Pushdown Timed Automata: a Binary Reachability Characterization and Safety Verification*

Zhe Dang

School of Electrical Engineering and Computer Science
Washington State University
Pullman, WA 99164, USA
zdang@eecs.wsu.edu

Abstract. We consider pushdown timed automata (PTAs) that are timed automata (with dense clocks) augmented with a pushdown stack. A configuration of a PTA includes a state, dense clock values and a stack word. By using the pattern technique, we give a decidable characterization of the binary reachability (i.e., the set of all pairs of configurations such that one can reach the other) of a PTA. Since a timed automaton can be treated as a PTA without the pushdown stack, we can show that the binary reachability of a timed automaton is definable in the additive theory of reals and integers. The results can be used to verify a class of properties containing linear relations over both dense variables and unbounded discrete variables. The properties previously could not be verified using the classic region technique nor expressed by timed temporal logics for timed automata and CTL* for pushdown systems. The results are also extended to other generalizations of timed automata.

Key words: Model-checking, Timed Automata, Pushdown Timed Automata, Binary Reachability, Presburger, Real-time Systems

1 Introduction

A timed automaton [3] can be considered as a finite automaton augmented with a number of dense (either real or rational) clocks. Clocks can be reset or progress at rate 1 depending upon the truth values of a number of clock constraints in the form of clock regions (i.e., comparisons of a clock or the difference of two clocks against an integer constant). Due to their ability to model and analyze a wide range of real-time systems, timed automata have been extensively studied in recent years (see [1, 36] for recent surveys). In particular, by using the standard region technique, it has been shown that region reachability for timed automata is decidable [3]. This fundamental result and the technique help researchers, both theoretically and practically, in formulating various

* A short version [19] of this paper appears in the *Proceedings of the 13th International Conference on Computer-aided Verification* (CAV'01), Lecture Notes in Computer Science 2102, pp. 506-517, Springer.

timed temporal logics [2, 4–6, 27, 32, 34, 35] and developing verification tools [11, 26, 30].

Region reachability is useful but has intrinsic limitations. In many real-world applications [14], we might also want to know whether a timed automaton satisfies a non-region property; e.g., for two given states s and s',

$$x_1 - 2x_2 + x'_3 > x'_1 + 4x'_2 - 3x_3$$

holds whenever clock values (x_1, x_2, x_3) at s can reach (x'_1, x'_2, x'_3) at s'. Comon and Jurski [17] have shown that the binary reachability of a timed automaton is definable in the additive theory of reals augmented with an integral predicate that tells whether a term is an integer, by flattening a timed automaton into a real-valued counter machine without nested cycles [16]. The result immediately paves the way for automatic verification of a class of non-region properties that previously were not possible using the region technique.

On the other hand, a strictly more powerful system, called a *pushdown timed automaton* (PTA), can be obtained by augmenting a timed automaton with a pushdown stack. PTAs are particularly interesting because they contain both dense clocks and unbounded discrete structures. They can be used to study, for instance, a timed version of pushdown processes [9, 24] or real-time programs with procedure calls. A configuration of a PTA is a tuple of a state, dense clock values, and a stack word. The binary reachability of a PTA is the set of all pairs of configurations such that one can reach the other. Comon and Jurski's result for timed automata inspires us to look for a similar result for PTAs. Is there a decidable binary reachability characterization for PTAs such that a class of non-region properties can be verified ? The main result in this paper answers this question positively.

There are several potential ways to approach the question. The first straightforward approach would be to treat a PTA as a Cartesian product of a timed automaton and a pushdown automaton. In this way, the binary reachability of a PTA can be formulated by simply combining Comon and Jurski's result and the fact that pushdown automata accept context-free languages. Obviously, this is wrong, since stack operations depend on clock values and thus can not be simply separated. The second approach is to closely look at the flattening technique of Comon and Jurski's to see whether the technique can be adapted by adding a pushdown stack. However, the second approach has an inherent difficulty: the flattening technique, as pointed out in their paper, destroys the structure of the original timed automaton, and thus, the sequences of stack operations can not be maintained after flattening.

Recently, the question has been answered positively, but only for integer-valued clocks (i.e., for discrete PTAs). It has been shown in [20] that the binary reachability of a discrete PTA can be accepted by a nondeterministic pushdown automaton augmented with reversal-bounded counters (NPCA), whose emptiness problem is known to be decidable [28]. However, as far as dense clocks are concerned, the automata-based technique used in [20] does not immediately apply. The reason is that traditional automata theories do not provide tools to deal with machines containing both real-valued counters (for dense clocks) and unbounded discrete data structures.

In order to handle dense clocks, inspired by Alur and Dill's region technique [3], we separate a dense clock into an integral part and a fractional part. Consider a pair $(\boldsymbol{v}_0, \boldsymbol{v}_1)$

of two tuples of clock values. We define (see Section 3 for details) an ordering, called the pattern of $(\boldsymbol{v}_0, \boldsymbol{v}_1)$, on the fractional parts of $\boldsymbol{v}_0$ and $\boldsymbol{v}_1$. The definition guarantees that there are only a finite number of distinct patterns. An equivalent relation "$\approx$" is defined such that $(\boldsymbol{v}_0, \boldsymbol{v}_1) \approx (\boldsymbol{v}'_0, \boldsymbol{v}'_1)$ iff $\boldsymbol{v}_0$ and $\boldsymbol{v}'_0$ ($\boldsymbol{v}_1$ and $\boldsymbol{v}'_1$ will also) have the same integral parts, and both $(\boldsymbol{v}_0, \boldsymbol{v}_1)$ and $(\boldsymbol{v}'_0, \boldsymbol{v}'_1)$ have the same pattern. The "$\approx$" essentially defines an equivalent relation with a countable number of equivalent classes such that the integral parts of $\boldsymbol{v}_0$ and $\boldsymbol{v}_1$ together with the pattern of the fractional parts of $\boldsymbol{v}_0$ and $\boldsymbol{v}_1$ determine the equivalent class of $(\boldsymbol{v}_0, \boldsymbol{v}_1)$. A good property of "$\approx$" is that it preserves the binary reachability: $\boldsymbol{v}_0$ can reach $\boldsymbol{v}_1$ by a sequence of transitions iff $\boldsymbol{v}'_0$ can reach $\boldsymbol{v}'_1$ by the (almost) same sequence of transitions, whenever $(\boldsymbol{v}_0, \boldsymbol{v}_1) \approx (\boldsymbol{v}'_0, \boldsymbol{v}'_1)$. Therefore, the fractional parts can be abstracted away from the dense clocks by using a pattern. In this way, by preserving the (almost) same control structure, a PTA can be transformed into a discrete transition system (called a pattern graph) containing discrete clocks (for the integral parts of the dense clocks) and a finite variable over patterns. By translating a pattern back to a relation over the fractional parts of the clocks, the decidable binary reachability characterization of the pattern graph derives the decidable characterization (namely, $(\mathbf{D} + \mathbf{NPCA})$-definable) for the PTA, since the relation is definable in the additive theory of reals. With this characterization, it can be shown that the particular class of safety properties that contain mixed linear relations over both dense variables (e.g., clock values) and discrete variables (e.g., word counts) can be automatically verified for PTAs. An example property is as follows: for two given states s and s', whenever configuration (s, x_1, x_2, w) can reach configuration (s', x'_1, x'_2, w'), the following condition holds,

$$x_1 + 2x'_2 - x_2 > \#_a(w) - \#_b(w'),$$

where $\#_a(w)$ is the number of symbol a's in the stack word w. The results can be easily extended to PTAs augmented with reversal-bounded counters. In particular, we can show that the binary reachability of a timed automaton is definable in the first-order additive theory over reals and integers with $\geq$ and $+$, i.e., $(\mathbf{R}, \mathbf{Z}, +, \geq, 0)$. Essentially, for timed automata, Comon and Jurski's characterization (the additive theory of reals augmented with an integral predicate) is equivalent to ours (the additive theory of reals and integers). The additive theory over reals and integers is decidable, for instance, by the Buchi-automata based decision procedure presented in [12].

Fractional orderings are an effective way to abstract the fractional parts of dense clocks. The idea of using fractional orderings can be traced back to the pioneering work of Alur and Dill in inventing the region technique [3]. Essentially, the region technique makes a finite partition of the clock space such that clock values in the same region give the same answer to each clock constraint in the system (i.e., the automaton of interest). Comon and Jurski [17] notice that Alur and Dill's partition is too coarse in establishing the binary reachability of a timed automata. They move one step further by bringing in the clock values before a transition was made. But Comon and Jurski's partition is still finite, since their partition, though finer than Alur and Dill's, is still based on answers to all the clock constraints (there are finitely many of them) in the system. In this paper, $\approx$ deduces an *infinite* partition of both the initial values $\boldsymbol{v}_0$ and the current values $\boldsymbol{v}_1$ of the clocks. Essentially, this partition is based on answers to all clock constraints (not just the ones in the system). That is, $\approx$ is finer than Comon and Jurski's partition as well as Alur and Dill's. This is why, when a PTA is concerned, the flattening technique [17] preserves the binary reachability of clock values but not stack words while the technique presented in this paper preserves the binary reachability of both. A class of Pushdown Timed Systems was discussed in [10]. However, that paper focuses on region reachability instead of binary reachability.

This paper is organized as follows. Section 2 reviews a number of definitions and, in particular, defines a decidable formalism in which the binary reachability of PTAs is expressed. Section 3 and Section 4 give the definition of patterns and show the correctness of using patterns as an abstraction for fractional clock values. Section 5 and Section 6 define PTAs and show that the pattern graph of a PTA has a decidable binary reachability characterization. Section 7 states the main results of the paper. In Section 8, we point out that the results in this paper can be extended to many other infinite state machine models augmented with dense clocks.

2 Preliminaries

A nondeterministic multicounter automaton is a nondeterministic automaton with a finite number of states, a one-way input tape, and a finite number of integer counters. Each counter can be incremented by 1, decremented by 1, or stay unchanged. Besides, a counter can be tested against an integer constant. It is well-known that counter machines with two counters have an undecidable halting problem [33], and obviously the undecidability holds for machines augmented with a pushdown stack. Thus, we have to restrict the behaviors of the counters. One such restriction is to limit the number of reversals a counter can make. A counter is *n-reversal-bounded* if it changes mode between nondecreasing and nonincreasing for at most n times. For instance, the following sequence of counter values:

$$0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 3, 2, 1, 1, 1, 1, \cdots$$

demonstrates only one counter reversal. A counter is *reversal-bounded* if it is n-reversal-bounded for some fixed number n independent of computations. A *reversal-bounded nondeterministic multicounter automaton (NCA)* is a nondeterministic multicounter automaton in which each counter is reversal-bounded. A *reversal-bounded nondeterministic pushdown multicounter automaton (NPCA)* is an NCA augmented with a pushdown stack. In addition to counter operations, an NPCA can pop the top symbol from the stack or push a word onto the top of the stack. It is known that the emptiness problem (i.e., whether a machine accepts some words?) for NPCAs (and hence NCAs) is decidable [28].

Lemma 1. *The emptiness problem for reversal-bounded nondeterministic pushdown multicounter automata is decidable.*

When an automaton does not have an input tape, we call it a machine. In this case, we are interested in the behaviors generated by the machine rather than the language accepted by the automaton. We shall use NPCM (resp. NCM) to stand for NCPA (resp. NCA) without an input tape.

Let $\mathbf{Z}$ be integers, $\mathbf{N}$ be nonnegative integers, $\mathbf{D} = \mathbf{Q}$ (rationals) or $\mathbf{R}$ (reals) be a dense domain, and Γ be an alphabet. $\mathbf{D}^+$ is used to denote nonnegative values in $\mathbf{D}$. Each value $v \in \mathbf{D}$ can be uniquely expressed as the sum of $\lceil v \rceil + \lfloor v \rfloor$, where $\lceil v \rceil \in \mathbf{Z}$ is the integral part of v, and $0 \leq \lfloor v \rfloor < 1$ is the fractional part of v. A *dense variable* is a variable over $\mathbf{D}$. An *integer variable* is a variable over $\mathbf{Z}$. A *word variable* is a variable over Γ^*. Let $m \geq 1$. For each $1 \leq i \leq m$, we use x_i, y_i, and w_i to denote a dense variable, an integer variable, and a word variable, respectively. We use $\#_a(w_i)$ to denote a *count variable* representing the number of symbol $a \in \Gamma$ in w_i. A *linear term* t is defined as follows:

$$t ::= n \mid x_i \mid y_i \mid \#_a(w_i) \mid t - t \mid t + t,$$

where $n \in \mathbf{Z}, a \in \Gamma$ and $1 \leq i \leq m$. A *mixed linear relation* l is defined as follows:

$$l ::= t > 0 \mid t = 0 \mid t_{discrete} \, mod \, n = 0 \mid \neg l \mid l \wedge l,$$

where t is a linear term, $0 \neq n \in \mathbf{N}$, and $t_{discrete}$ is a linear term not containing dense variables. Notice that a mixed linear relation could contain dense variables, integer variables and count variables. A *dense linear relation* is a mixed linear relation that contains dense variables only. A *discrete linear relation* is a mixed linear relation that does not contain dense variables. Obviously, any discrete linear relation is a Presburger formula over integer variables and count variables.

Each integer can be represented as a unary string, e.g., string "00000" (resp. "11111") for integer $+5$ (resp. -5). In this way, a tuple of integers and words can be encoded as a string by concatenating the unary representations of the integers and the words, with a separator $\$ \notin \Gamma$. For instance, $(2, -4, w)$ is encoded as string "00\$1111\$$w$". Consider a predicate H over integer variables and word variables. The domain of H is the set of tuples of integers and words that satisfy H. Under the encoding, the domain of H can be treated as a set of strings, i.e., a language. A predicate H over integer variables and word variables is an *NPCA predicate* (or simply NPCA) if there is an NPCA accepting the domain of H. A $(\mathbf{D} + \mathbf{NPCA})$-*formula* f is defined as follows:

$$f ::= l_{dense} \wedge H \mid l_{dense} \vee H \mid f \vee f,$$

where l_{dense} is a dense linear relation and H is an NPCA predicate. Therefore, a $(\mathbf{D} + \mathbf{NPCA})$ formula is a finite disjunction of formulas in the form of $l_{dense} \wedge H$ or $l_{dense} \vee H$, where dense variables (contained only in each l_{dense}) and discrete variables (contained only in each H) are separated. Let $p, q, r \geq 0$. A predicate A on tuples in $\mathbf{D}^p \times \mathbf{Z}^q \times (\Gamma^*)^r$ is $(\mathbf{D} + \mathbf{NPCA})$-*definable* if there is a $(\mathbf{D} + \mathbf{NPCA})$-formula f with p dense variables, $p + q$ integer variables, and r word variables, such that, for all $x_1, \cdots, x_p$ in $\mathbf{D}$, for all $y_1, \cdots, y_q$ in $\mathbf{Z}$, and for all $w_1, \cdots, w_r$ in Γ^*,

$$(x_1, \cdots, x_p, y_1, \cdots, y_q, w_1, \cdots, w_r) \in A$$

$$\text{iff } f(\lfloor x_1 \rfloor, \cdots, \lfloor x_p \rfloor, \lceil x_1 \rceil, \cdots, \lceil x_p \rceil, y_1, \cdots, y_q, w_1, \cdots, w_r) \text{ holds.}$$

Lemma 2. *(1). Both $l_{discrete} \wedge H$ and $l_{discrete} \vee H$ are NPCA predicates, if $l_{discrete}$ is a discrete linear relation and H is an NPCA predicate.*

(2). NPCA predicates are closed under existential quantifications (over integer variables and word variables).

(3). If A is $(\mathbf{D}+\mathbf{NPCA})$-definable and l is a mixed linear relation, then both $l \wedge A$ and $l \vee A$ are $(\mathbf{D}+\mathbf{NPCA})$-definable.

(4). The emptiness (or satisfiability) problem for $(\mathbf{D}+\mathbf{NPCA})$-definable predicates is decidable.

Proof. (1). $l_{discrete}$ is a Presburger formula. (The domain of) $l_{discrete}$ can therefore be accepted by a deterministic NCA [28]. Hence, $l_{discrete} \wedge H$ and $l_{discrete} \vee H$ can be accepted by NPCAs by "intersecting" and "joining" the deterministic NCA and the NPCA that accepts H, respectively.

(2). Let H be an NPCA predicate containing variable z (either an integer variable or a word variable). Assume that H is accepted by NPCA M. An NPCA M' can be constructed to accept $\exists z H$ by guessing each symbol in the encoding of z (on the input tape of M) and simulating M.

(3). We first show that any mixed linear relation l is definable by a *separately* mixed linear relation l' (i.e., l' is a Boolean combination of dense linear relations and discrete linear relations. So, l' does not have a term containing both dense variables and discrete variables.). That is, for all $x_1, \cdots, x_p \in \mathbf{D}, y_1, \cdots, y_q \in \mathbf{Z}$,

$$l(x_1, \cdots, x_p, y_1, \cdots, y_q) \text{ iff } l'(\lfloor x_1 \rfloor, \cdots, \lfloor x_p \rfloor, \lceil x_1 \rceil, \cdots, \lceil x_p \rceil, y_1, \cdots, y_q).$$

Instead of giving a lengthy proof, we look at an example of l: $x_1 - x_2 + y_1 > 2$. This can be rewritten as: $\lceil x_1 \rceil - \lceil x_2 \rceil + y_1 - 2 + \lfloor x_1 \rfloor - \lfloor x_2 \rfloor > 0$. Term $\lfloor x_1 \rfloor - \lfloor x_2 \rfloor$ is the only part containing dense variables. Since $\lfloor x_1 \rfloor - \lfloor x_2 \rfloor$ is bounded, separating cases for this term being at (and between) -1, 0, 1 will give a separately mixed linear relation l'. This separation idea can be applied for any mixed linear relation l. If A is definable by a $(\mathbf{D}+\mathbf{NPCA})$-formula f, then $l \wedge A$ (resp. $l \vee A$) is definable by $l' \wedge f$ (resp. $l' \vee f$). By re-organizing the dense linear relations (in l' and in f) and the discrete linear relations (in l') such that the discrete linear relations are grouped with the NPCA predicates in f, $l' \wedge f$ and $l' \vee f$ can be made $(\mathbf{D}+\mathbf{NPCA})$-formulas using Lemma 2 (1).

(4). The emptiness problem for $l_{dense} \wedge H$ and $l_{dense} \vee H$ is decidable, noticing that the emptiness for l_{dense}, which is expressible in the additive theory of reals (or rationals), is decidable, and the emptiness of NPCA predicate H is decidable (Lemma 1). Therefore, the emptiness of any $(\mathbf{D}+\mathbf{NPCA})$ formulas, as well as, from Lemma 2 (3), any $(\mathbf{D}+\mathbf{NPCA})$-definable predicates, is decidable. ∎

3 Clock Patterns and Their Changes

A dense clock is simply a dense variable over $\mathbf{D}^+$. Now we fix a $k > 0$ and consider $k+1$ clocks $\boldsymbol{x} = x_0, \cdots, x_k$. For technical reasons, x_0 is an auxiliary clock indicating the current time *now*. Let $K = \{0, \cdots, k\}$, and $K^+ = \{1, \cdots, k\}$. A subset K' of K is abused as a set of clocks; i.e., we say $x_i \in K'$ if $i \in K'$. A *(clock) valuation* $\boldsymbol{v}$ is a function $K \to \mathbf{D}^+$ that assigns a value in $\mathbf{D}^+$ to each clock in K. A *discrete (clock) valuation* $\mathbf{u}$ is a function $K \to \mathbf{N}$ that assigns a value in $\mathbf{N}$ to each clock in K. For each valuation $\boldsymbol{v}$ and $\delta \in \mathbf{D}^+$, $\lceil \boldsymbol{v} \rceil$, $\lfloor \boldsymbol{v} \rfloor$ and $\boldsymbol{v} + \delta$ are valuations satisfying $\lceil \boldsymbol{v} \rceil(i) = \lceil \boldsymbol{v}(i) \rceil$, $\lfloor \boldsymbol{v} \rfloor(i) = \lfloor \boldsymbol{v}(i) \rfloor$ and $(\boldsymbol{v} + \delta)(i) = \boldsymbol{v}(i) + \delta$, respectively, for each $i \in K$. The *relative representation* $\widehat{\boldsymbol{v}}$ of a valuation $\boldsymbol{v}$ is a valuation satisfying:

- $\lceil \widehat{\boldsymbol{v}} \rceil = \lceil \boldsymbol{v} \rceil$,
- $\lfloor \widehat{\boldsymbol{v}} \rfloor(0) = \lfloor 1 - \lfloor \boldsymbol{v} \rfloor(0) \rfloor$,
- $\lfloor \widehat{\boldsymbol{v}} \rfloor(i) = \lfloor \lfloor \boldsymbol{v} \rfloor(i) + \lfloor \widehat{\boldsymbol{v}} \rfloor(0) \rfloor$, for each $i \in K^+$.

A valuation $\boldsymbol{v}_0$ is *initial* if the auxiliary clock x_0 has value 0 in $\boldsymbol{v}_0$.

Example 1. Let $k = 4$ and $\boldsymbol{v}_1 = (4.296, 1.732, 1.414, 5.289, 3.732)$. It can be calculated that $\widehat{\boldsymbol{v}}_1 = (4.704, 1.436, 1.118, 5.993, 3.436)$. Let $\boldsymbol{v}_2 = \boldsymbol{v}_1 + .268 = (4.564, 2, 1.682, 5.557, 4)$. Then, $\widehat{\boldsymbol{v}}_2 = (4.436, 2.436, 1.118, 5.993, 4.436)$. Notice that all the fractional parts (except for $\widehat{\boldsymbol{v}}_1(0)$ and $\widehat{\boldsymbol{v}}_2(0)$) are the same in $\widehat{\boldsymbol{v}}_1$ and $\widehat{\boldsymbol{v}}_2$. It is easy to show that a clock progress (i.e., $x_0, \cdots, x_k$ progress by the same amount such as .268) will not change the fractional parts of clock values (for clocks $x_1, \cdots, x_k$) in a relative representation. ∎

3.1 Clock Patterns

We distinguish two disjoint sets, $K^0 = \{0^0, \cdots, k^0\}$ and $K^1 = \{0^1, \cdots, k^1\}$, of indices. A *pattern*, $\pi = p_0, \cdots, p_n$ with $0 \leq n < 2(k+1)$, is a partition of $K^0 \cup K^1$ satisfying:

- $0^0 \in p_0$ and
- $\cup_{0 \leq i \leq n} p_i = K^0 \cup K^1$.

In pattern π, p_i is called the *i-position*. A pair of valuations $(\boldsymbol{v}_0, \boldsymbol{v}_1)$ is *initialized* if $\boldsymbol{v}_0$ is initial. The pattern of $(\boldsymbol{v}_0, \boldsymbol{v}_1)$ characterizes the ordering between elements in $\lfloor \widehat{\boldsymbol{v}}_0 \rfloor$ and $\lfloor \widehat{\boldsymbol{v}}_1 \rfloor$ (where K^0 is for indices of $\boldsymbol{v}_0$ and K^1 is for indices of $\boldsymbol{v}_1$). Formally, an initialized pair $(\boldsymbol{v}_0, \boldsymbol{v}_1)$ *has pattern* $\pi = p_0, \cdots, p_n$, written $(\boldsymbol{v}_0, \boldsymbol{v}_1) \in \pi$, or $[(\boldsymbol{v}_0, \boldsymbol{v}_1)] = \pi$, if, for each $0 \leq m, m' \leq n$, each $b, b' \in \{0, 1\}$, and each $i, i' \in K$, $i^b \in p_m$ and $i'^{b'} \in p_{m'}$ imply that

$$\lfloor \widehat{\boldsymbol{v}}_b \rfloor(i) \sim \lfloor \widehat{\boldsymbol{v}}_{b'} \rfloor(i') \Leftrightarrow m \sim m', \text{ with } \sim \in \{<, =\}.$$

Though this definition of a pattern is quite complex, a pattern can be easily visualized after looking at the following example.

Example 2. Consider $\boldsymbol{v}_1$ in Example 1 and an initial valuation $\boldsymbol{v}_0 = (0, 3.118, 5.118, 2, 1.876)$. Since $\boldsymbol{v}_0$ is initial, $\widehat{\boldsymbol{v}}_0 = \boldsymbol{v}_0$. The fractional parts of $\widehat{\boldsymbol{v}}_0$ and $\widehat{\boldsymbol{v}}_1$ can be put on a big circle representing the interval $[0, 1)$ as shown in Figure 1. Each fractional value $\lfloor \widehat{\boldsymbol{v}}_0 \rfloor(i)$ for $\boldsymbol{v}_0$ is represented by an oval; each fractional value $\lfloor \widehat{\boldsymbol{v}}_1 \rfloor(i)$ for $\boldsymbol{v}_1$ is represented by a box. The pattern of $(\boldsymbol{v}_0, \boldsymbol{v}_1)$ can be drawn by collecting clockwisely (from the top, i.e., $\widehat{\boldsymbol{v}}_0(0) = 0$) the indices (superscripted with 0, e.g., 3^0 for $\widehat{\boldsymbol{v}}_0(3)$) for each component in $\widehat{\boldsymbol{v}}_0$ and the indices (superscripted with 1, e.g., 3^1 for $\widehat{\boldsymbol{v}}_1(3)$) for each component in $\widehat{\boldsymbol{v}}_1$; i.e., the pattern is

$$\pi = p_0, \cdots, p_5$$

with $p_0 = \{0^0, 3^0\}, p_1 = \{1^0, 2^0, 2^1\}, p_2 = \{1^1, 4^1\}, p_3 = \{0^1\}, p_4 = \{4^0\}, p_5 = \{3^1\}$. ∎

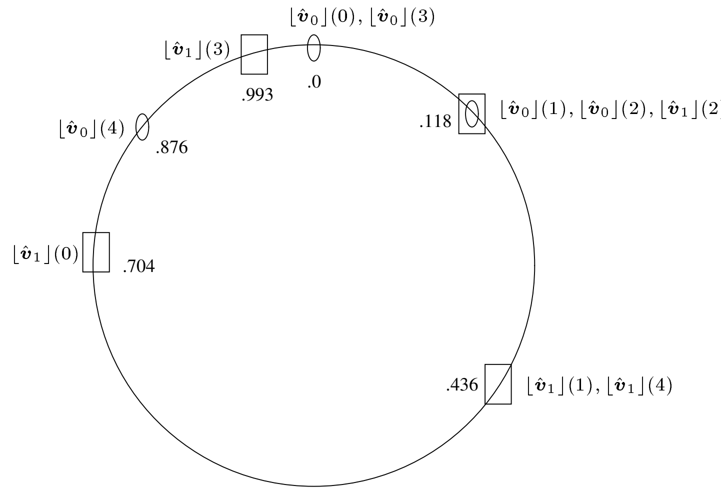

Fig. 1. A graphical representation of $\lfloor\widehat{\boldsymbol{v}}_0\rfloor$ and $\lfloor\widehat{\boldsymbol{v}}_1\rfloor$ in Example 2.

Notice that $0^1 \in K^1$ stands for the index for the value of clock x_0 (representing now) in $\boldsymbol{v}_1$. If $0^1 \in p_i$, then p_i is the *now-position* of π. A pattern is a *merge-pattern* if the now-position is a singleton set (i.e., 0^1 is the only element). A pattern is a *split-pattern* if it is not a merge-pattern, i.e., the now-position contains more than one element. ("Merge" and "split" will be made clear in a moment.) There are at most $2^{6(k+1)^2}$ distinct patterns. Let Π denote the set of all the patterns (for the fixed k).

Given two initialized pairs $(\boldsymbol{v}_0^1, \boldsymbol{v}_1)$ and $(\boldsymbol{v}_0^2, \boldsymbol{v}_2)$, we write $(\boldsymbol{v}_0^1, \boldsymbol{v}_1) \approx (\boldsymbol{v}_0^2, \boldsymbol{v}_2)$, if $(\boldsymbol{v}_0^1, \boldsymbol{v}_1)$ and $(\boldsymbol{v}_0^2, \boldsymbol{v}_2)$ have the same pattern, and have the same integral parts (i.e., $\lceil \boldsymbol{v}_0^1 \rceil = \lceil \boldsymbol{v}_0^2 \rceil, \lceil \boldsymbol{v}_1 \rceil = \lceil \boldsymbol{v}_2 \rceil$). A valuation $\boldsymbol{v}_1$ *has pattern* π if there is an initial $\boldsymbol{v}_0$ such that $(\boldsymbol{v}_0, \boldsymbol{v}_1)$ has pattern π. $\boldsymbol{v}_1$ may have a number of patterns, by different choices of $\boldsymbol{v}_0$. Observe that a pattern of $\boldsymbol{v}_1$ tells the truth values of all the *fractional orderings* $\lfloor \boldsymbol{v}_1 \rfloor(i) \sim \lfloor \boldsymbol{v}_1 \rfloor(j)$ and $\lfloor \boldsymbol{v}_1 \rfloor(i) \sim 0$ with $\sim \in \{<, >, \leq, \geq, =\}$, for all $i, j \in K^+$.

3.2 Clock Progresses

For each $0 < \delta \in \mathbf{D}^+$, $\boldsymbol{v} + \delta$ is the result of a clock progress from $\boldsymbol{v}$ by an amount of δ. How does a pattern change according to the progress? Let us first look at an example.

Example 3. Consider $\boldsymbol{v}_1, \boldsymbol{v}_2$ $(= \boldsymbol{v}_1 + .268)$ in Example 1, and $\boldsymbol{v}_0$ in Example 2. In Example 2, we indicated that the pattern π_1 of $(\boldsymbol{v}_0, \boldsymbol{v}_1)$ is

$$\{0^0, 3^0\}, \{1^0, 2^0, 2^1\}, \{1^1, 4^1\}, \{0^1\}, \{4^0\}, \{3^1\}.$$

Similar steps can be followed to show that the pattern π_2 of $(\boldsymbol{v}_0, \boldsymbol{v}_2)$ is

$$\{0^0, 3^0\}, \{1^0, 2^0, 2^1\}, \{1^1, 4^1, 0^1\}, \{4^0\}, \{3^1\}.$$

A helpful way to see the relationship between π_1 and π_2 is by looking at Figure 1. Holding the box labeled by $\lfloor \widehat{\boldsymbol{v}}_1 \rfloor(0)$ (for the current time) and sliding counter-clockwisely along the big circle for an amount of .268 will stop at the box labeled by $\lfloor \widehat{\boldsymbol{v}}_1 \rfloor(1)$ and $\lfloor \widehat{\boldsymbol{v}}_1 \rfloor(4)$. Thus, the pattern π_2 (after sliding) is exactly π_1 (before sliding) except that 0^1 in the 3-position in π_1 is merged into the 2-position in π_2. Notice that π_1 is a merge-pattern and the resulting π_2 is a split-pattern. The integral parts $\lceil \boldsymbol{v}_1 \rceil(1)$ and $\lceil \boldsymbol{v}_1 \rceil(4)$ change to $\lceil \boldsymbol{v}_2 \rceil(1) = \lceil \boldsymbol{v}_1 \rceil(1) + 1$ and $\lceil \boldsymbol{v}_2 \rceil(4) = \lceil \boldsymbol{v}_1 \rceil(4) + 1$. But all the other components of $\lceil \boldsymbol{v}_1 \rceil$ do not change. The reason is that, after merging 0^1 with 1^1 and 4^1 in π_2, the fractional parts $\lfloor \boldsymbol{v}_2 \rfloor(1)$ and $\lfloor \boldsymbol{v}_2 \rfloor(4)$ are "rounded" (i.e., become 0). What if we further make a clock progress from $\boldsymbol{v}_2$ for an amount of $\delta' = .12$? The resulting pattern π_3 of $(\boldsymbol{v}_0, \boldsymbol{v}_3)$ with $\boldsymbol{v}_3 = \boldsymbol{v}_2 + \delta'$ is the result of splitting 0^1 from the 2-position $\{1^1, 4^1, 0^1\}$. That is, π_3 is

$$\{0^0, 3^0\}, \{1^0, 2^0, 2^1\}, \{0^1\}, \{1^1, 4^1\}, \{4^0\}, \{3^1\},$$

which is a merge-pattern again. This process of merging and splitting can be formally defined as the following function $next$. ∎

Function $next : \mathit{\Pi} \times (\mathbf{N})^{k+1} \to \mathit{\Pi} \times (\mathbf{N})^{k+1}$ describes how a pattern changes upon a clock progress. Given any discrete valuation $\boldsymbol{u}$ and pattern $\pi = p_0, \cdots, p_n$ with the now-position being p_i for some i, $next(\pi, \boldsymbol{u})$ is defined to be $(\pi', \boldsymbol{u}')$ such that,

- (the case when π is a merge-pattern) if $|p_i| = 1$ and hence $i > 0$, then π' is

  $$p_0, \cdots, p_{i-1} \cup \{0^1\}, p_{i+1}, \cdots, p_n$$

  (that is, π' is the result of merging the now-position to the previous position), and for each $j \in K^+$, if $j^1 \in p_{i-1}$, then $\boldsymbol{u}'(j) = \boldsymbol{u}(j) + 1$ else $\boldsymbol{u}'(j) = \boldsymbol{u}(j)$. Besides, if $i = 1$ (i.e., the now-position is merged to p_0), then $\boldsymbol{u}'(0) = \boldsymbol{u}(0) + 1$ else $\boldsymbol{u}'(0) = \boldsymbol{u}(0)$,
- (the case when π is a split pattern) if $|p_i| > 1$, then π' is the result of splitting 0^1 from the now-position. That is, if $i > 0$, π' is

  $$p_0, \cdots, p_{i-1}, \{0^1\}, p_i - \{0^1\}, p_{i+1}, \cdots, p_n.$$

  However, if $i = 0$, π' is

  $$p_0 - \{0^1\}, p_1, \cdots, p_n, \{0^1\}.$$

  In either case, $\boldsymbol{u}' = \boldsymbol{u}$.

If $next(\pi, \boldsymbol{u}) = (\pi', \boldsymbol{u}')$, π' is called *the next pattern of* π, written $Next(\pi)$.

To better understand $Next(\cdot)$, we visualize pattern π as a circle shown in Figure 2. Applications of $Next(\cdot)$ can be regarded as moving the index 0^1 along the circle, by performing merge-operations (Figure 2 (a)) and split-operations (Figure 2 (b)) alternatively. After enough number of applications of $Next(\cdot)$, 0^1 will return to the original now-position after moving through the entire circle. That is, for each pattern π, there is a smallest positive integer m such that $Next^m(\pi) = \pi$; i.e., there are $\pi_0, \cdots, \pi_m$ satisfying $\pi_0 = \pi_m = \pi$ and $Next(\pi_i) = \pi_{i+1}$ for each $0 \le i < m$. More precisely, by looking at Figure 2, if π is a merge-pattern, $m = 2n$; if π is a split-pattern,

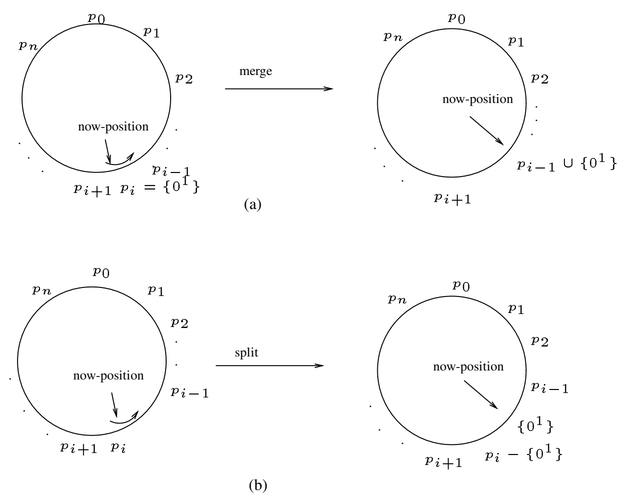

Fig. 2. A graphical representation of the $Next(\cdot)$ operator.

$m = 2(n+1)$. Notice that $next^m(\pi, \boldsymbol{u}) = (\pi, \boldsymbol{u}+1)$ for each $\boldsymbol{u}$. The sequence $\pi_0, \cdots, \pi_m$ is called a *pattern ring*.

Fix any initialized pair $(\boldsymbol{v}_0, \boldsymbol{v})$ and $0 < \delta \in \mathbf{D}^+$. Assume that the patterns of $(\boldsymbol{v}_0, \boldsymbol{v})$ and $(\boldsymbol{v}_0, \boldsymbol{v}+\delta)$ are π and π', respectively. $\boldsymbol{v}$ *has no pattern change for* δ if, for all $0 \leq \delta' \leq \delta$, $(\boldsymbol{v}_0, \boldsymbol{v}+\delta')$ has the same pattern. $\boldsymbol{v}$ *has one pattern change for* δ if $Next(\pi) = \pi'$ (hence $\pi \neq \pi'$) and,

- (when π is a merge-pattern) for all $0 \leq \delta' < \delta$, $(\boldsymbol{v}_0, \boldsymbol{v}+\delta')$ has pattern π, or,
- (when π is a split-pattern) for all $0 < \delta' \leq \delta$, $(\boldsymbol{v}_0, \boldsymbol{v}+\delta')$ has pattern π'.

The following lemma can be observed.

Lemma 3. *For any initialized pair* $(\boldsymbol{v}_0, \boldsymbol{v})$ *and any* $0 < \delta \in \mathbf{D}^+$*, the following statements are equivalent:*

(1). $next([(\boldsymbol{v}_0, \boldsymbol{v})], \lceil \boldsymbol{v} \rceil) = ([(\boldsymbol{v}_0, \boldsymbol{v}+\delta)], \lceil \boldsymbol{v}+\delta \rceil)$,
(2). $\boldsymbol{v}$ *has one pattern change for* δ.

$\boldsymbol{v}$ *has* n *pattern changes for* δ with $n \geq 1$, if there are positive $\delta_1, \cdots, \delta_n$ in $\mathbf{D}^+$ with $\Sigma_{1 \leq i \leq n} \delta_i = \delta$ such that $\boldsymbol{v} + \Sigma_{1 \leq i \leq j} \delta_i$ has one pattern change for δ_{j+1}, for each $j = 0, \cdots, n-1$. It is noticed that for any $\delta \leq 1$, $\boldsymbol{v}$ has at most m pattern changes, where m is the length of the pattern ring starting from the pattern π of $(\boldsymbol{v}_0, \boldsymbol{v})$. This m is uniformly bounded by $4(k+1)$.

3.3 Clock Resets

In addition to clock progresses, clock resets are the other form of clock behaviors. Let $r \subseteq K^+$ be (a set of) *clock resets*. $\boldsymbol{v} \downarrow_r$ denotes the result of resetting each clock $x_i \in r$

(i.e., $i \in r$). That is, for each $i \in K$,

$$(\boldsymbol{v} \downarrow_r)(i) = \begin{cases} 0 & \text{if } i \in r \\ \boldsymbol{v}(i) & \text{otherwise.} \end{cases}$$

Example 4. Consider $\boldsymbol{v}_0$ and $\boldsymbol{v}_1$ given in Example 1 and Example 2. Assume $r = \{4\}$. By definition, $\boldsymbol{v}_1 \downarrow_r = (4.296, 1.732, 1.414, 5.289, 0)$. It can be calculated that the relative representation of $\boldsymbol{v}_1 \downarrow_r$ is $(4.704, 1.436, 1.118, 5.993, 0.704)$. The pattern of $(\boldsymbol{v}_0, \boldsymbol{v}_1 \downarrow_r)$ can be figured out again by looking at Figure 1. The reset of clock x_4 can be conceptually regarded as moving the label $\lfloor\widehat{\boldsymbol{v}}_1\rfloor(4)$ from the box of $\lfloor\widehat{\boldsymbol{v}}_1\rfloor(1)$ and $\lfloor\widehat{\boldsymbol{v}}_1\rfloor(4)$ to the box of $\lfloor\widehat{\boldsymbol{v}}_1\rfloor(0)$ (the current time). Therefore, the pattern after the reset changes from

$$\{0^0, 3^0\}, \{1^0, 2^0, 2^1\}, \{1^1, 4^1\}, \{0^1\}, \{4^0\}, \{3^1\}$$

of $(\boldsymbol{v}_0, \boldsymbol{v}_1)$ to

$$\{0^0, 3^0\}, \{1^0, 2^0, 2^1\}, \{1^1\}, \{0^1, 4^1\}, \{4^0\}, \{3^1\}$$

of $(\boldsymbol{v}_0, \boldsymbol{v}_1 \downarrow_r)$ by moving 4^1 into the position containing 0^1. ∎

Functions $reset_r : \Pi \times (\mathbf{N})^{k+1} \to \Pi \times (\mathbf{N})^{k+1}$ for $r \subseteq K^+$ describe how a pattern changes after clock resets. Given any discrete valuation $\boldsymbol{u}$ and any pattern $\pi = p_0, \cdots, p_n$ with the now-position being p_i for some i, $reset_r(\pi, \boldsymbol{u})$ is defined to be $(\pi', \boldsymbol{u}')$ such that,

- π' is $p_0 - r^1, \cdots, p_{i-1} - r^1, p_i \cup r^1, p_{i+1} - r^1, \cdots, p_n - r^1$, where $r^1 = \{j^1 : j \in r\} \subseteq K^1$. Therefore, π' is the result of bringing every index in r^1 into the now-position. Notice that some of $p_m - r^1$ may be empty after moving indices in r^1 out of p_m, for $m \neq i$. In this case, these empty elements are removed from π' (to guarantee that π' is well defined.),
- for each $j \in K$, if $j \in r$, then $\boldsymbol{u}'(j) = 0$ else $\boldsymbol{u}'(j) = \boldsymbol{u}(j)$.

If $reset_r(\pi, \boldsymbol{u}) = (\pi', \boldsymbol{u}')$, π' is written as $Reset_r(\pi)$. The following lemma can be observed.

Lemma 4. *For any initialized pair $(\boldsymbol{v}_0, \boldsymbol{v})$ and any $r \subseteq K^+$,*

$$reset_r([(\boldsymbol{v}_0, \boldsymbol{v})], \lceil\boldsymbol{v}\rceil) = ([(\boldsymbol{v}_0, \boldsymbol{v} \downarrow_r)], \lceil\boldsymbol{v} \downarrow_r\rceil).$$

4 Clock Constraints and Patterns

An *atomic clock constraint* (over clocks $x_1, \cdots, x_k$, excluding x_0) is a formula in the form of $x_i - x_j \sim d$ or $x_i \sim d$ where $d \in \mathbf{Z}$ and $\sim \in \{<, >, \leq, \geq, =\}$. A *clock constraint* c is a Boolean combination of atomic clock constraints. Let $\mathcal{C}$ be the set of all clock constraints (over clocks $x_1, \cdots, x_k$). We say $\boldsymbol{v} \in c$ if clock valuation $\boldsymbol{v}$ (for $x_0, \cdots, x_k$) satisfies clock constraint c.

Any clock constraint c can be written as a Boolean combination $I(c)$ of clock constraints over discrete clocks $\lceil x_1\rceil, \cdots, \lceil x_k\rceil$ and fractional orderings $\lfloor x_i\rfloor \sim \lfloor x_j\rfloor$ and $\lfloor x_i\rfloor \sim 0$. For instance, $x_i - x_j < d$ is equivalent to $\lceil x_i\rceil - \lceil x_j\rceil < d \vee (\lceil x_i\rceil - \lceil x_j\rceil =$

$d \wedge \lfloor x_i \rfloor < \lfloor x_j \rfloor)$. $x_i > d$ is equivalent to $\lceil x_i \rceil > d \vee (\lceil x_i \rceil = d \wedge \lfloor x_i \rfloor > 0)$. Therefore, testing $\boldsymbol{v} \in c$ is equivalent to testing $\lceil \boldsymbol{v} \rceil$ and the fractional orderings on $\lfloor \boldsymbol{v} \rfloor$ satisfying $I(c)$.

Assume that $\boldsymbol{v} \in \pi$ (i.e., $\boldsymbol{v}$ has pattern π). As we mentioned earlier, the truth value for each fractional ordering on $\lfloor \boldsymbol{v} \rfloor$ can be told from π. We use $I(c)^\pi$, or simply c^π, to denote the result of replacing fractional orderings in $I(c)$ by the truth values given by π. c^π, without containing fractional orderings, is just a clock constraint (over discrete clocks). Notice that the pattern space $\mathit{\Pi}$ is finite, therefore, $\boldsymbol{v} \in c$ is equivalent to

$$\bigvee_{\pi \in \mathit{\Pi}} (\boldsymbol{v} \in \pi \wedge \lceil \boldsymbol{v} \rceil \in c^\pi).$$

Hence, the truth value of $\boldsymbol{v} \in c$ only depends on a pattern of $\boldsymbol{v}$ and the integral parts of $\boldsymbol{v}$.

Now, we consider two initialized pairs $(\boldsymbol{v}_0^1, \boldsymbol{v}_1)$ and $(\boldsymbol{v}_0^2, \boldsymbol{v}_2)$ such that

$$(\boldsymbol{v}_0^1, \boldsymbol{v}_1) \approx (\boldsymbol{v}_0^2, \boldsymbol{v}_2).$$

Assume that $\boldsymbol{v}_1$ can be reached from a valuation $\boldsymbol{v}^1$ via a clock progress by an amount of δ_1, i.e., $\boldsymbol{v}^1 + \delta_1 = \boldsymbol{v}_1$. We would like to know whether $\boldsymbol{v}_2$ can be reached from some valuation $\boldsymbol{v}^2$ also via a clock progress but probably by a slightly different amount of δ_2 such that $(\boldsymbol{v}_0^1, \boldsymbol{v}^1)$ and $(\boldsymbol{v}_0^2, \boldsymbol{v}^2)$ are still equivalent($\approx$). We also expect that for any test c, if during the progress of $\boldsymbol{v}^1$, c is consistently satisfied, then so is c for the progress of $\boldsymbol{v}^2$. The following lemma concludes that these, as well as the parallel case for clock resets, can be done. This result will be used later to show that if $\boldsymbol{v}_1$ is reached from $\boldsymbol{v}_0^1$ by a sequence of transitions that repeatedly perform clock progresses and clock resets, then $\boldsymbol{v}_2$ can be also reached from $\boldsymbol{v}_0^2$ via a very similar sequence such that no test c can distinguish the two sequences.

Lemma 5. *For any initialized pairs $(\boldsymbol{v}_0^1, \boldsymbol{v}_1)$ and $(\boldsymbol{v}_0^2, \boldsymbol{v}_2)$ with $(\boldsymbol{v}_0^1, \boldsymbol{v}_1) \approx (\boldsymbol{v}_0^2, \boldsymbol{v}_2)$,*

- *(1). for any positive $\delta_1 \in \mathbf{D}^+$, for any clock valuation $\boldsymbol{v}^1$, if $\boldsymbol{v}^1 + \delta_1 = \boldsymbol{v}_1$, then there exist a positive $\delta_2 \in \mathbf{D}^+$ and clock valuation $\boldsymbol{v}^2$ such that*
  - *(1.1). $\boldsymbol{v}^2 + \delta_2 = \boldsymbol{v}_2$ and $(\boldsymbol{v}_0^1, \boldsymbol{v}^1) \approx (\boldsymbol{v}_0^2, \boldsymbol{v}^2)$,*
  - *(1.2). $\boldsymbol{v}^1$ is initial iff $\boldsymbol{v}^2$ is initial, $\boldsymbol{v}^1 = \boldsymbol{v}_0^1$ iff $\boldsymbol{v}^2 = \boldsymbol{v}_0^2$, and, for any clock constraint $c \in \mathcal{C}$, $\boldsymbol{v}^1 \in c$ (resp. $\boldsymbol{v}_1 \in c$) iff $\boldsymbol{v}^2 \in c$ (resp. $\boldsymbol{v}_2 \in c$),*
  - *(1.3). for any clock constraint $c \in \mathcal{C}$, $\forall 0 \le \delta \le \delta_1 (\boldsymbol{v}^1 + \delta \in c)$ iff $\forall 0 \le \delta \le \delta_2 (\boldsymbol{v}^2 + \delta \in c)$.*
- *(2). for any $r \subseteq K^+$, for any clock valuation $\boldsymbol{v}^1$, if $\boldsymbol{v}^1 \downarrow_r = \boldsymbol{v}_1$, then there exists a valuation $\boldsymbol{v}^2$ such that*
  - *(2.1). $\boldsymbol{v}^2 \downarrow_r = \boldsymbol{v}_2$ and $(\boldsymbol{v}_0^1, \boldsymbol{v}^1) \approx (\boldsymbol{v}_0^2, \boldsymbol{v}^2)$,*
  - *(2.2). same as (1.2).*

Proof. (1). Assume that δ_1 is "small"; i.e., from $\boldsymbol{v}^1$ to $\boldsymbol{v}_1 = \boldsymbol{v}^1 + \delta_1$, there is at most one pattern change. Let $\pi = p_0, \cdots, p_n$ be the pattern for $(\boldsymbol{v}_0^2, \boldsymbol{v}_2)$ (and, hence, for $(\boldsymbol{v}_0^1, \boldsymbol{v}_1)$). Assume $0^1 \in p_i$ for some i. If δ_1 causes one pattern change for $\boldsymbol{v}^1$, then we put $(\boldsymbol{v}_0^2, \boldsymbol{v}_2)$ on a circle (e.g., Figure 1). If π is a split-pattern (i.e., $|p_i| > 1$), then we separate a new box (only labeled by $\lfloor \widehat{\boldsymbol{v}}_2 \rfloor(0)$) from the original box labeled by $\lfloor \widehat{\boldsymbol{v}}_2 \rfloor(0)$

and slide the new box backwards (i.e., clockwisely) for a small positive amount (taken as δ_2) without hitting any box or oval. If π is a merge-pattern (i.e., $|p_i| = 1$), then we slide the box labeled by $\lfloor \widehat{\boldsymbol{v}}_2 \rfloor(0)$ (this is the only label) backwards (i.e., clockwisely) for a positive amount Δ (taken as δ_2) until a box or an oval is hit. If δ_1 causes no pattern change for $\boldsymbol{v}^1$, then π must be a merge-pattern. In this case, δ_2 is any positive amount less than Δ. Take $\boldsymbol{v}^2 = \boldsymbol{v}_2 - \delta_2$. Obviously, $(\boldsymbol{v}_0^1, \boldsymbol{v}^1) \approx (\boldsymbol{v}_0^2, \boldsymbol{v}^2)$. It can be checked that (1.2) and (1.3) hold.

Any larger δ_1 that causes multiple pattern changes for $\boldsymbol{v}^1$ can be split into a finite sequence of small δ's such that each δ causes exactly one pattern change. This is because, as we mentioned earlier, $\boldsymbol{v}^1$ has at most $4(k+1)$ pattern changes for any $\delta \leq 1$. In this case, δ_2 can be calculated by working on each small δ (the last one first) as in the above proof.

(2). The case when $r = \emptyset$ is obvious. Assume that r contains only one element $j \in K^+$ and π is the pattern of $(\boldsymbol{v}_0^1, \boldsymbol{v}^1)$. A desired $\boldsymbol{v}^2$ is picked as follows. The integral parts of $\boldsymbol{v}^2$ are exactly those of $\boldsymbol{v}^1$; i.e., $\lceil \boldsymbol{v}^2 \rceil = \lceil \boldsymbol{v}^1 \rceil$. The fractional parts of $\boldsymbol{v}^2$ are exactly those of $\boldsymbol{v}_2$, except that $\lfloor \widehat{\boldsymbol{v}}^2 \rfloor(j)$ in the relative representation of $\boldsymbol{v}^2$ may be different from $\lfloor \widehat{\boldsymbol{v}}_2 \rfloor(j)$. Then what is $\lfloor \widehat{\boldsymbol{v}}^2 \rfloor(j)$? It is chosen such that the pattern of $(\boldsymbol{v}_0^2, \boldsymbol{v}^2)$ is π. For instance, if $\lfloor \widehat{\boldsymbol{v}}^1 \rfloor(j)$ equals to, say, $\lfloor \widehat{\boldsymbol{v}}_1 \rfloor(j_1)$ (resp. $\lfloor \widehat{\boldsymbol{v}}_0^1 \rfloor(j_1)$), for some j_1, then $\lfloor \widehat{\boldsymbol{v}}^2 \rfloor(j)$ is picked as $\lfloor \widehat{\boldsymbol{v}}_2 \rfloor(j_1)$ (resp. $\lfloor \widehat{\boldsymbol{v}}_0^2 \rfloor(j_1)$). If $\lfloor \widehat{\boldsymbol{v}}^1 \rfloor(j)$ lies strictly between, say, $\lfloor \widehat{\boldsymbol{v}}_1 \rfloor(j_1)$ (or, $\lfloor \widehat{\boldsymbol{v}}_0^1 \rfloor(j_1)$) and $\lfloor \widehat{\boldsymbol{v}}_1 \rfloor(j_2)$ (or, $\lfloor \widehat{\boldsymbol{v}}_0^1 \rfloor(j_2)$), for some j_1 and j_2, such that no other component in $\lfloor \widehat{\boldsymbol{v}}^1 \rfloor$ and $\lfloor \widehat{\boldsymbol{v}}_0^1 \rfloor$ lies strictly between these two values, then $\lfloor \widehat{\boldsymbol{v}}^2 \rfloor(j)$ is picked as any value lies strictly between $\lfloor \widehat{\boldsymbol{v}}_2 \rfloor(j_1)$ (or, $\lfloor \widehat{\boldsymbol{v}}_0^2 \rfloor(j_1)$) and $\lfloor \widehat{\boldsymbol{v}}_2 \rfloor(j_2)$ (or, $\lfloor \widehat{\boldsymbol{v}}_0^2 \rfloor(j_2)$) accordingly. Since $(\boldsymbol{v}_0^1, \boldsymbol{v}_1) \approx (\boldsymbol{v}_0^2, \boldsymbol{v}_2)$, one can show that $\lfloor \widehat{\boldsymbol{v}}^2 \rfloor(j)$ can always be picked. The choice of $\lfloor \widehat{\boldsymbol{v}}^2 \rfloor(j)$ guarantees that the pattern of $(\boldsymbol{v}_0^1, \boldsymbol{v}^1)$ is the same as the pattern of $(\boldsymbol{v}_0^2, \boldsymbol{v}^2)$. The rest of the conditions in (2) can be checked easily.

For the case when r contains more than one element, the above proof can be generalized by resetting clocks in r one by one. ∎

5 Pushdown Timed Automata

A *pushdown timed automaton* (PTA) $\mathcal{A}$ is a tuple

$$\langle S, \{x_1, \cdots, x_k\}, Inv, R, \Gamma, PD \rangle,$$

where

- S is a finite set of *states*,
- $x_1, \cdots, x_k$ are (dense) clocks,
- $Inv : S \to \mathcal{C}$ assigns a clock constraint over clocks $x_1, \cdots, x_k$, called an *invariant*, to each state,
- $R : S \times S \to \mathcal{C} \times 2^{\{x_1, \cdots, x_k\}}$ assigns a clock constraint over clocks $x_1, \cdots, x_k$, called a *reset condition*, and a subset of clocks, called clock resets, to a directed edge in $S \times S$,
- Γ is the *stack alphabet*. $PD : S \times S \to \Gamma \times \Gamma^*$ assigns a pair (a, γ) with $a \in \Gamma$ and $\gamma \in \Gamma^*$, called a *stack operation*, to each edge in $S \times S$. A stack operation (a, γ) replaces the top symbol a of the stack with a string (possibly empty) in Γ^*.

A *timed automaton* is a PTA without the pushdown stack.

The semantics of $\mathcal{A}$ is defined as follows. A *configuration* is a triple $(s, \boldsymbol{v}, w)$ of a state s, a clock valuation $\boldsymbol{v}$ on $x_0, \cdots, x_k$ (where x_0 is the auxiliary clock), and a stack word $w \in \Gamma^*$. $(s_1, \boldsymbol{v}_1, w_1) \to_{\mathcal{A}} (s_2, \boldsymbol{v}_2, w_2)$ denotes a *one-step transition* of $\mathcal{A}$ if one of the following conditions is satisfied:

- (*a progress transition*) $s_1 = s_2$, $w_1 = w_2$, and $\exists 0 < \delta \in \mathbf{D}^+$, $\boldsymbol{v}_2 = \boldsymbol{v}_1 + \delta$ and for all δ' satisfying $0 \leq \delta' \leq \delta$, $\boldsymbol{v}_1 + \delta' \in Inv(s_1)$. That is, a progress transition makes all the clocks synchronously progress by amount $\delta > 0$, during which the invariant is consistently satisfied, while the state and the stack content remain unchanged.
- (*a reset transition*) $\boldsymbol{v}_1 \in Inv(s_1) \wedge c$, $\boldsymbol{v}_1 \downarrow_r = \boldsymbol{v}_2 \in Inv(s_2)$, and $w_1 = aw, w_2 = \gamma w$ for some $w \in \Gamma^*$, where $R(s_1, s_2) = (c, r)$ for some clock constraint c and clock resets r, and $PD(s_1, s_2) = (a, \gamma)$ for some stack symbol $a \in \Gamma$ and string $\gamma \in \Gamma^*$. That is, a reset transition, by moving from state s_1 to state s_2, resets every clock in r to 0 and keeps all the other clocks unchanged. The stack content is modified according to the stack operation (a, γ) given on (s_1, s_2). Clock values before the transition satisfy the invariant $Inv(s_1)$ and the reset condition c; clock values after the transition satisfy the invariant $Inv(s_2)$. [1]

We write $\to^*_{\mathcal{A}}$ to be the transitive closure of $\to_{\mathcal{A}}$. Given two valuations $\boldsymbol{v}_0^1$ and $\boldsymbol{v}_1$, two states s_0 and s_1, and two stack words w_0 and w_1, assume the auxiliary clock x_0 starts from 0, i.e., $\boldsymbol{v}_0^1$ is initial. The following result is surprising. It states that, for **any** initialized pair $(\boldsymbol{v}_0^2, \boldsymbol{v}_2)$ with $(\boldsymbol{v}_0^1, \boldsymbol{v}_1) \approx (\boldsymbol{v}_0^2, \boldsymbol{v}_2)$, $(s_0, \boldsymbol{v}_0^1, w_0) \to^*_{\mathcal{A}} (s_1, \boldsymbol{v}_1, w_1)$ if and only if $(s_0, \boldsymbol{v}_0^2, w_0) \to^*_{\mathcal{A}} (s_1, \boldsymbol{v}_2, w_1)$. This result implies that, from the definition of $\approx$, for any fixed s_0, s_1, w_0 and w_1, the pattern of $(\lfloor \boldsymbol{v}_0^1 \rfloor, \lfloor \boldsymbol{v}_1 \rfloor)$ (instead of the actual values of $\lfloor \boldsymbol{v}_0^1 \rfloor$ and $\lfloor \boldsymbol{v}_1 \rfloor$), the integral values $\lceil \boldsymbol{v}_0^1 \rceil$, and the integral values $\lceil \boldsymbol{v}_1 \rceil$ are sufficient to determine whether $(s_0, \boldsymbol{v}_0^1, w_0)$ can reach $(s_1, \boldsymbol{v}_1, w_1)$ in $\mathcal{A}$.

Lemma 6. *Let $\mathcal{A}$ be a PTA. For any states s_0 and s_1, any two initial clock valuations $\boldsymbol{v}_0^1$ and $\boldsymbol{v}_0^2$, any two clock valuations $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$, and any two stack words w_0 and w_1, if $(\boldsymbol{v}_0^1, \boldsymbol{v}_1) \approx (\boldsymbol{v}_0^2, \boldsymbol{v}_2)$, then,*

$$(s_0, \boldsymbol{v}_0^1, w_0) \to^*_{\mathcal{A}} (s_1, \boldsymbol{v}_1, w_1) \textit{ iff } (s_0, \boldsymbol{v}_0^2, w_0) \to^*_{\mathcal{A}} (s_1, \boldsymbol{v}_2, w_1).$$

Proof. Lemma 5 already gives the result, but for $\to_{\mathcal{A}}$ instead of $\to^*_{\mathcal{A}}$, noticing that Lemma 5 guarantees that tests (and obviously stack operations) are consistent in $(s_0, \boldsymbol{v}_0^1, w_0) \to_{\mathcal{A}} (s_1, \boldsymbol{v}_1, w_1)$ and in $(s_0, \boldsymbol{v}_0^2, w_0) \to_{\mathcal{A}} (s_1, \boldsymbol{v}_2, w_1)$. An induction (on the length of $\to^*_{\mathcal{A}}$) can be used to show the lemma, by working from $(s_1, \boldsymbol{v}_1, w_1)$ back to $(s_0, \boldsymbol{v}_0^1, w_0)$. ∎

Example 5. It is the time to show an example to convince the reader that Lemma 6 indeed works. Consider a timed automaton $\mathcal{A}$ shown in Figure 3. Let $\boldsymbol{v}_0^1 = (0, 4.98, 2.52)$, $\boldsymbol{v}_3^1 = (5.36, 2.89, 7.88)$. $(s_1, \boldsymbol{v}_0^1) \to^*_{\mathcal{A}} (s_2, \boldsymbol{v}_3^1)$ is witnessed by: $(s_1, \boldsymbol{v}_0^1) \to_{\mathcal{A}}$ (progress

[1] In the definition, we do not have a stack operation on a progress transition. In fact, a translation can be worked out by expressing any PTA with stack operations on progress transitions in a PTA defined in this paper. Since we focus on the clock/stack behaviors of a PTA, instead of the ω-language accepted by it, input symbols are not considered in our definition.

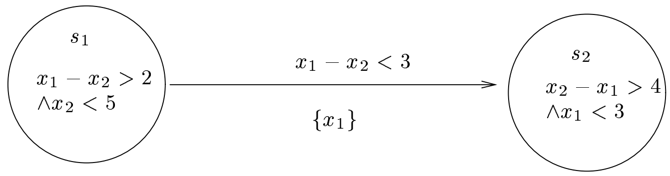

Fig. 3. An example timed automaton $\mathcal{A}$.

by 2.47 at s_1) $(s_1, \boldsymbol{v}_1^1) \to_{\mathcal{A}}$ (reset x_1 and transit to s_2) $(s_2, \boldsymbol{v}_2^1) \to_{\mathcal{A}}$ (progress by 2.89 at s_2) $(s_2, \boldsymbol{v}_3^1)$. Take a new pair $\boldsymbol{v}_0^2 = (0, 4.89, 2.11)$ and $\boldsymbol{v}_3^2 = (5.28, 2.77, 7.39)$. It is easy to check $(\boldsymbol{v}_0^1, \boldsymbol{v}_3^1) \approx (\boldsymbol{v}_0^2, \boldsymbol{v}_3^2)$. From Lemma 6, $(s_1, \boldsymbol{v}_0^2) \to_{\mathcal{A}}^* (s_2, \boldsymbol{v}_3^2)$. Indeed, this is witnessed by $(s_1, \boldsymbol{v}_0^2) \to_{\mathcal{A}}$ (progress by 2.51 at s_1) $(s_1, \boldsymbol{v}_1^2) \to_{\mathcal{A}}$ (reset x_1 and transit to s_2) $(s_2, \boldsymbol{v}_2^2) \to_{\mathcal{A}}$ (progress by 2.77 at s_2) $(s_2, \boldsymbol{v}_3^2)$. These two witnesses differ slightly (2.47 and 2.89, vs. 2.51 and 2.77). We choose 2.77 and 2.51 by looking at the first witness backwardly. That is, $\boldsymbol{v}_2^2$ is picked such that $(\boldsymbol{v}_0^2, \boldsymbol{v}_2^2) \approx (\boldsymbol{v}_0^1, \boldsymbol{v}_2^1)$. Then, $\boldsymbol{v}_1^2$ is picked such that $(\boldsymbol{v}_0^2, \boldsymbol{v}_1^2) \approx (\boldsymbol{v}_0^1, \boldsymbol{v}_1^1)$. The existence of $\boldsymbol{v}_2^2$ and $\boldsymbol{v}_1^2$ is guaranteed by Lemma 5. Finally, according to Lemma 5 again, $\boldsymbol{v}_1^2$ is able to go back to $\boldsymbol{v}_0^2$. This is because $\boldsymbol{v}_1^1$ goes back to $\boldsymbol{v}_0^1$ through a one-step transition and $\boldsymbol{v}_0^1$ is initial. ∎

Now, we express $\to_{\mathcal{A}}^*$ in a form treating the integral parts and the fractional parts of clock values separately. For any pattern $\pi \in \Pi$, any discrete valuations $\boldsymbol{u}_0$ and $\boldsymbol{u}_1$, and any stack words w_0 and w_1, define $(s_0, \boldsymbol{u}_0, w_0) \to_{\mathcal{A},\pi}^* (s_1, \boldsymbol{u}_1, w_1)$ to be

$$\exists \boldsymbol{v}_0 \exists \boldsymbol{v}_1 (\boldsymbol{v}_0(0) = 0 \wedge \lceil \boldsymbol{v}_0 \rceil = \boldsymbol{u}_0 \wedge \lceil \boldsymbol{v}_1 \rceil = \boldsymbol{u}_1$$
$$\wedge (\boldsymbol{v}_0, \boldsymbol{v}_1) \in \pi \wedge (s_0, \boldsymbol{v}_0, w_0) \to_{\mathcal{A}}^* (s_1, \boldsymbol{v}_1, w_1)).$$

Lemma 7. *Let $\mathcal{A}$ be a PTA. For any states s_0 and s_1, any initialized pair $(\boldsymbol{v}_0, \boldsymbol{v}_1)$, and any stack words w_0 and w_1, $(s_0, \boldsymbol{v}_0, w_0) \to_{\mathcal{A}}^* (s_1, \boldsymbol{v}_1, w_1)$ iff*

$$\bigvee_{\pi \in \Pi} ((\lfloor \boldsymbol{v}_0 \rfloor, \lfloor \boldsymbol{v}_1 \rfloor) \in \pi \wedge (s_0, \lceil \boldsymbol{v}_0 \rceil, w_0) \to_{\mathcal{A},\pi}^* (s_1, \lceil \boldsymbol{v}_1 \rceil, w_1)).$$

Proof. ($\Rightarrow$) is immediate.

($\Leftarrow$) uses the following observation (from the definition of $\to_{\mathcal{A},\pi}^*$ and Lemma 6): for any pattern π, $(\lfloor \boldsymbol{v}_0 \rfloor, \lfloor \boldsymbol{v}_1 \rfloor) \in \pi \wedge (s_0, \lceil \boldsymbol{v}_0 \rceil, w_0) \to_{\mathcal{A},\pi}^* (s_1, \lceil \boldsymbol{v}_1 \rceil, w_1)$ implies $(\lfloor \boldsymbol{v}_0 \rfloor, \lfloor \boldsymbol{v}_1 \rfloor) \in \pi \wedge (s_0, \lceil \boldsymbol{v}_0 \rceil + \lfloor \boldsymbol{v}_0 \rfloor, w_0) \to_{\mathcal{A}}^* (s_1, \lceil \boldsymbol{v}_1 \rceil + \lfloor \boldsymbol{v}_1 \rfloor, w_1)$. ∎

Once we give a characterization of $\to_{\mathcal{A},\pi}^*$, Lemma 7 immediately gives a characterization for $\to_{\mathcal{A}}^*$. Fortunately, the characterization of $\to_{\mathcal{A},\pi}^*$ is a decidable one, as shown in the next section.

6 The Pattern Graph of a Timed Pushdown Automaton

Let $\mathcal{A} = \langle S, \{x_1, \cdots, x_k\}, Inv, R, \Gamma, PD \rangle$ be a PTA specified in the previous section. The *pattern graph* G of $\mathcal{A}$ is a tuple

$$\langle S \times \Pi, \{y_0, \cdots, y_k\}, E, \mathcal{A} \rangle$$

where

- S is the states in $\mathcal{A}$ and Π is the set of all patterns,
- *Discrete clocks* $y_0, \cdots, y_k$ are the integral parts of the clocks $x_0, \cdots, x_k$ in $\mathcal{A}$,
- $E \subseteq S \times \Pi \times \{\mathrm{STAY}, \mathrm{PROG}, \mathrm{RESET}\} \times S \times \Pi$ is a set of (labeled) edges. For all $(s, \pi), (s', \pi') \in S \times \Pi, l \in \{\mathrm{STAY}, \mathrm{PROG}, \mathrm{RESET}\}$, $\langle (s, \pi), l, (s', \pi') \rangle$ is in E iff one of the followings is true:
  - (a stay edge) l is STAY, $s = s'$, and, $\pi' = \pi$ is a merge pattern.
  - (a progress edge) $l = \mathrm{PROG}$, $s = s'$, and $\pi' = Next(\pi)$.
  - (a reset edge) $l = \mathrm{RESET}$, and $\pi' = Reset_r(\pi)$, where $R(s, s') = (c, r)$ for some c and r.

  A stay edge corresponds to progress transitions in $\mathcal{A}$ that cause no pattern change. (Notice that a progress transition in $\mathcal{A}$ causes no pattern change only from a merge pattern.) A progress edge corresponds to progress transitions in $\mathcal{A}$ that cause one pattern change. A reset edge corresponds to a reset transition in $\mathcal{A}$.

A configuration of G is a tuple $(s, \pi, \boldsymbol{u}, w)$ of state $s \in S$, pattern $\pi \in \Pi$, discrete valuation $\boldsymbol{u} \in \mathbf{N}^{k+1}$ and stack word $w \in \Gamma^*$. $(s, \pi, \boldsymbol{u}, w) \to^e (s', \pi', \boldsymbol{u}', w')$ denotes a *one-step transition* through edge $e \in E$ of G if one of the followings is true:

- e is a stay edge $\langle (s, \pi), \mathrm{STAY}, (s, \pi) \rangle$, $s' = s$, $\pi' = \pi$, $\boldsymbol{u}' = \boldsymbol{u} \in Inv(s)^\pi$, and $w' = w$.
- e is a progress edge $\langle (s, \pi), \mathrm{PROG}, (s, \pi') \rangle$, $s' = s$, $(\pi', \boldsymbol{u}') = next(\pi, \boldsymbol{u})$, $\boldsymbol{u} \in Inv(s)^\pi$, $\boldsymbol{u}' \in Inv(s)^{\pi'}$, and $w' = w$. We say that y_i, $0 \le i \le k$, progresses on e if $\boldsymbol{u}'(i) = \boldsymbol{u}(i) + 1$.
- e is a reset edge $\langle (s, \pi), \mathrm{RESET}, (s', \pi') \rangle$, $\boldsymbol{u} \in (c \wedge Inv(s))^\pi$, $\boldsymbol{u}' \in Inv(s')^{\pi'}$, $reset_r(\pi, \boldsymbol{u}) = (\pi', \boldsymbol{u}')$ and $w = aw'', w' = \gamma w''$ for some $w'' \in \Gamma^*$, where $R(s, s') = (c, r)$ and $PD(s, s') = (a, \gamma)$ for some c, r, a and γ. Hence, w changes to w' according to the stack operation.

In above, $Inv(s)^\pi$, $Inv(s)^{\pi'}$, $(c \wedge Inv(s))^\pi$, and $Inv(s')^{\pi'}$ are called *tests* in G. From Section 4, the tests are clock constraints over discrete clocks $y_1, \cdots, y_k$. We write $(s, \pi, \boldsymbol{u}, w) \to_G (s', \pi', \boldsymbol{u}', w')$ if $(s, \pi, \boldsymbol{u}, w) \to^e (s', \pi', \boldsymbol{u}', w')$ for some e. The binary reachability $\to_G^*$ of G is the transitive closure of $\to_G$.

If π is the pattern of $(\boldsymbol{v}_0, \boldsymbol{v}_1)$, we use $init(\pi)$ to denote the pattern of $(\boldsymbol{v}_0, \boldsymbol{v}_0)$. $init(\pi)$ is unique for each π. We first show that G faithfully simulates $\mathcal{A}$ when the fractional parts of dense clocks are abstracted away by a pattern.

Lemma 8. *Let $\mathcal{A}$ be a PTA with pattern graph G. For any states s_0 and s_1 in S, any pattern $\pi \in \Pi$, any stack words w_0 and w_1 in Γ^*, and any discrete valuation pairs $(\boldsymbol{u}_0, \boldsymbol{u}_1)$ with $\boldsymbol{u}_0(0) = 0$, we have,*

$$(s_0, \boldsymbol{u}_0, w_0) \to_{\mathcal{A},\pi}^* (s_1, \boldsymbol{u}_1, w_1) \text{ \textit{iff} } (s_0, init(\pi), \boldsymbol{u}_0, w_0) \to_G^* (s_1, \pi, \boldsymbol{u}_1, w_1).$$

Proof. Fix any states $s_0, s_1 \in S$, any pattern $\pi \in \Pi$, any stack words w_0 and w_1 in Γ^*, and any discrete valuation pairs $(\boldsymbol{u}_0, \boldsymbol{u}_1)$ with $\boldsymbol{u}_0(0) = 0$.
($\Rightarrow$). By the definition of $(s_0, \boldsymbol{u}_0, w_0) \to_{\mathcal{A},\pi}^* (s_1, \boldsymbol{u}_1, w_1)$, there exists an initialized pair $(\boldsymbol{v}_0, \boldsymbol{v}_1)$ such that

- $(\boldsymbol{v}_0, \boldsymbol{v}_1)$ has pattern π,
- $\lceil \boldsymbol{v}_0 \rceil = \boldsymbol{u}_0, \lceil \boldsymbol{v}_1 \rceil = \boldsymbol{u}_1$,
- $(s_0, \boldsymbol{v}_0, w_0) \to^*_{\mathcal{A}} (s_1, \boldsymbol{v}_1, w_1)$.

In order to show that $(s_0, [(\boldsymbol{v}_0, \boldsymbol{v}_0)], \lceil \boldsymbol{v}_0 \rceil, w_0) \to^*_G (s_1, [(\boldsymbol{v}_0, \boldsymbol{v}_1)], \lceil \boldsymbol{v}_1 \rceil, w_1)$ (notice that $init(\pi) = [(\boldsymbol{v}_0, \boldsymbol{v}_0)]$), it suffices to show that each one-step transition in $\mathcal{A}$ can be simulated by $\to^*_G$ properly: for any valuations $\boldsymbol{v}, \boldsymbol{v}'$, any states s and s', and any stack words w and w', if $(s, \boldsymbol{v}, w) \to_{\mathcal{A}} (s', \boldsymbol{v}', w')$ then $(s, [(\boldsymbol{v}_0, \boldsymbol{v})], \lceil \boldsymbol{v} \rceil, w) \to^*_G (s', [(\boldsymbol{v}_0, \boldsymbol{v}')], \lceil \boldsymbol{v}' \rceil, w')$.

Case 1. For any valuation $\boldsymbol{v}$ and state s, consider a progress transition in $\mathcal{A}$, $(s, \boldsymbol{v}, w) \to_{\mathcal{A}} (s, \boldsymbol{v} + \delta, w')$, $\delta > 0$, such that (by definition) $w = w'$, and $\forall 0 \leq \delta' \leq \delta, \boldsymbol{v} + \delta' \in Inv(s)$. Let π_0 be the pattern of $(\boldsymbol{v}_0, \boldsymbol{v})$. If $\boldsymbol{v}$ has no pattern change for δ, then π_0 must be a merge-pattern. This progress transition in $\mathcal{A}$ can therefore be simply simulated by the stay edge $\langle (s, \pi_0), \mathrm{STAY}, (s, \pi_0) \rangle$ in G. If, however, $\boldsymbol{v}$ has at least one pattern change for δ, let the pattern ring of π_0 be $\pi_0, \cdots, \pi_m = \pi_0$. This progress transition in $\mathcal{A}$ can be simulated by the following path of progress edges in G: looping along

$$\langle (s, \pi_0), \mathrm{PROG}, (s, \pi_1) \rangle, \cdots, \langle (s, \pi_{m-1}), \mathrm{PROG}, (s, \pi_m = \pi_0) \rangle$$

for $\lceil \delta \rceil$ times, followed by a prefix of the loop ended with (s, π_i), for some i, with π_i being the pattern of $(\boldsymbol{v}_0, \boldsymbol{v} + \delta)$. From Lemma 3, it is not hard to show $(s, \pi_0, \lceil \boldsymbol{v} \rceil, w) \to^*_G (s, \pi_i, \lceil \boldsymbol{v} + \delta \rceil, w)$ through the path in G, noticing that tests for $Inv(s)$ are consistent in $\mathcal{A}$ and in G, and the stack word does not change for progress transitions in both $\mathcal{A}$ and G.

Case 2. For any valuation $\boldsymbol{v}$ and states s and s', consider a reset transition $(s, \boldsymbol{v}, w) \to_{\mathcal{A}} (s', \boldsymbol{v} \downarrow_r, w')$ in $\mathcal{A}$ such that (by definition) $w = aw'', w' = \gamma w''$ for some w'' with $PD(s, s') = (a, \gamma)$, $R(s, s') = (c, r)$ and $\boldsymbol{v} \in Inv(s) \wedge c$, $\boldsymbol{v} \downarrow_r \in Inv(s')$. Assume that the pattern of $(\boldsymbol{v}_0, \boldsymbol{v})$ is π_0 and the pattern of $(\boldsymbol{v}_0, \boldsymbol{v} \downarrow_r)$ is π'_0. This reset transition in $\mathcal{A}$ corresponds to the reset edge in G: $\langle (s, \pi_0), \mathrm{RESET}, (s', \pi'_0) \rangle$. From Lemma 4, it can be established $(s, \pi_0, \lceil \boldsymbol{v} \rceil, w) \to^*_G (s', \pi'_0, \lceil \boldsymbol{v} \downarrow_r \rceil, w')$ through this edge, noticing that tests for $Inv(s) \wedge c$ and $Inv(s')$ are consistent in $\mathcal{A}$ and G, and the stack operations are the same in $\mathcal{A}$ and G.

($\Leftarrow$). Suppose that $(s_0, init(\pi), \boldsymbol{u}_0, w_0) \to^*_G (s_1, \pi, \boldsymbol{u}_1, w_1)$. We would like to show

$$(s_0, \boldsymbol{u}_0, w_0) \to^*_{\mathcal{A}, \pi} (s_1, \boldsymbol{u}_1, w_1).$$

Pick any initial valuation $\boldsymbol{v}_0$ such that $(\boldsymbol{v}_0, \boldsymbol{v}_0)$ has pattern $init(\pi)$ and $\lceil \boldsymbol{v}_0 \rceil = \boldsymbol{u}_0$. Suppose that $(s^0, \pi_0, \boldsymbol{u}^0, w^0) \to^{e_1} \cdots \to^{e_m} (s^m, \pi_m, \boldsymbol{u}^m, w^m)$ is a path (in G) witnessing $(s_0, init(\pi), \boldsymbol{u}_0, w_0) \to^*_G (s_1, \pi, \boldsymbol{u}_1, w_1)$ through edges $e_1, \cdots, e_m$ such that

$$(s^0, \pi_0, \boldsymbol{u}^0, w^0) = (s_0, init(\pi), \boldsymbol{u}_0, w_0)$$

and

$$(s^m, \pi_m, \boldsymbol{u}^m, w^m) = (s_1, \pi, \boldsymbol{u}_1, w_1).$$

A path in $\mathcal{A}$

$$(s^0, \boldsymbol{v}^0, w^0) \to^{t_1} \cdots \to^{t_m} (s^m, \boldsymbol{v}^m, w^m)$$

is constructed as follows, where $\boldsymbol{v}^0 = \boldsymbol{v}_0$ and each transition t_i in $\mathcal{A}$ corresponds to each edge e_i in G. From $i = 1$ to m, each e_i belongs to one of the following three cases:

Case 1. e_i is a progress edge in G. Then, $next(\pi_{i-1}, \boldsymbol{u}^{i-1}) = (\pi_i, \boldsymbol{u}^i)$, $w^i = w^{i-1}$, and $s^{i-1} = s^i$. We pick t_i to be a progress transition (at state s^{i-1}) in $\mathcal{A}$ from $\boldsymbol{v}^{i-1}$ with an amount of $\delta > 0$ that causes exactly one pattern change. Take $\boldsymbol{v}^i = \boldsymbol{v}^{i-1} + \delta$. Notice that both the progress edge and the progress transition do not change the stack content, i.e., $w^i = w^{i-1}$.

Case 2. e_i is a stay edge in G. Then, $\pi_{i-1} = \pi_i$. π_{i-1} must be a merge-pattern with $w^i = w^{i-1}$ and $s^{i-1} = s^i$. We pick t_i to be a progress transition (at state s^{i-1}) in $\mathcal{A}$ from $\boldsymbol{v}^{i-1}$ with an amount of $\delta > 0$ that causes no pattern change. This δ always exists since the pattern π_{i-1} of $(\boldsymbol{v}^0, \boldsymbol{v}^{i-1})$ is a merge-pattern. Similarly to Case 1, $w^i = w^{i-1}$.

Case 3. e_i is a reset edge from state s^{i-1} to state s^i with clock resets r in G. Then, t_i is the reset transition from state s^{i-1} to state s^i with clock resets r in $\mathcal{A}$. Notice that both e_i and t_i have the same stack operation. Take $\boldsymbol{v}^i = \boldsymbol{v}^{i-1} \downarrow r$ and w^i is the result of the stack operation on w^{i-1}.

Notice that, for each $i = 1 \cdots m$,

- $(\boldsymbol{v}_0, \boldsymbol{v}^i)$ has pattern π_i,
- $\lceil \boldsymbol{v}^i \rceil = \boldsymbol{u}^i$.

This can be shown using Lemma 3 for Case 1, the definition of "no pattern change" for Case 2, and Lemma 4 for Case 3. Therefore, this constructed path of $\mathcal{A}$ keeps the exactly the same patterns and integral parts of clocks as well as the stack word as in the path for G. Clock tests (and obviously the stack operations) are consistent between the path in G and the constructed path in $\mathcal{A}$. Hence, $(s_0, \boldsymbol{u}_0, w_0) \to^*_{\mathcal{A},\pi} (s_1, \boldsymbol{u}_1, w_1)$ since, by taking $\boldsymbol{v}_1 = \boldsymbol{v}^m$,

- $(\boldsymbol{v}_0, \boldsymbol{v}_1)$ has pattern π,
- $\lceil \boldsymbol{v}_0 \rceil = \boldsymbol{u}_0$, $\lceil \boldsymbol{v}_1 \rceil = \boldsymbol{u}_1$,
- $(s_0, \boldsymbol{v}_0, w_0) \to^*_{\mathcal{A}} (s_1, \boldsymbol{v}_1, w_1)$.

∎

Before we proceed further to show that the binary reachability $\to^*_G$ of G is NPCA, we point out a property of G. Let y_i and y_j ($1 \le i, j \le k$) be two discrete clocks. Suppose that $(s, \pi, \boldsymbol{u}, w) \to^*_G (s', \pi', \boldsymbol{u}', w')$ through a sequence τ of one-step transitions during which y_i and y_j do not reset. Then, the absolute value of the difference between the net increment made toward y_i on τ and the one toward y_j is bounded by 1; i.e.,

$$(*) \qquad |(\boldsymbol{u}'(i) - \boldsymbol{u}(i)) - (\boldsymbol{u}'(j) - \boldsymbol{u}(j))| \le 1.$$

The reason is as follows. According to the definition of G, if i^1 and j^1 are in the same position in π, then y_i and y_j progress (i.e., $y_i := y_i + 1$ and $y_j := y_j + 1$) at the same time on τ. If i^1 and j^1 are in different positions in π, then y_i and y_j progress alternately on τ. Either case will give (*).

Recall that, in an NPCM, each counter can add 1, subtract 1, or stay unchanged. Those counter assignments are called *standard assignments*. The NPCM can also test whether a counter is equal to, greater than, or less than an integer constant. Those tests are called *standard tests*. G can be considered as an NPCM where discrete clocks $y_0, \cdots, y_k$ are treated as counters. However, carefully looking at the definition of G, we notice that, in addition to standard assignments $y_i := y_i + 1$ ($0 \le i \le k$), G also has nonstandard assignments $y_i := 0$ ($1 \le i \le k$). Moreover, the tests in G are in the form of Boolean combinations of $y_i \sim d$ and $y_i - y_j \sim d$ (wlog, $1 \le i \le j \le k$), which are not standard tests. A special case of G is that $y_1, \cdots, y_k$ always progress at the same time (hence, in (*), the bound is 0 instead of 1). Under this special case, a technique presented in [20, 22] can be directly used to replace the tests in G with finite table look-ups. In the following proof, the technique is modified to handle the general case of G. That is, $\rightarrow_G^*$ can be accepted by a reversal-bounded NPCA using standard tests and nonstandard assignments. Then, we show that the nonstandard assignments can be made standard and the counters are reversal-bounded.

Lemma 9. *For any PTA $\mathcal{A}$, the binary reachability $\rightarrow_G^*$ of the pattern graph G of $\mathcal{A}$ is NPCA. In particular, if $\mathcal{A}$ is a timed automaton, then the binary reachability $\rightarrow_G^*$ is Presburger.*

Proof. We construct the NPCA M that accepts $\rightarrow_G^*$. M is given a pair of string encodings of configurations $(s, \pi, \boldsymbol{u}, w)$ and $(s', \pi', \boldsymbol{u}', w')$ (separated by a delimiter not in the stack alphabet) on its one-way input tape. In the encodings, s, s', π, and π' are treated as integers in a bounded range. In particular, the stack word w' is reversed in the encoding (the reason will be made clear in a moment). On reading $(s, \pi, \boldsymbol{u}, w)$, M remembers s and π in its finite control, and copies $\boldsymbol{u}$ and w into its $k+1$ counters (we still use $y_0, \cdots, y_k$ to denote them) and the stack, respectively. Thus, M's input head stops at the beginning of $(s', \pi', \boldsymbol{u}', w')$. M starts simulating G from configuration $(s, \pi, \boldsymbol{u}, w)$ as follows with the stack operations in G being exactly simulated on its own stack.

Tests in G are Boolean combinations of $y_i \sim d$ and $y_i - y_j \sim d$ for $1 \le i \le j \le k$. Assume that m is two plus the maximal absolute value of all the d's that appear in the tests of G. For each $1 \le i \le j \le k$, let entry a_{ij} (resp. b_i) be a finite state variable in $\{-m, \cdots, 0, \cdots, m\}$ (resp. in $\{0, \cdots, m\}$). In the following, we demonstrate a technique to eliminate the tests in G, based on the definition of a finite table lookup $a_{ij} \sim d$ and $b_i \sim d$ to replace tests $y_i - y_j \sim d$ and $y_i \sim d$.

The initial values of the entries are constructed directly from the values $\boldsymbol{u}$ in configuration $(s, \pi, \boldsymbol{u}, w)$ on the input tape, for each $1 \le i \le j \le k$:

- $a_{ij} := \boldsymbol{u}(i) - \boldsymbol{u}(j)$ if $|\boldsymbol{u}(i) - \boldsymbol{u}(j)| < m$,
- $a_{ij} := m$ if $\boldsymbol{u}(i) - \boldsymbol{u}(j) \ge m$,
- $a_{ij} := -m$ if $\boldsymbol{u}(i) - \boldsymbol{u}(j) \le -m$,
- $b_i := \boldsymbol{u}(i)$ if $\boldsymbol{u}(i) < m$,
- $b_i := m$ if $\boldsymbol{u}(i) \ge m$.

The procedure for updating the entries is given below, in which "$\oplus 1$" means adding one if the result does not exceed m, else it keeps the same value. "$\ominus 1$" means subtracting one if the result is not less than $-m$, else it keeps the same value. We modify G as follows. Let e be an edge of G. If e is a stay edge, the entries remain unchanged on e. If

e is a progress edge, we use λ to denote all the y_i's that progress on e. In the case, the entries are updated by adding the following instructions to e, for each $1 \leq i \leq j \leq k$:

- $a_{ij} := a_{ij}$ if $y_i \in \lambda$ and $y_j \in \lambda$, or, $y_i \notin \lambda$ and $y_j \notin \lambda$,
- $a_{ij} := a_{ij} \oplus 1$ if $y_i \in \lambda$ and $y_j \notin \lambda$,
- $a_{ij} := a_{ij} \ominus 1$ if $y_i \notin \lambda$ and $y_j \in \lambda$,
- $b_i := b_i$ if $y_i \notin \lambda$,
- $b_i := b_i \oplus 1$ if $y_i \in \lambda$.

If e is a reset edge, we use r to denote all the y_i's that are reset on e. In the case, the entries are updated by adding the following instructions to e, for each $1 \leq i \leq j \leq k$:

- $a_{ij} := 0$ if $y_i \in r$ and $y_j \in r$,
- $a_{ij} := -b_j$ if $y_i \in r$ and $y_j \notin r$,
- $a_{ij} := b_i$ if $y_i \notin r$ and $y_j \in r$,
- $a_{ij} := a_{ij}$ if $y_i \notin r$ and $y_j \notin r$,

followed by adding the following instructions, for each $1 \leq i \leq k$:

- $b_i := b_i$ if $y_i \notin r$,
- $b_i := 0$ if $y_i \in r$.

By adding the above entry updating instructions to G, a configuration of G is now augmented with entry values. The configuration is *valid* if its counter values and entry values satisfy,

(1). for all $1 \leq i \leq j \leq k$ and for each integer $-(m-2) \leq d \leq m-2$,

$$y_i - y_j \sim d \text{ iff } a_{ij} \sim d,$$

(2). for all $1 \leq i \leq k$ and for each integer $-(m-1) \leq d \leq m-1$,

$$y_i \sim d \text{ iff } b_i \sim d.$$

That is, tests $y_i - y_j \sim d$ and $y_i \sim d$ can be replaced by $a_{ij} \sim d$ and $b_i \sim d$, respectively. (Recall that m is chosen such that $m-2$ is greater than or equal to the absolute value of any constant d in the tests of G.) Consider an execution of length t, $\alpha^0 \rightarrow_G \cdots \rightarrow_G \alpha^t$, in which α^0 is the initial configuration $(s, \pi, \boldsymbol{u}, w)$ (on the input tape) augmented with the initial entry values. The execution is *valid* if each α_i ($1 \leq i \leq t$) is valid. We will show,

> **Claim.** After adding the above entry updating instructions to G, any execution of G from α^0 is valid.

Proof of the Claim. We prove it by induction on length t. Obviously, α^0 is valid; i.e., the Claim holds for $t = 0$. Suppose that the Claim holds for t. Now, consider an execution of length $t+1$: $\alpha^0 \rightarrow_G \cdots \rightarrow_G \alpha^t \rightarrow_G \alpha^{t+1}$, in which $\alpha^0, \cdots, \alpha^t$ are all valid. It suffices for us to show that α^{t+1} is valid. Assume that e is the edge witnessing $\alpha^t \rightarrow_G \alpha^{t+1}$. We only deal with four special cases that would make α^{t+1} violate (1); the rest cases, which are omitted here, are completely analogous to a similar proof presented in [18,20]. Fix any $1 \le i < j \le k$. e is a progress edge with λ being the set of discrete clocks that progress on e. The cases are:

Case 1. On e, $y_i \notin \lambda$ and $y_j \in \lambda$. In α^t, $a_{ij} = m-1$ and $y_i - y_j \ge m$.
Case 2. On e, $y_i \notin \lambda$ and $y_j \in \lambda$. In α^t, $a_{ij} = m$ and $y_i - y_j = m-1$.
Case 3. On e, $y_i \in \lambda$ and $y_j \notin \lambda$. In α^t, $a_{ij} = -(m-1)$ and $y_i - y_j \le -m$.
Case 4. On e, $y_i \in \lambda$ and $y_j \notin \lambda$. In α^t, $a_{ij} = -m$ and $y_i - y_j = -(m-1)$.

Each of the cases makes α^{t+1} invalid. For instance, under Case 1, according to the entry updating instructions, $a_{ij} = m-2$ and $y_i - y_j \ge m-1$ in α^{t+1}. Take $d = m-2$, we have $a_{ij} = d$ but $y_i - y_j \ne d$, which contradicts (1). Fortunately, none of the four cases is possible. We only show Case 1; the rest are similar. Suppose that Case 1 is true. For convenience, we shall use a^t_{ij}, b^t_i, y^t_i to denote the values of a_{ij}, b_i, y_i in configuration α^t. t_0 denotes a number such that either y_i or y_j resets when G reaches α^{t_0} on the execution (if t_0 does not exist, take $t_0 = 0$). In addition, from α^{t_0} to α^{t+1}, y_i and y_j do not reset. Since, from the conditions of Case 1, y_j progresses but y_i does not progress on edge e that leads from α^t to α^{t+1}, one of the following items is true:

(a). there is a t_1 with $t_0 \le t_1 < t$ such that, from α^{t_1} to α^t, y_i progresses but y_j does not progress,
(b). from α^{t_0} to α^t, y_i and y_j do not progress.

This is because, from property (*) mentioned earlier, y_i and y_j must progress alternately. For (a), it is observed that $y^{t_1}_i - y^{t_1}_j = (y^t_i - 1) - y^t_j$ and $a^t_{ij} = a^{t_1}_{ij} \oplus 1$. From the conditions in Case 1, we have $y^{t_1}_i - y^{t_1}_j \ge m-1$ and $a^{t_1}_{ij} = m-2$. This contradicts to (1) for α^{t_1}. For (b), we have $y^{t_0}_i - y^{t_0}_j = y^t_i - y^t_j \ge m$ and $a^{t_0}_{ij} = a^t_{ij} = m-1$. From these two facts and the definition of t_0, if $t_0 \ne 0$, then $y^{t_0}_j = 0$ and $a^{t_0}_{ij} = b^{t_0}_i$. Therefore, $y^{t_0}_i \ge m$ but $b^{t_0}_i = m-1$. This contradicts to (2) for α^{t_0}. If, however, $t_0 = 0$, then the above two facts $y^{t_0}_i - y^{t_0}_j \ge m$ and $a^{t_0}_{ij} = m-1$ already contradicts the definition of the initial value for a_{ij}.

This ends the proof of the Claim. Thus, it is valid for M to use $a_{ij} \sim d$ to do test $y_i - y_j \sim d$ and to use $b_i \sim d$ to do test $y_i \sim d$. At some point, M guesses that it has reached the configuration $(s', \pi', \boldsymbol{u}', w')$ by comparing the counter values and the stack content with $(s', \pi', \boldsymbol{u}', w')$ on the rest of the input tape. M accepts iff such a comparison succeeds. Clearly M accepts $\to^*_G$. There is a slight problem when M compares its own stack content with $(s', \pi', \boldsymbol{u}', w')$ on the one-way input tape by popping the stack. The reason is that popping the stack contents reads the reverse of the stack content. However, recall that the encoding of the stack word in $(s', \pi', \boldsymbol{u}', w')$ on the input tape is reversed. Thus, such a comparison can be proceeded.

Now, M only uses standard tests. However, assignments in M, in the form of $y_i := y_i + 1$ ($0 \le i \le k$) and $y_i := 0$ ($1 \le i \le k$), are still not standard. We now show that these assignments can be made standard, while the machine is still reversal-bounded. Let M' be an NPCM that is exactly the same as M. M' simulates M's computation from the configuration $(s, \pi, \boldsymbol{u}, w)$. Initially, each y_i takes the value of $\boldsymbol{u}(i)$ and M' calculates the initial values for all entries a_{ij} and b_i from $\boldsymbol{u}$. The calculations can be implemented under the help of a number of auxiliary reversal-bounded counters. However, for each $1 \le i \le k$, each time that M executes an assignment $y_i := y_i + 1$ or $y_i := 0$, M' does nothing to y_i. The stack operations in M are faithfully simulated by M' on its own stack. For each $1 \leq i \leq k$, at some point, either initially or at a moment $y_i := 0$ is being executed by M, M' guesses (only once for each i) that y_i will not reset afterward. When the guess for i happens at $y_i := 0$, M' decrements y_i to 0. After such a guess for i, an execution of $y_i := y_i + 1$ will also cause y_i incremented by 1. However, a later execution of $y_i := 0$ in M will cause M' to abort abnormally (without accepting the input). At some point after all $1 \leq i \leq k$ have been guessed, M' guesses that it has reached the configuration $(s', \pi', \boldsymbol{u}', w')$. Then, M' compares its current configuration with $(s', \pi', \boldsymbol{u}', w')$ as M does. Clearly, M' accepts $\rightarrow_G^*$ and each y_i in M' is reversal-bounded. Hence, $\rightarrow_G^*$ is NPCA.

In particular, when $\mathcal{A}$ is a timed automaton, $\rightarrow_G^*$ defines a set of integer tuples (without stack words). It is known that the set is NPCA iff it is Presburger [28]. Therefore, if $\mathcal{A}$ is a timed automaton, $\rightarrow_G^*$ is Presburger. ∎

Now, we conclude this section by claiming that $\rightarrow_{\mathcal{A},\pi}^*$ is NPCA by combining Lemma 8 and Lemma 9.

Lemma 10. *For any PTA $\mathcal{A}$ and any fixed pattern $\pi \in \Pi$, $\rightarrow_{\mathcal{A},\pi}^*$ is NPCA. In particular, if $\mathcal{A}$ is a timed automaton, then $\rightarrow_{\mathcal{A},\pi}^*$ is Presburger.*

7 A Decidable Binary Reachability Characterization and Automatic Verification

Recall that PTA $\mathcal{A}$ actually has clocks $x_1, \cdots, x_k$. x_0 is the auxiliary clock. The *binary reachability* $\leadsto_{\mathcal{A}}^{*\mathbf{B}}$ of $\mathcal{A}$ is the set of tuples

$$\langle s, v_1, \cdots, v_k, w, s', v_1', \cdots, v_k', w' \rangle$$

such that there exist $v_0 = 0, v_0' \in \mathbf{D}^+$ satisfying

$$(s, v_0, \cdots, v_k, w) \leadsto_{\mathcal{A}}^* (s', v_0', \cdots, v_k', w').$$

The main theorem of this paper gives a decidable characterization for the binary reachability as follows.

Theorem 1. *The binary reachability $\leadsto_{\mathcal{A}}^{*\mathbf{B}}$ of a PTA $\mathcal{A}$ is $(\mathbf{D} + \mathbf{NPCA})$-definable. In particular, if $\mathcal{A}$ is a timed automaton, then the binary reachability $\leadsto_{\mathcal{A}}^{*\mathbf{B}}$ can be expressed in the additive theory of reals (or rationals) and integers.*

Proof. From Lemma 7, $\leadsto_{\mathcal{A}}^{*\mathbf{B}}$ is definable by the following formula:

$$\exists u_0' \in \mathbf{N} \exists v_0' \in \widehat{\mathbf{D}^+} (\bigvee_{\pi \in \Pi} ((0, v_1, \cdots, v_k), (v_0', \cdots, v_k')) \in \pi \wedge$$

$$(s, (0, u_1, \cdots, u_k), w) \leadsto_{\mathcal{A},\pi}^* (s', (u_0', \cdots, u_k'), w'))$$

on integer variables $s, u_1, \cdots, u_k, s', u_1', \cdots, u_k'$ (over $\mathbf{N}$), and dense variables $v_1, \cdots, v_k, v_1', \cdots, v_k'$ (over $\widehat{\mathbf{D}^+} = \mathbf{D}^+ \cap [0, 1)$), and on word variables w and w'. This formula is equivalent to

$$\bigvee_{\pi \in \Pi} P_\pi^{\mathbf{D}^+}(v_1, \cdots, v_k, v_1', \cdots, v_k') \wedge Q_\pi^{\mathbf{Z}}(s, u_1, \cdots, u_k, w, s', u_1', \cdots, u_k', w')$$

where $P_\pi^{\mathbf{D}^+}(v_1, \cdots, v_k, v'_1, \cdots, v'_k)$ stands for

$$\exists v'_0 \in \widehat{\mathbf{D}^+}(((0, v_1, \cdots, v_k), (v'_0, \cdots, v'_k)) \in \pi)$$

and $Q_\pi^{\mathbf{Z}}(s, u_1, \cdots, u_k, w, s', u'_1, \cdots, u'_k, w')$ stands for

$$\exists u'_0((s, (0, u_1, \cdots, u_k), w) \rightsquigarrow^*_{\mathcal{A},\pi} (s', (u'_0, \cdots, u'_k), w')).$$

From the definition of patterns, $P_\pi^{\mathbf{D}^+}$, after eliminating the existential quantification, is a dense linear relation. On the other hand, $Q_\pi^{\mathbf{Z}}$ (after eliminating the existential quantification, from Lemma 10 and Lemma 2) is NPCA. Therefore, $\rightsquigarrow^{*\mathbf{B}}_{\mathcal{A}}$ is $(\mathbf{D}+\mathbf{NPCA})$-definable.

In particular, if $\mathcal{A}$ is a timed automaton, $\rightsquigarrow^{*\mathbf{B}}_{\mathcal{A}}$ is $(\mathbf{D}+\mathbf{NPCA})$-definable by a formula in the additive theory of reals (or rationals) and integers. Hence, $\rightsquigarrow^{*\mathbf{B}}_{\mathcal{A}}$ itself can be expressed in the same theory. ∎

The importance of the above characterization for $\rightsquigarrow^{*\mathbf{B}}_{\mathcal{A}}$ is that, from Lemma 2, the emptiness of $(\mathbf{D}+\mathbf{NPCA})$-definable predicates is decidable. From Theorem 1 and Lemma 2 (3)(4), we have,

Theorem 2. *The emptiness of $l \cap \rightsquigarrow^{*\mathbf{B}}_{\mathcal{A}}$ with respect to a PTA $\mathcal{A}$ for any mixed linear relation l is decidable.*

The emptiness of $l \cap \rightsquigarrow^{*\mathbf{B}}_{\mathcal{A}}$ is called a *mixed linear property* of $\mathcal{A}$. Many interesting safety properties (or their negations) for PTAs can be expressed as a mixed linear property. For instance, consider the following property of a PTA $\mathcal{A}$ with three dense clocks x_1, x_2 and x_3:

"for any two configurations α and β with $\alpha \rightsquigarrow^{*\mathbf{B}}_{\mathcal{A}} \beta$, if the difference between β_{x_3} (the value of clock x_3 in β) and $\alpha_{x_1} + \alpha_{x_2}$ (the sum of clocks x_1 and x_2 in α) is greater than the difference between $\#_a(\alpha_{\mathbf{w}})$ (the number of symbol a appearing in the stack word in α) and $\#_b(\beta_{\mathbf{w}})$ (the number of symbol b appearing in the stack word in β), then $\#_a(\alpha_{\mathbf{w}}) - 2\#_b(\beta_{\mathbf{w}})$ is greater than 5."

The negation of this property can be expressed as the emptiness of

$$(s, x_1, x_2, x_3, w) \rightsquigarrow^{*\mathbf{B}}_{\mathcal{A}} (s', x'_1, x'_2, x'_3, w') \wedge l$$

where l is the negation of a mixed linear relation (hence l itself is also a mixed linear relation):

$$x'_3 - (x_1 + x_2) > \#_a(w) - \#_b(w') \ \rightarrow\ \#_a(w) - 2\#_b(w') > 5.$$

Thus, from Theorem 2, this property can be automatically verified. We need to point out that

- $x'_3 - (x_1 + x_2) > \#_a(w) - \#_b(w')$ is a linear relation on both dense variables and discrete variables. Thus, this property can not be verified by using the decidable characterization for discrete PTAs [20], where only integer-valued clocks are considered.

- Even without clocks, $\#_a(w) - 2\#_b(w') > 5$ expresses a non-regular set of stack word pairs. Therefore, this property can not be verified by the model-checking procedures for pushdown systems [9, 24].
- Even without the pushdown stack, $x_3' - (x_1 + x_2) > 0$ (by taking $\#_a(w) - \#_b(w')$ as a constant such as 0) is not a clock region, therefore, the classical region-based techniques can not verify this property. This is also pointed out in [17].
- With both dense clocks and the pushdown stack, this property can not be verified by using the region-based techniques for Timed Pushdown Systems [10].

When $\mathcal{A}$ is a timed automaton, by Theorem 1, the binary reachability $\rightsquigarrow_{\mathcal{A}}^{*\mathbf{B}}$ can be expressed in the additive theory of reals (or rationals) and integers. Notice that our characterization is essentially equivalent to the one given by Comon and Jurski [17] in which $\rightsquigarrow_{\mathcal{A}}^{*\mathbf{B}}$ can be expressed in the additive theory of reals augmented with a predicate telling whether a term is an integer. Because the additive theory of reals and integers is decidable [8, 12], we have,

Theorem 3. *The truth value for any closed formula expressible in the (first-order) additive theory of reals (or rationals) augmented with a predicate $\rightsquigarrow_{\mathcal{A}}^{*\mathbf{B}}$ for a timed automaton $\mathcal{A}$ is decidable. (also shown in [17])*

For instance, consider the following property for a timed automaton $\mathcal{A}$ with two real clocks:

"there are states s and s' such that, for any x_1, x_2, x_2', there exists x_1' such that if (s, x_1, x_2) can reach (s', x_1', x_2') in $\mathcal{A}$, then $x_1 - x_2 > x_1' - x_2'$."

It can be expressed as

$$\exists s, s' \forall x_1, x_2, x_2' \exists x_1' ((s, x_1, x_2) \rightsquigarrow_{\mathcal{A}}^{*\mathbf{B}} (s', x_1', x_2') \rightarrow x_1 - x_2 > x_1' - x_2'),$$

and thus can be verified according to Theorem 3.

8 Conclusions, Discussions and Future Work

In this paper, we consider PTAs that are timed automata augmented with a pushdown stack. A configuration of a PTA includes a state, finitely many dense clock values and a stack word. By introducing the concept of a clock pattern and using an automata-theoretic approach, we give a decidable characterization of the binary reachability of a PTA. Since a timed automaton can be treated as a PTA without the pushdown stack, we can show that the binary reachability of a timed automaton is definable in the additive theory of reals and integers. The results can be used to verify a class of safety properties containing linear relations over both dense variables and unbounded discrete variables.

A PTA studied here can be regarded as the timed version of a pushdown machine. Carefully looking at the proofs of the decidable binary reachability characterization, we find out that the underlying untimed machine (e.g., the pushdown machine) is not essential. We can replace it with many other kinds of machines and the resulting timed system still has a decidable binary reachability characterization. We will summarize some of these machines in this section.

Consider a class of machines **X**. We use **XCM** to denote machines in **X** augmented with reversal-bounded counters. We are looking at the binary reachability characterization of the timed version of machines in **X**. The characterization is established in the previous sections when **X** represents pushdown machines. In the proofs, the pattern technique is used, in which (1) a dense clock is separated into a fractional part and an integral part and (2) the fractional parts of dense clocks are abstracted as a pattern and the integral parts are translated into reversal-bounded counters. The result of the translation is the underlying untimed machine in **X** augmented with these reversal-bounded counters, i.e., a machine in **XCM**. Suppose that a class of automata **Y** accept the binary reachability of machines in **XCM**. In the case of **X** being pushdown machines, **XCM** represents NPCMs and **Y** can be chosen as NPCAs (it is known that the binary reachability of NPCMs can be accepted by NPCAs [20].). The fact that this **Y** (i.e., NPCA) satisfies Lemma 2 is the only condition we need in order to obtain the decidable reachability characterization in Theorem 1. Definitions like NPCA predicates and $(\mathbf{D}+\mathbf{NPCA})$-definability can be accordingly modified into **Y** predicates and (**D+Y**)-definability once **Y** is clear. The above discussions give the following result.

Theorem 4. *Let* **Y** *be a class of automata,* **X** *be a class of machines and* **XCM** *be the class of machines in* **X** *augmented with reversal-bounded counters. If, for each machine in* **XCM***, an automaton in* **Y** *can be constructed that accepts the binary reachability of the machine, and Lemma 2 holds (replacing NPCA with* **Y***), then the binary reachability of the timed version of* **X** *is (***D+Y***)-definable.*

Notice that Lemma 2 (4) requires that the emptiness problem for **Y** in Theorem 4 be decidable. Theorem 2 can be immediately followed from Theorem 4 for the timed version of **X**.

According to Theorem 4, the timed version of the following machines **X** has a decidable (**D+Y**)-definable characterization for binary reachability by properly choosing **Y**:

- NPCM. Here **Y**=NPCA;
- NCM with an unrestricted counter. Notice that the counter is a special case of a pushdown stack (when the stack alphabet is unary). Here, **Y**=NPCA;
- Finite-crossing NCM [28] (i.e., NCM augmented with a finite-crossing read-only worktape. The head on the worktape is two-way, but for each cell of the tape, the head crosses only a bounded number of times.). Here, **Y** is finite-crossing NCAs [28] that are NCM augmented with a finite-crossing input tape.
- Reversal-bounded multipushdown machines [18] that are multipushdown machines [13] augmented with reversal-bounded counters. Here, **Y** is reversal-bounded multipushdown automata [18].

Let **X** be a class of machines. The pattern technique tells us that, for a decidable binary reachability characterization of the timed version of **X**, the density of clocks (and even clocks themselves) is not the key issue. This is because, using the technique, these dense clocks can be reduced to reversal-bounded integer counters. The key issue is whether **X** and its reversal-bounded version **XCM** have a decidable binary reachability characterization (i.e., the binary reachability can be accepted by a class **Y** of automata with a decidable emptiness problem). In particular, when the binary reachability of **X** is effectively semilinear (and hence the binary reachability is decidable), in most cases, the binary reachability of **XCM** is also effectively semilinear. Such **X** includes all the machines mentioned above. In this case, once we can show that the untimed machines in **X** have a decidable binary reachability characterization, we are getting really close to the decidable characterization for their timed version. But, we do have exceptions. For instance, consider **X** to be a finite state machine with a two-way read only worktape. **X** has a decidable binary reachability characterization (witnessed by one-way multitape finite automata). However, augmenting **X** with reversal-bounded counters makes the binary reachability undecidable. The pitfall here is that a two-way tape makes reversal-bounded counters too powerful. In fact, the emptiness problem is undecidable for two-way automata augmented with reversal-bounded counters. In the case when there is only one reversal-bounded counter, the emptiness problem is decidable if the machines are deterministic. The nondeterministic case is still open [29].

In practice, augmenting timed automata with other unbounded data structures allows us to study more complex real-time applications. For instance, the decidable characterization of PTAs makes it possible to implement a tool verifying recursive real-time programs containing finite-state variables against safety properties containing linear constraints over dense clocks and stack word counts. This tool will be a good complement to available tools for recursive finite state programs (for *regular* safety properties, e.g., termination) [7, 23]. On the other hand, the pattern technique is not intended to replace the traditional region-based technique used in the existing tools analyzing real-time systems (such as UPPAAL [30] and its extensions [31], TREX [31], HyTECH [26], Kronos [11]). In fact, the pattern technique is also a good complement to the region technique. When verifying timing requirements in the form of clock regions, the region technique is employed. However, when verifying some complex timing requirements that may not be in the form of clock regions, one might find the pattern technique useful. Therefore, the tools may be enhanced with the pattern technique. The results in this paper can also be used to implement a model-checker for a subset of the real-time specification language ASTRAL [14]. The subset includes history-independent ASTRAL specifications containing both dense clocks and unbounded discrete control variables.

As mentioned in this section, the timed version of NPCM (i.e., PTAs further augmented with reversal-bounded counters) also has a decidable characterization. This timed model has many important applications. For instance, a real-time recursive program (containing unbounded integer variables) can be automatically debugged using the reversal-bounded approximation (i.e., assign a reversal-bound to the variables). Additionally, a free counter (i.e., an unrestricted counter) is a special case for a pushdown stack (when the stack alphabet is unary). Therefore, this model can also be used to specify real-time systems containing a free counter and many reversal-bounded counters. It seems that "reversal-bounded counters" appear unnatural and therefore their applications in practice are remote. However, a nondecreasing counter is also a reversal-bounded counter (with zero reversal-bound). This kind of counters have a lot of applications. For instance, a nondecreasing counter can be used to count digital time elapse, the number of external events, the number of a particular branch taken by a nondeterministic program (this is important, when fairness is taken into account), etc. For instance, consider a timed automaton with input symbols (i.e., a transition is triggered by an external event as well as the enabling condition). We use $\#_a$ to denote the number of event a occurred so far. The enabling condition of a transition, besides clock constraints, may also include comparisons of the counts $\#_a$ against an integer constant and comparisons of one specific linear term T (on all $\#_a$) against an integer constant. For instance, a transition may look like this (in pseudo-code):

s: `if event(`a`) and` $x_2 - x_1 > 10$ `and` $\#_b > 21$ `and` $2\#_c - 3\#_b < 5$, `then progress(); goto` s'

where x_1 and x_2 are dense clocks. Notice that comparisons of the linear term $2\#_c - 3\#_b$ against an integer constant may show up in other transitions. But this term is unique in the automaton: a comparison like $4\#_a - 3\#_b > 8$ that involves a different term $4\#_a - 3\#_b$ can not be used in the enabling conditions of the automaton. This timed automaton is a standard timed automaton augmented with reversal-bounded counters $\#_a$ (which are nondecreasing) and a free counter (representing the linear term $2\#_c - 3\#_b$). Hence, the following property can be automatically verified:

"It is always true that whenever $x_1 - 7\#_b + 3x_2 > 2\#_a$ holds, x_1 must be greater $\#_c - \#_a$."

A future research issue is to investigate whether the decidable results [22] for Presburger liveness of discrete timed automata can be extended to timed automata (with dense clocks) using the technique in this paper. We are also going to look at the possibility of extending the approximation approaches for parameterized discrete timed automata [21] to the dense clocks. This is particularly interesting, since the reachability set presented in [21] is not necessarily semilinear. Another issue is on the complexity analysis of the decision procedure presented in this paper. However, the complexity for the emptiness problem of NPCAs is still unknown, though it is believed that it can be derived along Gurari and Ibarra [25]. Future work may also include investigating a fragment of a dense time linear temporal logic that has a decidable model-checking problem for PTAs, following the work of Comon and Cortier [15].

The author would like to thank H. Comon and O. H. Ibarra for discussions on the topic of dense timed pushdown automata during CAV'00 in Chicago, B. Boigelot, P. San Pietro and J. Su for recent discussions on [12], J. Nelson, F. Sheldon and G. Xie for reading an earlier draft of this paper, and T. Bultan, H. Comon, J. Esparza and K. Larsen for comments on the short version of this paper presented in CAV'01 in Paris. Thanks also go to the anonymous referee for many useful suggestions.

References

1. R. Alur, *"Timed automata"*, *CAV'99*, LNCS 1633, pp. 8-22
2. R. Alur, C. Courcoibetis, and D. Dill, *"Model-checking in dense real time," Information and Computation*, **104** (1993) 2-34
3. R. Alur and D. Dill, *"A theory of timed automata," Theoretical Computer Science*, **126** (1994) 183-236
4. R. Alur, T. Feder, and T. A. Henzinger, *"The benefits of relaxing punctuality," J. ACM*, **43** (1996) 116-146

5. R. Alur, T. A. Henzinger, "Real-time logics: complexity and expressiveness," *Information and Computation*, **104** (1993) 35-77
6. R. Alur, T. A. Henzinger, "A really temporal logic," *J. ACM*, **41** (1994) 181-204
7. T. Ball and S. K. Rajamani, "Bebop: a symbolic model-checker for Boolean programs," *Spin Workshop'00*, LNCS 1885, pp. 113-130.
8. J. R. Buchi, "On a decision method in restricted second order arithmetic," *Proceedings of the International Congress on Logic, Method, and Philosophy of Sciences,* Stanford University Press, pp. 1-12, 1962
9. A. Bouajjani, J. Esparza, and O. Maler, "Reachability Analysis of Pushdown Automata: Application to Model-Checking,", *CONCUR'97*, LNCS 1243, pp. 135-150
10. A. Bouajjani, R. Echahed, and R. Robbana, "On the automatic verification of systems with continuous variables and unbounded discrete data structures," *Hybrid System II*, LNCS 999, 1995, pp. 64-85
11. M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine, "Kronos: A model-checking tool for real-time systems," *CAV'98*, LNCS 1427, pp. 546-550
12. B. Boigelot, S. Rassart and P. Wolper, "On the expressiveness of real and integer arithmetic automata," *ICALP'98*, LNCS 1443, pp. 152-163
13. A. Cherubini, L. Breveglieri, C. Citrini, and S. Crespi Reghizzi. "Multi-push-down languages and grammars," *International Journal of Foundations of Computer Science,* 7(3): 253-291, 1996
14. A. Coen-Porisini, C. Ghezzi and R. Kemmerer, "Specification of real-time systems using ASTRAL," *IEEE Transactions on Software Engineering*, **23** (1997) 572-598
15. H. Comon and V. Cortier, "Flatness is not a weakness," *CSL'00*, LNCS 1862, pp. 262–276
16. H. Comon and Y. Jurski, "Multiple counters automata, safety analysis and Presburger arithmetic," *CAV'98*, LNCS 1427, pp. 268-279.
17. H. Comon and Y. Jurski, "Timed automata and the theory of real numbers," *CONCUR'99*, LNCS 1664, pp. 242-257
18. Z. Dang, "Debugging and verification of infinite state real-time systems," PhD Dissertation, University of California, Santa Barbara, August 2000
19. Z. Dang, "Binary reachability analysis of pushdown timed automata with dense clocks," *CAV'01*, LNCS 2102, pp. 506-517
20. Z. Dang, O. H. Ibarra, T. Bultan, R. A. Kemmerer, and J. Su, "Binary reachability analysis of discrete pushdown timed automata," *CAV'00*, LNCS 1855, pp. 69-84
21. Z. Dang, O. H. Ibarra and R. A. Kemmerer, "Decidable approximations on generalized and parameterized discrete timed automata," *COCOON'01*, LNCS 2108, pp. 529-539
22. Z. Dang, P. San Pietro and R. A. Kemmerer, "On Presburger liveness of discrete timed automata," *STACS'01*, LNCS 2010, pp. 132-143
23. J. Esparza and S. Schwoon, "A BDD-based model-checker for recursive programs," *CAV'01*, LNCS 2102, pp. 324-336
24. A. Finkel, B. Willems and P. Wolper, "A direct symbolic approach to model checking pushdown systems," *INFINITY'97*.
25. E. Gurari and O. Ibarra, "The complexity of decision problems for finite-turn multicounter machines," *J. Computer and System Sciences*, **22** (1981) 220-229
26. T. A. Henzinger and Pei-Hsin Ho, "HyTech: the Cornell hybrid technology tool," *Hybrid Systems II*, LNCS 999, pp. 265-294
27. T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. "Symbolic model checking for real-time systems," *Information and Computation*, **111** (1994) 193-244
28. O. H. Ibarra, "Reversal-bounded multicounter machines and their decision problems," *J. ACM*, **25** (1978) 116-133
29. O. H. Ibarra, T. Jiang, N. Tran and H. Wang, "New decidability results concerning two-way counter machines," *SIAM J. Comput.,* **24** (1995) 123-137

30. K. G. Larsen, P. Pattersson, and W. Yi, *"UPPAAL in a nutshell," International Journal on Software Tools for Technology Transfer*, **1** (1997): 134-152
31. K. G. Larsen, G. Behrmann, Ed Brinksma, A. Fehnker, T. Hune, P. Pettersson, and J. Romijn, *"As cheap as possible: efficient cost-optimal reachability for priced timed automata," CAV'01*, LNCS 2102, pp. 493-505
32. F. Laroussinie, K. G. Larsen, and C. Weise, *"From timed automata to logic - and back," MFCS'95*, LNCS 969, pp. 529-539
33. M. L. Minsky, *"Computation: finite and infinite machines,"* Prentice-Hall, 1967
34. J. Raskin and P. Schobben, *"State clock logic: a decidable real-time logic," HART'97*, LNCS 1201, pp. 33-47
35. T. Wilke, *"Specifying timed state sequences in powerful decidable logics and timed automata,"* LNCS 863, pp. 694-715, 1994
36. S. Yovine, *"Model checking timed automata," Embedded Systems'98*, LNCS 1494, pp. 114-152