Event-Clock Visibly Pushdown Automata

Nguyen Van Tang and Mizuhito Ogawa

Graduate School of Information Science
Japan Advanced Institute of Science and Technology
Email: {*tang_nguyen*, *mizuhito*}@*jaist.ac.jp*

Abstract. Event-clock automata is a determinizable subclass of timed automata, where the clock valuations are determined by the input instead of by the transition relation. In this note, we propose *event-clock visibly pushdown automata* (ECVPAs) as a proper extension of event-clock automata. The class of ECVPAs is sufficiently expressive to model real-time pushdown systems, and to specify common non-regular context-free real-time requirements. We show that ECVPAs are closed under boolean operations, and thus the language-inclusion problem is decidable for ECVPAs. Using these properties together with the automata-theoretic approach, we prove that the model checking problem over ECVPA models against ECVPA specifications is EXPTIME-complete.

Key words: visibly pushdown automata, timed automata, pushdown systems, model checking, verification of real-time systems.

1 Introduction

One of the most successful techniques for automatic verification of reactive systems has been *model checking*. In this verification technique, one checks the correctness of a system with respect to a desired behavior by checking whether a mathematical model of the system satisfies a formal specification of this behavior (see [7] for an overview). Traditionally, model checking is applied to finite-state systems. Recently, model checking has been extended to infinite-state sequential systems. These are systems in which each state carries a finite, but unbounded, amount of information, e.g., a pushdown stack. Pushdown automata (PDAs) naturally model the control flow of sequential programs with nested and recursive procedure calls. Therefore, PDAs are the adequate model to deal with program analysis, and model checking questions can be formulated as decision problems for PDAs. While many analysis problems, such as identifying dead code and accesses to uninitialized variables, can be captured as regular requirements, many others require inspections of the stack or matching of calls and returns are non-regular context-free [10]. Although the general problem of checking context-free properties of PDAs is undecidable [10], weaker models have been proposed to verify different kinds of non-regular properties. One of the most promising approaches is that of *visibly pushdown automata* (VPAs) [3]. These are PDAs in which stack operations (i.e., whether to push, pop, or no stack operation) are

completely determined by the input alphabet. Such a restriction on the use of stack leads VPAs to enjoy all desirable closure properties and decision problems, though retaining an expressiveness that is adequate to formulate program analysis questions. Therefore, checking context-free properties of pushdown models is decidable as long as the calls and returns are made visible. This visibility seems quite natural while writing requirements about pre/post conditions or for inter-procedural flow properties. In particular, requirements that can be verified in this manner include all regular properties, and non-regular properties such as: partial correctness (if P holds when a procedure is invoked, then if the procedure returns, P' holds upon return), total correctness (if P holds when a procedure is invoked, then the procedure must return and P' must hold at the return state), local properties, access control, and stack limits.

Timed automata were introduced by Alur and Dill in [1], and have become a standard modeling formalism for real-time systems. A timed automaton is a finite automaton augmenting with a finite set of real-valued variables modeling clocks. Constraints on the clock variables are used to restrict the behaviors of an automaton. A timed automaton, then, accepts *timed words* in which each symbol is paired with a real-valued time stamp. The theory of timed automata allows the solution of certain verification problems for real-time systems with finite control [1, 9]. However, the general verification problem (i.e., language inclusion) is undecidable for timed automata [1, 11][1]. This is because, unlike the untimed case, timed automata are not determinizable. Hence, one must then work either with deterministic specifications or with a restricted class of timed automata which has the required closure properties. The *event-clock automata* (ECAs), proposed by Alur, Fix, and Henzinger in [2], is a subclass of timed automata which permit non-determinism (which is useful from a modeling point of view), and are closed under boolean operations, including complementation. The key feature of these automata is that they have a pair of implicit clocks associated with each action. The clocks record the time elapsed since the last occurrence of the associated action, as well as the time that will elapse before the next occurrence. As a result, common real-time regular requirements like bounded response time can be naturally modeled using these automata.

In hard real-time programming, one faces the challenge that the execution of a program must satisfy given timing constraints. Timed automata form a good framework to specify regular real-time properties [1, 9]. Unfortunately, some common non-regular context-free real-time properties such as "if P holds when a procedure is invoked, then the procedure must return within 5 time units and P' must hold at the return state" cannot be captured by any VPA [3] or any timed automaton [1]. Such kinds of properties can be captured by pushdown timed automata (PTAs) [5]. However, the general verification problem (i.e., language inclusion) for PTAs is undecidable.

In this paper, we introduce *event-clock visibly pushdown automata* (ECVPAs) as an extension of event-clock automata. The class of ECVPAs is suffi-

[1] Recently, Ouaknine and Worrell have showed that the inclusion problem $L(A) \subseteq L(B)$ becomes decidable when B has at most one clock [11].

ciently expressive to model real-time pushdown systems, and to specify common non-regular context-free real-time properties. To study the closure properties of ECVPAs, we first clarify that the essential restriction behind the notion of event clocks is *deterministic clock-valuation*. Clock valuation is deterministic if the value of each clock is not affected by the choice among nondeterministic transitions. This enables us to make a homomorphism from ECVPAs to VPAs. Thus, ECVPAs are closed under union, intersection, and complementation. As a result, the language inclusion problem for ECVPAs is decidable. Using these good closure properties together with the automata-theoretic approach, we prove that the model checking problem of event-clock visibly pushdown automata is EXPTIME-complete.

The remainder of this paper is organized as follows. Section 2 recalls the standard notions of timed automata and briefly explains why timed automata are not determinizable. Section 3 recalls the definition and properties of event-clock automata. Section 4 defines event-clock visibly pushdown automata. Section 5 shows that the class of ECVPAs is closed under union, intersection, and complementation. Thus, the language inclusion problem for ECVPAs is EXPTIME-complete. This section also describes how to model and verify real-time context-free properties for real-time recursive programs using ECVPAs. In Section 6, we consider ECVPAs with Büchi acceptance condition. We demonstrate that Büchi ECVPAs are closed under union, intersection and complementation. In Section 7, we discuss related work. Section 8 concludes the paper with some possible future directions.

2 Timed Words and Timed Automata

Let N denote the set of natural numbers $\{0, 1, ...\}$, let R^+ and Q^+ denote the set of non-negative real numbers and non-negative rational numbers, respectively. As usual, the set of all finite words over a finite input alphabet Σ will be denoted by Σ^*.

A *timed word* $\bar{w}$ over an alphabet Σ is a finite sequence $(a_0, t_0)(a_1, t_1) \cdots (a_n, t_n)$ of symbols $a_i \in \Sigma$ that are paired with nonnegative real numbers $t_i \in R^+$ such that the sequence $\bar{t} = t_0 t_1 \cdots t_n$ of time-stamps is nondecreasing (i.e., $t_i \leq t_{i+1}$ for all $0 \leq i < n$). Without loss of generality, it may be assumed that $t_0 = 0$. Sometimes, we denote the timed word $\bar{w}$ by the pair $(\bar{a}, \bar{t})$, where $\bar{a} \in \Sigma^*$ is an untimed word over Σ. The set of all finite timed words over Σ is denoted by $T\Sigma^*$. A *timed language* over the alphabet Σ is a subset of $T\Sigma^*$.

In the following, we recall the standard notion of a timed automaton [1]. Let $X = \{x_1, ..., x_n\}$ be a finite set of *clock variables*. Define the set $\Phi(X)$ of clock constraints over X by the grammar:

$$\varphi ::= \top \mid x \bowtie c \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2,$$

where $c \in Q^+$, $x \in X$, and $\bowtie \in \{<, \leq, \geq, >\}$.

Definition 1 (Timed Automaton). *A timed automaton over input alphabet Σ is a tuple $A = (Q, q_0, X, \Delta, F)$, where Q is a finite set of locations, $q_0 \in Q$ is*

an initial location, $F \subseteq Q$ is a set of final locations, X is a finite set of clocks, and $\Delta \subseteq Q \times \Sigma \times \Phi(X) \times 2^X \times Q$ is a finite set of edges. An edge (q, a, φ, Re, q') allows a-labelled transition from q to q', provided the precondition φ on clocks is satisfied. Afterwards, the clocks in Re are reset to 0, while all other clocks remain unchanged.

A *clock valuation* of A is a vector $\nu = (\upsilon_1, ..., \upsilon_n)$, where $\upsilon_i \in R^+$ gives the value of clock x_i. If $t \in R^+$, we let $\nu + t$ be the clock valuation whose i-th component is $\upsilon_i + t$. A *state* of A is a pair (q, ν), where $q \in Q$ is a location and ν is a clock valuation. A transition of A is defined by $(q, \nu) \xrightarrow{d} \xrightarrow{a} (q', \nu')$, iff there is an edge (q, a, φ, Re, q') such that $(\nu + d)$ satisfies φ, $\upsilon'_i = 0$ for all $x_i \in Re$ and $\upsilon'_i = (\upsilon_i + d)$ for all $x_i \notin Re$.

Let $\bar{w} = (\bar{a}, \bar{t})$ be a timed word and write $d_i = t_{i+1} - t_i$ for the time delay between the i-th and (i+1)-th actions. Define a *run* of A on $\bar{w}$ to be a sequence of discrete transitions: $(q_0, \nu_0) \xrightarrow{d_0} \xrightarrow{a_0} (q_1, \nu_1) \xrightarrow{d_1} \xrightarrow{a_1} (q_2, \nu_2) \rightsquigarrow \cdots \rightsquigarrow (q_{n-1}, \nu_{n-1}) \xrightarrow{d_n} \xrightarrow{a_n} (q_n, \nu_n)$, where q_0 is the initial location and ν_0 maps every clock to 0. A run is *accepting* if it ends in a final location. The timed language of A, denoted $TL(A)$, is the set of all timed words over which A has an accepting run.

For a given timed word $\bar{w}$, assume that there are two runs of A over $\bar{w}$:

- $\rho_1 = (q_0, \nu_0) \xrightarrow{d_0} \xrightarrow{a_0} (q_1, \nu_1) \rightsquigarrow \cdots \rightsquigarrow (q_{n-1}, \nu_{n-1}) \xrightarrow{d_n} \xrightarrow{a_n} (q_n, \nu_n)$.
- $\rho_2 = (q_0, \nu_0) \xrightarrow{d_0} \xrightarrow{a_0} (q'_1, \nu'_1) \rightsquigarrow \cdots \rightsquigarrow (q'_{n-1}, \nu'_{n-1}) \xrightarrow{d_n} \xrightarrow{a_n} (q'_n, \nu'_n)$.

Due to clock-reset conditions on transitions, the clock valuations at each step of two runs may be different, i.e., there exists i such that $\nu_i \neq \nu'_i$. In this case, we say that the clock valuations of ρ_1 and ρ_2 are not *deterministic*. In other words, the clock valuations is not deterministic if the value of each clock is affected by the choice among nondeterministic transitions. Due to this property of *nondeterministic* clock valuations, timed automata are neither determinizable nor closed under complementation.

Theorem 1 ([1]). *Timed automata are closed under union and intersection, but not closed under complementation. Thus, the language inclusion problem for timed automata is undecidable.*

3 Event-clock Automata: Deterministic Clock-valuations

As mentioned earlier, on reading the same word, the clock valuations of a timed automaton along different runs may be not deterministic. This leads timed automata to be not determinizable. To overcome this difficulty, one has to consider some restrictions on the set of clocks. In [2], Alur et al. proposed the notion of event-clock automata in which the clock valuations is determined only by the input word, i.e., deterministic clock valuations. Thus, on reading the same input word, the clock valuations of ECAs along different runs are deterministic.

An event-clock automaton over an alphabet Σ uses two implicit clocks x_a and y_a for each $a \in \Sigma$. Along a timed word, the clock x_a measures the time since the last occurrence of symbol a, and y_a measures the time to the next occurrence of symbol a. x_a and y_a are named *event-recording* and *event-predicting* clocks for event a, respectively. Let $C_\Sigma = \{x_a \mid a \in \Sigma\} \cup \{y_a \mid a \in \Sigma\}$ be the set of event-recording and event-predicting clocks. Let $\perp$ denotes the *undefined* value. Let $R^+_\perp = R^+ \cup \{\perp\}$ and $Q^+_\perp = Q^+ \cup \{\perp\}$.

Definition 2 (Deterministic clock valuation). *For each timed word* $\bar{w} = (a_0, t_0)(a_1, t_1) \cdots (a_n, t_n)$. *A clock valuation over* $\bar{w}$ *is a function* $\nu^{\bar{w}}_j : C_\Sigma \to R^+_\perp$ *which specifies the values of the clocks in* C_Σ *at position* j *in* $\bar{w}$.

$$\nu^{\bar{w}}_j(x_a) = \begin{cases} t_j - t_i & \text{If } \exists i < j : a_i = a, \text{and } \forall k : i < k < j \Rightarrow a_k \neq a \\ \perp & \text{otherwise} \end{cases}$$

$$\nu^{\bar{w}}_j(y_a) = \begin{cases} t_i - t_j & \text{If } \exists i > j : a_i = a, \text{and } \forall k : j < k < i \Rightarrow a_k \neq a \\ \perp & \text{otherwise} \end{cases}$$

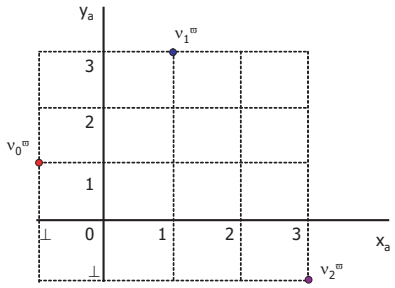

Fig. 1. values of x_a and y_a for $\bar{w}$

Figure 1, for example, shows the values of $\nu^{\bar{w}}_j$ for the clocks x_a and y_a for the timed word $\bar{w} = (a, 2)(a, 3)(a, 6)(b, 7)(b, 10)$.

Remark 1. The *event-recording clock* x_a behaves exactly like an automaton clock that is reset every time the automaton encounters the input symbol a. The value of x_a, therefore, is determined by the input word, not by the automaton. The *event-predicting clock* y_a can be viewed as an automaton clock that is reset, every time the automaton encounters the input symbol a, to a nondeterministic negative starting value, and checked for 0 at the subsequent occurrence of a.

Definition 3 (Event-clock Automaton). *An event-clock automaton (ECA) is a tuple $A = \langle \Sigma, Q, Q_0, F, E \rangle$ where Q is finite set of locations, $Q_0 \subseteq S$ is a set initial locations, $F \subseteq Q$ is a set of final locations, and $E \subseteq Q \times \Sigma \times \Phi(C_\Sigma) \times Q$ is a finite set of edges. An edge (q, a, φ, q') allows a-labeled transition from q to q', provided the clock constraint $\varphi \in C_\Sigma$ is satisfied.*

The *clock constraints* of ECAs compare clock values to rational constants or to the special value $\perp$. The clock constraints over C_Σ are interpreted with respect to the clock-valuation function ν from C_Σ to $R_\perp^+$: the atom $\perp \leq \perp$ evaluates to true, and all other comparisons that involve $\perp$ (e.g., $\perp \geq 3$) evaluate to false. For the clock-valuation ν and a clock constraint φ, we write $\nu \models \varphi$ to denote that ν satisfies the constraint φ.

A *run* of the event-clock automaton A over the timed input word $\bar{w}$ is a finite sequence: $q_0 \xrightarrow{e_1} q_1 \xrightarrow{e_2} q_2 \cdots \xrightarrow{e_{n-1}} q_n \xrightarrow{e_n} q_{n+1}$ of locations $q_i \in Q$ and edges $e_i = (q_i, a_i, \varphi_i, q_{i+1}) \in E$ such that $q_0 \in Q_0$ and for all $0 \leq i \leq n$, $\nu_i^{\bar{w}} \models \varphi_i$. A run is *accepting* iff it ends in a final control location, i.e., $q_{n+1} \in F$. The timed language $TL(A)$ defined by the event-clock automaton A consists of all timed words $\bar{w}$ such that A has an accepting run over $\bar{w}$. The next theorem summarizes the main results about event-clock automata.

Theorem 2 ([2]). *The class of ECAs is closed under all boolean operations, and the language-inclusion problem for ECAs is* PSPACE*-complete.*

4 Event-Clock Visibly Pushdown Automata

In this section, we extend event-clock automata to event-clock visibly pushdown automata to model real-time pushdown systems. For simplicity, to this end, let us fix a *pushdown alphabet* $\Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_i$ that comprises three *disjoint* finite alphabets - Σ_c is a finite set of *calls*, Σ_r is a finite set of *returns*, and Σ_i is a finite set of *internal actions*. By abuse of notation $\perp$ also denotes the *undefined* value, and the *bottom-of-stack* of an event-clock visibly pushdown automaton (see definition below).

Definition 4 (Event-clock visibly pushdown automaton). *An event-clock visibly pushdown automaton (ECVPA) on finite timed words over Σ is a tuple $M = \langle Q, \Sigma, C_\Sigma, Q_0, \Gamma, \delta, Q_f \rangle$, where Q is a finite set of locations, $Q_0 \subseteq Q$ is a finite set of initial locations, C_Σ is a set of event clocks associated with Σ, Γ is a finite stack alphabet that contains a special symbol $\perp$, $Q_f \subseteq Q$ is a set of final locations, and $\delta = \delta_c \cup \delta_r \cup \delta_i$ is the transition relation,*

- *$\delta_c \subseteq Q \times \Sigma_c \times \Phi(C_\Sigma) \times Q \times (\Gamma \backslash \{\perp\})$ is a push-transition relation,*
- *$\delta_r \subseteq Q \times \Sigma_r \times \Phi(C_\Sigma) \times \Gamma \times Q$ is a return-transition relation, and*
- *$\delta_i \subseteq Q \times \Sigma_i \times \Phi(C_\Sigma) \times Q$ is an internal-transition relation.*

Before defining configurations and runs of M, let us briefly explain the intuition behind the transition rules of M. We call $(q, a, \varphi, q', \gamma) \in \delta_c$ a push-transition, where on reading a when the clock valuation satisfies φ the symbol γ

is pushed onto the stack and the control location changes to q'; $(q, a, \varphi, \gamma, q') \in \delta_r$ a pop-transition, where on reading a when the clock valuation satisfies φ, γ is popped from the stack leading to the control location q'; and $(q, a, \varphi, q') \in \delta_i$ an internal transition, where its control location, on reading a when the clock valuation satisfies φ, changes from q to q' with no stack operation.

A stack content is a nonempty finite sequence over $(\Gamma \setminus \{\bot\})$ ending in the bottom-of-stack symbol $\bot$, let us denote the set of all stacks as $St = (\Gamma \setminus \{\bot\})^*.\ \bot$. A configuration of an ECVPA M is a pair (q, σ) where $q \in Q$, and σ is the stack content. For a timed word $\bar{w} = (a_0, t_0)(a_1, t_1) \cdots (a_k, t_k)$, a *run* of M on $\bar{w}$ is a sequence of configurations $\rho = (q_0, \sigma_0) \cdots (q_{k+1}, \sigma_{k+1})$, where each $q_i \in Q$, $\sigma_i \in St$, $q_0 \in Q_0, \sigma_0 = \bot$, and for every $1 \leq i \leq k$ the following condition holds:

- **Internal:** If a_i is an internal symbol, then $(q_i, a_i, \varphi_i, q_{i+1}) \in \delta_i$, $\nu_i^{\bar{w}} \models \varphi_i$, and $\sigma_{i+1} = \sigma_i$.
- **Push:** If a_i is a call symbol, then for some $\gamma \in \Gamma$, $(q_i, a_i, \varphi_i, q_{i+1}, \gamma) \in \delta_c$, $\nu_i^{\bar{w}} \models \varphi_i$, and $\delta_{i+1} = \gamma.\delta_i$.
- **Pop:** If a_i is a return symbol, then for some $\gamma \in \Gamma$, $(q_i, a_i, \varphi_i, \gamma, q_{i+1}) \in \delta_r$, $\nu_i^{\bar{w}} \models \varphi_i$, and either $\gamma \in \Gamma$ and $\sigma_i = \gamma.\sigma_{i+1}$, or $\gamma = \sigma_i = \sigma_{i+1} = \bot$.

A run is *accepting* if it ends in a final location. A timed word $\bar{w}$ is accepting if there is an accepting run of M on $\bar{w}$. The language accepted by an ECVPA M, denoted $TL(M)$, is the set of all accepting timed words $\bar{w}$. A language of finite timed words L is an *event-clock visibly pushdown language* (ECVPL) if there is an ECVPA M such that $L = TL(M)$.

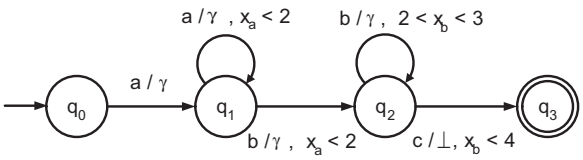

Fig. 2. Event-clock visibly pushdown automaton M

Example 1. The ECVPA M over $\Sigma = \langle \{a\}, \{b, c\}, \emptyset \rangle$ of Figure 2 uses two event-recording clocks, x_a and x_b. The alphabet Σ we use for M is partitioned in $\Sigma_c = \{a\}$, $\Sigma_r = \{b, c\}$, and $\Sigma_i = \emptyset$.

- **Push:** $(q_0, a, \top, q_1, \gamma)$, $(q_1, a, x_a < 2, q_1, \gamma)$
- **Pop:** $(q_1, b, x_a < 2, \gamma, q_2)$, $(q_2, b, 2 < x_b < 3, \gamma, q_2)$, $(q_2, c, x_b < 4, \bot, q_3)$

The automaton M accepts timed input words of the form $(\bar{\alpha}, \bar{t})$ with $\bar{\alpha} = a^n b^n c$ for some integer $n \geq 0$. All edges that are not labeled by clock constraints have, by default, the trivial clock constraint true. The clock constraint $x_a < 2$ that

is associated with the loop from q_1 to itself ensures that each a occurs within 2 time units of the preceding a. The clock constraint $x_a < 2$ that is associated with the edge from q_1 to q_2 ensures that b occurs within 2 time units of the preceding b. A similar mechanism for checking the value of x_b while reading b ensures that the time difference between each b and the subsequent b is always greater than 2 and less than 3. Thus, the timed language $L(M)$ defined by M consists of all timed words $(\bar{\alpha}, \bar{t})$ such that for all $1 \leq i < n$, we have $t_{i+1} \leq t_i + 2$, and for all $n+1 \leq j \leq 2n$, we have $t_j + 2 < t_{j+1} < t_j + 3$. However, $TL(M_1)$ cannot be accepted by any timed automaton.

Pushdown timed automata (PTAs) were considered in [5] for modeling real-time context-free processes. A PTA is a timed automaton augmented with a pushdown stack. The emptiness problem for PTAs is decidable. Unfortunately, PTAs are not closed under intersection and complementation, and language inclusion problem for PTAs is undecidable. The next theorem addresses the relationships between classes of ECAs, TAs, ECVPAs, and PTAs.

Theorem 3. *The following assertions hold:*
1) $ECAs \subsetneq ECVPAs$; 2) $ECVPAs \nsubseteq TAs$;
3) $TAs \nsubseteq ECVPAs$; 4) $ECVPAs \subsetneq PTAs$.

Definition 5. *An ECVPA $M = \langle Q, \Sigma, C_\Sigma, Q_0, \Gamma, \delta, F \rangle$, is deterministic iff all the following conditions hold.*

- *M has at most one initial location, i.e., $|Q_0| \leq 1$, and*
- *For every $a \in \Sigma_i$: If $(q, a, \varphi_1, q_1) \in \delta_i$ and $(q, a, \varphi_2, q_2) \in \delta_i$, then $\varphi_1 \wedge \varphi_2$ is mutually exclusive, i.e., $\nu \not\models \varphi_1 \wedge \varphi_2$ for all clock-valuation ν.*
- *For every $a \in \Sigma_c$: If $(q, a, \varphi_1, q_1, \gamma_1) \in \delta_c$ and $(q, a, \varphi_2, q_2, \gamma_2) \in \delta_c$, then $\varphi_1 \wedge \varphi_2$ is mutually exclusive, i.e., $\nu \not\models \varphi_1 \wedge \varphi_2$ for all clock-valuation ν.*
- *For $a \in \Sigma_r, \gamma \in \Gamma$: If $(q, a, \varphi_1, \gamma, q_1) \in \delta_r$ and $(q, a, \varphi_2, \gamma, q_2) \in \delta_r$, then $\varphi_1 \wedge \varphi_2$ is mutually exclusive, i.e., $\nu \not\models \varphi_1 \wedge \varphi_2$ for all clock-valuation ν.*

The determinism condition ensures that, at each step during a computation, the choice of the next transition is uniquely determined by the current location of the automaton, the input word, and by the current position of the automaton along the input word.

5 Closure properties and Decision problems

This section demonstrates the good closure properties and tractable decision problems for ECVPAs. Our key is to prove that ECVPAs are determinizable. To do so, we establish a homomorphism from ECVPAs to VPAs.

For a given ECVPA M, let $B = \{r_0, r_1, ..., r_k\}$ be the set of constants appearing in the clock constraints of M. Without loss of generality, let us assume that $0 \leq r_0 < r_1 < \cdots < r_k$. We define $Intv = \{[\bot, \bot], [0, 0], (0, r_0))\} \cup \{[r_i, r_i], (r_i, r_{i+1}) \mid 0 \leq i < n\} \cup \{[r_k, r_k], (r_k, \infty)\}$. An *interval pushdown alphabet* based on Σ is the set $\Lambda = \Lambda_i \cup \Lambda_c \cup \Lambda_r = \Sigma \times Intv^{|C_\Sigma|}$, where $\Lambda_i =$

$\Sigma_i \times Intv^{|C_\Sigma|}$ is the subset of internals, $\Lambda_c = \Sigma_c \times Intv^{|C_\Sigma|}$ is the subset of calls, and $\Lambda_r = \Sigma_r \times Intv^{|C_\Sigma|}$ is the subset of returns. It is easy to see that $|\Lambda| = |\Sigma| \times |Intv|^{2|\Sigma|} = |\Sigma| \times (2r_k + 2)^{2|\Sigma|}$.

Elements of an interval pushdown alphabet Λ are thus of the form (a, g) with $a \in \Sigma$ and $g : C_\Sigma \to Intv$. The component g is called *guard* and is meant to present the constraint: $\bigwedge_{x \in C_\Sigma} x \in g(x)$. Let $\lambda = (a_1, g_1) \cdots (a_n, g_n) \in \Lambda^*$. Then λ induces in a natural way a set of timed words, denoted $tw(\lambda)$, as follows. Let $\bar{w} = (b_1, t_1) \cdots (b_n, t_n) \in T\Sigma^*$. Then $\bar{w} \in tw(\lambda)$ iff for each i, $b_i = a_i$ and for each $x \in C_\Sigma$, $\nu_i^{\bar{w}}(x) \in g_i(x)$. The map tw is extended to work on subsets of Λ^* in the natural way. For each $\bar{L} \subseteq \Lambda^*$, we define $tw(\bar{L}) = \bigcup_{\lambda \in \bar{L}} tw(\lambda)$.

Lemma 1. *Let $\lambda, \lambda' \in \Lambda^*$, if $\lambda \neq \lambda'$ then $tw(\lambda) \cap tw(\lambda') = \emptyset$.*

Proof. Assume on the contrary that $tw(\lambda) \cap tw(\lambda') \neq \emptyset$. Let $\bar{w} = (a_1, t_1) \cdots (a_n, t_n) \in tw(\lambda) \cap tw(\lambda')$ then we see that $\lambda(i) = (a_i, g_i)$ and $\lambda'(i) = (a_i, g_i')$ and $g_i(z) \cap g_i'(z) \neq \emptyset$ for all $1 \leq i \leq n$, $z \in C_\Sigma$. Since $\lambda \neq \lambda'$, $g_i(z) \neq g_i'(z)$ for each $z \in C_\Sigma$. Thus, we obtain the property that $g_i(z) \neq g_i'(z)$ and $g_i(z) \cap g_i'(z) \neq \emptyset$ for all $1 \leq i \leq n$, $z \in C_\Sigma$. This contradict with the fact that there cannot exist $I, J \in Intv$ such that $I \neq J$ and $I \cap J \neq \emptyset$.

□

Definition 6 (Untimed Homomorphism). *For a given ECVPA $M = \langle Q, \Sigma, C_\Sigma, Q_0, \Gamma, \delta, Q_f \rangle$, let Λ be the interval pushdown alphabet constructed as above. We define a VPA $ut(M) = \langle Q, \Lambda, Q_0, \Gamma, \Delta, Q_f \rangle$ over Λ as follows:*

- **Push:** *If $(q, a, \varphi, q', \gamma) \in \delta_c$ then $(q, (a, g), q', \gamma) \in \Delta_c$, where $(a, g) \in \Lambda$ such that $\varphi = \bigwedge_{x \in C_\Sigma} x \in g(x)$.*
- **Pop:** *If $(q, a, \varphi, \gamma, q') \in \delta_r$ then $(q, (a, g), \gamma, q') \in \Delta_r$, where $(a, g) \in \Lambda$ such that $\varphi = \bigwedge_{x \in C_\Sigma} x \in g(x)$.*
- **Internal:** *If $(q, a, \varphi, q') \in \delta_i$ then $(q, (a, g), q') \in \Delta_i$, where $(a, g) \in \Lambda$ such that $\varphi = \bigwedge_{x \in C_\Sigma} x \in g(x)$.*

Lemma 2. *Let M be an ECVPA then $tw(L(ut(M))) = TL(M)$, and if M is deterministic then $ut(M)$ is also deterministic.*

Definition 7 (Timed Homomorphism). *For a given VPA $N = \langle Q, \Lambda, Q_0, \Gamma, \Delta, Q_f \rangle$ over interval pushdown alphabet Λ, we define an ECVPA $ec(N) = \langle Q, \Sigma, C_\Sigma, Q_0, \Gamma, \delta, Q_f \rangle$ as follows:*

- **Push:** *If $(q, (a, g), q', \gamma) \in \Delta_c$ then $(q, a, \varphi, q', \gamma) \in \delta_c$ where the clock constraint $\varphi = \bigwedge_{x \in C_\Sigma} x \in g(x)$.*
- **Pop:** *If $(q, (a, g), \gamma, q') \in \Delta_r$ then $(q, a, \varphi, \gamma, q') \in \delta_r$ where the clock constraint $\varphi = \bigwedge_{x \in C_\Sigma} x \in g(x)$.*
- **Internal:** *If $(q, (a, g), q') \in \Delta_i$ then $(q, a, \varphi, q') \in \delta_i$ where the clock constraint $\varphi = \bigwedge_{x \in C_\Sigma} x \in g(x)$.*

Lemma 3. *$TL(ec(N)) = tw(L(N))$, and if N is deterministic then $ec(N)$ is also deterministic.*

Theorem 4 (Determinization). *For any nondeterministic ECVPA M, there is a deterministic ECVPA $Det(M)$ such that $TL(Det(M)) = TL(M)$. Moreover, if M has n states, we can construct $Det(M)$ with $O(2^{n^2})$ states and $O(2^{n^2} \cdot |\Sigma_c| \cdot (2r)^{|2\Sigma|})$ stack symbols, where r is the largest constant appearing in the clock constraints of M. The set of clocks of $Det(M)$ coincides with that of M.*

Proof. Let $M = \langle Q, \Sigma, C_\Sigma, Q_0, \Gamma, \delta, Q_f \rangle$ be a nondeterministic ECVPA. Assume that $B = \{r_0, r_1, ..., r_k\}$ is the set of constants appearing in the clock constraints of M. Let $Intv$ be the set of interval defined as above. Without loss of generality, we can assume that each clock constraint in each transition of M is of the form $\bigwedge_{x \in C_\Sigma}(x \in I_x)$ where $I_x \in Intv$. We first construct a VPA $ut(M)$ over interval pushdown alphabet Λ using Definition 6. Second, determinize the VPA $ut(M)$ using determinization technique presented in [3]. We get an equivalent deterministic VPA $Det(ut(M))$ with $O(2^{n^2})$ locations and $O(2^{n^2} \cdot |\Lambda_c|)$ stack symbols. Finally, transform the VPA $Det(ut(M))$ into an ECVPA $Det(M) = ec(Det(ut(M)))$ using Definition 7. Based on Lemmas 1, 2, and 3, we obtain $TL(Det(M)) = TL(M)$ and $Det(M)$ is deterministic. The number of locations of $Det(M)$ is $O(2^{n^2})$ and the number of stack symbols is $O(2^{n^2} \cdot |\Sigma_c| \cdot (2r_k)^{2|\Sigma|})$. □

A direct determinization from nondeterministic ECVPAs to deterministic ECVPAs is discussed in Appendix D. The next theorem follows from Theorem 4 and the visibility on input symbols.

Theorem 5 (Closure properties). *The class of ECVPL of timed languages is closed under union, intersection, and complementation.*

Using Alur and Dill's region technique [1], Bouajjani et al. [5] showed that the emptiness problem for pushdown timed automata is decidable in $O(n^3 \cdot 2^{3mlogrm})$, where n is the size the automaton, m is the size of input alphabet, and r is the largest constant of clock constraints appearing in the automaton. Since each ECVPA is a pushdown timed automaton (Theorem 3), the emptiness problem for ECVPAs is decidable. Together with Theorem 5, we get

Theorem 6 (Language Inclusion). *The language inclusion problem for ECVPLs is* EXPTIME*-complete.*

Proof. Decidability and membership in EXPTIME for inclusion hold because, given ECVPAs M_1 and M_2, we can take the complement of M_2, take its intersection with M_1 and check for the emptiness. For the hardness, as shown in [3], the language inclusion problem for VPAs is EXPTIME-complete. Because a VPA can be viewed as an ECVPA whose all transitions have the trivial clock constraint *true*, which evaluates to true for all clock-valuations. Thus, the language inclusion problem for ECVPAs is EXPTIME-complete.

□

In the remaining part of this section, we briefly sketch how to model real-time verification problems using ECVPAs. Suppose that we are given a program

P with procedures (or method) that can call one another. We choose a suitable pushdown alphabet $\Sigma = \langle \Sigma_c, \Sigma_r, \Sigma_i \rangle$, assign a set of event-clocks corresponding to Σ, and associate a symbol with every transition of P with the restriction that calls are mapped to Σ_c, returns are mapped to Σ_r, and all other statements are mapped to Σ_i. Then, P can be viewed as an ECVPA over pushdown alphabet Σ, generated timed language $TL(P)$. The real-time specification is given as another ECVPA S over the same alphabet, and the program is correct iff $TL(P) \subseteq TL(S)$. Requirements that can be checked in this approach include common regular real-time properties specified by ECAs [2], and non-regular real-time properties such as: partial real-time correctness (if p holds when a procedure is invoked, then within d time units, if the procedure returns, q holds upon return), total real-time correctness (if p holds when a procedure is invoked, then the procedure must return within d time units and q must hold at the return state), and local real-time properties (every request is followed by a response within d time units). Moreover, Theorem 6 can be extended to verify whether a program described as a *visibly pushdown timed automaton* (PTAs with visibility on input alphabet) satisfies a specification given as an ECVPA.

6 Event-Clock Büchi Visibly Pushdown Automata

In this section, we consider extensions of the results of ECVPAs to ECVPAs with Büchi acceptance condition. We show that event-clock Büchi visibly pushdown automata are closed under union, intersection, and complementation.

An *infinite timed word* over an alphabet Σ is a member $\bar{w} = (a_0, t_0)(a_1, t_1) \cdots$ of $(\Sigma \times R^+)^\omega$ such that $t_i < t_{i+1}$, for each $i \in N$. Let $T\Sigma^\omega$ denote the set of infinite timed words over Σ. Clock valuation for an infinite timed word $\bar{w}$ is defined using the natural extension of Definition 2 of clock valuation for finite timed word.

Definition 8. *A Büchi ECVPA over Σ is a tuple $M = \langle Q, \Sigma, C_\Sigma, Q_0, \Gamma, \delta, Q_f \rangle$ whose components agree with those of a finite ECVPA given in Definition 4.*

For any infinite timed word $\bar{w} \in T\Sigma^w$, a run over $\bar{w}$ is an ω-sequence $\rho = (q_0, \sigma_0)(q_1, \sigma_1) \cdots$ that is defined using the natural extension of the definition of runs on finite timed words. To determine whether a run ρ is accepting, we consider the set $\inf(\rho) \subseteq Q$ which is the set of all locations that occur in ρ infinitely often. A run ρ is *accepting* if Q_f is met infinitely often along the run, i.e., $inf(\rho) \cap Q_f \neq \emptyset$. An infinite timed word $\bar{w}$ is accepted by M if there is an accepting run of M on $\bar{w}$. The set of all infinite timed words accepted by M is called the the *timed language* of M, denoted $TL^\omega(M)$. A language of infinite timed words $L \subseteq L\Sigma^\omega$ is said to be an w-ECVPL if there is a Büchi ECVPA M such that $L = TL^\omega(M)$. By using a similar technique as for ECVPAs, we can establish a homomorphism from Büchi ECVPAs to Büchi VPAs. Since Büchi VPAs are closed under all boolean operations, we obtain

Theorem 7 (Closure properties). *The class of ω-ECVPLs is closed under union, intersection, and complementation.*

7 Related works

There is an extensive body of works on formal verification of non-regular properties recently. Especially, VPAs were introduced in [3] as a formal framework for pushdown model checking non-regular properties. The synchronization of VPAs have been generalized to synchronized pushdown automata by Caucal [6], and to height-deterministic pushdown automata by Nowotka and Srba [13]. VPAs with multiple stacks have been considered recently by La Torre, Madhusudan and Parlato [15], who exploit the concept of visibility further to obtain a rich pushdown-automata class that can even model multi-thread programs and express properties beyond the class of context-free languages. At the same time, their automata are closed under boolean operations, come up with a decidable emptiness and inclusion problem. These results require a restricted version of VPAs with multiple stacks whose behavior can be split up into a fixed number of phases. In a recent report [4], Bollig considered 2-stack VPAs in their unrestricted form and showed that 2-stack VPAs are expressively equivalent to the existential fragment of monadic second order logic.

Pushdown systems were used to model the state space of sequential programs with nested procedure calls. When modeling recursive sequential programs via pushdown systems, it is necessary to abstract local variables (which have to be stored on the stack) with an infinite range to finite range, in order to obtain a finite pushdown alphabet. This abstraction may lead to spurious counterexample. In [12], Kuske and Lohrey introduced pushdown systems over infinite stack alphabets. Such a pushdown system can directly model programs with nested procedure calls, where procedures use variables with an infinite domain. They also showed that the FOREG-theory of the configuration graph $C(S)$ of a pushdown systems over an infinite stack is decidable. FOREG is the extension of first-order logic which allows to define new binary predicates by regular expressions over the binary predicates of a base structure.

8 Conclusion

In this paper, we have presented ECVPAs as a subclass of pushdown timed automata. We have shown that the class of ECVPAs enjoys good closure properties and desirable decision problems. Our work has been inspired by VPAs [3] and ECAs [2]. From the automata-theoretic point of view, ECVPAs can be seen as a proper extension of both VPAs and ECAs. From the formal verification point of view, the class of ECVPAs is sufficiently expressive to model real-time pushdown systems, and to specify common non-regular context-free real-time requirements. With such advantages, we believe that the class of ECVPAs is an adequate framework for model checking common context-free real-time properties of real-time programs with recursive procedure calls.

Several future directions are interesting. First, we plan to study a logical characterization of ECVPAs. We expect that such a logic is more expressive than EventClockLTL logic [9]. Second, state complexity of finite automata has been

intensively studied, and has important practical implications in finite-automata applications (see [14], e.g,). We believe that a study of state complexity of VPAs (ECVPAs) is an interesting issue. Third, weighted automata and weighted logics have been extensively studied by Droste et al. (see [8], e.g,), and have used to modeling quantitative systems. It would be interesting to incorporate weights to ECVPAs for modeling quantitative real-time pushdown systems.

Acknowledgements. This research is supported by the 21st Century COE program "Verifiable and Evolvable e-Society" funded by the Japanese Ministry of Education, Culture, Sports, Science and Technology.

References

1. A. Alur and D. Dill. A theory of timed automata. *Theor. Comp. Sci*, 126: pp.183-235, 1994.
2. R. Alur, L. Fix and T. A. Henzinger. Event-Clock Automata: A Determinizable Class of Timed Automata. *Theor. Comp. Sci*, 211: pp.253-273, 1999.
3. R. Alur and P. Madhusudan. Visibly pushdown languages. In *STOC'04*, pp.202-211.
4. B. Bollig. On the expressive power of 2-stack visibly pushdown automata. Research Report LSV-07-27, ENS Cachan, France, September 2007.
5. A. Bouajjani, R. Echahed, R. Robbana. On the automatic verification of systems with continuous variables and unbounded discrete data structures. In *Hybrid Systems II*, LNCS 999, pp. 64-85, 1995.
6. D. Caucal. Synchronization of pushdown automata. In *DLT'06*, LNCS 4036, pp. 120-132, 2006.
7. E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.
8. M. Droste and P. Gastin. Weighted Automata and Weighted Logic. *Theor. Comp. Sci*, 380: pp. 69-86, 2007.
9. T. A. Henzinger, J. Raskin, and P. Schobbens. The regular real-time languages. In *ICALP'98*, LNCS 1443, pp.580-598, 1998.
10. O. Kupferman, N. Piterman, and M. Vardi. Pushdown Specifications. In *LPAR'02*, LNCS 2514, pp. 262-277, 2002.
11. J. Ouaknine and J. Worrell. On the Language Inclusion Problem for Timed Automata: Closing a Decidability Gap. In *LICS'04*, pp. 54-63, 2004.
12. D. Kuske and M. Lohrey. Monadic chain logic over iterations and applications to pushdown systems. In *LICS'06*, pp. 91-100, 2006.
13. D. Nowotka and J. Srba. Height-Deterministic Pushdown Automata. In *MFCS'07*, LNCS 4708, pp. 125-134, 2007.
14. A. Salomaa, K. Salomaa, S. Yu. State complexity of combined operations. Theor. Comput. Sci, 383: pp. 140-152, 2007.
15. S. La Torre, P. Madhusudan, G. Parlato. A robust class of context-sensitive languages. In *LICS'07*, pp. 161-170, 2007.

A Proof of Theorem 3

Proof. Obviously, (1) holds. Recall the ECVPA M in Example 1, the timed language of M cannot be accepted by any timed automata. This proves $ECVPAs \nsubseteq$

TAs, (2) holds. To prove (3), consider the timed automaton A in Figure 3, which uses a single clock x to accept timed words over the unary alphabet $\{a\}$. The timed language $L(A)$ consists of all timed words of the form (a^k, t), for $k \geq 2$, such that $t_j - t_i = 1$ for some $0 \leq i < j < k$. However, this timed language cannot be accepted by any ECVPA. Hence $TAs \nsubseteq ECVPAs$. Finally, as shown in [2], an event-clock automaton can be transformed into a timed automaton. By using a similar technique, one can transform an ECVPA into a pushdown timed automaton. This indicates that $ECVPAs \subsetneq PTAs$, (4) holds.

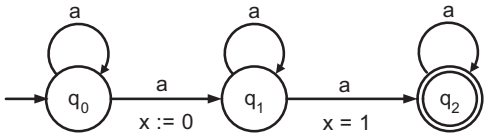

Fig. 3. Timed automaton A

□

B Proof of Lemma 2

Proof. First, we prove $TL(M) \subseteq tw(L(ut(M))$. Let $\bar{w} = (a_1, t_1)(a_2, t_2) \cdots (a_n, t_n) \in TL(M)$. There is an accepting run $\rho = (q_0, \sigma_0)(q_1, \sigma_1) \cdots (q_n, \sigma_n)$ of M on $\bar{w}$, where $q_n \in Q_f$. By Definition 6, we see that ρ is also a run of $ut(M)$ on $\lambda = (a_1, g_1) \cdots (a_n, g_n)$. Namely, $\lambda \in L(ut(M))$. Because $\bar{w} \in tw(\lambda)$, so $TL(M) \subseteq tw(L(ut(M))$ (1). Conversely, for each $\lambda' = (b_1, g'_1)(b_2, g'_2) \cdots (b_n, g'_n) \in L(ut(M))$, it is easy to verify that $\bar{w}' \in TL(M)$ for all $\bar{w}' \in tw(\lambda')$. This indicates that $tw(L(ut(M)) \subseteq TL(M)$ (2). From (1) and (2), we have $TL(M) = tw(L(ut(M))$. □

C Proof of Lemma 3

Proof. First, we prove that $tw(L(N) \subseteq TL(ec(N))$. For each $\lambda = (a_1, g_1)(a_2, g_2) \cdots (a_n, g_n) \in L(N)$, if $\rho = (q_0, \sigma_0)(q_1, \sigma_1) \cdots (q_n, \sigma_n)$ is a run of N on λ then ρ is also a run of $ec(N)$ on $\bar{w}$ for all $\bar{w} \in tw(\lambda)$. Namely, $tw(L(N) \subseteq TL(ec(N))$ (1). Conversely, let $\bar{w} = (a_1, t_1)(a_2, t_2) \cdots (a_n, t_n) \in TL(ec(N))$. There is an accepting run $\rho = (q_0, \sigma_0)(q_1, \sigma_1) \cdots (q_n, \sigma_n)$ of M on $\bar{w}$, where $q_n \in Q_f$. By Definition 7, we can find an untimed word $\lambda = (a_1, g_1) \cdots (a_n, g_n) \in \Lambda^*$ such that $\bar{w} \in tw(\lambda)$ and ρ is also a run of N on λ. Thus, $TL(ec(N) \subseteq tw(L(N))$ (2). The lemma follows from (1) and (2). □

D Another Proof of Theorem 4

Proof. This proof [2] uses the same basic technique as for determinizing VPAs [3]. The construction has two components: a set of *summary edges* S, that keeps track of what state transitions are possible from a push transition to the corresponding pop transition, and a set of *path edges* R, that keep track of all possible location reached from initial locations. In our real-time case, we have to consider clock constraints. Thus, we associate each transition of the target automaton with a suitable clock constraint.

Let $M = \langle Q, \Sigma, C_\Sigma, Q_0, \Gamma, \delta, F\rangle$. We first sketch the scenario by the following example. Let $\bar{w} = \bar{w}_1\bar{a}_1\bar{w}_2\bar{a}_2\bar{w}_3$ be a timed word, where in $\bar{w}_1$ every call is matched by a return, where there may be unmatched returns; $\bar{w}_2$ and $\bar{w}_3$ are timed words in which all calls and returns are matched; and $\bar{a}_1 = (a, t)$, $\bar{a}_2 = (a_2, t')$, where a_1 and a_2 are calls (that do not have matching returns in $\bar{w}$). Then after reading $\bar{w}$, the ECVPA we construct will have as its stack $(S_2, R_2, a_2)(S_1, R_1, a_1)\ \bot$, and its control location will be (S, R). Here, the set S_2 is the summary for $\bar{w}_2$, i.e., $S_2 = \{(q, q') \mid \exists q_0 \in Q_0 : (q_0, \sigma_0) \xrightarrow{\bar{w}_1\bar{a}_1} (q, \sigma) \xrightarrow{\bar{w}_2} (q', \sigma')\}$. Similarly, S_1 is the summary for $\bar{w}_1$ and S is the summary for $\bar{w}_3$. The set R_1 is the set of locations reachable by M from any initial location on $\bar{w}_1$, R_2 is the set of locations reachable from any initial location on $\bar{w}_1\bar{a}_1\bar{w}_2$, and R is the set of locations reached by M after reading $\bar{w}$.

When reading the next symbol $\bar{b} = (b, t'')$, there are there cases: b is (1) a call, (2) a return, (3) internal. If b is a call, we push (S, R, b), update R using all possible transitions on $\bar{b}$ to get R', and go to the new location (Id_Q, R'), where $Id_Q = \{(q, q) \mid q \in Q\}$ is the initialization of the summary. If b is a matching return of a_2, we pop (S_2, R_2, a_2) and update S_2 and R_2 using the current summary S along with a push-transition on a_2 and a corresponding pop-transition on $\bar{b}$. If b is an internal action, we update S' and R' by subset construction using all possible transitions on $\bar{b}$ (no stack operations).

We now formally define the ECVPA $Det(M) = \langle Q', \Sigma, C_\Sigma, Q'_0, \Gamma', \delta', F'\rangle$ as follows: $Q' = 2^{Q\times Q} \times 2^Q$, $Q'_0 = \{(Id_Q, Q_0)\}$ where $Id_Q = \{(q, q)|q \in Q\}$, $F' = \{(S, R)|R \cap F \neq \emptyset\}$, Γ' is the set of tuples (S, R, a) where $S \subseteq Q \times Q$, $R \subseteq Q$ and $a \in \Sigma_c$. The transition relation $\delta' = \delta'_c \cup \delta'_r \cup \delta'_i$ is given by:

- **Internal:** For each $(S, R) \in Q'$ and an internal symbol $a \in \Sigma_i$, let $E \subseteq \delta_i$ be the set of all internal transitions of M whose source locations are in R and whose input symbol is a. Then, for every non-empty subset $E' \subseteq E$, there is an internal transition $((S, R), a, \chi_E^{E'}, (S', R')) \in \delta'_i$, where $\chi_E^{E'}$ [3], and (S', R') satisfies the following conditions:

$$\begin{cases} S' = \{(q, q')|\exists q'' \in Q : (q, q'') \in S, (q'', a, \varphi, q') \in E'\} \\ R' = \{q'|\exists q \in R : (q, a, \varphi, q') \in E'\} \end{cases}$$

[2] This proof gives a direct determinization of ECVPA, by this way, the determinized automaton has $O(2^{n^2})$ locations and $O(2^{n^2} \cdot |\Sigma_c|)$ stack symbols.

[3] In the whole of this proof, we denote $\chi_E^{E'}$ is the conjunction of all clock constraints of transitions in E' and all negated clock constraints of transitions in $(E \setminus E')$.

– **Push:** For each $(S,R) \in Q'$ and a call symbol $a \in \Sigma_c$, let $E \subseteq \delta_c$ be the set of all call transitions of M whose source locations are in R and whose input symbol is a. Then, for every non-empty subset $E' \subseteq E$, there is a call transition $((S,R), a, \chi_E^{E'}, (Id_Q, R'), (S,R,a)) \in \delta'_c$ where R' satisfies the following conditions:

$$R' = \left\{ q' \,\middle|\, \exists q \in R, \gamma \in \Gamma \setminus \{\bot\} : (q, a, \varphi, q', \gamma) \in E' \right\}$$

– **Pop:** For each $(S,R) \in Q'$ and a return symbol $a \in \Sigma_r$, let $E \subseteq \delta_r$ be the set of all return transitions of M whose source locations are in R and whose input symbol is a. Then, for every non-empty subset $E' \subseteq E$, there is a return transition from (S,R) to (S'',K'') with the input a such that:

- if the stack is empty : $((S,R), a, \chi_E^{E'}, \bot, (S'', R'')) \in \delta'_r$ where

$$\begin{cases} S'' = \{(q,q') \mid \exists q'' \in Q : (q,q'') \in S, (q'', a, \varphi, \bot, q') \in E'\} \\ R'' = \{q' \mid \exists q \in R : (q, a, \varphi, \bot, q') \in E'\} \end{cases}$$

- otherwise: $((S,R), a, \chi_E^{E'}, (S', R', a'), (S'', R'')) \in \delta'_r$, where

$$\begin{cases} S'' & = \{(q,q') \mid \exists q_3 : (q,q_3) \in S', (q_3,q') \in Update\} \\ R'' & = \{q' \mid \exists q \in R' : (q,q') \in Update\} \\ Update = \left\{ (q,q') \,\middle|\, \begin{array}{l} \exists q_1, q_2 \in Q, \gamma \in \Gamma : (q, a', \varphi_1, q_1, \gamma) \in \delta_c, \\ (q_1, q_2) \in S, (q_2, a, \varphi_2, \gamma, q') \in E' \end{array} \right\} \end{cases}$$

□