Event-Clock Nested Automata

Laura Bozzelli, Aniello Murano, and Adriano Peron

Università degli Studi di Napoli Federico II, Italy

Abstract. In this paper we introduce and study Event-Clock Nested Automata (ECNA), a formalism that combines Event Clock Automata (ECA) and Visibly Pushdown Automata (VPA). ECNA allow to express real-time properties over non-regular patterns of recursive programs. We prove that ECNA retain the closure and decidability properties of ECA and VPA being closed under Boolean operations and having a decidable language-inclusion problem. In particular, we prove that emptiness, universality, and language-inclusion for ECNA are EXPTIME-complete problems. As for the expressiveness, we have that ECNA properly extend any previous attempt in the literature of combining ECA and VPA.

1 Introduction

Model checking is a well-established formal-method technique to automatically check for global correctness of reactive systems [7]. In this setting, automata theory over infinite words plays a crucial role: the set of possible (potentially infinite) behaviors of the system and the set of admissible behaviors of the correctness specification can be modeled as languages accepted by automata. The verification problem of checking that a system meets its specification then reduces to testing language inclusion between two automata over infinite words.

In the last two decades, model checking of pushdown automata (PDA) has received a lot of attention [11, 17, 20]. PDA represent an infinite-state formalism suitable to model the control flow of typical sequential programs with nested and recursive procedure calls. Although the general problem of checking context-free properties of PDA is undecidable [16], algorithmic solutions have been proposed for interesting subclasses of context-free requirements [3, 5, 6, 13]. A well-known approach is that of *Visibly Pushdown Automata* (VPA) [5, 6], a subclass of PDA where the input symbols over a *pushdown alphabet* control the admissible operations on the stack. Precisely, the alphabet is partitioned into a set of *calls*, representing a procedure call and forcing a push stack-operation, a set of *returns*, representing a procedure return and forcing a pop stack-operation, and a set of *internal* actions that cannot access or modify the content of the stack. This restriction makes the class of resulting languages (*visibly pushdown languages* or VPL) very similar in tractability and robustness to that of regular languages [5, 6]. VPL are closed under Boolean operations, and language inclusion is EXPTIME-complete. VPA capture all regular properties, and, additionally, allow to specify regular requirements over two kinds of *non-regular* patterns on input words: *abstract paths* and *caller paths*. An abstract path captures the local computation within a procedure with the removal of subcomputations corresponding to nested procedure calls, while a caller path represents the call-stack content at a given position of the input.

Recently, many works [1, 8, 10, 14, 15, 19] have investigated real-time extensions of PDA by combining PDA with *Timed Automata* (TA) [2], a model widely used to represent real-time systems. TA are finite automata augmented with a finite set of real-valued clocks, which operate over words where each symbol is paired with a real-valued timestamp (*timed words*). All clocks progress at same speed and can be reset by transitions (thus, each clock keeps track of the elapsed time since the last reset). Constraints on clocks are associated with transitions to restrict the behavior of the automaton. The emptiness problem for TA is decidable and Pspace complete [2]. However, since in TA, clocks can be reset nondeterministically and independently of each other, the resulting class of timed languages is not closed under complement and, in particular, language inclusion is undecidable [2]. As a consequence, the general verification problem (i.e., language inclusion) of formalisms combining unrestricted TA with robust subclasses of PDA such as VPA, i.e. *Visibly Pushdown Timed Automata* (VPTA), is undecidable as well. In fact, checking language inclusion for VPTAis undecidable even in the restricted case of specifications using at most one clock [15].

Event-clock automata (ECA) [4] are an interesting subclass of TA where the explicit reset of clocks is disallowed. In ECA, clocks have a predefined association with the input alphabet symbols. Precisely, for each symbol a, there are two clocks: the *global recorder clock*, recording the time elapsed since the last occurrence of a, and the *global predictor clock*, measuring the time required for the next occurrence of a. Hence, the clock valuations are determined only by the input timed word being independent of the automaton behavior. Such a restriction makes the resulting class of timed languages closed under Boolean operations, and in particular, language inclusion is Pspace-complete [4].

Recently, a robust subclass of VPTA, called *Event-Clock Visibly Pushdown Automata* (ECVPA), has been proposed in [18], combining ECA with VPA. ECVPA are closed under Boolean operations, and language inclusion is Exptime-complete. However, ECVPA do not take into account the nested hierarchical structure induced by a timed word over a pushdown alphabet, namely, they do not provide any explicit mechanism to relate the use of a stack with that of event clocks.

Our contribution. In this paper, we introduce an extension of ECVPA, called *Event-Clock Nested Automata* (ECNA) that, differently from ECVPA, allows to relate the use of event clocks and the use of the stack. To this end, we add for each input symbol a, three additional event clocks: the *abstract recorder clock* (resp., *abstract predictor clock*), measuring the time elapsed since the last occurrence (resp., the time for the next occurrence) of a along the maximal abstract path visiting the current position; the *caller clock*, measuring the time elapsed since the last occurrence of a along the caller path from the current position. In this way, ECNA allow to specify relevant real-time non-regular properties including:

– Local bounded-time responses such as "in the local computation of a procedure A, every request p is followed by a response q within k time units".

– Bounded-time total correctness requirements such as “if the pre-condition p holds when the procedure A is invoked, then the procedure must return within k time units and q must hold upon return”.
– Real-time security properties which require the inspection of the call-stack such as “a module A should be invoked only if module B belongs to the call stack and within k time units since the activation of module B”.

We show that ECNA are strictly more expressive than ECVPA and, as for ECVPA, the resulting class of languages is closed under all Boolean operations. Moreover, language inclusion and visibly model-checking of VPTA against ECNA specifications are decidable and EXPTIME-complete. The key step in the proposed decision procedures is a translation of ECNA into equivalent VPTA.

Related work. Pushdown Timed Automata (PTA) have been introduced in [10], and their emptiness problem is EXPTIME-complete. An extension of PTA, namely Dense-Timed Pushdown Automata (DTPA), has been studied in [1], where each symbol in the stack is equipped with a real-valued clock representing its ‘age’ (the time elapsed since the symbol has been pushed onto the stack). It has been shown in [14] that DTPA do not add expressive power and can be translated into equivalent PTA. Our proposed translation of ECNA into VPTA is inspired from the construction in [14]. In [9], an equally-expressive extension of ECVPA [18] over finite timed words, by means of a *timed* stack (like in DTPA), is investigated.

2 Preliminaries

In the following, $\mathbb{N}$ denotes the set of natural numbers and $\mathbb{R}_+$ the set of non-negative real numbers. Let w be a finite or infinite word over some alphabet. By $|w|$ we denote the length of w (we set $|w| = \infty$ if w is infinite). For all $i, j \in \mathbb{N}$, with $i \leq j$, w_i is i-th letter of w, while $w[i, j]$ is the finite subword $w_i \cdots w_j$.

An *infinite timed word* w over a finite alphabet Σ is an infinite word $w = (a_0, \tau_0)(a_1, \tau_1), \ldots$ over $\Sigma \times \mathbb{R}_+$ (intuitively, τ_i is the time at which a_i occurs) such that the sequence $\tau = \tau_0, \tau_1, \ldots$ of timestamps satisfies: (1) $\tau_i \leq \tau_{i+1}$ for all $i \geq 0$ (monotonicity), and (2) for all $t \in \mathbb{R}_+$, $\tau_i \geq t$ for some $i \geq 0$ (divergence). The timed word w is also denoted by the pair (σ, τ), where σ is the untimed word $a_0 a_1 \ldots$ and τ is the sequence of timestamps. An *ω-timed language* over Σ is a set of infinite timed words over Σ.

Pushdown alphabets, abstract paths, and caller paths. A *pushdown alphabet* is a finite alphabet $\Sigma = \Sigma_{call} \cup \Sigma_{ret} \cup \Sigma_{int}$ which is partitioned into a set Σ_{call} of *calls*, a set Σ_{ret} of *returns*, and a set Σ_{int} of *internal actions*. The pushdown alphabet Σ induces a nested hierarchical structure in a given word over Σ obtained by associating to each call the corresponding matching return (if any) in a well-nested manner. Formally, the set of *well-matched words* is the set of finite words σ_w over Σ inductively defined by the following grammar:

$$\sigma_w := \varepsilon \,\big|\, a \cdot \sigma_w \,\big|\, c \cdot \sigma_w \cdot r \cdot \sigma_w$$

where ε is the empty word, $a \in \Sigma_{int}$, $c \in \Sigma_{call}$, and $r \in \Sigma_{ret}$.

Fix an infinite word σ over Σ. For a call position $i \geq 0$, if there is $j > i$ such that j is a return position of σ and $\sigma[i+1, j-1]$ is a well-matched word (note that j is uniquely determined if it exists), we say that j is the *matching return* of i along σ. For a position $i \geq 0$, the *abstract successor of i along σ*, denoted $\mathsf{succ}(\mathsf{a}, \sigma, i)$, is defined as follows:

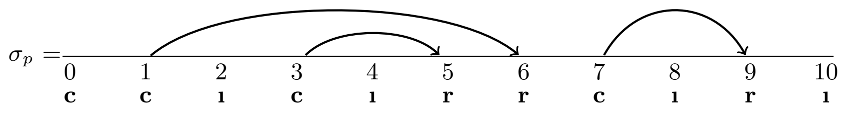
- If i is a call, then $\mathsf{succ}(\mathsf{a}, \sigma, i)$ is the matching return of i, if such a matching return exists; otherwise, $\mathsf{succ}(\mathsf{a}, \sigma, i) = \perp$ ($\perp$ denotes the *undefined* value).
- If i is not a call, then $\mathsf{succ}(\mathsf{a}, \sigma, i) = i+1$ if $i+1$ is not a return position, and $\mathsf{succ}(\mathsf{a}, \sigma, i) = \perp$, otherwise.

The *caller of i along σ*, denoted $\mathsf{succ}(\mathsf{c}, \sigma, i)$, is instead defined as follows:

- if there exists the greatest call position $j_c < i$ such that either $\mathsf{succ}(\mathsf{a}, \sigma, j_c) = \perp$ or $\mathsf{succ}(\mathsf{a}, \sigma, j_c) > i$, then $\mathsf{succ}(\mathsf{c}, \sigma, i) = j_c$; otherwise, $\mathsf{succ}(\mathsf{c}, \sigma, i) = \perp$.

In the analysis of recursive programs, a *maximal abstract path* captures the local computation within a procedure removing computation fragments corresponding to nested calls, while the *caller path* represents the call-stack content at a given position of the input. Formally, a *maximal abstract path* (*MAP*) of σ is a *maximal* (finite or infinite) increasing sequence of natural numbers $\nu = i_0 < i_1 < \ldots$ such that $i_j = \mathsf{succ}(\mathsf{a}, \sigma, i_{j-1})$ for all $1 \leq j < |\nu|$. Note that for every position i of σ, there is exactly one *MAP* of σ visiting position i. For each $i \geq 0$, the *caller path of σ from position i* is the maximal (finite) decreasing sequence of natural numbers $j_0 > j_1 \ldots > j_n$ such that $j_0 = i$ and $j_{h+1} = \mathsf{succ}(\mathsf{c}, \sigma, j_h)$ for all $0 \leq h < n$. Note that the positions of a *MAP* have the same caller (if any).

For instance, consider the finite untimed word σ_p of length 10 depicted below where $\Sigma_{call} = \{c\}$, $\Sigma_{ret} = \{r\}$, and $\Sigma_{int} = \{\imath\}$.



Let σ be $\sigma_p \cdot \imath^\omega$. Note that 0 is the unique unmatched call position of σ: hence, the *MAP* visiting 0 consists of just position 0 and has no caller. The *MAP* visiting position 1 is the infinite sequence $1, 6, 7, 9, 10, 11, 12, 13\ldots$ and the associated caller is position 0; the *MAP* visiting position 2 is the sequence $2, 3, 5$ and the associated caller is position 1, and the *MAP* visiting position 4 consists of just position 4 whose caller path is $4, 3, 1, 0$.

3 Event-clock nested automata

In this section, we define the formalism of *Event-Clock Nested Automata* (ECNA), which allow a combined used of event clocks and visible operations on the stack. To this end, we augment the standard set of event clocks [4] with a set of *abstract event clocks* and a set of *caller event clocks* whose values are determined by considering maximal abstract paths and caller paths of the given word, respectively.

In the following, we fix a pushdown alphabet $\Sigma = \Sigma_{call} \cup \Sigma_{ret} \cup \Sigma_{int}$. The set C_Σ of event clocks associated with Σ is given by $C_\Sigma := \bigcup_{b \in \Sigma} \{x_b^{\mathsf{g}}, y_b^{\mathsf{g}}, x_b^{\mathsf{a}}, y_b^{\mathsf{a}}, x_b^{\mathsf{c}}\}$.

Thus, we associate with each symbol $b \in \Sigma$, five event clocks: the *global recorder clock* x_b^{g} (resp., the *global predictor clock* y_b^{g}) recording the time elapsed since the last occurrence of b, if any, (resp., the time required to the next occurrence of b if any); the *abstract recorder clock* x_b^{a} (resp., the *abstract predictor clock* y_b^{a}) recording the time elapsed since the last occurrence of b, if any, (resp. the time required to the next occurrence of b) along the *MAP* visiting the current position; and the *caller (recorder) clock* x_b^{c} recording the time elapsed since the last occurrence of b if any along the caller path from the current position. Let $w = (\sigma, \tau)$ be an infinite timed word over Σ and $i \geq 0$. We denote by $Pos(\mathsf{a}, w, i)$ the set of positions visited by the *MAP* of σ associated with position i, and by $Pos(\mathsf{c}, w, i)$ the set of positions visited by the caller path of σ from position i. In order to allow a uniform notation, we write $Pos(\mathsf{g}, w, i)$ to mean the full set $\mathbb{N}$ of positions. Fixed the input word w, when the automaton reads the i-th position σ_i at time τ_i, the values of the clocks are uniquely determined as follows.

Definition 1 (Input determinisitic clock valuations). A *clock valuation* over C_Σ is a mapping $val : C_\Sigma \mapsto \mathbb{R}_+ \cup \{\bot\}$, assigning to each event clock a value in $\mathbb{R}_+ \cup \{\bot\}$ ($\bot$ denotes the *undefined* value). Given an infinite timed word $w = (\sigma, \tau)$ over Σ and a position i, the *clock valuation* val_i^w *over* C_Σ, specifying the values of the event clocks at position i along w, is defined as follows for each $b \in \Sigma$, where $dir \in \{\mathsf{g}, \mathsf{a}, \mathsf{c}\}$ and $dir' \in \{\mathsf{g}, \mathsf{a}\}$:

$$val_i^w(x_b^{dir}) = \begin{cases} \tau_i - \tau_j & \text{if } \exists j < i : \, b = \sigma_j, \, j \in Pos(dir, w, i), \text{ and} \\ & \quad \forall k : (j < k < i \text{ and } k \in Pos(dir, w, i)) \Rightarrow b \neq \sigma_k \\ \bot & \text{otherwise} \end{cases}$$

$$val_i^w(y_b^{dir'}) = \begin{cases} \tau_j - \tau_i & \text{if } \exists j > i : \, b = \sigma_j, \, j \in Pos(dir', w, i), \text{ and} \\ & \quad \forall k : (i < k < j \text{ and } k \in Pos(dir', w, i)) \Rightarrow b \neq \sigma_k \\ \bot & \text{otherwise} \end{cases}$$

It is worth noting that while the values of the global clocks are obtained by considering the full set of positions in w, the values of the abstract clocks (resp., caller clocks) are defined with respect to the *MAP* visiting the current position (resp., with respect to the caller path from the current position).

For $C \subseteq C_\Sigma$ and a clock valuation val over C_Σ, $val_{|C}$ denotes the restriction of val to the set C. A *clock constraint* over C is a conjunction of atomic formulas of the form $z \in I$, where $z \in C$, and I is either an interval in $\mathbb{R}_+$ with bounds in $\mathbb{N} \cup \{\infty\}$, or the singleton $\{\bot\}$ (also denoted by $[\bot, \bot]$). For a clock valuation val and a clock constraint θ, val satisfies θ, written $val \models \theta$, if for each conjunct $z \in I$ of θ, $val(z) \in I$. We denote by $\Phi(C)$ the set of clock constraints over C.

For technical convenience, we first introduce an extension of the known class of *Visibly Pushdown Timed Automata* (VPTA) [10, 15], called *nested* VPTA. Nested VPTA are simply VPTA augmented with event clocks. Therefore, transitions of *nested* VPTA are constrained by a pair of disjoint finite sets of clocks: a finite set C_{st} of standard clocks and a disjoint set $C \subseteq C_\Sigma$ of event clocks. As usual, a standard clock can be reset when a transition is taken; hence, its value at a position of an input word depends in general on the behaviour of the automaton and not only, as for event clocks, on the word.

The class of *Event-Clock Nested Automata* (ECNA) corresponds to the subclass of nested VPTA where the set of standard clocks C_{st} is empty.

A (standard) clock valuation over C_{st} is a mapping $sval : C_{st} \mapsto \mathbb{R}_+$ (note that the undefined value $\perp$ is not admitted). For $t \in \mathbb{R}_+$ and a reset set $Res \subseteq C_{st}$, $sval + t$ and $sval[Res]$ denote the valuations over C_{st} defined as follows for all $z \in C_{st}$: $(sval + t)(z) = sval(z) + t$, and $sval[Res](z) = 0$ if $z \in Res$ and $sval[Res](z) = sval(z)$ otherwise. For $C \subseteq C_\Sigma$ and a valuation val over C, $val \cup sval$ denotes the valuation over $C_{st} \cup C$ defined in the obvious way.

Definition 2 (Nested VPTA). A Büchi *nested* VPTA over $\Sigma = \Sigma_{call} \cup \Sigma_{int} \cup \Sigma_{ret}$ is a tuple $\mathcal{A} = (\Sigma, Q, Q_0, D = C \cup C_{st}, \Gamma \cup \{\top\}, \Delta, F)$, where Q is a finite set of (control) states, $Q_0 \subseteq Q$ is a set of initial states, $C \subseteq C_\Sigma$ is a set of event clocks, C_{st} is a set of standard clocks disjoint with C_Σ, $\Gamma \cup \{\top\}$ is a finite stack alphabet, $\top \notin \Gamma$ is the special *stack bottom symbol*, $F \subseteq Q$ is a set of accepting states, and $\Delta_c \cup \Delta_r \cup \Delta_i$ is a transition relation, where ($D = C \cup C_{st}$):
- $\Delta_c \subseteq Q \times \Sigma_{call} \times \Phi(D) \times 2^{C_{st}} \times Q \times \Gamma$ is the set of *push transitions*,
- $\Delta_r \subseteq Q \times \Sigma_{ret} \times \Phi(D) \times 2^{C_{st}} \times (\Gamma \cup \{\top\}) \times Q$ is the set of *pop transitions*,
- $\Delta_i \subseteq Q \times \Sigma_{int} \times \Phi(D) \times 2^{C_{st}} \times Q$ is the set of *internal transitions.*

We now describe how a nested VPTA $\mathcal{A}$ behaves over an infinite timed word w. Assume that on reading the i-th position of w, the current state of $\mathcal{A}$ is q, val_i^w is the event-clock valuation associated with w and i, $sval$ is the current valuation of the standard clocks in C_{st}, and $t = \tau_i - \tau_{i-1}$ is the time elapsed from the last transition (where $\tau_{-1} = 0$). If $\mathcal{A}$ reads a call $c \in \Sigma_{call}$, it chooses a push transition of the form $(q, c, \theta, Res, q', \gamma) \in \Delta_c$ and pushes the symbol $\gamma \neq \top$ onto the stack. If $\mathcal{A}$ reads a return $r \in \Sigma_{ret}$, it chooses a pop transition of the form $(q, r, \theta, Res, \gamma, q') \in \Delta_r$ such that γ is the symbol on top of the stack, and pops γ from the stack (if $\gamma = \top$, then γ is read but not removed). Finally, on reading an internal action $a \in \Sigma_{int}$, $\mathcal{A}$ chooses an internal transition of the form $(q, a, \theta, Res, q') \in \Delta_i$, and, in this case, there is no operation on the stack. Moreover, in all the cases, the constraint θ of the chosen transition must be fulfilled by the valuation $(sval + t) \cup (val_i^w)_{|C}$, the control changes from q to q', and all the standard clocks in Res are reset (i.e., the valuation of the standard clocks is updated to $(sval + t)[Res]$).

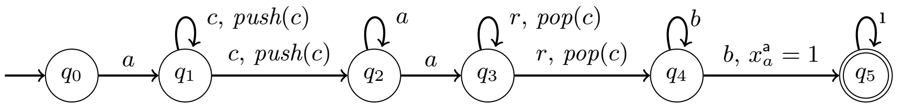
Formally, a configuration of $\mathcal{A}$ is a triple $(q, \beta, sval)$, where $q \in Q$, $\beta \in \Gamma^* \cdot \{\top\}$ is a stack content, and $sval$ is a valuation over C_{st}. A run π of $\mathcal{A}$ over $w = (\sigma, \tau)$ is an infinite sequence of configurations $\pi = (q_0, \beta_0, sval_0), (q_1, \beta_1, sval_1), \ldots$ such that $q_0 \in Q_0$, $\beta_0 = \top$, $sval_0(z) = 0$ for all $z \in C_{st}$ (initialization requirement), and the following holds for all $i \geq 0$, where $t_i = \tau_i - \tau_{i-1}$ ($\tau_{-1} = 0$):

- **Push:** If $\sigma_i \in \Sigma_{call}$, then for some $(q_i, \sigma_i, \theta, Res, q_{i+1}, \gamma) \in \Delta_c$, $\beta_{i+1} = \gamma \cdot \beta_i$, $sval_{i+1} = (sval_i + t_i)[Res]$, and $(sval_i + t_i) \cup (val_i^w)_{|C} \models \theta$.
- **Pop:** If $\sigma_i \in \Sigma_{ret}$, then for some $(q_i, \sigma_i, \theta, Res, \gamma, q_{i+1}) \in \Delta_r$, $sval_{i+1} = (sval_i + t_i)[Res]$, $(sval_i + t_i) \cup (val_i^w)_{|C} \models \theta$, and *either* $\gamma \neq \top$ and $\beta_i = \gamma \cdot \beta_{i+1}$, *or* $\gamma = \beta_i = \beta_{i+1} = \top$.
- **Internal:** If $\sigma_i \in \Sigma_{int}$, then for some $(q_i, \sigma_i, \theta, Res, q_{i+1}) \in \Delta_i$, $\beta_{i+1} = \beta_i$, $sval_{i+1} = (sval_i + t_i)[Res]$, and $(sval_i + t_i) \cup (val_i^w)_{|C} \models \theta$.

The run π is *accepting* if there are infinitely many positions $i \geq 0$ such that $q_i \in F$. The *timed language* $\mathcal{L}_T(\mathcal{A})$ of $\mathcal{A}$ is the set of infinite timed words w over Σ such that there is an accepting run of $\mathcal{A}$ on w. The *greatest constant of* $\mathcal{A}$, denoted $K_{\mathcal{A}}$, is the greatest natural number used as bound in some clock constraint of $\mathcal{A}$. For technical convenience, we also consider nested VPTA equipped with a *generalized Büchi acceptance condition* $\mathcal{F}$ consisting of a family of sets of accepting states. In such a setting, a run π is accepting if for each Büchi component $F \in \mathcal{F}$, the run π visits infinitely often states in F.

A VPTA [15] corresponds to a nested VPTA whose set C of event clocks is empty. An *ECNA* is a nested VPTA whose set C_{st} of standard clocks is empty. For ECNA, we can omit the reset component *Res* from the transition function and the valuation component *sval* from each configuration $(q, \beta, sval)$. Note the the class of Event-Clock Visibly Pushdown Automata (ECVPA) [18] corresponds to the subclass of ECNA where abstract and caller event-clocks are disallowed. We also consider three additional subclasses of ECNA: *abstract predicting* ECNA (AP_ECNA, for short) which do not use abstract recorder clocks and caller clocks, *abstract recording* ECNA (AR_ECNA, for short) which do not use abstract predictor clocks and caller clocks, and *caller* ECNA (C_ECNA, for short) which do not use abstract clocks. Note that these three subclasses of ECNA subsume ECVPA.

Example 1. Let us consider the AR_ECNA depicted below, where $\Sigma_{call} = \{c\}$, $\Sigma_{ret} = \{r\}$, and $\Sigma_{int} = \{a, b, \imath\}$. The control part of the transition relation ensures that for each accepted word, the *MAP* visiting the b-position associated with the transition *tr* from q_4 to q_5 cannot visit the a-positions following the call positions. This implies that the abstract recorder constraint $x_a^{\mathsf{a}} = 1$ associated with *tr* is fulfilled *only if* all the occurrences of calls c and returns r are matched. Hence, constraint $x_a^{\mathsf{a}} = 1$ ensures that the accepted language, denoted by $\mathcal{L}_T^{rec}$,



consists of all the timed words of the form $(\sigma, \tau) \cdot (\imath^\omega, \tau')$ such that σ is a well-matched word of the form $a \cdot c^n \cdot a^m \cdot r^n \cdot b^k$ with $n, m, k > 0$, and the time difference in (σ, τ) between the first and last symbols is 1, i.e. $\tau_{|\sigma|-1} - \tau_0 = 1$. The example shows that ECNA allow to express a meaningful real-time property of recursive systems, namely the ability of bounding the time required to perform an internal activity consisting of an unbounded number of returning recursive procedure calls.Similarly, it is easy to define an AP_ECNA accepting the timed language, denoted by $\mathcal{L}_T^{pred}$, consisting of all the timed words of the form $(\sigma, \tau) \cdot (\imath^\omega, \tau')$ such that σ is a well-matched word of the form $a^k \cdot c^n \cdot b^m \cdot r^n \cdot b$, with $n, m, k > 0$, and the time difference in (σ, τ) between the first and last symbol is 1. Finally, we consider the timed language $\mathcal{L}_T^{caller}$, which can be defined by a C_ECNA, consisting of the timed words of the form $(c, t_0) \cdot (\sigma, \tau) \cdot (\imath^\omega, \tau')$ such that σ is a well-matched word of the form $a \cdot c^n \cdot a^m \cdot r^n \cdot b^k$, with $n, m, k > 0$, and the time difference in $(c, t_0) \cdot (\sigma, \tau)$ between the first and last symbols is 1.

Closure properties of Büchi ECNA*.* As stated in the following theorem, the class of languages accepted by Büchi ECNA is closed under Boolean operations. The proof exploits a technique similar to that used in [18] to prove the analogous closure properties for ECVPA (for details, see Appendix A of [12]).

Theorem 1. *The class of ω-timed languages accepted by Büchi* ECNA *is closed under union, intersection, and complementation. In particular, given two Büchi* ECNA $\mathcal{A} = (\Sigma, Q, Q_0, C, \Gamma \cup \{\top\}, \Delta, F)$ *and* $\mathcal{A}' = (\Sigma, Q', Q'_0, C', \Gamma' \cup \{\top\}, \Delta', F')$ *over* Σ*, one can construct*

- *a Büchi* ECNA *accepting* $\mathcal{L}_T(\mathcal{A}) \cup \mathcal{L}_T(\mathcal{A}')$ *with* $|Q| + |Q'|$ *states,* $|\Gamma| + |\Gamma'| + 1$ *stacks symbols, and greatest constant* $\max(K_{\mathcal{A}}, K_{\mathcal{A}'})$*;*
- *a Büchi* ECNA *accepting* $\mathcal{L}_T(\mathcal{A}) \cap \mathcal{L}_T(\mathcal{A}')$ *with* $2|Q||Q'|$ *states,* $|\Gamma||\Gamma'|$ *stacks symbols, and greatest constant* $\max(K_{\mathcal{A}}, K_{\mathcal{A}'})$*;*
- *a Büchi* ECNA *accepting the complement of* $\mathcal{L}_T(\mathcal{A})$ *with* $2^{O(n^2)}$ *states,* $O(2^{O(n^2)} \cdot |\Sigma_{call}| \cdot |Const|^{O(|\Sigma|)})$ *stack symbols, and greatest constant* $K_{\mathcal{A}}$*, where* $n = |Q|$ *and Const is the set of constants used in the clock constraints of* $\mathcal{A}$*.*

Expressiveness results. We now summarize the expressiveness results for ECNA. First of all, the timed languages $\mathcal{L}_T^{rec}$, $\mathcal{L}_T^{pred}$, and $\mathcal{L}_T^{caller}$ considered in Example 1 and definable by AR_ECNA, AP_ECNA, and C_ECNA, respectively, can be used to prove that the three subclasses AR_ECNA, AP_ECNA, and C_ECNA of ECNA are mutually incomparable. Hence, these subclasses strictly include the class of ECVPA and are strictly included in ECNA. The incomparability result directly follows from Proposition 1 below, whose proof is in Appendix B of [12].

As for ECNA, we have that they are less expressive than Büchi VPTA. In fact, by Theorem 3 in Section 4, Büchi ECNA can be converted into equivalent Büchi VPTA. The inclusion is strict since, while Büchi ECNA are closed under complementation (Theorem 1), Büchi VPTA are not [15].

In [9], an equally-expressive extension of ECVPA over finite timed words, by means of a *timed* stack, is investigated. The Büchi version of such an extension can be trivially encoded in Büchi AR_ECNA. Moreover, the proof of Proposition 1 can also be used for showing that Büchi ECVPA with timed stack are less expressive than Büchi AR_ECNA, Büchi AP_ECNA, and Büchi C_ECNA.
The general picture of the expressiveness results is summarized by Theorem 2.

Proposition 1. *The language* $\mathcal{L}_T^{rec}$ *is* not *definable by Büchi* ECNA *which do* not *use abstract recorder clocks,* $\mathcal{L}_T^{pred}$ *is* not *definable by Büchi* ECNA *which do* not *use abstract predictor clocks, and* $\mathcal{L}_T^{caller}$ *is* not *definable by Büchi* ECNA *which do* not *use caller clocks. Moreover, the language* $\mathcal{L}_T^{rec} \cup \mathcal{L}_T^{pred} \cup \mathcal{L}_T^{caller}$ *is* not *definable by Büchi* AR_ECNA*, Büchi* AP_ECNA *and Büchi* C_ECNA*.*

Theorem 2. *The classes* AR_ECNA*,* AP_ECNA*, and* C_ECNA *are mutually incomparable, and* AP_ECNA $\cup$ AR_ECNA $\cup$ C_ECNA $\subset$ ECNA*. Moreover,*

(1) *ECVPA* $\subset$ *AR_ECNA* (2) *ECVPA* $\subset$ *AP_ECNA*
(3) *ECVPA* $\subset$ *C_ECNA* (4) *ECNA* $\subset$ *VPTA*

Note that the expressiveness results above also hold for finite timed words.

4 Decision procedures for Büchi ECNA

In this section, we first investigate emptiness, universality, and language inclusion problems for Büchi ECNA. Then, we consider the *Visibly model-checking problem against Büchi ECNA*, i.e., given a *visibly pushdown timed system* $\mathcal{S}$ over Σ (that is a Büchi VPTA where all the states are accepting) and a Büchi ECNA $\mathcal{A}$ over Σ, the problem whether $\mathcal{L}_T(\mathcal{S}) \subseteq \mathcal{L}_T(\mathcal{A})$ hold. We establish that the above mentioned problems are decidable and EXPTIME-complete. The key intermediate result is an exponential-time translation of Büchi ECNA into language-equivalent generalized Büchi VPTA. More precisely, we show that event clocks in *nested* VPTA can be removed with a single exponential blow-up.

Theorem 3 (Removal of event clocks from nested VPTA). *Given a generalized Büchi* nested *VPTA $\mathcal{A}$, one can construct in singly exponential time a generalized Büchi VPTA $\mathcal{A}'$ (which do not use event clocks) such that $\mathcal{L}_T(\mathcal{A}') = \mathcal{L}_T(\mathcal{A})$ and $K_{\mathcal{A}'} = K_{\mathcal{A}}$. Moreover, $\mathcal{A}'$ has $n \cdot 2^{O(p \cdot |\Sigma|)}$ states and $m + O(p)$ clocks, where n is the number of $\mathcal{A}$-states, m is the number of standard $\mathcal{A}$-clocks, and p is the number of* event-clock *atomic constraints used by $\mathcal{A}$.*

In the following we sketch a proof of Theorem 3. Basically, the result follows from a sequence of transformation steps all preserving language equivalence. At each step, an event clock is replaced by a set of fresh standard clocks. To remove global event clocks we use the technique from [4]. Here, we focus on the removal of an *abstract predictor* clock y_b^{a} with $b \in \Sigma$, referring to Appendix D and E of [12] for the treatment of abstract recorder clocks and caller clocks, respectively.

Fix a generalized Büchi nested VPTA $\mathcal{A} = (\Sigma, Q, Q_0, C \cup C_{st}, \Gamma \cup \{\top\}, \Delta, \mathcal{F})$ such that $y_b^{\mathsf{a}} \in C$. By exploiting nondeterminism, we can assume that for each transition *tr* of $\mathcal{A}$, there is exactly one atomic constraint $y_b^{\mathsf{a}} \in I$ involving y_b^{a} used as conjunct in the clock constraint of *tr*. If $I \neq \{\bot\}$, then $y_b^{\mathsf{a}} \in I$ is equivalent to a constraint of the form $y_b^{\mathsf{a}} \succ \ell \wedge y_b^{\mathsf{a}} \prec u$, where $\succ \in \{>, \geq\}$, $\prec \in \{<, \leq\}$, $\ell \in \mathbb{N}$, and $u \in \mathbb{N} \cup \{\infty\}$. We call $y_b^{\mathsf{a}} \succ \ell$ (resp., $y_b^{\mathsf{a}} \prec u$) a lower-bound (resp., upper-bound) constraint. Note that if $u = \infty$, the constraint $y_b^{\mathsf{a}} \prec u$ is always fulfilled, but we include it to have a uniform notation. We construct a generalized Büchi nested VPTA $\mathcal{A}_{y_b^{\mathsf{a}}}$ equivalent to $\mathcal{A}$ whose set of event clocks is $C \setminus \{y_b^{\mathsf{a}}\}$, and whose set of standard clocks is $C_{st} \cup C_{new}$, where C_{new} consists of the fresh standard clocks $z_{\succ\ell}$ (resp., $z_{\prec u}$), for each lower-bound constraint $y_b^{\mathsf{a}} \succ \ell$ (resp., upper-bound constraint $y_b^{\mathsf{a}} \prec u$) of $\mathcal{A}$ involving y_b^{a}.

We now report the basic ideas of the translation. Consider a lower-bound constraint $y_b^{\mathsf{a}} \succ \ell$. Assume that a prediction $y_b^{\mathsf{a}} \succ \ell$ is done by $\mathcal{A}$ at position i of the input word for the first time. Then, the simulating automaton $\mathcal{A}_{y_b^{\mathsf{a}}}$ exploits the standard clock $z_{\succ\ell}$ to check that the prediction holds by resetting it at position i. Moreover, if i is not a call (resp., i is a call), $\mathcal{A}_{y_b^{\mathsf{a}}}$ carries the obligation $\succ\ell$ in its control state (resp., pushes the obligation $\succ\ell$ onto the stack) in order to check that the constraint $z_{\succ\ell} \succ \ell$ holds when the next b occurs at a position j_{check} along the *MAP* ν visiting position i. We observe that:

- if a new prediction $y_b^{\mathsf{a}} \succ \ell$ is done by $\mathcal{A}$ at a position $j > i$ of ν strictly preceding j_{check}, $\mathcal{A}_{y_b^{\mathsf{a}}}$ resets the clock $z_{\succ\ell}$ at position j rewriting the old obligation. This is safe since the fulfillment of the lower-bound prediction $y_b^{\mathsf{a}} \succ \ell$ at j guarantees that prediction $y_b^{\mathsf{a}} \succ \ell$ is fulfilled at i along ν.

– If a call position $i_c \geq i$ occurs in ν before j_{check}, the next position of i_c in ν is the matching return i_r of i_c, and any *MAP* visiting a position $h \in [i_c+1, i_r-1]$ is finite and ends at a position $k < i_r$. Thus, the clock $z_{\succ\ell}$ can be safely reset to check the prediction $y_b^{\mathsf{a}} \succ \ell$ raised in positions in $[i_c+1, i_r-1]$ since this check ensures that $z_{\succ\ell} \succ \ell$ holds at position j_{check}.

Thus, previous obligations on a constraint $y_b^{\mathsf{a}} \succ \ell$ are always rewritten by more recent ones. At each position i, $\mathcal{A}_{y_b^{\mathsf{a}}}$ records in its control state the lower-bound obligations for the current *MAP* ν (i.e., the *MAP* visiting the current position i). Whenever a call i_c occurs, the lower-bound obligations are pushed on the stack in order to be recovered at the matching return i_r. If i_c+1 is not a return (i.e., $i_r \neq i_c+1$), then $\mathcal{A}_{y_b^{\mathsf{a}}}$ moves to a control state having an empty set of lower-bound obligations (position i_c+1 starts the *MAP* visiting i_c+1).

The treatment of an upper-bound constraint $y_b^{\mathsf{a}} \prec u$ is symmetric. Whenever a prediction $y_b^{\mathsf{a}} \prec u$ is done by $\mathcal{A}$ at a position i, and the simulating automaton $\mathcal{A}_{y_b^{\mathsf{a}}}$ has no obligation on the constraint $y_b^{\mathsf{a}} \prec u$, $\mathcal{A}_{y_b^{\mathsf{a}}}$ resets the standard clock $z_{\prec u}$. If i is not a call (resp., i is a call) the fresh obligation (*first*,$\prec u$) is recorded in the control state (resp., (*first*,$\prec u$) is pushed onto the stack). When, along the *MAP* ν visiting position i, the next b occurs at a position j_{check}, the constraint $z_{\prec u} \prec u$ is checked, and the obligation (*first*,$\prec u$) is removed or confirmed (in the latter case, resetting the clock $z_{\prec u}$), depending on whether the prediction $y_b^{\mathsf{a}} \prec u$ is asserted at position j_{check} or not. We observe that:

– if a new prediction $y_b^{\mathsf{a}} \prec u$ occurs in a position $j > i$ of ν strictly preceding j_{check}, $\mathcal{A}_{y_b^{\mathsf{a}}}$ simply ignores it (the clock $z_{\prec u}$ is not reset at position j) since checking the prediction $y_b^{\mathsf{a}} \prec u$ at the previous position i guarantees the fulfillment of the prediction $y_b^{\mathsf{a}} \prec u$ at the position $j > i$ along ν.

– If a call position $i_c \geq i$ occurs in ν before j_{check}, then all the predictions $y_b^{\mathsf{a}} \prec u$ occurring in a *MAP* visiting a position $h \in [i_c+1, i_r-1]$, with $i_r \leq j_{check}$ being the matching-return of i_c, can be safely ignored (i.e., $z_{\prec u}$ is not reset there) since they are subsumed by the prediction at position i.

Thus, for new obligations on an upper-bound constraint $y_b^{\mathsf{a}} \prec u$, the clock $z_{\prec u}$ is not reset. Whenever a call i_c occurs, the updated set O of upper-bound and lower-bound obligations is pushed onto the stack to be recovered at the matching return i_r of i_c. Moreover, if i_c+1 is not a return (i.e., $i_r \neq i_c+1$), then $\mathcal{A}_{y_b^{\mathsf{a}}}$ moves to a control state where the set of lower-bound obligations is empty and the set of upper-bound obligations is obtained from O by replacing each upper-bound obligation (f,$\prec u$), for $f \in \{$*live*,*first*$\}$, with the live obligation (*live*,$\prec u$). The latter asserted at the initial position i_c+1 of the *MAP* ν visiting i_c+1 (note that ν ends at i_r-1) is used by $\mathcal{A}_{y_b^{\mathsf{a}}}$ to remember that the clock $z_{\prec u}$ cannot be reset along ν. Intuitively, live upper-bound obligations are propagated from the caller *MAP* to the called *MAP*. Note that fresh upper-bound obligations (*first*, $\prec u$) always refer to predictions done along the current *MAP* and, differently from the live upper-bound obligations, they can be removed when the next b occurs along the current *MAP*.

Extra technicalities are needed. At each position i, $\mathcal{A}_{y^{\mathsf{a}}_b}$ guesses whether i is the last position of the current *MAP* (i.e., the *MAP* visiting i). For this, it keeps track in its control state of the guessed type (call, return, or internal symbol) of the next input symbol. In particular, when i is a call, $\mathcal{A}_{y^{\mathsf{a}}_b}$ guesses whether it has a matching return. If not, $\mathcal{A}_{y^{\mathsf{a}}_b}$ pushes onto the stack a special symbol, say *bad*, and the guess is correct iff the symbol is never popped from the stack. Conversely, $\mathcal{A}_{y^{\mathsf{a}}_b}$ exploits a special proposition p_∞ whose Boolean value is carried in the control state: p_∞ *does not hold* at a position j of the input iff the *MAP* visiting j has a caller whose matching return exists. Note that p_∞ holds at infinitely many positions. The transition function of $\mathcal{A}_{y^{\mathsf{a}}_b}$ ensures that the Boolean value of p_∞ is propagated consistently with the guesses. Doing so, the guesses about the matched calls are correct iff p_∞ is asserted infinitely often along a run. A Büchi component of $\mathcal{A}_{y^{\mathsf{a}}_b}$ ensures this last requirement. Finally, we have to ensure that the lower-bound obligations and fresh upper-bound obligations at the current position are eventually checked, i.e., the current *MAP* eventually visits a b-position. For finite *MAP*, this can be ensured by the transition function of $\mathcal{A}_{y^{\mathsf{a}}_b}$. For infinite *MAP*, we note that at most one infinite *MAP* ν exists along a word, and ν visits only positions where p_∞ holds. Moreover, each position i greater than the initial position i_0 of ν is either a ν-position, or a position where p_∞ does not hold. Thus, a Büchi component of $\mathcal{A}_{y^{\mathsf{a}}_b}$ using proposition p_∞ ensures the b-liveness requirements along the unique infinite *MAP* (if any). Full details of the construction of $\mathcal{A}_{y^{\mathsf{a}}_b}$ are in Appendix C of [12].

By exploiting Theorems 1 and 3, we establish the main result of the paper.

Theorem 4. *Emptiness, universality, and language inclusion for Büchi* ECNA*, and visibly model-checking against Büchi* ECNA *are* Exptime*-complete.*

Proof. For the upper bounds, first observe that by [10, 1] the emptiness problem of generalized Büchi VPTA is Exptime-complete and solvable in time $O(n^4 \cdot 2^{O(m \cdot \log Km)})$, where n is the number of states, m is the number of clocks, and K is the largest constant used in the clock constraints of the automaton (hence, the time complexity is polynomial in the number of states). Now, given two Büchi ECNA $\mathcal{A}_1$ and $\mathcal{A}_2$ over Σ, checking whether $\mathcal{L}_T(\mathcal{A}_1) \subseteq \mathcal{L}_T(\mathcal{A}_2)$ reduces to check emptiness of the language $\mathcal{L}_T(\mathcal{A}_1) \cap \overline{\mathcal{L}_T(\mathcal{A}_2)}$. Similarly, given a Büchi VPTA $\mathcal{S}$ where all the states are accepting and a Büchi ECNA $\mathcal{A}$ over the same pushdown alphabet Σ, model-checking $\mathcal{S}$ against $\mathcal{A}$ reduces to check emptiness of the language $\mathcal{L}_T(\mathcal{S}) \cap \overline{\mathcal{L}_T(\mathcal{A})}$. Since Büchi VPTA are polynomial-time closed under intersection and universality can be reduced in linear-time to language inclusion, by the closure properties of Büchi ECNA (Theorem 1) and Theorem 3, membership in Exptime for the considered problems directly follow.

For the matching lower-bounds, the proof of Exptime-hardness for emptiness of Büchi VPTA can be easily adapted to the class of Büchi ECNA. For the other problems, the result directly follows from Exptime-hardness of the corresponding problems for Büchi VPA [5, 6] which are subsumed by Büchi ECNA. □

Conclusions. In this paper we have introduced and studied ECNA, a robust subclass of VPTA allowing to express meaningful non-regular timed properties of recursive systems. The closure under Boolean operations, and the decidability of languages inclusion and visibly model-checking makes ECNA amenable to specification and verification purposes. As future work, we plan to investigate suitable extensions of the Event Clock Temporal Logic introduced for ECA so that a logical counterpart for ECNA can be similarly recovered.

References

1. Abdulla, P.A., Atig, M.F., Stenman, J.: Dense-timed pushdown automata. In: Proc. 27th LICS. pp. 35–44. IEEE Computer Society (2012)
2. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science 126(2), 183–235 (1994)
3. Alur, R., Etessami, K., Madhusudan, P.: A temporal logic of nested calls and returns. In: Proc. 10th TACAS. LNCS, vol. 2988, pp. 467–481. Springer (2004)
4. Alur, R., Fix, L., Henzinger, T.A.: Event-clock automata: A determinizable class of timed automata. Theoretical Computer Science 211(1-2), 253–273 (1999)
5. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: Proc. 36th STOC. pp. 202–211. ACM (2004)
6. Alur, R., Madhusudan, P.: Adding nesting structure to words. Journal of ACM 56(3), 16:1–16:43 (2009)
7. Baier, C., Katoen, J.P.: Principles of Model Checking. The MIT Press (2008)
8. Benerecetti, M., Peron, A.: Timed recursive state machines: Expressiveness and complexity. Theoretical Computer Science 625, 85–124 (2016)
9. Bhave, D., Dave, V., Krishna, S.N., Phawade, R., Trivedi, A.: A logical characterization for dense-time visibly pushdown automata. In: Proc. 10th LATA. LNCS, vol. 9618, pp. 89–101. Springer (2016)
10. Bouajjani, A., Echahed, R., Robbana, R.: On the automatic verification of systems with continuous variables and unbounded discrete data structures. In: Hybrid Systems II. LNCS, vol. 999, pp. 64–85. Springer (1994)
11. Bozzelli, L., Murano, A., Peron, A.: Pushdown Module Checking. Formal Methods in System Design 36(1), 65–95 (2010)
12. Bozzelli, L., Murano, A., Peron, A.: Event-clock nested automata. http://arxiv.org/abs/1711.08314 (2017)
13. Chatterjee, K., Ma, D., Majumdar, R., Zhao, T., Henzinger, T., Palsberg, J.: Stack size analysis for interrupt-driven programs. In: Proc. 10th SAS. LNCS, vol. 2694, pp. 109–126. Springer (2003)
14. Clemente, L., Lasota, S.: Timed pushdown automata revisited. In: Proc. 30th LICS. pp. 738–749. IEEE Computer Society (2015)
15. Emmi, M., Majumdar, R.: Decision problems for the verification of real-time software. In: Proc. 9th HSCC. LNCS, vol. 3927, pp. 200–211 (2006)
16. Kupferman, O., Piterman, N., Vardi, M.Y.: Pushdown specifications. In: Proc. 9th LPAR. LNCS, vol. 2514, pp. 262–277. Springer (2002)
17. Murano, A., Perelli, G.: Pushdown multi-agent system verification. In: Proc. IJCAI. pp. 1090–1097 (2015)
18. Tang, N.V., Ogawa, M.: Event-clock visibly pushdown automata. In: Proc. 35th SOFSEM. LNCS, vol. 5404, pp. 558–569. Springer (2009)
19. Trivedi, A., Wojtczak, D.: Recursive timed automata. In: Proc. 8th ATVA. LNCS, vol. 6252, pp. 306–324. Springer (2010)
20. Walukiewicz, I.: Pushdown Processes: Games and Model Checking. In: CAV'96. pp. 62–74 (1996)