A Perfect Class of Context-Sensitive Timed Languages

D. Bhave[1], V. Dave[1], S. N. Krishna[1], R. Phawade[1], and A. Trivedi[1,2]

[1] IIT Bombay and [2] CU Boulder

Abstract. Perfect languages—a term coined by Esparza, Ganty, and Majumdar—are the classes of languages that are closed under Boolean operations and enjoy decidable emptiness problem. Perfect languages form the basis for decidable automata-theoretic model-checking for the respective class of models. Regular languages and visibly pushdown languages are paradigmatic examples of perfect languages. Alur and Dill initiated the language-theoretic study of timed languages and introduced timed automata capturing a timed analog of regular languages. However, unlike their untimed counterparts, timed regular languages are not perfect. Alur, Fix, and Henzinger later discovered a perfect subclass of timed languages recognized by event-clock automata. Since then, a number of perfect subclasses of timed context-free languages, such as event-clock visibly pushdown languages, have been proposed. There exist examples of perfect languages even beyond context-free languages:—La Torre, Madhusudan, and Parlato characterized first perfect class of context-sensitive languages via multistack visibly pushdown automata with an explicit bound on number of stages where in each stage at most one stack is used. In this paper we extend their work for timed languages by characterizing a perfect subclass of timed context-sensitive languages and provide a logical characterization for this class of timed languages.

1 Introduction

A class $\mathcal{C}$ of languages is called *perfect* [9] if it is closed under Boolean operations and permits algorithmic emptiness-checking. Perfect languages are the key ingredient for the Vardi-Wolper recipe for automata-theoretic model-checking:—given a system specification $\mathcal{S}$ and a system implementation $\mathcal{M}$ as languages in $\mathcal{C}$, the model-checking involves deciding the emptiness of the language $\mathcal{M} \cap \neg\mathcal{S} \in \mathcal{C}$. The class of ($\omega$-)regular languages is a well-known class of perfect languages, while other classes of languages such as context-free languages (CFLs) or context-sensitive languages (CSLs) are, in general, not perfect. CFLs are not perfect since they are not closed under intersection and complementation, although emptiness is decidable.On the other hand, CSLs are closed under Boolean operations but emptiness, in general, is undecidable for CSLs [6].

Alur and Madhusudan [4] discovered a perfect subclass of CFLs, called visibly pushdown languages (VPLs), characterized by *visibly pushdown automata* that operate over words that dictate the stack operations. This notion is formalized by giving an explicit partition of the alphabet into three disjoint sets of *call*, *return*, and *internal* symbols and the visibly pushdown automata must push one symbol to stack while reading a call symbol, and must pop one symbol (given stack is non-empty) while reading a return symbol, and must not touch the stack while reading an internal symbol. This visibility enables closure of these automata under all of the Boolean operations, while retaining the decidable emptiness property. Building upon this work, La Torre, Madhusudan, and Parlato [10] introduced a perfect class of CSLs, called multistack visibly pushdown languages (MVPLs), recognized by visibly pushdown automata with multiple stacks (and call-return symbols for each stack) where the number of switches between various stacks for popping-purposes is bounded.

Example 1. $L = \{a^n b^n \,:\, n \geq 0\}$ is a VPL with a as call and b as return symbol for the unique stack, whereas $L' = \{a_1^n a_2^m b_1^n b_2^m \,:\, n, m \geq 0\}$ is a MVPL considering a_i and b_i as call and return symbols, respectively, for stack-i where $i \in \{1, 2\}$. Finally, $L'' = \{a^n b^n c^n \,:\, n \geq 0\}$ is neither VPL nor MVPL for any partition of alphabets as call and respectively alphabets of various stacks.

In this paper we introduce a timed extension of this context-sensitive language and study language-theoretic properties of the class in [13]. We characterize a perfect subclass of timed context-sensitive languages and provide a logical characterization for this class of timed languages.

Quest for Perfect Timed Languages. Alur and Dill [2] initiated automata-theoretic study of timed languages and characterized the class of timed-regular languages as the languages defined by timed automata. Unlike untimed regular languages, Alur and Dill showed that timed regular languages are not perfect as they are not closed under complementation. However, the emptiness of timed automata is a decidable using a technique known as region-construction. To overcome the limitation of timed automata for model-checking, Alur, Fix, and Henzinger introduced a perfect class of timed languages called the event-clock automata [3] (ECA)that achieves the closure under Boolean operations by making clock resets visible—the reset of each clock variable is determined by a fixed class of events and hence *visible* just by looking at the input word. The decidability of the emptiness for ECA follows from the decidability of regular timed languages.

Two of the well-known models for context-free timed languages include recursive timed automata (RTAs) [14] and dense-time pushdown automata (dtPDAs) [1]. RTAs generalize recursive state machines with clock variables, while dtPDAs generalize pushdown automata with clocks and stack with variable ages. In general, the emptiness problem for the RTA in undecidable, however [14] characterizes classes of RTA with decidable emptiness problem. However, without any further restrictions, such as event-clock or visible stack, the languages captured by these classes are not perfect, since they strictly generalize both timed regular languages and CFLs. Tang and Ogawa in [15] proposed a first perfect timed context-free language class characterized by *event-clock visibly pushdown automata* (ECVPA) that generalized both ECA and VPA. For the proposed model they showed determinizability as well as closure under Boolean operations, and proved the decidability of the emptiness problem. However, ECVPAs, unlike dtPDAs, do not support

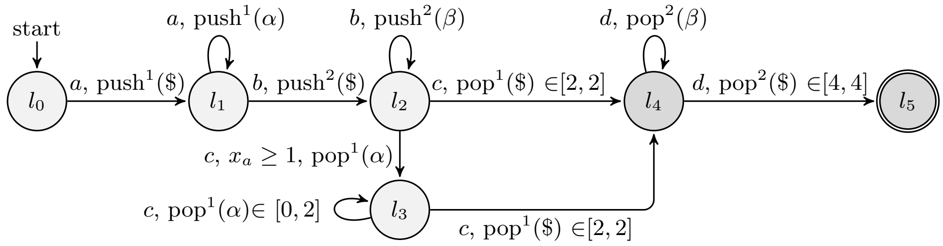

Fig. 1. Dense-time Multistack Visibly Pushdown Automata used in Example 2

pushing the clocks on the stack. We proposed [7] a generalization of ECVPA called dense-time visibly pushdown automata (dtVPA), that are strictly more expressive than ECVPA as they support stack with variable ages (like dtPDA) and showed that dtVPA characterize a perfect timed context-free language.

Contributions. We study a class of timed context-sensitive languages called dense-time multistack visibly pushdown languages (dtMVPLs), characterized by dense-time visibly pushdown multistack automata (dtMVPA), that generalize MVPLs with multiple stacks with ages as shown in the following example.

Example 2. Consider the timed language whose untimed component is of the form $\{a^y b^z c^y d^z \mid y, z \geq 1\}$ with the critical timing restrictions among various symbols in the following manner. The first c must appear after 1 time-unit of last a, the first d must appear within 3 time-unit after last b, and finally the last b must appear within 2 time units of the beginning and last d must appear precisely at 4 time unit. This language is accepted by a dtMVPA with two stacks shown in Figure 1. Let a and c (b and d, resp.) be call and return symbols for the first (second, resp.) stack. Stack alphabets for first stack is $\Gamma^1 = \{\alpha, \$\}$ and for second stack is $\Gamma^2 = \{\beta, \$\}$. In the figure a clock x_a measures the time since the occurrence of last a, while constraints $pop(\gamma) \in I$ checks if the age of the popped symbol is in a given interval I. The correctness of the model is easy to verify.

In this paper we show dtMVPLs are closed under Boolean operations and enjoy decidable emptiness problem. Although, the emptiness problem for restrictions of context sensitive languages has been studied extensively [5,8,13,12,11], ours is the first attempt to formalize perfect dense-time context-sensitive languages. We will also give a logical characterization of this class of languages. We believe that dtMVPLs provide an expressive yet decidable model-checking framework for concurrent time-critical software systems (See Appendix A for an example).

The paper is organized as follows. We begin by introducing dense-time visibly pushdown multistack automata in the next section. In Section 3 we show closure under Boolean operations for this model, followed by logical characterization in Section 4. Due to lack of space, the proof for the decidability of emptiness is deferred to the Appendix.

2 Dense-Time Visibly Pushdown Multistack Automata

We assume that the reader is comfortable with standard concepts from automata theory (such as context-free languages, pushdown automata, MSO logic), concepts from timed automata (such as clocks, event clocks, clock constraints, and valuations), and visibly pushdown automata. Due to space limitation, we only give a very brief introduction of required concepts in this section, and for a detailed background on these concepts we refer the reader to [2,3,4].

A finite timed word over Σ is a sequence $(a_1, t_1), (a_2, t_2), \ldots, (a_n, t_n) \in (\Sigma \times \mathbb{R}_{\geq 0})^*$ such that $t_i \leq t_{i+1}$ for all $1 \leq i \leq n-1$. Alternatively, we can represent timed words as tuple $(\langle a_1, \ldots, a_n \rangle, \langle t_1, \ldots, t_n \rangle)$. We use both of these formats depending on technical convenience. We represent the set of finite timed words over Σ by $T\Sigma^*$. Before we introduce our model, we recall the definitions of event-clock automata and visibly pushdown automata.

2.1 Preliminaries

Event-clock automata (ECA) [3] are a determinizable subclass of timed automata [2] that for every action $a \in \Sigma$ implicitly associate two clocks x_a and y_a, where the "recorder" clock x_a records the time of the last occurrence of action a, and the "predictor" clock y_a predicts the time of the next occurrence of action a. Hence, event-clock automata do not permit explicit reset of clocks and it is implicitly governed by the input timed word. This property makes ECA determinizable and closed under all Boolean operations.

Notice that since clock resets are "visible" in input timed word, the clock valuations after reading a prefix of the word is also determined by the timed word. For example, for a timed word $w = (a_1, t_1), (a_2, t_2), \ldots, (a_n, t_n)$, the value of the event clock x_a at position j is $t_j - t_i$ where i is the largest position preceding j where an action a occurred. If no a has occurred before the jth position, then the value of x_a is undefined denoted by a special symbol $\vdash$. Similarly, the value of y_a at position j of w is undefined if symbol a does not occur in w after the jth position. Otherwise, it is $t_k - t_j$ where k is the first occurrence of a after j.

We write C for the set of all event clocks and we use $\mathbb{R}^{\vdash}_{>0}$ for the set $\mathbb{R}_{>0} \cup \{\vdash\}$. Formally, the clock valuation after reading j-th prefix of the input timed word w, $\nu^w_j : C \mapsto \mathbb{R}^{\vdash}_{>0}$, is defined in the following way: $\nu^w_j(x_q) = t_j - t_i$ if there exists an $0 \leq i < j$ such that $a_i = q$ and $a_k \neq q$ for all $i < k < j$, otherwise $\nu^w_j(x_q) = \vdash$ (undefined). Similarly, $\nu^w_j(y_q) = t_m - t_j$ if there is $j < m$ such that $a_m = q$ and $a_l \neq q$ for all $j < l < m$, otherwise $\nu^w_j(y_q) = \vdash$. A clock constraint over C is a boolean combination of constraints of the form $z \sim c$ where $z \in C$, $c \in \mathbb{N}$ and $\sim \in \{\leq, \geq\}$. Given a clock constraint $z \sim c$ over C, we write $\nu^w_i \models (z \sim c)$ to denote if $\nu^w_j(z) \sim c$. For any boolean combination φ, $\nu^w_i \models \varphi$ is defined in an obvious way: if $\varphi = \varphi_1 \wedge \varphi_2$, then $\nu^w_i \models \varphi$ iff $\nu^w_i \models \varphi_1$ and $\nu^w_i \models \varphi_2$. Likewise, the other boolean combinations are defined.

Definition 3. *An event clock automaton is a tuple $A = (L, \Sigma, L^0, F, E)$ where L is a set of finite locations, Σ is a finite alphabet, $L^0 \in L$ is the set of initial* *locations, $F \in L$ is the set of final locations, and E is a finite set of edges of the form $(\ell, \ell', a, \varphi)$ where ℓ, ℓ' are locations, $a \in \Sigma$, and φ is a clock constraint.*

The class of languages accepted by event-clock automata are closed under boolean operations with decidable emptiness property [3].

Visibly pushdown automata [4] are a determinizable subclass of pushdown automata that operate over words that dictate the stack operations. This notion is formalized by giving an explicit partition of the alphabet into three disjoint sets of *call*, *return*, and *internal* symbols and the visibly pushdown automata must push one symbol to stack while reading a call symbol, and must pop one symbol (given stack is non-empty) while reading a return symbol, and must not touch the stack while reading the internal symbol.

Definition 4. *A visibly pushdown alphabet is a tuple $\Sigma = \langle \Sigma_c, \Sigma_r, \Sigma_{int} \rangle$ where Σ is partitioned into a* call *alphabet Σ_c, a* return *alphabet Σ_r, and an* internal *alphabet Σ_{int}. A visibly pushdown automata(VPA) over $\Sigma = \langle \Sigma_c, \Sigma_r, \Sigma_{int} \rangle$ is a tuple $(L, \Sigma, \Gamma, L^0, \delta, F)$ where L is a finite set of locations including a set $L^0 \subseteq L$ of initial locations, Γ is a finite stack alphabet with special end-of-stack symbol $\bot$, $\Delta \subseteq (L \times \Sigma_c \times L \times (\Gamma \setminus \bot)) \cup (L \times \Sigma_r \times \Gamma \times L) \cup (L \times \Sigma_{int} \times L)$ is the transition relation, and $F \subseteq L$ is final locations.*

The class of languages accepted by visibly pushdown automata are closed under boolean operations with decidable emptiness property [4].

2.2 Dense-Time Visibly Pushdown Multistack Automata(dtMVPA)

We introduce the dense-time visibly pushdown automata as an event-clock automaton equipped with multiple (say $n \geq 1$) timed stacks along with a visibly pushdown alphabet $\Sigma = \langle \Sigma_c^h, \Sigma_r^h, \Sigma_{int}^h \rangle_{h=1}^n$ where $\Sigma_x^i \cap \Sigma_x^j = \emptyset$ for $i \neq j$, and $x \in \{c, r, int\}$. Due to space limitation and notational convenience, we assume that the partitioning function is one-to-one, i.e. each symbol $a \in \Sigma^h$ has unique recorder x_a and predictor y_a clocks assigned to it. Let Γ^h be the stack alphabet of the h-th stack. Let $\Gamma = \bigcup_{h=1}^n \Gamma^h$ and let $\Sigma^h = \langle \Sigma_c^h, \Sigma_r^h, \Sigma_{int}^h \rangle$. Let C_{Σ^h} (or C_h when Σ^h is clear) be a finite set of event clocks. Let $\Phi(C_h)$ be the set of *clock constraints* over C_h and $\mathcal{I}$ be the set of intervals.

Definition 5. *A dense-time visibly pushdown multistack automata (dtMVPAs) over $\langle \Sigma_c^h, \Sigma_r^h, \Sigma_{int}^h \rangle_{h=1}^n$ is a tuple $(L, \Sigma, \Gamma, L^0, F, \Delta{=}(\Delta_c^h \cup \Delta_r^h \cup \Delta_{int}^h)_{h=1}^n)$ where*

- *L is a finite set of locations including a set $L^0 \subseteq L$ of initial locations,*
- *Γ^h is the finite alphabet of the hth stack with special end-of-stack symbol $\bot_h$,*
- *$\Delta_c^h \subseteq (L \times \Sigma_c^h \times \Phi(C_h) \times L \times (\Gamma^h \setminus \bot_h))$ is the set of call transitions,*
- *$\Delta_r^h \subseteq (L \times \Sigma_r^h \times \mathcal{I} \times \Gamma^h \times \Phi(C_h) \times L)$ is set of return transitions,*
- *$\Delta_{int}^h \subseteq (L \times \Sigma_{int}^h \times \Phi(C_h) \times L)$ is set of internal transitions, and*
- *$F \subseteq L$ is the set of final locations.*

Let $w = (a_0, t_0), \ldots, (a_e, t_e)$ be a timed word. A configuration of the dtMVPA is a tuple $(\ell, \nu_i^w, (((\gamma^1\sigma^1, age(\gamma^1\sigma^1)), \ldots, (\gamma^n\sigma^n, age(\gamma^n\sigma^n)))$ where ℓ is the current location of the dtMVPA, ν_i^w gives the valuation of all the event clocks at position $i \leq |w|$, $\gamma^h\sigma^h \in \Gamma^h(\Gamma^h)^*$ is the content of stack h with γ^h being the topmost symbol and σ^h is the string representing the stack content below γ^h, while $age(\gamma^h\sigma^h)$ is a sequence of real numbers encoding the ages of all the stack symbols (the time elapsed since each of them was pushed on to the stack). We follow the assumption that $age(\bot^h) = \langle \vdash \rangle$ (undefined). If for some string $\sigma^h \in (\Gamma^h)^*$ we have that $age(\sigma^h) = \langle t_1, t_2, \ldots, t_g \rangle$ and for $\tau \in \mathbb{R}_{\geq 0}$ we write $age(\sigma^h) + \tau$ for the sequence $\langle t_1 + \tau, t_2 + \tau, \ldots, t_g + \tau \rangle$. For a sequence $\sigma^h = \langle \gamma_1^h, \ldots, \gamma_g^h \rangle$ and a member γ^h we write $\gamma^h :: \sigma^h$ for $\langle \gamma^h, \gamma_1^h, \ldots, \gamma_g^h \rangle$.

A run of a dtMVPA on $w = (a_0, t_0), \ldots, (a_e, t_e)$ is a sequence of configurations $(\ell_0, \nu_0^w, (\langle \bot^1 \rangle, \langle \vdash \rangle), \ldots, (\langle \bot^n \rangle, \langle \vdash \rangle)), (\ell_1, \nu_1^w, ((\sigma_1^1, age(\sigma_1^1)), \ldots, (\sigma_1^n, age(\sigma_1^n))),$ $\ldots, (\ell_{e+1}, \nu_{e+1}^w, (\sigma_{e+1}^1, age(\sigma_{e+1}^1)), \ldots, (\sigma_{e+1}^n, age(\sigma_{e+1}^n))))$ where $\ell_i \in L$, $\ell_0 \in L^0$, $\sigma_i^h \in (\Gamma^h \cup \{\bot^h\})^+$, and for each $i,\ 0 \leq i \leq e$, we have:

- If $a_i \in \Sigma_c^h$, then there is $(\ell_i, a_i, \varphi, \ell_{i+1}, \gamma^h) \in \Delta_c^h$ such that $\nu_i^w \models \varphi$. The symbol $\gamma^h \in \Gamma^h \backslash \{\bot^h\}$ is then pushed onto the stack h, and its age is initialized to zero, i.e. $(\sigma_{i+1}^h, age(\sigma_{i+1}^h)) = (\gamma^h :: \sigma_i^h, 0 :: (age(\sigma_i^h) + (t_i - t_{i-1})))$. All symbols in all other stacks are unchanged, and age by $t_i - t_{i-1}$.
- If $a_i \in \Sigma_r^h$, then there is $(\ell_i, a_i, I, \gamma^h, \varphi, \ell_{i+1}) \in \Delta_r^h$ such that $\nu_i^w \models \varphi$. Also, $\sigma_i^h = \gamma^h :: \kappa \in \Gamma^h(\Gamma^h)^*$ and $age(\gamma^h) + (t_i - t_{i-1}) \in I$. The symbol γ^h is popped from stack h obtaining $\sigma_{i+1}^h = \kappa$ and $age(\sigma_{i+1}^h) = age(\sigma_i^h) + (t_i - t_{i-1})$. However, if $\gamma^h = \langle \bot^h \rangle$, then γ^h is not popped. The contents of all other stacks remains unchanged, and simply age by $(t_i - t_{i-1})$.
- If $a_i \in \Sigma_{int}^h$, then there is $(\ell_i, a_i, \varphi, \ell_{i+1}) \in \Delta_{int}^h$ such that $\nu_i^w \models \varphi$. In this case all stacks remain unchanged i.e. $\sigma_i^h = \sigma_{i+1}^h$, and $age(\sigma_{i+1}^h) = age(\sigma_i^h) + (t_i - t_{i-1})$ for all $1 \leq h \leq n$. All symbols in all stacks age by $t_i - t_{i-1}$.

A run ρ of a dtMVPA M is accepting if it terminates in a final location. A timed word w is an accepting word if there is an accepting run of M on w. The language $L(M)$ of a dtMVPA M, is the set of all timed words w accepted by M.

A dtMVPA $M = (L, \Sigma, \Gamma, L^0, F, \Delta)$ is said to be *deterministic* if it has exactly one start location, and for every configuration and input action exactly one transition is enabled. Formally, we have the following conditions: for every $(\ell, a, \phi_1, \ell', \gamma_1), (\ell, a, \phi_2, \ell'', \gamma_2) \in \Delta_c^h$, $\phi_1 \wedge \phi_2$ is unsatisfiable; for every $(\ell, a, I_1, \gamma, \phi_1, \ell'), (\ell, a, I_2, \gamma, \phi_2, \ell'') \in \Delta_r^h$, either $\phi_1 \wedge \phi_2$ is unsatisfiable or $I_1 \cap I_2 = \emptyset$; and for every $(\ell, a, \phi_1, \ell'), (\ell, a, \phi_2, \ell') \in \Delta_{int}^h$, $\phi_1 \wedge \phi_2$ is unsatisfiable. An ECMVPA is a dtMVPA where the stacks are untimed. A ECMVPA $(L, \Sigma, \Gamma, L^0, F, \Delta)$ is an dtMVPA if $I = [0, +\infty]$ for every $(\ell, a, I, \gamma, \phi, \ell') \in \Delta_r^h$.

Let $\Sigma = \langle \Sigma_c^h, \Sigma_r^h, \Sigma_{int}^h \rangle_{h=1}^n$be a visibly pushdown alphabet. A *context* over $\Sigma^h = \langle \Sigma_c^h, \Sigma_r^h, \Sigma_{int}^h \rangle$ is a timed word in $(\Sigma^h)^*$. The empty word ε is a context. For ease, we assume in this paper that any context *has at least one symbol from* Σ. A *round* over Σ is a timed word w over Σ of the form $w_1 w_2 \ldots w_n$ such that each w_h is a context over Σ^h. A k-round over Σ is a timed word w that can be obtained as a concatenation of k rounds over Σ. That is, $w = u_1 u_2 \ldots u_k$, where each u_i

is a round. Let $Round(\Sigma, k)$ denote the set of all k-round timed words over Σ. For any fixed k, a k-round dtMVPA over Σ is a tuple $A = (k, L, \Sigma, \Gamma, L^0, F, \Delta)$ where $M = (L, \Sigma, \Gamma, L^0, F, \Delta)$ is a dtMVPA over Σ. The language accepted by A is $L(A) = L(M) \cap Round(\Sigma, k)$ and is called k-round dense time multistack visibly push down language. The class of k-round dense time multistack visibly push down languages is denoted k-dtMVPL. The set $\bigcup_{k \geq 1} k$-dtMVPL is denoted bd-dtMVPL, and is the class of dense time multistack visibly push down languages with a bounded number of rounds. We define k-ECMVPL and bd-ECMVPL in a similar fashion. Also, we write k-dtMVPA and k-ECMVPA to denote k-round dtMVPA and k-round ECMVPA. The key result of the paper is the following.

Theorem 6 (A Perfect Timed Context-Sensitive Language). *The classes of languages accepted by k-dtMVPA and k-ECMVPA are perfect:— they are closed under Boolean operations with decidable emptiness problem.*

We sketch key lemmas towards this proof in the following section. As an application of this theorem we show Monadic second-order logic characterization of the languages accepted by k-dtMVPA in Section 4.

3 Proof of Theorem 6

The closure under union and intersection for both k-dtMVPA and k-ECMVPA is straightforward and is sketched in Appendix B. In order to show closure under complementation, the main hurdle is to show determinizability of these automata. We sketch the key ideas required to get determinizability for k-ECMVPA in Section 3.1 and for k-dtMVPA in Section 3.2. The decidability of the emptiness problem for k-ECMVPA follows as for every k-ECMVPA, via region construction [3], one can get an untimed-bisimilar k-MVPA, which has a decidable emptiness [13]. In Section 3.2 we show that for every k-dtMVPA we get an emptiness-preserving k-ECMVPA and hence this result in combination with previous remark yield decidability of emptiness for k-dtMVPA.

3.1 Determinizability of k-ECMVPA

For the determinizability proof the key observation is the since the words accepted by A is a catenation of k rounds, and the stacks (or contexts) do not interfere with each other, the k-ECMVPA A can be considered as a "composition" of n ECVPA $A_1, \ldots, A_n$, with stack of each A_i corresponds to i-th stack of the k-ECMVPA. A has to simulate the n ECVPAs in a round robin fashion for k rounds.

If $w \in L(A)$, then $w = u_1 u_2 \ldots u_k$, and $u_i = u_{i1} u_{i2} \ldots u_{in}$, where u_{ij} is the jth context in the ith round. Starting in an initial location ℓ_{11}, control is passed to A_1, which runs on u_{11} and enters location $\ell'_{11} = \ell_{12}$. Let $\nu'_{11} = \nu_{12}$ be the values of all clocks after processing u_{11}. At this point of time, A_2 runs on u_{12} starting in location ℓ_{12}, and so on, until A_n runs on u_{1n} starting in location ℓ_{1n}. Now first round is over, and u_1 is processed. A_n ends in some location $\ell'_{1n} = \ell_{21}$. Now A_1 starts again in ℓ_{21} and processes u_{21}. The values of all recorders and predictors change according to the time that elapsed during the simulation of $A_2, \ldots, A_n$. It must be noted that between two consecutive rounds i and $i+1$ of any A_j, none of the clocks pertaining to A_j get reset; they only reflect the time that has elapsed since the last round of A_j. This continues for k rounds, until u_{kn} is processed. A_j processes in order, $u_{1j}, u_{2j}, \ldots, u_{kj}$ over $(\Sigma^j)^*$ for $1 \leq j \leq n$. In round i, $1 \leq i \leq k$, each A_j, $1 \leq j \leq n-1$, starts in location ℓ_{ij}, runs on u_{ij} and "computes" a location ℓ_{ij+1}. Similarly, A_n moves from round i to round $i+1$, by starting in ℓ_{in}, runs on u_{in} and computes a location ℓ_{i+11}. The $(i+1)$th round begins in this location with A_1 running on u_{i+11}. Thus, by stitching together the locations needed to switch from A_j to A_{j+1}, we can obtain a simulation of A.

Let $u_{ij} = (a_j^1, t_{ij}^1) \ldots (a_j^{last}, t_{ij}^{last})$, where $t_{ij}^1, \ldots, t_{ij}^{last}$ are the time stamps on reading u_{ij}. Let $\kappa_j = u_{1j}(\#_1, t_{1j}^{last})u_{2j}(\#_2, t_{2j}^{last}) \ldots u_{kj}(\#_k, t_{kj}^{last})$. The new symbols $\#_i$ help disambiguate A_j processing $u_{1j}, \ldots, u_{kj}$ in k rounds. We first focus on each ECVPA A_j which processes $u_{1j}, u_{2j}, \ldots, u_{kj}$. Let $cmax$ be the maximum constant used in clock constraints of Σ^j in the ECMVPA A. Let $\mathcal{I} = \{[0,0], [0,1], \ldots, [cmax, cmax], [cmax, \infty)\}$ be a set of intervals. A *correct sequence of round switches* for A_j with respect to κ_j is a sequence of pairs $V_j = P_{1j}P_{2j} \ldots P_{kj}$, where $P_{hj} = ((\ell_{hj}, I_{hj}), \ell'_{hj})$, $2 \leq h \leq k$, $P_{1j} = ((\ell_{1j}, \nu_{1j}), \ell'_{1j})$ and $I_{hj} \in \mathcal{I}$ such that

1. Starting in ℓ_{1j}, with the jth stack containing $\perp_j$, and an initial valuation ν_{1j} of all recorders and predictors of Σ^j, the ECMVPA A processes u_{1j} and reaches some ℓ'_{1j} with stack content σ_{2j} and clock valuation ν'_{1j}. The processing of u_{2j} by A then starts at location ℓ_{2j}, and a time $t \in I_{2j}$ has elapsed between the processing of u_{1j} and u_{2j}. Thus, A starts processing u_{2j} in (ℓ_{2j}, ν_{2j}) where ν_{2j} is the valuation of all recorders and predictors updated from ν'_{1j} with respect to t. The stack content remains same as σ_{2j} when the processing of u_{2j} begins.
2. In general, starting in (ℓ_{hj}, ν_{hj}), $h > 1$ with the jth stack containing σ_{hj}, and ν_{hj} obtained from ν_{h-1j} by updating all recorders and predictors based on the time interval I_{hj} that records the time elapse between processing u_{hj-1} and u_{hj}, A processes u_{hj} and reaches (ℓ'_{hj}, ν'_{hj}) with stack content σ_{h+1j}. The processing of u_{h+1j} starts after time $t \in I_{h+1}$ has elapsed since processing u_{hj} in a location ℓ_{h+1j}, and stack content being σ_{h+1j}.

Lemma 7. *(Round Switching Lemma for A_j) Let $A = (k, L, \Sigma, \Gamma, L^0, F, \Delta)$ be a k-ECMVPA. Let $w = u_1u_2 \ldots u_k$ with $u_i = u_{i1}u_{i2} \ldots u_{in}$, $1 \leq i \leq k$. Then we can construct a ECVPA A_j over $\Sigma^j \cup \{\#_1, \ldots, \#_k\}$ which reaches a location V_j on reading κ_j iff V_j is a correct sequence of round switches for A_j.*

Proof. Recall that κ_j is defined by annotating $u_{1j}u_{2j} \ldots u_{kj}$ with new symbols $\{\#_1, \ldots, \#_k\}$ and appropriate time stamps. Let $V_j = P_{1j} \ldots P_{kj}$ be a correct sequence of round switches for A_j. Given the k-ECMVPA $A = (k, L, \Sigma, \Gamma, L^0, F, \Delta)$ with w, the ECVPA A_j is constructed by simulating the transitions of A on Σ^j by guessing V_j in its initial location. The alphabet of A_j is $\Sigma^j \cup \{\#_1, \ldots, \#_n\}$, and hence has event clocks $x_a, x_{\#_i}$, $a \in \Sigma^j$. Whenever A_j reads the $\#_i$, the control location as well as the valuation of all recorders and predictors are changed according to P_{i+1j}, $1 \le i \le k-1$. On reading $\#_k$, A_j enters the location V_j from its current location ℓ'_{kj}. The locations of A_j are $V_j \cup \{(i, \ell_{ij}, V_j), (i, \ell_{ij}, V_j, \#), (i, \ell_{ij}, V_j, a) \mid 1 \le i \le k, \ell \in L, a \in \Sigma^j, V_j \in ((L \times I) \times L)^k\}, \cup((L \times I) \times L)^k, I \in \mathcal{I}$. The set of initial locations are $\{(1, \ell_{1j}, V_j) \mid V_j \in ((L \times I) \times L)^k, I \in \mathcal{I}\}$. Starting in $(1, \ell_{1j}, V_j)$, A_j processes u_{1j}. When the last symbol a of u_{1j} is read, it enters a location $(1, \ell'_{1j}, V_j, a)$. From this location, only $\#_1$ transitions are enabled. On reading $\#_1$, we move from $(1, \ell'_{1j}, V_j, a)$ to a location $(2, \ell_{2j}, V_j, \#)$, where $P_2 = ((\ell_{2j}, I_{2j}), \ell'_{2j})$ and $P_1 = ((\ell_{1j}, \nu_{1j}), \ell'_{1j})$, after checking no time elapse since a (check x_a=0). This ensures that no time is spent in processing $\#_1$ after u_{1j}. Now A_j starts processing u_{2j} starting in location $(2, \ell_{2j}, V_j, \#)$. From $(2, \ell_{2j}, V_j, \#)$, on reading a symbol $a \in \Sigma^j$, we check that the time elapse since $\#_1$ lies in the interval I_{2j} (check $x_{\#_1} \in I_{2j}$) as given by P_2 and so on. When round k is reached, A_j starts processing in some location $(k, \ell_{kj}, V_j, \#)$, and reaches (k, ℓ'_{kj}, V_j, a). When $\#_k$ is read, A_j enters location V_j. The transitions δ^j of A_j are given in Appendix C. It is easy to see that V_j is reached by A_j only when the guessed V_j in the initial location is a correct sequence of round switches for A_j. □

While each V_j talks about the correct sequence of round switches, $1 \le j \le n$, the sequence $V_1 V_2 \dots V_n$ is called a *globally correct sequence* iff we can stitch together the individual V_i's to obtain a complete simulation of A on w by moving across contexts and rounds. For instance, consider $V_j = P_{1j} P_{2j} \dots P_{kj}$ and $V_{j+1} = P_{1j+1} P_{2j+1} \dots P_{kj+1}$ for $1 \le j \le n-1$. Recall that $P_{ij} = ((\ell_{ij}, I_{ij}), \ell'_{ij})$ and $P_{ij+1} = ((\ell_{ij+1}, I_{ij+1}), \ell'_{ij+1})$ for $1 \le i \le k$. The sequence $V_1 V_2 \dots V_n$ is globally correct iff $\ell'_{ij} = \ell_{ij+1}$, $j \le n-1$ and $\ell'_{in} = \ell_{i+11}$ for $1 \le i \le k$.

Lemma 8. *Let $w = u_1 u_2 \dots u_k$ be a timed word in $Round(\Sigma, k)$, with $A = (k, L, \Sigma, \Gamma, L^0, F, \Delta)$ being a k-ECMVPA over Σ, and let $u_i = u_{i1} u_{i2} \dots u_{in}$ and κ_j be as defined above. Then $w \in L(A)$ iff for $1 \le j \le n$, there exists a correct switching sequence V_j of the ECVPA A_j for κ_j such that $V_1 V_2 \dots V_n$ is a globally correct sequence for A with $\ell_{11} \in L^0$ and $\ell'_{kn} \in F$.*

Proof. The proof essentially shows how one can simulate A by composing the A_j's using a globally correct sequence $V_1 V_2 \dots V_n$. The idea is to simulate each A_j one after the other, allowing A_{j+1} to begin on u_{ij+1} iff the location reached ℓ'_{ij} at the end of u_{ij} by A_j matches with ℓ_{ij+1}, the proposed starting location of A_{j+1} on u_{ij+1}. Lets construct a composition of $A_1, \dots, A_n$ which runs on w, and accepts w iff there exists a globally correct sequence $V_1 V_2 \dots V_n$. The initial locations are of the form $(p_1, p_2, \dots, p_n, 1, 1)$, where the last two entries denote the current round number and context number and p_j is an initial location of A_j. The transitions Δ of the composition are defined using the transitions δ^j of A_j.

In some chosen initial location, we first run A_1 updating only the first entry p_1 of the tuple until u_{11} is completely read. The first entry of the tuple then has the form $p'_1 = (1, \ell'_{11}, V_1, a)$ where a is the last symbol of u_{11}. When A_1 reads $\#_1$, the current location in the composition is $(p'_1, p_2, \dots, p_n, 1, 1)$. In the composition of $A_1, \dots, A_n$, since there are no $\#$'s to be read, we start simulation of A_2 on u_{12} from $(p'_1, p_2, \dots, p_n, 1, 1)$ iff p_2 is $(2, \ell_{12}, V_2)$ such that the ℓ'_{11} in p_1

is same as the ℓ_{12} in p_2. We then add the transition from $(p'_1, p_2, \ldots, p_n, 1, 1)$ to $(p''_1 = (2, \ell_{21}, V_1, a), q, \ldots, p_n, 1, 2)$ where q is obtained from p_2 by a transition in A_2 on the first symbol of u_{12}. The a in p''_1 is the last symbol of u_{11} taken from $p'_1 = (1, \ell'_{11}, V_1, a)$, and the ℓ_{21} in p''_1 is obtained from $P_{21} = ((\ell_{21}, I_{21}), \ell'_{21})$ of V_1. We continue like this till we reach u_{1n}, the last context in round 1, and reach some location $(s_1, s_2, \ldots, s_{n-1}, p'_n, 1, 1)$ with $s_1 = (2, \ell_{21}, V_1, a_1), s_2 = (2, \ell_{22}, V_2, a_2), \ldots, s_{n-1} = (2, \ell_{2\ n-1}, V_{n-1}, a_{n-1})$ and $p'_n = (1, \ell'_{1n}, V_n, a_n)$.

Now, to start the second round, that is on u_{21}, we allow the transition from the above location iff $\ell'_{1n} = \ell_{21}$ and if $x_{a_1} \in I_{21}$ and we start simulating A_1 again, after updating p'_n, the context and round number. That is, we have the transition $(s_1, \ldots, s_{n-1}, p'_n, 1, n)$ on the first symbol of u_{21} to $(r, \ldots, s_{n-1}, s_n, 2, 1)$ where $s_n = (2, \ell_{2\ n}, V_n, a_n)$ iff $\ell'_{1n} = \ell_{21}$ and $x_{a_1} \in I_{21}$. Also, r is obtained from s_1 by a transition of A_1 on the first symbol of u_{21}. The check $x_{a_1} \in I_{21}$ is consistent with the check of $x_{\#_1} \in I_{21}$ in A_1. From $(r, \ldots, s_{n-1}, s_n, 2, 1)$, the processing of u_{21} happens as in A_1, and we continue till we finish processing u_{2n}. The same checks are repeated at the start of each fresh round.

It is clear that we have a run on w in the composition only when we have a globally correct sequence. On completing u_{kn}, this would lead to a location $(V_1, \ldots, V_{n-1}, V_n, k, n)$, each V_j obtained from the individual A_j. We define the accepting locations of the composition to be $\{(V_1, \ldots, V_n) \mid P_{kn} = (\ell'_{kn}, [0, \infty)), \ell'_{kn} \in F\}$. Clearly, whenever there is a run in A on w that ends up in $\ell'_{kn} \in F$, we have an accepting run on w in the composition. $\square$

The key idea of the determinization of k-ECMVPA follows from Lemma 8 and the determinizability of ECVPA [15]. Details are given in Appendix D.

Theorem 9. *k-ECMVPAs are determinizable.*

3.2 Determinizability of k-dtMVPA

Given a k-dtMVPA M, we first construct (untiming construction) a k-ECMVPA M' and a morphism h such that $L(M) = h(L(M'))$. We then use the determinizability of k-ECMVPA (Theorem 9) to obtain a deterministic k-ECMVPA M'' such that $L(M') = L(M'')$. We then show how to obtain a k-dtMVPA D from M'' preserving the determinism of M'' such that $L(D) = h(L(M'')) = h(L(M')) = L(M)$.

We give an intuition to the untiming construction, and give formal details in Appendix E. Each time a symbol is pushed on to a stack (say stack i), we guess its age (the time interval) at the time of popping the symbol. For instance, in the dtMVPA M, while pushing a symbol a on a stack, if we guess that the constraint checked at the time of the pop is $< \kappa$ for $\kappa \in \mathbb{N}$, then in the ECMVPA M', we push in the stack i, the symbol $(a, < \kappa, first)$ if this is the first symbol for which the guessed age is $< \kappa$. If $< \kappa$ has already been guessed as the age for a symbol pushed earlier, then we push $(a, < \kappa)$ onto the stack i. The guess $<_i \kappa$ is remembered in the finite control of the ECMVPA M'. Thus, for each symbol a pushed in stack i of the dtMVPA M, we push in stack i of the ECMVPA M', either $(a, < \kappa, first)$ or $(a, < \kappa)$ and remember $<_i \kappa$ in the finite control as a set of obligations. This information $<_i \kappa$ is retained in the finite control until popping the symbol $(a, < \kappa, first)$ from stack i. New symbols $<_i \kappa$ are added as internal symbols to the ECMVPA M'. The number of these symbols is finite since we have finitely many stacks and there is a maximum constant used in age comparisons of the dtMVPA M. After pushing $(a, < \kappa, first)$ onto the stack i, we read the internal symbol $<_i \kappa$, ensuring no time elapse since the last input symbol. Thus the event clock $x_{<_i\kappa}$ is reset at the same time as pushing $(a, < \kappa, first)$ on the stack. While popping $(a, < \kappa, first)$, we check that the value of the event clock $x_{<_i\kappa}$ is less than κ. Constraints of the form $> \kappa$ are handled similarly. Since the n stacks do not interfere with each other, this construction (adding extra symbols $<_i \kappa$ one per stack, retaining these symbols in the finite control until popping $(a, < \kappa, first)$ from stack i) can be done for all stacks, mimicking the timed stack. Note that the language accepted by the dtMVPA M is $h(L(M'))$, where h is the morphism which erases symbols of the form $<_i \kappa$ and $>_i \kappa$ from $L(M')$. This gives an ECMVPA preserving emptiness of the dtMVPA. We can determinize the ECMVPA M' obtaining $det(M')$ using Theorem 9. It remains to eliminate the transitions on the new symbols $<_i \kappa$ and $>_i \kappa$ from $det(M')$ and argue that the resulting machine stays deterministic and accepts $L(M)$.

Theorem 10. *k-dtMVPAs have decidable emptiness and are determinizable.*

4 Logical Characterization of k-dtMVPA

We consider a timed word $w = (a_0, t_0), (a_1, t_1), \ldots, (a_m, t_m)$ over alphabet $\Sigma = \langle \Sigma_c^i, \Sigma_{int}^i, \Sigma_r^i \rangle_{i=1}^n$ as a *word structure* over the universe $U = \{1, 2, \ldots, |w|\}$ of positions in the timed word. The predicates in the word structure are $Q_a(i)$ for $a \in \Sigma$ which evaluates to true at position i iff $w[i] = a$, where $w[i]$ denotes the ith position of w. Following [10], we use the matching binary relation $\mu_j(i, k)$ which evaluates to true iff the ith position is a call and the kth position is its matching return corresponding to the jth stack. We also introduce three predicates $\lhd_a$, $\rhd_a$, and θ_j capturing the following relations. For an interval I, the predicate $\lhd_a(i) \in I$ evaluates to true on the word structure iff $\nu_i^w(x_a) \in I$ for recorder clock x_a. For an interval I, the predicate $\rhd_a(i) \in I$ evaluates to true on the word structure iff $\nu_i^w(y_a) \in I$ for predictor clock y_a. For an interval I, the predicate $\theta_j(i) \in I$ evaluates to true on the word structure iff $w[i] \in \Sigma_r^j$, and there is some $k < i$ such that $\mu_j(k, i)$ evaluates to true and $t_i - t_k \in I$. The predicate $\theta_j(i)$ measures the time elapse between position k where a call was made on the stack j, and position i, its matching return. This time elapse is the age of the symbol pushed on to the stack during the call at position k. Since position i is the matching return, this symbol is popped at position i; if the age lies in the interval I, the predicate evaluates to true. We define MSO(Σ), the MSO logic over Σ, as:

$$\varphi := Q_a(x) \mid x \in X \mid \mu_j(x, y) \mid \lhd_a(x) \in I \mid \rhd_a(x) \in I \mid \theta_j(x) \in I \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists\, x.\varphi \mid \exists\, X.\varphi$$

where $a \in \Sigma$, $x_a \in C_\Sigma$, x is a first order variable and X is a second order variable.

The models of a formula $\phi \in$ MSO(Σ) are timed words w over Σ. The semantics of this logic is standard where first order variables are interpreted over positions of w and second order variables over subsets of positions. We define the language $L(\varphi)$ of an MSO sentence φ as the set of all words satisfying φ. Words in $Round(\Sigma, k)$, for some k rounds, can be captured by an MSO formula $Bd_k(\psi)$. For instance if $k = 1$, and n stacks, the formula $\exists x_1.(Q_{a^1}(x_1) \wedge \forall y_1(y_1 \leq x_1 \rightarrow Q_{a^1}(y_1)) \wedge \exists x_2.(x_1 < x_2 \wedge Q_{a^2}(x_2) \wedge \forall y_2(x_1 < y_2 < x_2 \rightarrow Q_{a^2}(y_2)) \wedge \ldots \wedge \exists x_n(x_{n-1} < x_n \wedge Q_{a^n}(x_n) \wedge last(x_n) \wedge \forall y_n(x_{n-1} < y_n < x_n \rightarrow Q_{a^n}(y_n)))))$, where $a^i \in \Sigma^i$ and $last(x)$ denotes x is the last position, captures a round. This can be extended to capture k-round words. Conjuncting the formula obtained from a dtMVPA M with $Bd_k(\psi)$ accepts only those words which lie in $L(M) \cap Round(\Sigma, k)$. Likewise, if one considers any MSO formula $\zeta = \varphi \wedge Bd_k(\psi)$, it can be shown that the dtMVPA M constructed for ζ will be a k-dtMVPA. The two directions, dtMVPA to MSO, as well as MSO to dtMVPA can be handled using standard techniques, and can be found in Appendix F.

Theorem 11. *A language L over Σ is accepted by an k-dtMVPA iff there is a MSO sentence φ over Σ such that $L(\varphi) \cap Round(\Sigma, k) = L$.*

References

1. Abdulla, P., Atig, M., Stenman, J.: Dense-timed pushdown automata. In: LICS. pp. 35–44 (2012)
2. Alur, R., Dill, D.: A theory of timed automata. TCS 126, 183–235 (1994)
3. Alur, R., Fix, L., Henzinger, T.A.: Event-clock automata: A determinizable class of timed automata. TCS 211(1-2), 253–273 (1999)
4. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: Symposium on Theory of Computing. pp. 202–211 (2004)
5. Atig, M.F.: Model-checking of ordered multi-pushdown automata. Logical Methods in Computer Science 8(3) (2012)
6. Bar-Hillel, Y., Perles, M., Shamir, E.: On formal properties of simple phrase structure grammars. Zeitschrift für Phonetik, Sprachwissenschaft und Kommunikationsforschung 14, 143–172 (1961)
7. Bhave, D., Dave, V., Krishna, S., Phawade, R., Trivedi, A.: A logical characterization for dense-time visibly pushdown automata. In: LATA. pp. 89–101 (2016)
8. Czerwinski, W., Hofman, P., Lasota, S.: Reachability problem for weak multi-pushdown automata. In: CONCUR, LNCS, vol. 7454, pp. 53–68. Springer (2012)
9. Esparza, J., Ganty, P., Majumdar, R.: A perfect model for bounded verification. In: LICS. pp. 285–294. IEEE Computer Society (2012)
10. La Torre, S., Madhusudan, P., Parlato, G.: A robust class of context-sensitive languages. In: LICS. pp. 161–170 (2007)
11. La Torre, S., Napoli, M., Parlato, G.: Scope-bounded pushdown languages. In: DLT, LNCS, vol. 8633, pp. 116–128. Springer International Publishing (2014)
12. La Torre, S., Napoli, M., Parlato, G.: A unifying approach for multistack pushdown automata. In: MFCS, LNCS, vol. 8634, pp. 377–389. Springer (2014)
13. Salvatore, L., Madhusudan, P., Parlato, G.: The language theory of bounded context-switching. In: LATIN. pp. 96–107 (2010)
14. Trivedi, A., Wojtczak, D.: Recursive timed automata. In: ATVA. LNCS, vol. 6252, pp. 306–324. Springer-Verlag (September 2010)
15. Van Tang, N., Ogawa, M.: Event-clock visibly pushdown automata. In: SOFSEM 2009, LNCS, vol. 5404, pp. 558–569. Springer (2009)

Appendix

A An Example of Concurrent Time-Critical Systems

In Android operating system every application has a main thread which is by default responsible for user interface (UI) management and hence can only be blocked for very short durations. If an application needs to perform blocking operations or asynchronous tasks which may block thread for longer durations, it forks additional threads. Android supports event-based architecture for UI management and inter-thread communication. Java class `Looper` implements incoming message queue and message processing loop which reads the next event in the queue and perform the corresponding action. Main thread maintains message queue for incoming messages using `Looper`. Other threads can send events like `Message` or `Runnable` (asynchronous function call) to the Main thread which are queued for processing. Additional threads may also use `Looper` and have their own incoming message queues.

We model such systems using an abstract event-based system architecture following Maiya et al. [1]. In this architecture, multiple processes communicate with each other using shared events and events are processed in the order of their arrival. We assume that each process has constant sized queue to store incoming events. Each process has event processing loop which reads next queued event and invoke corresponding event handler function. Additionally, events cannot remain pending in the queue for unbounded time. The *age* of an event is the time elapsed since an event is queued. When the age of an event exceeds some predefined threshold, it is dropped from the incoming event queue. Such condition is desirable to ensure responsiveness of interactive systems.

We propose formal modeling of such abstract multi-process event based architecture using dtMVPA. Let $P = \{P_1, P_2, \ldots, P_n\}$ be the set of processes that communicate among themselves using a shared set of events $E = \{e_1, e_2, \ldots, e_m\}$. Each process in P has its own fixed sized queue to store incoming events. Let Q_i be the incoming event queue for process P_i. Any process can send event to any other process by enqueuing it into receiver's incoming event queue. Each queued instance of an event has associated age which increases at the rate of one unit per unit time. Age is always initialized to zero. An event is dropped from the queue when its age exceeds some predefined threshold τ, which is assumed to be the same for all events.

We now describe dtMVPA model for the above architecture. We model k-sized event queues k-slot circular queues using finite automata control, where contents of the queue are remembered in the location. Send and receive operations are captured by introducing additional internal symbols as follows. We use symbol $S_k^{e_m, i \to j}$ to denote that the process P_i has sent event e_m to process P_j by enqueuing it into Q_j's k^{th} slot. Likewise, we use symbol $R_k^{e_m, i \to j}$ to denote

[1] Maiya, P., Gupta, R., Kanade, A., Majumdar, R.: Partial order reduction for event-driven multi-threaded programs. In: TACAS. Springer (2016)

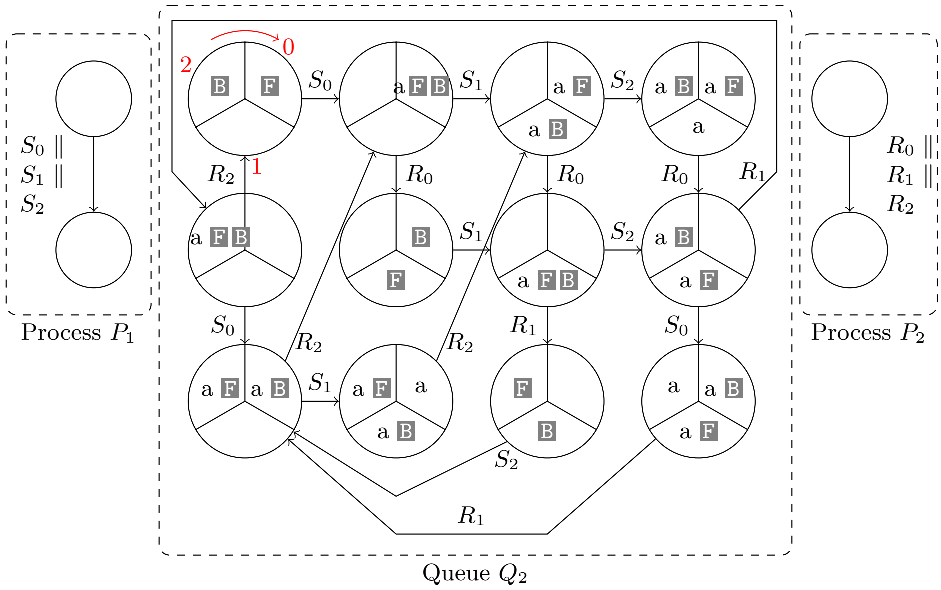

Fig. 2. dtMVPA model for Example 12

that the process P_j has received event e_m from process P_i by dequeuing it into Q_j's k^{th} slot. Thus, symbols $S_k^{e_m,i\to j}$ and $R_k^{e_m,i\to j}$ always occur in the pair along the run and the age of the event e_m at the time of dequeuing is the time difference between global timestamps of $S_k^{e_m,i\to j}$ and its corresponding $R_k^{e_m,i\to j}$. This model permits same event (but different instances) e_m to be enqueued more than once. But different instances of e_m occupy different lots in the queue and can be easily distinguished using their slot number. This is useful for modeling applications where different instances of the same type of service requests need to be distinguished. Although the stack is not used for modeling event queue, each process P_i has uses its own stack and is modeled using dtVPA. Direct product automaton of these processes and event queues yields desired dtMVPA.

Example 12. Assume we have only two processes $P = \{P_1, P_2\}$ and one event $E = \{e\}$. We assume the queue size $k = 3$. We use internal symbols $S_i^{e,1\to 2}$ and $R_i^{e,1\to 2}$ where $i \in \{0, 1, 2\}$, to capture send and receive operations. Figure A illustrates event processing for events sent from P_1 to P_2. For brevity we drop superscript ${}^{e,1\to 2}$ from internal symbols for the further discussion. When P_1 sends an event to P_2, it is marked by occurrence of any one of symbols S_0, S_1 or S_2. When P_2 receives the event, one of the symbols R_0, R_1 or R_2 occur. Whenever an event is enqueued in the i^{th} slot of the event queue, the occurrence a symbol S_i models this fact. Similarly R_i denotes the fact that an event is received from i^{th} slot. Event queue of size three is implemented using stackless event clock automaton. In figure A, automaton Q_2 models the circular queue.

Each of location of Q_2 pictorially shows three slot circular queue along with queue contents. Legends are shown (in red) on the initial location describing slot numbers and direction of enqueue. Marker F denotes the position of front pointer from where event is dequeued. Marker B denotes the position of back pointer. While enqueuing new event, B is first incremented and then new event is stored. When dequeuing, front element of the queue is read and front pointer is incremented. Here, processes P_1 and P_2 simply send and receive an event respectively, but they can be any arbitrary dtVPA involving stack operations. We add guard (not shown) $y_{R_i} \leq \tau$? on transition labeled S_i. This captures the requirement that no event waits for more than τ time units in the queue.

B Closure of k-dtMVPA under Boolean operators

Consider the same underlying alphabet $\Sigma = \langle \Sigma_c^i, \Sigma_r^i, \Sigma_{int}^i \rangle_{i=1}^n$, for two dtMVPA M_1, M_2 with stack alphabets $\Gamma_1 = \bigcup_{i=1}^n \Gamma_1^i$ and $\Gamma_2 = \bigcup_{i=1}^n \Gamma_2^i$ respectively. Without loss of generality, assume the locations of M_1, M_2 to be disjoint. Union then follows simply by taking the union of the (initial and final) locations and transitions of M_1, M_2.

For the intersection, we consider the product of $M_l = (L_l, \Sigma, \Gamma_l, L_l^0, L_l^f, \Delta_l)$, for l in $\{1,2\}$, where $\Delta_l = (\Delta_{cl}^i \cup \Delta_{rl}^i \cup \Delta_{intl}^i)_{i=1}^n$, we construct the dtMVPA $M = (L, \Sigma, \Gamma, L^0, F, \Delta)$ with initial locations $L^0 = \{(l_1^0, l_2^0) \mid l_1^0 \in L_1^0 \text{ and } l_2^0 \in L_2^0\}$, with final locations $F = \{(f_1, f_2) \mid f_1 \in L_1^f, f_2 \in L_2^f\}$ and with stack alphabet $\Gamma = \bigcup_{i=1}^n (\Gamma_1^i \times \Gamma_2^i)$, The transition function $\Delta = (\Delta_c^i \cup \Delta_r^i \cup \Delta_{int}^i)_{i=1}^n$, is follows:

1. For $a \in \Sigma_c^i$, transition $((q_1, q_2), a, \varphi_1 \wedge \varphi_2, (q_1', q_2'), (\gamma_1, \gamma_2)) \in \Delta_c^i$ iff transition $(q_1, a, \varphi_1, q_1', \gamma_1)$ is in Δ_{c1}^i and transition $(q_2, a, \varphi_2, q_2', \gamma_2)$ is in Δ_{c2}^i. The age of (γ_1, γ_2) is initialized to 0. The push operations of M_1, M_2 are synchronized.
2. For $a \in \Sigma_r^i$, transition $((q_1, q_2), a, I_1 \wedge I_2, (\gamma_1, \gamma_2), \varphi_1 \wedge \varphi_2, (q_1', q_2')) \in \Delta_r^i$ iff transition $(q_1, a, I_1, \gamma_1, \varphi_1, q_1')$ is in Δ_{r1}^i and transition $(q_2, a, I_2, \gamma_2, \varphi_2, q_2')$ is in Δ_{r2}^i. Note that the age of (γ_1, γ_2) must satisfy $I_1 \wedge I_2$ for popping. The pop operations of M_1, M_2 are synchronized.
3. For $a \in \Sigma_{intl}^i$, transition $((q_1, q_2), a, \varphi_1 \wedge \varphi_2, (q_1', q_2')) \in \Delta_l^i$ iff $(q_1, a, \varphi_1, q_1') \in \Delta_{int1}^i$ and $(q_2, a, \varphi_2, q_2') \in \Delta_{int2}^i$.

It is easy to see that $L(M) = L(M_1) \cap L(M_2)$. We thus have By Theorem 6 closure under determinizability which gives us closure under complementation.

Lemma 13. *The class of languages characterized by k-dtMVPA are closed under Boolean operators.*

C Details of Lemma 7: Round-Switching Lemma

Lemma 14. *(Round Switching Lemma for A_j) Let $A = (k, L, \Sigma, \Gamma, L^0, F, \Delta)$ be a k-ECMVPA. Let $w = u_1 u_2 \dots u_k$ with $u_i = u_{i1} u_{i2} \dots u_{in}$, $1 \leq i \leq k$. Then we can construct a ECVPA A_j over $\Sigma^j \cup \{\#_1, \dots, \#_k\}$ which reaches a location V_j on reading κ_j iff V_j is a correct sequence of round switches for A_j.*

Proof. The locations of A_j are $V_j \cup \{(i, \ell_{ij}, V_j), (i, \ell_{ij}, V_j, \#), (i, \ell_{ij}, V_j, a) \mid 1 \leq i \leq k, \ell \in L, a \in \Sigma^j, V_j \in ((L \times I) \times L)^k\}, \cup((L \times I) \times L)^k, I \in \mathcal{I}$. The set of initial locations are $\{(1, \ell_{1j}, V_j) \mid V_j \in ((L \times I) \times L)^k, I \in \mathcal{I}\}$. We provide the details of the transitions of the ECVPA A_j. Formally, the transitions δ^j of A_j can be summarized as follows:

- For $2 \leq h \leq k$, $\langle (h-1, \ell'_{h-1j}, V_j, a), \#_{h-1}, x_a = 0, (h, \ell_{hj}, V_j, \#) \rangle$ if $((\ell_{h-1j}, I_{h-1j}), \ell'_{h-1j}) = P_{h-1j}$, and $P_h = ((\ell_{hj}, I_{hj}), \ell'_{hj})$, (After reading the last symbol $a \in \Sigma^j$ of u_{h-1j}, A_j reads the symbol $\#_{h-1}$, and checks $x_a = 0$ to ensure that there is no time elapse in reading $\#_{h-1}$. The location $(h-1, \ell'_{h-1j}, V_j, a)$ signifies that A_j has read the last symbol of u_{h-1j}. The location $(h, \ell_{hj}, V_j, \#)$ is entered on reading $\#_{h-1}$.)
- $\langle (h, \ell_{hj}, V_j, \#), a, \varphi \wedge x_{\#_{h-1}} \in I_{hj}, (h, \ell, V_j) \rangle$ if $(\ell_{hj}, a, \varphi, \ell) \in \Delta^j_{int}$ and for $2 \leq h \leq k$ we have $((\ell_{hj}, I_{hj}), \ell'_{hj}) = P_{hj}$.
- $\langle (h, \ell_{hj}, V_j, \#), a, \varphi \wedge x_{\#_{h-1}} \in I_{hj}, (h, \ell, V_j), \gamma \rangle$ if $(\ell_{hj}, a, \varphi, \ell, \gamma) \in \Delta^j_c$ and for $2 \leq h \leq k$ we have $((\ell_{hj}, I_{hj}), \ell'_{hj}) = P_{hj}$,
- $\langle (h, \ell_{hj}, V_j, \#), a, \gamma, \varphi \wedge x_{\#_{h-1}} \in I_{hj}, (h, \ell, V) \rangle$ if $(\ell_{hj}, a, \gamma, \varphi, \ell) \in \Delta^j_r$ and $((\ell_{hj}, I_{hj}), \ell'_{hj}) = P_{hj}$, $2 \leq h \leq k$, (From the location $(h, \ell_{hj}, V_j, \#)$, A_j starts reading u_{hj}. To check that the time elapsed between the processing of u_{h-1j} and u_{hj} in A lies in the interval T_{hj}, we have the constraint $x_{\#_{h-1}} \in I_{hj}$. In addition, we also check the constraint φ that was checked by A while reading the first symbol a of u_{hj}. The location (h, ℓ_{hj}, V_j) is entered. A_j continues in this location until u_{hj} is completely read. On reading the last symbol a of u_{hj}, the location (h, ℓ_{hj}, V_j, a) is entered. Then on reading $\#_{h+1}$, the location $(h+1, \ell_{h+1j}, V_j, \#)$ is entered. A_j then enters location $(h+1, \ell_{h+1j}, V_j)$ and starts processing u_{h+1j} and so on.)

- $\langle (h, \ell, V_j), a, \varphi, (h, \ell', V_j) \rangle$ if $(\ell, a, \varphi, \ell') \in \Delta^j_{int}$
- $\langle (h, \ell, V_j), a, \varphi, (h, \ell', V_j, a) \rangle$ if $(\ell, a, \varphi, \ell') \in \Delta^j_{int}$
- $\langle (h, \ell, V_j), a, \varphi, (h, \ell', V_j), \gamma \rangle$ if $(\ell, a, \varphi, \ell', \gamma) \in \Delta^j_c$
- $\langle (h, \ell, V_j), a, \varphi, (h, \ell', V_j, a), \gamma \rangle$ if $(\ell, a, \varphi, \ell', \gamma) \in \Delta^j_c$
- $\langle (h, \ell, V_j), a, \gamma, \varphi, (h, \ell', V_j) \rangle$ if $(\ell, a, \gamma, \varphi, \ell') \in \Delta^j_r$
- $\langle (h, \ell, V_j), a, \gamma, \varphi, (h, \ell', V_j, a) \rangle$ if $(\ell, a, \gamma, \varphi, \ell') \in \Delta^j_r$ (A_j processes u_{hj} in location (h, ℓ, V_j). The next location can be either (h, ℓ, V_j) if the symbol a read is not the last symbol of u_{hj}, or (h, ℓ, V_j, a) if the symbol a read is the last symbol of u_{hj}.)
- $\langle (k, \ell'_{kj}, V_j, a), \#_k, x_a = 0, V_j \rangle$ if $P_{kj} = ((\ell_{kj}, I_{kj}), \ell'_{kj})$ (On reading the last symbol a of u_{kj}, A_j enters location V_j, after correctly processing $u_{1j}, \ldots, u_{kj}$)

The construction is now complete. The correctness of the construction is straightforward to verify. □

D Details for Theorem 9

We first recall some basic results for the reader's convenience.

D.1 VPA Determinization

Given a VPA $M = (Q, Q_{in}, \Gamma, \delta, Q_F)$, the idea in [4] is to do a subset construction. Let $w = w_1 a_1 w_2 a_2 w_3$ be a string such that every call in w_1, w_2, w_3 has a matching return, and a_1, a_2, a_3 are call symbols without matching returns. After reading w, the deterministic VPA has in its stack the contents $(S_2, R_2, a_2)(S_1, R_1, a_1)\bot$ and its control state is (S, R). Here, S_2 contains all pairs of states (q, q') such that starting with q on w_2 and an empty stack (contains only $\bot$), we reach q' with stack $\bot$. The set of pairs of states S_2 is called a summary for w_2. Likewise, S_1 is a summary for w_1 and S is the summary for w_3. Here R_i is the set of states reachable from the initial state after reading till the end of w_i, $i = 1, 2$ and R is the set of reachable states obtained on reading w.

After w_3, if a call a_3 occurs, then (S, R, a_3) is pushed on the stack, and the current state is (S', R') where $S' = \{(q, q) \mid q \in Q\}$, while R' is obtained by updating R using all transitions for a_3. The current control state (S, R) is updated to (S', R') where R' is all reachable states obtained from R, using all possible transitions on the current symbol read. The set S' is obtained as follows:

- On reading an internal symbol, S evolves into S' where $S' = \{(q, q') \mid \exists q'', (q, q'') \in S, (q'', a, q') \in \delta\}$.
- On reading a call symbol a, (S, R, a) is pushed onto the stack, and the control state is (S', R') where $S' = \{(q, q) \mid q \in Q\}$. On each call, S' is re-initialized.
- On reading a return symbol a', let the top of stack be (S_1, R_1, a). This is popped. Thus, a and a' are a matching call-return pair. Let the string read so far be $waw'a'$. Clearly, w, w' are well-nested, or all calls in them have seen their returns.

  For the well-nested string w preceding a, we have S_1 consisting of all (q, q') such that starting on q on w, we reach q' with empty stack. Also, S consists of pairs (q_1, q_2) that have been obtained since the call symbol a (corresponding to the return symbol a') was pushed onto the stack. The set S started out as $\{(q_1, q_1) \mid q_1 \in Q\}$ on pushing a, and contains pairs (q_1, q_2) such that on reading the well-nested string between a and a', starting in q_1, we reach q_2. The set S is updated to S' by "stitching" S_1 and S as follows: A pair $(q, q') \in S'$ if there is some $(q, q'') \in S_1$, and $(q'', a, q_1, \gamma) \in \delta$ (the push transition on a), $(q_1, q_2) \in S$, and $(q_2, a', \gamma, q') \in \delta$ (the pop transition on a').

The set of final locations of the determinized VPA are $\{(S, R) \mid R$ contains a final state of the starting VPA $\}$, and its initial location is the set of all pairs (S_{in}, R_{in}) where $S_{in} = \{(q, q) \mid q \in Q\}$ and R_{in} is the set of all initial states of the starting VPA.

D.2 ECVPA to VPA

We quickly recall the conversion from ECVPA to VPA [15]. For example, a transition of the form $(q, a, x_c < 2, q')$ in the ECVPA is replaced with the transitions $(q, (a, (c, (0, 0))), q')$, $(q, (a, (c, (0, 1))), q')$, $(q, (a, (c, (1, 1))), q')$ and $(q, (a, (c, (1, 2))), q')$ in the VPA. These transitions are deterministic since the intervals involved in the alphabet are disjoint. The VPA obtained like this is determinized as explained above. The resulting VPA is then converted back to a deterministic ECVPA by reverting to the original alphabet, and translating the interval alphabet to clock constraints. For instance, the transitions introduced above in the VPA become $(q, a, x_c = 0, q')$, $(q, a, 0 < x_c < 1, q')$, $(q, a, x_c = 1, q')$ and $(q, a, 1 < x_c < 2, q')$.

D.3 Proof of Theorem 9

Let $A = (k, L, \Sigma, \Gamma, L^0, F, \Delta)$ be the k-ECMVPA and let A_j be the ECVPA on $\Sigma^j \cup \{\#_1, \#_2, \ldots, \#_k\}$. Each A_j is determinizable [15]. Recall from [15] that an ECVPA A_j is untimed to obtain a VPA $ut(A_j)$ by encoding the clock constraints of A_j in an extended alphabet. This VPA can be converted back into an ECVPA $ec(ut(A_j))$ by using the original alphabet, and replacing the clock constraints. This construction is such that $L(ec(ut(A_j))) = L(A_j)$ and both steps involved preserve determinism. Determinization of VPA $ut(A_j)$ is done in the usual way [4]. This gives $Det(ut(A_j))$. Again, $ec(Det(ut(A_j)))$ converts this back into a ECVPA by simplifying the alphabet, and writing the clock constraints. The set of locations remain unchanged in $ec(Det(ut(A_j)))$ and $Det(ut(A_j))$. This translation also preserves determinism, hence $B_j = ec(Det(ut(A_j)))$ is a deterministic ECVPA language equivalent to ECVPA A_j.

The locations of B_j are thus of the form (S, R) where R is the set of all reachable control states of A_j and S is a set of ordered pairs of states of A_j as seen in section D.1. On reading κ_j, the R component of the state reached in B_j is the set $\{\langle V_j \rangle \mid V_j$ is a correct round switching sequence of $A_j\}$. Lemmas 7 and Lemma 8 follow easily using $B_j = ec(Det(ut(A_j)))$ in place of A_j. We now obtain a deterministic ECMVPA B which simulates $B_1, \ldots, B_n$ one after the other on reading w. Given $w = u_1 u_2 \ldots u_k$, B invokes $B_1, \ldots, B_n$ in round robin fashion for k times : the first time each B_j processes u_{1j}, the second time u_{2j} and so on till each B_j processes u_{kj} in the last round. Automaton B keeps track in its finite control, the locations of all the B_j's, along with the valuations of all the recorders and predictors of Σ. It also remembers the current round number and the current context number in its finite control to ensure a correct round robin simulation of the B_j's. To achieve this, we make use of correct sequence of round switches of nondeterministic A_js.

Let $B_j = (Q^j, \Sigma^j \cup \{\#_1, \ldots, \#_k\}, \Gamma^j, Q_0^j, F^j, \delta^j)$. Locations of B_j have the form (S_j, R_j). The initial state of B_j is the set consisting of all (S_{in}, R_{in}) where $S_{in} = \{(q, q) \mid q$ is a state of $A_j\}$, and R_{in} is the set of all initial states of A_j. Recall that a final state of A_j is V_j if V_j is a correct switching sequence of A_j. Thus, an accepting run in B_j goes through states $(S_{in}, R_{in}), (S_1, R_1) \ldots, (S_n, R_n), \langle V_j \rangle$ where $\langle V_j \rangle$ is a set that contains a correct switching sequence V_j of A_j.

Locations of B have the form $(q_1, \ldots, q_n, i, j)$ where q_y is a location of B_y, i, j are respectively the current round and context. The initial location of B is $(q_{11}, q_{12}, \ldots, q_{1n}, 1, 1)$ where q_{1j} is the initial location of B_j. We define the set of final locations of B to be $(\langle V_1 \rangle, \ldots \langle V_{n-1} \rangle \langle S_n, R_n \rangle)$ where R_n is a set containing a tuple of the form (k, l'_{kn}, V_n, a) and l'_{kn} is in F, the set of final locations of A.

We now explain the transitions Δ of B, using the transitions δ^j of B_j. Recall that B processes $w = u_1 u_2 \ldots u_k$, with $u_i = u_{i1} u_{i2} \ldots u_{in}$. Let $u_{ij} = (a_j^1, t_{ij}^1) \ldots (a_j^{last}, t_{ij}^{last})$, where $t_{ij}^1, \ldots, t_{ij}^{last}$ are the time stamps on reading u_{ij}. Let $\kappa_j = u_{1j}(\#_1, t_{1j}^{last}) u_{2j}(\#_2, t_{2j}^{last}) \ldots u_{kj}(\#_k, t_{kj}^{last})$. Each B_j processes κ_j.

Let $\eta = (q_{i1}, \ldots, q_{ij-1}, q_{ij}, q_{i-1,j+1}, \ldots, q_{i-1n}, i, j)$ and let $\zeta = (q_{i1}, \ldots, q_{ij-1}, q, q_{i-1,j+1}, \ldots, q_{i-1n}, i, j)$, where q_{ij} is the location reached by B_j after it has completed its round i, that is after processing u_{ij}.

1. Simulation of B_j when $j < n$ and the round is $i < k$.
   - $\langle \eta, a, \varphi, \zeta \rangle \in \Delta_{int}^j$ iff $(q_{ij}, a, \varphi, q) \in \delta_{int}^j$
   - $\langle \eta, a, \varphi, \zeta, \gamma \rangle \in \Delta_c^j$ iff $(q_{ij}, a, \varphi, \gamma, q) \in \delta_c^j$
   - $\langle \eta, a, \gamma, \varphi, \zeta \rangle \in \Delta_r^j$ iff $(q_{ij}, a, \gamma, \varphi, q) \in \delta_r^j$
2. Change context from j to $j+1$ in round $i > 1$. This is the time when B_j completes processing u_{ij}, reads a $\#_i$ in the location $q'_{ij} = (S'_{ij}, R'_{ij})$ which has been computed at the end of u_{ij}. Here, R'_{ij} is the set of all reachable states of A_j after i rounds. Recall that these are states of the form (i, ℓ'_{ij}, V_j, a), where V_j is a correct switching sequence of A_j and a is the last symbol of u_{ij}. The location $q_{i-1j+1} = (S_{i-1j+1}, R_{i-1j+1})$ of B_{j+1} at this point is such that R_{i-1j+1} is the set of states reached at the end of simulating $u_{i-1j+1}\#_{i-1}$, which is the set of states of the form $(i, \ell_{ij+1}, V_{j+1}, a)$ (refer the transition from p'_{11} to p''_{11} in Lemma 8 when changing context. We remember the last symbol of the current string u_{ij} and use it to check the time elapse on starting u_{i+1j}.)
   The location of B at this time is thus $(\ldots, q'_{ij}, q_{i-1j+1}, \ldots)$. Thanks to Lemma 8, we know that for each $(i, \ell'_{ij}, V_j, a) \in R'_{ij}$, there is a $(i, \ell_{ij+1}, V_{j+1}) \in R_{i-1j+1}$ with $\ell'_{ij} = \ell_{ij+1}$. That is, $V_j V_{j+1}$ is part of a globally correct sequence for A. We denote this fact by writing $L'_{ij} = L_{i-1j+1}$. When $L'_{ij} = L_{ij+1}$, B starts processing u_{ij+1} by running B_{j+1} on the first symbol of u_{ij+1} from location $(\ldots, q'_{ij}, q_{i-1j+1}, \ldots, i, j)$. Component location q_{i-1j+1} will be replaced based on a transition of B_{j+1}, and q'_{ij} is also replaced with q_{ij} to take care of the transition on $\#_i$ in B_j, where $q_{ij} = (S_{ij}, R_{ij})$ with R_{ij} containing all locations of the form $\langle i, \ell_{ij}, V_j, a \rangle$, where a is the last symbol of u_{ij} (refer the transition from p'_{11} to p''_{11} in Lemma 8)
   - $\langle (\ldots, q'_{ij}, q_{i-1j+1}, \ldots, i, j), a, \varphi, (\ldots, q_{ij}, q, \ldots, i, j+1) \rangle \in \Delta_{int}^{j+1}$ iff $(q_{i-1j+1}, a, \varphi, q) \in \delta_{int}^{j+1}$
   - $\langle (\ldots, q'_{ij}, q_{i-1j+1}, \ldots, i, j), a, \varphi, (\ldots, q_{ij}, q, \ldots, i, j+1), \gamma \rangle \in \Delta_c^{j+1}$ iff $(q_{i-1j+1}, a, \varphi, q, \gamma) \in \delta_c^{j+1}$
   - $\langle (\ldots, q'_{ij}, q_{i-1j+1}, \ldots, i, j), a, \gamma, \varphi, (\ldots, q_{ij}, q, \ldots, i, j+1) \rangle \in \Delta_r^{j+1}$ iff $(q_{i-1j+1}, a, \gamma, \varphi, q) \in \delta_r^{j+1}$

Transitions of B_{j+1} continue on $(\ldots, q_{ij}, q, \ldots, i, j+1)$ replacing only the $(j+1)$th entry until u_{ij+1} is read completely.
When $i = 1$, we have $q'_{1j} = (S'_{1j}, R'_{1j})$ which has been computed at the end of u_{1j}, where R'_{1j} is the set of all reachable states of A_j after the first round. The initial location was $(q_1, \ldots, q_n, 1, 1)$. The location reached now looks like $(q'_{11}, \ldots, q'_{1j}, q_{j+1}, \ldots, q_n, 1, 1)$, with $q_{j+1} = (S_{j+1}, R_{j+1})$, where R_{j+1} is all possible initial locations of A_{j+1}. We start processing B_{j+1} on u_{1j+1} when $L'_{1j} = L_{1j+1}$, as seen above.

3. Change context from n to 1 on consecutive rounds i and $i+1 < k$. This is the time when B_n completes processing u_{in} and B_1 starts processing u_{i+11}. As seen above, the location $q'_{in} = (S'_{in}, R'_{in})$ is reached in B_n after u_{in} with R'_{in} the set of locations of the form $(i, \ell'_{in}, V_n, a_n)$ where a_n is the last symbol of u_{in}. Also, B_1 is in location $q_{i+11} = (S_{i+11}, R_{i+11})$ after processing $u_{i1}\#_i$, where R_{i+11} is the set of all locations of the form $(i+1, \ell_{i+11}, V_1, a_1)$, and a_1 is the last symbol of u_{i1}. The location of B at this time is thus $(q_{i+11}, q_{i+12}, \ldots, q'_{in})$. Thanks to Lemma 8, we know that for each $(i, \ell'_{in}, V_n, a_n) \in R'_{in}$, there is a $(i+1, \ell_{i+11}, V_1, a_1) \in R'_{i+11}$ with $\ell'_{in} = \ell_{i+11}$. That is, V_n and V_1 are part of some globally correct sequence for A. We denote this fact by $L_{i+11} = L'_{in}$.
When $L_{i+11} = L'_{in}$, automata B starts running B_1 on u_{i+11} from the location $(q_{i+11}, \ldots, q'_{in}, i, n)$, state q_{i+11} will be replaced based on a transition of B_1. We also replace q'_{in} with q_{in} as it happens in B_n when the $\#_i$ is read. Initial state is $q_{in} = (S_{in}, R_{in})$ where R_{in} has all locations of the form (i, ℓ_{in}, V_n, a_n).
   - $\langle (q_{i+11}, \ldots, q'_{in}, i, n), a, \varphi, (q, \ldots, q_{in}, i+1, 1) \rangle \in \Delta^1_{int}$ iff $(q_{i+11}, a, \varphi, q) \in \delta^1_{int}$
   - $\langle (q_{i+11}, \ldots, q'_{in}, i, n), a, \varphi, (q, \ldots, q_{in}, i{+}1, 1), \gamma \rangle \in \Delta^1_c$ iff $(q_{i+11}, a, \varphi, q, \gamma) \in \delta^1_c$
   - $\langle (q_{i+11}, \ldots, q'_{in}, i, n), a, \gamma, \varphi, (q, \ldots, q_{in}, i{+}1, 1) \rangle \in \Delta^1_c$ iff $(q_{i+11}, a, \gamma, \varphi, q) \in \delta^1_r$

   Transitions of B_1 continue on $(q, \ldots, q_{in}, i{+}1, 1)$ replacing only the first entry until u_{i+11} is read completely.
4. Simulation of B_n in round k. This happens when B has completed reading u_{kn-1} by simulating B_{n-1} on u_{kn-1}. The location of B at this point of time is $(q_{k1}, q_{k2}, \ldots, q'_{kn-1}, q_{k-1n})$, where $q_{kj} = (S_{kj}, R_{kj})$ with R_{kj} is the set of locations $\langle V_j \rangle$. $q'_{kn-1} = (S_{kn-1}, R_{kn-1})$ where R_{kn-1} is the set of all locations of the form $\langle k, \ell'_{kn-1}, V_{n-1}, a \rangle$ where a is the last symbol of u_{kn-1} and $q_{k-1n} = (S_{k-1n}, R_{k-1n})$ with R_{k-1n} is the set of all locations (k, ℓ_{kn}, V_n, b). Again, thanks to Lemma 8, we know that for each $(k, \ell'_{kn-1}, V_{n-1}, a) \in R_{kn-1}$, there is a $(k, \ell_{kn}, V_n, b) \in R_{k-1n}$ such that $\ell'_{kn-1} = \ell_{kn}$. We denote this with $L'_{kn-1} = L_{k-1n}$.
   When $L_{kn-1} = L_{k-1n}$, automata B starts running B_n on u_{kn} from the location $(q_{k1}, q_{k2}, \ldots, q'_{kn-1}, q_{k-1n})$, and q_{k-1n} is replaced by a transition of B_n, while q'_{kn-1} is replaced with $q_{kn-1} = (S_{kn-1}, R_{kn-1})$ to simulate the transition on $\#_k$ by B_{n-1}, where $R_{kn-1} = \langle V_{n-1} \rangle$.
   Let $\eta = (q_{k1}, q_{k2}, \ldots, q'_{kn-1}, q_{k-1n}, k{-}1, n)$ and $\zeta = (q_{k1}, q_{k2}, \ldots, q_{kn-1}, q, k, n)$,
   - $\langle \eta, a, \varphi, \zeta \rangle \in \Delta^n_{int}$ iff $(q_{k-1n}, a, \varphi, q) \in \delta^n_{int}$,
   - $\langle \eta, a, \varphi, \zeta, \gamma \rangle \in \Delta^n_c$ iff $(q_{k-1n}, a, \varphi, q, \gamma) \in \delta^n_c$,

– $\langle \eta, a, \gamma, \varphi, \zeta \rangle \in \Delta_r^n$ iff $(q_{k-1n}, a, \gamma, \varphi, q) \in \delta_r^n$.

Transitions of B_n continue on $(q_{k1}, q_{k2}, \ldots, q_{kn-1}, q, k, n)$ replacing only the nth entry based on transitions of B_n. When B_n completes reading u_{kn}, it reaches the location $q'_{kn} = (S'_{kn}, R'_{kn})$ where R'_{kn} is the set of all locations of the form (k, ℓ'_{kn}, V_n, a), where a is the last symbol of u_{kn}. Since there are no more symbols to be read, the location reached is $q'_{kn} = (S'_{kn}, R'_{kn})$. Unlike the earlier rounds where we processed $\#_i$ on B_j in parallel (after completing u_{ij}) and started B_{j+1} on the first symbol of u_{ij+1}, when B_n finishes u_{kn}, there is no processing that remains. Hence, we are at $q'_{kn} = (S'_{kn}, R'_{kn})$ at the end of the kth round of B_n. This is accepting iff there exists $\ell'_{kn} \in R'_{kn}$ such that $\ell'_{kn} \in F$. The state reached in B is then $(\langle V_1 \rangle, \langle V_2 \rangle, \ldots, \langle V_{n-1} \rangle, q'_{kn})$. Note that we have ensured the following:

(a) $\langle V_j \rangle$ contains a correct switching sequence V_j for A_j, and we ensure that $V_j V_{j+1}$ is part of a correct global sequence, for all $1 \leq j \leq n-2$,

(b) We have the condition $L'_{kn-1} = L_{k-1n}$, ensuring the continuity between $\langle V_{n-1} \rangle$ and the start of B_n in the kth round.

(c) At the end of u_{kn}, we reach in B_n, $q'_{kn} = (S'_{kn}, R'_{kn})$ such that $\ell'_{kn} \in R'_{kn}$ such that $\ell'_{kn} \in F$.

The above conditions ensure correctness of local switching and a globally correct sequence in A. Clearly, $w \in L(B)$ iff $w \in L(A)$ iff there is some globally correct sequence $V_1 \ldots V_n$.

E Details of Theorem 10

We now describe the *untiming-the-stack* construction to obtain from a k-dtMVPA M over Σ, an k-ECMVPA M' over an extended alphabet Σ' such that $L(M) = h(L(M'))$ where h is a homomorphism $h : \Sigma' \times \mathbb{R}^{\geq 0} \to \Sigma \times \mathbb{R}^{\geq 0}$ defined as $h(a,t) = (a,t)$ for $a \in \Sigma$ and $h(a,t) = \varepsilon$ for $a \notin \Sigma$. Our construction builds upon that of [7].

Let κ be the maximum constant used in the k-dtMVPA M while checking the age of a popped symbol in any of the stacks. Let us first consider a call transition $(l, a, \varphi, l', \gamma) \in \Delta_c^i$ encountered in M. To construct an ECMVPA M' from M, we guess the interval used in the return transition when γ is popped from ith stack. Assume the guess is an interval of the form $[0, \kappa)$. This amounts to checking that the age of γ at the time of popping is $<\kappa$. In M', the control switches from l to a special location $(l'_{a,<\kappa}, \{<_i \kappa\})$, and the symbol $(\gamma, <\kappa, \texttt{first})$[2] is pushed onto the ith stack.

Let $Z_i^{\sim} = \{\sim_i c \mid c \in \mathbb{N}, c \leq k, \sim \in \{<, \leq, >, \geq\}\}$. Let $\Sigma'_i = \Sigma^i \cup Z_i^{\sim}$ be the extended alphabet for transitions on the ith stack. All symbols of $Z_i^{\sim}$ are internal symbols in M' i.e. $\Sigma'_i = \left\{\Sigma_c^i, \Sigma_{int}^i \cup Z_i^{\sim}, \Sigma_r^i\right\}$. At location $(l'_{a,<\kappa}, \{<_i \kappa\})$, the new symbol $<_i \kappa$ is read and we have the following transition : $((l'_{a,<\kappa}, \{<_i \kappa\}), <_i \kappa, x_a = 0, (l', \{<_i \kappa\}))$, which results in resetting the event recorder $x_{<_i \kappa}$ corresponding to the new symbol $<_i \kappa$. The constraint $x_a = 0$

[2] It is sufficient to push $(\gamma, <\kappa, \texttt{first})$ in stack i, since the stack number is known as i

ensures that no time is elapsed by the new transition. The information $<_i\kappa$ is retained in the control state until $(\gamma, <\kappa, \texttt{first})$ is popped from ith stack. At $(l', \{<_i\kappa\}))$, we continue the simulation of M from l'. Assume that we have another push operation on ith stack at l' of the form (l', b, ψ, q, β). In M', from $(l', \{<_i\kappa\})$, we first guess the constraint that will be checked when β will be popped from the ith stack. If the guessed constraint is again $<_i\kappa$, then control switches from $(l', \{<_i\kappa\})$ to $(q, \{<_i\kappa\})$, and $(\beta, <\kappa, -)$ is pushed onto the ith stack and simulation continues from $(q, \{<_i\kappa\})$. However, if the guessed pop constraint is $<_i\zeta$ for $\zeta \neq \kappa$, then control switches from $(l', \{<_i\kappa\})$ to $(q_{b,<\zeta}, \{<_i\kappa, <_i\zeta\})$ on reading b. The new obligation $<_i\zeta$ is also remembered in the control state. From $(q_{b,<\zeta}, \{<_i\kappa, <_i\zeta\})$, we read the new symbol $<_i\zeta$ which resets the event recorder $x_{<_i\zeta}$ and control switches to $(q, \{<_i\kappa, <_i\zeta\})$, pushing $(\beta, <\zeta, \texttt{first})$ on to the ith stack. The idea thus is to keep the obligation $<_i\kappa$ alive in the control state until γ is popped; the value of $x_{<_i\kappa}$ at the time of the pop determines whether the pop is successful or not. If a further $<_i\kappa$ constraint is encountered while the obligation $<_i\kappa$ is already alive, then we do not reset the event clock $x_{<_i\kappa}$. The $x_{<_i\kappa}$ is reset only at the next call transition after $(\gamma, <\kappa, \texttt{first})$ is popped from ith stack , when $<_i\kappa$ is again guessed. The case when the guessed popped constraint is of the form $>_i\kappa$ is similar. In this case, each time the guess is made, we reset the event recorder $x_{>_i\kappa}$ at the time of the push. If the age of a symbol pushed later is $>\kappa$, so will be the age of a symbol pushed earlier. In this case, the obligation $>\kappa$ is remembered only in the stack and not in the finite control. Handling guesses of the form $\geq \zeta \wedge \leq \kappa$ is similar, and we combine the ideas discussed above.

Now consider a return transition $(l, a, I, \gamma, \varphi, l') \in \Delta_r^i$ in M. In M', we are at some control state (l, P). On reading a, we check the top of the ith stack symbol in M'. It is of the form $(\gamma, S, \texttt{first})$ or $(\gamma, S, -)$, where S is a singleton set of the form $\{<\kappa\}$ or $\{>\zeta\}$, or a set of the form $\{<\kappa, >\zeta\}$[3]. Consider the case when the top of the ith stack symbol is $(\gamma, \{<\kappa, >\zeta\}, \texttt{first})$. In M', on reading a, the control switches from (l, P) to (l', P') for $P' = P \backslash \{<\kappa\}$ iff the guard φ evaluates to true, the interval I is (ζ, κ) (this validates our guess made at the time of push) and the value of clock $x_{<_i\kappa}$ is $<\kappa$, and the value of clock $x_{>_i\zeta}$ is $>\zeta$. Note that the third component $\texttt{first}$ says that there are no symbols in ith stack below $(\gamma, \{<\kappa, >\zeta\}, \texttt{first})$ whose pop constraint is $<\kappa$. Hence, we can remove the obligation $<_i\kappa$ from P in the control state. If the top of stack symbol was $(\gamma, \{<\kappa, >\zeta\}, -)$, then we know that the pop constraint $<\kappa$ is still alive for ith stack . That is, there is some stack symbol below $(\gamma, \{<\kappa, >\zeta\}, -)$ of the form $(\beta, S, \texttt{first})$ such that $<\kappa \in S$. In this case, we keep P unchanged and control switches to (l', P). Processing another jth stack continues exactly as above; the set P contains $<_i \kappa, \leq_j \eta$, and so on depending on what constraints are remembered per stack. Note that the set P in (l, P) only contains constraints of the form $<_i \kappa$ or $\leq_i \kappa$ for each ith stack, since we do not remember $> \zeta$ constraints in the finite control.

[3] This last case happens when the age checked lies between ζ and κ

E.1 Reduction from dtMVPA to ECMVPA

We now give the formal construction. Let $Z^{\sim} = \bigcup_{i=1}^{n} Z_i^{\sim}$ and and let $S^{\sim} = \{\sim c \mid c \in \mathbb{N}, c \leq k, \sim \in \{<, \leq, >, \geq, =\}\}$. Given k-dtMVPA $M = (L, \Sigma, \Gamma, L^0, F, \Delta)$ with max constant κ used in return transitions of all stacks, we construct k-ECMVPA $M' = (L', \Sigma', \Gamma', L'^0, F', \Delta')$ where $L' = (L \times 2^{Z^{\sim}}) \cup (L_{\Sigma_i \times S^{\sim}} \times 2^{Z^{\sim}}) \cup (L_{\Sigma_i \times S^{\sim} \times S^{\sim}} \times 2^{Z^{\sim}})$, $\Sigma'_i = (\Sigma_c^i, \Sigma_{int}^i \cup Z_i^{\sim}, \Sigma_r^i)$ and $\Gamma'_i = \Gamma_i \times 2^{S^{\sim}} \times \{\texttt{first}, -\}$, $L^0 = \{(l^0, \emptyset) \mid l^0 \in L^0\}$, and $F = \{(l^f, \emptyset) \mid l^f \in F\}$. The transitions Δ' are defined as follows:

Call Transitions. For every $(l, a, \varphi, l', \gamma) \in \Delta_c^i$, we have the following classes of transitions in M'.

1. The first class of transitions correspond to the guessed pop constraint being $<\kappa$. In the first case, $<\kappa$ is alive, and hence there is no need to reset the clock $x_{<_i\kappa}$. In the second case, the obligation $<\kappa$ is fresh and hence it is remembered as first in the ith stack , and the clock $x_{<_i\kappa}$ is reset.

$$((l, P), a, \varphi, (l', P), (\gamma, \{<\kappa\}, -)) \in {\Delta^{i}}'_c \quad \text{if } <_i\kappa \in P$$
$$((l, P), a, \varphi, (l'_{a,<\kappa}, P'), (\gamma, \{<\kappa\}, \texttt{first})) \in {\Delta^{i}}'_c \quad \text{if } <_i\kappa \notin P \text{ and } P' = P \cup \{<_i\kappa\}$$
$$((l'_{a,<\kappa}, P'), <_i\kappa, x_a = 0, (l', P')) \in {\Delta^{i}}'_{int}$$

2. The second class of transitions correspond to the case when the guessed pop constraint is $>\kappa$. The clock $x_{>_i\kappa}$ is reset, and obligation is stored in ith stack.

$$((l, P), a, \varphi, (l'_{a,>\kappa}, P), (\gamma, \{>\kappa\}, -)) \in {\Delta^{i}}'_c \text{ and } ((l'_{a,>\kappa}, P), >_i\kappa, x_a{=}0, (l', P)) \in {\Delta^{i}}'_{int}$$

3. Finally the following transitions consider the case when the guessed pop constraint is $>\zeta$ and $<\kappa$. Depending on whether $<\kappa$ is alive or not, we have two cases. If alive, then we simply reset the clock $x_{>_i\zeta}$ and remember both the obligations in ith stack . If $<\kappa$ is fresh, then we reset both clocks $x_{>_i\zeta}$ and $x_{<_i\kappa}$ and remember both obligations in ith stack , and $<_i\kappa$ in the state.

$$((l, P), a, \varphi, (l'_{a,<\kappa,>\zeta}, P'), (\gamma, \{<\kappa, >\zeta\}, \texttt{first})) \in {\Delta^{i}}'_c \quad \text{if } <_i\kappa \notin P, P' = P \cup \{<_i\kappa, >_i\zeta\}$$
$$((l'_{a,<\kappa,>\zeta}, P'), >_i\zeta, x_a = 0, (l'_{a,<\kappa}, P')) \in {\Delta^{i}}'_{int}$$
$$((l, P), a, \varphi, (l'_{a,>\zeta}, P), (\gamma, \{<\kappa, >\zeta\}, -)) \in {\Delta^{i}}'_c \quad \text{if } <_i\kappa \in P$$

Internal Transitions. For every $(l, a, \varphi, l') \in \Delta_{int}^i$ we have the set of transitions $((l, P), a, \varphi, (l', P)) \in {\Delta^{i}}'_{int}$.

Return Transitions. For every $(l, a, I, \gamma, \varphi, l') \in \Delta_r^i$, we have following transitions in ${\Delta^{i}}'_r$.

1. $((l, P), a, (\gamma, \{<\kappa, >\zeta\}, -), \varphi \wedge x_{<_i\kappa} < \kappa \wedge x_{>_i\zeta} > \zeta, (l', P))$ if $I = (\zeta, \kappa)$.
2. $((l, P), a, (\gamma, \{<\kappa, >\zeta\}, \texttt{first}), \varphi \wedge x_{<_i\kappa} < \kappa \wedge x_{>_i\zeta} > \zeta, (l', P'))$ where $P' = P \backslash \{<_i\kappa\}$, if $I = (\zeta, \kappa)$.
3. $((l, P), a, (\gamma, \{<\kappa\}, -), \varphi \wedge x_{<_i\kappa} < \kappa, (l', P))$ if $I = [0, \kappa)$.
4. $((l, P), a, (\gamma, \{<\kappa\}, \texttt{first}), \varphi \wedge x_{<_i\kappa} < \kappa, (l', P'))$ with $P' = P \backslash \{<_i\kappa\}$ if $I = [0, \kappa)$.

5. $((l, P), a, (\gamma, \{>\zeta\}, -), \varphi \wedge x_{>_i\zeta} > \zeta, (l', P))$ if $I = (\zeta, \infty)$.

For the pop to be successful in M', the guess made at the time of the push must be correct, and indeed at the time of the pop, the age must match the constraint. The control state (l^f, P) is reached in M' on reading a word w' iff M accepts a string w and reaches l^f. Accepting locations of M' are of the form (l^f, P) for $P \subseteq Z^{\sim}$. Let $w = (a_1, t_1) \dots (a_i, t_i) \dots (a_n, t_n) \in L(M)$. If $a_i \in \Sigma_c^i$, we have in $L(M')$, a string T_i between (a_i, t_i) and (a_{i+1}, t_{i+1}), with $|T_i| \leq 2$, and T_i is a timed word of the form $(b_{1i}, t_i)(b_{2i}, t_i)$ or (b_{1i}, t_i). The time stamp t_i remains unchanged, and either b_{1i} is $<_i \kappa$ or $\leq_i \kappa$ or b_{1i} is $>_i \zeta$, or b_{1i} is $>_i \zeta$ and b_{2i} is one of $<_i \kappa$ or $\leq_i \kappa$ for some $\kappa, \zeta \leq k$. This follows from the 3 kinds of call transitions in M'.

Theorem 15. *The emptiness problem for k-dtMVPA is decidable.*

(Proof sketch.) In the construction above, it can shown by inducting on the length of words accepted that $h(L(M')) = L(M)$. Thus, $L(M') \neq \emptyset$ iff $L(M) \neq \emptyset$. If M is a k-dtMVPA, then M' is a k-ECMVPA. Since M' is a k-ECMVPA, we can apply the standard region construction of event clock automata [3] to obtain a k-MVPA, which has a decidable emptiness [13].

Determinizability of k-dtMVPA. Next, we focus on the determinizability of $k-$dtMVPA. Consider a k-dtMVPA $M = (L, \Sigma, \Gamma, L^0, F, \Delta)$ and the corresponding k-ECMVPA $M' = (L', \Sigma', \Gamma', L'^0, F', \Delta')$ as constructed in section E.1. From Theorem 9 we know that M' is determinizable. Let $Det(M')$ be the determinized automaton such that $L(Det(M')) = L(M')$. That is, $L(M) = h(L(Det(M')))$. By construction of M', we know that the new symbols introduced in Σ' are $Z^{\sim}$ ($\Sigma'_i = \Sigma_i \cup Z_i^{\sim}$ for each ith stack) and (i) no time elapse happens on reading symbols from $Z_i^{\sim}$, and (ii) no stack operations happen on reading symbols of $Z_i^{\sim}$. Consider any transition in $Det(M')$ involving the new symbols. Since $Det(M')$ is deterministic, let $(s_1, \alpha, \varphi, s_2)$ be the unique transition on $\alpha \in Z_i^{\sim}$. In the following, we eliminate these transitions on $Z_i^{\sim}$ preserving the language accepted by M and the determinism of $det(M')$. In doing so, we will construct a k-dtMVPA M'' which is deterministic, and which preserves the language of M. We now analyze various types for $\alpha \in Z_i^{\sim}$.

1. Assume that α is of the form $>_i\zeta$. Let $(s_1, \alpha, \varphi, s_2)$ be the unique transition on $\alpha \in Z_i^{\sim}$. By construction of M' (and hence $det(M')$), we know that φ is $x_a = 0$ for some $a \in \Sigma^i$. We also know that in $Det(M')$, there is a unique transition $(s_0, a, \psi, s_1, (\gamma, \alpha, -))$ preceding $(s_1, \alpha, \varphi, s_2)$. Since $(s_1, \alpha, \varphi, s_2)$ is a no time elapse transition, and does not touch any stack, we can combine the two transitions from s_0 to s_1 and s_1 to s_2 to obtain the call transition $(s_0, a, \psi, s_2, \gamma)$ for ith stack . This eliminates transition on $>_i\zeta$.
2. Assume that α is of the form $<_i\kappa$. Let $(s_1, \alpha, \varphi, s_2)$ be the unique transition on $\alpha \in Z_i^{\sim}$. We also know that φ is $x_a = 0$ for some $a \in \Sigma^i$. From M', we also know that in $Det(M')$, there is a unique transition of one of the following forms preceding $(s_1, \alpha, \varphi, s_2)$:

(a) $(s_0, a, \psi, s_1, (\gamma, \alpha, -))$, (b) $(s_0, a, \psi, s_1, (\gamma, \alpha, \texttt{first}))$, or
(c) $(s_0, >_i\zeta, \varphi, s_1)$ where it is preceded by $(s'_0, a, \psi, s_0, (\gamma, \{\alpha, >\zeta\}, X))$ for $X \in \{\texttt{first}, -\}$.

Since $(s_1, \alpha, \varphi, s_2)$ is a no time elapse transition, and does not touch the stack, we can combine two transitions from s_0 to s_1 (cases (a), (b)) and s_1 to s_2 to obtain the call transition $(s_0, a, \psi, s_2, (\gamma, \alpha, -))$ or $(s_0, a, \psi, s_2, (\gamma, \alpha, \texttt{first}))$. This eliminates the transition on $<_i\kappa$.
In case of transition (c), we first eliminate the local transition on $>_i\zeta$ obtaining $(s'_0, a, \psi, s_1, \gamma)$. This can then be combined with $(s_1, \alpha, \varphi, s_2)$ to obtain the call transitions $(s'_0, a, \psi, s_2, \gamma)$. We have eliminated local transitions on $<_i\kappa$.

Merging transitions as done here does not affect transitions on any Σ^i as they simply eliminate the newly added transitions on $\Sigma'_i \setminus \Sigma_i$. Recall that checking constraints on recorders $x_{<_i\kappa}$ and $x_{>_i\zeta}$ were required during return transitions. We now modify the pop operations in $Det(M')$ as follows: Return transitions have the following forms, and in all of these, φ is a constraint checked on the clocks of C_{Σ^i} in M during return:

- transitions $(s, a, (\gamma, \{<\kappa\}, X), \varphi \wedge x_{<_i\kappa} < \kappa, s')$ for $X \in \{-, \texttt{first}\}$ are modified to $(s, a, [0, \kappa), \gamma, \varphi, s')$;
- transitions $(s, a, (\gamma, \{<\kappa, >\zeta\}, X), \varphi \wedge x_{>_i\zeta} > \zeta \wedge x_{<_i\kappa} < \kappa, s')$ for $X \in \{-, \texttt{first}\}$ are modified to $(s, a, (\zeta, \kappa), \gamma, \varphi, s')$; and
- transition $(s, a, (\gamma, \{>\zeta\}, -), \varphi \wedge x_{>_i\zeta} > \zeta, s')$ are modified to the transitions $(s, a, (\zeta, \infty), \gamma, \varphi, s')$.

Now it is straightforward to verify that the k-dtMVPA M'' obtained from the k-ECVPA $det(M')$ is deterministic. Also, since we have only eliminated symbols of $Z^\sim$, we have $L(M'') = L(M)$ and $h(L(M'')) = L(det(M'))$. This completes the proof of determinizability of k-dtMVPA.

F Details of Theorem 11

Here, we give the details of the translations from dtMVPA to MSO and conversely. A technical point is regarding the projection operation : in general, it is known that event clock automata (hence dtMVPA) are not closed under projections. However, we need to handle projections while quantifying out variables in the MSO to dtMVPA construction. We do this by working on Quasi dtMVPA where the underlying alphabet Σ is partitioned into finitely many buckets $P_1, \ldots, P_k$ via a ranking function $\rho : \Sigma \to \mathbb{N}$. All symbols in a P_j are then "equivalent" : we assign one event recorder and one event predictor per P_i. This helps in arguing the correctness of the constructed dtMVPA from an MSO formula, while projecting out variables. In Section F.1, we show the equi-expressiveness of quasi dtMVPA and dtMVPA which allows us to complete the logical characterization.

- **Logic to automata.** We first show that the language accepted by an MSO formula φ over $\Sigma = \langle \Sigma^i_c, \Sigma^i_{int}, \Sigma^i_r \rangle_{i=1}^n$, $L(\varphi)$ is accepted by a dtMVPA. Let

$Z = (x_1, \ldots, x_m, X_1, \ldots, X_n)$ be the free variables in φ. As usual, we work on the extended alphabet $\Sigma' = \langle {\Sigma^i}'_c, {\Sigma^i}'_{int}, {\Sigma^i}'_r \rangle_{i=0}^n$ where

$${\Sigma^i}'_s = \Sigma^i_s \times (Val : Z \to \{0,1\}^{m+n}),$$

for $s \in \{c, int, r\}$. A word w' over Σ' encodes a word over Σ along with the valuation of all first order and second order variables. Thus ${\Sigma^i}'$ consists of all symbols (a, v) where $a \in \Sigma^i$ is such that $v(x) = 1$ means that x is assigned the position i of a in the word w, while $v(x) = 0$ means that x is not assigned the position of a in w. Similarly, $v(X) = 1$ means that the position i of a in w belongs to the set X. Next we use quasi-event clocks for Σ' by assigning suitable ranking function. Quasi dtMVPA are equiexpressive to dtMVPA as explained in Section F.1. We partition each ${\Sigma^i}'$ such that for a fixed $a \in \Sigma^i$, all symbols of the form $(a, d_1, \ldots, d_{m+n})$ and $d_i \in \{0, 1\}$ lie in the same partition (a determines their partition). Let $\rho' : \Sigma' \to \mathbb{N}$ be the ranking function of Σ' wrt above partitioning scheme.
Let $L(\psi)$ be the set of all words w' over Σ' such that the underlying word w over Σ satisfies formula ψ along with the valuation Val. Structurally inducting over ψ, we show that $L(\psi)$ is accepted by a dtMVPA. The cases $Q_a(x), \mu_j(x, y)$ are exactly as in [10]. We only discuss the predicate θ_j here. Consider the atomic formula $\theta_j(x) \in I$. To handle this, we build a dtMVPA that keeps pushing symbols (a, v) onto the stack j whenever $a \in \Sigma^j_c$, initializing the age to 0 on push. It keeps popping the stack on reading return symbols (a', v'), $a' \in \Sigma^j_r$, and checks whether $v'(x) = 1$ and $age(a', v') \in I$. It accepts on finding such a pop. The check $v'(x) = 1$ ensures that this is the matching return of the call made at position x. The check $age(a', v') \in I$ confirms that the age of this symbol pushed at position x is indeed in the interval I. Negations, conjunctions and disjunctions follow from the closure properties of dtMVPA.
Existential quantifications correspond to projection by excluding the chosen variable from the valuation and renaming the alphabet Σ'. Let M be a dtMVPA constructed for $\varphi(x_1, \ldots, x_n, X_1, \ldots, X_m)$ over Σ'. Consider $\exists x_i.\varphi(x_1, \ldots, x_n, X_1, \ldots, X_m)$ for some first order variable x_i. Let $Z_i = (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n, X_1, \ldots, X_m)$ by removing x_i from Z. We simply work on the alphabet $\Sigma'{\downarrow}i = \Sigma \times (Val : Z_i \to \{0,1\}^{m+n-1})$. Note that $\Sigma'{\downarrow}i$ is partitioned exactly in the same way as Σ'. For a fixed $a \in \Sigma$, all symbols $(a, d_1, \ldots, d_{m+n-1})$ for $d_i \in \{0, 1\}$ lie in the same partition. Thus, Σ' and $\Sigma'{\downarrow}i$ have exactly the same number of partitions, namely $|\Sigma|$. Thus, an event clock $x_a = x_{(a,d_1,\ldots,d_{m+n})}$ used in M can be used the same way while constructing the automaton for $\exists x_i.\varphi(x_1, \ldots, x_n, X_1, \ldots, X_m)$. The case of $\exists X_i.\varphi(x_1, \ldots, x_n, X_1, \ldots, X_m)$ is similar. Hence we obtain in all cases, a dtMVPA that accepts $L(\psi)$ when ψ is an MSO sentence.

- **Automata to logic.** Consider a dtMVPA $M = (L, \Sigma, \Gamma, L^0, F, \Delta)$. For each stack i, let C^i_γ denote a second order variable which collects all positions where γ is pushed in stack i. Similarly, let R^i_γ be a second order variable which collects all positions where γ is popped from stack i. Let X_{l_i} be a

second order variable which collects all positions where the location is l_i in a run. Let $\mathcal{C}$, $\mathcal{R}$ and $\mathcal{L}$ respectively be the set of these variables.
The MSO formula encoding runs of the dtMVPA is: $\exists\mathcal{L}\ \exists\mathcal{C}\ \exists\mathcal{R}\ \varphi(\mathcal{L},\mathcal{C},\mathcal{R})$. We assert that the starting position must belong to X_l for some $l \in L^0$. Successive positions must be connected by an appropriate transition. To complete the reduction we list these constraints.

- For call transitions $(\ell_i, a, \psi, \ell_j, \gamma) \in \Delta_c^h$, for positions x, y, assert

$$X_{\ell_i}(x) \wedge X_{\ell_j}(y) \wedge Q_a(x) \wedge C_\gamma^h(x) \wedge$$
$$\bigwedge_{b \in \Sigma^h} \Big(\big(\bigwedge_{(x_b \in I) \in \psi} \lhd_b(x) \in I \big) \wedge \big(\bigwedge_{(y_b \in I) \in \psi} \rhd_b(x) \in I \big) \Big).$$

- For return transitions $(\ell_i, a, I, \gamma, \psi, \ell_j) \in \Delta_r^h$ for positions x and y we assert that

$$X_{\ell_i}(x) \wedge X_{\ell_j}(y) \wedge Q_a(x) \wedge R_\gamma^h(x) \wedge \theta^h(x) \in I \wedge$$
$$\bigwedge_{b \in \Sigma^h} \Big(\big(\bigwedge_{(x_b \in I) \in \psi} \lhd_b(x) \in I \big) \wedge \big(\bigwedge_{(y_b \in I) \in \psi} \rhd_b(x) \in I \big) \Big).$$

- Finally, for internal transitions $(\ell_i, a, \psi, \ell_j) \in \Delta_{int}^h$ for positions x and y we assert

$$X_{\ell_i}(x) \wedge X_{\ell_j}(y) \wedge Q_a(x) \wedge \bigwedge_{b \in \Sigma^h} \Big(\big(\bigwedge_{(x_b \in I) \in \psi} \lhd_b(x) \in I \big) \wedge \big(\bigwedge_{(y_b \in I) \in \psi} \rhd_b(x) \in I \big) \Big).$$

We also assert that the last position of the word belongs to some X_l such that there is a transition (call, return, local) from l to an accepting location. The encoding of all 3 kinds of transitions is as above. Additionally, we assert that corresponding call and return positions should match, i.e.

$$\forall x \forall y\ \mu_j(x, y) \Rightarrow \bigvee_{\gamma \in \Gamma^j \setminus \perp_j} C_\gamma^j(x) \wedge R_\gamma^j(y).$$

F.1 Remaining part: Quasi dtMVPA

A quasi k-dtMVPA is a weaker form of k-dtMVPA where more than one input symbols share the same event clock. Let the finite input alphabet Σ be partitioned into finitely many classes via a ranking function $\rho : \Sigma \to \mathbb{N}$ giving rise to finitely many partitions $P_1, \ldots, P_k$ of Σ where $P_i = \{a \in \Sigma \mid \rho(a) = i\}$. The event recorder x_{P_i} records the time elapsed since the last occurrence of some action in P_i, while the event predictor y_{P_i} predicts the time required for any action of P_i to occur. Notice that since clock resets are "visible" in input timed word, the clock valuations after reading a prefix of the word is also determined by the timed word.

Definition 16 (Quasi k-dtMVPA). *A quasi dense-time visibly pushdown multistack automata over* $\Sigma = \left\{\Sigma_c^i, \Sigma_r^i, \Sigma_{int}^i\right\}_{i=1}^n$ *is a tuple* $M = (L, \Sigma, \rho, \Gamma, L^0, F, \Delta)$ *where* L *is a finite set of locations including a set* $L^0 \subseteq L$ *of initial locations,* ρ *is the ranking function,* Γ *is the stack alphabet and* $F \subseteq L$ *is a set of final locations.*

Lemma 17. *Quasi k-dtMVPA and k-dtMVPA are effectively equivalent.*

Proof. Let $A = (L_A, \Sigma_A, \rho_A, \Gamma_A, L_A^0, F_A, \Delta_A)$ be a given quasi k-dtMVPA. Let $P_A = \{p_0, p_1, \ldots, p_{n-1}\}$ be the set of partitions induced by ρ_A. If P_A contains a partition having more than one alphabet, without the loss of generality, we assume it to be partition p_0. We now describe a construction to create another quasi-event clock automaton $B = (L_B, \Sigma_B, \rho_B, \Gamma_B, L_B^0, F_B, \Delta_B)$ such that number of partitions with more than one alphabet is one less than that of A. Additionally this construction also ensures $L(A) = L(B)$. Repeated application of such construction will eventually yield quasi-event clock automaton having only singleton set partitions and which is language equivalent to A.

Let x_0 and y_0 be the event recording and predicting clocks for a partition p_0 in A. Crucial observation here is along any run of A, value of an event clock x_0 matches with an event recording clock x_a if most recent occurring alphabet of p_0 is a and a is (say) allowed to have its own event clock. Using this observation, we assign an event clock x_a in B to alphabet a and replace x_0 in the guard by x_a whenever above condition holds. Similarly, in case of event predicting clocks, if the current position along the run is i and first future position where some alphabet b in p_0 occurs is j $(j > i)$, the value of event predicting clock y_0 at position i matches with y_b. Again we assign an event clock y_b in B to b and replace y_0 by y_b in the guards when above condition is known to hold. However whether such condition will hold or not in the future is decided using nondeterministic guess. This necessitates some additional mechanism to verify the correctness of the guess. When x_0 or y_0 is $\vdash$ in A, any replacement x_a or y_b is valid.

We remember our above choices about the replacement of event recording and event predicting clocks in the locations of B along with additional information which helps us to verify correctness of our event predicting clock guess. The locations of B, $L_B = L_A \times p_0 \times 2^{\Sigma} \times p_0$ is four component tuple, where second component remembers last occurring alphabet of p_0, third component is the set of alphabets that are permitted to occur on the outgoing transitions from current location and fourth component is the alphabet of p_0 that is predicted to occur in the future. Initial locations of B are $L_B^0 = L_A^0 \times p_0 \times 2^{\Sigma} \times p_0$ and the final locations are $F_B = F_A \times p_0 \times 2^{\Sigma} \times p_0$. Partition function f_B is such that it assigns separate partitions for each alphabet in p_0 while keeping rest of partitions unchanged. Let $g = (x_0 \in I_0^x) \wedge (y_0 \in I_0^y) \bigwedge_{i=1}^{n-1} (x_i \in I_i^x) \wedge (y_i \in I_i^y)$ be the guard condition. Then the guard $g[x_0/x_a, y_0/y_b] = (x_a \in I_0^x) \wedge (y_b \in I_0^y) \bigwedge_{i=1}^{n-1} (x_i \in I_i^x) \wedge (y_i \in I_i^y)$ denotes the guard expression obtained by replacing event clock x_0 by x_a and y_0 by y_b. The transitions of B are given as $E_B = \{\langle \ell, a, \alpha, b \rangle \xrightarrow[g[x_0/x_a, y_0/y_b]]{d} \langle \ell', a', \alpha', b' \rangle\}$ such that all following conditions hold.

(c.1) $(\ell, d, g, \ell') \in E_A$ and $d \in \alpha$ and

(c.2) if $d \in p_0$ then $a' = d$, otherwise $a' = a$ and
(c.3) either $\alpha' = \{b\}, b' \in p_0$ or $\alpha' = \Sigma - p_0, b' = b$

Condition (c.1) enforces that only permissible outgoing transitions can be taken. Condition (c.2) updates the event recording component in the location whenever it sees any of the alphabet from p_0. While, Condition (c.3) covers two cases. First case is applicable when b occurs at immediate next position in the run. In this case, we must enforce next transition by setting third component to b. Run cannot proceed if any alphabet other than b occurs. This amounts to checking that our guess about b is correct. Second case covers the possibility that b does occur in the future but not immediately next. Then all alphabets in Σ other than those in p_0 are permitted to occur.

Valid homomorphisms for quasi k-dtMVPA Let $\Sigma = \left\{\Sigma_c^i, \Sigma_r^i, \Sigma_{int}^i\right\}_{i=1}^n$ and $\Pi = \left\{\Pi_c^i, \Pi_r^i, \Pi_{int}^i\right\}_{i=1}^n$ be the set of alphabets of two k-dtMVPAs $M_1 = (L_1, \Sigma, \rho_1, \Gamma_1, L_1^0, F_1, \Delta_1)$ and $M_2 = (L_2, \Pi, \rho_2, \Gamma_2, L_2^0, F_2, \Delta_2)$ respectively. A homomorphism $h : \Sigma \mapsto \Pi$ is said to *valid* iff following conditions are satisfied

- h preserves stack mapping *i.e.* $a \in \Sigma_c^i$ iff $h(a) \in \Pi_c^i$, $b \in \Sigma_r^i$ iff $h(b) \in \Pi_r^i$ and $c \in \Sigma_{int}^i$ iff $h(c) \in \Pi_{int}^i$
- h preserves event clock partition *i.e.* $\rho_1(a) = \rho_2(h(a))$ for all $a \in \Sigma$.