On Timed Scope-Bounded Context-Sensitive Languages

D. Bhave[1], S. N. Krishna[1], R. Phawade[2](✉), and A. Trivedi[3]

[1] IIT Bombay, Mumbai, India
{devendra,krishnas}@cse.iitb.ac.in
[2] IIT Dharwad, Dharwad, India
prb@iitdh.ac.in
[3] CU Boulder, Boulder, USA
ashutosh.trivedi@colorado.edu

Abstract. Perfect languages, characterized by closure under Boolean operations and decidable emptiness problem, form the basis for decidable automata-theoretic model-checking for the corresponding class of models. Regular languages and visibly pushdown languages are paradigmatic examples of perfect languages. In a previous work authors have established a timed context-sensitive perfect language characterized by multistack pushdown automata (MPA) with an explicit bound on number of rounds where in each round at most one stack is used. This paper complements the results of on bounded-round timed MPA by characterizing an alternative restriction on timed context-sensitive perfect languages called the scope-bounded multi-stack timed push-down automata where every stack symbol must be popped within a bounded number of stack contexts. The proposed model uses visibly-pushdown alphabet and event clocks to recover a bounded-scope MPA with decidable emptiness, closure under Boolean operations, and an equivalent logical characterization.

1 Introduction

The Vardi-Wolper [18] recipe for an automata-theoretic model-checking for a class of languages requires that class to be closed under Boolean operations and have decidable emptiness problem. Esparza, Ganty, and Majumdar [11] coined the term "perfect languages" for the classes of languages satisfying these properties. However, several important extensions of regular languages, such as pushdown automata and timed automata, do not satisfy these requirements. In order to lift the automata-theoretic model-checking framework for these classes of languages, appropriate restrictions have been studied including visibly pushdown automata [5] (VPA) and event-clock automata [4] (ECA). Tang and Ogawa [17] introduced a perfect class of timed context-free languages generalized both visibly pushdown automata and event-clock automata to introduce event-clock visibly pushdown automata (ECVPA). This paper proposes a perfect class of

Partially supported by grant MTR/2018/001098.

timed context-sensitive languages inspired by the scope-bounded restriction on multi-stack visibly pushdown languages introduced by La Torre, Napoli, and Parlato [13] and presents a logical characterization for the proposed subclass.

Automata Characterizing Perfect Context-Free Languages. Alur and Madhusudan [5] introduced visibly pushdown automata as a specification formalism where the call and return edges are made visible in a structure of the word. This notion is formalized by giving an explicit partition of the alphabet into three disjoint sets of call, return, and internal or local symbols and the visibly pushdown automata must push one symbol to the stack while reading a call symbol, and must pop one symbol (given the stack is non-empty) while reading a return symbol, and must not touch the stack while reading an internal symbol.

Automata Characterizing Perfect Timed Regular Languages. Alur-Dill timed automata [3] is a generalization of finite automata with continuous variables called clocks that grow with uniform rate in each control location and their valuation can be used to guard the transitions. Each transition can also reset clocks, and that allows one to constrain transitions based on the duration since a previous transition has been taken. However, the power of reseting clocks contributed towards timed automata not being closed under complementation. In order to overcome this limitation, Alur, Fix, and Henzinger [4] introduced event-clock automata where input symbol dictate the resets of the clocks. In an event-clock automata every symbol a is implicitly associated with two clocks x_a and y_a, where the recorder clock x_a records the time since the last occurrence of the symbol a, and the predictor clock y_a predicts the time of the next occurrence of symbol a. Hence, event-clock automata do not permit explicit reset of clocks and it is implicitly governed by the input timed word.

Proposed Model For Perfect Context-Sensitive Timed Languages. We study dense-time event-clock multistack visibly pushdown automata (dt-ECMVPA) that combines event-clock dynamics of event-clock automata with multiple visibly pushdown stacks. We assume a partition of the alphabet among various stacks, and partition of the alphabet of each stack into call, return, and internal symbols. Moreover, we associate recorder and predictor clocks with each symbol. Inspired by Atig et al. [1] we consider our stacks to be dense-timed, i.e. we allow stack symbols to remember the time elapsed since they were pushed to the stack.

A finite timed word over an alphabet Σ is a sequence $(a_1, t_1), \ldots, (a_n, t_n) \in (\Sigma \times \mathbb{R}_{\geq 0})^*$ 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 may use both of these formats depending on the context and for technical convenience. Let $T\Sigma^*$ denote the set of finite timed words over Σ.

We briefly discuss the concepts of rounds and scope as introduced by [13]. Consider an pushdown automata with n stacks. We say that for a stack h, a (timed) word is a stack-h context if all of its symbols belong to the alphabet of stack h. A *round* is fixed sequence of exactly n contexts one for each stack. Given a timed word, it can be partitioned into sequences of contexts of various stacks. The word is called k-round if it can be partitioned into k rounds. We say that a timed word is k-scoped if for each return symbol of a stack its matching call symbol occurs within the last k contexts of that stack. A visibly-pushdown multistack event-clock automata is *scope-bounded* if all of the accepting words are k-scoped for a fixed $k \in \mathbb{N}$.

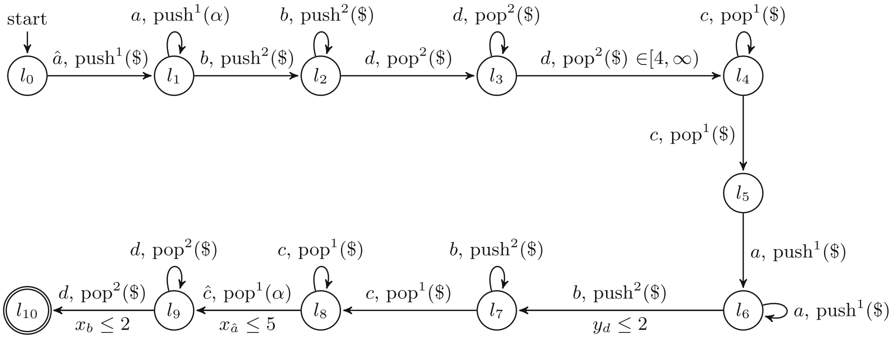

Fig. 1. A dense-time multistack visibly pushdown automata (from Example 1).

Example 1. Consider the timed language whose untimed component is of the form $L = \{\hat{a}a^x b^y d^y c^l a^l b^z c^x \hat{c} d^z \mid x, l, z \geq 1,\ y \geq 2\}$ with the critical timing restrictions among various symbols in the following manner. The time delay between the first occurrence of b and the last occurrence of d in the substring $b^y d^y$ is at least 4 time-units. The time-delay between this last occurrence of d and the next occurrence of b is at most 2 time-units. Finally the last d of the input string must appear within 2 time units of the last b, and $\hat{c}$ must occur within 5 time units of corresponding $\hat{a}$. This language is accepted by a dt-ECMVPA with two stacks shown in Fig. 1. We annotate a transition with the symbol and corresponding stack operations if any. We write pop^i or $push^i$ to emphasize pushes and pops to the i-th stack. We also use $pop^i(X) \in I$ to check if the age of the popped symbol X belongs to the interval I. In addition, we use simple constraints on predictor/recorder clock variables corresponding to the symbols. Let $a, \hat{a}$ and $c, \hat{c}$ (b and d, resp.) be call and return symbols for the first (second, resp.) stack. The Stack alphabet for the first stack is $\Gamma^1 = \{\alpha, \$\}$ and for the second stack is $\Gamma^2 = \{\$\}$. In Fig. 1 clock x_a measures the time since the occurrence of the last a, while constraints $pop(\gamma) \in I$ checks if the age of the popped symbol γ is in a given interval I. This language is 3-scoped and is accepted by a 6-round dt-ECMVPA. But if we consider the Kleene star of this language, it will be still 3-scoped and its machine can be built by fusing states l_0 and l_{10} in Fig. 1.

The formalisms of timed automata and pushdown stack have been combined before. First such attempt was timed pushdown automata [9] by Bouajjani, et al. and was proposed as a timed extension of pushdown automata which uses global clocks and timeless stack. We follow the dense-timed pushdown automata by Abdulla et al. [1]. The reachability checking of a given location from an initial one was shown to be decidable for this model. Trivedi and Wojtczak [16] studied the recursive timed automata in which clock values can be pushed onto a stack using mechanisms like pass-by-value and pass-by-reference. They studied reachability and termination problems for this model. Nested timed automata (NeTA) proposed by Li et al. [15] is a relatively recent model which, an instance of timed automata itself can be pushed on the stack along with the clocks. The clocks of pushed timed automata progress uniformly while on the stack. From the perspective of logical characterization, timed matching logic, an existential fragment of second-order logic, identified by Droste and Perevoshchikov [10] characterizes dense-timed pushdown automata.

We earlier [6] studied MSO logic for dense-timed visibly pushdown automata which form a subclass of timed context-free languages. This subclass is closed under union, intersection, complementation and determinization. The work presented in this paper extends the results from [7] for bounded-round dt-ECMVPA to the case of bounded-scope dt-ECMVPA. *We study bounded-scope dt-ECMVPA and show that they are closed under Boolean operations with decidable emptiness problem. We also present a logical characterization for these models.*

In the next section we recall the definitions of event clock and visibly pushdown automata. In Sect. 3 we define k-scope dense time multiple stack visibly push down automata with event clocks and its properties. In the following section these properties are used to decide emptiness checking and determinizability of k-scope ECMVPA with event clocks. Building upon these results, we show decidability of these properties for k-scope dt-ECMVPA with event clocks. In Sect. 5 we give a logical characterization for models introduced.

2 Preliminaries

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 [2,4,5]. We assume that the reader is comfortable with standard concepts such as context-free languages, pushdown automata, MSO logic from automata theory; and clocks, event clocks, clock constraints, and valuations from timed automata. Before we introduce our model, we revisit the definitions of event-clock automata.

The general class of TA [2] are not closed under Boolean operations. An important class of TA which is determinizable is Event-clock automata (ECA) [4], and hence closed under Boolean operations. Here the determinizability is achieved by making clock resets "visible".

To make clock resets visible we have two clocks which are associated with every action $a \in \Sigma$: x_a the "recorder" clock which records the time of the last occurrence of action a, and y_a the "predictor" clock which predicts the time of the next occurrence of action a. 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, $\nu_j^w : C \mapsto \mathbb{R}^{\vdash}_{>0}$, is defined as follows: $\nu_j^w(x_q) = t_j - t_i$ if there exists an $0 \leq i < j$ such that $a_i = q$ and $a_k \neq q$ for all $i < k < j$, otherwise $\nu_j^w(x_q) = \vdash$ (undefined). Similarly, $\nu_j^w(y_q) = t_m - t_j$ if there is $j < m$ such that $a_m = q$ and $a_l \neq q$ for all $j < l < m$, otherwise $\nu_j^w(y_q) = \vdash$. 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 $\nu_i^w \models (z \sim c)$ to denote if $\nu_j^w(z) \sim c$. For any boolean combination φ, $\nu_i^w \models \varphi$ is defined in an obvious way: if $\varphi = \varphi_1 \wedge \varphi_2$, then $\nu_i^w \models \varphi$ iff $\nu_i^w \models \varphi_1$ and $\nu_i^w \models \varphi_2$. Likewise, the other Boolean combinations are defined. Let $\Phi(C)$ define all the clock constraints defined over C.

3 Dense-Time Visibly Pushdown Multistack Automata

A visibly pushdown alphabet is a tuple $\langle \Sigma_c, \Sigma_r, \Sigma_l \rangle$ where Σ_c is *call* alphabet, Σ_r is a *return* alphabet, and Σ_l is *internal* alphabet. This section introduces scope-bounded dense-timed multistack visibly pushdown automata and give some properties about words and languages accepted by these machines.

Let $\Sigma = \langle \Sigma_c^h, \Sigma_r^h, \Sigma_l^h \rangle_{h=1}^n$ where $\Sigma_x^i \cap \Sigma_x^j = \emptyset$ whenever either $i \neq j$ or $x \neq y$, and $x, y \in \{c, r, l\}$. Let $\Sigma^h = \langle \Sigma_c^h, \Sigma_r^h, \Sigma_l^h \rangle$. Let Γ^h be the stack alphabet of the h-th stack and $\Gamma = \bigcup_{h=1}^n \Gamma^h$. For notational convenience, we assume that each symbol $a \in \Sigma^h$ has an unique recorder x_a and predictor y_a clock assigned to it. Let C_h denote the set of event clocks corresponding to stack h and $\Phi(C_h)$ denote the set of clock constraints defined over C_h. Let *cmax* be the maximum constant used in the clock constraints $\Phi(C^h)$ of all stacks. Let $\mathcal{I}$ denote the finite set of intervals $\{[0,0], (0,1), [1,1], (1,2), \ldots, [cmax, cmax], (cmax, \infty)\}$.

Definition 2. *A dense-timed visibly pushdown multistack automata (dt-ECMVPA) over* $\langle \Sigma_c^h, \Sigma_r^h, \Sigma_l^h \rangle_{h=1}^n$ *is a tuple* $(L, \Sigma, \Gamma, L^0, F, \Delta = (\Delta_c^h \cup \Delta_r^h \cup \Delta_l^h)_{h=1}^n)$ *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 stack* h *and has special end-of-stack symbol* $\perp_h$*,*
- $\Delta_c^h \subseteq (L \times \Sigma_c^h \times \Phi(C_h) \times L \times (\Gamma^h \backslash \{\perp_h\}))$ *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,*
- $\Delta_l^h \subseteq (L \times \Sigma_l^h \times \Phi(C_h) \times L)$ *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 dt-ECMVPA 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 dt-ECMVPA, function 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 the string representing stack contents below γ^h, while $age(\gamma^h\sigma^h)$ is a sequence of real numbers denoting the ages (the time elapsed since a stack symbol was pushed on to the stack) of all the stack symbols in $\gamma^h\sigma^h$. We follow the assumption that $age(\bot^h) = \langle \vdash \rangle$ (undefined). If for some string $\sigma^h \in (\Gamma^h)^*$ we have $age(\sigma^h) = \langle t_1, t_2, \ldots, t_g \rangle$ and for $\tau \in \mathbb{R}_{\geq 0}$ then 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 stack symbol γ^h we write $\gamma^h{::}\sigma^h$ for $\langle \gamma^h, \gamma_1^h, \ldots, \gamma_g^h \rangle$.

A run of a dt-ECMVPA on a timed word $w = (a_0, t_0), \ldots, (a_e, t_e)$ 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 $\ell_i \in L$, $\ell_0 \in L^0$, $\sigma_i^h \in (\Gamma^h)^*\bot^h$, 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 they 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 ages of remaining stack symbols are updated i.e., $age(\sigma_{i+1}^h) = age(\kappa) + (t_i - t_{i-1})$. 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 $a_i \in \Sigma_l^h$, then there is $(\ell_i, a_i, \varphi, \ell_{i+1}) \in \Delta_l^h$ such that $\nu_i^w \vDash \varphi$. In this case all stacks remain unchanged i.e. $\sigma_{i+1}^h{=}\sigma_i^h$, but their contents age by $t_i - t_{i-1}$ i.e. $age(\sigma_{i+1}^h){=}age(\sigma_i^h) + (t_i - t_{i-1})$ for all $1 \leq h \leq n$.

A run ρ of a dt-ECMVPA 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 dt-ECMVPA M, is the set of all timed words w accepted by M and is called dt-ECMVPL.

A dt-ECMVPA $M = (L, \Sigma, \Gamma, L^0, F, \Delta)$ 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 any two moves $(\ell, a, \phi_1, \ell', \gamma_1)$ and $(\ell, a, \phi_2, \ell'', \gamma_2)$ of Δ_c^h, condition $\phi_1 \wedge \phi_2$ is unsatisfiable; for any two moves $(\ell, a, I_1, \gamma, \phi_1, \ell')$ and $(\ell, a, I_2, \gamma, \phi_2, \ell'')$ in Δ_r^h, either $\phi_1 \wedge \phi_2$ is unsatisfiable or $I_1 \cap I_2 = \emptyset$; and for any two moves (ℓ, a, ϕ_1, ℓ') and (ℓ, a, ϕ_2, ℓ') in Δ_l^h, condition $\phi_1 \wedge \phi_2$ is unsatisfiable.

An Event clock multi stack visibly push down automata (ECMVPA) is a dt-ECMVPA where the stacks are untimed i.e., a dt-ECMVPA $(L, \Sigma, \Gamma, L^0, F, \Delta)$, with $I = [0, +\infty]$ for every $(\ell, a, I, \gamma, \phi, \ell') \in \Delta_r^h$, is an ECMVPA.

A dtECVPA is a dt-ECMVPA restricted to single stack.

We now define a *matching relation* $\sim_h$ on the positions of input timed word w which identifies matching call and return positions for each stack h. Note that this is possible because of the visibility of the input symbols.

Definition 3 (Matching relation). *Consider a timed word w over Σ. Let $\mathcal{P}_c^h$ (resp. $\mathcal{P}_r^h$) denote the set of positions in w where a symbol from Σ_c^h i.e. a call symbol (resp. Σ_r^h i.e. a return symbol) occurs. Position i (resp. j) is called* call position (resp. return position). *For each stack h the timed word w, defines a* matching relation $\sim_h \subseteq \mathcal{P}_c^h \times \mathcal{P}_r^h$ *satisfying the following conditions:*

1. *for all positions i, j with $i \sim_h j$ we have $i < j$,*
2. *for any call position i of $\mathcal{P}_c^h$ and any return position j of $\mathcal{P}_r^h$ with $i < j$, there exists l with $i \leq l \leq j$ for which either $i \sim_h l$ or $l \sim_h j$,*
3. *for each call position $i \in \mathcal{P}_c^h$ (resp. $i \in \mathcal{P}_r^h$) there is at most one return position $j \in \mathcal{P}_r^h$ (resp. $j \in \mathcal{P}_c^h$) with $i \sim_h j$ (resp. $j \sim_h i$).*

For $i \sim_h j$, position i (resp. j) is called *matching call (resp. matching return).*

This definition of matching relation extends that defined by La Torre, et al. [14] to timed words. As matching relation is completely determined by stacks and timestamps of the input word does not play any role, we claim that above definition uniquely identifies matching relation for a given input word w using uniqueness proof from [14].

Fix a k from $\mathbb{N}$. A *stack-h context* is a word in $\Sigma^h(\Sigma^h)^*$. Given a word w and a stack h, the word w has k maximal h-contexts if $w \in (\Sigma^h)^*((\bigcup_{h \neq h'} \Sigma^{h'})^*(\Sigma^h)^*)^{k-1}$. A timed word over Σ is **k-scoped** if for each matching call of stack h, its corresponding return occurs within at most k maximal stack-h contexts.

Let $Scope(\Sigma, k)$ denote the set of all k-scope timed words over Σ. For any fixed k, a k-scope dt-ECMVPA over Σ is a tuple $A = (k, M)$ where $M = (L, \Sigma, \Gamma, L^0, F, \Delta)$ is a dt-ECMVPA over Σ. The language accepted by A is $L(A) = L(M) \cap Scope(\Sigma, k)$ and is called k-scope dense-timed multistack visibly pushdown language (k-scoped-dt-ECMVPL). We define k-scoped-ECMVPL in a similar fashion. We now recall some key definitions from La Torre [13,14] which help us extend the notion of scoped words from untimed to timed words.

Definition 4 (k-scoped splitting [13,14]). *A* cut *of w is $w_1{:}w_2$ where $w = w_1w_2$. The cutting of w is marked by ":". A* cut is *h-consistent with matching relation $\sim_h$ if no call occuring in w_1 matches with a return in w_2 in $\sim_h$. A* splitting *of w is a set of cuts $w_1 \dots w_i : w_{i+1} \dots w_m$ such that $w = w_1 \dots w_i w_{i+1} \dots w_m$ for each i in $\{1, \dots, m-1\}$. An* h-consistent splitting *of w is the one in which each specified cut is h-consistent. A* context-splitting *of word w is a splitting $w_1 : w_2 : \dots : w_m$ such that each w_i is an h-context for some stack h and $i \in \{1, \dots, m\}$. A* canonical context-splitting *of word is a context-splitting of w in which no two consecutive contexts belong to the same stack.*

Given a context-splitting of timed word w, we obtain its h-projection by removing all non stack-h contexts. Observe that an h-projection is a context-splitting. An ordered tuple of m h-contexts is k-bounded if there there exists a h-consistent splitting of this tuple, where each component of the cut in the splitting is a concatenation of at most k consecutive h-contexts of given tuple. A k-scoped splitting of word w is the canonical splitting of w equipped with additional cuts for each stack h such that, if we take h-projection of w with these cuts it is k-bounded. The main purpose for introducing all the above definitions is to come up with a scheme which will permit us to split any arbitrary length input timed word into k-scoped words. Using [13,14] for untimed words we get the following Lemma.

Lemma 5. *A timed word w is k-scoped iff there is a k-scoped splitting of w.*

Next we describe the notion of switching vectors for timed words [6], which are used in determinization of k-scope dt-ECMVPA.

3.1 Switching Vectors

Let A be k-scoped dt-ECMVPA over Σ and let w be a timed word accepted by A. Our aim is to simulate A on w by n different dtECVPAs, A^h for each stack-h inputs. We insert a special symbol # at the end of each maximal context, to obtain word w' over $\Sigma \cup \{\#, \#'\}$. We also have recorder clocks $x_\#$ and predictor clocks $y_\#$ for symbol #. For h-th stack, let dtECVPA A^h be the restricted version of A over alphabet $\Sigma \cup \{\#, \#'\}$ which simulates A on input symbols from Σ^h. Then, it is clear that at the symbol before #, stack h may be touched by dt-ECMVPA A and at the first symbol after #, stack h may be touched again. But it may be the case that at positions where # occurs stack h may not be empty i.e., cut defined position of # may be not be h-consistent.

To capture the behaviour of A^h over timed word w we have a notion of switching vector. Let m be the number of maximal h-contexts in word w and w^h be the h-projection of w i.e., $w^h = u_1^h \dots u_m^h$. In particular, m could be more than k. A switching vector $\mathbb{V}^h$ of A for word w is an element of $(L, \mathcal{I}, L)^m$, where $\mathbb{V}^h[l] = (q, I_l, q')$ if in the run of A over w^h we have $q \xrightarrow{u_l^h} q'$. Let $w'^h = u_1^h \# u_2^h \# \dots u_m^h \#$, where $u_i^h = (a_{i1}^h, t_{i1}^h), (a_{i2}^h, t_{i2}^h) \dots (a_{i,s_i}^h, t_{i,s_i}^h)$ is a stack-h context, where $s_i = |u_i^h|$. Now we assign time stamps of the last letter read in the previous contexts to the current symbol # to get the word $\kappa^h = u_1^h(\#, t_{1,s_1}^h) u_2^h(\#, t_{2,s_2}^h) \dots u_m^h(\#, t_{m,s_m}^h)$.

We take the word w'^h and looking at this word we construct another word $\bar{w}^h$ by inserting symbols #′ at places where the stack is empty after popping some symbol, and if #′ is immediately followed by # then we drop # symbol. We do this in a very canonical way as follows: In this word w'^h look at the first call position c_1 and its corresponding return position r_1. Then we insert #′ after position r_1 in w^h. Now we look for next call position c_2 and its corresponding return position r_2 and insert symbol #′ after r_2. We repeat this construction for all call and its corresponding return positions in w'^h to get a timed word $\bar{w}^h$ over $\Sigma \cup \{\#, \#'\}$. Let $\bar{w}^h = \bar{u}_1^h \widehat{\#} \bar{u}_2^h \widehat{\#} \dots \widehat{\#} \bar{u}_z^h$, where $\widehat{\#}$ is either # or #′, and $\bar{u}_i^h = (\bar{a}_{i1}^h, \bar{t}_{i1}^h), (\bar{a}_{i2}^h, \bar{t}_{i2}^h) \dots (\bar{a}_{i,s_i}^h, \bar{t}_{i,s_i}^h)$, is a timed word.

The restriction of A which reads $\bar{w}^h$ is denoted by A_k^h. Assign timestamps of the last letter read in the previous contexts to the current symbol $\hat{\#}$ to get the word $\bar{\kappa}^h = \bar{u}_1^h(\hat{\#}, \bar{t}_{1,s_1}^h) \bar{u}_2^h(\hat{\#}, \bar{t}_{2,s_2}^h) \dots \bar{u}_z^h(\hat{\#}, \bar{t}_{z,s_z}^h)$, where $s_i = |\bar{u}_i^h|$ for i in

$\{1,\ldots,z\}$. A stack-h *switching vector* $\bar{\mathbb{V}}^h$ is a z-tuple of the form $(L,\mathcal{I},L)^z$, where $z>0$ and for every $j\le z$ if $\bar{\mathbb{V}}^h[j]=(q_j,I_j,q'_j)$ then there is a run of A^h from location q_j to q'_j.

By definition of k-scoped word we are guaranteed to find maximum k number of # symbols from c_j to r_j. And we also know that stack-h is empty whenever we encounter $\#'$ in the word. In other words, if we look at the switching vector $\bar{\mathbb{V}}^h$ of A reading $\bar{w}^h$, it can be seen as a product of switching vectors of A each having a length less than k. Therefore, $\bar{\mathbb{V}}^h=\Pi_{i=1}^r V_i^h$ where $r\le z$ and $V_i^h=(L\times\mathcal{I}\times L)^{\le k}$. When we look at a timed word and refer to the switching vector corresponding to it, we view it as tuples of switching pairs, but when we look at the switching vectors as a part of state of A_k^h then we see at a product of switching vectors of length less than k.

A *correct sequence of context switches* for A_k^h wrt $\bar{\kappa}^h$ is a sequence of pairs $\bar{\mathbb{V}}^h=P_1^hP_2^h\ldots P_z^h$, where $P_i^h=(\ell_i^h,I_i^h,\ell_i'^h)$, $2\le h\le n$, $P_1^h=(\ell_1^h,\nu_1^h,\ell_1'^h)$ and $I_i^h\in\mathcal{I}$ such that

1. Starting in ℓ_1^h, with the h-th stack containing $\perp^h$, and an initial valuation ν_1^h of all recorders and predictors of Σ^h, the dt-ECMVPA A processes u_1^h and reaches some $\ell_1'^h$ with stack content σ_2^h and clock valuation $\nu_1'^h$. The processing of u_2^h by A then starts at location ℓ_2^h, and a time $t\in I_2^h$ has elapsed between the processing of u_1^h and u_2^h. Thus, A starts processing u_2^h in (ℓ_2^h,ν_2^h) where ν_2^h is the valuation of all recorders and predictors updated from $\nu_1'^h$ with respect to t. The stack content remains same as σ_2^h when the processing of u_2^h begins.
2. In general, starting in (ℓ_i^h,ν_i^h), $i>1$ with the h-th stack containing σ_i^h, and ν_i^h obtained from ν_{i-1}^h by updating all recorders and predictors based on the time interval I_i^h that records the time elapse between processing u_{i-1}^h and u_i^h, A processes u_i^h and reaches $(\ell_i'^h,\nu_i'^h)$ with stack content σ_{i+1}^h. The processing of u_{i+1}^h starts after time $t\in I_{i+1}^h$ has elapsed since processing u_i^h in a location ℓ_{i+1}^h, and stack content being σ_i^h.

These switching vectors were used in to get the determinizability of k-round dt-ECMVPA [6] In a k-round dt-ECMVPA, we know that there at most k-contexts of stack-h and hence the length of switching vector (whichever it is) is at most k for any given word w. See for example the MVPA corresponding to Kleene star of language given in the Example 1. In k-scope MVPA for a given w, we do not know beforehand what is the length of switching vector. So we employ not just one switching vector but many one after another for given word w, and we maintain that length of each switching vector is at most k. This is possible because of the definition of k-scope dt-ECMVPA and Lemma 5.

Lemma 6 *(Switching Lemma for A_k^h). Let $A=(k,L,\Sigma,\Gamma,L^0,F,\Delta)$ be a k-scope-dt-ECMVPA. Let w be a timed word with m maximal h-contexts and accepted by A . Then we can construct a dtECVPA A_k^h over $\Sigma^h\cup\{\#,\#'\}$ such that A_k^h has a run over $\bar{w}^h$ witnessed by a switching sequence $\bar{\mathbb{V}}^h=\Pi_{i=1}^r\bar{\mathbb{V}}_i^h$ where $r\le z$ and $\bar{\mathbb{V}}_i^h=(L\times\mathcal{I}\times L)^{\le k}$ which ends in the last component $\bar{\mathbb{V}}_r^h$ of $\bar{\mathbb{V}}^h$ iff there exists a k-scoped switching sequence $\bar{\mathbb{V}}'^h$ of switching vectors of A such that for any v' of $\bar{\mathbb{V}}'^h$ there exist v_i and v_j in $\bar{\mathbb{V}}'$ with $i\le j$ and $v'[1]=v_i[1]$ and $v'[|v'|]=v_j[|v_j|]$.*

Proof. We construct a dtECVPA $A_h^k = (L^h, \Sigma \cup \{\#, \#'\}, \Gamma^h, L^0, F^h = F, \Delta^h)$ where, $L^h \subseteq (L \times \mathcal{I} \times L)^{\leq k} \times \Sigma \cup \{\#, \#'\}$ and Δ^h are given below.

1. For a in Σ:
$(P_1^h, \ldots, P_i^h = (q, I_i^h, q'), b) \xrightarrow{a,\phi} (P_1^h, \ldots, P_i'^h = (q, I_i'^h, q''), a)$, when $q' \xrightarrow{a,\phi} q''$ is in Δ, and $b \in \Sigma$.
2. For a in Σ:
$(P_1^h, \ldots, P_i^h = (q, I_i^h, q'), \#) \xrightarrow{a,\phi \wedge x_\# = 0} (P_1^h, \ldots, P_i'^h = (q, I_i'^h, q''), a)$, when $q' \xrightarrow{a,\phi} q''$ is in Δ, and $b \in \Sigma$.
3. For a in Σ:
$(P_1^h, \ldots, P_i^h = (q, I_i^h, q'), \#') \xrightarrow{a,\phi \wedge x_{\#'} = 0} (P_1^h, \ldots, P_i'^h = (q, I_i'^h, q''), a)$, when $q' \xrightarrow{a,\phi} q''$ is in Δ, and $b \in \Sigma$.
4. For $a = \#$,
$(P_1^h, \ldots, P_i^h = (q, I_i^h, q'), b) \xrightarrow{a,\phi \wedge x_b \in I_{i+1}'^h} (P_1^h, \ldots, P_{i+1}'^h = (q'', I_{i+1}'^h, q''), \#)$, when $q' \xrightarrow{a,\phi} q''$ is in Δ.
5. For $a = \#'$,
$(P_1^h, \ldots, P_i^h = (q, I_i^h, q'), a) \xrightarrow{a,\phi, x_{\#'} \in \hat{I}_1^h} (\hat{P}_1^h = (q', \hat{I}_1^h, q'), \#')$, when $q' \xrightarrow{a,\phi} q''$ is in Δ.

Given a timed word w accepted by A, when A is restricted to A^h then it is running on w'^h, the projection of w on Σ^h, interspersed with # separating the maximal h-contexts in original word w. Let $v_1, v_2, \ldots, v_m$ be the sequence of switching vectors witnessed by A^h while reading w'^h.

Now when w'^h is fed to the constructed machine A_h^k, it is interspersed with new symbols $\#'$ whenever the stack is empty just after a return symbol is read. Now $\bar{w}^h$ thus constructed is again a collection of z stack-h contexts which possibly are more in number than in w'^h. And each newly created context is either equal to some context of w'^h or is embedded in exactly one context of w'^h. These give rise to sequence of switching vectors $v_1', v_2', \ldots, v_z'$, where $m \leq z$. That explains the embedding of switching vectors witnessed by A_h^k, while reading $\bar{w}^h$, into switching vectors of A, while reading w^h. □

Let w be in $L(A)$. Then as described above we can have a sequence of switching vectors $\bar{\mathbb{V}}_h$ for stack-h machine A_k^h. Let d^h be the number of h-contexts in the k-scoped splitting of w i.e., the number of h-contexts in $\bar{w}^h$. Then we have those many tuples in the sequence of switching vectors $\bar{\mathbb{V}}^h$. Therefore, $\bar{\mathbb{V}}^h = \Pi_{y \in \{1,\ldots,d_h\}} \langle l_y^h, I_y^h, l_y'^h \rangle$.

We define the relation between elements of $\bar{\mathbb{V}}^h$ across all such sequences. While reading the word w, for all h and h' in $\{1, \ldots, n\}$ and for some y in $\{1, \ldots, d_h\}$ and some y' in $\{1, \ldots, d_{h'}\}$ we define a relation *follows*$(h, y) = (h'y')$ if y-th h-context is followed by y'-th h'-context.

A collection of correct sequences of context switches given via switching vectors $(\bar{\mathbb{V}}^1, \ldots, \bar{\mathbb{V}}^n)$ is called **globally correct** if we can stitch together runs of all A_k^hs on $\bar{w}^h$ using these switching vectors to get a run of A on word w.

In the reverse direction, if for a given k-scoped word w over Σ which is in $L(A)$ then we have, collection of globally correct switching vectors $(\bar{\mathbb{V}}^1, \ldots, \bar{\mathbb{V}}^n)$.

The detailed proof of the following lemma is given in [8].

Lemma 7 (Stitching Lemma). *Let $A = (k, L, \Sigma, \Gamma, L^0, F, \Delta)$ be a k-scope dt-ECMVPA. Let w be a k-scoped word over Σ. Then $w \in L(A)$ iff there exist a collection of globally correct sequences of switching vectors for word w.*

4 Scope-Bounded ECMVPA and dt-ECMVPA

Fix a $k \in \mathbb{N}$. Decidability of emptiness checking of k-round ECMVPA has been shown in [7]. This proof works for any general ECMVPA as the notion k-round has not been used and we use the same for emptiness checking of k-scope ECMVPA. Detailed proofs for the theorems in this section are given in [8].

Theorem 8. *Emptiness checking for k-scope ECMVPA is decidable.*

Using Lemmas 6 and 7 we get the following theorem.

Theorem 9. *The class of k-scope ECMVPA are determinizable.*

To prove the decidability of emptiness checking for k-scope dt-ECMVPA, we first do untime its stack to get k-scope ECMVPA for which emptiness is shown to be decidable in Theorem 8.

Theorem 10. *The emptiness checking for k-scope dt-ECMVPA is decidable.*

Theorem 11. *The k-scope dt-ECMVPA are determinizable.*

Proof (sketch). From a k-scope dt-ECMVPA using the stack untiming construction we get a k-scope ECMVPA, which is determinized using Theorem 9. We convert this back to get deterministic k-scope dt-ECMVPA.

It is easy to show that k-scoped ECMVPAs and k-scoped dt-ECMVPAs are closed under union and intersection; using Theorems 9 and 11 we get closure under complementation.

Theorem 12. *The classes of k-scoped ECMVPLs and k-scoped dt-ECMVPLs are closed under Boolean operations.*

5 Logical Characterization of k-dt-ECMVPA

Let $w = (a_1, t_1), \ldots, (a_m, t_m)$ be a timed word over alphabet $\Sigma = \langle \Sigma_c^i, \Sigma_l^i, \Sigma_r^i \rangle_{i=1}^n$ as a *word structure* over the universe $U = \{1, 2, \ldots, |w|\}$ of positions in w. We borrow definitions of predicates $Q_a(i), \lhd_a(i), \rhd_a(i)$ from [6]. Following [12], 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 introduce the predicate $\theta_j(i) \in I$ which evaluates to true on the word structure iff

$w[i] = (a, t_i)$ with $a \in \Sigma_r^j$ and $w[i] \in \Sigma_r^j$, 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 time elapsed 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 onto the stack during the call at position k. Since position i is the matching return, this symbol is popped at 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 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 $Scope(\Sigma, k)$, for some k, can be captured by an MSO formula $Scope_k(\psi) = \bigwedge_{1 \le j \le n} Scope_k(\psi)^j$, where n is number of stacks, where

$$Scope_k(\psi)^j \overset{\text{def}}{=} \forall y Q_a(y) \wedge a \in \Sigma_j^r \Rightarrow (\exists x \mu_j(x, y) \wedge (\psi^j_{kcnxt} \wedge \psi^j_{matcnxt} \wedge \psi_{noxcnxt}))$$

where ψ^j_{kcnxt}, $\psi^j_{matcnxt}$, and $\psi_{noxcnxt}$ are defined as

$$\psi^j_{kcnxt} = \exists x_{1\ldots k}(x_1 \le \ldots \le x_k \le y \bigwedge_{1 \le q \le k} (Q_a(x_q) \wedge a \in \Sigma_j \wedge (Q_b(x_q - 1) \Rightarrow b \notin \Sigma_j)),$$

$$\psi^j_{matcnxt} = \bigvee_{1 \le q \le k} \forall x_i (x_q \le x_i \le x (Q_c(x_i) \Rightarrow c \in \Sigma_j)), \text{ and}$$

$$\psi_{noxcnxt} = \exists x_l (x_1 \le x_l \le y)(Q_a(l) \wedge a \in \Sigma_j \wedge Q_b(x_l - 1) \wedge b \in \Sigma_j) \Rightarrow 1 \le l \le k.$$

Formulas $\psi_{noextracnxt}$ and ψ_{kcnxt} say that there are at most k contexts of j-th stack, while formula $\psi_{matcnxt}$ says where matching call position x of return position y is found. Conjuncting the formula obtained from a dt-ECMVPA M with $Scope(\psi)$ accepts only those words which lie in $L(M) \cap Scope(\Sigma, k)$. Likewise, if one considers any MSO formula $\zeta = \varphi \wedge Scope(\psi)$, it can be shown that the dt-ECMVPA M constructed for ζ will be a k-dt-ECMVPA. Hence we have the following MSO characterization.

Theorem 13. *A language L over Σ is accepted by an k-scope dt-ECMVPA iff there is a MSO sentence φ over Σ such that $L(\varphi) \cap Scope(\Sigma, k) = L$.*

The two directions, dt-ECMVPA to MSO, as well as MSO to dt-ECMVPA can be handled using standard techniques, and can be found in [8].

References

1. Abdulla, P.A., Atig, M.F., Stenman, J.: Dense-timed pushdown automata. In: Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, pp. 35–44, 25–28 June 2012. IEEE Computer Society. https://doi.org/10.1109/LICS.2012.15

2. Alur, R., Dill, D.: A theory of timed automata. Theoret. Comput. Sci. **126**, 183–235 (1994)
3. Alur, R., Dill, D.: Automata for modeling real-time systems. In: Paterson, M.S. (ed.) ICALP 1990. LNCS, vol. 443, pp. 322–335. Springer, Heidelberg (1990). https://doi.org/10.1007/BFb0032042
4. Alur, R., Fix, L., Henzinger, T.A.: Event-clock automata: a determinizable class of timed automata. Theoret. Comput. Sci. **211**(1–2), 253–273 (1999)
5. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: Babai, L. (ed.) Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, pp. 202–211, 13–16 June 2004. ACM. https://doi.org/10.1145/1007352.1007390
6. Bhave, D., Dave, V., Krishna, S.N., Phawade, R., Trivedi, A.: A logical characterization for dense-time visibly pushdown automata. In: Dediu, A.-H., Janoušek, J., Martín-Vide, C., Truthe, B. (eds.) LATA 2016. LNCS, vol. 9618, pp. 89–101. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-30000-9_7
7. Bhave, D., Dave, V., Krishna, S.N., Phawade, R., Trivedi, A.: A perfect class of context-sensitive timed languages. In: Brlek, S., Reutenauer, C. (eds.) DLT 2016. LNCS, vol. 9840, pp. 38–50. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53132-7_4
8. Bhave, D., Krishna, S.N., Phawade, R., Trivedi, A.: On timed scope-bounded context-sensitive languages. Technical report, IIT Bombay (2019). www.cse.iitb.ac.in/internal/techreports/reports/TR-CSE-2019-77.pdf
9. Bouajjani, A., Echahed, R., Habermehl, P.: On the verification problem of nonregular properties for nonregular processes. In: Proceedings, 10th Annual IEEE Symposium on Logic in Computer Science, San Diego, California, USA, pp. 123–133, 26–29 June 1995. IEEE Computer Society. https://doi.org/10.1109/LICS.1995.523250
10. Droste, M., Perevoshchikov, V.: A logical characterization of timed pushdown languages. In: Beklemishev, L.D., Musatov, D.V. (eds.) CSR 2015. LNCS, vol. 9139, pp. 189–203. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-20297-6_13
11. Esparza, J., Ganty, P., Majumdar, R.: A perfect model for bounded verification. In: Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, pp. 285–294, 25–28 June 2012. https://doi.org/10.1109/LICS.2012.39
12. La Torre, S., Madhusudan, P., Parlato, G.: A robust class of context-sensitive languages. In: LICS, pp. 161–170 (2007)
13. La Torre, S., Napoli, M., Parlato, G.: Scope-bounded pushdown languages. In: Shur, A.M., Volkov, M.V. (eds.) DLT 2014. LNCS, vol. 8633, pp. 116–128. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-09698-8_11
14. La Torre, S., Napoli, M., Parlato, G.: Scope-bounded pushdown languages. Int. J. Found. Comput. Sci. **27**(2), 215–234 (2016)
15. Li, G., Cai, X., Ogawa, M., Yuen, S.: Nested timed automata. In: Braberman, V., Fribourg, L. (eds.) FORMATS 2013. LNCS, vol. 8053, pp. 168–182. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40229-6_12
16. Trivedi, A., Wojtczak, D.: Recursive timed automata. In: Bouajjani, A., Chin, W.-N. (eds.) ATVA 2010. LNCS, vol. 6252, pp. 306–324. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15643-4_23

17. Van Tang, N., Ogawa, M.: Event-clock visibly pushdown automata. In: Nielsen, M., Kučera, A., Miltersen, P.B., Palamidessi, C., Tůma, P., Valencia, F. (eds.) SOFSEM 2009. LNCS, vol. 5404, pp. 558–569. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-540-95891-8_50
18. Vardi, M., Wolper, P.: Reasoning about infinite computations. Inform. Comput. **115**(1), 1–37 (1994)