### On Timed Scope-bounded Context-sensitive Languages

D. Bhave<sup>1</sup>, S. N. Krishna<sup>1</sup>, R. Phawade<sup>2</sup>, and A. Trivedi<sup>3</sup>

 $^1$  IIT Bombay and  $^2$  IIT Dharwad  $^3$  CU Boulder {devendra,krishnas}@cse.iitb.ac.in,prb@iitdh.ac.in, ashutosh.trivedi@colorado.edu

**Abstract.** In (DLT 2016) we studied timed context sensitive languages characterized by multiple stack push down automata (MPA), with an explicit bound on number of stages where in each stage at most one stack is used (k-round MPA).

In this paper, we continue our work on timed MPA and study a subclass in which a symbol corresponding to a stack being pushed in it must be popped within fixed number of contexts of that stack —scope-bounded push-down automata with multiple stacks (k-scope MPA). We use Visibly Push-down Alphabet and Event Clocks to show that timed k-scope MPA have decidable reachability problem; are closed under Boolean operations; and have an equivalent logical characterization.

#### 1 Introduction

The Vardi-Wolper [24] 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 [12] 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 [6] (VPA) and event-clock automata [5] (ECA). Tang and Ogawa [23] 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).

In this paper we study a previously unexplored class of timed context-sensitive languages inspired by the scope-bounded restriction on multi-stack visibly pushdown languages introduced by La Torre, Napoli, and Parlato [17], and show that it is closed under Boolean operations and has decidable emptiness problem. Moreover, we also present a logical characterization for the proposed subclass. Visible Stack Operations. Alur and Madhusudan [6] 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.

Visible Clock Resets. Alur-Dill timed automata [4] 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 [5] 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.

Visible Stack Operations and Clock Resets in Multistack Setting. 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.

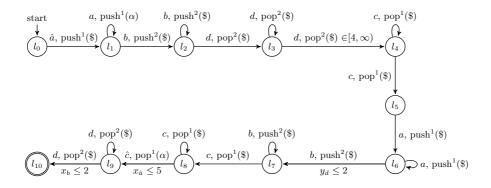


Fig. 1. Dense-time Multistack Visibly Pushdown Automata used in Example 1

A finite timed word over an alphabet  $\Sigma$  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  $\Sigma$ .

We briefly discuss the concepts of rounds and scope as introduced by [17]. 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  $k \in \mathbb{N}$ .

To introduce some of the key ideas of our model, let us consider the following example.

Example 1. Consider the timed language whose untimed component is of the form  $L = \{\hat{a}a^xb^yd^yc^la^lb^zc^x\hat{c}d^z \mid x,l,z\geq 1,\ y\geq 2\}$  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 â. This language is accepted by a dt-ECMVPA with two stacks shown in Figure 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 Figure 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  $\gamma$  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  $l_{10}$  of the MVPA in Figure 1.

Related Work. 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 intial one was shown to be decidable for this model. Trivedi and Wojtczak [21] 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 [19] 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 [11] characterizes dense-timed pushdown automata. We earlier [7] 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 [8] for bounded-round dt-ECMVPA to the case of bounded-scope dt-ECMVPA.

Contributions. We study bounded-scope dt-ECMVPA and show that they are closed under Boolean operations and the emptiness problem for these models is decidable. We also present a logical characterization for these models.

Organization of the paper: In the next section we recall the definitions of event clock and visibly pushdown automata. In Section 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 Section 6 we give a logical characterization for models introduced.

#### 2 Preliminaries

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 [3,5,6]. 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.

#### 2.1 Event-Clock Automata

The general class of TA [3] are not closed under Boolean operations. An important class of TA which is determinizable is Event-clock automata (ECA) [5], 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.

Hence, event-clock automata do not permit explicit reset of clocks and it is implicitly governed by the input timed word which makes them determinizable and closed under all Boolean operations.

We write C for the set of all event clocks and we use  $\mathbb{R}_{>0}^{\vdash}$  for the set  $\mathbb{R}_{>0} \cup \{\vdash\}$ . Formally, the clock valuation after reading j-th prefix of the input timed word w,  $\nu_j^w : C \mapsto \mathbb{R}_{>0}^{\vdash}$ , is defined as follows:  $\nu_j^w(x_q) = t_j - t_i$  if there exists an  $0 \le i < j$  such that  $a_i = q$  and  $a_k \ne q$  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  $a_l \ne q$  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  $\varphi$ ,  $\nu_i^w \models \varphi$  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.

**Definition 2.** An event clock automaton is a tuple  $A = (L, \Sigma, L^0, F, E)$  where L is a set of finite locations,  $\Sigma$  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  $\ell, \ell'$  are locations,  $a \in \Sigma$ , and  $\varphi$  in  $\Phi(C)$ .

The class of languages accepted by ECA have Boolean closure and decidable emptiness [5].

#### 2.2 Visibly Pushdown Automata

The class of push down automata are not determinable and also not closed under Boolean operations [15]. The determinizability is achieved by making input alphabet "visible" that is for a given input letter only one kind of stack operations is allowed giving an important subclass of Visibly

pushdown automata [6] which operate over words that dictate the stack operations. This notion is formalized by giving an explicit partition of the alphabet. 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 3.** A visibly pushdown alphabet is a tuple  $\langle \Sigma_c, \Sigma_r, \Sigma_l \rangle$  where  $\Sigma_c$  is call alphabet,  $\Sigma_r$  is a return alphabet, and  $\Sigma_l$  is internal alphabet.

A visibly pushdown automata(VPA) over  $\Sigma = \langle \Sigma_c, \Sigma_r, \Sigma_l \rangle$  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,  $\Gamma$  is a finite stack alphabet with special end-of-stack symbol  $\bot$ ,  $\Delta \subseteq (L \times \Sigma_c \times L \times (\Gamma \setminus \bot)) \cup (L \times \Sigma_r \times \Gamma \times L) \cup (L \times \Sigma_l \times L)$  is the transition relation, and  $F \subseteq L$  is final locations.

Alur and Madhusudan [6] showed that VPAs are determinizable and closed under boolean operations. A language L of finite words defined over visibly pushdown alphabet  $\Sigma$  is a visibly pushdown language(VPL) if there exist a VPA M such that L(M) = L. The class of languages accepted by visibly pushdown automata are closed under boolean operations with decidable emptiness property [6].

#### 3 Dense-Time Visibly Pushdown Multistack Automata

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  $\Gamma^h$  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 4** ([8]). 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,
- $-\Gamma^h$  is the finite alphabet of stack h and has special end-of-stack symbol  $\perp_h$ ,
- $-\Delta_c^h \subseteq (L \times \Sigma_c^h \times \Phi(C_h) \times L \times (\Gamma^h \setminus \{\bot_h\})) \text{ is the set of call transitions,} \\ -\Delta_r^h \subseteq (L \times \Sigma_r^h \times \mathcal{I} \times \Gamma^h \times \Phi(C_h) \times L) \text{ is set of return transitions,} \\ -\Delta_l^h \subseteq (L \times \Sigma_l^h \times \Phi(C_h) \times L) \text{ is set of internal transitions, and} \\ -F \subseteq L \text{ is the set of final locations.}$

Let  $w = (a_0, t_0), \dots, (a_e, t_e)$  be a timed word. A configuration of the dt-ECMVPA is a tuple  $(\ell, \nu_i^w, (((\gamma^1 \sigma^1, age(\gamma^1 \sigma^1)), \dots, (\gamma^n \sigma^n, age(\gamma^n \sigma^n))))$  where  $\ell$  is the current location of the dt-ECMVPA, function  $\nu_i^w$  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  $\gamma^h$  being the topmost symbol, and  $\sigma^h$  the string representing stack contents below  $\gamma^h$ , 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  $age(\sigma^h) = \langle t_1, t_2, \dots, t_g \rangle$ and for  $\tau \in \mathbb{R}_{\geq 0}$  then we write  $age(\sigma^h) + \tau$  for the sequence  $\langle t_1 + \tau, t_2 + \tau, \dots, t_g + \tau \rangle$ . For a sequence  $\sigma^h = \langle \gamma_1^h, \dots, \gamma_g^h \rangle$  and a stack symbol  $\gamma^h$  we write  $\gamma^h :: \sigma^h$  for  $\langle \gamma^h, \gamma_1^h, \dots, \gamma_g^h \rangle$ .

A run of a dt-ECMVPA on a timed word  $w = (a_0, t_0), \dots, (a_e, t_e)$  is a sequence of configurations:

 $(\ell_0, \nu_0^w, (\langle \bot^1 \rangle, \langle \vdash \rangle), \dots, (\langle \bot^n \rangle, \langle \vdash \rangle)), \ (\ell_1, \nu_1^w, ((\sigma_1^1, age(\sigma_1^1)), \dots, (\sigma_1^n, age(\sigma_1^n)))), \dots, (\ell_{e+1}, \nu_{e+1}^w, (\sigma_{e+1}^1, age(\sigma_{e+1}^1)), \dots, (\sigma_{e+1}^n, age(\sigma_{e+1}^n))) \text{ where } \ell_i \in L, \ \ell_0 \in L^0, \ \sigma_i^h \in (\Gamma^h)^* \bot^h, \text{ and } \ell_i \in L, \ \ell_0 \in L^0, \ \sigma_i^h \in (\Gamma^h)^* \bot^h, \text{ and } \ell_i \in L, \ \ell_0 \in L^0, \ \sigma_i^h \in (\Gamma^h)^* \bot^h, \text{ and } \ell_i \in L, \ \ell_0 \in L^0, \ \sigma_i^h \in (\Gamma^h)^* \bot^h, \text{ and } \ell_i \in L, \ \ell_0 \in L^0, \ \sigma_i^h \in (\Gamma^h)^* \bot^h, \text{ and } \ell_i \in L, \ \ell_0 \in L^0, \ \sigma_i^h \in (\Gamma^h)^* \bot^h, \ \ell_0 \in L^0, \ \ell_0 \in L$ for each  $i, 0 \le i \le e$ , 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  $\gamma^h \in \Gamma^h \setminus \{\bot^h\}$  is then pushed onto the stack h, and its age is initialized to zero, i.e.  $(\sigma_{i+1}^h, age(\sigma_{i+1}^h)) = (\gamma^h :: \varphi^h)$  $\sigma_i^h, 0 :: (age(\sigma_i^h) + (t_i - t_{i-1})))$ . 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  $\gamma^h$  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  $\gamma^h = \langle \perp^h \rangle$ , then  $\gamma^h$  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  $\nu_i^w \models \varphi$ . 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  $1 \le h \le n$ .

A run  $\rho$  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  $\Delta_c^h$ , 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  $\Delta_r^h$ , either  $\phi_1 \wedge \phi_2$  is unsatisfiable or  $I_1 \cap I_2 = \emptyset$ ; and for any two moves  $(\ell, a, \phi_1, \ell')$  and  $(\ell, a, \phi_2, \ell')$  in  $\Delta_l^h$ , 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  $(\ell, a, I, \gamma, \phi, \ell') \in \Delta_x^h$ , 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 5** (Matching relation). Consider a timed word w over  $\Sigma$ . Let  $\mathcal{P}_c^h$  (resp.  $\mathcal{P}_r^h$ ) denote the set of positions in w where a symbol from  $\Sigma_c^h$  i.e. a call symbol (resp.  $\Sigma_r^h$  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 [18] 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 [18].

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  $\Sigma$  is k-scoped if for each matching call of stack h, its corresponding return occurs within at most kmaximal stack-h contexts.

Let  $Scope(\Sigma, k)$  denote the set of all k-scope timed words over  $\Sigma$ . For any fixed k, a k-scope dt-ECMVPA over  $\Sigma$  is a tuple A=(k,M) where  $M=(L,\Sigma,\Gamma,L^0,F,\Delta)$  is a dt-ECMVPA over  $\Sigma$ . 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 [17,18] which help us extend the notion of scoped words from untimed to timed words.

**Definition 6** (k-scoped splitting [17,18]). 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 occurring 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  $i in \{1, \ldots, m-1\}$ . 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  $w_1: w_2: \ldots : w_m$  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 stackh contexts. See that an h-projection is a context-splitting. An ordered tuple of m h-contexts is k-bounded if 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 hsuch 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 [17,18] for untimed words we get the following Lemma.

**Lemma 7.** 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 [7], which are used in determinization of k-scope dt-ECMVPA.

#### Switching vectors 3.1

Let A be k-scoped dt-ECMVPA over  $\Sigma$  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  $\Sigma^h$ . 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 mbe 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  $w^{\prime h}$  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  $\bar{w}^h = \bar{u}^h_1 \# \bar{u}^h_2 \# \dots \# \bar{u}^h_z$ , where # is either # or #', and  $\bar{u}^h_i = (\bar{a}^h_{i1}, \bar{t}^h_{i1}), (\bar{a}^h_{i2}, \bar{t}^h_{i2}) \dots (\bar{a}^h_{i,s_i}, \bar{t}^h_{i,s_i})$ , 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  $j \leq z$  if  $\overline{\mathbb{V}}^h[j] = (q_j, I_j, q'_j)$  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  $r \leq z$  and  $V_i^h = (L \times \mathcal{I} \times L)^{\leq k}$ . 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  $\bar{\mathbb{V}}^h = P_1^h P_2^h \dots P_z^h$ , 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  $\ell_1^h$ , with the h-th stack containing  $\perp^h$ , and an initial valuation  $\nu_1^h$  of all recorders and predictors of  $\Sigma^h$ , the dt-ECMVPA A processes  $u_1^h$  and reaches some  $\ell_1'^h$  with stack content  $\sigma_2^h$  and clock valuation  $\nu_1'^h$ . The processing of  $u_2^h$  by A then starts at location  $\ell_2^h$ , 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  $(\ell_2^h, \nu_2^h)$ where  $\nu_2^h$  is the valuation of all recorders and predictors updated from  $\nu_1^{th}$  with respect to t. The stack content remains same as  $\sigma_2^h$  when the processing of  $u_2^h$  begins.

2. In general, starting in  $(\ell_i^h, \nu_i^h)$ , i > 1 with the h-th stack containing  $\sigma_i^h$ , and  $\nu_i^h$  obtained from  $\nu_{i-1}^h$  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  $\sigma_{i+1}^h$ . 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  $\ell_{i+1}^h$ , and stack content being  $\sigma_i^h$ .

These switching vectors were used in to get the determinizability of k-round dt-ECMVPA [7] 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 7.

**Lemma 8.** (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  $\bar{\mathbb{V}}^h = \prod_{i=1}^r \bar{\mathbb{V}}_i^h$  where  $r \leq z$  and  $\bar{\mathbb{V}}_i^h = (L \times \mathcal{I} \times L)^{\leq k}$  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  $i \leq j$  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  $\Delta^h$  are given below.

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

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  $\Sigma^h$ , 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,  $\bar{\mathbb{V}}^h = \Pi_{y \in \{1, \dots, d_h\}} \langle l_y^h, l_y^h, l_y'^h \rangle$ .

We define the relation between elements of  $\overline{\mathbb{V}}^h$  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  $(\bar{\mathbb{V}}^1, \dots, \bar{\mathbb{V}}^n)$  is called **globally correct** if we can stitch together runs of all  $A_k^h$ s 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  $\Sigma$  which is in L(A) then we have, collection of globally correct switching vectors  $(\bar{\mathbb{V}}^1, \dots, \bar{\mathbb{V}}^n)$ .

The following lemma enables us to construct a run of k-scope ECMVPA on word w over  $\Sigma$  from the runs of ECVPA  $A_k^j$  on h-projection of w over  $\Sigma^j$  with the help of switching vectors.

**Lemma 9 (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  $\Sigma$ . Then  $w \in L(A)$  iff there exist a collection of globally correct sequences of switching vectors for word w.

*Proof.* ( $\Rightarrow$ ): Assuming  $w \in L(A)$  to prove existence of a collection of globally correct sequences is easy.

 $(\Leftarrow)$ : Assume that we have a collection of globally correct switching vectors  $(\bar{\mathbb{V}}^1, \dots, \bar{\mathbb{V}}^n)$  of A, for a word w. For each h in  $\{1, \dots, n\}$  we have k-scoped splitting of word w and we have a run of  $A_k^h$  on  $\bar{w}^h$ , which uses  $\bar{\mathbb{V}}^h$ , the corresponding switching vector of A.

Let  $\bar{w}$  be the word over  $\Sigma \cup \{\#^1, \dots, \#^n\} \cup \{\#'^1, \dots, \#'^n\}$ , obtained from w by using k-splitting as follows. Let  $w = w_1 w_2 \dots w_d$  We first insert  $\#^j$  before  $w_1$  if  $w_1$  is a j-context. Then for all i in  $\{1, \dots, d\}$ , insert  $\#^l$  in between  $w_i$  and  $w_{i+1}$  if  $w_{i+1}$  is l-th context. Let us insert special symbol  $\#^s$  to mark the end of word w. Now using k-splitting of word w we throw in  $\#'^h$  as done in the case of obtaining  $\bar{w}^h$  from  $w^h$ .

Now we build a composite machine whose constituents are  $A_k^h$  for all h in  $\{1,\ldots,n\}$ . Its initial state is  $(p^1,\ldots,p^j,\ldots,p^n,j)$  where  $p^j$  s are the initial states of  $A_k^j$ s and the last component tells which component is processing current symbol. According to this, initially we are processing first j-context in  $\bar{w}^j$ .

We first run  $A_k^j$  on  $\bar{w}^j$  updating location  $p^j$  to  $p'^j$  where  $p'^j = (\bar{v}^j = (l_1^j, I_1^j, l_1'^j), a)$ . When it reads  $\#^g$  then  $p'^j = (\bar{v}^j = (l_1^j, I_1^j, l_1'^j), \#^g)$  and the composite state is changed to  $(p^1, \ldots, p'^j, \ldots, p^n, g)$  meaning that next context belongs to g-th stack. So we start simulating A on first g-context in  $\bar{w}^g$ . But to do that  $A_k^j$  should have left us in a state where  $A_k^g$  can begin. That is if  $p^g = (\bar{v}^g = (l_1^g, I_1^g, l_1'^g), \#^g)$  we should have  $l_1^g = l_1'^j$  and  $x_{\#'g} = 0$  which will be the case as this is the first time we accessing g-th stack.

In general, if we are leaving a j-context and the next context belongs to g-th stack then the time elapsed from reading the last symbol of last g-context should falls in the interval of the first element of next switching vector processing next g-context in  $\bar{w}^g$ , along with the matching of state in which the previous context has left us in.

# 4 Emptiness checking and Determinizability of scope-bounded ECMVPA

First we show that emptiess problem is decidable using the ideas from [8]. Fix a  $k \in \mathbb{N}$ .

**Theorem 10.** Emptiness checking for k-scope ECMVPA is decidable.

*Proof (Proof sketch)*. Decidability of emptiness checking of k-round ECMVPA has been shown in [8]. This proof works for any general ECMVPA as the notion k-round has not been used. So, the same proof can be used to decide emptiness checking of k-scope ECMVPA.

Rest of the section is devoted for the proof of following theorem.

**Theorem 11.** The class of k-scope ECMVPA are determinizable.

To show this we use the determinization of VPA [6] and we recall this construction here for the reader's convenience.

**Determinization of VPA** [6] Given a VPA  $M = (Q, Q_{in}, \Gamma, \delta, Q_F)$ , the idea in [6] is to do a subset construction. Let  $w = w_1 a_1 w_2 a_2 w_3$  be a word 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 stack contents  $(S_2, R_2, a_2)(S_1, R_1, a_1) \perp$  and is in control state  $(S_2, R_2, R_2, a_2)$  such that starting with q on  $w_2$  and an empty stack (contains only  $\perp$ ), we reach q' with stack  $\perp$ . 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' in the case of call and internal symbols is the set of all reachable states obtained from R, using all possible transitions on the current symbol read, and where 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:

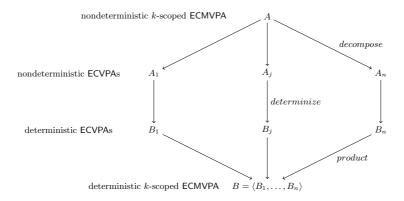
A pair  $(q, q') \in S'$  if there is some  $(q, q'') \in S_1$ , and  $(q'', a, q_1, \gamma) \in \delta$  (the push transition on a),  $(q_1, q_2) \in S$ , and  $(q_2, a', \gamma, q') \in \delta$  (the pop transition on a').

In this case, a state  $q' \in R'$  if there exists some location q in  $R_1$  and there exists and  $(q, a, q_1, \gamma) \in \delta$  (the push transition on a),  $(q_1, q_2) \in S'_1$ , and  $(q_2, a', \gamma, q') \in \delta$  (the pop transition on a'). The important thing is the reachable set of states of non-deterministic machine after the call transition is updated using all possible summaries of well-matched words possible after the corresponding push transition in the past.

The set of final locations of the determinized VPA are  $\{(S,R) \mid R \text{ contains a final state of the starting VPA}\}$ , 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.

**Determinization of k-scope ECMVPA** In this section we show that k-scope ECMVPA are determinizable using the result from [23] that single stack ECVPA are determinizable.

Let  $A=(k,L,\Sigma,\Gamma,L^0,F,\Delta)$  be the k-scoped ECMVPA and let  $A_k^h$  be the ECVPA on  $\Sigma^h \cup \{\#^h,\#'^h\}$ . Each  $A_k^h$  is determinizable [23]. Recall from [23] that an ECVPA  $A_k^h$  is untimed to obtain a VPA  $ut(A_k^h)$  by encoding the clock constraints of  $A_k^h$  in an extended alphabet. This VPA can be converted back into an ECVPA  $ec(ut(A_k^h))$  by using the original alphabet, and replacing the clock constraints. This construction is such that  $L(ec(ut(A_k^h))) = L(A_k^h)$  and both steps involved preserve determinism. Determinization of VPA  $ut(A_k^h)$  is done in the usual way [6]. This gives  $Det(ut(A_k^h))$ . Again,  $ec(Det(ut(A_k^h)))$  converts this back into a ECVPA by simplifying the alphabet, and writing the clock constraints. The set of locations remain unchanged in  $ec(Det(ut(A_k^h)))$  and  $Det(ut(A_k^h))$ . This translation also preserves determinism, hence  $B_k^h = ec(Det(ut(A_k^h)))$  is a deterministic ECVPA language equivalent to ECVPA  $A_k^h$ . This process is also explained in the Figure 2.



**Fig. 2.** Determinazation of k-scoped ECMVPA

The locations of  $B_k^h$  are thus of the form (S,R) where R is the set of all reachable control states of  $A_k^h$  and S is a set of ordered pairs of states of  $A_k^h$  as seen in section 4. On reading  $\kappa_j$ , the R component of the state reached in  $B_k^h$  is the set  $\{\langle V^h \rangle \mid V^h \text{ is a last component of switching sequence } \bar{\mathbb{V}}^h \text{ of } A_k^h\}$ . Lemmas 8 and Lemma 9 follow easily using  $B_k^h = ec(Det(ut(A_k^h)))$  in place of  $A_k^h$ . We now obtain a deterministic k-scoped ECMVPA B for language of k-scoped ECMVPA A by simulating  $B_k^1, \ldots, B_k^h$  one after the other on reading w, with the help of globally correct sequences of k-scoped switching vectors of  $A_k^h$  s.

Automaton B in its finite control, keeps track of the locations of all the  $B_k^h$ 's, along with the valuations of all the recorders and predictors of  $\Sigma$ . It also remembers the current context number of the  $B_k^h$ 's to ensure correct simulation.

Let  $B_k^h = (Q^h, \Sigma^h \cup \{\#, \#'\}, \Gamma^h, Q_0^h, F^h, \delta^h)$ . Locations of  $B_k^h$  have the form  $(S^h, R^h)$ . The initial state of  $B_k^h$  is the set consisting of all  $(S_{in}, R_{in})$  where  $S_{in} = \{(q, q) \mid q \text{ is a state of } A_k^h\}$ , and  $R_{in}$  is the set of all initial states of  $A_k^h$ . Recall that a final state of  $A_k^h$  is  $V^h$  the last component switching vector  $V^h$  of a correct switching sequence  $\bar{\mathbb{V}}^h$  of  $A_k^h$ . Thus, an accepting run in  $B_k^h$  goes through states  $(S_{in}, R_{in}), (S_1, R_1), \dots, (S_m, R_m), \langle V^h \rangle$ .

Locations of B have the form  $(q_1, \ldots, q_n, h)$  where  $q_y$  is a location of  $B_y$ , h is the stack number of current context. Without loss of generality we assume that the first context in any word belongs to stack 1. The initial location of B is  $(q_{11}, q_{12}, \ldots, q_{1n}, 1)$  where  $q_{1h}$  is the initial location of  $B_k^h$ . 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  $\Delta$  of B, using the transitions  $\delta^j$  of  $B_k^h$ . Recall that B processes  $w=w_1w_2\dots w_m$  where each  $w_i$  is context of some stack. When w is k-scoped we can see w as the concatenation of contexts  $w=\bar{w}_1\bar{w}_2\dots\bar{w}_{\bar{m}}$  where consecutive contexts need not belong to different stacks, and  $\bar{m}\geq m$ . Let  $\eta=(q_1,\dots,q_{h-1},q_h,q_{h+1},\dots,q_n,h)$  and let  $\zeta=(q_1,\dots,q_{h-1},q,q_{h+1},\dots,q_n,h)$ , where  $q_h$  is some state of  $B_k^h$  while processing some context of h-th stack.

1. Simulation of  $B_k^h$  when the  $\bar{w}_{\bar{m}}$  is not an h-context.

```
 \begin{aligned} & - \langle \eta, a, \varphi, \zeta \rangle \in \Delta_l^j \text{ iff } (q_{ij}, a, \varphi, q) \in \delta_l^j \\ & - \langle \eta, a, \varphi, \zeta, \gamma \rangle \in \Delta_c^j \text{ iff } (q_{ij}, a, \varphi, \gamma, q) \in \delta_c^j \\ & - \langle \eta, a, \gamma, \varphi, \zeta \rangle \in \Delta_r^j \text{ iff } (q_{ij}, a, \gamma, \varphi, q) \in \delta_r^j \end{aligned}
```

2. Change context from h to h+1. Assume that we have processed h-context  $\bar{w}_g$  and next context is  $\bar{w}_{g+1}$  which is an (h+1)-context. Let the current state of B is  $\eta = (q_1, \ldots, q_{h-1}, q_h, q_{h+1}, \ldots, q_n, h)$  after processing  $\bar{w}_g$ . At this point  $B_k^h$  reads symbol # in  $\bar{\kappa}^h$  and moves from state  $q_h$  to  $q_h'$ . Therefore, current state of B becomes  $\eta' = (q_1, \ldots, q_{h-1}, q_h', q_{h+1}, \ldots, q_n, h)$  At this point, B invokes  $B_k^{h+1}$  and starts reading first symbol of  $\bar{w}_{g+1}$  when B is in state  $\eta'$ . This is meaningful because of Lemma 9. States of  $q_{h+1}$  are of the form  $(S^{h+1}, R^{h+1})$  where  $R^{h+1}$  have states of  $A_k^{h+1}$  of the form  $(\bar{\mathbb{V}}^{h+1}, a)$ , which is a possible state after processing last h+1 context. Globally correct stitching sequence guarantees that for all  $(\bar{\mathbb{V}}^h, a) \in R'^h$  we have at least one  $(\bar{\mathbb{V}}^{h+1}, a) \in R'^{h+1}$  such that  $\bar{\mathbb{V}}'^h[last] = (x, I, y)$  then  $\bar{\mathbb{V}}'^{h+1}[first] = (y, I', z)$  and valuation of clocks at  $\eta'$  belongs to I', where x, y are locations of A, and where last is the last component of  $\bar{\mathbb{V}}'^h$  after processing  $\bar{w}_g$ , and first is the first componet of switching vector  $\bar{\mathbb{V}}'^{h+1}$  for processing  $\bar{\mathbb{V}}'^h + 1$ . Component location  $q_{h+1}$  will be replaced based on a transition of  $B_k^{h+1}$ , and  $q'^h$  is also replaced with  $q^h$  to take care of the transition on # in  $B_k^h$ , where  $q^h = (S^h, R^h)$  with  $R^h$  containing all locations of the form  $(\bar{\mathbb{V}}^h, a)$ , where a is the last symbol of  $\bar{w}_g$ .

```
 - \langle (\dots, q'_h, q_{h+1}, \dots, h), a, \varphi, (\dots, q_h, q, \dots, h+1) \rangle \in \Delta_l^{h+1} \text{ iff } 
 (q_{h+1}, a, \varphi, q) \in \delta_l^{h+1} 
 - \langle (\dots, q'_h, q_{h+1}, \dots, h), a, \varphi, (\dots, q_h, q, \dots, h+1), \gamma \rangle \in \Delta_c^{h+1} \text{ iff } 
 (q_{h+1}, a, \varphi, q, \gamma) \in \delta_c^{h+1} 
 - \langle (\dots, q'_h, q_{h+1}, \dots, h), a, \gamma, \varphi, (\dots, q_h, q, \dots, h+1) \rangle \in \Delta_r^{h+1} \text{ iff } 
 (q_{h+1}, a, \gamma, \varphi, q) \in \delta_r^{h+1}
```

Transitions of  $B^{h+1}$  continue on  $(\ldots, q_h, q, \ldots, h+1)$  replacing only the (h+1)-th entry until  $\bar{w}_{q+1}$  is read completely.

3. Reading  $\bar{w}_{\bar{m}}$  the last context of word w. Without loss of generality assume that this is the context of  $B_k^n$  and the previous context  $\bar{w}_{\bar{m}-1}$  was the context of  $B_k^{n-1}$ . When  $B_k^{n-1}$  stops reading last symbol of  $\bar{w}_{\bar{m}}$  then it is in the state  $\eta = (\mathcal{V}^1, \mathcal{V}^2, \dots, \mathcal{V}^{n-1}, (S'^n, R'^n), n)$  where no more symbols are to be read, and each  $B_k^h$  is in the state  $\langle \mathcal{V}^h \rangle$ , where  $\mathcal{V}^h$  is the last component of k-scoped switching vector  $\bar{\mathbb{V}}^h$ , where  $R'_n$  is the set of all locations of the form  $(\mathcal{V}^n, a)$ , where a is the last symbol of  $\bar{\mathbb{V}}^h$ . This is accepting iff there exists  $\ell'_{kn}$  as the second destination of last tuple in  $\mathcal{V}^n$  and  $\ell'_{kn} \in F$ . Note that we have ensured the following:

- (a)  $\bar{\mathbb{V}}^h = \Pi_{1 \leq i \leq m_h} \mathcal{V}_i^h$  and is part of correct global sequence for all  $1 \leq h \leq n$ , and  $m_h$  is the number of k-switching vectors of  $A_k^h$  used in the computation.
- (b) At the end of  $\bar{w}_{arm}$ , we reach in  $B_k^h$ ,  $(S'_{kn}, R'_{kn})$  such that there exists  $\ell'_{kn}$  as the second destination of the last tuple in  $\mathcal{V}^n$  and  $\ell'_{kn} \in F$ .
- (c) When switching from one context to another continuity of state is used as given in the correct global sequence.

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  $\bar{\mathbb{V}}^1 \dots \bar{\mathbb{V}}^n$ .

## 5 Emptiness checking and Determinizability of scope-bounded dt-ECMVPA

Fix a  $k \in \mathbb{N}$ . The proof is via untiming stack construction to get k-scope ECMVPA for which emptiness is shown to be decidable in Theorem 10.

We first informally describe the untiming-the-stack construction to obtain from a k-scope-dt-ECMVPA M over  $\Sigma$ , an k-scope-ECMVPA M' over an extended alphabet  $\Sigma'$  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  $\kappa$  be the maximum constant used in the k-scope-dt-ECMVPA M while checking the age of a popped symbol in any of the stacks. Let us first consider a call transition  $(l,a,\varphi,l',\gamma)\in\Delta^i_c$  encountered in M. To construct an k-scope-ECMVPA M' from M, we guess the interval used in the return transition when  $\gamma$  is popped from ith stack. Assume the guess is an interval of the form  $[0,\kappa]$ . This amounts to checking that the age of  $\gamma$  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  $(\gamma,<\kappa,\mathrm{first})^1$  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 *i*th stack. All symbols of  $Z_i^{\sim}$  are internal symbols in M' i.e.  $\Sigma_i' = \{(\Sigma_i', \Sigma_i') \mid (\Sigma_i', \Sigma_$  $\{\Sigma_c^i, \Sigma_l^i \cup Z_i^{\sim}, \Sigma_r^i\}$ . 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  $x_a=0$  ensures that no time is elapsed by the new transition. The information  $<_i \kappa$  is retained in the control state until  $(\gamma, <\kappa, \text{first})$  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  $(l', b, \psi, q, \beta)$ . In M', from  $(l', \{<_i \kappa\})$ , we first guess the constraint that will be checked when  $\beta$  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  $(\beta, <\zeta, first)$  on to the ith stack. The idea thus is to keep the obligation  $<_i\kappa$  alive in the control state until  $\gamma$  is popped; the value of  $x_{\leq,\kappa}$  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  $x_{\leq_i \kappa}$ . The  $x_{\leq_i \kappa}$  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  $x_{\geq_i \kappa}$  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  $\geq \zeta \land \leq \kappa$  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  $(\gamma, S, \mathtt{first})$  or  $(\gamma, S, -)$ , where S is a singleton set of the form  $\{<\kappa\}$  or  $\{>\zeta\}$ , or a set of the form  $\{<\kappa, >\zeta\}^2$ . Consider the case when the top of the ith stack symbol is  $(\gamma, \{<\kappa, >\zeta\}, \mathtt{first})$ . In M', on reading a, the control switches from (l, P) to (l', P') for  $P' = P \setminus \{<\kappa\}$  iff the guard  $\varphi$  evaluates to true, the interval I is  $(\zeta, \kappa)$  (this validates our guess made at the time of push) and the value of clock  $x_{<_i\kappa}$  is

<sup>&</sup>lt;sup>1</sup> It is sufficient to push  $(\gamma, < \kappa, \text{first})$  in stack i, since the stack number is known as i

<sup>&</sup>lt;sup>2</sup> This last case happens when the age checked lies between  $\zeta$  and  $\kappa$ 

 $<\kappa$ , and the value of clock  $x_{>i\zeta}$  is  $>\zeta$ . Note that the third component first says that there are no symbols in ith stack below  $(\gamma, \{<\kappa,>\zeta\}, \mathtt{first})$  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  $(\beta,S,\mathtt{first})$  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  $<_i\kappa, \le_j\eta$ , 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  $\le_i\kappa$  for each ith stack, since we do not remember  $>\zeta$  constraints in the finite control.

We now give the formal construction.

#### Reduction from k-scope-dt-ECMVPA to k-scope-ECMVPA:

Let  $Z^{\sim} = \bigcup_{i=1}^n Z_i^{\sim}$  and let  $S^{\sim} = \{ \sim c \mid c \in \mathbb{N}, c \leq \kappa, \sim \in \{ <, \leq, >, \geq, = \} \}$ . Given k-scope-dt-ECMVPA  $M = (L, \Sigma, \Gamma, L^0, F, \Delta)$  with max constant  $\kappa$  used in return transitions of all stacks, we construct k-scope-ECMVPA  $M' = (L', \Sigma', \Gamma', L'^0, F', \Delta')$  where  $L' = (L \times 2^{Z^{\sim}}) \cup (L_{\Sigma_i \times S^{\sim}} \times 2^{Z^{\sim}}) \cup (L_{\Sigma_i \times S^{\sim}} \times 2^{Z^{\sim}})$ ,  $\Sigma_i' = (\Sigma_c^i, \Sigma_l^i \cup Z_i^{\sim}, \Sigma_r^i)$  and  $\Gamma_i' = \Gamma_i \times 2^{S^{\sim}} \times \{ \text{first}, - \}, \ L^0 = \{ (l^0, \emptyset) \mid l^0 \in L^0 \},$  and  $F = \{ (l^f, \emptyset) \mid l^f \in F \}.$ 

The transitions  $\Delta'$  are defined as follows:

Internal Transitions. For every  $(l, a, \varphi, l') \in \Delta_l^i$  we have the set of transitions  $((l, P), a, \varphi, (l', P)) \in \Delta_l^{i'}$ . 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 corresponds to the guessed pop constraint being  $<\kappa$ . In the case that, obligation is  $<\kappa$  is alive in the state, hence there is no need to reset the clock  $x_{<i\kappa}$ . Otherwise, the obligation  $<\kappa$  is fresh and hence it is remembered as first in the *i*th stack, and the clock  $x_{<i\kappa}$  is reset.

$$\begin{split} &((l,P),a,\varphi,(l',P),(\gamma,\{<\!\kappa\},-))\!\in\!\!\Delta^{i'}_{\phantom{i'}c}\quad \text{if } \!<_i\!\kappa\!\in\!\!P\\ &((l,P),a,\varphi,(l'_{a,<\!\kappa},P'),(\gamma,\{<\!\kappa\},\text{first}))\!\in\!\!\Delta^{i'}_{\phantom{i'}c}\quad \text{if } \!<_i\!\kappa\!\notin\!\!P \text{ and } P'=P\cup\{<_i\!\kappa\}\\ &((l'_{a,<\!\kappa},P'),<_i\!\kappa,x_a=0,(l',P'))\!\in\!\!\Delta^{i'}_{\phantom{i'}l} \end{split}$$

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

$$((l,P),a,\varphi,(l'_{a,>\kappa},P),(\gamma,\{>\!\!\kappa\},-))\in \! \Delta^{i'}_{c} \text{ and } ((l'_{a,>\kappa},P),>_{l}\! \kappa,x_{a}=0,(l',P))\in \! \Delta^{i'}_{l}$$

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.

$$\begin{split} &((l,P),a,\varphi,(l'_{a,<\kappa,>\zeta},P'),(\gamma,\{<\kappa,>\zeta\},\mathtt{first})) \in \varDelta^{i'}_{c} \ \ \mathrm{if} <_{i}\kappa \not \in P, P' = P \cup \{<_{i}\kappa\} \\ &((l'_{a,<\kappa,>\zeta},P'),>_{i}\zeta,x_{a}=0,(l'_{a,<\kappa},P')) \in \varDelta^{i'}_{l} \\ &((l'_{a,<\kappa},P'),>_{i}\kappa,x_{a}=0,(l'_{a,<\kappa},P')) \in \varDelta^{i'}_{l} \\ &((l,P),a,\varphi,(l'_{a,>\zeta},P),(\gamma,\{<\kappa,>\zeta\},-)) \in \varDelta^{i'}_{c} \ \ \mathrm{if} <_{i}\kappa \in P \end{split}$$

Return Transitions. For every  $(l, a, I, \gamma, \varphi, l') \in \Delta_r^i$ , transitions in  $\Delta_r^i$  are:

- 1.  $((l, P), a, (\gamma, \{<\kappa, >\zeta\}, -), \varphi \land x_{<\iota\kappa} < \kappa \land x_{>\iota\zeta} > \zeta, (l', P))$  if  $I = (\zeta, \kappa)$ .
- $\begin{array}{l} 2. \ ((l,P),a,(\gamma,\{<\!\kappa,>\!\zeta\},\mathtt{first}),\varphi \wedge x_{<_i\kappa}<\!\kappa \wedge x_{>_i\zeta}>\!\zeta,(l',P')) \\ \text{where } P'=P\backslash \{<_i\kappa\}, \text{ if } I=(\zeta,\kappa). \end{array}$
- 3.  $((l,P),a,(\gamma,\{<\kappa\},-),\varphi \wedge x_{<_{l}\kappa}<\kappa,(l',P))$  if  $I=[0,\kappa).$
- 4.  $((l,P),a,(\gamma,\{<\kappa\},\mathtt{first}),\varphi\wedge x_{<_i\kappa}<\kappa,(l',P'))$  with  $P'=P\setminus\{<_i\kappa\}$  if  $I=[0,\kappa)$ .
- 5.  $((l, P), a, (\gamma, \{>\zeta\}, -), \varphi \land x_{>i\zeta} > \zeta, (l', P))$  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  $P\subseteq Z^{\sim}$ . If w is a matched word then P is empty, and there must be any obligations which are pending at the end.

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  $\le_i \kappa$  or  $b_{1i}$  is  $>_i \zeta$ , or  $b_{1i}$  is  $>_i \zeta$  and  $b_{2i}$  is one of  $<_i \kappa$  or  $\le_i \kappa$  for some  $\kappa, \zeta \leq k$ . This follows from the three kinds of call transitions in M'.

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-scope-dt-ECMVPA, then M' is a k-scope-ECMVPA. Since M' is a k-scope-ECMVPA, its emptiness check is decidable using Theorem 10, which uses the standard region construction of event clock automata [5] to obtain a k-scope-MVPA, which has a decidable emptiness [20].

#### **Theorem 12.** The emptiness checking for k-scope dt-ECMVPA is decidable.

In [8] we have shown that k-round dt-ECMVPA are determinizable. Using an untiming construction to get k-round ECMVPA, determinize it and again converting this to get deterministic k-round dt-ECMVPA.

For k-scope dt-ECMVPA using the stack untiming construction we get a k-scope ECMVPA. This is determinized to get deterministic k-scope ECMVPA. We convert this back to get deterministic k-scope dt-MVPA. The morphisms used for this conversion are same as in [8].

#### **Theorem 13.** The k-scope dt-ECMVPA are determinizable.

Proof. Consider a k-scope dt-ECMVPA  $M=(L,\Sigma,\Gamma,L^0,F,\Delta)$  and the corresponding k-scope ECMVPA  $M'=(L',\Sigma',\Gamma',L'^0,F',\Delta')$  as constructed in section  $\ref{thm:property}.$  From Theorem 11 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  $\Sigma'$  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-scope dt-ECMVPA 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  $\alpha$  is of the form  $>_i \zeta$ . Let  $(s_1, \alpha, \varphi, s_2)$  be the unique transition on  $\alpha \in \mathbb{Z}_i^{\sim}$ . By construction of M' (and hence det(M')), we know that  $\varphi$  is  $x_a = 0$  for some  $a \in \mathbb{Z}^i$ . 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  $>_i \zeta$ .
- Assume that α is of the form <<sub>i</sub>κ. Let (s<sub>1</sub>, α, φ, s<sub>2</sub>) be the unique transition on α∈Z<sub>i</sub><sup>∞</sup>. We also know that φ is x<sub>a</sub> = 0 for some a∈Σ<sup>i</sup>. From M', we also know that in Det(M'), there is a unique transition of one of the following forms preceding (s<sub>1</sub>, α, φ, s<sub>2</sub>):

   (a) (s<sub>0</sub>, a, ψ, s<sub>1</sub>, (γ, α, −)),
   (b) (s<sub>0</sub>, a, ψ, s<sub>1</sub>, (γ, α, 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  $X \in \{\text{first}, -\}$ . As  $(s_1, \alpha, \varphi, s_2)$  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  $(s_0, a, \psi, s_2, (\gamma, \alpha, \text{first}))$ . 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  $\Sigma^i$  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,  $\varphi$  is a constraint checked on the clocks of  $C_{\Sigma^i}$  in M during return: transitions  $(s, a, (\gamma, \{<\kappa\}, X), \varphi \land x_{<_i\kappa} < \kappa, s')$  for  $X \in \{-, \text{first}\}$  are modified to  $(s, a, [0, \kappa), \gamma, \varphi, s')$ ; transitions  $(s, a, (\gamma, \{<\kappa, >\zeta\}, X), \varphi \land x_{>_i\zeta} > \zeta \land x_{<_i\kappa} < \kappa, s')$  for  $X \in \{-, \text{first}\}$  are modified to  $(s, a, (\zeta, \kappa), \gamma, \varphi, s')$ ; and transition  $(s, a, (\gamma, \{>\zeta\}, -), \varphi \land x_{>_i\zeta} > \zeta, s')$  are modified to the transitions  $(s, a, (\zeta, \infty), \gamma, \varphi, s')$ . Now it is straightforward to verify that the k-scope dt-ECMVPA M'' obtained from the k-scope ECMVPA 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-scope dt-ECMVPA.

It is easy to show that k-scoped ECMVPAs and k-scoped dt-ECMVPAs are closed under union and intersection; using Theorem 11 and Theorem 13 we get closure under complementation.

**Theorem 14.** The classes of k-scoped ECMVPLs and k-scoped dt-ECMVPLs are closed under Boolean operations.

### 6 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 [7]. Following [16], 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( $\Sigma$ ), the MSO logic over  $\Sigma$ , as:

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

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  $\phi \in \mathrm{MSO}(\Sigma)$  are timed words w over  $\Sigma$ . 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  $\varphi$  as the set of all words satisfying  $\varphi$ .

Words in  $Scope(\Sigma, k)$ , for some k, can be captured by an MSO formula  $Scope_k(\psi) = \bigwedge_{1 \leq j \leq n} Scope_k(\psi)^j$ , where n is number of stacks, where

$$Scope_{k}(\psi)^{j} = \forall y Q_{a}(y) \land a \in \Sigma_{j}^{r} \Rightarrow (\exists x \mu_{j}(x, y) \land (\psi_{kcnxt}^{j} \land \psi_{matcnxt}^{j} \land \psi_{noextracnxt}))$$
where  $\psi_{kcnxt}^{j} = \exists x_{1}, \dots, x_{k}(x_{1} \leq \dots \leq x_{k} \leq y \bigwedge_{1 \leq q \leq k} (Q_{a}(x_{q}) \land a \in \Sigma_{j} \land (Q_{b}(x_{q} - 1) \Rightarrow b \notin \Sigma_{j})),$ 
and  $\psi_{matcnxt}^{j} = \bigvee_{1 \leq q \leq k} \forall x_{i}(x_{q} \leq x_{i} \leq x(Q_{c}(x_{i}) \Rightarrow c \in \Sigma_{j})),$  and

 $\psi_{noextracnxt} = \exists x_l (x_1 \leq x_l \leq y) (Q_a(l) \land a \in \Sigma_j \land Q_b(x_l-1) \land b \in \Sigma_j) \Rightarrow x_l \in \{x_1, \dots, x_k\}.$  Formulas  $\psi_{noextracnxt}$  and  $\psi_{kcnxt}$  says 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  $\zeta = \varphi \land Scope(\psi)$ , it can be shown that the dt-ECMVPA M constructed for  $\zeta$  will be a k-dt-ECMVPA.

Hence we have the following MSO characterization.

**Theorem 15.** A language L over  $\Sigma$  is accepted by an k-scope dt-ECMVPA iff there is a MSO sentence  $\varphi$  over  $\Sigma$  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 Appendix B.

#### 7 Conclusion

In this work we have seen timed context languages characterized by k-scope ECMVPA and densetimed k-scope dt-ECMVPA, along with logical characterizations for both classes (for k-scope ECMVPA, equivalent MSO is obtained by dropping predicate  $\theta_j(x) \in I$ —which checks if the aging time of pushed symbol x of stack j falls in interval I—from MSO of k-scope dt-ECMVPA).

Here, while reading an input symbol of a stack, clock constraints involve the clocks of symbols associated with the same stack. It would be interesting to see if our results hold without this restriction. Another direction, would be to use alternate methods like Ramsey theorem based [2,14] or Anti-chain based [13,10] methods avoiding complementation and hence determinization to do language inclusion check.

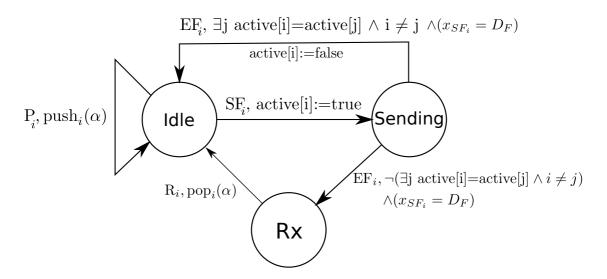
#### References

- 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, June 25-28, 2012. pp. 35-44. IEEE Computer Society (2012). https://doi.org/10.1109/LICS.2012.15, https://doi.org/10.1109/LICS.2012.15
- 2. Abdulla, P.A., Chen, Y., Clemente, L., Holík, L., Hong, C., Mayr, R., Vojnar, T.: Advanced Ramsey-based Büchi automata inclusion testing. In: CONCUR (2011)
- 3. Alur, R., Dill, D.: A theory of timed automata. TCS 126, 183–235 (1994)
- Alur, R., Dill, D.L.: Automata for modeling real-time systems. In: Paterson, M. (ed.) Automata, Languages and Programming, 17th International Colloquium, ICALP90, Warwick University, England, UK, July 16-20, 1990, Proceedings. Lecture Notes in Computer Science, vol. 443, pp. 322–335. Springer (1990). https://doi.org/10.1007/BFb0032042, https://doi.org/10.1007/BFb0032042
- 5. Alur, R., Fix, L., Henzinger, T.A.: Event-clock automata: A determinizable class of timed automata. TCS 211(1-2), 253–273 (1999)
- 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, June 13-16, 2004. pp. 202–211. ACM (2004). https://doi.org/10.1145/1007352.1007390, https://doi.org/10.1145/1007352.1007390
- Bhave, D., Dave, V., Krishna, S.N., Phawade, R., Trivedi, A.: A logical characterization for dense-time visibly pushdown automata. In: Dediu, A., Janousek, J., Martín-Vide, C., Truthe, B. (eds.) Language and Automata Theory and Applications 10th International Conference, LATA 2016, Prague, Czech Republic, March 14-18, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9618, pp. 89–101. Springer (2016). https://doi.org/10.1007/978-3-319-30000-9\_7, https://doi.org/10.1007/978-3-319-30000-9\_7
- 8. 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.) Developments in Language Theory 20th International Conference, DLT 2016, Montréal, Canada, July 25-28, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9840, pp. 38–50. Springer (2016). https://doi.org/10.1007/978-3-662-53132-7\_4, https://doi.org/10.1007/978-3-662-53132-7\_4
- 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, June 26-29, 1995. pp. 123-133. IEEE Computer Society (1995). https://doi.org/10.1109/LICS.1995.523250, https://doi.org/10.1109/LICS.1995.523250
- 10. Bruyère, V., Ducobu, M., Gauwin, O.: Visibly pushdown automata: Universality and inclusion via antichains. In: LATA. LNCS, vol. 7810, pp. 190–201 (2013)
- 11. Droste, M., Perevoshchikov, V.: A Logical Characterization of Timed Pushdown Languages, pp. 189–203. Springer International Publishing (2015)
- 12. 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, June 25-28, 2012. pp. 285–294 (2012). https://doi.org/10.1109/LICS.2012.39, https://doi.org/10.1109/LICS.2012.39
- 13. Fogarty, S., Vardi, M.Y.: Efficient Büchi universality checking. In: TACAS (2010)
- 14. Friedmann, O., Klaedtke, F., Lange, M.: Ramsey-based inclusion checking for visibly pushdown automata. ACM Trans. Comput. Logic 16(4), 34:1–34:24 (2015)
- 15. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley Publishing Company (1979)
- 16. La Torre, S., Madhusudan, P., Parlato, G.: A robust class of context-sensitive languages. In: LICS. pp. 161–170 (2007)
- 17. La Torre, S., Napoli, M., Parlato, G.: Scope-bounded pushdown languages. In: DLT, LNCS, vol. 8633, pp. 116–128. Springer International Publishing (2014)
- 18. La Torre, S., Napoli, M., Parlato, G.: Scope-bounded pushdown languages. Int. J. Found. Comput. Sci. **27**(2), 215–234 (2016)
- 19. Li, G., Cai, X., Ogawa, M., Yuen, S.: Nested timed automata. In: FORMATS (2013)
- 20. Salvatore, L., Madhusudan, P., Parlato, G.: The language theory of bounded context-switching. In: LATIN. pp. 96–107 (2010)
- 21. Trivedi, A., Wojtczak, D.: Recursive timed automata. In: ATVA (2010)
- 22. Tsybakov, B.S., Vvedenskaya, N.D.: Random multiple-access stack algorithm. Problems Inform. Transmission 16(3) (1980)
- 23. Van Tang, N., Ogawa, M.: Event-clock visibly pushdown automata. In: SOFSEM, pp. 558–569. Springer Berlin Heidelberg (2009)
- 24. Vardi, M., Wolper, P.: Reasoning about infinite computations. Information and Computation 115(1), 1 37 (1994)

## **Appendix**

# A Case study: Modeling A single channel packet switching communication network

A single channel packet switching communication network uses a common medium to send fixed sized packets from a transmitter to one or many receivers. Such medium may be a bus or a cable or any other transmission medium. The communication network consists of multiple stations sharing a single error free communication channel supported by the medium. All stations transmit fixed sized packets of data over the channel and stations receive them. The time is slotted into frames and the transmission of new packet can start only at the beginning of the frame. It is possible that one than one stations may transmit packets simultaneously, and in that case packets are said to collide. Such collision results in the transmission error and whether a collision has occured or not is decided at the end of frame. If the collision has happended, it is visible to all stations and failure of communication is concluded. If no collision occurs, packet is assumed to be transmitted successfully. All stations employ same protocol to avoid collision and retransmission. Such choice of such protocol determines various characteristics of the network like stability and maximum throughput. Here we model the Capetanakis-Tsybakov-Mikhailov (CTM) protocol [22] using a dt-ECMVPA shown in Figure 3. It shows state transitions for station i. The actual automaton for CTM protocol is constructed by taking cross product of automata for all stations.



**Fig. 3.** Model of CTM protocol for station i

For each station i, the symbol  $SF_i$ ,  $EF_i$ ,  $P_i$  and  $R_i$  denote the start of frame, end of frame, arrival of new packet and receipt of packet respectively. We use a global shared variable array active to record stations which are actively sending packets. active[i] is true iff station i is sending packet. We detect the occurance of collision if more that one entry in active is true. The constant  $D_F$  denote the durations of frame (i.e. time difference between SF and EF). There is an idle period between two successive frames (i.e. time difference between EF for first frame and SF of the next frame). We denote it using constant  $D_I$ . The event clocks have been used to measure such timing requirements. CTM algorithm expects that start and end of frame is synchronized for all stations. Such requirement can be taken care by introducing extra symbols during initialization phase (not shown in Figure 3). We introduce additional return input symbols  $P_0$ . At global time zero input symbol  $P_0$  occurs which pushes  $P_0$  on the stack at station 0. We then require that input symbols  $P_0$  occurs which pushes  $P_0$  occur when the age of popped  $P_0$  is zero. This ensures that the all  $P_0$  are synchronized when global time is zero.

#### B Details of Theorem 15

Here, we give the details of the translations from dt-ECMVPA to MSO and conversely. A technical point is regarding the projection operation: in general, it is known that event clock automata

(hence dt-ECMVPA) are not closed under projections. However, we need to handle projections while quantifying out variables in the MSO to dt-ECMVPA construction. We do this by working on Quasi dt-ECMVPA where the underlying alphabet  $\Sigma$  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 dt-ECMVPA from an MSO formula while projecting out variables. In Section B.1, we show the equi-expressiveness of quasi dt-ECMVPA and dt-ECMVPA which allows us to complete the logical characterization.

- **Logic to automata.** We first show that the language accepted by an MSO formula  $\varphi$  over  $\Sigma = \langle \Sigma_c^i, \Sigma_l^i, \Sigma_r^i \rangle_{i=1}^n$ ,  $L(\varphi)$  is accepted by a dt-ECMVPA. Let  $Z = (x_1, \ldots, x_m, X_1, \ldots, X_n)$  be the free variables in  $\varphi$ . As usual, we work on the extended alphabet  $\Sigma' = \langle \Sigma_c^{i'}, \Sigma_l^{i'}, \Sigma_r^{i'} \rangle_{i=0}^n$  where

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

for  $s \in \{c, l, r\}$ . A word w' over  $\Sigma'$  encodes a word over  $\Sigma$  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  $\Sigma'$  by assigning suitable ranking function. Quasi dt-ECMVPA are equiexpressive to dt-ECMVPA as explained in Section B.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  $\Sigma'$  wrt above partitioning scheme.

Let  $L(\psi)$  be the set of all words w' over  $\Sigma'$  such that the underlying word w over  $\Sigma$  satisfies formula  $\psi$  along with the valuation Val. Structurally inducting over  $\psi$ , we show that  $L(\psi)$  is accepted by a dt-ECMVPA. The cases  $Q_a(x), \mu_j(x,y)$  are exactly as in [16]. We only discuss the predicate  $\theta_j$  here. Consider the atomic formula  $\theta_j(x) \in I$ . To handle this, we build a dt-ECMVPA that keeps pushing symbols (a,v) onto the stack j whenever  $a \in \Sigma_c^j$ , initializing the age to 0 on push. It keeps popping the stack on reading return symbols  $(a',v'), a' \in \Sigma_r^j$ , 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 dt-ECMVPA.

Existential quantifications correspond to a projection by excluding the chosen variable from the valuation and renaming the alphabet  $\Sigma'$ . Let M be a dt-ECMVPA constructed for  $\varphi(x_1,\ldots,x_n,X_1,\ldots,X_m)$  over  $\Sigma'$ . 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  $\Sigma'$ . 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,  $\Sigma'$  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 dt-ECMVPA that accepts  $L(\psi)$  when  $\psi$  is an MSO sentence.

- Automata to logic. Consider a dt-ECMVPA  $M = (L, \Sigma, \Gamma, L^0, F, \Delta)$ . For each stack i, let  $C^i_{\gamma}$  denote a second order variable which collects all positions where  $\gamma$  is pushed in stack i. Similarly, let  $R^i_{\gamma}$  be a second order variable which collects all positions where  $\gamma$  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 dt-ECMVPA 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

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

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

$$X_{\ell_i}(x) \wedge X_{\ell_i}(y) \wedge Q_a(x) \wedge R_{\gamma}^h(x) \wedge \theta^h(x) \in I \wedge$$

$$\bigwedge_{b \in \Sigma^h} \Big( \Big( \bigwedge_{(x_b \in I) \in \psi} \lhd_b(x) \in I \Big) \land \Big( \bigwedge_{(y_b \in I) \in \psi} \rhd_b(x) \in I \Big) \Big).$$

• Finally, for internal transitions  $(\ell_i, a, \psi, \ell_j) \in \Delta_l^h$  for positions x and y we assert

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

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 three kinds of transitions are given as above. Additionally, we assert that corresponding call and return positions should match, i.e.

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

### B.1 Remaining part: Quasi dt-ECMVPA

A quasi k-dt-ECMVPA is a weaker form of k-dt-ECMVPA where more than one input symbols share the same event clock. Let the finite input alphabet  $\Sigma$  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  $\Sigma$  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 are also determined by the timed word.

**Definition 16 (Quasi k-dt-ECMVPA).** A quasi k-dt-ECMVPA over alphabet  $\Sigma = \{\Sigma_c^i, \Sigma_r^i, \Sigma_l^i\}_{i=1}^n$  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,  $\rho$  is the ranking function,  $\Gamma$  is the stack alphabet and  $F \subseteq L$  is a set of final locations.

Lemma 17. Quasi k-dt-ECMVPA and k-dt-ECMVPA are effectively equivalent.

The proof of Lemma 17 is obtained by using the construction proposed in the proof of Lemma 18 for single stack machines.

Lemma 18 ([7]). The class of q-dtVPA is equally expressive as the class of dtVPA.

Valid homomorphisms for quasi k-dtMVPA Let  $\Sigma = \left\{ \Sigma_c^i, \Sigma_r^i, \Sigma_l^i \right\}_{i=1}^n$  and  $\Pi = \left\{ \Pi_c^i, \Pi_r^i, \Pi_l^i \right\}_{i=1}^n$  be the alphabets of 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  $c \in \Sigma_l^i$  iff  $h(c) \in \Pi_l^i$
- h preserves event clock partition i.e.  $\rho_1(a) = \rho_2(h(a))$  for all  $a \in \Sigma$ .