# A Logical Characterization for Dense-time Visibly Pushdown Automata

Devendra Bhave, Vrunda Dave, Shankara Narayanan Krishna, Ramchandra Phawade, and Ashutosh Trivedi

Indian Institute of Technology Bombay, India
{devendra, vrunda, krishnas, ramchandra, trivedi}@cse.iitb.ac.in

**Abstract.** Two of the most celebrated results that effectively exploit visual representation to give logical characterization and decidable model-checking include visibly pushdown automata (VPA) by Alur and Madhusudan and event-clock automata (ECA) by Alur, Fix and Henzinger. VPA and ECA—by making the call-return edges visible and by making the clock-reset operation visible, respectively—recover decidability for the verification problem for pushdown automata implementation against visibly pushdown automata specification and timed automata implementation against event-clock timed automata specification, respectively. In this work we combine and extend these two works to introduce dense-time visibly pushdown automata that make both the call-return as well as resets visible. We present MSO logic characterization of these automata and prove the decidability of the emptiness problem for these automata paving way for verification problem for dense-timed pushdown automata against dense-timed visibly pushdown automata specification.

**Keywords:** visibly pushdown, event-clock, logical characterization

## 1 Introduction

Timed automata [2] are simple yet powerful generalization of finite automata where a finite set of continuous variables with uniform rates, aptly named clocks, are used to measure critical timing constraints among various events by permitting reset of these clocks to remember occurrence of an event. Due to the carefully crafted dynamics, the emptiness of timed automata is a decidable problem using a technique known as region-construction that computes time-abstract finitary bisimulation of the automata. While, timed automata are closed under union and intersection, they are not closed under complementation and determinization. For this reason it is not possible to verify timed automata implementation against specifications given as timed automata. Event-clock automata [3] are a determinizable subclass of timed automata that enjoy a nice set of closure properties: they are closed under union, intersection, complementation, determinization, and projection. Event-clock automata achieve the closure under determinization by making clock resets visible—the reset of each clock variable is determined by a fixed class of event and hence visible just by looking at the input word. Partially thanks to these closure properties, they are known to be precisely capture timed languages defined by an appropriate class of monadic second-order logic [6].

Recursive timed automata (RTA) [7] and dense-time pushdown automata (dtPDA) [1] are generalization of timed automata that accept certain real-time extensions of context-free languages. In general, the emptiness problem for the RTA in undecidable, however [7] characterizes classes of RTA with decidable emptiness problem. The emptiness problem for the dtPDA is known to be decidable. RTA and dtPDA naturally model the flow of control in time-critical software systems with potentially recursive procedure calls. Alur and Madhusudan [4] argued the need for context-free (representable using pushdown automata) specification while verifying systems modeled as pushdown systems. The goal of this paper is to develop decidable verification framework for RTA and dtPDA by introducing an appropriate class of specification formalism for context-free and time-critical properties that permit decidable verification.

Non-closure under determinization and complementation makes verification against general context-free specification impossible. Alur and Madhusudan [4] introduced *visibly pushdown automata* as a specification formalism where the call and return edges are made visible in a structure of the word. This visibility enabled closure of these automata under determinization and hence complementation, and allowed them to be used in a decidable verification framework. Also, again owing to these closure properties, visibly pushdown automata are known to precisely capture the context-free languages definable by an appropriate class of monadic second order (MSO) logic [4].

In this paper we present dense-time visibly pushdown automata (dtVPA) that form a subclass of dense-time pushdown automata of Abdulla, Atig, and Stenman [1] and generalize both visibly pushdown automata and event-clock automata. We show that dtVPA are determinizable and are closed under Boolean operations (union, intersection, and complementation) as well as projection. We build on these closure properties to give a logical characterization of the timed languages captured by dtVPA.

*Related Work.* Tang and Ogawa in [8] proposed a model called *event-clock visibly pushdown automata* (ECVPA) that generalized both ECA and VPA. For the proposed model they showed determinizability as well as closure under boolean operations, and proved the decidability of the verification problem for timed visibly pushdown automata against such event-clock visibly pushdown automata specifications. However, unlike dtVPAs, ECVPAs do not permit pushing the clocks on the stack and hence dtVPA capture a larger specification class than ECVPA. Moreover [8] did not explore any logical characterization of ECVPA. Our paper builds upon the ideas presented in D'Souza [6] for event-clock automata and Alur and Madhusudan [4] to present a visualized specification framework for dense-time pushdown automata. For the decidability of the emptiness problem, we exploit the recent untiming construction proposed by Clemente and Lasota [5]. For a survey of models related to recursive timed automata and dense-time pushdown automata we refer the reader to [7] and [1].

## 2 Preliminaries

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

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

In order to develop a logical characterization of ECA D'Souza [6] required a class of ECA that is closed under projections. For this purpose, he introduced an equi-expressive generalization of event-clock automata – called quasi-event clock automata (qECA) – where event recorders and predictors are associated with a set of actions rather than a single action. Here, the finite alphabet $\Sigma$ is partitioned into finitely many classes via a ranking function $\rho : \Sigma \to \mathbb{N}$ giving rise to finitely many partitions $P_1, \dots, P_k$ 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 is also determined by the timed word. For example, for a timed word $w = (a_1, t_1), (a_2, t_2), \dots, (a_n, t_n)$, the value of the event clock $x_{\rho(a)}$ at position $j$ is $t_j - t_i$ where $i$ is the largest position preceding $j$ where an action of $P_{\rho(a)}$ has occurred. If no symbols from $P_{\rho(a)}$ have occurred before the $j$th position, then the value of $x_{\rho(a)}$ is undefined denoted by a special symbol $\vdash$. Similarly, he value of $y_{\rho(a)}$ at position $j$ of $w$ is undefined if no symbols of $P_{\rho(a)}$ occur in $w$ after the $j$th position. Otherwise, it is defined as $t_k - t_j$, where $k$ is the first position after $j$ where a symbol of $P_{\rho(a)}$ occurs. We write $C_\rho$ for the set of all event clocks for a ranking function $\rho$ and we use $\mathbb{R}^{\vdash}_{>0}$ for the set $\mathbb{R}_{>0} \cup \{\vdash\}$. Formally, the clock valuation after reading $j$-th prefix of the input timed word $w$, $\nu^w_j : C_\rho \mapsto \mathbb{R}^{\vdash}_{>0}$, is defined in the following fashion: $\nu^w_j(x_q) = t_j - t_i$ if there exists an $0 \leq i < j$ such that $\rho(a_i) = q$ and $a_k \notin P_q$ for all $i < k < j$, otherwise $\nu^w_j(x_q) = \vdash$ (undefined). Similarly, $\nu^w_j(y_q) = t_m - t_j$ if there is $j < m$ such that $\rho(a_m) = q$ and $a_l \notin P_q$ for all $j < l < m$, otherwise $\nu^w_j(y_q) = \vdash$.

A quasi-event clock automaton [6] is a tuple $A = (L, \Sigma, \rho, L^0, F, E)$ where $L$ is a set of finite locations, $\Sigma$ is a finite alphabet, $\rho$ is the alphabet ranking function, $L^0 \in L$ 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 $\varphi$ is a clock constraint over the clocks $C_\rho$. A clock constraint over $C_\rho$ is a boolean combination of constraints of the form $z \sim c$ where $z \in C_\rho$, $c \in \mathbb{N}$ and $\sim \in \{\leq, \geq\}$. Event clock automata are a special kind of quasi-event clock automata when the ranking function $\rho$ is a one-to-one function.

**Theorem 1 (ECA [3, 6]).** *Quasi event-clock automata and event-clock automata are equi-expressive. Quasi event-clock automata are determinizable and closed under Boolean operations, concatenation, Kleene closure, and projection. The language accepted by (quasi) event-clock automata can be characterized by MSO logic over timed words augmented with timed modalities.*

*Visibly Pushdown Automata.* Visibly pushdown automata [4] are a determinizable subclass of pushdown automata that operate over words that dictate the stack operations. This notion is formalized by giving an explicit partition of the alphabet into three disjoint sets of *call*, *return*, and *local* 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 local symbol.

A visibly pushdown alphabet is a tuple $\Sigma = \langle \Sigma_c, \Sigma_r, \Sigma_l \rangle$ where $\Sigma$ is partitioned into a *call* alphabet $\Sigma_c$, a *return* alphabet $\Sigma_r$, and a *local* alphabet $\Sigma_l$. A visibly pushdown automata 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, a finite stack alphabet $\Gamma$ with special end-of-stack symbol $\perp$, and $\Delta \subseteq (L \times \Sigma_c \times L \times (\Gamma \backslash \perp)) \cup (L \times \Sigma_r \times \Gamma \times L) \cup (L \times \Sigma_l \times L)$ and $F \subseteq L$ is final locations.

**Theorem 2 (VPA [4]).** *Visibly pushdown automata are determinizable and closed under Boolean operations, concatenation, Kleene closure, and projection. The language accepted by visibly pushdown automata can be characterized by MSO logic over words augmented with binary matching predicate.*

## 3 Dense-time Visibly Pushdown Automata (dtVPA)

We introduce the dense-time visibly pushdown automata as an event-clock automaton equipped with a timed stack along with visibly pushdown alphabet $\Sigma = \langle \Sigma_c, \Sigma_r, \Sigma_l \rangle$. For notational convenience, we assume that the partitioning function is one-to-one, i.e. each symbol $a \in \Sigma$ has unique recorder $x_a$ and predictor $y_a$ clocks assigned to it. This permits us to drop the ranking function $\rho$ for the further discussion. Let $C_\Sigma$ (or $C$ when $\Sigma$ is clear) be a finite set of event clocks. Let $\Phi(C)$ be the set of *clock constraints* over $C$ and $\mathcal{I}$ be the set of intervals of the form $\langle a, b \rangle$ with $a \in \mathbb{N}$, $a \leq b$ and $b \in \mathbb{N} \cup \{\infty\}$.

*Syntax.* A dense-time visibly pushdown automata over $\Sigma = \{\Sigma_c, \Sigma_r, \Sigma_l\}$ is a tuple $M = (L, \Sigma, \Gamma, L^0, F, \Delta = \Delta_c \cup \Delta_r \cup \Delta_l)$, 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 $\perp$, $\Delta_c = (L \times \Sigma_c \times \Phi(C) \times L \times (\Gamma \backslash \perp))$ is the set of call transitions, $\Delta_r = (L \times \Sigma_r \times \mathcal{I} \times \Gamma \times \Phi(C) \times L)$ is the set of return transitions, $\Delta_l = (L \times \Sigma_l \times \Phi(C) \times L)$ is the set of local transitions, and $F \subseteq L$ is a set of final locations.

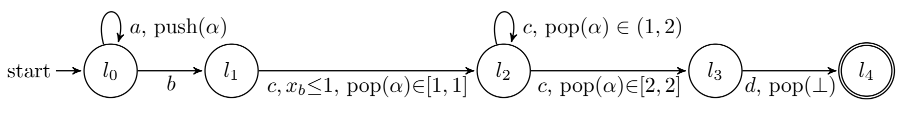
*Semantics.* Let $w = (a_0, t_0), \ldots, (a_n, t_n)$ be a timed word. A configuration of the dtVPA is a tuple $(\ell, \nu_i^w, (\gamma\sigma, age(\gamma\sigma)))$ where $\ell$ is the current location of the dtVPA, $\nu_i^w$ gives the valuation of all the event clocks at position $i \leq |w|$, $\gamma\sigma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

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

– If $a_i \in \Sigma_c$, then there is a transition $(\ell_i, a_i, \varphi, \ell_{i+1}, \gamma) \in \Delta$ s.t. $\nu_i^w \models \varphi$. The symbol $\gamma \in \Gamma \backslash \{\bot\}$ is then pushed onto the stack, and its age is initialized to zero, obtaining $(\sigma_{i+1}, age(\sigma_{i+1})) = (\gamma :: \sigma_i, 0 :: (age(\sigma_i) + (t_i - t_{i-1})))$. Note that all symbols in the stack excluding the topmost age by $t_i - t_{i-1}$.
– If $a_i \in \Sigma_r$, then there is a transition $(\ell_i, a_i, I, \gamma, \varphi_i, \ell_{i+1}) \in \Delta$. The configuration $(\ell_i, \nu_i, (\sigma_i, age(\sigma_i)))$ evolves to $(\ell_{i+1}, \nu_{i+1}, (\sigma_{i+1}, age(\sigma_{i+1})))$ iff $\nu_i^w \models \varphi_i$, $\sigma_i = \gamma :: \kappa \in \Gamma\Gamma^*$ and $age(\gamma) + (t_i - t_{i-1}) \in I$. Then we obtain $\sigma_{i+1} = \kappa$, with $age(\sigma_{i+1}) = age(\kappa) + (t_i - t_{i-1})$. However, if $\gamma = \langle \bot \rangle$, the symbol is not popped, and the attached interval $I$ is irrelevant.
– If $a_i \in \Sigma_l$, then there is a transition $(\ell_i, a_i, \varphi_i, \ell_{i+1}) \in \Delta$ such that $\nu_i^w \vDash \varphi_i$. In this case stack remains unchanged i.e. $\sigma_i = \sigma_{i+1}$, and $age(\sigma_{i+1}) = age(\sigma_i) + (t_i - t_{i-1})$. All symbols in the stack age by $t_i - t_{i-1}$.

A run $\rho$ of a dtVPA $M$ 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 dtVPA $M$, is the set of all timed words $w$ accepted by $M$.

*Example 3.* Consider the timed languages of the form $a^n b c^n d$ where the first $c$ comes precisely 1 time-unit after last $a$ and the first $a$ and the last $c$ are 2 time-units apart, and every other matching $a$ and $c$ are within $(1, 2)$ time-unit apart, i.e. $\{(a^n b c^n d, \langle t_1, \ldots, t_n, t, t'_n, \ldots, t'_1, t' \rangle) \mid t'_n - t_n = 1, t'_1 - t_1 = 2, t'_i - t_i \in (1, 2) \text{ for all } i \leq n\}$. Given a partition $\Sigma_c = \{a\}$, $\Sigma_l = \{b, d\}$, $\Sigma_r = \{c\}$ and $\Gamma = \{\alpha\}$ this language can be accepted by the dtVPA shown in below.


start
$l_0$
$a$, push($\alpha$)
$b$
$l_1$
$c, x_b \leq 1$, pop($\alpha$)$\in[1,1]$
$l_2$
$c$, pop($\alpha$) $\in (1,2)$
$c$, pop($\alpha$)$\in[2,2]$
$l_3$
$d$, pop($\bot$)
$l_4$

Here $l_0$ is the initial location and $l_4$ is only accepting location. The transitions relation contains the following transitions: the call transition $(l_0, a, \texttt{true}, l_0, \alpha) \in \Delta_c$, the local transition $(l_0, b, \texttt{true}, l_1) \in \Delta_l$ and the following set of return transitions $(l_1, c, [1, 1], \alpha, x_b \leq 1, l_2)$, $(l_2, c, (1, 2), \alpha, \texttt{true}, l_2)$, $(l_2, c, [2, 2], \alpha, \texttt{true}, l_3)$,

$(l_3, d, \texttt{true}, \perp, \texttt{true}, l_4) \in \Delta_r$. In the figure we have not shown clock constraints that evaluate to $\texttt{true}$ and we have depicted testing the age of the top symbol as $pop(\cdot) \in I$.

*Deterministic dtVPA.* A dtVPA $M = (L, \Sigma, L^0, F, \Delta)$ is said to be *deterministic* if it has exactly one start location, and for every configuration and input action exactly one transition is enabled. Formally, we have the following conditions: for every $(\ell, a, \phi_1, \ell', \gamma_1), (\ell, a, \phi_2, \ell'', \gamma_2) \in \Delta_c$, $\phi_1 \wedge \phi_2$ is unsatisfiable; for every $(\ell, a, I_1, \gamma, \phi_1, \ell'), (\ell, a, I_2, \gamma, \phi_2, \ell'') \in \Delta_r$, either $\phi_1 \wedge \phi_2$ is unsatisfiable or $I_1 \cap I_2 = \emptyset$; and for every $(\ell, a, \phi_1, \ell'), (\ell, a, \phi_2, \ell') \in \Delta_l$, $\phi_1 \wedge \phi_2$ is unsatisfiable.
The following is one of the central of the paper.

**Theorem 4 (Determinizability, Emptiness and Closure).** *Dense-time visibly pushdown automata are determinizable and closed under Boolean operations, concatenation, Kleene closure and projection. Their emptiness is also decidable.*

The proofs for the union, intersection, concatenation, and Kleene closure are straightforward extensions of the closure of visibly pushdown automata and event-clock automata under these operations. The proof for the determinizability (and hence the complementation) is slightly more involved. In the next section we present a proof for the determinizability as well as decidability of the emptiness problem for dtVPA. Section 5 presents a logical characterization of dtVPA.

## 4 Untiming the Stack in dtVPA

Event-clock visibly-pushdown automata (ECVPA) [8] can be considered as subclasses of dtVPA where the ages are not pushed on the stack. Hence a dtVPA $M = (L, \Sigma, L^0, F, \Delta)$ is an ECVPA if for every $(\ell, a, I, \gamma, \phi, \ell') \in \Delta_r$ we have that $I = [-\infty, +\infty]$. Tang and Ogawa, in [8], proved the following for ECVPA.

**Theorem 5.** *ECVPAs are determinizable and closed under Boolean operations.*

We now describe the *untiming-the-stack* construction to obtain from a dtVPA $M$ over $\Sigma$, an ECVPA $M'$ 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 [5]. However, [5] cannot directly be used here since [5] introduces extra clocks that require resets which is not available under event-clock restriction.

Given a dtVPA $M$, let $k$ be the maximum constant used in any interval $I$ to check the age of a popped symbol. We first explain the proof idea. Consider the first call transition $(l, a, \varphi, l', \gamma)$ encountered in $M$. To construct an ECVPA $M'$ from $M$, we guess the interval used in a constraint in return transition when $\gamma$ will be popped from the 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 the location $(l'_{a,<\kappa}, \{<\kappa\})$, and the symbol $(\gamma, <\kappa, \texttt{first})$ is pushed onto the stack.

Let $Z_k^{\sim} = \{\sim c \mid c \in \mathbb{N}, c \leq k, \sim \in \{<, \leq, >, \geq, =\}\}$. Then the extended alphabet $\Sigma' = \Sigma \cup Z_k^{\sim}$. All symbols of $Z_k^{\sim}$ are local symbols in $M'$. $\Sigma' = (\Sigma_c, \Sigma_l \cup Z_k^{\sim}, \Sigma_r)$. At $(l'_{a,<\kappa}, \{<\kappa\})$, the new symbol $<\kappa$ is read and we have the following transition : $((l'_{a,<\kappa}, \{<\kappa\}), <\kappa, x_a = 0, (l', \{<\kappa\}))$, which results in resetting the event recorder $x_{<\kappa}$ corresponding to the new symbol $<\kappa$. The constraint $x_a = 0$ ensures that no time is elapsed by the new transition. The information $<\kappa$ is retained in the control state until $(\gamma, <\kappa, \texttt{first})$ is popped. At $(l', \{<\kappa\}))$, we continue the simulation of $M$ from $l'$. Assume that we have another push operation at $l'$ of the form $(l', b, \psi, q, \beta)$. In $M'$, from $(l', \{<\kappa\})$, we first guess the constraint that will be checked when $\beta$ will be popped from the stack. If the guessed constraint is again $<\kappa$, then control switches from $(l', \{<\kappa\})$ to $(q, \{<\kappa\})$, and $(\beta, <\kappa, -)$ is pushed onto the stack and simulation continues from $(q, \{<\kappa\})$. However, if the guessed pop constraint is $<\zeta$ for $\zeta \neq \kappa$, then control switches from $(l', \{<\kappa\})$ to $(q_{b,<\zeta}, \{<\kappa, <\zeta\})$. The new obligation $<\zeta$ is also remembered in the control state. From $(q_{b,<\zeta}, \{<\kappa, <\zeta\})$, we read the new symbol $<\zeta$ which resets the event predictor $x_{<\zeta}$ and control switches to $(q, \{<\kappa, <\zeta\})$, pushing $(\beta, <\zeta, \texttt{first})$ on to the stack. The idea thus is to keep the obligation $<\kappa$ alive in the control state until $\gamma$ is popped; the value of $x_{<\kappa}$ at the time of the pop determines whether the pop is successful or not. If a further $<\kappa$ constraint is encountered while the obligation $<\kappa$ is already alive, then we do not reset the event clock $x_{<\kappa}$. The $x_{<\kappa}$ is reset only at the next call transition after $(\gamma, <\kappa, \texttt{first})$ is popped, when $<\kappa$ is again guessed.

The case when the guessed popped constraint is of the form $>\kappa$ is similar. In this case, each time the guess is made, we reset the event recorder $x_{>\kappa}$ 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. Handling guesses of the form $\geq \zeta \wedge \leq \kappa$ is similar, and we combine the ideas discussed above.

Now consider a return transition $(l, a, I, \gamma, \varphi, l')$ in $M$. In $M'$, we are at some control state $(l, P)$. On reading $a$, we check the top of stack symbol in $M'$. It is of the form $(\gamma, S, \texttt{first})$ or $(\gamma, S, -)$, where $S$ is either a singleton set of the form $\{<\kappa\}$ or $\{>\zeta\}$, or a set of the form $\{<\kappa, >\zeta\}$. Consider the case when the top of stack symbol is $(\gamma, \{<\kappa, >\zeta\}, \texttt{first})$. In $M'$, on reading $a$, the control switches from $(l, P)$ to $(l', P')$ for $P' = P \backslash \{<\kappa\}$ iff

1. The guard $\varphi$ evaluates to true.
2. the interval $I$ is $(\zeta, \kappa)$. This validates our guess made at the time of push.
3. The value of clock $x_{<\kappa}$ is $<\kappa$, and the value of clock $x_{>\zeta}$ is $>\zeta$.
4. Note that the third component *first* says that there aren't any symbols in the stack below $(\gamma, \{<\kappa, >\zeta\}, \texttt{first})$ whose pop constraint is $<\kappa$. Hence, we can remove the obligation $<\kappa$ from $P$ in the control state.
5. If the top of stack symbol was $(\gamma, \{<\kappa, >\zeta\}, -)$, then we know that the pop constraint $<\kappa$ is still alive. That is, there is some stack symbol below $(\gamma, \{<\kappa, >\zeta\}, -)$ of the form $(\beta, S, \texttt{first})$ such that $<\kappa \in S$. In this case, we keep $P$ unchanged and control switches to $(l', P)$.

To formalize the construction, given $M = (L, \Sigma, \Gamma, L^0, F, \Delta)$ with max constant $k$ used in return transitions, we construct $M' = (L', \Sigma', \Gamma', L'^0, L'^f, \delta')$ where $L' = (L \times 2^{Z_k^\sim}) \cup (L_{\Sigma \times Z_k^\sim} \times 2^{Z_k^\sim}) \cup (L_{\Sigma \times Z_k^\sim \times Z_k^\sim} \times 2^{Z_k^\sim})$, $\Sigma' = (\Sigma_c, \Sigma_l \cup Z_k^\sim, \Sigma_r)$, $\Gamma' = \Gamma \times 2^{Z_k^\sim} \times \{\texttt{first}, -\}$, $L^0 = \{(l^0, \emptyset) \mid l^0 \in L^0\}$, and $F = \{(l^f, \emptyset) \mid l^f \in F\}$. The transitions $\Delta'$ are defined below. Let $(l, a, \varphi, l', \gamma) \in \Delta_c$. Then we have the following transitions in $M'$.

1. $((l, P), a, \varphi, (l', P), (\gamma, \{<\kappa\}, -)) \in \Delta'_c$ if $<\kappa \in P$
2. $((l, P), a, \varphi, (l'_{a,<\kappa}, P'), (\gamma, \{<\kappa\}, \texttt{first})) \in \Delta'_c$ if $<\kappa \notin P$. $P' = P \cup \{<\kappa\}$.
3. $((l'_{a,<\kappa}, P'), <\kappa, x_a = 0, (l', P')) \in \Delta'_l$.
   These transitions consider the guessed pop constraint as $<\kappa$. In the first case, $<\kappa$ is alive, and hence there is no need to reset the clock $x_{<\kappa}$. In the second case, the obligation $<\kappa$ is fresh. Hence it is remembered as *first* in the stack, and the clock $x_{<\kappa}$ is reset.
4. $((l, P), a, \varphi, (l'_{a,>\kappa}, P), (\gamma, \{>\kappa\}, -)) \in \Delta'_c$.
5. $((l'_{a,>\kappa}, P), >\kappa, x_a = 0, (l', P)) \in \Delta'_l$.
   These transitions consider the case when the guessed pop constraint is $>\kappa$. The clock $x_{>\kappa}$ is reset, and the obligation is remembered in the stack.
6. $((l, P), a, \varphi, (l'_{a,<\kappa,>\zeta}, P'), (\gamma, \{<\kappa, >\zeta\}, \texttt{first})) \in \Delta'_c$ if $<\kappa \notin P$. $P' = P \cup \{<\kappa, >\zeta\}$.
7. $((l'_{a,<\kappa,>\zeta}, P'), >\zeta, x_a = 0, (l'_{a,<\kappa}, P')) \in \Delta'_l$.
8. $((l, P), a, \varphi, (l'_{a,>\zeta}, P), (\gamma, \{<\kappa, >\zeta\}, -)) \in \Delta'_c$ if $<\kappa \in P$.
   These 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 $x_{>\zeta}$ and remember both the obligations in the stack. If $<\kappa$ is fresh, then we reset both clocks $x_{>\zeta}$ and $x_{<\kappa}$ and remember both obligations in the stack, and $<\kappa$ in the control state.

Let $(l, a, \varphi, l') \in \Delta_l$. Then we have the transitions $((l, P), a, \varphi, (l', P)) \in \Delta'_l$. Next we consider return transitions. Let $(l, a, I, \gamma, \varphi, l') \in \Delta_r$. Then we have the following transitions in $\Delta'_r$.

1. $((l, P), a, (\gamma, \{<\kappa, >\zeta\}, -), \varphi \wedge x_{<\kappa} < \kappa \wedge x_{>\zeta} > \zeta, (l', P))$ if $I = (\zeta, \kappa)$.
2. $((l, P), a, (\gamma, \{<\kappa, >\zeta\}, \texttt{first}), \varphi \wedge x_{<\kappa} < \kappa \wedge x_{>\zeta} > \zeta, (l', P'))$
   where $P' = P \backslash \{<\kappa\}$, if $I = (\zeta, \kappa)$.
3. $((l, P), a, (\gamma, \{<\kappa\}, -), \varphi \wedge x_{<\kappa} < \kappa, (l', P))$ if $I = [0, \kappa)$.
4. $((l, P), a, (\gamma, \{<\kappa\}, \texttt{first}), \varphi \wedge x_{<\kappa} < \kappa, (l', P'))$ where $P' = P \backslash \{<\kappa\}$, if $I = [0, \kappa)$.
5. $((l, P), a, (\gamma, \{>\zeta\}, -), \varphi \wedge x_{>\zeta} > \zeta, (l', P))$ 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 $P \subseteq Z_k^\sim$. For any $w = (a_1, t_1) \dots (a_n, t_n) \in L(M)$, we have $w' = (a_1, t_1)T_1(a_2, t_2)T_2 \dots (a_n t_n)T_n$ accepted by $L(M')$, where for $1 \leq l \leq n$, $|T_l| \leq 2k$, and $T_l$ is a timed word $(b_1, t_l) \dots (b_j, t_l)$ where $j \leq 2k$ and $b_i \in Z_k^\sim$ for $1 \leq i \leq j$ and the only time stamp used in $T_i$ is $t_i$, since no time elapses in $M'$ while remembering obligations and resetting the appropriate clocks.

*Emptiness and Determinizability.* 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 $L(M) \neq \emptyset$. Since $M'$ is ECVPA, we can apply the standard region construction of event clock automata [3] to obtain a PDA preserving emptiness.

Next, we focus on the determinizability problem for dtVPA. We start with a dtVPA $M = (L, \Sigma, \Gamma, L^0, F, \Delta)$ with $n$ locations, with max constant $k$ used in the intervals of return transitions. Let $\chi = max(|\Sigma|, |Z_k^{\sim}|)$. The deterministic dtVPA $M''$ corresponding to a dtVPA $M$ is obtained as follows:
(1) Obtain a ECVPA $M'$ as seen in section 4. Note that $L(M) = h(L(M'))$, and $M' = (L', \Sigma, \Gamma', L'^0, F', \Delta')$ with $|L'| \leq \chi^3.2^{|Z_k^{\sim}|}$. Let $\pi = \chi^3.2^{|Z_k^{\sim}|}$.
(2) Use [8] to obtain a deterministic ECVPA $Det(M')$. This construction uses the determinization of VPA [4], which results in $O(2^{\pi^2})$ locations in $Det(M')$.
(3) Construct the deterministic dtVPA $M''$ from $Det(M')$ as explained below. No blow up in the locations and transitions are incurred in this construction. We thus obtain a deterministic dtVPA $M''$ with at most $2^{\pi^2}$ locations. The construction of $M''$ is explained below.

Consider a dtVPA $M = (L, \Sigma, \Gamma, L^0, F, \Delta)$ and the corresponding ECVPA $M' = (L', \Sigma', \Gamma', L'^0, F', \Delta')$ as constructed in section 4. From Theorem 5 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 $Z_k^{\sim}$ ($\Sigma' = \Sigma \cup Z_k^{\sim}$) and (i) no time elapse happens on reading these symbols, and (ii) no stack operations happen on reading these symbols. 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 $\alpha \in Z_k^{\sim}$. In the following, we eliminate these transitions on $Z_k^{\sim}$ preserving the language accepted by $M$ and the determinism of $det(M')$. In doing so, we will construct a dtVPA $M''$ which is deterministic, and which preserves the language of $M$. We now analyze various types for $\alpha \in Z_k^{\sim}$.

Assume that $\alpha$ is of the form $>\zeta$. Let $(s_1, \alpha, \varphi, s_2)$ be the unique transition on $\alpha \in Z_k^{\sim}$. By construction of $M'$ (and hence $det(M')$), we know that $\varphi$ has the form $x_a = 0$ for some $a \in \Sigma$. 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 the stack, we can combine the two transitions from $s_0$ to $s_1$ and $s_1$ to $s_2$ to obtain the call transition $(s_0, a, \psi, s_2, (\gamma, \alpha, -))$. This eliminates transition on $>\zeta$.

Assume that $\alpha$ is of the form $<\kappa$. Let $(s_1, \alpha, \varphi, s_2)$ be the unique transition on $\alpha \in Z_k^{\sim}$. We know that $\varphi$ has the form $x_a = 0$ for some $a \in \Sigma$. From $M'$, we also know that in $Det(M')$, there is a unique transition of one of the following forms preceding $(s_1, \alpha, \varphi, s_2)$.

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

Since $(s_1, \alpha, \varphi, s_2)$ is a no time elapse transition, and does not touch the stack, we can combine the two transitions from $s_0$ to $s_1$ (cases (a), (b)) and $s_1$ to $s_2$

to obtain the call transition $(s_0, a, \psi, s_2, (\gamma, \alpha, -))$ or $(s_0, a, \psi, s_2, (\gamma, \alpha, \texttt{first}))$. This eliminates the transition on $<\kappa$. In case of transition (c), we first eliminate the local transition on $>\zeta$ obtaining $(s'_0, a, \psi, s_1, (\gamma, \{\alpha, >\zeta\}, X))$ and then obtain the call transitions $(s'_0, a, \psi, s_2, (\gamma, \{\alpha, >\zeta\}, X))$. We have thus eliminated local transitions on $<\kappa$.

Merging transitions as done here does not affect transitions on $\Sigma$; they simply eliminate the newly added transitions on $\Sigma' - \Sigma$. Recall that checking constraints on these clocks were required on popping the stack. 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 $C_\Sigma$ in $M$ during pop.

- $(s, a, (\gamma, \{<\kappa\}, X), \varphi \wedge x_{<\kappa}<\kappa, s')$, for $X \in \{-, \texttt{first}\}$. This is modified to $(s, a, [0, \kappa), (\gamma, \{<\kappa\}, X), \varphi, s')$.
- $(s, a, (\gamma, \{<\kappa, >\zeta\}, X), \varphi \wedge x_{>\zeta}>\zeta \wedge x_{<\kappa}<\kappa, s')$ for $X \in \{-, \texttt{first}\}$. This is modified to $(s, a, (\zeta, \kappa), (\gamma, \{<\kappa, >\zeta\}, X), \varphi, s')$.
- $(s, a, (\gamma, \{>\zeta\}, -), \varphi \wedge x_{>\zeta}>\zeta, s')$. This is modified to $(s, a, (\zeta, \infty), (\gamma, \{>\zeta\}, -), \varphi, s')$.

Thus, we obtain a deterministic dtVPA $M''$ from $det(M')$ such that $L(M'') = L(M)$ and $h(L(M'')) = L(det(M'))$.

## 5 Logical Characterization of dtVPA

*Monadic Second-Order Logic on Timed Words.* We consider a timed word $w = (a_0, t_0), (a_1, t_1), \ldots, (a_m, t_m)$ over $\Sigma$ as a *word structure* over the universe $U = \{1, 2, \ldots, |w|\}$ of positions in the timed word. The predicates in the word structure are $Q_a(i)$ which evaluates to true at position $i$ iff $w[i] = a$, where $w[i]$ denotes the $i$th position of $w$. Following [4], we use the matching binary relation $\mu(i, j)$ which evaluates to true iff the $i$th position is a call and the $j$th position is its matching return. We also introduce three predicates $\lhd_a$, $\rhd_a$, and $\theta$ capturing the following relations. For an interval $I$, the predicate $\lhd_a(i) \in I$ evaluates to true on the word structure iff $\nu_i^w(x_a) \in I$ for recorder clock $x_a$. For an interval $I$, the predicate $\rhd_a(i) \in I$ evaluates to true on the word structure iff $\nu_i^w(y_a) \in I$ for predictor clock $y_a$. For an interval $I$, the predicate $\theta(i) \in I$ evaluates to true on the word structure iff $w[i] \in \Sigma_r$, and there is some $k < i$ such that $\mu(k, i)$ evaluates to true and $t_i - t_k \in I$. The predicate $\theta(i)$ measures the time elapse between position $k$ where a call was made, and position $i$, its matching return. This time elapse is the age of the symbol pushed on to the stack during the call at position $k$. Since position $i$ is the matching return, this symbol is popped at position $i$; if the age lies in the interval $I$, the predicate evaluates to true. We define MSO($\Sigma$), the MSO logic over $\Sigma$, as:

$$\varphi := Q_a(x) \mid x{\in}X \mid \mu(x,y) \mid \lhd_a(x){\in}I \mid \rhd_a(x){\in}I \mid \theta(x){\in}I \mid \neg\varphi \mid \varphi{\vee}\varphi \mid \exists\, x.\varphi \mid \exists\, X.\varphi$$

where $a{\in}\Sigma$, $x_a{\in}C_\Sigma$, $x$ is a first order variable and $X$ is a second order variable. The models of a formula $\phi \in$ MSO($\Sigma$) are timed words $w$ over $\Sigma$. The semantics of these logic is standard where first order variables are interpreted over positions of $w$ and second order variables over subsets of positions. As an example consider the formula $\varphi = \forall x(Q_a(x) \to \exists y[Q_c(y) \wedge \theta(y) \in (1,2)])$ over $\Sigma = (\{a\},\{b\},\{c\}))$. It expresses that for every $a \in \Sigma_c$, there exists a $c \in \Sigma_r$ as the matching return such that the time elapse between the call and return is in the interval $(1,2)$. The word $w = (a,0)(a,0.2)(b,0.5)(c,1.3)(b,1.7)(b,1.9)(c,1.99)$ satisfies $\varphi$. We define the language $L(\varphi)$ of an MSO sentence $\varphi$ as the set of all words satisfying $\varphi$.

*Logic to automata.* We first show that for any MSO formula $\varphi$ over $\Sigma = (\Sigma_c, \Sigma_l, \Sigma_r)$, $L(\varphi)$ is accepted by a dtVPA. Let $Z = (x_1, \dots, x_m, X_1, \dots, X_n)$ be the free variables in $\varphi$. We work on the extended alphabet $\Sigma' = (\Sigma'_c, \Sigma'_l, \Sigma'_r)$ where $\Sigma'_s{=}\Sigma_s{\times}(Val : Z \to \{0,1\}^{m+n})$, 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 $\Sigma'$ consists of all symbols $(a,v)$ where $a \in \Sigma$ 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. We partition $\Sigma'$ such that for a fixed $a \in \Sigma$, all symbols of the form $(a, d_1, \dots, d_{m+n})$ 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 dtVPA. The cases $Q_a(x), \mu(x,y)$ are exactly as in [4]. We only discuss the new predicates here.

Consider the atomic formula $\lhd_a(x) \in I$. We construct a dtVPA that on reading a symbol $(b,v) \in \Sigma'$ with $v(x) = 1$ checks the constraint $x_a \in I$ for acceptance. The case of $\rhd_a(x) \in I$ is similar, and the check is done on clock $y_a$. Consider the atomic formula $\theta(x) \in I$. To handle this, we build a dtVPA that keeps pushing symbols $(a,v)$ onto the stack whenever $a \in \Sigma_c$, initializing the age to 0 on push. It keeps popping the stack on reading return symbols $(a', v')$, 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 dtVPA. Existential quantifications correspond to projection by excluding the chosen variable from the valuation and renaming the alphabet $\Sigma'$. Let $M$ be an dtVPA constructed for $\varphi(x_1, \dots, x_n, X_1, \dots, X_m)$ over $\Sigma'$. Consider $\exists x_i.\varphi(x_1, \dots, x_n, X_1, \dots, X_m)$ for some first order variable $x_i$. Let $Z_i = (x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n, X_1, \dots, X_m)$ by removing $x_i$ from $Z$. We simply work on the alphabet $\Sigma'_i = \Sigma \times (Val : Z_i \to \{0,1\}^{m+n-1})$. Note that $\Sigma'_i$ is partitioned exactly in the same way as $\Sigma'$. For a fixed $a \in \Sigma$, all symbols $(a, d_1, \dots, d_{m+n-1})$ for $d_i \in \{0,1\}$ lie in the same partition. Thus, $\Sigma'$ and $\Sigma'_i$ have exactly the same number of partitions, namely $|\Sigma|$. Thus, an event clock $x_a = x_{(a,d_1,\dots,d_{m+n})}$ used in $M$ can be used the same way while constructing the automaton for $\exists x_i.\varphi(x_1, \dots, x_n, X_1, \dots, X_m)$. 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 dtVPA that accepts $L(\psi)$ when $\psi$ is an MSO sentence.

*Automata to logic.* Consider a dtVPA $M = (L, \Sigma, \Gamma, L^0, F, \Delta)$. Let $L = \{l_1, \ldots l_n\}$ and $\Gamma = \{\gamma_1, \ldots, \gamma_m\}$. The MSO formula encoding accepting runs of dtVPA is: $\exists X_{l_1} \ldots X_{l_n} C_{\gamma_1} \ldots C_{\gamma_m} R_{\gamma_1} \ldots R_{\gamma_m}\, \varphi(X_{l_1}, \ldots, X_{l_n}, C_{\gamma_1}, \ldots, C_{\gamma_m}, R_{\gamma_1}, \ldots, R_{\gamma_m})$, where $X_q$ denotes the set of positions in the word where the run is in location $q$, $C_\gamma, R_\gamma$ stand for the set of positions in the run where $\gamma$ is pushed and popped from the stack respectively. 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 $(\ell_i, a, \psi, \ell_j, \gamma) \in \Delta_c$, for positions $x, y$, we assert that $X_{\ell_i}(x) \wedge X_{\ell_j}(y) \wedge Q_a(x) \wedge C_\gamma(x) \wedge \bigwedge_{b \in \Sigma} \Big( \big( \bigwedge_{(x_b \in I) \in \psi} \lhd_b(x) \in I \big) \wedge \big( \bigwedge_{(y_b \in I) \in \psi} \rhd_b(x) \in I \big) \Big)$. For return transitions $(\ell_i, a, I, \gamma, \psi, \ell_j) \in \Delta_r$ for positions $x$ and $y$ we assert that $X_{\ell_i}(x) \wedge X_{\ell_j}(y) \wedge Q_a(x) \wedge R_\gamma(x) \wedge \theta(x) \in I \wedge \bigwedge_{b \in \Sigma} \Big( \big( \bigwedge_{(x_b \in I) \in \psi} \lhd_b(x) \in I \big) \wedge \big( \bigwedge_{(y_b \in I) \in \psi} \rhd_b(x) \in I \big) \Big)$. Finally, for local transitions $(\ell_i, a, \psi, \ell_j) \in \Delta_l$ for positions $x$ and $y$ we assert $X_{\ell_i}(x) \wedge X_{\ell_j}(y) \wedge Q_a(x) \wedge \bigwedge_{b \in \Sigma} \Big( \big( \bigwedge_{(x_b \in I) \in \psi} \lhd_b(x) \in I \big) \wedge \big( \bigwedge_{(y_b \in I) \in \psi} \rhd_b(x) \in I \big) \Big)$. We also assert that the last position of the word belongs to some $X_l$ such that there is a transition (call, return,local) from $l$ to an accepting location. The encoding of all 3 kinds of transitions is as above. Additionally, we assert that corresponding call and return positions should match, i.e. $\forall x \forall y\, \mu(x, y) \Rightarrow \bigvee_{\gamma \in \Gamma \setminus \bot} C_\gamma(x) \wedge R_\gamma(y)$.

These two parts together finish the proof of the main result of the paper.

**Theorem 6.** *A language $L$ over $\Sigma$ is accepted by an dtVPA iff there is a MSO sentence $\varphi$ over $\Sigma$ such that $L(\varphi) = L$.*

## References

1. Abdulla, P., Atig, M., Stenman, J.: Dense-timed pushdown automata. In: LICS. pp. 35–44 (2012)
2. Alur, R., Dill, D.: A theory of timed automata. Theoretical Computer Science 126, 183–235 (1994)
3. Alur, R., Fix, L., Henzinger, T.A.: Event-clock automata: A determinizable class of timed automata. TCS 211(1-2), 253–273 (1999)
4. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: Symposium on Theory of Computing. pp. 202–211 (2004)
5. Clemente, L., Lasota, S.: Timed pushdown automata revisited. In: LICS. pp. 738–749 (2015)
6. D'Souza, D.: A logical characterisation of event clock automata. Int. J. Found. Comput. Sci. 14(4), 625–640 (2003), `http://dx.doi.org/10.1142/S0129054103001923`
7. Trivedi, A., Wojtczak, D.: Recursive timed automata. In: ATVA. LNCS, vol. 6252, pp. 306–324. Springer-Verlag (September 2010)
8. Van Tang, N., Ogawa, M.: Event-clock visibly pushdown automata. In: SOFSEM 2009, LNCS, vol. 5404, pp. 558–569. Springer (2009)