Decision Problems for the Verification of Real-Time Software*

Michael Emmi and Rupak Majumdar

University of California, Los Angeles
{mje,rupak}@cs.ucla.edu

Abstract. We study two questions in the theory of timed automata concerning timed language inclusion of real-time programs modeled as timed pushdown automata in real-time specifications with just one clock. We show that if the specification B is modeled as a timed automaton with one clock, then the language inclusion problem $L(A) \subseteq L(B)$ for a timed pushdown automaton A is decidable. On the other hand, we show that the universality problem of timed visibly pushdown automata with only one clock is undecidable. Thus there is no algorithm to check language inclusion of real-time programs for specifications given by visibly pushdown specifications with just one clock.

1 Introduction

Timed automata [4] are a standard modeling formalism for real-time systems. Alur and Dill [4] showed the untimed reachability problem for timed automata is decidable. However, the universality problem (whether all timed traces are accepted) is undecidable, and therefore the timed language inclusion problem (whether all finite timed traces accepted by A are also accepted by B) is also undecidable. These bounds were recently tightened. The timed language inclusion problem $L(A) \subseteq L(B)$ for timed automata A and B *is* decidable if B has at most one clock [13], while the proof of [4] shows two clocks are sufficient for the universality problem to become undecidable. On the other hand, over infinite timed words, one clock is enough to make the language inclusion problem undecidable [2].

When verifying real-time software, the basic model of timed automata must be augmented by a program stack to model procedure calls. In this case, the model is a timed pushdown automaton: a timed automaton augmented with a stack. Untimed reachability is decidable for timed pushdown automata [8], in fact, the binary reachability relation for timed pushdown automata is also decidable [9]. Since the timed language inclusion problem $L(A) \subseteq L(B)$ is undecidable for timed automata if B has more than one clock, one remaining open question is when A is a timed pushdown automaton and B is a timed automaton that has exactly one clock (if B has no clocks, then the question is decidable by a

* This research was supported in part by the grants NSF CCR-0427202 and NSF CNS-0541606.

reduction to reachability, using closure properties of finite automata). We show that this problem is decidable, by extending the proof of [13]. The question is not just of theoretical interest. Many network protocol specifications can be modeled as timed automata with just one clock, and their software implementations are usually modeled as timed pushdown automata [12].

The main technical content of our proof is a generic decidability result for well-structured infinite software systems that is of independent interest in software verification. A large class of infinite state systems have been shown decidable using *well quasi-ordering* relations on the state space [1, 3, 10]. However, these formalisms are not immediately applicable to software verification, where a program is organized into procedures with possibly recursive calls. For software with a *finite* data state space, the standard technique to compute the set of reachable states is context free reachability [14, 7]. Our result unifies the two worlds by providing a context free reachability algorithm for infinite data state spaces, whose termination is proved using well quasi-ordering relations of [1, 10].

What if we extend the expressive power of the specification formalism beyond timed automata? The universality (and so language inclusion) problem for timed pushdown automata is undecidable, since the corresponding problems are already undecidable for (untimed) pushdown automata. We must therefore contend ourselves with formalisms of lesser expressive power. One such candidate is *visibly* pushdown automata [6] —where the stack pushes and pops are determined explicitly by the input alphabet. (Untimed) visibly pushdown automata are already sufficient to specify many interesting properties of software systems [6, 5]. Moreover, they have the nice decidability properties akin to regular languages: for example, universality and language inclusion problems are decidable for (untimed) visibly pushdown automata, and one can hope for similar decidability results in the timed case. We therefore study the universality problem for *timed* visibly pushdown automata with one clock. Unfortunately, we exhibit that this problem (and hence the language inclusion problem even when there is exactly one clock) is undecidable, thus precluding algorithmic solutions to the problem.

Our undecidability result encodes the operation of two counter machines using a timed visibly pushdown automaton with exactly one clock. We cannot directly apply the undecidability proof for the universality of pushdown automata, since the standard proof [11] is not *visibly* pushdown (indeed, universality is decidable for visibly pushdown automata). Instead, we represent a configuration of a two counter machine using two "identical" copies, one with the pop alphabet of the visibly pushdown automaton, and the other with the push alphabet, and use the single clock to make sure that the two copies are identical.

Thus, our results show that model checking real-time software modeled as timed pushdown automata against real time specifications with one clock and no stack remains decidable, however, the problem becomes undecidable as soon as the specification formalism is allowed a visibly pushdown stack.

2 Timed Pushdown Automata

Given an alphabet Σ, a *timed word* $(\sigma, \tau) \in \Sigma^* \times \mathbb{R}^*$ of length n is a word $\sigma = \sigma_1\sigma_2 \ldots \sigma_n$ paired with a time sequence $\tau = \tau_1\tau_2 \ldots \tau_n$ such that τ is monotonically increasing. A *timed language* is a set of timed words.

Let C be a set of *clock variables.* A *clock constraint* ϕ is defined inductively by

$$\phi := x \leq c \mid c \leq x \mid \neg\phi \mid \phi_1 \wedge \phi_2$$

where $x \in C$ and $c \in \mathbb{Q}$. $\Phi(C)$ is the set of all clock constraints over C. For a set of clocks C, a *clock valuation* is a function $\nu : C \to \mathbb{R}$ which describes the values of each clock $c \in C$ at an instant. A clock constraint $\phi \in \Phi(C)$ is *satisfied* by the clock valuation ν (written $\nu \vdash \phi$) if $[\nu(c)/c]_{c \in C}\phi$ is true. Given a set of clocks λ and a clock valuation ν, let $\nu \downarrow \lambda$ be defined as

$$(\nu \downarrow \lambda)(c) = \begin{cases} 0 & \text{when } c \in \lambda \\ \nu(c) & \text{otherwise} \end{cases}$$

Given a clock valuation ν and a time $t \in \mathbb{R}$ define $(\nu + t)(c) = \nu(c) + t$.

A *timed pushdown automaton* (TPDA) is a tuple $M = (\tilde{\Sigma}, \Gamma, Q, S, F, C, \delta)$, where $\tilde{\Sigma}$ is a finite alphabet of input symbols, Γ is finite stack alphabet (we write $\Gamma_\epsilon = \Gamma \cup \{\epsilon\}$ where ϵ is a fresh symbol not in Γ), Q is a finite set of states, $S \subseteq Q$ is a set of start states, $F \subseteq Q$ is a set of final states, C is a finite set of real-valued clocks, and $\delta \subseteq (Q \times \Sigma \times \Gamma_\epsilon \times \Phi(C) \times Q \times \Gamma_\epsilon \times 2^C)$ is a discrete transition relation.

A transition $(q, a, \gamma, \phi, q', \gamma', \lambda) \in \delta$ is taken if the current location is q, the input symbol is a, the stack is popped and the popped symbol is γ (if $\gamma = \epsilon$, then the stack is not popped), and the current valuation ν satisfies ϕ, and then the new location is q', the symbol γ' is pushed on the stack (if $\gamma' = \epsilon$, then no symbol is pushed) and the new clock valuation is $\nu' = \nu \downarrow \lambda$. Given a timed word (σ, τ) of length n, a *run* of a TPDA M on (σ, τ) is a sequence $(q_1, \nu_1, \gamma_1), (q_2, \nu_2, \gamma_2), \ldots, (q_{n+1}, \nu_{n+1}, \gamma_{n+1}) \in (Q \times (C \to \mathbb{R}) \times \Gamma^*)^*$ if for each $i \in \{1, \ldots, n\}$ there exists $t \in \mathbb{R}$ such that $(q_i, \sigma_i, \gamma, \phi, q_{i+1}, \hat{\gamma}, \lambda) \in \delta$, $\nu_i \vdash \phi$, $\nu_{i+1} = \nu_i \downarrow \lambda + t$, and there is some $\gamma' \in \Gamma^*$ such that $\gamma_i = \gamma' \cdot \gamma$ and $\gamma_{i+1} = \gamma' \cdot \hat{\gamma}$. A run $\rho = (q_1, \nu_1, \gamma_1) \ldots (q_{n+1}, \nu_{n+1}, \gamma_{n+1})$ of a TPDA M is *initialized* if $q_1 \in S$, $\nu_1(c) = 0$ for all $c \in C$, and $\gamma_1 = \epsilon$. The run ρ is *accepting* if $q_{n+1} \in F$. A timed word (σ, τ) of length n is accepted by a TVPA M if there exists a run of M on (σ, τ) that is initialized and accepting. The timed language of a TPDA M, denoted $L(M)$, is the set of all timed words that are accepted by M. A TPDA M is called *universal* if $L(M) = (\tilde{\Sigma} \times \mathbb{R})^*$, i.e., if it accepts all timed words.

A TPDA M is *visibly pushdown* (TVPA) if the input alphabet $\tilde{\Sigma}$ can be partitioned into three disjoint sets $\tilde{\Sigma} = \Sigma_{int} \cup \Sigma_c \cup \Sigma_r$ of internal, call, and return input symbols respectively, and $\delta \subseteq (Q \times \Sigma_{int} \times \Phi(C) \times Q \times 2^C) \cup (Q \times \Sigma_c \times \Phi(C) \times Q \times \Gamma \times 2^C) \cup (Q \times \Sigma_r \times \Gamma \times \Phi(C) \times Q \times 2^C)$ is the visible pushdown transition relation. The transitions of a TVPA come in three varieties. Let ν be a clock valuation. An *internal transition* $(q, a, \phi, q', \lambda) \in \delta$ at clock valuation ν is a move on the (internal) input symbol a from the state q to q' such that ν satisfies

ϕ and the resulting clock valuation $\nu' = \nu \downarrow \lambda$. A *call transition* $(q, a, \phi, q', \gamma, \lambda)$ is a move on the (call) input symbol a from q to q' where ν satisfies ϕ, the clock valuation is updated from ν to $\nu \downarrow \lambda$, and γ is pushed on the stack. A *return transition* $(q, a, \gamma, \phi, q', \lambda)$ is a move on the (return) input symbol a and stack symbol γ, from q to q' where ϕ is satisfied and ν is updated to $\nu \downarrow \lambda$. Given a timed word (σ, τ) of length n, a *run* of the TVPA M on (σ, τ) is a sequence $(q_1, \nu_1, \gamma_1), (q_2, \nu_2, \gamma_2), \ldots, (q_{n+1}, \nu_{n+1}, \gamma_{n+1}) \in (Q \times (C \to \mathbb{R}) \times \Gamma^*)^*$ if for each $i \in \{1, \ldots, n\}$ there exists $t \in \mathbb{R}$ such that one of the following holds:

1. $(q_i, \sigma_i, \phi, q_{i+1}, \lambda) \in \delta$, $\nu_i \vdash \phi$, $\nu_{i+1} = \nu_i \downarrow \lambda + t$, and $\gamma_{i+1} = \gamma_i$
2. $(q_i, \sigma_i, \phi, q_{i+1}, \gamma, \lambda) \in \delta$, $\nu_i \vdash \phi$, $\nu_{i+1} = \nu_i \downarrow \lambda + t$, and $\gamma_{i+1} = \gamma_i \gamma$
3. $(q_i, \sigma_i, \gamma, \phi, q_{i+1}, \lambda) \in \delta$, $\nu_i \vdash \phi$, $\nu_{i+1} = \nu_i \downarrow \lambda + t$, and $\gamma_{i+1} \gamma = \gamma_i$

Initialized and accepted runs, languages, and universality for TVPAs are defined by restriction from TPDAs. A TVPA is a timed automaton if $\Sigma_c = \Sigma_r = \emptyset$.

The *universality problem* for TPDAs (resp. TVPAs) takes as input a TPDA (resp. TVPA) M, and returns "yes" if M is universal, and returns "no" otherwise. Notice that the universality problem for TPDAs is undecidable, even if there is no clock, i.e., if $C = \emptyset$, since the universality of PDAs is undecidable [11].

3 One-Clock Language Inclusion Problem

We now show that the language inclusion problem $L(B) \subseteq L(A)$ where B is a TPDA and A is a timed automaton with one clock is decidable. This extends the result of [13]. Our main technical tool is a decidability result for context free reachability for well-structured transition systems.

3.1 Well-structured Infinite Pushdown Automata

A *quasi-order* (or *preorder*) $\leq$ over a set A is a reflexive and transitive relation $\leq \subseteq A \times A$. A *well-quasi-order* (wqo) is a quasi-order where for every infinite sequence $a_1, a_2, a_3, \ldots$ from A, there exist $i, j \in \mathbb{N}$ where $i < j$ and $a_i \leq a_j$. We say that a *dominates* a' if $a' \leq a$.

An *infinite pushdown automaton* (∞PDA) M is a quintuple $M = (Q, \Sigma, \Gamma, S, \delta)$ where Q is an infinite set of states, Σ and Γ are input and tape alphabets, $S \subseteq Q$ is a finite set of initial states, and $\delta \subseteq (Q \times \Sigma \times Q) \cup (Q \times \Sigma \times Q \times \Gamma) \cup (Q \times \Sigma \times \Gamma \times Q)$ is a transition relation. We say M is *finitely branching* if for all $q \in Q$, $\{(q, \sigma, q') \in \delta \mid \sigma \in \Sigma, q' \in Q\} \cup \{(q, \sigma, q', \gamma) \in \delta \mid \sigma \in \Sigma, q' \in Q, \gamma \in \Gamma\} \cup \{(q, \sigma, \gamma, q') \in \delta \mid \sigma \in \Sigma, \gamma \in \Gamma, q' \in Q\}$ is a finite set. Given a quasi-order $\leq \subseteq Q \times Q$ we say that $\leq$ is *strictly (downward) compatible* with δ if for all $p \leq q$:

(i) if $(q, \sigma, q') \in \delta$ for some $q' \in Q$ then there exists $p' \in Q$ such that $(p, \sigma, p') \in \delta$ and $p' \leq q'$;
(ii) if $(q, \sigma, q', \gamma) \in \delta$ for some $q' \in Q$ and $\gamma \in \Gamma$ then there exists $p' \in Q$ such that $(p, \sigma, p', \gamma) \in \delta$ and $p' \leq q'$; and
(iii) if $(q, \sigma, \gamma, q') \in \delta$ for some $q' \in Q$ and $\gamma \in \Gamma$ then there exists $p' \in Q$ such that $(p, \sigma, \gamma, p') \in \delta$ and $p' \leq q'$.

A *well-structured infinite pushdown automaton* ($\infty_{\sqsubseteq}$PDA) $M = (Q, \Sigma, \Gamma, S, \delta, \sqsubseteq)$ is a finitely branching infinite pushdown automaton where $\sqsubseteq \subseteq Q \times Q$ is a wqo over the set of states Q that is strictly compatible with δ.

Let M be a well-structured infinite pushdown automaton over states Q, and let $\sqsubseteq$ be a well-quasi-order over Q. Define $\sqsubseteq$-*reachability* to be the decision problem of whether or not for a given state $q \in Q$ there exists a state $q' \in Q$ such that $q' \sqsubseteq q$ and q' is reachable from an initial state of M.

Theorem 1. *The $\sqsubseteq$-reachability problem is decidable for well-structured infinite pushdown automata with decidable $\sqsubseteq$.*

Proof. The algorithm $\sqsubseteq$-*Reachability* shown in Figure 1 computes the set $Paths \subseteq S \times Q$. The set $Paths$ contains pairs of states (s, q) where s is a start state, and there is some series of transitions from M in which to arrive at q, taking into account constraints on stack symbols. Thus the pair $(s, q) \in Paths$ corresponds to the existence of a path from s to q. To see that there is a corresponding pair in $Paths$ for every reachable state, if a state q remains unexplored either $Update$ was never invoked with q, or there is some state q' which is dominated by q, and q' is explored. In the latter case, since $\sqsubseteq$ is compatible with δ, we can be sure that for any state p which q could have transitioned to, there will be some state $p' \sqsubseteq p$ which q' could transition to. If $Update$ is never invoked with q then either there is no explored state which made a transition to q, or at some point along a path to q there is a state which dominates a state that is explored. Thus to decide whether a given state $p \in Q$ is $\sqsubseteq$-reachable in M, it suffices to find a pair (s, p') from the finite set $Paths$ such that $p' \sqsubseteq p$.

To show that the algorithm $\sqsubseteq$-*Reachability* indeed terminates in a finite number of steps, consider the following argument. Since $\sqsubseteq$ is a wqo, there is no infinite strictly decreasing sequence $q_1 \sqsupset q_2 \sqsupset q_3 \sqsupset \ldots$ from Q, and so any number of repeated invocations of the subroutine $Update$ can only add a finite number of states to the set $Basis$. Because $Basis$ is always a finite set, and only states from $Basis$ are considered in the sets $Paths$, $Summaries$ and $ToExplore$, these sets are also finite; thus the second phase of the subroutine $Update$ is only invoked a finite number of times, and only a finite sequence of states is added to $ToExplore$. For each state that is explored there are only a finite number of successors (by the definition of ∞PDA.) Combined with a finite bound on the number of states inserted into $ToExplore$, it is clear that the algorithm $\sqsubseteq$-*Reachability* terminates in a finite number of steps. ∎

3.2 Decidability of Language Inclusion

We now show the decidability of language inclusion when the specification is a timed automaton with one clock and the implementation a TPDA. Our proof follows that of [13]. The idea is to construct an infinite pushdown automaton by a product construction between A and B, in which we identify "bad" states as product states which are accepting in B, and non-accepting in A. Then we apply Theorem 1 to decide if any bad states are reachable. If some bad state is

$$
\begin{array}{l}
\texttt{subroutine}\ \mathit{Update}(q,p,s)\\
\quad \texttt{if}\ \nexists b \in \mathit{Basis}.b \sqsubseteq q\ \texttt{then}\\
\qquad \mathit{Basis} := \mathit{Basis} \cup \{q\}\\
\quad \texttt{if}\ q \in \mathit{Basis}\ \texttt{and (}\ p \not\subseteq \mathit{Paths}\ \texttt{or}\ s \not\subseteq \mathit{Summaries}\ \texttt{) then}\\
\qquad \mathit{Paths} := \mathit{Paths} \cup p\\
\qquad \mathit{Summaries} := \mathit{Summaries} \cup s\\
\qquad \mathit{ToExplore} := \mathit{ToExplore} \cup \{q\}\\
\\
\texttt{algorithm}\ \sqsubseteq\text{-}\mathit{Reachability}(M = (Q, \Sigma, \Gamma, S, \delta, \sqsubseteq))\\
\quad \texttt{let}\ \mathit{Basis} := \{q \mid q \in S\}\\
\qquad \mathit{ToExplore} := \{q \mid q \in S\}\\
\qquad \mathit{Paths} := \{(q,q) \mid q \in S\}\\
\qquad \mathit{Summaries} := \emptyset\\
\quad \texttt{in until}\ \mathit{ToExplore} = \emptyset\ \texttt{repeat}\\
\qquad \texttt{remove some}\ q\ \texttt{from}\ \mathit{ToExplore}\\
\qquad \texttt{for each}\ q' \in Q\ \texttt{and}\ \sigma \in \Sigma\ \texttt{where}\ (q,\sigma,q') \in \delta\ \texttt{do}\\
\qquad\quad \texttt{let}\ p = \{(q'',q') \mid (q'',q) \in \mathit{Paths}\}\\
\qquad\qquad s = \{(q'',q')_\gamma \mid (q'',q)_\gamma \in \mathit{Summaries}\}\\
\qquad\quad \texttt{in}\ \mathit{Update}(q',p,s)\\
\qquad \texttt{for each}\ q' \in Q\ \texttt{and}\ \sigma \in \Sigma\ \texttt{and}\ \gamma \in \Gamma\ \texttt{where}\ (q,\sigma,q',\gamma) \in \delta\ \texttt{do}\\
\qquad\quad \texttt{let}\ p = \{(q'',q') \mid (q'',q) \in \mathit{Paths}\}\\
\qquad\qquad s = \{(q,q')_\gamma\}\\
\qquad\quad \texttt{in}\ \mathit{Update}(q',p,s)\\
\qquad \texttt{for each}\ q' \in Q\ \texttt{and}\ \sigma \in \Sigma\ \texttt{and}\ \gamma \in \Gamma\ \texttt{where}\ (q,\sigma,\gamma,q') \in \delta\ \texttt{do}\\
\qquad\quad \texttt{let}\ p = \{(q'',q') \mid \exists q''' \in Q \text{ where}\\
\qquad\qquad\qquad (q'',q''') \in \mathit{Paths} \text{ and } (q''',q)_\gamma \in \mathit{Summaries}\}\\
\qquad\qquad s = \{\lceil q'',q' \rceil_\gamma \mid (q'',q)_\gamma \in \mathit{Summaries}\}\\
\qquad\quad \texttt{in}\ \mathit{Update}(q',p,s)
\end{array}
$$

Fig. 1. Algorithm to decide $\sqsubseteq$-reachability.

reachable, then B accepts some string which A does not, and $L(B) \nsubseteq L(A)$. If there are no reachable bad states, then we know $L(B) \subseteq L(A)$.

Theorem 2. *For a single-clock timed automaton A and timed pushdown automaton B, $L(B) \subseteq L(A)$ is decidable.*

Proof. Given a single-clock timed automaton $A = (Q_A, \Sigma, S_A, F_A, C_A = \{x\}, \delta_A)$, and a TPDA $B = (Q_B, \Sigma, \Gamma, S_B, F_B, C_B, \delta_B)$, we now define a $\infty_{\sqsubseteq}$PDA P which is the product automaton of a determinized A and non-deterministic B. We define the product as $P = (Q, \Sigma, \Gamma, S, \delta, \sqsubseteq)$ where

$$\begin{aligned}
Q &= \mathcal{P}(Q_A \times (C_A \to \mathbb{R})) \times Q_B \times (C_B \to \mathbb{R})\\
S &= \mathcal{P}(S_A \times (C_A \to \{0\})) \times S_B \times (C_B \to \{0\})\\
\delta &= \{((q_A, q_B), \sigma, (q'_A, q'_B)) \mid q'_B \in \delta'_B(q_B, \sigma), q'_A = \delta'_A(q_A, \sigma)\}\\
&\quad \cup \{((q_A, q_B), \sigma, (q'_A, q'_B), \gamma) \mid (q'_B, \gamma) \in \delta'_B(q_B, \sigma), q'_A = \delta'_A(q_A, \sigma)\}\\
&\quad \cup \{((q_A, q_B), \sigma, \gamma, (q'_A, q'_B)) \mid q'_B \in \delta'_B(q_B, \sigma, \gamma), q'_A = \delta'_A(q_A, \sigma)\}
\end{aligned}$$

and

$$\begin{aligned}
\delta'_A(q_A, \sigma) &= \{(q', \nu') \mid (q, \nu) \in q_A, (q, \sigma, \phi, q', \lambda) \in \delta_A, \nu \vdash \phi, \nu' = \nu \downarrow \lambda\}\\
\delta'_B((q, \nu), \sigma) &= \{(q', \nu') \mid (q, \sigma, \phi, q', \lambda) \in \delta_B, \nu \vdash \phi, \nu' = \nu \downarrow \lambda\}\\
&\quad \cup \{(q', \nu', \gamma) \mid (q, \sigma, \phi, q', \lambda, \gamma) \in \delta_B, \nu \vdash \phi, \nu' = \nu \downarrow \lambda\}\\
\delta'_B((q, \nu), \sigma, \gamma) &= \{(q', \nu') \mid (q, \sigma, \gamma, \phi, q', \lambda) \in \delta_B, \nu \vdash \phi, \nu' = \nu \downarrow \lambda\}
\end{aligned}$$

are defined for convenience. Note that P is finitely branching since clock valuations are not arbitrarily increased by a transition of A or B; rather they must either remain constant or be (partially) reset.

As in [13], the wqo $\sqsubseteq$ over Q is defined as follows. Without loss of generality we assume that all numeric values in the clock constraints of A and B are integral. Define the set of regions $Reg = \{r_0, r_0^1, r_1, r_1^2, \ldots, r_K\}$, where K is the largest constant appearing in a clock constraint of A and B. For all $t \in \mathbb{R}$, let $\bar{t} \in [0, 1)$ be the fractional part of t (i.e., $\bar{t} = t - \lfloor t \rfloor$). Let $reg : \mathbb{R} \to Reg$ be the function mapping clock values to regions defined by

$$reg(t) = \begin{cases} r_K & \text{when } t \geq K\\ r_i & \text{when } t < K, \bar{t} = 0 \text{ and } t = i\\ r_i^{i+1} & \text{when } t < K, \bar{t} \neq 0 \text{ and } i < t < i+1 \end{cases}.$$

Let $F : Q \to \mathcal{P}((Q_A \times Reg \times [0, 1)) \cup (Q_B \times C_B \times Reg \times [0, 1)))$ be a function which disassembles the state structure of Q into sets of its constituents:

$$\begin{aligned}
F((q_A, q_B, \nu_B)) = \{(\eta, \overline{\nu(x)}) \mid (q, \nu) \in q_A, \{x\} = C_A \text{ and } \eta = (q, reg(\nu(x)))\}\\
\cup \{(\eta, \overline{\nu_B(y)}) \mid y \in C_B \text{ and } \eta = (q_B, y, reg(\nu_B(y)))\}
\end{aligned}$$

Now define G to group together tuples with the same clock fraction:

$$G(q) = \left\{ \left(\bigcup \{\rho' \mid (\rho', t) \in F(q)\}, t \right) \middle| (\rho, t) \in F(q) \right\}$$

and finally let H be defined as

$$H(q) = \rho_1 \rho_2 \ldots \rho_{|G(q)|},$$

where $(\rho_i, t_i) \in G(q)$ for $1 \leq i \leq |G(q)|$ and $t_i < t_{i+1}$ for $1 \leq i < |G(q)|$. The codomain of H is the set of finite words on a finite alphabet. By Higman's Lemma, this set is well quasi-ordered with respect to the subword ordering $\preceq$. We define the quasi-order $\sqsubseteq$ as $q_1 \sqsubseteq q_2$ if and only if $H(q_1) \preceq H(q_2)$. As shown in [13], this is a well quasi-ordering on states.

To see that $\sqsubseteq$ is strictly compatible with δ consider two states $p \sqsubseteq q$. By the definition of $\sqsubseteq$, p and q are in the same Q_A and Q_B states, and the valuations of their corresponding clocks are in the same regions (i.e., they satisfy the same clock constraints.) Thus any transition that is enabled out of q has a corresponding transition enabled out of p.

Define $Bad = \mathcal{P}((Q_A \setminus F_A) \times (C_A \to \mathbb{R})) \times F_B \times (C_B \to \mathbb{R})$ to be the set of "bad" states of P. Since P is a $\infty_{\sqsubseteq}$PDA with decidable $\sqsubseteq$, Theorem 1 states that $\sqsubseteq$-reachability is decidable on P. Let $Paths$ be computed from $\sqsubseteq$-$Reachability(P)$. Now the language containment question $L(B) \subseteq L(A)$ is reduced to finding a state from the finite set $\{q \mid (s, q) \in Paths\}$ which also belongs to Bad, and is thus decidable. ∎

4 Universality of Timed Visibly Pushdown Automata

We now prove a negative result that shows that the specification formalism cannot be extended from finite state one-clock timed automata to one-clock TVPA. The universality problem for (untimed) visibly pushdown automata is decidable [6]. On the other hand, the universality problem for timed automata with two clocks is undecidable, and (from [13]) the universality problem for timed automata with one clock is decidable. We now show that the universality problem for TVPAs with one clock is undecidable, thus completing the decidability picture.

The proof is by a reduction from the halting problem for two counter machines. A *two counter machine* is a tuple $M = (I, C, D)$ where $I : \mathbb{N} \to (\{C, D\} \times \mathbb{N} \times \mathbb{N}) \cup (\{Inc, Dec\} \times \{C, D\})$ and the domain of I is finite. That is, M has a finite set of instructions I and counters C and D and at each instruction M can either increment or decrement one counter and proceed to the next instruction, or conditionally jump to another instruction upon a given counter having the value 0. For example the instruction $(3, (C, 5, 7)) \in I$ means that the instruction at location 3 is a jump to location 5 if the value of C is 0, and is otherwise a jump to location 7. The instruction $(5, (Dec, D)) \in I$ means that at location 5 the value of counter D is decremented before advancing to location 6.

A configuration of M is represented by the triple $(l, c, d) \in \mathbb{N} \times \mathbb{N} \times \mathbb{N}$ where l is a location and $c, d \geq 0$ are the values of counters C and D. The unique *initial configuration* is the triple $(1, 0, 0)$. The set of *final configurations* is $\{(2, c, d) \mid c, d \in \mathbb{N}\}$. A *run* of the two-counter machine $M = (I, C, D)$ is a sequence of configurations $(l_1, c_1, d_1)(l_2, c_2, d_2)\dots(l_n, c_n, d_n) \in (\mathbb{N} \times \mathbb{N} \times \mathbb{N})^*$ where for each $i \in \{1, 2, \dots, n-1\}$, we have

$$(l_{i+1}, c_{i+1}, d_{i+1}) = \begin{cases} (l_i + 1, c_i + 1, d_i) & \text{when } I(l_i) = (Inc, C) \\ (l_i + 1, c_i - 1, d_i) & \text{when } I(l_i) = (Dec, C) \\ (l_i + 1, c_i, d_i + 1) & \text{when } I(l_i) = (Inc, D) \\ (l_i + 1, c_i, d_i - 1) & \text{when } I(l_i) = (Dec, D) \\ (b_1, c_i, d_i) & \text{when } I(l_i) = (C, b_1, b_2) \text{ and } c_i = 0 \\ (b_2, c_i, d_i) & \text{when } I(l_i) = (C, b_1, b_2) \text{ and } c_i \neq 0 \\ (b_1, c_i, d_i) & \text{when } I(l_i) = (D, b_1, b_2) \text{ and } d_i = 0 \\ (b_2, c_i, d_i) & \text{when } I(l_i) = (D, b_1, b_2) \text{ and } d_i \neq 0 \end{cases}.$$

A run $\alpha_1 \dots \alpha_n$ of M is accepting if α_1 an initial configuration and α_n is a final configuration.

Theorem 3. *Universality of single clock TVPA's is undecidable.*

Proof. We reduce from the accepting problem for two counter machines. For a two-counter machine $M = (I, C, D)$, fix $\Sigma_{int} = \{h_i \mid i \in dom(I)\}$, $\Sigma_c = \{f_c, g_c\}$, $\Sigma_r = \{f_r, g_r\}$ and $\tilde{\Sigma} = \Sigma_{int} \cup \Sigma_c \cup \Sigma_r$. Given a two-counter machine M, we build a TVPA N that accepts any string in $w \in (\tilde{\Sigma} \times \mathbb{R})^*$ such that w does not represent an accepting run of M. The problem of finding the existence of an accepting run of M is then reduced to verifying that N is not universal (i.e., if the language of N is the universe, then M has no accepting run).

We represent each configuration (i, j, k) of the two-counter machine M by a timed word $\Pi(i, j, k) = (h_i f_r^j g_r^k g_c^k f_c^j, \tau)$ where $\tau_1 \in \mathbb{N}$, and $\tau_1 \leq \tau_{j+1} < (\tau_1 + \frac{1}{4}) < \tau_{j+2} \leq \tau_{j+k+1} < (\tau_1 + \frac{1}{2}) < \tau_{j+k+2} \leq \tau_{j+2k+1} < (\tau_1 + \frac{3}{4}) < \tau_{j+2k+2} \leq \tau_{2j+2k+1} < (\tau_1 + 1)$. In addition we require that for every g_r there is a g_c that follows at exactly $\frac{1}{4}$ time units, and for every f_r there is a f_c that follows at exactly $\frac{3}{4}$ time units. A sequence of configurations $\beta_1\beta_2\dots\beta_n$ is represented as the concatenation of timed words $\Pi(\beta_1)\Pi(\beta_2)\dots\Pi(\beta_n)$ where for each $1 \leq i < n$ we have $\beta_i = (\alpha^i, \tau^i)$, $\tau_1^{i+1} = \tau_1^i + 1$, and $\tau_1^1 = 1$.

We will build N to be a disjunction of several smaller TVPA's which each try to find a particular reason why the input string is not an accepting run of M. The input alphabet of N is $\tilde{\Sigma}$, the stack alphabet is $\Gamma = \{C, D, X, Y\}$, and there is one clock x. The state and transition structure is taken as the disjunction of the smaller automata that we now describe.

One possibility is that the input string does not represent some sequence of configurations. The regular automaton $N_{\neg format}$ accepts strings that are not matched by the regular expression $R_{format} = ((h_1 \cup h_2 \cup \dots h_m) f_r^* g_r^* g_c^* f_c^*)^*$, where $m = |dom(I)|$.

Another possibility is that at least one timing constraint is broken in the input string. The single-clock automaton $N_{\neg schedule}$ accepts any string in which either the first symbol does not occur at time 1, or there exist $h_i, h_j \in \Sigma_{int}$ from successive configurations where h_i does not occur exactly one time unit before h_j, or any of the symbols f_r, g_r, f_c, g_c don't fit into the intervals $(\tau + 0, \tau + \frac{1}{4})$, $(\tau + \frac{1}{4}, \tau + \frac{1}{2})$, $(\tau + \frac{3}{4}, \tau + 1)$, and $(\tau + \frac{1}{2}, \tau + \frac{3}{4})$ respectively, where τ is the time

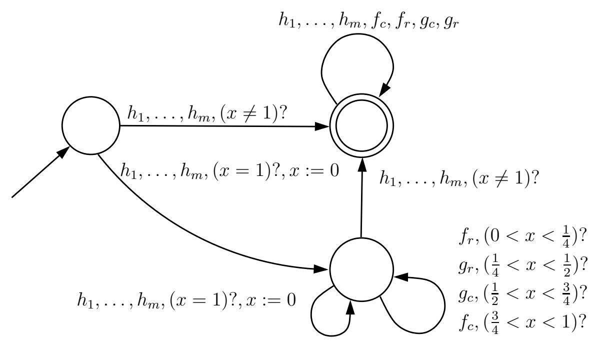

Fig. 2. The one-counter automaton $N_{\neg schedule}$ recognizes strings that are not properly timed configuration sequences.

of the nearest preceding h_i. The single-clock automaton $N_{\neg schedule}$ is shown in Figure 2.

The regular automaton $N_{\neg init}$ accepts strings that start with a configuration which is not initial. Since an initial configuration has location 1 and both counter values of 0, the regular expression $R_{init} = h_1\ (h_1 \cup h_2 \cup \ldots \cup h_m)\ (h_1 \cup h_2 \cup \ldots \cup h_m \cup f_c \cup f_r \cup g_c \cup g_r)^*$ matches all strings that represent correct initial configurations, where $m = |dom(I)|$.

The regular automaton $N_{\neg final}$ accepts strings that end with a configuration that is not final. Since a final configuration has location 2, the regular expression $R_{final} = (h_1 \cup h_2 \cup \ldots \cup h_m \cup f_c \cup f_r \cup g_c \cup g_r)^*\ h_2\ (f_c \cup f_r \cup g_c \cup g_r)^*$ matches all strings that represent correct final configurations, where $m = |dom(I)|$.

In our textual representation of a configuration there should be a f_c following each f_r by $\frac{3}{4}$ time units, a f_r $\frac{3}{4}$ time units before each f_c, and g_r if and only if there is a g_c following at $\frac{1}{4}$ time units. The single-clock automata $N_{\neg f_r \rightarrow f_c}$, $N_{\neg f_r \leftarrow f_c}$, $N_{\neg g_r \rightarrow g_c}$ and $N_{\neg g_r \leftarrow g_c}$ accept strings with unpaired f_r's and f_c's (or g_r's and g_c's) within a configuration. As an example $N_{\neg f_r \rightarrow f_c}$ is shown in Figure 3.

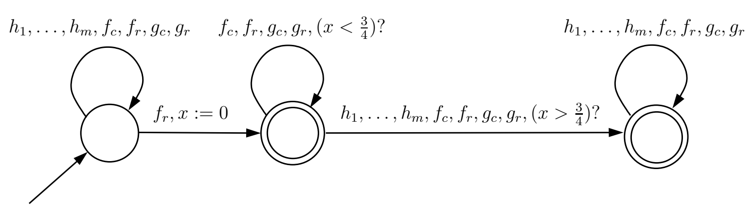

Fig. 3. The single-clock automaton $N_{\neg f_r \rightarrow f_c}$ recognizes strings in which some configuration has a symbol f_r without a matching symbol f_c following at $\frac{3}{4}$ time units.

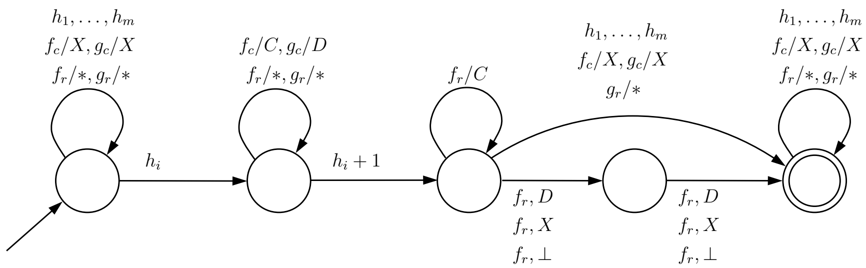

Fig. 4. The visibly pushdown automaton $N^i_{\neg c\uparrow}$ recognizes strings in which a configuration invoking instruction (Inc, C) is followed by a configuration where the C counter is not properly incremented.

The automata that have been described so far accept when there is some problem with the format of a particular configuration representation, and accept timed (or untimed) regular languages. The remaining automata will accept when there is a particular problem with a sequence of two configurations and will use the pushdown stack of a visibly pushdown automaton.

For each instruction/location we will use several automata to recognize an invalid sequence. For each instruction i that is not a branch instruction, the regular automaton $N^i_{\neg step}$ accepts when the following instruction in the configuration sequence is not $i+1$ (e.g., represented by h_{i+1}). For each instruction i which increments counter C, the automata $N^i_{\neg c\uparrow}$ and $N^i_{c\uparrow\wedge\neg d=}$ accept when counter C is not incremented, or C is incremented and D does not remain the same. These automata function by using the pushdown stack to remember how many g_c's and f_c's appear before the f_r's and g_r's of the following configuration. Note that the pushdown stack can only compare the counters of successive configurations, since symbols can only be pushed on the stack when f_c or g_c are read, and can only be popped from the stack when f_r or g_r are read. Figure 4 depicts this functionality.

The remaining $N^i_{c\uparrow\wedge\neg d=}$, $N^i_{\neg c\downarrow}$, $N^i_{c\downarrow\wedge\neg d=}$, $N^i_{\neg c=}$, $N^i_{c=\wedge\neg d\uparrow}$, $N^i_{c=\wedge\neg d\downarrow}$, and $N^i_{c=\wedge\neg d=}$ automata all function in a similar manner, by counting between configurations using the pushdown stack.

The automata $N^i_{c=0\wedge\neg goto-l_1}$ and $N^i_{c\neq0\wedge\neg goto-l_2}$ require no clock or stack, and simply recognize when the configuration following a branch has a location that does not match with the branch location.

As mentioned earlier, N is simply the disjunction of each of the machines mentioned above. Since each automaton uses at most one clock, and are either regular or visibly pushdown, the resulting disjunction N is a single-clock visibly pushdown automaton. By construction N accepts any string that either doesn't encode a sequence of configurations of M, or encodes a sequence of configurations that is not an initialized and accepting run of M. Thus by reduction, the universality problem for N decides membership for M. ∎

As a corollary, the language inclusion problem $L(A) \subseteq L(B)$ where B is a timed visibly-pushdown automaton with at least one clock is undecidable. What can we say if the specification automaton B has no clocks (i.e., is an untimed visibly pushdown automaton)? If A is a TVPA, then the language inclusion problem is decidable using the closure properties of visibly pushdown automata (i.e., we can reduce the problem to the emptiness question $L(A \cap \neg B) = \emptyset$, and $A \cap \neg B$ is again a TVPA). On the other hand, if A is a PDA, then the problem is already undecidable from the untimed case.

5 Conclusions

We have sharpened the frontier for decidability of timed language inclusion. On the one hand, we show that the language problem remains decidable if the implementation is strengthened to be a TPDA and the specification is a timed automaton with at most one clock. On the other hand, if we strengthen the specification to TVPA with one clock, the problem becomes undecidable.

References

1. P. A. Abdulla, K. Čerāns, B. Jonsson, and Yih-Kuan Tsay. General decidability theorems for infinite-state systems. In *LICS 96: Logic in Computer Science*, pages 313–321. IEEE Press, 1996.
2. P. A. Abdulla, J. Deneux, J. Ouaknine, and J. Worrell. Decidability and complexity results for timed automata via channel machines. In *ICALP 05: International Conference on Automata, Languages, and Programming*, LNCS 3580, pages 1089–1101. Springer-Verlag, 2005.
3. P. A. Abdulla, K. Čerāns, B. Jonsson, and Yih-Kuan Tsay. Algorithmic analysis of programs with well quasi-ordered domains. *Information and Computation*, 160:109–127, 2000.
4. R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.
5. R. Alur, K. Etessami, and P. Madhusudan. A temporal logic of nested calls and returns. In *TACAS 04: Tools and Algorithms for the Construction and Analysis of Systems*, LNCS 2988, pages 467–481. Springer-Verlag, 2004.
6. R. Alur and P. Madhusudan. Visibly pushdown automata. In *STOC 04: Symposium on Theory of Computing*, pages 202–211. ACM Press, 2004.
7. T. Ball and S. K. Rajamani. Bebop: A symbolic model checker for Boolean programs. In *SPIN 00: SPIN Workshop*, LNCS 1885, pages 113–130. Springer-Verlag, 2000.
8. A. Bouajjani, R. Echahed, and R. Robbana. On the automatic verification of systems with continuous variables and unbounded discrete data structures. In *Hybrid Systems II*, LNCS 999, pages 64–85. Springer-Verlag, 1994.
9. Z. Dang. Pushdown timed automata: a binary reachability characterization and safety verification. *Theoretical Computer Science*, 302:93–121, 2003.
10. A. Finkel and Ph. Schnoebelen. Well-structured transition systems everywhere. *Theoretical Computer Science*, 256:63–92, 2001.

11. J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Publishing Company, 1979.
12. V.K. Nandivada and J. Palsberg. Timing analysis of TCP servers for surviving denial-of-service attacks. In *RTAS 05: IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 541–549. IEEE Press, 2005.
13. J. Ouaknine and J. Worrell. On the language inclusion problem for timed automata: Closing a decidability gap. In *LICS 2004: Logic in Computer Science*, pages 54–63. IEEE Press, 2004.
14. T. Reps, S. Horwitz, and M. Sagiv. Precise interprocedural dataflow analysis via graph reachability. In *POPL 95: Principles of Programming Languages*, pages 49–61. ACM, 1995.