Eventual Timed Automata

Deepak D'Souza Raj Mohan M

IISc-CSA-TR-2005-8
http://archive.csa.iisc.ernet.in/TR/2005/8/

Computer Science and Automation
Indian Institute of Science, India

June 2005

Eventual Timed Automata

Deepak D'Souza   Raj Mohan M
Dept. of Computer Science and Automation,
Indian Institute of Science,
Bangalore 560012, India.
Email: {deepakd,raj}@csa.iisc.ernet.in

Abstract

We study the class of timed automata obtained using guards based on the operator $\diamond$. These automata, which we call *Eventual Timed Automata*, are a special case of the generic class of "input determined automata" studied in [DT04] which were shown to have robust logical properties. It was also shown that the well known timed temporal logic Metric Temporal Logic (MTL) corresponds to the first-order fragment of eventual timed automata. In this paper we show that ETA's form a decidable class of timed automata. This is done in two steps: we first show that the general class of ETA's which allow "recursion" can be "flattened" to a non-recursive version. We then show how non-recursive ETA's can be decided via a reduction to one-clock alternating timed automata which have been shown to be decidable [LW05, OW05]. We also study the expressiveness of the class of ETA's and show that they compare favourably with other classes in the literature. Finally we show that class obtained using the dual operator $\diamondminus$ is also decidable, though the two together lead to an undecidable class of languages.

1 Introduction

The timed automata of Alur and Dill [AD94] are a popular model of time dependent systems. While they are very expressive and have a decidable emptiness problem, they suffer from the fact that they are not closed under complement and have an undecidable language inclusion problem. As a result of non-closure under complementation, they also do not admit natural monadic second order logic characterisations.

In an attempt to describe a robust class of timed automata which are closed under boolean operations and which have more robust logical properties, the generic class of Input Determined Automata (IDA's) were identified in [DT04]. In contrast to a timed automaton, an IDA does not have explicit clocks but uses distance operators whose meaning is determined solely by the given timed word and a position in it. These generic classes of automata have several desirable properties. They are determinizable, closed under boolean operations, admit natural monadic second order logic characterisations, and have expressively complete timed temporal logics.

In this paper we focus on the class of IDA's based on the input determined operator $\Diamond$, and call this class *eventual timed automata* or ETA's. The operator $\Diamond_a$, where a is a letter of the alphabet, identifies a set of distances to future positions in the timed word which are labelled a. An eventual timed automaton can use guards of the form $(\Diamond_a \in I)$ which is then true at a point in a timed word provided there is a position in the future where an a occurs and the distance to that point lies in the interval I. In general, the class of recursive ETA's can make use of operators of the form $\Diamond_{\mathcal{B}}$, where $\mathcal{B}$ is in turn such an ETA which accepts timed words *at* a designated position in them.

The class of ETA's are an interesting class of timed automata for reasons other than the fact that they inherit the useful properties of generic IDA's listed above. The well known timed temporal logic Metric Temporal Logic (MTL) [Koy90, AFH96, OW05] corresponds to the first order fragment of recursive ETA's, in that MTL is expressively equivalent to the first order fragment of the timed MSO which characterises recursive ETA's [DT04]. Further, they form an orthogonal class of timed languages with respect to Alur-Dill timed automata, since they can express some languages not expressible by timed automata [DT04]. However nothing much was known about decision procedures for this class of languages.

In this paper we address this issue and show that recursive ETA's form a decidable class of timed languages. We do this in two steps. We first show that recursive ETA's can be "flattened" to a non-recursive one over an extended alphabet. We then show that the emptiness problem for ETA's can be decided by showing how they can be simulated by 1-clock alternating timed automata which were recently shown to be decidable [LW05, OW05].

We also study the expressiveness of this class more closely, and show that it is indeed incomparable with the class of timed automata in that there is also a language which timed automata can express but a recursive ETA cannot. The class of recursive ETA's can also be seen to be strictly more expressive than the class of recursive Event Predicting Automata [AFH94, HRS98] which are another class with robust logical properties [DT04]. This, together with the properties described above, make the class of recursive

ETA's one of the most expressive classes of timed automata of its kind in the literature.

Finally we show that the class of automata obtained by considering the dual operator $\diamondminus$ is also decidable. We do this by reducing its emptiness problem to that of ETA's. However, the two operators taken together lead to an undecidable class of automata.

Some proofs are omitted due to lack of space and can be found in the full version of this paper [DM05].

2 Eventual Timed Automata

For an alphabet A we use A^* to denote the set of finite words over A, and $|w|$ to denote the length of a word w. The set of non-negative real numbers will be denoted by $\mathbb{R}^{\geq 0}$.

A (finite) *timed word* over an alphabet Σ is an element of $(\Sigma \times \mathbb{R}^{\geq 0})^*$ of the form $\sigma = (a_0, t_0)(a_1, t_1) \cdots (a_n, t_n)$, satisfying $t_i < t_{i+1}$ for all $i \in \{0, \ldots, n-1\}$. Wherever convenient, we will also use the representation of the timed word σ above as (w, τ) where $w = a_0 \cdots a_n$ and $\tau = t_0 \cdots t_n$. We denote the set of all timed words over Σ by $T\Sigma^*$.

We will use rational bounded intervals to specify timing constraints. These intervals can be open or closed, and we allow ∞ as an open right end. These intervals denote a subset of reals in the usual manner – for example $[2, \infty)$ denotes the set $\{t \in \mathbb{R}^{\geq 0} \mid 2 \leq t\}$. The set of all intervals is denoted $\mathcal{I}_{\mathbb{Q}}$.

An eventual timed automaton is essentially a timed automaton which has no explicit clocks, but uses atomic guards of the form "$\diamond_a \in I$", which assert that with respect to the current position in a timed word there is an occurrence of an action a at some point in the future, such that the distance to that point lies in the interval I. To define these automata more formally, we begin by defining the guards used by them. For an alphabet Σ, we use $\mathcal{G}(\Sigma)$ to denote the set of guards given by the syntax $g ::= \top \mid \diamond_a \in I \mid \neg g \mid g \vee g \mid g \wedge g$, where $a \in \Sigma$ and $I \in \mathcal{I}_{\mathbb{Q}}$. The guards are interpreted at the "action points" in a timed word σ (we call these *positions* and denote them by natural numbers between 0 and $|\sigma| - 1$). Let $\sigma = (a_0, t_0) \cdots (a_n, t_n)$ and $i \in \{0, \ldots, |\sigma| - 1\}$. Then σ at position i satisfies the guard $\diamond_a \in I$ iff there exists $j \geq i$ such that $a_j = a$ and $t_j - t_i \in I$. The guard $\top$ is always true, and the boolean operators are interpreted in the expected manner.

An *eventual timed automaton (ETA)* over Σ is of the form $\mathcal{A} = (Q, s, \delta, F)$ where Q is a finite set of states, $s \in Q$ is the start state, δ which is the transition relation is a finite subset of $Q \times \Sigma \times \mathcal{G}(\Sigma) \times Q$, and $F \subseteq Q$ is a set of final states. A run of $\mathcal{A}$ on a timed word $\sigma = (a_0, t_0) \cdots (a_n, t_n)$ is a sequence of states $q_0, \ldots, q_{n+1}$ satisfying

- $q_0 = s$,
- for each $i \in \{0, \ldots, n\}$ there exist guards g_i such that $(q_i, a_i, g_i, q_{i+1}) \in \delta$ and $\sigma, i \models g_i$.

The run is accepting if $q_{n+1} \in F$. The language of timed words accepted by $\mathcal{A}$ is denoted $L(\mathcal{A})$ and defined to be the set of all timed words in $T\Sigma^*$ on which $\mathcal{A}$ has an accepting run.

The figure below shows an example of a simple ETA with a single state that is both the start and the final state. It accepts the language L_0 of all timed words of a's in which there are no two a's at a distance of 1 time unit apart.

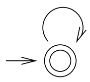

Figure 1: An example ETA

It will be convenient to recall the general class of input determined automata (IDA) defined in [DT04]. An IDA over an alphabet Σ has a structure similar to an ETA except that it is parameterised by a set of "input determined" operators Op. Each operator $\Delta \in Op$ has a semantic function $[\![\Delta]\!]$ which we assume in this paper associates a set of distances with a given timed word σ and a position i in it. Thus $[\![\Delta]\!](\sigma, i) \subseteq \mathbb{R}^{\geq 0}$. The atomic guards in an IDA are now of the form $(\Delta \in I)$ and the truth of the guard at position i in a timed word σ is given by $\sigma, i \models \Delta \in I$ iff $[\![\Delta]\!](\sigma, i) \cap I \neq \emptyset$. The timed language accepted by an IDA $\mathcal{A}$ over an alphbet Σ and a set of operators Op is denoted $L(\mathcal{A})$ and is defined as for ETA's, with the guards interpreted as described above.

Thus the class of ETA's over an alphabet Σ defined above is nothing but the class of IDA's corresponding to the set of operators $\{\Diamond_a \mid a \in \Sigma\}$ (we denote this class by IDA($\Diamond$)) with the semantic function $[\![\Diamond_a]\!]$ given by

$$[\![\Diamond_a]\!](\sigma, i) = \{\tau(j) - \tau(i) \mid \sigma = (w, \tau),\ j \geq i \text{ and } w(j) = a\}.$$

We now introduce the *recursive* version of ETA's. The main idea is to index the $\Diamond$ operator with an *automaton* $\mathcal{B}$ and use the operator $\Diamond_{\mathcal{B}}$ to mean the set of distances to the points in the future where the automaton $\mathcal{B}$ "accepts".

To make this notion more precise, we first introduce the notion of a *floating* language. A *floating language* over Σ is a set of *floating timed words* over Σ which in turn are pairs of the form (σ, i) where σ is a timed word over Σ and i is a position in σ. We will represent a "floating" word (σ, i) as a timed word over the extended alphabet $\Sigma \times \{0,1\}$. Thus a timed word ν over $\Sigma \times \{0,1\}$ represents the floating word (σ, i), iff $\nu = (w, v, \tau)$, with $v \in \{0,1\}^*$ with a *single* 1 in the i-th position, and $\sigma = (w, \tau)$. We use *fw* to denote the (partial) map which given a timed word ν over $\Sigma \times \{0,1\}$ returns the floating word (σ, i) corresponding to ν, and extend it to apply to timed languages over $\Sigma \times \{0,1\}$ in the natural way.

We can now define recursive ETA's and the timed languages they accept as follows. The class of *recursive eventual timed automata* (recursive ETA's) over the alphabet Σ is the union of the classes of level i recursive eventual timed automata over Σ (for $i \geq 0$) which are defined inductively below:

- A *level 0 recursive ETA* over Σ is an ETA $\mathcal{A}$ over Σ that uses only the guard $\top$. It accepts the language $L(\mathcal{A})$ as defined for ETA's.

- Similarly, a *level 0 recursive floating ETA* over Σ is an ETA $\mathcal{B}$ over the alphabet $\Sigma \times \{0,1\}$ that uses only the guard $\top$. It accepts the floating language $L^f(\mathcal{B})$ defined to be $fw(L(\mathcal{B}))$.

- Let D be a finite collection of recursive floating ETA of level i or less over Σ, and consider the set of operators $Op = \{\Diamond_{\mathcal{B}} \mid \mathcal{B} \in D\}$. We define the semantics of $\Diamond_{\mathcal{B}}$ in Op by:

$$[\![\Diamond_{\mathcal{B}}]\!](\sigma, i) = \{\tau(j) - \tau(i) \mid \sigma = (w, \tau),\ j \geq i, \text{ and } (\sigma, j) \in L^f(\mathcal{B})\}.$$

  A *level* $i+1$ *recursive ETA* over Σ is then an IDA $\mathcal{A}$ over (Σ, Op), and accepts the language $L(\mathcal{A})$ as defined for IDA's. We also require that $\mathcal{A}$ should use at least one operator of the form $\Diamond_{\mathcal{B}}$ for a recursive floating ETA $\mathcal{B}$ of level i.

- To define level $i+1$ recursive floating ETA's, let D be a finite collection of recursive floating ETA's of level i or less over Σ, and consider the set of operators $Op = \{\Diamond_{\mathcal{B}} \mid \mathcal{B} \in D\}$. For floating automata the operator $\Diamond$ works slightly differently in that it ignores the extended part of the alphabet. We define the semantics of $\Diamond_{\mathcal{C}}$ when used in a floating automaton as follows. Let σ' be a timed word over $\Sigma \times \{0,1\}$. Then

$$[\![\Diamond_{\mathcal{C}}]\!](\sigma', i) = \{\tau(j) - \tau(i) \mid \sigma' = (w, v, \tau),\ j \geq i, \text{ and } ((w, \tau), j) \in L^f(\mathcal{C})\}.$$

A *level* $i+1$ *recursive floating ETA* over Σ is then an IDA $\mathcal{B}$ over $(\Sigma \times \{0,1\}, Op)$, and accepts the floating language $L^f(\mathcal{B})$ defined to be $fw(L(\mathcal{B}))$. Once again we require that $\mathcal{B}$ should use at least one operator of the form $\Diamond_{\mathcal{C}}$ for a recursive floating ETA $\mathcal{C}$ of level i.

Below we give an example of a recursive ETA over the alphabet $\{a, b, c\}$.

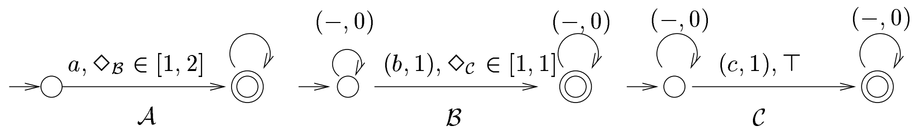

Figure 2: A level 2 recursive ETA.

Fig. 2 shows a level 2 recursive ETA $\mathcal{A}$ which uses a level 1 floating automaton $\mathcal{B}$, which in turn calls a level 0 floating automaton $\mathcal{C}$. $\mathcal{A}$ accepts timed words σ which start with an a at some time t, followed eventually by a b at time t' such that $t' - t \in [1,2]$, and a c at time $t'' = t' + 1$. For example, the timed word $(a, 0.2)(a, 1.2)(b, 1.4)(c, 2.4)$ satisfies all the conditions given above and therefore belongs to $L(\mathcal{A})$.

Before we close this section, we recall some of the results of [DT04] specialized to the case of recursive ETA's. We recall briefly the timed monadic second order logic rec-TMSO(Σ) based on the operator $\Diamond$, which characterises the class of timed languages defined by recursive ETA's. The syntax of the logic is given by

$$\varphi ::= Q_a(x) \mid \Diamond_\psi(x) \in I \mid x \in X \mid x < y \mid \neg\varphi \mid (\varphi \vee \varphi) \mid \exists x \varphi \mid \exists X \varphi.$$

In the predicate $\Diamond_\psi(x) \in I$, we have ψ a rec-TMSO(Σ) formula with a single free variable z, x a first-order variable, and $I \in \mathcal{I}_{\mathbb{Q}}$.

The logic is interpreted over timed words in $T\Sigma^*$. The semantics of the predicate "$\Diamond_\psi(x) \in I$" is defined inductively as follows. If ψ is a formula which uses no $\Diamond$ predicates, then the satisfaction relation $\sigma, \mathbb{I} \models \psi$ is defined as for standard MSO. Inductively, assuming the semantics of ψ has already been defined, $\Diamond_\psi$ is interpreted as an input determined operator as follows. Then

$$[\![\Diamond_\psi]\!](\sigma, i) = \{\tau(j) - \tau(i) \mid \sigma = (w, \tau),\ j \geq i,\ \text{and } \sigma, [j/z] \models \psi\}.$$

Then we say

$$\sigma, \mathbb{I} \models \Diamond_\psi(x) \in I \quad \text{iff} \quad [\![\Diamond_\psi]\!](\sigma, \mathbb{I}(x)) \cap I \neq \emptyset.$$

A sentence φ in rec-TMSO(Σ) defines the language $L(\varphi) = \{\sigma \in T\Sigma^\omega \mid \sigma \models \varphi\}$.

Theorem 1 ([DT04]) *The class of recursive ETA's over an alphabet Σ satisfy the following properties:*

1. *They are determinizable and closed under boolean operations.*
2. *The timed monadic second order logic* rec-TMSO(Σ) *is expressively equivalent to them.*
3. *The timed temporal logic MTL is expressively equivalent to the first order fragment of* rec-TMSO(Σ). $\square$

3 Deciding emptiness for ETA's

In this section we give a decision procedure for the language emptiness problem for ETA's. We do this by showing how to translate an ETA to a 1-clock alternating timed automaton, the emptiness problem of which has been shown to be decidable [LW05, OW05].

We first recall the definition of a 1-clock alternating timed automaton. Let $\mathcal{G}_x$ denote the set of guards over the clock x given by the following syntax: $g ::= \top \mid x \in I \mid \neg g \mid g \vee g$. A valuation for the clock x, denoted $d \in \mathbb{R}^{\geq 0}$, satisfies $x \in I$, written $d \models x \in I$, iff $d \in I$. The other guards are interpreted in the expected manner. For $X \subseteq \{x\}$, we use $d[0/X]$ to denote the valuation d or 0, depending on whether $X = \emptyset$ or $\{x\}$.

A 1-clock *alternating timed automaton* (ATA) over an alphabet Σ is of the form $\mathcal{T} = (Q, s, \Delta, F)$ where Q is a finite set of states, $s \in Q$ is the initial state, $\Delta \subseteq Q \times \Sigma \times \mathcal{G}_x \times (2^{\{\emptyset,\{x\}\}\times Q} - \{\emptyset\})$ is the transition relation, and $F \subseteq Q$ is the set of final states.

An *extended state* of $\mathcal{T}$ is a pair of the form (q, d) where $q \in Q$ and $d \in \mathbb{R}^{\geq 0}$ is a valuation for x. A *configuration* of $\mathcal{T}$ is a finite collection of extended states. To describe a run of the ATA over a timed word it is convenient to first define a consecution relation between configurations. Let $C = \{(q_0, d_0), \dots, (q_k, d_k)\}$ be a configuration of $\mathcal{T}$. Let $t \in \mathbb{R}^{\geq 0}$ and $a \in \Sigma$. Then a configuration D is an (a, t)-successor of C iff D is the union of configurations $D_0, \dots, D_k$ which satisfy: there exist guards g_i and sets of reset-state pairs $T_i = \{(r_0^i, p_0^i), \dots, (r_{n_i}^i, p_{n_i})\}$ such that

- $(q_i, a, g_i, T_i) \in \Delta$,

- $d_i + t \models g_i$, and
- $D_i = \{(p^i_j, (d_i + t)[0/r^i_j]) \mid j \in \{0, \ldots, n_i\}\}$.

An accepting run of $\mathcal{T}$ over a timed word $\sigma = (a_0, t_0) \cdots (a_n, t_n)$ is a sequence of configuration $C_0, \ldots, C_{n+1}$ satisfying:

1. $C_0 = \{(s, 0)\}$
2. For each $i \in \{0, \ldots, n\}$, the configuration C_{i+1} is an $(a_i, t_i - t_{i-1})$-successor of C_i. (We use the convention that $t_{-1} = 0$.)
3. C_{n+1} contains only final states – i.e. $C_{n+1} \subseteq F \times \mathbb{R}^{\geq 0}$.

The timed language accepted by $\mathcal{T}$, denoted $L(\mathcal{T})$, is the set of timed words over Σ on which $\mathcal{T}$ has an accepting run.

As an example, the timed language over $\{a, b\}$ in which every a is followed by a b exactly one time unit later is accepted by the ATA $\mathcal{T}$ shown in the diagram below. The transition relation Δ of $\mathcal{T}$ is depicted by using an arc covering the egdes corresponding to an "And" transition. For example the state s has two entries in Δ corresponding to its two outgoing transitions: $(s, b, \top, \{(\emptyset, s)\})$ and $(s, a, \top, \{(\emptyset, s), (\{x\}, t)\})$.

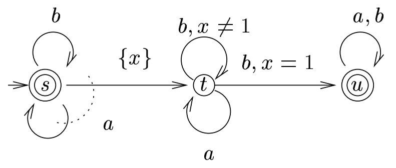

Figure 3: An example ATA

We recall here the recent result of Lasota-Walukiewicz and Ouaknine-Worell:

Theorem 2 ([LW05, OW05]) *The emptiness problem of one-clock alternating timed automata over finite words is decidable.* □

We now show how to translate a given ETA to a language-equivalent 1-clock ATA. The idea is fairly simple: for every guarded transition in the ETA we spawn off a copy of the automaton which verifies a literal guard of the form $\diamond_a \in I$ or $\neg\diamond_a \in I$. The translation is depicted in Fig. 4.

More precisely, let $\mathcal{A} = (Q, s, \delta, F)$ be an ETA over the alphabet Σ. We construct an ATA $\mathcal{T} = (Q', s', \Delta, F')$ as follows. The set of states Q' of $\mathcal{T}$ consists of the original states Q of $\mathcal{A}$, a new state u, and a new state p_c for each literal guard c (i.e. of the form $\Diamond_a \in I$ or $\neg\Diamond_a \in I$) in $\mathcal{A}$. The initial state s' is the initial state s of $\mathcal{A}$. The transition relation Δ contains the following transitions:

1. For each transition (p, a, g, q) of $\mathcal{A}$ we have the following transition in Δ. Without loss of generality let $g = c_0 \wedge \cdots \wedge c_k$ be a conjunction of literal guards c_i. Then

$$(p, a, g, \{(\emptyset, q)\} \cup \{(\{x\}, p_{c_i}) \mid 0 \leq i \leq k\}) \in \Delta$$

2. For each positive literal c of the form $\Diamond_a \in I$ we have the transitions

$$\begin{array}{ll} (p_c, b, \top, \{(\emptyset, p_c)\}) & \in \Delta \quad \text{(for each } b \in \Sigma\text{)}, \\ (p_c, a, x \in I, \{(\emptyset, u)\}) & \in \Delta. \end{array}$$

3. For each negative literal c of the form $\neg\Diamond_a \in I$ we have the transitions

$$\begin{array}{ll} (p_c, b, \top, \{(\emptyset, p_c)\}) & \in \Delta \quad \text{(for each } b \neq a\text{)}, \\ (p_c, a, \neg(x \in I), \{(\emptyset, p_c)\}) & \in \Delta. \end{array}$$

4. $(u, a, \top, \{(\emptyset, u)\}) \in \Delta$.

The set of final states F' is given by

$$F' = F \cup \{u\} \cup \{p_c \mid c \text{ is a negative literal in } \mathcal{A}\}.$$

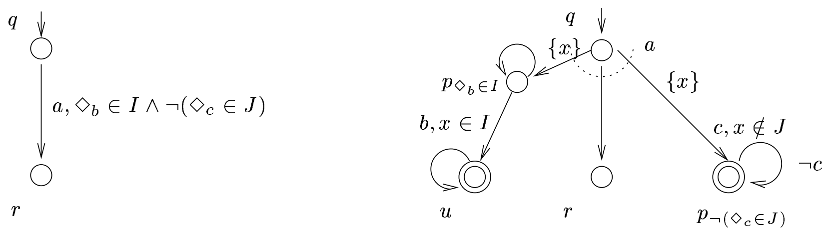

Figure 4: Translation of an ETA transition

Lemma 1 *$L(\mathcal{A}) = L(\mathcal{T})$.* □

Using Theorem 2 and Lemma 1 we conclude that:

Theorem 3 *The emptiness problem for ETA's is decidable.* □

4 Deciding Recursive ETA's

In this section we show how we can decide the emptiness problem for recursive ETA's by showing how they can be simulated by non-recursive ETA's, though over an extended alphabet.

To explain the idea of our "flattening" construction, consider a recursive ETA $\mathcal{A}$ that uses the floating recursive ETA's $\mathcal{B}_0, \ldots, \mathcal{B}_k$ in its guards. We can then consider a timed word over the extended alphabet $\Sigma \times \{0,1\}^{k+1}$ which represents the positions where the automata B_i accept the underlying timed word. Thus if $\sigma = (a_0, t_0) \cdots (a_n, t_n)$ is a timed word over Σ, the canonical extension of σ (wrt $\mathcal{B}_0, \ldots, \mathcal{B}_k$) is the timed word $\bar{\sigma}$ which we represent as $((a_0, v_0), t_0) \cdots ((a_n, v_n), t_n)$ where for each $i \in \{0, \ldots, n\}$ and $j \in \{0, \ldots, k\}$, $v_i[j] = 1$ iff $\mathcal{B}_j$ accepts (σ, i). The figure below shows the canonical extension of the timed word $(a, 0.2)(a, 1.2)(b, 1.4)(c, 2.4)$ with respect to the recursive ETA of Fig. 2.

$\overset{Pos}{\rightarrow}$	0	1	2	3
T	0.2	1.2	1.4	2.4
Σ	a	a	b	c
$\mathcal{B}$	0	0	1	0
$\mathcal{C}$	0	0	0	1

Now the idea is essentially to run a modified version of $\mathcal{A}$, which we call $\hat{\mathcal{A}}$, over the extended timed words, and replace each atomic guard of the form $\diamond_{\mathcal{B}_j} \in I$ in $\mathcal{A}$ by an appropriate non-recursive guard of the form $\diamond_{(a,v)} \in I$ where $a \in \Sigma$ and v is such that $v[j] = 1$. We must further make sure that $\hat{\mathcal{A}}$ does not accept "bad" extended words in which some 0 and 1 guesses are incorrect. For this we use ETA's $\hat{\mathcal{B}}_j^0$ and $\hat{\mathcal{B}}_j^1$ which verify that the 0 entries (respectively the 1 entries) in the j-th row are correct.

The ETA $\hat{\mathcal{B}}_j^0$ which verifies that the 0 entries in the j-th row are correct, works by maintaining two components in its state: the first a state of $\mathcal{B}_j$ and the second a subset of states of $\mathcal{B}_j$. Whenever it gets an (a, v) with $v[j] = 0$ it pretends it has received a 1 entry instead, and pushes the resulting state to its second component. Apart from this, it runs its second component similar to a subset construction, assuming all entries are 0. Thus, if the extended word contained a 0 entry that should have been a 1, at the end of the word $\hat{\mathcal{B}}_j^0$ will have a *final* state of $\mathcal{B}_j$ and it will reject the extended word.

We now give the formal construction of the flattened ETA. Let $\mathcal{A} = (Q, s, \delta, F)$ be a recursive ETA over the alphabet Σ, and let it make use of the recursive floating ETA's $\mathcal{B}_0, \ldots, \mathcal{B}_k$, listed in the order of increasing levels. Without loss of generality (cf. Theorem 1) we assume that each $\mathcal{B}_j$ is deterministic. Let each $\mathcal{B}_j$ have the structure $(Q_j, s_j, \delta_j, F_j)$.

The required flattened ETA $\mathcal{A}'$ over the extended alphabet $\Sigma' = \Sigma \times \{0,1\}^{k+1}$ is defined to be the intersection of the following ETA's as below:

$$\mathcal{A}' = \hat{\mathcal{A}} \cap \bigcap_{j=0}^{k} \hat{\mathcal{B}}_j^0 \cap \bigcap_{j=0}^{k} \hat{\mathcal{B}}_j^1.$$

The ETA $\hat{\mathcal{A}}$ has the same structure as $\mathcal{A}$ and is defined to be $(Q, s, \hat{\delta}, F)$ where the transition relation $\hat{\delta}$ is given as follows. For a guard g in $\mathcal{A}$, we denote by $\hat{g}$ the guard obtained by replacing each occurrence of an atomic guard of the form $\Diamond_{\mathcal{B}_j} \in I$ by the guard

$$\bigvee_{a \in \Sigma, v[j]=1} (\Diamond_{(a,v)} \in I).$$

Now let $(p, a, g, q) \in \delta$. Then we have the transition $(p, (a, v), \hat{g}, q)$ in $\hat{\delta}$ for all v.

The automaton $\hat{\mathcal{B}}_j^0$ is defined as follows. $\hat{\mathcal{B}}_j^0 = (\hat{Q}_j^0, \hat{s}_j^0, \hat{\delta}_j^0, \hat{F}_j^0)$, where:

- $\hat{Q}_j^0 = Q_j \times 2^{Q_j}$
- $\hat{s}_j^0 = (s_j, \emptyset)$
- $\hat{\delta}_j^0$ is given as follows. Let (q, S) be a state of $\hat{\mathcal{B}}_j^0$, with $S = \{q_0, \ldots, q_m\}$. Let $(a, v) \in \Sigma'$ with $v[j] = 0$. Then we have the transition $((q, S), (a, v), h, (r, T))$ in $\hat{\delta}_j^0$ provided the following conditions are satisfied:
  - There exists g such that $(q, (a, 0), g, r) \in \delta_j$.
  - There exists g' such that $(q, (a, 1), g', t) \in \delta_j$.
  - There exist $g_0, \ldots, g_m$ such that $(q_i, (a, 0), g_i, r_i) \in \delta_j$
  - $h = \hat{g}_0 \wedge \cdots \wedge \hat{g_m} \wedge \hat{g} \wedge \hat{g'}$, and,
  - $T = \{r_0, \ldots, r_m\} \cup \{t\}$.

  If $(a, v) \in \Sigma'$ with $v[j] = 1$, then we have the transition $((q, S), (a, v), h, (r, T))$ in $\hat{\delta}_j^0$ provided:
  - There exists g such that $(q, (a, 0), g, r) \in \delta_j$.
  - There exist $g_0, \ldots, g_m$ such that $(q_i, (a, 0), g_i, r_i) \in \delta_j$
  - $h = \hat{g}_0 \wedge \cdots \wedge \hat{g_m} \wedge \hat{g}$, and,
  - $T = \{r_0, \ldots, r_m\}$.

- $\hat{F}_j^0 = Q_j \times 2^{Q_j - F_j}$.

The automaton $\hat{\mathcal{B}}_j^1$ is defined as follows. $\hat{\mathcal{B}}_j^1 = (\hat{Q}_j^1, \hat{s}_j^1, \hat{\delta}_j^1, \hat{F}_j^1)$, where:

- $\hat{Q}_j^1 = Q_j \times 2^{Q_j}$
- $\hat{s}_j^1 = (s_j, \emptyset)$
- $\hat{\delta}_j^1$ is given as follows. Let (q, S) be a state of $\hat{\mathcal{B}}_j^1$, with $S = \{q_0, \ldots, q_m\}$. Let $(a, v) \in \Sigma'$ with $v[j] = 1$. Then we have the transition $((q, S), (a, v), h, (r, T))$ in $\hat{\delta}_j^1$ provided the following conditions are satisfied:
  - There exists g such that $(q, (a, 0), g, r) \in \delta_j$.
  - There exists g' such that $(q, (a, 1), g', t) \in \delta_j$.
  - There exist $g_0, \ldots, g_m$ such that $(q_i, (a, 0), g_i, r_i) \in \delta_j$
  - $h = \hat{g}_0 \wedge \cdots \wedge \hat{g_m} \wedge \hat{g} \wedge \hat{g'}$, and,
  - $T = \{r_0, \ldots, r_m\} \cup \{t\}$.

  If $(a, v) \in \Sigma'$ with $v[j] = 0$, then we have the transition $((q, S), (a, v), h, (r, T))$ in $\hat{\delta}_j^1$ provided:
  - There exists g such that $(q, (a, 0), g, r) \in \delta_j$.
  - There exist $g_0, \ldots, g_m$ such that $(q_i, (a, 0), g_i, r_i) \in \delta_j$
  - $h = \hat{g}_0 \wedge \cdots \wedge \hat{g_m} \wedge \hat{g}$, and,
  - $T = \{r_0, \ldots, r_m\}$.
- $\hat{F}_j^1 = Q_j \times 2^{F_j}$.

To reason about the correctness of the construction, it is convenient to use the notion of a "projection" of an extended word to a standard timed word. Let σ' be a timed word over the extended alphabet Σ'. Let us represent σ' as (w, β, τ) where $w \in \Sigma^*$, $\beta \in (\{0,1\}^{k+1})^*$, and $\tau \in (\mathbb{R}^{\geq 0})^*$. Then the projection of σ' to Σ, denoted $\sigma' \lfloor \Sigma$, is defined to be the timed word (w, τ). We extend the projection operator to work on timed languages over Σ' in the expected way.

Lemma 2 *$\sigma \in L(\mathcal{A})$ iff $\bar{\sigma} \in L(\hat{\mathcal{A}})$ where $\bar{\sigma}$ is the canonical extention σ.*

Proof: We prove that $\sigma, i \models g_i$ iff $\bar{\sigma}, i \models \hat{g}_i$. Without loss of generality we assume that the guard g_i is atomic.

$\Rightarrow$ Let $\sigma, i \models g_i$. There are two cases depending on the guard g_i.
Case 1: Let $g_i = (\Diamond_{\mathcal{B}_j} \in I)$ and $\hat{g}_i = (\bigvee_{(a\in\Sigma)} \Diamond_{(a,v[j]=1)} \in I)$. Since g_i is

satisfied in $\mathcal{A}$ there exists an a at some position l such that $\tau(l) - \tau(i) \in I$ and $(\sigma, l) \in L(\mathcal{B}_j)$ which implies $\bar{\sigma}, i \models \hat{g}_i$.
Case 2: Let $g_i = \neg(\diamond_{\mathcal{B}_j} \in I)$ and $\hat{g}_i = \neg(\bigvee_{(a \in \Sigma)} \diamond_{(a, v[j]=1)} \in I)$. Since g_i is satisfied in $\mathcal{A}$ there does not exists an a at some position l such that $\tau(l) - \tau(i) \in I$ and $(\sigma, l) \in L(\mathcal{B}_j)$ which implies $\bar{\sigma}, i \models \hat{g}_i$.

By construction both $\mathcal{A}$ and $\hat{\mathcal{A}}$ have the same run on σ and $\bar{\sigma}$ respectively. Therefore we conclude that $\sigma \in L(\mathcal{A})$ implies $\bar{\sigma} \in L(\hat{\mathcal{A}})$.

$\Leftarrow$ It can easily be argued along similar lines that if $\bar{\sigma} \in L(\hat{\mathcal{A}})$ then $\sigma \in L(\mathcal{A})$. □

Lemma 2 shows that $\hat{\mathcal{A}}$ accepts the canonical extention of all the timed words in $L(\mathcal{A})$. We now prove that $\mathcal{A}'$ accepts precisely the canonical extention of all the timed words in $L(\mathcal{A})$. Then it follows that $L(\mathcal{A})$ is empty iff $L(\mathcal{A}')$ is empty.

Let σ' be a timed word over the extended alphabet. Let $\bar{\sigma}_j^0$ (and symmetrically $\bar{\sigma}_j^1$) represent an extended time word σ' such that all the guesses in the rows lower than j and all the *zero* guesses in the j-th row of σ' (and respectively *one* guesses in σ') are correct. Let Γ_j^0 (and Γ_j^1) be the set of all the timed words $\bar{\sigma}_j^0$ (and $\bar{\sigma}_j^1$).

Lemma 3 *$\hat{\mathcal{B}}_j^0$ and $\hat{\mathcal{B}}_j^1$ are deterministic.*

Proof: $\Rightarrow$ Recall that we assume the floating automaton $\mathcal{B}_j$ is deterministic. Suppose $\hat{\mathcal{B}}_j^0$ is nondeterministic. This implies there exists a timed word $\bar{\sigma}_j^0$ such that $\bar{\sigma}_j^0, i \models \hat{g}_0$ and $\bar{\sigma}_j^0, i \models \hat{g}_1$ where $\hat{g}_0$ and $\hat{g}_1$ are guards on two different transitions from some state s in $\hat{\mathcal{B}}_j^0$. Then by construction $\sigma, i \models g_0$ and $\sigma, i \models g_1$ where g_0 and g_1 are guards on two different transitions from the state s in $\mathcal{B}_j$ which implies $\mathcal{B}_j$ nondeterministic, a contradiction. □

Lemma 4 *Let $\sigma \in L(\mathcal{B})$ and let σ' be an extended timed word such that all the guesses in the rows lower than j are correct. Then $\sigma' \in L(\hat{\mathcal{B}}_j^0)$ iff $\sigma' \in \Gamma_j^0$.*

Proof: Let S represent the second component of the automaton $\hat{\mathcal{B}}_j^0$.

$\Rightarrow$ Let $\sigma' \in \Gamma_j^0$. Suppose $\sigma' \notin L(\hat{\mathcal{B}}_j^0)$. This implies S contains at least one final state of $\mathcal{B}_j$ on σ'. Since all the *zero* and *one* guesses of lower level automata are assumed to be correct this implies there exists a position l in σ' such that $(\sigma', l) \in L(\mathcal{B}_j)$. Then by construction this implies the guess $v_l[j] = 0$ is incorrect, a contradiction to the fact that $\sigma' \in \Gamma_j^0$.

$\Leftarrow$ It is easy to see that if $\sigma' \notin \Gamma_j^0$ then there is an incorrect guess in σ' and therefore $\sigma' \notin L(\hat{\mathcal{B}}_j^0)$. □

Lemma 5 *Let $\sigma \in L(\mathcal{B})$ and let σ' be an extended timed word such that all the guesses in the rows lower than j are correct. Then $\sigma' \in L(\hat{\mathcal{B}}^1_j)$ iff $\sigma' \in \Gamma^1_j$.*
□

Lemma 6 *Let $\mathcal{B} = \bigcap_{j=0}^{k} \hat{\mathcal{B}}^0_j \cap \bigcap_{j=0}^{k} \hat{\mathcal{B}}^1_j$. Then $\sigma' \in L(\mathcal{B})$ iff σ' is a canonical extention of σ.*

Proof: $\Rightarrow$ Let σ' be a canonical extention of σ. We use induction on the level of automata to prove the lemma. Proof is trivial for the level 0 automaton. Let us assume $\sigma' \in L(\bigcap_{j=0}^{i} \hat{\mathcal{B}}^0_j \cap \bigcap_{j=0}^{i} \hat{\mathcal{B}}^1_j)$ where $i < k$. Since σ' is the canonical extension of σ then by Lemma 4 and by induction hypothesis $\sigma' \in L(\hat{\mathcal{B}}^0_{i+1})$ and by Lemma 5 and by induction hypothesis $\sigma' \in L(\hat{\mathcal{B}}^1_{i+1})$ and hence $\sigma' \in L(\bigcap_{j=0}^{i+1} \hat{\mathcal{B}}^0_j \cap \bigcap_{j=0}^{i+1} \hat{\mathcal{B}}^1_j)$. Therefore by induction $\sigma' \in L(\mathcal{B})$.

$\Leftarrow$ Let $\sigma' \in L(\mathcal{B})$ be an extended timed word such that it is not a canonical extension of σ. This implies there exists a row i such that all the guesses below the i-th row are correct and some guess in the i-th row is incorrect. Then by Lemma 4 and by Lemma 5 $\sigma' \notin L(\hat{\mathcal{B}}^0_i \cap \hat{\mathcal{B}}^1_i)$ which implies $\sigma' \notin L(\mathcal{B})$, a contradiction. □

Using Lemma 2 and Lemma 6 we conclude that:

Lemma 7 *$\sigma \in L(\mathcal{A})$ iff $\bar{\sigma} \in L(\mathcal{A}')$ where $\bar{\sigma}$ is the canonical extention of σ.*□

Using Lemma 7 we conclude that:

Theorem 4 *The recursive ETA $\mathcal{A}$ accepts the projection of the language accepted by the ETA $\mathcal{A}'$, i.e $L(\mathcal{A}) = L(\mathcal{A}') \lfloor \Sigma$.* □

From Theorem 3 and Theorem 4 we conclude that:

Theorem 5 *The emptiness problem for recursive ETA's is decidable.* □

5 Expressiveness of ETA's

In this section we compare the expressiveness of the class of recursive ETA's with a couple of well known classes of timed automata in the literature. We show that recursive ETA's are incomparable in expressiveness with Alur-Dill timed automata, and that they are strictly more expressive than the class of recursive Event Predicting Automata (EPA's) of [HRS98].

The timed language L_0 of section 2 was shown to be definable by an ETA whereas it is not definable by an Alur-Dill timed automaton. Conversely, consider the language L_{ni} (for "no insertions") over the alphabet $\{a, b\}$, in which for any two consecutive a's the corresponding time period translated by one time unit contains no events. The complement of this language is definable by a non-deterministic timed automaton. However, as shown in [DP05], the language L_{ni} is not definable by a 1-clock alternating timed automaton (and hence neither is its complement, since this class is closed under complement). Recursive ETA's can be seen to be a subclass of 1-clock ATA's using Theorem 4 and the fact that 1-clock ATA's are closed under projection. It then follows that recursive ETA's cannot define the complement of L_{ni} either. Thus the class of recursive ETA's is incomparable to the class of Alur-Dill timed automata.

The class of recursive Event Predicting Automata [AFH96, HRS98] can be seen to be strictly contained in the class of recursive ETA's. Recall that EPA's use an input determined operator $\rhd_a$ which measures the distance to the *next* a event. A level 1 EPA guard $\rhd_a \in I$ can be simulated by the ETA guard $\Diamond_a \in I \wedge \neg(\Diamond_a \in J)$ and $\neg(\rhd_a \in I)$ can be simulated by the ETA guard $\neg(\Diamond_a \in I \wedge \neg(\Diamond_a \in J))$, where the interval J is of the form $(0, l)$ if $I = [l, r\rangle$, and $(0, l]$ if $I = (l, r\rangle$ and $'\rangle'$ is either $')'$ or $']'$. We now show that recursive EPA can be simulated by a recursive ETA and hence the class is contained in the class of recursive ETA. We use induction on the level of automata to prove the result. The base case is level 0 automaton and therefore the proof is trivial. So let us assume that the claim is true for level i. Let $\mathcal{A}$ be a level $i+1$ EPA and let $\mathcal{B}$ be a level $j < i$ floating EPA. Let $\hat{\mathcal{B}}$ be the ETA which simulates $\mathcal{B}$. We construct a level $i+1$ ETA $\hat{\mathcal{A}}$ from $\mathcal{A}$ by replacing all the guards $\rhd_{\mathcal{B}} \in I$ in $\mathcal{A}$ with $\Diamond_{\hat{\mathcal{B}}} \in I \wedge \neg(\Diamond_{\hat{\mathcal{B}}} \in J)$ and $\neg(\rhd_{\mathcal{B}} \in I)$ with $\neg(\Diamond_{\hat{\mathcal{B}}} \in I \wedge \neg(\Diamond_{\hat{\mathcal{B}}} \in J))$. Let σ be a timed word. It is easy to see that $(\sigma, i) \models \rhd_{\mathcal{B}} \in I$ iff there exists $j \geq i$ such that $(\sigma, j) \in L(\mathcal{B})$ and $\tau(j) - \tau(i) \in I$ and $\forall k : i \leq k < j$ $(\sigma, k) \notin L(\mathcal{B})$. This implies $(\sigma, i) \models \rhd_{\mathcal{B}} \in I$ iff $(\sigma, i) \models \Diamond_{\hat{\mathcal{B}}} \in I \wedge \neg(\Diamond_{\hat{\mathcal{B}}} \in J)$. Similarly $(\sigma, i) \models \neg(\rhd_{\mathcal{B}} \in I)$ iff $(\sigma, i) \models \neg(\Diamond_{\hat{\mathcal{B}}} \in I \wedge \neg(\Diamond_{\hat{\mathcal{B}}} \in J))$. Hence recursive EPA $\subseteq$ recursive ETA.

A recursive EPA can be flattened to a projection equivalent non recursive EPA which runs over an extended alphabet. A non recursive EPA is shown to a subclass of Alur-Dill timed automata[AFH94]. Since timed automata are closed under projection this implies there exists a timed automaton which accepts the langauge accepted by the recursive ETA. Therefore the class of ETA's is strictly contained in the class of ETA's. The figure below summarises the arguments of this section:

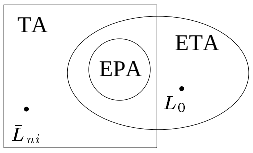

Figure 5: Expressiveness of recursive ETA's, EPA's and timed automata

6 Decision procedures for the dual of $\diamond$

In this section we consider the class of IDA's obtained by considering the dual operator $\diamondminus$. The semantics of the operator is given as follows:

$$[\![\diamondminus_a]\!](\sigma, i) = \{\tau(i) - \tau(j) \mid \sigma = (w, \tau),\ j \leq i,\ \text{and } w(j) = a\}.$$

We denote this class IDA($\diamondminus$). We prove that the emptiness problem of the class of IDA($\diamondminus$) over finite words is decidable by reducing it to the emptiness problem of ETA's.

We now define the notion of the *reverse* of a timed word. Let $\sigma = (a_0, t_0) \cdots (a_n, t_n)$ be a timed word. We define the set of reversed timed words σ^R corresponding to σ as follows:

$$\sigma^R = \{(a_n, d - t_n) \cdots (a_0, d - t_0) \mid d > t_n\}.$$

For a timed language $L \subseteq T\Sigma^*$ we define $L^R = \bigcup_{\sigma \in L} \sigma^R$.

Let g be a $\diamondminus$-guard, and let us denote by g^R the $\diamond$-guard obtained from g by replacing each atomic guard $\diamondminus_a \in I$ in g by the atomic guard $\diamond_a \in I$. We now construct an ETA $\mathcal{A}^R$ from the given IDA($\diamondminus$) $\mathcal{A}$ by replacing every guard g in $\mathcal{A}$ with the guard g^R, reversing the direction of all the edges, and interchanging the start and the final states. For convenience we allow $\mathcal{A}^R$ to have multiple start states.

Let $\sigma = (w, \tau)$ be a timed word and $\sigma' = (w', \tau')$ be such that $\sigma' \in \sigma^R$. It is easy to see that $\tau'(n - k) = d - \tau(k)$.

Lemma 8 $\sigma, i \models g_i$ *then* $\sigma', (n - i) \models g^R_{n-i}$.

Proof: Without loss of generality we assume that the guard g is atomic. Let $g = (\diamondminus_a \in I)$ and $\sigma, i \models g_i$. This implies that there exists an a at some position j such that $\tau(i) - \tau(j) \in I$ which in turn implies the guard $g^R = (\diamond_a \in I)$ is satisfied at the position $n - i$ since $\tau'(n - j) - \tau'(n - i) = \tau(i) - \tau(j) \in I$. Similarly for the guard $g = (\diamondminus_a \in I)$. Therefore $\sigma, i \models g_i$ implies $\sigma', n - i \models g^R_{n-i}$. $\square$

Lemma 9 $\sigma, i \models g_i^R$ *then* $\sigma', (n-i) \models g_{n-i}$.

Proof: Without loss of generality we assume that the guard g is atomic. Let $g^R = (\Diamond_a \in I)$ and let $\sigma, i \models g_i^R$. This implies that there exists an a at some position j such that $\tau'(j) - \tau'(i) \in I$ which in turn implies the guard $g = (\diamondminus_a \in I)$ is satisfied at the position $n-i$ since $\tau(n-i) - \tau(n-j) = (d - \tau'(i)) - (d - \tau'(j)) \in I$. Similarly for the guard $g = (\Diamond_a \in I)$. Therefore $\sigma, i \models g_i^R$ implies $\sigma', n-i \models g_{n-i}$. $\Box$

Given an IDA($\diamondminus$) $\mathcal{A} = \langle Q, s, \delta, F \rangle$ we construct an ETA $\mathcal{A}^R = \langle Q, Q_0, \delta^R, G \rangle$ where

- $Q_0 = F$
- $G = \{s\}$
- $\delta^R = \{(t, a, g^R, s) \mid (s, a, g, t) \in \delta\}$

Let σ be a timed word and let $\sigma' \in \sigma^R$. Then from Lemma 8 and by construction of $\mathcal{A}^R$ we conclude that:

Lemma 10 $\sigma \in L(\mathcal{A}) \Rightarrow \sigma' \in L(\mathcal{A}^R)$. $\Box$

Again from Lemma 9 and by construction of $\mathcal{A}^R$ we conclude that:

Lemma 11 $\sigma \in L(\mathcal{A}^R) \Rightarrow \sigma' \in L(\mathcal{A})$. $\Box$

Thus $\mathcal{A}^R$ preserves the language emptiness of $\mathcal{A}$. In general, it can be seen that $L(\mathcal{A}^R) = L(\mathcal{A})^R$. Since the emptiness problem for ETA's is decidable we conclude that:

Theorem 6 *The emptiness problem for IDA($\diamondminus$) is decidable.* $\Box$

It is easy to see that recursive IDA($\diamondminus$) can be flattened in a similar manner to ETA's. Therefore:

Theorem 7 *The emptiness problem for recursive IDA($\diamondminus$) is decidable.* $\Box$

7 Undecidability of IDA($\Diamond, \ominus$)'s

In this section we show that the emptiness problem for IDA($\Diamond, \ominus$) is undecidable by reducing the halting problem of 2 counter machine on an empty tape to it. The halting problem for a 2 counter machine M on the empty tape is problem of deciding whether M halts on the empty tape which is shown to be undecidable.

A 2 counter machine M has a read only input tape, two non negative counters (say C and D) which can be incremented or decremented independently and a sequence of n instructions where each instruction does one of the following.

- Increments the counter C (or D) and jumps to another instruction.
- Decrements the counter C (or D) and jumps to another instruction.
- Checks for a zero counter value for C (or D) and jumps to the instruction j if the counter value is zero or jumps to the instruction k if it is nonzero.

The machine M also has a halt instruction on the execution of which M halts. We assume that the halt instruction is the $n+1$-th instruction. A computation of M is a sequence of triplets of the form $\langle i, c, d\rangle$, $i \in \{1, \ldots, n+1\}$ where i is the instruction counter, and c and d are non negative counter values of the counters C and D respectively.

We encode computations of the machine for the reduction. Our encoding of computations of M is same as the encoding given in [AD94]. A computation is encoded as a timed word σ over the alphabet $\Sigma = \{b_1, \ldots, b_n, b_h, a_1, a_2\}$ where b_i, $i \in \{1, \ldots, n\}$, denotes the instruction i and b_h denotes the halt instruction. a_1 and a_2 are used to encode the values of the counters C and D respectively. A configuration $\langle i, c, d\rangle$ is represented as $b_i a_1^c a_2^d$. The successive configurations are concatenated to represent a computation. The time sequence associated with σ is used to express the consecution property. We require that l-th configuration of a valid computation occurs in the interval $[l, l+1)$.

We also denote a timed word $\sigma \in T\Sigma^\omega$ by (α, τ) where $\alpha \in \Sigma^\omega$. σ encodes a valid halting computation of M iff it satisfies the following conditions.

- C_0 - α is of the form $b_1(b_i a_1^* a_2^*)^* b_h \beta, \beta \in \Sigma^\omega$
- C_1 - Each event in $\{b_1, ..., b_n, b_h\}$ occurs at integral position relative to b_1 and relative to b_1, at every integral position an event in $\{b_1, ..., b_n, b_h\}$ occurs. We assume that the machine always executes the instruction number *one* first.

- C_2 - For every a_1 (and a_2) except for the last one, occurring at time t in the interval $[l, l+1)$ there exists an a_1 (and a_2) at time $t+1$ in the interval $[l+1, l+2)$.

- C_3 - For every a_1 (and a_2) except for the last one, occurring at time t in the interval $[l+1, l+2)$ there exists an a_1 (and a_2) at time $t-1$ in the interval $[l, l+1)$.

- C_4 - $D_1 \wedge ... \wedge D_n$ where D_i says that the instruction i is carried out correctly. Depending on the instruction i following cases arise. We assume that the instruction i is executed in the interval $[l, l+1)$.

  - If the instruction i increments the counter C (or D) and jumps to the instruction j then for every a_1 (or a_2) occurring at time t in the interval $[l, l+1)$ there exists an a_1 (or a_2) at $t+1$. Also b_j should succeed b_i exactly one time unit later.
  - If the instruction i decrements the counter C (or D) and jumps to the instruction j then for every a_1 (or a_2) occurring at time t in the interval $[l+1, l+2)$ there exists an a_1 (or a_2) at $t-1$. Also b_j should succeed b_i exactly one time unit later.
  - If the instruction i is a branch instruction and can jump to the instruction j or k depending on the value of the counter then conditions C_2 and C_3 should hold. Also either b_j or b_k should succeed b_i exactly one time unit later.

- We also impose the following two conditions on the valid computations of M.

  - For every instruction i the next instruction executed by M is $j \neq i$.
  - For every instruction i if the next instruction executed by M is h then only the condition C_1 needs to hold.

Let L be the set of all $\sigma \in T\Sigma^\omega$ such that $\sigma = (\alpha_0, \tau_0)(b_h, t)(\alpha_1, \tau_1)$ and $(\alpha_0, \tau_0)(b_h, t)$ encodes a valid halting computation of M, i.e $L = \{\sigma \mid (\alpha_0, \tau_0)(b_h, t)$ satisfies $\bigwedge_{i=0}^{4} C_i\}$. It is easy to see that L is non empty iff M has a valid halting computation.

For the easy understanding of the automata given below we will restate the five conditions (C_0 to C_4) mentioned above as follows. $\sigma \in L$ iff it satisfies the following conditions.

- E_0 - α should begin with b_1, i.e initial configuration of a valid computation of M must be $\langle 1, 0, 0 \rangle$. $L(E_0)$ is accepted by $\mathcal{A}^0$.

- E_1 - Each subsequence $b_i a_1^{c1} a_2^{d1} b_j a_1^{c2} a_2^{d2}$ should be valid. $L(E_1)$ is accepted by $\bigcap_{i=1}^{n} \mathcal{A}^i$.
- E_2 - The sequence α should have an b_h. $L(E_2)$ is accepted by $\mathcal{A}^h$.

Then the language $L = L(E_0) \cap L(E_1) \cap L(E_2)$ is precisely set of all the timed words encoding the valid halting computations of M. We now discuss the construction of automata, the intersection of which accepts the language L.

Any valid computation of M will have the initial configuration $\langle 1, 0, 0 \rangle$. The automaton $\mathcal{A}^0$ shown in Fig.6 ensures that the initial configuration of M is $\langle 1, 0, 0 \rangle$. We assume that the instruction i is the only valid successor of the first instruction.

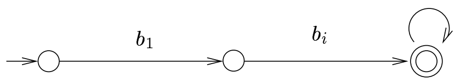

Figure 6: Automaton $\mathcal{A}^0$

Now we will construct automata $\mathcal{A}^i$, $i \in \{1, \ldots, n\}$, such that $L(E_1) = \bigcap_{i=1}^{n} \mathcal{A}^i$. We observe that the transitions on b_h in $\mathcal{A}^i$ which is to the final state s_f^i is suppressed for brevity. Once the control reaches the s_f^i automaton $\mathcal{A}^i$ loops in s_f^i for all subsequent executions. Note that these automata are instruction dependent.

Let $i \neq n+1$ be an instruction which increments the counter C and jumps to the instruction j. The automaton $\mathcal{A}^i$ in Fig.7 ensures that the instruction i is executed properly.

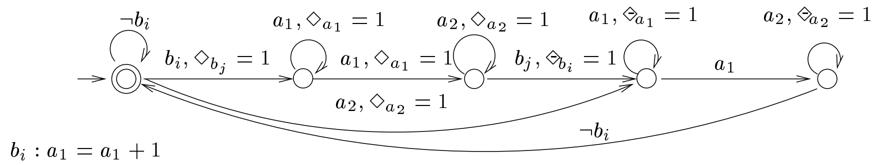

Figure 7: Automaton $\mathcal{A}^i$ when i increments

If i decrements counter the C then the automaton $\mathcal{A}^i$ is as shown in Fig.8. Here we assume $c > 0$. Automaton handling the case $c = 0$ can be similarly constructed.

Automaton $\mathcal{A}^h$ shown in Fig.9 accepts σ which has an b_h in it.

The automaton for other cases can be constructed along the similar lines. It is easy to see that $L = L(\mathcal{A}^0 \cap \mathcal{A}^h \cap \bigcap_{i=1}^{n} \mathcal{A}^i)$. Since IDA($\diamond, \ominus$) is closed under intersection[DT04] we conclude that:

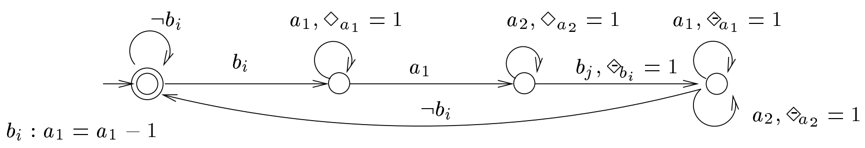

Figure 8: Automaton $\mathcal{A}^i$ when i decrements

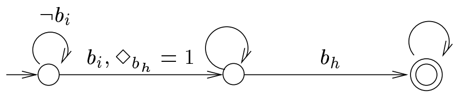

Figure 9: Automaton $\mathcal{A}^h$

Theorem 8 *The emptiness problem for IDA($\Diamond, \diamondminus$) is undecidable.* $\square$

The universality problem for an automaton is problem of deciding whether is accepts all the timed words over Σ. We show that for IDA($\Diamond, \diamondminus$) the universality problem is also undecidable.

Theorem 9 *The universality problem for IDA($\Diamond, \diamondminus$) is undecidable.*

Proof: Theorem 9 follows from Theorem 8 and from the fact that IDA($\Diamond, \diamondminus$) is closed under complementation[DT04]. $\square$

References

[AD94] R. Alur and D. Dill: A theory of timed automata, *Theoretical Computer Science* **126**, Elsevier, 183–235 (1994).

[AFH96] R. Alur, T. Feder and T.A. Henzinger: The benefits of relaxing punctuality, *J. ACM* **43**, 116–146 (1996).

[AFH94] R. Alur, L. Fix, T. A. Henzinger: Event-clock automata: a determinizable class of timed automata, *Proc. 6th CAV,* LNCS **818**, 1–13, Springer-Verlag (1994).

[DM05] D. D'Souza and Raj Mohan M: Eventual Timed Automata, IISc-CSA-TR-2005-8, Technical Report, Indian Institute of Science, Bangalore (2005).

[DP05] D. D'Souza and P. Prabhakar: On the expressiveness of MTL in the pointwise and continuous semantics, IISc-CSA-TR-2005-7, Technical Report, Indian Institute of Science, Bangalore (2005).

[DT04] D. D'Souza and N. Tabareau: On timed automata with input-determined guards, *Proc. FORMATS-FTRTFT*, LNCS **3253** (2004).

[HRS98] T. Henzinger, J. Raskin and P. Schobbens: The regular real-time languages, *Proc. 25th ICALP*, LNCS **1443**, 580–591 (1998).

[Koz97] D. Kozen: Automata and Computability, *Springer-Verlag* (1997).

[Koy90] R. Koymans: Specifying real-time properties with metric temporal logic, *Real Time Systems*, Vol. **2**, No.4, 255–299 (1990).

[LW05] S. Lasota and I. Walukiewicz: Alternating timed automata, *Proc. FOSSACS 2005* (2005).

[OW05] J. Ouaknine and J. Worrell: On the decidability of metric temporal logic, *Proc. LICS 2005* (2005).