On the expressiveness of MTL in the pointwise and continuous semantics

Deepak D'Souza Pavithra Prabhakar

IISc-CSA-TR-2005-7
http://archive.csa.iisc.ernet.in/TR/2005/7/

Computer Science and Automation
Indian Institute of Science, India

May 2005

On the expressiveness of MTL in the pointwise and continuous semantics

Deepak D'Souza and Pavithra Prabhakar
Department of Computer Science and Automation,
Indian Institute of Science,
Bangalore 560012, India.
Email: {deepakd,pavithra}@csa.iisc.ernet.in

Abstract

The Metric Temporal Logic (MTL) introduced by Koymans [7] has been interpreted in two ways in the literature: the "pointwise" version and the "continuous" version. We show that the pointwise version of the logic is strictly less expressive than the continuous version. The proof is constructive in the sense that we exhibit a timed language which is definable in the continuous semantics but is not definable in the pointwise semantics. This result holds for the case where we consider *finite* timed models. The technique used essentially exploits the fact that the pointwise logic is decidable [10] while the continuous logic is undecidable.

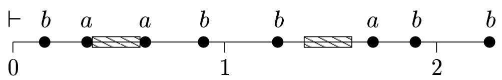
1 Introduction

The timed temporal logic *Metric Temporal Logic* (MTL) [7] has received much attention in the literature on the verification of real-time systems. MTL extends the until operator of classical temporal logic with an interval which specifies the time distance within which the formula must be satisfied. Over dense time the logic has traditionally been interpreted in either of two ways which have come to be known as the "pointwise" and the "continuous" semantics. In the pointwise version temporal assertions are interpreted only at time points where an "action" or "event" happens in the observed timed behaviour of a system, whereas in the continuous version one is allowed to assert formulas at arbitrary time points between events as well. For instance consider a timed word comprising two events: an a which happens at time 1 and a b which occurs at time 3. Then the MTL formula $\Diamond_{[1,1]}b$ (a "b occurs at a distance of 1 time unit") is not true at any point in this model in the pointwise semantics, since there is no action point from which the action b happens at a distance of 1 time unit. However in the continuous semantics the formula is true at the time instant 2 in the model since at this point the event b occurs at a time distance of 1.

There have been a variety of results regarding decision procedures and to some extent expressiveness of the logic in these two semantics. Alur, Feder, and Henzinger [3, 2] argue that the logic is undecidable in the continuous semantics, though a restriction to non-singular intervals (called "Metric Interval Temporal Logic", or MITL) is shown to be decidable. These results hold for both finite and infinite models. Recently Ouaknine and Worrell [10] have shown that MTL over finite words in the pointwise semantics is decidable, using a translation to one-clock alternating timed automata which in turn was recently introduced and shown to be decidable ([8, 10]). There have also been a few expressiveness results regarding MTL, namely expressive completeness with respect to a class of "input-determined" timed automata in the pointwise semantics [4], and expressive equivalence of MITL with a recursive event-clock logic in the continuous semantics [5].

However the issue of the relative expressiveness of the pointwise and continuous semantics has not been addressed in the literature. One expects the continuous version to be at least as expressive as the pointwise one since we can characterise the action points in the continuous semantics, and hence mimic the pointwise semantics by restricting the interpretation only to these points. Intuitively, one would also expect the continuous version to be strictly more expressive, given its ability to quantify over arbitrary time points.

In this paper we confirm this intuition for the case of finite models by exhibiting a language of finite timed words which is definable in the continuous semantics but *not* in the pointwise semantics. We begin by first giving a way of interpreting the continuous version of the logic over timed words (we note that the continuous version was originally interpreted over timed state sequences or "signals", in which the boolean states are assumed to carry through till the next transition point). The semantics we give is fairly natural if we view a system's behaviour as a sequence of events and their associated time stamps. This gives us a common set of models to compare the expressiveness of the two logics. It is then easy to see that pointwise assertions can be captured in the continuous semantics. We then show the strict containment by considering the timed 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.



This language can easily be seen to be expressible in the continuous semantics. However, we argue that it is not expressible in the pointwise semantics, since it would essentially lead to the undecidability of the pointwise version of the logic, which would contradict the result of [10].

In general, this technique – i.e. of obtaining an inexpressiveness result for a decidable class of languages which lies on the borderline of undecidability by identifying a property whose expressibility in the class would lead to a reduction from an undecidable problem – seems to be quite fruitful in obtaining similar results in the setting of timed languages. For example, a similar argument can be used to argue that the complement of the language L_{ni} cannot be accepted by an alternating timed automaton with one clock, while it can be by a classical Alur-Dill timed automaton.

We should be careful to point out that in such arguments it is important that the property which we claim to be inexpressible, is in a sense, independent of the alphabet used for encoding instances of the undecidable problem. In fact such a property may not always exist. This is borne out by the fact that there are logics [9] with two equally expressive versions, but one of whose satisfiability problem is decidable while the others isn't.

2 Syntax and semantics of MTL

We begin with some preliminary definitions. As usual, A^* will denote the set of finite words over an alphabet A. For a finite word $\alpha = a_0 a_1 \cdots a_n$ we use $|\alpha|$ to denote the length of α (in this case $n+1$), and $\alpha(i)$ to denote a_i, the i-th symbol of α.

For convenience we define timed words with a special left-end marker '$\vdash$' which we assume does not occur in any of the alphabets we consider. A (finite) *timed word* σ over an alphabet Σ is an element of $((\Sigma \cup \{\vdash\}) \times \mathbb{R}_{\geq 0})^*$ of the form $(\vdash, 0)(a_1, t_1) \cdots (a_n, t_n)$, satisfying $t_i < t_{i+1}$ for all $0 \leq i < n$. Wherever convenient we will also represent a timed word $\sigma = (a_0, t_0)(a_1, t_1)...(a_n, t_n)$ as a pair $\sigma = (\alpha, \tau)$, where $\alpha = a_0 a_1 \cdots a_n$, and $\tau = t_0 t_1 \cdots t_n$. We use $time(\sigma)$ to denote the time of the last action, namely t_n. We write $T\Sigma^*$ for the set of timed words over Σ.

We now give the syntax and semantics of the two versions of MTL. The models for both versions will be timed words. For ease of comparison we also use a common syntax for the two logics, using an until operator which is "strict" in its first argument. While the operator is the standard one used in the continuous version, the more natural operators of the pointwise semantics ("next" and a non-strict "until") continue to be expressible using this until operator. This is shown later.

The formulas of MTL over an alphabet Σ are built up from symbols in Σ and $\vdash$ by boolean connectives and a time-constrained version of the *until* operator U. The formulas of MTL over an alphabet Σ are inductively defined as follows:

$$\varphi \; := \top \mid a \mid \neg\varphi \mid (\varphi \vee \varphi) \mid (\varphi \; U_I^s \, \varphi),$$

where $a \in (\Sigma \cup \{\vdash\})$ and I is an interval with rational end points.

We first define the *pointwise* semantics for MTL. Given an MTL formula φ, a timed word $\sigma = (\alpha, \tau)$, and a position i in the timed word (i.e. $0 \leq i < |\sigma|$), the satisfaction relation $\sigma, i \models_{pw} \varphi$ (read "σ at position i satisfies φ in the pointwise semantics") is inductively defined as:

$$\begin{array}{lll}
\sigma, i \models_{pw} \top. & & \\
\sigma, i \models_{pw} a & \text{iff} & \alpha(i) = a. \\
\sigma, i \models_{pw} \neg\varphi & \text{iff} & \sigma, i \not\models_{pw} \varphi. \\
\sigma, i \models_{pw} \varphi_1 \vee \varphi_2 & \text{iff} & \sigma, i \models_{pw} \varphi_1 \text{ or } \sigma, i \models_{pw} \varphi_2. \\
\sigma, i \models_{pw} \varphi_1 \; U_I^s \, \varphi_2 & \text{iff} & \exists j \geq i \,\text{such that}\, \tau(j) - \tau(i) \in I,\; \sigma, j \models_{pw} \varphi_2, \\
& & \;\text{and } \forall k \text{ such that } i < k < j,\; \sigma, k \models_{pw} \varphi_1.
\end{array}$$

The language defined by an MTL formula φ in pointwise semantics is given by:

$$L^{pw}(\varphi) = \{\, \sigma \in T\Sigma^* * \mid \sigma, 0 \models_{pw} \varphi \}.$$

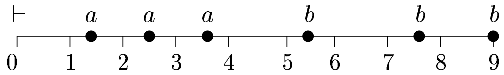
We will use MTL^{pw} to denote the pointwise interpretation of this logic.

The pointwise version of MTL is usually given in terms of a "next" operator and a non-strict "until" similar to classical LTL, with the semantics:

$$\begin{array}{lll}
\sigma, i \models_{pw} O\varphi & \text{iff} & \sigma, i+1 \models \varphi \\
\sigma, i \models_{pw} \varphi_1 \; U_I \, \varphi_2 & \text{iff} & \exists j \geq i \,\text{such that}\, \tau(j) - \tau(i) \in I,\; \sigma, j \models_{pw} \varphi_2 \\
& & \text{and}\, \forall k \,\text{such that}\; i \leq k < j,\; \sigma, k \models_{pw} \varphi_1.
\end{array}$$

It is easy to see that these operators can be expressed using the U_I^s operator as below:

$$\begin{array}{lll}
O\varphi & \equiv & \bot U^s_{(0,\infty)} \varphi. \\[1em]
\varphi_1 U_I \varphi_2 & \equiv & \left\{ \begin{array}{ll} \varphi_1 \wedge (\varphi_1 U_I^s \varphi_2) & \text{if} \;\; 0 \notin I. \\ \varphi_2 \vee (\varphi_1 \wedge (\varphi_1 U_I^s \varphi_2)) & \text{if} \;\; 0 \in I. \end{array} \right.
\end{array}$$



A timed word

The derived operators $\Diamond_I$, $\Box_I$, U, $\Diamond$, and $\Box$ (defined in the expected manner below) are thus also expressible in the pointwise interpretation, and we will freely make use of them in the sequel.

$$
\begin{array}{lcl}
\Diamond_I \varphi & = & \top U_I \varphi, \\
\Box_I \varphi & = & \neg \Diamond_I \neg \varphi, \\
\varphi_1 U \varphi_2 & = & \varphi_1 U_{[0,\infty)} \varphi_2, \\
\Diamond \varphi & = & \top U \varphi, \\
\Box \varphi & = & \neg \Diamond \neg \varphi.
\end{array}
$$

We can note that the operator U_I^s in turn can be expressed using the O and U_I operators. This is done as follows:

$$
\varphi_1 U_I^s \varphi_2 \equiv \begin{cases}
\Diamond_I \varphi_2 \wedge \Box_{(0,a)}(\varphi_1 U_{[0,\infty)}(\varphi_1 \wedge O\varphi_2)) & \text{if } I = [a,b), a > 0. \\
\Diamond_I \varphi_2 \wedge \Box_{(0,a]}(\varphi_1 U_{[0,\infty)}(\varphi_1 \wedge O\varphi_2)) & \text{if } I = (a,b), a > 0. \\
\Diamond_I \varphi_2 \wedge (\varphi_2 \vee O(\varphi_1 U_{[0,\infty)} \varphi_2)) & \text{if } I = [0,b). \\
\Diamond_I \varphi_2 \wedge O(\varphi_1 U \varphi_2) & \text{if } I = (0,b).
\end{cases}
$$

Here are a couple of examples of MTL formulas and their pointwise interpretations. Consider the timed word $\sigma = (\vdash, 0)(a, 1.3)(a, 2.5)(a, 3.6)(b, 5.3)(b, 7.6)(b, 9)$ depicted below.

1. $aU^s_{[5,6]}b$: The timed word in the above figure satisfies this formula, since b is true at some time in the interval [5,6] and a is true at all the action points before that.

2. $aU^s_{[6,7]}(a \vee b)$: This formula is not satisfied by the above timed word, since there is neither an occurrence of a nor that of b anywhere in the interval [6,7].

We now turn to the *continuous* semantics for MTL. The continuous version of the logic is typically interpreted over timed state sequences (e.g. in [2]), where a system is assumed to continue in a state till the next transition or state change takes place. Here we interpret the logic over timed words, while keeping intact the essence of the continuous interpretation, namely the ability to assert formulas at arbitrary time points. A minor point of departure here is that the atomic formula a in our interpretation can only hold at an action point labelled a, and not during an interval of time after the event happens as in the case of timed state sequences. This view is appropriate since we are talking about event occurrences and not propositions that make up a state.

Given an MTL formula φ, a timed word $\sigma = (\alpha, \tau)$, and a time t such that $0 \leq t \leq time(\tau)$, the satisfaction relation $\sigma, t \models_c \varphi$ (read "σ at time t satisfies φ in the continuous semantics") is inductively defined as follows:

$$
\begin{array}{lll}
\sigma, t \models_c \top. & & \\
\sigma, t \models_c a & \text{iff} & \exists i \text{ such that } \tau(i) = t \text{ and } \alpha(i) = a. \\
\sigma, t \models_c \neg\varphi & \text{iff} & \sigma, t \not\models_c \varphi. \\
\sigma, t \models_c \varphi_1 \vee \varphi_2 & \text{iff} & \sigma, t \models_c \varphi_1 \text{ or } \sigma, t \models_c \varphi_2. \\
\sigma, t \models_c \varphi_1 \; U_I^s \, \varphi_2 & \text{iff} & \exists t' \text{ such that } t' \geq t,\ t' - t \in I,\ \sigma, t' \models_c \varphi_2 \\
 & & \quad \text{and } \forall t'' \text{ such that } t < t'' < t',\ \ \sigma, t'' \models_c \varphi_1.
\end{array}
$$

The language defined by an MTL formula φ in the continuous semantics is denoted $L^c(\varphi)$ and defined as

$$L^c(\varphi) = \{\, \sigma \in T\Sigma^* \mid \sigma, 0 \models_c \varphi \}.$$

We will use MTL^c to denote this continuous interpretation of the MTL formulas.

Here are a couple of examples of MTL formulas and their interpretations in the continuous semantics:

1. $a \, U_{[5,6]}^s \, b$: The timed word in figure 2 does not satisfy this formula. In fact, there is no model which satisfies this formula, since it requires that b is true at some point in the interval [5,6] (in fact an action point), and a is true at all the previous points (not just the action points). However, the formula $(\neg \bigvee_{d \in \Sigma} d \vee a) U_{[5,6]}^s b$ (which defines in the continuous semantics, the language defined by the above formula in pointwise semantics) is satisfied by the above timed word, as it asks for the satisfaction of b at some action point and the satisfaction of a at all the previous action points.

2. $\top U_{[4,5]}^s (\top U_{[1,1]}^s b)$: The timed word in figure 2 satisfies this formula in continuous semantics, since we have a point in the interval [4,5] such that there is a b at distance one from this point. But the timed word does not satisfy this formula in the pointwise semantics as there is no action point in the interval [4,5] (and hence no action point which satisfies $(\top U_{[1,1]}^s b)$.

3 Expressing MTL^{pw} in MTL^{c}

In this section we show that MTL^c is at least as expressive as MTL^{pw}. By the expressiveness of a logic, we mean the class of languages definable by the formulas of the logic. We present a function *pw-c* which maps an MTL formula φ to an MTL formula $pw\text{-}c(\varphi)$ such that $L^{pw}(\varphi) = L^c(pw\text{-}c(\varphi))$. Hence any language definable in MTL^{pw} is also definable in MTL^c. The formula φ_{action} is a shorthand for $\bigvee_{a\in(\Sigma\cap\{\vdash\})} a$ and captures the notion of an action point in a timed word.

$$
\begin{array}{lcl}
pw\text{-}c(a) & = & a \text{ where } a \in (\Sigma \cup \{\vdash\}). \\
pw\text{-}c(\neg\varphi_1) & = & \neg pw\text{-}c(\varphi_1). \\
pw\text{-}c(\varphi_1 \vee \varphi_2) & = & pw\text{-}c(\varphi_1) \vee pw\_c(\varphi_2). \\
pw\text{-}c(\varphi_1 U_I^s \varphi_2) & = & (\neg\varphi_{action} \vee pw\text{-}c(\varphi_1)) U_I^s (\varphi_{action} \wedge pw\text{-}c(\varphi_2)).
\end{array}
$$

In the next section it will be convenient to make use of the pointwise operators O, $\Diamond_I$, and $\Box_I$ as MTL^c operators. We denote these operators O^p, $\Diamond_I^p$, and $\Box_I^p$ respectively and define them as:

$$
\begin{array}{lcl}
O^p\varphi & = & pw\text{-}c(O\varphi). \\
\Diamond_I^p\varphi & = & pw\text{-}c(\Diamond_I\varphi). \\
\Box_I^p\varphi & = & pw\text{-}c(\Box_I\varphi).
\end{array}
$$

4 Undecidability of MTL^c

In this section we give a detailed proof of undecidability of the satisfiability problem for MTL^c. (The satisfiability problem for MTL^c is to decide for a given MTL^c formula φ whether $L^c(\varphi)$ is non-empty or not.) This result is sketched in [3] and [2]. The reduction we give in the proof below is essentially the same as the the ones used in [3] and [1]. However, the encoding has been adapted for our specific purpose of proving an inexpressibility result for the pointwise version of the logic in the next section.

Theorem 4.1 ([3, 2]) *The satisfiability problem for MTL^c is undecidable.*

Proof The problem of deciding whether a deterministic 2-counter machine has a halting computation on an empty tape is known to be undecidable. We reduce this problem to the satisfiability problem of MTL^c. We give an algorithm which given a deterministic 2-counter machine M outputs an MTL formula φ_M such that φ_M is satisfiable in the continuous semantics for MTL if and only if M has a halting computation.

A *deterministic 2-counter machine* M has 2 counters C and D and a sequence of n instructions. The n-th instruction is a halt instruction and the rest of them are one of the following:

- Increment or decrement of one of the counters and a jump to a specified instruction.
- A conditional jump upon one of the counters being zero. (We assume that the machine does not decrement when the counter value is 0).

The value of the location counter, which points to the instruction to be executed next and the values of the two counters C and D capture the *state* or *configuration* of M. We represent a configuration of M by a triple $< i, c, d >$, where $1 \leq i \leq n$, $c \geq 0$, and $d \geq 0$ give the values of the location counter and the two counters C and D.

A *computation* of M is a finite sequence of related configurations (the present configuration is the result of the execution of the instruction indicated by the location counter of the previous configuration), starting with the initial configuration $< 1, 0, 0 >$. A computation of M is said to be *halting* if the last instruction executed is a halt instruction.

We encode the computations of M using timed words over the alphabet $\Sigma_M = \{b_1, b_2, ..., b_n, a_1, a_2\}$. The encoding is such that each computation has at least one timed word which represents its encoding. We define a timed language L_M over Σ_M which contains all timed words corresponding to halting computations. We then give an MTL formula φ_M such that $L^c(\varphi_M) = L_M$. It then follows that with such an encoding M has a halting computation if and only if L_M is non-empty, which in turn holds if and only if φ_M is satisfiable.

The encoding referred to above is done in the following way. A configuration $< i, c, d >$ is represented by the sequence $b_i a_1^c a_2^d$. We encode a computation by concatenating the sequences representing the individual configurations. We use the time sequence associated with a timed word σ to ensure that the successive configurations are related as per the requirements of the program instructions. In the encoding, the subsequence of σ corresponding to the time interval $[j, j+1)$ encodes the j-th configuration of the computation and b_{i_j}, the location counter value of the j-th configuration, occurs at time j. Note that the underlying time domain allows the counter values to get arbitrarily large.

In case of a conditional jump instruction, the number of a_1's and a_2's in the next configuration do not differ from their corresponding values in the current configuration. To enforce such a requirement, we require that every

a_1 in the first interval has a matching a_1 at distance 1 and vice versa. We have a similar requirement for a_2's.

If the location counter of a configuration points to a decrement operation on counter C, then it requires that the number of a_1's in the succeeding configuration is one less than that of the current configuration. We encode the same by forcing every a_1 in the interval encoding the present configuration, except the last one, to have an a_1 at distance one from it in the succeeding interval and every a_1 in the succeeding interval to have an a_1 at distance one from it in the present interval. However, the number of a_2's remain the same and are encoded as in the case of the conditional jump instruction.

For an increment operation on counter C, we need an a_1 in the next interval at distance one for every a_1 in the present interval and an a_1 in the present interval at distance one from every a_1, except the last one, in the next interval. The corresponding conditions for operations on counter D can be obtained by exchanging the roles of a_1 and a_2.

Moreover, we restrict all a_1's in an interval $[j, j+1)$ encoding the j-th configuration to occur in the interval $[j+0.25, j+0.5)$, with the first one (if any) in that interval occurring at time $j+0.25$ and all a_2's to occur in the interval $[j+0.5, j+1)$, with the first one (if any) occurring at time $j+0.5$. The last restriction is a departure from the encoding used in [3] and [1]. It is however quite crucial to our argument in the next section.

Let M be a 2-counter machine with n instructions. We formally define the language L_M over the alphabet $\Sigma_M = \{b_1, ..., b_n, a_1, a_2\}$ as follows. $\sigma = (\alpha, \tau)$ is in L_M provided it satisfies the following:

1. $\alpha = \vdash b_{i_1} a_1^{c_1} a_2^{d_1} b_{i_2} a_1^{c_2} a_2^{d_2} ... b_{i_k} a_1^{c_k} a_2^{d_k}$ where $1 \leq i_j \leq n$, $c_j \geq 0$ and $d_j \geq 0$ for $1 \leq j \leq k$, and $i_1 = 1$, $c_1 = 0$ and $d_1 = 0$. This is to ensure that the timed word is a concatenation of valid encodings of configurations, starting with the encoding of the initial configuration $< 1, 0, 0 >$.

2. For all $1 \leq j \leq k$, the time of occurrence of b_{i_j} is j.

3. In every interval $[j, j+1)$ (encoding the j-th configuration), all a_1's occur in the interval $[j+0.25, j+0.5)$ and all a_2's occur in the interval $[j+0.5, j+1)$. The first a_1 in the interval (if any) occurs at time $j+0.25$ and the first a_2 in the interval (if any) occurs at time $j+0.5$.

4. There does not exist a time between the times of occurrences of two consecutive a_1's such that at distance one from that time there is a symbol

of the alphabet. Formally, let t_1 and t_2 be the times of occurrences of two consecutive a_1's. Then, there does not exist t, such that $t_1 < t < t_2$ and there is an event at time $(t+1)$. We have a similar condition for a_2.

5. For every a_1 at time t in the interval $[j, j+1)$, except the last one, there is an a_1 at time $t+1$. This is required only if there is a b_m at time $j+1$ for some $1 \leq m \leq n$. We have a similar condition for a_2.

6. Let the l-th configuration point to the i-th instruction.

   (a) If the i-th instruction is "If $C = 0$, then jump to p, else jump to q": Then to ensure that the number of a_1's in the interval $[l, l+1)$ and $[l+1, l+2)$ remain the same, we have:

   - If the value of counter C is 0: Then there is no a_1 in the interval $[l+1, l+2)$.
   - If the value of counter C is not 0: Then there is an a_1 at time $t+1$, where t is the time of occurrence of the last a_1 in the interval $[l, l+1)$ and there are no a_1's in the interval $(t+1, l+2)$.

   We obtain the conditions required to ensure that the number of a_2's in the interval $[l, l+1)$ and $[l+1, l+2)$ remain the same, by replacing C by D and a_1 by a_2 in the above clause.

   If the value of counter C is 0, then there is a b_p at time $l+1$, otherwise there is a b_q at time $l+1$.

   (b) If the i-th instruction is "Increment C and jump to p":

   - If the value of counter C is 0: There is exactly one a_1 in the interval $[l+1, l+2)$ at time $l+0.25$.
   - If the value of counter C is not 0: There is an a_1 at time $t+1$, where t is the time of occurrence of the last a_1 in the interval $[l, l+1)$ and there is exactly one a_1 in the interval $(t, l+2)$.

   We have a condition similar to that in the case of (a) to ensure that the number of a_2's in the interval $[l, l+1)$ and $[l+1, l+2)$ is the same. We have a b_p at time $t+1$.

   (c) If the i-th instruction is "Decrement C and jump to p":

   - If the value of counter C is 1: There is are no a_1's in the interval $[l+1, l+2)$.

- If the value of counter C is greater than 1: There are no a_1's in the interval $(t+1, l+2)$ where t is the time of occurrence of the last but one a_1 in the interval $[l, l+1)$.

We have a condition similar to that in the case of (a) to ensure that the number of a_2's in the interval $[l, l+1)$ and $[l+1, l+2)$ is the same. We have a b_p at time $t+1$.

(d) If the i-th instruction is a halt instruction, then there are no symbols in the interval $[l+1, \infty)$.

We get the corresponding conditions for operations on counter D by replacing C by D and exchanging a_1 and a_2 in the above clauses.

7. Every computation has a configuration which points to the halt instruction, i.e, there is a b_n somewhere in the timed word.

One can verify that the language L_M defined above is non-empty if and only if M has a halting computation.

We now proceed to give the formula φ_M over Σ_M which describes the language L_M. We will use B as a shorthand for $\bigvee_{i \in \{1,\ldots,n\}} b_i$ and φ_{action} as a shorthand for $\bigvee_{a \in \Sigma} a$. Whenever the interval in the operators $\Diamond^p$ and $\Box^p$ is not specified, it is taken to be $[0, \infty)$.

The formula φ_M is the conjunction of the following formulas, φ_1,...,φ_7. Each φ_i corresponds to the clause i in the description of the language L_M.

- φ_1:

$$O^p(b_1 \wedge \neg O^p a_1 \wedge \neg O^p a_2) \wedge \Box^p(a_2 \Rightarrow \neg O^p a_1).$$

- φ_2:

$$\Diamond^p_{[1,1]} B \wedge \neg \Diamond^p_{(0,1)} \varphi_{action} \wedge \Box^p((B \wedge \Diamond^p_{(0,\infty)} B) \Rightarrow (\Diamond^p_{[1,1]} B \wedge \Box^p_{(0,1)} \neg B)).$$

- φ_3:

$$\Box^p(B \Rightarrow (\Box^p_{[0,0.25)} \neg(a_1 \vee a_2) \wedge \Box^p_{[0.25,0.5)} \neg a_2 \wedge \Box^p_{[0.5,1)} \neg a_1 \wedge \\ (\Diamond^p_{(0,1)} a_1 \Rightarrow \Diamond^p_{[0.25,0.25]} a_1) \wedge (\Diamond^p_{(0,1)} a_2 \Rightarrow \Diamond^p_{[0.5,0.5]} a_2))).$$

- φ_4:

$$\neg \Diamond^p(a_1 \wedge O^p a_1 \wedge (\neg a_1 U^s_{(0,\infty)} (\neg a_1 \wedge \Diamond^p_{[1,1]} \varphi_{action}))) \wedge \\ \neg \Diamond^p(a_2 \wedge O^p a_2 \wedge (\neg a_2 U^s_{(0,\infty)} (\neg a_2 \wedge \Diamond^p_{[1,1]} \varphi_{action}))).$$

- φ_5:

$$\Box^p((B \wedge \Diamond^p_{[1,1]} B) \Rightarrow (\Box^p_{(0,1)}((a_1 \wedge O^p a_1) \Rightarrow (\Diamond^p_{[1,1]} a_1)) \wedge \\ \Box^p_{[0,1]}((a_2 \wedge O^p a_2) \Rightarrow (\Diamond^p_{[1,1]} a_2)))).$$

- φ_6: This describes the properties specific to instructions that the timed word should satisfy. It is a conjunction of various formulas as defined below:

$$\varphi_6 = \bigwedge_{i \in \{1,..,n\}} \varphi_6^i.$$

ψ_{a_1} along with φ_5 verifies that the number of a_1's in two consecutive encodings of configurations is the same. ψ_{a_2} along with φ_5 verifies the number of a_2's in two consecutive encodings of configurations is the same.

$$\psi_{a_1} = \Box^p_{[0,1]}((a_1 \wedge \neg O^p a_1) \Rightarrow (\Diamond^p_{[1,1]}(a_1 \wedge \neg O^p a_1))) \wedge (\Box^p_{[0,1]} \neg a_1 \Rightarrow \Box^p_{[1,2]} \neg a_1).$$

$$\psi_{a_2} = \Box^p_{[0,1]}((a_2 \wedge \neg O^p a_2) \Rightarrow (\Diamond^p_{[1,1]}(a_2 \wedge \neg O^p a_2))) \wedge (\Box^p_{[0,1]} \neg a_2 \Rightarrow \Box^p_{[1,2]} \neg a_2).$$

We define φ_6^i depending on the i-th instruction which is one of the following:

  - If $C = 0$, then jump to p, otherwise jump to q.

$$\Box^p(b_i \Rightarrow (\psi_{a_1} \wedge \psi_{a_2} \wedge (\Box^p_{(0,1)} \neg a_1 \Rightarrow \Diamond^p_{[1,1]} b_p) \wedge (\Diamond^p_{[0,1]} a_1 \Rightarrow \Diamond^p_{[1,1]} b_q))).$$

  - Increment counter C and jump to p.

$$\Box^p(b_i \Rightarrow (\psi_{a_2} \wedge \Box^p_{(0,1)}((a_1 \wedge \neg O^p a_1) \Rightarrow \Diamond^p_{[1,1]}(a_1 \wedge O^p a_1 \wedge \neg O^p O^p a_1)) \\ \wedge (\Box^p_{(0,1)} \neg a_1 \Rightarrow \Diamond^p_{[1.25,1.25]} a_1)) \wedge \Diamond^p_{[1,1]} b_p)).$$

  - Decrement counter C and jump to p.

$$\Box^p(b_i \Rightarrow (\psi_{a_2} \wedge \Box^p_{(0,1)}(((a_1 \wedge O^p a_1 \wedge \neg O^p O^p a_1) \Rightarrow (\Diamond^p_{[1,1]} \neg O^p a_1)) \wedge \\ ((a_1 \wedge \neg O^p a_1) \Rightarrow \neg \Diamond^p_{[1,1]} a_1)) \wedge \Diamond^p_{[1,1]} b_p)).$$

  We can write similar formulas for counter D by exchanging a_1 and a_2.

– Halt instruction.

$$\Box^p(b_i \Rightarrow \Box^p_{[1,\infty)}(\neg\varphi_{action})).$$

- $\varphi_7 : \Diamond^p b_n$.

It is not difficult to verify that each formula φ_i asserts precisely the clause i in the definition of the language L_M. Hence $L^c(\varphi_M)$ coincides with L_M. This completes the undecidability proof. $\Box$

5 Strict containment of MTL^{pw} in MTL^c

We have already seen that MTL^{pw} is contained in MTL^c, i.e, any property expressible in MTL^{pw} is also expressible in MTL^c. In this section we argue that MTL^c is strictly more expressive that MTL^{pw}:

Theorem 5.1 *The logic MTL^c is strictly more expressive than MTL^{pw}.*

The rest of this section is devoted to a proof of this theorem. We basically elicit a language which is closely related to a property used in the previous undecidability result, and show that it is definable by MTL^c but not by MTL^{pw}.

We claim that there does not exist an MTL^{pw} formula φ which defines the language L_{ni} over $\Sigma = \{a, b\}$ defined below: L_{ni} consists of timed words over {a,b} which do not contain two consecutive a's such that there exists a time between their times of occurrences, at distance one from which there is an a or b. Formally, let $\sigma = (a_0, t_0)...(a_n, t_n)$ be a timed word over $\Sigma = \{a, b\}$. σ is in L_{ni} if and only if for all i such that $a_i = a_{i+1} = a$ and for all t such that $t_i < t < t_{i+1}$, there does not exist j such that $t_j = t + 1$.

In general, let $L^{\Sigma,c}_{ni}$, where $c \in \Sigma$, denote the language consisting of timed words over Σ which do not contain two consecutive c's such that there exists a time between their times of occurrences, at distance one from which there is a symbol of Σ. Formally, let $\sigma = (a_0, t_0)...(a_n, t_n)$ be a timed word over Σ. σ is in $L^{\Sigma,c}_{ni}$ if and only if for all i such that $a_i = a_{i+1} = c$ and for all t such that $t_i < t < t_{i+1}$, there does not exist j such that $t_j = t + 1$.

We first show that if L_{ni} is definable by an MTL^{pw} formula φ, then for all Σ, $L^{\Sigma,c}_{ni}$ is definable by some formula φ' (obtainable from φ).

We define the notion of a *substitution* on formulas and languages. Let Σ_1, Σ_2 be two alphabets. A substitution ρ is a mapping $\rho : \Sigma_1 \to 2^{\Sigma_2}$ satisfying:

1. $\rho(a) \neq \phi$ for all $a \in \Sigma_1$.

2. $\rho(a) \cap \rho(b) = \phi$ if $a \neq b$.

3. $\forall a' \in \Sigma_2$, there exists an $a \in \Sigma_1$ such that $a' \in \rho(a)$.

The substitution ρ works on timed languages as follows. Let L be a timed language over Σ_1. We define $\rho(L)$ to be: $\rho(L) = \{\sigma' = (\alpha', \tau') \in T\Sigma_2^* \mid \exists \sigma = (\alpha, \tau') \in L, \alpha'(i) \in \rho(\alpha(i)), \forall 0 \leq i < |\sigma|\}$.

On formulas, the substitution ρ works as follows. Let φ be a formula over Σ_1. By $\rho(\varphi)$ we mean the formula obtained by replacing each a in φ by $\bigvee_{d \in \rho(a)} d$.

The lemma below says that the effect of a substitution on a formula is "equivalent" to carrying out the substitution on the language defined by the formula.

Lemma 5.2 *Let Σ_1, Σ_2 be two alphabets and $\rho : \Sigma_1 \to 2^{\Sigma_2}$ be a substitution. Let φ be a formula over Σ_1. Then $L(\rho(\varphi)) = \rho(L(\varphi))$.*

Proof Let $\sigma' = (\alpha', \tau') \in T\Sigma_2^*$. Then $\sigma = (\alpha, \tau')$ such that $\alpha'(i) \in \rho(\alpha(i)), \forall 0 \leq i < |\sigma|$. We claim that $\sigma, i \models \varphi$ iff $\sigma', i \models \rho(\varphi)$. Proof is by induction on the structure of φ.

- $\varphi = a$
  $\sigma, i \models \varphi$ iff $\sigma, i \models a$ iff $\alpha(i) = a$ iff $\alpha'(i) \in \rho(\alpha(i))$ iff $\sigma', i \models \bigvee_{d \in \rho(\alpha(i))} d$ iff $\sigma', i \models \rho(\varphi)$

- $\varphi = \neg\varphi_1$
  $\sigma, i \models \varphi$ iff $\sigma, i \models \neg\varphi_1$ iff $\sigma, i \not\models \varphi_1$ iff $\sigma', i \not\models \rho(\varphi_1)$ iff $\sigma', i \models \neg\rho(\varphi_1)$ iff $\sigma', i \models \rho(\varphi)$

- $\varphi = \varphi_1 \vee \varphi_2$
  $\sigma, i \models \varphi$ iff $\sigma, i \models \varphi_1 \vee \varphi_2$ iff $\sigma, i \models \varphi_1$ or $\sigma, i \models \varphi_2$ iff $\sigma', i \models \rho(\varphi_1)$ or $\sigma', i \models \rho(\varphi_2)$ iff $\sigma', i \models \rho(\varphi_1) \vee \rho(\varphi_2)$ iff $\sigma', i \models \rho(\varphi)$

- $\varphi = \varphi_1 U_I^s \varphi_2$
  $\sigma, i \models \varphi$ iff $\sigma, i \models \varphi_1 U_I^s \varphi_2$ iff $\exists j \geq i$, such that $\tau(j) - \tau(i) \in I, \sigma, j \models \varphi_2$ and for all $i < k < j, \sigma, k \models \varphi_1$ iff $\exists j \geq i$, such that $\tau(j) - \tau(i) \in I, \sigma', j \models \rho(\varphi_2)$ and for all $i < k < j, \sigma', k \models \rho(\varphi_1)$ iff $\sigma', i \models \rho(\varphi_1) U_I^s \rho(\varphi_2)$ iff $\sigma', i \models \rho(\varphi)$

It follows from the claim that $L(\rho(\varphi)) = \rho(L(\varphi))$: $\sigma' \in L(\rho(\varphi))$ iff $\sigma' \models \rho(\varphi)$ iff $\sigma \models \varphi$ iff $\sigma \in L(\varphi)$ iff $\sigma' \in \rho(L(\varphi))$. $\square$

$\square$

It is easy to see that $\rho(L_{ni}) = L_{ni}^{\Sigma,c}$ where $\rho(a) = c$ and $\rho(b) = \Sigma - c$. Let φ be a formula such that $L_{ni} = L(\varphi)$. Then, it follows from the lemma that $L_{ni}^{\Sigma,c} = L(\rho(\varphi))$.

Before we proceed, we note that the formulas φ_i (all except for φ_4) of the previous section were all essentially pointwise formulas. Let φ'_i denote the corresponding formulas in the pointwise syntax.

Now, suppose there exists a formula which in the pointwise semantics defines L_{ni}. Let $\varphi_{4,1}, \varphi_{4,2}, ...$ be an enumeration of the MTL^{pw} formulas over the alphabet $\{a, b\}$. One of these defines the language L_{ni}. Consider a sequence of turing machines $T_0, T_1, ...$ where T_i is a turing machine which takes as input a 2-counter machine M and outputs

$$\varphi'_M = \varphi'_1 \wedge \varphi'_2 \wedge \varphi'_3 \wedge (\rho^M_{a_1}(\varphi_{4,i}) \wedge \rho^M_{a_2}(\varphi_{4,i})) \wedge \varphi'_5 \wedge \varphi'_6 \wedge \varphi'_7$$

where the substitution $\rho^M_c : \{a, b\} \to \Sigma_M$ (for $c \in \Sigma_M$) is defined as: $\rho^M_c(a) = \{c\}$ and $\rho^M_c(b) = \Sigma_M - \{c\}$. One of these turing machines takes as input M and outputs a formula φ'_M which in the pointwise interpretation defines the language L_M. Hence, there exists a turing machine which computes the reduction of the halting problem of 2-counter machine to the satisfiability problem of MTL in pointwise semantics. This implies that the satisfiability problem of MTL^{pw} is undecidable. But this contradicts the decidability of MTL^{pw} proved in [10]. Hence, there does not exist an MTL^{pw} formula which defines L_{ni}.

However, there exists an MTL^c formula which defines L_{ni}. The formula over $\{a, b\}$ which in the continuous semantics defines L_{ni} is given by:

$$\neg\Diamond^p(a \wedge O^p a \wedge \neg a U^s_{(0,\infty)}(\neg a \wedge \Diamond^p_{[1,1]}(a \vee b))).$$

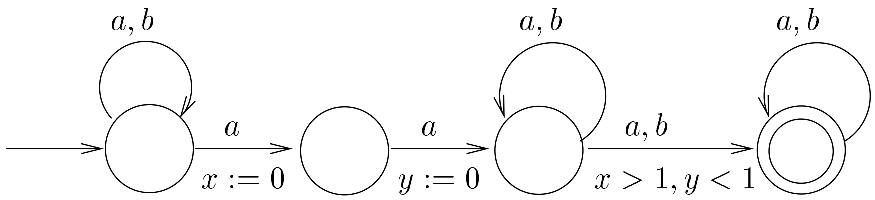
Hence we conclude that MTL^c is more expressive than MTL^{pw}.

This completes the proof of Theorem 5.1. $\square$

6 An inexpressiveness result for alternating timed automata

In the previous section we proved that MTL^c is strictly more expressive than MTL^{pw} by eliciting the language L_{ni} and showing that the expressibility of this language by MTL^{pw} would render it undecidable. In general, this technique seems applicable to obtain similar results in the setting of timed languages. In this section we will exemplify the same by using it to show that one clock alternating timed automata (OCATA) and timed automata (TA) express orthogonal classes of languages.

In one direction, we have the language L_a – consisting of timed words over the singleton alphabet $\{a\}$ in which no two a's are separated by exactly one time unit – which is accepted by no timed automaton [6] but is accepted by a OCATA [10]. In the other direction, we show that the language L_i which is the complement of L_{ni} is not accepted by any OCATA but is accepted by the timed automaton below.



We will show in what follows that the language L_{ni} is not acceptable by any OCATA as that would essentially lead to the undecidability of the emptiness problem for OCATA which in turn would contradict the results of its decidability [8]. The proof basically follows the one used to disprove the inclusion of L_{ni} in MTL^{pw}.

Suppose that there exists a OCATA $\mathrm{A}_{ni} = (\{a,b\}, S, S_0, F, \delta)$ which expresses the language L_{ni}. Let $\mathrm{A}_{ni}^{\Sigma,c} = (\Sigma, S, S_0, F, \delta')$, where for any state s and any symbol d of the alphabet,

$$\delta'(s,d) = \begin{cases} \delta(s,a) & \text{if} \quad d = c. \\ \delta(s,b) & \text{if} \quad d \neq c. \end{cases}$$

Informally, $\mathrm{A}_{ni}^{\Sigma,c}$ is the automaton obtained by replacing an edge labelled with a in A_{ni} by an edge labelled with c and an edge labelled with b by an edge labelled with d for each $d \in \Sigma - \{c\}$. It is easy to check that $\mathrm{A}_{ni}^{\Sigma,c}$ accepts exactly the timed words in the language $L_{ni}^{\Sigma,c}$.

In [10], Ouaknine and Worrell have shown that all MTL pointwise formulas can be expressed using OCATA. In fact they give a way of constructing the automaton A_φ for the MTL formula φ such that $L^{pw}(\varphi) = L(\mathrm{A}_\varphi)$. Hence given a 2-counter machine M, we have a procedure for constructing a one clock timed alternating automaton which accepts the language L_M or $L^{pw}(\varphi_M)$ as explained below.

Let M be a 2-counter machine. Let A_1,...,A_3, A_5,...,A_7 represent the automata corresponding to the pointwise versions of the formulas φ_1,...,φ_3,φ_5,...,φ_7.

Let A^1_{ni}, A^2_{ni},... be an enumeration of the OCATA. We can give an argument similar to the previous one to show that there exists a turing machine which takes as input a 2-counter machine and outputs A_M, the automaton corresponding to the intersection of A_1,...,A_3, A_5,...,A_7, A^{Σ,a_1}_{ni} and A^{Σ,a_2}_{ni}. Since the languages accepted by these automata coincide with the clauses in the definition of L_M, one can easily convince oneself that the automaton A_M accepts the language L_M.

In essence, we have reduced the halting problem for 2-counter machine to the emptiness problem of OCATA. This in turn implies that the emptiness problem for OCATA is undecidable which contradicts the decidability results in [8]. Hence the language L_{ni} is not accepted by any OCATA. Since these automata are closed under complementation, the language L_i is also not accepted by any OCATA. This completes the proof of the incomparability of the class of languages accepted by timed automata and one clock alternating timed automata.

7 Conclusion

In this paper we have addressed the problem of comparing the expressiveness of MTL in the pointwise and continuous semantics. We give a natural way of interpreting the continuous version of the logic over timed words, so that both versions are interpreted over the same models. We then show that the continuous semantics is strictly more expressive than the pointwise one. This result holds for the case when models are *finite* timed words.

It would be interesting to have an expressiveness result that arises out of a characterisation of the languages definable in the two semantics. The technique used in this paper is not applicable to differentiating the two semantics when we look at infinite words, since the decidability of the pointwise logic over infinite words is not yet known. Similarly, we cannot say anything about MITL in the pointwise versus the continuous semantics (for neither finite nor infinite words). Our argument also does not apply to reasoning about MTL with past operators, since both the pointwise and continuous versions of the logic are undecidable. These would be interesting questions to answer.

References

[1] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[2] Rajeev Alur, Tomás Feder, and Thomas A. Henzinger. The Benefits of Relaxing Punctuality. *Journal of the ACM*, 43(1):116–146, 1996.

[3] Rajeev Alur and Thomas A. Henzinger. Real-time Logics: Complexity and Expressiveness. In *LICS*, pages 390–401. IEEE Computer Society, 1990.

[4] Deepak D'Souza and Nicolas Tabareau. On timed automata with input-determined guards. In Yassine Lakhnech and Sergio Yovine, editors, *FORMATS/FTRTFT*, volume 3253 of *Lecture Notes in Computer Science*, pages 68–83. Springer, 2004.

[5] Thomas A. Henzinger, Jean-François Raskin, and Pierre-Yves Schobbens. The Regular Real-Time Languages. In Kim Guldstrand Larsen, Sven Skyum, and Glynn Winskel, editors, *ICALP*, volume 1443 of *Lecture Notes in Computer Science*, pages 580–591. Springer, 1998.

[6] Philippe Herrmann. Timed Automata and Recognizability. *Inf. Process. Lett.*, 65(6):313–318, 1998.

[7] Ron Koymans. Specifying Real-Time Properties with Metric Temporal Logic. *Real-Time Systems*, 2(4):255–299, 1990.

[8] Slawomir Lasota and Igor Walukiewicz. Alternating Timed Automata. In Vladimiro Sassone, editor, *FoSSaCS*, volume 3441 of *Lecture Notes in Computer Science*, pages 250–265. Springer, 2005.

[9] Étienne Lozes. Adjuncts elimination in the static ambient logic. *Electr. Notes Theor. Comput. Sci.*, 96:51–72, 2004.

[10] Joel Ouaknine and James Worrel. On the Decidability of Metric Temporal Logic. *Proc. LICS 2005*, 2005.