Making Metric Temporal Logic Rational

S. Krishna\(^1\), Khushraj Madnani\(^1\), and P. K. Pandya\(^2\)

\(^1\) krishnas,khushraj@cse.iitb.ac.in
\(^2\) pandya@tifr.res.in

Abstract
We study an extension of MTL in pointwise time with rational expression guarded modality \(\text{Rat}_I(re)\) where \(re\) is a rational expression over subformulae. We study the decidability and expressiveness of this extension \((\text{MTL} + \varphi \text{URat}_I re \varphi + \text{Rat}_I re \varphi)\), called RatMTL, as well as its fragment SfrMTL where only star-free rational expressions are allowed. Using the technique of temporal projections, we show that RatMTL has decidable satisfiability by giving an equisatisfiable reduction to MTL. We also identify a subclass MITL + URat of RatMTL for which our equi-satisfiable reduction gives rise to formulae of MITL, yielding elementary decidability. As our second main result, we show a tight automaton-logic connection between SfrMTL and partially ordered (or very weak) 1-clock alternating timed automata.

Digital Object Identifier 10.4230/LIPIcs...

1 Introduction

Temporal logics provide constructs to specify qualitative ordering between events in time. Real time logics are quantitative extensions of temporal logics with the ability to specify real time constraints amongst events. Logics MTL and TPTL are amongst the prominent real time logics [2]. Two notions of MTL semantics have been studied in the literature: continuous and pointwise [3]. The expressiveness and decidability results vary considerably with the semantics used: while the satisfiability checking of MTL is undecidable in the continuous semantics even for finite timed words [1], it is decidable in pointwise semantics with non-primitive recursive complexity [15]. Due to limited expressive power of MTL, several additional modalities have been proposed: the threshold counting modality \(C_n\phi\) states that in time interval \(I\) relative to current point, \(\phi\) occurs at least \(n\) times. The Pnueli modality \(P_n(\phi_1, \ldots, \phi_n)\) states that there is a subsequence of \(n\) time points inside interval \(I\) where at \(i\)th point the formula \(\phi_i\) holds. In a recent result, Hunter [9] showed that, in continuous time semantics, MTL enriched with \(C\) modality (denoted MTL + \(C\)) is as expressive as FO with distance FO[\(<, +1\)], which is as expressive as TPTL. Unfortunately, satisfiability and model checking of all these logics are undecidable. This has led us to focus on the pointwise case with only the future modality, i.e. logic MTL[\(U_I\)], which we abbreviate as MTL in rest of the paper. Also, MTL + \(op\) means MTL with modalities \(U_I\) as well as \(op\).

In pointwise semantics, it can be shown that MTL \(\subset\) MTL + \(C\) \(\subset\) MTL + \(P_n\) (see [11]). In this paper, we propose a generalization of threshold counting and Pnueli modalities by a rational expression modality \(\text{Rat}_{Ire}(\phi_1, \ldots, \phi_k)\), which specifies that the truth of the subformulae, \(\phi_1, \ldots, \phi_k\), at the set of points within interval \(I\) is in accordance with the rational expression \(re(\phi_1, \ldots, \phi_k)\). The resulting logic is called RatMTL and is the subject of this paper. The expressive power of logic RatMTL raises several points of interest. It can be shown that MTL + \(P_n\) \(\subset\) RatMTL, and it can express several new and interesting properties: (1) Formula \(\text{Rat}_{I(1,2)}((aa)^*)\) states that within time interval \((1, 2)\) there is an even number of occurrences of \(a\). We will define a derived modulo counting modality which states this directly as the formula MC_{\(0\leq\)2}. (2) An exercise regime lasting between 60 to 70 seconds
consists of arbitrary many repetitions of three pushup cycles which must be completed within 2 seconds. There is no restriction on delay between two cycles to accommodate weak athletes. This is given by $\text{Rat}_{[0,70]}(\text{UPP,up,up})$ where $\text{UPP} = \text{(up UC,0,2,up)}$. The inability to specify rational expression constraints has been an important lacuna of LTL and its practically useful extensions such as PSL sugar [7], [6] (based on Dynamic Logic [5]) which extend LTL with both counting and rational expressions. This indicates that our logic $\text{RatMTL}$ is a natural and useful logic for specifying properties. However, to our knowledge, impact of rational expression constraints on metric temporal modalities have not been studied before. As we show in the paper, timing and regularity constraints interact in a fairly complex manner.

As our first main result, we show that satisfiability of $\text{RatMTL}$ is decidable by giving an equisatisfiable reduction to MTL. The reduction makes use of the technique of over-sampled temporal projections which was previously proposed [10], [11] and used for proving the decidability of MTL+$C$. The reduction given here has several novel features such as an MTL encoding of the run tree of an alternating automaton which restarts the DFA of a given rational expression at each time point (section 3.1). We identify two syntactic subsets of $\text{RatMTL}$ denoted MTL+$URat$ with 2EXPSPACE hard satisfiability, and its further subset MTL+$UM$ with EXPSPACE-complete satisfiability. As our second main result, we show that the star-free fragment $\text{SfrMTL}$ of $\text{RatMTL}$ characterizes exactly the class of partially ordered 1-clock alternating timed automata, thereby giving a tight logic automaton connection. The most non-trivial part of this proof is the construction of $\text{SfrMTL}$ formula equivalent to a given partially ordered 1-clock alternating timed automaton $\mathcal{A}$ (Lemma 7).

## 2 Timed Temporal Logics

This section describes the syntax and semantics of the timed temporal logics needed in this paper: MTL and TPTL. Let $\Sigma$ be a finite set of propositions. A finite word over $\Sigma$ is a tuple $\rho = (\sigma, \tau)$. $\sigma$ and $\tau$ are sequences $\sigma_1 \sigma_2 \ldots \sigma_n$ and $\tau_1 \tau_2 \ldots \tau_n$ respectively, with $\sigma_i \in \mathcal{P}(\Sigma) - \emptyset$, and $\tau_i \in \mathbb{R}_{\geq 0}$ for $1 \leq i \leq n$ and $\forall i \in \text{dom}(\rho)$, $\tau_i \leq \tau_{i+1}$, where $\text{dom}(\rho)$ is the set of positions $\{1, 2, \ldots, n\}$ in the timed word. For convenience, we assume $\tau_n = 0$. The $\sigma_i$'s can be thought of as labeling positions $i$ in $\text{dom}(\rho)$. For example, given $\Sigma = \{a, b, c\}$, $\rho = \{(a, c), 0\}(\{a\}, 0.7)(\{b\}, 1.1)$ is a timed word. $\rho$ is strictly monotonic iff $\tau_i < \tau_{i+1}$ for all $i$, $i + 1 \in \text{dom}(\rho)$. Otherwise, it is weakly monotonic. The set of finite timed words over $\Sigma$ is denoted $T\Sigma^*$. Given $\rho = (\sigma, \tau)$ with $\sigma = \sigma_1 \ldots \sigma_n$, $\sigma_{\text{single}}$ denotes the set of words $\{w_1 w_2 \ldots w_n \mid w_i \in \sigma_i\}$. For $\rho$ as above, $\sigma_{\text{single}}$ consists of $\{(a), 0\}(\{a\}, 0.7)(\{b\}, 1.1)$ and $\{(c), 0\}(\{a\}, 0.7)(\{b\}, 1.1)$. Let $I_{\nu}$ be a set of open, half-open or closed time intervals. The end points of these intervals are in $\mathbb{N} \cup \{0, \infty\}$. For example, $[1,3), [2, \infty)$. For $\tau \in \mathbb{R}_{\geq 0}$ and interval $[a, b)$, with $< \in \{, [\} \text{ and } > \in \{, ]\} \text{, } \tau + (a, b) \text{ stands for the interval } (\tau + a, \tau + b)$.

**Metric Temporal Logic (MTL).** Given a finite alphabet $\Sigma$, the formula $\varphi$ of MTL are built from $\Sigma$ using boolean connectives and time constrained version of the modality $U$ as follows: $\varphi ::= a \in (\Sigma) | \text{true} | \varphi \land \varphi | \neg \varphi | \varphi U_I \varphi$, where $I \in I_{\nu}$. For a timed word $\rho = (\sigma, \tau) \in T\Sigma^*$, a position $i \in \text{dom}(\rho)$, and an MTL formula $\varphi$, the satisfaction of $\varphi$ at a position $i$ of $\rho$ is denoted $(\rho, i) \models \varphi$, and is defined as follows: (i) $\rho, i \models a \iff a \in \sigma_i$, (ii) $\rho, i \models \neg \varphi \iff \rho, i \not\models \varphi$, (iii) $\rho, i \models \varphi_1 \land \varphi_2 \iff \rho, i \models \varphi_1 \text{ and } \rho, i \models \varphi_2$, (iv) $\rho, i \models \varphi_1 U_I \varphi_2 \iff \exists j > i$, $\rho, j \models \varphi_2, \tau_j = \tau_i \in I$, and $\rho, k \models \varphi_1 \forall i < k < j$.

The language of a MTL formula $\varphi$ is $L(\varphi) = \{\rho \mid \rho, 1 \models \varphi\}$. Two formulae $\varphi$ and $\phi$ are said to be equivalent denoted as $\varphi \equiv \phi$ iff $L(\varphi) = L(\phi)$. Additional temporal connectives are defined in the standard way: we have the constrained future eventuality operator $\square_I a \equiv
true $U_I a$ and its dual $□_I a \equiv \neg \diamond_I \neg a$. We also define the next operator as $O_I \phi \equiv \perp U_I \phi$.

Non strict versions of operators are defined as $\bigcirc_a \phi = a \lor \bigcirc_I \phi$, $\bigdiamond_a \phi \equiv a \land \bigdiamond_I a$, $a U_I b \equiv b \lor [a \land (a U_I b)]$ if $0 \notin I$, and $\bigvee a \land (a U_I b)]$ if $0 \notin I$. Also, $a W b$ is a shorthand for $\bigcirc a \lor (a U b)$.

The subclass of $MTL$ obtained by restricting the intervals $I$ in the until modality to non-punctual intervals is denoted $MTL$.

**Timed Propositional Temporal Logic (TPTL).** TPTL is a prominent real time extension of LTL, where timing constraints are specified with the help of freeze clocks. The set of TPTL formulas are defined inductively as $\varphi := a(\in \Sigma) \mid true \mid \varphi \land \varphi \mid \neg \varphi \mid \varphi \lor \varphi \mid y, \varphi \mid y \in I$. $C$ is a set of clock variables progressing at the same rate, $y \in C$, and $I$ is an interval as above.

For a timed word $\rho = (\sigma_1, \tau_1) \ldots (\sigma_n, \tau_n)$, we define the satisfiability relation, $\rho, i, \nu \models \phi$ saying that the formula $\phi$ is true at position $i$ of the timed word $\rho$ with valuation $\nu$ of all the clock variables as follows: (1) $\rho, i, \nu \models a \leftrightarrow a \in \sigma_i$, (2) $\rho, i, \nu \models \neg \varphi \leftrightarrow \rho, i, \nu \not\models \varphi$, (3) $\rho, i, \nu \models \varphi_1 \land \varphi_2 \leftrightarrow \rho, i, \nu \models \varphi_1$ and $\rho, i, \nu \models \varphi_2$, (4) $\rho, i, \nu \models \varphi \leftrightarrow \rho, i, \nu[\tau_i \leftarrow \tau_i]$ $\models \varphi$, (5) $\rho, i, \nu \models \varphi \in I \leftrightarrow \tau_i - \nu(x) \in I$, (6) $\rho, i, \nu \models \varphi \lor \varphi \leftrightarrow \\exists j > i, \rho, j, \nu \models \varphi_2$, and $\rho, k, \nu \models \varphi_1 \land i < k < j$. $\rho$ satisfies $\phi$ denoted $\rho \models \phi$ iff $\rho, 1, 0 \models \phi$. Here $0$ is the valuation obtained by setting all clock variables to $0$. We denote by $k$−TPTL the fragment of TPTL using at most $k$ clock variables.

**Theorem 1 ([15]).** $MTL$ satisfiability is decidable over finite timed words and is non-primitive recursive.

**MTL with Rational Expressions (RatMTL)**

We propose an extension of $MTL$ with rational expressions, that forms the core of the paper. These modalities can assert the truth of a rational expression (over subformulae) within a particular time interval with respect to the present point. For example, $\text{Rat}_{(0,1)}(\varphi_1, \varphi_2) \uplus$ when evaluated at a point $i$, asserts the existence of $2k$ points $\tau_i < \tau_{i+1} < \tau_{i+2} < \cdots < \tau_{i+2k} < \tau_{i+1}$, $k > 0$, such that $\varphi_1$ evaluates to true at $\tau_{i+2j+1}$, and $\varphi_2$ evaluates to true at $\tau_{i+2j+2}$, for all $0 \leq j < k$.

**RatMTL Syntax:** Formulae of RatMTL are built from $\Sigma$ (atomic propositions) as follows: $\varphi := a(\in \Sigma) \mid true \mid \varphi \land \varphi \mid \neg \varphi \mid \varphi \lor \varphi \mid y, \varphi \mid y \in I \mid \text{Rat} \text{re}(S) \mid \text{URat} \text{re}(S) \varphi$, where $I \in \text{In}$ and $S$ is a finite set of formulæ of interest, and $\text{re}(S)$ is defined as a rational expression over $S$. $\text{re}(S) := \varphi(\in S) \mid \text{re}(S) \cdot \text{re}(S) \mid \text{re}(S) + \text{re}(S) \mid [\text{re}(S)]^*$. Thus, RatMTL is $MTL$ + $URat$ + $Rat$. An atomic rational expression $\text{re}$ is any well-formed formula $\varphi \in \text{RatMTL}$.

**RatMTL Semantics:** For a timed word $\rho = (\sigma, \tau) \in \Sigma^*$, a position $i \in \text{dom}(\rho)$, and a RatMTL formula $\varphi$, a finite set $S$ of formulæ, we define the satisfaction of $\varphi$ at a position $i$ as follows. For positions $i < j \in \text{dom}(\rho)$, let $\text{Seg}(S, i, j)$ denote the untimed word over $P(S)$ obtained by marking the positions $k$ in $\{i + 1, \ldots, j - 1\}$ of $\rho$ with $\psi \in S$ iff $\rho, k \models \psi$. For a position $i \in \text{dom}(\rho)$ and an interval $I$, let $\text{Seg}(S, I, i)$ denotes the untimed word over $P(S)$ obtained by marking all the positions $k$ such that $\tau_k - \tau_i \in I$ of $\rho$ with $\psi \in S$ iff $\rho, k \models \psi$.

1. $\rho, i \models \text{Rat} \text{re} \leftrightarrow [\text{Seg}(S, I, i)]^\text{single} \land \text{L}(\text{re}(S)) \neq \emptyset$, where $\text{L}(\text{re}(S))$ is the language of the rational expression $\text{re}$ formed over the set $S$. The subclass of RatMTL using only the URat modality is denoted $\text{RatMTL}[\text{URat}]$ or $\text{MTL} + \text{URat}$ and if only non-punctual intervals are used, then it is denoted $\text{RatMTL}[\text{URat}]$ or $\text{MTL} + \text{URat}$.

2. $\rho, i \models \text{Rat} \text{re} \leftrightarrow [\text{Seg}(S, I, i)]^\text{single} \land \text{L}(\text{re}(S)) \neq \emptyset$.

The language accepted by a RatMTL formula $\varphi$ is given by $L(\varphi) = \{\rho \mid \rho, 0 \models \varphi\}$.

**Example 1.** Consider the formula $\varphi = a\text{URat}_{(0,1)}ab$. Then $\text{re} = ab^*$, and the subformulae of interest are $a, b$. For $\varphi = \{(a) \cup (a, b), 0.3\}((a, b), 0.99)$, $\rho, 1 \models \varphi$, since $a \in \sigma_2, b \in \sigma_3$. 

- Krishna, Madnani, Pandya. XX:3
\[ \tau_3 - \tau_1 \in (0, 1) \text{ and } a \in \text{Seg}([a, b], 1, 3)^{\text{single}} \cap L(ab^*). \] On the other hand, for the word \[ \rho = ([a], 0) ([a], 0.3) ([a], 0.5) ([a], 0.9) ([b], 0.99), \] we know that \( \rho, 1 \neq \varphi \), since even though \( b \in \sigma_5, a \in \tau_i \) for \( i < 5 \), \([\text{Seg}([a, b], 1, 5)]^{\text{single}} = \text{aaa} \) and \( \text{aaa} \notin L(ab^*). \)

Example 2. Consider the formula \( \varphi = \text{Rat}_{(0, 1]} \neg \text{Rat}_{(0, 1]} a \). For the word \( \rho = ([a, b], 0) ([a, b], 0.91) ([a], 1, 2) \), to check \( \varphi \) at position 1, we check position 2 of the word, since \( \tau_2 - \tau_1 \in (0, 1) \). The formulae of interest for marking is \( \{ \neg \text{Rat}_{(0, 1]} a \} \). Position 2 is not marked, since \( \rho, 2 \models \text{Rat}_{(0, 1]} a \). Then \([\text{TSeg}([0, 1, 1)]^{\text{single}} = \emptyset \notin L(\neg \text{Rat}_{(0, 1]} a) \). However, for the word \( \rho = ([a, b], 0) ([a, b], 0.91) ([b], 1, 1) \), \( \rho, 1 \models \varphi \), since position 2 is marked with \( \neg \text{Rat}_{(0, 1]} a \), and \( \neg \text{Rat}_{(0, 1]} a \in L(\neg \text{Rat}_{(0, 1]} a) \cap [\text{TSeg}([0, 1, 1)]^{\text{single}}\). Example 3. Consider the formula \( \varphi = \text{Rat}_{(0, 1]} \neg \text{Rat}_{(0, 1]} a \). For \( \rho = ([a, b], 0) ([a, b], 0.7) ([b], 0.98) ([a, b], 1, 4) \), we have \( \rho, 1 \models \text{Rat}_{(0, 1]} \neg \text{Rat}_{(0, 1]} a \), even though point 3 is.

Generalizing Counting, Pnueli & Mod Counting Modalities The following reductions show that RatMTL subsumes most of the extensions of MTL studied in the literature. (1) \( \text{Threshold Counting} \) constraints [17, 12, 11] specify the number of times a property holds within some time region is at least (or at most) \( n \). These can be expressed in RatMTL: (i) \( \text{Seg}_{(t, \sigma)} \varphi \equiv \text{Rat}_t \rho_{\text{th}} \), (ii) \( \text{UT}_\sigma \varphi \geq n \phi_2 \equiv \text{Rat}_t \rho_{\text{th}} \phi_2 \), where \( \rho_{\text{th}} = \text{true}^* \varphi \text{true}^* \ldots \varphi \text{true}^* \). (n times)

(2) \( \text{Pnueli Modalities} \) [17], which enhance the expressiveness of MITL in continuous semantics preserving the complexity, can be written in RatMTL: \( \text{Pn}(\phi_1, \phi_2, \ldots, \phi_k) \) can be written as \( \text{Rat}_t(\text{true}^* \phi_1 \text{true}^* \phi_2 \ldots \text{true}^* \phi_k \text{true}^*). \)

(3) \( \text{Modulo Counting} \) constraints [3, 13] specify the number of times a property holds modulo \( n \in \mathbb{N} \), in some region. We extend these to the timed setting by proposing two modalities \( \text{MC}^{\%a}_t \varphi \) and \( \text{UM}_{t, \varphi = k \% n} \varphi \). \( \text{MC}^{\%a}_t \varphi \) checks if the number of times \( \varphi \) is true in interval \( I = M(n) + k \), where \( M(n) \) denotes a non-negative integer multiple of \( n \), and \( 0 \leq k \leq n - 1 \), while \( \text{UM}_{t, \varphi = k \% n} \varphi \) checks if the number of times \( \varphi \) is true at some point \( i \) in \( I \), \( \varphi_2 \) is true at \( j \), \( \varphi_1 \) holds in between \( i, j \), and the number of times \( \psi \) is true between \( i, j \) is \( M(n) + k \), \( 0 \leq k \leq n - 1 \). As an example, \( \psi = \text{true} \text{UM}(0, 1, 3 \% 2, a \lor b) \), when asserted at point \( i \), checks the number of points \( i \) in \( I \), \( \varphi_2 \) is true at \( j \), \( \varphi_1 \) holds between \( i, j \), and the number of points \( i \) where \( b \) is true is odd. Both these modalities can be rewritten equivalently in RatMTL as follows: \( \text{MC}^{\%a}_t \varphi \equiv \text{Rat}_t \rho_{\text{mod}} \) and \( \text{UM}_{t, \varphi = k \% n} \varphi \equiv \text{Rat}_t \rho_{\text{mod}} \phi_2 \), where \( \rho_{\text{mod}} = ([\varphi]^* \varphi \ldots (\neg \varphi)^* \varphi]^* ([\neg \varphi]^* \varphi \ldots (\neg \varphi)^* \varphi]^* \). The extension of MTL (MITL) with only \( \text{UM} \) is denoted \( \text{MTL} + \text{UM} \) (MITL + UM) while \( \text{MTL} + \text{MC} \) (MITL + MC) denotes the extension using MC.

3 Satisfiability of RatMTL and Complexity

The main results of this section are as follows.

\( \text{Theorem 2.} \) (1) Satisfiability of RatMTL is decidable. (2) Satisfiability of MTL + UM is EXPSPACE-complete. (3) Satisfiability of MTL + URat is in 2EXPSPACE.

(4) Satisfiability of MTL + MC is \( \mathcal{F}_{\omega^n} \)-hard.

Details of Theorems 2 and 2.3 can be found in Appendices E.4 and E.5.

\( \text{Theorem 3.} \) MTL + URat \( \subseteq \) MTL + Rat, MTL + UM \( \subseteq \) MTL + MC.

Theorem 3 shows that the Rat modality can capture URat (and likewise, MC captures UM). Thus, RatMTL \( \equiv \) MTL + Rat. Observe that any \( \rho_e \) can be decomposed into finitely many
Equisatisfiability

We will use the technique of equisatisfiability modulo oversampling \cite{10} in the proof of Theorem 2. Using this technique, formulae \( \varphi \) in one logic (say \( \text{RatMTL} \)) can be transformed into formulae \( \psi \) over a simpler logic (say \( \text{MTL} \)) such that whenever \( \rho \models \varphi \) for a timed word \( \rho \) over alphabet \( \Sigma \), one can construct a timed word \( \rho' \) over an extended set of positions and an extended alphabet \( \Sigma' \) such that \( \rho' \models \psi \) and vice-versa \cite{10, 11}. In oversampling, (i) \( \text{dom}(\rho') \) is extended by adding some extra positions between the first and last point of \( \rho \), (ii) the labeling of a position \( i \in \text{dom}(\rho) \) is over the extended alphabet \( \Sigma' \supset \Sigma \) and can be a superset of the previous labeling over \( \Sigma \), while the new positions are labeled using only the new symbols \( \Sigma' \setminus \Sigma \). We can recover \( \rho \) from \( \rho' \) by erasing the new points and the new symbols. A restricted use of oversampling, when one only extends the alphabet and not the set of positions of a timed word \( \rho \) is called simple extension. In this case, if \( \rho' \) is a simple extension of \( \rho \), then \( \text{dom}(\rho) = \text{dom}(\rho') \), and by erasing the new symbols from \( \rho' \), we obtain \( \rho \). See Figure 1 for an illustration. The formula \( \psi \) over the larger alphabet \( \Sigma' \supset \Sigma \) such that \( \rho' \models \psi \) if \( \rho \models \varphi \) is said to be equisatisfiable modulo temporal projections to \( \varphi \). In particular, \( \psi \) is equisatisfiable to \( \varphi \) modulo simple extensions or modulo oversampling, depending on how the word \( \rho' \) is constructed from the word \( \rho \). The oversampling technique

\[
\begin{array}{cccccccccccc}
\rho & a & b & c & d & e & a & b & c & d & e & \text{red points} \\
\rho_1 & a & b & c & d & e & a & b & c & d & e & \text{red points} \\
\rho_2 & c & a & b & d & e & a & b & c & d & e & \text{red points} \\
\rho_3 & c & a & b & d & e & a & b & c & d & e & \text{red points} \\
\end{array}
\]

**Figure 1** \( \rho \) is over \( \Sigma = \{a\} \) and satisfies \( \varphi = \square_{(0,1)}a \). \( \rho_1 \) is an oversampling of \( \rho \) over an extended alphabet \( \Sigma_1 = \Sigma \cup \{b, d\} \) and satisfies \( \psi_1 = \square(b \leftrightarrow \neg a) \land (\neg b \cup_{(0,1)} b) \). The red points in \( \rho_1 \) are the oversampling points. \( \rho_2 \) is a simple extension of \( \rho \) over an extended alphabet \( \Sigma_2 = \Sigma \cup \{c\} \) and satisfies \( \psi_2 = \square(c \leftrightarrow \square_{(0,1)} a) \land c \). It can be seen that \( \psi_1 \) is equivalent to \( \varphi \) modulo oversampling, and \( \psi_2 \) is equivalent to \( \varphi \) modulo simple extensions using the (respectively oversampling, simple) extensions \( \rho_1, \rho_2 \) of \( \rho \). However, \( \rho_3 \) above, obtained by merging \( \rho_1, \rho_2 \), eventhough an oversampling of \( \rho \), is not a good model for the formula \( \psi_1 \land \psi_2 \) over \( \Sigma_1 \cup \Sigma_2 \). However, we can relativize \( \psi_1 \) and \( \psi_2 \) with respect to \( \Sigma \) as \( \square(\text{act}_1 \rightarrow (b \leftrightarrow \neg a)) \land (\text{act}_1 \rightarrow \neg b) \cup_{(0,1)} b \land \text{act}_1 \) and \( \square(\text{act}_2 \rightarrow (c \leftrightarrow \square_{(0,1)} c \land \text{act}_2) \rightarrow a) \) where \( \text{act}_1 = \sqrt[\Sigma_1]{} \), \( \text{act}_2 = \sqrt[\Sigma_2]{} \). The relativized formula \( \kappa = \text{Rel}(\psi_1, \Sigma) \land \text{Rel}(\psi_2, \Sigma) \) is then equisatisfiable to \( \varphi \) modulo oversampling, and \( \rho_3 \) is indeed an oversampling of \( \rho \) satisfying \( \kappa \). This shows that while combining formulae \( \psi_1, \psi_2 \) which are equivalent to formulae \( \varphi_1, \varphi_2 \) modulo oversampling, we need to relativize \( \psi_1, \psi_2 \) to obtain a conjunction which will be equisatisfiable to \( \varphi_1 \land \varphi_2 \) modulo oversampling. See \cite{10} for details.

Equisatisfiable Reduction: \( \text{RatMTL} \) to \( \text{MTL} \)

Let \( \varphi \) be a \( \text{RatMTL} \) formula. To obtain equisatisfiable MTL formula \( \psi \), we do the following.
1. We “flatten” the reg modalities to simplify the alphabet. Each of the modalities \( \text{Rat}_i, \text{URat} \) that appear in the formula \( \varphi \) are replaced with fresh witness propositions to obtain a flattened formula. For example, if \( \varphi = \text{Rat}_{(0,1)}[a \text{URat}_{(1,2)} \text{Rat}_{(0,1)}(a+b)]_b \), then flattening yields the formula \( \varphi_{\text{flat}} = \varphi_1 \land \box{}[w_1 \leftrightarrow \text{Rat}_{(0,1)}w_2] \land \box{}[w_2 \leftrightarrow a \text{URat}_{(1,2)}w_3] \land \box{}[w_3 \leftrightarrow \text{Rat}_{(0,1)}(a+b)] \), where \( w_1, w_2, w_3 \) are fresh witness propositions. Let \( W \) be the set of fresh witness propositions such that \( \Sigma \cap W = \emptyset \). After flattening, the modalities \( \text{Rat}_i, \text{URat} \) appear only in this simplified form as \( \box{}[w \leftrightarrow \text{Rat}_i, \text{URat}] \). This simplified appearance of reg modalities are called temporal definitions and have the form \( \box{}[w \leftrightarrow \text{Rat}_i, \text{URat}] \) or \( \box{}[w \leftrightarrow x \text{URat}_{i', \text{atom}}y] \), where \( \text{atom} \) is a rational expression over \( \Sigma \cup W_i \), \( W_i \) being the set of fresh witness propositions used in the flattening, and \( i' \) is either a unit length interval or an unbounded interval.

2. The elimination of reg modalities is achieved by obtaining equisatisfiable MTL formulae \( \psi_i \) over \( X_i \), possibly a larger set of propositions than \( \Sigma \cup W_i \). Let \( Th_i = \bigwedge_1^i \text{Rel}(\psi_i, \Sigma) \) that is equisatisfiable to \( \varphi \) (see Figure 1 for relativization).

The above steps are routine [10], [11]. What remains is to handle the temporal definitions.

**Embedding the Runs of the DFA**

For any given \( \rho \) over \( \Sigma \cup W_i \), where \( W_i \) is the set of witness propositions used in the temporal definitions \( T_i \) of the forms \( \box{}[w \leftrightarrow \text{Rat}_i, \text{URat}] \) or \( \box{}[w \leftrightarrow x \text{URat}_{i', \text{atom}}y] \), the rational expression \( \text{atom} \) has a corresponding minimal DFA recognizing it. We define an LTL formula \( \text{GOODRUN}(\phi_e) \) which takes a formula \( \phi_e \) as a parameter with the following behaviour. \( \rho, i \models \text{GOODRUN}(\phi_e) \) iff for all \( k > i \), \( (\rho, k \models \phi_e) \rightarrow (\rho[i,k] \in L(\text{atom})) \). To achieve this, we use two new sets of symbols Threads and Merge for this information. This results in the extended alphabet \( \Sigma \cup W \cup \text{Threads} \cup \text{Merge} \) for the simple extension \( \rho' \) of \( \rho \). The behaviour of Threads and Merge are explained below.

Consider \( \text{atom} = \text{re}(S) \). Let \( A_{\text{atom}} = (Q, 2^Q, \delta, q_1, Q_F) \) be the minimal DFA for \( \text{atom} \) and let \( Q = \{q_1, q_2, \ldots, q_m\} \). Let \( n = \{1, 2, \ldots, m\} \) be the indices of the states. Conceptually, we consider multiple runs of \( A_{\text{atom}} \) with a new run (new thread) started at each point in \( \rho \). \text{Threads} records the state of each previously started run. At each step, each thread is updated from it previous value according to the transition function \( \delta \) of \( A_{\text{atom}} \) and also augmented with a new run in initial state. Potentially, the number of threads would grow unboundedly in size but notice that once two runs are the same state at position \( i \) they remain identical in future. Hence they can be merged into single thread (see Figure 2). As a result, \( m \) threads suffice. We record whether threads are merged in the current state using variables \( \text{Merge} \). An LTL formula records the evolution of Threads and Merge over any behaviour \( \rho \). We can define formula \( \text{GOODRUN}(\phi_e) \) in LTL over Threads and Merge.

1. At each position, let \( \text{Th}_i(q_x) \) be a proposition that denotes that the \( i \)th thread is active and is in state \( q_x \), while \( \text{Th}_i(\bot) \) be a proposition that denotes that the \( i \)th thread is not active. The set Threads consists of propositions \( \text{Th}_i(q_x), \text{Th}_i(\bot) \) for \( 1 \leq i, x \leq m \).

2. If at a position \( e \), we have \( \text{Th}_i(q_x) \) and \( \text{Th}_j(q_y) \) for \( i < j \), and if \( \delta(q_x, \sigma_e) = \delta(q_y, \sigma_e) \), then we can merge the threads \( i, j \) at position \( e+1 \). Let merge\((i, j)\) be a proposition that signifies that threads \( i, j \) have been merged. In this case, merge\((i, j)\) is true at position \( e + 1 \). Let Merge be the set of all propositions merge\((i, j)\) for \( 1 \leq i < j \leq m \).

We now describe the conditions to be checked in \( \rho' \).
- **Initial condition**($\varphi_{init}$)- At the first point of the word, we start the first thread and initialize all other threads as $\bot: \varphi_{init} = ((\mathsf{Th}_1(q_1)) \land \bigwedge_{1 < i \leq m} \mathsf{Th}_i(\bot))$.

- **Initiating runs at all points**($\varphi_{start}$)- To check the rational expression within an arbitrary interval, we need to start a new run from every point. $\varphi_{start} = \Box_{m}^{\mathsf{ns}} (\bigvee_{1 \leq i \leq m} \mathsf{Th}_i(q_1))$

- **Disallowing Redundancy**($\varphi_{no-red}$)- At any point of the word, if $i < j$ and $\mathsf{Th}_i(q_x)$ and $\mathsf{Th}_j(q_x)$ are both true, $q_x \neq q_y$. $\varphi_{no-red} = \bigwedge_{x \in \mathbb{L}} \Box_{m}^{\mathsf{ns}} \neg \bigvee_{1 \leq i < j \leq m} (\mathsf{Th}_i(q_x) \land \mathsf{Th}_j(q_x))$.

\[
\delta(q_1, a) = q_2, \delta(q_2, a) = q_3, \delta(q_3, a) = q_1
\]

\[
\delta(q_1, b) = q_2, \delta(q_2, b) = q_3, \delta(q_3, b) = q_1
\]

| a | a | a | a | b | b |
|---|---|---|---|---|---|
| $\mathsf{Th}_0$ | $\mathsf{Th}_1$ | $\mathsf{Th}_2$ | $\mathsf{Th}_3$ | $\mathsf{Th}_4$ | $\mathsf{Th}_5$ |
| $q_0$ | $q_1$ | $q_2$ | $q_3$ | $q_2$ | $q_1$ |
| $\mathsf{merge}(1, 2)$ | $\mathsf{merge}(1, 3)$ | $\mathsf{merge}(1, 2)$ | $\mathsf{merge}(1, 3)$ | $\mathsf{merge}(1, 3)$ | $\mathsf{merge}(1, 2)$ |

**Figure 2** Depiction of threads and merging. At time point 2.7, thread 2 is merged with 1, since they both had the same state information. This thread remains inactive till time point 8.8, where it becomes active, by starting a new run in state $q_1$. At time point 8.8, thread 3 merges with thread 1, while at time point 11, thread 2 merges with 1, but is reactivated in state $q_1$.

- **Merging Runs**($\varphi_{merge}$)- If two different threads $\mathsf{Th}_i, \mathsf{Th}_j (i < j)$ reach the same state $q_x$ on reading the input at the present point, then we merge thread $\mathsf{Th}_j$ with $\mathsf{Th}_i$. We remember the merge with the proposition $\mathsf{merge}(i, j)$. We define a macro $\mathsf{Nxt}(\mathsf{Th}_i(q_x))$ which is true at a point $e$ if and only if $\mathsf{Th}_i(q_y)$ is true at $e$ and $\delta(q_y, \sigma_e) = q_x$, where $\sigma_e \subseteq \mathsf{AP}$ is the maximal set of propositions true at $e$:

\[
\forall \{(q_y, \text{prop}) \in (\mathbb{Q}, (2 \cup \mathbb{F})) | \delta(q_y, \text{prop}) = q_x\}
\]

Let $\psi(i, j, k, q_x)$ be a formula that says that at the next position, $\mathsf{Th}_i(q_x)$ and $\mathsf{Th}_k(q_x)$ are true for $k > i$, but for all $j < i$, $\mathsf{Th}_j(q_x)$ is not. $\psi(i, j, k, q_x)$ is given by $\mathsf{Nxt}(\mathsf{Th}_i(q_x)) \land \bigwedge_{j < i} \neg \mathsf{Nxt}(\mathsf{Th}_j(q_x)) \land \mathsf{Nxt}(\mathsf{Th}_k(q_x))$. In this case, we merge threads $\mathsf{Th}_i, \mathsf{Th}_k$, and either restart $\mathsf{Th}_k$ in the initial state, or deactivate the $k$th thread at the next position. This is given by the formula $\mathsf{NextMerge}(i, k) = \mathsf{O}[^{(\mathsf{merge}(i, k) \lor (\mathsf{Th}_k(\bot) \lor (\mathsf{Th}_i(q_1) \land \mathsf{Th}_k(q_x))))}]$. $\varphi_{merge} = \bigwedge_{x, z, k \in \mathbb{L} \land k > i} \Box_{m}^{\mathsf{ns}} [\psi(i, j, k, q_x) \rightarrow \mathsf{NextMerge}(i, k)]$.

- **Propagating runs**($\varphi_{pro}, \varphi_{NO-pro}$)- If $\mathsf{Nxt}(\mathsf{Th}_i(q_x))$ is true at a point, and if for all $j < i$, $\neg \mathsf{Nxt}(\mathsf{Th}_j(q_x))$ is true, then at the next point, we have $\mathsf{Th}_i(q_x)$. Let $\mathsf{NextTh}(i, j, q_x)$ denote the formula $\mathsf{Nxt}(\mathsf{Th}_i(q_x)) \land \neg \mathsf{Nxt}(\mathsf{Th}_j(q_x))$. The formula $\varphi_{pro}$ is given by $\bigwedge_{i, j \in \mathbb{L} \land i < j} \Box_{m}^{\mathsf{ns}} [\mathsf{NextTh}(i, j, q_x) \rightarrow \mathsf{O}[^{(\mathsf{merge}(i, j))}]]$. If $\mathsf{Th}_i(\bot)$ is true at the current point, then at the next point, either $\mathsf{Th}_i(\bot)$ or $\mathsf{Th}_i(q_1)$. The latter condition corresponds to starting a new run on thread $\mathsf{Th}_i$. $\varphi_{NO-pro} = \bigwedge_{i \in \mathbb{L}} \Box_{m}^{\mathsf{ns}} \mathsf{Th}_i(\bot) \rightarrow \mathsf{O}[^{(\mathsf{Th}_i(\bot) \lor \mathsf{Th}_i(q_1))}]$

Let $\mathsf{Run}$ be the formula obtained by conjuncting all formulas explained above. Once we construct the simple extension $\rho'$, checking whether the rational expression $\mathsf{atom}$ holds in some interval $I$ in the timed word $\rho$, is equivalent to checking that if $u$ is the first action point within $I$, and if $\mathsf{Th}_i(q_1)$ holds at $u$, then after a series of merges of the form $\mathsf{merge}(i_1, i_1), \mathsf{merge}(i_2, i_1), \ldots, \mathsf{merge}(j, i_n)$, at the last point $v$ in the interval $I$, $\mathsf{Th}_j(q_f)$ is true, for some final state $q_f$. This is encoded as $\mathsf{GOODRUN}(q_f)$. It can be seen that the number of possible sequences of merges is bounded. Figure 2 illustrates the threads and merging. We can easily write a 1- TPTL formula that will check the truth of $\mathsf{Rat}^i_{(u, q_1)}\mathsf{atom}$ at a point $v$ on the simple extension $\rho'$ (see Appendix C). However, to write an MTL formula
that checks the truth of $\text{Rat}_{[\mu]}^{\text{atom}}$ at a point $v$, we need to oversample $\rho'$ as shown below.

![Diagram](image)

**Figure 3** The linking thread at $c_{j_\oplus u}$. The points in red are the oversampling integer points, and so are $\tau_v + l$ and $\tau_v + u$.

**Lemma 4.** Let $T = \Box[\mu][w \leftrightarrow \text{Rat}_{\text{atom}}]$ be a temporal definition built from $\Sigma \cup W$. Then we synthesize a formula $\psi \in \text{MTL}$ over $\Sigma \cup W \cup X$ such that $T$ is equivalent to $\psi$ modulo oversampling.

**Proof.** Let’s first consider the case when the interval $I$ is bounded of the form $[l, u)$. Consider a point in $\rho'$ with time stamp $\tau_v$. To assert $w$ at $\tau_v$, we look at the first action point after time point $\tau_v + l$, and check that $\text{GOODRUN}(\text{last}(qf))$ holds, where $\text{last}(qf)$ identifies the last action point just before $\tau_v + u$. The first difficulty is the possible absence of time points $\tau_v + l$ and $\tau_v + u$. To overcome this difficulty, we oversample $\rho'$ by introducing points at times $t + l, t + u$, whenever $t$ is a time point in $\rho'$. These new points are labelled with a new proposition $\text{ov}_{s}$. Sadly, $\text{last}(qf)$ cannot be written in $\text{MTL}$.

To address this, we introduce new time points at every integer point of $\rho'$. The starting point 0 is labelled $c_0$. Consecutive integer time points are marked $c_i, c_j \oplus 1$, where $\oplus$ is addition modulo the maximum constant used in the time interval in the $\text{RatMTL}$ formula. This helps in measuring the time elapsed since the first action point after $\tau_v + l$, till the last action point before $\tau_v + u$ as follows: if $\tau_v + l$ lies between points marked $c_j, c_j \oplus 1$, then the last integer point before $\tau_v + u$ is uniquely marked $c_{j_\oplus u}$.

- Anchoring at $\tau_v$, we assert the following at distance $l$: no action points are seen until the first action point where $Th_i(q_i)$ is true for some thread $Th_i$. Consider the next point where $c_j \oplus u$ is seen. Let $Th_{k_1}$ be the thread to which $Th_i$ has merged at the last action point just before $c_{j_\oplus u}$. Let us call $Th_{k_1}$ the “last merged thread” before $c_{j_\oplus u}$. The sequence of merges from $Th_i$ till $Th_{k_1}$ asserts a prefix of the run that we are looking for between $\tau_v + l$ and $\tau_v + u$. To complete the run we mention the sequence of merges from $Th_{k_1}$ which culminates in some $Th_{k_2}(q_f)$ at the last action point before $\tau_v + u$.

- Anchoring at $\tau_v$, we assert the following at distance $u$: we see no action points since $Th_{k_2}(q_f)$ at the action point before $\tau_v + u$ for some thread $Th_{k_2}$, and there is a path linking thread $Th_{k_2}$ to $Th_{k_1}$ since the point $c_{j_\oplus u}$. We assert that the “last merged thread”, $Th_{k_2}$ is active at $c_{j_\oplus u}$ : this is the linking thread which is last merged into before $c_{j_\oplus u}$, and which is the first thread which merges into another thread after $c_{j_\oplus u}$. These two formulae thus “stitch” the actual run observed between points $\tau_v + l$ and $\tau_v + u$. The formal technical details can be seen in Appendix D. If $I$ was an unbounded interval of the form $[l, \infty)$, then we will go all the way till the end of the word, and assert $Th_{k_2}(q_f)$ at the last action point of the word. Thus, for unbounded intervals, we do not need any oversampling at integer points.
In a similar manner, we can eliminate the URat modality, the proof of which can be found in Appendix C. If we choose to work on logic MITL + URat, we obtain a 2EXPSPACE upper bound for satisfiability checking, since elimination of URat results in an equisatisfiable MITL formula. This is an interesting consequence of the oversampling technique: without oversampling, we can eliminate URat obtaining 1-TPTL (Appendix C). However, 1-TPTL does not enjoy the benefits of non-punctuality, and is non-primitive recursive (Appendix F).

4 Automaton-Metric Temporal Logic-Freeze Logic Equivalences

The focus of this section is to obtain equivalences between automata, temporal and freeze logics. First of all, we identify a fragment of RATMTL denoted SfrMTL, where the rational expressions in the formulae are all star-free. We then show the equivalence between po-1-clock ATA, 1-TPTL, and SfrMTL (po-1-clock ATA ⊆ SfrMTL ⊆ 1-TPTL ≡ po-1-clock ATA). The main result of this section gives a tight automaton-logic connection in Theorem 1 and is proved using Lemmas 7-8 and 9.

Theorem 5. 1-TPTL, SfrMTL and po-1-clock ATA are all equivalent.

We first show that partially ordered 1-clock alternating timed automata (po-1-clock ATA) capture exactly the same class of languages as 1-TPTL. We also show that 1-TPTL is equivalent to the subclass SfrMTL of RATMTL where the rational expressions re involved in the formulae are such that \(L(re)\) is star-free.

A 1-clock ATA \((\Sigma, S, s_0, F, \delta)\), where \(\Sigma\) is a finite alphabet, \(S\) is a finite set of locations, \(s_0 \in S\) is the initial location and \(F \subseteq S\) is the set of final locations. Let \(x\) denote the clock variable in the 1-clock ATA, and \(x \triangleright c\) denote a clock constraint where \(c \in \mathbb{N}\) and \(\triangleright \in \{<, \leq, >, \geq\}\). Let \(X\) denote a finite set of clock constraints of the form \(x \triangleright c\). The transition function is defined as \(\delta : S \times \Sigma \rightarrow \Phi(S \cup \Sigma \cup X)\) where \(\Phi(S \cup \Sigma \cup X)\) is a set of formulae defined by the grammar \(\varphi ::= \top | \bot \land \varphi_1 \lor \varphi_2 | \varphi_1 \lor \varphi_2 \mid x \triangleright c \mid \varphi x \mid \varphi\) where \(s \in S\), and \(\varphi\) is a binding construct corresponding to resetting the clock \(x\) to 0.

The notation \(\Phi(S \cup \Sigma \cup X)\) thus allows boolean combinations as defined above of locations, symbols of \(\Sigma\), clock constraints and \(\top, \bot\), with or without the binding construct \((x)\). A configuration of a 1-clock ATA is a set consisting of locations along with their clock valuation. Given a configuration \(C\), we denote by \(\delta(C, a)\) the configuration \(D\) obtained by applying \(\delta(s, a)\) to each location \(s\) such that \((s, \nu) \in C\). A run of the 1-clock ATA starts from the initial configuration \(\{(s_0, 0)\}\), and proceeds with alternating time elapse transitions and discrete transitions obtained on reading a symbol from \(\Sigma\). A configuration is accepting iff it is either empty, or is of the form \(\{(s, \nu) \mid s \in F\}\). The language accepted by a 1-clock ATA \(A\), denoted \(L(A)\) is the set of all timed words \(\rho\) that starts from \(\{(s_0, 0)\}\), reading \(\rho\) leads to an accepting configuration. A po-1-clock ATA is one in which (i) there is a partial order denoted \(<\) on the locations, such that whenever \(s_j\) appears in \(\Phi(s_i)\), \(s_j \prec s_i\), or \(s_j = s_i\). Let \(\downarrow s_i = \{x \mid x < s_i\}\), (ii) \(x.s\) does not appear in \(\delta(s, a)\) for all \(s \in S, a \in \Sigma\).

Example 6. Consider the po-1-clock ATA \(A = \{[a, b], \{s_0, s_a, s_e\}, s_0, \{s_0, s_e\}, \delta\}\) with transitions \(\delta(s_0, b) = s_0, \delta(s_0, a) = (s_0 \land x < 1) \lor (x > 1) = \delta(s_a, b), \delta(s_e, a) = \bot\). The automaton accepts all strings where every non-last \(a\) has no symbols at distance 1 from it, and has some symbol at distance > 1 from it.

Lemma 7. po-1-clock ATA and 1-TPTL are equivalent in expressive power.

The translation from 1-TPTL to po-1-clock ATA is easy, as in the translation from MTL to po-1-clock ATA. For the reverse direction, we start from the lowest location (say \(s\)) in the
Making Metric Temporal Logic Rational

where □ partial order, and replace the transitions of which are accepted, when started in Beh. The idea is to iteratively keep replacing the □ modality level by level, starting with the innermost one, until we have eliminated the topmost one. Consider the SfrMTL formula \( \varphi = \text{Rat}(a_1)[\text{Rat}(a_2)(a + b)^+] \). To eliminate \( \text{Rat}(a_2)(a + b)^+ \) at a point, we freeze a clock \( x \), and wait till \( x \in (1, 2) \), and check \( (a + b)^+ \) on this region. The LTL formula for \( (a + b)^+ \) is \( \Box(a \lor b) \). Treating \( x \in (1, 2) \) as a proposition, we obtain \( \zeta = x \cdot (x \not\in (1, 2) \lor [\psi_1 \land \psi_2]) \) where \( \psi_1 = x \in (1, 2) \land (x \in (1, 2) \lor [\Box x \not\in (1, 2)]) \) and \( \psi_2 = [\Box x \in (1, 2) \to (a \lor b)] \). ζ asserts \( \Box(a \lor b) \) exactly on the region \( (1, 2) \), eliminating the modality \( \text{Rat}(a_2) \). To eliminate the outer \( \text{Rat}(a_1) \), we assert the existence of a point in \( (0, 1) \) where \( \text{Rat}(a_2)(a + b)^+ \) is true by saying \( x \cdot (x \not\in (0, 1) \lor [x \in (0, 1) \land \zeta \land \Box x \not\in (0, 1)]) \). This is 1-TPTL equivalent to \( \varphi \).

Lemma 8. SfrMTL \( \subseteq 1 \text{- TPTL} \)

The idea is to iteratively keep replacing the \( \text{Rat} \) modality level by level, starting with the innermost one, until we have eliminated the topmost one. Consider the SfrMTL formula \( \varphi = \text{Rat}(a_1)[\text{Rat}(a_2)(a + b)^+] \). To eliminate \( \text{Rat}(a_2)(a + b)^+ \) at a point, we freeze a clock \( x \), and wait till \( x \in (1, 2) \), and check \( (a + b)^+ \) on this region. The LTL formula for \( (a + b)^+ \) is \( \Box(a \lor b) \). Treating \( x \in (1, 2) \) as a proposition, we obtain \( \zeta = x \cdot (x \not\in (1, 2) \lor [\psi_1 \land \psi_2]) \) where \( \psi_1 = x \in (1, 2) \land (x \in (1, 2) \lor [\Box x \not\in (1, 2)]) \) and \( \psi_2 = [\Box x \in (1, 2) \to (a \lor b)] \). ζ asserts \( \Box(a \lor b) \) exactly on the region \( (1, 2) \), eliminating the modality \( \text{Rat}(a_2) \). To eliminate the outer \( \text{Rat}(a_1) \), we assert the existence of a point in \( (0, 1) \) where \( \text{Rat}(a_2)(a + b)^+ \) is true by saying \( x \cdot (x \not\in (0, 1) \lor [x \in (0, 1) \land \zeta \land \Box x \not\in (0, 1)]) \). This is 1-TPTL equivalent to \( \varphi \).

Lemma 9. (po-1-clock ATA to SfrMTL) Given a po-1-clock ATA \( A \), we can construct a SfrMTL formula \( \varphi \) such that \( L(A) = L(\varphi) \).

Proof. (Sketch) We give a proof sketch here, a detailed proof can be found in Appendix [3]. Let \( A \) be a po-1-clock ATA with locations \( S = \{s_0, s_1, \ldots, s_n\} \). Let \( K \) be the maximal constant used in the guards \( x \sim c \) occurring in the transitions. Let \( R_{2i} = [i, i], R_{2i+1} = (i, i + 1), 0 \leq i < K \) and \( R_K = (K, \infty) \) be the regions \( R \) of \( x \). Let \( R_h \prec R_k \) denote that region \( R_h \) precedes region \( R_k \). For each location \( s \), \( \text{Beh}(s) \) as computed in Lemma [3] \( A \) is a 1-TPTL formula that gives the timed behaviour starting at \( s \), using constraints \( x \sim c \) since the point where \( x \) was frozen. In example [5] \( \text{Beh}(s_{a}) = (x < 1) \cup^n x > 1 \), allows symbols \( a, b \) as long as \( x < 1 \) keeping the control in \( s_{a} \), has no behaviour at \( x = 1 \), and allows control to leave \( s_{a} \) when \( x > 1 \). For any \( s \), we “distribute” \( \text{Beh}(s) \) across regions by untiming it. In example [6] \( \text{Beh}(s_{a}) = \Box^n (a \lor b) \) for regions \( R_0, R_1 \), it is \( \perp \) for \( R_2 \) and is \( (a \lor b) \) for \( R_1 \). Given any \( \text{Beh}(s) \), and a pair of regions \( R_j \preceq R_k \), such that \( s \) has a non-empty behaviour in region \( R_j \), and control leaves \( s \) in \( R_k \), the untimed behaviour of \( s \) between regions \( R_j, R_k \) is written as LTL formulae \( \varphi_j, \ldots, \varphi_k \). This results in a “behaviour description” (or BD for short) denoted \( \text{BD}(s, R_j, R_k) \) : this is a \( 2K + 1 \) tuple with \( \text{BD}[R_j] = \varphi_j \) for \( j \leq l \leq k \), and \( \text{BD}[R] = \perp \) denoting “dont care” for the other regions. Let \( \text{BDS}(s) \) denote the set of all BDs for a location \( s \). For the initial location \( s_0 \), consider all \( \text{BD}(s_0, R_j, R_k) \in \text{BDS}(s_0) \) that have a behaviour starting in \( R_j \), and ends in an accepting configuration in \( R_k \). Each LTL formula \( \text{BD}(s_0, R_j, R_k)[R_i] \) (or BD for \( R_i \) when \( s_j, R_j, R_k \) are clear) is replaced with a star-free expression denoted \( \text{re}(\text{BD}(s_0, R_j, R_k)[R_i]) \). Then \( \text{BD}(s_0, R_j, R_k) \) is transformed into a SfrMTL formula \( \varphi(s_0, R_j, R_k) = \bigwedge_{0 \leq j \leq k \leq 2K} \text{Rat}_{R_j} \text{re}(\text{BD}(s_0, R_j, R_k)[R_i]) \). The language accepted by the po-1-clock ATA \( A \) is then given by \( \bigvee_{0 \leq j \leq k \leq 2K} \varphi(s_0, R_j, R_k) \). Computing \( \text{BD}(s, R_{i}, R_{j}) \) for a location \( s \) and pair of regions \( R_i \preceq R_j \). We first compute \( \text{BD}(s, R_i, R_j) \) for locations \( s \) which are lowest in the partial order, followed by computing \( \text{BD}(s', R_i, R_j) \) for locations \( s' \) which are higher in the order. For any location \( s \), \( \text{Beh}(s) \) has the form \( \varphi \lor \varphi_1 \text{W} \varphi_2 \) or \( \varphi \lor \varphi_1 \text{U} \varphi_2 \), where \( \varphi, \varphi_1, \varphi_2 \) are disjunctions of conjunctions over \( \Phi(S \cup \Sigma \cup X) \), where \( S \) is the set of locations with or without the binding construct
If the next point for Rat is described above, Rat are no region order, then BD defined for a region x.

This is done to detect if the “next point” for P x.

BD computed above for s,R.

BD for location x.

BD for location s lowest in po. Let s be a location that is lowest in the partial order. In general, if s is the lowest in the partial order, then Beh(s) has the form ϕ1 Wϕ2 or ϕ1 U∗ϕ2 or ϕ where ϕ, ϕ1, ϕ2 are disjunctions of conjunctions over Φ(Σ∪X). Each conjunct has the form ψ ∧ x ∈ R where ψ ∈ Φ(Σ∪S) and R ∈ R. Let ϕ1 = ∨(P1 ∧ C1), ϕ2 = ∨(Qj ∧ Ej) where P1, Qj ∈ Φ(Σ∪S) and C1, Ej ∈ R. Let C and E be shorthands for any Ck, Ek.

If Beh(s) is an expression without U, W (the case of ϕ above), then BD(s, R1, R2) is defined for a region R; if ϕ = ∨(Qj ∧ Ej) and there is some E1 with x ∈ R1. It is a 2K + 1 tuple with BD(s, R1, R2)[R1] = Q1, and the rest of the entries are ⊤ (don’t care). If Beh(s) has the form ϕ1 Wϕ2 or ϕ1 U*ϕ2, then for R1 ∼ R2, and a location s, BD(s, R1, R2) is a 2K+1 tuple with (i) formula ⊤ in regions R0, . . . , Rk−1, Rk+1, . . . , Rk of, (ii) if Ck = E1 = (x ∈ Rj) for some Ck, Ek, then the LTL formula in region Rj is Pb UQ1 if s is not accepting, and is Pb WQ1 if s is accepting, (iii) if no Ck is equal to any E1, and if E1 = (x ∈ Rj) for some l, then the formula in region Rj is Q1. If Cm = (x ∈ Rj) for some m, then the formula for region Rj is ⊤ Ow. If there is some Ch = (x ∈ Rw) for i < w < j, then the formula in region Rw is ⊤ Ow ∧ ϵ, where ϵ signifies that there may be no points in regions Rw. If there are no Cm’s such that Cm = (x ∈ Rw) for Rl ∼ Rw ∼ Rj, then the formula in region Rw is ϵ. ϵ is used as a special symbol in LTL whenever there is no behaviour in a region.

BD(s, R1, R2) for location s lowest in po. Let s be a location that is lowest in the partial order. In general, if s is the lowest in the partial order, then Beh(s) has the form ϕ1 Wϕ2 or ϕ1 U∗ϕ2 or ϕ where ϕ, ϕ1, ϕ2 are disjunctions of conjunctions over Φ(Σ∪X). Each conjunct has the form ψ ∧ x ∈ R where ψ ∈ Φ(Σ) and R ∈ R. See Figure 3 with regions R0, R1, R2, R∗ 1′, and some example BDs. In Figure 7 using the BDs of the lowest location s3, we write the SfrMTL formula for Beh(s3) : ψ(s3) = ϕR0(R3) ∧ ϕR1(R3) ∧ ϕR2(R3) ∧ ϕR3(R3), where each ϕR describes the behaviour of s3 starting from region R. For a fixed region R, ϕR = (ψ(s3, R1)) is described above. RatRε means that there is no behaviour in R. ϕR(R3) is given by RatRε ∨ ∨ (RatRε ρ ∧ RatRε [a + ϵ] ∧ RatRε [a + ϵ] ∧ RatRε [a + ϵ b])).

BD(s, R1, R2) for a location s which is higher up. If s is not the lowest in the partial order, then Beh(s) can have locations s′ ∈ s. s′ occurs as O(s′) or x O(s′) in Beh(s). For x OBeh(s3) in BD(s, R1, R2), since the clock is frozen, we plug-in the SfrMTL formula ψ(s3) computed above for x OBeh(s3) in BD(s1, R1, R2). For instance, in Figure 4 x OBeh(s3) appears in BD(s2, R1, R2)[R3]. We simply plug in the SfrMTL formula ψ(s3) in its place. Likewise, for locations s, t, if OBeh(t) occurs in BD(s, R1, R2)[Rk], we look up BD(t, R1, R2) in BDSet(t) for all Rk < Rl and combine BD(s, R1, R2), BD(t, Rk, Rl) in a manner described below. This is done to detect if the “next point” for t has a behaviour in Rk or later.

(a) If the next point for t is in Rk itself, then we combine BD1 = BD(s, R1, R2) with BD2 ∈ {BD(t, Rk, Rl) | Rk < Rl} ⊆ BDSet(t) as follows. combine(BD1, BD2) results in BD3 such that BD3[R] = BD1[R] for R < Rk, BD3[R] = BD1[R] ∧ BD2[R] for Rk < R, where

![Figure 4 A po-1-clock ATA with initial location s1 and s2, s3 are accepting.](image-url)
(b) Assume the next point for \( t \) lies in \( R_b \), \( R_b \prec R_b \). The difference with case (a) is that we combine \( BD_1 = BD(s, R_1, R_j) \) with \( BD_2 \in \{ BD(t, R_b, R_t) \mid R_b \prec R_b \leq R_t \} \subseteq BDSet(t) \). Then combine\((BD_1, BD_2)\) results in a \( BD \), say \( BD_3 \) such that \( BD_3[R] = BD_2[R] \) for \( R \prec R_b \), \( BD_3[R] = BD_1[R] \land BD_2[R] \) for all \( R_b \leq R \), and \( BD_3[R] = \epsilon \) for \( R_b \prec R \prec R_b \). The \( OBeh(t) \) in \( BD_1[R_b] \) is replaced with \( \square \bot \) to signify that the next point is not enabled for \( t \). See Figure 5 where \( R_b = R_2 \). The conjunction with \( \square \bot \) in \( R_b \) signifies that the next point for \( s_2 \) is not in \( R_0 \); the \( \epsilon \) in \( R_1 \) signifies that there are no points in \( R_1 \) for \( s_2 \). Conjoining \( \square \bot \) in a region signifies that the next point does not lie in this region.

\[
\begin{align*}
\text{BD}_1 & \quad \text{BD}(s_1, R_0, R_0) & \quad \alpha \land \Box \text{Beh}(s_2) \quad \text{don’t care} \quad \text{don’t care} \quad \text{don’t care} \\
\text{BD}_2 & \quad \text{BD}(s_2, R_2, R_2) & \quad \text{don’t care} \quad \text{don’t care} \quad \text{\( \square \)\( \alpha \land \epsilon \leq \Box \text{Beh}(s_3) \)} \quad \text{don’t care} \\
\text{combine} & \quad \text{BD}_1, \text{BD}_2 & \quad \alpha \land \square \bot \quad \epsilon \quad \text{\( \square \)\( \alpha \land \epsilon \leq \Box \text{Beh}(s_3) \)} \quad \text{don’t care} \\
\end{align*}
\]

\textbf{Figure 5} Combining BDs

We look at the “accepting” BDs in \( BDSet(s_0) \), viz., all \( BD(s_0, R_j, R_k) \), such that acceptance happens in \( R_k \), and \( s_0 \) has a behaviour starting in \( R_j \). The LTL formulae \( BD(s_0, R_j, R_k)[R] \) are replaced with star-free expressions \( \text{re}(BD(s_0, R_j, R_k)[R]) \). Each accepting \( BD(s_0, R_j, R_k) \) gives an \( \text{SfrMTL} \) formula \( \bigwedge_{R_j \leq R_k \leq R} \text{RatRE}(BD(s_0, R_j, R_k)[R]) \). The disjunction of these across all accepting BDs is the \( \text{SfrMTL} \) formula equivalent to \( L(A) \).

5 Discussion

We propose \( \text{RatMTL} \) which significantly increases the expressive power of \( \text{MTL} \) and yet retains decidability over pointwise finite words. The \( \text{Rat} \) operator added to \( \text{MTL} \) syntactically subsumes several other modalities in literature including threshold counting, modulo counting and the pnueli modality. Decidability of \( \text{RatMTL} \) is proved by giving an equisatisfiable reduction to \( \text{MTL} \) using oversampled temporal projections. This reduction has elementary complexity and allows us to identify two fragments of \( \text{RatMTL} \) with \( 2\text{EXPSPACE} \) and \( \text{EXPSPACE} \) satisfiability. In previous work \[10\], oversampled temporal projections were used to reduce \( \text{MTL} \) with punctual future and non-punctual past to \( \text{MTL} \). Our reduction can be combined with the one in \[10\] to obtain decidability of \( \text{RatMTL} \) and elementary decidability of \( \text{MITL} + \text{URat} \) + non-punctual past. These are amongst the most expressive decidable extensions of \( \text{MTL} \) known so far. We also show an exact logic-automaton correspondence between the fragment \( \text{SfrMTL} \) and \text{po-a-clock} ATA. Ouaknine and Worrell reduced \( \text{MTL} \) to \text{po-1 clock} ATA. Our \( \text{SfrMTL} \) achieves the converse too. It is not difficult to see that full \( \text{RatMTL} \) can be reduced to equivalent 1 clock alternating timed automata. This provides an alternative proof of decidability of \( \text{RatMTL} \) but the proof will not extend to decidability of \( \text{RatMTL} + \text{non-punctual past} \). Hence, we believe that our proof technique has some advantages. An interesting related formalism of timed regular expressions was defined by Asarin, Maler, Caspi, and shown to be expressively equivalent to timed automata. Our \( \text{RatMTL} \) has orthogonal expressivity, and it is boolean closed. The exact expressive power of \( \text{RatMTL} \) which is between 1-clock ATA and \text{po-1-clock} ATA is open.
References

1. R. Alur, T. Feder, and T. Henzinger. The benefits of relaxing punctuality. *J. ACM*, 43(1):116–146, 1996.
2. Rajeev Alur and Thomas A. Henzinger. Real-time logics: Complexity and expressiveness. *Inf. Comput.*, 104(1):35–77, 1993.
3. Augustin Baziramwabo, Pierre McKenzie, and Denis Thérien. Modular temporal logic. In *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*, pages 344–351, 1999.
4. Patricia Bouyer, Fabrice Chevalier, and Nicolas Markey. On the expressiveness of TPTL and MTL. In *FSTTCS 2005: Foundations of Software Technology and Theoretical Computer Science, 25th International Conference, Hyderabad, India, December 15-18, 2005, Proceedings*, pages 432–443, 2005.
5. Stéphane Demri and Ranko Lazic. LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Log.*, 10(3), 2009.
6. Cindy Eisner and Dana Fisman. *A Practical Introduction to PSL*. Springer, 2006.
7. IEEE P1850-Standard for PSL-Property Specification Language, 2005.
8. Jesper G. Henriksen and P. S. Thiagarajan. Dynamic linear time temporal logic. *Ann. Pure Appl. Logic*, 96(1-3):187–207, 1999.
9. P. Hunter. When is metric temporal logic expressively complete? In *CSL*, pages 380–394, 2013.
10. S. N. Krishna K. Madnani and P. K. Pandya. Partially punctual metric temporal logic is decidable. In *TIME*, pages 174–183, 2014.
11. Shankara Narayanan Krishna, Khushraj Madnani, and Paritosh K. Pandya. Metric temporal logic with counting. In *Foundations of Software Science and Computation Structures - 19th International Conference, FOSSACS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, pages 335–352, 2016.
12. F. Laroussinie, A. Meyer, and E. Petonnet. Counting ltl. In *TIME*, pages 51–58, 2010.
13. K. Lodaya and A. V. Sreejith. Ltl can be more succinct. In *ATVA*, pages 245–258, 2010.
14. M. Minsky. *Finite and Infinite Machines*. Prentice-Hall, 1967.
15. J. Ouaknine and J. Worrell. On the decidability of metric temporal logic. In *LICS*, pages 188–197, 2005.
16. Pavithra Prabhakar and Deepak D’Souza. On the expressiveness of MTL with past operators. In *FORMATS*, pages 322–336, 2006.
17. A. Rabinovich. Complexity of metric temporal logic with counting and pnueli modalities. In *FORMATS*, pages 93–108, 2008.
Appendix

A Rational Expressions and Star-Free Expressions

We briefly introduce rational expressions and star-free expressions over an alphabet $\Sigma$. A rational expression over $\Sigma$ is constructed inductively using the atomic expressions $a \in \Sigma, \epsilon, \emptyset$ and combining them using concatenation, Kleene-star and union.

A star-free expression also has the same atomic expressions, and allows combination using union, concatenation and complementation. For instance, $\Sigma^*$ is star-free since it can be written as $\neg \emptyset$.

B Exclusive Normal Form

We eliminate $\text{Rat}_{\text{atom}}$ and $x \text{\ul{U}} \text{Rat}_{\text{atom}}$ respectively from temporal definitions $\Box_{\text{ns}}[w \leftrightarrow \text{Rat}_{\text{atom}}]$ or $\Box_{\text{ns}}[w \leftrightarrow x \text{\ul{U}} \text{Rat}_{\text{atom}} y]$. The idea is to first mark each point of the timed word $\rho$ over $\Sigma \cup W$ with the information whether $\text{atom}$ is true or not at that point, obtaining a simple extension $\rho'$ of $\rho$, and then to refine this information by checking if $\text{atom}$ is true within an interval $I$.

Assume $\text{atom} = \text{re}(S)$. To say that $\text{re}(S)$ is true starting at a point in the timed word, we have to look at the truth of subformulae in $S$. The alphabet of the minimal DFA to check $\text{re}(S)$ is hence $2^S = S'$. This results in the minimal DFA accepting an expression $\text{re}'(S')$, and not $\text{re}(S)$. In the following, we show that $\text{re}'(S')$ is equivalent to $\text{re}(S)$.

The first thing we do to avoid dealing with sets of formulae of $S$ being true at each point is to assume that the sets $S$ are exclusive: that is, at any point, exactly one formula from $S$ can be true. If the sets $S$ are all exclusive, then the formula is said to be in Exclusive Normal Form. If $S$ is exclusive, then we will be marking positions in the word over $S$ and not $P(S)$. This way, the untimed words $\text{Seg}(S, i, j)$ as well as $\text{TSeg}(S, i, I)$ that were used in the semantics of $\varphi_1 \text{\ul{U}} \text{Rat}_{\text{re}(S)} \varphi_2$, $\text{Rat}_{\text{re}(S)}$ respectively will be words over $S$. The satisfaction of $\varphi_1 \text{\ul{U}} \text{Rat}_{\text{re}(S)} \varphi_2$, $\text{Rat}_{\text{re}(S)}$ at any point $i$ will then amount to simply checking if $\text{Seg}(S, i, j), \text{TSeg}(S, i, I) \in \text{L}(\text{re}(S))$.

We now show that the exclusiveness of $S$ can be achieved by a simple translation.

Lemma 10. Given any $\text{RatMTL}$ formula $\varphi$ of the form $\text{Rat}_{\text{re}(S)}[\varphi_1 \varphi_2]$, or $\varphi_1 \text{\ul{U}} \text{Rat}_{\text{re}(S)}[\varphi_2]$, there exists an equivalent formula $\psi \in \text{RatMTL}$ in exclusive normal form.

Proof. Let $S = \{\varphi_1, \ldots, \varphi_n\}$. Construct set $S'$ consisting of all formulae of the form $\bigwedge_{i \in K} \varphi_i \wedge \bigwedge_{i \in K^c} \neg \varphi_i$ for all possible subsets $K \subseteq \{1, 2, \ldots, n\}$.

Consider any formula of the form $\text{Rat}(\text{re}(S))$. Let $W_i$ denote the set consisting of all subsets of $\{1, 2, \ldots, K\}$ which contains $i$. The satisfaction of $\varphi_i$ is then equivalent to that of $\sum_{W \in W_i} \varphi_W$. We can thus replace any $\varphi_i$ occurring in $\text{re}(S)$ with $\sum_{W \in W_i} \varphi_W$. This results in obtaining a rational expression $\text{re}'$ over $S$.

It can be shown that $\text{re}(S)$ is equivalent to $\text{re}'(S')$ by inducting on the structure of $\text{re}$.

Thus, the minimal DFA we construct for $\text{re}(S)$ in the temporal definition will end up accepting $\text{re}'(S')$, equivalent to $\text{re}(S)$.

C 1-TPTL for $\text{Rat}_{(l,u)}$atom

We encode in 1-TPTL an accepting run going through a sequence of merges capturing $\text{Rat}_{(l,u)}$atom at a point $e$. To encode an accepting run going through a sequence of merges
we assert $\chi_{chk1} \lor \chi_{chk2} \lor \chi_{chk3}$. Recall that $m$ is the number of states in the minimal DFA accepting $\text{atom}$. 

Let $\text{cond1} = 0 \leq n < m$, and 

Let $\text{cond2} = 1 \leq i_1 < i_2 < \ldots < i_n < i \leq m$. 

\[ \begin{align*} 
\chi_{chk1} &= \bigvee_{\text{cond2}} \bigvee_{\text{cond1}} x. \hat{\phi}(x < l \land O[(x \geq l) \land \text{GoodRun}]) \\
\chi_{chk2} &= \bigvee_{\text{cond2}} \bigvee_{\text{cond1}} x. (O[(x \geq l) \land \text{GoodRun}]) \\
\chi_{chk3} &= \bigvee_{\text{cond1}} \bigvee_{\text{cond2}} x. \text{GoodRun}
\end{align*} \]

where $\text{GoodRun}$ is the formula which describes the run starting in $q_1$ in thread $Th_i$, going through a sequence of merges, and witnesses $q_f$ in a merged thread $Th_i$, at a point when $x \in [l, u)$, and is the maximal point in $[l, u)$. 

$\text{GoodRun}$ is given by $Th_i(q_1) \land \{\neg \text{Mrg}(i)\} \cup \text{merge}(i_1, i) \land \{\neg \text{Mrg}(i_n)\} \cup \text{merge}(i_{n-1}, i_n) \ldots \{\neg \text{Mrg}(i_2)\} \cup \text{merge}(i_1, i_2) \land \bigvee_{q \in Q_F} \text{Nxt}(Th_i(q)) \land x \in [l, u) \land O(x > u)]\}$. The next expected state of thread $Th_i$, is one of the final states if and only if the sub-string within the interval $[l, u)$ from the point $e$ satisfies the regular expression $\text{atom}$. Note that when the frozen clock is $\geq l$, we start the run with $Th_i(q_1)$, go through the merges, and check that $x \in I$ when we encounter a thread $Th_i(q_f)$, with $q_f$ being a final state. To ensure that we have covered checking all points in $\tau_e + I$, we ensure that at the next point after $Th_i(q_f)$, $x > u$. The decidability of $1^{-}\text{TPTL}$ gives the decidability of $\text{RatMTL}$. 

D Proof of Lemma 4 

Proof. Starting with the simple extension $\rho'$ having the information about the runs of $A_{\text{atom}}$, we explain the construction of the oversampled extension $\rho''$ as follows:

- We first oversample $\rho'$ at all the integer timestamps and mark them with propositions in $C = \{c_0, \ldots, c_{\text{max}-1}\}$ where $\text{max}$ is the maximum constant used in timing constraints of the input formulae. An integer timestamp $k$ is marked $c_i$ if and only if $k = M(\text{max}) + i$ where $M(\text{max})$ denotes a non-negative integral multiple of $\text{max}$ and $0 \leq i \leq \text{max} - 1$. This can be done easily by the formula $c_0 \land \bigwedge_{i \in \{0, \ldots, \text{max}-1\}} \bigwedge_{i=0}^{\text{max}} (c_i \rightarrow \neg \hat{\phi}_{(0, 1)}(\bigvee C) \land \hat{\phi}_{(0, 1)}[c_i \oplus 1])$ where $x \oplus y$ is addition of $x, y$ modulo $\text{max}$.

- Next, a new point marked $\text{ovs}$ is introduced at all time points $\tau$ whenever $\tau - t$ or $\tau - u$ is marked with $\bigvee \Sigma$. This ensures that for any time point $t$ in $\rho''$, the points $t + l, t + u$ are also available in $\rho''$.

After the addition of integer time points, and points marked $\text{ovs}$, we obtain the oversampled extension $(\Sigma \cup W \cup \text{Threads} \cup \text{Merge}, C \cup \{\text{ovs}\}) \rho''$ of $\rho'$.

To check the truth of $\text{Rat}_{[l, u]} \text{atom}$ at a point $v$, we need to assert the following: starting from the time point $\tau_v + l$, we have to check the existence of an accepting run $R$ in $A_{\text{atom}}$ such that the run starts from the first action point in the interval $[\tau_v + l, \tau_v + u)$, is a valid run.
which goes through some possible sequence of merging of threads, and witnesses a final state at the last action point in \([\tau_v + l, \tau_v + u]\). To capture this, we start at the first action point in \([\tau_v + l, \tau_v + u]\) with initial state \(q_1\) in some thread \(Th_1\), and proceed for some time with \(Th_1\) active, until we reach a point where \(Th_1\) is merged with some other thread. This is followed by \(Th_1\) remaining active until we reach a point where \(Th_1\) is merged with some other thread \(Th_2\) and so on, until we reach the last such merge where some thread say \(Th_n\) witnesses a final state at the last action point in \([\tau_v + l, \tau_v + u]\). A nesting of until formulae captures this sequence of merges of the threads, starting with \(Th_1\) in the initial state \(q_1\). Starting at \(v\), we have the point marked \(ovs\) at \(\tau_v + l\), which helps us to anchor there and start asserting the existence of the run.

The issue is that the nested until can not keep track of the time elapse since \(\tau_v + l\). However, note that the greatest integer point in \([\tau_v + l, \tau_v + u]\) is uniquely marked with \(c_{j\oplus u}\) whenever \(c_i \leq \tau_v \leq c_{i+1}\) are the closest integer points to \(\tau_v\). We make use of this by (i) asserting the run of \(A_{atom}\) until we reach \(c_{j\oplus u}\) from \(\tau_v + l\). Let the part of the run \(R\) that has been witnessed until \(c_{j\oplus u}\) be \(R_{pref}\). Let \(R = R_{pref} \cap R_{suf}\) be the accepting run. (ii) From \(\tau_v + l\), we jump to \(\tau_v + u\), and assert the reverse of \(R_{suf}\) till we reach \(c_{j\oplus u}\). This ensures that \(R = R_{pref} \cap R_{suf}\) is a valid run in the interval \([\tau_v + l, \tau_v + u]\).

Let \(Mrg(j) = \bigvee_{k<j} merge(k, j) \lor c_{j\oplus u}\).

We first write a formula that captures \(R_{pref}\). Given a point \(v\), the formula captures a sequence of merges through threads \(i > i_1 > \cdots > i_{k_1}\), and \(n\) is the number of states of \(A_{atom}\).

Let \(\varphi_{pref, k_1} = \bigvee_{m \geq i > i_1 > \cdots > i_{k_1}} MergeseqPref(k_1)\) where \(MergeseqPref(k_1)\) is the formula

\[
\bigvee_{l, j} \{ \neg(\bigvee c_{\oplus u}) \cup [Th_i(q_1) \land (\neg Mrg(i) \cup \text{merge}(i_1, i) \land \\
(\neg Mrg(i_1) \cup \text{merge}(i_2, i_1) \land \cdots \neg Mrg(i_{k_1}) \cup c_{j\oplus u})]\] \}
\]

Note that this asserts the existence of a run till \(c_{j\oplus u}\) going through a sequence of merges starting at \(\tau_v + l\). Also, \(Th_{k_1}\) is the guessed last active thread till we reach \(c_{i\oplus u}\) which will be merged in the continuation of the run from \(c_{i\oplus u}\).

Now we start at \(\tau_v + u\) and assert that we witness a final state sometime as part of some thread \(Th_{k_1}\), and walk backwards such that some thread \(i_t\) got merged to \(i_k\), and so on, we reach a thread \(Th_{i_1}\) to which thread \(Th_{k_1}\) merges with. Note that \(Th_{i_1}\) was active when we reached \(c_{i\oplus u}\). This thread \(Th_{i_1}\) is thus the “linking point” of the forward and reverse runs. See Figure 6.
Let \( \varphi_{\text{Suf},k,k_1} = \bigvee_{1 \leq i_k < \ldots < i_{k_1} \leq n} \text{MergeseqSuf}(k,k_1) \) where \( \text{MergeseqSuf}(k,k_1) \) is the formula

\[
\bigwedge_{[u,v]} \{ \neg( \forall \Sigma \cup c_{i\oplus u}) S[(\text{Th}_i(q_j)) \land \neg \text{Mrg}(i_k) \ S \ [\text{merge}(i_k, i_{k-1}) \land \neg \text{Mrg}(i_{k-1}) \ S \ [\text{merge}(i_{k-1}, i_{k-2}) \land \ldots \text{merge}(i_c, i_{k_1}) \land \neg \text{Mrg}(i_{k_1}) \ S \ c_{i\oplus u})]]]\}.
\]

For a fixed sequence of merges, the formula

\[
\varphi_{k,k_1} = \bigvee_{k \geq k_1 \geq 1} [\text{MergeseqPref}(k_1) \land \text{MergeseqSuf}(k,k_1)]
\]
captures an accepting run using the merge sequence. Disjuncting over all possible sequences for a starting thread \( \text{Th}_i \), and disjuncting over all possible starting threads gives the required formula capturing an accepting run. Note that this resultant formulae is also relativized with respect to \( \Sigma \) and also conjunction with \( \text{Rel}(\text{Run}, \Sigma) \) (where \( \text{Run} \) is the formula capturing the run information in \( \rho' \) as seen in section \( 3.1 \)) to obtain the equisatisfiable MTL formula. The relativization of \( \text{Run} \) with respect to \( \Sigma \) can be done as illustrated in Figure \( 1 \). Note that \( S \) can be eliminated obtaining an equisatisfiable MTL[\( U_I \)] formula modulo simple projections \( \text{[10]} \).

If \( I \) was an unbounded interval of the form \( [l, \infty) \), then in formula \( \varphi_{k,k_1} \), we do not require \( \text{MergeseqSuf}(k,k_1) \); instead, we will go all the way till the end of the word, and assert \( \text{Th}_i(q_j) \) at the last action point of the word. Thus, for unbounded intervals, we do not need any oversampling at integer points. ▶

**E Elimination of URat_{f,re}**

**Lemma 11.** Let \( T = [\forall x \psi_1 \leftrightarrow x \text{URat}_{f,re}] \) be a temporal definition built from \( \Sigma \cup W \). Then we synthesize a formula \( \psi \in \text{MTL} \) over \( \Sigma \cup W \cup X \) such that \( T \) is equivalent to \( \psi \) modulo oversampling.

**Proof.** We discuss first the case of bounded intervals. The proof technique is very similar to Lemma \( [4] \) The differences that arise are as below.

1. Checking \( \text{re} \) in \( \text{Rat}_{f,re} \) at point \( v \) is done at all points \( j \) such that \( \tau_j - \tau_v \in I \). To ensure this, we needed the punched modalities \( \diamond_{[u,v]} \), \( \diamond_{[l,q]} \). On the other hand, to check \( \text{URat}_{f,re} \) from a point \( v \), the check on \( \text{re} \) is done from the first point after \( \tau_v \), and ends at some point within \( [\tau_v + l, \tau_v + u] \). Assuming \( \tau_v \) lies between integer points \( c_1, c_{i\oplus 1} \), we can witness the forward run in \( \text{MergeseqPref} \) from the next point after \( \tau_v \) till \( c_{i\oplus 1} \), and for the reverse run, go to some point in \( \tau_v + I \) where the final state is witnessed in a merged thread, and walk back till \( c_{i\oplus 1} \), ensuring the continuity of the threads merged across \( c_{i\oplus 1} \). The punctual modalities are hence not required and we do not need points marked ovs.

2. The formulae \( \text{MergeseqPref}(k_1) \), \( \text{MergeseqSuf}(k,k_1) \) of the lemma \( [4] \) are replaced as follows:

   - \( \text{MergeseqPref}(k_1) : [\neg( \forall \Sigma \cup c_{i\oplus 1}) \cup \text{Th}_i(q_j) \land \neg \text{Mrg}(i) \cup \text{merge}(i_2, i_1) \land \ldots \neg \text{Mrg}(i_{k_1}) \cup \text{merge}(i_{k_1}) \cup c_{i\oplus 1})]\).
   - \( \text{MergeseqSuf}(k,k_1) : [\text{merge}(i_{k-1}, i_{k-2}) \land \ldots \text{merge}(i_c, i_{k_1}) \land \neg \text{Mrg}(i_{k_1}) \cup c_{i\oplus 1})]\).

The above takes care of \( \text{re} \) in \( x \text{URat}_{f,re} \) : we also need to say that \( x \) holds continuously from the current point to some point in \( I \). This is done by pushing \( x \) into \( \text{re} \) (see the translation of \( \varphi_{x \text{URat}_{f,re} \varphi_2} \) to \( \text{Rat}_{f,re} \) in Appendix \( [3] \)). The resultant formulae is relativized with respect to \( \Sigma \) and also conjunction with \( \text{Rel}(\text{Run}, \Sigma) \) to obtain the equisatisfiable MTL formula.
Now we consider unbounded intervals. The major challenge for the unbounded case is that the point where we asserting $\text{Th}_{i_k}(q_f)$ (call this point $w$) may be far away from the point $v$ where we begin: that is, if $\tau_v$ is flanked between integer points marked $c_i$ and $c_{i+1}$, it is possible to see multiple occurrences of $c_{i+1}$ between $\tau_v$ and the point in $\tau_v + l$ which witnesses $\text{Th}_{i_k}(q_f)$. In this case, when walking back reading the reverse of the suffix, it is not easy to stitch it back to the first $c_{i+1}$ seen after $\tau_v$. The possible non-uniqueness of $c_{i+1}$ thus poses a problem in reducing our technique in the bounded interval case. Thus we consider two cases:

Case 1: In this case, we assume that our point $w$ lies within $[\tau_v + l, [\tau_v + l])$. Note that $[\tau_v + l]$ is the nearest point from $v$ marked with $c_{i+1}$. This can be checked by asserting $\neg c_{i+1}$ all the way till $c_{i+1}$ while walking backward from $w$, where $\text{Th}_{i_k}(q_f)$ is witnessed. The formula $\text{MergeseqPref}(k_1)$ does not change. $\text{MergeseqSuf}(k, k_1)$ is as follows:

$$\hat{\Diamond}_{[I+1,J+1]}\left\{\left(\text{Th}_{i_k}(q_f)\right) \land \left(\neg \text{Mrg}'(i_k) \text{Smerge}(i_k, i_{k-1}) \land \neg \text{Mrg}'(i_{k-1})\right)\right\}$$

where $\text{Mrg}'(i) = \left\lceil \bigvee_{j < i} \text{merge}(j, i) \land c_{i+1} \right\rceil$

Case 2: In this case, we assume the complement. That is the point $w$ occurs after $[\tau_v + l]$. In this case, we assert the prefix till $c_{i+1}$ and then continue asserting the suffix from this point in the forward fashion unlike other cases. The changed $\text{MergeseqPref}$ and $\text{MergeseqSuf}$ are as follows:

- $\text{MergeseqPref}(k_1)$:
  $$\neg \left\lceil \bigvee_{i \leq l} \text{merge}(i, j) \land \text{Th}_{i_1}(q_f) \right\rceil \land \left(\neg \text{Mrg}(i_1) \text{Umerge}(i_1, i) \land \neg \text{Mrg}(i_1) \text{Umerge}(i_1, i_1) \land \ldots \right)$$

- $\text{MergeseqSuf}(k, k_1)$:
  $$\hat{\Diamond}_{[I+1,J+2]}\left\{\left|c_{i+1}\right| \land \left(\neg \text{Mrg}(i_{k_1}) \text{Umerge}(i_{k_1}, i_{k_1}) \land \neg \text{Mrg}(i_{k_1}) \text{Umerge}(i_{k_1}, i_{k_1}) \land \ldots \right) \text{merge}(i_{k_1}, i_{k_1}) \land \ldots \text{merge}(i_{k_1} - 1, i_{k_1} - 2) \land \neg \text{Mrg}(i_{k_1} - 1) \text{Umerge}(i_{k_1} - 1, i_{k_1} - 2) \right\}$$

E.1 Complexity of RatMTL Fragments

Given a formula $\varphi$ in (MITL or MTL or RatMTL), the size of $\varphi$ denoted $|\varphi|$ is defined by taking into consideration, the number of temporal modalities $U_f$, $O_f$, the number of boolean connectives, as well as the maximal constant occurring in the formulae (encoded in binary). The size is defined as $\log K \times (\text{the number of temporal modalities in } \varphi + \text{number of boolean connectives in } \varphi)$, where $K$ is the max constant appearing in the formulae. For example, $|a \cup b (\neg b \land c) \cup d| = \log 2 \times 4$. In all our complexity results, we assume a binary encoding of all constants involved in the formulae.

To prove the complexity results we first need the following lemma.

 Lemma 12. Given any MITL formula $\varphi$ with $|\varphi| = O(2^n)$ (for some $n \in \mathbb{N}$) with maximum constant $K$ used in timing intervals, the satisfiability checking for $\varphi$ is EXPSPACE in $n$ and $\log(K)$.
Proof. Given any MITL formula \( \varphi \) with \(|\varphi| = \mathcal{O}(2^n) \), there are at most \( \expn = \mathcal{O}(2^n) \) number of temporal modalities and boolean connectives. Let \( K \) be the maximal constant used in \( \varphi \). We give a satisﬁability preserving reduction from \( \varphi \) to \( \psi \in MITL[U_{0,\infty}, S] \). MITL[\( U_{0,\infty}, S \)] is the fragment of MITL with untimed past and the intervals in future modalities are only of the form \((0, u)\) or \((l, \infty)\). The satisﬁability checking for MITL[\( U_{0,\infty}, S \)] is in PSPACE [1]. Hence, the reduction from a MITL formula with \( \expn = \mathcal{O}(2^n) \) number of modalities to an MITL[\( U_{0,\infty}, S \)] formula with \( \mathcal{O}(\text{poly}(K, \expn)) \) modalities preserving the max constant \( K \), gives an EXPSPACE upper bound in \( n, \log K \). The EXPSPACE hardness of MITL can be found in [1]. The reduction from MITL to MITL[\( U_{0,\infty}, S \)] is achieved as follows:

\[
K = 5 \quad [l, l + 1] = [2, 3] \quad \tau_p = 0.01 \quad \tau_j = 2.9
\]

![Figure 7](image)

The point \( p \) has \( \tau_p = 0.01 \), and for \( l = 2 \), \( \tau_p + [l, l + 1] = [2.01, 3.01] \). \( \tau_p \) lies between points \( c_{i-1} = c_0 \) and \( c_i = c_1 \). In this case, \( \tau_j = 2.9 \) where \( \phi_2 \) holds and \( c_3 = c_4 \) does not lie between \( \tau_p, \tau_j \).

\[
K = 5 \quad [l, l + 1] = [2, 3] \quad \tau_p = 0.5 \quad \tau_j = 3.2
\]

![Figure 8](image)

The point \( p \) has \( \tau_p = 0.5 \), and for \( l = 2 \), \( \tau_p + [l, l + 1] = [2.5, 3.5] \). \( \tau_p \) lies between points \( c_{i-1} = c_0 \) and \( c_i = c_1 \). In this case, \( \tau_j = 3.2 \) where \( \phi_2 \) holds and \( c_3 = c_4 \) lies between \( \tau_p, \tau_j \).

(a) Break each \( U_I \) formulae in MITL where \( I \) is a bounded interval, into disjunctions of \( U_I \) modality, where each \( I_i \) is a unit length interval and union of all \( I_i \) is equal to \( I \). That is, \( \phi_1 U_{[l,u]} \phi_2 \equiv \phi_1 U_{[l,l+1]} \phi_2 \lor \phi_1 U_{[l+1,l+2]} \phi_2 \lor \ldots \lor \phi_1 U_{[u-1,u]} \phi_2 \). This increases the size from \( \expn \) to \( \expn \times K \).

(b) Next, we flatten all the modalities containing bounded intervals. This results in replacing subformulae of the form \( \phi_1 U_{[l,l+1]} \phi_2 \) with new witness variables. This results in the conjunction of temporal deﬁnitions of the form \( \Box^\mathcal{M} [a \leftrightarrow \phi_1 U_{[l,l+1]} \phi_2] \) to the formula, and creates only a linear blow up in the size.

Now consider any temporal deﬁnition \( \Box^\mathcal{M} [a \leftrightarrow \phi_1 U_{[l,l+1]} \phi_2] \). We show a reduction to an equisatisﬁable MITL[\( U_{0,\infty}, S \)] formula by eliminating each \( \Box^\mathcal{M} [a \leftrightarrow \phi_1 U_{[l,l+1]} \phi_2] \) and replacing it with untimed \( S \) and \( U \) modalities with intervals \( <0, u> \) and \( <l, \infty> \).

- First we oversample the words at integer points \( C = \{c_0, c_1, c_2, \ldots, c_{K-1}\} \). An integer timestamp \( k \) is marked \( c_i \) if and only if \( k = M(K) + i \), where \( M(K) \) denotes a non-negative integer multiple of \( K \), and \( 0 \leq i \leq K - 1 \). This can be done easily by the formula

\[
c_{0} \land \bigwedge_{i \in \{0, \ldots, K-1\}} \Box^\mathcal{M} (c_i \rightarrow \neg \Diamond (0,1)(\forall C) \land \Diamond (0,1)\neg c_{i+1})
\]

where \( x \oplus y \) is \((x+y)\%K\) (recall that \((x+y)\%K = M(K) + (x+y), 0 \leq x+y \leq K - 1\)).
Consider any point \( p \) within a unit integer interval whose end points are marked \( c_{i-1}, c_i \). Then \( \phi_1 U_{[p, l+1]} \phi_2 \) is true at that point \( p \) if and only if, \( \phi_1 \) is true on all the action points till a point \( j \) in the future, such that

- either \( j \) occurs within \([l, \infty)\) from \( p \) and there is no \( c_i \) between \( p \) and \( j \) (\( \tau_j \in [\tau_p + l, [\tau_p + l]] \)) (see figure 7)

\[
\phi_{C1,p} = (\phi_1 \land \neg c_i) U_{[l, \infty]} \phi_2
\]

- or, \( j \) occurs within \([0, l+1)\) from \( p \), and \( j \) is within a unit interval whose end points are marked \( c_i \) and \( c_i \) (\( \tau_i \in [[\tau_p + l], [\tau_p + l + 1)] \)) (see figure 8)

\[
\phi_{C2,p} = \phi_1 U_{[0, l+1]} (\phi_2 \land (\neg (\forall C) Sc_i) \ depressions
\]

The temporal definition \( \Box^{\sigma}[a \leftrightarrow \phi_1 U_{[l, l+1]} \phi_2] \) is then captured by

\[
\bigvee_{i=1}^{K-1} \Box^{\sigma}[(a \land (\neg (\forall C) U c_i)) \leftrightarrow \phi_{C1,i} \lor \phi_{C2,i}]
\]

To eliminate each bounded interval modality as seen in (a),(b) above, we need an \( O(K) \) increase in size. Each temporal definition is replaced with a formula with of size \( O(K) \). Thus the size of the new formula is \( O(2^n) \times O(K) \times O(K) \), and the total number of propositions needed is \( 2^n \times \{c_0, \ldots, c_{K-1}\} \). Assuming binary encoding for \( K \), we get a \( \text{MITL} U_{[0, \infty]} S \) formulae whose size is exponential in \( n \) and \( \log K \). As the satisfiability checking for \( \text{MITL} U_{[0, \infty]} S \) is in \( \text{PSPACE} \), we get an \( \text{EXPSPACE} \) upper bound in \( n, \log K \). The \( \text{EXPSPACE} \) hardness of \( \text{MITL} \) can be found in \([7]\).

**E.2 Proof of Theorem 2.2** : \( \text{MITL} + \text{UM} \) is \( \text{EXPSPACE-complete} \)

Starting from an \( \text{MITL} + \text{UM} \) formula, we first show how to obtain an equivalent \( \text{MITL} \) formula modulo simple projections. The constants appearing in a \( \text{MITL} + \text{UM} \) formula come from those which are part of the time intervals \( I \) decorating the temporal modalities, as well as those from counting constraints \( k\%n \). If we consider some \( U \) modality, say \( U_{(1,n)}, \#b=\#k\%n \), then the number of bits needed to encode this modality is \( (\log l + \log g + \log k + \log n) = O(\log (u \times (n \times \log (n)))) \). Let \( n_{max} \) and \( u_{max} \) be the maximal constants appearing in the counting constraints as well as time intervals of a \( \text{MITL} + \text{UM} \) formula \( \phi \). Then \( |\phi| = (\log(n_{max}) \times \log(u_{max})) \times (\text{the number of temporal modalities in } \phi \text{ } \text{number of boolean connectives in } \phi) \).

**Elimination of UM**

In this section, we show how to eliminate \( \text{UM} \) from \( \text{MTL} + \text{UM} \) over strictly monotonic timed words. This can be extended to weakly monotonic timed words. Given any \( \text{MTL} + \text{UM} \) formula \( \varphi \) over \( \Sigma \), we first “flatten” the \( \text{UM} \) modalities of \( \varphi \) and obtain a flattened formula. **Example.** The formula \( \varphi = [a U (e \land (\# U_{(2,3), \#b=2\%5}))] \) can be flattened by replacing the \( \text{UM} \) with a fresh witness proposition \( w \) to obtain \( \varphi_{flat} = [a U (e \land w)] \land \Box^{\sigma} [w \leftrightarrow (f U_{(2,3), \#b=2\%5})] \).

Starting from \( \chi \in \text{MTL} + \text{UM} \), in the following, we now show how to obtain equisatisfiable \( \text{MTL} \) formulae corresponding to each temporal projection containing a \( \text{UM} \) modality.

1. **Flattening** : Flatten \( \chi \) obtaining \( \chi_{flat} \) over \( \Sigma \cup W \), where \( W \) is the set of witness propositions used, \( \Sigma \cap W = \emptyset \).

2. **Eliminate Counting** : Consider, one by one, each temporal definition \( T_i \) of \( \chi_{flat} \). Let \( \Sigma_i = \Sigma \cup W \cup X_i \), where \( X_i \) is a set of fresh propositions, \( X_i \cap X_j = \emptyset \) for \( i \neq j \).
3. **Putting it all together**: The formula $\zeta = \bigwedge_{i=1}^{k} \zeta_i \in \text{MTL}$ is such that it is equisatisfiable to $T_i$ modulo simple extensions.

For elimination of UM, marking witnesses correctly is ensured using an extra set of symbols $B = \{b_0, \ldots, b_n\}$ which act as counters incremented in a circular fashion. Each time a witness of the formula which is being counted is encountered, the counter increments, else it remains same. The evaluation of the mod counting formulae can be reduced to checking the difference between indices between the first and the last symbol in the time region where the counting constraint is checked.

### Construction of Simple Extension

Consider a temporal definition $T = \Box^{\text{ns}}[a \leftrightarrow x \text{UM}_{I, \#b=k\%n} y]$, built from $\Sigma \cup W$. Let $\oplus$ denote addition modulo $n + 1$.

1. **Construction of a $(\Sigma \cup W, B)$- simple extension.** We introduce a fresh set of propositions $B = \{b_0, b_1, \ldots, b_{n-1}\}$ and construct a family of simple extensions $\rho' = (\rho', \tau)$ from $\rho = (\sigma, \tau)$ as follows:
   - $C1$: $\sigma'_i = \sigma_i \cup \{b_i\}$. If $b_k \in \sigma'_i$ and if $b \in \sigma_{i+1}$, $\sigma'_{i+1} = \sigma_{i+1} \cup \{b_{k+1}\}$.
   - $C2$: If $b_k \in \sigma'_i$ and $b \notin \sigma_{i+1}$, then $\sigma'_{i+1} = \sigma_{i+1} \cup \{b_k\}$.
   - $C3$: $\sigma'_i$ has exactly one symbol from $B$ for all $1 \leq i \leq |\text{dom}(\rho)|$.

2. **Formula specifying the above behaviour.** The variables in $B$ help in counting the number of $b$'s in $\rho$. $C1, C2$ and $C3$ are written in MTL as follows:
   - $\delta_1 = \bigwedge_{k=0}^{n} \Box^{\text{ns}}[\Theta b \land b_k] \rightarrow \Theta b_{k+1}$ and
   - $\delta_2 = \bigwedge_{k=0}^{n} \Box^{\text{ns}}[\Theta \neg b \land b_k] \rightarrow \Theta b_k$
   - $\delta_3 = \bigwedge_{k=0}^{n} \Box^{\text{ns}}[b_k \rightarrow \bigwedge_{j \neq k} \neg b_j]$

3. **Lemma 13.** Consider a temporal definition $T = \Box^{\text{ns}}[a \leftrightarrow x \text{UM}_{I, \#b=k\%n} y]$, built from $\Sigma \cup W$. Then we synthesize a formula $\psi \in \text{MTL}$ over $\Sigma \cup W \cup X$ which is equivalent to $T$ modulo simple extensions.

**Proof.**

1. Construct a simple extension $\rho'$ as shown in section E.2

2. Now checking whether at point $i$ in $\rho$, $x \text{UM}_{I, \#b=k\%n} y$ is true, is equivalent to checking that at point $i$ in $\rho'$ there exist a point $j$ in the future where $y$ is true and for all the points between $j$ and $i$, $x$ is true and the difference between the index values of the symbols from $B$ at $i$ and $j$ is $k\%n$. $\phi_{\text{mark}, a} = \Box^{\text{ns}} \bigwedge_{i \in \{1, \ldots, n-1\}} (a \land b_i \leftrightarrow [x \cup I(y \land b_j)])$ where $j = k + i\%n$.

3. The formula $\delta_1 \land \delta_2 \land \delta_3 \land \phi_{\text{mark}, a}$ is equivalent to $T$ modulo simple projections.

Notice that in the above reduction, if we start with an MITL + UM formula, we will obtain an MITL formula since we do not introduce any new punctual intervals.

4. **Lemma 14.** Satisfiability of MITL + UM is EXPSPACE-complete.
Proof. Assume that we have a MITL + UM formula \( \phi \) with \(|\phi| = m \), and hence \( \leq m \) UM modalities. Let \( \phi \) be over the alphabet \( \Sigma \). The number of propositions used is hence \( 2^2 \), and let \( K \) be the maximal constant appearing in the intervals of \( \phi \). Let \( k_1\%n_1, \ldots, k_m\%n_m \) be the modulo counting entities in \( \phi \). Let \( n_{\text{max}} \) be the maximum of \( n_1, \ldots, n_m \). Going by the construction above, we obtain \( m \) temporal definitions \( T_1, \ldots, T_m \), corresponding to the \( m \) UM modalities.

To eliminate each \( T_i \), we introduce \( n_{\text{max}} \) formulae of the form \( \phi_{\text{mark},a} \), evaluated on timed words over \( 2^E \cup B_1 \cup \cdots \cup B_m \). This is enforced by \( \delta_1, \delta_2, \delta_3 \). The number of propositions in the obtained MITL formula is hence \( |2^E| |B_1 + B_2 + \cdots + B_m| \). The size of the new formula is \( O(m.n_{\text{max}}) \), while the maximum constant appearing in the intervals is same as \( K \). Thus we have an exponential size (the size now is \( O(m.2^{\log n_{\text{max}}}) \)) MITL formulae with max constant as \( K \). The EXPSPACE-hardness of MITL + UM follows from that of MITL. Lemma \( \square \) now shows that satisfiability checking for MITL + UM is EXPSPACE-complete. □

E.3 Proof of Theorem 2.3 : MITL + URat is in 2EXPSPACE

Proof. Consider a URat modality \( a\text{URat}_{T_0, \Sigma} \), where \( a \) is a rational expression over \( S \) and \( a, b \in \Sigma \). The size of a URat modality is \( \text{size}(re) + \log(l) + \log(u) \), where \( l, u \) are the lower and upper bounds of the interval \( I \), and \( \text{size}(re) \) is the size of the rational expression \( re \). We first flatten \( \varphi \) by introducing witness propositions for each URat modality obtaining temporal definitions of the form \( \square^{\varphi_0}(w \leftrightarrow \text{URat}_{T_0, \Sigma}(\psi)) \). Flattening only creates a linear blow up in formula size. Assume that \( \varphi \) is flattened. Let \( T_i = \square^{\varphi_i}(w \leftrightarrow a\text{URat}_{T_0, \Sigma}(\psi_i)) \) be a temporal definition, and let there be \( t \) temporal definitions. Let \( l_i, u_i \) be the bounds of the interval \( I_i \). The size of \( \varphi_i \), \( |\varphi_i| \) is then defined as \( O(\sum_{i=1}^t (n_i + \log(l_i) + \log(u_i))) \), where \( n_i \) is the size of \( re_i \). Let \( w \) be the maximum constant appearing in the intervals \( I_i \).

Let us consider a temporal definition \( T = \square^{\varphi_0}(w \leftrightarrow a\text{URat}_{T_0, \Sigma}(\psi)) \).

1. We look at the number of propositions needed in obtaining the equisatisfiable MTL formula.
   a. The size of the rational expression \( re \) in \( T \) is \( n \). The DFA accepting \( re \) has \( \leq 2^n \) states.
      The transitions of this DFA are over formulae from \( S \). Since we convert the formulae into ExNF, we also convert this DFA into one whose transitions are over \( 2^5 \). Hence, the number of transitions in the DFA is \( \leq 2^n \times 2^5 \). Let \( S' = 2^5 \).
   b. This DFA is simulated using the symbols Threads, Merge. There can be at most \( 2^n \) threads, and each thread be in one of the \( 2^n \) states. Thus, the number of propositions \( Th_i(q) \) is at most \( 2^n 2^n \). Given that there are \( t \) temporal definitions, we need \( t \times 2^{n^2} \) extra symbols.
   c. Each integer point in the timed word is marked with a symbol \( c_i \), \( 0 \leq i \leq u - 1 \) (see the proof of lemma \( \square \) in Appendix \( \square \)).
   d. The number of propositions merge\((i,j)\) is \( \leq 2^n \times 2^n \).
   e. Thus, the number of symbols needed is \( 2^{5^2} \times u \times t \times 2^n 2^n \times (2^n \times 2^n) \).

2. Next, we count the size of the formulae needed while constructing the equisatisfiable MTL formula.
   a. For each temporal definition, we define the formulae \( \text{Nxt}(Th_i(q_x)) \) for each thread \( Th_i \). The argument of \( \text{Nxt} \) can take at most \( 2^n \) possibilities (\( 2^n \) states of a DFA) on each of the \( 2^n \) threads. Thus, the total number of \( \text{Nxt}(Th(q)) \) formulae is \( 2^n \times 2^n = O(poly(2^n)) \). Note that each \( \text{Nxt} \) formulae simulates the transition function of the DFA. \( \text{Nxt}(Th_i(q')) \) is determined depending on the present state \( q \) of the thread \( Th_i \), and the formulae (in \( S' \)) that are true at the present point. Thus, the size
of each Nxt formula is $2^n \times 2^{[S]}$. Thus, the total size of all the Nxt formulae is $O(poly(2^n)) \times O(poly(2^n \times 2^{[S]})) = O(poly(2^{n+1}[S]))$.

b. Next, we look at formulae NextMerge(i, k). Note that both the arguments refer to threads, and hence can take at most $2^n$ values. Thus, the total number of formulae is $2^n \times 2^n = O(poly(2^n))$. Each NextMerge formula checks whether the states at the 2 threads $\text{Th}_i, \text{Th}_k$ are equal or not. Thus, the size of each formulae is $O(2^n)$. The total blow up due to NextMerge formulae is hence, $O(2^n \times 2^n \times 2^n) = O(poly(2^n))$.

c. Next, we look at formulae MergeseqPref(k_1). This formula states all the possible merges from the present point to the integer point within the interval $(l-1, l)$. There are at most $2^n$ merges possible, as the merge always happens from a higher indexed thread to a lower one. The number of merges is equal to the nesting depth of the formula MergeseqPref(k_1). Note that the nesting depth can be at most $2^n$. The number of propositions merge(i, j) is $2^n \times 2^n$. Let there be $k \leq 2^n$ merges until we see the integer point in $(l-1, l)$. At each of these $k$ merges, we have $2^n \times 2^n$ possibilities, the maximum possible number of propositions merge(i, j) $(i, j \leq 2^n)$. Hence, the number of possible merge sequences we can generate is $(2^n \times 2^n)^k \leq (2^n \times 2^n)^2^n$. There are $2^n$ possible values of $k$ and the possible number of disjunctions of the formulae is at most $(2^n \times 2^n)^2^n \times 2^n = O(poly(2^{2^n}(2^n)))$.

d. The counting for Mergeseq Suff(k_1, k) is symmetric.

Adding all the blow ups due to various formulae Nxt(Th(q)), NextMerge(i, k), MergeseqPref(k_1) and Mergeseq Suff(k_1, k), we see the number to be doubly exponential $O(poly(2^{2^n}(2^n)))$.

Thus, we obtain an MITL formula of doubly exponential size, with doubly exponential number of new propositions. By applying the reduction as in lemma [12] we will obtain a formula in MITL[$U_{0, \infty}, S$], which is still doubly exponential, and which preserves the max constant. The PSPACE procedure of MITL[$U_{0, \infty}, S$] thus ensures that we have a 2EXPSPACE procedure for satisfiability checking for MITL + URat. Arriving at a tighter complexity for this class is an interesting problem and is open.

\[ \]

E.4 Proof of Theorem 2.4: MITL + MC is F_{\omega_\omega}-hard

In this section, we discuss the complexity of MITL + MC, proving Theorem 2.4. To prove this, we obtain a reduction from the reachability problem of Insertion Channel Machines with Emptiness Testing (ICMET). We now show how to encode the reachability problem of ICMET in MITL + MC.

Recalling ICMET

A channel machine $\mathcal{A}$ consists of a tuple having a finite set of states $S$, a finite alphabet $M$ used to write on the channels, a finite set $C$ of channels, and a transition relation $\Delta \subseteq S \times Op \times S$ where $Op$ is a finite set of operations on the channels. These operations have the forms $c_0 a$, $c_1 a$ and $c = \epsilon$ which respectively write a message $a$ to the tail of channel $c$, read the first message $a$ from a channel $c$, and test if channel $c$ is empty.

A configuration of the channel machine $\mathcal{A}$ is a pair $(s, U)$ where $s$ is a state and $U$ is a tuple of length $|C|$ which describes the contents of all the $|C|$ channels. Each entry in this tuple is hence a string over the alphabet $M$. We use Conf to denote the configurations of the channel machine. The configurations are connected to each other depending on the operations performed. In particular,
(a) From a configuration \((q, U)\), the transition \((q, c!a, q')\) results in a configuration \((q', U')\) where \(U'\) is the \(|C|\)-tuple which does not alter the contents of channels other than \(c\), and appends \(a\) to channel \(c\).

(b) From a configuration \((q, U)\), the transition \((q, c?a, q')\) results in a configuration \((q', U')\) where \(U'\) is the \(|C|\)-tuple which does not alter the contents of channels other than \(c\), and reads \(a\) from the head of channel \(c\).

(c) From a configuration \((q, U)\), the transition \((q, c = \epsilon, q')\) results in the configuration \((q', U)\) if channel \(c\) is empty. The contents of all the channels are unaltered. If channel \(c\) is non-empty, then the machine is stuck.

If the only transitions allowed are as above, then we call \(\mathcal{A}\) an error-free channel-machine. We now look at channel machines with insertion errors. These allow extra transitions between configurations as follows.

(d) If a configuration \((q, U)\) can evolve into \((q', V)\) using one transition as above, then we allow any configuration \((q, U')\), where \(U'\) is a \(|C|\)-tuple of words obtained by deleting any number of letters from any word in \(U\), to evolve into \((q', V')\) where \(V'\) is obtained by adding any number of letters to any word in \(V\). Thus insertion errors are created by inserting arbitrarily many symbols into some word.

The channel machines as above are called ICMET. A run of an ICMET is a sequence of transitions \(\gamma_0 \xrightarrow{op_0} \gamma_1 \cdots \xrightarrow{op_{n-1}} \gamma_n\ldots\) that is consistent with the above operational semantics.

**Reduction from ICMET reachability to satisfiability of MITL + MC**

Consider any ICMET \(C = (S, M, \Delta, C)\), with set of states \(S = \{s_0, \ldots, s_n\}\) and channels \(C = \{c_1, \ldots, c_k\}\). Let \(M\) be a finite set of messages used for communication in the channels.

We encode the set of all possible configurations of \(C\), with a timed language over the alphabet \(\Sigma = M_a \cup M_b \cup \Delta \cup S \cup \{H\}\), where \(M_a = \{m_a | m \in M\}\), \(M_b = \{m_b | m \in M\}\), and \(H\) is a new symbol.

1. The \(j\)th configuration for \(j \geq 0\) is encoded in the interval \([(2k + 2)j, (2k + 2)(j + 1) - 1]\) where \(k\) refers to number of channels. The \(j\)th configuration begins at the time point \((2k + 2)j\). At a distance \([2i - 1, 2i]\) from this point, \(1 \leq i \leq k\), the contents of the \(i\)th channel are encoded as shown in the point 7. The intervals of the form \((2i, 2i + 1)\), \(0 \leq i \leq k + 1\) from the start of any configuration are time intervals within which no action takes place. The current state at the \(j\)th configuration is encoded at \((2k + 2)j\), and the transition that connects configurations \(j, j + 1\) is encoded at \((2k + 2)j + (2k + 1)\).

\[
\beta = (2k + 2)j
\]

\[\text{Figure 9 Illustrating the } j\text{th configuration, with the current state encoded at } (2k + 2)j, \text{ and transition between configurations } j, j + 1 \text{ encoded at } (2k + 2)j + (2k + 1), \text{ and the contents of channel } i \text{ encoded in the interval } (2k + 2)j + [2i - 1, 2i].\]

2. Let \(m_{b_i}\) be the message at the head of the channel \(i\). Each message \(m_i\) is encoded using
We introduce a special symbol $m_{i,a}$ and $m_{i,b}$. In our encoding of channel $i$, the first point marked $m_{b_{i,a}}$ in the interval $(2k+2)j+[2i-1,2i]$ is the head of the channel $i$ and denotes that $m_{b_{i}}$ is the message at the head of the channel. The last point marked $m_{i,b}$ in the interval is the tail of the channel, and denotes that message $m_{t_{i}}$ is the message stored at the tail of the channel.

**Figure 10** Illustrating the channel contents with each message $m_{i}$ encoded as $m_{i,a}m_{i,b}$. $H$ is a separator for the head of the channel.

3. Exactly at $2k+1$ time units after the start of the $j^{th}$ configuration, we encode the transition from the state at the $j^{th}$ configuration to the $(j+1)^{st}$ configuration (starts at $(2k+2)(j+1)$). Note that the transition has the form $(s,clm,s')$ or $(s,c?m,s')$ or $(s,c=e,s')$.

4. We introduce a special symbol $H$, which acts as separator between the head of the message and the remaining contents, for each channel.

5. A sequence of messages $w_1w_2w_3\ldots w_z$ in any channel is encoded as a sequence $w_{1,a}w_{1,b}Hw_{2,a}w_{2,b}w_{3,a}w_{3,b}\ldots w_{z,a}w_{z,b}$.

Let $S = \bigvee_{i=0}^{m} S_{i}$ denote the states of the ICME$T$, $\alpha = \bigvee_{i=0}^{m} \alpha_{i}$, denote the transitions $\alpha_{i}$ of the form $(s,clm,s')$ or $(s,c?m,s')$ or $(s,c=e,s')$. Let action $= \text{true}$ and let $M_a = \bigvee_{m_{x,a}\in M_{}} (m_{x,a})$, $M_b = \bigvee_{m_{x,b}\in M_{}} (m_{x,b})$, with $M = M_a \lor M_b$.

1. All the states must be at distance $2k+2$ from the previous state (first one being at 0) and all the propositions encoding transitions must be at the distance $2k+1$ from the start of the configuration.

   $$\varphi_S = s_0 \land \square[\varphi] \Rightarrow \{ \diamond\ Diamond_{0,2k+2}(\Diamond S) \land \square_{0,2k+2}(\Diamond \lnot S) \land \Diamond_{0,2k+1}(\Diamond \alpha \land \Diamond_{2k+1,2k+2}(\Diamond \lnot \alpha)) \}$$

2. All the messages are in the interval $[2i-1,2i]$ from the start of configuration. No action takes place at $(2i-2,2i-1)$ from the start of any configuration.

   $$\varphi_m = \square[\Diamond S \Rightarrow \bigwedge_{j=1}^{b} \square_{2j-1,2j} (M \lor H) \land \square_{2j-2,2j-1}(\lnot \text{action})]$$

3. Consecutive source and target states must be in accordance with a transition $\alpha$. For example, $s_j$ appears consecutively after $s_i$ reading $\alpha_i$ iff $\alpha_i$ is of the form $(s_i,y,s_j) \in \Delta$, with $y \in \{c_1,m,c_2\}$.

   $$\varphi_{\Delta} = \bigwedge_{s,s' \in S_{i}} \square[\Diamond (s \land \Diamond_{0,2k+2}(s') \Rightarrow (\Diamond_{0,2k+1}(\Diamond \Delta_{s,s'})) \text{ where } \Delta_{s,s'} \text{ are possible } \alpha_i \text{ between } s_i,s_j]$$

4. We introduce a special symbol $H$ along with other channel contents which acts as a separator between the head of the channel and rest of the contents. Thus $H$ has following properties.

   There is one and only one time-stamp in the interval $(2i-1,2i)$ from the start of the configuration where $H$ is true. The following formula says that there is an occurrence of a $H$:

   $$\varphi_{H_1} = \square[S \land \Diamond_{2i-1,2i}M \Rightarrow \bigwedge_{j=1}^{k} \Diamond_{2j-1,2j}(H)]$$

The following formula says that there can be only one $H$: $\varphi_{H_2} = \square[H \Rightarrow \Diamond_{0,1}(H)]$
Every message \( m_x \) is encoded by truth of proposition \( m_{x,a} \) immediately followed by \( m_{x,b} \). Thus for any message \( m_x \), the configuration encoding the channel contents has a sub-string of the form \( (m_{x,a}m_{x,b})^* \) where \( m_x \) is some message in \( M \).

\[
\varphi_m = \Box[m_{x,a} \Rightarrow \Diamond(0,1)M_{x,b} \land \Diamond(0,1)M_a \lor \Diamond(\Diamond(0,1)(\land H))] \land (\neg M_b \lor M_a)
\]

If the channel is not empty (there is at least one message \( m_{x,b} \) in the interval \( (2i-1,2i) \) corresponding to channel \( i \) contents) then there is one and only one \( m_{b} \) before \( H \). The following formula says that there can be at most one \( m_{b} \) before \( H \).

\[
\varphi_{H_b} = \Box[\neg(M_b \land \Diamond(0,1)(M_b \land \Diamond(0,1)(\land H)))]
\]

The following formula says that there is one \( M_b \) before \( H \) in the channel, if the channel is non-empty.

\[
\varphi_{H} = \Box[S \Rightarrow (M_{b,1} \land \Diamond(2j-1,2j)(M_b) \Rightarrow \Diamond(2j-1,2j)(M_b \land \Diamond(0,1)(\land H)))]
\]

Let \( \varphi_H = \varphi_{H_1} \land \varphi_{H_2} \land \varphi_{H_3} \land \varphi_{H_4} \).

5. Encoding transitions:

(a) We first define a macro for copying the contents of the \( i^{th} \) channel to the next configuration with insertion errors. If there were some \( m_{x,a}, m_{x,b} \) at times \( t, t' \), \( m_{x,b} \) is copied to \( t''+2k+2 \) (where \( t'' \in [t, t') \)), representing the channel contents in the next configuration. This is specified by means of an even count check. From any 3 consecutive points \( u, v, w \) such that \( m_{x,a} \) and \( m_{x,b} \) are true at \( v \) and \( w \), respectively, we assert that there are even (or odd) number of \( m_{x,b} \) within \((0, 2k+2)\) from both \( v \) and \( w \). This implies that there must be an odd number of \( m_{x,b} \)'s within time interval \([\tau_v+2k+2, \tau_w+2k+2] \). Thus, there must be at least one \( m_{x,b} \) copied from the point \( w \) to some point in the interval \([\tau_v+2k+2, \tau_w+2k+2] \). The rest of the even number of erroneous \( m_{x,b} \) in \([\tau_v+2k+2, \tau_w+2k+2] \), along with the arbitrary insertion errors within \([\tau_v+2k+2, \tau_w+2k+2] \) models the insertion error of the ICMET (see Figure 11). The formula copy is as follows.

\[
\Box[\text{copy}_{\tau}](m_{x,a} \land \text{seven}(0,2k+2)(m_{x,b})) \Rightarrow \Diamond(\text{seven}(0,2k+2)(m_{x,b})) \land \Box[\text{even}_{\tau}](m_{x,a} \land \text{seven}(0,2k+2)(m_{x,b})) \Rightarrow \Diamond(\text{seven}(0,2k+2)(m_{x,b}))]
\]
(b) If the transition is of the form $c_i = \epsilon$. The following formula checks that there are no events in the interval $(2i - 1, 2i)$ corresponding to channel $i$, while all the other channel contents are copied.

$$\varphi_{c_i = \epsilon} \equiv S \land \Box_{(2i-1,2i)}(\neg\text{action}) \land \bigwedge_{g=1}^{k} \text{copy}_g$$

(c) If the transition is of the form $c_i m_x$ where $m \in M$. An extra message is appended to the tail of channel $i$, and all the $m_a m_b$’s are copied to the next configuration. $M_b \land \Box_{(0,1)}(\neg M)$ denotes the last time point of channel $i$; if this occurs at time $t$, we know that this is copied at a timestamp strictly less than $2k + 2 + t$ (by 5(a)). Thus we assert that truth of $\Diamond (2k+2,2k+3) m_{x,b}$ at $t$.

$$\varphi_{c_i m_x} = S \land \bigwedge_{g=1}^{k} \text{copy}_g \land \Diamond_{(2i-1,2i)} \left\{ (M \land \Box_{(0,1)}(\neg M)) \Rightarrow (\Diamond (2k+2,2k+3) (m_{x,b}) \right\}$$

(d) If the transition is of the form $c_i ? m$ where $m \in M$. The contents of all channels other than $i$ are copied to the intervals encoding corresponding channel contents in the next configuration. We also check the existence of a first message in channel $i$; such a message has a $H$ at distance $(0,1)$ from it.

$$\varphi_{c_i ? m_x} = S \land \bigwedge_{j \neq i}^{k} \text{copy}_j \land \Diamond_{(2i-1,2i)} \left\{ m_{x,b} \land \Diamond_{(0,1)}(H) \right\} \land \bigwedge_{m_a \in M} \Box_{(2i-1,2i)} \left( m_{x,a} \land \neg \Diamond_{(0,1)}(H) \Rightarrow O(\neg \text{seven}(0,2k+2)(m_{x,b})) \right) \land \bigwedge_{m_a \in M} \Box_{(2i-1,2i)} \left( m_{x,a} \land \neg \Diamond_{(0,1)}(H) \Rightarrow O(\neg \text{seven}(0,2k+2)(m_{x,b})) \right)$$

6. Channel contents must change in accordance to the relevant transition. Let $L$ be a set of labels (names) for the transitions. Let $l \in L$ and $a_l$ be a transition labeled $l$.

$$\varphi_{C} = \Box[S \Rightarrow \bigwedge_{i \in L} \left( \Diamond_{(0,2k+1)}(\forall a_l \Rightarrow \varphi_i) \right)]$$

where $\varphi_i$ are the formulae as seen in 5 above ($\varphi_{c_i m_x}, \varphi_{c_i m_m}, \varphi_{c_i = \epsilon}$).

7. Let $s_t$ be a state of the ICMEt whose reachability we are interested in. We check $s_t$ is reachable from $s_0$ : $s_{\text{reach}} = \Diamond(s_t)$

Thus the formula encoding ICMEt is: $\varphi_S \land \varphi_\Delta \land \varphi_m \land \varphi_H \land \varphi_C \land \varphi_{\text{reach}}$

This is a formula in MITL + UM, and we have reduced the reachability problem of ICMEt with insertion errors to checking satisfiability of this formula.

F Non-punctual 1-TPTL is NPR

In this section, we show that non-punctuality does not provide any benefits in terms of complexity of satisfiability for TPTL as in the case of MITL. We show that satisfiability checking of non-punctual TPTL is itself non-primitive recursive. This highlights the importance of our oversampling reductions from RatMTL and RatMITL to MTL and MITL respectively, giving RegMITL an elementary complexity. It is easier to reduce RatMITL[URat] to 1-variable, non-punctual, TPTL without using oversampling, but this gives a non-primitive recursive bound on complexity. Our reduction of RatMITL[URat] to equisatisfiable MITL using oversampling, however has a 2EXPSPACE upperbound.

Non-punctual TPTL with 1 Variable (1 − OpTPTL)

We study a subclass of 1 − TPTL called open 1 − TPTL and denoted as 1 − OpTPTL. The restrictions are mainly on the form of the intervals used in comparing the clock $x$ as follows:

- Whenever the single clock $x$ lies in the scope of even number of negations, $x$ is compared only with open intervals, and
Whenever the single clock $x$ lies in the scope of an odd number of negations, $x$ is compared to a closed interval.

Note that this is a stricter restriction than non-punctuality as it can assert a property only within an open timed region. Our complexity result hence applies to TPTL with non-punctual intervals. Our hardness result uses a reduction from counter machines.

**Counter Machines**

A deterministic $k$-counter machine is a $k + 1$ tuple $\mathcal{M} = (P, C_1, \ldots, C_k)$, where

1. $C_1, \ldots, C_k$ are counters taking values in $\mathbb{N} \cup \{0\}$ (their initial values are set to zero);
2. $P$ is a finite set of instructions with labels $p_1, \ldots, p_{n-1}, p_n$. There is a unique instruction labelled HALT. For $E \in \{C_1, \ldots, C_k\}$, the instructions $P$ are of the following forms:
   a. $p_i$: $Inc(E)$, goto $p_j$,
   b. $p_i$: If $E = 0$, goto $p_j$, else go to $p_k$,
   c. $p_i$: $Dec(E)$, goto $p_j$,
   d. $p_n$: HALT.

A configuration $W = (i, c_1, \ldots, c_k)$ of $\mathcal{M}$ is given by the value of the current program counter $i$ and values $c_1, c_2, \ldots, c_k$ of the counters $C_1, C_2, \ldots, C_k$. A move of the counter machine $(l, c_1, c_2, \ldots, c_k) \rightarrow (l', c'_1, c'_2, \ldots, c'_k)$ denotes that configuration $(l', c'_1, c'_2, \ldots, c'_k)$ is obtained from $(l, c_1, c_2, \ldots, c_k)$ by executing the $l^\text{th}$ instruction $p_l$. If $p_l$ is an increment or decrement instruction, $c'_i = c_i + 1$ or $c_i - 1$, while $c_i = c_i$ for $i \neq l$ and $p'_l$ is the respective next instruction, while if $p_l$ is a zero check instruction, then $c'_i = c_i$ for all $i$, and $p'_l = p_j$ if $c_l = 0$ and $p_k$ otherwise.

**Incremental Error Counter Machine (IECM)**

An incremental error counter machine (IECM) is a counter machine where a particular configuration can have counter values with arbitrary positive error. Formally, an incremental error $k$-counter machine is a $k + 1$ tuple $\mathcal{M} = (P, C_1, \ldots, C_k)$ where $P$ is a set of instructions like above and $C_1$ to $C_k$ are the counters. The difference between a counter machine with and without incremental counter error is as follows:

1. Let $(l, c_1, c_2, \ldots, c_k) \rightarrow (l', c'_1, c'_2, \ldots, c'_k)$ be a move of a counter machine without error when executing $l^\text{th}$ instruction.
2. The corresponding move in the increment error counter machine is

   $$\{(l, c_1, c_2, \ldots, c_k) \rightarrow \{(l', c''_1, c''_2, \ldots, c''_k)|c''_i \geq c'_i, 1 \leq i \leq k\}\}$$

Thus the value of the counters are non deterministic. We use these machines for proving lower bound complexity in section F.1.

**Theorem 15.** [[14]] The halting problem for deterministic $k$ counter machines is undecidable for $k \geq 2$.

**Theorem 16.** [[2]] The halting problem for incremental error $k$-counter machines is non primitive recursive for $k \geq 5$.

**F.1 Satisfiability Checking for $1 – OpTPTL$**

**Theorem 17.** Satisfiability Checking of $1 – OpTPTL[\emptyset, \emptyset]$ is decidable with non primitive recursive lower bound over finite timed words.
Proof. We encode the runs of $k$ counter incremental error counter machine using $1 - \text{OpTPTL}$ formulae with $\emptyset, \text{O}$ modalities. We will encode a particular computation of any counter machine using timed words. The main idea is to construct a $1 - \text{OpTPTL}[\emptyset, \text{O}]$ formula $\varphi_{\text{IECM}}$ for any given $k$-incremental counter machine $\text{IECM}$ such that $\varphi_{\text{IECM}}$ is satisfied by only those timed words that encode the halting computation of $\text{IECM}$. Moreover, for every halting computation $C$ of the $\text{IECM}$, at least one timed word $\rho_C$ encodes $C$ and satisfies $\varphi_{\text{IECM}}$.

We encode each computation of a $k$-incremental counter machine $(P, \Sigma)$ where $P = \{p_1, \ldots, p_n\}$ is the set of instructions and $\Sigma = \{c_1, \ldots, c_k\}$ is the set of counters using timed words over the alphabet $\Sigma_{\text{IECM}} = \bigcup_{j \in \{1, \ldots, k\}} (\{S \cup F \cup \{a_j, b_j\}\})$ where $S = \{s^p|p \in 1, \ldots, n\}$ and $F = \{f^p|p \in 1, \ldots, n\}$ as follows: The $i$th configuration, $(p, c_1, \ldots, c_k)$ is encoded in the timed region $[i, i + 1)$ with the sequence $s^p(a_1b_1)^{c_1}(a_2b_2)^{c_2} \ldots (a_kb_k)^{c_k} f^p$.

![Figure 12](image.png)

Assume there are 3 counters, and that the $i$th configuration is $(p, 2, 0, 1)$. Let the instruction $p$ increment counter 2 and go to instruction $q$. Then the $i+1$st configuration is $(q, 2, 1, 1)$. Note that the $i$th configuration is encoded between integer points $i, i + 1$, while configuration $i + 1$ is encoded between integer points $i + 1, i + 2$.

The concatenation of these time segments of a timed word encodes the whole computation. Untiming our language yields the language

$$(S(a_1b_1)^{(a_2b_2)^{(a_kb_k)^*}})^*$$

where $S = \bigvee_{p \in \{1, 2, \ldots, n\}} s^p$ and $F = \bigvee_{p \in \{1, 2, \ldots, n\}} f^p$.

To construct a formula $\varphi_{\text{IECM}}$, the main challenge is to propagate the behaviour from the time segment $[i, i + 1)$ to the time segment $[i + 1, i + 2)$ such that the latter encodes the $i + 1$th configuration of the $\text{IECM}$ in accordance with the counter values of the $i$th configuration. The usual idea is to copy all the $a$’s from one configuration to another using punctuality. This is not possible in a non-punctual logic. We preserve the number of $a$s and $b$s using the following idea:

- Given any non last $(a_j, t)(b_j, t')$ before $F$ (for some counter $c_j$), of a timed word encoding a computation. We assert that the last symbol in $(t, t + 1)$ is $a_j$ and the last symbol in $(t', t' + 1)$ is $b_j$.
- We can easily assert that the untimed sequence of the timed word is of the form

$$(S(a_1b_1)^{(a_2b_2)^{(a_kb_k)^*}})^*$$

The above two conditions imply that there is at least one $a_j$ within time $(t + 1, t' + 1)$. Thus, all the non last $a_j, b_j$ are copied to the segment encoding next configuration. Now appending one $a_jb_j$, two $a_jb_j$’s or no $a_jb_j$’s depends on whether the instruction was copy, increment or decrement operation.

$\varphi_{\text{IECM}}$ is obtained as a conjunction of several formulae. Let $S, F$ be a shorthand for $\bigwedge_{p \in \{1, \ldots, n\}} s^p$ and $\bigwedge_{p \in \{1, \ldots, n\}} f^p$, respectively. We also define macros $A_j = \bigvee_{w \geq j} a_w$ and $A_{k+1} = \bot$.

We now give formula for encoding the machine. Let $C = \{1, \ldots, k\}$ and $P = \{1, \ldots, n\}$ be the indices of the counters and the instructions.
Expressing untimed sequence: The words should be of the form

\((S(a_1b_1)^*(a_2b_2)^* \ldots (a_kb_k)^*F)^*\)

This could be expressed in the formula below

\[\varphi_1 = \bigwedge_{j \in C, p \in P} \square^{ns}[s^p \to O(A_1 \lor f^p)] \land \square^{na}[a_j \to O(b_j)] \land \square^{ns}[b_j \to O(A_{j+1} \lor f^p)] \land \square^{na}[f^p \to O(S \lor \square^{na}(false))]\]

Initial Configuration: There is no occurrence of \(a_jb_j\) within \([0, 1]\). The program counter value is 1.

\[\varphi_2 = x.\{s^1 \land O(f^1 \land x \in (0, 1))\}\]

Copying \(S, F\): Any \((S, u)\) (read as any symbol from \(S\) at time stamp \(u\)) \((F, v)\) (read as (read as any symbol from \(F\) at time stamp \(v\)) has a next occurrence \((S, u')\), \((F, v')\) in the future such that \(u' - u \in (k, k + 1)\) and \(v' - v \in (k - 1, k)\). Note that this condition along with \(\varphi_1\) and \(\varphi_2\) makes sure that \(S\) and \(F\) occur only within the intervals of the form \([i, i + 1]\) where \(i\) is the configuration number. Recall that \(s^n, f^n\) represents the last instruction (HALT).

\[\varphi_3 = \square^{ns}x.\{(S \land \neg s^n) \to \neg \diamond(x \in [0, 1] \land S) \land \diamond(S \land x \in (1, 2))\} \land \square^{ns}x.\{(F \land \neg f^n) \to \diamond(F \land x \in (0, 1))\}\]

Note that the above formula ensures that subsequent configurations are encoded in smaller and smaller regions within their respective unit intervals, since consecutive symbols from \(S\) grow apart from each other (a distance > 1), while consecutive symbols from \(F\) grow closer to each other (a distance < 1). See Figure 13

![Figure 13 Subsequent configurations in subsequent unit intervals grow closer and closer.](image)

Beyond \(p_n=\text{HALT}\), there are no instructions

\[\varphi_4 = \square^{ns}[f^n \to \square(false)]\]

At any point of time, exactly one event takes place. Events have distinct time stamps.

\[\varphi_6 = [\bigwedge_{y \in \Sigma_{\text{ecm}}} \square^{na}[y \to \neg (\bigwedge_{x \in \Sigma_{\text{ecm}} \setminus \{y\}} (x))] \land \square^{na}[\square(false) \lor O(x \in (0, \infty))]]\]

Eventually we reach the halting configuration \((p_n, c_1, \ldots, c_k)\): \(\varphi_6 = \diamond^{na}s^n\)

Every non last \((a_j, t)(b_j, t')\) occurring in the interval \((i, i + 1)\) should be copied in the interval \((i + 1, i + 2)\). We specify this condition as follows:

state that from every non last \(a_j\) the last symbol within \((0, 1)\) is \(a_j\). Similarly from every non last \(b_j\), the last symbol within \((0, 1)\) is \(b_j\). Thus \((a_j, t)(b_j, t')\) will have a \((b_j, t' + 1 - \epsilon)\) where \(\epsilon \in (0, t' - t)\).
Figure 14 Consider a $a_j b_j$ where $a_j$ is at time $t$ and $b_j$ is at time $t'$. There are further $a, b$ symbols in the unit interval, like as shown above $a_{j+3} b_{j+3}$ occur after $a_j b_j$ in the same unit interval. Then the $a_j, b_j$ are copied such that the last symbol in the interval $(t, t + 1)$ is an $a_j$ and the last symbol in the interval $(t', t' + 1)$ is a $b_j$. There are no points between the $a_j$ in $(i + 1, i + 2)$ and the time stamp $t + 1$ as shown above. Likewise, there are no points between the $b_j$ in $(i + 1, i + 2)$ and the time stamp $t' + 1$ as shown above. Note that the time stamp of the copied $b_j$ in $(i + 1, i + 2)$ lies in the interval $(t + 1, t' + 1)$.

Thus all the non last $a_j b_j$ will incur a $b_j$ in the next configuration. $\varphi_1$ makes sure that there is an $a_j$ between two $b_j$'s. Thus this condition along with $\varphi_1$ makes sure that the non last $a_j b_j$ sequence is conserved. Note that there can be some $a_j b_j$ which are arbitrarily inserted. These insertion errors model the incremental error of the machine. Any such inserted $(a_j, t_{ins})(b_j, t'_{ins})$ in $(i + 1, i + 2)$ is such that there is a $(a_j, t)(b_j, t')$ in $(i, i + 1)$ with $t_{ins} \in (t + 1, t' + 1)$. Just for the sake of simplicity we assume $a_{k+1} = false$.

Let $nl(a_j) = a_j \land \neg last(a_j)$, $nl(b_j) = b_j \land \neg last(b_j)$, $\psi_{nh} = \neg \diamond (f^n \land x \in [0, 1])$, $last(a_j) = a_j \land O(\mathcal{O}(F \lor A_{j+1}))$ and $last(b_j) = b_j \land O(\mathcal{O}(F \lor A_{j+1}))$.

$\varphi_2 = \bigwedge_{j \in C} \mathcal{D}^n x. [nl(a_j) \land \psi_{nh} \rightarrow \diamond (a_j \land x \in (0, 1) \land O(x \in (1, 2)])] \land \mathcal{D}^n x. [nl(b_j) \land \psi_{nh} \rightarrow \diamond (b_j \land x \in (0, 1) \land O(x \in (1, 2)])]$.

We define a short macro $Copy_{\mathcal{W}}$: Copies the content of all the intervals encoding counter values except counters in $W$. Just for the sake of simplicity we denote

$Copy_{\mathcal{W}} = \bigwedge_{j \in C \setminus W} \mathcal{D}^n x. \{last(a_j) \rightarrow (a_j \land x \in (0, 1) \land O(b_j \land x \in (1, 2) \land O(F)))\}$

Using this macro we define the increment, decrement and jump operation.

1. Consider the zero check instruction $p_2$: If $C_j = 0$ goto $p_h$, else goto $p_d$. $\delta_1$ specifies the next configuration when the check for zero succeeds. $\delta_2$ specifies the else condition.

$\varphi_3^{j=0} = Copy_{\mathcal{W}} \land \delta_1 \land \delta_2$

$\delta_1 = \mathcal{D}^n[\{s^g \land ((\neg a_j) \cup F)\} \rightarrow (\neg \mathcal{S}) \cup s^h]$

$\delta_2 = \mathcal{D}^n[\{s^g \land ((\neg a_j) \cup a_j)\} \rightarrow (\neg \mathcal{S}) \cup s^h].$

2. $p_2$: $inc(C_j)$ goto $p_h$. The increment is modelled by appending exactly one $a_j b_j$ in the next interval just after the last copied $a_j b_j$

$\varphi_3^{inc j} = Copy_{\mathcal{W}} \land \mathcal{D}^n s^g \rightarrow (\neg \mathcal{S}) \cup s^h \land \psi_0^{inc} \land \psi_1^{inc}$

- The formula $\psi_0^{inc} = \mathcal{D}^n[(s^g \land (\neg a_j) \cup F)] \rightarrow (\neg \mathcal{S}) \cup s^h \land (\neg \mathcal{x} \land \mathcal{O}(last(a_j) \land x \in (0, 1) \land (a_j \land O(last(a_j) \land x \in (1, 2))))])$ specifies the increment of the counter $j$ when the value of $j$ is zero.

- The formula $\psi_1^{inc} = \mathcal{D}^n[(s^g \land (\neg F) \cup a_j)] \rightarrow (\neg \mathcal{F}) \cup s^h \land (\neg \mathcal{x} \land \mathcal{O}(last(a_j) \land x \in (0, 1) \land (a_j \land O(last(a_j) \land x \in (1, 2))))])$ specifies the increment of counter $j$ when $j$ value is non zero by appending exactly one pair of $a_j b_j$ after the last copied $a_j b_j$ in the next interval.
3. \( p_q^i \): Dec \((C_j)\) goto \( p_n \). Let \( \text{second} - \text{last}(a_j) = a_j \land \mathcal{O}(\text{last}(a_j)) \). Decrement is modelled by avoiding copy of last \( a_j b_j \) in the next interval.

\[
\varphi_{8,\text{dec}}^q = \text{Copy}_{C_j} \land \Box^p s^g \rightarrow (\neg S) \cup s^k \land \psi_8^\text{dec} \land \psi_8^\text{dec}.
\]

- The formula \( \psi_8^\text{dec} = \Box^p s^g \rightarrow (\neg S) \cup s^k \land ((\neg \text{a_j}) \cup (\text{a_j} \cup \mathcal{O}(\text{last}(a_j)))) \) specifies that the counter remains unchanged if decrement is applied to the \( j \) when it is zero.

- The formula \( \psi_8^\text{dec} = \Box^p s^g \rightarrow (\neg S) \cup s^k \land ((\neg \text{a_j}) \cup (\text{a_j} \cup \mathcal{O}(\text{last}(a_j)))) \) decrements the counter \( j \), if the present value of \( j \) is non zero. It does that by disallowing copy of last \( a_j b_j \) of the present interval to the next.

The formula \( \varphi_{\text{IECM}} = \bigwedge_{i \in \{1, \ldots, 7\}} \varphi_i \land \bigwedge_{p \in P} \varphi_{8, p} \).

### G Details on Expressiveness

- **Theorem 18.** 1. MTL + URat \( \subseteq \) MTL + Rat
2. MTL + UM \( \subseteq \) MTL + MC

**Proof.** 1. We first prove MTL + URat \( \subseteq \) MTL + Rat.

Note that \( \phi_1 \text{URat}_{i, \text{re}'} \phi_2 \) is equivalent to \( \text{trueURat}_{i, \text{re}'} \phi_2 \), where \( \text{re}' \) is a regular expression obtained by conjuncting \( \phi_1 \) to all formulae \( \psi \) occurring in the top level subformulae of \( \text{re} \), and \( S' = S \cup \{ \phi_1 \} \). For example, if we had \( \text{aURat}_{i,1} \), then we obtain \( \text{trueURat}_{i,1} \). When evaluated at a point \( i \), the conjunction ensures that \( \phi_1 \) holds good at all the points between \( i \) and \( j \), where \( \tau_j - \tau_i \in I \). To reduce \text{trueURat}_{i, \text{re}'} \phi_2 \) to a Rat formula, we need the following lemma.

- **Lemma 19.** Given any regular expression \( \text{re} \), there exist finitely many regular expressions \( R_1, R_2, \ldots, R_n, R_m \) such that \( R = \bigcup_{i=1}^n R_1, R_2 \). That is, for any string \( \sigma \in R \) and for any decomposition of \( \sigma \) as \( \sigma_1 \sigma_2 \), there exists some \( i \leq n \) such that \( \sigma_1 \sigma_2 \in R_i \).

**Proof.** Let \( \mathcal{A} \) be the minimal DFA for \( R \). Let the number of states in \( \mathcal{A} \) be \( n \). The set of strings that leads to some state \( q_i \) from the initial state \( q_0 \) is definable by a regular expression \( R_1 \). Likewise, the set of strings that lead from \( q_i \) to some final state of \( \mathcal{A} \) is also definable by some regular expression \( R_2 \). Given that there are \( n \) states in the DFA \( \mathcal{A} \), we have \( L(\mathcal{A}) = \bigcup_{i=1}^n R_1, R_2 \). Consider any string \( \sigma \in L(\mathcal{A}) \), and any arbitrary decomposition of \( \sigma \) as \( \sigma_1 \sigma_2 \). If we run the word \( \sigma_1 \) over \( \mathcal{A} \), we might reach at some state \( q_i \). Thus \( \sigma_1 \in L(R_i) \). If we read \( \sigma_2 \) from \( q_i \), it should lead us to one of the final states (by assumption that \( \sigma \in R \) ). Thus \( \sigma_2 \in L(R_j) \).

Let’s consider \( \text{trueURat}_{i, \text{re}'} \phi_2 \) when \( I = [l, u] \). If \( \text{trueURat}_{i, \text{re}'} \phi_2 \) evaluates to true at a point \( i \), we know that \( \phi_2 \) holds good at some point \( j \) such that \( \tau_j - \tau_i \in [l, u] \), and that \( \text{Seg}(S, i, j) \cap L(\text{re}') \neq \emptyset \). By the above lemma, for any word \( \sigma \in L(\text{re}') \), and any decomposition \( \sigma = \sigma_1 \sigma_2 \), there exist \( i \in \{1, 2, \ldots, n\} \) such that \( \sigma_1 \in L(R_i) \) and \( \sigma_2 \in L(R_j) \). Thus we decompose at a point \( j' \) with every possible \( R_i, R_j \) pair such that

\[1\text{ If } I = [l, l], \text{ then } \text{trueURat}_{i, \text{re}'} \phi_2 = \text{Rat}_{[0, l]} \text{re}' \phi_2 \]
We first show that the $\phi$ formula

$\text{In this section, we explain the algorithm which converts a $\psi$ equivalent form as (i) $C_1 \lor C_2$ or (ii) $C_1$ or (iii) $C_2$ where}

- $C_1$ has the form $s \land \varphi_1$, where $\varphi_1 \in \Phi(\downarrow s \cup \{a\} \cup X)$,
- $C_2$ has the form $\varphi_2$, where $\varphi_2 \in \Phi(\downarrow s \cup \{a\} \cup X)$

2. We first show that the UM modality can be captured by MC. Consider any formula $\phi_1 \cup \phi_2$. At any point $i$ this formula is true if and only if there exists a point $j$ in future such that $\tau_j - \tau_i \in I$ and the number of points between $i$ and $j$ where $\phi_3$ is true at all points between $i$ and $j$. To count between $i$ and $j$, we can first count the behaviour $\phi_3$ from $i$ to the last point of the word, followed by the counting from $j$ to the last point of the word. Then we check that the difference between these counts to be $k \% n$.

Let $\tau_{cnt}(x, \phi_3) = \{\phi \land \text{MC}_{x \% n}(\varphi_3)\}$. Using this macro, $\phi_1 \cup \phi_2$ is equivalent to $\bigvee_{i=0}^{n-1} [\psi_1 \lor \psi_2]$ where

- $\psi_1 = \{\tau_{cnt}(1, \phi_1) \land (\phi_1 \cup \tau_{cnt}(2, \phi_2))\}$,
- $\psi_2 = \{\tau_{cnt}(1, \phi_3) \land (\phi_1 \cup \tau_{cnt}(2, \phi_2))\}$,
- $k_1 - k_2 = k$

$$\#\phi_3 = (M(n) + k_1) \quad \#\phi_3 = (M(n) + k_2)$$

**Figure 15** The case of $\psi_1$

$$\#\phi_3 = (M(n) + (k_2 - 1)) \quad \#\phi_3 = (M(n) + k_2)$$

**Figure 16** The case of $\psi_2$

The only difference between $\psi_1, \psi_2$ is that in one, $\phi_3$ holds at position $j$, while in the other, it does not. The $k_2 - 1$ is to avoid the double counting in the case $\phi_3$ holds at $j$.

H po-1-clock ATA to 1-TPTL

In this section, we explain the algorithm which converts a po-1-clock ATA $\mathcal{A}$ into a 1-TPTL formula $\varphi$ such that $L(\mathcal{A}) = L(\varphi)$.

1. **Step 1.** Rewrite the transitions of the automaton. Each $\delta(s, a)$ can be written in an equivalent form as (i) $C_1 \lor C_2$ or (ii) $C_1$ or (iii) $C_2$ where

- $C_1$ has the form $s \land \varphi_1$, where $\varphi_1 \in \Phi(\downarrow s \cup \{a\} \cup X)$,
- $C_2$ has the form $\varphi_2$, where $\varphi_2 \in \Phi(\downarrow s \cup \{a\} \cup X)$
In particular, if s is the lowest location in the partial order, then \( \varphi_1, \varphi_2 \in \Phi(\{a\} \cup X) \). Denote this equivalent form by \( \delta'(s, a) \).

For the example above, we obtain \( \delta'(s_0, a) = (s_0 \land (a \land x . s_a)) \lor (a \land s_t), \delta'(s_0, b) = s_0 \land b, \delta'(s_0, a) = (s_0 \land x < 1) \lor (x > 1) \delta'(s_2) = (s_t \land b) \).

2. **Step 2.** For each location \( s \), construct \( \Delta(s) \) which combines \( \delta'(s, a) \) for all \( a \in \Sigma \), by disjuncting them first, and again putting them in the form in step 1. Thus, we obtain \( \Delta(s) = \bigvee_a \delta'(s, a) \) which can be written as a disjunction \( D_1 \lor D_2 \) or simply \( D_1 \) or simply \( D_2 \) where \( D_1, D_2 \) have the forms \( s \land \varphi_1 \) and \( \varphi_2 \) respectively, where \( \varphi_1, \varphi_2 \in \Phi(\downarrow s \cup \Sigma \cup X) \).

For the example above, we obtain \( \Delta(s_0) = (s_0 \land [(a \land x . s_a)) \lor b]) \lor (a \land s_t), \Delta(s_1) = (s_a \land x < 1) \lor (x > 1) \Delta(s_1) = s_t \land b \).

3. **Step 3.** We now convert each \( \Delta(s) \) into a normal form \( N(s) \). \( N(s) \) is obtained from \( \Delta(s) \) as follows.

- If \( s \) occurs in \( \Delta(s) \), replace it with \( Os \).
- Replace each \( s' \) occurring in each \( \Phi_i(\downarrow s) \) with \( Os' \).

Let \( N(s) = N_1 \lor N_2 \), where \( N_1, N_2 \) are normal forms. Intuitively, the states appearing on the right side of each transition are those which are taken up in the next step. The normal form explicitly does this, and takes us a step closer to 1–TPTL.

Continuing with the example, we obtain \( N(s_0) = (O s_0 \land [(a \land x . O s_a)) \lor b]) \lor (a \land O s_t) \), \( N(s_1) = (O s_a \land x < 1) \lor (x > 1) N(s_1) = O s_t \land b \).

4. **Step 4.**

- Start with the state \( s_n \) which is the lowest in the partial order.

Let \( N(s_n) = (O s_n \land \varphi_1) \lor \varphi_2 \), where \( \varphi_1, \varphi_2 \in \Phi(\Sigma, X) \). Solving \( N(s_n) \), one obtains the solution \( \text{Beh}(s_n) = \varphi_1 W \varphi_2 \) if \( s_n \) is an accepting location, and as \( \varphi_1 \lor \varphi_2 \) if \( s_n \) is non-accepting. Intuitively, \( \text{Beh}(s_n) \) is the behaviour of \( s_n \); that is, it describes the timed words that are accepted when we start in \( s_n \). In the running example, we obtain \( \text{Beh}(s_t) = b W \bot = \Box^a b \) and \( \text{Beh}(s_0) = (x < 1) \lor \Box^a x > 1 \).

- Consider now some \( N(s_i) = (O s_i \land \varphi_1) \lor \varphi_2 \). First replace each \( s' \) in \( \varphi_1 \) with \( \text{Beh}(s') \).

\( \text{Beh}(s_i) \) is then obtained as \( \text{Beh}(\varphi_1) W \text{Beh}(\varphi_2) \) if \( s_i \) is an accepting location, and as \( \text{Beh}(\varphi_1) \lor \text{Beh}(\varphi_2) \) if \( s_i \) is non-accepting.

Substituting \( \text{Beh}(s_n) \) and \( \text{Beh}(s_t) \) in \( N(s_0) \), we obtain \( (O s_0 \land [(a \land x . O s_a)) \lor b]) \lor (a \land O \text{Beh}(s_t)) \), solving which, we get \( \text{Beh}(s_0) = [(a \land x . O \text{Beh}(s_a)) \lor b] W (a \land O \text{Beh}(s_t)) \).

Thus, \( \text{Beh}(s_0) \) which represents all timed words which are accepted when we start at \( s_0 \) is given by \( ((a \land x . O ((x < 1) \land x > 1)) \lor b) W (a \land O \Box^a b) \). The 1–TPTL formula equivalent to \( L(A) \) is then given by \( \text{Beh}(s_0) \).

**H.1 Correctness of Construction**

The above algorithm is correct; that is, the 1–TPTL formula \( \text{Beh}(s_0) \) indeed captures the language accepted by the 1-clock ATA.

For the proof of correctness, we define a 1-clock ATA with a TPTL look ahead. That is, \( \delta : S \times X \rightarrow \Phi(S \cup X \cup \chi(\Sigma \cup \{x\})) \), where \( \chi(\Sigma \cup \{x\}) \) is a TPTL formula over alphabet \( \Sigma \) and clock variable \( x \). We allow open TPTL formulae for look ahead; that is, one which is not of the form \( x . \varphi \). All the freeze quantifications \( x \) lie within \( \varphi \). The extension now allows to take a transition \( (s, \nu) \rightarrow [\kappa \land \psi(x)] \), where \( \psi(x) \) is a TPTL formula, if and only if the suffix of the input word with value of \( x \) being \( \nu \) satisfies \( \psi(x) \). We induct on the level of the partial order on the states.

Base Case: Let the level of the partial order be zero. Consider 1-clock ATA having only one location \( s_0 \). Let the transition function be \( \delta(s_0, a) = B_a(\psi_a(x), X, s_0) \) for every
Given $a \in \Sigma$. By our construction, we reduce $s_0$ into $\Delta(s_0) = \bigvee_{a \in \Sigma} [B_a(\psi_a(x), X, O(s_0))]$. Let $\Delta(s_0) = \bigvee (P_1 \land \psi_1(x) \land X_1 \land O(s_0)) \lor \bigvee (Q_j \land \psi_j(x) \land X_j)$. $\delta(s_0, a) = s_0 \land X_1 \land \psi_1(x)$ specifies that the clock constraints $X_1$ are satisfied and the suffix satisfies the formula $\psi_1(x)$ on reading an $a$. Thus for this $\delta(s_0, a)$, we have $O(s_0) \land X_1 \land \psi_1(x) \land a$ as a corresponding disjunct in $\Delta$ which specifies the same constraints on the word from the current point onwards. Thus the solution to the above will be satisfied at a point with some $x = \nu$ if and only if there is an accepting run from $s_0$ to the final configuration with $x = \nu$.

If the $s_0$ is a final location, the solution to this is $\varphi = \bigvee (P_1 \land \psi_1(x) \land X_1 \land O(s_0)) \lor \bigvee (Q_j \land \psi_j(x) \land X_j)$. If it is non-final, then it would be $U$ instead of $W$. Note that this implies that whenever $s_0$ is invoked with value of $x$ being $\nu$, the above formula would be true with $x = \nu$ thus getting an equivalent $1$-TPTL formulae.

Assume that for automata with $n-1$ levels in the partial order, we can construct an equivalent $1$-TPTL formula as per our construction. Consider an automaton with $n$ levels. Consider all the locations at the lowest level (that is, those location that can only call itself), $s_0, \ldots, s_k$. Apply the same construction. As explained above, the constructed formulae, while eliminating a location will be true at a point if and only if there is an accepting run starting from the corresponding location with the same clock value. Let the formula obtained for any $s_i$ be $\varphi_i$.

The occurrence of an $s_i$ in any $\Delta(s_{i<n})$ can be substituted with $\varphi_i$ as a look ahead. This gives us an $n-1$ level 1-clock $\mathsf{ATA}$ with TPTL look ahead. By induction, we obtain that every 1-clock $\mathsf{po-ATA}$ can be reduced to $1$-TPTL formulae.

## Proof of Lemma 8

**Proof.** Let $\rho$ be a timed word such that $\rho, i \models \mathsf{Rat}_\mathsf{re}$. $\mathsf{re}$ can be either a simple star-free expression over $\Sigma$, or can be of the form $\mathsf{Rat}_\mathsf{re}'$ or $\mathsf{Rat}_\mathsf{re}_1 + \mathsf{Rat}_\mathsf{re}_2$ or $\mathsf{Rat}_\mathsf{re}_1 \mathsf{Rat}_\mathsf{re}_2$ or $\mathsf{Rat}_\mathsf{re}_1^*$. Recursively, each of $\mathsf{re}', \mathsf{re}_1, \mathsf{re}_2$ also can be expanded out as above. The idea of the proof is to eliminate “all levels” of $\mathsf{Rat}_\mathsf{f}$ starting from the inner most one, by replacing them with $1$-TPTL formulae using structural induction. We first explain the proof in the case of $\mathsf{Rat}_\mathsf{f}\mathsf{re}$, where $\mathsf{re}$ is a star-free expression over $\Sigma$, and then look at general cases.

We first consider the case when all intervals are bounded.

1. Given $\mathsf{Rat}_\mathsf{f}\mathsf{re}$, and a point $i$ in a word $\rho$, $\mathsf{Rat}_\mathsf{f}$ checks $\mathsf{re}$ at all points $j$ in $\rho$ such that $\tau_j - \tau_i \in I$. We first eliminate the interval $I$ from $\mathsf{Rat}_\mathsf{f}\mathsf{re}$ by imagining a witness variable $w_I$ that evaluates to true at all points $j$ of $\rho$ such that $\tau_j - \tau_i \in I$. $w_I$ is used to cover all points distant $I$ from $i$.

2. We eliminate the interval $I$ in $\mathsf{Rat}_\mathsf{f}$ by rewriting $\mathsf{Rat}_\mathsf{f}\mathsf{re}$ as $\mathsf{Rat}((-w_I)^* \mathsf{re} \cdot (-w_I))$, which, when asserted at a point $i$, checks the truth of the expression $(-w_I)^* \mathsf{re} \cdot (-w_I)$ in the suffix from $i$. If $w_I$ indeed captures $\tau_j - \tau_i \in I$, then indeed we are checking $\mathsf{re}$ in the interval $I$. Let $\varphi_{\mathsf{re}}$ be an LTL formula that is equivalent to $\mathsf{re}$. This is possible since $L(\mathsf{re})$ is a star-free language.

3. We next replace $w_I$ by using a freeze clock variable $x$ which checks $x \in I$ whenever we assert $w_I$.

a. We will look at the simplest case when $\mathsf{re}$ is a regular expression over $\Sigma$. Let $\varphi_{\mathsf{re}}$ be the LTL formula equivalent to $\mathsf{re}$.

i. Let $\mathsf{re} = a$ for $a \in \Sigma$. We expand the alphabet by allowing proposition $x \in I$ (and its negation $x \notin I$). The formula $\psi = ((x \notin I) \cup [\varphi_{\mathsf{re}} \land (x \in I) \land O(\square(x \notin I))])$ is then an LTL formula that says that there is a single point in the region $I$, and $a$ holds at that point. Then $\psi_{\mathsf{re}} = x.\psi$ is a 1-TPTL formula that captures $\mathsf{Rat}_\mathsf{f}a$. 


The Main Idea

1. Let \( \text{re} = a.b \) for \( b \in \Sigma \). Then we inductively assume LTL formulae \( \varphi_a \) and \( \varphi_b \) that capture \( a \) and \( b \). As above, we allow the proposition \( x \in I \). Then the formula\
   \[
   \psi_{\text{re}} = x \cdot \{(x \notin I) \cup ([x \in I) \land \varphi_b \land \Diamond \varphi_a \land \Diamond (x \notin I))\}\
   \]
   asserts the existence of two points in the interval \( I \) respectively satisfying in order, \( a \) and \( b \). This argument can be extended to work for any finite concatenation \( \text{re} = a_1.a_2 \ldots a_n \).

2. Let \( \text{re} = (r_1)^* \) be a rational expression over \( \Sigma \). Let \( \varphi_{r_1} \) be the LTL formula equivalent to \( r_1 \). Then the formula \( x \cdot \{(x \notin I) \cup ([x \in I) \land \varphi_{r_1} \land \Diamond \varphi_{r_1} \land \Diamond (x \notin I))\} \) is a 1-TPTL formula that asserts the formula \( \varphi_{r_1} \) at all points in \( I \).

3. Let \( \text{re} = r_1 + r_2 \) be a rational expression over \( \Sigma \). Let \( \varphi_{r_1} \) and \( \varphi_{r_2} \) be LTL formulae equivalent to \( r_1 \), \( r_2 \). Then \( ([x \notin I) \cup ([x \in I) \land \varphi_{r_1} \land \Diamond \varphi_{r_2} \land \Diamond (x \notin I))\} \) is then an LTL formula that says that there is a single point in the region \( I \), and one of \( \varphi_{r_1}, \varphi_{r_2} \) holds at that point.

b. Finish the base case of the structural induction, where \( \text{re} \) was a rational expression over \( \Sigma \), we now move on to general cases.

i. Let us now consider the case when we have a formula \( \text{Rat}_1 \cdot \text{Rat}_2 \cdot \text{re} \).

   Then we first obtain as seen above, \( x. \zeta_{\text{re}} \) equivalent to \( \text{Rat}_1 \cdot \text{Rat}_2 \cdot \text{re} \). Let \( x. \zeta_{\text{re}} = w \).

   Then, \( \zeta_{\text{out}} = ([x \notin I_1] \cup ([x \in I_1) \land \Diamond \varphi_{\text{precedes region } I_2} \land \Diamond (x \notin I_1))\} \) is an LTL formula which asserts the existence of a single point lying in the interval \( I_1 \) where \( w \) is true. Then \( x. \zeta_{\text{out}} \) is a 1-TPTL formula over \( \Sigma \) that is equivalent to \( \text{Rat}_1 \cdot \text{Rat}_2 \cdot \text{re} \).

ii. If we have \( \text{Rat}_1 \cdot \text{Rat}_2 \cdot \text{re} \), then let \( w_1 = x. \zeta_{\text{re}}, w_2 = x. \zeta_{\text{re}} \). Then the formula \( x \cdot \{(x \notin I) \cup ([x \in I) \land \varphi_{r_1} \land \Diamond \varphi_{r_2} \land \Diamond (x \notin I))\} \) asserts the existence of two points in the interval \( I \) respectively satisfying in order, \( \text{Rat}_1 \cdot \text{re}_1 \) and \( \text{Rat}_2 \cdot \text{re}_2 \).

iii. If we have \( \text{Rat}_1 \cdot \text{Rat}_2 \cdot \text{re} \), then let \( w_1 = x. \zeta_{\text{re}} \). Then \( w = x \cdot \{(x \notin I) \cup ([x \in I) \land \varphi_{r_1} \land \Diamond \varphi_{r_2} \land \Diamond (x \notin I))\} \). Then \( x.w \) is a 1-TPTL formula that asserts that at all points in \( I \), the formula \( \text{Rat}_1 \cdot \text{re}_1 \) evaluates to true.

iv. If we have \( \text{Rat}_1 \cdot \text{Rat}_2 \cdot \text{re} \), then let \( w_1 = x. \zeta_{\text{re}} \) and \( w_2 = x. \zeta_{\text{re}} \). Then \( x \cdot \{(x \notin I) \cup ([w \in I) \land \varphi_{r_1} \land \Diamond \varphi_{r_2} \land \Diamond (x \notin I))\} \) is a 1-TPTL formula that checks that \( \text{Rat}_1 \cdot \text{re}_1 \) or \( \text{Rat}_2 \cdot \text{re}_2 \) evaluates to true at the single point in the interval \( I \).

Note that the boolean combinations like conjunction, disjunction and unary operations like negation can be handled in a straightforward way, once we are done with the above. While encountering boolean combinations, we simply combine the 1-TPTL formulae obtained so far.

In case \( I \) is an unbounded interval, then we need not concatenate \( \Diamond \varphi_{(x \notin I)} \) at the end, since the time stamps of all points in the suffix lie in \( I \). The rest of the proof is the same.

\[\text{J} \quad \text{Proof of Lemma 9} \]

The Main Idea: Let \( A \) be a po-1-clock ATA with locations \( S = \{s_0, s_1, \ldots, s_n\} \). Let \( K \) be the maximal constant used in the guards \( x \sim c \) occurring in the transitions. Let \( R_{2i} = [i, i], R_{2i+1} = (i, i+1), 0 \leq i < K \) and \( R_K^+ = (K, \infty) \) be the regions \( R \) of \( x \). Let \( R_h < R_k \) denote that region \( R_h \) precedes region \( R_k \). For each location \( s \), \( \text{Beh}(s) \) as seen above (also Figure 7) gives the timed behaviour starting at \( s \), using constraints \( x \sim c \) since the point where \( x \) was frozen. In example 6, \( \text{Beh}(s_a) = (x < 1) \cup (x > 1) \), allows symbols \( a, b \) as long as \( x < 1 \) keeping the control in \( s_a \), has no behaviour at \( x = 1 \), and allows control to leave \( s_a \) when \( x > 1 \). For any \( s \), we “distribute” \( \text{Beh}(s) \) across regions by untiming it. In example 6, \( \text{Beh}(s_a) = \Diamond^{s_a}(a \lor b) \) for regions \( R_0, R_1 \), it is \( \bot \) for \( R_2 \) and ...
is \((a \lor b)\) for \(R^+_l\). Given any \(\text{Beh}(s)\), and a pair of regions \(R_j \leq R_k\), such that \(s\) has a non-empty behaviour in region \(R_j\), and control leaves \(s\) in \(R_k\), the untimed behaviour of \(s\) between regions \(R_1, \ldots, R_e\) is written as LTL formulae \(\varphi_j, \ldots, \varphi_e\). This results in a “behaviour description” (or BD for short) denoted \(\text{BD}(s, R_j, R_k)\): this is a \(2K + 1\) tuple with \(\text{BD}(R_l) = \varphi_l\) for \(j \leq l \leq k\), and \(\text{BD}(R) = \top\) denoting “dont care” for the other regions. Each LTL formula \(\text{BD}(s, R_j, R_k)[R_l]\) (or \(\text{BD}(R_l)\) when \(s, R_j, R_k\) are clear) is replaced with a star-free rational expression denoted \(\text{re}(\text{BD}(s, R_j, R_k)[R_l])\). Then \(\text{BD}(s, R_j, R_k)\) is transformed into a \(\text{FrhMTL}\) formula \(\varphi(s, R_j, R_k) = \bigwedge_{j \leq g \leq k} \text{Rat}_{R_g} \text{re}(\text{BD}(s, R_j, R_k)[R_g])\). The language accepted by the \(a\)-1-clock \(\mathcal{A}TA\) is then given by \(\bigvee_{0 \leq i \leq k \leq 2K} \varphi(s_0, R_j, R_k)\) where \(s_0\) is the initial location, and the word is accepted while in region \(R_k\). This disjunction allows all possible accepting behaviours from the initial location \(s_0\).

Each location \(s\) is associated with a set of BDs. Let \(\text{BDSet}(s)\) denote the of BDs that are associated with \(s\). If \(s\) is the lowest location in the partial order, then \(\text{BDSet}(s) = \{\text{BD}(s, R_i, R_j) \mid R_i \leq R_j\}\).

**Computing \(\text{BD}(s, R_i, R_j)\) for a location \(s\) and pair of regions \(R_i \leq R_j\).** The proof proceeds by first computing \(\text{BD}(s, R_i, R_j)\) for locations \(s\) which are lowest in the partial order, followed by computing \(\text{BD}(s', R_i, R_j)\) for locations \(s'\) which are higher in the order. For any location \(s\), \(\text{Beh}(s)\) has the form \(\varphi_1 \text{W} \varphi_2\) or \(\varphi_1 \text{U}^a \varphi_2\), or \(\varphi\), where \(\varphi, \varphi_1, \varphi_2\) are conjunctions of \(\Phi(S \cup \Sigma \cup X)\), where \(S\) is the set of locations with or without the binding construct \(x\), and \(X\) is a set of clock constraints of the form \(x \sim c\). Each conjunct has the form \(\psi \land x \in R\) where \(\psi \in \Phi(S \cup \Sigma)\) and \(R \in \mathcal{R}\). Let \(\varphi_1 = \bigvee(P_i \land C_i), \varphi_2 = \bigvee(Q_j \land E_j)\) where \(P_i, Q_j \in \Phi(S \cup \Sigma)\) and \(C_i, E_j \in \mathcal{R}\). Let \(C\) and \(E\) be a shorthand notation to represent any \(C_k, E_k\).

For \(R_i \leq R_j\), and a location \(s\), \(\text{BD}(s, R_i, R_j)\) is empty if \(\text{Beh}(s)\) has no constraint \(x \in R_i\) occurring in \(C, E\), and if control cannot exit \(s\) in \(R_j\). If \(\text{Beh}(s)\) has no \(U, W\) modalities, then \(\text{BD}(s, R_i, R_j)\) is computed when \(\text{Beh}(s) = \bigvee(Q_j \land E_j)\) and there is some \(E_j\) with \(x \in R_i\). In this case, \(\text{BD}(s, R_i, R_j)[R_i] = Q_i\), and the remaining entries are \(\top\) representing “dont care”. If \(\text{Beh}(s)\) has \(U, W\) modalities, then \(\text{BD}(s, R_i, R_j)\) is computed when (1) there is a constraint \(x \in R_i\) in \(C\) or \(E\) (this allows us to start observing the behaviour in region \(R_j\)) (2) there is a constraint \(x \in R_j\) in some \(E\) (this allows us to exit the control location \(s\) while in region \(R_j\). If so, the \(\text{BD}(s, R_i, R_j)\) is a \(2K + 1\) tuple with (i) formula \(\top\) in regions \(R_0, \ldots, R_{i-1}, R_i+1, \ldots, R_{K-1}\) (denoting dont care), (ii) if \(C_k = E_k = (x \in R_j)\) for some \(C_k, E_k\), then the LTL formula in region \(R_j\) is \(P_k \text{UQ} Q_i\) if \(s\) is not an accepting location, and is \(P_k \text{WQ} Q_i\) if \(s\) is an accepting location, (iii) If no \(C_k\) is equal to any \(E_l\) for any \(k, l\), and if \(E_i = (x \in R_j)\) for some \(i\), then the formula in region \(R_j\) is \(Q_i\). If \(C_m = (x \in R_i)\) for some \(m\), then the formula for region \(R_j\) is \(\text{D}^a P_m\). If there is some \(C_h = (x \in R_w)\) for \(i < w < j\), then the formula in region \(R_w\) is \(\text{D}^a P_i \lor \epsilon\), where \(\epsilon\) signifies the fact that there may be no points in regions \(R_w\). If there are no \(C_m\)’s such that \(C_m = (x \in R_w)\) for \(R_i \prec R_w \prec R_j\), then the formulae in region \(R_w\) is \(\epsilon\). We allow \(\epsilon\) as a special symbol in LTL to signify that there is no behaviour in a region.

**BD(s, R_i, R_j) for location s lowest in po.** Let \(s\) be a location that is lowest in the partial order. The locations \(s_i, s_a\) in Example 6 are lowest in the partial order, and \(\text{Beh}(s_i) = bW \bot = \text{D}^a b\), \(\text{Beh}(s_a) = [(a \lor b) \land (x < 1)] \text{U}^a [(a \lor b) \land (x > 1)]\). In general, if \(s\) is the lowest in the partial order, then \(\text{Beh}(s)\) has the form \(\varphi_1 \text{W} \varphi_2\) or \(\varphi_1 \text{U}^a \varphi_2\), or \(\varphi\), where \(\varphi, \varphi_1, \varphi_2\) are conjunctions of \(\Phi(S \cup \Sigma \cup X)\). Each conjunct has the form \(\psi \land x \in R\) where \(\psi \in \Phi(S)\) and \(R \in \mathcal{R}\). In example 7 the regions are \(R_0 = [0, 0], R_1 = (0, 1), R_2 = [1, 1], R_3 = (1, \infty)\). \(\text{Beh}(s_i, R_i, R_j^+) = (\top, \text{D}^a b, \text{D}^a b \lor \epsilon, bW \bot)\), and \(\text{Beh}(s_a, R_0, R_i^+) = (\text{D}^a (a \lor b), \text{D}^a (a \lor b) \lor \epsilon, (a \lor b))\). If \(\epsilon\) in the sole entry in a region, it represents that there
is no behaviour in that region. If \( \epsilon \) is a disjunct \( \psi \lor \epsilon \), then it represents a possibility of no behaviour, or a behaviour \( \psi \).

Using the BDs of \( s_0 \), we can write the SfrMTL formula that describes the behaviour of \( s_0 \). This formula is given by \( \psi(s_0) = \varphi_{R_0}(s_0) \land \varphi_{R_1}(s_0) \land \varphi_{R_2}(s_0) \land \varphi_{R_1}^+(s_0) \), where each \( \varphi_{R_i} \) describes the behaviour starting from region \( R_i \), while in location \( s_0 \). For a fixed region \( R_i \), \( \varphi_{R_i}(s_0) \) is \( \bigwedge_{R_0 \prec R_i} \text{Rat}_{R_i} \Sigma \land \text{Rat}_{R_i}^+ \rightarrow \{ \bigvee_{R_0 \prec R_i} \varphi(s_0, R_i, R_j) \} \), where \( \varphi(s_0, R_i, R_j) \) is described above. \( \text{Rat}_{R_i} \Sigma \) represents that there is no behaviour in \( R_i \). Recall that \( \varphi(s_0, R_0, R_i) \) describes a possible behaviour of \( s_0 \) that starts at \( R_0 \) and ends in \( R_i \). For instance, \( \varphi_{R_0}(s_0) \) is \( \text{Rat}_{R_0} \Sigma \rightarrow \{(\text{Rat}_{R_0}(a+b)^* \land \text{Rat}_{R_0} [(a+b)^* + \epsilon] \land \text{Rat}_{R^0} \land \text{Rat}_{R^0}^+ (a+b)^*\} \) while \( \varphi_{R_1}(s_0) \) is \( \text{Rat}_{R_1} \Sigma \rightarrow \{(\text{Rat}_{R_1}(a+b)^* \land \text{Rat}_{R_1} \land \text{Rat}_{R_1}^+ (a+b)^*\} \). Similarly, \( \varphi_{R_2}(s_0) \) is empty since \( s_0 \) has no behaviour in \( R_2 \). Finally, \( \varphi_{R_1}^+(s_0) \) is \( \bigwedge_{R_0 \prec R_1} \text{Rat}_{R_1} \Sigma \land \text{Rat}_{R_1}^+ \rightarrow \text{Rat}_{R_1}^+ \Sigma \rightarrow \text{Rat}_{R_1}^+ (a+b)^* \). In a similar manner, we can write the SfrMTL formula \( \psi(s_1) \) that describes the behaviour of \( s_1 \) across regions.

**BD(s, R_i, R_j) for a location s which is higher up.** If \( s \) is not the lowest in the partial order, then \( \text{Beh}(s) \) has locations \( s' \in \downarrow \ s \). \( s' \) occurs as \( O(s') \) or \( X(s') \) in \( \text{Beh}(s) \). We now elaborate the operations needed to combine BDs.

**Boolean Combinations of BDs.** Let \( s_1, s_2 \) be two locations of the po-1-clock ATA \( A \). Assume \( \text{Beh}(s_1) = \varphi_1 U^\alpha \varphi_2 \lor \varphi_1 W \varphi_2 \) and \( \text{Beh}(s_2) = \psi_1 U^\alpha \psi_2 \lor \psi_1 W \psi_2 \). We have already seen how to handle \( x.O\text{Beh}(s_1) \) or \( x.O\text{Beh}(s_2) \). So let us assume \( s_1, s_2 \) appear in \( \text{Beh}(s) \) as \( \text{OBeh}(s_1) \) and \( \text{OBeh}(s_2) \).

Consider \( \text{BDSet}(s_1) \) and \( \text{BDSet}(s_2) \), and consider any pair of BDs, say \( \text{BD}(s_1, R_i, R_j) \) and \( \text{BD}(s_2, R_i, R_k) \) from these respectively. The boolean operations are defined for each pair taken from \( \text{BDSet}(s_1) \) and \( \text{BDSet}(s_2) \).

Take \( \text{BD}(s_1, R_i, R_j) \) and \( \text{BD}(s_2, R_i, R_k) \) respectively from \( \text{BDSet}(s_1) \) and \( \text{BDSet}(s_2) \). We now define boolean operations \( \land \) and \( \lor \) on these BDs.

The BDSet for \( s_1 \land s_2 \): Consider \( \text{BD}_1 = \text{BD}(s_1, R_i, R_j) \) and \( \text{BD}_2 = \text{BD}(s_2, R_i, R_k) \), both which describe behaviours of \( s_1, s_2 \) starting in region \( R_i \). Assume \( R_j \prec R_k \) (the case of \( R_k \prec R_j \) is similar). To obtain a BD conjunction these two, starting in region \( R_i \), we do the following. Construct \( \text{BD}' \) by conjunction the entries of \( \text{BD}_1, \text{BD}_2 \) component wise. This will ensure that we take the possible behaviour of \( \text{Beh}(s_1) \) at region \( R_i \) and conjunct it with the possible behaviour of \( \text{Beh}(s_2) \) in the same region. \( \text{BD}' \in \text{BDSet}(s_1 \land s_2) \). In a similar way, we can also compute the \( \text{BDSet}(s_1 \lor s_2) \).

**Elimination of OBeh(s') from BD(s, R_i, R_j)\)**

Given any \( \text{Beh}(s) \) of the form \( \bigvee_i (P_i \land C_i) \) \( U^\alpha \bigvee_j (Q_j \land E_j) \) or \( \bigvee_i (P_i \land C_i) \) \( W \bigvee_j (Q_j \land E_j) \) with \( P_i, Q_j \in \Phi(\Sigma \cup S) \), and \( C_i, E_j \) are clock constraints of the form \( x \in R \). Assume that we have calculated \( \text{BD}(s', R', R') \) for all \( s' \in \downarrow \ s \) and all regions \( R, R' \). There might be some propositions of the form \( \text{OBeh}(s') \) as a conjunct in some entries of \( \text{BD}(s, R_i, R_j) \). This occurrence of \( \text{OBeh}(s') \) is eliminated by "stitching" the behaviour of \( s' \) with \( \text{BD}(s, R_i, R_j) \) as follows:

1. We consider three cases here, depending on how \( \text{OBeh}(s') \) occurs in \( \text{BD}_1 = \text{BD}(s, R_i, R_j) \).
   - As a first case, let \( \text{BD}_1 = (X_0, \ldots, X_0, Q_j, O(\text{Beh}(s')), X_{g+1}, \ldots, X_{2K}) \).
   1. To eliminate \( \text{OBeh}(s') \) from \( \text{BD}_1 \), we first recall that \( s' \in \downarrow \ s \) and that \( \text{BD}(s', R_k, R_l) \) has been computed for all regions \( R_k, R_l \). \( \text{Beh}(s) \) will not occur in any of these BDs corresponding to \( s' \).
   2. The first thing to check is which region \( R_g \) (or later) where the next point can be enabled, based on the behaviour of \( s' \). There are \( 2K - g + 1 \) possibilities, depending on which region \( \geq g \) the next point lies with respect to \( Q_j \land O(\text{Beh}(s')) \).
Suppose the next point can be taken in $R_g$ itself. This means that from the next point, all the possible behaviours described by any of the BD’s $BD(s', R_g, R_h)$ will apply along with $BD_1$. We define a binary operation combine which combines two BDs, $BD_1 = BD(s, R_i, R_j)$ and $BD_2 = BD(s', R_g, R_h)$, producing a new BD $BD_3 = \text{combine}(BD_1, BD_2)$. To combine the behaviours from the point where $\text{Beh}(s')$ is encountered, we substitute $O\text{Beh}(s')$ with the LTL formula asserted at region $R_g$ in $BD_2$. If $BD_2[R_h]$ represents the $h$th component, then we replace $O\text{Beh}(s')$ in $BD_1[R_g]$ with $BD_2[R_h]$. Thus, $BD_3[R_g] = Q_j \land BD_2[R_g]$. For all $R_w < R_g$, $BD_3[R_w] = BD_1[R_w] \land BD_2[R_w]$.

Now consider the case when the next point is taken a region $> R_g$. The next point can occur in region $R_g+1$ or higher. Let $b \in \{g + 1, \ldots, 2K\}$, and assume that the next point where $\text{Beh}(s')$ has a behaviour is in $R_b$. Then given $BD_1 = BD(s, R_i, R_j)$ and $BD_2 = BD(s', R_g, R_h)$ such that $BD_2[g + 1], \ldots, BD_2[b - 1] = 1$, we obtain $BD_3$ as follows. For all $R_w < R_g$, $BD_3[R_w] = BD_1[R_w] \land \epsilon = 1$. The conjunction with $\square \bot$ signifies that the next point in $R_b$ is not available for $s'$, since $s'$ has no behaviour in $R_g$. For all $b > w > g$, $BD_3[R_w] = BD_1[R_w] \land \epsilon = 1$. This implies the next point from where the assertion $Q_j \land O(\text{Beh}(s'))$ was made is in a region $> R_b$. For all $w \geq h$, $BD_3[R_w] = BD_1[R_w] \land BD_2[R_w]$. This combines the assertions of both the behaviours from the next point onwards.

As a second case, consider $BD_1 = [X_0, \ldots, X_{g-1}, \square \text{mark}(P_j \land O(\text{Beh}(s'))), X_{g+1}, \ldots, X_{2K}]$. Elimination of $O\text{Beh}(s')$ in this case is similar to case 1.

As the third case, let $BD_1 = [X_0, \ldots, X_{g-1}, P_i \land O(\text{Beh}(s_1)) \sqcup n Q_j \land O(\text{Beh}(s_2)), X_{g+1}, \ldots, X_{2K}]$, and we have to eliminate both $O\text{Beh}(s_1)$ and $O\text{Beh}(s_2)$. Either $Q_j \land O(\text{Beh}(s_2))$ is true at the present point or, $P_i \land O(\text{Beh}(s_1))$ is true until some point in the future within the region $R_g$, at which point, $Q_j \land O(\text{Beh}(s_2))$ becomes true. Thus, $BD_1$ can be replaced with two BDs

$BD_1' = [X_0, \ldots, X_{g-1}, Q_j \land O(\text{Beh}(s_2)), X_{g+1}, \ldots, X_{2K}]$, and

$BD_2' = [X_0, \ldots, X_{g-1}, P_i \land O(\text{Beh}(s_1)) \sqcup n Q_j \land O(\text{Beh}(s_2)), X_{g+1}, \ldots, X_{2K}]$. Elimination of $O\text{Beh}(s_2)$ is done from $BD_1'$ as seen in case 1. Consider $BD_1''$ which guarantees that the next point from which the assertion $P_i \land O(\text{Beh}(s_1)) \sqcup n Q_j \land O(\text{Beh}(s_2))$ is made is within region $R_g$, and that $O(\text{Beh}(s_1))$ is called for the last time within $R_g$. $BD_1''$ has to be combined with any $BD(s_1, R_g, R_h)$, which has a starting behaviour of $s_1$ from region $R_g$. $s_2$ can have an enabled transition from any point either within region $R_g$ or a succeeding region.

Consider the case where $s_2$ has an enabled transition from within the region $R_g$. In this case, we have to combine $BD_1''$ with some $BD_3 = BD(s_1, R_i, R_j)$ and with some $BD_4 = BD(s_2, R_g, R_h)$. Let $BD_3 = (Y_0, \ldots, Y_{2K})$ and let $BD_4 = (Z_0, \ldots, Z_{2K})$. We now show to combine $BD_1''$, $BD_3$ and $BD_4$ obtaining a BD $(A_0, \ldots, A_{2K})$.

For every $w < g$, $A_w = X_w$. For $w = g$, $A_g$ is obtaining by replacing $O\text{Beh}(s_1)$ with $Y_g$ and $O\text{Beh}(s_2)$ with $Z_g$. For all $w > g$, $A_w = X_w \land Y_w \land Z_w$.

Now consider the case where $s_2$ has an enabled transition from a region $R_0$ such that $R_g < R_0$. In this case, $A_w = X_w$ for $w < g$. The main difference with the earlier case is that we have to assert that from the last point in $R_g$, the next point only occurs in the region $R_0$. Thus all the regions between $R_g$ and $R_0$ should be in $(A_0, \ldots, A_{2K})$. That is, $A_w = 1$ for $g < w < b$. For $w = g$, $A_g = (P_i \land Y_g) \sqcup (Q_j \land \square \bot)$, where $P_i, Q_j$ are obtained from $BD_i'[R_g]$. Here again, conjuncting $\square \bot$ with $Q_j$ signifies that the next point is not enabled for $s_2$. Finally, for $w \geq b$, $A_w = X_w \land Y_w \land Z_w$. 

\[\text{Krishna, Madnani, Pandya}\]
Note that elimination of $O\text{Beh}(s')$ from any BD in the set $\text{BDSet}(s)$ results in stitching some BD from $\text{BDSet}(s')$ to certain elements of $\text{BDSet}(s)$. At the end of the stitching, we obtain $\text{BDSet}(s)$ such that in each BD of $\text{BDSet}(s)$, $O\text{Beh}(s')$ has been replaced.

### Obtaining SfrMTL Formulae

Finally, we show that given a BD for $\text{Beh}(s)$, we can construct an SfrMTL formula, $\psi_s$, equivalent to $x.O(s)$. That is, $\rho, i, \nu \models \psi_s$ if and only if $\rho, i, \nu \models x.O(\text{Beh}(s))$, for any $\nu$. Recall that $\text{Beh}(s)$ is a 1-TPTL formula, as computed in lemma 7. We give a constructive proof as follows:

Assume $\rho, i, \nu \models x.O(\text{Beh}(s))$. Note that according to the syntax of TPTL, every constraint $x \in I$ checks the time elapse between the last point where $x$ was frozen. Thus satisfaction of formulae of the form $x.\phi$ at a point is independent of the clock valuation. 

$\rho, i, \nu \models x.O(\text{Beh}(s))$ iff $\rho, i, \nu[x \leftarrow \tau_i] \models O\text{Beh}(s)$. We have precomputed $\text{BD}(s, R_i, R_j)$ for all regions $R_i \preceq R_j$; and $\text{BD}(s, R_i, R_j)$ is guided by the 1-TPTL formula $\text{Beh}(s)$. The entry in region $R_i$ of $\text{BD}(s, R_i, R_j)$ depends on the behaviour allowed in region $R_i$ from location $s$; likewise, the entry in each region $R_k$ of $\text{BD}(s, R_i, R_j)$ is obtained by looking up $\text{Beh}(s)$. In case $\text{Beh}(s)$ does not admit any behaviour in a region $R_k$, then the $g$th entry in $\text{BD}(s, R_i, R_j)$ is $\perp$. Thus, $\rho, i, \nu \models x.O(\text{Beh}(s))$ iff for all $w \in 0, \ldots, 2K$, such that $\text{Beh}(s)$ has an allowed behaviour in region $R_w$, $\rho, i + 1, \tau_i \models (x \in R_w)$. In addition, we also know that there is some $\text{BD}(s, R_w, R_j)$ such that $\text{BD}[R_k]$ is the LTL formula that describes the behaviour in region $R_k$ of location $s$.

Note that, $\rho, i + 1, \tau_i \models (x \in R_w) \begin{array}{l} \text{true, iff, } \rho, i \models \bigwedge_{g \in \{1, \ldots, w-1\}} \text{Rat}_{R_g} \epsilon \wedge \text{Rat}_{R_w} \Sigma^+ \end{array}$. This is true iff $\rho, i \models \bigvee_{\text{BD} = \text{BD}(s, R_w, R_j)} \bigwedge_{k \in \{1, \ldots, 2K\}} \text{Rat}_{R_k}(\text{re(\text{BD}[R_k])})$, where $\text{re(\text{BD}[R_k])}$ is a star-free rational expression equivalent to the LTL formula $\text{BD}[R_k]$.

Thus, $\rho, i, \nu \models x.O(\text{Beh}(s))$, iff, $\rho, i \models (\psi_1 \rightarrow \psi_2)$ where

\begin{align*}
\psi_1 &= \bigwedge_{w \in \{0, \ldots, 2K\} \setminus E} \bigwedge_{g \in \{1, \ldots, w-1\}} \text{Rat}_{R_g} \epsilon \wedge \text{Rat}_{R_w} \Sigma^+ \\
\psi_2 &= \bigvee_{\text{BD} = \text{BD}(s, R_w, R_j)} \bigwedge_{k \in \{1, \ldots, 2K\}} \text{Rat}_{R_k}(\text{re(\text{BD}[R_k])}).
\end{align*}

where $E$ is the set of regions where $\text{Beh}(s)$ has no behaviour. The SfrMTL formula $\psi_{s_0}$ is one which begins in the initial location $s_0$, stitches the behaviours of locations $s_i$ that appear in a run from $s_0$ such that $L(\psi_{s_0})$ is non-empty iff the language accepted by the po-1-clock $\text{ATA} \lambda$ is non-empty, and $L(\psi_{s_0}) = L(\lambda)$.

Consider the po-1-clock $\text{ATA} \lambda = \{(a, b), \{s_0, s_a, s_t\}, s_0, \{s_0, s_t\}, \delta\}$ with transitions $\delta(s_0, b) = s_0, \delta(s_0, a) = (s_0 \wedge x.\text{Beh}(s_a)) \vee s_t, \delta(s_a, a) = (s_a \wedge x < 1) \vee (x > 1) = \delta(s_a, b)$, and $\delta(s_t, b) = s_t, \delta(s_t, a) = \perp$.

Consider the subset of $L(\lambda)$ consisting of timed words whose first symbol occurs at a time $> 1$. We write a SfrMTL formula that captures this subclass.

Let us consider the formula we obtain if we consider allowed behaviours from $s_0$ that begin in the region $R^+_1$; this is the subset of $\text{BDSet}(s_0)$ consisting of $\text{BD}(s_0, R^+_1, R^+_1) = (\top, \top, \top, \top) \cup \{(a \wedge x.\text{Beh}(s_a))\}$. We look at the SfrMTL formula $\psi_{s_0}$ corresponding to $x.\text{Beh}(s_a)$, which is given by

\begin{align*}
\text{Rat}_{R_0} \Sigma^+ &\rightarrow \{\text{Rat}_{R_0} (a + b)^* \wedge \text{Rat}_{R_1} ((a + b)^* + \epsilon) \wedge \text{Rat}_{R_2} \epsilon \wedge \text{Rat}_{R^+_1} (a + b)^* \} \\
\text{Rat}_{R_0} \epsilon \wedge \text{Rat}_{R_1} \Sigma^+ &\rightarrow \{\text{Rat}_{R_0} \top \wedge \text{Rat}_{R_1} (a + b)^* \wedge \text{Rat}_{R_2} \emptyset \wedge \text{Rat}_{R^+_1} (a + b)^* \} \\
\text{Rat}_{R_0} \epsilon \wedge \text{Rat}_{R_1} \epsilon \wedge \text{Rat}_{R^+_1} \Sigma^+ &\rightarrow \text{Rat}_{R^+_1} (a + b)^*.
\end{align*}

This formula $\psi_{s_0}$ is plugged in place of $x.\text{Beh}(s_a)$ in $\text{BD}(s_0, R^+_1, R^+_1)$. We now combine
BD(s_\ell, R^+_1, R^+_1) \in BDSet(s_\ell) with BD(s_0, R^+_1, R^+_1)[R^+_1] to obtain the combined behaviour of locations s_\ell from the next point along with that of s_0. We know that BD(s_\ell, R^+_1, R^+_1) = (\top, \top, \top, b W \perp). Thus, we obtain BD(s_0, R^+_1, R^+_1) after combining BD(s_\ell, R^+_1, R^+_1) and \psi_s as (\top, \top, \top, [(a \land \psi_s) \lor b] W(a \land (b W \perp))). Translating this into an SfrMTL formula, we obtain the formula \varphi_{R^+_1}(s_0)

\varphi_{R^+_1}(s_0) is the formula which captures the subset of L(A) which consists of timed words of the form (a_1, t_1)(a_2, t_2)\ldots(a_n, t_n) such that t_1 > 1. We can also write the formulae \varphi_{R_0}(s_0), \varphi_{R_1}(s_0), \varphi_{R_2}(s_0), which capture respectively, the subset of words of L(A) which consists of timed words of the form (a_1, t_1)(a_2, t_2)\ldots(a_n, t_n) where t_1 = 0, 0 < t_1 < 1 and t_1 = 1 respectively. Thus, L(A) is the union of the languages L(\varphi_{R^+_1}(s_0)), L(\varphi_{R_0}(s_0)), L(\varphi_{R_2}(s_0)) and L(\varphi_{R_1}(s_0)).