Path-Checking for MTL and TPTL over Data Words

Shiguang Feng[1]⋆, Markus Lohrey[2], and Karin Quaas[1]

[1] Institut für Informatik, Universität Leipzig, Germany
[2] Department für Elektrotechnik und Informatik, Universität Siegen, Germany

Abstract. Precise complexity results are derived for the model-checking problem for TPTL (timed propositional temporal logic) on (in)finite data words and deterministic one-counter machines. Depending on the number of register variables and the encoding of numbers in constraints (unary or binary), the complexity is either P-complete or PSPACE-complete.

1 Introduction

Metric temporal logic and timed propositional temporal logic. *Linear time temporal logic* (LTL) is nowadays one of the main logical formalisms used for the specification and verification of reactive systems, and has found applications in industrial tools. In this context, satisfiability and model-checking are the main computational problems for LTL. The complexity of these problems in various settings is well-understood, see e.g. [3] for background. Triggered by applications in real time systems, various timed extensions of LTL have been invented. Two of the most prominent examples are MTL (metric temporal logic) [11] and TPTL (timed propositional temporal logic) [2]. In MTL, the temporal operators next (X) and until (U) are indexed by time intervals. For instance, the formula $p \,\mathsf{U}_{[2,3)}\, q$ holds at a certain time t, if there is a time $t' \in [t+2, t+3)$, where q holds, and p holds during the interval $[t, t')$. TPTL is a more powerful logic that is equipped with a freeze formalism. It uses register variables, which can be set to the current time value and later these register variables can be compared with the current time value. For instance, the above MTL-formula $p \,\mathsf{U}_{[2,3)}\, q$ is equivalent to the TPTL-formula $x.(p \,\mathsf{U}\, (q \wedge 2 \leq x < 3))$. Here, the constraint $2 \leq x < 3$ should be read as: The difference of the current time value and the value stored in x is in the interval $[2, 3)$. It is a simple observation that every MTL formula can be translated into an equivalent TPTL formula with only one register variable.

For both MTL and TPTL, two different semantics exist: the continuous semantics, where the time domain are the reals, and the discrete semantics, where the time domain are the naturals. In this paper, we will be only interested in the discrete semantics, where formulae are evaluated over finite or infinite sequences $(P_0, d_0)(P_1, d_1)\ldots$ of pairs (P_i, d_i). Here $P_i \subseteq \mathcal{P}$ is a finite set of atomic propositions (from some prespecified finite set $\mathcal{P}$) and $d_i \in \mathbb{N}$ is a time stamp such that $d_i \leq d_{i+1}$ for all $i \geq 0$. Satisfiability and model-checking for MTL and TPTL under both semantics has been studied intensively in the past, see e.g. [2,4,13,14,15].

⋆ The author is supported by the German Research Foundation (DFG), GRK 1763.

Freeze LTL. The freeze mechanism from TPTL has also received attention in connection with data words. A data word is a finite or infinite sequence $(P_0, d_0)(P_1, d_1)\ldots$ of the above form, where we do not require the data values d_i to be monotonic, and we speak of *non-monotonic data words*. As for TPTL, freeze LTL can store the current data value in a register x. But in contrast to TPTL, the value of x can only be compared for equality with the current data value. Results on satisfiability and model-checking for freeze LTL can be found in [8,9]. For model-checking the authors of [9] consider one-counter machines (OCM) as a mechanism for generating infinite data words, where the data values are the counter values along the unique computation path. Whereas freeze LTL model-checking for non-deterministic OCM turned out to be Σ_1^1-complete, the problem becomes PSPACE-complete for deterministic OCM [9].

New results. In this paper, we extend the PSPACE-completeness result for freeze LTL to TPTL *over non-monotonic data words*. This logic extends both freeze LTL (over non-monotonic data words) and TPTL: As for freeze LTL, data values are natural numbers that can vary arbitrarily over time. In contrast to the latter, one can express that the difference of the current data value and the value stored in a register belongs to a certain interval, whereas freeze LTL only allows to say that this difference is zero. Applications for TPTL over non-monotonic data values can be seen in areas, where data streams of discrete values have to be analyzed and the focus is on the dynamic variation of the values (e.g. streams of discrete sensor data or stock charts).

The third author proved recently that model-checking for non-monotonic TPTL over deterministic OCM is still decidable [16], but the complexity remained open. Our first main result states that model-checking for TPTL over deterministic OCM is PSPACE-complete. This sharpens the decidability result from [16] and at the same time generalizes the PSPACE-completeness result for freeze LTL. We also show that PSPACE-hardness already holds (i) for the fragment of TPTL with only two register variables and (ii) for full TPTL, where all interval borders are encoded in unary (the latter result can be shown by a straightforward adaptation of the PSPACE-hardness proof in [9]). On the other hand, if we restrict TPTL to (i) a constant number of register variables and unary encoded numbers in constraints, or (ii) one register variable but allow binary en coded numbers, then model-checking over deterministic OCM is P-complete. Note that the latter covers MTL over non-monotonic data words.

We actually prove our upper complexity bounds for infinite periodic data words. Such a data word is given by two finite data words $u = (P_1, d_1)\cdots(P_m, d_m)$ and $v = (Q_1, e_1)\cdots(Q_n, e_n)$ (where $d_1, \ldots, d_m, e_1, \ldots, e_n \in \mathbb{N}$) together with an offset number K. The resulting infinite data word is $u\prod_{i\geq 0}(v + iK)$, where $v + M$ denotes the data word $(Q_1, e_1 + M)\cdots(Q_n, e_n + M)$. It can be easily seen that the infinite data word produced by a deterministic OCM is such a periodic data word.

Further related work. It is long known that in the monotonic setting, TPTL and MTL are equally expressive [1]. Recently, it has been shown that in the non-monotonic setting, TPTL is strictly more powerful than MTL [6] and that satisfiability for MTL is undecidable [7].

For periodic words without data values (i.e., infinite words of the form uv^ω), the complexity of LTL model-checking (also known as LTL path-checking) belongs to the

class NC (more precisely, $\mathsf{AC}^1(\mathsf{LogDCFL})$) [12]. This result solved a long standing open problem. For finite monotonic data words, the same complexity bound has been shown for MTL in [5]. It is open, whether this result can be extended to MTL over non-monotonic data words.

2 Preliminaries

Computational complexity. We assume some standard knowledge in complexity theory. An *alternating Turing machine* is an ordinary non-deterministic Turing-machine, whose state set Q is partitioned in four disjoint sets Q_{acc} (accepting states), Q_{rej} (rejecting states), $Q_\exists$ (existential states), and $Q_\forall$ (universal states). A configuration c, where the current state is q, is *accepting* if (i) $q \in Q_{\text{acc}}$ or (ii) $q \in Q_\exists$ and there exists an accepting successor configuration of c or (iii) $q \in Q_\forall$ and all successor configurations of c are accepting. The machine accepts an input w if and only if the initial configuration for w is accepting. It is well-known that the class of all languages that can be accepted on an alternating Turing-machine in polynomial time (APTIME) is equal to PSPACE, and that the class of all languages that can be accepted on an alternating Turing-machine in logarithmic space (ALOGSPACE) is equal to P.

Data words. Let $\mathcal{P}$ be a finite set of *atomic propositions*. A *data word* over $\mathcal{P}$ is a finite or infinite sequence $(P_0, d_0)(P_1, d_1)\cdots$ of pairs from $2^{\mathcal{P}} \times \mathbb{N}$. It is *monotonic* (*strictly monotonic*), if $d_i \leq d_{i+1}$ ($d_i < d_{i+1}$) for all appropriate i. It is *pure*, if $P_i = \emptyset$ for all $i \geq 0$. A pure data word is just written as a sequence of natural numbers. We denote with $(2^{\mathcal{P}} \times \mathbb{N})^*$ and $(2^{\mathcal{P}} \times \mathbb{N})^\omega$, respectively, the set of finite and infinite, respectively, data words over $\mathcal{P}$. The *length* of a data word w is denoted by $|w|$, where we set $|w| = \infty$ for the case that w is infinite.

We use u_1u_2 to denote the concatenation of two data words u_1 and u_2, where u_1 has to be finite. Let $u = (P_0, d_0)(P_1, d_1)\cdots$ be a data word and $k \in \mathbb{N}$. We denote with u_{+k} the data word $(P_0, d_0 + k)(P_1, d_1 + k)\cdots$. Given two finite data words u_1, u_2 and $k \in \mathbb{N}$, we use $u_1(u_2)^\omega_{+k}$ to denote the infinite data word $u_1u_2(u_2)_{+k}(u_2)_{+2k}(u_2)_{+3k}\cdots$.

For complexity considerations, the encoding of the data values and the offset number k (in an infinite data word) makes a difference. We speak of *unary* (resp., *binary*) encoded data words if all these numbers are given in unary (resp., binary) encoding.

2.1 Metric temporal logic

The set of formulae of MTL is built up from $\mathcal{P}$ by Boolean connectives, the *next* and the *until* modality using the following grammar, where $p \in \mathcal{P}$ and $I \subseteq \mathbb{Z}$ is an interval with endpoints in $\mathbb{Z} \cup \{-\infty, +\infty\}$:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathsf{X}_I\varphi \mid \varphi\mathsf{U}_I\varphi$$

We use pseudo-arithmetic expressions to denote intervals. For instance, ≥ 1 denotes the interval $[1, \infty)$. If $I = \mathbb{Z}$, then we may omit the index I in U_I.

Formulae of MTL are interpreted over data words. Let $w = (P_0, d_0)(P_1, d_1)\cdots$ be a data word, and let $i \leq |w|$. We define the *satisfaction relation for* MTL inductively as follows (we omit the obvious cases for $\neg$ and $\wedge$):

- $(w,i) \models p$ if and only if $p \in P_i$
- $(w,i) \models \mathsf{X}_I\varphi$ if and only if $i+1 \leq |w|$, $d_{i+1} - d_i \in I$ and $(w,i+1) \models \varphi$
- $(w,i) \models \varphi_1 \mathsf{U}_I \varphi_2$ if and only if there exists a position j with $i \leq j \leq |w|$, $(w,j) \models \varphi_2$, $d_j - d_i \in I$, and $(w,t) \models \varphi_1$ for all $t \in [i,j)$.

We say that a data word *satisfies* an MTL-formula φ, written $w \models \varphi$, if $(w,0) \models \varphi$. Two formulae φ and ψ are *equivalent*, written $\varphi \equiv \psi$, if for every data word w we have $w \models \varphi$ if, and only if, $w \models \psi$. We use the following standard abbreviations: $\varphi_1 \vee \varphi_2 := \neg(\neg\varphi_1 \wedge \neg\varphi_2)$, $\varphi_1 \rightarrow \varphi_2 := \neg\varphi_1 \vee \varphi_2$, $\mathsf{true} := p \vee \neg p$, $\mathsf{false} := \neg\mathsf{true}$, $\mathsf{F}_I\varphi := \mathsf{true}\mathsf{U}_I\varphi$, $\mathsf{G}_I\varphi := \neg\mathsf{F}_I\neg\varphi$, $\varphi_1\mathsf{R}_I\varphi_2 := \neg(\neg\varphi_1\mathsf{U}_I\neg\varphi_2)$. We define the *length* of a formula ψ, denoted by $|\psi|$, as the number of symbols occurring in ψ.

2.2 Timed propositional temporal logic

Next we define formulae of TPTL. For this, let V be a countable set of *register variables*. The set of TPTL-formulae is given by the following grammar, where $p \in \mathcal{P}$, $x \in V$, $c \in \mathbb{Z}$, and $\sim \in \{<, \leq, =, \geq, >\}$:

$$\varphi ::= p \mid x \sim c \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathsf{X}\varphi \mid \varphi\mathsf{U}\varphi \mid x.\varphi$$

We use the same syntactical abbreviations as for MTL. We may also use formulae of the form $x \in I$ as abbreviation for conjunctions of constraints; e.g., we may write $x \in [a,b]$ for $x \geq a \wedge x \leq b$. The *length* of a TPTL-formula is defined as for MTL-formulae. Given $r \geq 1$, we use TPTL^r to denote the fragment of TPTL that uses at most r different register variables.

A *register valuation* ν is a function from V to $\mathbb{Z}$. Given a register valuation ν, a data value $d \in \mathbb{Z}$, and a variable $x \in V$, we define the register valuations $\nu + d$ and $\nu[x \mapsto d]$ as follows: $(\nu + d)(y) = \nu(y) + d$ for every $y \in V$, $(\nu[x \mapsto d])(y) = \nu(y)$ for every $y \in V \backslash \{x\}$, and $(\nu[x \mapsto d])(x) = d$.

Let $w = (P_0, d_0)(P_1, d_1)\cdots$ be a data word, let ν be a register valuation, and let $i \in \mathbb{N}$. The satisfaction relation for TPTL is inductively defined in a similar way as for MTL; we only give the definitions for the new formulae:

- $(w,i,\nu) \models \mathsf{X}\varphi$ if and only if $i+1 \leq |w|$ and $(w,i+1,\nu) \models \varphi$
- $(w,i,\nu) \models \varphi_1\mathsf{U}\varphi_2$ if and only if there exists a position j with $i \leq j \leq |w|$, $(w,j,\nu) \models \varphi_2$, and $(w,t,\nu) \models \varphi_1$ for all $t \in [i,j)$
- $(w,i,\nu) \models x \sim c$ if and only if $d_i - \nu(x) \sim c$
- $(w,i,\nu) \models x.\varphi$ if and only if $(w, \nu[x \mapsto d_i], i) \models \varphi$.

We say that a data word w satisfies a TPTL-formula φ, written $w \models \varphi$, if $(w,0,\bar{0}) \models \varphi$, where $\bar{0}$ denotes the valuation that maps all variables to the initial data value d_0.

For complexity considerations, it makes a difference, whether the numbers c in constraints $x \sim c$ are encoded in binary or unary notation, and similarly for the interval borders in MTL. We write TPTL^r_u, TPTL_u and MTL_u (resp. TPTL^r_b, TPTL_b, and MTL_b) if we want to emphasize that numbers are encoded in unary (resp., binary) notation.

Let w be a data word, $i \in \mathbb{N}$ be a position in w, and let δ be a register valuation. For technical reasons, we define a new *relative* satisfaction relation for TPTL, denoted by $\models^{\mathsf{rel}}$, as follows. For Boolean formulae $\models^{\mathsf{rel}}$ is defined like $\models$. For the other operators we define:

– $(w,i,\delta) \models^{\mathsf{rel}} \mathsf{X}\varphi$ if and only if $i+1 \leq |w|$ and $(w,i+1,\delta+(d_{i+1}-d_i)) \models^{\mathsf{rel}} \varphi$
– $(w,i,\delta) \models^{\mathsf{rel}} \varphi_1 \mathsf{U} \varphi_2$ if and only if there is a position j such that $i \leq j \leq |w|$, $(w,j,\delta+(d_j-d_i)) \models^{\mathsf{rel}} \varphi_2$, and $(w,t,\delta+(d_t-d_i)) \models^{\mathsf{rel}} \varphi_1$ for all $t \in [i,j)$
– $(w,i,\delta) \models^{\mathsf{rel}} x \sim c$ if and only if $\delta(x) \sim c$
– $(w,i,\delta) \models^{\mathsf{rel}} x.\varphi$ if and only if $(w,i,\delta[x \mapsto 0]) \models^{\mathsf{rel}} \varphi$.

We say that data word w satisfies formula φ under the relative semantics, written $w \models^{\mathsf{rel}} \varphi$, if $(w,0,\tilde{0}) \models \varphi$, where $\tilde{0}$ denotes the valuation function that maps all register variables to 0.

Lemma 1. *Let w be a data word and d_i the data value at position i. If $\delta(x) = d_i - \nu(x)$ for every register variable x, then for every* TPTL*-formula φ, $(w,i,\nu) \models \varphi$ if and only if $(w,i,\delta) \models^{\mathsf{rel}} \varphi$.*

Proof. The proof is by induction on φ. We give the proof only for the nontrivial cases.

– If $\varphi = x \sim c$, then $(w,i,\nu) \models x \sim c$ iff $d_i - \nu(x) \sim c$ iff $\delta(x) \sim c$ iff $(w,i,\delta) \models^{\mathsf{rel}} x \sim c$.
– If $\varphi = x.\varphi_1$, then $(w,i,\nu) \models x.\varphi_1$ iff $(w,i,\nu[x \mapsto d_i]) \models \varphi_1$. By induction, the latter holds iff $(w,i,\delta[x \mapsto 0]) \models^{\mathsf{rel}} \varphi_1$, i.e., $(w,i,\delta) \models^{\mathsf{rel}} x.\varphi_1$.
– If $\varphi = \mathsf{X}\varphi_1$, then $(w,i,\nu) \models \mathsf{X}\varphi_1$ iff $i+1 \leq |w|$ and $(w,i+1,\nu) \models \varphi_1$. By induction, the later holds iff $i+1 \leq |w|$ and $(w,i+1,\delta+(d_{i+1}-d_i)) \models^{\mathsf{rel}} \varphi_1$. This is equivalent to $(w,i,\delta) \models^{\mathsf{rel}} \mathsf{X}\varphi_1$.
– If $\varphi = \varphi_1 \mathsf{U} \varphi_2$, then $(w,i,\nu) \models \varphi_1 \mathsf{U} \varphi_2$ iff there exists $i \leq j \leq |w|$ such that $(w,j,\nu) \models \varphi_2$ and $(w,t,\nu) \models \varphi_1$ for all $t \in [i,j)$. By induction, this holds iff there exists $i \leq j \leq |w|$ such that $(w,j,\delta+(d_j-d_i)) \models^{\mathsf{rel}} \varphi_2$ and $(w,t,\delta+(d_t-d_i)) \models^{\mathsf{rel}} \varphi_1$ for all $t \in [i,j)$. This is equivalent to $(w,i,\delta) \models^{\mathsf{rel}} \varphi_1 \mathsf{U} \varphi_2$. □

Corollary 1. *For every data word w and every* TPTL*-formula φ, we have $w \models \varphi$ if and only if $w \models^{\mathsf{rel}} \varphi$.*

2.3 Succinct metric temporal logic

In MTL, U and X modalities are indexed by intervals I. If we replace the interval I by a union of intervals $I_1 \cup I_2 \cup \cdots \cup I_n$, then we call this logic *succinct* MTL (SMTL). Formally, the syntax and semantics of SMTL is the same as for MTL, except that the set I in X_I and U_I can be a finite union $I = I_1 \cup I_2 \cup \cdots \cup I_n$ of intervals $I_i \subseteq \mathbb{Z}$.

Let $I = \bigcup_{i=1}^{n} I_i$. It is easily seen that

$$\mathsf{X}_I\varphi \equiv \bigvee_{i=1}^{n} \mathsf{X}_{I_i}\varphi \text{ and } \varphi_1 \mathsf{U}_I \varphi_2 \equiv \bigvee_{i=1}^{n} \varphi_1 \mathsf{U}_{I_i} \varphi_2,$$

$$\mathsf{X}_I\varphi \equiv x.\mathsf{X}((\bigvee_{i=1}^{n} x \in I_i) \wedge \varphi) \text{ and } \varphi_1 \mathsf{U}_I \varphi_2 \equiv x.\varphi_1 \mathsf{U}((\bigvee_{i=1}^{n} x \in I_i) \wedge \varphi_2).$$

Fact 1 *Each* SMTL*-formula is equivalent to an* MTL*-formula which can be exponentially larger.*

Fact 2 *Each* SMTL*-formula is equivalent to a* TPTL^1*-formula of polynomial size, which, moreover, can be computed in logspace.*

3 Path-checking problems for TPTL and MTL

In this section, we study the path-checking problems for our logics over data words. Data words can be (i) finite or infinite, (ii) monotonic or non-monotonic, (iii) pure or non-pure, and (iv) unary encoded or binary encoded. For one of our logics L and a class of data words C, we consider the *path-checking problem for* L *over* C:

Input: A data word $w \in \mathsf{C}$ and a formula $\varphi \in \mathsf{L}$.
Output: yes if $w \models \varphi$, no otherwise.

3.1 Lower bounds

In this section, we will prove several lower bounds for the classes P and PSPACE. All lower bounds will hold for unary encoded pure monotonic data words. Whereas the first two constructions hold for finite words (and can be easily adapted for infinite data words), the last construction needs an infinite data word.

P-hardness. We prove our P-hardness results by a reduction from a restricted version of the Boolean circuit value problem. A Boolean circuit is *monotone* if it has only $\wedge$ and $\vee$ gates. A *synchronous alternating monotone circuit with fanin 2 and fanout 2* (briefly, SAM2-circuit) is a monotone circuit divided into levels $1, \ldots, l$ ($l \geq 2$) such that the following properties hold:

- The input gates for a gate on level $i < l$ belongs to level $i + 1$.
- All output gates are in level 1 and all input gates are in level l.
- All gates in the same level $i > 0$ are of the same type ($\wedge$ or $\vee$) and the levels alternate between $\wedge$-levels and $\vee$-levels.
- All gates except the output gates have fanout 2 and all gates except the input gates have fanin 2.

By the restriction to fanin 2 and fanout 2, we know that each level contains the same number of gates. The circuit value problem for SAM2-circuits, called SAM2CVP in [10], is the following problem:

Input: An SAM2-circuit α, inputs $x_1, \ldots, x_n$ and designated output gate y.
Output: yes if output y of α evaluates to 1 on inputs $x_1, \ldots, x_n$, no otherwise.

It is shown in [10] that SAM2CVP is P-complete.

Theorem 1. *Path-checking for* SMTL_u *over finite unary encoded strictly monotonic pure data words is* P*-hard.*

Proof. We reduce from SAM2CVP. Let α be an SAM2-circuit with $l \geq 2$ levels and n gates in each level. The idea will be to encode the wires between two consecutive layers by a suitably shifted version of the data word

$$w_n = \prod_{i=1}^{n} i \cdot \prod_{i=1}^{n} i(n+1) = (1, 2, \ldots, n,\ 1 \cdot (n+1), 2 \cdot (n+1), \ldots, n \cdot (n+1)).$$

Note that for all $i_1, i_2 \in \{1, \ldots, n\}$ and $j_1, j_2 \in \{1 \cdot (n+1), 2 \cdot (n+1), \ldots, n \cdot (n+1)\}$ we have the following: If $j_1 - i_1 = j_2 - i_2$ then $i_1 = i_2$ and $j_1 = j_2$. This is best seen by viewing numbers in their base $(n+1)$ expansion. Let us denote with $\Delta = n(n+1) - 1$ the maximal difference between a number from $\{1, \ldots, n\}$ and a number from $\{1 \cdot (n+1), 2 \cdot (n+1), \ldots, n \cdot (n+1)\}$.

We define the pure and strictly monotonic data word $w_{n,l}$ as

$$w_{n,l} = \prod_{j=0}^{l-2} (w_n)_{+j \cdot n(n+2)}.$$

The offset number $j \cdot n(n+2)$ is chosen such that the difference between a number from $\{1, \ldots, n\}$ and a number from $\{1 + j \cdot n(n+2), \ldots, n + j \cdot n(n+2)\}$ is larger than Δ for every $j \geq 1$.

Note that the unary encoding of the data word $w_{n,l}$ can be computed in logspace from the circuit. For instance, $w_{4,3}$ is the following data word:

$$(1, 2, 3, 4,\ 5, 10, 15, 20)(25, 26, 27, 28,\ 29, 34, 39, 44)$$

For each $1 \leq j \leq l-1$, define

$$S_j = \{i_2(n+1) - i_1 \mid \text{ the } i_1^{\text{th}} \text{ gate in level } j \text{ connects to the } i_2^{\text{th}} \text{ gate in level } j+1\}.$$

Suppose o_k $(1 \leq k \leq n)$ is the designated output gate. Let I be the set of all $i \in [1, n]$ such that the i^{th} gate in layer l is set to the Boolean value 1. We construct the SMTL-formula $\psi = \mathsf{X}^{k-1}\varphi_1$, where φ_j $(1 \leq j \leq l-1)$ is defined inductively as follows:

$$\varphi_j = \begin{cases} \mathsf{F}_{S_j} \mathsf{X}^n \varphi_{j+1} & \text{if } j < l-1 \text{ and level } j \text{ is a } \vee\text{-level,} \\ \mathsf{G}_{S_j} \mathsf{X}^n \varphi_{j+1} & \text{if } j < l-1 \text{ and level } j \text{ is a } \wedge\text{-level,} \\ \mathsf{F}_{S_j} (\bigvee_{i \in I} \mathsf{X}^{n-i} \neg \mathsf{X}\, \mathsf{true}) & \text{if } j = l-1 \text{ and level } j \text{ is a } \vee\text{-level,} \\ \mathsf{G}_{S_j} (\bigvee_{i \in I} \mathsf{X}^{n-i} \neg \mathsf{X}\, \mathsf{true}) & \text{if } j = l-1 \text{ and level } j \text{ is a } \wedge\text{-level.} \end{cases}$$

Note that the formula $\neg \mathsf{X}\, \mathsf{true}$ is only true in the last position of a data word. The purpose of the prefix X^n in front of φ_{j+1} is to move from a certain position within the second half of the j^{th} copy of w_n to the corresponding position within the first half of the $(j+1)^{\text{th}}$ copy of w_n in $w_{n,l}$.

It is straightforward to check that $w_{n,l} \models \psi$ iff the circuit α evaluates to 1. □

By Fact 2, we can obtain the following corollary.

Corollary 2. *Path-checking for* TPTL_u^1 *over finite unary encoded strictly monotonic pure data words is* P*-hard.*

Recall that finite path-checking for MTL (which is a proper fragment of TPTL^1) over monotonic data words is in the parallel complexity class $\mathsf{AC}^1(\mathsf{LogDCFL})$ [5].

PSPACE-**hardness.**

Theorem 2. *Path-checking for* TPTL_u *over finite unary encoded strictly monotonic pure data words is* PSPACE*-hard.*

Proof. We prove PSPACE-hardness by a reduction from the PSPACE-complete quantified Boolean formula problem (QBF, for short). Let $\Psi = Q_0 x_0 Q_1 x_1 \cdots Q_{n-1} x_{n-1} \phi$ be a quantified Boolean formula, where $Q_i \in \{\forall, \exists\}$ and ϕ is a quantifier-free propositional formula. We construct the finite pure strictly monotonic data word

$$w_n = 0, 1, 2, \ldots, 2n-1, 2n$$

For every $i \in \{0, \ldots, n-1\}$, the subword $2i, 2i+1$ is used to quantify over the Boolean variable x_i. We use a corresponding register variable x_i. If we assign to this register variable the data value $2i + b$, then the corresponding Boolean variable x_i is set to $b \in \{0, 1\}$.

We define the TPTL-formula $x.\Psi'$, where Ψ' is defined inductively by the following rules.

- If $\Psi = \forall x_i \Phi$, then $\Psi' = \mathsf{G}((x_i = 2i \vee x_i = 2i+1) \rightarrow x_i.\Phi')$.
- If $\Psi = \exists x_i \Phi$, then $\Psi' = \mathsf{F}((x_i = 2i \vee x_i = 2i+1) \wedge x_i.\Phi')$.
- If Ψ is a quantifier-free formula, then

  $$\Psi' = \mathsf{F}(x = 2n \wedge \Psi[x_0/x_0 = 2n-1, x_1/x_1 = 2n-3, \ldots, x_{n-1}/x_{n-1} = 1]).$$

  Here, $\Psi[x_0/x_0 = a_0, \ldots, x_{n-1}/x_{n-1} = a_{n-1}]$ denotes the TPTL-formula obtained from Ψ by replacing every occurrence of x_i by $x_i = a_i$.

It is easy to see that Ψ is `true` if and only if $w \models x.\Psi'$. □

The constructions in the proof of Theorem 1 and 2 can be easily adapted to *infinite* data words. The next lower bound only holds for infinite data words.

Theorem 3. *Path-checking for* TPTL_b^2 *over the infinite strictly monotonic pure data word* $w = 0(1)_{+1}^{\omega} = 0, 1, 2, 3, 4, \ldots$ *is* PSPACE*-hard.*

Proof. The theorem is proved by a reduction from the PSPACE-complete quantified subset sum problem (QSS), see [17]:

Input: A sequence $a_1, a_2, \ldots, a_{2n}, b$ of binary encoded natural numbers.
Output: `yes` if $\forall x_1 \in \{0,1\} \exists x_2 \in \{0,1\} \cdots \forall x_{2n-1} \in \{0,1\} \exists x_{2n} \in \{0,1\}$ such that $\sum_{i=1}^{2n} x_i a_i = b$, `no` otherwise.

Given an instance $a_1, a_2, \ldots, a_{2n}, b$ of QSS, we construct the TPTL-formula $x.y.\varphi_n$, where φ_n is defined inductively by

$$\varphi_i = \mathsf{G}((x = 0 \vee x = a_{2(n-i)+1}) \rightarrow x.\mathsf{F}((x = 0 \vee x = a_{2(n-i)+2}) \wedge x.\varphi_{i-1}))$$

for $1 \leq i \leq n$ and $\varphi_0 = \mathsf{F}(y = b \wedge x = 0)$. We can show that $w \models x.y.\varphi_n$ iff the answer for the QSS-input $(a_1, a_2, \ldots, a_{2n}, b)$ is `yes`. □

3.2 Upper bounds

In this section we prove our upper complexity bounds. All bounds hold also for non-monotonic and non-pure data words (and we will not mention this explicitly in the theorems). But we have to distinguish whether (i) data words are unary or binary encoded, and (ii) whether data words are finite or infinite.

We start with a couple of lemmas. We always assume that u_1 and u_2 are finite data words, $k \geq 0$, $w := u_1(u_2)^{\omega}_{+k}$, $i \geq |u_1|$, and ϕ is a TPTL-formula.

Lemma 2. *For every register valuation δ, $(w, i, \delta) \models^{\mathsf{rel}} \phi$ iff $(w, i + |u_2|, \delta) \models^{\mathsf{rel}} \phi$.*

Proof. Let δ be a register valuation. Define two register valuations ν, ν' by $\nu(x) = d_i - \delta(x)$ and $\nu'(x) = \nu(x) + k$ for every register variable x. By Lemma 1, we have $(w, i, \delta) \models^{\mathsf{rel}} \phi$ iff $(w, i, \nu) \models \phi$, and $(w, i + |u_2|, \delta) \models^{\mathsf{rel}} \phi$ iff $(w, i + |u_2|, \nu') \models \phi$ (since $\nu'(x) = \nu(x) + k = d_i - \delta(x) + k = d_{i+|u_2|} - \delta(x)$). We prove that $(w, i, \nu) \models \phi$ iff $(w, i + |u_2|, \nu') \models \phi$; the claim then follows. For $l \geq 0$, let $w_{\geq l}$ denote the suffix of w starting in position l. Since $i \geq |u_1|$, we have $w_{\geq (i+|u_2|)} = (w_{\geq i})_{+k}$. Hence, we only need to show that $(w_{\geq i}, 0, \nu) \models \phi$ iff $((w_{\geq i})_{+k}, 0, \nu') \models \phi$. This can be shown by a simple induction on the structure of ϕ. □

Lemma 3. *Let δ_1, δ_2 be register valuations. If for every $j \geq i$ and every subformula $x \sim c$ of ϕ we have*

$$(w, j, \delta_1 + (d_j - d_i)) \models^{\mathsf{rel}} x \sim c \;\Leftrightarrow\; (w, j, \delta_2 + (d_j - d_i)) \models^{\mathsf{rel}} x \sim c, \qquad (1)$$

then, we have

$$(w, i, \delta_1) \models^{\mathsf{rel}} \phi \;\Leftrightarrow\; (w, i, \delta_2) \models^{\mathsf{rel}} \phi. \qquad (2)$$

Proof. Assume that (1) holds for all $j \geq i$ and all subformulae $x \sim c$ of ϕ. We prove (2) by induction on the structure of ϕ. The induction base is clear.

Next, let $\phi = x.\phi'$. Then we have $(w, i, \delta_1) \models^{\mathsf{rel}} x.\phi'$ iff $(w, i, \delta_1[x \mapsto 0]) \models^{\mathsf{rel}} \phi'$, and $(w, i, \delta_2) \models^{\mathsf{rel}} x.\phi'$ iff $(w, i, \delta_2[x \mapsto 0]) \models^{\mathsf{rel}} \phi'$. Note that $\delta_1[x \mapsto 0]$ and $\delta_2[x \mapsto 0]$ again satisfy the premise of the lemma. By induction, we obtain $(w, i, \delta_1[x \mapsto 0]) \models^{\mathsf{rel}} \phi'$ iff $(w, i, \delta_2[x \mapsto 0]) \models^{\mathsf{rel}} \phi'$. Hence $(w, i, \delta_1) \models^{\mathsf{rel}} x.\phi'$ iff $(w, i, \delta_2) \models^{\mathsf{rel}} x.\phi'$.

Now let $\phi = \mathsf{X}\phi'$. Then $(w, i, \delta_1) \models^{\mathsf{rel}} \mathsf{X}\phi'$ iff $(w, i + 1, \delta_1 + (d_{i+1} - d_i)) \models^{\mathsf{rel}} \phi'$ and similarly $(w, i, \delta_2) \models^{\mathsf{rel}} \mathsf{X}\phi'$ iff $(w, i + 1, \delta_2 + (d_{i+1} - d_i)) \models^{\mathsf{rel}} \phi'$. Note that $\delta_1 + (d_{i+1} - d_i)$ and $\delta_2 + (d_{i+1} - d_i)$ again satisfy the premise of the lemma. By induction, we get $(w, i+1, \delta_1 + (d_{i+1} - d_i)) \models^{\mathsf{rel}} \phi'$ iff $(w, i+1, \delta_2 + (d_{i+1} - d_i)) \models^{\mathsf{rel}} \phi'$, yielding the result. The proof for the other cases is similar. □

For a TPTL-formula ϕ and a finite data word v we define:

$$C_\phi = \max\{c \in \mathbb{Z} \mid x \sim c \text{ is a subformula of } \phi\} \qquad (3)$$

$$M_v = \max\{d_i - d_j \mid d_i \text{ and } d_j \text{ are data values in } v\} \geq 0 \qquad (4)$$

We may always assume that $C_\phi \geq 0$ (we can add a dummy constraint $x \geq 0$). Note that in the infinite data word v^{ω}_{+k}, for all positions $i < j$ we have $d_j - d_i + M_v \geq 0$ (where as usual d_l is the data value at position l).

Lemma 4. *Let δ a register valuation and define the register valuation δ' by $\delta'(x) = \min\{\delta(x), C_\phi + M_{u_2} + 1\}$ for all x. For every subformula θ of ϕ, we have $(w, i, \delta) \models^{\mathsf{rel}} \theta$ iff $(w, i, \delta') \models^{\mathsf{rel}} \theta$.*

Proof. Let θ be a subformula of ϕ. We prove that the premise of Lemma 3 holds for $\delta_1 = \delta$ and $\delta_2 = \delta'$. The claim then follows from Lemma 3.

Let $j \geq i$, and let $x \sim c$ be a subformula of ϕ. If $\delta(x) \leq C_\phi + M_{u_2} + 1$, and hence $\delta'(x) = \delta(x)$, then it is clear that the premise of Lemma 3 is satisfied. So assume $\delta(x) > C_\phi + M_{u_2} + 1$, and hence $\delta'(x) = C_\phi + M_{u_2} + 1$. Then $\delta(x) + d_j - d_i > C_\phi + M_{u_2} + 1 + d_j - d_i \geq C_\phi + 1$ and $\delta'(x) + d_j - d_i = C_\phi + M_{u_2} + 1 + d_j - d_i \geq C_\phi + 1$. Thus, the premise of Lemma 3 holds. □

Theorem 4. *Path-checking for TPTL_b over infinite binary encoded data words is in* PSPACE.

Proof. Fix two finite data words u_1, u_2, a number $k \in \mathbb{N}$ and a TPTL-formula ψ, and let $w = u_1(u_2)^\omega_{+k}$. We show that one can decide in APTIME = PSPACE whether $w \models \psi$ holds. We first deal with the case $k > 0$ and later sketch the necessary adaptations for the (simpler) case $k = 0$. Without loss of generality, we further assume ψ to be in negation normal form, i.e., negations only appear in front of atomic propositions. Define $C := C_\psi$ and $M := M_{u_2}$ by (3) and (4).

The non-trivial cases in our alternating polynomial time algorithm are the ones for $\psi = \varphi_1 \mathsf{U} \varphi_2$ and $\psi = \varphi_1 \mathsf{R} \varphi_2$. Consider a position i and a register valuation δ. We have $(w, i, \delta) \models^{\mathsf{rel}} \varphi_1 \mathsf{U} \varphi_2$ iff $(w, i, \delta) \models^{\mathsf{rel}} \varphi_2$ or there exists $j > i$ such that $(w, j, \delta + d_j - d_i) \models^{\mathsf{rel}} \varphi_2$ and $\forall i \leq t < j : (w, t, \delta + d_t - d_i) \models^{\mathsf{rel}} \varphi_1$. Because w is an infinite word, j could be arbitrarily large. Our first goal is to derive a bound on j. Suppose that $0 \leq i \leq |u_1| + |u_2| - 1$; this is no restriction by Lemma 2. Define

$$m_\delta = \min\{\delta(x) \mid x \text{ is a register variable in } \psi\}, \tag{5}$$

$$m_1 = \max\{d_i - d_j \mid d_i \text{ and } d_j \text{ are data values in } u_1 u_2\} \text{ and} \tag{6}$$

$$m_2 = \min\{d \mid d \text{ is a data value in } u_2\}. \tag{7}$$

Let $n \geq 2$ be the minimal number such that $m_\delta + m_2 + (n-1)k - d_i \geq C + M + 1$, *i.e.* (here we assume $k > 0$),

$$n = \max\{2, \left\lceil \frac{C + M + 1 + d_i - m_\delta - m_2}{k} \right\rceil + 1\}. \tag{8}$$

If $h \geq |u_1| + (n-1)|u_2|$, then for every register variable x from ψ we have

$$\delta(x) + d_h - d_i \geq m_\delta + d_h - d_i \geq m_\delta + m_2 + (n-1)k - d_i \geq C + M + 1.$$

By Lemmas 2 and 4, for every $h \geq |u_1| + (n-1)|u_2|$ we have

$$(w, h, \delta + d_h - d_i) \models^{\mathsf{rel}} \varphi_2 \Leftrightarrow (w, h + |u_2|, \delta + d_{h+|u_2|} - d_i) \models^{\mathsf{rel}} \varphi_2.$$

Therefore, the position j witnessing $(w, j, \delta + d_j - d_i) \models^{\mathsf{rel}} \varphi_2$ can be bounded by $|u_1| + n|u_2|$. Similarly, we can get the same result for $\varphi_1 \mathsf{R} \varphi_2$.

We sketch an alternating Turing machine $\mathcal{A}$ that, given a TPTL_b-formula ψ and a data word w, has an accepting run iff $w \models \psi$. The machine $\mathcal{A}$ first computes and stores the value $C + M + 1$. In every configuration, $\mathcal{A}$ stores a triple (i, δ, φ), where i is a position in the data word, δ is a register valuation (with respect to the relative semantics), and φ is a subformula of ψ. By Lemma 2, we can restrict i to the interval $[0, |u_1| + |u_2|)$, and by Lemma 4, we can restrict the range of δ to the interval $[-m_1, \max\{m_1, C + M + 1\}]$. The machine $\mathcal{A}$ starts with the triple $(0, \tilde{0}, \psi)$, where $\tilde{0}(x) = 0$ for each register variable x. Then, $\mathcal{A}$ branches according to the following rules, where we define the function $\rho : \mathbb{N} \to [0, |u_1| + |u_2|)$ by $\rho(z) = z$ for $z < |u_1|$ and $\rho(z) = ((z - |u_1|) \bmod |u_2|) + |u_1|$ otherwise.

If φ is of the form p, $\neg p$, or $x \sim c$, then accept if $(w, i, \delta) \models^{\mathsf{rel}} \varphi$, and reject otherwise.

If $\varphi = \varphi_1 \wedge \varphi_2$, then branch universally to (i, δ, φ_1) and (i, δ, φ_2).

If $\varphi = \varphi_1 \vee \varphi_2$, then branch existentially to (i, δ, φ_1) and (i, δ, φ_2).

If $\varphi = x.\varphi_1$, then go to $(i, \delta[x \mapsto 0], \varphi_1)$.

If $\varphi = \mathsf{X}\varphi_1$, then continue as follows:

- If $i < |u_1| - 1$, then go to $(i + 1, \delta + d_{i+1} - d_i, \varphi_1)$.
- If $|u_1| - 1 \leq i < |u_1| + |u_2| - 1$, then go to $(i + 1, \delta', \varphi_1)$, where $\delta'(x) = \min\{\delta(x) + d_{i+1} - d_i, C + M + 1\}$.
- If $i = |u_1| + |u_2| - 1$, then proceed with the triple $(|u_1|, \delta', \varphi_1)$, where $\delta'(x) = \min\{\delta(x) + d_{|u_1|} + k - d_{|u_1|+|u_2|-1}, C + M + 1\}$.

If $\varphi = \varphi_1 \mathsf{U} \varphi_2$, then branch existentially to the following two alternatives.

- Go to (i, δ, φ).
- Compute the value n according to (5), (7), and (8), then branch existentially to each value $j \in (i, |u_1| + n|u_2|]$, and finally branch universally to each triple from $\{(\rho(t), \delta_t, \varphi_1) \mid i \leq t < j\} \cup \{(\rho(j), \delta_j, \varphi_2)\}$, where for all x:

$$\delta_j(x) = \begin{cases} \min\{\delta(x) + d_j - d_i, C + M + 1\} & \text{if } j \geq |u_1|, \\ \delta(x) + d_j - d_i & \text{otherwise,} \end{cases}$$

$$\delta_t(x) = \begin{cases} \min\{\delta(x) + d_t - d_i, C + M + 1\} & \text{if } t \geq |u_1|, \\ \delta(x) + d_t - d_i & \text{otherwise.} \end{cases}$$

If $\varphi = \varphi_1 \mathsf{R} \varphi_2$, then compute the value n according to (5), (7), and (8) and branch existentially to the following two alternatives:

- Branch universally to all triples from $\{(\rho(j), \delta_j, \varphi_2) \mid i \leq j \leq |u_1| + n|u_2|\}$, where

$$\delta_j(x) = \begin{cases} \min\{\delta(x) + d_j - d_i, C + M + 1\} & \text{if } j \geq |u_1|, \\ \delta(x) + d_j - d_i & \text{otherwise.} \end{cases}$$

– Branch existentially to each value $j \in [i, |u_1| + n|u_2|]$, and then branch universally to all triples from $\{(\rho(t), \delta_t, \varphi_2) \mid i \leq t \leq j\} \cup \{(\rho(j), \delta_j, \varphi_1)\}$, where for all x:

$$\delta_j(x) = \begin{cases} \min\{\delta(x) + d_j - d_i, C + M + 1\} & \text{if } j \geq |u_1|, \\ \delta(x) + d_j - d_i & \text{otherwise,} \end{cases}$$

$$\delta_t(x) = \begin{cases} \min\{\delta(x) + d_t - d_i, C + M + 1\} & \text{if } t \geq |u_1|, \\ \delta(x) + d_t - d_i & \text{otherwise.} \end{cases}$$

The machine $\mathcal{A}$ clearly works in polynomial time.

Let us briefly discuss the necessary changes for the case $k = 0$ (i.e., $w = u_1(u_2)^\omega$). The main difficulty in the above algorithm is to find the upper bound of the witnessing position j for the formulae $\varphi_1 \mathsf{U} \varphi_2$ and $\varphi_1 \mathsf{R} \varphi_2$. If $k = 0$, then it is easily seen that for every $i \geq |u_1|$, formula φ and valuation ν, $(w, i, \nu) \models \varphi$ iff $(w, i + |u_2|, \nu) \models \varphi$. We see at once that the witnessing position j can be bounded by $|u_1| + 2|u_2|$. It is straightforward to implement the necessary changes in the above algorithm. □

If all numbers are unary encoded and the number of register variables is fixed, then the alternating Turing-machine from the proof of Theorem 4 works in logarithmic space. Since $\mathsf{ALOGSPACE} = \mathsf{P}$, we obtain:

Theorem 5. *For every fixed $r \in \mathbb{N}$, path-checking for TPTL^r_u over infinite unary encoded data words is in P.*

Proof. In the algorithm from the proof of Theorem 4, if all numbers are given in unary, then the numbers $C + M + 1$, m_1, m_2 and n can be computed in logspace and are bounded polynomially in the input size. Moreover, a configuration triple (i, δ, φ) needs only logarithmic space: Clearly, the position $i \in [0, |u_1| + |u_2|)$ and the subformula φ only need logarithmic space. The valuation δ is an r-tuple over $[-m_1, \max\{m_1, C + M + 1\}]$ and hence needs logarithmic space too, since r is a constant. Hence, the alternating machine from the proof of Theorem 4 works in logarithmic space. The theorem follows, since $\mathsf{ALOGSPACE} = \mathsf{P}$. □

Actually, for finite data words, we obtain a polynomial time algorithm also for binary encoded data words (assuming again a fixed number of register variables):

Theorem 6. *For every fixed $r \in \mathbb{N}$, path-checking for TPTL^r_b over finite binary encoded data words is in P.*

Proof. Let the input data word w be of length n and let $d_1, \ldots, d_n$ be the data values appearing in w. Moreover, let $x_1, \ldots, x_r$ be the variables appearing in the input formula ψ. Then, we only have to consider the n^r many valuation mappings $\delta : \{x_1, \ldots, x_r\} \to \{d_1, \ldots, d_n\}$. For each of these mappings δ, every subformula φ of ψ, and every position i in w we check whether $(w, i, \delta) \models \varphi$. This information is computed bottom-up (with respect to the structure of φ) in the usual way. □

For infinite data words we have to reduce the number of register variables to one in order to get a polynomial time complexity for binary encoded numbers.

Theorem 7. *Path-checking for* TPTL_b^1 *over infinite binary encoded data words is in* P.

Proof. Given two finite data words u_1, u_2, a number $k \in \mathbb{N}$, and a formula ψ, let $w = u_1(u_2)^\omega_{+k}$, we show how to decide whether $w \models \psi$ in P. We say a TPTL-formula φ is closed if every occurrence of a register variable x in φ is within the scope of the corresponding freeze quantifier. It is straightforward to prove the following two claims:

Claim 1: If φ is a closed formula, then for any two valuations ν' and ν'', $(w, i, \nu') \models \varphi$ iff $(w, i, \nu'') \models \varphi$.

Claim 2: If φ is a closed formula and $i \geq |u_1|$, then for every valuation ν', $(w, i, \nu') \models \varphi$ iff $(w, i + |u_2|, \nu') \models \varphi$.

Given a valuation ν and formula φ, we define $S_{\varphi,\nu} = (S_0, S_1, \ldots, S_{|u_2|})$, where $S_0 \subseteq \{0, \ldots, |u_1| - 1\}$ and $S_h \subseteq \mathbb{N}\,(1 \leq h \leq |u_2|)$, such that, for $0 \leq i < |u_1|$, $i \in S_0$ iff $(w, i, \nu) \models \varphi$, and for each $1 \leq h \leq |u_2|$ and $j \geq 0$, $j \in S_h$ iff $(w, |u_1| + j \cdot |u_2| + h - 1, \nu) \models \varphi$, i.e., S_h contains all the numbers j such that φ holds in the h^{th} position of the j^{th} repetition of u_2.

We use $S^h_{\varphi,\nu}$ to denote the h^{th} $(0 \leq h \leq |u_2|)$ component S_h of $S_{\varphi,\nu}$. If φ is a closed formula, then, by Claim 1, $S_{\varphi,\nu'} = S_{\varphi,\nu''}$ for all valuations ν' and ν''. In this case, we skip the subscript ν and abbreviate $S_{\varphi,\nu}$ ($S^h_{\varphi,\nu}$) by S_φ (S^h_φ).

For $0 \leq i < |u_1u_2|$, ν_i denotes the valuation with $\nu_i(x) = d_i$. We will compute for every $0 \leq i < |u_1u_2|$ and every subformula φ of our input formula ψ the tuple S_{φ,ν_i}. Every set S^h_{φ,ν_i} will be represented by a union of polynomially many intervals that are pairwise disjoint, each of which is either a closed interval $[a, b]$ or a half closed interval $[a, +\infty)$, where $a, b \in \mathbb{N}$. Note that $w \models \psi$ if and only if $0 \in S^0_{\psi,\nu_0}$ (we assume without loss of generality that u_1 is not the empty word).

For a set $S \subseteq \mathbb{N}$, let $S - 1 = \{a - 1 \mid a \geq 1, a \in S\}$. We compute the tuples S_{φ,ν_i} bottom-up with respect to the structure of the formula φ as follows:

Case 1. φ is an atomic proposition. We can compute S^0_φ easily from u_1. For each $1 \leq h \leq |u_2|$, by Claim 2, S^h_φ is either $\mathbb{N}$ if $(u_2, h - 1) \models \varphi$ or $\emptyset$ otherwise.

Case 2. φ is a constraint formula $x \sim c$. We can compute S^0_{φ,ν_i} by checking whether $(u_1, i, \nu_i) \models x \sim c$ for each $0 \leq i < |u_1|$. For each S^h_{φ,ν_i} $(1 \leq h \leq |u_2|)$, note that the sequence of data values of w in positions $|u_1| + n \cdot |u_2| + h - 1\,(n \geq 0)$ is a non-decreasing arithmetic progression $d_{|u_1|+h-1}, d_{|u_1|+h-1} + k, d_{|u_1|+h-1} + 2k, \ldots$. Then, the interval borders for S^h_{φ,ν_i} can be easily computed. For example, suppose $\varphi = (x \geq c)$. We need to find the minimal number $n \geq 0$ such that $d_{|u_1|+h-1} + nk - \nu_i(x) \geq c$, which is

$$n = \max\{\left\lceil \frac{c + \nu_i(x) - d_{|u_1|+h-1}}{k} \right\rceil, 0\}.$$

Then, we set $S^h_{\varphi,\nu_i} = [n, +\infty)$. Similar calculation works for the other constraint formulae.

Case 3. $\varphi = \varphi_1 \wedge \varphi_2$. Then $S^h_{\varphi,\nu_i} = S^h_{\varphi_1,\nu_i} \cap S^h_{\varphi_2,\nu_i}$.

Case 4. $\varphi = \neg\varphi_1$. Then, $S^0_{\varphi,\nu_i} = \{0, \ldots, |u_1| - 1\} \setminus S^0_{\varphi_1,\nu_i}$ and $S^h_{\varphi,\nu_i} = \mathbb{N} \setminus S^j_{\varphi_1,\nu_i}$ for $1 \leq h \leq |u_2|$.

Case 5. $\varphi = x.\varphi_1$. Then $S^0_\varphi = \{i \mid 0 \leq i < |u_1|, i \in S^0_{\varphi_1,\nu_i}\}$ and, for each $1 \leq h \leq |u_2|$, $S^h_\varphi = \mathbb{N}$ if $|u_1| + h - 1 \in S^h_{\varphi_1,\nu_{|u_1|+h-1}}$, and $S^h_\varphi = \emptyset$ otherwise.

Case 6. $\varphi = \mathsf{X}\varphi_1$. Then $S^0_{\varphi,\nu_i} = (S^0_{\varphi_1,\nu_i} - 1) \cup \{|u_1| - 1 \mid 0 \in S^1_{\varphi_1,\nu_i}\}$ and, for each $1 \leq h < |u_2|$, $S^h_{\varphi,\nu_i} = S^{h+1}_{\varphi_1,\nu_i}$, and $S^{|u_2|}_{\varphi,\nu_i} = S^1_{\varphi_1,\nu_i} - 1$.

Case 7. $\varphi = \varphi_1 \mathsf{U} \varphi_2$. First, let $0 \leq j < |u_1|$. Then, we have $j \in S^0_{\varphi,\nu_i}$ iff one of the following two cases holds:

- There exists $s \in [j, |u_1|)$ such that $s \in S^0_{\varphi_2,\nu_i}$ and $[j, s) \subseteq S^0_{\varphi_1,\nu_i}$.
- $[j, |u_1|) \subseteq S^0_{\varphi_1,\nu_i}$ and there are $g_1 \geq 0$ and $1 \leq g_2 \leq |u_2|$ such that $[0, g_1 - 1] \subseteq \bigcap_{i=1}^h S^i_{\varphi_1,\nu_i}$ and $g_1 \in \bigcap_{i=1}^{g_2-1} S^i_{\varphi_1,\nu_i} \cap S^{g_2}_{\varphi_2,\nu_i}$.

Both cases can be easily checked in polynomial time.

In order to compute the sets S^h_{φ,ν_i} for $1 \leq h \leq |u_2|$, we initially set $R^1_1 = S^h_{\varphi_1,\nu_i}$ and $R^1_2 = S^h_{\varphi_2,\nu_i}$. For $1 \leq s < |u_2|$, if $h + s \leq |u_2|$, we set

$$R^{s+1}_1 = R^s_1 \cap S^{h+s}_{\varphi_1,\nu_i} \text{ and } R^{s+1}_2 = R^s_2 \cup (R^s_1 \cap S^{h+s}_{\varphi_2,\nu_i}).$$

If $h + s > |u_2|$, we set

$$R^{s+1}_1 = R^s_1 \cap (S^{h+s-|u_2|}_{\varphi_1,\nu_i} - 1) \text{ and } R^{s+1}_2 = R^s_2 \cup (R^s_1 \cap (S^{h+s-|u_2|}_{\varphi_2,\nu_i} - 1)).$$

Set $R = R^{|u_2|}_1 \setminus (S^h_{\varphi_2,\nu_i} \cup \cdots \cup S^{|u_2|}_{\varphi_2,\nu_i} \cup (S^1_{\varphi_2,\nu_i} - 1) \cup \cdots \cup (S^{h-1}_{\varphi_2,\nu_i} - 1))$ and

$$S^h_{\varphi,\nu_i} = R^{|u_2|}_2 \cup \bigcup \{[a, b] \mid [a, b] \text{ is a closed interval contained in } R \text{ and } b + 1 \in R^{|u_2|}_2\}.$$

The set $R^{|u_2|}_2$ contains all j such that (w.r.t. valuation ν_i) φ_2 holds in a position that is at most $|u_2| - 1$ positions after position $|u_1| + j \cdot |u_2| + h - 1$ and up to this position φ_1 holds. The set R contains all j such that (w.r.t. valuation ν_i) φ_1 holds in the interval of length $|u_2|$ starting at $|u_1| + j \cdot |u_2| + h - 1$ and φ_2 does not hold in this interval. If $[a, b]$ is a closed interval contained in R and $b + 1 \in R^{|u_2|}_2$, then it is easily seen that $\varphi_1 \mathsf{U} \varphi_2$ holds in each position $|u_1| + j \cdot |u_2| + h - 1$ for $j \in [a, b]$. Conversely, if $\varphi_1 \mathsf{U} \varphi_2$ holds in position $q = |u_1| + j \cdot |u_2| + h - 1$, then either φ_2 holds in a position that is at most $|u_2| - 1$ positions after position q and up to this position φ_1 holds (hence $j \in R^{|u_2|}_2$), or there exists $j' > j$ such that $j' \in R^{|u_2|}_2$ and φ_1 holds from position q up to position $|u_1| + j' \cdot |u_2| + h - 1$. In the latter case, we can choose j' minimal with this property. This implies $[j, j' - 1] \subseteq R$.

Finally, we need to show that each S^h_{φ,ν_i} contains only polynomially many intervals. It is sufficient to show that the number of all interval borders in the algorithm above is polynomially bounded. We use four kinds of set operations: union, intersection, complementation and subtraction. Union and intersection do not add any new interval borders. So we only need to consider complementation and subtraction of 1.

For a set $B \subseteq \mathbb{N}$ and $k \geq 1$, write $B - k = \{a - k \mid a \geq k, a \in B\}$ and $B + k = \{a + k \mid a \in B\}$. Let $\gamma_1, \ldots, \gamma_m$ be all constraint formulae $x \sim c$ that appear in ψ.

$$B_0 = \bigcup_{i=0}^{|u_1 u_2|-1} \bigcup_{j=1}^{m} \bigcup_{h=1}^{|u_2|} \{a \in \mathbb{N} \mid a \text{ is a border of an interval in } S^h_{\gamma_j,\nu_i}\}.$$

Thus, B_0 is the set of all interval borders that arise from constraint subformulae. W.l.o.g. we assume that $0 \in B_0$. For $n \geq 1$ define

$$B_n = B_{n-1} \cup (B_{n-1} - 1) \cup (B_{n-1} + 1).$$

By induction on n, one can show that

$$B_n = B_0 \cup \bigcup_{k=1}^{n} (B_0 - k) \cup (B_0 + k).$$

Hence $|B_n| \leq (2n + 1)|B_0|$. Subtraction decreases each interval border by 1, and complementation may decrease or increase an interval border by 1. Suppose that all interval borders are in B_n. If we do complementation or subtraction, then the new interval borders are in B_{n+1}. There are polynomially many complementation and subtraction operations and $|B_0|$ is polynomially bounded. So the number of all interval borders is polynomially bounded. □

3.3 Summary of the results for finite and infinite path-checking

Figure 1 collects our complexity results for path checking problems. The lower bounds all hold for pure strictly monotonic unary encoded data words. The upper bound hold for general (non-pure and non-monotonic) data words that are binary encoded, except for $\mathsf{TPTL}_u^{<\infty}$ (membership in P), where the data word has to be unary encoded.

4 Model-checking for deterministic one-counter machines.

A *one-counter machine* (OCM) $\mathcal{A}$ is a triple $\mathcal{A} = (Q, q_0, \Delta)$, where Q is a finite set of states, $q_0 \in Q$ is the initial state, and $\Delta = Q \times \{-1, 0, 1\} \times Q$ is the transition transition relation. A configuration is a pair $(q, n) \in Q \times \mathbb{N}$. For configurations (p, m) and (q, n) we write $(p, m) \vdash_{\mathcal{A}} (q, n)$ if one of the following three cases holds:

- $(p, -1, q) \in \Delta$ and $n = m - 1$
- $(p, 1, q) \in \Delta$ and $n = m + 1$
- $(p, 0, q) \in \Delta$ and $n = m = 0$

An infinite run of $\mathcal{A}$ is an infinite sequence

$$(q_0, 0) \vdash_{\mathcal{A}} (q_1, n_1) \vdash_{\mathcal{A}} (q_2, n_2) \vdash_{\mathcal{A}} (q_3, n_3) \vdash_{\mathcal{A}} \cdots .$$

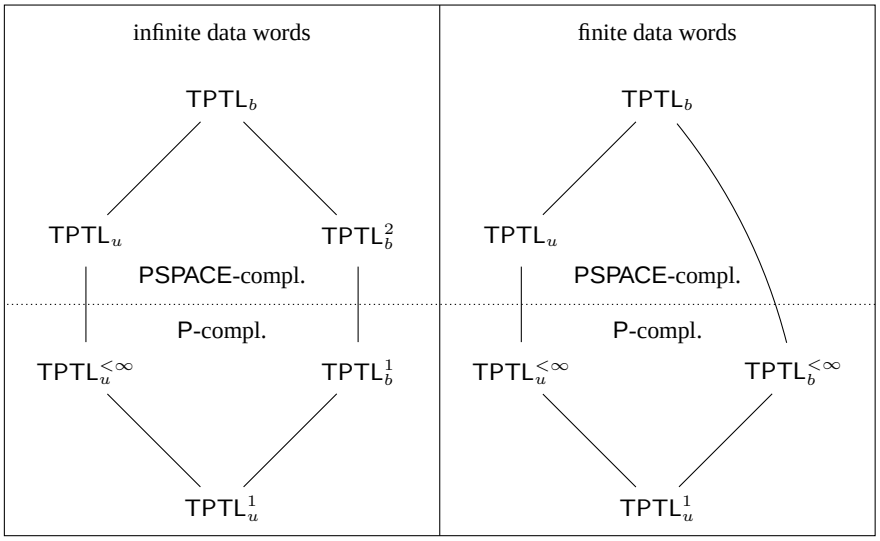

Fig. 1. Complexity results for path checking

A finite run of $\mathcal{A}$ is a finite sequence

$$(q_0, 0) \vdash_{\mathcal{A}} (q_1, n_1) \vdash_{\mathcal{A}} (q_2, n_2) \vdash_{\mathcal{A}} \cdots \vdash_{\mathcal{A}} (q_l, n_l)$$

such that there does not exist a configuration (q, n) with $(q_l, n_l) \vdash_{\mathcal{A}} (q, n)$. We identify this run with the finite data word $(\{q_0\}, 0)(\{q_1\}, n_1)(\{q_2\}, n_2) \cdots (\{q_l\}, n_l)$, and an infinite run is viewed as an infinite data word in the same way.

An OCM is *deterministic* (and called a DOCM) if for every state $p \in Q$, either there is exactly one outgoing transition (p, a, q) or there are exactly two outgoing transitions, which have to be of the form $(p, 0, q_1)$ and $(p, -1, q_2)$ for states $q_1, q_2 \in Q$. This implies that $\mathcal{A}$ has a unique run (either finite or infinite), which we denote with $\mathsf{run}(\mathcal{A})$, which is viewed as a data word as explained above.

Lemma 5. *For a given DOCM $\mathcal{A}$ one can check in logspace, whether $\mathsf{run}(\mathcal{A})$ is finite or infinite. Moreover, the following holds:*

- *If $\mathsf{run}(\mathcal{A})$ is finite, then the corresponding data word in unary encoding can be computed in logspace.*
- *If $\mathsf{run}(\mathcal{A})$ is infinite, then one can compute in logspace two unary encoded data words u_1 and u_2 and a unary encoded number k such that $\mathsf{run}(\mathcal{A}) = u_1(u_2)^{\omega}_{+k}$.*

Proof. In [9], the following statement was shown: If $\mathsf{run}(\mathcal{A})$ is infinite, then $\mathsf{run}(\mathcal{A}) = u_1(u_2)^{\omega}_{+k}$ with $k \leq |Q|$ and $|u_1 u_2| \leq |Q|^3$. Hence, in order to check whether $\mathsf{run}(\mathcal{A})$ is infinite, we have to simulate $\mathcal{A}$ for at most $|Q|^3$ many steps. Thereby we check, whether a configurations (q, n) is reached such that before, already a configuration (q, m) with $m < n$ has been reached. We store the current configuration with the counter value

in binary together with a step counter t, which only needs logarithmic space (since the counter and the step counter are bounded by $|Q|^3$). Each time, we produce a new configuration (q, n) (at step t), we have to check, whether we have seen a configuration (q, m) with $m < n$ before. Since we cannot store the whole sequence of configuration, we have to "freeze" the simulation of $\mathcal{A}$ at the configuration (q, n) and then start a new simulation from the initial configuration for at most t steps. Thereby, the current configuration is compared with (q, n).

In a similar way, we can produce the data word $\mathsf{run}(\mathcal{A})$ itself in logarithmic space. We only have to print out the current configuration. Internally, our machine stores counter values in binary encoding. Since we want the output data word to be unary encoded, we have to transform the binary encoded counter values into unary encoding, which can be done with a logspace machine. □

Let L be one of the logics considered in this paper. The *model-checking problem for* L *over DOCM* is defined as follows:

Input: A DOCM $\mathcal{A}$ and a formula $\varphi \in \mathsf{L}$.
Output: yes if $\mathsf{run}(\mathcal{A}) \models \varphi$, no otherwise.

Lemma 6. *Let* L *be one of the logics considered in this paper. The model-checking problem for* L *over DOCM is equivalent with respect to logspace reductions to the path-checking problem for* L *over infinite unary encoded data words.*

Proof. The reduction from the model-checking problem for L over DOCM to the path-checking problem for L over infinite unary encoded data words follows from Lemma 5. For the other direction take a unary encoded infinite data word $w = u_1(u_2)^{\omega}_{+k}$ and a formula $\psi \in \mathsf{L}$. Of course, w does not have to be of the form $\mathsf{run}(\mathcal{A})$ for a DOCM $\mathcal{A}$, since in a data word $\mathsf{run}(\mathcal{A})$ the data value can only change by at most 1 for neighboring positions. On the other hand, the latter can be easily enforced by inserting dummy positions in between the positions of w. Let $w' = v_1(v_2)^{\omega}_{+k}$ be the resulting data words. Then, we can easily construct in logspace a DOCM $\mathcal{A}$ such that the sequence of counter values produced by $\mathcal{A}$ is the sequence of data values of w'. Moreover, no state of $\mathcal{A}$ repeats among the first $|v_1 v_2| - 1$ many positions. It is then straightforward to construct a formula $\psi' \in \mathsf{L}$ such that $w \models \psi$ if and only if $\mathsf{run}(\mathcal{A}) \models \psi'$. □

By Lemma 6, the left diagram from Figure 1 also shows the complexity results for TPTL-model-checking over DOCM.

5 Open problems

An interesting open problem is the complexity status of model-checking MTL over non-monotonic data words. This problem is in P, but we do not know whether it is P-complete. For MTL over monotonic data words, model-checking was shown to be in $\mathsf{AC}^1(\mathsf{LogDCFL})$ [12].

References

1. Rajeev Alur and Thomas A. Henzinger. Real-Time Logics: Complexity and Expressiveness. *Inf. Comput.*, 104(1):35–77, 1993.
2. Rajeev Alur and Thomas A. Henzinger. A really temporal logic. *J. ACM*, 41(1):181–204, 1994.
3. Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.
4. Patricia Bouyer, Kim Guldstrand Larsen, and Nicolas Markey. Model checking one-clock priced timed automata. *Log. Meth. Comput. Sci.*, 4(2), 2008.
5. Daniel Bundala and Joël Ouaknine. On the complexity of temporal-logic path checking. In *Proc. ICALP 2014, Part II*, LNCS 8573, pages 86–97. Springer, 2014.
6. Claudia Carapelle, Shiguang Feng, Oliver Fernandez Gil, and Karin Quaas. On the Expressiveness of TPTL and MTL over ω-Data Words. In *Proc. AFL 2014*, volume 151 of *EPTCS*, pages 174–187, 2014.
7. Claudia Carapelle, Shiguang Feng, Oliver Fernandez Gil, and Karin Quaas. Satisfiability for MTL and TPTL over non-monotonic data words. In *Proc. LATA 2014*, LNCS 8370, pages 248–259. Springer, 2014.
8. Stéphane Demri and Ranko Lazić. LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Log.*, 10(3), 2009.
9. Stéphane Demri, Ranko Lazić, and Arnaud Sangnier. Model checking memoryful linear-time logics over one-counter automata. *Theor. Comput. Sci.*, 411(22-24):2298–2316, 2010.
10. Raymond Greenlaw, H. James Hoover, and Walter L. Ruzzo. *Limits to Parallel Computation: P-completeness Theory*. Oxford University Press, 1995.
11. R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.
12. Lars Kuhtz and Bernd Finkbeiner. Efficient parallel path checking for linear-time temporal logic with past and bounds. *Log. Meth. Comput. Sci.*, 8(4), 2012.
13. François Laroussinie, Nicolas Markey, and Ph. Schnoebelen. On model checking durational Kripke structures. In *Proc. FoSSaCS 2002*, LNCS 2303, pages 264–279. Springer, 2002.
14. Joël Ouaknine and James Worrell. On metric temporal logic and faulty Turing machines. In *Proc. FoSSaCS 2006*, LNCS 3921, pages 217–230. Springer, 2006.
15. Joël Ouaknine and James Worrell. On the decidability and complexity of metric temporal logic over finite words. *Log. Meth. Comput. Sci.*, 3(1), 2007.
16. Karin Quaas. Model checking metric temporal logic over automata with one counter. In *Proc. LATA 2013*, LNCS 7810, pages 468–479. Springer, 2013.
17. Stephen Travers. The complexity of membership problems for circuits over sets of integers. *Theor. Comput. Sci.*, 369(1):211–229, December 2006.