PATH CHECKING FOR MTL AND TPTL OVER DATA WORDS

SHIGUANG FENG[♣], MARKUS LOHREY, AND KARIN QUAAS[♠]

Institut für Informatik, Universität Leipzig, Germany

Department für Elektrotechnik und Informatik, Universität Siegen, Germany

Institut für Informatik, Universität Leipzig, Germany

Abstract. Metric temporal logic (MTL) and timed propositional temporal logic (TPTL) are quantitative extensions of linear temporal logic, which are prominent and widely used in the verification of real-timed systems. It was recently shown that the path-checking problem for MTL, when evaluated over finite timed words, is in the parallel complexity class NC. In this paper, we derive precise complexity results for the path-checking problem for MTL and TPTL when evaluated over infinite data words over the non-negative integers. Such words may be seen as the behaviours of one-counter machines. For this setting, we give a complete analysis of the complexity of the path-checking problem depending on the number of register variables and the encoding of constraint numbers (unary or binary). As the two main results, we prove that the path-checking problem for MTL is P-complete, whereas the path-checking problem for TPTL is PSPACE-complete. The results yield the precise complexity of model checking deterministic one-counter machines against formulas of MTL and TPTL.

1. Introduction

Linear temporal logic (LTL) is nowadays one of the main logical formalisms for describing system behaviour. Triggered by real-time applications, various timed extensions of LTL have been invented. Two of the most prominent examples are MTL (metric temporal logic) [19] and TPTL (timed propositional temporal logic) [2]. In MTL, the temporal until modality (U) is annotated with time intervals. For instance, the formula $p\,\mathrm{U}_{[2,3)}\,q$ holds at 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 [6] 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

1998 ACM Subject Classification: F.4 MATHEMATICAL LOGIC AND FORMAL LANGUAGES - F.4.1 Mathematical Logic - Temporal Logic;

Key words and phrases: Metric Temporal Logic, Timed Propositional Temporal Logic, Freeze LTL, Path Checking Problem, Deterministic One Counter Machines, Data Words.

An extended abstract of this paper has been published at *Developments in Language Theory* (DLT), 2015.

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

[♠]The author is supported by DFG, projects QU 316/1-1 and QU 316/1-2.

DOI:10.2168/LMCS-???

	Satisfiability	**Model Checking**
Real-Timed Words		
MTL finite words	$\mathcal{F}_{\omega^\omega}$-complete [26, 28]	$\mathcal{F}_{\omega^\omega}$-complete [26, 28]
MTL infinite words	undecidable [25]	undecidable [25]
TPTL	undecidable [1]	undecidable [1]
Discrete Timed Words		
MTL	EXPSPACE-complete [1]	EXPSPACE-complete [1]
TPTL	EXPSPACE-complete [2]	EXPSPACE-complete [2]
Data Words		
MTL	undecidable [7]	undecidable [27]
TPTL	undecidable [7]	undecidable [27]
FreezeLTL	undecidable [8]	undecidable [9]
FreezeLTL^1 finite words	$\mathcal{F}_\omega$-complete [8, 12]	undecidable [9]
FreezeLTL^1 infinite words	undecidable [8]	undecidable [9]

Table 1: Complexity results for the satisfiability problem and the model-checking problem for the logics MTL, TPTL, FreezeLTL, and its one-variable fragment FreezeLTL^1. By *real-timed words* (*discrete timed words*, respectively) we mean timed words with timestamps in the non-negative reals (non-negative integers, respectively). Model checking for timed words is done for timed automata, and model checking for data words is done for one-counter machines. We do not distinguish between finite and infinite words if this does not influence the complexity.

be compared with the current time value. For instance, the above MTL formula $p\,\mathrm{U}_{[2,3)}\,q$ is equivalent to the TPTL formula $x.(p\,\mathrm{U}\,(q \wedge 2 \leq x < 3))$. 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 $[2, 3)$. Formulas in MTL and TPTL are evaluated over finite or infinite *timed words* of the form $(a_0, t_0)(a_1, t_1)\ldots$, where $t_0, t_1, \ldots$ is a monotonically increasing sequence of real-valued *timestamps*, and the a_i are actions that take place at timestamp t_i. In the context of formal verification of timed automata, *satisfiability* and *model checking* for MTL and TPTL have been studied intensively in the past, see Table 1 for a summary of the most important results.

The freeze mechanism from TPTL has also received attention in connection with *data words*. Data words generalize timed words and are of the form $(a_0, d_0)(a_1, d_1)\ldots$, where the data values $d_0, d_1, \ldots$ come from an arbitrary infinite data domain. Following [9], we regard data words as computation paths of *one-counter machines*, *i.e.*, we study data words over the domain of the non-negative integers. Note that, different to timed words, the sequence of data values $d_0, d_1, \ldots$ does not have to be monotonically increasing. Applications for data words 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).

For reasoning about data words, besides MTL and TPTL in [7], a strict fragment of TPTL called FreezeLTL is studied in [8, 9]. With formulas in FreezeLTL one can only test *equality* of the value of a variable x and the data value at the current position in a data word. Table 1 shows that for all these logics, model checking one-counter machines and the satisfiability problem are undecidable, with the exception of the satisfiability problem for the one-variable fragment of FreezeLTL when evaluated over finite data words. The

situation changes if one considers model checking of *deterministic* one-counter machines: in this case, FreezeLTL-model checking is PSPACE-complete [9]. Note that model checking of deterministic one-counter machines is a special case of the following *path-checking problem*: given a data word w and some formula φ, does w satisfy φ?

The path-checking problem plays a key role in *run-time verification* [13, 24, 23], specifically in *offline monitoring*, where the satisfaction of a specification formula is tested only for an individual single computation path of the observed system. Run-time verification may be the only practical alternative to check that certain temporal properties hold, in situations where the source code of the system under consideration is not available, or where model checking of the system is unfeasible if not undecidable, as it is the case for one-counter machines.

For LTL and periodic words without data values, it was shown in [21] that the path-checking problem can be solved using an efficient parallel algorithm. More precisely, the problem belongs to $\mathrm{AC}^1(\mathrm{LogDCFL})$, a subclass of NC. This result solved a long standing open problem; the best known lower bound is NC^1, arising from NC^1-hardness of evaluating Boolean expressions. The $\mathrm{AC}^1(\mathrm{LogDCFL})$-upper complexity bound was later even established for the path-checking problem for MTL over finite timed words [5].

In this paper, we continue the study of the path-checking problem started in [9] for TPTL and data words over the non-negative integers, *i.e.*, we regard data words as the behaviours of one-counter machines. Note that TPTL is strictly more expressive than FreezeLTL in that, in contrast to the latter, with TPTL one can express that the difference of the current data value in a data word and the value stored in a register belongs to a certain interval. We further investigate path checking for MTL, because, as it was recently proved [6], MTL is strictly less expressive than TPTL in the setting of data words over the non-negative integers. More specifically, we investigate the path-checking problems for TPTL over data words that can be either finite or infinite periodic; in the latter case the data word is specified by an initial part, a period, and an offset number, which is added to the data values in the period after each repetition of the period.

We show that the $\mathrm{AC}^1(\mathrm{LogDCFL})$-membership result of [5] for path checking of MTL over finite timed words is quite sharp in the following sense: path checking for MTL over (finite or infinite) data words as well as path checking for the one-variable fragment of TPTL evaluated over monotonic (finite or infinite) data words is P-complete. Moreover, path checking for TPTL (with an arbitrary number of register variables) over finite as well as infinite periodic data words becomes PSPACE-complete. 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]). These results yield a rather complete picture on the complexity of path checking for MTL and TPTL, see Figure 7 at the end of the paper. We also show that PSPACE-membership for the path-checking problem for TPTL still holds if data words are specified succinctly by so called straight-line programs, which have the ability to generate data words of exponential length. For ordinary LTL it was shown in [24] that path checking is PSPACE-complete if paths are represented by straight-line programs. Our result extends the PSPACE upper bound from [24].

Since the infinite data word produced by a deterministic one-counter machines is periodic, we can transfer all complexity results for the infinite periodic case to deterministic one-counter machines, assuming that update numbers are encoded in unary notation. The

third author proved recently that model checking for TPTL over deterministic one-counter machines is decidable [27], but the complexity remained open. Our results show that the precise complexity is PSPACE-complete. This also generalizes the PSPACE-completeness result for FreezeLTL over deterministic one-counter machines [9]. Our PSPACE upper bound for TPTL model checking over deterministic one-counter machines also holds when the update numbers of the one-counter machines are given in binary notation. These are called *succinct one-counter automata* in [15, 17], where the complexity of reachability problems and model-checking problems for various temporal logics over succinct nondeterministic one-counter automata were studied.

An extended abstract of this paper without full proofs appeared in [11].

2. Temporal Logics over Data Words

Data Words. Let $\mathbb{P}$ be a finite set of *atomic propositions*. A *word* over $\mathbb{P}$ is a finite or infinite sequence $P_0P_1P_2\ldots$, where $P_i \subseteq \mathbb{P}$ for all $i \in \mathbb{N}$. If $P_i = \{p_i\}$ is a singleton set for every position i in the word, then we may also write $p_0p_1p_2\ldots$. A *data word* over $\mathbb{P}$ is a finite or infinite sequence $(P_0, d_0)(P_1, d_1)(P_2, d_2)\ldots$, where $(P_i, d_i) \in (2^{\mathbb{P}} \times \mathbb{N})$ for all $i \in \mathbb{N}$. A data word is *monotonic* (*strictly monotonic*), if $d_i \leq d_{i+1}$ ($d_i < d_{i+1}$) for all $i \in \mathbb{N}$. A *pure* data word is a finite or infinite sequence $d_0d_1d_2\ldots$ of natural numbers; it can be identified with the data word $(\emptyset, d_0)(\emptyset, d_1)(\emptyset, d_2)\ldots$. We use $(2^{\mathbb{P}} \times \mathbb{N})^*$ and $(2^{\mathbb{P}} \times \mathbb{N})^\omega$, respectively, to denote the set of finite and infinite, respectively, data words over $\mathbb{P}$. We use $|w|$ to denote the *length* of a data word w, *i.e.*, the number of all pairs (P_i, d_i) in w. If w is infinite, then $|w| = +\infty$.

Let $w = (P_0, d_0)(P_1, d_1)\ldots$ be a data word, and let $i \in \{0, \ldots, |w|\}$ be a position in w. We define $w[i{:}]$ to be the suffix of w starting in position i, *i.e.*, $w[i{:}] := (P_i, d_i)(P_{i+1}, d_{i+1})\ldots$. For an integer $k \in \mathbb{Z}$ satisfying $d_i + k \geq 0$ for all $i \geq 0$, we define the data word $w_{+k} := (P_0, d_0 + k)(P_1, d_1 + k)\ldots$.

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. For finite data words u_1, u_2 and $k \in \mathbb{N}$, we define

$$u_1(u_2)^\omega_{+k} := u_1u_2(u_2)_{+k}(u_2)_{+2k}(u_2)_{+3k}\ldots$$

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.

Linear Temporal Logic. Given a finite set $\mathbb{P}$ of propositions, the set of formulas of linear temporal logic (LTL, for short) is built up from $\mathbb{P}$ by Boolean connectives and the *until* modality U using the following grammar:

$$\varphi ::= \text{true} \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \mathrm{U} \varphi$$

where $p \in \mathbb{P}$. Formulas of LTL are interpreted over *words* over $\mathbb{P}$. Let $w = P_0\, P_1\, P_2\ldots$ be a word over $\mathbb{P}$, and let i be a position in w. We define the *satisfaction relation for* LTL inductively as follows:

- $(w, i) \models_{\mathrm{LTL}} \text{true}$.
- $(w, i) \models_{\mathrm{LTL}} p$ if, and only if, $p \in P_i$.
- $(w, i) \models_{\mathrm{LTL}} \neg\varphi$ if, and only if, $(w, i) \not\models_{\mathrm{LTL}} \varphi$.
- $(w, i) \models_{\mathrm{LTL}} \varphi_1 \wedge \varphi_2$ if, and only if, $(w, i) \models_{\mathrm{LTL}} \varphi_1$ and $(w, i) \models_{\mathrm{LTL}} \varphi_2$.

- $(w,i) \models_{\mathrm{LTL}} \varphi_1 \mathrm{U} \varphi_2$ if, and only if, there exists a position $j > i$ in w such that $(w,j) \models_{\mathrm{LTL}} \varphi_2$, and $(w,k) \models_{\mathrm{LTL}} \varphi_1$ for all positions k with $i < k < j$.

We say that a word *satisfies* an LTL formula φ, written $w \models \varphi$, if $(w,0) \models \varphi$.

We use the following standard abbreviations:

$$\begin{aligned} \mathrm{false} &:= \neg\mathrm{true} & \mathrm{F}\varphi &:= \mathrm{true}\mathrm{U}\varphi \\ \varphi_1 \vee \varphi_2 &:= \neg(\neg\varphi_1 \wedge \neg\varphi_2) & \mathrm{G}\varphi &:= \neg\mathrm{F}\neg\varphi \\ \varphi_1 \rightarrow \varphi_2 &:= \neg\varphi_1 \vee \varphi_2 & \mathrm{X}\varphi &:= \mathrm{false}\mathrm{U}\,\varphi \\ \varphi_1 \,\mathrm{R}\, \varphi_2 &:= \neg(\neg\varphi_1 \mathrm{U} \neg\varphi_2) & \mathrm{X}^m\varphi &:= \underbrace{\mathrm{X}\ldots\mathrm{X}}_{m}\varphi \end{aligned}$$

The modalities X (*next*), F (*finally*) and G (*globally*), respectively, are *unary* operators, which refer to the *next* position, *some* position in the future and *all* positions in the future, respectively. The binary modality R is the *release* operator, which is useful to transform a formula into an equivalent *negation normal form*, where the negation operator ($\neg$) may only be applied to true or to propositions.

Metric Temporal Logic and Timed Propositional Temporal Logic. Metric Temporal Logic, MTL for short, extends LTL in that the until modality U may be annotated with an interval over $\mathbb{Z}$. More precisely, the set of MTL formulas is defined by the following grammar:

$$\varphi ::= \mathrm{true} \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \mathrm{U}_I \varphi$$

where $p \in \mathbb{P}$ and $I \subseteq \mathbb{Z}$ is an interval with endpoints in $\mathbb{Z} \cup \{-\infty, +\infty\}$.

Formulas in MTL are interpreted over data words. Let $w = (P_0, d_0)(P_1, d_1)\ldots$ be a data word over $\mathbb{P}$, and let i be a position in w. We define the *satisfaction relation for* MTL, denoted by $\models_{\mathrm{MTL}}$, inductively as follows (we omit the obvious cases for $\neg$ and $\wedge$):

- $(w,i) \models_{\mathrm{MTL}} p$ if, and only if, $p \in P_i$.
- $(w,i) \models_{\mathrm{MTL}} \varphi_1 \mathrm{U}_I \varphi_2$ if, and only if, there exists a position $j > i$ in w such that $(w,j) \models_{\mathrm{MTL}} \varphi_2$, $d_j - d_i \in I$, and $(w,k) \models_{\mathrm{MTL}} \varphi_1$ for all positions k with $i < k < j$.

We say that a data word *satisfies* an MTL formula φ, written $w \models_{\mathrm{MTL}} \varphi$, if $(w,0) \models_{\mathrm{MTL}} \varphi$. We use the same syntactic abbreviations as for LTL, where every temporal operator is annotated with an interval, *e.g.*, $\mathrm{F}_I\varphi := \mathrm{true}\mathrm{U}_I\varphi$ and $\mathrm{X}_I\varphi := \mathrm{false}\mathrm{U}_I\varphi$. For annotating the temporal operators, we may also use pseudo-arithmetic expressions of the form $x \sim c$, where $\sim \in \{<, \leq, =, \geq, >\}$ and $c \in \mathbb{Z}$. For instance, we may write $\mathrm{F}_{=2}p$ as abbreviation for $\mathrm{F}_{[2,2]}p$. If $I = \mathbb{Z}$, then we may omit the annotation I on U_I.

Some of our results for lower bounds already hold for fragments of MTL, which we explain in the following. We write MTL(F, X) to denote the *unary* fragment of MTL in which the only temporal modalities allowed are F and X, and we write MTL(F) to denote the fragment of MTL in which F is the only allowed temporal modality. We write PureMTL (PureMTL(F, X), respectively) to denote the set of MTL formulas (MTL(F, X) formulas, respectively), in which no propositional variable from $\mathbb{P}$ is used.

Next, we define Timed Propositional Temporal Logic, TPTL for short. Let V be a countable set of *register variables*. Formulas of TPTL are built by the following grammar:

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

where $p \in \mathbb{P}$, $x \in V$, $c \in \mathbb{Z}$, and $\sim \in \{<, \leq, =, \geq, >\}$.

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 $w = (P_0, d_0)(P_1, d_1)\ldots$ be a data word over $\mathbb{P}$, let ν be a register valuation, and let i be a position in w. The satisfaction relation for TPTL, denoted by $\models_{\text{TPTL}}$, is defined inductively in the obvious way; we only give the definitions for the new formulas:

- $(w, i, \nu) \models_{\text{TPTL}} x \sim c$ if, and only if, $d_i - \nu(x) \sim c$.
- $(w, i, \nu) \models_{\text{TPTL}} x.\varphi$ if, and only if, $(w, i, \nu[x \mapsto d_i]) \models_{\text{TPTL}} \varphi$.

Intuitively, $x.\varphi$, means that x is *updated* to the data value at the current position of the data word, and $x \sim c$ means that, compared to the last time that x was updated, the data value has changed by at least (at most, or exactly, respectively) by c. We say that a data word w satisfies a TPTL formula φ, written $w \models \varphi$, if $(w, 0, \mathbf{d_0}) \models_{\text{TPTL}} \varphi$, where $\mathbf{d_0}$ denotes the valuation that maps all register variables to the initial data value d_0 of the data word w.

We use the same syntactic abbreviations as for LTL. We define the fragments TPTL(F, X), TPTL(F), and PureTPTL like the corresponding fragments of MTL. Additionally, we define FreezeLTL to be the subset of TPTL formulas φ where every subformula $x \sim c$ of φ must be of the form $x = 0$. Further, for every $r \in \mathbb{N}$, we use TPTL^r to denote the fragment of TPTL in which at most r different register variables occur; similarly for other TPTL-fragments.

For complexity considerations, it makes a difference whether the numbers c in constraints $x \sim c$ are binary or unary encoded, and similarly for the interval borders in MTL. We annotate our logics $\mathbb{L}$ by an index u or b, *i.e.*, we write $\mathbb{L}_{\text{u}}$ respectively $\mathbb{L}_{\text{b}}$, to emphasize that numbers are encoded in unary (resp., binary) notation. The *length* of a (TPTL or MTL) formula ψ, denoted by $|\psi|$, is the number of symbols occurring in ψ.

In the rest of the paper, 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 a logic $\mathbb{L}$ and a class of data words $\mathbb{C}$, we consider the *path-checking problem for* $\mathbb{L}$ *over* $\mathbb{C}$: given some data word $w \in \mathbb{C}$ and some formula $\varphi \in \mathbb{L}$, does $w \models_{\mathbb{L}} \varphi$ hold?

3. Background from Complexity Theory

We assume that the reader is familiar with the complexity classes P (deterministic polynomial time) and PSPACE (polynomial space), more background can be found for instance in [3]. Recall that, by Savitch's theorem, nondeterministic polynomial space is equal to deterministic polynomial space. All completeness results in this paper refer to logspace reductions.

We will make use of well known characterizations of P and PSPACE in terms of alternating Turing machines. An *alternating Turing machine* is a nondeterministic 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 by

an alternating Turing machine in polynomial time (APTIME) is equal to PSPACE, and that the class of all languages that can be accepted by an alternating Turing machine in logarithmic space (ALOGSPACE) is equal to P.

A couple of times we will mention the complexity class $\mathrm{AC}^1(\mathrm{LogDCFL})$. For completeness, we present the definition: the class $\mathsf{LogDCFL}$ is the class of all languages that are logspace reducible to a deterministic context-free language. Then $\mathrm{AC}^1(\mathrm{LogDCFL})$ is the class of all problems that can be solved by a logspace-uniform circuit family of polynomial size and logarithmic depth, where in addition to ordinary Boolean gates (NOT, AND, OR) also oracle gates for problems from $\mathsf{LogDCFL}$ can be used. More details can be found in [21]. The class $\mathrm{AC}^1(\mathrm{LogDCFL})$ belongs to NC (the class of all problems that can be solved on a parallel random-access machine (PRAM) in polylogarithmic time with polynomially many processors), which, in turn, is contained in P.

4. Upper Complexity Bounds

In this section we prove the upper complexity bounds for the path-checking problems. We distinguish between (i) unary or binary encoded data words, and (ii) finite and infinite data words.

4.1. Polynomial Space Upper Bound for TPTL.

For the most general path-checking problem ($\mathrm{TPTL_b}$ over infinite binary encoded data words) we can devise an alternating polynomial time (and hence a polynomial space) algorithm by constructing an alternating Turing machine that, given a $\mathrm{TPTL_b}$ formula φ and an infinite binary encoded data word w, has an accepting run if, and only if, $w \models_{\mathrm{TPTL}} \varphi$. The main technical difficulty is to bound the position in the infinite data word and the values of the register valuation, so that they can be stored in polynomial space.

Theorem 4.1. *Path checking for* $\mathrm{TPTL_b}$ *over infinite binary encoded data words is in* PSPACE.

Before we give the proof of Theorem 4.1, we introduce some helpful notions and prove some lemmas.

Relative semantics. 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 introduce a *relative satisfaction relation* for TPTL, denoted by $\models^{\mathsf{rel}}$, as follows. For Boolean formulas, $\models^{\mathsf{rel}}$ is defined like $\models_{\mathrm{TPTL}}$. For the other operators we define:

- $(w, i, \delta) \models^{\mathsf{rel}} \varphi_1 \mathrm{U} \varphi_2$ if, and only if, there exists a position $j > i$ in w such that $(w, j, \delta + (d_j - d_i)) \models^{\mathsf{rel}} \varphi_2$, and $(w, k, \delta + (d_t - d_i)) \models^{\mathsf{rel}} \varphi_1$ for all positions k with $i < k < j$
- $(w, i, \delta) \models^{\mathsf{rel}} x \sim c$ 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 the data word w satisfies the formula φ under the relative semantics, written $w \models^{\mathsf{rel}} \varphi$, if $(w, 0, \mathbf{0}) \models^{\mathsf{rel}} \varphi$, where $\mathbf{0}$ denotes the valuation function that maps all register variables to 0.

The main advantage of the relative semantics is the following: under the normal TPTL semantics, a constraint $x \sim c$ is true under a valuation ν at a position with data value d, if $d - \nu(x) \sim c$ holds. In contrast, under the relative semantics, a constraint $x \sim c$ is true

under a valuation δ, if $\delta(x) \sim c$ holds, *i.e.*, the data value at the current position is not important. The following lemma implies that $w \models_{\text{TPTL}} \varphi$ if, and only if, $w \models^{\mathsf{rel}} \varphi$, which allows us to work with the relative semantics.

Lemma 4.2. *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, $(w, i, \nu) \models_{\text{TPTL}} \varphi$ if, and only if, $(w, i, \delta) \models^{\mathsf{rel}} \varphi$.*

Proof. The proof is by induction on the structure of the formula φ. We only consider the non-trivial cases.

Base case. Assume $\varphi = x \sim c$.

$$\begin{aligned}(w, i, \nu) \models_{\text{TPTL}} \varphi &\Leftrightarrow d_i - \nu(x) \sim c \quad \text{(definition } \models_{\text{TPTL}}\text{)} \\ &\Leftrightarrow d_i - d_i + \delta(x) \sim c \quad \text{(assumption)} \\ &\Leftrightarrow (w, i, \delta) \models^{\mathsf{rel}} \varphi \quad \text{(definition } \models^{\mathsf{rel}}\text{)}.\end{aligned}$$

Induction step.

- Assume $\varphi = \varphi_1 \mathrm{U} \varphi_2$. We have

  $(w, i, \nu) \models_{\text{TPTL}} \varphi_1 \mathrm{U} \varphi_2$

  $\Leftrightarrow$ there exists $i < j < |w|.(w, j, \nu) \models_{\text{TPTL}} \varphi_2$, and

  $(w, t, \nu) \models_{\text{TPTL}} \varphi_1$ for all $i < t < j$ (definition $\models_{\text{TPTL}}$)

  $\Leftrightarrow$ there exists $i < j < |w|.(w, j, \delta + (d_j - d_i)) \models^{\mathsf{rel}} \varphi_2$, and

  $(w, t, \delta + (d_t - d_i)) \models^{\mathsf{rel}} \varphi_1$ for all $i < t < j$ (induction hypothesis)

  $\Leftrightarrow (w, i, \delta) \models^{\mathsf{rel}} \varphi_1 \mathrm{U} \varphi_2$ (definition $\models^{\mathsf{rel}}$)

- Assume $\varphi = x.\varphi_1$. By definition

  $$(w, i, \nu) \models_{\text{TPTL}} x.\varphi_1 \Leftrightarrow (w, i, \nu[x \mapsto d_i]) \models_{\text{TPTL}} \varphi_1,$$

  and

  $$(w, i, \delta) \models^{\mathsf{rel}} x.\varphi_1 \Leftrightarrow (w, i, \delta[x \mapsto 0]) \models^{\mathsf{rel}} \varphi_1.$$

  Clearly,

  $$(\delta[x \mapsto 0])(x) = 0 = d_i - d_i = d_i - (\nu[x \mapsto d_i])(x),$$

  and

  $$(\delta[x \mapsto 0])(y) = \delta(y) = d_i - \nu(y) = d_i - (\nu[x \mapsto d_i])(y)$$

  for every register variable $y \neq x$. The result follows by induction hypothesis on $\delta[x \mapsto 0]$ and $\nu[x \mapsto d_i]$.

□

For the next three lemmas, we let $w = u_1(u_2)^{\omega}_{+k}$, where u_1 and u_2 are finite data words, and $k \geq 0$. Further assume $i \geq |u_1|$, and let ψ be a TPTL formula.

Lemma 4.3. *For all register valuations δ, $(w, i, \delta) \models^{\mathsf{rel}} \psi$ if, and only if, $(w, i + |u_2|, \delta) \models^{\mathsf{rel}} \psi$.*

Proof. Let δ be a register valuation. Define $\nu = d_i - \delta$ and $\nu' = \nu + k$. Lemma 4.2 yields

$$(w, i, \delta) \models^{\mathsf{rel}} \psi \Leftrightarrow (w, i, \nu) \models_{\text{TPTL}} \psi,$$

and, together with $d_{i+|u_2|} = d_i + k$,

$$(w, i + |u_2|, \delta) \models^{\mathsf{rel}} \psi \Leftrightarrow (w, i + |u_2|, \nu') \models_{\text{TPTL}} \psi.$$

We prove that $(w,i,\nu) \models_{\mathrm{TPTL}} \psi \Leftrightarrow (w, i+|u_2|, \nu') \models_{\mathrm{TPTL}} \psi$; 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 $(w_{\geq i}, 0, \nu) \models_{\mathrm{TPTL}} \psi \Leftrightarrow ((w_{\geq i})_{+k}, 0, \nu') \models_{\mathrm{TPTL}} \psi$. This can be shown by a simple induction on the structure of ψ. □

Lemma 4.4. *Let δ_1, δ_2 be two register valuations. If for every $j \geq i$ and every subformula $x \sim c$ of ψ we have*

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

then we also have $(w,i,\delta_1) \models^{\mathsf{rel}} \phi \;\Leftrightarrow\; (w,i,\delta_2) \models^{\mathsf{rel}} \phi$.

Proof. The proof is by induction on the structure of ψ. We only prove the non-trivial cases.
Base case. Let $\psi = x \sim c$. Clearly, $\delta_k = \delta_k + (d_i - d_i)$ for $k = 1, 2$. Hence

$$\begin{aligned}(w,i,\delta_1) \models^{\mathsf{rel}} x \sim c &\Leftrightarrow (w, i, \delta_1 + (d_i - d_i)) \models^{\mathsf{rel}} x \sim c \\ &\Leftrightarrow (w, i, \delta_2 + (d_i - d_i)) \models^{\mathsf{rel}} x \sim c \Leftrightarrow (w,i,\delta_2) \models^{\mathsf{rel}} x \sim c.\end{aligned}$$

Induction step. Assume $\psi = x.\psi_1$. By definition,

$$(w,i,\delta_k) \models^{\mathsf{rel}} x.\psi_1 \Leftrightarrow (w,i,\delta_k[x \mapsto 0]) \models^{\mathsf{rel}} \psi_1$$

for $k = 1, 2$. Note that the valuations $\delta_1[x \mapsto 0]$ and $\delta_2[x \mapsto 0]$ satisfy the premise of the lemma. The result follows by induction hypothesis. □

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

$$C_\varphi = \max\{c \in \mathbb{Z} \mid x \sim c \text{ is a subformula of } \varphi\} \tag{4.1}$$

$$M_v = \max\{d_i - d_j \mid d_i \text{ and } d_j \text{ are data values in } v\} \geq 0 \tag{4.2}$$

We may always assume that $C_\varphi \geq 0$ (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 d_i and d_j denote the data values at positions i and j, respectively.

Lemma 4.5. *Let δ be a register valuation and define δ' by $\delta'(x) = \min\{\delta(x), C_\psi + M_{u_2} + 1\}$ for all x. For every subformula θ of ψ, we have $(w,i,\delta) \models^{\mathsf{rel}} \theta$ if, and only if, $(w,i,\delta') \models^{\mathsf{rel}} \theta$.*

Proof. We prove that the premise of Lemma 4.4 holds for $\delta_1 = \delta$ and $\delta_2 = \delta'$. The claim then follows from Lemma 4.4. So let $j \geq i$, and let $x \sim c$ be a subformula of ψ. If $\delta(x) \leq C + M_{u_2} + 1$, and hence $\delta'(x) = \delta(x)$, then it is clear that the premise of Lemma 4.4 is satisfied. So assume $\delta(x) > C + M_{u_2} + 1$, and hence $\delta'(x) = C + M_{u_2} + 1$. Then

$$\delta(x) + d_j - d_i > C + M_{u_2} + 1 + d_j - d_i \geq C + 1$$

and

$$\delta'(x) + d_j - d_i = C + M_{u_2} + 1 + d_j - d_i \geq C + 1.$$

Thus, the premise of Lemma 4.4 holds. □

Proof of Theorem 4.1. 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 $w \models_{\mathrm{TPTL}} \psi$ 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. Define $C := C_\psi$ and $M := M_{u_2}$ by (4.1) and (4.2).

The non-trivial cases in our alternating polynomial time algorithm are the ones for $\psi = \varphi_1 \mathrm{U} \varphi_2$ and $\psi = \varphi_1 \mathrm{R} \varphi_2$. Consider a position i and a register valuation δ. We have $(w, i, \delta) \models^{\mathsf{rel}} \varphi_1 \mathrm{U} \varphi_2$ if, and only if, there exists some position $j > i$ in w such that $(w, j, \delta + (d_j - d_i) \models^{\mathsf{rel}} \varphi_2$, and for all t with $i < t < j$ we have $(w, t, \delta + d_t - d_i) \models^{\mathsf{rel}} \varphi_1$. 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 4.3. Define

$$m_\delta = \min\{\delta(x) \mid x \text{ is a register variable in } \psi\}, \tag{4.3}$$

$$m_1 = \max\{d_i - d_j \mid d_i \text{ and } d_j \text{ are data values in } u_1 u_2\} \text{ and} \tag{4.4}$$

$$m_2 = \min\{d \mid d \text{ is a data value in } u_2\}. \tag{4.5}$$

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

$$n_\delta = \max\{2, \left\lceil \frac{C + M + 1 + d_i - m_\delta - m_2}{k} \right\rceil + 1\}. \tag{4.6}$$

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

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

By Lemmas 4.3 and 4.5, for every $h \geq |u_1| + (n_\delta - 1)|u_2|$ 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 $|u_1| + n_\delta |u_2|$. Similarly, we get the same result for $\varphi_1 \mathrm{R} \varphi_2$.

We sketch an alternating Turing machine $\mathcal{T}$ that, given a $\mathrm{TPTL_b}$ formula ψ and a data word w, has an accepting run if, and only if, $w \models_{\mathrm{TPTL}} \psi$. The machine $\mathcal{T}$ first computes and stores the value $C + M + 1$. In every configuration, $\mathcal{T}$ 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 4.3, we can restrict i to the interval $[0, |u_1| + |u_2|)$, and by Lemma 4.5, we can restrict the range of δ to the interval $[-m_1, \max\{m_1, C + M + 1\}]$. The machine $\mathcal{T}$ starts with the triple $(0, \mathbf{0}, \psi)$, where $\mathbf{0}(x) = 0$ for each register variable x. Then, $\mathcal{T}$ 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 $\varphi = \varphi_1 \mathrm{U} \varphi_2$, then branch existentially to the following two alternatives.

- Go to (i, δ, φ).

- Compute the value n_δ according to (4.3), (4.5), and (4.6), then branch existentially to each value $j \in (i+1, |u_1| + n_\delta|u_2|]$, and finally branch universally to each triple from $\{(\rho(t), \delta_t, \varphi_1) \mid i < t < j\} \cup \{(\rho(j), \delta_j, \varphi_2)\}$, 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 $\varphi = \varphi_1 \mathrm{R} \varphi_2$, then compute the value n_δ according to (4.3), (4.5), and (4.6) and branch existentially to the following two alternatives:

- branch universally to all triples from $\{(\rho(j), \delta_j, \varphi_2) \mid i \leq j \leq |u_1| + n_\delta|u_2|\}$, 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 $j \in [i+1, |u_1| + n_\delta|u_2|]$, and then branch universally to all triples from $\{(\rho(t), \delta_t, \varphi_2) \mid i < t \leq j\} \cup \{(\rho(j), \delta_j, \varphi_1)\}$, 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 $\mathcal{T}$ 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 formulas $\varphi_1 \mathrm{U} \varphi_2$ and $\varphi_1 \mathrm{R} \varphi_2$. If $k = 0$, then it is easily seen that for every $i \geq |u_1|$, formula φ and valuation ν, $(w, i, \nu) \models_{\mathrm{TPTL}} \varphi$ if, and only if, $(w, i + |u_2|, \nu) \models_{\mathrm{TPTL}} \varphi$. One can easily see 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. □

Note that one can easily adapt the proof of Theorem 4.1 to obtain the following result for *finite* data words:

Theorem 4.6. *Path checking for* $\mathrm{TPTL_b}$ *over finite binary encoded data words is in* PSPACE.

Straight-line programs for data words. In this section we prove an extension of Theorem 4.1, where the data words are succinctly specified by so called straight-line programs. Straight-line programs allow to represent data words of exponential length. In Section 6 we will apply the result of this section to the model-checking problem for deterministic one-counter machines with binary encoded updates.

A *straight-line program*, briefly SLP, is a tuple $\mathcal{G} = (V, A_0, \mathsf{rhs})$, where V is a finite set of variables, $A_0 \in V$ is the output variable, and rhs (for right-hand side) is a mapping that associates with every variable $A \in V$ an expression $\mathsf{rhs}(A)$ of the form BC, $B + d$, or (P, d), where $B, C \in V$, $d \in \mathbb{N}$, and $P \subseteq \mathbb{P}$. Moreover, we require that the relation

$\{(A,B) \in V \times V \mid B \text{ occurs in } \mathsf{rhs}(A)\}$ is acyclic. This allows to assign to every variable A inductively a data word $\mathsf{val}_{\mathcal{G}}(A)$:

- if $\mathsf{rhs}(A) := (P,d)$ then $\mathsf{val}_{\mathcal{G}}(A) = (P,d)$,
- if $\mathsf{rhs}(A) := B + d$ then $\mathsf{val}_{\mathcal{G}}(A) = \mathsf{val}_{\mathcal{G}}(B)_{+d}$, and
- if $\mathsf{rhs}(A) := BC$ then $\mathsf{val}_{\mathcal{G}}(A) = \mathsf{val}_{\mathcal{G}}(B)\mathsf{val}_{\mathcal{G}}(C)$.

Finally, we set $\mathsf{val}(\mathcal{G}) = \mathsf{val}_{\mathcal{G}}(A_0)$. We assume that natural numbers appearing in right-hand sides are binary encoded. The size of the SLP $\mathcal{G}$ is defined as the sum of the sizes of all right-hand sides, where a right-hand side of the form BC (resp. (P,d) or $B+d$) has size 1 (resp., $\lceil \log_2 d \rceil$).

Example 4.7. *Consider the straight-line program* $\mathcal{G} = (\{A_0, A_1, A_2, A_3, A_4, A_5\}, A_0, \mathsf{rhs})$, *where* rhs *is defined as follows:*

$$\mathsf{rhs}(A_5) = (\{a,b\},2),\ \mathsf{rhs}(A_4) = (\{b,c\},3), \mathsf{rhs}(A_3) = A_5A_4,$$
$$\mathsf{rhs}(A_2) = A_3A_3,\ \mathsf{rhs}(A_1) = A_2 + 4,\ \mathsf{rhs}(A_0) = A_2A_1.$$

We get

$$\begin{aligned}
\mathsf{val}_{\mathcal{G}}(A_3) &= (\{a,b\},2)(\{b,c\},3)\\
\mathsf{val}_{\mathcal{G}}(A_2) &= (\{a,b\},2)(\{b,c\},3)(\{a,b\},2)(\{b,c\},3)\\
\mathsf{val}_{\mathcal{G}}(A_1) &= (\{a,b\},6)(\{b,c\},7)(\{a,b\},6)(\{b,c\},7)\\
\mathsf{val}_{\mathcal{G}}(A_0) &= (\{a,b\},2)(\{b,c\},3)(\{a,b\},2)(\{b,c\},3)(\{a,b\},6)(\{b,c\},7)(\{a,b\},6)(\{b,c\},7)\\
&= \mathsf{val}(\mathcal{G}).
\end{aligned}$$

Lemma 4.8. *For a given SLP* $\mathcal{G}$ *we can compute the numbers* $\min(\mathsf{val}(\mathcal{G}))$ *and* $\max(\mathsf{val}(\mathcal{G}))$ *in polynomial time.*

Proof. Let $\mathcal{G} = (V, A_0, \mathsf{rhs})$. We compute $\min(\mathsf{val}_{\mathcal{G}}(A))$ for every $A \in V$ bottom-up according to the following rules:

- If $\mathsf{rhs}(A) = (P,d)$, then $\min(\mathsf{val}_{\mathcal{G}}(A)) := d$.
- If $\mathsf{rhs}(A) = B + k$, then $\min(\mathsf{val}_{\mathcal{G}}(A)) := \min(\mathsf{val}_{\mathcal{G}}(B)) + k$.
- If $\mathsf{rhs}(A) = BC$, then $\min(\mathsf{val}_{\mathcal{G}}(A)) := \min\{\min(\mathsf{val}_{\mathcal{G}}(B)), \min(\mathsf{val}_{\mathcal{G}}(C))\}$.

Of course, for computing $\max(\mathsf{val}(\mathcal{G}))$ in polynomial time one can proceed analogously. □

The following result extends Theorem 4.1 to SLP-encoded infinite data words, *i.e.*, infinite data words $u_1(u_2)^{\omega}_{+k}$ that are succinctly represented by two SLPs for u_1 and u_2, respectively, and the binary encoding of the number k.

Theorem 4.9. *Path checking for* $\mathrm{TPTL_b}$ *over infinite (resp., finite) SLP-encoded data words is in* PSPACE.

Proof. We only show the statement for infinite data words. The proof is almost identical to the proof of Theorem 4.1. We explain the necessary adaptations. The input consists of a TPTL formula ψ (without loss of generality in negation normal form), two SLPs $\mathcal{G}_1$ and $\mathcal{G}_2$ and a binary encoded number k. Let $u_1 = \mathsf{val}(\mathcal{G}_1)$, $u_2 = \mathsf{val}(\mathcal{G}_2)$ and $w = u_1(u_2)^{\omega}_{+k}$. We have to check whether $w \models_{\mathrm{TPTL}} \psi$ holds. For this we use the alternating polynomial time algorithm from the proof of Theorem 4.1. We only consider the more difficult case $k > 0$. Define $C := C_{\psi}$ by (4.1), $M := M_{u_2}$ by (4.2), m_1 by (4.4), and m_2 by (4.5). Note that the binary encodings of these four numbers can be computed in polynomial time by Lemma 4.8.

As before, in every configuration, $\mathcal{T}$ stores a triple (i, δ, φ), where φ is a subformula of ψ, i is a position in the data word from the interval $[0, |u_1| + |u_2|)$, and δ is a register valuation (with respect to the relative semantics) whose range is $[-m_1, \max\{m_1, C + M + 1\}]$. For each such δ we define m_δ as in (4.3) and n_δ as in (4.6). Given δ, one can compute the binary encodings of these numbers in polynomial time. Furthermore, note that a triple (i, δ, φ) with the above properties can be stored in polynomial space.

The machine $\mathcal{T}$ starts with the triple $(0, \mathbf{0}, \psi)$, where $\mathbf{0}(x) = 0$ for each register variable x. Then, $\mathcal{T}$ branches according to the rules from the proof of Theorem 4.1. The numbers j and t guessed there can be still stored in polynomial space, and all arithmetic manipulations can be done in polynomial time. In particular, the function $\rho : \mathbb{N} \to [0, |u_1| + |u_2|)$ defined by $\rho(z) = z$ for $z < |u_1|$, and $\rho(z) = ((z - |u_1|) \bmod |u_2|) + |u_1|$ otherwise, can be computed in polynomial time on binary encoded numbers. □

4.2. **Polynomial Time Upper Bounds for TPTL with Fixed Number of Registers.** If the number of register variables is fixed and all numbers are unary encoded (or their unary encodings can be computed in polynomial time), then the alternating Turing-machine in the proof of Theorem 4.1 works in logarithmic space. Since ALOGSPACE = P, we obtain the following statement for (i):

Theorem 4.10. *For every fixed $r \in \mathbb{N}$, path checking for $\mathrm{TPTL}^r_{\mathrm{u}}$ over (i) infinite unary encoded data words or (ii) infinite binary encoded monotonic data words is in* P.

Proof. We start with the proof of the statement for (i). In the algorithm from the proof of Theorem 4.1, if all numbers are given in unary, then the numbers $C + M + 1$, m_1, m_2 and n can be computed in logarithmic space and are bounded polynomially in the input size. Moreover, a configuration of the form (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 4.1 works in logarithmic space. The theorem follows, since $\mathsf{ALOGSPACE} = \mathsf{P}$.

Let us now prove statement (ii) in Theorem 4.10. We actually prove a slightly stronger statement for so called quasi-monotonic data words instead of monotonic data words.

For a finite data word u, let $\min(u)$ (resp., $\max(u)$) be the minimal (resp., maximal) data value that occurs in u. Given $k \in \mathbb{N}$, and two finite data words u_1 and u_2, we say that the infinite data word $u_1(u_2)^\omega_{+k}$ is *quasi-monotonic* if $\max(u_1) \leq \max(u_2) \leq \min(u_2) + k$. Note that if $u_1(u_2)^\omega_{+k}$ is monotonic, then $u_1(u_2)^\omega_{+k}$ is also quasi-monotonic.

Let us now prove statement (ii) (with "monotonic" replaced by "quasi-monotonic"). Suppose that k, u_1 and u_2 are binary encoded, and $u_1(u_2)^\omega_{+k}$ is quasi-monotonic. The idea is that we construct in polynomial time two unary encoded finite data words v_1, v_2 and $l \in \mathbb{N}$ encoded in unary notation such that $u_1(u_2)^\omega_{+k} \models_{\mathrm{TPTL}} \psi$ if, and only if, $v_1(v_2)^\omega_{+l} \models_{\mathrm{TPTL}} \psi$, where ψ is the input $\mathrm{TPTL}^r_{\mathrm{u}}$ formula. Then we can apply the alternating logarithmic space algorithm from the above proof for (i).

Let $x_1 \sim_1 c_1, \ldots, x_m \sim_m c_m$ be all constraint formulas in ψ. Without loss of generality, we suppose that $c_i \leq c_{i+1}$ for $1 \leq i < m$. We define an equivalence relation $\equiv_\psi$ on $\mathbb{N}$ such that $a \equiv_\psi b$ if $a = b$ or a and b both belong to one of the intervals $(-\infty, c_1)$, $(c_m, +\infty)$, (c_i, c_{i+1}) for some $1 \leq i < m$. Define $C = \max\{|c_1|, \ldots, |c_m|\}$.

Suppose that $|u_1| = n_1$ and $|u_2| = n_2$. Let $d_1, \ldots, d_{n_1+n_2}$ be an enumeration of all data values in u_1u_2 such that $d_j \leq d_{j+1}$ for $1 \leq j < n_1 + n_2$. Without loss of generality, we

suppose that $d_1 = 0$. For $1 < i \leq n_1 + n_2$ let $\delta_i = d_i - d_{i-1}$. We define a new sequence $d'_1, \ldots, d'_{n_1+n_2}$ inductively as follows: $d'_1 = 0$ and for all $1 < i \leq n_1 + n_2$,

$$d'_i = \begin{cases} d'_{i-1} + \delta_i & \text{if } \delta_i \leq C, \\ d'_{i-1} + C + 1 & \text{if } \delta_i > C. \end{cases}$$

Intuitively, the data values d'_j are obtained by shrinking the d_j so that the largest difference between two different data values is bounded by $C + 1$.

We obtain the new data words v_1 and v_2 by replacing in u_1 and u_2 every data value d_j by d'_j for every $j \in [1, n_1 + n_2]$. Note that $d'_{n_1+n_2} \leq (C+1) \cdot (n_1 + n_2 - 1)$. Since C is given in unary notation, we can compute in polynomial time the unary encodings of the numbers $d'_1, \ldots, d'_{n_1+n_2}$.

To define the number l, note that $\delta := \min(u_2) + k - \max(u_2)$ is the difference between the smallest data value in $(u_2)_{+k}$ and the largest data value in u_2 (which is the largest data value of u_1u_2). Since $u_1(u_2)^\omega_{+k}$ is quasi-monotonic, we have $\delta \geq 0$. We define the number l as

$$l = \begin{cases} \max(v_2) - \min(v_2) + \delta & \text{if } \delta \leq C, \\ \max(v_2) - \min(v_2) + C + 1 & \text{if } \delta > C. \end{cases}$$

Again, the unary encoding of l can be computed in polynomial time. Let e_i (resp., e'_i) be the data value in the i-th position of $u_1(u_2)^\omega_{+k}$ (resp., $v_1(v_2)^\omega_{+l}$). Then, for every $j > i$ we have $e_j - e_i \equiv_\psi e'_j - e'_i$. This implies that $u_1(u_2)^\omega_{+k} \models_{\mathrm{TPTL}} \varphi$ if, and only if, $v_1(v_2)^\omega_{+l} \models_{\mathrm{TPTL}} \varphi$. Finally, we can use the alternating logarithmic space algorithm to check whether $v_1(v_2)^\omega_{+l} \models_{\mathrm{TPTL}} \varphi$ holds. □

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 4.11. *For every fixed $r \in \mathbb{N}$, path checking for $\mathrm{TPTL}^r_{\mathrm{b}}$ 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 register variables appearing in the input formula ψ. Then, we only have to consider the n^r many register valuation mappings $\nu : \{x_1, \ldots, x_r\} \to \{d_1, \ldots, d_n\}$. For each of these mappings ν, for every subformula φ of ψ, and for every position i in w we check whether $(w, i, \delta) \models_{\mathrm{TPTL}} \varphi$. 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, *cf.* Theorem 5.12:

Theorem 4.12. *Path checking for $\mathrm{TPTL}^1_{\mathrm{b}}$ over infinite binary encoded data words is in* P.

For the proof of Theorem 4.12 we first show some auxiliary results of independent interest.

Lemma 4.13. *For a given* LTL *formula ψ, words $u_1, \ldots, u_n, u \in (2^{\mathbb{P}})^*$ and binary encoded numbers $N_1, \ldots, N_n \in \mathbb{N}$, the question whether $u_1^{N_1} u_2^{N_2} \ldots u_n^{N_n} u^\omega \models \psi$ holds, belongs to* P *(more precisely,* $\mathrm{AC}^1(\mathrm{LogDCFL})$*).*

Proof. We first prove that for all finite words $u, v \in (2^{\mathbb{P}})^*$, every infinite word $w \in (2^{\mathbb{P}})^\omega$ and every number $N \geq |\psi|$, we have $uv^N w \models \psi$ if, and only if, $uv^{|\psi|}w \models \psi$. For the proof,

we use the Ehrenfeucht-Fraïssé game for LTL (EF-game, for short), introduced in [10] and briefly explained in the following.

Let $w_0 = (P_0, d_0)(P_1, d_1)\dots$ and $w_1 = (P'_0, d'_0)(P'_1, d'_1)\dots$ be two infinite data words over $2^{\mathbb{P}}$, and let $k \geq 0$. The k-round EF-game is played by two players, called Spoiler and Duplicator, on w_0 and w_1. A game configuration is a pair of positions $(i_0, i_1) \in \mathbb{N} \times \mathbb{N}$, where i_0 is a position in w_0, and i_1 is a position in w_1. In each round of the game, if the current configuration is (i_0, i_1), Spoiler chooses an index $l \in \{0, 1\}$ and a position $j_l > i_l$ in the data word w_l. Then, Duplicator responds with a position $j_{1-l} > i_{1-l}$ in the other data word w_{1-l}. After that, Spoiler has two options:

- He chooses to finish the current round. The current round then finishes with configuration (j_0, j_1).
- He chooses a position j'_{1-l} so that $i_{1-l} < j'_{1-l} < j_{1-l}$. Then Duplicator responds with a position j'_l so that $i_l < j'_l < j_l$. The current round then finishes with configuration (j'_0, j'_1).

The *winning condition for Duplicator* is defined inductively by the number k of rounds of the EF-game: duplicator wins the 0-round EF-game starting in configuration (i_0, i_1) if $P_{i_0} = P'_{i_1}$. Duplicator wins the $k+1$-round EF-game starting in configuration (i_0, i_1) if $P_{i_0} = P'_{i_1}$ and for every choice of moves of Spoiler in the first round, Duplicator can respond such that Duplicator wins the k-round EF-game starting in the finishing configuration of the first round (*i.e.*, in the above configuration (j_0, j_1) or (j'_0, j'_1)).

The *until rank* of an LTL formula φ, denoted by $\mathsf{Rank}(\varphi)$, is defined inductively on the structure of φ:

- If φ is true, $p \in \mathbb{P}$ or $x \sim c$, then $\mathsf{Rank}(\varphi) = 0$.
- If φ is $\neg\varphi_1$, then $\mathsf{Rank}(\varphi) = \mathsf{Rank}(\varphi_1)$.
- If φ is $\varphi_1 \wedge \varphi_2$, then $\mathsf{Rank}(\varphi) = \max\{\mathsf{Rank}(\varphi_1), \mathsf{Rank}(\varphi_2)\}$.
- If φ is $\varphi_1 \mathrm{U} \varphi_2$, then $\mathsf{Rank}(\varphi) = \max\{\mathsf{Rank}(\varphi_1), \mathsf{Rank}(\varphi_2)\} + 1$.

Theorem 4.14 ([10])**.** *Duplicator wins the k-round EF-game on w_0 and w_1 starting in configuration $(0, 0)$ if, and only if, for every* LTL *formula φ with $\mathsf{Rank}(\varphi) \leq k$, we have $w_0 \models_{\mathrm{LTL}} \varphi$ if, and only if, $w_1 \models_{\mathrm{LTL}} \varphi$.*

Now, let $k \geq 0$, and assume that $w_0 = uv^{m_0}w$ and $w_1 = uv^{m_1}w$, where $m_0, m_1 \geq k$, u and v are finite words, and w is an infinite word. One can easily prove that Duplicator can win the k-round EF-game starting from configuration $(0, 0)$. The point is that Duplicator can enforce that after the first round the new configuration (i_0, i_1) satisfies one of the following two conditions:

- $w_0[i_0{:}] = w_1[i_1{:}]$, *i.e.*, after the first round the suffix of w_0 starting in position i_0 is equal to the suffix of w_1 starting in i_1. This implies that Duplicator can win the remaining $(k-1)$-round EF-game starting in configuration (i_0, i_1).
- $w_0[i_0{:}]$ (respectively, $w_1[i_1{:}]$) has the form $u'v^{n_0}w$ (respectively, $u'v^{n_1}w$), where $n_0, n_1 \geq k-1$. Hence, by induction, Duplicator can win the $(k-1)$-round EF-game from configuration (i_0, i_1).

By Theorem 4.14, we have $uv^{m_0}w \models_{\mathrm{LTL}} \varphi$ if, and only if, $uv^{m_1}w \models_{\mathrm{LTL}} \varphi$ for every LTL formula φ with $\mathsf{Rank}(\varphi) \leq k$. It follows that for two infinite words $u_1^{m_1}u_2^{m_2}\dots u_n^{m_n}u_{n+1}^{\omega}$ and $u_1^{m'_1}u_2^{m'_2}\dots u_n^{m'_n}u_{n+1}^{\omega}$ satisfying $m_i, m'_i \geq k$ for all $i \in \{1, \dots, n\}$, we have

$$u_1^{m_1}u_2^{m_2}\dots u_l^{m_n}u_{n+1}^{\omega} \models_{\mathrm{LTL}} \varphi \text{ if, and only if, } u_1^{m'_1}u_2^{m'_2}\dots u_l^{m'_n}u_{n+1}^{\omega} \models_{\mathrm{LTL}} \varphi$$

for every LTL formula φ with $\mathsf{Rank}(\varphi) \leq k$.

We use this to prove the lemma as follows: replace every binary encoded exponent N_i in the word $u_1^{N_1} u_2^{N_2} \dots u_l^{N_n} u_{n+1}^{\omega}$ by $\min\{N_i, |\psi|\}$. By Theorem 3.6 in [24], infinite path checking for LTL can be reduced in logspace to finite path checking for LTL. Finite path checking for LTL is in $\mathrm{AC}^1(\mathrm{LogDCFL}) \subseteq \mathrm{P}$ [21]. □

Lemma 4.15. *Path checking for* $\mathrm{TPTL_b}$ *formulas, which do not contain the freeze quantifier* $x.(\cdot)$*, over infinite binary encoded data words is in* P *(in fact,* $\mathrm{AC}^1(\mathrm{LogDCFL})$*).*

Proof. We reduce the question whether $w \models_{\mathrm{TPTL}} \psi$ in logspace to an instance of the special LTL path-checking problem from Lemma 4.13. Let $w = u_1(u_2)^{\omega}_{+k}$. We use $w[i]$ to denote the pair $(P_i, d_i) \in 2^{\mathbb{P}} \times \mathbb{N}$ occurring at the i-th position of w. Let $n_1 = |u_1|$ and $n_2 = |u_2|$. Without loss of generality we may assume that the only register variable that appears in constraint formulas in φ is x. Note that since φ does not contain the freeze quantifier, the value of x is always assigned the initial value d_0.

In order to construct an LTL formula from ψ, it remains to eliminate occurrences of constraint formulas $x \sim c$ in ψ. Without loss of generality, we may assume that all constraints are of the form $x < c$ or $x > c$. Let $x \sim_1 c_1, \dots, x \sim_m c_m$ be a list of all constraints that appear in ψ. We introduce for every $j \in \{1, \dots, m\}$ a new atomic proposition p_j, and we define $\mathbb{P}' := \mathbb{P} \cup \{p_1, \dots, p_m\}$. Let ψ' be obtained from ψ by replacing every occurrence of $x \sim_j c_j$ by p_j, and let $w' \in (2^{\mathbb{P}'})^{\omega}$ be the ω-word defined by $w'[i] = P_i \cup \{p_j \mid 1 \leq j \leq m, d_i - d_0 \sim_j c_j\}$. Clearly $w \models_{\mathrm{TPTL}} \psi$ if, and only if, $w' \models_{\mathrm{LTL}} \psi'$. Next we will show that the word w' can be written in the form required in Lemma 4.13.

First of all, we can write w' as $w' = u_1' u_{2,0}' u_{2,1}' u_{2,2}' \dots$, where $|u_1'| = n_1$ and $|u_{2,i}'| = n_2$. The word u_1' can be computed in logspace by evaluating all constraints at all positions of u_1. Moreover, every word $u_{2,i}'$ is obtained from u_2 (without the data values) by adding the new propositions p_j at the appropriate positions. Consider the equivalence relation $\equiv$ on $\mathbb{N}$ with $a \equiv b$ if, and only if, $u_{2,a}' = u_{2,b}'$. The crucial observations are that (i) every equivalence class of $\equiv$ is an interval, and (ii) the index of $\equiv$ is bounded by $1 + n_2 \cdot m$ (one plus the length n_2 of u_2 times the number m of constraint formulas). To see this, consider a position $i \in \{0, \dots, n_2 - 1\}$ in the word u_2 and a constraint $x \sim_j c_j$ for some $j \in \{1, \dots, m\}$. Then, the truth value of "proposition p_j is present at the i-th position of $u_{2,x}'$" switches (from true to false or from false to true) at most once when x grows. The reason for this is that the data value at position $n_1 + i + n_2 \cdot x$ is $d_{n_1+i+n_2 \cdot x} = d_{n_1+i} + k \cdot x$ for $x \geq 0$, i.e., it grows monotonically with x. Hence, the truth value of $d_{n_1+i} + k \cdot x - d_0 \sim_j c_j$ switches at most once, when x grows. So, we get at most $n_2 \cdot m$ many "switching points" in $\mathbb{N}$ which produce at most $1 + n_2 \cdot m$ many intervals.

Let $I_1, \dots, I_l$ be a list of all $\equiv$-classes (intervals), where $a < b$ whenever $a \in I_i$, $b \in I_j$ and $i < j$. The borders of these intervals can be computed in logspace using arithmetics on binary encoded numbers (addition, multiplication and division with remainder can be carried out in logspace on binary encoded numbers [18]). Hence, we can compute in logspace the lengths $N_i = |I_i|$ of the intervals, where $N_l = \omega$. Also, for all $i \in \{1, \dots, l\}$ we can compute in logspace the unique word v_i such that $v_i = u_{2,a}'$ for all $a \in I_i$. Hence, $w' = u_1' v_1^{N_1} \dots v_l^{N_l}$. We can now apply Lemma 4.13. □

Proof of Theorem 4.12. Consider an infinite binary encoded data word $w = u_1(u_2)^{\omega}_{+k}$ and a $\mathrm{TPTL_b^1}$ formula ψ. Let $n = |u_1| + |u_2|$. We check in polynomial time whether $w \models_{\mathrm{TPTL}} \psi$.

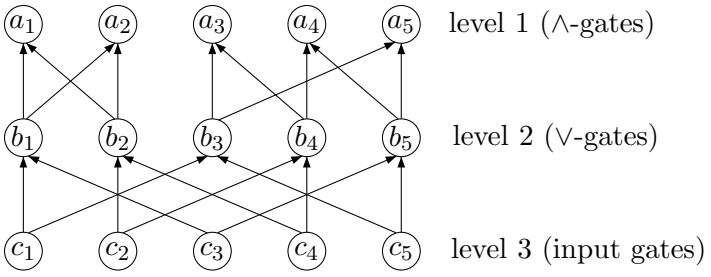

Figure 1: An SAM2-circuit with three levels.

A TPTL formula φ is *closed* if every occurrence of a register variable x in φ is under the scope of a freeze quantifier $x.(\cdot)$. The proofs of the following two claims are straightforward:

Claim 1: If φ is closed, then for all valuations ν, ν', $(w, i, \nu) \models_{\text{TPTL}} \varphi$ if, and only if, $(w, i, \nu') \models_{\text{TPTL}} \varphi$.

Claim 2: If φ is closed and $i \geq |u_1|$, then for every valuation ν, $(w, i, \nu) \models_{\text{TPTL}} \varphi$ if, and only if, $(w, i + |u_2|, \nu) \models_{\text{TPTL}} \varphi$.

By Claim 1 we can write $(w, i) \models_{\text{TPTL}} \varphi$ for $(w, i, \nu) \models_{\text{TPTL}} \varphi$. It suffices to compute for every (necessarily closed) subformula $x.\varphi$ of ψ the set of all positions $i \in [0, n-1]$ such that $(w, i) \models_{\text{TPTL}} x.\varphi$, or equivalently $w[i:] \models_{\text{TPTL}} \varphi$. We do this in a bottom-up process. Consider a subformula $x.\varphi$ of ψ and a position $i \in [0, n-1]$. We have to check whether $w[i:] \models_{\text{TPTL}} \varphi$. Let $x.\varphi_1, \ldots, x.\varphi_l$ be a list of all subformulas of φ that are not in the scope of another freeze quantifier within φ. We can assume that for every $s \in \{1, \ldots, l\}$ we have already determined the set of positions $j \in [0, n-1]$ such that $(w, j) \models_{\text{TPTL}} x.\varphi_s$. We can therefore replace every subformula $x.\varphi_s$ of φ by a new atomic proposition p_s and add in the data words u_1 (resp., u_2) the proposition p_s to all positions j (resp., $j - |u_1|$) such that $(w, j) \models_{\text{TPTL}} x.\varphi_s$, where $j \in [0, n-1]$. Here, we make use of Claim 2. We denote the resulting formula and the resulting data word with φ' and $w' = u_1'(u_2')^\omega_{+k}$, respectively. Next, we explain how to compute from u_1' and u_2' new finite data words v_1 and v_2 such that $v_1(v_2)^\omega_{+k} = w'[i:]$. If $i < |u_1'|$ then we take $v_1 = u_1'[i:]$ and $v_2 = u_2'$. If $|u_1'| \leq i \leq n-1$, then we take $v_1 = \varepsilon$ and $v_2 = u_2'[i:](u_2'[:i-1] + k)$, where $u_2'[:i-1]$ denotes the prefix of u_2' up to position $i-1$. Finally, using Lemma 4.15 we can check in polynomial time whether $w'[i:] \models_{\text{TPTL}} \varphi'$. □

5. Lower Complexity Bounds

In this section we prove several P-hardness and PSPACE-hardness results for the path-checking problem. Together with the upper bounds from Section 4 we obtain sharp complexity results for the various path-checking problems.

5.1. **P-Hardness Results.** We prove our P-hardness results by a reduction from a restricted version of the Boolean circuit value problem. A *Boolean circuit* is a finite directed acyclic graph, where each node is called a *gate*. An *input gate* is a node with indegree 0. All other gates are of a certain type, which is either $\vee$, $\wedge$ or $\neg$ (*i.e.*, the corresponding logical OR, AND and NOT operations). An *output* gate is a node with outdegree 0. A Boolean circuit is *monotone* if it does not have gates of type $\neg$.

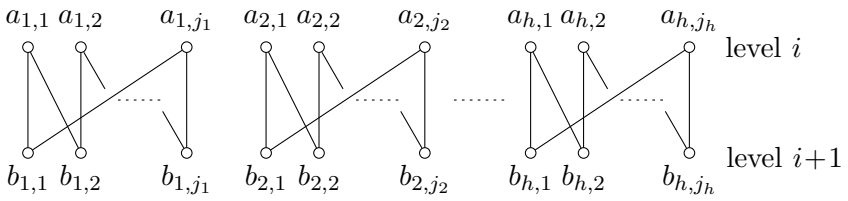

Figure 2: The induced subgraph between level i and $i+1$

A *synchronous alternating monotone circuit with fanin 2 and fanout 2* (SAM2-circuit, for short) is a monotone circuit divided into levels $1, \ldots, l$ for some $l \geq 2$ such that the following properties hold:

- All wires go from a gate in level $i+1$ to a gate from level i.
- All output gates are in level 1 and all input gates are in level l.
- All gates in the same level are of the same type ($\wedge$, $\vee$ or input) and the types of the levels between 1 and $l-1$ alternate between $\wedge$ and $\vee$.
- All gates except the output gates have outdegree 2, and all gates except the input gates have indegree 2. For all $i \in \{2, \ldots, l-1\}$, the two input gates for a gate at level i are different.

By the restriction to fanin 2 and fanout 2, each level contains the same number of gates. Figure 1 shows an example of an SAM2-circuit; the node names a_i, b_i, c_i will be used later. The *circuit value problem* for SAM2-circuits, called SAM2CVP in [16], is the following problem: given an SAM2-circuit α, inputs $x_1, \ldots, x_n \in \{0,1\}$, and a designated output gate y, does the output gate y of α evaluate to 1 on inputs $x_1, \ldots, x_n$? The circuit value problem for SAM2-circuits is P-complete [16].

Theorem 5.1. *Path checking for* $\mathrm{PureMTL}(\mathrm{F}, \mathrm{X})_{\mathrm{u}}$ *over finite unary encoded pure data words is* P*-hard.*

Proof. The proof is a reduction from SAM2CVP. Let α be an SAM2-circuit. We first encode each pair of consecutive levels of α into a data word, and combine these data words into a data word w, which is the encoding of the whole circuit. Then we construct a $\mathrm{PureMTL}(\mathrm{F}, \mathrm{X})_{\mathrm{u}}$ formula ψ such that $w \models_{\mathrm{MTL}} \psi$ if, and only if, α evaluates to 1. The data word w that we are going to construct contains gate names of α (and some copies of the gates) as atomic propositions. These propositions are only needed for the construction. At the end, we remove all propositions from the data word w to obtain a pure data word. The whole construction can be done in logarithmic space. The reader might look at Example 5.2, where the construction is carried out for the circuit shown in Figure 1.

Let α be an SAM2-circuit with $l \geq 2$ levels and n gates in each level. By the restriction to fanin 2 and fanout 2, the induced undirected subgraph containing the nodes in level i and level $i+1$ is comprised of several cycles; see Figure 2. For instance, for the circuit in Figure 1, there are two cycles between level 1 and 2, and we have the same number of cycles between level 2 and 3.

We can enumerate in logarithmic space the gates of level i and level $i+1$ such that they occur in the order shown in Figure 2. To see this, let $a_1, \ldots, a_n$ (respectively, $b_1, \ldots, b_n$) be the nodes in level i (respectively, $i+1$) in the order in which they occur in the input description. We start with b_1 and enumerate the nodes in the cycle containing b_1 (from

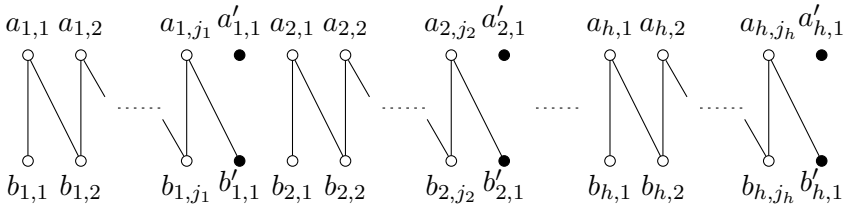

Figure 3: The graph after adding dummy nodes

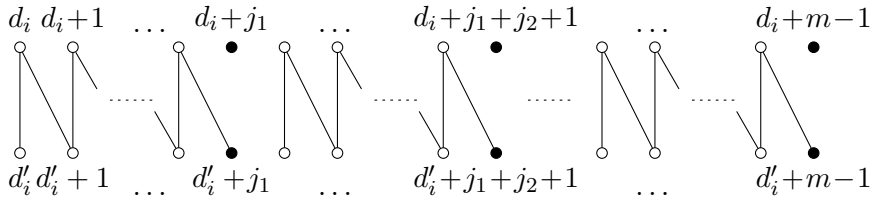

Figure 4: The graph after assigning data values to the nodes

b_1 we go to the smaller neighbour among $a_1, \ldots, a_n$. Then from this node the next node on the cycle is uniquely determined since the graph has degree 2. Thereby we store the current node in the cycle and the starting node b_1. As soon as we come back to b_1, we know that the first cycle is completed. To find the next cycle, we search for the first node from the list $b_2, \ldots, b_n$ that is not reachable from b_1 (reachability in undirected graphs is in LOGSPACE), and continue in this way.

So, assume that the nodes in level i and $i+1$ are ordered as in Figure 2. Assume we have h cycles. Next, we want to get rid of the crossing edges between the rightmost node in level i and the leftmost node in level $i+1$ in each cycle. For this, we introduce for each cycle two dummy nodes which are basically copies of the leftmost node in each level. Formally, for each $t \in \{1, \ldots, h\}$, add a node $a'_{t,1}$ (respectively, $b'_{t,1}$) after a_{t,j_t} (respectively, b_{t,j_t}), and then replace the edge $(a_{t,j_t}, b_{t,1})$ by a new edge $(a_{t,j_t}, b'_{t,1})$. In this way we obtain the graph shown in Figure 3. Again, the construction can be done in logarithmic space.

By adding dummy nodes, we can assume that for every $i \in \{1, \ldots, l-1\}$, the subgraph between level i and $i+1$ has the same number (say h) of cycles (this is only done for notational convenience, and we still suppose that there are n gates in each level). Thus, after the above step we have $m = n + h$ many nodes in each level. Define $d_i = (i-1) \cdot 2m$ and $d'_i = d_i + m$. Next we are going to label the nodes from Figure 3: in level i (respectively, $i+1$) with the numbers $d_i, d_i+1, \ldots, d_i+m-1$ (respectively $d'_i, d'_i+1 \ldots, d'_i+m-1$) in this order, see Figure 4. Note that this labelling is the crucial point for encoding the wiring between the gates of two levels: the difference between two connected nodes in level i and level $i+1$ is always m or $m+1$. We will later exploit this fact and use the modality $\mathrm{F}_{[m,m+1]}$ (respectively, $\mathrm{G}_{[m,m+1]}$) to jump from a $\vee$-gate (respectively, $\wedge$-gate) in level i to a successor

gate in level $i+1$. We now obtain in logarithmic space the data word $w_i = w_{i,1}w_{i,2}$, where

$$w_{i,1} = \begin{cases} (a_{1,1}, d_i)(a_{1,2}, d_i+1)\cdots(a_{1,j_1}, d_i+j_1-1) \\ (a_{2,1}, d_i+j_1+1)(a_{2,2}, d_i+j_1+2)\cdots(a_{2,j_2}, d_i+j_1+j_2)\cdots \\ (a_{h,1}, d_i+\sum_{t=1}^{h-1} j_t + h - 1)(a_{h,2}, d_i+\sum_{t=1}^{h-1} j_t + h)\cdots(a_{h,j_h}, d_i+m-2) \end{cases}$$

$$w_{i,2} = \begin{cases} (b_{1,1}, d'_i)\cdots(b_{1,j_1}, d'_i+j_1-1)(b'_{1,1}, d'_i+j_1) \\ (b_{2,1}, d'_i+j_1+1)\cdots(b_{2,j_2}, d'_i+j_1+j_2)(b'_{2,1}, d'_i+j_1+j_2+1)\cdots \\ (b_{h,1}, d'_i+\sum_{t=1}^{h-1} j_t + h - 1)\cdots(b_{h,j_h}, d'_i+m-2)(b'_{h,1}, d'_i+m-1) \end{cases}$$

which is the encoding of the wires between level i and level $i+1$ from Figure 4. Note that the new nodes $a'_{1,1}, a'_{2,1}, \ldots, a'_{h,1}$ in level i of the graph in Figure 3 do not occur in $w_{i,1}$.

Suppose now that for all $i \in \{1, \ldots, l-1\}$, the data words w_i is constructed. We combine $w_1, w_2, \ldots, w_{l-1}$ to obtain the data word w that encodes the complete circuit as follows. Suppose that

$$w_{i,2} = (\tilde{b}_1, y_1)\ldots(\tilde{b}_m, y_m) \text{ and } w_{i+1,1} = (b_1, z_1)\ldots(b_n, z_n).$$

Note that every $\tilde{b}_s$ is either one of the b_j or b'_j (the copy of b_j). Let

$$v_{i+1,1} = (\tilde{b}_1, z'_1)\ldots(\tilde{b}_m, z'_m),$$

where the data values z'_s are determined as follows: if $\tilde{b}_s = b_j$ or $\tilde{b}_s = b'_j$, then $z'_s = z_j$. Then, the data word w is $w = w_{1,1}w_{1,2}v_{2,1}w_{2,2}\ldots v_{l-1,1}w_{l-1,2}$.

Next, we explain the idea of how to construct the MTL formula. Consider a gate a_j of level i for some $i \in \{2, \ldots, l-1\}$, and assume that level i consists of $\vee$-gates. Let b_{j_1} and b_{j_2} (from level $i+1$) be the two input gates for a_j. In the above data word $v_{i,1}$ there is a unique position where the proposition a_j occurs, and possibly a position where the copy a'_j occurs. If both positions exist, then they carry the same data value. Let us point to one of these positions. Using an MTL formula, we want to branch (existentially) to the positions in the factor $v_{i+1,1}$, where the propositions $b_{j_1}, b'_{j_1}, b_{j_2}, b'_{j_2}$ occur (where b'_{j_1} and b'_{j_2} possibly do not exist). For this, we use the modality $\mathrm{F}_{[m,m+1]}$. By construction, this modality branches existentially to positions in the factor $w_{i,2}$, where the propositions $b_{j_1}, b'_{j_1}, b_{j_2}, b'_{j_2}$ occur. Then, using the iterated next modality X^m, we jump to the corresponding positions in $v_{i+1,1}$.

In the above argument, we assumed that $i \in \{2, \ldots, l-1\}$. If $i = 1$, then we can argue similarly, if we assume that we are pointing to the unique a_j-labelled position of the prefix $w_{1,1}$ of w. Now consider level $l-1$. Suppose that

$$w_{l-1,2} = (\tilde{e}_1, d_1)\ldots(\tilde{e}_m, d_m).$$

Let $e_1, \ldots, e_n$ be the original gates of level l, which all belong to $\{\tilde{e}_1, \ldots, \tilde{e}_m\}$, and let $x_i \in \{0, 1\}$ be the input value for gate e_i. Define

$$J = \{j \mid j \in [1, m], \exists i \in [1, n] : \tilde{e}_j \in \{e_i, e'_i\}, x_i = 1\}. \tag{5.1}$$

Let the designated output gate be the k-th node in level 1. We construct the PureMTL(F, X) formula $\psi = \mathrm{X}^{k-1}\varphi_1$, where, for every $i \in \{1, \ldots, l-1\}$, the formula φ_i is inductively defined

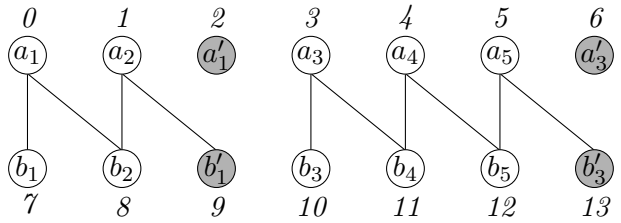

Figure 5: The labelling for level 1 and 2

as follows:

$$\varphi_i := \begin{cases} \mathrm{F}_{[m,m+1]}\mathrm{X}^m\varphi_{i+1} & \text{if } i < l-1 \text{ and level } i \text{ is a } \vee\text{-level,} \\ \mathrm{G}_{[m,m+1]}\mathrm{X}^m\varphi_{i+1} & \text{if } i < l-1 \text{ and level } i \text{ is a } \wedge\text{-level,} \\ \mathrm{F}_{[m,m+1]}(\bigvee_{j\in J}\mathrm{X}^{m-j}\neg\mathrm{X}\,\mathrm{true}) & \text{if } i = l-1 \text{ and level } i \text{ is a } \vee\text{-level,} \\ \mathrm{G}_{[m,m+1]}(\bigvee_{j\in J}\mathrm{X}^{m-j}\neg\mathrm{X}\,\mathrm{true}) & \text{if } i = l-1 \text{ and level } i \text{ is a } \wedge\text{-level.} \end{cases}$$

The formula $\neg\mathrm{X}\,\mathrm{true}$ is only true in the last position of a data word. Suppose data word w is the encoding of the circuit. From the above consideration, it follows that $w \models_{\mathrm{MTL}} \psi$ if, and only if, the circuit α evaluates to 1. Note that we only use the unary modalities $\mathrm{F}, \mathrm{G}, \mathrm{X}$ and do not use any propositions in ψ. We can thus ignore the propositional part in the data word w to get a pure data word. Since the number m is bounded by $2n$, and all data values in w are bounded by $4nl$, where n is the number of gates in each level and l is the number of levels, we can compute the formula ψ and data word w where the interval borders and data values are encoded in unary notation in logarithmic space. □

Example 5.2. *Let α be the SAM2-circuit from Figure 1. It has 3 levels and 5 gates in each level. Level 1 contains $\wedge$-gates and level 2 contains $\vee$-gates. There are 2 cycles in the subgraph between level 1 and 2, and also 2 cycles in the subgraph between level 2 and 3. The encoding for level 1 and level 2 is*

$$\begin{aligned} &(a_1,0)(a_2,1)(a_3,3)(a_4,4)(a_5,5) \\ &(b_1,7)(b_2,8)(b'_1,9)(b_3,10)(b_4,11)(b_5,12)(b'_3,13), \end{aligned} \tag{5.2}$$

which can be obtained from Figure 5. The new nodes a'_1 and a'_3 in level 1 are not used for the final encoding in the data word. The encoding for level 2 and 3 is

$$\begin{aligned} &(b_1,14)(b_5,15)(b_3,16)(b_2,18)(b_4,19) \\ &(c_1,21)(c_3,22)(c_5,23)(c'_1,24)(c_2,25)(c_4,26)(c'_2,27), \end{aligned} \tag{5.3}$$

which can be obtained from Figure 6. We skip the new nodes b'_1 and b'_2 in level 2 in this encoding.

We combine (5.2) and (5.3) to obtain the following data word (5.4) which is the encoding of the circuit α. The encoding for level 1 and 2 determines the order of the propositional part of the third line in (5.4), and the encoding for level 2 and 3 determines its data values.

$$\begin{aligned} &(a_1,0)(a_2,1)(a_3,3)(a_4,4)(a_5,5) \\ &(b_1,7)(b_2,8),(b'_1,9)(b_3,10)(b_4,11)(b_5,12)(b'_3,13) \\ &(b_1,14)(b_2,18)(b'_1,14)(b_3,16)(b_4,19)(b_5,15)(b'_3,16) \\ &(c_1,21)(c_3,22)(c_5,23)(c'_1,24)(c_2,25)(c_4,26)(c'_2,27). \end{aligned} \tag{5.4}$$

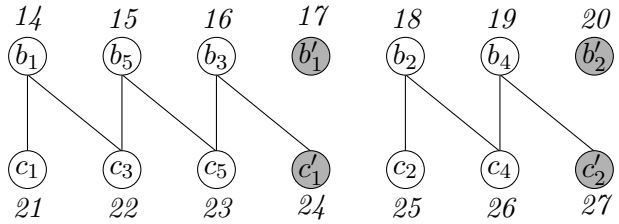

Figure 6: The labelling for level 2 and 3

Let the designated output gate be a_3 in level 1, and assume that the input gates c_1, c_4, c_5 (respectively, c_2, c_3) receive the value 0 (respectively, 1). Then the set J from (5.1) is $J = \{2, 5, 7\}$ and the formula ψ is

$$\psi = \mathrm{X}^2(\mathrm{G}_{[7,8]}\mathrm{X}^7(F_{[7,8]}(\bigvee_{j\in\{2,5,7\}} \mathrm{X}^{7-j}\neg\mathrm{X}\,\mathrm{true}))).$$

Theorem 5.3. *Path checking for* $\mathrm{MTL}(\mathrm{F},\mathrm{X})_{\mathrm{u}}$ *over infinite unary encoded data words is* P*-hard.*

Proof. We use an adaptation of the proof for Theorem 5.1. Let p be an atomic proposition that is not used in the data word w defined in the proof of Theorem 5.1. Define the infinite data word $w' = w\,(p, 5ml)^{\omega}_{+0}$, and redefine for every $i \in \{1, \ldots, l-1\}$ the formula φ_i by:

$$\varphi_i := \begin{cases} \mathrm{F}_{[m,m+1]}\mathrm{X}^m\varphi_{i+1} & \text{if } i < l-1 \text{ and level } i \text{ is a } \vee\text{-level,} \\ \mathrm{G}_{[m,m+1]}\mathrm{X}^m\varphi_{i+1} & \text{if } i < l-1 \text{ and level } i \text{ is a } \wedge\text{-level,} \\ \mathrm{F}_{[m,m+1]}(\bigvee_{j\in I}\mathrm{X}^{m-j}(\neg p \wedge \mathrm{X}p)) & \text{if } i = l-1 \text{ and level } i \text{ is a } \vee\text{-level,} \\ \mathrm{G}_{[m,m+1]}(\bigvee_{j\in I}\mathrm{X}^{m-j}(\neg p \wedge \mathrm{X}p)) & \text{if } i = l-1 \text{ and level } i \text{ is a } \wedge\text{-level.} \end{cases}$$

Then $w' \models_{\mathrm{MTL}} \psi$ if, and only if, the circuit α evaluates to 1. □

Note that the construction in the proof of Theorem 5.1 uses data words that are not monotonic. This is indeed unavoidable: path checking for MTL over finite monotonic data words is in $\mathrm{AC}^1(\mathrm{LogDCFL})$ [5]. In the following, we prove that in contrast to MTL, the path-checking problem for $\mathrm{TPTL}^1_{\mathrm{u}}$ over finite monotonic data words is P-hard, even for the pure fragment that only uses only the unary modalities F and X.

Theorem 5.4. *Path checking for* $\mathrm{PureTPTL}^1(\mathrm{F},\mathrm{X})_{\mathrm{u}}$ *over finite unary encoded strictly monotonic pure data words is* P*-hard.*

Before we prove Theorem 5.4, we prove P-hardness of path checking for some extension of MTL.

In MTL, the U modality is annotated by some interval I. If, instead, we allow the U modality to be annotated by a finite union of intervals $I_1 \cup I_2 \cup \cdots \cup I_n$, then we call this logic *succinct* MTL, SMTL for short. Formally, the syntax and semantics of SMTL is the same as for MTL, except that the set I in U_I can be a finite union $I = I_1 \cup I_2 \cup \cdots \cup I_n$ of intervals $I_i \subseteq \mathbb{Z}$. The corresponding fragments of SMTL are defined in the expected way.

Let $I = \bigcup_{i=1}^n I_i$. It is easily seen that ($\equiv$ denotes logical equivalence)

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

The following two propositions are easy to prove.

Proposition 5.5. *Each* SMTL *formula is equivalent to an* MTL *formula which can be exponentially larger.*

Proposition 5.6. *For a given* SMTL *formula φ one can compute in logspace an equivalent* TPTL^1 *formula of size polynomial in the size of φ.*

In the following, we prove P-hardness of path checking for SMTL_u over finite unary encoded strictly monotonic pure data words. Like the P-hardness proof for MTL in Theorem 5.1, the proof is by reduction from SAM2CVP. Unlike the data word in the proof of Theorem 5.1, we encode the wires between two levels of an SAM2-circuit by a *strictly monotonic* data word. This is possible due to the succinct usage of unions of intervals annotating the finally and globally modalities in the SMTL formula ψ. The reader might look at Example 5.8 below, where the construction is carried out for the circuit shown in Figure 1.

Theorem 5.7. *Path checking for* $\mathrm{PureSMTL}(\mathrm{F},\mathrm{X})_\mathrm{u}$ *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

$$w_{n,l} = \prod_{j=0}^{l-2} (w_n)_{+j\cdot n(n+2)} \tag{5.5}$$

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 each $j \in \{1, \ldots, l-1\}$, define

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

Suppose o_k, for some $k \in \{1, \ldots, n\}$, 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 $\psi = \mathrm{X}^{k-1}\varphi_1$, where φ_j, for all $j \in \{1, \ldots, l-1\}$, is defined inductively as follows:

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

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 $(j+1)$-th copy of w_n in $w_{n,l}$.

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

Example 5.8. *We illustrate the proof of Theorem 5.7 with the SAM2 from Figure 1. The formula in equation* (5.5) *yields*

$$w_{5,3} = (1, 2, 3, 4, 5,\ 6, 12, 18, 24, 30)(36, 37, 38, 39, 40\ 41, 47, 53, 59, 65).$$

We compute

$$S_1 = \{5, 11, 4, 10, 15, 21, 20, 26, 13, 25\} \text{ and } S_2 = \{5, 17, 10, 22, 3, 27, 8, 20, 13, 25\}.$$

Note that the values in S_1 correspond to all distances between gates in level 1 which are wired to gates in level 2. Assuming that a_3 is the designated output gate and that c_1, c_4, d_5 (c_2, c_3, respectively) receive the input value 0 (1, respectively), we have $I = \{2, 3\}$ and we obtain the formula

$$\psi = \mathrm{X}^2\mathrm{G}_{S_1}\mathrm{X}^5\ \mathrm{F}_{S_2}(\mathrm{X}^3\neg\mathrm{X}\mathrm{true} \vee \mathrm{X}^2\neg\mathrm{X}\mathrm{true}).$$

Proof of Theorem 5.4. The theorem is a direct consequence of Theorem 5.7 and Proposition 5.6. □

Similarly to the proof of Theorem 5.3, we can adapt the proof of Theorem 5.7 and extend the results to infinite data words.

Theorem 5.9. *Path checking for* $\mathrm{SMTL}(\mathrm{F}, \mathrm{X})_{\mathrm{u}}$ *and* $\mathrm{TPTL}(\mathrm{F}, \mathrm{X})^1_{\mathrm{u}}$ *over infinite unary encoded data words are* P*-hard.*

5.2. **PSPACE-Hardness Results.** Next we prove three PSPACE lower bounds, which complete the picture about the complexity of the path-checking problem for MTL and TPTL.

Theorem 5.10. *Path checking for* $\mathrm{PureTPTL}(\mathrm{F})_{\mathrm{u}}$ *over finite unary encoded strictly monotonic pure data words is* PSPACE*-hard.*

Proof. The proof is a reduction from the quantified Boolean formula problem: does a given closed formula $\Psi = Q_1 x_1 \dots Q_n x_n \phi$, where $Q_i \in \{\forall, \exists\}$, and ϕ is a quantifier-free propositional formula, evaluate to true? This problem is PSPACE-complete [14].

Let $\Psi = Q_1 x_1 \dots Q_n x_n \phi$ be an instance of the quantified Boolean formula problem. We construct the finite pure strictly monotonic data word

$$w = 0, 1, 2, \dots, 2n-1, 2n, 2n+1.$$

For every $i \in \{1, \dots, n\}$, the subword $2i-1, 2i$ is used to quantify over the Boolean variable x_i.

For the formula, we use a register variable x and register variables x_i corresponding to the variables used in Ψ, for every $i \in \{1, \dots, n\}$. Intuitively, if we assign to register variable x_i the data value $2i - b$, then the corresponding Boolean variable x_i is set to $b \in \{0, 1\}$.

We define the $\mathrm{PureTPTL}(\mathrm{F})_{\mathrm{u}}$ formula $x.x_1.x_2.\dots.x_n.\Psi'$, where Ψ' is defined inductively by the following rules.

- If $\Psi = \forall x_i \Phi$, then $\Psi' = \mathrm{G}((x_i = 2i - 1 \vee x_i = 2i) \to x_i.\Phi')$.

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

$$\Psi' = \mathrm{F}(x = 2n+1 \wedge \Psi[x_1/x_1 = 2n, \ldots, x_i/x_i = 2(n-i)+2, \ldots, x_n/x_n = 2]).$$

  Here, $\Psi[x_1/x_1 = a_0, \ldots, x_n/x_n = a_n]$ denotes the TPTL formula obtained from Ψ by replacing every occurrence of x_i by $x_i = a_i$ for every $i \in \{1, \ldots, n\}$.

Recall that the subformula $x_i = 2i-1 \vee x_i = 2i$ is `true` if, and only if, the difference between the current data value and the value to which x_i is bound (and which, initially, is set to 0) is $2i-1$ or $2i$. Hence, the subformula is only true at the two positions where the data values are $2i-1$ and $2i$, respectively. Clearly, Ψ is true if, and only if, $w \models_{\mathrm{TPTL}} x.\Psi'$. □

The *quantified subset sum problem* [29] (QSS, for short) is to decide for a given sequence $a_1, a_2, \ldots, a_{2n}, b \in \mathbb{N}$ of binary encoded numbers, whether

$$\forall x_1 \in \{0, a_1\} \exists x_2 \in \{0, a_2\} \ldots \forall x_{2n-1} \in \{0, a_{2n-1}\} \exists x_{2n} \in \{0, a_{2n}\} : \sum_{i=1}^{2n} x_i = b$$

holds. This problem is PSPACE-complete [29].

We define a variant of QSS, called *positive quantified subset sum problem* (PQSS, for short), in which for a given sequence $a_1, a_2, \ldots, a_{2n}, b \in \mathbb{N} \setminus \{0\}$ of binary encoded numbers, we want to decide whether

$$\forall x_1 \in \{1, a_1\} \exists x_2 \in \{1, a_2\} \ldots \forall x_{2n-1} \in \{1, a_{2n-1}\} \exists x_{2n} \in \{1, a_{2n}\} : \sum_{i=1}^{2n} x_i = b.$$

One can easily see that QSS and PQSS are polynomial time-interreducible, and thus PQSS is PSPACE-complete.

Theorem 5.11. *Path checking for* $\mathrm{PureTPTL}^2(\mathrm{F})_{\mathrm{b}}$ *over the infinite strictly monotonic pure data word* $w = 0(1)^{\omega}_{+1} = 0, 1, 2, 3, 4, \ldots$ *is* PSPACE*-hard.*

Proof. The theorem is proved by a reduction from PQSS. Given an instance $a_1, a_2, \ldots, a_{2n}, b$ of PQSS, we construct the $\mathrm{PureTPTL}^2(\mathrm{F})_{\mathrm{b}}$ formula $x.\varphi_1$, where the formula φ_i, for every $i \in \{1, \ldots, , 2n+1\}$, is defined inductively by

$$\varphi_i = \begin{cases} y.\mathrm{G}((y = 1 \vee y = a_i) \rightarrow \varphi_{i+1}) & \text{for } i < 2n \text{ odd}, \\ y.\mathrm{F}((y = 1 \vee y = a_i) \wedge \varphi_{i+1}) & \text{for } i \leq 2n \text{ even}, \\ x = b & \text{for } i = 2n+1. \end{cases}$$

The intuition is the following: note that in the data word w the data value is increasing by one in each step. Assume we want to evaluate $y.\mathrm{G}((y = 1 \vee y = a_i) \rightarrow \varphi_{i+1})$ in a position where the data value is currently d. The initial freeze quantifier sets y to d. Then, $\mathrm{G}((y = 1 \vee y = a_i) \rightarrow \varphi_{i+1})$ means that in every future position, where the current data value is either $d+1$ (in such a position $y = 1$ holds by the TPTL-semantics) or $d + a_i$ (in such a position $y = a_i$ holds), the formula φ_{i+1} has to hold. In this way, we simulate the quantifier $\forall x_i \in \{1, a_i\}$. At the end, we have to check that the current data value is b, which can be done with the constraint $x = b$ (note that x is initially set to 0 and never reset). One can show that $(0)^{\omega}_{+1} \models_{\mathrm{TPTL}} x.\varphi_1$ if, and only if, $(a_1, a_2, \ldots, a_{2n}, b)$ is a positive instance of PQSS. □

Theorem 5.12. *Path checking for* FreezeLTL^2 *over infinite binary encoded pure data words is* PSPACE*-hard.*

Proof. The proof is by a reduction from PQSS. We first prove the claim for infinite binary encoded data words. Then we show how one can change the proof to obtain the result for *pure* data words.

Given an instance $a_1, a_2, \ldots, a_{2n}, b$ of PQSS, we define the infinite data word

$$w := (r,b)\left((q,0)\prod_{i=1}^{2n}(p,1)(p,a_i)(r,0)\right)^{\omega}_{+1}$$

For defininig the FreezeLTL^2 formula, we first define for every FreezeLTL^2 formula ψ the auxiliary formulas

$$\mathrm{F}_p\psi := p\mathrm{U}(p \wedge \psi) \text{ and } \mathrm{G}_p\psi := \neg\mathrm{F}_p\neg\psi.$$

The formula $\mathrm{F}_p\psi$ holds in position i if, and only if, there exists a future position $j > i$ such that ψ holds, and p holds in all positions $k \in \{i+1, \ldots, j\}$. Note that the formula $\mathrm{G}_p\psi$ is equivalent to $\neg p\mathrm{R}(p \to \psi)$; it thus holds in a position i if, and only if, for all future positions $j > i$, ψ holds in position j whenever p holds in all positions $k \in \{i+1, \ldots, j\}$.

Define for every $i \in \{1, \ldots, 2n\}$ the FreezeLTL^2 formula φ_i as follows:

$$\varphi_i := \begin{cases} \mathrm{X}^{3(i-1)}\mathrm{G}_p y.\mathrm{F}(q \wedge y = 0 \wedge \varphi_{i+1}) & \text{for } i < 2n \text{ odd}, \\ \mathrm{X}^{3(i-1)}\mathrm{F}_p y.\mathrm{F}(q \wedge y = 0 \wedge \varphi_{i+1}) & \text{for } i \leq 2n \text{ even}, \\ x = 0 & \text{for } i = 2n+1. \end{cases}$$

Finally set $\varphi = x.y.\mathrm{X}\varphi_1$.

Note that in φ, we require the auxiliary formulas $\mathrm{F}_p\psi$ and $\mathrm{G}_p\psi$ to hold in w *only* at positions in which also q holds, *i.e.*, at positions corresponding to the beginning of the periodic part of w. Plainly put, formula $\mathrm{F}_p\psi$ holds at some position in w in which also q holds, if, and only if, ψ holds at the next *or* the next but one position; analogously, the formula $\mathrm{G}_p\psi$ holds, if, and only if, ψ holds at the next *and* the next but one position. Note that we resign from using the next modality here to avoid an exponential blow-up of the formula.

We explain the idea of the reduction. Assume $w \models_{\mathrm{TPTL}} \varphi$. Then, formula φ_1 holds at the beginning of the first iteration of the periodic part of w, *i.e.*, at the position with letter $(q,0)$. By formula $\mathrm{G}_p y.(q \wedge y = 0 \wedge \varphi_2)$, we know that $y.(q \wedge y = 0 \wedge \varphi_2)$ holds both at the position with letter $(p,1)$ *and* at the position with letter (p,a_1) (universal quantification by the G_p-modality). This is the case if, and only if, φ_2 holds at the beginning of the 2nd iteration of the periodic part of w (*i.e.*, the position with letter $(q,1)$) *and* at the beginning of the (a_1+1)-th iteration of the periodic part of w (the position with letter (q,a_1)). In the former case, we can conclude that the formula $y.\mathrm{F}(q \wedge y = 0 \wedge \varphi_3)$ holds at the position with letter $(p, 1+1)$ *or* at the position with letter $(p, 1+a_2)$ (existential quantification of the F_p-modality); in the second case, the formula $y.\mathrm{F}(q \wedge y = 0 \wedge \varphi_3)$ holds at the position with letter (p, a_1+1) *or* at the position with letter (p, a_1+a_2) (again, existential quantification of the F_p-modality). Note how the intermediate sums computed so far, *i.e.*, $(1+1$ or $1+a_2$, and, a_1+1 or $a_1+a_2)$, are stored in the corresponding data values with proposition p.

Note that the register variable x is set to the first data value occurring in w, which is b. Hence, in the formula φ_{2n}, the constraint $x = 0$ expresses that the current data value has to be b.

Clearly, one can prove that $w \models_{\mathrm{TPTL}} x.y.\mathrm{X}\varphi_1$ if, and only if, $a_1, a_2, \ldots, a_{2n}, b$ is a positive instance of PQSS.

Next, we explain how we can encode the propositional variables occurring in w and φ by data values, to obtain the result for pure data words and the pure logics. Note that r does not occur in φ. It thus suffices to encode q and p, respectively, which we do by $(0,1,1)$ and $(0,0,0)$, respectively. We obtain the pure data word w' from w by replacing all occurrences of p and q by their corresponding data values, and by removing r, as follows, where we underline the data values that occurred in the w:

$$w' := \underline{b}\left(0,1,1,\underline{0},\prod_{i=1}^{2n}(0,0,0,\underline{1},0,0,0,\underline{a_i},\underline{0})\right)^{\omega}_{+1}.$$

Define $\varphi_q = x.\mathrm{X}(\neg(x=0) \wedge x.\mathrm{X}(x=0))$ and $\varphi_p = x.\mathrm{X}(x=0 \wedge \mathrm{X}(x=0))$. We replace the formula $\mathrm{F}_p\psi$ by

$$\mathrm{F}'_p\psi = [\varphi_p \vee \mathrm{X}^3(\varphi_p \wedge \mathrm{X}\neg\varphi_p) \vee \mathrm{X}^2(\varphi_p \wedge \mathrm{X}\neg\varphi_p) \vee \mathrm{X}(\varphi_p \wedge \mathrm{X}\neg\varphi_p)]\mathrm{U}[\varphi_p \wedge \psi]$$

and define $\mathrm{G}'_p\psi = \neg\mathrm{F}'_p\neg\psi$. Then we define:

$$\varphi'_i := \begin{cases} \mathrm{X}^{9(i-1)}\,\mathrm{G}'_p\,\mathrm{X}^3 y.\mathrm{F}(\varphi_q \wedge \mathrm{X}^4\varphi_p \wedge \mathrm{X}^3(y=0) \wedge \mathrm{X}^3\varphi'_{i+1}) & \text{for } i < 2n \text{ odd},\\ \mathrm{X}^{9(i-1)}\,\mathrm{F}'_p\,\mathrm{X}^3 y.\mathrm{F}(\varphi_q \wedge \mathrm{X}^4\varphi_p \wedge \mathrm{X}^3(y=0) \wedge \mathrm{X}^3\varphi'_{i+1}) & \text{for } i \leq 2n \text{ even},\\ x=0 & \text{for } i = 2n+1. \end{cases}$$

Analysing the formulas yields that $w' \models_{\mathrm{TPTL}} x.y.\mathrm{X}^4\,\varphi'_1$ if, and only if, $a_1, a_2, \ldots, a_{2n}, b$ is positive instance of PQSS. □

6. Model Checking for Deterministic One-Counter Machines

A *one-counter machine* (OCM, for short) is a nondeterministic finite-state machine extended with a single counter that takes values in the non-negative integers. Formally, a one-counter machine is a tuple $\mathcal{A} = (Q, q_0, E)$, where Q is a finite set of control states, $q_0 \in Q$ is the initial control state, and $E \subseteq Q \times \mathsf{Op} \times Q$ is a finite set of labelled edges, where $\mathsf{Op} = \{\mathsf{zero}\} \cup \{\mathsf{add}(a) \mid a \in \mathbb{Z}\}$. We use the operation zero to test whether the current value of the counter is equal to zero, and we use $\mathsf{add}(a)$ for adding a to the current value of the counter. A *configuration* of the one-counter machine $\mathcal{A}$ is a pair (q, c), where $q \in Q$ is a state and $c \in \mathbb{N}$ is the current value of the counter. We define a transition relation $\to_{\mathcal{A}}$ over the set of all configurations by $(q,c) \to_{\mathcal{A}} (q',c')$ if, and only if, there is an edge $(q, \mathsf{op}, q') \in E$ and one of the following two cases holds:

- $\mathsf{op} = \mathsf{zero}$ and $c = c' = 0$,
- $\mathsf{op} = \mathsf{add}(a)$ and $c' = c + a \geq 0$.

If $\mathcal{A}$ is clear from the context, we write $\to$ for $\to_{\mathcal{A}}$. A *finite computation* of $\mathcal{A}$ is a finite sequence $(q_0,c_0)(q_1,c_1)\ldots(q_n,c_n)$ over $Q \times \mathbb{N}$ such that $c_0 = 0$ and $(q_i,c_i) \to (q_{i+1},c_{i+1})$ for all $i \in \{0,\ldots,n-1\}$, and such that there does not exist a configuration (q,c) with $(q_n,c_n) \to (q,c)$. We identify such a computation with the finite data word of the same form. An *infinite computation* of $\mathcal{A}$ is an infinite sequence $(q_0,c_0)(q_1,c_1)\ldots$ over $Q \times \mathbb{N}$ such that, again, $c_0 = 0$, and $(q_i,c_i) \to (q_{i+1},c_{i+1})$ for all $i \geq 0$. We identify such a computation with the infinite data word of the same form. A *deterministic one-counter machine* $\mathcal{A} = (Q, q_0, E)$, briefly DOCM, is an OCM such that for every configuration (q,c) with $(q_0,0) \to^* (q,c)$ there is *at most one* configuration (q',c') such that $(q,c) \to (q',c')$. This implies that

$\mathcal{A}$ has a unique (finite or infinite) computation, which we denote by $\mathsf{comp}(\mathcal{A})$, and which we view as a data word as explained above. For complexity considerations, it makes a difference whether the numbers $a \in \mathbb{Z}$ in operations $\mathsf{add}(a)$ occurring at edges in $\mathcal{A}$ are encoded in unary or in binary. We will therefore speak of unary encoded (resp., binary encoded) OCMs in the following. Let $\alpha(\mathcal{A})$ be the largest number a such that $\mathsf{add}(a)$ appears in an edge from E. We use the following lemma from [9]

Lemma 6.1 ([9, Lemma 9])**.** *Let $\mathcal{A} = (Q, q_0, E)$ be a DOCM. Then, the following holds:*

- *If $\mathsf{comp}(\mathcal{A})$ is infinite then $\mathsf{comp}(\mathcal{A}) = u_1(u_2)^\omega_{+k}$ with $0 \leq k \leq |Q|$ and $|u_1u_2| \leq \alpha(\mathcal{A}) \cdot |Q|^3$.*
- *If $\mathsf{comp}(\mathcal{A})$ is finite then $|\mathsf{comp}(\mathcal{A})| \leq \alpha(\mathcal{A}) \cdot |Q|^3$.*

Proof. The first statement is shown in [9] for a DOCM $\mathcal{A}$ with the operations zero, $\mathsf{add}(-1)$ and $\mathsf{add}(1)$ (and hence $\alpha(\mathcal{A}) = 1$). To get the full first statement of the lemma, it suffices to simulate every $\mathsf{add}(a)$ operation by at most $\alpha(\mathcal{A})$ many operations $\mathsf{add}(-1)$ or $\mathsf{add}(1)$. To the resulting DOCM one can then apply [9, Lemma 9].

The second statement is implicitly shown in the proof of [9, Lemma 9]. Like for the proof of the first statement, it suffices to consider a DOCM such that $a \in \{1, -1\}$ for all instructions $\mathsf{add}(a)$ and afterwards multiply the length of $\mathsf{comp}(\mathcal{A})$ by $\alpha(\mathcal{A})$. Assume that $\mathsf{comp}(\mathcal{A})$ is finite and let $\mathsf{comp}(\mathcal{A}) = (q_0, c_0)(q_1, c_1) \dots (q_n, c_n)$. Consider $i < j$ such that $c_i = c_j = 0$. By [9, Lemma 10], we must have $j - i \leq |Q|^2$. Moreover, there can be at most $|Q|$ many $i \geq 0$ such that $c_i = 0$. Let k be maximal such that $c_k = 0$. From the previous discussion, we get $k \leq |Q|^2(|Q| - 1)$.

Assume that there exist $j > i > k$ with $q_i = q_j$. If $c_i \leq c_j$ then $\mathsf{comp}(\mathcal{A})$ would be infinite, and if $c_i > c_j$ then the counter would hit zero again, which contradicts the choice of k. It follows that $n \leq k + |Q| - 1$ and hence $|\mathsf{comp}(\mathcal{A})| = n + 1 \leq k + |Q| \leq |Q|^2(|Q| - 1) + |Q| \leq |Q|^3$. □

For unary encoded DOCMs we make use of the following result:

Lemma 6.2. *For a given unary encoded DOCM $\mathcal{A} = (Q, q_0, E)$ one can check in logspace, whether $\mathsf{comp}(\mathcal{A})$ is finite or infinite. Moreover, the following holds:*

- *If $\mathsf{comp}(\mathcal{A})$ is finite, then the corresponding data word in unary encoding can be computed in logspace.*
- *If $\mathsf{comp}(\mathcal{A})$ 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 $\mathsf{comp}(\mathcal{A}) = u_1(u_2)^\omega_{+k}$.*

Proof. In order to check whether $\mathsf{comp}(\mathcal{A})$ is infinite, it suffices by Lemma 6.1 to simulate $\mathcal{A}$ for at most $\alpha(\mathcal{A}) \cdot |Q|^3$ many steps. For this we store (i) the current configuration (q, c) with c encoded in binary notation and (ii) a step counter t in binary notation, which is initially zero and incremented after each transition of $\mathcal{A}$. The algorithm stops if (q, c) has no successor configuration or t reaches the value $\alpha(\mathcal{A}) \cdot |Q|^3$. In the latter case, $\mathsf{comp}(\mathcal{A})$ is infinite. Note that the counter value c is bounded by $\alpha(\mathcal{A})^2 \cdot |Q|^3$. Therefore, logarithmic space suffices to store (q, c) and t. More precisely, (q, c) can be stored with $4 \log |Q| + 2 \log \alpha(\mathcal{A})$ bits (note that in the input representation, $\alpha(\mathcal{A})$ is represented with $\alpha(\mathcal{A})$ bits) and t can be stored with $3 \log |Q| + \log \alpha(\mathcal{A})$ bits. In a similar way, we can produce the data word $\mathsf{comp}(\mathcal{A})$ itself in logarithmic space. We only have to print out the current configuration in each step. Internally, our machine stores counter values in binary encoding. Since we want to output the data word in unary encoding, we transform the binary encoded counter values into

unary encoding, which can be done with a logspace machine. In case $\mathsf{comp}(\mathcal{A})$ is finite, the machine outputs the unary encoding of $\mathsf{comp}(\mathcal{A})$ in this way. In case $\mathsf{comp}(\mathcal{A})$ is infinite, we can output with a first logspace machine the unary encoded data word consisting of the first $\alpha(\mathcal{A}) \cdot |Q|^3$ many configurations. From this data word, a second logspace machine can easily compute two unary encoded data words u_1 and u_2 and a unary encoded number k such that $\mathsf{comp}(\mathcal{A}) = u_1(u_2)^\omega_{+k}$. We then use the fact that the composition of two logspace machines can be compiled into one logspace machine. □

In order to extend Lemma 6.2 to binary encoded DOCMs, we need SLPs:

Lemma 6.3. *For a given binary encoded DOCM $\mathcal{A}$ one can check in polynomial time, whether $\mathsf{comp}(\mathcal{A})$ is finite or infinite. Moreover, the following holds:*

- *If $\mathsf{comp}(\mathcal{A})$ is finite, then an SLP $\mathcal{G}$ with $\mathsf{val}(\mathcal{G}) = \mathsf{comp}(\mathcal{A})$ can be computed in polynomial time.*
- *If $\mathsf{comp}(\mathcal{A})$ is infinite, then one can compute in polynomial time two SLPs $\mathcal{G}_1$ and $\mathcal{G}_2$ and a binary encoded number k such that $\mathsf{comp}(\mathcal{A}) = \mathsf{val}(\mathcal{G}_1)(\mathsf{val}(\mathcal{G}_2))^\omega_{+k}$.*

For the proof of Lemma 6.3 we need the following lemma:

Lemma 6.4. *Let u be a finite data word, and let $m \in \mathbb{N}$ and $k \in \mathbb{Z}$ be binary encoded numbers such that $d + ik \geq 0$ for all data values d occuring in u and all $0 \leq i \leq m$. From u, m, and k one can construct in polynomial time an SLP for the data word $\prod_{i=1}^m u_{+ik}$.*

Proof. We first consider the case $k > 0$. First, assume that $m = 2^n$ for some $n \geq 0$. If $U_n = \prod_{i=1}^{2^n} u_{+ik}$ then we obtain the following recurrence:

$$U_0 = u_{+k} \text{ and } U_{n+1} = U_n(U_n)_{+2^n k}.$$

This recurrence can be directly translated into an SLP. Second, assume that m is not necessarily a power of two and let $m = 2^{n_1} + 2^{n_2} + \cdots + 2^{n_l}$ be the binary expansion of m, where $n_1 < n_2 < \cdots < n_l$. Let $m_j = 2^{n_1} + 2^{n_2} + \cdots + 2^{n_j}$ for $1 \leq j \leq l$. If $V_j = \prod_{i=1}^{m_j} u_{+ik}$ then we obtain the following recurrence:

$$V_1 = U_{n_1} \text{ and } V_{j+1} = V_j(U_{n_{j+1}})_{+m_j k}.$$

Again, this recurrence can be directly translated into an SLP.

Let us finally show how to reduce the case $k < 0$ to the case $k > 0$ (the case $k = 0$ is easier). Assume that $k < 0$ and let $v = (u_{+(m+1)k})^{\mathrm{rev}}$, where w^{rev} denotes the data word w reversed. Then, for $l = -k > 0$ we get the following identity:

$$\prod_{i=1}^{m} u_{+ik} = (\prod_{i=1}^{m} v_{+il})^{\mathrm{rev}}.$$

From an SLP for $\prod_{i=1}^m v_{+il}$ it is easy to compute in polynomial time an SLP for $(\prod_{i=1}^m v_{+il})^{\mathrm{rev}}$: one just has to replace every right-hand side of the form BC by CB. This shows the lemma. □

Proof of Lemma 6.3. Fix a DOCM $\mathcal{A} = (Q, q_0, E)$. We define below a procedure $\mathsf{comp}(q)$, where $q \in Q$. This procedure constructs an SLP for the unique computation that starts in the configuration $(q, 0)$. Hence the call $\mathsf{comp}(q_0)$ computes an SLP for $\mathsf{comp}(\mathcal{A})$. The algorithm stores as an auxiliary data structure a graph G with vertex set $Q \cup \{f\}$ (where $f \notin Q$) which initially is the empty graph and to which edges labelled with data words are added. These data words will be represented by SLPs. The idea is that an edge $q \xrightarrow{u} p$

is added if the data word u represents a computation of $\mathcal{A}$ from configuration $(q,0)$ to configuration $(p,0)$. The overall algorithm terminates as soon as (i) an edge to node f is added or (ii) the graph G contains a cycle.

The call $\mathsf{comp}(q)$ starts simulating $\mathcal{A}$ in the configuration $(q,0)$. Let (p_i,c_i) be the configuration reached after $i \geq 0$ steps (thus, $p_0 = q$). After at most $|Q|+1$ transitions, one of the following situations has to occur:

- There exists $0 \leq i \leq |Q|$ such that that the computation terminates with (p_i,c_i). We add the edge $q \xrightarrow{u} f$ to the graph G, where $u = (p_0,0)(p_1,c_1)\dots(p_i,c_i)$ and the call $\mathsf{comp}(q)$ terminates.
- There exists $1 \leq i \leq |Q|+1$ such that $c_i = 0$. We add the edge $q \xrightarrow{u} p_i$ to the graph G, where $u = (p_0,0)(p_1,c_1)\dots(p_{i-1},c_{i-1})$ and call $\mathsf{comp}(p_i)$.
- $c_i > 0$ for all $1 \leq i \leq |Q|+1$ and there exists $1 \leq j < l \leq |Q|+1$ such that $p_j = p_l$ and $c_j \leq c_l$. We add the edge $q \xrightarrow{u} f$ to the graph G, where $u = u_1(u_2)^{\omega}_{+k}$, $u_1 = (p_0,0)(p_1,c_1)\dots(p_{j-1},c_{j-1})$, $u_2 = (p_j,c_j)(p_{j+1},c_{j+1})\dots(p_{l-1},c_{l-1})$ and $k = c_l - c_j$, and the call $\mathsf{comp}(q)$ terminates.
- $c_i > 0$ for all $1 \leq i \leq |Q|+1$ and there exists $1 \leq j < l \leq |Q|+1$ such that $p_j = p_l$ and $c_j > c_l$. Let $k = c_l - c_j < 0$. We compute in polynomial time the binary encoding of the largest number $m \geq 0$ such that $c_i + mk > 0$ for all $j \leq i \leq l-1$. That means that the computation of $\mathcal{A}$ starts from $(q,0)$ with the data word $u\prod_{i=0}^{m} v_{+ik}$, where $u = (p_0,0)(p_1,c_1)\dots(p_{j-1},c_{j-1})$ and $v = (p_j,c_j)(p_{j+1},c_{j+1})\dots(p_{l-1},c_{l-1})$.

  Moreover, by the choice of m, there exists an $i \in [j,l-1]$ such that $c_i + (m+1)k \leq 0$. Let i be minimal with this property and define

  $$w = (p_j, c_j + (m+1)k)\dots(p_{i-1}, c_{i-1} + (m+1)k).$$

  This implies that the computation of $\mathcal{A}$ starts from $(q,0)$ with the data word $u(\prod_{i=0}^{m} v_{+ik})w$. Moreover, if $c_i + (m+1)k < 0$ the computation terminates in the configuration $(p_{i-1}, c_{i-1} + (m+1)k)$ and we add to G an edge from q to f labelled with $u(\prod_{i=0}^{m} v_{+ik})w$. On the other hand, if $c_i + (m+1)k = 0$ then we add to G an edge from q to p_i labelled with $u(\prod_{i=0}^{m} v_{+ik})w$ and call $\mathsf{comp}(p_i)$.

We did not make the effort to make the above four cases non-overlapping; ties are broken in an arbitrary way. As mentioned above we start the overall algorithm with the call $\mathsf{comp}(q_0)$ and terminate as soon as (i) an edge to node f is added or (ii) the graph G contains a cycle. This ensures that there is a unique path in G starting in q_0 that either ends in f (in case (i)) or enters a cycle (in case (ii)). In case (i) the data word $\mathsf{comp}(\mathcal{A})$ is $u_1u_2\dots u_n$, where the data words $u_1, u_2, \dots, u_n$ label the path from q_0 to f (note that u_n can be an infinite data word). In case (ii) the data word $\mathsf{comp}(\mathcal{A})$ is $u_1u_2\dots u_n(u_{n+1}\dots u_m)^{\omega}_{+0}$, where the data words $u_1, u_2, \dots, u_n$ label the path from q_0 to the first state on the cycle, and $u_{n+1}, \dots, u_m$ label the cycle. Finally, note that all data words that appear as labels in G can be represented by SLPs that can be computed in polynomial time. For the label $u(\prod_{i=0}^{m} v_{+ik})w$ this follows from Lemma 6.4. □

The algorithm from the above proof can be also used to check in polynomial time whether a given binary encoded OCM is deterministic, which is not clear from our definition of DOCMs. We run the same algorithm as in the above proof but check in every step whether more than one successor configuration exists.

Let $\mathbb{L}$ be one of the logics considered in this paper. The *model-checking problem for* $\mathbb{L}$ *over unary (resp., binary) encoded DOCMs* asks for a given unary (resp., binary) encoded DOCM $\mathcal{A}$ and a formula $\varphi \in \mathbb{L}$, whether $\mathsf{comp}(\mathcal{A}) \models_{logic} \varphi$ holds. Based on Lemma 6.2 we can now easily show:

Theorem 6.5. *Let* $\mathbb{L}$ *be one of the logics considered in this paper. The model-checking problem for* $\mathbb{L}$ *over unary encoded DOCMs is equivalent with respect to logspace reductions to the path-checking problem for* $\mathbb{L}$ *over infinite unary encoded data words.*

Proof. The reduction from the model-checking problem for $\mathbb{L}$ over DOCMs to the path-checking problem for $\mathbb{L}$ over infinite unary encoded data words follows from Lemma 6.2. For the other direction take a unary encoded infinite data word $w = u_1(u_2)^{\omega}_{+k}$ and a formula $\psi \in \mathbb{L}$. It is straightforward to construct (in logspace) from u_1, u_2 a unary encoded DOCM $\mathcal{A}$ such that the infinite sequence of counter values produced by $\mathcal{A}$ is equal to the sequence of data values in w with an initial 0 (that comes from the initial configuration $(q_0, 0)$) added. Moreover, no state of $\mathcal{A}$ repeats among the first $|u_1u_2|$ many positions in $\mathsf{comp}(\mathcal{A})$. Hence, by replacing every proposition in ψ by a suitable disjunction of states of $\mathcal{A}$ we easily obtain a formula $\psi' \in \mathbb{L}$ such that $w \models \psi$ if, and only if, $\mathsf{comp}(\mathcal{A}) \models \psi'$. □

By Theorem 6.5, the left diagram from Figure 7 also shows the complexity results for TPTL-model checking over DOCMs.

Finally, for binary encoded DOCMs, Lemma 6.3 and Theorem 4.9 directly imply the following result (PSPACE-hardness follows by a reduction similar to the one from the proof of Theorem 6.5):

Theorem 6.6. *The model-checking problem for* TPTL *over binary encoded DOCMs is* PSPACE*-complete.*

7. Summary and Open Problems

Figure 7 collects our complexity results for path-checking problems. We use $\mathrm{TPTL}^{\geq 2}$ to denote the fragments in which at least 2 registers are used. We observe that in all cases whether data words are pure or not does not change the complexity. For finite data words, the complexity does not depend upon the encoding of data words (unary or binary), and for TPTL and SMTL, it does not depend on whether a data word is monotonic or not. In contrast to this, for infinite data words, these distinctions indeed influence the complexity: for binary encoded data words the complexity picture looks different from the picture for unary encoded or (quasi-)monotonic data words.

We leave open the precise complexity of the model-checking problem for MTL and TPTL^1 over DOCMs. Model checking MTL over data words that are represented by SLPs is PSPACE-complete (and this even holds for LTL, see [24]), but the SLPs that result from DOCMs have a very simple form (see the proof of Lemma 6.2). This may be helpful in proving a polynomial time upper bound.

Kuhtz proved in his thesis [20] that the tree checking problem for CTL (i.e., the question whether a given CTL formula holds in the root of a given tree) can be solved in $\mathrm{AC}^2(\mathrm{LogDCFL})$. One might try to combine this result with the $\mathrm{AC}^1(\mathrm{LogDCFL})$-algorithm of Bundala and Ouaknine [5] for MTL path checking over monotonic data words. There is an obvious CTL-variant of MTL that might be called MCTL, which to our knowledge has not been studied so far. Then, the question is whether the tree checking problem for MCTL

is in AC^2(LogDCFL). Here the tree nodes are labelled with data values and this labelling should be monotonic in the sense that if v is a child of u, then the data value of v is at least as large as the data value of u.

Similarly to MTL, also TPTL has a CTL-variant. One might try extend our polynomial time path checking algorithm for the one-variable TPTL-fragment to this CTL-variant.

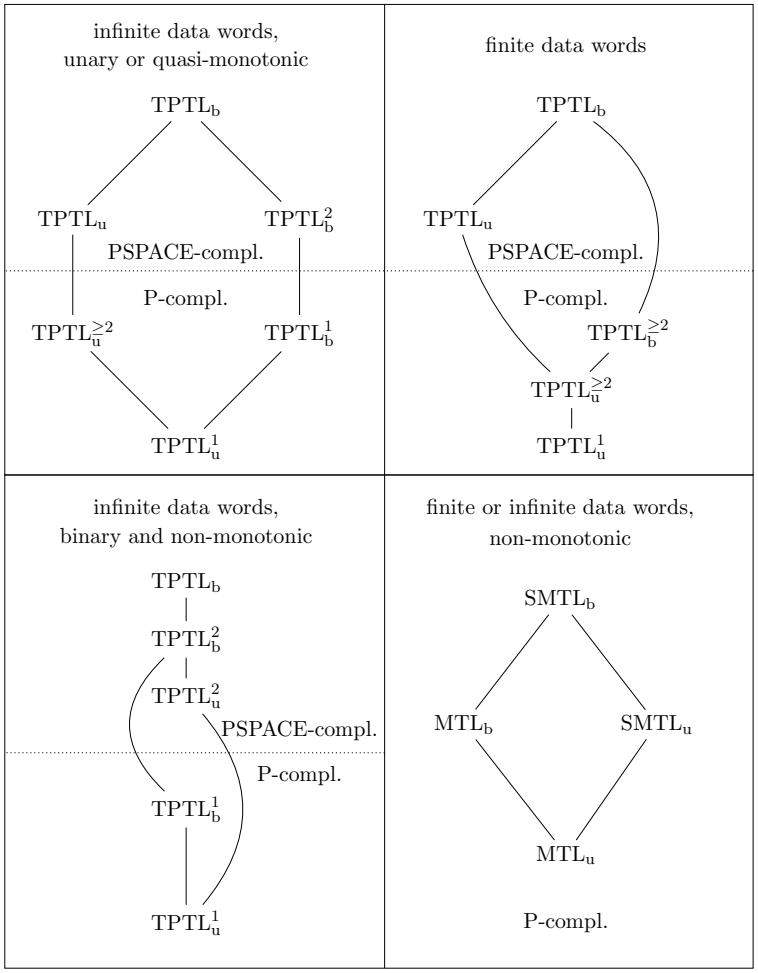

Figure 7: Complexity results of path checking

References

[1] Rajeev Alur and Thomas A. Henzinger. Real-time logics: Complexity and expressiveness. *Information and Computation*, 104(1):35–77, 1993.

[2] Rajeev Alur and Thomas A. Henzinger. A really temporal logic. *Journal of the ACM*, 41(1):181–204, 1994.

[3] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.

[4] Patricia Bouyer, Kim Guldstrand Larsen, and Nicolas Markey. Model checking one-clock priced timed automata. *Logical Methods in Computer Science*, 4(2), 2008.

[5] Daniel Bundala and Joël Ouaknine. On the complexity of temporal-logic path checking. In *Proceedings of the 41st International Colloquium Automata, Languages, and Programming, ICALP 2014, Part II*, volume 8573 of *LNCS*, pages 86–97. Springer, 2014.

[6] Claudia Carapelle, Shiguang Feng, Oliver Fernández Gil, and Karin Quaas. Ehrenfeucht-Fraïssé games for TPTL and MTL over non-monotonic data words. In *Proceedings 14th International Conference on Automata and Formal Languages, AFL 2014*, volume 151 of *EPTCS*, pages 174–187, 2014.

[7] Claudia Carapelle, Shiguang Feng, Oliver Fernández Gil, and Karin Quaas. Satisfiability for MTL and TPTL over non-monotonic data words. In *Proceedings of the 8th International Conference on Language and Automata Theory and Applications, LATA 2014*, volume 8370 of *LNCS*, pages 248–259. Springer, 2014.

[8] Stéphane Demri and Ranko Lazic. LTL with the freeze quantifier and register automata. *ACM Transactions on Computational Logic*, 10(3), 2009.

[9] Stéphane Demri, Ranko Lazic, and Arnaud Sangnier. Model checking memoryful linear-time logics over one-counter automata. *Theoretical Computer Science*, 411(22-24):2298–2316, 2010.

[10] Kousha Etessami and Thomas Wilke. An until hierarchy and other applications of an Ehrenfeucht-Fraïssé game for temporal logic. *Information and Computation*, 160(1-2):88–108, 2000.

[11] Shiguang Feng, Markus Lohrey, and Karin Quaas. Path checking for MTL and TPTL over data words. In *Proceedings of the 19th International Conference on Developments in Language Theory, DLT 2015*, volume 9168 of *LNCS*, pages 326–339. Springer, 2015.

[12] Diego Figueira, Santiago Figueira, Sylvain Schmitz, and Philippe Schnoebelen. Ackermannian and primitive-recursive bounds with Dickson's lemma. In *Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science, LICS 2011*, pages 269–278. IEEE Computer Society, 2011.

[13] Bernd Finkbeiner and Henny Sipma. Checking finite traces using alternating automata. *Electronic Notes in Theoretical Computer Science*, 55(2):147–163, 2001.

[14] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, 1979.

[15] Stefan Göller, Christoph Haase, Joël Ouaknine, and James Worrell. Model checking succinct and parametric one-counter automata. In *Proceedings of the 37th International Colloquium on Automata, Languages and Programming, ICALP 2010, Part II*, volume 6199 of *LNCS*, pages 575–586. Springer, 2010.

[16] Raymond Greenlaw, H. James Hoover, and Walter L. Ruzzo. *Limits to Parallel Computation: P-Completeness Theory*. Oxford University Press, Inc., New York, NY, USA, 1995.

[17] Christoph Haase, Stephan Kreutzer, Joël Ouaknine, and James Worrell. Reachability in succinct and parametric one-counter automata. In *Proceedings of the 20th International Conference on Concurrency Theory, CONCUR 2009*, volume 5710 of *LNCS*, pages 369–383. Springer, 2009.

[18] William Hesse, Eric Allender, and David A. Mix Barrington. Uniform constant-depth threshold circuits for division and iterated multiplication. *Journal of Computer and System Sciences*, 65(4):695–716, 2002.

[19] Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.

[20] Lars Kuhtz. *Model Checking Finite Paths and Trees*. PhD thesis, Universität des Saarlandes, 2010.

[21] Lars Kuhtz and Bernd Finkbeiner. LTL path checking is efficiently parallelizable. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming, ICALP 2009, Part II*, volume 5556 of *LNCS*, pages 235–246. Springer, 2009.

[22] François Laroussinie, Nicolas Markey, and Philippe Schnoebelen. On model checking durational Kripke structures. In *Proceedings of the 5th International Conference on Foundations of Software Science and Computation Structures, FOSSACS 2002*, volume 2303 of *LNCS*, pages 264–279. Springer, 2002.

[23] Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems, Joint International Conferences on Formal Modelling and Analysis of Timed Systems, FORMATS 2004 and Formal Techniques in Real-Time and Fault-Tolerant Systems, FTRTFT 2004, Proceedings*, volume 3253 of *LNCS*, pages 152–166. Springer, 2004.

[24] Nicolas Markey and Philippe Schnoebelen. Model checking a path. In *Proceedings of the 14th International Conference on Concurrency Theory, CONCUR 2003*, volume 2761 of *LNCS*, pages 248–262. Springer, 2003.

[25] Joël Ouaknine and James Worrell. On metric temporal logic and faulty Turing machines. In *Proceedings of the 9th International Conference on Foundations of Software Science and Computation Structures, FOSSACS, 2006*, volume 3921 of *LNCS*, pages 217–230. Springer, 2006.

[26] Joël Ouaknine and James Worrell. On the decidability and complexity of metric temporal logic over finite words. *Logical Methods in Computer Science*, 3(1), 2007.

[27] Karin Quaas. Model checking metric temporal logic over automata with one counter. In *Proceedings of the 7th International Conference on Language and Automata Theory and Applications, LATA 2013*, volume 7810 of *LNCS*, pages 468–479. Springer, 2013.

[28] Sylvain Schmitz, and Philippe Schnoebelen. Multiply-Recursive Upper Bounds with Higman's Lemma. In *Proceedings of the 38th International Colloquium on Automata, Languages and Programming, ICALP 2011*, volume 6756 of *LNCS*, pages 441–452, 2011. Springer, 2011.

[29] Stephen Travers. The complexity of membership problems for circuits over sets of integers. *Theoretical Computer Science*, 369(1):211–229, 2006.