Path Checking for MTL and TPTL over Data Words

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

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

Abstract. Precise complexity results are derived for the model checking problems for MTL and TPTL on (in)finite data words and deterministic one-counter machines. Depending on the number of register variables and the encoding of constraint numbers (unary or binary), the complexity is P-complete or PSPACE-complete. Proofs can be found in the long version [10].

1 Introduction

Linear time 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) [13] and TPTL (timed propositional temporal logic) [2]. In MTL, the operators next (X) and until (U) are indexed by time intervals. For instance, the formula $p \,\mathsf{U}_{[2,3)}\, q$ 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 that is equipped with a freeze formalism. It uses register variables, which can be set to the current time value and later these register variables can be compared with the current time value. For instance, the above MTL-formula $p \,\mathsf{U}_{[2,3)}\, q$ is equivalent to the TPTL-formula $x.(p \,\mathsf{U}\, (q \wedge 2 \leq x < 3))$. Here, the constraint $2 \leq x < 3$ should be read as: The difference of the current time value and the value stored in x is in the interval $[2, 3)$. In this paper, we always use the discrete semantics (opposed to the continuous semantics), where formulae are interpreted over (in)finite timed sequences $(P_0, d_0)(P_1, d_1) \ldots$, where the d_i are time stamps and the P_i are sets of atomic propositions.

The freeze mechanism from TPTL has also received attention in connection with data words. A data word is a finite or infinite sequence $(P_0, d_0)(P_1, d_1) \ldots$ of the above form, where we do not require the data values d_i to be monotonic, and we speak of *non-monotonic data words*. As for TPTL, freezeLTL can store the current data value in a register x. But in contrast to TPTL, the value of x can only be compared for equality with the current data value.

Satisfiability and model checking for MTL, TPTL and freezeLTL have been studied intensively in the past [2–4, 7, 8, 15–17]. For model checking freezeLTL the authors of [8] consider one-counter machines (OCM) as a mechanism for generating infinite non-monotonic data words, where the data values are the counter values along the unique computation path. Whereas freezeLTL model checking for non-deterministic OCM is Σ_1^1-complete, the problem becomes PSPACE-complete for deterministic OCM [8].

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

In this paper, we study MTL and TPTL over non-monotonic data words. The latter logic extends both freezeLTL over non-monotonic data words and TPTL over monotonic data words: As for freezeLTL, data values are natural numbers that can vary arbitrarily over time. In contrast to the latter, one can express that the difference of the current data value and the value stored in a register belongs to a certain interval, whereas freezeLTL only allows to say that this difference is zero. Applications for TPTL over non-monotonic data values can be seen in areas, where data streams of discrete values have to be analyzed and the focus is on the dynamic variation of the values (e.g. streams of discrete sensor data or stock charts). Recently, it has been shown that (in contrast to the monotonic setting [1]) in the non-monotonic setting, TPTL is strictly more powerful than MTL [5].

We investigate the complexity of model checking problems for TPTL over non-monotonic data words. These data words 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. For periodic words without data values (i.e., ω-words of the form uv^ω), the complexity of LTL model checking (also known as LTL path checking) belongs to $\mathsf{AC}^1(\mathsf{LogDCFL})$ (a subclass of NC) [14]. This result solved a long standing open problem. For finite monotonic data words, the same complexity bound has been shown for MTL in [4].

We show that the latter result of [4] is quite sharp in the following sense: Path checking for MTL over non-monotonic (finite or infinite) data words as well as path checking for TPTL with one register variable 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 [8]). These results yield a rather complete picture on the complexity of path checking for MTL and TPTL, see Fig. 5.

2 Temporal Logics over Data Words

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

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

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

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

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

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

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

We say that a data word *satisfies* an MTL-formula φ, written $w \models \varphi$, if $(w, 0) \models \varphi$. We use the following standard abbreviations: $\varphi_1 \vee \varphi_2 := \neg(\neg\varphi_1 \wedge \neg\varphi_2)$, $\varphi_1 \rightarrow \varphi_2 := \neg\varphi_1 \vee \varphi_2$, $\mathsf{true} := p \vee \neg p$, $\mathsf{false} := \neg\mathsf{true}$, $\mathsf{F}_I\varphi := \mathsf{true}\mathsf{U}_I\varphi$, $\mathsf{G}_I\varphi := \neg\mathsf{F}_I\neg\varphi$.

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

$$\varphi ::= p \mid x \sim c \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathsf{X}\varphi \mid \varphi \mathsf{U} \varphi \mid x.\varphi \tag{1}$$

We use the same syntactical abbreviations as for MTL. The fragment freezeLTL is obtained by restricting $\sim$ in (1) to $=$. Ordinary LTL is obtained by disallowing the use of register variables. Given $r \geq 1$, we use TPTL^r (resp., $\mathsf{freezeLTL}^r$) to denote the fragment of TPTL (resp., freezeLTL) that uses at most r different register variables.

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

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

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

Note that $x \sim c$ does not mean that the current value $v = \nu(x)$ of x satisfies $v \sim c$, but expresses that $d_i - v \sim c$, where d_i is the current data value. We say that a data word w satisfies a TPTL-formula φ, written $w \models \varphi$, if $(w, 0, \bar{0}) \models \varphi$, where $\bar{0}$ denotes the valuation that maps all variables to the initial data value d_0.

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

3 Path Checking Problems for TPTL and MTL

In this section, we study the path checking problems for our logics over data words. Data words can be (i) finite or infinite, (ii) monotonic or non-monotonic, (iii) pure or non-pure, and (iv) unary encoded or binary encoded. For one of our logics L and a class of data words C, we consider the *path checking problem for* L *over* C. It asks whether for a given data word $w \in \mathsf{C}$ and a given formula $\varphi \in \mathsf{L}$, $w \models \varphi$ holds.

3.1 Upper Bounds

In this section we prove our upper complexity bounds. All bounds hold for non-monotonic and non-pure data words (and we will not mention this explicitly in the theorems). But we have to distinguish whether (i) data words are unary or binary encoded, and (ii) whether data words are finite or infinite. For the most general path checking problem (TPTL_b over infinite binary encoded data words) we can devise an alternating polynomial time algorithm (and hence a polynomial space algorithm). The only 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, see [10] for details.

Theorem 1. *Path checking for* TPTL_b *over infinite binary encoded data words is in* PSPACE.

If the number of register variables is fixed and all data values are unary encoded, then the alternating Turing-machine in the proof of Theorem 1 works in logarithmic space. Since $\mathsf{ALOGSPACE} = \mathsf{P}$, we obtain the following statement for (i). For (ii) we show that an infinite *binary* encoded monotonic data word can be replaced by an infinite *unary* encoded data word, which allows to apply (i).

Theorem 2. *For every fixed* $r \in \mathbb{N}$*, path checking for* TPTL_u^r *over (i) infinite unary encoded data words or (ii) infinite binary encoded monotonic data words is in* P.

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

Theorem 3. *For every fixed* $r \in \mathbb{N}$*, path checking for* TPTL_b^r *over finite binary encoded data words is in* P.

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

Theorem 4. *Path checking for* TPTL_b^1 *over infinite binary encoded data words is in* P.

For the proof of Theorem 4 we need the following two lemmas.

Lemma 5. *For a given* LTL*-formula ψ, words $u_1, \ldots, u_k, u \in (2^{\mathcal{P}})^*$ and binary encoded numbers $N_1, \ldots, N_k \in \mathbb{N}$, the question whether $u_1^{N_1} u_2^{N_2} \cdots u_k^{N_k} u^\omega \models \psi$ holds, belongs to* P *(actually,* $\mathsf{AC}^1(\mathsf{LogDCFL})$*).*

The crucial point is that for all finite words $u, v \in (2^{\mathcal{P}})^*$, every infinite word $w \in (2^{\mathcal{P}})^\omega$ and every number $N \geq |\psi|$, we have $uv^N w \models \psi$ if and only if $uv^{|\psi|}w \models \psi$. This can be shown by using the Ehrenfeucht-Fraïssé game for LTL from [9]. Hence, one can replace all exponents N_i by small numbers of size at most $|\psi|$. Then, one can use a polynomial time algorithm (or $\mathsf{AC}^1(\mathsf{LogDCFL})$ algorithm) for LTL path checking [14].

Lemma 6. *Path checking for* TPTL_b*-formulae, which do not contain subformulae of the form $x.\theta$ for a register variable x, over infinite binary encoded data words is in* P *(in fact,* $\mathsf{AC}^1(\mathsf{LogDCFL})$*).*

Proof. We reduce the question, whether $w \models \psi$ in logspace to an instance of the succinct LTL path checking problem from Lemma 5. Let $w = u_1(u_2)^\omega_{+k}$ and let $w[i] = (P_i, d_i) \in 2^{\mathcal{P}} \times \mathbb{N}$. Let $n_1 = |u_1|$ and $n_2 = |u_2|$. We can assume that only one register variable x appears in ψ (since we do not use the freeze construct $x.()$ in ψ all register variables remain at the initial value d_0).

In order to construct an LTL-formula from ψ, it remains to eliminate occurrences of constraints $x \sim c$ in ψ. W.l.o.g. all constraints are of the form $x < c$ or $x > c$. Let $x \sim_1 c_1, \ldots, x \sim_m c_m$ be a list of all constraints that appear in ψ. We introduce for every $1 \leq j \leq m$ a new atomic proposition p_j and let $\mathcal{P}' = \mathcal{P} \cup \{p_1, \ldots, p_m\}$. Let ψ' be obtained from ψ by replacing every occurrence of $x \sim_j c_j$ by p_j, and let $w' \in (2^{\mathcal{P}'})^\omega$ be the ω-word with $w'[i] = P_i \cup \{p_j \mid 1 \leq j \leq m, d_i - d_0 \sim_j c_j\}$. Clearly $w \models \psi$ if and only if $w' \models \psi'$. We will show that the word w' can be written in the form considered in Lemma 5.

First of all, we can write w' as $w' = u'_1 u'_{2,0} u'_{2,1} u'_{2,2} \cdots$, 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 length of u_2 times number of constraints). To see this, consider a position $0 \leq i \leq n_2 - 1$ in the word u_2 and a constraint $x \sim_j c_j$ ($1 \leq j \leq m$). 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 $I_1, \ldots, I_l$ 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 arithmetic on binary encoded numbers (addition, multiplication and division with remainder can be carried out in logspace on binary encoded numbers [12]). Hence, we

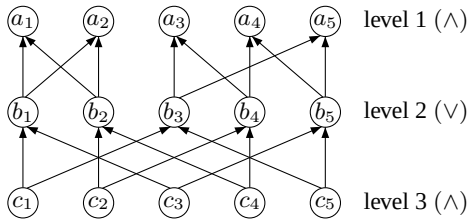

Fig. 1. An SAM2-circuit

can compute in logspace the lengths $N_i = |I_i|$ of the intervals, where $N_l = \omega$. Also, for all $1 \leq i \leq l$ we can compute in logspace the unique word v_i such that $v_i = u'_{2,a}$ for all $a \in I_i$. Hence, $w' = u'_1 v_1^{N_1} \cdots v_l^{N_l}$. We can now apply Lemma 5. □

Proof of Theorem 4. Consider an infinite binary encoded data word $w = u_1(u_2)^{\omega}_{+k}$ and a TPTL^1_b-formula ψ. Let $n = |u_1|+|u_2|$. We check in polynomial time whether $w \models \psi$. A TPTL-formula φ is closed if every occurrence of a register variable x in φ appears within a subformula of the form $x.\theta$. The following two claims are straightforward:

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

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

By Claim 1 we can write $(w, i) \models \varphi$ for $(w, i, \nu) \models \varphi$. 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 $(w, i) \models x.\varphi$, or equivalently $w[i:] \models \varphi$. 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 $w[i:] \models \varphi$. Let $x.\varphi_1, \ldots, x.\varphi_l$ be all maximal (with respect to the subformula relation) subformulae of φ of the form $x.\theta$. We can assume that for every $1 \leq s \leq l$ we have already determined the set of positions $j \in [0, n-1]$ such that $(w, j) \models x.\varphi_s$. 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 $(w, j) \models x.\varphi_s$, 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, it is easy 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 $v_2 = u'_2[i:](u'_2[:i-1] + k)$. Finally, using Lemma 6 we can check in polynomial time whether $w'[i:] \models \varphi'$. □

3.2 Lower Bounds

We prove several P-hardness and PSPACE-hardness results for path checking.

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

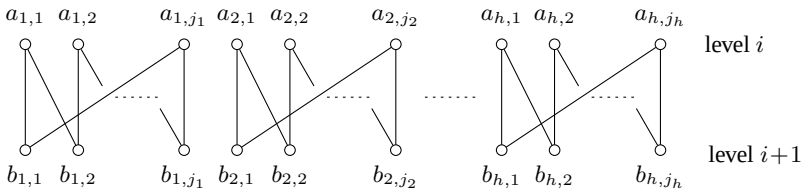

Fig. 2. The induced subgraph between level i and $i+1$

- All wires go from a gate in level $i+1$ to a gate from level i ($1 \leq i \leq l-1$).
- All output gates are in level 1 and all input gates are in level l, and the latter are labelled with input bits. Moreover, there is a distinguished output gate on level 1.
- All gates in the same level $1 \leq i \leq l-1$ are of the same type ($\wedge$ or $\vee$) and the levels alternate between $\wedge$-levels and $\vee$-levels.
- All gates except the output gates have fanout 2 and all gates except the input gates have fanin 2. The two input gates for a gate at level $i \leq l-1$ are different.

By the restriction to fanin 2 and fanout 2, we know that each level contains the same number of gates. Fig. 1 shows an example of an SAM2-circuit (the node names a_i, b_i, c_i will be needed later). The circuit value problem for SAM2-circuits (i.e., the question whether the distinguished output gate of a given SAM2-circuit evaluates to 1), which is called SAM2CVP, is P-complete [11].

Recall that finite path checking for MTL (a fragment of TPTL^1) over monotonic data words is in the parallel complexity class $\mathsf{AC}^1(\mathsf{LogDCFL})$ [4]. We will show that for both (i) MTL_u over non-monotonic data words and (ii) TPTL^1_u over monotonic data words the path checking problem becomes P-hard (and hence P-complete).

Theorem 7. *Path checking for* MTL_u *over finite unary encoded pure data words is* P*-hard.*

Proof. We reduce from SAM2CVP. Let α be the input circuit. We first encode each two 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 formula ψ such that $w \models \psi$ if and only if α evaluates to 1. The data word w that we are constructing contains gate names of α (and some copies of the gates) as atomic propositions. These propositions will be only needed for the construction. At the end, we can remove all propositions from the data word w and hence obtain a pure data word. The whole construction can be done in logspace. The reader might look at the example in [10], where the construction is carried out for the circuit from Fig. 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 which contains the nodes in level i and $i+1$ ($1 \leq i < l$) consists of several cycles; see Fig. 2. For instance, for the circuit in Fig. 1 the number of cycles between level 1 and 2 (resp., 2 and 3) is 2.

We can enumerate in logspace the gates of level i and $i+1$ such that they occur in the order shown in Fig. 2. For this, let $a_1, \ldots, a_n$ (resp., $b_1, \ldots, b_n$) be the nodes in level i (resp., $i+1$) in the order in which they occur in the input description. We start with

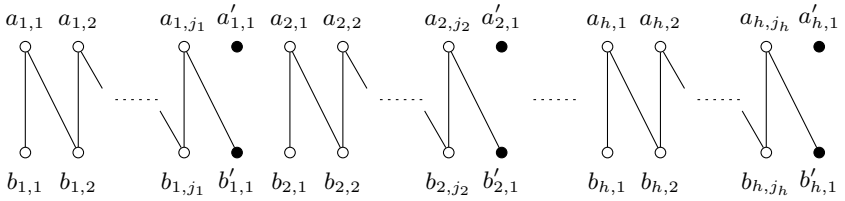

Fig. 3. The graph obtained from the induced subgraph

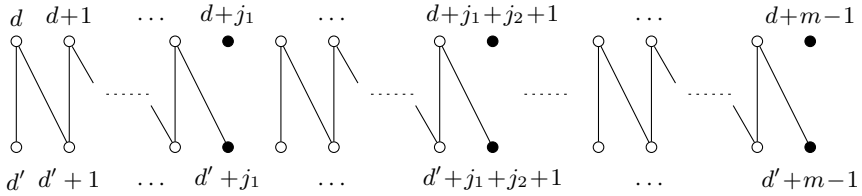

Fig. 4. Labeling the new graph

a_1 and enumerate the nodes in the cycle containing a_1 (from a_1 we go to the smaller neighbor among $b_1, \dots, b_n$, then 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 a_1. As soon as we return to a_1, we know that the first cycle is completed. To find the next cycle, we search for the first node from $a_2, \dots, a_n$ that is not reachable from a_1 (reachability in undirected graphs is in logspace), and continue this way.

So, assume that the nodes in layer i and $i+1$ are ordered as in Fig. 2. In particular, we have h cycles. For each $1 \leq t \leq h$, we add a new node $a'_{t,1}$ (resp., $b'_{t,1}$) after a_{t,j_t} (resp., b_{t,j_t}). Then we replace the edge $(a_{t,j_t}, b_{t,1})$ by the edge $(a_{t,j_t}, b'_{t,1})$ $(1 \leq t \leq h)$. In this way we obtain the graph from Fig. 3. Again, the construction can be done in logspace by adding the new nodes and new edges once a cycle was completed in the enumeration procedure from the previous paragraph.

By adding dummy nodes, we can assume that for every $1 \leq i \leq l-1$, the subgraph between level i and $i+1$ has the same number (say h) of cycles. We still denote by n the number of nodes in each level. Thus, after the above step we have $m = n+h$ nodes in each level. Let $d = (i-1) \cdot 2m$ and $d' = d+m$. In Fig. 3, we label the nodes in level i (resp., $i+1$) with the numbers $d, d+1, \dots, d+m-1$ (resp. $d', d'+1 \dots, d'+m-1$) in this order, see Fig. 4. By this labeling, the difference between two connected nodes in level i and level $i+1$ is always m or $m+1$. So we can use the modality $\mathsf{F}_{[m,m+1]}$ (resp., $\mathsf{G}_{[m,m+1]}$) to jump from an $\vee$-gate (resp., $\wedge$-gate) in level i to a successor gate in level $i+1$. We now obtain in logspace the data word $w_i = w_{i,1}w_{i,2}$, where

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

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

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

Suppose now that all data words w_i ($1 \leq i \leq l-1$) are constructed. We then combine them to obtain the data word w for the whole circuit as follows. Suppose that

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

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

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

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

Let us explain the idea. Consider a gate a_j of level $2 \leq i \leq l-1$, 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 $\mathsf{F}_{[m,m+1]}$. By the 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 modality X^m (which is an abbreviation for m copies of the MTL-modality $X_{\mathbb{Z}}$), we jump to the corresponding positions in $v_{i+1,1}$.

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

$$w_{l-1,2} = (\tilde{d}_1, v_1) \ldots (\tilde{d}_m, v_m).$$

Let $d_1, \ldots, d_n$ be the original gates of level l, which all belong to $\{\tilde{d}_1, \ldots, \tilde{d}_m\}$, and let $x_i \in \{0,1\}$ be the input value for gate d_i. Define

$$I = \{j \mid j \in [1,m], \exists i \in [1,n] : \tilde{d}_j \in \{d_i, d'_i\}, x_i = 1\}. \tag{2}$$

Let the designated output gate be the k^{th} node in level 1. We construct the MTL-formula $\psi = \mathsf{X}^{k-1} \varphi_1$, where φ_i ($1 \leq i \leq l-1$) is defined inductively as follows:

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

Note that the formula $\neg \mathsf{X}\,\mathsf{true}$ is only true in the last position of a data word. Suppose data word w is the encoding of the circuit. From the above consideration, it follows that $w \models \psi$ if and only if the circuit α evaluates to 1. Note that we do not use any propositional variables in the formula ψ. So we can ignore the propositional part in the data word w to get a pure data word. □

Note that the above construction uses non-monotonic data words. This is unavoidable since finite path checking for MTL over monotonic data words is in NC [4]. On the other hand, for the extension TPTL^1_u of MTL_u we can show, using again a reduction from SAM2CVP (see [10]), P-hardness also for monotonic data words:

Theorem 8. *Path checking for* TPTL^1_u *over finite unary encoded strictly monotonic pure data words is* P*-hard.*

PSPACE**-Hardness.** In [10], we prove three PSPACE lower bounds, which complete our complexity picture. The first one is shown by a reduction from QBF, whereas the latter two results are shown by a reduction from a quantified variant of the subset sum problem [19].

Theorem 9. *Path checking for* TPTL_u *over finite unary encoded strictly monotonic pure data words is* PSPACE*-hard.*

Theorem 10. *Path checking for* TPTL^2_b *over the infinite strictly monotonic pure data word* $w = 0(1)^{\omega}_{+1} = 0, 1, 2, 3, 4, \ldots$ *is* PSPACE*-hard.*

Theorem 11. *Path checking for* $\mathsf{freezeLTL}^2$ *(and hence* TPTL^2_u*) over infinite binary encoded pure data words is* PSPACE*-hard.*

Recall from Theorem 2 that for every fixed r, path checking for TPTL^r_u over infinite binary encoded monotonic data words can be solved in polynomial time. Hence, Theorem 11 shows that monotonicity is important for Theorem 2.

3.3 Summary of the Results

Figure 5 collects our complexity results for path checking problems (here the superscript $<\infty$ is a place holder for any number $r \geq 2$). Whether data words are pure or not does not influence the complexity in all cases. Moreover, for finite data words, the complexity does not depend upon the encoding of data words (unary or binary) and the fact whether data words are monotonic or non-monotonic. On the other hand, for infinite data words, these distinctions influence the complexity: For binary and non-monotonic data words we get another picture than or unary encoded or (quasi-)monotonic data words. Note that for MTL_b and MTL_u the complexity is P-complete for all classes of data words (since MTL translates in logspace into TPTL^1).

One may also study the complexity of path checking problems for various fragments of MTL and TPTL. In this context, it is interesting to note that all lower bounds already hold for the corresponding unary fragments (where the until-operator is replaced by F and G) with only one exception: Our proof for Theorem 11 in [10] for $\mathsf{freezeLTL}^2$

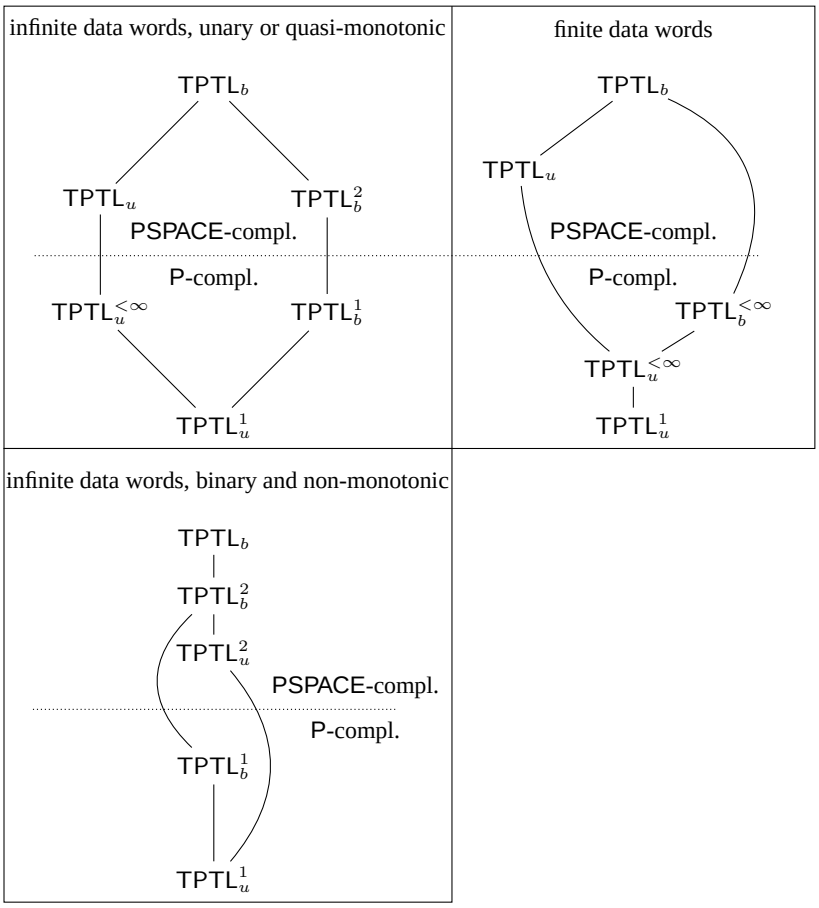

Fig. 5. Complexity results for path checking

needs the until operator. It is not clear, whether path checking for the unary fragment of freezeLTL^2 over infinite binary encoded data words is still PSPACE-complete.

Our complexity results for infinite unary encoded data words also hold for *deterministic one-counter machines (DOCMs)*, see [10] for a precise definition. A DOCM produces in general an infinite data word, where the sequence of atomic propositions is the sequence of states of the machine, and the sequence of data values is the sequence of counter values produced by the DOCM (the DOCM can block in which case it produces a finite data word). It is an easy observation that the data word produced by a DOCM $\mathcal{A}$ is periodic in case it is infinite, and one can in fact compute in logspace from $\mathcal{A}$ two unary encoded finite data words u_1 and u_2 and a unary encoded number k such that $u_1(u_2)^\omega_{+k}$ is the data word produced by $\mathcal{A}$, see also [8, Lemma 9]. For this it is crucial that the counter can be incremented or decremented in each step by at most one (or, more general, a unary encoded number). This, in turn implies that for

each of the logics L considered in this paper, the model checking problem for L over DOCM (i.e., the question, whether a given formula $\varphi \in \mathsf{L}$ holds in the data word produced by a given DOCM) is equivalent with respect to logspace reductions to the path checking problem for L over infinite unary encoded data words. Hence, the upper left diagram from Figure 5 also shows the complexity results for TPTL model checking over DOCM. In particular we strengthen the third author's recent decidability result for model checking non-monotonic TPTL over DOCMs [18]. Our results also generalizes the PSPACE-completeness result for freezeLTL over DOCMs from [8].

References

1. R. Alur and T. A. Henzinger. Real-Time Logics: Complexity and Expressiveness. *Inf. Comput.*, 104(1):35–77, 1993.
2. R. Alur and T. A. Henzinger. A really temporal logic. *J. ACM*, 41(1):181–204, 1994.
3. P. Bouyer, K. G. Larsen, and N. Markey. Model checking one-clock priced timed automata. *Log. Meth. Comput. Sci.*, 4(2), 2008.
4. D. Bundala and J. Ouaknine. On the complexity of temporal-logic path checking. In *Proc. ICALP 2014, Part II*, LNCS 8573, pages 86–97. Springer, 2014.
5. C. Carapelle, S. Feng, O. F. Gil, and K. Quaas. On the expressiveness of TPTL and MTL over ω-data words. In *Proc. AFL 2014*, volume 151 of *EPTCS*, pages 174–187, 2014.
6. C. Carapelle, S. Feng, O. F. Gil, and K. Quaas. Satisfiability for MTL and TPTL over non-monotonic data words. In *Proc. LATA 2014*, LNCS 8370, pages 248–259. Springer, 2014.
7. S. Demri and R. Lazić. LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Log.*, 10(3), 2009.
8. S. Demri, R. Lazić, and A. Sangnier. Model checking memoryful linear-time logics over one-counter automata. *Theor. Comput. Sci.*, 411(22-24):2298–2316, 2010.
9. K. Etessami and T. Wilke. An until hierarchy and other applications of an Ehrenfeucht-Fraïssé game for temporal logic. *Inf. Comput.*, 160(1-2):88–108, 2000.
10. S. Feng, M. Lohrey, and K. Quaas. Path-Checking for MTL and TPTL. http://arxiv.org/abs/1412.3644. arXiv.org, 2014.
11. R. Greenlaw, H. J. Hoover, and W. L. Ruzzo. Limits to Parallel Computation: P-completeness Theory. *Oxford University Press*, 1995.
12. W. Hesse, E. Allender, and D. A. M. Barrington. Uniform constant-depth threshold circuits for division and iterated multiplication. *J. Comput. System Sci.*, 65:695–716, 2002.
13. R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.
14. L. Kuhtz and B. Finkbeiner. Efficient parallel path checking for linear-time temporal logic with past and bounds. *Log. Meth. Comput. Sci.*, 8(4), 2012.
15. F. Laroussinie, N. Markey, and Ph. Schnoebelen. On model checking durational Kripke structures. In *Proc. FoSSaCS 2002*, LNCS 2303, pages 264–279. Springer, 2002.
16. J. Ouaknine and J. Worrell. On metric temporal logic and faulty Turing machines. In *Proc. FoSSaCS 2006*, LNCS 3921, pages 217–230. Springer, 2006.
17. J. Ouaknine and J. Worrell. On the decidability and complexity of metric temporal logic over finite words. *Log. Meth. Comput. Sci.*, 3(1), 2007.
18. K. Quaas. Model checking metric temporal logic over automata with one counter. In *Proc. LATA 2013*, LNCS 7810, pages 468–479. Springer, 2013.
19. S. Travers. The complexity of membership problems for circuits over sets of integers. *Theor. Comput. Sci.*, 369(1):211–229, December 2006.