SMT-Based Model Checking of Max-Plus Linear Systems

Muhammad Syifa'ul Mufid
Department Computer Science, University of Oxford, UK

Andrea Micheli
Fondazione Bruno Kessler, Trento, Italy

Alessandro Abate
Department Computer Science, University of Oxford, UK

Alessandro Cimatti
Fondazione Bruno Kessler, Trento, Italy

Abstract

Max-Plus Linear (MPL) systems are an algebraic formalism with practical applications in transportation networks, manufacturing and biological systems. MPL systems can be naturally modeled as infinite-state transition systems, and exhibit interesting structural properties (e.g. periodicity or steady state), for which analysis methods have been recently proposed. In this paper, we tackle the open problem of specifying and analyzing user-defined temporal properties for MPL systems. We propose Time-Difference LTL (TDLTL), a logic that encompasses the delays between the discrete-time events governed by an MPL system, and characterize the problem of model checking TDLTL over MPL. We propose a family of specialized algorithms leveraging the periodic behaviour of an MPL system. We prove soundness and completeness, showing that the transient and cyclicity of the MPL system induce a completeness threshold for the verification problem. The algorithms are cast in the setting of SMT-based verification of infinite-state transition systems over the reals, with variants depending on the (incremental vs upfront) computation of the bound, and on the (explicit vs implicit) unrolling of the transition relation. Our comprehensive experiments show that the proposed techniques can be applied to MPL systems of large dimensions and on general TDLTL formulae, with remarkable performance gains against a dedicated abstraction-based technique and a translation to the nuXmv symbolic model checker.

2012 ACM Subject Classification Theory of computation → Verification by model checking

Keywords and phrases Max-Plus Linear Systems, Satisfiability Modulo Theory, Model Checking, Linear Temporal Logic

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2021.22

1 Introduction

Max-Plus Linear (MPL) systems are a class of discrete-event systems (DES) that are based on the so-called max-plus algebra, an algebraic system using the two binary operations of maximisation and addition. MPL systems are employed to model applications with features of synchronization without concurrency, and as such are widely used for applications in transportation networks [7], manufacturing [29] and biological systems [14, 20]. In MPL models, the states correspond to time instances related to discrete events. Traditional dynamical analysis of MPL systems is associated with their algebraic and graph representation (cf. Definition 2), that allows the investigation of several structural problems such as eigenproblems [21], optimisation [13] and periodicity [24, 35].

In this paper, we tackle the problem of formally specifying and analyzing user-defined temporal properties for MPL systems. This can be considered an open problem in practice, despite the line of work in [3, 4, 5] that deals with reachability analysis and formal verification

for MPL systems. These methods are in general not complete: they rely on the construction of an abstraction that overapproximates the concrete MPL system [2, 33, 34]. Furthermore, the underlying abstraction procedures suffer from state-explosion problems, given that the size of the abstraction is exponential in the size of the MPL, and are unable to deal with more than few variables. Finally, no general specification language is provided to express properties at the MPL system level.

We make the following contributions. First, we propose Time-Difference LTL (TDLTL), a logic that encompasses the delays between the discrete-time events governed by an MPL system, and characterize the problem of model checking TDLTL over MPL. Second, we propose a family of specialized algorithms for TDLTL model checking, cast in the setting of infinite-state transition systems, with a symbolic representation in Satisfiability Modulo Theories (SMT) over the reals [9]. The algorithms, that we prove sound and complete, leverage the periodic behaviour of an MPL system: intuitively, the transient and cyclicity of the MPL system induce a completeness threshold for a bounded encoding of the verification problem. The family of algorithms has several variants, depending on two independent factors. One is the *computation of the bound*, that could be carried out either upfront before calling the SMT solver, or incrementally, interleaving it with multiple solver calls. The other is the *unrolling of the transition relation*, that can either follow the explicit approach of Bounded Model Checking (BMC), or – thanks to some algebraic properties of MPL systems – be left implicit, so that the number of SMT variables is significantly reduced. Third, we demonstrate the practical effectiveness of the approach. We run a comprehensive set of experiments, showing that the proposed techniques can be applied to general TDLTL formulae on large MPL systems that are completely out-of-reach for existing abstraction-based techniques [33, 34]. The comparison also shows that the new algorithms yield orders-of-magnitude speed ups against a generic translational approach to the nuXmv symbolic model checker [15].

The structure is as follows. Section 2 describes the basics of MPL systems and SMT. In Section 3, we formalize the TDLTL logic and Sections 4 and 5 present the verification algorithms and the related work, respectively. Our experimental analysis is reported in Section 6 and we conclude in Section 7. Proofs and additional experiments are provided in the appendices.

2 Model and Preliminaries

2.1 Max-Plus Linear Systems

Max-plus algebra is a modification of the canonical linear algebra, and is defined over the max-plus semiring $(\mathbb{R}_{\max}, \oplus, \otimes)$, where $\mathbb{R}_{\max} := \mathbb{R} \cup \{\varepsilon := -\infty\}$ and

$$a \oplus b := \max\{a, b\}, \qquad a \otimes b := a + b, \quad \forall a, b \in \mathbb{R}_{\max} \tag{1}$$

The zero and unit elements of $\mathbb{R}_{\max}$ are ε and 0, respectively. By $\mathbb{R}_{\max}^{m\times n}$, we denote the set of $m \times n$ matrices over the max-plus algebra. Max-plus algebraic operations can be extended to vectors and matrices as follows. Given $A, B \in \mathbb{R}_{\max}^{m\times n}, C \in \mathbb{R}_{\max}^{m\times p}, D \in \mathbb{R}_{\max}^{p\times n}$ and $\alpha \in \mathbb{R}_{\max}$,

$$\begin{aligned}
[\alpha \otimes A](i,j) &= \alpha + A(i,j),\\
[A \oplus B](i,j) &= A(i,j) \oplus B(i,j),\\
[C \otimes D](i,j) &= \bigoplus_{k=1}^{p} C(i,k) \otimes D(k,j),
\end{aligned}$$

for all $i = 1, \ldots, m$ and $j = 1, \ldots, n$. Given $A \in \mathbb{R}_{\max}^{n\times n}$ and $t \in \mathbb{N}$, $A^{\otimes t}$ denotes $A \otimes \ldots \otimes A$ (t times).

A dynamical system over the max-plus algebra is called a Max-Plus Linear (MPL) system and is defined as

$$\mathbf{x}(k+1) = A \otimes \mathbf{x}(k), \quad k = 0, 1, \ldots, \tag{2}$$

where $A \in \mathbb{R}_{\max}^{n \times n}$ is the system matrix, and vector $\mathbf{x}(k) = [x_1(k) \ \ldots \ x_n(k)]^\top$ encodes the state variables [7]. Vector $\mathbf{x}$ is used to represent the time stamps associated to the discrete events, while k corresponds to the event counter. Applications of MPL systems are found in dynamical systems where modelling the time variable is essential, such as in transportation networks [29], in scheduling [6] or manufacturing [30] problems, or for biological systems [14, 20].

▶ **Definition 1** (Orbit and Lasso). Given an MPL system (2) with an initial vector $\mathbf{x}(0)$, a sequence $\mathbf{x}(0)\mathbf{x}(1)\ldots$ is called an orbit from $\mathbf{x}(0)$ w.r.t. A. Furthermore, if there exist $\alpha \in \mathbb{R}$ and $k \geq l \geq 0$ such that $\mathbf{x}(k+1+j) = \alpha \otimes \mathbf{x}(l+j)$ for $j \geq 0$ then such sequence is called a (k, l)-lasso. Furthermore, we call l the *loopback* bound. The illustration of a lasso is shown in Figure 1.

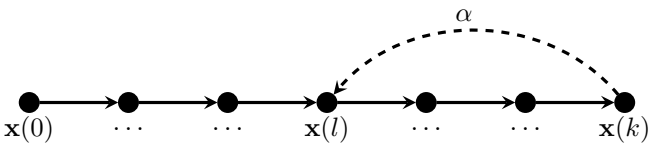


Figure 1 An illustration of a (k, l)-lasso. The dashed arrow represents the transition $\mathbf{x}(k+1) = \alpha \otimes \mathbf{x}(l)$.

Let us remark that an orbit represents the execution (or path) of (2) originating from an initial state that is a vector. It is important to note that the definition of lasso is slightly different from the *canonical* one found in literature [11, 12], which requires that the l-th and $(k+1)$-th states are the same. The notation $\mathsf{Orb}(A) = \{\mathbf{x}(0)\mathbf{x}(1)\ldots \mid \mathbf{x}(0) \in \mathbb{R}^n\}$ represents the set of all orbits w.r.t. A. Likewise, $\mathsf{Orb}(A, X) = \{\mathbf{x}(0)\mathbf{x}(1)\ldots \mid \mathbf{x}(0) \in X\}$ is the set of orbits w.r.t. A starting from a set of initial vectors X. For the sake of simplicity we may use the following notation to refer to an orbit: $\pi = \mathbf{x}(0)\mathbf{x}(1)\ldots$. Given an orbit π and $j \geq 0$, $\pi[j] = \mathbf{x}(j)$ denotes the j-th vector of π while $\pi[j..]$ is the j-th suffix of π, i.e., $\pi[j..] = \mathbf{x}(j)\mathbf{x}(j+1)\ldots$. We say that orbit π is *similar* to orbit σ iff there exists $\beta \in \mathbb{R}$ such that $\pi[0] = \beta \otimes \sigma[0]$ (which implies $\pi[j] = \beta \otimes \sigma[j]$ for $j \geq 0$).

▶ **Definition 2** (Precedence Graph [7]). The precedence graph of $A \in \mathbb{R}_{\max}^{n \times n}$, denoted by $\mathcal{G}(A)$, is a weighted directed graph with nodes $1, \ldots, n$ and an edge from j to i with weight $A(i, j)$ for each $A(i, j) \neq \varepsilon$.

▶ **Definition 3** (Irreducible Matrix [7]). A matrix $A \in \mathbb{R}_{\max}^{n \times n}$ is called *irreducible* if $\mathcal{G}(A)$ is strongly connected.

A directed graph is strongly connected if, for any two different nodes i, j, there exists a path from i to j. The weight of a path $p = i_1 i_2 \ldots i_k$ is equal to the sum of the edge weights in p. A circuit, namely a path that begins and ends at the same node, is called *critical* if it has maximum average weight, which is the weight divided by the length of the path [7].

Each irreducible matrix $A \in \mathbb{R}_{\max}^{n \times n}$ admits a unique max-plus eigenvalue $\lambda \in \mathbb{R}$ and a corresponding max-plus eigenspace $E(A) = \{x \in \mathbb{R}_{\max}^n \mid A \otimes x = \lambda \otimes x\}$. The scalar λ is equal to the average weight of critical circuits in $\mathcal{G}(A)$, and $E(A)$ can be computed from

$A_\lambda^+ = \bigoplus_{k=1}^{n}((-\lambda)\otimes A)^{\otimes k}$. A reducible matrix may have multiple eigenvalues, where the maximum one equals to the average weight of critical circuits of $\mathcal{G}(A)$. Another important property of irreducible MPL systems is the periodicity of the powers of matrix $A^{\otimes k}$.

▶ **Proposition 4** (Transient [7, 29])**.** For an irreducible matrix $A \in \mathbb{R}_{\max}^{n\times n}$ and its max-plus eigenvalue $\lambda \in \mathbb{R}$, there exist $l, c \in \mathbb{N}_0$, such that $A^{\otimes(k+c)} = (\lambda \times c) \otimes A^{\otimes k}$ for all $k \geq l$. The smallest such l and c are called the *transient* and the *cyclicity* of A, respectively.

For the rest of this paper, we denote the transient and the cyclicity of A as $\mathsf{tr}(A)$ and $\mathsf{cyc}(A)$, respectively. While $\mathsf{cyc}(A)$ is related to critical circuits in the precedence graph $\mathcal{G}(A)$ [7, Def 3.94], $\mathsf{tr}(A)$ is unrelated to the dimension of A. Even for a small n, the transient of $A \in \mathbb{R}_{\max}^{n\times n}$ can be large. Upper bounds of the transient have been discussed in [16, 32, 35, 36].

By Proposition 4, all orbits of an irreducible MPL system induce a *periodic* behaviour with a rate λ: for each initial vector $\mathbf{x}(0) \in \mathbb{R}^n$ we have $\mathbf{x}(k + \mathsf{cyc}(A)) = (\lambda \times \mathsf{cyc}(A)) \otimes \mathbf{x}(k)$ for all $k \geq \mathsf{tr}(A)$. A similar condition may be found on reducible MPL systems: we denote the corresponding transient and cyclicity to be global, as per Proposition 4. The local transient and cyclicity for a specific initial vector $\mathbf{x}(0) \in \mathbb{R}^n$ and for a set of initial vectors $X \subseteq \mathbb{R}^n$ has been studied in [1] and is defined as follows.

▶ **Definition 5** ([34])**.** Given $A \in \mathbb{R}_{\max}^{n\times n}$ with max-plus eigenvalue λ and an initial vector $\mathbf{x}(0) \in \mathbb{R}^n$, the local transient and cyclicity of A w.r.t. $\mathbf{x}(0)$ are respectively the smallest $l, c \in \mathbb{N}_0$ such that $\mathbf{x}(j + c) = \lambda c \otimes \mathbf{x}(j), \forall j \geq l$. We denote those scalars as $\mathsf{tr}(A, \mathbf{x}(0))$ and $\mathsf{cyc}(A, \mathbf{x}(0))$, respectively. Furthermore, for $X \subseteq \mathbb{R}^n$, $\mathsf{tr}(A, X) = \max\{\mathsf{tr}(A, \mathbf{x}(0)) \mid \mathbf{x}(0) \in X\}$ and $\mathsf{cyc}(A, X) = \mathtt{lcm}\{\mathsf{cyc}(A, \mathbf{x}(0)) \mid \mathbf{x}(0) \in X\}$, where $\mathtt{lcm}$ is the "least common multiple".

Following [1], any MPL system (2) can be classified into three categories: 1) *never periodic* if $\mathsf{tr}(A, \mathbf{x}(0))$ does not exist for all $\mathbf{x}(0) \in \mathbb{R}^n$; 2) *boundedly periodic* if $\mathsf{tr}(A, \mathbf{x}(0))$ exists for all $\mathbf{x}(0) \in \mathbb{R}^n$ and $\mathsf{tr}(A)$ exists; and 3) *unboundedly periodic* if $\mathsf{tr}(A, \mathbf{x}(0))$ exists for all $\mathbf{x}(0) \in \mathbb{R}^n$ but $\mathsf{tr}(A)$ does not. We call (2) *periodic* if it is either *unboundedly periodic* or *boundedly periodic*. It has been shown in [1] that an MPL system (2) is periodic if and only if the state matrix A admits a finite eigenvector (all elements are not equal to ε). The following proposition shows the relation between the periodicity of the MPL system (2) and its corresponding orbits.

▶ **Proposition 6.** An orbit π is a lasso iff $\pi[0]$ admits local transient and cyclicity.

▶ **Corollary 7.** An MPL system (2) is periodic iff all orbits $\pi \in \mathsf{Orb}(A)$ are lassos.

▶ Remark 8. The nature of the periodicity of an MPL system (2), as discussed above, plays an important role for the verification procedures in Section 4. For instance, over boundedly periodic MPL systems it entails the decidability of verification problems (cf. Corollary 18).

▶ **Example 9.** Consider a two-dimensional MPL system

$$\mathbf{x}(k+1) = A \otimes \mathbf{x}(k), \quad A = \begin{bmatrix} 2 & 5 \\ 3 & 3 \end{bmatrix}, \tag{3}$$

that models a simple railway network shown in Figure 2, where $x_i(k)$ represents the time of the k-th departure at station S_i for $i \in \{1, 2\}$. The element $A(i, j)$ for $i \neq j$ corresponds to the time needed to travel from station S_j to S_i. The element $A(i, i)$ represents the delay for the next departure of a train from station S_i.

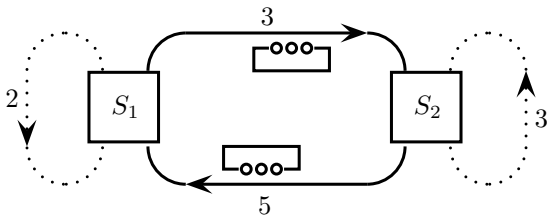


Figure 2 A simple railway network represented by an MPL system in (3).

Suppose the vector of initial departures is $\mathbf{x}(0) = [0\ 1]^\top$, then its corresponding orbit is

$$\pi = \begin{bmatrix}0\\1\end{bmatrix}, \begin{bmatrix}6\\4\end{bmatrix}, \begin{bmatrix}9\\9\end{bmatrix}, \begin{bmatrix}14\\12\end{bmatrix}, \dots$$

Notice that, the above orbit is periodic with transient $\mathsf{tr}(A, \mathbf{x}(0)) = 1$ and cyclicity $\mathsf{cyc}(A, \mathbf{x}(0)) = 2$, and is a $(2, 1)$-lasso.

2.2 Satisfiability Modulo Theory

Given a first-order formula ψ in a background theory T, Satisfiability Modulo Theory (SMT) refers to the problem of deciding whether there exists a model (i.e. an assignment to the free variables in ψ) that satisfies ψ [9]. For example, the formula $(x \le y) \wedge (x + 3 = z) \vee (z \ge y)$ within the theory of real numbers is satisfiable, and a valid model is $\{x := 5,\ y := 6,\ z := 8\}$.

SMT solvers can support different theories. A widely used theory is Quantifier-Free Linear Real Arithmetic (QF_LRA). A QF_LRA formula is an arbitrary Boolean combination of atoms in the form $\sum_i a_i \mathtt{x}_i \sim \alpha$, where $\sim \in \{>, \ge\}$, every $\mathtt{x}_i$ is a real variable, and every a_i and α are rational constants. Quantifier-Free Real Difference Logic (QF_RDL) is the subset of QF_LRA in which all atoms are restricted to the form $\mathtt{x}_i - \mathtt{x}_j \sim \alpha$. Both theories are decidable [9, Section 26.2]. Core to the main results of this work, it has been shown in [1] that any inequality in max-plus algebra can be translated into an RDL formula as follows.

▶ **Proposition 10** ([1]). Given real-valued variables $\mathtt{x}_1, \dots, \mathtt{x}_n$ and max-plus scalars $a_1, \dots, a_n$, $b_1, \dots, b_n \in \mathbb{R}_{\max}$, the inequality $F \equiv \bigoplus_{i=1}^n (\mathtt{x}_i \otimes a_i) \sim \bigoplus_{j=1}^n (\mathtt{x}_j \otimes b_j)$ is equivalent to $F^* \equiv \bigoplus_{i \in S_1} (\mathtt{x}_i \otimes a_i) \sim \bigoplus_{j \in S_2} (\mathtt{x}_j \otimes b_j)$, where $S_1 = \{1, \dots, n\} \setminus \{1 \le k \le n \mid a_k = \varepsilon \text{ or } \neg(a_k \sim b_k)\}$ and $S_2 = \{1, \dots, n\} \setminus \{1 \le k \le n \mid b_k = \varepsilon \text{ or } a_k \sim b_k\}$, respectively. Furthermore,

$$F^* \equiv \bigwedge_{j \in S_2} \left(\bigvee_{i \in S_1} (\mathtt{x}_i - \mathtt{x}_j \sim b_j - a_i) \right) \equiv \bigvee_{i \in S_1} \left(\bigwedge_{j \in S_2} (\mathtt{x}_i - \mathtt{x}_j \sim b_j - a_i) \right). \tag{4}$$

If $S_1 = \emptyset$ then $F^* \equiv \mathtt{false}$. On the other hand, if $S_2 = \emptyset$ then $F^* \equiv \mathtt{true}$.

Proposition 10 ensures that any inequality expression in max-plus algebra can be reduced to a simpler one in which no a variable appears on both sides. Then, the reduced inequality can be expressed as a QF_RDL formula either in conjunctive or disjunctive normal form[1].

[1] The readers are referred to the longer version of [1] in `https://arxiv.org/pdf/2007.00505.pdf` for a detailed proof.

As a direct consequence of Proposition 10, the MPL system dynamics in (2) can be expressed as a QF_RDL formula as follows:

$$\texttt{SymbMPL}(A, \mathcal{V}^{(k-1)}, \mathcal{V}^{(k)}) := \bigwedge_{i=1}^{n} (\texttt{ge}_i \wedge \texttt{eq}_i), \tag{5}$$

where $\texttt{ge}_i = \bigwedge_{j \in \texttt{fin}_i} (\texttt{x}_i^{(k)} - \texttt{x}_j^{(k-1)} \geq A(i,j))$ and $\texttt{eq}_i = \bigvee_{j \in \texttt{fin}_i} (\texttt{x}_i^{(k)} - \texttt{x}_j^{(k-1)} = A(i,j))$. The set $\mathcal{V}^{(k)} = \{\texttt{x}_1^{(k)}, \ldots, \texttt{x}_n^{(k)}\}$ contains SMT instances to encompass the states of the MPL system at the k-th bound while $\texttt{fin}_i$ is the indices of the finite elements of $A(i,\cdot)$.

3 Time-Difference Linear Temporal Logic

This section describes the logic we propose to express properties over MPL systems. We start by introducing the notions of Time-Difference (TD) proposition and of TD formula.

▶ **Definition 11.** A time-difference proposition over a vector of variables $\vec{\texttt{x}} = \langle \texttt{x}_1, \cdots, \texttt{x}_n \rangle$ is an atomic formula $p = \texttt{x}_i^{(k)} - \texttt{x}_j^{(l)} \sim \alpha$, where $i, j \in \{1, \ldots, n\}, k, l \in \mathbb{N}, \alpha \in \mathbb{R}$ and $\sim \in \{>, \geq\}$. We call p *initial* if $k = l = 0$: for the sake of simplicity, we write $\texttt{x}_i - \texttt{x}_j \sim \alpha$ instead. For $m \geq 0$, we write $p^{(m)} = \texttt{x}_i^{(k+m)} - \texttt{x}_j^{(l+m)} \sim \alpha$.

▶ **Definition 12.** A TD formula for a vector of variables $\vec{x}$ is defined according to the following grammar

$$f ::= \texttt{true} \mid p \mid \neg f \mid f_1 \wedge f_2,$$

where $p = \texttt{x}_i^{(k)} - \texttt{x}_j^{(l)} \sim \alpha$ is a TD proposition over $\vec{\texttt{x}}$. We call a TD formula f initial if all TD propositions appearing in f are initial ones.

Semantically, we interpret TD formulae on orbits[2]: given an orbit π and a TD formula f we say that $\pi \models f$ according to the following recursive rules.

$$\left.\begin{array}{ll} \pi \models \texttt{true} & \\ \pi \models \texttt{x}_i^{(k)} - \texttt{x}_j^{(l)} \sim \alpha & \text{iff } \pi[k]_i \sim \pi[l]_j + \alpha \\ \pi \models \neg f_1 & \text{iff } \pi \not\models f_1, \\ \pi \models f_1 \wedge f_2 & \text{iff } \pi \models f_1 \wedge \pi \models f_2. \end{array}\right\} \tag{6}$$

In the case of MPL systems, an orbit is uniquely determined by its initial vector $\mathbf{x}(0) = \pi[0]$; hence, one can write

$$\pi[0] \models f \text{ iff } \pi \models f, \tag{7}$$

or in general $\pi[i] \models f$ iff $\pi[i \ldots] \models f$. Finally, given an MPL system defined by matrix $A \in \mathbb{R}_{\max}^{n \times n}$ and a TD formula f, we say that $A \models f$ if all the orbits of A satisfy f (i.e., $\forall \pi \in \mathsf{Orb}(A). \pi \models f$).

Given an MPL system defined by matrix $A \in \mathbb{R}_{\max}^{n \times n}$ and a TD formula f, we can always rewrite f into an initial TD formula by means of the $\mathsf{get\_initial}(A, f)$ translation defined as follows. First, we translate all the TD propositions of f as the following inequality

$$\bigoplus_{r=1}^{n} \left(\texttt{x}_r + A^{\otimes k}(i,r)\right) \sim \bigoplus_{s=1}^{n} \left(\texttt{x}_s + \alpha + A^{\otimes l}(j,s)\right), \tag{8}$$

then, we can translate f into an initial TD formula by applying the rewriting (4) by Proposition 10.

[2] Equivalently, we could define the semantics on traces over $\vec{\texttt{x}}$.

▶ **Proposition 13.** Given a TD formula f and $A \in \mathbb{R}_{\max}^{n\times n}$, let g be $\mathsf{get\_initial}(A, f)$. Then, for any orbit $\pi \in \mathsf{Orb}(A)$, $\pi \models f$ iff $\pi \models g$.

Therefore, each (non-initial) TD formula w.r.t. an MPL system characterised by $A \in \mathbb{R}_{\max}^{n\times n}$ can be translated into an initial one, with the same satisfaction relation over any orbit $\pi \in \mathsf{Orb}(A)$. It is important to note that the translation of a non-initial TD formula f into an initial TD formula g by Proposition 13 may result in a larger formula, in terms of the number of its propositions. Notice that the number of propositions in (4) is at most $\lfloor \frac{n}{2} \rfloor \times (n - \lfloor \frac{n}{2} \rfloor)$.

▶ **Proposition 14.** Given a TD formula f and two similar orbits π, σ, $\pi \models f$ iff $\sigma \models f$.

We are now in the position to discuss TD specifications, which generalise TD formulae to temporal requirements. Formally, TD specifications are defined as LTL formulae in Release Positive Normal Form (PNF) [8, Definition 5.23], according to the following grammar:

$$\varphi := \mathtt{true} \mid \mathtt{false} \mid p \mid \neg p \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \bigcirc \varphi \mid \varphi_1 \ \mathsf{U}\ \varphi_2 \mid \varphi_1 \ \mathsf{R}\ \varphi_2, \tag{9}$$

where p is an initial TD proposition. We call this logic Time-Difference Linear Temporal Logic (TDLTL). It is important to note that one can express a TDLTL specification that contains a non-initial TD formula f by first translating f into an initial TD formula, as per Proposition 13. Furthermore, any initial TD formula is a TDLTL formula (without any temporal operators). Given an orbit π, the semantics of TDLTL formulae (9) is defined as:

$$\left.\begin{array}{ll}
\pi \models \mathtt{true} & \text{for all } \pi \in \mathsf{Orb}(A), \\
\pi \not\models \mathtt{false} & \text{for all } \pi \in \mathsf{Orb}(A), \\
\pi \models p & \text{iff } \pi[0] \models p, \\
\pi \models \neg p & \text{iff } \pi \not\models p, \\
\pi \models \varphi_1 \wedge \varphi_2 & \text{iff } \pi \models \varphi_1 \wedge \pi \models \varphi_2, \\
\pi \models \varphi_1 \vee \varphi_2 & \text{iff } \pi \models \varphi_1 \vee \pi \models \varphi_2, \\
\pi \models \bigcirc\varphi & \text{iff } \pi[1..] \models \varphi, \\
\pi \models \varphi_1 \ \mathsf{U}\ \varphi_2 & \text{iff } \exists j \geq 0.\ \pi[j..] \models \varphi_2 \text{ and } \forall 0 \leq i < j.\ \pi[i\ldots] \models \varphi_1, \\
\pi \models \varphi_1 \ \mathsf{R}\ \varphi_2 & \text{iff } \forall j \geq 0.\ \pi[j] \models \varphi_2 \text{ or } \exists i \geq 0.\ (\pi[i..] \models \varphi_1 \wedge \forall h \leq i.\ \pi[h..] \models \varphi_2) \\
\pi \models \Diamond\varphi & \text{iff } \exists j \geq 0.\ \pi[j..] \models \varphi, \\
\pi \models \Box\varphi & \text{iff } \forall j \geq 0.\ \pi[j..] \models \varphi.
\end{array}\right\} \tag{10}$$

Similar to (7), the semantics of TDLTL formulae over initial vectors in the case of an MPL system is as follows: $\pi[0] \models \varphi$ iff $\pi \models \varphi$.

▶ **Example 15.** Consider a TDLTL formula $\varphi = \Diamond\Box(\bigwedge_{i=1}^{2}(3 \leq \mathtt{x}_i^{(1)} - \mathtt{x}_i^{(0)} \leq 5))$[3] defined for the MPL system in (3). The specification assesses whether the system eventually reaches a state after which the delays of consecutive departures from both stations are always between 3 and 5 time units. φ has four non-initial TD propositions; the "initialised" translations are:

$$\begin{array}{llll}
\mathtt{x}_1^{(1)} - \mathtt{x}_1^{(0)} \geq 3 & \Leftrightarrow & \max\{\mathtt{x}_1 + 2, \mathtt{x}_2 + 5\} \geq \mathtt{x}_1 + 3 & \Leftrightarrow \quad \mathtt{x}_2 - \mathtt{x}_1 \geq -2, \\
\mathtt{x}_2^{(1)} - \mathtt{x}_2^{(0)} \geq 3 & \Leftrightarrow & \max\{\mathtt{x}_1 + 3, \mathtt{x}_2 + 3\} \geq \mathtt{x}_2 + 3 & \Leftrightarrow \quad \mathtt{true}, \\
\mathtt{x}_1^{(0)} - \mathtt{x}_1^{(1)} \geq -5 & \Leftrightarrow & \mathtt{x}_1 + 5 \geq \max\{\mathtt{x}_1 + 2, \mathtt{x}_2 + 5\} & \Leftrightarrow \quad \mathtt{x}_1 - \mathtt{x}_2 \geq 0, \\
\mathtt{x}_2^{(0)} - \mathtt{x}_2^{(1)} \geq -5 & \Leftrightarrow & \mathtt{x}_2 + 5 \geq \max\{\mathtt{x}_1 + 3, \mathtt{x}_2 + 3\} & \Leftrightarrow \quad \mathtt{x}_2 - \mathtt{x}_1 \geq -2.
\end{array}$$

Hence, the "initialised version" for φ is $\varphi' = \Diamond\Box((\mathtt{x}_2 - \mathtt{x}_1 \geq -2) \wedge (\mathtt{x}_1 - \mathtt{x}_2 \geq 0))$; equivalent to $\Diamond\Box(0 \leq \mathtt{x}_1 - \mathtt{x}_2 \leq 2)$.

[3] We write $k \leq x - y \leq w$ as a shorthand for $(x - y \leq w) \wedge (y - x \leq -k)$.

Given an MPL system (2), characterised by matrix $A \in \mathbb{R}_{\max}^{n\times n}$, and a TDLTL formula φ, we say that $A \models \phi$ if all the orbits of A satisfy ϕ (i.e., $\forall \pi \in \mathsf{Orb}(A).\ \pi \models \phi$). Furthermore, if the dynamics of (2) are defined over a set of initial conditions $X \subseteq \mathbb{R}^n$,

$$\mathsf{Orb}(A, X) \models \varphi \text{ iff } \forall \pi \in \mathsf{Orb}(A).\ \pi[0] \in X \implies \pi \models \varphi. \tag{11}$$

3.1 Encoding Bounded Counterexamples

This section describes the procedure to generate an SMT instance corresponding to a bounded counterexample of the TDLTL formula φ. By Corollary 7, such a counterexample can be generated over a lasso. First, we define the bounded version of (10) up to bound k for a (k, l)-lasso π as follows:

$$\left.\begin{array}{ll}
\pi \models_k p & \text{iff } \pi[0] \models p, \\
\pi \models_k \neg p & \text{iff } \pi \not\models_k p, \\
\pi \models_k \varphi_1 \wedge \varphi_2 & \text{iff } \pi \models_k \varphi_1 \wedge \pi \models_k \varphi_2, \\
\pi \models_k \varphi_1 \vee \varphi_2 & \text{iff } \pi \models_k \varphi_1 \vee \pi \models_k \varphi_2, \\
\pi \models_k \bigcirc\varphi & \text{iff } \pi[1..] \models_k \varphi, \\
\pi \models_k \varphi_1 \mathsf{U} \varphi_2 & \text{iff } \exists 0 \le j \le k.\ \pi[j..] \models_k \varphi_2 \text{ and } \forall 0 \le i < j.\ \pi[i..] \models_k \varphi_1, \\
\pi \models_k \varphi_1 \mathsf{R} \varphi_2 & \text{iff } \forall 0 \le j \le k.\ \pi[j] \models_k \varphi_2 \text{ or} \\
 & \quad \exists 0 \le i \le k.\ (\pi[i..] \models_k \varphi_1 \wedge \forall h \le i.\ \pi[h..] \models_k \varphi_2), \\
\pi \models_k \Diamond\varphi & \text{iff } \exists 0 \le j \le k.\ \pi[j,,] \models_k \varphi, \\
\pi \models_k \Box\varphi & \text{iff } \forall 0 \le j \le k.\ \pi[j..] \models_k \varphi.
\end{array}\right\} \tag{12}$$

Notice that, for a (k, l)-lasso π, $\pi[(k+1)..]$ is similar to $\pi[l..]$. Thus, it is straightforward to see that $\pi \models_k \varphi$ implies $\pi \models \varphi$.

▶ **Example 16.** For the TDLTL formula $\varphi' = \Diamond\Box(0 \le \mathtt{x}_1 - \mathtt{x}_2 \le 2)$ in Example 15 and a $(2, 1)$-lasso π in Example 9, one could check that $\pi \models_k \varphi'$ for $k = 2$.

We now describe how to translate a bounded counterexample of a TDLTL formula into an SMT instance. Suppose ψ is the negation of φ i.e. $\psi \equiv \neg\varphi$. We recall that φ and ψ are assumed to be in positive normal form (9). The notation ${}_l[\psi]_k^m$ denotes the witness encoding of ψ (equivalently, the counterexample encoding of φ) at position $0 \le m \le k$ over a (k, l)-lasso. Similar to the description in [10], the encoding can be formulated as follows:

$$\begin{aligned}
{}_l[p]_k^m &:= p^{(m)}, & {}_l[\neg p]_k^m &:= \neg({}_l[p]_k^m), \\
{}_l[\psi_1 \wedge \psi_2]_k^m &:= {}_l[\psi_1]_k^m \wedge {}_l[\psi_2]_k^m, & {}_l[\psi_1 \vee \psi_2]_k^m &:= {}_l[\psi_1]_k^m \vee {}_l[\psi_2]_k^m, \\
{}_l[\bigcirc\psi]_k^m &:= \begin{cases} {}_l[\psi]_k^{m+1}, & \text{if } m < k \\ {}_l[\psi]_k^l, & \text{otherwise} \end{cases} & {}_l[\Diamond\psi]_k^m &:= \bigvee_{j=\min\{m,l\}}^{k} {}_l[\psi]_k^j \\
{}_l[\psi_1 \ \mathsf{U}\ \psi_2]_k^m &:= \bigvee_{j=m}^{k} \left({}_l[\psi_2]_k^j \wedge \bigwedge_{n=m}^{j-1} {}_l[\psi_1]_k^n \right) \vee & {}_l[\Box\psi]_k^m &:= \bigwedge_{j=\min\{m,l\}}^{k} {}_l[\psi]_k^j \\
 & \quad \bigvee_{j=l}^{m-1} \left({}_l[\psi_2]_k^j \wedge \bigwedge_{n=m}^{k} {}_l[\psi_1]_k^n \wedge \bigwedge_{n=l}^{j-1} {}_l[\psi_1]_k^n \right) \\
{}_l[\psi_1 \ \mathsf{R}\ \psi_2]_k^m &:= \left(\bigwedge_{j=\min\{m,l\}}^{k} {}_l[\psi_2]_k^j \right) \vee \bigvee_{j=m}^{k} \left({}_l[\psi_1]_k^j \wedge \bigwedge_{n=m}^{j} {}_l[\psi_2]_k^n \right) \vee \\
 & \quad \bigvee_{j=l}^{m-1} \left({}_l[\psi_1]_k^j \wedge \bigwedge_{n=m}^{k} {}_l[\psi_2]_k^n \wedge \bigwedge_{n=l}^{j} {}_l[\psi_2]_k^n \right),
\end{aligned}$$

where p is an initial TD proposition. The final formula, which is satisfiable iff there exists a (k,l)-lasso π such that $\pi \not\models_k \varphi$, is given by:

$$\bigwedge_{i=0}^{k} \mathtt{SymbMPL}(A, \mathcal{V}^{(i)}, \mathcal{V}^{(i+1)}) \wedge \mathsf{Loop}(A, k+1, l, \lambda) \wedge \ {}_l[\psi]_k^0, \tag{13}$$

where λ is the max-plus eigenvalue of A and $\mathsf{Loop}(A, k+1, l, \lambda)$ represents the looping constraint i.e., $\bigwedge_{i=1}^{n}(\mathbf{x}_i^{(k+1)} - \mathbf{x}_i^{(l)} = \lambda \times (k-l+1))$. Recall that, if the orbit of $\mathbf{x}(0)$ w.r.t. A is (k,l)-lasso, then $\mathbf{x}(k+1) = (\lambda \times (k-l+1)) \otimes \mathbf{x}(l)$. Furthermore, the first conjunct of (13) corresponds to the executions of (2) up to bound k.

By the same procedure used in Proposition 13, one can translate $\mathsf{Loop}(A, k+1, l, \lambda) \wedge {}_l[\psi]_k^0$ into an SMT formula over the variables $\mathcal{V}^{(0)}$ only (i.e., instead of representing the variables at each time in the orbit, we only define the formula over the initial state). Abusing the notation of TD formulae, we indicate the "initialised" version of (13) as

$$\mathsf{get\_initial}\left(A, \bigwedge_{i=0}^{k} \mathsf{Loop}(A, k+1, l, \lambda) \wedge \ {}_l[\psi]_k^0\right). \tag{14}$$

The first conjunct of (13) is not included because all TD propositions in (14) are initial ones. The number of TD propositions in (14) may be much larger compared to (13), especially for TDLTL formulae with multiple temporal operators. On the other hand, the encoding (14) has an advantage w.r.t. the number of variables: notice that (13) consists of variables from $\mathcal{V}^{(0)} \cup \ldots \cup \mathcal{V}^{(k+1)}$, whereas (14) involves $\mathcal{V}^{(0)}$ only.

3.2 An Upper Bound for the Completeness Threshold

The notion of *completeness threshold* refers to an index k such that, if no counterexample with length k or less for a LTL formula φ is found, then φ in fact holds over all infinite behaviours in the model. The discussion of how to compute the (upper bound of the) completeness threshold is given in [19, 31]. In this section, we show that the upper bound of the completeness threshold to verify (11) for any TDLTL formula over an MPL system (2) is determined by its pair transient/cyclicity.

▶ **Proposition 17.** Given a periodic MPL system (2) with a set of initial conditions X and a TDLTL formula φ, the upper bound of the completeness threshold to verify $\mathsf{Orb}(A, X) \models \varphi$ is given by $\mathsf{tr}(A, X) + \mathsf{cyc}(A, X) - 1$.

4 Model Checking of Max-Plus Linear Systems

This section describes the model-checking algorithms we devise for MPL systems. We build on the basic idea of BMC, that is to find a bounded counterexample of a given specification with a specific length k. If no such counterexample is found, then one increases k by one and searches corresponding counterexamples, until a known completeness threshold is reached, or until the problem becomes intractable. The reader is referred to [10, 11, 12] for a more detailed description.

Given an MPL system (2) with a set of initial conditions $X \subseteq \mathbb{R}^n$ and a TDLTL formula φ (9), we present procedures to verify whether $\mathsf{Orb}(A, X) \models \varphi$. In this paper, we assume that the underlying MPL system is periodic and the set of initial conditions X can be expressed as a general LRA formula. The procedures are integrated with the method computing the transient and cyclicity of MPL from a given set of initial conditions in [1]. We recall that such pair of transient and cyclicity can be used as an upper bound of the completeness threshold.

In the first part of Section 4, we present incremental algorithms to verify (11) where the bounded counterexample of a TDLTL formula φ is encoded for each iteration. Due to the periodic behaviour of MPL systems, such a counterexample corresponds to an orbit with transient l and cyclicity c. Unlike the usual BMC where the bound is increased by one, the novel procedures increase the step bound by finding the existence of a larger orbit with transient $l' > l$ or cyclicity c' not divides c. In the second part of the section, we describe "upfront" procedures where one only needs to encode the bounded counterexample of φ w.r.t. the upper bound of the completeness threshold given by Proposition 17.

4.1 Incremental Approaches

Orbits of MPL systems are potentially periodic with transient l and cyclicity c. Hence, in incremental procedures, we search a finite counterexample of a TDLTL formula φ with length k in a shape of (k, l)-lasso, where initially we set $l = 0$ and $c = 1$ (the smallest possible values). From these values, we generate the looping constraint $\mathsf{Loop}(A, k+1, l, \lambda)$ and ${}_l[\neg\varphi]_k^0$, where λ is the max-plus eigenvalue of A. The formula $X \wedge F$, with F given by (13) or (14), corresponds to a counterexample of φ i.e., a (k, l)-lasso $\pi \in \mathsf{Orb}(A, X)$, such that $\pi \not\models \varphi$.

We use an SMT solver to check the satisifiability of $X \wedge F$. If the SMT solver reports `SAT` then a counterexample is found. On the other hand, if the SMT solver reports `UNSAT`, we increment the step bound. Instead of increasing the bound by one, we use the SMT solver to check whether there exists an orbit with larger transient l' or cyclicity c'. The value of $l' + c' - 1$ becomes the new bound. These two steps are repeated until either 1) a counterexample is found; 2) the bound cannot be incremented; or 3) the bound is deemed too large. The second outcome suggests that the specification is valid, since the bound exceeds the upper bound of the completeness threshold given by Proposition 17. For the last outcome, we use a large integer as a termination condition.

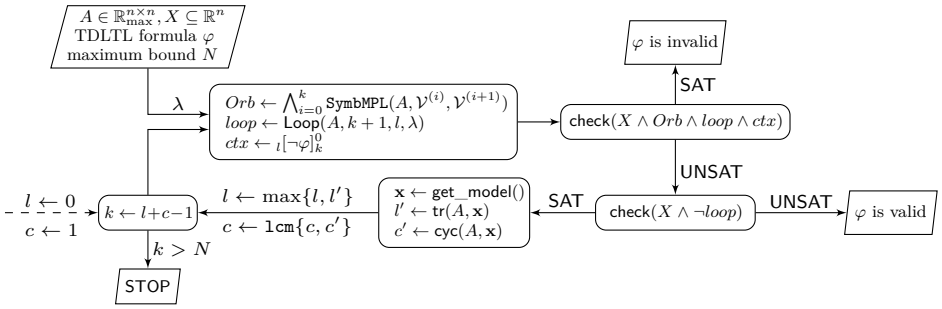
The incremental approaches are illustrated in Figure 3. We name the procedure that uses the encoding in (13) *unrolled-incremental*, since the first conjunct of (13) represents the execution of (2) up to bound k. On the other hand, the alternative procedure with the encoding in (14) is called *initialised-incremental*, due to the translation from Proposition 13.

4.2 Upfront Approaches

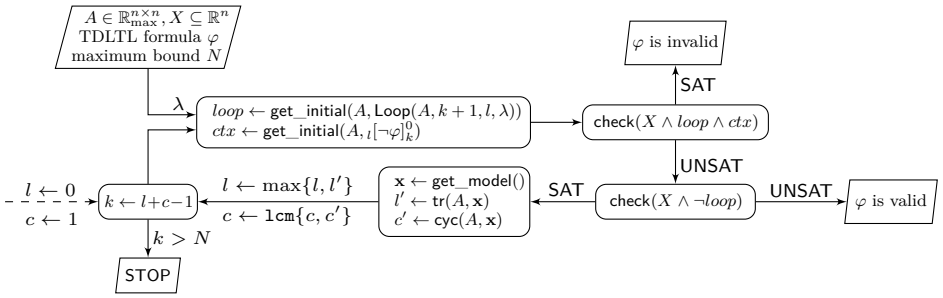
Upfront approaches exploit the fact that the upper bound of the completeness threshold in Proposition 17 is unrelated to the TDLTL formula φ, and that it can be computed via SMT-based techniques, as in [1, Algoritm 3]. Hence, the upfront versions of the procedures in Figure 3 is obtained by generating (13) or (14) with $l = \mathsf{tr}(A, X)$ and $k = \mathsf{tr}(A, X) + \mathsf{cyc}(A, X) - 1$. Together with the set of initial conditions X, we check the satisfaction of the resulting SMT instance. If it is satisfiable, then φ is invalid. On the other hand, if it is unsatisfiable, then φ is valid. The resulting procedures are then called *unrolled-upfront* and *initalised-upfront*, respectively.

4.3 Completeness and Decidability

Proposition 17 suggests that the completeness of the procedures in Sections 4.1-4.2 depends on the existence of $\mathsf{tr}(A, X)$ and $\mathsf{cyc}(A, X)$ for a given set X of initial conditions. From the discussed classification of MPL systems over their periodic behaviour, we can conclude that if (2) is boundedly periodic, then all procedures are complete. It is also shown in [1] that the computation of transient for unboundedly periodic MPL systems is semi-complete. We further summarise this discussion in Corollary 18.



(a) Unrolled-incremental procedure using the encoding in (13).



(b) Initialised-incremental procedure using the encoding in (14).

Figure 3 Incremental Approaches. The function check is implemented in an SMT solver. The integer N represents the allowed maximum bound.

▶ **Corollary 18.** Suppose we have an MPL system (2) with a set of initial condition X and a TDLTL formula φ. If the underlying MPL system is boundedly periodic (resp. unboundedly periodic) then the proposed procedures are complete (resp. semi-complete) and verifying $\mathsf{Orb}(A, X) \models \varphi$ is a decidable (resp. semi-decidable) problem.

5 Related Work

A verification by model checking procedure for MPL systems has been firstly discussed in [2]. It employs the *abstraction* [8, 27] of the underlying MPL system into an equivalent Piecewise Affine (PWA) model [28]. This results in an abstract model with a finite number of (abstract) states expressed as Difference Bound Matrices (DBM) [25, 33]. The approach allows to verify specifications (as LTL formulae) over the abstract model: if the specification is true, then it is also valid for the original MPL system [8]. However, the invalidity of specification on the abstract model does not necessarily imply the same conclusion on the concrete model. A refinement procedure is then proposed in [2]: the abstract model can be refined, so that it is in a bisimulation relation [8, Definition 7.1] with the original MPL system. This means that the specification is true on MPL system iff it holds on the bisimilar abstract model. Unfortunately, the proposed refinement procedure in general does not terminate, even for irreducible MPL systems. A sufficient condition for the existence of bisimilar abstract model is given in [2, Theorem 5]. The surveyed techniques suffer from the curse of dimensionality, since the abstraction computation runs on $\mathrm{O}(n^{(n+3)})$ time where n is the dimension of the state matrix.

The recent work [34] applies a different approach to verify MPL systems. A set of predicates is used to generate the abstraction of an MPL system. Predicates are automatically selected from the state matrix, as well as from the specifications under consideration. This

predicate abstraction of MPL systems, reminiscent of [18, 27], is shown to be more scalable than the PWA-based abstraction in [2]. A standard BMC procedure is then applied to verify given specifications. It also has been shown in [34, Lemma 2] that the completeness threshold for this BMC procedure is determined by the pair of transient and cyclicity.

Despite successive ameliorations, the main drawback of the aforementioned methods is their scalability, as they can only be applied to MPL systems with relatively few variables (the dimension n of vector $\mathbf{x}$ in this work). There are a few elements contributing to the computational bottleneck (time and memory requirements) of these approaches. First, the worst-case complexity to generate the abstraction of n-dimensional MPL systems is $O(n^{n+3})$ [2]. As a result, the number of abstract states grows exponentially, as n increases. Second, the refinement procedures in [2, 34] potentially lead to state-explosion problems. Another disadvantage is the limitations related to utilising the DBM data structure: in [34], each proposition in the form of $\mathbf{x}_i - \mathbf{x}_j \sim c$ where $1 \leq i, j \leq n, \sim \in \{>, \geq\}$ and $\alpha \in \mathbb{R}$ is transformed into a DBM in $\mathbb{R}^n$. As such, the more propositions are in an LTL formula, the more DBMs are needed, and therefore the larger number of abstract states.

6 Experiments

In our experimental evaluation, we compare the procedures introduced in Section 4 against alternative approaches based on the symbolic model checker nuXmv [15] and the existing abstraction-based techniques presented in [2, 33, 34].

6.1 Encoding in nuXmv (IC3)

We present a procedure to verify (11) using nuXmv [15], a symbolic model checker for the analysis of synchronous, finite- and infinite-state systems. This can be done by encoding the maximization operation into SMV language. For instance, $x_1' = \max\{x_1 + a_1, x_2 + a_2\}$ can be expressed as:

```
TRANS ((next(x_1) >= (a_1 + x_1)) & (next(x_1) >= (a_2 + x_2))) &
      ((next(x_1) = (a_1 + x_1)) | (next(x_1) = (a_2 + x_2)));
```

Notice that the above expression is similar to (5). However, unlike the procedures in Figure 3, nuXmv requires that the lasso is in the canonical form. This means that, in Figure 1, the states $\mathbf{x}(l)$ and $\mathbf{x}(k+1)$ must be equal. This condition can be achieved if the corresponding matrix has eigenvalue equal to 0. It is straightforward that if matrix A in (2) has eigenvalue λ then $A \otimes (-\lambda)$ has eigenvalue 0.

The procedure to verify (11) using nuXmv is as follows. First, we update the matrix by subtracting all elements with the corresponding eigenvalue. Then, we generate an SMV file from the matrix and the specification. Finally, we call nuXmv to verify the specification using command check_ltlspec_ic3 that implements the algorithm described in [22]. The algorithm works by trying to prove that any existing abstract fair loop is covered by a given set of well-founded relations, and leverages the efficiency and incrementality of the IC3ia [17] underlying safety checker.

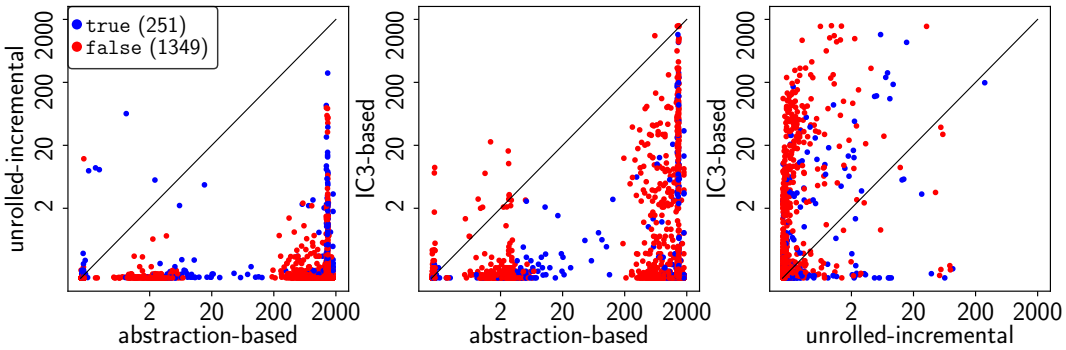
6.2 Experimental Setup

For the experiments, we generate 20 irreducible matrices of dimension $n \in \{4, 6, 8, 10\} \cup \{12, 16, \ldots, 40\}$ with $\frac{n}{2}$ finite elements in each row, where the values of the finite elements are integers between 1 and 20. The locations of the finite elements are chosen randomly. We

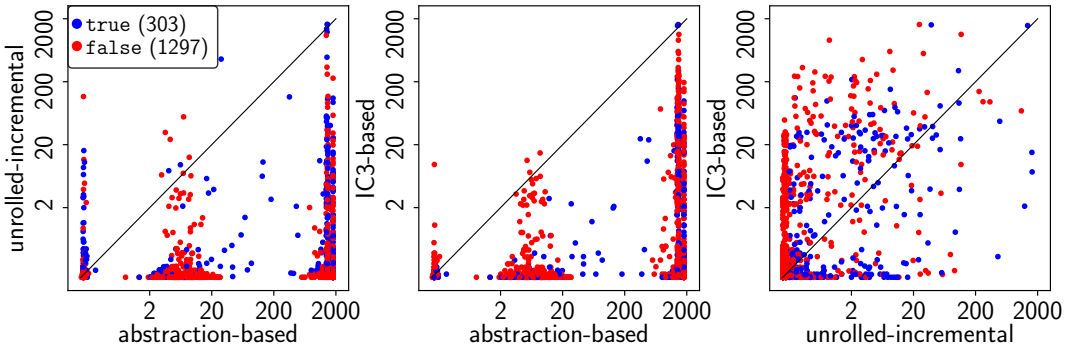
focus on irreducible matrices to ensure the termination of the procedures. With regards to the specifications, for each n, we generate randomly 20 TDLTL formulae where the propositions are in the form of $\mathtt{x}_i - \mathtt{x}_j \sim \alpha$, $i, j \in \{1, \ldots, n\}, \sim \in \{>, \geq\}$, and α is an integer within the interval $[-20, 20]$. The randomised TDLTL formulae are generated using Spot [26], which categorises them according to their size: namely, the size of a TDLTL formula φ is intended as number of operators and propositions in φ. For instance, the size of $p \wedge q$ and $p \mathbin{\mathsf{U}} q$ is both 3 while $\Box p$ has size of 2. The experiments have been implemented using the SMT solver Z3 [23] on an Intel(R) Xeon(R) Gold 6248 CPU @ 2.50GHz with 120GB of RAM memory. The set of initial conditions for each experiment is $X = \mathbb{R}^n$. The experiments are implemented for each pair of matrix and specification. Thus, there are $20 \times 20 \times 2$ experiments for each n. We set 30 minutes as a `timeout` limit.

6.3 Results

Figure 4 illustrates the performance comparison of abstraction-based, unrolled-incremental, and IC3-based algorithms for $n \in \{4, 6, 8, 10\}$. These three algorithms are similar in the sense that they unroll the dynamics of MPL systems up to the underlying step bound. The scattered plots (in logarithmic scale) represent the running times (in second) for a pair of algorithms. It is clear that the abstraction-based algorithm is in general the least efficient one. As expected, the dimension of MPL systems heavily affects the runtime of the abstraction-based procedure: all experiments for 10-dimensional MPL systems yield `timeout` (see Table 1 in Appendix B). For this reason, we do not pursue the abstraction-based experiments for higher dimensions.



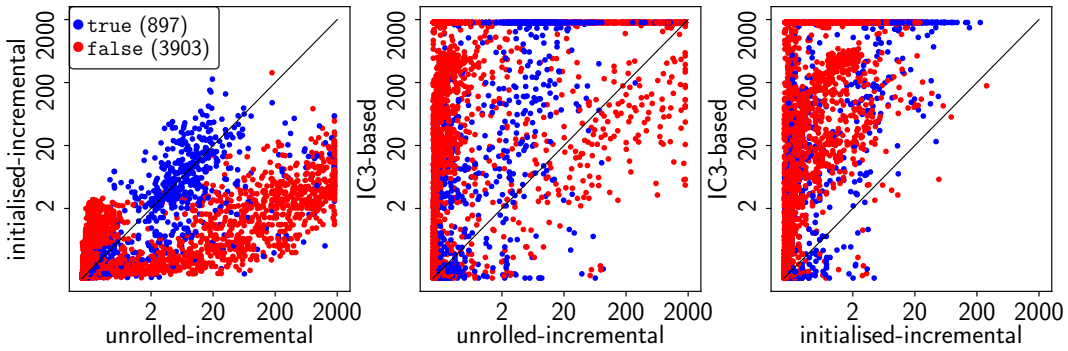
(a) The plots of experiments with TDLTL formulae of size 5.



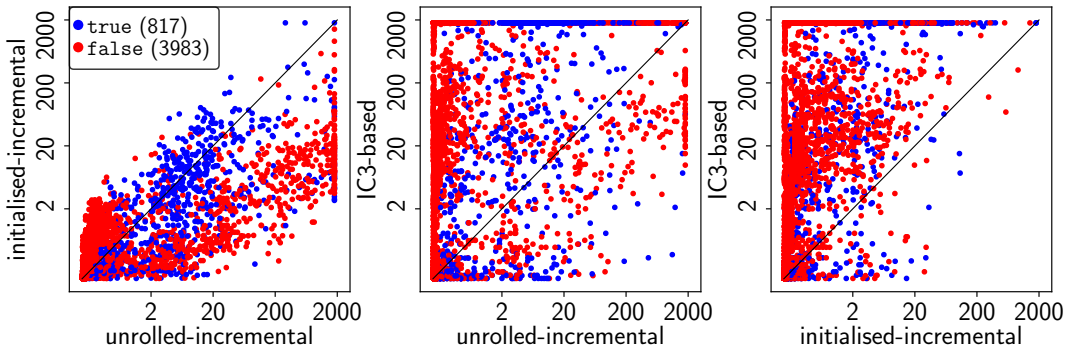
(b) The plots of experiments with TDLTL formulae of size 10.

Figure 4 Comparison of abstraction-based, unrolled-incremental and IC3-based algorithms.

The plots in Figure 4 also indicate that the unrolled-incremental algorithm is more efficient than the IC3-based technique. To shed light on this finding, we then compare the performance of IC3-based technique with SMT-based incremental (unrolled and initialised) algorithms. As depicted in Figure 5, the proposed algorithms outperform the procedure that employs IC3. We recall that, in incremental algorithms, the bound of the counterexample is not increased by one. Hence, they are indeed more effective to find a long counterexample. Furthermore, in each iteration, the search of counterexample is implemented with a fixed loopback bound. Such a bound is related to the current value of transient l.



(a) The plots of experiments with TDLTL formulae of size 5.



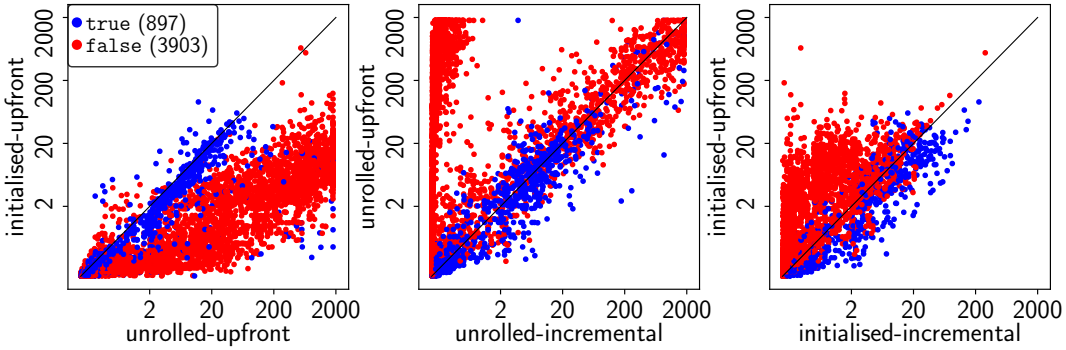
(b) The plots of experiments with TDLTL formulae of size 10.

Figure 5 Comparison of incremental algorithms and IC3-based technique.

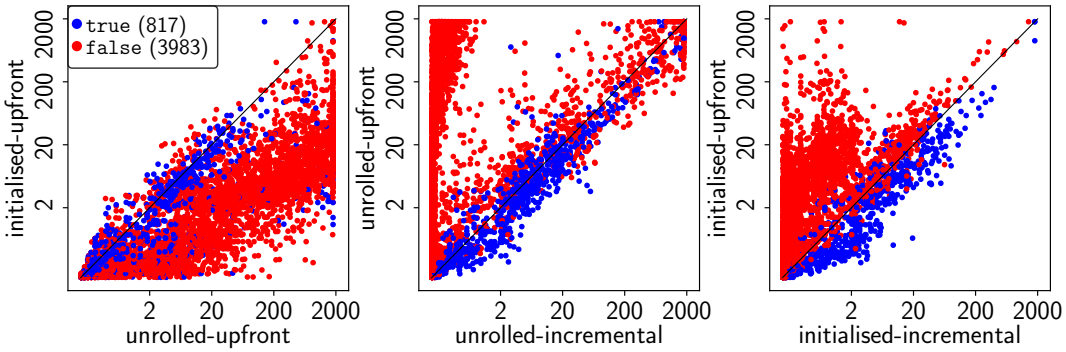
While the SMT-based incremental algorithms take a longer time to verify specifications with size of 10, the performance of the IC3-based algorithm is relatively similar. In fact, it is the dimension of the matrix which affects the running time of IC3-based procedure: the number of experiments that yield `timeout` increases as the dimension grows. On the other hand, the performance incremental algorithms are affected by the upper bound of completeness threshold given by Proposition 17; in particular for experiments whose corresponding specification is valid. Between the unrolled-incremental and initialised-incremental algorithms, it seems that the latter one is the winner. We recall that in Figure 3(b), all TD propositions in (14) are initial ones. Therefore, there are only one set of variables that appeared in (14). In comparison, there are $k+1$ sets of variables in (13) which correspond to the states of MPL system (2) from bound 0 until k.

We then compare the performance between incremental and upfront algorithms, as shown in Figure 6. As expected, upfront procedures are faster than incremental ones when the specification is valid. On the other hand, incremental algorithms are much more efficient when the specification is invalid. This is due to the fact that the counterexample may be

found at a smaller bound than the completeness threshold given in Proposition 17. As in incremental approaches, the procedure which uses one set of variables (initialised-upfront) is faster than the other one that employs multiple sets of variables (unrolled-upfront).



(a) The plots of experiments with TDLTL formulae of size 5.



(b) The plots of experiments with TDLTL formulae of size 10.

Figure 6 Comparison of incremental and upfront algorithms.

7 Conclusions

In this paper, we addressed the problem of proving temporal properties over MPL systems. We defined TDLTL as a suitable formalism for the specification of temporal properties, and proposed a family of correct and complete SMT-based algorithms for the problem of checking TDLTL over MPL systems. We derived suitable completeness thresholds from the periodicity of MPL, and optimized the encoding by means of max-plus algebraic transformations. The results from a broad set of benchmarks demonstrate that the proposed approach can handle large models, which is completely out of reach for existing abstraction-based approaches, and that it outperforms a the baseline encoding into nuXmv.

As future work, we plan to investigate the case of reducible matrices, and to generalize the approach for *parametric* MPL, where the matrices may contain symbolic expressions.

References

1 Alessandro Abate, Alessandro Cimatti, Andrea Micheli, and Muhammad Syifa'ul Mufid. Computation of the transient in max-plus linear systems via SMT-solving. In Nathalie Bertrand and Nils Jansen, editors, *Formal Modeling and Analysis of Timed Systems*, pages 161–177, Cham, 2020. Springer International Publishing.

2 Dieky Adzkiya, Bart De Schutter, and Alessandro Abate. Finite abstractions of max-plus-linear systems. *IEEE Transactions on Automatic Control*, 58(12):3039–3053, 2013.

3 Dieky Adzkiya, Bart De Schutter, and Alessandro Abate. Backward reachability of autonomous max-plus-linear systems. *IFAC Proceedings Volumes*, 47(2):117–122, 2014.

4 Dieky Adzkiya, Bart De Schutter, and Alessandro Abate. Forward reachability computation for autonomous max-plus-linear systems. In Erika Abraham and Klaus Havelund, editors, *Intl. Conf. on Tools and Algorithms for the Construction and Analysis of Systems* (TACAS'14), volume 8413 of *LNCS*, pages 248–262. Springer, 2014.

5 Dieky Adzkiya, Bart De Schutter, and Alessandro Abate. Computational techniques for reachability analysis of max-plus-linear systems. *Automatica*, 53:293–302, 2015.

6 Mohsen Alirezaei, Ton JJ van den Boom, and Robert Babuska. Max-plus algebra for optimal scheduling of multiple sheets in a printer. In *Proc. 31st American Control Conference (ACC), 2012*, pages 1973–1978, June 2012.

7 François Baccelli, Guy Cohen, Geert Jan Olsder, and Jean-Pierre Quadrat. *Synchronization and linearity: an algebra for discrete event systems*. John Wiley & Sons Ltd, 1992.

8 Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

9 Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. Satisfiability modulo theories. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 825–885. IOS Press, 2009. doi:10.3233/978-1-58603-929-5-825.

10 Armin Biere, Alessandro Cimatti, Edmund Clarke, and Yunshan Zhu. Symbolic model checking without BDDs. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 193–207. Springer, 1999. doi:10.1007/3-540-49059-0_14.

11 Armin Biere, Alessandro Cimatti, Edmund M Clarke, Ofer Strichman, Yunshan Zhu, et al. Bounded model checking. *Advances in Computers*, 58(11):117–148, 2003.

12 Armin Biere, Keijo Heljanko, Tommi Junttila, Timo Latvala, and Viktor Schuppan. Linear encodings of bounded LTL model checking. *Logical Methods in Computer Science*, 2(5):1–64, 2006. doi:10.2168/LMCS-2(5:5)2006.

13 J-L Bouquard, Christophe Lenté, and J-C Billaut. Application of an optimization problem in max-plus algebra to scheduling problems. *Discrete Applied Mathematics*, 154(15):2064–2079, 2006.

14 Chris A Brackley, David S Broomhead, M Carmen Romano, and Marco Thiel. A max-plus model of ribosome dynamics during mRNA translation. *Journal of Theoretical Biology*, 303:128–140, 2012.

15 Roberto Cavada, Alessandro Cimatti, Michele Dorigatti, Alberto Griggio, Alessandro Mariotti, Andrea Micheli, Sergio Mover, Marco Roveri, and Stefano Tonetta. The nuxmv symbolic model checker. In *International Conference on Computer Aided Verification*, pages 334–342. Springer, 2014.

16 Bernadette Charron-Bost, Matthias Függer, and Thomas Nowak. New transience bounds for max-plus linear systems. *Discrete Applied Mathematics*, 219:83–99, 2017.

17 Alessandro Cimatti, Alberto Griggio, Sergio Mover, and Stefano Tonetta. IC3 modulo theories via implicit predicate abstraction. In Erika Ábrahám and Klaus Havelund, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 20th International Conference, TACAS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014. Proceedings*, volume 8413 of *Lecture Notes in Computer Science*, pages 46–61. Springer, 2014.

18 Edmund Clarke, Orna Grumberg, Muralidhar Talupur, and Dong Wang. Making predicate abstraction efficient. In *Proc. International Conference on Computer Aided Verification 2003 (CAV'03)*, pages 126–140. Springer, 2003.

19 Edmund Clarke, Daniel Kroening, Joël Ouaknine, and Ofer Strichman. Completeness and complexity of bounded model checking. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*, pages 85–96. Springer, 2004. doi:10.1007/978-3-540-24622-0_9.

20 J-P Comet. Application of max-plus algebra to biological sequence comparisons. *Theoretical computer science*, 293(1):189–217, 2003.

21 RA Cuninghame-Green and P Butkovic. Generalised eigenproblem in max-algebra. In *International Workshop on Discrete Event Systems*, pages 236–241. IEEE, 2008.

22 Jakub Daniel, Alessandro Cimatti, Alberto Griggio, Stefano Tonetta, and Sergio Mover. Infinite-state liveness-to-safety via implicit abstraction and well-founded relations. In Swarat Chaudhuri and Azadeh Farzan, editors, *Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part I*, volume 9779 of *Lecture Notes in Computer Science*, pages 271–291. Springer, 2016.

23 Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *Intl. Conf. on Tools and Algorithms for the Construction and Analysis of Systems* (TACAS'08), volume 4963 of *LNCS*, pages 337–340. Springer, 2008.

24 Bart De Schutter. On the ultimate behavior of the sequence of consecutive powers of a matrix in the max-plus algebra. *Linear Algebra and its Applications*, 307(1-3):103–117, 2000.

25 David L Dill. Timing assumptions and verification of finite-state concurrent systems. In J. Sifakis, editor, *Intl. Conf. on Computer Aided Verification* (CAV'89), volume 407 of *Lecture Notes in Computer Science*, pages 197–212, Hiedelberg, 1989. Springer.

26 Alexandre Duret-Lutz, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Etienne Renault, and Laurent Xu. Spot 2.0 - a framework for ltl and *omega*-automata manipulation. In *International Symposium on Automated Technology for Verification and Analysis*, pages 122–129. Springer, 2016.

27 Susanne Graf and Hassen Saïdi. Construction of abstract state graphs with PVS. In *In Proc. International Conference on Computer Aided Verification (CAV'97)*, pages 72–83. Springer, 1997.

28 Wilhemus Heemels, Bart De Schutter, and Alberto Bemporad. Equivalence of hybrid dynamical models. *Automatica*, 37(7):1085–1091, July 2001.

29 Bernd Heidergott, Geert Jan Olsder, and Jacob Van der Woude. *Max Plus at work: modeling and analysis of synchronized systems: a course on Max-Plus algebra and its applications*. Princeton University Press, 2014.

30 Aleksey Imaev and Robert P Judd. Hierarchial modeling of manufacturing systems using max-plus algebra. In *Proc. American Control Conference, 2008*, pages 471–476, June 2008.

31 Daniel Kroening, Joël Ouaknine, Ofer Strichman, Thomas Wahl, and James Worrell. Linear completeness thresholds for bounded model checking. In *International Conference on Computer Aided Verification*, pages 557–572. Springer, 2011.

32 Glenn Merlet, Thomas Nowak, and Sergei Sergeev. Weak CSR expansions and transience bounds in max-plus algebra. *Linear Algebra and its Applications*, 461:163–199, 2014.

33 M.S. Mufid, D. Adzkiya, and A. Abate. Tropical abstractions of max-plus linear systems. In D.N. Jansen and P. Prabhakar, editors, *Int. Conf. Formal Modeling and Analysis of Timed Systems* (FORMATS'18), pages 271–287. Springer, 2018.

34 Muhammad Syifa'ul Mufid, Dieky Adzkiya, and Alessandro Abate. Bounded model checking of max-plus linear systems via predicate abstractions. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 142–159. Springer, 2019.

35 Thomas Nowak and Bernadette Charron-Bost. An overview of transience bounds in max-plus algebra. *Tropical and Idempotent Mathematics and Applications*, 616:277–289, 2014.

36 Gerardo Soto Y Koelemeijer. *On the behaviour of classes of min-max-plus systems*. PhD thesis, Delft University of Technology, 2003.

A Proofs of Propositions

▶ **Proposition 6.** An orbit π is a lasso iff $\pi[0]$ admits local transient and cyclicity.

Proof. ($\Leftarrow$) Suppose the transient and cyclicity for $\pi[0]$ is l anc c, respectively. Thus, $\pi[l+c+j] = \lambda c \otimes \pi[l+j]$ for $j \geq 0$ and π is a $(l+c-1, l)$-lasso.
($\Rightarrow$) Suppose π is a (k, l)-lasso. Then, there exists $\beta \in \mathbb{R}$ such that $\pi[k+1+j] = \beta \otimes \pi[l+j]$ for $j \geq 0$, or equivalently $\pi[j+k-l+1] = \beta \otimes \pi[j]$ for $j \geq l$. This means that $\mathsf{tr}(A, \pi[0]) = l$ and $\mathsf{cyc}(A, \pi[0]) = k - l + 1$. ◀

▶ **Corollary 7.** An MPL system (2) is periodic iff all orbits $\pi \in \mathsf{Orb}(A)$ are lasso.

Proof. Direct consequence of Proposition 6. ◀

▶ **Proposition 13.** Given a TD formula f and $A \in \mathbb{R}_{\max}^{n\times n}$, let g be $\mathsf{get\_initial}(A, f)$. Then, for any $\pi \in \mathsf{Orb}(A)$, $\pi \models f$ iff $\pi \models g$.

Proof. Due to Definition 12 and (6), it suffices to prove for a non-initial TD proposition $p = \mathbf{x}_i^{(k)} - \mathbf{x}_j^{(l)} \sim \alpha$. Notice that that p is equivalent to an inequality (8) which can be translated into an initial TD formula by Proposition 10. This completes the proof. ◀

▶ **Proposition 14.** Given a TD formula f and two similar orbits π, σ, we have $\pi \models f$ iff $\sigma \models f$.

Proof. Suppose $\pi = \mathbf{x}(0)\mathbf{x}(1)\ldots$ and $\lambda = \mathbf{y}(0)\mathbf{y}(1)\ldots$ with $\mathbf{x}(m) = \beta \otimes \mathbf{y}(m)$ for $\beta \in \mathbb{R}$ and $m \geq 0$. The proof follows from the fact that for any TD proposition $p = \mathbf{x}_i^{(k)} - \mathbf{x}_j^{(l)} \sim \alpha$,

$$\begin{aligned}\pi \models p &\text{ iff } x_i(k) \sim x_j(l) + \alpha,\\ &\text{iff } y_i(k) + \beta \sim y_j(l) + \beta + \alpha,\\ &\text{iff } y_i(k) \sim y_j(l) + \alpha.\end{aligned}$$

The last assertion indicates that $\sigma \models p$. ◀

▶ **Proposition 17.** Given a periodic MPL system (2) with a set of initial conditions X and a TDLTL formula φ, the upper bound of completeness threshold to verify whether $\mathsf{Orb}(A, X) \models \varphi$ is given by $\mathsf{tr}(A, X) + \mathsf{cyc}(A, X) - 1$.

Proof. By Corollary 7, all orbits $\pi \in \mathsf{Orb}(A, X)$ are lasso. Recall that, if an initial vector $\mathbf{x}(0) \in X$ admits local transient $\mathsf{tr}(A, \mathbf{x}(0)) = l$ and cyclicity $\mathsf{cyc}(A, \mathbf{x}(0)) = c$ then the corresponding orbit is a (k, l)-lasso where $k = l + c - 1$. Consequently, the upper bound of completeness threshold to verify (11) is given by the largest possible of such k i.e., $\mathsf{tr}(A, X) + \mathsf{cyc}(A, X) - 1$. ◀

▶ **Corollary 18.** Suppose we have an MPL system (2) with a set of initial condition X and a TDLTL formula φ. If the underlying MPL system is boundedly periodic (resp. unboundedly periodic) then the proposed procedures are complete (resp. semi-complete) and verifying $\mathsf{Orb}(A, X) \models \varphi$ is a decidable (resp. semi-decidable) problem.

Proof. The completeness of the procedures follows from the fact that computing transient $\mathsf{tr}(A, X)$ is complete for boundedly periodic MPL systems, but semi-complete for unboundedly periodic ones. As a result, and because of Proposition 17, verifying $\mathsf{Orb}(A, X) \models \varphi$ is decidable for boundedly MPL systems and semi-decidable for unboundedly periodic ones. ◀

B The Runtime and Memory Consumption of Experiments

The following tables present the average and maximum running time (in second) and memory consumption (in MB) of the algorithms. The notation `timeout`(r) means that there are r (out of 400) failed experiments due to `timeout`.

Table 1 The runtime and memory consumption of abstraction-based algorithm.

n	Experiments with TDLTL formulae of size 5		Experiments with TDLTL formulae of size 10	
	runtime	memory	runtime	memory
4	{0.08, 1.42}	{26.29, 61.71}	{0.12, 12.11}	{28.3, 73.61}
6	`timeout`(9)	{69.45, 427.21}	`timeout`(5)	{110.84, 1238.51}
8	`timeout`(96)	{2715.27, 6401.11}	`timeout`(314)	{3517.04, 9837.71}
10	`timeout`(400)	{318.0, 569.61}	`timeout`(400)	{329.22, 608.91}

Table 2 The runtime and memory consumption of IC3-based algorithm.

n	Experiments with TDLTL formulae of size 5		Experiments with TDLTL formulae of size 10	
	runtime	memory	runtime	memory
4	{0.1, 8.89}	{9.94, 84.71}	{0.09, 9.69}	{11.21, 93.41}
6	{0.41, 22.71}	{22.34, 110.11}	{0.54, 24.81}	{21.32, 122.21}
8	{8.04, 1101.53}	{34.79, 275.71}	{5.02, 194.61}	{37.29, 170.11}
10	`timeout`(3)	{61.81, 262.21}	`timeout`(3)	{63.46, 1525.81}
12	`timeout`(12)	{69.12, 309.71}	`timeout`(10)	{71.51, 375.81}
16	`timeout`(150)	{96.37, 427.81}	`timeout`(115)	{92.51, 301.91}
20	`timeout`(243)	{105.9, 562.91}	`timeout`(233)	{105.39, 904.31}
24	`timeout`(189)	{100.78, 1326.61}	`timeout`(244)	{103.34, 345.71}
28	`timeout`(208)	{119.45, 1052.21}	`timeout`(255)	{104.97, 951.31}
32	`timeout`(228)	{125.82, 745.61}	`timeout`(254)	{103.79, 250.21}
36	`timeout`(232)	{126.33, 685.81}	`timeout`(293)	{118.79, 365.01}
40	`timeout`(245)	{149.08, 398.61}	`timeout`(268)	{154.99, 677.11}

Table 3 The runtime and memory consumption of unrolled-incremental algorithm.

n	Experiments with TDLTL formulae of size 5		Experiments with TDLTL formulae of size 10	
	runtime	memory	runtime	memory
4	{0.28, 63.77}	{6.63, 23.91}	{0.59, 116.33}	{9.24, 25.31}
6	{0.1, 5.62}	{8.84, 22.51}	{2.12, 457.88}	{9.23, 30.21}
8	{0.18, 3.37}	{10.99, 24.11}	{2.51, 227.76}	{11.62, 38.31}
10	{2.92, 280.91}	{22.94, 73.11}	`timeout`(1)	{23.78, 91.01}
12	{0.85, 34.69}	{13.76, 41.71}	{2.11, 120.39}	{13.44, 44.71}
16	{2.43, 57.04}	{21.3, 83.91}	{8.31, 1111.33}	{19.94, 78.61}
20	{13.79, 428.42}	{34.51, 230.91}	`timeout`(2)	{32.85, 220.91}
24	`timeout`(1)	{51.14, 544.51}	{29.31, 1130.4}	{42.59, 424.01}
28	`timeout`(17)	{120.89, 638.31}	`timeout`(14)	{66.89, 516.71}
32	`timeout`(16)	{132.17, 693.31}	`timeout`(24)	{77.32, 787.61}
36	`timeout`(31)	{123.9, 643.51}	`timeout`(19)	{82.76, 681.31}
40	`timeout`(71)	{179.88, 748.51}	`timeout`(73)	{152.97, 777.01}

Table 4 The runtime and memory consumption of initialised-incremental algorithm.

n	Experiments with TDLTL formulae of size 5		Experiments with TDLTL formulae of size 10	
	runtime	memory	runtime	memory
4	{0.03, 2.02}	{6.22, 22.61}	{0.15, 14.68}	{10.25, 23.41}
6	{0.07, 4.42}	{9.15, 22.61}	{0.46, 40.3}	{11.03, 23.41}
8	{0.1, 2.5}	{11.57, 22.51}	{0.39, 36.57}	{13.58, 23.41}
10	{0.35, 10.33}	{21.26, 23.41}	{5.18, 584.62}	{21.89, 23.91}
12	{0.39, 43.91}	{14.22, 22.21}	{0.64, 31.41}	{14.42, 23.41}
16	{0.32, 8.44}	{14.97, 18.81}	{1.2, 36.03}	{14.62, 18.81}
20	{0.82, 61.72}	{16.02, 61.71}	{1.47, 58.78}	{15.64, 23.21}
24	{1.65, 141.43}	{17.17, 181.51}	{1.49, 51.42}	{17.05, 29.41}
28	{3.27, 286.22}	{19.24, 30.41}	{3.8, 185.61}	{17.89, 58.01}
32	{2.56, 21.97}	{19.94, 29.71}	`timeout`(3)	{19.32, 58.61}
36	{8.1, 226.56}	{22.18, 41.31}	{8.0, 396.32}	{22.06, 52.31}
40	{8.93, 84.29}	{23.84, 43.01}	{18.64, 918.9}	{24.39, 50.91}

Table 5 The runtime and memory consumption of unrolled-upfront algorithm.

n	Experiments with TDLTL formulae of size 5		Experiments with TDLTL formulae of size 10	
	runtime	memory	runtime	memory
4	{0.41, 76.63}	{18.57, 27.21}	`timeout`(2)	{20.08, 34.01}
6	{0.44, 40.4}	{20.06, 29.11}	{12.6, 772.53}	{21.14, 33.41}
8	{0.63, 25.38}	{22.55, 34.31}	{8.98, 615.95}	{23.27, 39.11}
10	{4.54, 274.57}	{29.26, 81.31}	`timeout`(6)	{29.35, 98.21}
12	{1.44, 14.62}	{28.94, 53.81}	{7.41, 241.33}	{30.23, 55.41}
16	{7.15, 179.75}	{45.82, 169.31}	{17.22, 1122.95}	{44.93, 152.81}
20	{35.35, 610.82}	{87.02, 431.31}	`timeout`(4)	{81.37, 451.91}
24	`timeout`(1)	{142.51, 570.51}	`timeout`(2)	{138.93, 567.01}
28	`timeout`(3)	{223.51, 852.31}	`timeout`(14)	{241.62, 853.41}
32	`timeout`(30)	{307.83, 1095.81}	`timeout`(45)	{334.34, 1162.61}
36	`timeout`(29)	{381.8, 806.91}	`timeout`(40)	{395.85, 804.91}
40	`timeout`(125)	{587.1, 1096.21}	`timeout`(153)	{633.39, 1141.11}

Table 6 The runtime and memory consumption of initialised-upfront algorithm.

n	Experiments with TDLTL formulae of size 5		Experiments with TDLTL formulae of size 10	
	runtime	memory	runtime	memory
4	{0.05, 3.41}	{16.63, 22.81}	{4.49, 1393.45}	{18.43, 23.61}
6	{0.15, 10.63}	{18.75, 22.81}	{2.71, 730.72}	{18.92, 23.61}
8	{0.21, 3.94}	{18.97, 23.11}	{0.94, 44.02}	{20.22, 23.61}
10	{1.0, 183.35}	{21.68, 23.51}	`timeout`(1)	{22.24, 23.81}
12	{0.53, 15.0}	{20.64, 23.01}	{1.32, 20.2}	{21.15, 23.51}
16	{0.6, 9.53}	{21.01, 22.01}	{6.28, 1725.02}	{21.21, 22.81}
20	{1.66, 23.53}	{21.99, 24.51}	{5.05, 469.43}	{22.18, 24.51}
24	{3.3, 118.33}	{22.96, 27.71}	`timeout`(1)	{23.09, 28.01}
28	{7.15, 656.1}	{24.03, 32.61}	`timeout`(1)	{24.26, 34.41}
32	{4.77, 82.64}	{25.34, 35.81}	`timeout`(3)	{25.82, 36.61}
36	{12.3, 115.36}	{27.82, 42.21}	`timeout`(1)	{27.82, 41.61}
40	{15.82, 124.64}	{29.18, 47.41}	`timeout`(1)	{29.83, 47.21}