Bounded Model Checking of Max-Plus Linear Systems via Predicate Abstractions

Muhammad Syifa'ul Mufid[1(✉)], Dieky Adzkiya[2], and Alessandro Abate[1]

[1] Department of Computer Science, University of Oxford, Oxford, UK
{muhammad.syifaul.mufid,alessandro.abate}@cs.ox.ac.uk
[2] Department of Mathematics, Institut Teknologi Sepuluh Nopember, Surabaya, Indonesia
dieky@matematika.its.ac.id

Abstract. This paper introduces the abstraction of max-plus linear (MPL) systems via predicates. Predicates are automatically selected from system matrix, as well as from the specifications under consideration. We focus on verifying time-difference specifications, which encompass the relation between successive events in MPL systems. We implement a bounded model checking (BMC) procedure over a predicate abstraction of the given MPL system, to verify the satisfaction of time-difference specifications. Our predicate abstractions are experimentally shown to improve on existing MPL abstractions algorithms. Furthermore, with focus on the BMC algorithm, we can provide an explicit upper bound on the completeness threshold by means of the transient and the cyclicity of the underlying MPL system.

1 Introduction

Max-Plus-Linear (MPL) systems are a class of discrete-event systems, with dynamics based on two binary operations (maximisation and addition) over a max-plus semiring. MPL systems are used to model synchronisation phenomena without concurrency. These systems have been used in many areas, such as manufacturing [27], transportation [24], and biological systems [10,18].

Classical analysis of MPL systems is conducted using algebraic approaches [4,24]. Recently, an alternative take based on formal abstractions has been developed to verify MPL systems against quantitative specifications [1] that are general and expressive. The performance and scalability of the abstraction approach has been later improved by employing tropical operations [29] that are native to the max-plus semiring.

This work pushes the envelop on scalability of formal abstractions of MPL systems. We newly apply predicate abstractions (PA) and bounded model checking (BMC) for the verification of MPL systems over time-difference specifications. Predicate abstractions are an abstraction approach that leverages a set of predicates, and have been classically used for software and hardware verification

© Springer Nature Switzerland AG 2019
É. André and M. Stoelinga (Eds.): FORMATS 2019, LNCS 11750, pp. 142–159, 2019.
https://doi.org/10.1007/978-3-030-29662-9_9

[16,21], for the abstraction of programs [6,15], and for reachability analysis of hybrid systems [3].

BMC is a symbolic model checking approach that leverages SAT solvers. The basic idea is to attempt finding counterexamples with a length bounded by some integer. If no counterexamples are found, the length is greedily increased. The approach is sound (counterexamples are correct), and complete (no counterexamples are admitted) whenever a completeness threshold (CT) for the length is reached [8,9]. Whilst there exist results on correct upper-bounds on the CT, in practice BMC is run until the underlying problem becomes intractable.

This paper has two specific contributions. The first contribution is related to the abstraction approach. Moving beyond [1,29], where the abstraction procedures are based on the translation of MPL systems into piecewise affine (PWA) systems, in this work we newly employ PA. Namely, we determine a set of predicates such that the dynamics within each partitioning region is affine. In other words, there is no need to compute PWA systems anymore.

The second contribution is related to the model-checking approach. [1] employs standard model checking to verify the abstract transition system. In this paper, we leverage BMC: notice that PA naturally yield Boolean encodings that can be relevant for the SAT-based BMC procedure. We focus on time-difference specifications. Since we are working on abstractions, counterexample generated by the BMC procedure needs to be checked for spuriousness (cf. Algorithms 5 and 6). Whenever a counterexample is spurious, we refine the abstract transition using the procedure in [2], combined with lazy abstraction [26]. Finally, for the considered time-difference specifications, we show that the CT can be upper-bounded by means of the transient and cyclicity of the concrete MPL system - such bounds are in general tighter than those obtained working on the abstract transition system. As a side result, we provide a few instance of "direct verification", where the model checking of MPL models can be performed straightforwardly for time-difference specifications.

The paper is organised as follows. Section 2 describes the basics of models, abstraction techniques and temporal logic formulae used in this work. It also contains the notion of time-difference over MPL systems. The contributions of this paper are contained in Sects. 3 and 4. Proofs of the propositions and lemmas are provided in the longer version of this paper [30]. The comparison of abstraction procedures is presented in Sect. 5, with PA implemented in C++ and model checking run over NuSMV [11]. We also compare the completeness threshold w.r.t. the transient and cyclicity of MPL systems with those that are computed by NuSMV. The paper is concluded in Sect. 6.

2 Model and Preliminaries

2.1 Max-Plus Linear Systems

By max-plus semiring we understand an algebraic structure $(\mathbb{R}_{\max}, \oplus, \otimes)$ where $\mathbb{R}_{\max} := \mathbb{R} \cup \{\varepsilon := -\infty\}$ and $a \oplus b := \max\{a, b\}$, $a \otimes b := a + b \quad \forall a, b \in \mathbb{R}_{\max}$. The set of $n \times m$ matrices over max-plus semiring is denoted as $\mathbb{R}_{\max}^{n \times m}$.

Two binary operations of a max-plus semiring can be extended to matrices as follows

$$[A \oplus B](i,j) = A(i,j) \oplus B(i,j),$$
$$[A \otimes C](i,j) = \bigoplus_{k=1}^{m} A(i,k) \otimes C(k,j),$$

where $A, B \in \mathbb{R}_{\max}^{n\times m}, C \in \mathbb{R}_{\max}^{m\times p}$. Given $r \in \mathbb{N}$, the max-plus algebraic power of $A \in \mathbb{R}_{\max}^{n\times n}$ is denoted by $A^{\otimes r}$ and corresponds to $A \otimes \ldots \otimes A$ (r times).

A Max-Plus Linear (MPL) system is defined as

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

where $A \in \mathbb{R}_{\max}^{n\times n}$ is the system matrix and $\mathbf{x}(k) = [x_1(k) \ldots x_n(k)]^\top$ is the state variables [4]. In particular, for $i \in \{1,\ldots,n\}$, $x_i(k+1) = \max\{A(i,1) + x_1(k), \ldots, A(i,n) + x_n(k)\}$. In applications, $\mathbf{x}$ represents the time stamps of the discrete events, while k corresponds to the event counter. Therefore, it is more convenient to take $\mathbb{R}^n$ (instead of $\mathbb{R}_{\max}^n$) as the state space.

Definition 1 (Precedence Graph [4]). The precedence graph of A, denoted by $\mathcal{G}(A)$, is a weighted directed graph with nodes $1,\ldots,n$ and an edge from j to i with weight $A(i,j)$ if $A(i,j) \neq \varepsilon$. □

Definition 2 (Regular Matrix [24]). A matrix $A \in \mathbb{R}_{\max}^{n\times n}$ is called regular if there is at least one finite element in each row. □

Definition 3 (Irreducible Matrix [4]). A matrix $A \in \mathbb{R}_{\max}^{n\times n}$ is called irreducible if the corresponding precedence graph $\mathcal{G}(A)$ is strongly connected. □

Recall that a directed graph is strongly connected if for two different nodes i, j of the graph, there exists a path from i to j [4,20]. The weight of a path $p = i_1 i_2 \ldots i_k$ is equal to the total weight of the corresponding edges i.e. $|p| = A(i_2, i_1) + \ldots + A(i_k, i_{k-1})$. 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 path [4].

Every irreducible matrix $A \in \mathbb{R}_{\max}^{n\times n}$ admits a unique max-plus eigenvalue $\lambda \in \mathbb{R}$, which corresponds to the weight of critical circuit in $\mathcal{G}(A)$. Furthermore, by Proposition 1 next, A satisfies the so-called transient condition:

Proposition 1 (Transient Condition [4]). For an irreducible matrix $A \in \mathbb{R}_{\max}^{n\times n}$ and its corresponding max-plus eigenvalue $\lambda \in \mathbb{R}$, there exist $k_0, c \in \mathbb{N}$ such that $A^{\otimes(k+c)} = \lambda c \otimes A^{\otimes k}$ for all $k \geq k_0$. The smallest such k_0 and c are called the transient and the cyclicity of A, respectively. □

Example 1. Consider a 2×2 MPL system that represents a simple railway network [24]:

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

Its max-plus eigenvalue is $\lambda = 4$, whereas the transient and cyclicity for the matrix are $k_0 = c = 2$. □

Any given MPL system can be translated into a Piece-Wise Affine (PWA) system [23]. A PWA system comprises of spatial regions with corresponding PWA dynamics. The regions are generated from all possible coefficients $\mathbf{g} = (g_1, \ldots, g_n) \in \{1, \ldots, n\}^n$, which satisfies $A(i, g_i) \neq \varepsilon$ for $1 \leq i \leq n$. As shown in [1], the region corresponding to $\mathbf{g}$ is

$$\mathsf{R}_{\mathbf{g}} = \bigcap_{i=1}^{n} \bigcap_{j=1}^{n} \left\{ \mathbf{x} \in \mathbb{R}^n | x_{g_i} - x_j \geq A(i,j) - A(i, g_i) \right\}. \quad (3)$$

One could check that for each non-empty $\mathsf{R}_{\mathbf{g}}$ and $\mathbf{x}(k) \in \mathsf{R}_{\mathbf{g}}$, the MPL system (1) can be rewritten as the following affine dynamics:

$$x_i(k+1) = x_{g_i}(k) + A(i, g_i), \quad i = 1, \ldots, n. \quad (4)$$

Notice that (4) can be expressed as $\mathbf{x}(k+1) = A_{\mathbf{g}} \otimes \mathbf{x}(k)$, where $A_{\mathbf{g}}$ is a region matrix [29] for the coefficient $\mathbf{g}$.

2.2 Time Differences in MPL Systems

We consider delays occurring between events governed by (1). Delays can describe the difference of two events corresponding to the same event counter but at different variable indices (i.e. $x_i(k) - x_j(k)$), or the difference of two consecutive events for the same index (i.e. $x_i(k+1) - x_i(k)$). This paper focuses on the later case although, in general, the results of this paper can be applied to the former case.

We write the $(k+1)^{\text{th}}$ time difference for the ith component as $t_i(k) = x_i(k+1) - x_i(k)$. One can see that

$$t_i(k) = \max_{j^* \in \mathtt{fin}_i} \{x_{j^*}(k) + A(i, j^*)\} - x_i(k), \quad (5)$$

where $\mathtt{fin}_i$ is the set containing the indices of finite elements of $A(i, \cdot)$.[1]

2.3 Transition Systems and Linear Temporal Logic

Definition 4 (Transition System [5]). A transition system is formulated by a tuple $(S, T, I, \mathcal{AP}, L)$, where

- S is a set of states,
- $T \subseteq S \times S$ is a transition relation,
- $I \subseteq S$ is a set of initial states,
- $\mathcal{AP}$ is a set of atomic propositions, and
- $L : S \to 2^{\mathcal{AP}}$ is a labelling function. □

[1] For the sake of simplicity, we write the elements of $\mathtt{fin}_i$ in a strictly increasing order.

A *path* of TS is defined as a sequence of states $\pi = s_0 s_1 \ldots$, where $s_0 \in I$ and $(s_i, s_{i+1}) \in T$ for all $i \geq 0$. We denote $\pi[i] = s_{i-1}$ as the ith state of π. Furthermore, $|\pi|$ represents the number of transitions in π.

Linear temporal logic (LTL) is one of the predominant logics that are used for specifying properties over the set of atomic propositions [5]. LTL formulae are recursively defined as follows.

Definition 5 (Syntax of LTL [5]). LTL formulae over the set of atomic propositions $\mathcal{AP}$ are constructed according to the following grammar:

$$\varphi := \text{true} \mid a \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \bigcirc \varphi \mid \varphi_1 \,\mathsf{U}\, \varphi_2,$$

where $a \in \mathcal{AP}$. □

The symbol $\bigcirc$ (next) and U (until) are called temporal operators. Two additional operators, $\Diamond$ (eventually) and $\Box$ (always), are generated via the until operators: $\Diamond\varphi = \text{true} \,\mathsf{U}\, \varphi$ and $\Box\varphi = \neg\Diamond\neg\varphi$. We refer to [5] for the semantics of LTL formulae including the satisfaction relation $\models$ over transition systems.

2.4 Abstractions and Predicate Abstractions

Abstractions are techniques to generate a finite and smaller model from a large or even infinite-space (i.e., a continuous-space model, e.g., an MPL system) model. Abstractions can reduce the verification of a temporal property φ over the original model (a *concrete* model with state space S), to checking a related property on a simpler *abstract* model (over $\hat{S}$) [5]. The mapping from S to $\hat{S}$ is called *abstraction function*.

From a (concrete) transition system $TS = (S, T, I, \mathcal{AP}, L)$ and an abstraction function $f : S \to \hat{S}$, the (abstract) transition system $TS_f = (\hat{S}, T_f, I_f, \mathcal{AP}, L_f)$ is generated from TS as follows: (i) $I_f = \{f(s) \mid s \in I\}$, (ii) $(f(s), f(s')) \in T_f$ if $(s, s') \in T$, and (iii) $L_f(f(s)) = L(s)$, for all $s \in S$.

The important relation between TS and TS_f is that the former is *simulated* by the latter (which is denoted by $TS \preceq TS_f$). In detail, all behaviour on concrete transition system occur on the abstract one. The formal definition of simulation relation can be found in [5, Definition 7.47]. Furthermore, given an LTL formula φ, $TS_f \models \varphi$ implies $TS \models \varphi$ [5,13].

Predicate abstractions [13,17,19,22] denote abstraction methods that use a set of *predicates* $P = \{p_1, \ldots, p_k\}$ to characterise the abstract states. Predicates are identified from the concrete model, and possibly from the specification(s) under consideration. Each predicate p_i corresponds to a Boolean variable b_i and each abstract state $\hat{s} \in \hat{S}$ corresponds to a Boolean assignment of these k Boolean variables [13]. Therefore, we obtain that $|\hat{S}| \leq 2^k$. An abstract state will be labelled with predicate p_i if the corresponding b_i is true in that state. For this reason, predicates also serve as atomic propositions [13].

The predicates are also used to define an abstraction function between the concrete and abstract state spaces. A concrete state $s \in S$ will be related to an abstract state $\hat{s} \in \hat{S}$ iff the truth value of p_i on s equals the value of b_i

on $\hat{s}$. The abstraction function for predicate abstractions is defined as $f(s) = \bigwedge_{i=1}^{k} \text{val}(s, p_i)$, where $\text{val}(s, p_i) = b_i$ if p_i is satisfied in s, otherwise $\neg b_i$.

3 Predicate Abstractions of MPL Systems

3.1 Related Work

The notion of abstractions of an MPL system has been first introduced in [1]: there, it leverages translation of an MPL system into the corresponding PWA system. The resulting abstract states are expressed as Difference-Bound Matrices (DBM). A more efficient procedure for MPL abstractions via max-plus algebraic operations is later discussed in [29].

3.2 Generation of the Predicates

Considering an abstraction via a set of predicates, the first issue is to find appropriate predicates. Recall that related abstraction techniques [1,29] explore the connection between MPL and PWA systems and use DBMs to represent the abstract states. Similarly, predicates here are chosen such that the dynamics in the resulting abstract states are affine as in (3) and can be expressed as DBMs. Following these considerations, the predicates are defined as an inequality $p \equiv x_i - x_j \sim c$ where $\sim \in \{>, \geq\}$[2], $c \in \mathbb{R}$. For simplicity, we may write a predicate as a tuple $p \equiv (i, j, c, s)$ where $s = 1$ if $\sim \; = \; \geq$, otherwise $s = 0$. The negation of p then can be written as $\neg p \equiv (j, i, -c, 1 - s)$.

From the PWA region in (3), c can be chosen from the difference of two finite elements of the state matrix $A \in \mathbb{R}_{\max}^{n \times n}$ at the same row. In detail, if $A(k, j) \neq \varepsilon$ and $A(k, i) \neq \varepsilon$ with $i < j$ and $1 \leq k \leq n$, then we get a predicate $(i, j, A(k, j) - A(k, i), 1)$.

Algorithm 1 shows a procedure to generate the predicates from an MPL system. For each $k \in \{1, \ldots, n\}$, P_k is a set of predicates generated from $A(k, \cdot)$. If there are exactly $m > 1$ finite elements at each row of A then $|P_k| = \binom{m}{2}$ and in general $|\bigcup_{k=1}^{n} P_k| \leq n\binom{m}{2}$: indeed, it is possible to get the same predicate from two different rows when $A(k_1, j) - A(k_1, i) = A(k_2, j) - A(k_2, i)$ for $k_1 \neq k_2$.

As mentioned before, predicates can also be associated to given specifications. In this paper, we focus on time-difference specifications that are generated from a set of *time-difference propositions*. For $\alpha \in \mathbb{R}$, we define a time-difference proposition '$t_i \sim \alpha$' to reason the condition that $x_i' - x_i \sim \alpha$. We remove the counter event k for the sake of simplicity.

One can rewrite (5) as $t_i = \max_{j^* \in \texttt{fin}_i}\{x_{j^*} + A(i, j^*)\} - x_i$. Therefore, from $t_i \sim \alpha$ for $\sim \in \{>, \geq, <, \leq\}$ we have $\max_{j^* \in \texttt{fin}_i}\{x_{j^*} + A(i, j^*)\} - x_i \sim \alpha$. The number of predicates corresponding to '$t_i \sim \alpha$' is bounded by $|\texttt{fin}_i|$. For each $j^* \in \texttt{fin}_i$ we get a predicate $x_{j^*} - x_i \sim \alpha - A(i, j^*)$. However, in case of $i \in \texttt{fin}_i$, or in other words $A(i, i) \neq \varepsilon$, $x_i - x_i \sim \alpha - A(i, j^*)$ is not a predicate. Algorithm 2 shows how to generate the predicates w.r.t. a time-difference proposition.

[2] In this paper, we always use $p \equiv x_i - x_j \geq c$ as a predicate.

Algorithm 1. Generation of predicates from an MPL system

Input: $A \in \mathbb{R}_{\max}^{n \times n}$,
Output: P_{mat}, a set of predicates

```
 1: procedure mpl2pred(A, k)              ▷ generation of predicates from the k^th row of A
 2:   P_k ← ∅
 3:   fin_k := Find(A(k,·) ≠ ε)            ▷ fin_k is a vector consisting the index of
 4:   for j ∈ {2,...,|fin_k|}                finite elements of A(k,·), fin_k[i] is
 5:     for i ∈ {1,...,j−1}                  the i^th element of fin_k
 6:       P_k ← P_k ∪ {(fin_k[i], fin_k[j], A(k, fin_k[j]) − A(k, fin[i]), 1)}
 7:     end
 8:   end
 9:   return P_k
10: end

11: procedure mpl2pred(A)                 ▷ generation of predicates from matrix A
12:   P_mat ← ∅
13:   for k ∈ {1,...,n}                   ▷ generation of predicates for each row of matrix A
14:     P_mat ← P_mat ∪ mpl2pred(A, k)       and storing the resulting predicates in P_mat
15:   end
16:   return P_mat
17: end
```

Algorithm 2. Generation of predicates from a time-difference proposition

Input: $A \in \mathbb{R}_{\max}^{n \times n}$, a matrix containing exactly m finite elements in each row
$t_i \sim \alpha$, a time-difference proposition
Output: P_{time}, a set of predicates

```
 1: procedure td2pred(A, t_i ∼ α)
 2:   P_time ← ∅
 3:   A(i,i) ← ε
 4:   fin_i ← Find(A(i,·) ≠ ε)
 5:   if ∼ ∈ {>, ≥} then
 6:     for j* ∈ fin_i
 7:       P_time ← P_time ∪ {(j*, i, α − A(i, j*), s)}     ▷ s is 0 if ∼ is > and s is 1 if ∼ is ≥
 8:     end
 9:   else if ∼ ∈ {<, ≤} then
10:     for j* ∈ fin_i
11:       P_time ← P_time ∪ {(i, j*, A(i, j*) − α, s)}     ▷ each predicate uses operator > or ≥
12:     end
13:   end
14:   return P_time
15: end
```

3.3 Generation of Abstract States

This section starts by describing the procedure to generate abstract states via a set of predicates. We denote P as the set of predicates generated by Algorithms 1 and 2, i.e. $P = P_{mat} \cup P_{time} = \{p_1, \ldots, p_k\}$. Let $\hat{S}$ be a set of abstract states defined over Boolean variables $B = \{b_1, \ldots, b_k\}$, where the truth value of b_i depends on that of p_i. For each Boolean variable b_i, we define the corresponding DBM as follows: $\texttt{DBM}(b_i) = \{\mathbf{x} \in \mathbb{R}^n \mid p_i \text{ is true in } \mathbf{x}\}$ and $\texttt{DBM}(\neg b_i) = \{\mathbf{x} \in \mathbb{R}^n \mid p_i \text{ is false in } \mathbf{x}\}$. One can show that $\texttt{DBM}(b_i \wedge b_j) = \texttt{DBM}(b_i) \cap \texttt{DBM}(b_j)$.

Algorithm 3 shows the steps to generate the abstract states of an MPL system given a set of predicates P. For each $i \in \{1, \ldots, |P|\}$, we manipulate DBMs: the complexity of Algorithm 3 depends on emptiness checking of DBM (line 11), which runs in $\mathcal{O}(n^3)$, where n is the dimension of the state matrix [1]. Therefore, the worst-case complexity of Algorithm 3 is $\mathcal{O}(2^{|P|}n^3)$.

Algorithm 3. Generation of the abstract states from a set of predicates

```
Input: P, a set of predicates                                  ▷ P = P_mat ∪ P_time
Output: Ŝ, a set of abstract states
        D, a partition of ℝ^n w.r.t. Ŝ                          ▷ D is a set of DBMs
 1: procedure pred_abs(P)
 2:   B ← {b_1, ..., b_|P|}                                     ▷ a set of Boolean variables
 3:   D ← {ℝ^n}
 4:   Ŝ ← {true}
 5:   for i ∈ {1, ..., |P|}
 6:     Ŝ ← ⋃_{ŝ∈Ŝ}{ŝ ∧ ¬b_i} ∪ ⋃_{ŝ∈Ŝ}{ŝ ∧ b_i}
 7:     D_neg ← ⋃_{E∈D}{E ∩ DBM(¬b_i)}      ▷ each DBM in D is intersected with DBM(¬b_i)
 8:     D_pos ← ⋃_{E∈D}{E ∩ DBM(b_i)}                  ▷ both D_neg and D_pos are set of DBMs
 9:     D ← D_neg ∪ D_pos
10:     D_temp ← ∅                                              ▷ temporary variable for D
11:     Ŝ_temp ← ∅                                              ▷ temporary variable for Ŝ
12:     for j ∈ {1, ..., |D|}
13:       if D[j] is not empty then                                  ▷ DBM emptiness check
14:         add D[j] to D_temp
15:         add Ŝ[j] to Ŝ_temp
16:       end
17:     end
18:     D ← D_temp
19:     Ŝ ← Ŝ_temp
20:   end
21:   return (Ŝ, D)
22: end
```

3.4 Generation of Abstract Transitions

Having obtained the abstract states, one needs to generate the abstract transitions, which can be obtained via one-step reachability, as described in [1]. Namely, there is a transition from $\hat{s}_i$ to $\hat{s}_j$ if $\mathsf{Im}(\mathtt{DBM}(\hat{s}_i)) \cap \mathtt{DBM}(\hat{s}_j) \neq \emptyset$, where $\mathtt{Im}(\mathtt{DBM}(\hat{s}_i)) = \{A \otimes \mathbf{x} \mid \mathbf{x} \in \mathtt{DBM}(\hat{s}_i)\}$. The computation of $\mathtt{Im}(\mathtt{DBM}(\hat{s}_i))$ corresponds to the image of $\mathtt{DBM}(\hat{s}_i)$ w.r.t. the affine dynamics of $\hat{s}_i$ which has complexity $\mathcal{O}(n^2)$ [29].

However, unlike [29, Algorithm 2], Algorithm 3 does not produce the affine dynamics for each abstract state. For each $\hat{s} \in \hat{S}$, we need to find $\mathbf{g}$ as in (4). One can generate the affine dynamics for $\hat{s} \in \hat{S}$ from the value (either true or false) of $p \in P_{mat}$ on $\hat{s}$. Given a predicate $p \equiv (i, j, c, s)$, we call i and j as the left and right index of p (as $x_i \sim x_j + c$) and denoted them by $\mathtt{left}(p)$ and $\mathtt{right}(p)$, respectively.

If $p \equiv (i, j, A(k,j) - A(k,i), 1)$ is true in $\hat{s}$, we have $x_i + A(k,i) \geq x_j + A(k,j)$, otherwise $x_j + A(k,j) > x_i + A(k,i)$. Hence, the left index of predicates can be used to determine the affine dynamics. Algorithm 4 provides the procedure to find the affine dynamic associated to $\hat{s} \in \hat{S}$.

For each k, $\mathtt{fin}_k$ is computed. Initially, the elements of $\mathtt{fin}_k$ are in strictly increasing order. Then, for each predicate $p \in P_k$, we swap the location of $\mathtt{left}(p)$ and $\mathtt{right}(p)$ whenever p is false on $\hat{s}$. Suppose i is the first element of $\mathtt{fin}_k$ after swapping. One could show that $x_i + A(k,i) \sim x_j + A(k,j)$ for all $j \in \mathtt{fin}_k \setminus \{i\}$.

Algorithm 4. Generation of the affine dynamics for an abstract state

```
Input: A ∈ R_max^{n×n}, a m-regular matrix with m > 1
       ŝ ∈ Ŝ, an abstract state
       P_1, ..., P_n, sets of predicates generated by Algorithm 1
Output: g, the finite coefficient representing the affine dynamics for ŝ
 1: procedure get_affine(A, ŝ, P_1, ..., P_n)
 2:   g ← zeros(1, n)
 3:   for k ∈ {1, ..., n}
 4:     fin_k ← Find(A(k, ·) ≠ ε)          ▷ recall that elements in fin_k is
 5:     for p ∈ P_k                          in strictly-increasing order
 6:       if p is false in ŝ then
 7:         swap left(p) with right(p) in fin_k
 8:       end
 9:     end
10:     g[k] ← fin_k[1]                     ▷ insertion of the k^th element of g
11:   end
12:   return g
13: end
```

3.5 Model Checking MPL Systems over Time-Difference Specifications: Direct Verification

This section discusses the verification of MPL systems over time-difference specifications. First, we define a (concrete) transition system w.r.t. a given MPL system.

Definition 6 (Transition system associated with MPL system). A transition system TS for an MPL system in (1) is a tuple $(S, T, \mathcal{X}, \mathcal{AP}, L)$ where

- the set of states S is $\mathbb{R}^n$,
- $(\mathbf{x}, \mathbf{x}') \in T$ if $\mathbf{x}' = A \otimes \mathbf{x}$,
- $\mathcal{X} \subseteq \mathbb{R}^n$ is a set of initial conditions,
- $\mathcal{AP}$ is a set of time-difference propositions,
- the labelling function $L : S \to 2^{\mathcal{AP}}$ is defined as follows: a state $\mathbf{x} \in S$ is labeled by '$t_i \sim \alpha$' if $[A \otimes \mathbf{x} - \mathbf{x}]_i \sim \alpha$, where $\sim \; \in \{>, \geq, <, \leq\}$. □

We express the time-difference specifications as LTL formulae over a set of time-difference propositions.[3] For instance, $\bigcirc(t_i \leq \alpha)$ represents 'the next time difference for the ith component is $\leq \alpha$' while $\Diamond\Box(t_i \leq \alpha)$ corresponds to 'after some finite executions, the time difference for the ith component is always $\leq \alpha$'. To check the satisfaction of these specifications, we generate the abstract version of MPL system.

The abstract transition system $TS_f = (\hat{S}, T_f, I_f, P_{mat} \cup P_{time}, L_f)$ for an MPL system is generated via predicate abstraction where P_{mat} and P_{time} is the set of predicates generated by Algorithms 1 and 2, respectively. The (abstract) labelling function L_f is defined over predicates $p \in P_{mat} \cup P_{time}$: for $\hat{s} \in \hat{S}$, $p \in L_f(\hat{s})$ iff p is true in $\hat{s}$. We show the relation between predicates in P_{time} and a time-difference proposition in $\mathcal{AP}$.

Proposition 2. Suppose P_{time} is a set of predicates corresponding to a time-difference proposition '$t_i \sim \alpha$' and an abstract state $\hat{s} \in \hat{S}$.

[3] Notice that, in Definition 6 we consider $\mathcal{AP}$ as a set of time-difference propositions.

i. For $\sim \{>, \geq\}$, a (concrete) state $\mathbf{x} \in \mathtt{DBM}(\hat{s})$ is labeled by '$t_i \sim \alpha$' iff at least one predicate in P_{time} is true in $\hat{s}$.
ii. For $\sim \{<, \leq\}$, a (concrete) state $\mathbf{x} \in \mathtt{DBM}(\hat{s})$ is labeled by '$t_i \sim \alpha$' iff all predicates in P_{time} are true in $\hat{s}$. $\square$

Example 2. Suppose we have an MPL system (2) and $\mathcal{AP} = \{t_1 \leq 5\}$. We consider two time-difference specifications $\Diamond(t_1 \leq 5)$ and $\Diamond\Box(t_1 \leq 5)$ and a set of initial conditions $\mathcal{X} = \mathbb{R}^2$. By Algorithms 1 and 2, we have $P_{mat} = \{(1,2,3,1),(1,2,0,1)\}$ and $P_{time} = \{(1,2,0,1)\}$. Thus, $P = \{p_1, p_2\}$ where $p_1 \equiv (1,2,3,1)$ and $p_2 \equiv (1,2,0,1)$.

The resulting abstract transition is depicted in Fig. 1. All abstract states are initial. The corresponding LTL formulae for the time-difference specifications are $\Diamond p_2$ and $\Diamond\Box p_2$.

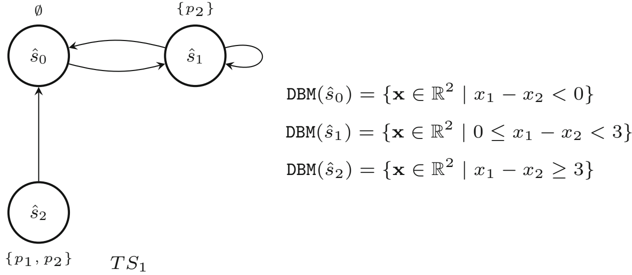

Fig. 1. The abstract transition system via predicate abstractions with a time-difference proposition.

It is clear that $TS_1 \models \Diamond p_2$. Therefore, the underlying MPL system satisfies $\Diamond(t_1 \leq 5)$. However, $TS_1 \not\models \Diamond\Box p_2$ and we can not conclude whether $\Diamond\Box(t_1 \leq 5)$ is false. We will show how to deal with this problem in Sect. 4. $\square$

Direct Verification. In some cases, it is possible to check the satisfaction of time-difference specifications directly, namely without generating the abstraction of the MPL system. We call a time-difference proposition $t_i \sim \alpha$ is a *contradiction* if there is no $\mathbf{x} \in \mathbb{R}^n$ such that $[A \otimes \mathbf{x} - \mathbf{x}]_i \sim \alpha$. On the other hand, $t_i \sim \alpha$ is a *tautology* if all $\mathbf{x} \in \mathbb{R}^n$ satisfy $[A \otimes \mathbf{x} - \mathbf{x}]_i \sim \alpha$.

Proposition 3. Given an MPL system (1) with $A(i,i) = \beta \in \mathbb{R}$.

i. For $\sim \{>, \geq\}$, $t_i \sim \alpha$ is a tautology if $\beta \sim \alpha$.
ii. For $\sim \{<, \leq\}$, $t_i \sim \alpha$ is a contradiction if $\alpha < \beta$. $\square$

The consequence of Proposition 3 is that any time-difference specification defined from a tautology (resp., contradiction) time-difference proposition, is guaranteed to be true (resp., false). For instance, from Example 2, the specification $(t_1 \geq 2)\mathsf{U}(t_2 \geq 3)$ is satisfied, while $\Diamond(t_2 \leq 2)$ is not. As a second instance of

direct verification, in the case of irreducible MPL systems, the dissatisfaction of specifications in the form of $\Diamond\Box(t_i \sim \alpha)$ is related to the max-plus eigenvalue of the corresponding system matrix.

Proposition 4. Consider an MPL system characterised by an irreducible matrix $A \in \mathbb{R}_{\max}^{n\times n}$ and a time-difference specification $\Diamond\Box(t_i \sim \alpha)$. Suppose λ is the max-plus eigenvalue of A. The following holds:

i. For $\sim \{>, \geq\}$, if $\lambda < \alpha$ then $\Diamond\Box(t_i \sim \alpha)$ is false.
ii. For $\sim \{<, \leq\}$, if $\lambda > \alpha$ then $\Diamond\Box(t_i \sim \alpha)$ is false. □

4 Bounded Model Checking of MPL Systems

In this section, we implement bounded model checking (BMC) algorithm to check the satisfaction of time-difference specifications over MPL system. The basic idea of BMC is to find a bounded counterexample of a given length k. If no such counterexample is found, then one increases k by one until a pre-known completeness threshold is reached, or until the problem becomes intractable. The readers are referred to [7–9] for a more detailed description of BMC.

We use NuSMV 2.6.0 [11] via command `check_ltlspec_bmc_onepb` to apply BMC. It performs non-incremental BMC to find a counterexample with length k. If no such bug is present then the command is reapplied for length $k+1$, otherwise we apply spurious checking (cf. Sect. 4.1). In case of non-spurious witness, one can conclude that the time-difference specification is false. Otherwise, we refine the transition system (cf. Sect. 4.2) such that the counterexample is removed and then reapply BMC command for length k. This procedure is repeated until we reach a completeness threshold (cf. Sect. 4.3).

4.1 Checking Spuriousness of Counterexamples

There are two types of k-length bounded abstract counterexamples $\pi = \hat{s}_0\hat{s}_1 \ldots \hat{s}_k$ in BMC: either no-loop or lasso-shaped paths. The former one can be used to express the violation of invariant properties $\Box p$. A lasso-shaped path is $\pi = \hat{s}_0\hat{s}_1 \ldots \hat{s}_k$ such that there exists $1 \leq l \leq k$ where $s_{l-1} = s_k$ [8,9]. Although it is finite, it can represent an infinite path $\overline{\pi} = (\hat{s}_0\hat{s}_1\hat{s}_{l-1})(\hat{s}_l \ldots \hat{s}_k)^\omega$ where $\hat{s}_{l-1+m} = \hat{s}_{k+m}$ for $m \geq 0$. It can be used to represent the counterexample of LTL formulae with eventuality, such as $\Diamond p$ and $\Diamond\Box p$.

From now, we write a lasso-shaped path as $(\pi_{stem})(\pi_{loop})^\omega$, where $\pi_{stem} = \hat{s}_0 \ldots \hat{s}_{l-1}$ and $\pi_{loop} = \hat{s}_l \ldots \hat{s}_k$. To avoid ambiguity, we consider that the length of a lasso-shaped path is equal to $|\pi_{stem}| + |\pi_{loop}|$.[4] Furthermore, any no-loop path cannot be expressed as a lasso-shaped one. That is, if π is a no-loop path then the states in π are all different.

The spuriousness of no-loop paths can be checked via forward-reachability analysis. In detail, $\pi = \hat{s}_0\hat{s}_1 \ldots \hat{s}_k$ is not spurious iff the sequence of DBMs

[4] Notice a loop-back transition from $\hat{s}_k$ to $\hat{s}_l$ in π_{loop}.

$D_1, \ldots, D_{k+1}$ where $D_1 = \mathtt{DBM}(\hat{s}_0)$ and $D_{i+1} = \mathsf{Im}(D_i) \cap \mathtt{DBM}(\hat{s}_i)$ for $1 \le i \le k$, are not empty. Simply put, there exists $\mathbf{x}(0) \in \mathtt{DBM}(\hat{s}_0)$ such that $\mathbf{x}(i+1) = A \otimes \mathbf{x}(i) \in \mathtt{DBM}(\hat{s}_{i+1})$ for $0 \le i \le k$. Algorithm 5 summarises the procedure of spuriousness checking for no-loop paths.

Algorithm 5. Spuriousness checking of no-loop paths

```
Input: π = ŝ0 ŝ1 ... ŝk, a no-loop path with length of k
Output: b, a boolean value                          ▷ b = true iff π is spurious
        D, a set of DBMs
 1: procedure is_spurious(π)
 2:   b ← false
 3:   E ← DBM(π[1])                                 ▷ π[i + 1] = ŝi for 0 ≤ i ≤ k
 4:   D ← {E}                                       ▷ E is the first DBM in D
 5:   k ← |π| − 1
 6:   i ← 1
 7:   while (i ≤ k and b == false)
 8:     E ← Im(E) ∩ DBM(π[i + 1])
 9:     if E is empty then
10:       b ← true
11:     else
12:       add E to D                                ▷ E is now the (i + 1)th DBM in D
13:     end
14:     i ← i + 1
15:   end
16:   return (b, D)
17: end
```

The spuriousness checking for lasso-shaped paths is computed via Algorithm 6. We use periodicity checking to deal with the infinite suffix $(\pi_{loop})^\omega$. In lines 14–22, we check the spuriousness of $(\pi_{stem})(\pi_{loop})^{it}$ where π_{loop} is repeated it times. If it is not spurious then we check the periodicity of the DBM (line 25). We can conclude that $(\pi_{stem})(\pi_{loop})^\omega$ is not spurious if the periodicity is found. In case of an irreducible MPL system, by Proposition 1, the periodicity is no greater than its cyclicity. On the other hand, after 1000 iterations, if the periodicity cannot be found then the algorithm is stopped with an 'undecided' result.

One can see that the spuriousness checking for no-loop paths (Algorithm 5) is guaranteed to be complete. However, this is not the case for Algorithm 6. In the case of irreducible MPL systems, it is complete due to the fact that the periodicity is related to Proposition 1. However for reducible MPL systems, it is incomplete as it may provide undecided results.

Lemma 1 relates the spuriousness of an abstract path, either no-loop or lasso-shaped path, with the value of transient and cyclicity of an irreducible matrix.

Lemma 1. Consider an irreducible $A \in \mathbb{R}_{\max}^{n \times n}$ with transient k_0 and cyclicity c and the resulting abstract transition system $TS_f = (\hat{S}, T_f, I_f, P_{mat} \cup P_{time}, L_f)$. Suppose that π is a path over TS_f. Then,

i. If π is a no-loop path with $|\pi| \ge k_0 + c$, then it is spurious.
ii. If $\pi = (\pi_{stem})(\pi_{loop})^\omega$ with $|\pi_{stem}| + |\pi_{loop}| > k_0 + c$, then it is spurious. □

Algorithm 6. Spuriousness checking of a lasso-shaped path

```
Input: π_stem = ŝ_0 ŝ_1 ... ŝ_{l−1}
       π_loop = ŝ_l ŝ_1 ... ŝ_k
Output: b, a boolean value                                  ▷ b = true iff π is spurious
        D, a set of DBMs
1: procedure is_spurious(π_stem, π_loop)
2:   (b, D) ← is_spurious(π_stem)
3:   if (b == true) then
4:     go to line 35                                        ▷ π_stem is already spurious
5:   else
6:     l ← |π_stem| + 1                                     ▷ the number of states in π_stem
7:     E ← D[l]                                             ▷ E is the last DBM in D
8:     m ← |π_loop|                                         ▷ the number of states in π_loop
9:     it ← 0                                               ▷ the number of iterations
10:    p ← false                                            ▷ boolean value to represent the periodicity
11:    while (it ≤ 1000 and p == false and b == false)      ▷ maximum number of
12:      it ← it + 1                                          iterations is 1000
13:      i ← 1
14:      while (i ≤ m and b == false)
15:        E ← Im(E) ∩ DBM(π_loop[i])
16:        if E is empty then
17:          b ← true
18:        else
19:          add E to D
20:        end
21:        i ← i + 1
22:      end
23:      j, num ← |D|                                       ▷ the number of DBMs in D, notice
24:      while (j − m > l and p == false and b == false)      that mod(|D|, m) = l
25:        if (D[j − m] == E) then
26:          p ← true
27:        end
28:        j ← j − m
29:      end
30:    end
31:    if (it > 1000 and p == false and b == false) then
32:      print 'undecided'
33:    else
34:      return (b, D)
35:    end
36:  end
37: end
```

4.2 Refinement Procedure

Provided that the counterexample is spurious, one needs to refine the abstract transition. Instead of adding new predicates as in CEGAR [12], we are inspired by the refinement procedure described in [2, Sec. 3.3]: for each abstract state $\hat{s}$ with more than one outgoing transitions, it partitions $\mathtt{DBM}(\hat{s})$ according to its successors.

Our approach for the refinement procedure is slightly different. We refine the abstract transition based on a spurious counterexample $\pi = \hat{s}_0 \ldots \hat{s}_k$ using the concept of lazy abstraction [26]. This starts by finding a pivot state, namely a state in which the spuriousness starts. Then, it splits the pivot state using the procedure in [2].

Notice that, from Algorithm 5, the pivot state can be found from the number of DBMs we have in D. One could find that $\hat{s}_{|D|-1}$ is a pivot state. On the other

hand, from Algorithm 6, a pivot state is $\hat{s}_i$ where $i = |D| - 1$, if $|D| < |\pi_{stem}| + 1$ (the spuriousness is found in π_{stem}), otherwise $i = |\pi_{stem}| + 1 + \texttt{mod}(D - |\pi_{stem}| - 1, |\pi_{loop}|)$.

With regards to the refined abstract transitions, the labels and affine dynamics for the new abstract states are equal to those of the pivot state. Furthermore, the outgoing (resp. ingoing) transitions from (resp. to) new abstract states are determined similarly using one-step reachability.

Example 3. We use abstract transition in Fig. 1 with specification $\Diamond\Box p_2$. The NuSMV model checker reports a counterexample of length 2: $\pi = \hat{s}_1(\hat{s}_0\hat{s}_1)^\omega$. By Algorithm 6, it is spurious and the pivot state is $\hat{s}_1$. The resulting post-refinement abstract transition is depicted in Fig. 2. $\square$

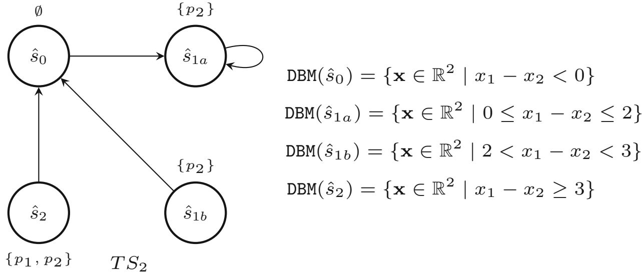

Fig. 2. The refinement of the abstract transition in Fig. 1. The abstract state $\hat{s}_1$ is split into $\hat{s}_{1a}, \hat{s}_{1b}$.

4.3 Upper-Bound on the Completeness Threshold

Given a transition system TS and a specification φ, a completeness threshold is a bound k such that, if no counterexample of φ with length k or less can be found in TS, then φ is satisfied by TS [8,9].

We recall from above that for specific formulae, the completeness threshold is related to the structure of the underlying transition system. For instance, the CT for safety properties of the form $\Box p$ is equal to the *diameter* of transition system: the length of longest shortest distance between two states [7]. Likewise, the CT for liveness specifications in the form of $\Diamond p$ is given by the *recurrent diameter* (the length of loop-free path) [14]. Computing the completeness threshold for general LTL formulae is still an open problem [14].

We show that the CT for (abstract) transition system that generated from an irreducible MPL system is related to the transient and cyclicity of the corresponding matrix.

Lemma 2. Consider an irreducible $A \in \mathbb{R}^{n\times n}_{\max}$ with transient k_0 and cyclicity c and the resulting abstract transition system $TS_f = (\hat{S}, T_f, I_f, P_{mat} \cup P_{time}, L_f)$. The CT for TS_f and for any LTL formula φ over $P_{mat} \cup P_{time}$ is bounded by $k_0 + c$. $\square$

Lemma 2 ensures that the CT is not greater than the sum of the transient and cyclicity of the MPL systems. Looking back to the transition system in Fig. 1, the completeness threshold for $\Diamond p_2$ is 2. In comparison, the transient and cyclicity of matrix in (2) are $k_0 = c = 2$.

By Lemma 2, one could say that the BMC algorithm for irreducible MPL systems is complete for any LTL formula. However, this is not the case for reducible MPL systems, due to the incompleteness of Algorithm 6.

5 Computational Benchmarks

We compare the run-time of the predicate abstractions in this paper with related abstraction procedures in [29], which use max-plus algebraic operations ("tropical abstractions") and are enhanced versions of the earlier work in [1]. For increasing n, we generate matrices $A \in \mathbb{R}_{\max}^{n \times n}$ with two finite elements in each row, each with values ranging between 1 and 10. Location and value of the finite elements are chosen randomly. The computational benchmark has been implemented on an Intel(R) Xeon(R) CPU E5-1660 v3, 16 cores, 3.0 GHz each, and 16 GB of RAM.

We run the experiments for both procedures using C++. Over 10 independents experiments for each dimension, Table 1 shows the running time to generate (specification-free) abstractions of MPL systems, where entry represents the average and maximal values. We do not compare the running time for the generation of abstract transitions because both methods apply the same algorithm.

Table 1. Average and maximal running times of abstraction procedures

n	Tropical abstractions from [29]	Predicate abstractions (this work)
3	$\{0.15, 0.21\}$ [ms]	$\{0.27, 0.38\}$ [ms]
4	$\{0.26, 0.35\}$ [ms]	$\{0.49, 0.72\}$ [ms]
5	$\{0.41, 0.44\}$ [ms]	$\{0.79, 0.88\}$ [ms]
6	$\{1.12, 1.20\}$ [ms]	$\{1.92, 2.10\}$ [ms]
7	$\{2.68, 3.74\}$ [ms]	$\{3.19, 4.60\}$ [ms]
8	$\{8.78, 10.02\}$ [ms]	$\{9.13, 13.74\}$ [ms]
9	$\{32.12, 36.66\}$ [ms]	$\{30.38, 42.02\}$ [ms]
10	$\{0.12, 0.14\}$ [s]	$\{0.11, 0.17\}$ [s]
11	$\{0.57, 0.66\}$ [s]	$\{0.54, 0.81\}$ [s]
12	$\{3.82, 4.67\}$ [s]	$\{2.58, 4.19\}$ [s]
13	$\{23.71, 28.28\}$ [s]	$\{15.80, 28.52\}$ [s]
14	$\{1.39, 1.59\}$ [min]	$\{0.89, 1.27\}$ [min]
15	$\{27.73, 31.06\}$ [min]	$\{4.68, 8.40\}$ [min]

As we can see in Table 1, for large dimensions (beyond 8), the average running time of predicate abstractions is faster than that of tropical abstractions. We recall that the (specification-free) predicate abstractions of MPL systems are computed by Algorithms 1, 2 and 4. Whereas for tropical abstractions, they are computed by [29, Algorithm 2].

We also provide a comparison over values of CT. NuSMV is able to compute CT via an incremental BMC command `check_ltlspec_sbmc_inc -c`. For each bound k, in addition to counterexample searching, it generates a SAT (i.e. boolean satisfiability) problem to verify whether the LTL formula can be concluded to hold. This method of computation of completeness check can be found in [25,28].

Table 2 shows the comparison of the CT values specified by Lemma 2 and those computed by NuSMV. For dimension of $n \in \{3, 4, 5\}$, we generate 20 random irreducible matrices $A \in \mathbb{R}_{\max}^{n \times n}$ with two finite elements in each row. We use the same time-difference specification $\Diamond\Box(t_1 \leq 10)$ for all experiments.

Table 2. The comparison of completeness thresholds.

n	#stf	#($\mathtt{ct}_1 < \mathtt{ct}_2$)	#($\mathtt{ct}_1 = \mathtt{ct}_2$)	#($\mathtt{ct}_1 > \mathtt{ct}_2$)
3	14	0	1	13
4	15	1	0	14
5	14	0	0	14

The 2$^{\text{nd}}$ column of Table 2 represents the number of experiments whose the specification $\Diamond\Box(t_1 \leq 10)$ is satisfied. The last three columns describe the comparison of CT. We use $\mathtt{ct}_1$ and $\mathtt{ct}_2$ to respectively denote the CT that computed by NuSMV and Lemma 2. As we can see, the CT upper bounds specified by Lemma 2 are relatively smaller than those computed by NuSMV.

6 Conclusions

This paper has introduced a new technique to generate the abstractions of MPL systems via a set of predicates. The predicates are chosen automatically from system matrix and the time-difference specifications under consideration. Having obtained the abstract states and transition, this paper has implemented bounded model checking to check the satisfaction of time-difference specifications.

The abstraction performance has been tested on a numerical benchmark, which has displayed an improvement over existing procedures. The comparison for completeness thresholds suggests that the cyclicity and transient of MPL systems can be used as an upper bound. Yet, this bound is relatively smaller than the CT bounds computed by NuSMV.

Acknowledgements. The first author is supported by Indonesia Endowment Fund for Education (LPDP), while the third acknowledges the support of the Alan Turing Institute, London, UK.

References

1. Adzkiya, D., De Schutter, B., Abate, A.: Finite abstractions of max-plus-linear systems. IEEE Trans. Autom. Control. **58**(12), 3039–3053 (2013). https://doi.org/10.1109/TAC.2013.2273299
2. Adzkiya, D., Zhang, Y., Abate, A.: VeriSiMPL 2: an open-sourcesoftware for the verification of max-plus-linear systems. Discrete Event Dyn. Syst. **26**(1), 109–145 (2016). https://doi.org/10.1007/s10626-015-0218-x
3. Alur, R., Dang, T., Ivančić, F.: Progress on reachability analysis of hybrid systems using predicate abstraction. In: Maler, O., Pnueli, A. (eds.) HSCC 2003. LNCS, vol. 2623, pp. 4–19. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36580-X_4
4. Baccelli, F., Cohen, G., Olsder, G.J., Quadrat, J.P.: Synchronization and Linearity: An Algebra for Discrete Event Systems. Wiley, Chichester (1992)
5. Baier, C., Katoen, J.P.: Principles of Model Checking. MIT Press, Cambridge (2008)
6. Ball, T., Majumdar, R., Millstein, T., Rajamani, S.K.: Automatic predicate abstraction of C programs. In: Proceedings of Programming Language Design and Implementation 2001 (PLDI 2001), vol. 36, pp. 203–213. ACM (2001). https://doi.org/10.1145/381694.378846
7. Biere, A., Cimatti, A., Clarke, E., Zhu, Y.: Symbolic model checking without BDDs. In: Cleaveland, W.R. (ed.) TACAS 1999. LNCS, vol. 1579, pp. 193–207. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-49059-0_14
8. Biere, A., Cimatti, A., Clarke, E.M., Strichman, O., Zhu, Y., et al.: Bounded model checking. Adv. Comput. **58**(11), 117–148 (2003)
9. Biere, A., Heljanko, K., Junttila, T., Latvala, T., Schuppan, V.: Linear encodings of bounded LTL model checking. Log. Methods Comput. Sci. **2**(5), 1–64 (2006). https://doi.org/10.2168/LMCS-2(5:5)2006
10. Brackley, C.A., Broomhead, D.S., Romano, M.C., Thiel, M.: A Max-plus model of ribosome dynamics during mRNA translation. J. Theor. Biol. **303**, 128–140 (2012). https://doi.org/10.1016/j.jtbi.2012.03.007
11. Cimatti, A., et al.: NuSMV 2: an opensource tool for symbolic model checking. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 359–364. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45657-0_29
12. Clarke, E., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 154–169. Springer, Heidelberg (2000). https://doi.org/10.1007/10722167_15
13. Clarke, E., Grumberg, O., Talupur, M., Wang, D.: Making predicate abstraction efficient. In: Hunt, W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 126–140. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45069-6_14
14. Clarke, E., Kroening, D., Ouaknine, J., Strichman, O.: Completeness and complexity of bounded model checking. In: Steffen, B., Levi, G. (eds.) VMCAI 2004. LNCS, vol. 2937, pp. 85–96. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24622-0_9

15. Clarke, E., Kroening, D., Sharygina, N., Yorav, K.: Predicate abstraction of ANSI-C programs using SAT. Form. Methods Syst. Des. **25**(2–3), 105–127 (2004). https://doi.org/10.1023/B:FORM.0000040025.89719.f3
16. Clarke, E., Talupur, M., Veith, H., Wang, D.: SAT based predicate abstraction for hardware verification. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 78–92. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24605-3_7
17. Clarke, E.M., Grumberg, O., Long, D.E.: Model checking and abstraction. ACM Trans. Program. Lang. Syst. (TOPLAS) **16**(5), 1512–1542 (1994). https://doi.org/10.1145/186025.186051
18. Comet, J.P.: Application of max-plus algebra to biological sequence comparisons. Theor. Comput. Sci. **293**(1), 189–217 (2003). https://doi.org/10.1016/S0304-3975(02)00237-2
19. Das, S., Dill, D.L., Park, S.: Experience with predicate abstraction. In: Halbwachs, N., Peled, D. (eds.) CAV 1999. LNCS, vol. 1633, pp. 160–171. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48683-6_16
20. De Schutter, B.: On the ultimate behavior of the sequence of consecutive powers of a matrix in the max-plus algebra. Linear Algebra Its Appl. **307**(1–3), 103–117 (2000). https://doi.org/10.1016/S0024-3795(00)00013-6
21. Flanagan, C., Qadeer, S.: Predicate abstraction for software verification. In: Proceedings of the 29th Principles of Programming Languages (POPL 2002), vol. 37, pp. 191–202. ACM (2002). https://doi.org/10.1145/503272.503291
22. Graf, S., Saidi, H.: Construction of abstract state graphs with PVS. In: Grumberg, O. (ed.) CAV 1997. LNCS, vol. 1254, pp. 72–83. Springer, Heidelberg (1997). https://doi.org/10.1007/3-540-63166-6_10
23. Heemels, W., De Schutter, B., Bemporad, A.: Equivalence of hybrid dynamical models. Automatica **37**(7), 1085–1091 (2001). https://doi.org/10.1016/S0005-1098(01)00059-0
24. Heidergott, B., Olsder, G.J., Van der Woude, J.: Max Plus at Work: Modeling and Analysis of Synchronized Systems: A Course on Max-Plus Algebra and Its Applications. Princeton University Press, Princeton (2014)
25. Heljanko, K., Junttila, T., Latvala, T.: Incremental and complete bounded model checking for full PLTL. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, pp. 98–111. Springer, Heidelberg (2005). https://doi.org/10.1007/11513988_10
26. Henzinger, T.A., Jhala, R., Majumdar, R., Sutre, G.: Lazy abstraction. In: Proceedings of the ACM Symposium on Principles of Programming Languages (POPL 2002), pp. 58–70 (2002). https://doi.org/10.1145/503272.503279
27. Imaev, A., Judd, R.P.: Hierarchial modeling of manufacturing systems using max-plus algebra. In: Proceedings of American Control Conference 2008, pp. 471–476 (2008)
28. Latvala, T., Biere, A., Heljanko, K., Junttila, T.: Simple is better: efficient bounded model checking for past LTL. In: Cousot, R. (ed.) VMCAI 2005. LNCS, vol. 3385, pp. 380–395. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-30579-8_25
29. Mufid, M.S., Adzkiya, D., Abate, A.: Tropical abstractions of max-plus linear systems. In: Jansen, D.N., Prabhakar, P. (eds.) FORMATS 2018. LNCS, vol. 11022, pp. 271–287. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00151-3_16
30. Mufid, M.S., Adzkiya, D., Abate, A.: Bounded model checking of max-plus linear systems via predicate abstractions. arXiv e-prints arXiv:1907.03564, July 2019