Abstractions and Formal Verification of Max-Plus Linear Systems

Muhammad Syifa'ul Mufid
University of Oxford
Linacre College

This thesis is submitted to the Computer Science Department of the University of Oxford, in partial fulfilment of the requirements for the degree of Doctor of Philosophy

Michaelmas 2021

To my fathers who never saw this adventure and
To my sons who joined this adventure

Acknowledgements

All praise is due to the Almighty God, the Most Gracious and the Most Merciful. Despite many shortcomings, He always bestowed His blessings upon me. I hope and pray that His mercy continuous upon me and my family unabatedly and that we remain only His servant till the end of our life. Verily, He is the source of all strength.

I would like to acknowledge and give my warmest thank to my supervisor Prof. Alessandro Abate who made this work possible. His guidance helped me in all the time of research and writing of this thesis. Besides my supervisor, I would like to thank the examiners of the D.Phil Viva, Prof. Christoph Haase and Prof. Laurent Hardouin, for their encouragement, insightful comments and questions.

I would like to thank all people in Oxford Control and Verification (OXCAV) research group. Often, I got the inspiration for my research from the group's members. Thanks also to the supporting staff at CS Oxford. With their help, I can focus on research without worrying about paperwork, especially for conferences' registration.

Thanks to the Indonesian community in Oxford for creating such a wonderful experience in the past four years. Special thanks to the Indonesian families who lived at Castle Mill for helping us during critical times and inviting us to dinners and vacations.

I would like to thank all colleagues in the Mathematics department, Institut Teknologi Sepuluh Nopember, Surabaya, Indonesia, for encouraging me to take a DPhil at University of Oxford. Also, for Lembaga Pengelola Dana Pendidikan (LPDP), thank you for the scholarship and other expenses. As promised, I returned to Indonesia after finishing my DPhil.

Finally, I would like to express my sincerest thank to my family. The prayers from my parents and my parents in law have been the best help that I will ever need. My warmest thank goes to Adiba, Mizan and Izman. Adiba, thanks for being supportive, sharing every moment of happiness and sadness, and keeping up with me. Mizan and Izman, thanks for your presence in our life. Lastly, I would like to thank all people who helped my family and me directly and indirectly during these years. I hope that all of you have a fruitful life. I wish you all the best.

Muhammad Syifa'ul Mufid

Abstract

Max-Plus Linear (MPL) systems are the class of discrete-event systems (DES) with dynamics based on two binary operations (maximisation and addition) over the so-called max-plus semiring. In practical applications, MPL systems are used to model synchronisation phenomena without concurrency. Such are widely used in railway networks, manufacturing plants, and modelling and studying biological systems. The dual of MPL systems is Min-Plus Linear (MiPL) systems which use minimisation and addition operations. Furthermore, as a natural extension, Interval Max-Plus Linear (IMPL) systems are MPL systems where real-valued intervals characterise the delays between successive discrete events. In general, IMPL systems are more realistic than simple MPL ones. For instance, in a model of a railway network, the travel between two stations may take longer (or possibly faster) than the expected time due to unforeseen external factors such as weather conditions, driver's behaviour, and the number of departing passengers at stations. The fundamental problems for the systems mentioned above are reachability analysis and formal verification, respectively assessing whether the dynamics of the underlying system eventually reaches a particular set and satisfies the intended specifications or requirements. The state-of-the-art approaches to tackle these problems employ the abstraction technique by leveraging the translation of an MPL (including MiPL and IMPL) system into an equivalent Piecewise Affine (PWA) system and using Difference-Bound Matrix (DBM) to express the resulting abstract states as well as the set of atomic propositions. Such an abstraction technique results in an abstract transition system that allows replacing the verification of specifications over the original model. However, the existing techniques are not complete since they cannot determine whether the counterexample found on the abstract transition system is spurious. The other disadvantages are related to the scalability of the abstraction procedure and the limited expressiveness of DBM when dealing with specifications that contain conjunction or negation. This work addresses these issues by proposing novel abstraction procedures by identifying the propositions from the underlying model and specifications before generating the abstract states. These novel approaches allow the verification of more expressive specifications in Linear Temporal Logic (LTL). In addition to this, using the Bounded Model Checking (BMC) algorithm combined with spuriousness checking (of counterexample) and refinement procedures, we present the complete verification framework where the corresponding completeness threshold is related to the periodic behaviour of the model. Such behaviour is determined by a pair of transient (i.e., the starting index of periodic behaviour) and cyclicity (also called periodicity). Despite these developments, the proposed abstraction-based procedures remain not scalable. We then propose alternative methods utilising Satisfiability Modulo Theory (SMT). The main idea

underpinning the SMT-based methods is to transform the dynamics of the model and possibly the specifications under consideration into a formula in quantifier-free Real Difference Logic (QF-RDL) or Linear Real Arithmetics (QF-LRA). The satisfaction of the resulting formula, which can be checked by SMT solvers, corresponds to the output for reachability and verification problems. Finally, the performance of the proposed methods (abstraction-based and SMT-based) is evaluated via a set of computational benchmarks, where we observe a significant improvement for the SMT-based procedures w.r.t. the scalability, compared to existing approaches whenever available.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

This thesis discusses the abstractions and formal verification of Max-Plus Linear (MPL) systems. This chapter introduces MPL systems and their classical problems, followed by the description of reachability analysis and formal verification. First, we mention the existing procedures for those problems and discuss their drawbacks. Then we propose novel approaches to tackle these problems, which will be further elaborated throughout this thesis. Finally, the description of the organisation of this thesis concludes this chapter.

1.1 Motivation

Max-plus algebra is an algebraic structure which employs maximisation and addition operations, respectively. Traditionally, it is denoted as $(\mathbb{R}_{\max}, \oplus, \otimes)$ where $\mathbb{R}_{\max} = \mathbb{R} \cup \{\varepsilon = -\infty\}$,

$$a \oplus b = \max\{a, b\} \text{ and } a \otimes b = a + b,$$

for all $a, b \in \mathbb{R}_{\max}$. Together with these binary operations, $(\mathbb{R}_{\max}, \oplus, \otimes)$ is a semiring with ε and 0 as the null and unit elements, respectively [10, 64].

The class of discrete-event systems (DES) based on max-plus algebra are called Max-Plus Linear (MPL) systems [10, 54]. In these systems, the underlying state-space represents the timing of discrete-time events synchronised over max-plus algebra. It means that the next event occurs right after the last of the previous events have finished. The applications of MPL systems are significantly found on models where time variable is essential such as in transportation networks [64], in scheduling [8] or manufacturing [67] problems, or for biological systems [22, 44].

In addition to those applications, there are some connections between max-plus algebra and theoretical computer science. Some of the prominent examples are max-plus automata [20, 49]. A max-plus automaton is defined as a weighted automaton over a semiring $\mathbb{Z}_{\max} = \{\mathbb{Z} \cup \{\varepsilon\}, \oplus, \otimes\}$ with finite alphabets Σ. Furthermore, there are matrix representations for the initial and accepting states and all labels $w \in \Sigma$. These matrices are used as tools to derive algebraic proofs of automata results [49]. Recently, the max-plus based neural network is introduced in [94] by employing $\oplus$ and $\otimes$ (instead of $+$ and $\times$) for the activation function of the perceptron.

Over the past three decades, several fundamental problems for MPL systems and max-plus algebra have been explored such as eigenproblems [46, 89], optimization [21], definite form [84], periodicity [52, 72, 80]. The classical approach for those problems is developed based on their algebraic features [10] as well as the precedence graph representation of the state matrix (cf. Definition 2.3). For instance, from the precedence graph, one can define the notion of *reducibility*: an MPL system is called *irreducible* if the precedence graph is strongly connected, otherwise, it is *reducible* [10]. This notion plays an important role since it is directly related to the periodic behaviour: the trajectories of discrete events generated by irreducible MPL systems are eventually periodic, in max-plus algebraic manner, from an index called *transient* with a specific periodicity (also called as *cyclicity*) [10].

The reachability analysis and formal verification problems have been recently discussed in [4–6]. The former problem determines whether a specific target set of states of an MPL system is attainable from a given set of initial states, while the latter investigates whether the underlying MPL system satisfies the given specifications. However, unlike the mentioned ones in the preceding paragraph, these two problems cannot be solved solely from the algebraic analysis of the state matrix. Furthermore, the continuous state-space of MPL systems makes it hard to simulate the trajectories of MPL systems at each time horizon.

The state-of-the-art methods for formal verification and reachability analysis employ the finite abstractions of MPL systems. Generally speaking, abstraction [11,37,43,48,62] is a technique to generate abstract versions of large or even infinite models. Such a procedure results in less complex abstract models, which allow replacing the verification of specifications over the original or concrete ones with automated and scalable techniques. Normally, those specifications are expressed as Linear Temporal Logic (LTL) formulae [11]. If a specification is satisfied over the abstract models, then it also holds for concrete ones [11]. However, the invalidity of the specification on abstract models does not necessarily imply the same conclusion for the concrete models.

In the case of MPL systems, the abstraction procedure is implemented by generating an equivalent Piece-Wise Affine (PWA) system [63], characterised by several domains (partitions, or PWA regions) and the corresponding affine dynamics. The PWA regions are represented by Difference-Bound Matrices (DBM) [55, 73], which correspond to the abstract states of MPL systems. Thus, to simulate the trajectories of MPL systems, one needs to generate the transition relations among abstract states by computing the image of DBM (or the inverse image of DBM) with the respective affine dynamics. The image computation also allows one to solve reachability analysis of MPL systems by using a forward approach, i.e., recursively computing the image of the initial set. Similarly, the backward approach is implemented by successively computing the inverse image of the target set.

While the approaches in [2, 4–7] are much scalable beyond existing methods based on max-plus algebraic operations, they admit several disadvantages. First, the abstraction procedures in [2,4] can be applied only for MPL systems with relatively small dimensions. Second, the verification procedure is not complete, especially if the specification is false on the abstract version of the MPL system. Indeed, one may generate the refined abstraction such that it has a bisimulation equivalence

relation with the original MPL system [4, 7]. However, the refinement procedure is not necessarily terminated. Third, using DBM as data structures makes it difficult to verify specifications with negation or disjunction. We recall that, in general, the complement of DBM and the union of DBM cannot be expressed as a single DBM. Fourth, as strange as it may seem, the specifications for an MPL system can only be chosen after generating the abstraction.

We propose some approaches to address these shortcomings. First, we develop novel abstraction procedures that are more scalable than those in [2, 4]. One particular procedure uses a set of predicates to generate the abstract version of MPL systems. Those predicates are identified from the underlying MPL system and possibly from the specifications under consideration. This technique is well-known as predicate abstraction [62], and is widely used for verification of software and hardware [42, 59] and programs [12, 40, 41].

Second, we implement the Bounded Model Checking (BMC) technique [17–19] to check the satisfaction of the desired specifications. The basic idea of BMC is finding a bounded counterexample with a given length of k. If no such counterexample presents, then one increases k by one until a pre-known *completeness threshold* is reached, or until the problem becomes intractable. Since we are working on abstractions, each counterexample generated by the BMC procedure needs to be checked for its spuriousness. If a spurious counterexample is found, we refine the abstract version of the MPL system using the procedure in [7], combined with lazy abstraction [66]. We prove that, for *boundedly periodic* (cf. Definition 2.8) MPL systems, this technique is complete, where the completeness threshold is related to the pair of transient and cyclicity.

Third, we use Real Difference Logic (RDL) [15, 45] to encode the trajectories of MPL systems and the specifications. By using RDL, the mentioned disadvantages of DBM no longer occur. Furthermore, employing a Satisfiability Modulo Theory (SMT) solver, we propose the idea of solving reachability analysis and formal verification of MPL systems symbolically. An SMT problem deals with the satisfaction of a logical formula w.r.t. a given theory (e.g., linear arithmetics, or bit-vectors) [15]. It should be noted that this technique is not using abstraction as it verifies the specifications directly on the original MPL system. Moreover, based on the computational benchmarks presented in Chapters 5-6, the SMT-based procedure is much more scalable than the existing abstraction-based method.

Finally, we formally define the logic of Time-Difference LTL (TDLTL) to express the specifications over MPL systems. The atomic formulae for TDLTL are time-difference propositions that correlate the delays on the discrete-time events of MPL systems. Each proposition can be chosen independently from the abstract representation of MPL systems. Furthermore, it can be equivalently expressed as DBM or RDL, respectively, allowing to solve verification problems over MPL systems via abstraction- or SMT-based technique.

To show the effectiveness of the developed theories and algorithms, we extend our resulting techniques for other related models, namely Min-Plus Linear (MiPL) systems and Interval Max-Plus Linear (IMPL) systems. The former can be considered as the dual of MPL systems because they use minimisation instead of maximisation. On the other hand, IMPL systems are the generalisation of MPL systems in

the sense that the finite elements of the state matrix are real-valued intervals. While most methods for MPL systems can be applied directly to MiPL systems due to the duality between maximisation and minimisation, only a few approaches have been developed in the literature to study the behaviour of IMPL systems; for example, eigenproblems [29,61], solving the linear equations over interval matrices [78,79] and reachability analysis [27].

1.2 Contributions

This research aims to develop a novel and general framework for the abstractions and formal verification of MPL systems. In the process, we also obtained several results related to the periodic behaviour of MPL systems, which become the foundation for the formal verification and reachability analysis procedures. The resulting outcomes are then applied to similar models such as MiPL and IMPL systems. The details of the contributions are described as follows.

Abstractions of MPL systems. We propose novel abstraction techniques that are more scalable and efficient than those in [4]. First, we develop the notion of *tropical abstraction* of MPL systems that uses the max-plus algebraic operations. Second, we apply *predicate abstraction* to generate the abstractions of MPL systems. Both methods leverage the translation of an MPL system into the equivalent PWA system. Furthermore, they outperform the existing abstraction procedure in [4]. We also show that the proposed abstraction procedures can be applied for MiPL and IMPL systems.

Computing transient of MPL systems via SMT-solving. While the transient of irreducible MPL systems can be computed using spectral analysis of the corresponding state matrix, we propose an alternative method by transforming the problem into SMT instances. In general, this SMT-based procedure is less efficient than the existing approach. However, it has advantages for two cases: 1) to compute transient of MPL systems with any set of initial conditions and 2) to classify or synthesise the state-space of MPL systems w.r.t. a specific pair of transient and cyclicity. To the best of the author's knowledge, there are no direct approaches for solving the latter case. The extension of the procedures to MiPL systems is also discussed.

Reachability analysis of MPL Systems. We develop an SMT-based procedure to solve the reachability analysis of MPL systems. It encodes the MPL system and the initial and targets sets into a formula in QF-RDL and then checks the satisfaction of the resulting formula via an SMT solver. We show that this procedure is complete for any MPL system as long as the transient w.r.t. a set of initial conditions exists. As shown in the computational benchmarks, the proposed algorithms clearly outperform the existing procedure that generates the abstraction of MPL systems and explicitly computes the reach sets for each step bound. The extension of the procedures (abstraction- and SMT-based) to MiPL and IMPL systems are then thoroughly discussed.

Bounded model checking of MPL systems. We designed two novel procedures to verify the satisfaction of time-difference specifications, grounded on the Bounded Model Checking (BMC) algorithm. The first one employs the abstractions

of MPL systems, while the second technique translates the trajectories of MPL systems and the time-difference specifications (which are later written as LTL formulae) into a QF-RDL formula. Again, both procedures are complete if the transient of MPL system w.r.t. initial conditions exists. We also show that the resulting procedures can be applied to MiPL and IMPL systems.

1.3 Overview of the Thesis

This thesis discusses the reachability analysis and formal verification of MPL, MiPL and IMPL systems. We present the abstraction- and SMT-based algorithms to solve the problems. Additionally, for IMPL systems, we present an SMT-based procedure to solve quantified reachability analysis by allowing quantifiers over the set of initial conditions and state matrices. The thesis is organised as follows and is depicted in Figure 1.1:

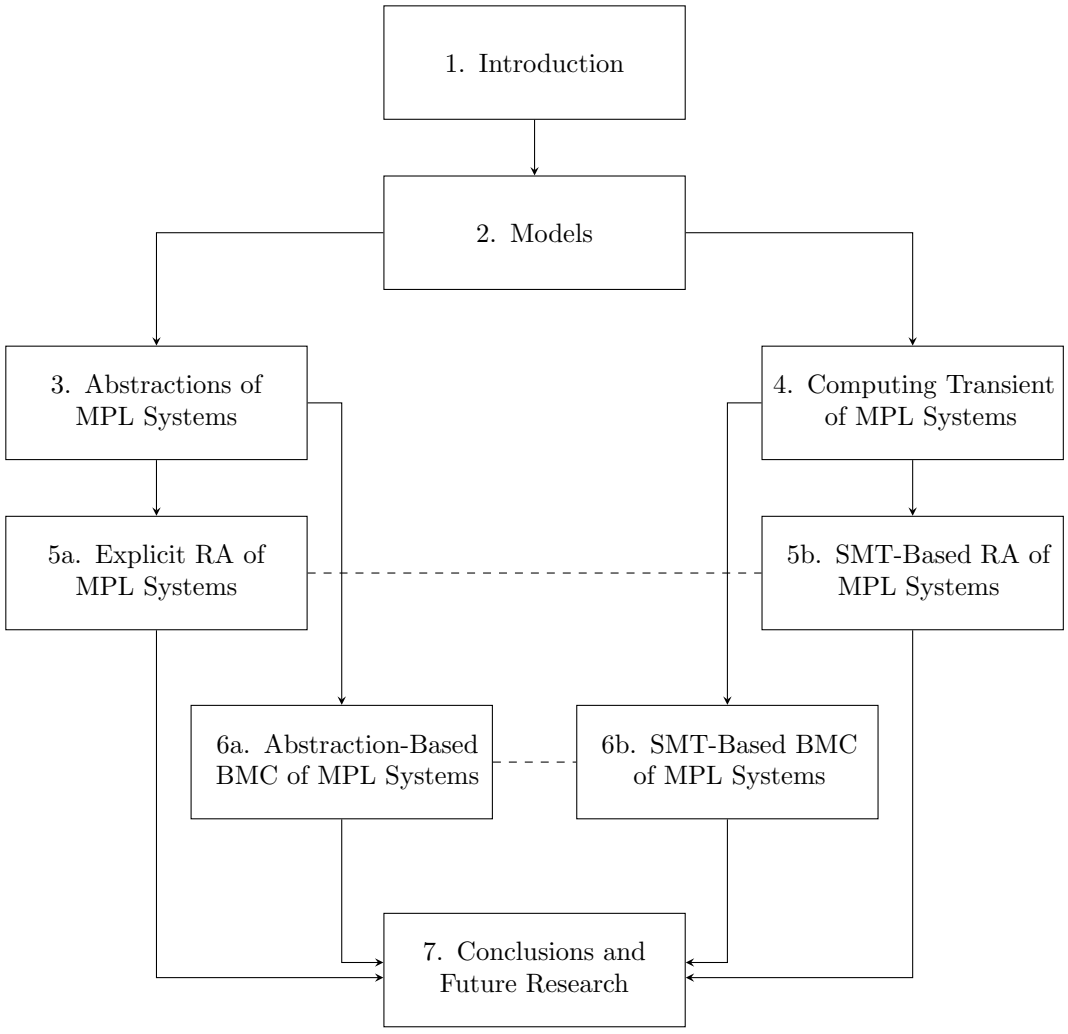


Figure 1.1: The structure of the thesis. Pointed arrows indicate relations of inter-dependence while dashed lines represent independence within the same chapter.

- **Chapter 2** introduces the basic definition of MPL systems and recalls some algebraic properties. Then, some related models are briefly discussed, such as Piece-wise Affine (PWA) systems, Min-Plus Linear (MiPL) and Interval Max-Plus Linear (IMPL) systems. Finally, we describe Difference Bound Matrices (DBM), which are used as the data structure for the procedures in Chapter 3.
- **Chapter 3** describes the novel procedures to generate the abstract version of MPL systems: tropical and predicate abstractions of MPL systems. First of all, we recall the state-of-the-art methods for finite-abstraction of MPL systems in [4] and then present some preliminary concepts such as transition systems and abstractions.
- SMT-based techniques to compute the transient bound of MPL systems are discussed in **Chapter 4**. The resulting bound plays an essential role in the SMT-based procedures in the following two chapters.
- **Chapter 5** describes the abstraction- and SMT-based approaches to solve reachability analysis of MPL systems. We cover both forward and backward reachability analysis.
- We develop abstraction- and SMT-based procedures for formal verification of MPL systems in **Chapter 6**. The properties of interest for this study are time-difference specifications.
- **Chapter 7** summarises the results of the thesis and outlines some directions for future research.

1.4 Publications by the Author

Most of the results presented in Chapters 3-6 of this thesis have appeared in international conference proceedings or published in peer-reviewed journals. The following are the list of publications:

- ***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), pages 271–287. Springer, 2018.*** [73]

  The first author was responsible for idea development, algorithm implementation and experiments. The first two authors contribute equally in manuscript writing and proofs, and they received valuable guidance from the third author.

- ***M. S. Mufid, D. Adzkiya, and A. Abate. "Bounded model checking of max-plus linear systems via predicate abstractions". In International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS), pages 142–159. Springer, 2019.*** [74]

  The first author was responsible for idea development, algorithm implementation and experiments. The first two authors contribute equally in manuscript writing and proofs, and they received valuable guidance from the third author.

- ***A. Abate, A. Cimatti, A. Micheli, and M. S. Mufid. Computation of the transient in max-plus linear systems via SMT-solving. In International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS), pages 161–177. Springer, 2020.*** [1]

  The order of authors is purely alphabetical. The last author was responsible for idea development. The third and fourth authors contributed equally in manuscript writing and algorithm implementation, and they received valuable guidance from the first two authors.

- ***M. S. Mufid, D. Adzkiya, and A. Abate. Symbolic reachability analysis of high dimensional max-plus linear systems. IFAC-PapersOnLine, 53(4):459-465, 2020. 15th IFAC Workshop on Discrete Event Systems (WODES) - Rio de Janeiro, 2020*** [75]

  The first author was responsible for idea development, algorithm implementation, manuscript writing and proofs, and he received valuable guidance from the co-authors.

- ***M. S. Mufid, D. Adzkiya, and A. Abate. SMT-based reachability analysis of high dimensional interval max-plus linear systems. IEEE Transactions on Automatic Control (TAC), 2021.*** [76]

  The first author was responsible for idea development, algorithm implementation, manuscript writing and proofs, and he received valuable guidance from the co-authors.

- ***M. S. Mufid, A. Micheli, A. Abate, and A. Cimatti. SMT-Based Model Checking of Max-Plus Linear Systems. In S. Haddad and D. Varacca, editors, 32nd International Conference on Concurrency Theory (CONCUR), volume 203 of Leibniz International Proceedings in Informatics (LIPIcs), pages 22:1–22:20, Dagstuhl, 2021.*** [77]

  The first author was responsible for idea development. The first and second authors contributed equally in manuscript writing and algorithm implementation, and they received valuable guidance from the co-authors.

The overlap between the content of each chapter and the published materials is as follows:

- Chapter 3
  The materials, including the proofs and algorithms, presented in Section 3.3 (resp. Section 3.4) are based on [73] (resp. [74]).

- Chapter 4
  All materials in Section 4.2 are based on [1].

- Chapter 5
  The materials, including the proofs and algorithms, presented in Section 5.2 (resp. Section 3.4) are based on [75, 76].

- Chapter 6
  The materials, including the proofs and algorithms, presented in Section 6.3 (resp. Section 6.4) are based on [74] (resp. [77]).

Chapter 2

Models

In this chapter, we describe a brief overview of Max-Plus Linear (MPL) systems and some related models such as Min-Plus Linear (MiPL) systems, Interval Max-Plus Linear (IMPL) systems, and Piece-wise Affine (PWA) systems. We also present the notion of Difference-Bound Matrix (DBM) and the procedures to compute the image and inverse image of a DBM w.r.t. the dynamics of the systems mentioned above.

2.1 Max-Plus Linear Systems

This section introduces the syntax and semantics in the max-plus algebra, followed by a discussion on Max-Plus Linear (MPL) systems. In max-plus algebra, $\mathbb{R}, \mathbb{R}_{\max}, \varepsilon$ are respectively defined as the set of real numbers, $\mathbb{R} \cup \{\varepsilon\}$ and $-\infty$. The set $\mathbb{R}_{\max}$ is equipped with two binary operations, $\oplus$ and $\otimes$, where

$$a \oplus b := \max\{a, b\} \quad \text{and} \quad a \otimes b := a + b, \tag{2.1}$$

for all $a, b \in \mathbb{R}_{\max}$. The algebraic structure $(\mathbb{R}_{\max}, \oplus, \otimes)$ is a semiring with ε and 0 as the null and unit element, respectively [10, 64].

By $\mathbb{R}_{\max}^{n\times m}$, we denote the set of $n \times m$ max-plus algebraic matrices whose elements are in $\mathbb{R}_{\max}$. The notation $A(i, j)$ represents the entry of matrix A at i-th row and j-th column. Furthermore, $A(i, \cdot)$ and $A(\cdot, j)$ are the i-th row and j-th column of A, respectively. The operations in (2.1) can be extended to matrices as follows. For $A, B \in \mathbb{R}_{\max}^{n\times m}, C \in \mathbb{R}_{\max}^{m\times p}$ and $\alpha \in \mathbb{R}_{\max}$,

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

for all i, j in the corresponding dimensions. Given two matrices with the same dimension, A and B, the partial order relations are defined as follows

$$\left.\begin{array}{lll}
A \ge B & \text{iff} & A \oplus B = A, \\
A \le B & \text{iff} & A \oplus B = B, \\
A = B & \text{iff} & A \ge B \text{ and } A \le B.
\end{array}\right\} \tag{2.2}$$

For a natural number r, the r-th max-plus algebraic power of a square matrix $A \in \mathbb{R}_{\max}^{n\times n}$ is denoted by $A^{\otimes r}$ and corresponds to $A \otimes \ldots \otimes A$ (r times). The matrix $A^{\otimes 0}$ is defined as an n-dimensional identity matrix I_n, where all diagonals and non-diagonal elements are 0 and ε, respectively. Similarly, for $a, b \in \mathbb{R}$, the max-plus algebraic power of a w.r.t. b is denoted by $a^{\otimes b}$ and equals to ab in the conventional algebra.

Given $V = \{v_1, \ldots, v_m\}$ as a set of vectors in $\mathbb{R}_{\max}^n$, we use the same notation to denote a matrix whose all columns are in V i.e., $V(\cdot, i) = v_i$ for $1 \leq i \leq m$. A vector $v \in \mathbb{R}_{\max}^n$ is *a max-plus linear combination* of V if $v = \alpha_1 \otimes v_1 \oplus \ldots \oplus \alpha_p \otimes v_m$ for some scalars $\alpha_1, \ldots, \alpha_m \in \mathbb{R}_{\max}$ or equivalently there exists $w \in \mathbb{R}_{\max}^m$ such that $V \otimes w = v$. The set of all max-plus linear combinations of V is called *max-plus cone* and denoted by $\mathsf{cone}(V)$ [25]. Furthermore, we call $v_1, \ldots, v_p$ as the *bases* of $\mathsf{cone}(V)$. In this thesis, we may consider a max-plus cone whose all vectors do not contain ε,

$$\mathsf{rcone}(V) = \left\{ \bigoplus_{i=1}^{p} \alpha_i \otimes V(\cdot, i) \;\middle|\; \alpha_i \in \mathbb{R} \right\}. \tag{2.3}$$

This restriction can be achieved if V is a regular matrix (cf. Definition 2.1). It is straightforward to see that $\mathbb{R}_{\max}^n = \mathsf{cone}(I_n)$ and $\mathbb{R}^n = \mathsf{rcone}(I_n)$.

Definition 2.1 (Regular Matrix [64])**.** A matrix in max-plus algebra is called regular if there is at least one finite element in each row. □

The following notations deal with a regular matrix $A \in \mathbb{R}_{\max}^{n\times n}$ and are taken from [73]. Given a square matrix $A \in \mathbb{R}_{\max}^{n\times n}$, a tuple $\mathbf{g} = (g_1, \ldots, g_n) \in \{1, \ldots, n\}^n$ is *finite coefficient* of A if $A(i, g_i) \neq \varepsilon$ for all $i \in \{1, \ldots, n\}$. The *region matrix* w.r.t. a finite coefficient $\mathbf{g}$ is defined as

$$A_{\mathbf{g}}(i, j) = \begin{cases} A(i, j), & \text{if } g_i = j \\ \varepsilon, & \text{otherwise.} \end{cases} \tag{2.4}$$

The *conjugate* of A is defined as follows

$$A^{\mathsf{c}}(j, i) = \begin{cases} -A(i, j), & \text{if } A(i, j) \neq \varepsilon \\ \varepsilon, & \text{otherwise.} \end{cases} \tag{2.5}$$

A Max-Plus-Linear (MPL) system is a discrete-event dynamics over max-plus algebra defined as

$$\mathbf{x}(k) = A \otimes \mathbf{x}(k-1), \tag{2.6}$$

where $A \in \mathbb{R}_{\max}^{n\times n}$ is the matrix system and $\mathbf{x}(k) = [x_1(k) \cdots x_n(k)]^\top \in \mathbb{R}^n$ is the state variables [10]. Traditionally, $\mathbf{x}$ represents the time stamps of the discrete-events, while $k \geq 0$ corresponds to an event counter. Therefore, it is more convenient to take $\mathbb{R}^n$ (instead of $\mathbb{R}_{\max}^n$) as the state space.

Definition 2.2 (Orbit)**.** Given an MPL system (2.6) with an initial vector $\mathbf{x}(0)$, a sequence of vectors $\mathbf{x}(0)\mathbf{x}(1)\mathbf{x}(2)\ldots$ is called as *orbit* of $\mathbf{x}(0)$ w.r.t. A. □

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 a notation to refer an orbit e.g., $\pi = \mathbf{x}(0)\mathbf{x}(1)\ldots$. 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., $\pi[j..] = \mathbf{x}(j)\mathbf{x}(j+1)\ldots$. These notations are adopted similarly for a *path of transition system* (cf. Definition 3.3). We call 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 $j \geq 0$).

The state matrix of (2.6) has significant roles w.r.t. algebraic properties such as eigenproblems [46, 89], periodic properties [72, 80], ultimate behaviour [52] and definite form [84]. Such properties are related to the notion of precedence graph and irreducibility.

Definition 2.3 (Precedence Graph [10])**.** 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 $A(i, j)$ if $A(i, j) \neq \varepsilon$. □

Definition 2.4 (Irreducible Matrix [10])**.** A matrix $A \in \mathbb{R}_{\max}^{n\times n}$ is called irreducible if the corresponding precedence $\mathcal{G}(A)$ 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 [10, 52]. 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., $|p| = A(i_2, i_1) + \ldots + A(i_k, i_{k-1})$. Furthermore, its average weight is defined as the total weight divided by its length i.e., $|p|/(k-1)$.

Definition 2.5 (Critical Graph [10])**.** A circuit, a path that begins and ends at the same node, is called *critical* if it has maximum average weight. The *critical graph* $\mathcal{G}^{crit}(A)$ is a subgraph of $\mathcal{G}(A)$ whose the nodes and edges correspond to the critical circuit of $\mathcal{G}(A)$. □

Generally speaking, $\mathcal{G}^{crit}(A)$ consists of the nodes and edges of in $\mathcal{G}(A)$ that belong to a critical circuit of $\mathcal{G}(A)$. Given a directed graph $\mathcal{G}$, a subgraph $\mathcal{G}'$ is called *strongly connected subgraph* of $\mathcal{G}$ if it is strongly connected and maximal w.r.t. inclusion (if a node or edge is added to $\mathcal{G}'$, then it is no longer strongly connected). The *cyclicity* of a strongly connected graph $\mathcal{G}'$ is the greatest common divisor (GCD) of the lengths of all circuits in $\mathcal{G}'$ [10]. Furthermore, the cyclicity of a graph, in general, is equal to the least common multiple (LCM) of the cyclicities of all its strongly connected subgraphs [51]. With regards to $A \in \mathbb{R}_{\max}^{n\times n}$, the cyclicity of A is denoted as $\mathsf{cyc}(A)$ and equals to the cyclicity of its critical graph $\mathcal{G}^{crit}(A)$ [10].

Every irreducible matrix $A \in \mathbb{R}_{\max}^{n\times n}$ admits a unique max-plus eigenvalue $\lambda \in \mathbb{R}$ and its corresponding eigenspace $E(A) = \{\mathbf{x} \in \mathbb{R}^n \mid A \otimes \mathbf{x} = \lambda \otimes \mathbf{x}\}$[1] [10, 89]. Both eigenvalue and eigenspace are related to the corresponding precedence graph $\mathcal{G}(A)$. The scalar λ is equal to the *maximum cycle mean* of $\mathcal{G}(A)$; that is, the average weight of critical circuits in $\mathcal{G}(A)$ [10]. Furthermore, the eigenspace $E(A)$ can be computed from $A_\lambda^+ = \bigoplus_{k=1}^{n}((-\lambda) \otimes A)^{\otimes k}$. More specifically, $E(A)$ is the max-plus linear combination of the i^{th} column of A_λ^+, for i such that $A_\lambda^+(i, i) = 0$ [10].

[1] In this thesis, we only consider eigenvectors whose all elements are finite.

Thus, $E(A)$ is a max-plus cone. A reducible matrix may has multiple eigenvalues with the largest one (maximum eigenvalue) equals to the maximum cycle mean in $\mathcal{G}(A)$. However, it is important to note that the eigenvectors corresponding to non-maximum eigenvalue always contain infinite element ε.

Algorithm 2.1 describes the procedure to solve the eigenproblem for $A \in \mathbb{R}_{\max}^{n\times n}$. It yields the (maximum) eigenvalue and the corresponding eigenspace. In the case of reducible matrices, the corresponding eigenspace may be an empty set as we only seek eigenvectors whose all elements are finite.

Algorithm 2.1 Solving eigenproblem for MPL systems

Input: $A \in \mathbb{R}_{\max}^{n\times n}$,
Outputs: eigenvalue $\lambda \in \mathbb{R}^n$,
eigenspace expessed as a max-plus matrix

function MPL_EIGEN(A)
  $n \leftarrow$ ROW(A) ▷ the number of rows of A
  $\lambda \leftarrow \max\{A(1,1), A(2,2), \ldots, A(n,n)\}$
  $\mathbf{M} \leftarrow$ EMPTYSTACK()
  $\mathbf{M}$.push_back(A) ▷ a stack consisting of $\{A, A^2, \ldots, A^n\}$
  $B \leftarrow A$
  for $2 \le i \le n$ **do**
    $B \leftarrow B \otimes A$
    $\lambda \leftarrow \max\{\lambda, \dfrac{\max\{B(1,1), B(2,2), \ldots, B(n,n)\}}{i}\}$
    $\mathbf{M}$.push_back(B)
  $B \leftarrow (\mathbf{M}[0] - \lambda) \oplus (\mathbf{M}[1] - 2\lambda) \oplus \ldots \oplus (\mathbf{M}[n-1] - n\lambda)$ ▷ a matrix corresponding to A_λ^+
  $E \leftarrow$ MATRIX()
  for $1 \le i \le n$ **do**
    if $B(\cdot, i) \in \mathbb{R}^n$ and $B(i,i) = 0$ **then** ▷ we only consider real-valued eigenvectors
      $m \leftarrow \max\{B(1,i), B(2,i), \ldots, B(n,i)\}$
      E.add_column($B(\cdot, i) - m$)
  $E \leftarrow$ unique_cols(E)
  return $\langle \lambda, E \rangle$

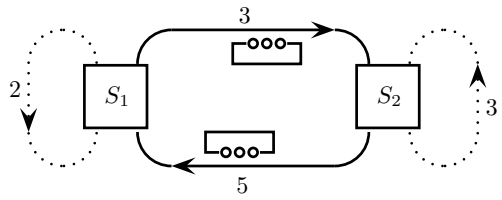


Figure 2.1: A simple railway network represented by an MPL system in (2.7).

Example 2.1. Consider a two-dimensional MPL system $\mathbf{x}(k+1) = A \otimes \mathbf{x}(k)$ that models a simple railway network shown in Figure 2.1, where

$$A = \begin{bmatrix} 2 & 5 \\ 3 & 3 \end{bmatrix}, \tag{2.7}$$

and $x_i(k)$ 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. For instance, after a train departs from S_1 at m time unit then the next departure cannot be earlier than $(m+2)$ time unit.

The network in Figure 2.1 can be interpreted as the precedence graph $\mathcal{G}(A)$. Furthermore, the critical graph $\mathcal{G}^{crit}(A)$ corresponds to a circuit $\{S_1 \to S_2 \to S_1\}$. Hence, $\mathsf{cyc}(A) = 2$. Furthermore, the max-plus eigenvalue is $\lambda = 4$ with the eigenspace $E(A) = \{\mathbf{x} \in \mathbb{R}^2 \mid x_1 - x_2 = 1\}$. □

Another important feature of MPL systems is the cycle-time vector which governs the periodic behaviour of the orbits.

Definition 2.6 (Cycle-Time Vector [64])**.** Consider a regular MPL system (2.6), and assume that for all $j \in \{1, \ldots, n\}$ the quantity η_j, defined by

$$\eta_j = \lim_{k \to +\infty} (x_j(k)/k),$$

exists. Then the vector $\chi = [\eta_1 \ \ldots \ \eta_n]^\top$ is called the the cycle-time vector of the orbit $\mathbf{x}(0)\mathbf{x}(1)\ldots$ with respect to A. □

The computation of the cycle-time vector is important, as it can shed light on the asymptotic behaviour of MPL systems. Furthermore, it can be used to determine whether the states of an MPL system are eventually periodic (in a max-plus algebraic sense). It has been shown in [64, Theorem 3.11] that if the cycle-time vector of A exists for at least one initial condition, then it exists for any initial condition, and such vector is unique. Instead of computing the limit as in Definition 2.6, the cycle-time vector can be generated using a procedure [58, Algorithm 31] that is summarized in Algorithm 2.2.

Algorithm 2.2 Computation of cycle-time vector for MPL systems

Input: $A \in \mathbb{R}_{\max}^{n \times n}$,
Outputs: cycle-time vector of A

```
1: function MPL_CTV(A)
2:     n ← ROW(A)
3:     B ← MATRIX(n, n)
4:     for i ∈ {1, ..., n} do
5:         for j ∈ {1, ..., n} do
6:             B(i, j) = (A(i, j) > ε) ? 0 : ε
7:     M ← EMPTYSTACK()
8:     x ← ZEROS(n, 1)                          ▷ zero-column vector as initial vector
9:     M.push_back(x)                           ▷ add a vector into M from the back
10:    it ← 0
11:    while true do
12:        it ← it + 1
13:        M.push_back(A ⊗ M[it − 1])
14:        d ← 1
15:        while (it − 2d > 0) do
16:            v1 ← M[it] − M[it − d]
17:            v2 ← M[it − d] − M[it − 2d]
18:            v3 ← B ⊗ v1
19:            d ← d + 1
20:            if ((v1 == v2) ∧ (v1 == v3)) then
21:                return (1/d) v1
```

The algorithm starts by defining a matrix B, for which the element $B(i, j)$ is 0 if $A(i, j)$ is finite, and otherwise $B(i, j) = \varepsilon$. Then, given an initial vector $\mathbf{x}(0) \in \mathbb{R}^n$,

we iterate (2.6) until there exist non-negative integers it, d such that $\mathbf{x}(it) - \mathbf{x}(it - d) = \mathbf{x}(it - d) - \mathbf{x}(it - 2d)$ and $B \otimes v = v$ where $v = \mathbf{x}(it) - \mathbf{x}(it - d)$. As we mentioned before, the cycle-time vector is unique and is independent of the chosen initial vector. In line 8 of Algorithm 2.2, we set the initial vector to a zero-column vector.

The following results relate the cycle-time vector with the orbits of a regular MPL system (2.6) and the emptiness of its eigenspace. In general, the elements of χ may be different, as shown in [58, Example 1]. However, if $E(A) \neq \emptyset$ then the elements of χ are all equal as stated in Proposition 2.1.

Theorem 2.1 ([58, Theorem 1])**.** Suppose we have a regular MPL system (2.6). For each $\mathbf{x}(0) \in \mathbb{R}^n$ there exist natural numbers p, q such that $\mathbf{x}(k+q) = (q \times \chi) + \mathbf{x}(k)$ for all $k \geq p$, where $\chi = [\eta_1 \ \ldots \ \eta_n]^\top$ is the cycle-time vector of A and the multiplication $q \times \chi$ is defined in the classical algebra. □

Proposition 2.1 ([1, Proposition 1])**.** Suppose a regular MPL system (2.6) has a max-plus eigenvalue λ. The eigenspace $E(A)$ is not empty iff $\chi = [\lambda \ \ldots \ \lambda]^\top$.

Proof. ($\Rightarrow$) Suppose $E(A) \neq \emptyset$. By taking $\mathbf{x}(0) \in E(A)$, we have $\mathbf{x}(k+1) = \lambda \otimes \mathbf{x}(k)$ for $k \geq 0$ which implies $x_j(k) = (\lambda \times k) + x_j(0)$ for $j \in \{1, \ldots, n\}$. It is straightforward that $\lim_{k \to +\infty} x_j(k)/k = \lambda$ for all $j \in \{1, \ldots, n\}$. Therefore, the resulting cycle-time vector is $\chi = [\lambda \ \ldots \ \lambda]^\top$
($\Leftarrow$) Suppose $\mathbf{x}(0) \in \mathbb{R}^n$. By Theorem 2.1, there exist p, q such that $\mathbf{x}(p+q) = q \times \chi + \mathbf{x}(p)$. Because $\chi = [\lambda \ \ldots \ \lambda]^\top$, it can be written as $\mathbf{x}(p+q) = (q \times \lambda) \otimes \mathbf{x}(p)$. Let

$$v = \bigoplus_{i=1}^{q} (\lambda \times (q - i)) \otimes \mathbf{x}(p + i - 1),$$

one could check that $v \in \mathbb{R}^n$ and $A \otimes v = \lambda \otimes v$. Thus, $E(A) \neq \emptyset$. □

We will now describe the notions of transient and periodicity of MPL systems and their relation with cycle-time vector.

Proposition 2.2 (Transient [10, 80])**.** For an irreducible matrix $A \in \mathbb{R}_{\max}^{n \times n}$ with its corresponding max-plus eigenvalue $\lambda \in \mathbb{R}$ and cyclicity $\mathsf{cyc}(A) = c$, there exists $l \geq 0$ such that $A^{\otimes(k+c)} = (\lambda \times c) \otimes A^{\otimes k}$ for all $k \geq l$. The smallest such l is called as the transient of A. □

For the rest of this thesis, we denote $\mathsf{tr}(A)$ as the transient of A. While the cyclicity of A is related to the critical graph $\mathcal{G}^{crit}(A)$, its transient is unrelated to the dimension of A. Even a small n, it is possible that the transient bound of $A \in \mathbb{R}_{\max}^{n \times n}$ is relatively large. The upper bounds of $\mathsf{tr}(A)$, which are much larger than the actual transient, have been discussed in [32, 71, 80, 88].

By Proposition 2.2, each irreducible MPL system (2.6) enjoys an *eventually periodic* behaviour: for each $\mathbf{x}(0) \in \mathbb{R}^n$ we have $\mathbf{x}(k + \mathsf{cyc}(A)) = (\lambda \times \mathsf{cyc}(A)) \otimes \mathbf{x}(k)$ for all $k \geq \mathsf{tr}(A)$. As it will be clear in Theorem 2.2, a similar condition may be found on reducible MPL systems: we denote the corresponding transient and cyclicity as global, as per Proposition 2.2. The local transient and cyclicity for a specific initial vector $\mathbf{x} \in \mathbb{R}^n$ and for a set $X \subseteq \mathbb{R}^n$ is defined as follows.

Definition 2.7 ([1, Definition 5])**.** Given $A \in \mathbb{R}_{\max}^{n\times n}$ with a max-plus eigenvalue λ and an initial vector $\mathbf{x} \in \mathbb{R}^n$, the local transient and cyclicity of $\mathbf{x}(0)$ w.r.t. A are respectively the smallest $l, c \geq 0$ s.t. $\mathbf{x}(k+c) = (\lambda \times c) \otimes \mathbf{x}(k)$ for all $k \geq l$. 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 $\mathtt{lcm}$ stands for the "least common multiple". □

Example 2.2. For the preceding matrix in Example 2.1, its max-plus eigenvalue and transient bound is $\lambda = 4$ and $\mathsf{tr}(A) = 2$, respectively. The orbit of $\mathbf{x}(0) = [0\ 0]^\top$ is

$$\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}5\\3\end{bmatrix}, \begin{bmatrix}8\\8\end{bmatrix}, \begin{bmatrix}13\\11\end{bmatrix}, \dots$$

Notice that $\mathbf{x}(k+2) = (4 \times 2) \otimes \mathbf{x}(k)$ for $k \geq 0$. Hence, $\mathsf{cyc}(A, \mathbf{x}(0)) = 0$ and $\mathsf{tr}(A, \mathbf{x}(0)) = 2$. □

By definition, we have $\mathsf{tr}(A, \mathbb{R}^n) = \mathsf{tr}(A)$. For a max-plus cone $\mathsf{rcone}(V)$ where $V = \{v_1, \dots, v_m\}$, the cyclicity and transient can be computed from the corresponding bases, provided that $\mathsf{tr}(A, v_i)$ exists for all $1 \leq i \leq m$.

Proposition 2.3 ([1, Proposition 3])**.** For a max-plus cone $X = \mathsf{rcone}(V)$ where $V = \{v_1, \dots, v_p\}$, we have $\mathsf{tr}(A, X) = \mathsf{tr}(A, V) = \max\{\mathsf{tr}(A, v) \mid v \in V\}$ and $\mathsf{cyc}(A, X) = \mathsf{cyc}(A, V) = \mathtt{lcm}\{\mathsf{cyc}(A, v) \mid v \in V\}$.

Proof. Suppose $\mathbf{x} \in X$. It follows that there exist scalars $\alpha_1, \dots, \alpha_p$ such that $\mathbf{x} = \bigoplus_{i=1}^{p} (\alpha_i \otimes v_i)$. Let $l^* = \max\{\mathsf{tr}(A, v_i) \mid 1 \leq i \leq p\}$ and $c^* = \mathtt{LCM}\{\mathsf{cyc}(A, v_i) \mid 1 \leq i \leq p\}$. Then, we obtain

$$\begin{aligned}
A^{\otimes(l^*+c^*)} \otimes \mathbf{x} &= A^{\otimes(l^*+c^*)} \otimes \bigoplus_{i=1}^{p} (\alpha_i \otimes v_i), \\
&= \bigoplus_{i=1}^{p} \left(\alpha_i \otimes A^{\otimes(l^*+c^*)} \otimes v_i\right), \\
&= \bigoplus_{i=1}^{p} \left(\alpha_i \otimes (\lambda \times c^*) \otimes A^{\otimes l^*} \otimes v_i\right), \\
&= (\lambda \times c^*) \otimes A^{\otimes l^*} \otimes \bigoplus_{i=1}^{p} (\alpha_i \otimes v_i) = (\lambda \times c^*) \otimes (A^{\otimes l^*} \otimes \mathbf{x}),
\end{aligned}$$

which shows that $\mathsf{tr}(A, \mathbf{x}) \leq l^*$ and $\mathsf{cyc}(A, \mathbf{x}) \leq c^*$. Thus, $\mathsf{tr}(A, X) \leq \mathsf{tr}(A, V)$ and $\mathsf{cyc}(A, X) \leq \mathsf{cyc}(A, V)$. On the other hand, because $V \subseteq X$, we have $\mathsf{tr}(A, V) \leq \mathsf{tr}(A, X)$ and $\mathsf{cyc}(A, V) \leq \mathsf{cyc}(A, X)$. Hence, we can conclude that $\mathsf{tr}(A, V) = \mathsf{tr}(A, X)$ and $\mathsf{cyc}(A, V) = \mathsf{cyc}(A, X)$. □

With regards to periodic behaviour, we classify MPL systems (2.6) based on the existence of the global and local transient as follows.

Definition 2.8 ([1, Definition 6])**.** Suppose we have a regular matrix $A \in \mathbb{R}_{\max}^{n\times n}$. The underlying MPL system (2.6) is classified into three categories:

i. *never periodic*: $\mathsf{tr}(A, \mathbf{x}(0))$ does not exist for all $\mathbf{x}(0) \in \mathbb{R}^n$,

ii. *boundedly periodic*: $\mathsf{tr}(A, \mathbf{x}(0))$ exists for all $\mathbf{x}(0) \in \mathbb{R}^n$ and $\mathsf{tr}(A)$ exists,

iii. *unboundedly periodic*: $\mathsf{tr}(A, \mathbf{x}(0))$ exists for all $\mathbf{x}(0) \in \mathbb{R}^n$ but $\mathsf{tr}(A)$ does not.

We call (2.6) *periodic* if it is either *unboundedly periodic* or *boundedly periodic*. □

It is important to note that an irreducible MPL system is guaranteed to be *periodic*. On the other hand, a reducible MPL system may belong to any category. As summarised in the following results, the periodic behaviour of an MPL system is indeed related to the eigenspace and cycle-time vector of its corresponding state matrix.

Theorem 2.2 ([1, Theorem 2])**.** Suppose we have a regular matrix $A \in \mathbb{R}_{\max}^{n\times n}$ with a max-plus eigenvalue λ and cycle-time vector χ. The following statements are equivalent.

(a) The underlying MPL system (2.6) is *periodic*.

(b) The corresponding cycle-time vector is $\chi = [\lambda \ \ldots \ \lambda]^\top \in \mathbb{R}^n$.

(c) The eigenspace $E(A)$ is not empty. □

Proof. By Proposition 2.1, it is sufficient to prove $(a) \Rightarrow (c)$ and $(b) \Rightarrow (a)$.
$(a) \Rightarrow (c)$. Suppose $\mathbf{x}(0) \in \mathbb{R}^n$. Since A is periodic, there exist natual numbers l, c such that $\mathbf{x}(k + c) = \lambda c \otimes \mathbf{x}(k)$ for all $k \geq l$. One could check that

$$v = \bigoplus_{i=1}^{c} (\lambda \times (c - i)) \otimes \mathbf{x}(l + i - 1).$$

is an eigenvector of A.
$(b) \Rightarrow (a)$. By Theorem 2.1, for each $\mathbf{x}(0) \in \mathbb{R}^n$ there exist natural numbers p, q such that $\mathbf{x}(k + q) = (q \times \chi) + \mathbf{x}(k)$ for all $k \geq p$. Because $\chi = [\lambda \ \ldots \ \lambda]^\top$, it can be written as $\mathbf{x}(k + q) = (q \times \lambda) \otimes \mathbf{x}(k)$. This shows that $\mathsf{tr}(A, \mathbf{x}(0))$ exists for all $\mathbf{x}(0) \in \mathbb{R}^n$. Therefore, (2.6) is *periodic*. □

Proposition 2.4 ([1, Proposition 4])**.** Suppose we have a regular matrix $A \in \mathbb{R}_{\max}^{n\times n}$ with max-plus eigenvalue λ and non-empty eigenspace $E(A)$. If there exist $i \in \{1, \ldots, n\}$ and natural numbers l', c' such that $A^{\otimes k+c'}(\cdot, i) = \mu c' \otimes A^{\otimes k}(\cdot, i)$ for all $k \geq l'$ with $\mu < \lambda$ then (2.6) is *unboundedly periodic*.

Proof. Because $E(A) \neq \emptyset$, by Theorem 2.2, the corresponding MPL system (2.6) is *periodic*. However, as $A^{\otimes k+c'}(\cdot, i) = \mu c' \otimes A^{\otimes k}(\cdot, i)$ where $\mu < \lambda$ for all $k \geq l'$, it is deemed impossible to find l, c such that $A^{\otimes k+c} = \lambda c \otimes A^{\otimes k}$ for $k \geq l$. Consequently, (2.6) is *unboundedly periodic*. □

Although $\mathsf{tr}(A)$ does not exist for *unboundedly periodic* MPL systems, the local transient $\mathsf{tr}(A, X)$ may exist. The computation of $\mathsf{tr}(A, X)$ for $X \subseteq \mathbb{R}^n$ will be discussed in Chapter 4 and plays an important role to solve formal verification and reachability analysis of MPL systems (cf. Chapters 5 and 6).

2.2 Related Models

This section describes models that have a relation with MPL systems. The first one is Min-Plus Linear (MiPL) systems which can be viewed as the dual of MPL systems. The second one is the so-called Interval Max-Plus-Linear (IMPL) systems: an extension of MPL systems where the finite elements of the state matrix are defined over real-valued intervals. We also present the description of Piece-Wise Affine (PWA) systems that will be used to construct the abstractions of MPL, MiPL, and IMPL systems.

2.2.1 Min-Plus Linear Systems

Define $\mathbb{R}_{\min}$ and ξ respectively as $\mathbb{R} \cup \{\xi\}$ and $+\infty$. The set $\mathbb{R}_{\min}$ is equipped with two binary operations

$$a \oplus' b := \min\{a, b\} \quad \text{and} \quad a \otimes b := a + b, \tag{2.8}$$

for $a, b \in \mathbb{R}_{\min}$. The operation $\oplus'$ in (2.8) is the dual of $\oplus$ in (2.1) under the relation $\min\{a, b\} = -\max\{-a, -b\}$. The algebraic structure $(\mathbb{R}_{\min}, \oplus', \otimes)$ is a semiring with ξ and 0 respectively as the null and unit elements, and famously known as min-plus algebra [70]. Similar to max-plus algebra, the operations in min-plus algebra can be applied to matrices. For $A, B \in \mathbb{R}_{\min}^{m\times n}, C \in \mathbb{R}_{\min}^{n\times p}$ and $\alpha \in \mathbb{R}_{\min}$,

$$\begin{aligned}
[A \oplus' B](i,j) &= A(i,j) \oplus' B(i,j) = \min\{A(i,j), B(i,j)\},\\
[A \otimes' C](i,j)^2 &= \bigoplus_{k=1}^{n}{}' A(i,k) \otimes C(k,j) = \min_{1\le k\le n}\{A(i,k) + C(k,j)\},\\
[\alpha \otimes A](i,j) &= \alpha \otimes A(i,j) = \alpha + A(i,j),
\end{aligned}$$

for all i, j in the corresponding dimensions. Notice that, the matrix multiplication in min-plus algebra can be computed using $\otimes$ (instead of $\otimes'$) as follows

$$A \otimes' C = -((-A) \otimes (-C)). \tag{2.9}$$

As in max-plus algebra, given $V = \{v_1, \ldots, v_p\}$ as a set of vectors in $\mathbb{R}_{\min}^n$, a vector $v \in \mathbb{R}_{\min}^n$ is *a min-plus linear combination* of V if $v = \alpha_1 \otimes v_1 \oplus' \ldots \oplus' \alpha_p \otimes v_p$ for some scalars $\alpha_1, \ldots, \alpha_p \in \mathbb{R}_{\min}$. Moreover, we define a min-plus cone as follows

$$\mathsf{rcone}_{\min}(V) = \left\{ \bigoplus_{i=1}^{p} \alpha_i \otimes' V(\cdot, i) \;\middle|\; \alpha_i \in \mathbb{R} \right\}.$$

The power operator for min-plus algebra is defined similarly as in max-plus algebra. For $a, b \in \mathbb{R}$, the min-plus algebraic power of a w.r.t. b is denoted by $a^{\otimes b}$ and equals to ab. Furthermore, given $m \in \mathbb{N}$ and $A \in \mathbb{R}_{\min}^{n\times n}$, the m^{th} min-plus algebraic power of A is $A^{\otimes' m} = A \otimes' \ldots \otimes' A$ (m times). For $m = 0$, $A^{\otimes' 0}$ is an $n \times n$ min-plus identity matrix whose the diagonal and non-diagonal elements are 0 and ξ, respectively.

[2] To avoid ambiguity, $\otimes'$ is used for matrices multiplication in min-plus algebra.

A Min-Plus-Linear (MiPL) system [9] is defined as

$$\mathbf{x}(k) = B \otimes' \mathbf{x}(k-1), \tag{2.10}$$

where $B \in \mathbb{R}_{\min}^{n\times n}$ and $\mathbf{x}(k) \in \mathbb{R}^n$. One could check that (2.10) can be expressed as

$$-\mathbf{x}(k) = (-B) \otimes (-\mathbf{x}(k-1)). \tag{2.11}$$

The orbit of MiPL systems (2.10) is defined similarly as that of MPL systems (2.6).

The applications of MiPL systems (or min-plus algebra in general) are found on traffic modelling [70] and all-pairs shortest path (APSP) problem [31]. The latter reference provides an APSP algorithm based on matrices multiplication in min-plus algebra.

Definition 2.9 (Precedence Graph for MiPL Systems)**.** The precedence graph of $B \in \mathbb{R}_{\min}^{n\times n}$, denoted by $\mathcal{G}_{\min}(B)$, is a weighted directed graph with nodes $1, \ldots, n$ and an edge from j to i with weight $B(i,j)$ if $B(i,j) \neq \xi$. □

The notions of regularity and irreducibility for MiPL systems are defined similarly for precedence graph $\mathcal{G}_{\min}(B)$ as in Definitions 2.1-2.4. However, due to the minimisation operator in (2.8), the definition of critical circuits for MiPL systems is slightly different from that of MPL systems. Instead, the circuits with minimum average weight are the critical ones. The cyclicity for $B \in \mathbb{R}_{\min}^{n\times n}$ is denoted as $\mathsf{cyc}_{\min}(B)$ and is equal to the cyclicity of $\mathcal{G}_{\min}^{crit}(B)$.

The procedures to solve the eigenproblem and computing the cycle-time vector of MiPL systems are illustrated in Algorithm 2.3-2.4. Due to the duality between MPL and MiPL systems, they can be computed via their MPL counterparts. Furthermore, one could prove the existence of transient for irreducible MiPL systems.

Algorithm 2.3 Solving eigenproblem for MiPL systems

Input: $B \in \mathbb{R}_{\min}^{n\times n}$,
Outputs: eigenvalue $\lambda \in \mathbb{R}^n$,
eigenspace expessed as a min-plus matrix

```
1: function MIPL_EIGEN(B)
2:     ⟨λ, E⟩ ← MIPL_EIGEN(−B)          ▷ computed by Algorithm 2.1
3:     if E is not an empty matrix then
4:         E ← −E
5:     return ⟨−λ, E⟩
```

Algorithm 2.4 Computation of cycle-time vector for MiPL systems

Input: $B \in \mathbb{R}_{\min}^{n\times n}$,
Outputs: cycle-time vector of B

```
1: function MIPL_CTV(B)
2:     χ ← MPL_CTV(−B)                  ▷ computed by Algorithm 2.2
3:     return −χ
```

Proposition 2.5 (Transient for MiPL systems)**.** For an irreducible matrix $B \in \mathbb{R}_{\min}^{n\times n}$ with its corresponding min-plus eigenvalue $\lambda_{\min} \in \mathbb{R}$ and cyclicity $\mathsf{cyc}_{\min}(B) = c_{\min}$, there exists $\mathsf{tr}_{\min}(B) \geq 0$ such that $B^{\otimes'(k+c_{\min})} = (\lambda_{\min} \times c_{\min}) \otimes B^{\otimes' k}$ for all $k \geq \mathsf{tr}_{\min}(B)$.

Proof. Let us define $A = -B$. Indeed, A is an irreducible MPL matrix. One could check that $\mathsf{cyc}(A) = \mathsf{cyc}_{\min}(B)$ and $\lambda = -\lambda_{\min}$ where λ is the max-plus eigenvalue for A. By Proposition 2.2, A has transient bound $\mathsf{tr}(A)$ such that

$$A^{\otimes(k+c_{\min})} = (\lambda \times c_{\min}) \otimes A^{\otimes k}, \forall k \geq \mathsf{tr}(A).$$

Since $A^{\otimes i} = -(B^{\otimes' i})$ for $i \geq 0$, we have $B^{\otimes'(k+c_{\min})} = (-\lambda \times c_{\min}) \otimes B^{\otimes' k} \ \ \forall k \geq \mathsf{tr}(A)$. This concludes that $\mathsf{tr}(A) = \mathsf{tr}_{\min}(B)$. □

Example 2.3. Let us consider an MiPL system $\mathbf{x}(k) = B \otimes' \mathbf{x}(k-1)$ with $B = A$ for A in Example 2.1. Without giving the details, The min-plus eigenvalue, cyclicity and transient for B is respectively, $\lambda_{\min}(B) = 2$, $\mathsf{cyc}_{\min}(B) = 1$, and $\mathsf{tr}_{\min}(B) = 2$. Furthermore, the min-plus orbit of $\mathbf{x}(0) = [0\ 0]^\top$ is

$$\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}2\\3\end{bmatrix}, \begin{bmatrix}4\\5\end{bmatrix}, \begin{bmatrix}6\\7\end{bmatrix}, \begin{bmatrix}8\\9\end{bmatrix}, \dots$$

Notice that $\mathbf{x}(k+1) = 2 \otimes \mathbf{x}(k)$ for $k \geq 1$. Hence, $\mathsf{cyc}_{\min}(B, \mathbf{x}(0)) = 1$ and $\mathsf{tr}_{\min}(B, \mathbf{x}(0)) = 1$. □

Similarly in Definition 2.8, we also classify MiPL systems (2.10) regarding the existence of $\mathsf{tr}_{\min}(B, \mathbf{x}(0))$ and $\mathsf{tr}_{\min}(B)$: each category is defined analogously. Moreover, Theorem 2.2 and Proposition 2.4 also hold for regular MiPL systems in min-plus algebraic setup.

2.2.2 Interval Max-Plus Linear Systems

With regard to a simple transportation network in Example 2.1, in reality, there may be perturbations on the model. Namely, the travel between two stations may take longer (or possibly faster) than the expected time due to unforeseen external factors, e.g. weather conditions and the number of departing passengers at stations. While the exact travel time for each departure is impossible to measure, one can set the lower and upper bound to address the uncertainty behaviour. Therefore, instead of a fixed value, an interval is used to represent the possible travelling time.

Before the description of Interval Max-Plus-Linear (IMPL) system, we present some basic properties of interval in max-plus algebra. An interval over max-plus algebra is defined as

$$\mathbf{a} = [\underline{a}, \overline{a}] = \{a \in \mathbb{R}_{\max} \mid \underline{a} \leq a \leq \overline{a}\}, \tag{2.12}$$

where $\underline{a} \leq \overline{a}$. The extension of max-plus algebraic operations to intervals is defined as follows:

$$\left.\begin{aligned}[\underline{a}, \overline{a}] \oplus [\underline{b}, \overline{b}] &= [\underline{a} \oplus \underline{b}, \overline{a} \oplus \overline{b}],\\ [\underline{a}, \overline{a}] \otimes [\underline{b}, \overline{b}] &= [\underline{a} \otimes \underline{b}, \overline{a} \otimes \overline{b}].\end{aligned}\right\} \tag{2.13}$$

By $(\mathbb{IR}_{\max}, \oplus, \otimes)$, we define the interval max-plus algebraic structure where $\mathbb{IR}_{\max}$ is the set of intervals defined in (2.12). Similarly, $\mathbb{IR}_{\max}^{m\times n}$ denotes the set of $m \times n$ interval matrices over $\mathbb{IR}_{\max}$. For each interval max-plus algebraic interval matrix (interval matrix, in short) $\mathbf{A} = \{\mathbf{a}_{ij}\} \in \mathbb{IR}_{\max}^{m\times n}$, we define $\underline{A}, \overline{A} \in \mathbb{R}_{\max}^{m\times n}$ respectively

as the lower and upper matrix of $\mathbf{A}$ i.e., $\underline{A}(i,j) = \underline{a}_{ij}$ and $\overline{A}(i,j) = \overline{a}_{ij}$ for $1 \leq i \leq m, 1 \leq j \leq n$. Hence, $\mathbf{A}$ can be writen as $[\underline{A}, \overline{A}]$ and $A \in \mathbf{A}$ iff $\underline{A} \leq A \leq \overline{A}$. For $\mathbf{A}, \mathbf{B} \in \mathbb{IR}_{\max}^{m\times n}, \mathbf{C} \in \mathbb{IR}_{\max}^{n\times p}$ and $\alpha \in \mathbb{IR}_{\max}$, the operations for interval matrices are defined as follows

$$\begin{aligned} \mathbf{A} \oplus \mathbf{B} &= [\underline{A} \oplus \underline{B}, \overline{A} \oplus \overline{B}], \\ \mathbf{A} \otimes \mathbf{C} &= [\underline{A} \otimes \underline{C}, \overline{A} \otimes \overline{C}], \\ \alpha \otimes \mathbf{A} &= [\underline{\alpha} \otimes \underline{A}, \overline{\alpha} \otimes \overline{A}]. \end{aligned}$$

It is straightforward that if $A_1, A_2 \in \mathbf{A}$ then so is $A_1 \oplus A_2$. The k^{th} power of $\mathbf{A} \in \mathbb{IR}_{\max}^{n\times n}$ is given by $\mathbf{A}^{\otimes k} = [\underline{A}^{\otimes k}, \overline{A}^{\otimes k}]$.

Given an interval matrix $\mathbf{A} \in \mathbb{IR}_{\max}^{n\times n}$, an Interval Max-Plus-Linear (IMPL)[3] system is defined as

$$\mathbf{x}(k) = A_{k-1} \otimes \mathbf{x}(k-1), \tag{2.14}$$

where $A_{k-1} \in \mathbf{A}$ for all $k \geq 1$. In comparison to (2.6), the state matrix in (2.14) is not fixed and can be changed for each time horizon $(k-1)$. Consequently, the sequence of vectors $\mathbf{x}(0)\mathbf{x}(1)\mathbf{x}(2)\ldots$ is not unique as it depends on the corresponding state matrices $A_0, A_1, \ldots$. Notice that, in (2.14), $\mathbf{x}(k) \in [\underline{A} \otimes \mathbf{x}(k-1), \overline{A} \otimes \mathbf{x}(k-1)]$ which can be expressed as

$$\bigoplus_{j=1}^{n} (\underline{A}(i,j) + x_j(k-1)) \leq x_i(k) \leq \bigoplus_{j=1}^{n} (\overline{A}(i,j) + x_j(k-1)), \tag{2.15}$$

for each $i \in \{1, \ldots, n\}$.

In recent years, IMPL systems and interval max-plus algebra have been used to control and analyse models with uncertainties of parameters, e.g., fault-tolerant control for automated guided vehicles [93], and robust controllers in disturbance decoupling [85]. Several problems and basic properties of IMPL systems have been studied, such as eigenproblems [29, 61], linear equations over interval matrices [78, 79], and reachability analysis [27]. In this thesis, we define the notion of uniform IMPL systems as follows.

Definition 2.10 (Uniform IMPL Systems)**.** An IMPL system (2.14) with an interval matrix $\mathbf{A} = [\underline{A}, \overline{A}] \in \mathbb{IR}_{\max}^{n\times n}$ is called *uniform* if the following conditions hold:

- for $1 \leq i, j \leq n$, both $\underline{A}(i,j)$ and $\overline{A}(i,j)$ are either finite or infinite,
- $\underline{A} \neq \overline{A}$. □

For the rest of this thesis, unless otherwise stated, whenever we mention IMPL systems, we refer to the uniform ones.

Definition 2.11 (Precedence Graph of IMPL Systems)**.** The precedence graph of $\mathbf{A} = [\underline{A}, \overline{A}]$, denoted by $\mathcal{G}(\mathbf{A})$, is a weighted directed graph with nodes $1, \ldots, n$ and an edge from j to i with an interval weight $\mathbf{A}(i,j)$ if $\underline{A}(i,j) \neq \varepsilon$ and $\overline{A}(i,j) \neq \varepsilon$. □

[3] In [27], it is equivalently called by Uncertain Max-Plus-Linear System.

We can define the irreducibility of $\mathbf{A}$ as that of $\overline{A}$ due to the first rule of Definition 2.10. However, the notion of the critical circuit cannot be defined for IMPL systems because two different state matrices in $\mathbf{A}$ may have different maximum cycle mean. Similarly, the transient of IMPL systems is not "inherited" from neither the upper nor lower matrix. In fact, the orbits of an IMPL system may never be periodic, as demonstrated in Example 2.4.

Example 2.4. Suppose Figure 2.2 depicts the perturbed version of the railway network in Example 2.1.

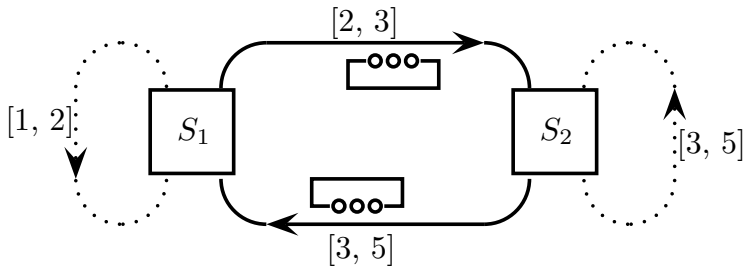


Figure 2.2: A simple railway network represented by an IMPL system. The intervals represent the time instance required by the trains to pass through the corresponding tracks.

The corresponding interval matrix is $\mathbf{A} = [\underline{A}, \overline{A}]$ where

$$\mathbf{A} = \begin{bmatrix} [1,2] & [3,5] \\ [2,3] & [3,5] \end{bmatrix}, \quad \underline{A} = \begin{bmatrix} 1 & 3 \\ 2 & 3 \end{bmatrix}, \quad \overline{A} = \begin{bmatrix} 2 & 5 \\ 3 & 5 \end{bmatrix}. \tag{2.16}$$

It is straightforward that matrix A in (2.7) belongs to $\mathbf{A}$. Suppose we have two state matrices from $\mathbf{A}$, namely

$$M_1 = \begin{bmatrix} 1.8 & 4 \\ 2.5 & 3.5 \end{bmatrix} \text{ and } M_2 = \begin{bmatrix} 1.5 & 4.5 \\ 2.8 & 3 \end{bmatrix}.$$

One could check that $\underline{A} \leq M_1, M_2 \leq \overline{A}$. The orbit from $\mathbf{x}(0) = [0\ 0]^\top$ for (2.14) where $M_1 = A_0 = A_1 = \ldots$ is

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 4 \\ 3.5 \end{bmatrix}, \begin{bmatrix} 7.5 \\ 7 \end{bmatrix}, \begin{bmatrix} 11 \\ 10.5 \end{bmatrix}, \begin{bmatrix} 14.5 \\ 14 \end{bmatrix}, \ldots.$$

On the other hand, the orbit from $\mathbf{x}(0)$ w.r.t. $M_2 = A_0 = A_1 = \ldots$ is

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 4.5 \\ 3 \end{bmatrix}, \begin{bmatrix} 7.5 \\ 7.3 \end{bmatrix}, \begin{bmatrix} 11.8 \\ 10.3 \end{bmatrix}, \begin{bmatrix} 14.8 \\ 14.6 \end{bmatrix}, \ldots.$$

Notice that, both orbits are periodic. Lastly, the orbit (up to $k = 8$) with $A_0 = A_3 = A_5 = A_6 = M_1$ and $A_1 = A_2 = A_4 = A_7 = M_2$ is

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 4 \\ 3.5 \end{bmatrix}, \begin{bmatrix} 8 \\ 6.8 \end{bmatrix}, \begin{bmatrix} 11.3 \\ 10.8 \end{bmatrix}, \begin{bmatrix} 14.8 \\ 14.3 \end{bmatrix}, \begin{bmatrix} 18.8 \\ 17.6 \end{bmatrix}, \begin{bmatrix} 21.6 \\ 21.3 \end{bmatrix}, \begin{bmatrix} 25.3 \\ 24.8 \end{bmatrix}, \begin{bmatrix} 29.3 \\ 28.1 \end{bmatrix}.$$

This orbit satisfies $\mathbf{x}(3) = 7.3 \otimes \mathbf{x}(1), \mathbf{x}(4) = 3.5 \otimes \mathbf{x}(3)$, and $\mathbf{x}(7) = 10.5 \otimes \mathbf{x}(4)$. However, it does not justify that the orbit is periodic. □

2.2.3 Piecewise-Affine Systems

Piecewise-Affine (PWA) systems [87] are defined by partitioning the input-state space into several domains characterised by polyhedra. Each domain, or PWA region, is associated with an affine function. PWA systems are sufficiently expressive for identifying or approximating generic nonlinear dynamics via multiple linearisations at different operating points [16].

It is shown in [63] that every MPL system has an equivalent PWA system. Formally, the PWA system w.r.t. an MPL system characterized by $A \in \mathbb{R}_{\max}^{n\times n}$ is generated by partitioning $\mathbb{R}^n$ into several PWA regions. Each region is formulated from a tuple $\mathbf{g} = (g_1, \ldots, g_n) \in \{1, \ldots, n\}^n$. For each i, g_i characterises the maximum term for the equation $x_i' = \max\{A(i,1) + x_1, \ldots, A(i,n) + x_n\}$[4] that is, $A(i, g_i) + x_{g_i} \geq A(i,j) + x_j$ for all $j \in \{1, \ldots, n\}$. As shown in [4], the PWA region corresponding to $\mathbf{g}$ is

$$\mathsf{R}_{\mathbf{g}} = \bigcap_{i=1}^{n} \bigcap_{j=1}^{n} \{\mathbf{x} \in \mathbb{R}^n | x_{g_i} - x_j \geq A(i,j) - A(i,g_i)\}\,. \tag{2.17}$$

Notice that, if $\mathbf{g}$ is not a finite coefficient, then $\mathsf{R}_{\mathbf{g}}$ is empty. However, a finite coefficient might lead to an empty set. The corresponding affine dynamics for non-empty $\mathsf{R}_{\mathbf{g}}$ can be written as

$$x_i' = x_{g_i} + A(i, g_i), \quad i = 1, \ldots, n. \tag{2.18}$$

The generation of a PWA system from a regular MPL matrix $A \in \mathbb{R}_{\max}^{n\times n}$ can be done using an algorithm with worst-case complexity $\mathcal{O}(n^{n+3})$ in [4, Algorithm 2]. This worst-case complexity happens only if A does not have infinite elements and all PWA regions (2.17) are non-empty. However, this worst-case rarely occurs since many regions can happen to be empty. A more efficient procedure (also with worst-case complexity $\mathcal{O}(n^{n+3})$) using the max-plus algebraic operations $\otimes, \oplus$ is described in [73, Algorithm 1].

Remark 2.1. Due to the duality with MPL systems, every MiPL system with state matrix $B \in \mathbb{R}_{\min}^{n\times n}$ can also be transformed to a PWA representation. The PWA region and the corresponding affine dynamics, are constructed from the tuple $\mathbf{g} = (g_1, \ldots, g_n) \in \{1, \ldots, n\}^n$. For each i, g_i characterises the minimum term for the equation $x_i' = \min\{B(i,1) + x_1, \ldots, B(i,n) + x_n\}$ that is, $B(i, g_i) + x_{g_i} \leq B(i,j) + x_j$ for all $j \in \{1, \ldots, n\}$. It follows that the PWA region (from an MiPL system) corresponding to $\mathbf{g}$ is

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

The corresponding affine dynamics for non-empty $\mathsf{R}_{\mathbf{g}}$ can be written as

$$x_i' = x_{g_i} + B(i, g_i), \quad i = 1, \ldots, n.$$

Notice that the resulting PWA region and affine dynamics from MiPL systems are similar to that of the MPL system. □

[4] For convenience, the "next" event is expressed as primed variables $\mathbf{x}' = [x_1' \ldots x_n']^\top$ while the variables for the "current" event is denoted by $\mathbf{x} = [x_1 \ldots x_n]^\top$.

The generation of PWA representations from IMPL systems is slightly different due to the interval matrix $[\underline{A}, \overline{A}]$. As in (2.15), the i-th next state x_i' can be expressed as $\max\{\underline{A}(i,1)+x_1,\ldots,\underline{A}(i,n)+x_n\} \leq x_i' \leq \max\{\overline{A}(i,1)+x_1,\ldots,\overline{A}(i,n)+x_n\}$. Here, we can characterise the maximum term of the lower or upper bounds. Notice that $\max\{\underline{A}(i,1)+x_1,\ldots,\underline{A}(i,n)+x_n\} \leq x_i'$ can be alternatively expressed as $\bigcap_{j=1}^{n}\{x_i' - x_j \geq \underline{A}(i,j)\}$ while $x_i' \leq \max\{\overline{A}(i,1)+x_1,\ldots,\overline{A}(i,n)+x_n\}$ is equivalent to $\bigcup_{j=1}^{n}\{x_j - x_i' \geq -\overline{A}(i,j)\}$.

Indeed, it is preferable to avoid the union of sets since checking the emptiness of the intersection of sets is easier than that of the union of sets. Hence, we need to characterise the upper bound i.e., $x_i' \leq \max\{\overline{A}(i,1)+x_1,\ldots,\overline{A}(i,n)+x_n\}$. As shown in [27], this can be done by generating the PWA representations of the upper matrix $\overline{A}$ as follows

$$\overline{\mathsf{R}}_{\mathbf{g}} = \bigcap_{i=1}^{n}\bigcap_{j=1}^{n}\left\{\mathbf{x}\in\mathbb{R}^n | x_{g_i} - x_j \geq \overline{A}(i,j) - \overline{A}(i,g_i)\right\}. \tag{2.19}$$

If the PWA region (2.19) is not empty then the corresponding affine dynamics is

$$\max\{\underline{A}(i,1)+x_1,\ldots,\underline{A}(i,n)+x_n\} \leq x_i' \leq \overline{A}(i,g_i)+x_{g_i}, \tag{2.20}$$

for $i=1,\ldots,n$ which can be expressed as $x_i' \in [A(i,\cdot)\otimes\mathbf{x},\ \overline{A}(i,g_i)+x_{g_i}]$. The dynamics in (2.20) are simpler than (2.15). Notice that, (2.20) can be further expressed as

$$\begin{aligned}&\bigcap_{i=1}^{n}\{[\mathbf{x}^\top,(\mathbf{x}')^\top]^\top \in \mathbb{R}^{2n} \mid x_{g_i} - x_i' \geq -\overline{A}(i,g_i)\} \cap \\ &\bigcap_{i=1}^{n}\bigcap_{j=1}^{n}\{[\mathbf{x}^\top,(\mathbf{x}')^\top]^\top \in \mathbb{R}^{2n} \mid x_i' - x_j \geq \underline{A}(i,j)\}.\end{aligned} \tag{2.21}$$

The following examples show the generation of PWA representations from MPL and IMPL systems.

Example 2.5. Consider the MPL system in Example 2.1 with a state matrix (2.7). The resulting PWA representations are

$$\mathbf{x}' = \begin{cases} \begin{bmatrix} x_1+2 \\ x_1+3 \end{bmatrix}, & \text{if } \mathbf{x}\in\mathsf{R}_{(1,1)}, \\ \begin{bmatrix} x_2+5 \\ x_1+3 \end{bmatrix}, & \text{if } \mathbf{x}\in\mathsf{R}_{(2,1)}, \\ \begin{bmatrix} x_2+3 \\ x_2+3 \end{bmatrix}, & \text{if } \mathbf{x}\in\mathsf{R}_{(2,2)}, \end{cases}$$

where $\mathsf{R}_{(1,1)} = \{\mathbf{x} \mid \mathbb{R}^2 \mid x_1 - x_2 \geq 3\}$, $\mathsf{R}_{(2,1)} = \{\mathbf{x} \mid \mathbb{R}^2 \mid 0 \leq x_1 - x_2 \leq 3\}$, and $\mathsf{R}_{(2,2)} = \{\mathbf{x} \mid \mathbb{R}^2 \mid x_1 - x_2 \leq 0\}$ as depicted in Figure 2.3 (left). Region $\mathsf{R}_{(1,2)}$ does not appear as it corresponds to an empty set.

On the other hand, the PWA representations for IMPL system (2.16) are

$$\mathbf{x}' \in \begin{cases} \begin{bmatrix} [\max\{x_1+1, x_2+3\},\ x_1+2] \\ [\max\{x_1+2, x_2+3\},\ x_1+3] \end{bmatrix}, & \text{if } \mathbf{x} \in \overline{\mathsf{R}}_{(1,1)}, \\ \begin{bmatrix} [\max\{x_1+1, x_2+3\},\ x_2+5] \\ [\max\{x_1+2, x_2+3\},\ x_1+3] \end{bmatrix}, & \text{if } \mathbf{x} \in \overline{\mathsf{R}}_{(2,1)}, \\ \begin{bmatrix} [\max\{x_1+1, x_2+3\},\ x_2+5] \\ [\max\{x_1+2, x_2+3\},\ x_2+5] \end{bmatrix}, & \text{if } \mathbf{x} \in \overline{\mathsf{R}}_{(2,2)}, \end{cases}$$

where $\overline{\mathsf{R}}_{(1,1)} = \{\mathbf{x} \mid \mathbb{R}^2 \mid x_1 - x_2 \geq 3\}$, $\overline{\mathsf{R}}_{(2,1)} = \{\mathbf{x} \mid \mathbb{R}^2 \mid 2 \leq x_1 - x_2 \leq 3\}$, and $\overline{\mathsf{R}}_{(2,2)} = \{\mathbf{x} \mid \mathbb{R}^2 \mid x_1 - x_2 \leq 2\}$ as shown in Figure 2.3 (right). The region $\overline{\mathsf{R}}_{(1,2)}$ is an empty set. The affine dynamics are expressed as vector of intervals. □

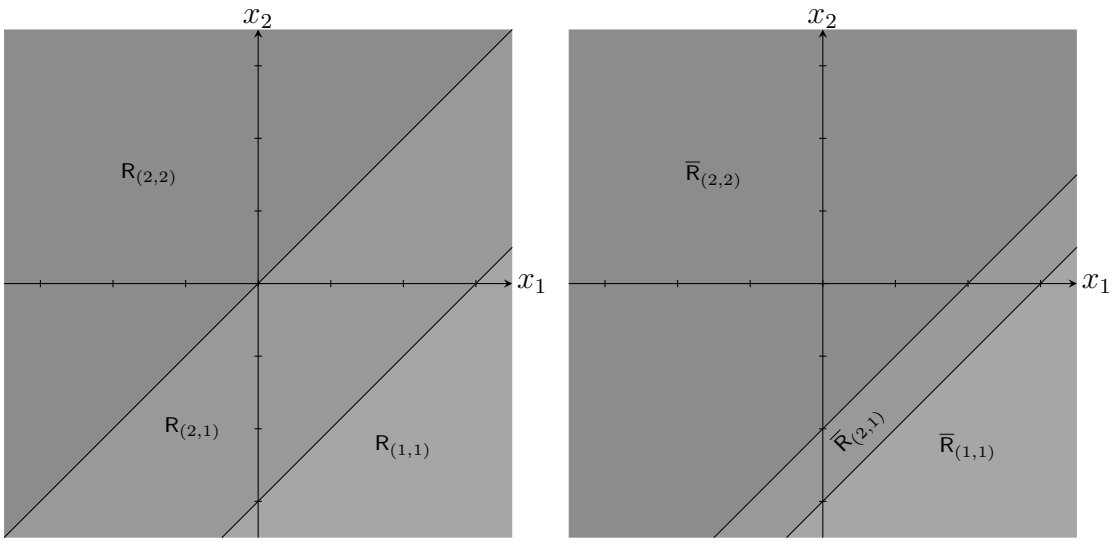


Figure 2.3: (left) Plot of PWA regions generated from MPL system (2.7). (right) Plot of PWA regions generated from IMPL system (2.16).

2.3 Difference-Bound Matrices

We present some basic notions of Difference-Bound Matrices, which are used as the data structure of abstraction procedures in Chapter 3. Some relations with max-plus algebra are mentioned.

Definition 2.12 (Difference-Bound Matrix [73])**.** A Difference-Bound Matrix (DBM) in $\mathbb{R}^n$ is the intersection of sets defined by $x_i - x_j \sim_{i,j} d_{i,j}$, where $\sim_{i,j} \in \{>, \geq\}$ and $d_{i,j} \in \mathbb{R} \cup \{-\infty\}$ for $0 \leq i, j \leq n$. The variable x_0 is set to be equal to 0. □

The dummy variable x_0 is used to represent inequalities with a single variable: $x_i \sim \alpha$ can be written as $x_i - x_0 \sim \alpha$. Definition 2.12 slightly differs from [55] which uses operators in $\{<, \leq\}$. A DBM in $\mathbb{R}^n$ can be expressed as a pair of matrices (D, S). The element $D(i, j)$ stores the bound variable $d_{i,j}$, while S represents the *sign matrix* of the operators i.e., $S(i, j) = 1$ if $\sim_{i,j} = \geq$ and $S(i, j) = 0$ otherwise.

Notice that, under Definition 2.12, D in $\mathbb{R}^n$ is an $(n+1)$-dimensional max-plus algebraic matrix while S is a binary matrix. Throughout this thesis, we may only use the bound matrix when refering a DBM. Some operations and properties in max-plus algebra can be used for DBM operations such as cross-product, intersection, computation of the canonical form, and emptiness checking.

Proposition 2.6 ([73, Proposition 2])**.** The intersection of two DBM D_1 and D_2 is $D_1 \oplus D_2$. □

Proof. Suppose a DBM D is the resulting intersection. For each pair (i, j), the (lower) bound for $x_i - x_j$ in D is equal to the tighter between bound in D_1 and D_2. This can be expressed by $D(i,j) = \max\{D_1(i,j), D_2(i,j)\} = D_1(i,j) \oplus D_2(i,j)$. Hence, $D = D_1 \oplus D_2$. □

The sign matrix for $D_1 \oplus D_2$ is computed separately as it depends on the operator of the tighter bound. More precisely, suppose S_1, S_2, and S are respectively as the sign matrices for D_1, D_2, and $D_1 \oplus D_2$, then

$$S(i,j) = \begin{cases} S_1(i,j), & \text{if } D_1(i,j) > D_2(i,j), \\ S_2(i,j), & \text{if } D_1(i,j) < D_2(i,j), \\ \min\{S_1(i,j), S_2(i,j)\}, & \text{if } D_1(i,j) = D_2(i,j). \end{cases}$$

Any DBM admits a graphical representation, called the potential graph, interpreting the DBM D as a weighted directed graph [82]. Because the bound matrix of a DBM is a max-plus algebraic matrix, the potential graph of D can be viewed as a precedence graph $\mathcal{G}(D)$. The canonical-form of a DBM D denoted as $\mathtt{cf}(D)$, is a DBM with the tightest possible bounds [55]. Computing the canonical-form representation is done by the all-pairs shortest path (APSP) problem over the corresponding potential graph [55, 82]. As the definition of the DBM is altered (see Definition 2.12), it is now implemented by all-pairs longest path (APLP) problem. One of the prominent algorithms is Floyd-Warshall [60] which has a cubic complexity w.r.t. its dimension. It has been shown in [73] the canonical form of a DBM can be computed using max-plus algebra operations.

Proposition 2.7 ([73, Proposition 3])**.** Given a DBM D in $\mathbb{R}^n$, the canonical form of D is $\mathtt{cf}(D) = \bigoplus_{m=0}^{n+1} D^{\otimes m}$. □

Proof. The element $D(i,j)$ represents the lower bound for $x_i - x_j$. We show the proof by using the precedence graph $\mathcal{G}(D)$. Notice that, $[D^{\otimes m}](i,j)$ is equal to the maximal total weight of paths with length m from j to i in $\mathcal{G}(D)$ while $[\bigoplus_{m=0}^{n+1} D^{\otimes m}](i,j)$ is equal to the maximal total weights of paths (with any length between 1 and $n+1$) from j to i. As a result, $[\bigoplus_{m=0}^{n+1}]D^{\otimes m}(i,j)$ corresponds to the tightest bound for $x_i - x_j$. Therefore, we can conclude that $\bigoplus_{m=0}^{n+1} D^{\otimes m}$ represents the canonical form of D. □

The advantage of the canonical-form representation is that the emptiness checking of a DBM can be done efficiently. Whenever a DBM D is in canonical form, if there is an i such that $d_{i,i} > 0$ or $\sim_{i,i}$ is $>$ then D is empty. Thus, the complexity of emptiness checking is linear w.r.t. dimension of the DBM.

The drawback of DBM as a data structure is that it cannot handle a complement or set difference operation. Given a DBM D in $\mathbb{R}^n$, it is possible that $\mathbb{R}^n \setminus D$ cannot be expressed as a single DBM. In general, the set difference of two DBM is a union of finitely many DBM.

In the relation with PWA system generated from an MPL system (resp. IMPL system), it is evident that the corresponding PWA expressions (2.17)-(2.18) (resp. (2.19)-(2.20)) can be expressed as a DBM. Furthermore, the image and inverse image of a DBM w.r.t. affine dynamics generated by an MPL system or an IMPL system is also a DBM (cf. Propositions 2.8-2.9).

Proposition 2.8 ([4, Theorem 1])**.** The image and inverse image of a DBM w.r.t. affine dynamics (2.18) generated by an MPL system is a DBM. □

Proposition 2.9 ([27, Proposition 2])**.** The image and inverse image of a DBM w.r.t. affine dynamics (2.21) generated by an IMPL system is a DBM. □

As mentioned in [4], the general procedure computing the image of a DBM D in $\mathbb{R}^n$ w.r.t. affine dynamic (2.18) involves: 1) generating the cross product of D and $\mathbb{R}^n$ i.e., $\{[\mathbf{x}^\top, (\mathbf{x}')\top]^\top \in \mathbb{R}^{2n} \mid \mathbf{x} \in D\}$; 2) intersecting the cross product with (2.18); then 3) calculating the canonical form of the obtained intersection; and finally 4) projecting the canonical-form representation over $x'_1, \ldots, x'_n$ by removing the complementary variables, i.e., $x_1, \ldots, x_n$. The complexity of this procedure depends critically on the second step, which runs in cubic time.

Likewise, the general procedure computing the inverse image of a DBM D' in $\mathbb{R}^n$ w.r.t. affine dynamic (2.18) involves: 1) generating the cross product of D' and $\mathbb{R}^n$; 2) intersecting the cross product with (2.18); then 3) calculating the canonical form of the obtained intersection; and finally 4) projecting the canonical-form representation over $x_1, \ldots, x_n$ by removing the primed variables, i.e., $x'_1, \ldots, x'_n$. The complexity of the inverse image computation is also in cubic time.

Remark 2.2. For the rest of this thesis, a matrix A_{ext} is the extension of $A \in \mathbb{R}_{\max}^{n\times n}$ by adding the 0-th row and column with $A(0,0) = 0, A(0,i) = A(i,0) = \varepsilon$ for $1 \leq i \leq n$. Following this, a finite coefficient $\mathbf{g}$ is also extended into $(g_0, g_1, \ldots, g_n)$ with $g_0 = 0$. These extensions are mainly used for DBM manipulations to accommodate the special variable x_0. Moreover, we may add square brackets to represent the region matrix (2.4) and conjugate (2.5) of A_{ext} e.g., $[A_{\text{ext}}]_{\mathbf{g}}$ and $[A_{\text{ext}}]^{\mathsf{c}}_{\mathbf{g}}$. □

We now will present the alternative procedure by incorporating the DBM with the corresponding affine dynamics. Notice that, in Propositions 2.10 and 2.11, the state matrix of MPL system is extended as per Remark 2.2.

Proposition 2.10 ([73, Propositions 6-7])**.** The image of a DBM D w.r.t. affine dynamics (2.18) is a DBM $D' = \bigcap_{i=0}^{n}\bigcap_{j=0}^{n}\{\mathbf{x}' \in \mathbb{R}^n \mid x'_i - x'_j = x_{g_i} - x_{g_j} + A_{\text{ext}}(i,g_i) - A_{\text{ext}}(j,g_j)\}$ where the bound of $x_{g_i} - x_{g_j}$ is taken from D. Furthermore, considering the bound matrix only, it can be expressed as $D' = [A_{\text{ext}}]_{\mathbf{g}} \otimes D \otimes [A_{\text{ext}}]^{\mathsf{c}}_{\mathbf{g}}$.

Proof. Suppose D' is the image of D w.r.t. the given affine dynamics (2.18). For each pair (i,j) where $i,j \in \{0,\ldots,n\}$, we have $x'_i - x'_j = x_{g_i} - x_{g_j} + A_{\text{ext}}(i,g_i) - A_{\text{ext}}(j,g_j)$. Therefore, $D' = \bigcap_{i=0}^{n}\bigcap_{j=0}^{n}\{\mathbf{x}' \in \mathbb{R}^n \mid x'_i - x'_j = x_{g_i} - x_{g_j} + A_{\text{ext}}(i,g_i) - A_{\text{ext}}(j,g_j)\}$.

Considering the bound matrix only, one can write $D'(i,j) = D(g_i,g_j) + A_{\text{ext}}(i,g_i) - A_{\text{ext}}(j,g_j)$ for $i,j \in \{0,\ldots,n\}$. Since $A_{\text{ext}}(i,g_i) = [A_{\text{ext}}]_{\mathbf{g}}(i,g_i)$ for $i \in \{0,\ldots,n\}$ (see the notion of region matrix in (2.4)) then $D'(i,j) = D(g_i,g_j) + [A_{\text{ext}}]_{\mathbf{g}}(i,g_i) - [A_{\text{ext}}]_{\mathbf{g}}(j,g_j)$.

On the other hand, $[[A_{\text{ext}}]_{\mathbf{g}} \otimes D \otimes [A_{\text{ext}}]^{\mathsf{c}}_{\mathbf{g}}](i,j) = \bigoplus_{k=0}^{n}([A_{\text{ext}}]_{\mathbf{g}}(i,k) \otimes (\bigoplus_{l=0}^{n} D(k,l) \otimes [A_{\text{ext}}]^{\mathsf{c}}_{\mathbf{g}}](l,j)))$. Notice that, for a fixed j there is an unique l such that $[A_{\text{ext}}]^{\mathsf{c}}_{\mathbf{g}}(l,j) \neq \varepsilon$ i.e. $l = g_j$. Similarly, for a fixed i, $[A_{\text{ext}}]_{\mathbf{g}}(i,k) \neq \varepsilon$ iff $k = g_i$. Therefore,

$$\begin{aligned}[[A_{\text{ext}}]_{\mathbf{g}} \otimes D \otimes [A_{\text{ext}}]^{\mathsf{c}}_{\mathbf{g}}](i,j) &= [A_{\text{ext}}]_{\mathbf{g}}(i,g_i) + D(g_i,g_j) + [A_{\text{ext}}]^{\mathsf{c}}_{\mathbf{g}}(g_j,j) \\ &= D(g_i,g_j) + [A_{\text{ext}}]_{\mathbf{g}}(i,g_i) - [A_{\text{ext}}]_{\mathbf{g}}(j,g_j) \\ &= D'(i,j)\end{aligned}$$

This completes the proof. □

Proposition 2.11 ([73, Propositions 8-9])**.** The inverse image of a DBM D' w.r.t. affine dynamics (2.18) is a DBM $\bigcap_{i=0}^{n}\bigcap_{j=0}^{n}\{\mathbf{x} \in \mathbb{R}^n \mid x_{g_i} - x_{g_j} = x'_i - x'_j + A_{\text{ext}}(j,g_j) - A_{\text{ext}}(i,g_i)\}$ where the bound of $x'_i - x'_j$ is taken from D'. Furthermore, considering the bound matrix only, it can be expressed as $D = ([A_{\text{ext}}]^{\mathsf{c}}_{\mathbf{g}} \otimes D' \otimes [A_{\text{ext}}]_{\mathbf{g}}) \oplus I_{n+1}$.

Proof. Suppose D is the inverse image of D' w.r.t. the given affine dynamics (2.18). For each pair (i,j) where $i,j \in \{0,\ldots,n\}$, we have $x_{g_i} - x_{g_j} = x'_i - x'_j + A_{\text{ext}}(j,g_j) - A_{\text{ext}}(i,g_i)$. Therefore, the resulting inverse image can be expressed as $D = \bigcap_{i=0}^{n}\bigcap_{j=0}^{n}\{\mathbf{x} \in \mathbb{R}^n \mid x_{g_i} - x_{g_j} = x'_i - x'_j + A_{\text{ext}}(j,g_j) - A_{\text{ext}}(i,g_i)\}$.

Now, let us starts from $[[A_{\text{ext}}]^{\mathsf{c}}_{\mathbf{g}} \otimes D' \otimes [A_{\text{ext}}]_{\mathbf{g}}](g_i,g_j) = \bigoplus_{k=0}^{n}([A_{\text{ext}}]^{\mathsf{c}}_{\mathbf{g}}(g_i,k) \otimes (\bigoplus_{l=0}^{n} D'(k,l) \otimes [A_{\text{ext}}]_{\mathbf{g}}](l,g_j)))$. Again, $[A_{\text{ext}}]_{\mathbf{g}}(k,g_i) \neq \varepsilon$ if $g_k = g_i$. Thus, for each $i,j \in \{0,\ldots,n\}$, one can write

$$\begin{aligned}[[A_{\text{ext}}]^{\mathsf{c}}_{\mathbf{g}} \otimes D' \otimes [A_{\text{ext}}]_{\mathbf{g}}](g_i,g_j) &= \bigoplus_{\substack{i^* \\ j^*}}([A_{\text{ext}}]^{\mathsf{c}}_{\mathbf{g}}(g_i,i^*) + D'(i^*,j^*) + [A_{\text{ext}}]_{\mathbf{g}}(j^*,g_j)), \\ &= \bigoplus_{\substack{i^* \\ j^*}}(-[A_{\text{ext}}]_{\mathbf{g}}(i^*,g_i) + D'(i^*,j^*) + [A_{\text{ext}}]_{\mathbf{g}}(j^*,g_j)),\end{aligned}$$

where $i^*, j^* \in \{0,\ldots,n\}$ such that $g_{i^*} = g_i$ and $g_{j^*} = g_j$. For all i^* and j^*, we have $x_{g_i} - x_{g_j} = x'_{i^*} - x'_{j^*} + [A_{\text{ext}}]_{\mathbf{g}}(j^*,g_j) - [A_{\text{ext}}]_{\mathbf{g}}(i^*,g_i)$ which shows that $x_{g_i} - x_{g_j}$ has multiple bounds up to the number of different pairs (i^*,j^*). The tightest bound of $x_{g_i} - x_{g_j}$ is equal to the maximum one; that is,

$$\begin{aligned}D(g_i,g_j) &= \bigoplus_{\substack{i^* \\ j^*}}(-[A_{\text{ext}}]_{\mathbf{g}}(i^*,g_i) + D'(i^*,j^*) + [A_{\text{ext}}]_{\mathbf{g}}(j^*,g_j)) \\ &= [[A_{\text{ext}}]^{\mathsf{c}}_{\mathbf{g}} \otimes D' \otimes [A_{\text{ext}}]_{\mathbf{g}}](g_i,g_j).\end{aligned}$$

From here, we have $D(k,l) = [[A_{\text{ext}}]^{\mathsf{c}}_{\mathbf{g}} \otimes D' \otimes [A_{\text{ext}}]_{\mathbf{g}}](k,l)$ if both k and k are in $G = \{g_0, g_1, \ldots, g_n\}$. On the other hand, if $k \notin G$ or $l \notin G$, then $D(k,l) = \varepsilon = [[A_{\text{ext}}]^{\mathsf{c}}_{\mathbf{g}} \otimes D' \otimes [A_{\text{ext}}]_{\mathbf{g}}](k,l)$. Since the diagonal elements of D are not allowed be to ε, we have $D = [A_{\text{ext}}]^{\mathsf{c}}_{\mathbf{g}} \otimes D' \otimes [A_{\text{ext}}]_{\mathbf{g}} \oplus I_{n+1}$. □

Algorithm 2.5 Computing the image of a DBM w.r.t. affine dynamics (2.18).

Inputs: $A \in \mathbb{R}_{\max}^{n\times n}$,
D, the bound matrix of the DBM,
S, the sign matrix of the DBM,
$\mathbf{g} = (g_1, \ldots, g_n)$, the finite coefficient w.r.t. (2.18)
Output: a DBM (D', S')

```
1: function IMAGE_DBM(A, D, S, g)
2:     n ← ROW(A)
3:     A ← EXTENT(A)
4:     g ← EXTENT(g)
5:     D' ← I_{n+1}
6:     S' ← EYE(n + 1)
7:     for i ∈ {0, ..., n} do
8:         for j ∈ {0, ..., n} do
9:             D'(i, j) ← D(g_i, g_j) + A(i, g_i) − A(j, g_j)
10:            S'(i, j) ← S(g_i, g_j)
11:    return ⟨D', S'⟩
```

Algorithm 2.6 Computing the inverse image of a DBM w.r.t. affine dynamics (2.18).

Inputs: $A \in \mathbb{R}_{\max}^{n\times n}$,
D', the bound matrix of the DBM,
S', the sign matrix of the DBM,
$\mathbf{g} = (g_1, \ldots, g_n)$, the finite coefficient w.r.t. (2.18)
Output: a DBM (D, S)

```
1: function INVIMAGE_DBM(A, D', S', g)
2:     n ← ROW(A)
3:     A ← EXTENT(A)
4:     g ← EXTENT(g)
5:     D' ← I_{n+1}
6:     S' ← EYE(n + 1)
7:     for ∈ {0, ..., n} do
8:         for j ∈ {0, ..., n} do
9:             b ← D'(i, j) + A(j, g_j) − A(i, g_i)
10:            s ← S'(i, j)
11:            if b > D(g_i, g_j) then
12:                D(g_i, g_j) ← b
13:                S(g_i, g_j) ← s
14:            else if b = D(g_i, g_j) then
15:                S(g_i, g_j) ← min{s, S(g_i, g_j)}
16:    return ⟨D, S⟩
```

Inspired by Propositions 2.10-2.11, Algorithms 2.5 and 2.6 show the novel procedures to compute the image and the inverse image of a DBM, respectively. At the beginning of both algorithms, we extent the matrix A and finite coefficient $\mathbf{g}$ as per Remark 2.2. In lines 5-6 of Algorithm 2.5 (resp. Algorithm 2.6), the candidate of image (resp. inverse image) is initialised with $\mathbb{R}^n$: the corresponding bound matrix is expressed as a max-plus identity matrix while the sign matrix is the usual identity matrix. Notice that, the bound and sign matrices for the image computation at pair of indexes (i, j) (lines 9-10 of Algorithm 2.5) is updated once. On the other hand, the bound and sign for the inverse image computation at pair of indexes (g_i, g_j) may be updated multiple times (lines 11-15 of Algorithm 2.6). This happens when the values in $\{g_0, \ldots, g_n\}$ are not distinct. The worst-case complexity of both algo-

rithms is $\mathcal{O}(n^2)$. They are more efficient than the existing procedures in [4] which have $\mathcal{O}(n^3)$ complexity.

The procedure to compute the image of DBM D w.r.t. MPL system (2.6) characterised by $A \in \mathbb{R}_{\max}^{n\times n}$ can be viewed as the extension of Algorithm 2.5, and involves the following steps: 1) intersecting D with each region of the PWA system (2.17); 2) computing the image of nonempty intersections by Algorithm (2.5); finally, 3) collecting the resulting images. The worst-case complexity is $\mathcal{O}(|\mathsf{R}|n^2)$, where $|\mathsf{R}|$ is the number of regions in the PWA system generated from an MPL system (2.6). Similarly, the procedure to compute the inverse image of DBM D' w.r.t. MPL system (2.6) can be done by the following steps: 1) computing the inverse image of DBM D' w.r.t. each affine dynamics of the PWA system; then 2) intersecting the inverse image with the corresponding PWA region; 3) collecting the non-empty intersection. The worst-case complexity is $\mathcal{O}(|\mathsf{R}|n^2)$. In general, computing the image and inverse image of a DBM (also, a union of finitely many DBM) w.r.t. an MPL system (2.6) results in a union of finitely many DBM. The following corollary generalises Proposition 2.8.

Corollary 2.1 ([2, Corollary 3.1])**.** The image and inverse image of a DBM w.r.t. an MPL system result in a union of finitely many DBM. □

Remark 2.3. By Remark 2.1, Algorithms 2.5 and 2.6 respectively can be applied to compute the image and inverse image of a DBM w.r.t. affine dynamics (2.18) generated from an MiPL system. Furthermore, Corollary 2.1 also applies to MiPL systems. □

We will now describe the methods to compute the image and inverse image of a DBM w.r.t. affine dynamics (2.20). As shown in [26], the steps as mentioned earlier (two paragraphs after Proposition 2.9) can be applied for affine dynamics generated from an IMPL system. The only difference is that, on the second step, one needs to intersect the cross product (of the first step) with (2.21). The worst-case complexity is again in cubic time. Next, we show that the computation of image and inverse image of a DBM w.r.t. affine dynamics (2.20) can be done using matrix operations in the max-plus algebra, which does not involve the canonical-form computation of DBM. These operations later underpin the new Algorithms 2.7-2.8.

Proposition 2.12 ([76, Proposition 1])**.** The image of DBM D w.r.t. the affine dynamics (2.20) is a DBM $D' = \bigcap_{i=0}^{n}\bigcap_{j=0}^{n}\{\mathbf{x}' \in \mathbb{R}^n \mid x'_i - x'_j \sim_{ij} d'_{ij}\}$, where $d'_{ij} = \bigoplus_{k=0}^{n}\{\underline{A}_{\text{ext}}(i,k)+D(k,g_j)-\overline{A}_{\text{ext}}(j,g_j)\}$ and $D(k,g_j)$ represents the bound of $x_j - x_{g_j}$ in DBM D. The operator $\sim_{ij}$ depends on the argmax of d'_{ij} (see Algorithm 2.7). Furthermore, considering the bound matrix only, the image computation can be expressed as $D' = (\underline{A}_{\text{ext}} \otimes D \otimes [\overline{A}_{\text{ext}}]^{\mathsf{c}}_{\mathbf{g}}) \oplus I_{n+1}$.

Proof. Let us denote $\overline{a}_{ij} = \overline{A}_{\text{ext}}(i,j)$ and $\underline{a}_{ij} = \underline{A}_{\text{ext}}(i,j)$ for $i,j \in \{0,\dots,n\}$. By (2.20), for each $i,j \in \{0,\dots,n\}$, we have

$$x'_i - x'_j \geq \max\{x_0 + \underline{a}_{i0}, \dots, x_n + \underline{a}_{in}\} - (x_{g_j} + \overline{a}_{jg_j}). \tag{2.22}$$

Because $x_i - x_j$ is (lower) bounded by $D(i,j)$ in DBM D, (2.22) can be rewritten as $x'_i - x'_j \geq \max\{D(0,g_j) + \underline{a}_{i0}, \ldots, D(n,g_j) + \underline{a}_{in}\} - \overline{a}_{jg_j}$, which can be expressed as

$$x'_i - x'_j \geq \bigoplus_{k=0}^{n} \left(\underline{a}_{ik} + D(k,g_j) - \overline{a}_{jg_j}\right). \tag{2.23}$$

By setting d'_{ij} as the right-hand side of (2.23), the resulting image can be expressed as $D' = \bigcap_{i=0}^{n}\bigcap_{j=0}^{n}\{\mathbf{x}' \in \mathbb{R}^n \mid x'_i - x'_j \sim_{ij} d'_{ij}\}$, where $\sim_{ij}$ depends on $\operatorname{argmax}(d'_{ij})$: if $\operatorname{argmax}(d'_{ij}) = k$, then $\sim_{ij}$ is taken the operator for $x_k - x_{g_j}$ in D.

We recall that $[\overline{A}_{\text{ext}}]_{\mathbf{g}}$ is a region matrix with only one finite element in each row. Furthermore, we have $[\overline{A}_{\text{ext}}]^{\mathsf{c}}_{\mathbf{g}}(g_j,j) = -[\overline{A}_{\text{ext}}]_{\mathbf{g}}(j,g_j) = -\overline{a}_{jg_j}$. Thus, the expression (2.23) is equivalent to $x'_i - x'_j \geq [\underline{A}_{\text{ext}} \otimes D \otimes [\overline{A}_{\text{ext}}]^{\mathsf{c}}_{\mathbf{g}}](i,j)$. Considering the bound matrix only, one can write $D'(i,j) = [\underline{A}_{\text{ext}} \otimes D \otimes [\overline{A}_{\text{ext}}]^{\mathsf{c}}_{\mathbf{g}}](i,j)$. It is possible that, for $i = j$, $[\underline{A}_{\text{ext}} \otimes D \otimes [\overline{A}_{\text{ext}}]^{\mathsf{c}}_{\mathbf{g}}](i,j) < 0$. Hence, the final expression is $D' = (\underline{A}_{\text{ext}} \otimes D \otimes [\overline{A}_{\text{ext}}]^{\mathsf{c}}_{\mathbf{g}}) \oplus I_{n+1}$. □

Proposition 2.13 ([76, Proposition 2])**.** The inverse image of DBM D' w.r.t. the affine dynamics (2.20) is a DBM $D = \bigcap_{i=0}^{n}\bigcap_{j=0}^{n}\{\mathbf{x} \in \mathbb{R}^n \mid x_i - x_j \sim_{ij} d_{ij}\}$, where $d_{ij} = \bigoplus_{j=0}^{n}(-\overline{A}_{\text{ext}}(j,g_j) + \bigoplus_{i=0}^{n}(D'(j,i) + \underline{A}_{\text{ext}}(i,k)))$ and $D'(i,j)$ represents the bound of $x'_i - x'_j$ in DBM D'. The operator $\sim_{ij}$ depends on the argmax of d_{ij} (see Algorithm 2.8). Furthermore, considering the bound matrix only, the inverse image computation can be expressed as $D = ([\overline{A}_{\text{ext}}]^{\mathsf{c}}_{\mathbf{g}} \otimes D' \otimes \underline{A}_{\text{ext}}) \oplus I_{n+1}$.

Proof. Notice that (2.22) is equivalent to $\bigwedge_{k=0}^{n}(x_{g_j} - x_k \geq x'_j - x'_i + \underline{A}_{\text{ext}}(i,k) - \overline{A}_{\text{ext}}(j,g_j))$. Thus, for each fixed $j,k \in \{0,\ldots,n\}$, we have $x_{g_j} - x_k \geq -\overline{A}_{\text{ext}}(j,g_j) + \bigoplus_{i=0}^{n}(D'(j,i) + \underline{A}_{\text{ext}}(i,k))$. In general, the latter expression can be rewritten as

$$x_{g_j} - x_k \geq \bigoplus_{j=0}^{n}(-\overline{A}_{\text{ext}}(j,g_j) + \bigoplus_{i=0}^{n}(D'(j,i) + \underline{A}_{\text{ext}}(i,k))), \tag{2.24}$$

because it is possible that there exists $l,j \in \{0,\ldots,n\}$ such that $l \neq j$ and $g_l = g_j$. By setting d_{ij} as the right-hand side of (2.24), the resulting inverse image can be expressed as $D = \bigcap_{i=0}^{n}\bigcap_{j=0}^{n}\{\mathbf{x} \in \mathbb{R}^n \mid x_i - x_j \sim_{ij} d_{ij}\}$ where $\sim_{ij}$ depends on $\operatorname{argmax}(d_{ij})$.

Considering the bound matrix only, (2.24) can be expressed as

$$D(g_j,k) = [[\overline{A}_{\text{ext}}]^{\mathsf{c}}_{\mathbf{g}} \otimes D' \otimes \underline{A}_{\text{ext}}](g_j,k).$$

Again, it is possible that $[[\overline{A}_{\text{ext}}]^{\mathsf{c}}_{\mathbf{g}} \otimes D' \otimes \underline{A}_{\text{ext}}](g_j,k) < 0$ for $j,k \in \{0,\ldots,n\}$ such that $g_j = k$. Thus, the final expression is $D = ([\overline{A}_{\text{ext}}]^{\mathsf{c}}_{\mathbf{g}} \otimes D' \otimes \underline{A}_{\text{ext}}) \oplus I_{n+1}$. □

Bolstered by Propositions 2.12-2.13, Algorithm 2.7 (resp. Algorithm 2.8) shows a novel procedure to compute the image (resp. inverse image) of a DBM w.r.t. the affine dynamics in (2.20). At the start of both algorithms, the lower and upper matrices and finite coefficient $\mathbf{g}$ are extented as per Remark 2.2. Then, the candidate of the output is initialised with $\mathbb{R}^n$. The worst-case complexity of Algorithm 2.7 is

in $\mathcal{O}(n^3)$: the for loops in lines 8-9 involve $(n+1)^2$ iterations, and the steps in lines 10-12 run in linear time. Similarly, the complexity of Algorithm 2.8 is $\mathcal{O}(n^3)$.

Algorithm 2.7 Image computation of a DBM (D, S) w.r.t. affine dynamics (2.20) for IMPL system

Inputs: $\mathbf{A} = [\underline{A}, \overline{A}]$, where $\underline{A}, \overline{A} \in \mathbb{R}_{\max}^{n \times n}$ with $\underline{A} \leq \overline{A}$,
D, the bound matrix of the DBM,
S, the sign matrix of the DBM,
$\mathbf{g} = (g_1, \ldots, g_n)$, the finite coefficient w.r.t. (2.20)
Output: a DBM (D', S')

1: **function** IMAGE_DBM_INTERVAL($\underline{A}, \overline{A}, D, S, \mathbf{g}$)
2: $n \leftarrow \textsc{Row}(\overline{A})$
3: $\underline{A}_{\text{ext}} \leftarrow \textsc{Extend}(\underline{A})$
4: $\overline{A}_{\text{ext}} \leftarrow \textsc{Extend}(\overline{A})$
5: $\mathbf{g} \leftarrow \textsc{Extent}(\mathbf{g})$
6: $D' \leftarrow I_{n+1}$
7: $S' \leftarrow \textsc{Eye}(n+1)$
8: **for** $i \in \{0, \ldots, n\}$ **do**
9: **for** $j \in \{0, \ldots, n\}$ **do**
10: $v \leftarrow \underline{A}_{\text{ext}}(i, \cdot) + D(\cdot, g_j)^\top - \overline{A}_{\text{ext}}(j, g_j)$
11: $val \leftarrow \max(v)$
12: $idx \leftarrow \operatorname{argmax}(v)$
13: **if** $val > D'(i,j)$ **then** ▷ $D'(i,j)$ is either 0 or ε
14: $D'(i,j) \leftarrow val$
15: $S'(i,j) \leftarrow \min_{k \in idx} S(k, g_j)$
16: **return** (D', S')

Algorithm 2.8 Inverse image computation of a DBM (D', S') w.r.t. affine dynamics (2.20) for IMPL system

Inputs: $\mathbf{A} = [\underline{A}, \overline{A}]$, where $\underline{A}, \overline{A} \in \mathbb{R}_{\max}^{n \times n}$ with $\underline{A} \leq \overline{A}$,
D', the bound matrix of the DBM,
S', the sign matrix of the DBM,
$\mathbf{g} = (g_1, \ldots, g_n)$, the finite coefficient w.r.t. (2.20)
Output: a DBM (D, S)

1: **function** INVIMAGE_DBM_INTERVAL($\underline{A}, \overline{A}, D', S', \mathbf{g}$)
2: $n \leftarrow \textsc{Row}(\overline{A})$
3: $\underline{A}_{\text{ext}} \leftarrow \textsc{Extend}(\underline{A})$
4: $\overline{A}_{\text{ext}} \leftarrow \textsc{Extend}(\overline{A})$
5: $\mathbf{g} \leftarrow \textsc{Extent}(\mathbf{g})$
6: $D \leftarrow I_{n+1}$
7: $S \leftarrow \textsc{Eye}(n+1)$
8: **for** $j \in \{0, \ldots, n\}$ **do**
9: **for** $k \in \{0, \ldots, n\}$ **do**
10: $v \leftarrow \underline{A}_{\text{ext}}(\cdot, k)^\top + D'(j, \cdot) - \overline{A}_{\text{ext}}(j, g_j)$
11: $val \leftarrow \max(v)$
12: $idx \leftarrow \operatorname{argmax}(v)$
13: $sign \leftarrow \min_{i \in idx} S'(j,i)\}$
14: **if** $val > D(g_j, k)$ **then**
15: $D(g_j, k) \leftarrow val$
16: $S(g_j, k) \leftarrow sign$
17: **else if** $val = D(g_j, k)$ **then**
18: $S(g_j, k) \leftarrow \min\{S(g_j, k), sign\}$
19: **return** (D, S)

We recall that the translation of an IMPL system into PWA representations simplifies (2.15) into (2.20). Thus, to compute the image of a DBM D w.r.t. the IMPL system dynamics (2.15), it is necessary to intersect D with all the non-empty PWA regions in (2.19). In general, the image computation involves the following steps: 1) intersecting the DBM with each region of the corresponding PWA system; 2) computing the image of non-empty intersections w.r.t. its corresponding affine dynamics (2.20) using Algorithm 2.7; 3) collecting the resulting images. Likewise, the inverse image of a DBM w.r.t. IMPL system dynamics (2.15) can be computed by the following steps: 1) computing the inverse image of the DBM w.r.t each affine dynamics (2.20) of the PWA system by Algorithm 2.8, 2) intersecting the resulting inverse image with the corresponding PWA region, and 3) collecting the non-empty intersections.

Corollary 2.2 ([26, Corollary 5.2])**.** The image and inverse image of a union finitely many DBM w.r.t. an IMPL system are also a union finitely many DBM. □

We present the examples to compute the image and inverse image of a DBM w.r.t. an MPL system (2.7) and an IMPL system (2.16).

Example 2.6. Let us compute the image of $X_0 = \{\mathbf{x} \in \mathbb{R}^2 \mid 0 \leq x_1 \leq 2, 0 \leq x_2 \leq 2\}$ w.r.t. an MPL system (2.7). We recall that, in Example 2.5, the resulting PWA representations for the MPL system have three regions, namely $\mathsf{R}_{(0,1,1)}, \mathsf{R}_{(0,2,1)}$, and $\mathsf{R}_{(0,2,2)}$. The intersection of X_0 and each region are $X_0 \cap \mathsf{R}_{(0,1,1)} = \emptyset, X_0 \cap \mathsf{R}_{(0,2,1)} = \{\mathbf{x} \in \mathbb{R}^2 \mid 0 \leq x_1 \leq 2, 0 \leq x_2 \leq 2, 0 \leq x_1 - x_2 \leq 2\}$, and $X_0 \cap \mathsf{R}_{(0,2,2)} = \{\mathbf{x} \in \mathbb{R}^2 \mid 0 \leq x_1 \leq 2, 0 \leq x_2 \leq 2, -2 \leq x_1 - x_2 \leq 0\}$. Skipping the details, the image of $X_0 \cap \mathsf{R}_{(0,2,1)}$ and $X_0 \cap \mathsf{R}_{(0,2,2)}$ is $\{\mathbf{x} \in \mathbb{R}^2 \mid 5 \leq x_1 \leq 7, 3 \leq x_2 \leq 5, 0 \leq x_1 - x_2 \leq 2\}$ and $\{\mathbf{x} \in \mathbb{R}^2 \mid 5 \leq x_1 \leq 7, 3 \leq x_2 \leq 5, x_1 - x_2 = 2\}$. It is coincidence that the latter set is a subset of the former one. Hence, the image of X_0 w.r.t. MPL system (2.7) is $\mathsf{Img}(A, X_0) = \{\mathbf{x} \in \mathbb{R}^2 \mid 5 \leq x_1 \leq 7, 3 \leq x_2 \leq 5, 0 \leq x_1 - x_2 \leq 2\}$.

Now, let us determine the inverse image of X_0 w.r.t. an MPL system (2.7). Without going into details, the inverse image of X_0 w.r.t. the affine dynamics for $\mathsf{R}_{(0,1,1)}, \mathsf{R}_{(0,2,1)}$ and $\mathsf{R}_{(0,2,2)}$ is $\{\mathbf{x} \in \mathbb{R}^2 \mid -2 \leq x_1 \leq -1\}$, $\{\mathbf{x} \in \mathbb{R}^2 \mid -3 \leq x_1 \leq -1, -5 \leq x_2 \leq -3, 0 \leq x_1 - x_2 \leq 4\}$, and $\{\mathbf{x} \in \mathbb{R}^2 \mid x_2 = -3\}$ respectively. The intersection of the resulting inverse images w.r.t. the corresponding PWA region is $\{\mathbf{x} \in \mathbb{R}^2 \mid -2 \leq x_1 \leq -1, x_2 \leq -4\}, \{\mathbf{x} \in \mathbb{R}^2 \mid -3 \leq x_1 \leq -1, -5 \leq x_2 \leq -3, 0 \leq x_1 - x_2 \leq 3\}$, and $\{\mathbf{x} \in \mathbb{R}^2 \mid x_1 \leq -3, x_2 = -3\}$. Hence the inverse image of X_0 is $\mathsf{Inv}(\mathbf{A}, X_0) = \{\mathbf{x} \in \mathbb{R}^2 \mid -2 \leq x_1 \leq -1, x_2 \leq -4\} \cup \{\mathbf{x} \in \mathbb{R}^2 \mid -3 \leq x_1 \leq -1, -5 \leq x_2 \leq -3, 0 \leq x_1 - x_2 \leq 3\} \cup \{\mathbf{x} \in \mathbb{R}^2 \mid x_1 \leq -3, x_2 = -3\}$

The plots for the resulting image and inverse image are shown in Figure 2.4 (left). The left-pointing (resp. down-pointing) arrow in $\mathsf{Inv}(A, X_0)$ indicates that there is no lower bound for variable x_1 (resp. x_2). □

Example 2.7. Let us compute the image of $X_0 = \{\mathbf{x} \in \mathbb{R}^2 \mid 0 \leq x_1 \leq 2, 0 \leq x_2 \leq 2\}$ w.r.t. an IMPL system in (2.16). We recall that, in Example 2.5, the resulting PWA representations (of the IMPL system) have three regions, namely $\overline{\mathsf{R}}_{(0,1,1)}, \overline{\mathsf{R}}_{(0,2,1)}$, and $\overline{\mathsf{R}}_{(0,2,2)}$. The intersection of X_0 and each region are $X_0 \cap \overline{\mathsf{R}}_{(0,1,1)} = \emptyset, X_0 \cap \overline{\mathsf{R}}_{(0,2,1)} = \{\mathbf{x} \in \mathbb{R}^2 \mid x_1 = 2, x_2 = 0\}$, and $X_0 \cap \overline{\mathsf{R}}_{(0,2,2)} = \{\mathbf{x} \in \mathbb{R}^2 \mid 0 \leq x_1 \leq 2, 0 \leq$

$x_2 \leq 2, -2 \leq x_1 - x_2 \leq 2\}$. Skipping the details, the image of $X_0 \cap \overline{\mathsf{R}}_{(0,2,1)}$ and $X_0 \cap \overline{\mathsf{R}}_{(0,2,2)}$ is $\{\mathbf{x} \in \mathbb{R}^2 \mid 3 \leq x_1 \leq 7, 4 \leq x_2 \leq 5, -2 \leq x_1 - x_2 \leq 1\}$ and $\{\mathbf{x} \in \mathbb{R}^2 \mid 3 \leq x_1 \leq 7, 3 \leq x_2 \leq 7, -2 \leq x_1 - x_2 \leq 2\}$. Notice that the former set is a subset of the latter one. Hence, the image of X_0 w.r.t. an IMPL system (2.16) is $\mathsf{Img}(\mathbf{A}, X_0) = \{\mathbf{x} \in \mathbb{R}^2 \mid 3 \leq x_1 \leq 7, 3 \leq x_2 \leq 7, -2 \leq x_1 - x_2 \leq 2\}$.

Now, let us determine the inverse image of X_0 w.r.t. an IMPL system (2.16). Without going into details, the inverse image of X_0 w.r.t. the affine dynamics for $\overline{\mathsf{R}}_{(0,1,1)}, \overline{\mathsf{R}}_{(0,2,1)}$ and $\overline{\mathsf{R}}_{(0,2,2)}$ are $\{\mathbf{x} \in \mathbb{R}^2 \mid -2 \leq x_1 \leq 0, x_2 \leq -1, x_1 - x_2 \geq 1\}$, $\{\mathbf{x} \in \mathbb{R}^2 \mid -3 \leq x_1 \leq 0, -5 \leq x_2 \leq -1, 0 \leq x_1 - x_2 \leq 4\}$, and $\{\mathbf{x} \in \mathbb{R}^2 \mid x_1 \leq 0, -5 \leq x_2 \leq -1, x_1 - x_2 \leq 3\}$ respectively. The intersections of the resulting inverse images w.r.t. the corresponding PWA regionare $\{\mathbf{x} \in \mathbb{R}^2 \mid -2 \leq x_1 \leq 0, x_2 \leq -3, x_1 - x_2 \geq 3\}, \{\mathbf{x} \in \mathbb{R}^2 \mid -3 \leq x_1 \leq 0, -5 \leq x_2 \leq -2, 2 \leq x_1 - x_2 \leq 3\}$, and $\{\mathbf{x} \in \mathbb{R}^2 \mid x_1 \leq 0, -5 \leq x_2 \leq -1, x_1 - x_2 \leq 2\}$. Hence the inverse image of X_0 is $\mathsf{Inv}(\mathbf{A}, X_0) = \{\mathbf{x} \in \mathbb{R}^2 \mid -2 \leq x_1 \leq 0, x_2 \leq -3, x_1 - x_2 \geq 3\} \cup \{\mathbf{x} \in \mathbb{R}^2 \mid -3 \leq x_1 \leq 0, -5 \leq x_2 \leq -2, 2 \leq x_1 - x_2 \leq 3\} \cup \{\mathbf{x} \in \mathbb{R}^2 \mid x_1 \leq 0, -5 \leq x_2 \leq -1, x_1 - x_2 \leq 2\}$.

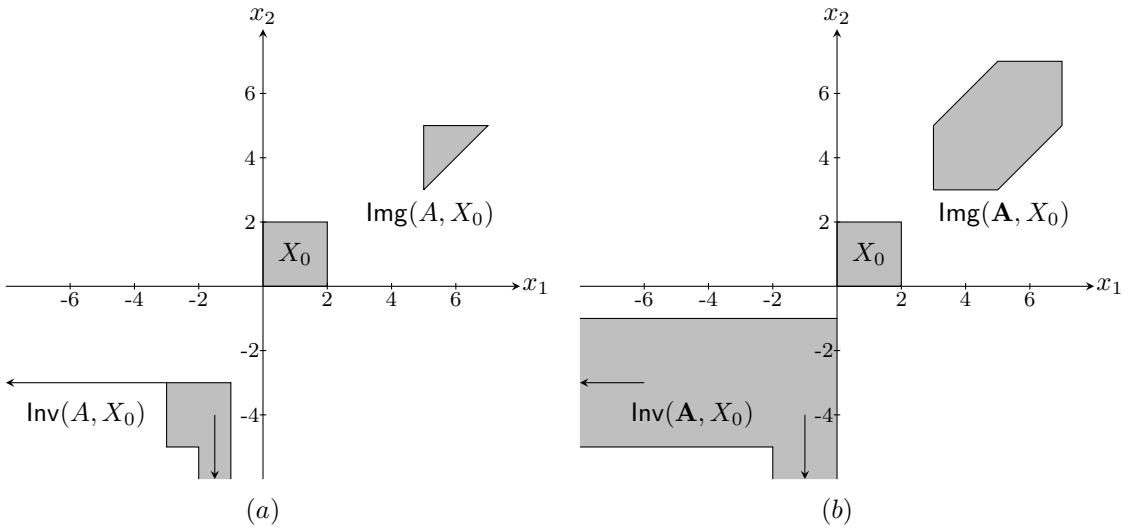


Figure 2.4: (a) The image and inverse image of X_0 w.r.t. an MPL system (2.7). (b) The image and inverse image of X_0 w.r.t. an IMPL system (2.16).

The plots for image and inverse image of X_0 are depicted in Figure 2.4 (right). The left-pointing (resp. down-pointing) arrow in $\mathsf{Inv}(A, X_0)$ indicates that there is no lower bound for variable x_1 (resp. x_2). □

Notice that, in Figure 2.4, $\mathsf{Img}(A, X_0) \subset \mathsf{Img}(\mathbf{A}, X_0)$ and $\mathsf{Inv}(A, X_0) \subset \mathsf{Inv}(\mathbf{A}, X_0)$. This is in line with the fact that A in (2.7) belongs to an interval matrix $\mathbf{A}$ in (2.16).

2.4 Summary

This chapter has discussed Max-Plus Linear (MPL) systems and some of their basic notions. We have then briefly described some related models such as Min-Plus Linear (MiPL) systems, Interval Max-Plus Linear (IMPL) systems, and Piece-wise Affine (PWA) systems. Following this, a procedure to generate PWA representations

from the other models has been presented. Finally, we have presented procedures to compute the image and inverse-image of Difference-Bound Matrices (DBM) w.r.t. the above systems.

Chapter 3

Abstractions of Max-Plus Linear Systems

This chapter discusses two procedures to generate finite-state abstractions of MPL systems. The first method (tropical abstraction) employs the max-plus algebraic operations ($\oplus$ and $\otimes$) to generate the abstract states. On the other hand, the second one (predicate abstraction) utilises a set of predicates chosen from the original system and possibly from specifications of interest. The former method is considered a specification-free technique, while the latter is a specification-induced method. Despite this difference, both techniques leverage the PWA representations of MPL dynamics and yield an abstract transition system that simulates the original MPL system.

Having obtained an abstract transition system, one can determine whether a specification is satisfied by model checking approaches. In this thesis, such a specification is expressed as a Linear Temporal Logic (LTL) formula. If it is satisfied, then the original MPL system also satisfies the specification. Otherwise, if the LTL formula is not satisfied, it does not imply the same conclusion for the underlying MPL system. In this case, a refinement procedure is implemented to obtain a more precise abstraction. We show that the procedures mentioned above can also be applied to MiPL and IMPL systems by modifying those developed for MPL systems.

A few elements contribute to the computational bottleneck (time and memory requirements) of these approaches. First, the PWA representations of MPL dynamics are prone to the state-explosion problem as the models' dimension increases. Second, the use of DBM as a data structure for representations and manipulation of abstract states might have heavy memory requirements, especially if there are a large number of abstract states. The overall performance of procedures is tested in a computational benchmark.

The discussion in this chapter is more about generating abstractions of MPL, MiPL, and IMPL systems. However, we present several examples of verification procedures on those models (see Examples 3.3-3.4 and Examples 3.6-3.7) to emphasise the benefit of such abstraction technique. A more thorough discussion about formal verification techniques will be presented in Chapter 6, where we also define the logic formulation for the specifications of interest. Moreover, Chapter 5 presents the abstraction-based procedures to solve reachability analysis for those models.

3.1 Related Work

The concept of abstractions for MPL systems has been firstly explored in [4] by leveraging the PWA representations of the MPL dynamics. Each PWA region corresponds to abstract states and is expressed as DBM. The transition relations among abstract states are then generated using one-step reachability analysis: first, the image of each abstract state w.r.t. the corresponding affine dynamics (2.18) is computed; then each image is intersected with all abstract states; finally, transition relations are defined for each non-empty intersection. This technique is adopted for PWA-based abstractions of IMPL systems in [27].

3.2 Preliminaries

This section describes some preliminary concept about transition systems, linear temporal logic and abstractions.

3.2.1 Transition Systems

In this subsection, we introduce transition systems, a standard class of models to describe the behaviour of systems, both software and hardware systems [11].

Definition 3.1 (Transition System [11, Definition 2.1])**.** A transition system TS is formulated by a tuple $(S, \longrightarrow, I, \mathcal{AP}, L)$ where

- S is a set of states,
- $\longrightarrow \subseteq S \times S$ is a transition relation,
- $I \subseteq S$ is a set of initial states[5],
- $\mathcal{AP}$ is a set of atomic propositions, and
- $L : S \to 2^{\mathcal{AP}}$ is a labeling function. □

This thesis follows the definition of transition systems in [11] except we do not use a set of actions that is usually used to describe the activity or mechanism in the transitions. In the transition system, transition relation reasons the behaviour of the states. For convenience, instead of (s, s'), we write $s \longrightarrow s'$. Atomic propositions express the characteristics of the states (i.e., state labels), while the labelling function relates each state to a set of atomic propositions that are satisfied by the state.

Definition 3.2 (Direct Predecessors and Direct Successors [11])**.**
Let $TS = (S, \longrightarrow, I, \mathcal{AP}, L)$ be a transition system. For $s \in S$, the set of direct successors and predecessor of s are respectively defined as

$$Post(s) = \{s' \in S \mid s \longrightarrow s'\} \text{ and } Pre(s) = \{s' \in S \mid s' \longrightarrow s\}.$$

A state s is called terminal if $Post(s) = \emptyset$. □

[5]The notation I does not represent a max-plus algebraic identity matrix, unless stated explicitly.

A transition systems $TS = (S, \longrightarrow, I, \mathcal{AP}, L)$ is called deterministic if $|I| \leq 1$ and $|Post(s)| \leq 1$ for all states $s \in S$ [11].

Definition 3.3 (Paths and Traces [11])**.** Let $TS = (S, \longrightarrow, I, \mathcal{AP}, L)$ be a transition system without terminal states. An infinite state sequence $\pi = s_0 s_1 \ldots$ is called path of transition system if $s_0 \in I$ and $s_i \in Post(s_{i-1})$ for all $i > 0$. $Paths(s)$ and $Paths(TS)$ are respectively the set of all paths start from s and the set of all paths in TS.

The trace of a path $\pi = s_0 s_1 \ldots$ is defined as $trace(\pi) = L(s_0)L(s_1)\ldots$. Furthermore, $Traces(s) = trace(Paths(s))$ and

$$Traces(TS) = \bigcup_{s \in I} Traces(s).$$

□

Given a path $\pi = s_0 s_1 \ldots$, the following notations are adopted for the sake of convenience. For $j \geq 0$, $\pi[j] = s_j$ denotes the j-th state of π and $\pi[..j]$ is the j-th prefix of π i.e., $\pi[..j] = s_0 s_1 \ldots s_j$. Likewise, the j-th suffix of π is $\pi[j..] = s_j s_{j+1} \ldots$. These notations are defined analogously for a finite path $\sigma = s_0 \ldots s_k$. Besides, we denote $\text{len}(\sigma) = k$ as the length of σ.

Following Definition 3.1, transition systems related to an MPL system (and also an IMPL system) are defined as follows.

Definition 3.4 (Trans. Sys. Associated with MPL Systems [4, Definition 3.8])**.** A transition system TS for an MPL system (2.6) characterised by $A \in \mathbb{R}_{\max}^{n \times n}$ is a tuple $(S, \longrightarrow, X, \mathcal{AP}, L)$ where

- the set of states S is $\mathbb{R}^n$,
- $(\mathbf{x}, \mathbf{x}') \in \longrightarrow$ if $\mathbf{x}' = A \otimes \mathbf{x}$,
- $X \subseteq \mathbb{R}^n$ is a set of initial conditions,
- $\mathcal{AP}$ is a set of atomic propositions,
- $L : S \to 2^{\mathcal{AP}}$ is a labeling function. □

Definition 3.5 (Trans. Sys. Associated with IMPL Systems)**.** A transition system TS for an IMPL system (2.14) characterised by an interval matrix $\mathbf{A} = [\underline{A}, \overline{A}] \in \mathbb{IR}_{\max}^{n \times n}$ is a tuple $(S, \longrightarrow, X, \mathcal{AP}, L)$ where

- the set of states S is $\mathbb{R}^n$,
- $(\mathbf{x}, \mathbf{x}') \in \longrightarrow$ if there exists $A \in \mathbf{A}$ such that $\mathbf{x}' = A \otimes \mathbf{x}$,
- $X \subseteq \mathbb{R}^n$ is a set of initial conditions,
- $\mathcal{AP}$ is a set of atomic propositions,
- $L : S \to 2^{\mathcal{AP}}$ is a labeling function. □

In this thesis, for transition systems in Definitions 3.4-3.5, we assume that the set of states satisfying each atomic proposition is a DBM: for each $p \in \mathcal{AP}$, the set of states $\{\mathbf{x} \mid p \in L(\mathbf{x})\}$ is a DBM. For the sake of simplicity, we denote such a set as $\mathtt{DBM}(p)$. For each $\mathbf{x} \in \mathbb{R}^n$, the set of direct successors of $Post(\mathbf{x})$ for a transition system associated with an MPL system (in Definition 3.4) always satisfies $|Post(\mathbf{x})| = 1$ since the image computation $A \otimes \mathbf{x}$ is uniquely defined for all $\mathbf{x} \in \mathbb{R}^n$. Hence, all states are not terminal. The set of paths $Paths(\mathbf{x})$ (resp. $Paths(TS)$) is indeed corresponding to the set of orbits $\mathsf{Orb}(A, \mathbf{x})$ (resp. $\mathsf{Orb}(A, X)$). However, for a transition system in Definition 3.5, we have $|Post(\mathbf{x})| \geq 1, Paths(\mathbf{x}) = \mathsf{Orb}(\mathbf{A}, \mathbf{x})$, and $Paths(TS) = \mathsf{Orb}(\mathbf{A}, X)$.

3.2.2 Linear Temporal Logic

Linear temporal logic (LTL) characterises logical formulae, which can be used for specifying properties over the set of atomic propositions [11]. Formally, LTL formulae are recursively defined as follows.

Definition 3.6 (Syntax of LTL [11, Definition 5.1])**.** LTL formulae over the set of atomic propositions $\mathcal{AP}$ are constructed according to the following grammar:

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

where $p \in \mathcal{AP}$. □

The symbol $\bigcirc$ (next) and U (until) are called temporal operators. Two additional operators, $\Diamond$ (eventually) and $\Box$ (always), are generated with use of until operator:

$$\Diamond\varphi = \mathtt{true}\ \mathsf{U}\ \varphi \quad \text{and} \quad \Box\varphi = \neg\Diamond\neg\varphi.$$

Throughout this thesis, we consider LTL formulae in release positive normal form (release PNF, or simply PNF) defined as follows:

Definition 3.7 (Positive Normal Form [11, Definition 5.23])**.** LTL formulae in release positive normal are given by

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

where $p \in \mathcal{AP}$. □

The R-modality is called *release until* operator, and is defined by $\varphi_1 \,\mathsf{R}\, \varphi_2 = \neg\varphi_1 \,\mathsf{U}\, \neg\varphi_2$. Any LTL formula φ in Definition 3.6 can be translated into a PNF formula φ' by "pushing the negations inside" with the following transformations:

$$\begin{array}{lcl}
\neg\mathtt{true} & \rightsquigarrow & \mathtt{false}, \\
\neg\neg\varphi & \rightsquigarrow & \varphi, \\
\neg(\varphi_1 \wedge \varphi_2) & \rightsquigarrow & \neg\varphi_1 \vee \neg\varphi_2, \\
\neg \bigcirc \varphi & \rightsquigarrow & \bigcirc\neg\varphi, \\
\neg(\varphi_1 \,\mathsf{U}\, \varphi_2) & \rightsquigarrow & \neg\varphi_1 \,\mathsf{R}\, \neg\varphi_2 \\
\neg(\Diamond\varphi) & \rightsquigarrow & \Box\neg\varphi.
\end{array}$$

Such translation can be done in a linear time w.r.t. the number of operators in φ [11, Theorem 5.24].

The semantics of LTL formulae are defined over all infinite words over the alphabet $2^{\mathcal{AP}}$ [11]. For $\sigma = \sigma_0\sigma_1 \ldots \in (2^{\mathcal{AP}})^\omega$, $\sigma[j..] = \sigma_j\sigma_{j+1}\ldots$ is the suffix of σ starting from an index j. The satisfaction relation for LTL formulae is defined as follows:

$$
\begin{array}{ll}
\sigma \models \texttt{true}, & \\
\sigma \not\models \texttt{false}, & \\
\sigma \models p \in \mathcal{AP} & \text{iff } p \in \sigma_0, \\
\sigma \models \neg p & \text{iff } \sigma \not\models p, \\
\sigma \models \varphi_1 \wedge \varphi_2 & \text{iff } \sigma \models \varphi_1 \wedge \sigma \models \varphi_2, \\
\sigma \models \varphi_1 \vee \varphi_2 & \text{iff } \sigma \models \varphi_1 \vee \sigma \models \varphi_2, \\
\sigma \models \bigcirc\varphi & \text{iff } \sigma[1..] = \sigma_1\sigma_2 \ldots \models \varphi, \\
\sigma \models \varphi_1 \,\mathsf{U}\, \varphi_2 & \text{iff } \exists j \geq 0 \text{ s.t. } \varphi[j..] \models \varphi_2 \text{ and} \\
 & \quad \sigma[i..] \models \varphi_1, \text{ for all } 0 \leq i < j, \\
\sigma \models \varphi_1 \,\mathsf{R}\, \varphi_2 & \text{iff } \forall j \geq 0 \text{ s.t. } \sigma[j..] \models \varphi_2 \text{ or} \\
 & \quad \exists i \geq 0.\ (\sigma[i..] \models \varphi_1 \wedge \forall k \leq i.\ \sigma[k..] \models \varphi_2), \\
\sigma \models \Diamond\varphi & \text{iff } \exists j \geq 0.\ \sigma[j..] \models \varphi, \\
\sigma \models \Box\varphi & \text{iff } \forall j \geq 0.\ \sigma[j..] \models \varphi.
\end{array}
$$

Given a transition system TS, the satisfaction relation of an LTL formula for a path π is induced by its trace; that is, $\pi \models \varphi$ iff $trace(\pi) \models \varphi$. Furthermore, $TS \models \varphi$ iff $\pi \models \varphi$ for all $\pi \in Paths(TS)$ [11].

3.2.3 Abstractions and Predicate Abstractions

Abstraction is a technique to generate a finite and smaller model from a large or even infinite (e.g., continuous-space) model. An abstraction is characterised by: 1) a set of abstract states $\widehat{S}$; 2) a function $f : S \to \widehat{S}$ that associates each (concrete) state $s \in S$ to the corresponding abstract state $f(s) \in \widehat{S}$; and 3) a set of atomic propositions that labels the concrete and abstract states. The mapping from S to $\widehat{S}$ is called *abstraction function*.

Definition 3.8 (Abstraction function [11, Definition 7.50])**.**
Let $TS = (S, \longrightarrow, I, \mathcal{AP}, L)$ be a transition system, and $\widehat{S}$ be a set of (abstract) states. $f : S \longrightarrow \widehat{S}$ is an abstraction function if for any $s, s' \in S : f(s) = f(s')$ implies $L(s) = L(s')$. □

Given a transition TS, an abstraction function produces a smaller transition TS_f. The states in TS and TS_f are called concrete and abstract states, respectively.

Definition 3.9 (Abstract transition system [11, Definition 7.51])**.**
Let $TS = (S, \longrightarrow, I, \mathcal{AP}, L)$ be a transition system, $\widehat{S}$ be a set of (abstract) states, and $f : S \longrightarrow \widehat{S}$ an abstraction function. The abstract transition system $TS_f = (\widehat{S}, \longrightarrow_f, I_f, \mathcal{AP}, L_f)$ induced by f on TS is defined by:

- $\longrightarrow_f$ is defined by the rule: $\frac{s \longrightarrow s'}{f(s) \longrightarrow_f f(s')}$

- $I_f = \{f(s) \mid s \in I\}$
- $L_f(f(s)) = L(s)$ for all states $s \in S$

□

The rule for transition relation in Definition 3.9 is usually called as *existential abstraction* [43]. It generates a transition from an abstract state if at least one corresponding concrete state has the transition. The dual of existential abstraction is *universal abstraction*: there is an (abstract) transition from an abstract state if all the corresponding concrete state have the (concrete) transition. There is a close connection between abstraction functions and partitions. For the abstraction function $f : S \to \widehat{S}$ in Definition 3.8, the set $\bigcup_{\widehat{s}\in\widehat{S}}\{s \mid f(s) = \widehat{s}\}$ is a partition of S.

Typically an abstract transition system simulates the concrete transition system. Simulation relations are used as a basis for abstraction techniques where the rough idea is to replace the model to be verified by a smaller (and possibly finite) abstract model and to verify the latter instead of the former one. The formal definition of simulation order is given below.

Definition 3.10 (Simulation order [11, Definition 7.47])**.**
Suppose $TS_i = (S_i, \longrightarrow_i, I_i, \mathcal{AP}, L_i)$ for $i = 1, 2$ be two transition systems over $\mathcal{AP}$. A simulation for (TS_1, TS_2) is a binary relation $\mathcal{R} \subseteq S_1 \times S_2$ such that

1. for each $s_1 \in I_1$ there exists $s_2 \in I_2$ such that $(s_1, s_2) \in \mathcal{R}$
2. for all $(s_1, s_2) \in \mathcal{R}$ it holds:
   (a) $L_1(s_1) = L_2(s_2)$
   (b) if $s_1' \in Post(s_1)$ then there exists $s_2' \in Post(s_2)$ with $(s_1', s_2') \in \mathcal{R}$

TS_1 is simulated by TS_2 (or, equivalently, TS_2 simulates TS_1) if there exists a simulation $\mathcal{R}$ for (TS_1, TS_2). □

Intuitively, TS_1 is simulated by TS_2 means that for every path in TS_1 there exists the corresponding path in TS_2 such that their traces coincide. Here, we refer to the paths and traces in Definition 3.3. The following proposition shows that if TS_f is an abstract transition system induced by abstraction function f on (concrete) transition system TS then TS is simulated by TS_f.

Proposition 3.1 ([11, Lemma 7.52])**.** Let $TS = (S, \longrightarrow, \mathcal{AP}, L)$ be a (concrete) transition system, $\widehat{S}$ a set of (abstract) states, and $f : S \to \widehat{S}$ an abstraction function. Then TS_f simulates TS. □

Proposition 3.2 ([11, Corollary 7.68 and Theorem 7.70])**.** Suppose TS_2 simulates TS_1 and TS_1 does not have any terminal state. For a linear-time property φ, if TS_2 satisfies φ then so is TS_1. □

Informally speaking, the linear-time property specifies the admissible (or desired) behaviour of the system under consideration [11]. Given a set of atomic proposition $\mathcal{AP}$, such properties can be expressed as a subset of $(2^{AP})^\omega$ [11, Definition 3.10].

Proposition 3.2 also applies when φ is an LTL formula since each LTL formula is a linear-time property [11]. However, if TS_2 does not satisfy φ, one cannot deduce that the same conclusion for TS_1 since the trace of the path that violates φ might be behaviours that do not exist in TS_1.

Definition 3.11 (Bisimulation equivalence [11, Definition 7.1])**.**
Suppose $TS_i = (S_i, \longrightarrow_i, I_i, \mathcal{AP}, L_i)$ for $i = 1, 2$ be two transition systems over $\mathcal{AP}$. A bisimulation for (TS_1, TS_2) is a binary relation $\mathcal{R} \subseteq S_1 \times S_2$ such that

1. for each $s_1 \in I_1$ there exists $s_2 \in I_2$ such that $(s_1, s_2) \in \mathcal{R}$ and for each $s_2 \in I_2$ there exists $s_1 \in I_1$ such that $(s_1, s_2) \in \mathcal{R}$
2. for all $(s_1, s_2) \in \mathcal{R}$ it holds:
   (a) $L_1(s_1) = L_2(s_2)$
   (b) if $s_1' \in Post(s_1)$ then there exists $s_2' \in Post(s_2)$ with $(s_1', s_2') \in \mathcal{R}$
   (c) if $s_2' \in Post(s_2)$ then there exists $s_1' \in Post(s_1)$ with $(s_1', s_2') \in \mathcal{R}$

TS_1 and TS_2 are bisimulation-equivalent (bisimilar, for short) if there exists a bisimulation $\mathcal{R}$ for (TS_1, TS_2). □

Intuitively, bisimilar relation between TS_1 and TS_2 means that TS_1 simulates TS_2 and also TS_2 simulates TS_1. Thus, it preserves all formulae that can be formulated in CTL* [11, Definition 6.80] (which contain in particular LTL). In other words, for any LTL formula φ, TS_1 satisfies φ iff TS_2 satisfies φ.

The bisimulation-quontienting algorithms [11, Section 7.3] can be used to obtain a bisimulation quotient transition system if the concrete transition system is finite. However, if the concrete transition system is infinite, then the termination of the algorithms is not guaranteed.

Predicate Abstractions

Predicate abstractions [37, 43, 48, 62] denote abstraction methods that use a set of *predicates* $P = \{p_1, \ldots, p_k\}$ to characterise the abstract states. Such predicates are identified from the concrete model and possibly from the specifications under consideration. Each predicate is used to evaluate the concrete model: a predicate is either satisfied or not satisfied by a concrete state. Therefore, we obtain that $|\widehat{S}| \leq 2^k$. An abstract state will be labelled with predicate p_i if it is `true` in that state. For this reason, predicates also serve as atomic propositions [37].

The predicates are also used to define an abstraction function between the concrete and abstract state spaces. The abstraction function for predicate abstractions is defined as

$$f(s) = \bigwedge_{i=1}^{k} \mathrm{val}(s, p_i), \tag{3.1}$$

where $\mathrm{val}(s, p_i) = p_i$ if p_i is satisfied in s, otherwise $\neg p_i$. Notice that, the abstraction function in (3.1) also ensures that $\bigcup_{\widehat{s} \in \widehat{S}} \{s \in S \mid f(s) = \widehat{s}\}$ is a partition of S. The labeling function can be simply defined from the predicates that are true on each abstract states i.e., $p \in L_f(\widehat{s})$ if p is true on $\widehat{s}$.

The common problem for predicate abstraction is *predicate redundancy*. In detail, a predicate p_i is redundant if there are two propositional formulas in terms of other predicates that are logically equivalent to p_i and $\neg p_i$, respectively [37]. In that case, any redundant predicate needs to be removed. The preventive approach has been proposed in [30] by using the minimum number of predicates.

3.3 Tropical Abstractions of Max-Plus Linear Systems

This section discusses the novel technique to generate the abstraction of MPL systems by employing max-plus algebraic operations $\oplus$ and $\otimes$. The name "tropical" is inspired by the fact that max-plus algebra is also known as tropical algebra. The method is a modification of the existing technique in [4] which leverages an MPL system into an equivalent PWA system.

3.3.1 States: Partitioning Procedure

We start by recalling that the PWA regions in (2.17) is a DBM in $\mathbb{R}^n$, and each DBM can be expressed as a max-plus algebraic matrix. Considering the bound matrix of the DBM, the following proposition shows the generation of the PWA region using max-plus algebraic operations.

Proposition 3.3 ([73, Proposition 5])**.** Given an MPL system (2.6), the PWA region (2.17) w.r.t. a finite coefficient $\mathbf{g} = (g_0, g_1, \ldots, g_n)$ can be expressed as

$$\mathsf{R}_{\mathbf{g}} = [A_{\text{ext}}]^{\mathsf{c}}_{\mathbf{g}} \otimes A_{\text{ext}} \oplus I_{n+1}. \tag{3.2}$$

Proof. Notice that, the value of $A_{\text{ext}}(i,j) - A_{\text{ext}}(i,g_i)$ in (2.17) corresponds to $\mathsf{R}_{\mathbf{g}}(g_i, j)$. Furthermore, we have $[[A_{\text{ext}}]^{\mathsf{c}}_{\mathbf{g}} \otimes A_{\text{ext}}](g_i,j) = \bigoplus_{k=0}^{n}([A_{\text{ext}}]^{\mathsf{c}}_{\mathbf{g}}(g_i,k) + A_{\text{ext}}(k,j))$. Since $[A_{\text{ext}}]^{\mathsf{c}}_{\mathbf{g}}(g_i,k) \neq \varepsilon$ iff $k = i$ then,

$$\begin{aligned}[[A_{\text{ext}}]^{\mathsf{c}}_{\mathbf{g}} \otimes A_{\text{ext}}](g_i,j) &= [A_{\text{ext}}]^{\mathsf{c}}_{\mathbf{g}}(g_i,i) + A_{\text{ext}}(i,j) \\ &= -A_{\text{ext}}(i,g_i) + A_{\text{ext}}(i,j)\end{aligned}$$

As a result, $\mathsf{R}_{\mathbf{g}}(g_i,j) = [[A_{\text{ext}}]^{\mathsf{c}}_{\mathbf{g}} \otimes A_{\text{ext}}](g_i,j)$ for all $i,j \in \{0,\ldots,n\}$. However, for $g_i = j$, it is possible that $[[A_{\text{ext}}]^{\mathsf{c}}_{\mathbf{g}} \otimes A_{\text{ext}}](g_i,j) < 0$. Therefore, the final expression is $\mathsf{R}_{\mathbf{g}} = [A_{\text{ext}}]^{\mathsf{c}}_{\mathbf{g}} \otimes A_{\text{ext}} \oplus I_{n+1}$. □

Recall that A_{ext} is the extension of matrix A (see Remark 2.2). Furthermore, the first matrix on the right hand side of (3.2), $[A_{\text{ext}}]^{\mathsf{c}}_{\mathbf{g}}$, corresponds to region matrix (2.4) and conjugate operation (2.5). Notice that, the resulting PWA regions are not a partition of $\mathbb{R}^n$ since it is possible that for two different finite coefficients $\mathbf{g}, \mathbf{h}$ we have $\mathsf{R}_{\mathbf{g}} \cap \mathsf{R}_{\mathbf{h}} \neq \emptyset$. The non-empty intersection is found on adjacent regions defined as follows.

Definition 3.12 (Adjacent Regions [4, Definition 3.10])**.** Two non-empty regions generated by (2.17) $\mathsf{R}_{\mathbf{g}}$ and $\mathsf{R}_{\mathbf{h}}$ are called adjacent, denoted by $\mathsf{R}_{\mathbf{g}} > \mathsf{R}_{\mathbf{h}}$, if there exists a single $i \in \{1,\ldots,n\}$ such that $g_i > h_i$ and $g_j = h_j$ for each $j \neq i$. □

To obtain an abstraction of an MPL system, we need to generate a partition from PWA regions (2.17). The procedure in [4] proposes to remove the intersection from the region with a lower finite coefficient. It has been shown in [4] that removing the intersection from adjacent regions still yields a DBM and the worst-case complexity to remove non-empty intersection from all adjacent regions is $\mathcal{O}(n^{2n+1})$.

Proposition 3.4 ([4, Proposition 3])**.** If $\mathsf{R}_{\mathbf{g}} > \mathsf{R}_{\mathbf{h}}$ then $\mathsf{R}_{\mathbf{h}} \setminus \mathsf{R}_{\mathbf{g}} = \mathsf{R}_{\mathbf{h}} \cap \{\mathbf{x} \in \mathbb{R}^n \mid A(i,h_i) + x_{h_i} > A(i,g_i) + x_{g_1}\}$, which is a DBM. □

Example 3.1. Consider the resulting PWA regions in Figure 2.3 (left). We have $\mathsf{R}_{(0,2,1)} > \mathsf{R}_{(0,1,1)}$ and $\mathsf{R}_{(0,2,2)} > \mathsf{R}_{(0,2,1)}$. By applying Proposition 3.4, the new regions are $\mathsf{R}'_{(0,1,1)} = \{\mathbf{x} \mid \mathbb{R}^2 \mid x_1 - x_2 > 3\}$, $\mathsf{R}'_{(0,2,1)} = \{\mathbf{x} \mid \mathbb{R}^2 \mid 0 < x_1 - x_2 \leq 3\}$, and $\mathsf{R}'_{(0,2,2)} = \{\mathbf{x} \mid \mathbb{R}^2 \mid x_1 - x_2 \leq 0\}$. Notice that $\mathsf{R}'_{(0,2,2)} = \mathsf{R}_{(0,2,2)}$. □

Instead of removing the intersection of adjacent regions, as shown in [73], the partition of PWA regions can be established by choosing the sign matrix for $\mathsf{R}_{\mathbf{g}}$ (let us denoted by $\mathsf{S}_{\mathbf{g}}$). As we can see in (2.17), all operators are $\geq$. Thus, $\mathsf{S}_{\mathbf{g}}(i,j) = 1$ for all $i,j \in \{0,1,\ldots,n\}$. In this thesis, we use a rule to decide the sign matrix of $\mathsf{R}_{\mathbf{g}}$ as follows

$$\mathsf{S}_{\mathbf{g}}(i,j) = \begin{cases} 1 & \text{if } \mathsf{R}_{\mathbf{g}}(i,j) \geq 0, \\ 0 & \text{if } \mathsf{R}_{\mathbf{g}}(i,j) < 0. \end{cases} \tag{3.3}$$

It is straightforward to see that this rule guarantees an empty intersection for each pair of adjacent regions.

Algorithm 3.1 presents the steps to generate the PWA representations from an MPL system using max-plus algebraic operations, $\oplus$ and $\otimes$. Its worst-case complexity is $O(n^{n+3})$ since the emptiness checking in line 11 has cubic complexity, and there are n^n iterations at line 5. However, we do not expect to incur this worst-case complexity, especially when matrix A has several ε elements in each row. In general, if there are exactly m finite elements at each row of A, then the complexity will be $\mathcal{O}(m^n n^3)$.

Algorithm 3.1 Generating PWA representations of an MPL system via tropical operations

```
1:  function MAXPLUS_TROP_ABS(A)
2:      R ← ∅
3:      n ← ROW(A)
4:      A ← EXTENT(A)
5:      for g ∈ {1,...,n}^n do
6:          g ← EXTENT(g)
7:          if g is a finite coefficient of A then
8:              generate [A_g]^c according to (2.4) and (2.5)
9:              R_g ← [A_g]^c ⊗ A ⊕ I_{n+1}
10:             generate S_g according to (3.3)        ▷ a sign matrix for R_g
11:             if (R_g, S_g) is not empty then        ▷ emptiness checking of a DBM
12:                 R ← R ∪ {R_g, S_g, g}
13:     return R
```

Example 3.2. Let us consider again the PWA region in Figure 2.3 (left). By applying a rule (3.3) and using Algorithm 3.1, the resulting (new) regions are $\mathsf{R}'_{(1,1)} = \{\mathbf{x} \mid \mathbb{R}^2 \mid x_1 - x_2 \geq 3\}$, $\mathsf{R}'_{(2,1)} = \{\mathbf{x} \mid \mathbb{R}^2 \mid 0 \leq x_1 - x_2 < 3\}$, and $\mathsf{R}'_{(2,2)} = \{\mathbf{x} \mid \mathbb{R}^2 \mid x_1 - x_2 < 0\}$. □

Finally, after generating the partition from PWA regions, we can define the abstraction function f and the labeling function L_f. Suppose for a partition in Example 3.2 we have $AP = \{p\}$ where $\texttt{DBM}(p) = \{\mathbf{x} \in \mathbb{R}^2 \mid x_1 - x_2 \geq 0\}$. The abstraction and labeling function is respectively defined as follows

$$f(\mathbf{x}) = \begin{cases} \widehat{s}_1, & \text{if } x_1 - x_2 \geq 3, \\ \widehat{s}_2, & \text{if } 0 \leq x_1 - x_2 < 3, \\ \widehat{s}_3, & \text{if } x_1 - x_2 < 0, \end{cases} \qquad L_f(\widehat{s}_i) = \begin{cases} \{p\}, & \text{if } i \in \{1, 2\}, \\ \emptyset, & \text{if } i = 3. \end{cases} \tag{3.4}$$

Thus, the abstraction of an MPL system in (2.7) results in three abstract states. Each abstract state corresponds to a PWA region and affine dynamics.

Remark 3.1. The partitions in Example 3.1 and Example 3.2 are indeed different as the former one inspects the finite coefficient of the adjacent regions while the latter one looks at the elements of the bound matrix. This condition does not affect the correctness of the abstraction but may affect the choice of a set of atomic propositions. For instance, if we use partition in Example 3.1 then we have

$$f(\mathbf{x}) = \begin{cases} \widehat{s}_1, & \text{if } x_1 - x_2 > 3, \\ \widehat{s}_2, & \text{if } 0 < x_1 - x_2 \leq 3, \\ \widehat{s}_3, & \text{if } x_1 - x_2 \leq 0. \end{cases}$$

An atomic proposition p with $\texttt{DBM}(p) = \{\mathbf{x} \in \mathbb{R}^2 \mid x_1 - x_2 \geq 0\}$ cannot be used since it is possible that there are $\mathbf{x}, \mathbf{y} \in \mathbb{R}^2$ such that $f(\mathbf{x}) = f(\mathbf{y}) = \widehat{s}_3$ and p is only satisfied by one of them (or in other words, $L(\mathbf{x}) \neq L(\mathbf{y})$). This violates the definition of abstraction function in Definition 3.8. If we chose $\texttt{DBM}(p) = \{\mathbf{x} \in \mathbb{R}^2 \mid x_1 - x_2 > 0\}$ instead, then such violation disappears. The main point here is that the choice of an atomic proposition is dependent on the resulting partition. □

Notice that the procedure in Algorithm 3.1 is a *specification-free* method since it does not depend on specifications nor the given set of atomic propositions. As such, one can only determine the specifications or atomic propositions after generating the abstract states.

3.3.2 Transitions: One-Step Reachability

This subsection describes the technique to determine the transition relations within the abstract states obtained in Subsection 3.3.1. From the abstraction function f and an abstract state $\widehat{s}$, the corresponding concrete state can be achieved from $f^{-1}(\widehat{s}) = \{\mathbf{x} \mid f(\mathbf{x}) = \widehat{s}\}$. Recall that $f^{-1}(\widehat{s})$ is a DBM which has a specific affine dynamics. For instance, from abstraction function (3.4), $f^{-1}(\widehat{s}_1) = \{\mathbf{x} \in \mathbb{R}^2 \mid x_1 - x_2 \geq 3\}$ and its corresponding affine dynamics is given by

$$\begin{bmatrix} x_1' \\ x_2' \end{bmatrix} = \begin{bmatrix} x_1 + 2 \\ x_1 + 3 \end{bmatrix}.$$

By *existential abstraction* rule in Definition 3.9, if there exists a transition from $\mathbf{x}$ to $\mathbf{x}'$, i.e., $\mathbf{x} \longrightarrow \mathbf{x}'$, in the concrete transition system (see Definition 3.4) then there exists a transition from $f(\mathbf{x}) = \widehat{s}$ to $f(\mathbf{x}') = \widehat{s}'$ in the abstract transition system, i.e.,

$\widehat{s} \longrightarrow_f \widehat{s}'$. The former transition can be determined by checking the non-emptiness of $Post(f^{-1}(\widehat{s})) \cap f^{-1}(\widehat{s}')$. It is straightforward that

$$Post(f^{-1}(\widehat{s})) = \{A \otimes \mathbf{x} \mid \mathbf{x} \in f^{-1}(\widehat{s})\}.$$

Since $f^{-1}(\widehat{s})$ is a DBM and there is only one active affine dynamics in $f^{-1}(\widehat{s})$, $Post(f^{-1}(\widehat{s}))$ is a DBM (see Proposition 2.10). Furthermore, the computation of $Post(f^{-1}(\widehat{s}))$ can be done by Algorithm 2.5. This procedure is called *one-step reachability analysis*. The complete algorithm to generate the abstract transition is given in [2, Algorithm 3.3]. The worst-case complexity of the algorithm is in $\mathcal{O}(n^3|\widehat{S}|^2)$ since the emptiness checking of DBM is in cubic time and there are $|\widehat{S}|^2$ pairs of abstract states.

Example 3.3. For the abstract states in (3.4), we have $f^{-1}(\widehat{s}_1) = \{\mathbf{x} \in \mathbb{R}^2 \mid x_1 - x_2 \geq 3\}$, $f^{-1}(\widehat{s}_2) = \{\mathbf{x} \in \mathbb{R}^2 \mid 0 \leq x_1 - x_2 < 3\}$, and $f^{-1}(\widehat{s}_3) = \{\mathbf{x} \in \mathbb{R}^2 \mid x_1 - x_2 < 0\}$. Furthermore, let the details aside, we have $Post(f^{-1}(\widehat{s}_1)) = \{\mathbf{x} \in \mathbb{R}^2 \mid x_1 - x_2 = -1\}$, $Post(f^{-1}(\widehat{s}_2)) = \{\mathbf{x} \in \mathbb{R}^2 \mid -1 < x_1 - x_2 \leq 2\}$, and $Post(f^{-1}(\widehat{s}_3)) = \{\mathbf{x} \in \mathbb{R}^2 \mid x_1 - x_2 = 2\}$. The intersection $Post(f^{-1}(\widehat{s}_i)) \cap f^{-1}(\widehat{s}_j)$ is not empty for $(i, j) \in \{(1, 3), (2, 2), (2, 3), (3, 2)\}$. The abstract transition system in shown in Figure 3.1 where we consider all abstract states are initial.

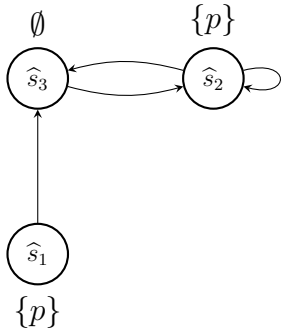


Figure 3.1: Abstract transition system generated from an MPL system in (2.7). All abstract states are initial.

Suppose we want to verify two specifications $\Diamond p$ and $\Diamond\Box p$. The first specification is satisfied by the abstract transition system in Figure 3.1. Thus the original MPL system (2.7) also satisfies $\Diamond p$. On the other hand, the second one does not hold. However, this does not imply that $\Diamond\Box p$ is not satisfied by the concrete MPL system in (2.7). □

We recall that the abstract transition system simulates the original MPL system. Thus, it makes sense to attempt deriving an abstract transition system that bisimulates the MPL system. The following theorem implies that the abstract transition system bisimulates the concrete transition system iff there is one outgoing transition from each abstract state.

Theorem 3.1 ([2, Theorem 3.2])**.** Let TS be the concrete transition system generated by an MPL system and TS_f be the abstract transition system induced by an abstraction function $f : S \to \widehat{S}$. The binary relation $\mathcal{R} = \{(f(\mathbf{x}), \mathbf{x}) \mid \mathbf{x} \in S\}$ is a simulation for (TS_f, TS) iff $|Post(\widehat{s})| = 1$ for all $\widehat{s} \in \widehat{S}$. □

We recall that, in concrete transition system TS associated with an MPL system, $|Post(\mathbf{x})| = 1$ for each $\mathbf{x} \in S$. For $\mathcal{R} = \{(f(\mathbf{x}), \mathbf{x}) \mid \mathbf{x} \in S\}$ to be a simulation order, by Condition 2(b) in Definition 3.10, it must be the case that $|Post(f(\mathbf{x}))| = 1$. This condition justifies the proof for Theorem 3.1.

The procedure to obtain an abstract transition system that bisimulates the concrete transition system works as follows. For each abstract state $\widehat{s}$ with $|Post(\widehat{s})| > 1$, the corresponding concrete state $f^{-1}(\widehat{s})$ is refined (or, partitioned) according to the direct successors of $\widehat{s}$. Then the incoming and outgoing transitions are updated. The preceding steps are repeated until all abstract states have one outgoing transition. Let us now describe the refinement step of the procedure. Suppose that an abstract state $\widehat{s}$ has more than one outgoing transitions. The refinement step generates a partition of $f^{-1}(\widehat{s})$ w.r.t. the direct successors of $\widehat{s}$. More precisely for each $\widehat{s}' \in Post(\widehat{s})$, we define a partition of $f^{-1}(\widehat{s})$ such that its direct successors is in $f^{-1}(\widehat{s}')$, i.e., $\{s \in f^{-1}(\widehat{s}) \mid f(Post(\mathbf{x})) = \widehat{s}'\} = f^{-1}(\widehat{s}) \cap Pre(f^{-1}(\widehat{s}')$. This can be done by computing the inverse image of $f^{-1}(\widehat{s}')$ w.r.t. the affine dynamics that are active in $f^{-1}(\widehat{s})$ via Algorithm 2.6. Again, this computation yields a DBM (see Proposition 2.11).

Unfortunately, such a refinement procedure, in general, does not necessarily terminate, especially for abstract transition systems generated from high-dimensional MPL systems; moreover, for reducible ones. There are several known sufficient conditions in [2, Propositions 3.8-3.10] for the existence of an abstract transition that bisimulates the concrete transition system. Some of them require that the state matrix of the underlying MPL system is irreducible and has a cyclicity of 1.

Example 3.4. Let us apply the refinement procedure to the abstract transition in Figure 3.1. The abstract state $\widehat{s}_2$ has two outgoing transitions. The corresponding concrete state $f^{-1}(\widehat{s}_2)$ is partitioned into two sets $\{\mathbf{x} \in \mathbb{R}^2 \mid 2 < x_1 - x_2 < 3\}$ and $\{\mathbf{x} \in \mathbb{R}^2 \mid 0 \leq x_1 - x_2 \leq 2\}$. Next we characterised the (new) abstract transition system associated with the refined partition. The new abstraction and labeling functions are defined as follows:

$$f(\mathbf{x}) = \begin{cases} \widehat{r}_1, & \text{if } x_1 - x_2 \geq 3, \\ \widehat{r}_2, & \text{if } 2 < x_1 - x_2 < 3, \\ \widehat{r}_3, & \text{if } 0 \leq x_1 - x_2 \leq 2, \\ \widehat{r}_4, & \text{if } x_1 - x_2 < 0, \end{cases} \qquad L_f(\widehat{r}_i) = \begin{cases} \{p\}, & \text{if } i \in \{1,2,3\}, \\ \emptyset, & \text{if } i = 4. \end{cases}$$

The new abstract transition system is depicted in Figure 3.2 and it bisimulates the concrete transition system since all abstract states have one outgoing transition. Regarding the specification $\Diamond\Box p$, it is satisfied by the abstract transition. Therefore, the original MPL system also satisfies $\Diamond\Box p$. □

3.4 Predicate Abstractions of Max-Plus Linear Systems

This section presents an alternative method to generate the abstraction of MPL systems via a set of predicates. Such predicates can be determined from the MPL system as well as the underlying specifications.

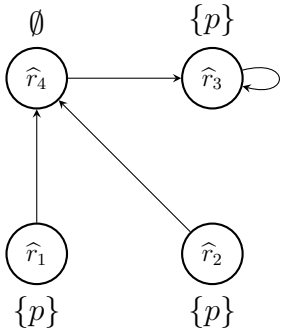


Figure 3.2: Refined abstract transition system generated from an MPL system in (2.7). All abstract states are initial.

3.4.1 States: Choosing the Predicates

The first issue for predicate abstraction is to find the appropriate predicates. Recall that the related abstraction techniques [4, 73] explore the connection between MPL and PWA systems and use DBM to represent the abstract states. Therefore, the predicates here are chosen such that the dynamics in the resulting abstract states are affine as in (2.17) and can be expressed as DBM. Following these considerations, the predicates are defined as an inequality $x_i - x_j \sim \alpha$ where $\sim \in \{>, \geq\}, \alpha \in \mathbb{R}$. From the PWA region in (2.17), the scalar α 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 $x_i - x_j \sim A(k,j) - A(k,i)$.

As mentioned before, a predicate (which also serves as a proposition) can also be chosen from the specifications of interest. Since the orbits of an MPL system represent the time-stamps of the discrete events, it is admissible to consider the delays occurring between events as atomic propositions. Again, this type of atomic propositions can be expressed as an inequality $x_i - x_j \sim \alpha$. In Section 3.3 (see Examples 3.3-3.4), the atomic propositions are chosen after we generate the tropical abstractions. However, in predicate abstractions, one can define them in advance. For instance, a specification "the absolute difference between i-th and j-th events never exceeds 3 time unit" can be expressed as $\Box(p \wedge q)$ where $p = x_i - x_j \geq -3$ and $q = x_j - x_i \geq -3$. Consequently, unlike the (*specification-free*) tropical abstraction approach, the predicate abstraction presented in this section is a *specification-induced* method.

Remark 3.2. In Subsection 6.2.2, we formally define the specifications of interest over the LTL formulation, where one can express more sophisticated delay requirements such as "the delay between two consecutive i-th events is never exceeds five time-unit". Such a specification could be expressed as $\Box(x_i - x_i' \geq -5)$. Unfortunately, we cannot consider $x_i - x_i' \geq -5$ as a predicate due to the primed variable x_i'. In the same subsection, we will describe the steps to obtain the corresponding predicates by utilising max-plus algebraic manipulations. □

Algorithm 3.2 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 $k_1 \neq k_2$.

Algorithm 3.2 Generating a set of predicates from an MPL system

1: **function** MPL2PRED(A, k) ▷ generating a set predicates from the k^{th} row of A
2: $P_k \leftarrow \emptyset$
3: **for** $j \in \{2, \ldots, n\}$ **do**
4: **for** $i \in \{1, \ldots, j-1\}$ **do**
5: **if** $A(k,i) \neq \varepsilon$ and $A(k,j) \neq \varepsilon$ **then**
6: $P_k \leftarrow P_k \cup \{x_i - x_j \geq A(k,j) - A(k,i)\}$ ▷ we pick $\sim = \geq$
7: **return** P_k
8: **function** MPL2PRED(A) ▷ generating a set of predicates from matrix A
9: $P \leftarrow \emptyset$
10: **for** $k \in \{1, \ldots, n\}$ **do**
11: $P \leftarrow P \cup$ MPL2PRED(A, k)
12: **return** P

After obtaining the set of predicates, the next step is generating the abstract states. Let $\widehat{S}$ be a set of abstract states defined over Boolean assignments of $P = \{p_1, \ldots, p_k\}$. Hence, we have $|\widehat{S}| \leq 2^k$. For a predicate p_i, we define the corresponding DBM as follows: $\texttt{DBM}(p_i) = \{\mathbf{x} \in \mathbb{R}^n \mid p_i \text{ is satisfied by } \mathbf{x}\}$ and $\texttt{DBM}(\neg p_i) = \{\mathbf{x} \in \mathbb{R}^n \mid p_i \text{ is not satisfied by } \mathbf{x}\}$. It is straightforward that $\texttt{DBM}(p_i \wedge p_j) = \texttt{DBM}(p_i) \cap \texttt{DBM}(p_j)$.

Algorithm 3.3 shows the steps to generate the abstract states of an MPL system given a set of predicates P. At the start, we initialise $\widehat{S} = \{\texttt{true}\}$ and the partition $\mathbf{R}$ as the whole state-space of MPL system i.e., $\mathbb{R}^n$. For each $p \in P$, we manipulate the Boolean assignments of p (line 5) and their corresponding partitions (lines 6-8). In lines 9-12, we only collect the (partial) assignments whose corresponding DBM are not empty.

Algorithm 3.3 Generating an abstraction of MPL system from a set of predicates

Input: P, a set of predicates
Outputs: $\widehat{S}$, a set of abstract states
$\mathbf{R}$, a partition of $\mathbb{R}^n$ w.r.t. $\widehat{S}$ ▷ $\mathbf{R}$ is a set of DBM
1: **function** PRED_ABS(P)
2: $\mathbf{R} \leftarrow \{\mathbb{R}^n\}$
3: $\widehat{S} \leftarrow \{\texttt{true}\}$
4: **for** $p \in P$ **do**
5: $\widehat{S} \leftarrow \bigcup_{\widehat{s} \in \widehat{S}} \{\widehat{s} \wedge \neg p\} \cup \bigcup_{\widehat{s} \in \widehat{S}} \{\widehat{s} \wedge p\}$
6: $\mathbf{R}_{neg} \leftarrow \bigcup_{D \in \mathbf{R}} \{D \cap \texttt{DBM}(\neg p)\}$ ▷ each DBM in $\mathbf{R}$ is intersected with $\texttt{DBM}(\neg p)$
7: $\mathbf{R}_{pos} \leftarrow \bigcup_{D \in \mathbf{R}} \{D \cap \texttt{DBM}(p_i)\}$ ▷ both $\mathbf{R}_{neg}$ and $\mathbf{R}_{pos}$ are set of DBM
8: $\mathbf{R} \leftarrow \mathbf{R}_{neg} \cup \mathbf{R}_{pos}$
9: **for** $j \in \{1, \ldots, |\mathbf{R}|\}$ **do**
10: **if** $\mathbf{R}[j]$ is empty **then** ▷ DBM emptiness check
11: **remove** $\mathbf{R}[j]$ from $\mathbf{R}$
12: **remove** $\widehat{S}[j]$ from $\widehat{S}$
13: **return** $(\mathbf{R}, \widehat{S})$

The worst-case complexity of Algorithm 3.3 is $\mathcal{O}(2^{|P|}n^3)$. If there are exactly $m > 1$ finite elements at each row of A then its worst-case complexity is $\mathcal{O}(2^{n\binom{m}{2}}n^3)$. Indeed, for $2 < m \leq n$, the predicate abstraction is less efficient than PWA-based abstraction whose complexity is $\mathcal{O}(m^n n^3)$. In case $m = 2$, both procedures have the same worst-case complexity. In this thesis, we call a predicate p_i is *logically similar* to p_j if either $p_i = p_j$ or $p_i = \neg p_j$. In this situation, we can simply remove one of them. Two logically similar predicates can be found from the state matrix as well as from the specifications.

Example 3.5. Let us consider again an MPL system in (2.7). Suppose we want to verify two specifications $\Diamond\Box(0 \leq x_1 - x_2 \leq 2)$ and $\Diamond(x_1 - x_2 = 1)$. From these specifications, we get four predicates $p_1 = x_1 - x_2 \geq 0, p_2 = x_2 - x_1 \geq -2, p_3 = x_1 - x_2 \geq 1$, and $p_4 = x_2 - x_1 \geq -1$. By Algorithm 3.2, we obtain additional two predicates from the state matrix, namely $p_5 = x_1 - x_2 \geq 3$ and $p_6 = x_1 - x_2 \geq 0$. Notice that p_6 is logically similar to p_1. Hence, we remove p_6. Out of 2^5 possible abstract states, from these five predicates, only six of them are non-empty e.g.,

$$\begin{array}{lll} \widehat{s}_1 = \neg p_1 p_2 \neg p_3 p_4 \neg p_5, & \widehat{s}_2 = p_1 p_2 \neg p_3 p_4 \neg p_5, & \widehat{s}_3 = p_1 p_2 p_3 p_4 \neg p_5, \\ \widehat{s}_4 = p_1 p_2 p_3 \neg p_4 \neg p_5, & \widehat{s}_5 = p_1 \neg p_2 p_3 \neg p_4 \neg p_5, & \widehat{s}_6 = p_1 \neg p_2 p_3 \neg p_4 p_5. \end{array}$$

The corresponding DBM for each abstract state is

$$\begin{array}{ll} \texttt{DBM}(\widehat{s}_1) = \{\mathbf{x} \in \mathbb{R}^2 \mid x_1 - x_2 < 0\}, & \texttt{DBM}(\widehat{s}_2) = \{\mathbf{x} \in \mathbb{R}^2 \mid 0 \leq x_1 - x_2 < 1\}, \\ \texttt{DBM}(\widehat{s}_3) = \{\mathbf{x} \in \mathbb{R}^2 \mid x_1 - x_2 = 1\}, & \texttt{DBM}(\widehat{s}_4) = \{\mathbf{x} \in \mathbb{R}^2 \mid 1 < x_1 - x_2 \leq 2\}, \\ \texttt{DBM}(\widehat{s}_5) = \{\mathbf{x} \in \mathbb{R}^2 \mid 2 < x_1 - x_2 < 3\}, & \texttt{DBM}(\widehat{s}_6) = \{\mathbf{x} \in \mathbb{R}^2 \mid x_1 - x_2 \geq 3\}. \end{array}$$

□

3.4.2 Transitions: Finding the Affine Dynamics

Having obtained the abstract states, one needs to generate the abstract transitions, which can be obtained via one-step reachability as described in Subsection 3.3.2. However, unlike Algorithm 3.1, Algorithm 3.3 does not produce the affine dynamics for each abstract state. For each $\widehat{s} \in \widehat{S}$, we need to find $\mathbf{g} = (g_1, \ldots, g_n)$ which characterise the maximum term of $x'_k = \max\{A(k,1) + x_1, \ldots, A(k,n) + x_n\}$ for $k \in \{1, \ldots, n\}$. Given a predicate $p = x_i - x_j \sim \alpha$, we call i and j as the left and right index of p (since $x_i \sim x_j + c$) and denote them by $\texttt{left}(p)$ and $\texttt{right}(p)$, respectively. Notice that $\texttt{left}(p) = \texttt{right}(\neg p)$.

Suppose $p = x_i - x_j \geq A(k,j) - A(k,i) \in P_k$ is a predicate generated by MPL2PRED(A,k) in Algorithm 3.2. If p is true on abstract state $\widehat{s}$ then we have $x_i + A(k,i) \geq x_j + A(k,j)$. This means that $\texttt{right}(p) = j$ cannot characterise the maximum term for $\max\{A(k,1) + x_1, \ldots, A(k,n) + x_n\}$. On the other hand, if p is false on $\widehat{s}$ then $\texttt{left}(p) = i$ is not the maximum term. Algorithm 3.4 illustrates the steps to generate the affine dynamics from an abstract states. The set idx contains the indexes of finite elements of $A(k,\cdot)$. It will be updated according to the truth value of p on $\widehat{s}$ in lines 6-9. At the end, the maximum term is the only remaining element in idx.

Algorithm 3.4 Generation of the affine dynamics for an abstract state

Inputs: $A \in \mathbb{R}_{\max}^{n\times n}$, a m-regular matrix with $m > 1$
$\widehat{s} \in \widehat{S}$, an abstract state
$P_1, \dots, P_n$, sets of predicates generated by Algorithm 3.2
Output: $\mathbf{g} = (g_0, \dots, g_n)$, the finite coefficient representing the affine dynamics for $\widehat{s}$

```
1: function GET_MAXPLUS_AFFINE(A, ŝ, P_1, ..., P_n)
2:     g ← zeros(1, n + 1)                          ▷ g_0 is always 0
3:     for k ∈ {1, ..., n} do
4:         idx ← Find(A(k, ·) ≠ ε)
5:         for p ∈ P_k do
6:             if p is true in ŝ then
7:                 idx ← idx \ {right(p)}
8:             else
9:                 idx ← idx \ {left(p)}
10:        g_k ← idx[0]
11:    return g
```

Example 3.6. With the preceding abstract states in Example 3.5, the corresponding finite coefficient for $\widehat{s}_1, \widehat{s}_6$ is $(0, 2, 2)$ and $(0, 1, 1)$, respectively. The other abstract states yield finite coefficient $(0, 2, 1)$. Leaving the details aside, the resulting abstract transition system is depicted in Figure 3.3

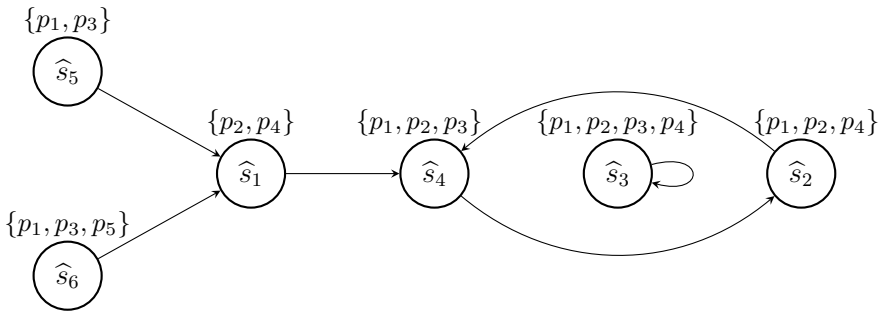


Figure 3.3: An abstract transition system obtained from predicate abstraction of an MPL system (2.7). All abstract states are initial.

Coincidentally, the abstract transition system bisimulates the original MPL system in (2.7) since there is only one outgoing transition from each abstract state. Regarding the specifications $\Diamond\Box(0 \le x_1 - x_2 \le 2)$ and $\Diamond(x_1 - x_2 = 1)$ in Example 3.5, they can be expressed as $\Diamond\Box(p_1 \wedge p_2)$ and $\Diamond(p_3 \wedge p_4)$, respectively. Only the first specification is satisfied by the abstract transition system. Hence, we can draw the same conclusion for the original MPL system. The only abstract state that satisfies $\Box(p_3 \wedge p_4)$ is $\widehat{s}_3$. Interestingly, the set $\texttt{DBM}(\widehat{s}_3) = \{\mathbf{x} \in \mathbb{R}^2 \mid x_1 - x_2 = 1\}$ is actually the eigenspace of A (see Example 2.1). □

In general, it is possible that the resulting abstract transition system generated via predicate abstraction does not bisimulate the original MPL system. In that case, the refined abstract transition system can be obtained using the refinement procedure in Subsection 3.3.3.

3.5 Abstractions of Other Models

This section discusses the tropical and predicate abstraction procedures for the model related to MPL systems, e.g., Min-Plus Linear Systems and Interval Max-Plus Linear Systems. Some algorithms are obtained by modifying the procedures from Subsections 3.3-3.4.

3.5.1 Min-Plus Linear Systems

Since MiPL system is a dual of MPL system, the abstraction procedures for the former model can be achieved from the latter one with a few modifications. We start with the PWA representations of an MiPL system $\mathbf{x}(k) = B \otimes' \mathbf{x}(k-1)$ where $B \in \mathbb{R}_{\min}^{n\times n}$. Suppose we denote $A = -B$ and $\mathbf{y}(k) = -\mathbf{x}(k)$ for $k \geq 0$. Hence, the MiPL system can be extressed as an MPL system $\mathbf{y}(k) = A \otimes \mathbf{y}(k-1)$. Finally, by (2.17), the PWA region (deriving from max-plus matrix A) w.r.t. a finite coefficient $\mathbf{g}$ is a DBM $D = \bigcap_{i=1}^{n}\bigcap_{j=1}^{n}\{\mathbf{y} \in \mathbb{R}^n | y_{g_i} - y_j\} \geq A(i,j) - A(i,g_i)\}$.

Since the dynamics of MiPL system are given by $\mathbf{x}(k) = -\mathbf{y}(k)$ for $k \geq 0$ then the respective PWA region from an MiPL system $\mathbf{x}(k) = B \otimes' \mathbf{x}(k-1)$ is a DBM $E = \{-\mathbf{y} \mid \mathbf{y} \in D\}$. From the matrix bound representation of both DBM, it can be expressed as $E = D^{\top}$ where $\top$ is a transpose operator. Algorithm 3.5 describes the steps to generate the tropical abstractions of an MiPL system, and is a modification of Algorithm 3.1.

Algorithm 3.5 Generating PWA representations of an MiPL system via tropical operations

function MINPLUS_TROP_ABS(B) ▷ B is a min-plus matrix
  $\mathbf{R} \leftarrow \emptyset$
  $n \leftarrow$ ROW(B)
  $A \leftarrow$ EXTENT($-B$) ▷ A is a max-plus matrix
  for $\mathbf{g} \in \{1,\ldots,n\}^n$ **do**
    $\mathbf{g} \leftarrow$ EXTENT($\mathbf{g}$)
    if $\mathbf{g}$ is a finite coefficient of A **then**
      generate $[A_{\mathbf{g}}]^{\mathsf{c}}$ according to (2.4) and (2.5)
      $\mathsf{R}_{\mathbf{g}} \leftarrow [A_{\mathbf{g}}]^{\mathsf{c}} \otimes A \oplus I_{n+1}$
      $\mathsf{R}_{\mathbf{g}} \leftarrow [\mathsf{R}_{\mathbf{g}}]^{\top}$ ▷ transposing the matrix
      generate $\mathsf{S}_{\mathbf{g}}$ according to (3.3) ▷ a sign matrix for $\mathsf{R}_{\mathbf{g}}$
      if $(\mathsf{R}_{\mathbf{g}}, \mathsf{S}_{\mathbf{g}})$ is not empty **then** ▷ emptiness checking of a DBM
        $\mathbf{R} \leftarrow \mathbf{R} \cup \{\mathsf{R}_{\mathbf{g}}, \mathsf{S}_{\mathbf{g}}, \mathbf{g}\}$
  return R

The predicate abstraction for an MiPL system can be done by similar modification. Following the procedure Section 3.4, a predicate from a max-plus matrix $A \in \mathbb{R}_{\max}^{n\times n}$ is chosen as an inequality $x_i - x_j \sim A(k,j) - A(k,i)$ for each $k \in \{1,\ldots,n\}$ and $1 \leq i < j \leq n$. Hence, a predicate from a min-plus $B \in \mathbb{R}_{\min}^{n\times n}$ is an equality $x_j - x_i \sim B(k,i) - B(k,j)$ for each $k \in \{1,\ldots,n\}$ and $1 \leq i < j \leq n$. Algorithm 3.6 is the modification of Algorithm 3.2 to generate a set of predicates from an min-plus matrix.

The generation of abstract states can be done similarly as in Algorithm 3.3. However, the steps to find affine dynamics are slightly difference from Algorithm 3.4. Now suppose $p = x_j - x_i \geq B(k,i) - B(k,j) \in P_k$ i.e., a predicate generated by

MIPL2PRED(A, k) in Algorithm 3.6. If p is true on abstract state $\widehat{s}$ then we have $x_j + B(k,j) \geq x_i + B(k,i)$. Hence, $\texttt{left}(p) = j$ cannot characterised the maximum term for $\min\{B(k,1) + x_1, \ldots, B(k,n) + x_n\}$. On the other hand, if p is false then $\texttt{right}(p) = i$ is not the maximum term. Algorithm 3.7 illustrates the steps to generate the affine dynamics from an abstract states.

Algorithm 3.6 Generating a set of predicates from an MiPL system

Input: $B \in \mathbb{R}_{\min}^{n \times n}$
Output: P, a set of predicates

```
1: function MIPL2PRED(B, k)
2:     n ← ROW(B)
3:     P_k ← ∅
4:     for j ∈ {2, ..., n} do
5:         for i ∈ {1, ..., j − 1} do
6:             if B(k, i) ≠ ε and B(k, j) ≠ ε then
7:                 P_k ← P_k ∪ {x_j − x_i ≥ B(k, i) − B(k, j)}      ▷ notice the difference with
8:     return P_k                                                   line 6 of Algorithm 3.2
9: function MIPL2PRED(B)
10:    n ← ROW(B)
11:    P ← ∅
12:    for k ∈ {1, ..., n} do
13:        P ← P ∪ MIPL2PRED(B, k)
14:    return P
```

Algorithm 3.7 Generation of the affine dynamics for an abstract state

Inputs: $B \in \mathbb{R}_{\min}^{n \times n}$,
$\widehat{s} \in \widehat{S}$, an abstract state
$P_1, \ldots, P_n$, sets of predicates generated by Algorithm 3.2
Output: $\mathbf{g} = (g_0, \ldots, g_n)$, the finite coefficient representing the affine dynamics for $\widehat{s}$

```
1: function GET_MINPLUS_AFFINE(B, ŝ, P_1, ..., P_n)
2:     g ← zeros(1, n + 1)                                  ▷ g_0 is always 0
3:     for k ∈ {1, ..., n} do
4:         idx ← Find(B(k, ·) ≠ ε)
5:         for p ∈ P_k do
6:             if p is true in ŝ then
7:                 idx ← idx \ {left(p)}      ▷ notice the difference with line 7 of Algorithm 3.4
8:             else
9:                 idx ← idx \ {right(p)}     ▷ notice the difference with line 9 of Algorithm 3.4
10:        g_k ← idx[0]
11:    return g
```

After generating the abstract states (either using tropical or predicate abstraction), the next step is to generate the abstract transition system that simulates the original MiPL system. This can be done by via a *one-step reachability* procedure in Subsection 3.3.2. By Remark 2.3, the image of a DBM w.r.t. affine dynamics generated from an MiPL system yields a DBM and can be computed by Algorithm 2.5. Furthermore, the attempt to derive an abstract transition system that bisimulates the MiPL system can be done according to the refinement procedure in Subsection 3.3.2. We argue that Theorem 3.1 also applies for MiPL system since the image computation $B \otimes' \mathbf{x}$ is uniquely defined for each $\mathbf{x} \in \mathbb{R}^n$.

3.5.2 Interval Max-Plus Linear Systems

The abstractions of IMPL systems are more straightforward than that of MiPL systems since they can be generated using algorithms in Sections 3.3 and 3.4 without any modification. We recall that the PWA representations of an IMPL system with interval matrix $\mathbf{A} = [\underline{A}, \overline{A}]$ are generated from its upper matrix $\overline{A}$. Hence, tropical abstraction of an IMPL system can be generated by MAXPLUS_TROP_ABS($\overline{A}$) in Algorithm 3.1. Similarly, predicate abstraction for an IMPL system can be done by taking the upper matrix $\overline{A}$ as the input of Algorithms 3.2-3.4.

Generating the abstract transition system that simulates the IMPL system can be done similarly using *one-step reachability* procedure in Subsection 3.3.2. Suppose TS is a concrete transition system generated by an IMPL system (see Definition 3.5) and TS_f be the abstract transition system induced by an abstraction function $f : S \to \widehat{S}$ (either using tropical or predicate abstractions). Given two abstract states $\widehat{s}$ and $\widehat{s}'$, there exists an abstract transition from $\widehat{s}$ to $\widehat{s}'$ if $Post(f^{-1}(\widehat{s})) \cap f^{-1}(\widehat{s}')$ is not empty. We recall that both $Post(f^{-1}(\widehat{s}))$ and $f^{-1}(\widehat{s}')$ are a DBM, and the latter one can be computed using Algorithm 2.7.

Example 3.7. Let us consider an IMPL system with an interval matrix $\mathbf{A} = [\underline{A}, \overline{A}]$ in (2.16). By MAXPLUS_TROP_ABS($\overline{A}$) in Algorithm 3.1, the resulting PWA regions are $\overline{\mathsf{R}}_{(0,1,1)} = \{\mathbf{x} \mid \mathbb{R}^2 \mid x_1 - x_2 \geq 3\}$, $\overline{\mathsf{R}}_{(0,2,1)} = \{\mathbf{x} \mid \mathbb{R}^2 \mid 2 \leq x_1 - x_2 < 3\}$, and $\overline{\mathsf{R}}_{(0,2,2)} = \{\mathbf{x} \mid \mathbb{R}^2 \mid x_1 - x_2 < 2\}$. Suppose we have a set of atomic proposition $AP = \{p\}$ where $\texttt{DBM}(p) = \{\mathbf{x} \in \mathbb{R}^2 \mid x_1 - x_2 < 2\}$. The abstraction and labeling function is respectively defined as follows

$$f(\mathbf{x}) = \begin{cases} \widehat{s}_1, & \text{if } x_1 - x_2 \geq 3, \\ \widehat{s}_2, & \text{if } 2 \leq x_1 - x_2 < 3, \\ \widehat{s}_3, & \text{if } x_1 - x_2 < 2, \end{cases} \qquad L_f(\widehat{s}_i) = \begin{cases} \emptyset, & \text{if } i \in \{1, 2\}, \\ \{p\}, & \text{if } i = 3. \end{cases}$$

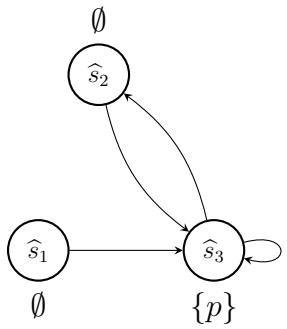


Figure 3.4: Abstract transition system generated from an IMPL system in (2.16). All abstract states are initial.

Without giving the details, we have $Post(f^{-1}(\widehat{s}_1)) = \{\mathbf{x} \in \mathbb{R}^2 \mid -2 \leq x_1 - x_2 \leq 0\}$, $Post(f^{-1}(\widehat{s}_2)) = \{\mathbf{x} \in \mathbb{R}^2 \mid -2 \leq x_1 - x_2 \leq 1\}$, and $Post(f^{-1}(\widehat{s}_3)) = \{\mathbf{x} \in \mathbb{R}^2 \mid -2 \leq x_1 - x_2 \leq 2\}$. The intersection $Post(f^{-1}(\widehat{s}_i)) \cap f^{-1}(\widehat{s}_j)$ is not empty for $(i, j) \in \{(1, 3), (2, 3), (3, 2), (3, 3)\}$. The resulting abstract transition is shown in Figure 3.4. Suppose we want to verify a specification $\square\Diamond p$. It is straightforward to conclude that the specification is valid on the abstract transition system. Thus, the specification also holds for the original IMPL system (2.16). $\square$

We now will discuss the condition for the abstract transition system that bisimulates the original IMPL system. Let TS be the concrete transition system associated with an IMPL system and TS_f be the abstract transition system induced by an abstraction function $f : S \to \widehat{S}$. As in Theorem 3.1, we aim to obtain a simulation for (TS_f, TS) with binary relation $\mathcal{R} = \{(f(\mathbf{x}), \mathbf{x}) \mid \mathbf{x} \in S\}$. Recall that, in TS, we have $|Post(\mathbf{x})| > 1$ for $\mathbf{x} \in \mathbb{R}^n$. Thus, the requirement $|Path(\widehat{s})| = 1$ is not necessary.

By Definition 3.10, we can formulate a condition such that TS simulates TS_f. From Definition 3.10(b), $\mathcal{R}$ is a simulation for (TS_f, TS) if for each abstract transition $\widehat{s} \longrightarrow_f \widehat{s}'$, the following condition holds: for each concrete state $\mathbf{x}$ associated with $\widehat{s}$ (namely, $\mathbf{x} \in f^{-1}(\widehat{s})$), there exists a concrete transition from $\mathbf{x}$ to $\mathbf{x}'$ (namely, $\mathbf{x} \longrightarrow \mathbf{x}'$) such that $\mathbf{x}'$ is associated with $\widehat{s}'$ (namely, $f(\mathbf{x}') = \widehat{s}'$). The preceding condition is equivalent with the inverse image of the region corresponding to $\widehat{s}'$ w.r.t. the affine dynamics of $\widehat{s}$ is a superset of the region corresponding to $\widehat{s}$ i.e., $Pre(f^{-1}(\widehat{s}')) \supseteq f^{-1}(\widehat{s})$.

Suppose there exists an abstract transition $\widehat{s} \longrightarrow_f \widehat{s}'$ that does not satisfy the condition in the preceding paragraph. In the refinement procedure, we partition the region associated with $\widehat{s}$ and associate each partitioning region to an abstract state. Then, we compute the incoming and outgoing transitions of the abstract states corresponding to those newly created partitioning regions. These steps are repeated until all abstract transitions satisfy the condition in the preceding paragraph.

The partitioning procedure of the region associated with $\widehat{s}$ is as follows. The region associated with $\widehat{s}$ is divided into two parts. The first part contains the set of concrete states $\mathbf{x}$ associated with $\widehat{s}$, i.e. $f(\mathbf{x}) = \widehat{s}$, such that there exists a transition from $\mathbf{x}$ to $\mathbf{x}'$, i.e. $\mathbf{x} \longrightarrow \mathbf{x}'$, and $\mathbf{x}'$ is associated with $\widehat{s}'$, i.e. $f(\mathbf{x}') = \widehat{s}'$. The first part is obtained by computing the intersection of the region associated with $\widehat{s}$ and the inverse image of the region corresponding to $\widehat{s}'$ w.r.t. the affine dynamics in the region corresponding to $\widehat{s}$. The first part is a DBM due to Proposition 2.9 and the fact that the intersection of two DBM is a DBM. The second part contains the set of concrete states $\mathbf{x}$ associated with $\hat{s}$, i.e. $f(\mathbf{x}) = \widehat{s}$, that does not have any transition to concrete states associated with $\widehat{s}'$. The second part is obtained by computing the set difference of the region associated with $\widehat{s}$ and the inverse image of the region corresponding to $\widehat{s}'$ w.r.t. the dynamics in the region corresponding to $\widehat{s}$. Since the set difference of two DBM is a union of finitely many DBM, the second part is a union of finitely many DBM.

Remark 3.3. Compared to the refinement procedure in Subsection 3.3.3, the above procedure is more complicated since one needs to check each transition in the abstract transition system. Moreover, it involves the set difference of two DBM, which cannot be expressed as a single DBM. Therefore, we do not elaborate more on the refinement procedure in this subsection for these two reasons.

3.6 Computational Benchmarks

This section presents the performance comparison of the proposed algorithms to perform the (specification-free) abstraction of MPL systems. For increasing dimen-

sions n of the given MPL system, we compute the running time required to generate abstract states and transitions of the obtained abstract transition system. We also keep track of the number of states and of transitions that are directly related to the memory consumption of the algorithms.

For each n, we generate 50 regular matrices $A \in \mathbb{R}_{\max}^{n\times n}$ with $m \in \{2, 4\}$ finite elements in each row, where the values of the finite elements are uniformly generated integers between 1 and 20. The locations of the finite elements are chosen randomly. The experiments have been implemented in C++ on an Intel(R) Xeon(R) Gold 6248 CPU @ 2.50GHz with 120GB of RAM. We use Armadillo [83] to enable matrix operations in max-plus algebra as well as DBM manipulations. For each experiment, we set 30 minutes as `timeout` limit and 20GB as the maximum allowed memory, i.e., `memout` (out of memory) limit.

Table 3.1: The average runtime for tropical and predicate abstractions of MPL systems.

(n, m)	avg. time generation of abstract states		avg. time generation of abstract transitions	
	Trop. Abs.	Pred. Abs.	Trop. Abs.	Pred. Abs.
$(4, 2)$	0.0 [sec]	0.0 [sec]	0.0 [sec]	0.0 [sec]
$(5, 2)$	0.0 [sec]	0.0 [sec]	0.01 [sec]	0.01 [sec]
$(6, 2)$	0.0 [sec]	0.0 [sec]	0.02 [sec]	0.02 [sec]
$(7, 2)$	0.0 [sec]	0.01 [sec]	0.06 [sec]	0.06 [sec]
$(8, 2)$	0.03 [sec]	0.01 [sec]	0.35 [sec]	0.33 [sec]
$(9, 2)$	0.07 [sec]	0.05 [sec]	2.28 [sec]	2.34 [sec]
$(10, 2)$	0.15 [sec]	0.11 [sec]	8.62 [sec]	8.78 [sec]
$(11, 2)$	0.36 [sec]	0.35 [sec]	38.49 [sec]	39.15 [sec]
$(12, 2)$	0.77 [sec]	0.78 [sec]	162.42 [sec]	163.55 [sec]
$(13, 2)$	1.53 [sec]	1.92 [sec]	668.35 [sec]	658.52 [sec]
$(14, 2)$	4.14 [sec]	4.55 [sec]	`timeout`(41)	`timeout`(42)
$(15, 2)$	12.82 [sec]	10.83 [sec]	`timeout`(50)	`timeout`(50)
$(16, 2)$	21.61 [sec]	17.75 [sec]	`timeout`(43)	`timeout`(45)
$(17, 2)$	27.48 [sec]	27.06 [sec]	`memout`(50)	`memout`(50)
$(18, 2)$	82.25 [sec]	58.06 [sec]	`memout`(50)	`memout`(50)
$(19, 2)$	157.34 [sec]	114.77 [sec]	`memout`(50)	`memout`(50)
$(20, 2)$	`memout`(50)	`memout`(50)	`memout`(50)	`memout`(50)
$(4, 4)$	0.03 [sec]	0.57 [sec]	0.01 [sec]	9.65 [sec]
$(5, 4)$	0.12 [sec]	3.67 [sec]	0.1 [sec]	418.98 [sec]
$(6, 4)$	0.63 [sec]	30.71 [sec]	6.11 [sec]	`timeout`(20)
$(7, 4)$	1.39 [sec]	`memout`(50)	21.38 [sec]	`memout`(50)
$(8, 4)$	4.01 [sec]	`memout`(50)	79.02 [sec]	`memout`(50)

Over 50 independent experiments, Table 3.1 reports the average time needed to generate the abstract transition system via tropical and predicate abstractions, broken down into two successive procedures for the generation of the abstract states and the transitions, respectively. The notation `timeout`(r) and `memout`(r) correspond to a condition that there are r failed experiments due to `timeout` and `memout`, respectively. For matrices with two finite elements at each row, it is evidence that the

predicate abstraction (Algorithm 3.3) is slightly more efficient than the tropical abstraction (Algorithm 3.1) to generate the abstract states. The average running time to generate the abstraction transitions is relatively similar since both algorithms use *one-step reachability analysis* to determine the transitions among abstract states. Interestingly, as shown in Table 3.2, both procedures always result in the same number of abstract states and transitions.

Table 3.2: The average number of abstract states and transitions for tropical and predicate abstractions of MPL systems.

(n, m)	avg. number of abstract states		avg. number of abstract transitions	
	Trop. Abs.	Pred. Abs.	Trop. Abs.	Pred. Abs.
$(4, 2)$	11.24	11.24	33.08	33.08
$(5, 2)$	23.78	23.78	114.12	114.12
$(6, 2)$	46.08	46.08	389.76	389.76
$(7, 2)$	83.02	83.02	956.02	956.02
$(8, 2)$	183.40	183.40	3.80×10^3	3.80×10^3
$(9, 2)$	339.04	339.04	1.02×10^4	1.02×10^4
$(10, 2)$	709.60	709.60	4.15×10^4	4.15×10^4
$(11, 2)$	1.28×10^3	1.28×10^3	1.07×10^5	1.07×10^5
$(12, 2)$	2.70×10^3	2.70×10^3	3.73×10^5	3.73×10^5
$(13, 2)$	5.25×10^3	5.25×10^3	1.13×10^6	1.13×10^6
$(14, 2)$	1.04×10^4	1.04×10^4	-	-
$(15, 2)$	2.07×10^4	2.07×10^4	-	-
$(16, 2)$	3.83×10^4	3.83×10^4	-	-
$(17, 2)$	7.79×10^4	7.79×10^4	-	-
$(18, 2)$	1.61×10^5	1.61×10^5	-	-
$(19, 2)$	3.29×10^5	3.29×10^5	-	-
$(20, 2)$	-	-	-	-
$(4, 4)$	31.75	852.20	69.15	2.30×10^3
$(5, 4)$	106.40	7.56×10^3	319.30	4.28×10^4
$(6, 4)$	346.20	6.76×10^4	1.60×10^3	-
$(7, 4)$	1.09×10^3	-	8.10×10^3	-
$(8, 4)$	3.42×10^3	-	3.78×10^4	-

On the other hand, for matrices with four finite elements at each row, the performance of predicate abstraction is inferior to that of tropical abstraction. For 7-dimensional MPL systems, the former procedure fails to generate the abstract states within the memory limit, let alone generate the abstract transitions. As presented in Table 3.2, for $m = 4$, predicate abstraction produces more abstract states and transitions. We recall that the worst-case complexity to generated abstract states using tropical and predicate abstractions are $\mathcal{O}(n^3 m^n)$ and $\mathcal{O}(n^3 2^{n\binom{m}{2}})$, respectively.

Table 3.1-3.2 clearly exhibit the computational bottleneck (time and memory requirements) of both algorithms. For $n \geq 14$, most experiments for both algorithms fail to generate the abstract transition systems within the time limit but still manage to generate the abstract states. As the dimension increases (and with more finite elements), the number of abstract states and transitions grows steeply, eventually

exceeding the maximum memory allowed.

Remark 3.4. We do not include the performance of the state-of-the-art procedure in [4] since it uses MATLAB. Furthermore, we also do not present the benchmarks for MiPL and IMPL systems since the corresponding abstraction procedures are similar to MPL systems. □

3.7 Summary

This chapter has introduced two novel techniques to generate abstractions of MPL systems by leveraging the translation of MPL systems into the equivalent PWA systems. The resulting abstraction is represented as a transition system that can be either simulate or bisimulate the original MPL system.

The computational complexity of the approaches has been fully explored, and their performance has been tested on a computational benchmark. In addition, the benchmark has displayed a bottleneck related to the dimension of the underlying state matrix and the number of finite elements at each row. Finally, it has been shown that the proposed techniques can be applied to similar models, namely MiPL and IMPL systems.

Chapter 4

Computing Transient of Max-Plus Linear Systems via Satisfiability Modulo Theory

This chapter proposes the computation of the transient bound of MPL systems through the Satisfiability Modulo Theory (SMT). Generally speaking, a transient is a starting bound of periodic behaviour (with a specific period called cyclicity) in MPL systems. As will be apparent in Chapters 5 and 6, such a bound will play an essential role in solving reachability analysis and bounded model checking of MPL systems. The main idea underpinning the new method is to transform the problem instance into a formula in quantifier-free Real Difference Logic (QF-RDL) and then pass the formula into an SMT solver, which outputs the transient. More precisely, to check the formula's validity, we check the unsatisfiability of its negation. If the SMT solver reports "satisfied", then the original formula admits a counterexample, from which we can refine the formula. On the other hand, if the SMT solver reports "unsatisfied", then from the formula, we obtain the transient and the corresponding cyclicity. Additionally, we provide a procedure to synthesise the subset of the state space of an MPL system that corresponds to a specific pair of transient and cyclicity. Finally, we show that the resulting procedures can be applied to MiPL systems.

4.1 Preliminaries

The first part of this section elaborates on the transient over MPL systems, including several known upper bounds and the classical method to compute it. Then, in the second part, we briefly describe SMT and the logical theories used throughout this thesis.

4.1.1 Transient in Max-Plus Linear Systems

Given a square matrix $A \in \mathbb{R}_{\max}^{n \times n}$, a transient is the starting bound of the periodicity of $A^{\otimes k}$ (see Proposition 2.2). The transient is key to solving several fundamental problems of MPL systems, such as reachability analysis [5,6,75] and bounded model checking [74]. Furthermore, it plays a crucial role as the "completeness threshold"

(namely, the maximum iteration sufficient for the termination of the algorithm) [38] for those two problems.

The computation of the transient is an interesting problem, as it is not linearly correlated to the dimension of the MPL system. Thus, the resulting transient can be relatively large for a small-dimensional MPL system. However, there are several known upper bounds for the transient. The following theorems relate these upper bounds of $\mathtt{tr}(A)$ to the values of the cycle mean in the precedence graph $\mathcal{G}(A)$.

Theorem 4.1 ([80])**.** The transient for irreducible matrices $A \in \mathbb{R}_{\max}^{n\times n}$ is at most

$$2n^2 + \frac{2n^2\|A\|}{\lambda(A) - \lambda_2(A)},$$

where $\|A\|$ denotes the difference between the largest and smallest finite elements of A and $\lambda_2(A)$ is the second-largest cycle mean in $\mathcal{G}(A)$. □

Theorem 4.2 ([88])**.** Let $A \in \mathbb{R}_{\max}^{n\times n}$ be a max-plus algebraic matrix with only finite elements. The transient for A is at most

$$\max\{2n^2, \left\lceil \frac{2\|A\|}{\lambda(A) - \lambda_2(A)} \right\rceil + n - 1\}.$$

□

A few (and more sophisticated) graph-based approaches to derive the upper bounds are given in [71, 80]. In general, these upper bounds are much larger than the actual transient.

As per Proposition 2.2, the common method to obtain the transient of $A \in \mathbb{R}_{\max}^{n\times n}$ is by computing the power of the matrix $A^{\otimes 0}, A^{\otimes 1}, \ldots$ until we find $l \geq 0$ such that $A^{\otimes(l+c)} = \lambda^{\otimes c} \otimes A^{\otimes l}$ where λ, c are respectively the max-plus eigenvalue and cyclicity of A. Notice that, there are at most $\frac{(l+c+1)(l+c)}{2}$ equality checking[6] of two matrices in the process. Similarly, to find the transient of A w.r.t. a max-plus cone $X = \mathsf{rcone}(V)$ one needs to compute $A^{\otimes 0} \otimes V, A^{\otimes 1} \otimes V, \ldots$. This classical method is indeed not efficient when the resulting transient is relatively large.

Algorithm 4.1 illustrates the procedure to compute transient (and cyclicity) for a max-plus cone w.r.t. $A \in \mathbb{R}_{\max}^{n\times n}$. While originally designed for irreducible matrices, it also can be applied to find the transient of reducible matrices (if it exists). For this reason, we assign a maximum bound N as a termination condition. It is important to note that Algorithm 4.1 can also be used to compute the local transient and cyclicity for a vector: when V has only one column. The algorithm starts by computing the cycle-time vector χ of the state matrix. By Theorem 2.2, if the entries of χ are not all the same, then the transient for $\mathsf{cone}(V)$ does not exist. In line 11, we perform equality checking between $A^{\otimes it-m} \otimes V$ and $A^{\otimes it} \otimes V$.

[6] At the it-th iteration, one needs to check whether $A^{\otimes it} = (\lambda \times m) \otimes A^{\otimes it-m}$ for $m \in \{1, \ldots, it\}$.

Algorithm 4.1 Computation of cyclicity and transient of A w.r.t. $\mathsf{rcone}(V)$

Inputs: $A \in \mathbb{R}_{\max}^{n\times n}$,
a max-plus cone expressed as a matrix $V \in \mathbb{R}_{\max}^{n\times n}$,
positive integer N
Output: a pair of transient and cyclicity

```
1: function MPL_TRANS_CONE(A, V, N)
2:     M ← EMPTYSTACK()                          ▷ empty stack used to store A^0 ⊗ V, A^1 ⊗ V, ...
3:     M.push_back(V)
4:     it ← 0                                    ▷ number of iterations
5:     χ ← MPL_CTV(A)                            ▷ computing cycle-time vector by Algorithm 2.2
6:     if elements of χ are all equal then
7:         while (it ≤ N) do
8:             M.push_back(A ⊗ M[it])
9:             it ← it + 1
10:            for 1 ≤ m < it do
11:                if (M[it] = (λ × m) ⊗ M[it − m]) then
12:                    return ⟨it − m, m⟩
13:        if (it > N) then
14:            print "terminated after reaching maximum bound"
15:    else
16:        print "the transient does not exist"
```

By Theorem 2.2 and Proposition 2.4, one can classify an MPL system (2.6) into a category in Definition 2.8. For *boundedly periodic* MPL systems, computing the transient is a decidable problem. This is because they ensure the existence of a finite transient, meaning that Algorithm 4.1 eventually terminates. However, Algorithm 4.1 is sound but does not necessarily terminate (in general) for *unboundedly periodic* MPL systems.

Remark 4.1. The procedure in Algorithm 4.1 only employs matrix operations in the max-plus algebra. It can be improved by computing the cyclicity of the matrix from the corresponding precedence graph. If the resulting cyclicity is c then the range in line 10 of Algorithm 4.1 can be taken between 1 and c. □

4.1.2 Satisfiability Modulo Theory

Satisfiability Modulo Theory (SMT) deals with the problem of determining the satisfaction of a first-order logical formula w.r.t. some logical theory background, such as Boolean logic (which generalises SAT theory), bit-vectors, real and integer arithmetics, and so on [15]. For instance, the following formula

$$(x \geq 0) \wedge (y < 2) \wedge (x - y < -1)$$

has solutions for $x, y \in \mathbb{R}$ but no solution for $x, y \in \mathbb{Z}$. In general, an SMT formula may contain conjunctions ($\wedge$), disjunctions ($\vee$), and quantifiers ($\exists, \forall$). An SMT solver reports whether the given formula is satisfiable (`SAT`) or not satisfiable (`UNSAT`). For the former case, it could provide a *model*, i.e., a satisfying assignment for the formula.

SMT has grown into a very active research subject: it has a standardised library and a collection of benchmarks, developed by the SMT community [14], and an annual international competition for SMT solvers [13]. As a result, there are several

powerful SMT solvers, such as MATHSAT5 [35], Yices 2.2 [57], and Z3 [50]. Applications of SMT-solving arise on supervisory control of discrete-event systems [86], verification of neural networks [68], optimization [69], and beyond.

SMT solvers can support different theories. A widely used theory is Linear Real Arithmetic (LRA). A formula in LRA is an arbitrary Boolean combination, or universal ($\forall$) and existential ($\exists$) quantifications, of atoms in the form $\sum_i a_i x_i \bowtie c$ where $\bowtie \in \{>, <, \geq, \leq, \neq, =\}$, every x_i is a real variable, and every a_i and c are rational constants. Difference logic (RDL) is the subset of LRA in which all atoms are restricted to the form $x_i - x_j \bowtie c$. The quantifier-free fragment of LRA and RDL is denoted as QF-LRA and QF-RDL, respectively. Satisfiability checking over both theories are decidable [15, Section 26.2].

4.2 SMT-Based Computation of Transient of Max-Plus Linear Systems

This section presents the computation of transient for MPL systems via SMT-solving. The first part of the section discusses the translation of inequalities over max-plus algebra into QF-RDL formulae. The second part provides the procedure to compute the transient and cyclicity, while the third one shows how to synthesise the state-space of MPL systems w.r.t. a specific pair of transient and cyclicity.

4.2.1 From Max-Plus Algebra to Difference Logic

In this subsection, we show that the inequalities in max-plus algebra can be expressed as formulae in QF-RDL. For the rest of this thesis, $\sim$ is either $\geq$ or $>$. We write $\neg(a \sim b)$ if it is not the case that $a \sim b$.

Proposition 4.1 ([1, Proposition 5])**.** Given $a_1, \ldots, a_p, a, b \in \mathbb{R}_{\max}$, real-valued variables $\mathtt{x}_1, \ldots, \mathtt{x}_p$, and $1 \leq j \leq p$, we have

$$\bigoplus_{i=1}^{p}(\mathtt{x}_i + a_i) \sim a \;\equiv\; \bigvee_{i=1}^{p}(\mathtt{x}_i + a_i \sim a), \tag{4.1}$$

$$a \sim \bigoplus_{i=1}^{p}(\mathtt{x}_i + a_i) \;\equiv\; \bigwedge_{i=1}^{p}(a \sim \mathtt{x}_i + a_i), \tag{4.2}$$

$$\bigoplus_{i=1}^{p}(\mathtt{x}_i + a_i) \sim \mathtt{x}_j + b \equiv \begin{cases} \mathtt{true}, & \text{if } (a_j \sim b), \quad (4.3) \\ \bigvee\limits_{\substack{i=1 \\ i \neq j}}^{p}(\mathtt{x}_i + a_i \sim \mathtt{x}_j + b), & \text{otherwise}, \quad (4.4) \end{cases}$$

$$\mathtt{x}_j + b \sim \bigoplus_{i=1}^{p}(\mathtt{x}_i + a_i) \equiv \begin{cases} \bigwedge\limits_{\substack{i=1 \\ i \neq j}}^{p}(\mathtt{x}_j + b \sim \mathtt{x}_i + a_i), & \text{if } (b \sim a_j), \quad (4.5) \\ \mathtt{false}, & \text{otherwise}. \quad (4.6) \end{cases}$$

Proof. The equation $\bigoplus_{i=1}^{p}(\mathtt{x}_i + a_i) \sim a$ holds iff there exists $1 \leq i \leq p$ such that $\mathtt{x}_i + a_i \sim a$. Similarly, $a \sim \bigoplus_{i=1}^{p}(\mathtt{x}_i + a_i)$ holds iff $a \sim \mathtt{x}_i + a_i$ for all $1 \leq i \leq p$. Hence, we get (4.1) and (4.2).

By applying (4.1), we have

$$\bigoplus_{i=1}^{p}(\mathtt{x}_i + a_i) \sim \mathtt{x}_j + b \equiv \bigvee_{i=1}^{p}(\mathtt{x}_i + a_i \sim \mathtt{x}_j + b) \equiv (a_j \sim b) \vee \bigvee_{\substack{i=1 \\ i\neq j}}^{p}(\mathtt{x}_i + a_i \sim \mathtt{x}_j + b).$$

If it is true that $a_j \sim b$ then we get (4.3), otherwise we get (4.4). The proof for (4.5)-(4.6) is similar to that of (4.3)-(4.4). □

Proposition 4.2 ([1, Proposition 6])**.** Given real valued variables $\mathtt{x}_1, \ldots, \mathtt{x}_p$ and max-plus scalars $a_1, \ldots, a_p, b_1, \ldots, b_p \in \mathbb{R}_{\max}$, the inequality

$$F \equiv \bigoplus_{i=1}^{p}(\mathtt{x}_i + a_i) \sim \bigoplus_{j=1}^{p}(\mathtt{x}_j + b_j) \tag{4.7}$$

is equivalent to

$$F^* \equiv \bigoplus_{i\in S_1}(\mathtt{x}_i + a_i) \sim \bigoplus_{j\in S_2}(\mathtt{x}_j + b_j), \tag{4.8}$$

where $S_1 = \{1, \ldots, p\} \backslash \{1 \leq k \leq p \mid a_k = \varepsilon \text{ or } \neg(a_k \sim b_k)\}$ and $S_2 = \{1, \ldots, p\} \backslash \{1 \leq k \leq p \mid b_k = \varepsilon \text{ or } a_k \sim b_k\}$, respectively.

Proof. Suppose we set initially $S_1 = S_2 = \{1, \ldots, p\}$. By applying (4.1), (4.7) can be expressed as

$$F \equiv \bigvee_{i\in S_1}\left(\mathtt{x}_i + a_i \sim \bigoplus_{j\in S_2}(\mathtt{x}_j + b_j)\right).$$

Indeed we can ignore any scalar k if $a_k = \varepsilon$. Furthermore, for each l such that $\neg(a_l \sim b_l)$, by (4.6), we have

$$F \equiv \mathtt{false} \vee \bigvee_{i\in S_1-\{l\}}\left(\mathtt{x}_i + a_i \sim \bigoplus_{j\in S_2}(\mathtt{x}_j + b_j)\right) \equiv \bigvee_{i\in S_1-\{l\}}\left(\mathtt{x}_i + a_i \sim \bigoplus_{j\in S_2}(\mathtt{x}_j + b_j)\right).$$

This shows that l can be removed from S_1. Similarly, by (4.2), we have

$$F \equiv \bigwedge_{j\in S_2}\left(\bigoplus_{i\in S_1}(\mathtt{x}_i + a_i) \sim \mathtt{x}_j + a_j\right).$$

Again we can ignore any scalar k when $b_k = \varepsilon$. By (4.3), for each $l \in S_2$ such that $a_l \sim b_l$ we have

$$F \equiv \mathtt{true} \wedge \bigwedge_{j\in S_2-\{l\}}\left(\bigoplus_{i\in S_1}(\mathtt{x}_i + a_i) \sim \mathtt{x}_j + a_j\right) \equiv \bigwedge_{j\in S_2-\{l\}}\left(\bigoplus_{i\in S_1}(\mathtt{x}_i + a_i) \sim \mathtt{x}_j + a_j\right).$$

Hence l can be removed from S_2. □

Proposition 4.2 ensures that any inequality expression in max-plus algebra (4.7) can be reduced to a simpler one (4.8) in which no a variable appears on both sides i.e., $S_1 \cap S_2 = \emptyset$. However, S_1 and S_2 cannot be both empty if there exists at least one finite scalar in both sides of (4.7). Finally, Proposition 4.3 shows that any reduced formula of (4.8) can be expressed as a QF-RDL formula in disjunctive and conjunctive normal forms.

Proposition 4.3 ([1, Proposition 7])**.** Given a reduced formula in (4.8), then

$$F^* \equiv \bigwedge_{j\in S_2}\left(\bigvee_{i\in S_1}(\mathtt{x}_i - \mathtt{x}_j \sim b_j - a_i)\right) \equiv \bigvee_{i\in S_1}\left(\bigwedge_{j\in S_2}(\mathtt{x}_i - \mathtt{x}_j \sim b_j - a_i)\right). \tag{4.9}$$

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

Proof. By applying (4.1) on (4.8), one can obtain

$$\bigvee_{i\in S_1}\left(x_i + a_i \sim \bigoplus_{j\in S_2}(\mathtt{x}_j + b_j)\right).$$

Following this, by applying (4.2) on the above expression, one can obtain

$$\bigvee_{i\in S_1}\left(\bigwedge_{j\in S_2}(x_i + a_i \sim x_j + b_j)\right).$$

Similarly, by applying (4.2) and then (4.1) on (4.8), one can get the left hand side of (4.9). In case $S_1 = \emptyset$, the inequality (4.8) can be written as $\max\{\emptyset\} \sim \bigoplus_{j\in S_2}(x_j + b_j)$. Under convention that $\max\{\emptyset\} = \varepsilon$, that inequality is never satisfied. The proof for the case $S_2 = \emptyset$ can be deducted analogously. □

4.2.2 Procedure to Compute Transient of MPL Systems with SMT

This subsection will describe the procedure to compute the transient (and cyclicity) of an MPL system via SMT-solving. The idea behind the SMT-based procedure is by transforming the equality checking in line 11 of Algorithm 4.1 into satisfiability checking of a QF-RDL formula. Notice that the quantity $\mathbf{M}[it]$ in Algorithm 4.1 corresponds to $A^{\otimes it} \otimes V$, and $\mathsf{rcone}(V)$ can be expressed as matrix V. Thus, it can be equivalently written as

$$(A^{\otimes it} \otimes V) \otimes \mathbf{x} = (\lambda \times m) \otimes (A^{\otimes it-m} \otimes V) \otimes \mathbf{x}, \quad \forall \mathbf{x} \in \mathbb{R}^p, \tag{4.10}$$

where p is the number of columns of V. By denoting $R = A^{\otimes it} \otimes V$ and $S = (\lambda \times m) \otimes A^{\otimes it-m} \otimes V$, (4.10) can be expressed as

$$\bigwedge_{k=1}^{n}\left(\left(\bigoplus_{i=1}^{p}(\mathtt{x}_i + r_{ki}) \geq \bigoplus_{j=1}^{p}(\mathtt{x}_j + s_{kj})\right) \wedge \left(\bigoplus_{i=1}^{p}(\mathtt{x}_i + s_{ki}) \geq \bigoplus_{j=1}^{p}(\mathtt{x}_j + r_{kj})\right)\right), \tag{4.11}$$

where r_{ki} (resp. s_{ki}) is the element of R (resp. S) at row k and column i. For simplicity, we denote (4.11) as $\texttt{EqFunc}(R, S)$. By Proposition 4.3, each disjunct in (4.11) can be expressed as a formula in QF-RDL.

Algorithm 4.2 summarizes the SMT-based version of Algorithm 4.1. If the corresponding cycle-time vector elements are all equal, we set the value for transient and cyclicity respectively to $l = 0$ and $c = 1$ (the smallest possible for both). Then, we generate the corresponding QF-RDL formula F w.r.t. (4.11) in line 9. To check the validity of F, we use an SMT solver to check the unsatisfiability of the negation instead. If it is not satisfiable, then the original formula is valid. As a result, the current value of l and c are the largest possible for transient and cyclicity, respectively.

On the other hand, if it is satisfiable then there exists a counterexample falsifying formula F. We express the counterexample from a satisfying assignment of $\neg F$ as a real-valued vector $w \in \mathbb{R}^p$ (line 14). The vector $v = V \otimes w$ corresponds to the counterexample: its transient l' is greater than l or its cyclicity c' is not the factor of c ($c' \nmid c$). The value for (l, c) is updated w.r.t. l' and c' (line 17) and then generate the QF-RDL formula w.r.t. (4.11) accordingly. This process is repeated until either the SMT solver reports "unsatisfiable" in line 11 or $l + c$ exceeds the maximum bound N (which corresponds to the termination condition of Algorithm 4.1).

Algorithm 4.2 Computation of transient and cyclicity of A w.r.t. $\mathsf{rcone}(V)$ via SMT-solving

Inputs: $A \in \mathbb{R}_{\max}^{n\times n}$, a max-plus cone expressed as a matrix $V \in \mathbb{R}_{\max}^{n\times n}$, positive integer N
Output: a pair of transient and cyclicity

```
1: function MPL_TRANS_CONE_SMT(A, V, N)
2:     χ ← MPL_CTV(A)
3:     if elements of χ are all equal then
4:         n ← ROW(A), p ← COL(V)                          ▷ p is the number of columns of V
5:         for i ∈ {1 ··· p} do
6:             x_i ← SMT_VAR()                              ▷ SMT variables
7:         l ← 0, c ← 1
8:         while (l + c ≤ N) do
9:             F ← EqFunc(A^{⊗l+c} ⊗ V, (λ × c) ⊗ A^{⊗l} ⊗ V)
10:            model ← GET_SMT_MODEL(¬F)
11:            if model = ⊥ then                            ▷ formula is unsatisfiable
12:                return ⟨l, c⟩
13:            else                                         ▷ formula is satisfiable
14:                w ← ⟨model(x_1), ··· model(x_p)⟩         ▷ vector in ℝ^p
15:                v ← V ⊗ w                                ▷ vector in ℝ^n
16:                ⟨l', c'⟩ ← MPL_TRANS_CONE(A, v, N)       ▷ computed by Algorithm 4.1
17:                l ← max{l, l'}, c ← lcm{c, c'}
18:        if ((l + c) > N) then
19:            print "terminated after reaching maximum bound"
20:    else
21:        print "the transient does not exist"
```

Unlike Algorithm 4.1, which only works on max-plus cones, Algorithm 4.2 can be modified (into Algorithm 4.3) so that it can be applied on any set of initial conditions $X \subseteq \mathbb{R}^n$. Although (4.11) is can be translated exclusively to QF-RDL,

we can extend X as a QF-LRA formula. In line 10 of Algorithm 4.3, we generate a formula F which corresponds to the equality checking between $A^{\otimes l}$ and $A^{\otimes l+c}$. If $X \to F$ is valid then for all $\mathbf{x}(0) \in X$ we have $\mathsf{tr}(A, \mathbf{x}(0)) \leq l$ and $\mathsf{cyc}(A, \mathbf{x}(0)) \leq c$. Again, to check the validity of $X \to F$, we check the unsatisfiability of its negation i.e., $X \wedge \neg F$.

Algorithm 4.3 Computation of transient and cyclicity of A w.r.t. a set of initial conditions X via SMT-solving

Inputs: $A \in \mathbb{R}_{\max}^{n\times n}$, a set of initial condition $X \subseteq \mathbb{R}^n$ as QF-LRA formula, positive integer N
Output: a pair of transient and cyclicity

```
1: function MPL_TRANS_SMT(A, X, N)
2:     χ ← MPL_CTV(A)
3:     if elements of χ are all equal then
4:         n ← ROW(A)                                   ▷ number of rows of A
5:         for i ∈ {1 ··· n} do
6:             x_i ← SMT_VAR()                          ▷ symbolic variables
7:         l ← 0, c ← 1
8:         while (l + c) ≤ N do
9:             F ← EqFunc(A^{⊗l+c}, (λ × c) ⊗ A^{⊗l})
10:            model ← GET_SMT_MODEL(X ∧ ¬F)
11:            if model = ⊥ then                        ▷ formula is unsatisfiable
12:                return ⟨l, c⟩
13:            else                                     ▷ formula is satisfiable
14:                v ← ⟨model(x_1), ··· model(x_n)⟩
15:                ⟨l', c'⟩ ← MPL_TRANS_CONE(A, v, N)
16:                l ← max{l, l'}, c ← lcm{c, c'}
17:        if ((l + c) > N) then
18:            print "terminated after reaching maximum bound"
19:    else
20:        print "the transient does not exist"
```

Based on a computational benchmark in [75], the SMT-based procedures (Algorithms 4.2-4.3) are much faster than the existing technique (Algorithm 4.1) when the resulting transient is relatively large. This is due to the fact that, in SMT-based algorithms, the candidates for the transient and cyclicity are not incremented by one (see line 17 of Algorithm 4.2 and 16 of Algorithm 4.3).

4.2.3 A Synthesis Problem

In addition to computing the transient and cyclicity of an MPL system (2.6) w.r.t. a set of initial conditions, we show that by means of SMT over QF-RDL theory, one can synthesise sets of states corresponding to specific transient (and cyclicity) defined as follows

$$\mathcal{S}_{p,q}(A) = \{\mathbf{x} \in \mathbb{R}^n \mid \mathsf{tr}(A, \mathbf{x}) = p, \mathsf{cyc}(A, \mathbf{x}) = q\}, \tag{4.12}$$

$$\mathcal{S}_{p}(A) = \{\mathbf{x} \in \mathbb{R}^n \mid \mathsf{tr}(A, \mathbf{x}) = p\}. \tag{4.13}$$

On the one hand, the computation of (4.13) has been discussed in [5, Section 4.2] by applying backward reachability analysis. On the other hand, to the best of the author's knowledge, there is no approach to generate (4.12). The following

proposition shows that both (4.12) and (4.13) can be computed symbolically by expressing them as QF-RDL formulae: the set (4.12) (resp. (4.13)) is not empty if and only if the corresponding formula (4.14) (resp. (4.15)) is satisfiable.

Proposition 4.4 ([1, Proposition 8])**.** Given $A \in \mathbb{R}_{\max}^{n\times n}$ with global cyclicity c and max-plus eigenvalue λ, we have

$$\mathcal{S}_p(A) = \begin{cases} \mathsf{EqFunc}(A^{\otimes p+c}, \lambda c \otimes A^{\otimes p}), & \text{if } p = 0, \\ \mathsf{EqFunc}(A^{\otimes p+c}, \lambda c \otimes A^{\otimes p}) \wedge \\ \quad \neg\mathsf{EqFunc}(A^{\otimes p-1+c}, \lambda c \otimes A^{\otimes p-1}), & \text{if } p > 0, \end{cases} \tag{4.14}$$

and

$$\mathcal{S}_{p,q}(A) = \begin{cases} \mathsf{EqFunc}(A^{\otimes p+q}, \lambda q \otimes A^{\otimes p}) \wedge \\ \quad \bigwedge\limits_{d\in \mathtt{Div}(q)-\{q\}} \neg\mathsf{EqFunc}(A^{\otimes p+d}, \lambda d \otimes A^{\otimes p}), & \text{if } p = 0, \\ \mathsf{EqFunc}(A^{\otimes p+q}, \lambda q \otimes A^{\otimes p}) \wedge \neg\mathsf{EqFunc}(A^{\otimes p-1+q}, \lambda q \otimes A^{\otimes p-1}) \wedge \\ \quad \bigwedge\limits_{d\in \mathtt{Div}(q)-\{q\}} \neg\mathsf{EqFunc}(A^{\otimes p+d}, \lambda d \otimes A^{\otimes p}), & \text{if } p > 0, \end{cases} \tag{4.15}$$

where $\mathtt{Div}(q)$ is a set of divisors of q.

Proof. Notice that, the formula $\mathsf{EqFunc}(A^{\otimes p+c}, \lambda c \otimes A^{\otimes p})$ corresponds to a periodic behavior at bound p and $p + c$ with cyclicity at most c. More precisely, if $\mathbf{x}(0)$ satisfies $\mathsf{EqFunc}(A^{\otimes p+c}, \lambda c \otimes A^{\otimes p})$ then $\mathbf{x}(p + c) = \lambda c \otimes \mathbf{x}(p)$ or in other words $\mathsf{tr}(A, \mathbf{x}(0)) \leq p$. On the other hand, if $\mathbf{x}(0)$ satisfies $\neg\mathsf{EqFunc}(A^{\otimes p-1+c}, \lambda c \otimes A^{\otimes p-1})$ then $\mathsf{tr}(A, \mathbf{x}(0)) > p - 1$.

Similarly, $\mathsf{EqFunc}(A^{\otimes p+q}, \lambda q \otimes A^{\otimes p}) \wedge \neg\mathsf{EqFunc}(A^{\otimes p-1+q}, \lambda q \otimes A^{\otimes p-1})$ corresponds to a periodic behavior with transient p and cyclicity at most q. The remaining conjuncts guarantee that the cyclicity cannot be smaller than q. □

As both (4.12) and (4.13) can be expressed as formulae in QF-RDL, the problem of determining the emptiness of both sets is decidable. By definition, for *never periodic* MPL system, $\mathcal{S}_{p,q} = \mathcal{S}_p = \emptyset$ for all p, q. Furthermore, for irreducible MPL systems, the emptiness of (4.12) and (4.13) is related to the global transient and cyclicity of A.

Proposition 4.5 ([1, Proposition 9])**.** For an irreducible matrix $A \in \mathbb{R}_{\max}^{n\times n}$ with global transient l and cyclicity c we have $\mathcal{S}_0(A) = E(A^{\otimes c})$ and $\mathcal{S}_{0,1}(A) = E(A)$. Furthermore,

i. $\mathcal{S}_p(A) \neq \emptyset$ iff $p \leq l$,

ii. If $p > l$ or q is not a divisor of c then $\mathcal{S}_{p,q}(A) = \emptyset$,

iii. If $\mathcal{S}_{p,q}(A)$ is empty then so is $\mathcal{S}_{p+1,q}(A)$.

Proof. Let us assume the eigenvalue for A is λ. Notice that, $\mathbf{x}(0) \in \mathcal{S}_0(A)$ if and only if $\mathbf{x}(c) = (\lambda \times c) \otimes \mathbf{x}(0)$, or equivalently $A^{\otimes c} \otimes \mathbf{x}(0) = (\lambda \times c) \otimes \mathbf{x}(0)$. This shows that $\mathbf{x}(0)$ is an eigenvector of $A^{\otimes c}$. Thus, $\mathsf{S}_0(A) = E(A^{\otimes c})$. The proof for $\mathcal{S}_{0,1}(A) = E(A)$ can obtained similarly.

i. As the global transient for A is k_0, it is by default that $\mathcal{S}_p(A) = \emptyset$ for $p > k_0$. Suppose $\mathbf{x}(0) \in \mathbb{R}^n$ such that $\mathtt{tr}(A, \mathbf{x}(0)) = k_0$. It is straightforward that $\mathbf{x}(p) \in \mathcal{S}_{p-k_0}(A)$ for $p \leq k_0$. This completes the proof.

ii. If $p > k_0$ we have $\mathcal{S}_p(A) = \emptyset$ which implies $\mathcal{S}_{p,q} = \emptyset$. Suppose q is not a divisor of c and $\mathcal{S}_{p,q} \neq \emptyset$. Then there exists $\mathbf{x}(0)$ such that $\mathtt{cyc}(A, \mathbf{x}(0)) = \mathtt{lcm}(c, q) > c$. This contradicts the fact that c is the global cyclicity.

iii. The proof is from the fact that if $\mathbf{x} \in \mathcal{S}_{p+1,q}(A)$ then $A \otimes \mathbf{x} \in \mathcal{S}_{p,q}(A)$. □

Example 4.1. Let us consider an MPL system in Example 2.1. The corresponding matrix A in (2.7) has transient $l = 2$ and cyclicity $c = 2$. Without giving the details, the non-empty sets w.r.t. (4.12) are

$$\mathcal{S}_0(A) = \{\mathbf{x} \in \mathbb{R}^2 \mid 0 \leq x_1 - x_2 \leq 2\}, \quad \mathcal{S}_1(A) = \{\mathbf{x} \in \mathbb{R}^2 \mid x_1 - x_2 < 0\},$$
$$\mathcal{S}_2(A) = \{\mathbf{x} \in \mathbb{R}^2 \mid x_1 - x_2 > 2\}.$$

On the other hand, the sets w.r.t. (4.13) are

$$\mathcal{S}_{0,1}(A) = \{\mathbf{x} \in \mathbb{R}^2 \mid x_1 - x_2 = 1\}, \quad \mathcal{S}_{1,2}(A) = \mathcal{S}_1(A), \quad \mathcal{S}_{2,2}(A) = \mathcal{S}_2(A),$$
$$\mathcal{S}_{0,2}(A) = \{\mathbf{x} \in \mathbb{R}^2 \mid 0 \leq x_1 - x_2 < 1 \text{ or } 1 < x_1 - x_2 \leq 2 \}.$$

□

4.3 Extension to Min-Plus Linear Systems

This section discusses briefly the extension of the procedures in Section 4.2 to be applied for MiPL systems. Due to the duality between $\oplus$ and $\oplus'$ operations, the inequalities over min-plus algebra can also be expressed as QF-RDL formulae.

Proposition 4.6. Given real variables $\mathtt{x}_1, \ldots, \mathtt{x}_p$ and min-plus scalars $a_1, \ldots, a_p$, $b_1, \ldots, b_p \in \mathbb{R}_{\min}$, the inequality

$$F \equiv \bigoplus_{i=1}^{p}{}'(\mathtt{x}_i + a_i) \sim \bigoplus_{j=1}^{p}{}'(\mathtt{x}_j + b_j) \tag{4.16}$$

is equivalent to

$$F^* \equiv \bigoplus_{i \in S_1}{}'(\mathtt{x}_i + a_i) \sim \bigoplus_{j \in S_2}{}'(\mathtt{x}_j + b_j), \tag{4.17}$$

where $S_1 = \{1, \ldots, p\} \setminus \{1 \leq k \leq p \mid a_k = \xi \text{ or } a_k \sim b_k\}$ and $S_2 = \{1, \ldots, p\} \setminus \{1 \leq k \leq p \mid b_k = \xi \text{ or } \neg(a_k \sim b_k)\}$, respectively. Furthermore, (4.17) can be expressed as

$$F^* \equiv \bigwedge_{i \in S_1} \left(\bigvee_{j \in S_2} (\mathtt{x}_i - \mathtt{x}_j \sim b_j - a_i) \right) \equiv \bigvee_{j \in S_2} \left(\bigwedge_{i \in S_1} (\mathtt{x}_i - \mathtt{x}_j \sim b_j - a_i) \right). \tag{4.18}$$

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

Proof. Notice that by switching the sides and due to the duality between $\oplus$ and $\oplus'$, the min-plus inequality (4.16) is equivalent to max-plus inequality $\bigoplus_{j=1}^{p}(-\mathtt{x}_j + (-b_j)) \sim \bigoplus_{i=1}^{p}(-\mathtt{x}_i + (-a_i))$. Then, by Proposition 4.2, it can be reduced into

$$\bigoplus_{j\in R_1}(-\mathtt{x}_j + (-b_j)) \sim \bigoplus_{i\in R_2}^{p}(-\mathtt{x}_i + (-a_i)), \tag{4.19}$$

where $R_1 = \{1,\ldots,p\} \setminus \{1 \le k \le p \mid -b_k = \varepsilon \text{ or } \neg(-b_k \sim -a_k)\}$ and $R_2 = \{1,\ldots,p\} \setminus \{1 \le k \le p \mid -a_k = \varepsilon \text{ or } -b_k \sim -a_k\}$. The set R_1 (resp. R_2) is indeed equivalent to S_2 (resp. S_1). After switching the sides of (4.19), one can obtain (4.17). Now, by Proposition 4.3, (4.19) can be expressed as

$$\bigwedge_{i\in R_2}\left(\bigvee_{j\in R_1}(-\mathtt{x}_j - (-\mathtt{x}_i) \sim -a_i - (-b_j)\right) \equiv \bigvee_{j\in R_1}\left(\bigwedge_{i\in R_2}(-\mathtt{x}_j - (-\mathtt{x}_i) \sim -a_i - (-b_j)\right),$$

which is equaivalent to (4.18). If $R_1 = S_2 = \emptyset$ then $F^* \equiv \mathtt{false}$. On the other hand, if $R_2 = S_1 = \emptyset$ then $F^* \equiv \mathtt{true}$. □

Algorithms 4.4-4.5 summarise the SMT-based computation of transient and cyclicity from an MiPL system. There are several modifications from Algorithms 4.2-4.3: 1) the computation of cycle-time vector in line 2; 2) the generation of QF-RDL formula in line 9; and 3) the computation of l', c' in line 16 of Algorithm 4.4 and line 15 of Algorithm 4.5. It is important to note that the QF-RDL formula in line 9 of Algorithms 4.4-4.5 is generated by (4.18).

It is also straightforward that one can synthesise the set of states corresponding to transient and cyclicity from an MiPL system $\mathbf{x}(k) = B \otimes' \mathbf{x}(k-1)$ as follows

$$\mathcal{S}_{p,q}(B) = \{\mathbf{x} \in \mathbb{R}^n \mid \mathsf{tr}_{\min}(B,\mathbf{x}) = p, \mathsf{cyc}_{\min}(B,\mathbf{x}) = q\}, \tag{4.20}$$

$$\mathcal{S}_{p}(B) = \{\mathbf{x} \in \mathbb{R}^n \mid \mathsf{tr}_{\min}(B,\mathbf{x}) = p\}. \tag{4.21}$$

The generation of the above sets can be done similarly using Propositions 4.4-4.5.

Corollary 4.1. Given $B \in \mathbb{R}_{\min}^{n\times n}$ with global cyclicity c and min-plus eigenvalue λ, we have

$$\mathcal{S}_p(B) = \begin{cases} \mathsf{EqFunc}(B^{\otimes' p+c}, \lambda c \otimes B^{\otimes' p}), & \text{if } p = 0, \\ \mathsf{EqFunc}(B^{\otimes' p+c}, \lambda c \otimes B^{\otimes' p}) \wedge & \\ \quad \neg\mathsf{EqFunc}(B^{\otimes' p-1+c}, \lambda c \otimes B^{\otimes' p-1}), & \text{if } p > 0, \end{cases} \tag{4.22}$$

and

$$\mathcal{S}_{p,q}(B) = \begin{cases} \mathsf{EqFunc}(B^{\otimes' p+q}, \lambda q \otimes B^{\otimes' p}) \wedge & \\ \quad \bigwedge_{d\in \mathtt{Div}(q)-\{q\}} \neg\mathsf{EqFunc}(B^{\otimes' p+d}, \lambda d \otimes B^{\otimes' p}), & \text{if } p = 0, \\ \mathsf{EqFunc}(B^{\otimes' p+q}, \lambda q \otimes B^{\otimes' p}) \wedge \neg\mathsf{EqFunc}(B^{\otimes' p-1+q}, \lambda q \otimes B^{\otimes' p-1}) \wedge & \\ \quad \bigwedge_{d\in \mathtt{Div}(q)-\{q\}} \neg\mathsf{EqFunc}(B^{\otimes' p+d}, \lambda d \otimes B^{\otimes p}), & \text{if } p > 0, \end{cases} \tag{4.23}$$

where $\mathtt{Div}(q)$ is a set of divisors of q. □

Algorithm 4.4 Computation of transient and cyclicity of $B \in \mathbb{R}_{\min}^{n\times n}$ w.r.t. $\mathsf{rcone}(V)$ via SMT-solving

Inputs: $B \in \mathbb{R}_{\min}^{n\times n}$, a min-plus cone expressed as a matrix $V \in \mathbb{R}_{\min}^{n\times n}$, positive integer N
Output: a pair of transient and cyclicity

function MIPL_TRANS_CONE_SMT(B, V, N)
  $\chi \leftarrow$ MIPL_CTV(B)
  if elements of χ are all equal **then**
    $n \leftarrow$ ROW(B), $p \leftarrow$ COL(V) ▷ p is the number of columns of V
    for $i \in \{1 \cdots p\}$ **do**
      $\mathtt{x}_i \leftarrow$ SMT_VAR() ▷ SMT variables
    $l \leftarrow 0, c \leftarrow 1$
    while $(l + c \leq N)$ **do**
      $F \leftarrow \mathtt{EqFunc}(B^{\otimes' l+c} \otimes V, (\lambda \times c) \otimes B^{\otimes' l} \otimes' V)$
      $model \leftarrow$ GET_SMT_MODEL($\neg F$)
      if $model = \bot$ **then** ▷ formula is unsatisfiable
        return $\langle l, c\rangle$
      else ▷ formula is satisfiable
        $w \leftarrow \langle model(\mathtt{x}_1), \cdots model(\mathtt{x}_p)\rangle$ ▷ vector in $\mathbb{R}^p$
        $v \leftarrow V \otimes' w$ ▷ vector in $\mathbb{R}^n$
        $\langle l', c'\rangle \leftarrow$ MPL_TRANS_CONE($-B$, $-v, N$) ▷ computed by
        $l \leftarrow \max\{l, l'\}, c \leftarrow \mathtt{lcm}\{c, c'\}$ Algorithm 4.1
    if $((l + c) > N)$ **then**
      print "terminated after reaching maximum bound"
  else
    print "the transient does not exist"

Algorithm 4.5 Computation of transient and cyclicity of $B \in \mathbb{R}_{\min}^{n\times n}$ w.r.t. a set of initial conditions X via SMT-solving

Inputs: $B \in \mathbb{R}_{\min}^{n\times n}$, a set of initial condition $X \subseteq \mathbb{R}^n$ as QF-LRA formula, positive integer N
Output: a pair of transient and cyclicity

function MIPL_TRANS_SMT(B, X, N)
  $\chi \leftarrow$ MIPL_CTV(B)
  if elements of χ are all equal **then**
    $n \leftarrow$ ROW(B) ▷ number of rows of B
    for $i \in \{1 \cdots n\}$ **do**
      $\mathtt{x}_i \leftarrow$ SMT_VAR() ▷ symbolic variables
    $l \leftarrow 0, c \leftarrow 1$
    while $(l + c) \leq N$ **do**
      $F \leftarrow \mathtt{EqFunc}(B^{\otimes' l+c}, (\lambda \times' c) \otimes' B^{\otimes' l})$
      $model \leftarrow$ GET_SMT_MODEL($X \wedge \neg F$)
      if $model = \bot$ **then** ▷ formula is unsatisfiable
        return $\langle l, c\rangle$
      else ▷ formula is satisfiable
        $v \leftarrow \langle model(\mathtt{x}_1), \cdots model(\mathtt{x}_n)\rangle$
        $\langle l', c'\rangle \leftarrow$ MPL_TRANS_CONE($-B$, $-v, N$) ▷ computed by
        $l \leftarrow \max\{l, l'\}, c \leftarrow \mathtt{lcm}\{c, c'\}$ Algorithm 4.1
    if $((l + c) > N)$ **then**
      print "terminated after reaching maximum bound"
  else
    print "the transient does not exist"

Corollary 4.2. For an irreducible matrix $B \in \mathbb{R}_{\min}^{n\times n}$ with global transient l and cyclicity c we have $\mathcal{S}_0(B) = E(B^{\otimes' c})$ and $\mathcal{S}_{0,1}(B) = E(B)$. Furthermore,

i. $\mathcal{S}_p(B) \neq \emptyset$ iff $p \leq l$,

ii. If $p > l$ or q is not a divisor of c then $\mathcal{S}_{p,q}(B) = \emptyset$,

iii. If $\mathcal{S}_{p,q}(B)$ is empty then so is $\mathcal{S}_{p+1,q}(B)$. □

Remark 4.2. We do not present the extension of the notions above to IMPL systems, since the transient cannot be easily defined for those models. Furthermore, it is possible that the orbits w.r.t. IMPL systems may never be periodic (see Example 2.4).

4.4 Summary

This chapter has discussed a novel technique to compute the transient and cyclicity of MPL systems by utilising SMT solving. Unlike the state-of-the-art technique, which works only for max-plus cones, the SMT-based procedure can be applied on any set of initial conditions expressed as a QF-LRA formula. Furthermore, it can be used to partition the state-space of MPL systems w.r.t. a given transient and cyclicity pair. It has been shown that the proposed procedure can be applied to MiPL systems.

Chapter 5

Reachability Analysis of Max-Plus Linear Systems

In this chapter, we elaborate on reachability analysis for MPL systems. Given a set of initial and target sets, we develop procedures to determine whether there exist orbits of MPL system that, starting from the initial set, eventually reach the target set. We first discuss the existing methods in [5, 6] which explicitly compute the reach sets at each time horizon. Such computation can be done compactly, thanks to the abstraction of MPL systems and manipulations of DBM. Furthermore, similar procedures are extendable to MiPL and IMPL systems. The extension to the latter models is firstly described in [27]. The main drawback of these methods is that they can be applied only to the models with a small dimension since generating the abstraction and computing the reach sets runs exponentially.

We then propose novel procedures by employing SMT checking. Instead of computing reach sets explicitly, we use SMT instances to encode the states of orbits of MPL systems and the initial and target sets into a QF-RDL formula and then check the satisfaction of the resulting logical formula via an off-the-shelf SMT solver. Similarly, the orbits of IMPL systems can be encoded into a QF-LRA formula. We also develop procedures to solve *quantified reachability analysis* by allowing quantifiers (universal and existential) over initial conditions and the sequence of state matrices. Based on the computational benchmarks, the performance and scalability of the SMT-based algorithms are shown to outperform state-of-the-art techniques, newly allowing to investigate reachability analysis of high-dimensional MPL, MiPL and IMPL systems.

5.1 Explicit Reachability Analysis of Max-Plus Linear Systems

This section discusses the reachability analysis (RA) problem for MPL systems. We first present the existing procedures to solve the bounded RA. Then, we describe the completeness threshold for the unbounded RA, which is related to the pair of transient and cyclicity of the underlying MPL system.

Problem 5.1. Suppose we have an MPL system (2.6), $X_0, Y_0 \subseteq \mathbb{R}^n$ respectively as the initial and target sets and a positive integer N. The *bounded reachability analysis* refers to the problem of determining whether Y_0 is reachable in at most N-steps from X_0: there exists $\mathbf{x}(0) \in X_0$ and $1 \leq k \leq N$ such that $\mathbf{x}(k) = A \otimes \mathbf{x}(k-1) \in Y_0$. It is assumed[7] that $X_0 \cap Y_0 = \emptyset$ and both X_0 and Y_0 can be expressed as the union of finitely many DBM. □

Problem 5.2. Suppose we have an MPL system (2.6), $X_0, Y_0 \subseteq \mathbb{R}^n$ respectively as the initial and target sets. The *unbounded reachability analysis* refers to the problem of determining whether there exists $N > 0$ such that Y_0 is reachable from X_0 at N-steps. □

The procedures to solve Problem 5.1 has been discussed in [5, 6] by explicitly computing the reach sets. From an initial set X_0, the forward reach set X_k is recursively defined as

$$X_k = \texttt{Im}(A, X_{k-1}) = \{A \otimes \mathbf{x} \mid \mathbf{x} \in X_{k-1}\}. \tag{5.1}$$

Likewise, from the target set Y_0, the backward reach set Y_{-k} is defined as

$$Y_{-k} = \texttt{Inv}(A, Y_{-k+1}) = \{\mathbf{y} \in \mathbb{R}^n \mid A \otimes \mathbf{y} \in Y_{-k+1}\}. \tag{5.2}$$

Alternatively, the forward and backward reach sets can be computed using one-shot procedures as follows:

$$X_k = \texttt{Im}(A^k, X_0) = \{A^{\otimes k} \otimes \mathbf{x} \mid \mathbf{x} \in X_0\}, \tag{5.3}$$

and

$$Y_{-k} = \texttt{Inv}(A^k, Y_0) = \{\mathbf{y} \in \mathbb{R}^n \mid A^{\otimes k} \otimes \mathbf{y} \in Y_0\}. \tag{5.4}$$

To compute forward and backward reach sets, one needs to represent an MPL system (2.6) as a PWA representation which can be done by Algorithm 3.1. Furthermore, since the initial and target sets are the union of DBM, the forward and backward reach set can be computed by Algorithm 2.5 and Algorithm 2.6, respectively. Notice that, for the forward reach sets, $X_k \neq \emptyset$ for $k \geq 0$. For the backward reach sets, it is possible that there is an $l > 0$ such that $Y_{-k} = \emptyset$ for all $k \geq l$.

Algorithms 5.1-5.4 illustrate the steps to solve Problem 5.1 by means of the computation of forward and backward reach sets up to a given bound $N \in \mathbb{N}$. We refer Algorithms 5.1 and 5.3 as the "sequential" ones since we successively compute the reach sets. In Algorithms 5.2 and 5.4, one needs to generate the PWA system for $A^{\otimes k}$ for each iteration k. Notice that these "one-shot" implementations do not simply compute the reach set at the final time N, as they still run over the entire time horizon from 1 to N. For Algorithms 5.3-5.4, if the backward reach set $Y_{-k} = \emptyset$, then the algorithms are terminated at the k-th iteration with `false` as the output.

[7] The first assumption is applied to exclude zero-step reachability analysis which can be considered as a trivial case.

Algorithm 5.1 RA of MPL systems (forward)

Inputs: $A \in \mathbb{R}_{\max}^{n\times n}$, initial and target sets $X_0, Y_0 \in \mathbb{R}^n$, $N \in \mathbb{N}$,
Output: a Boolean value

1: $reach \leftarrow$ `false`
2: generate PWA system of A
3: $k \leftarrow 1$
4: **while** $k \leq N$ **do**
5: compute X_k by (5.1)
6: **if** $X_k \cap Y_0 \neq \emptyset$ **then**
7: $reach \leftarrow$ `true`
8: **break**
9: $k \leftarrow k + 1$
10: **return** *reach*

Algorithm 5.2 RA of MPL systems (one-shot forward)

Inputs: $A \in \mathbb{R}_{\max}^{n\times n}$, initial and target sets $X_0, Y_0 \in \mathbb{R}^n$, $N \in \mathbb{N}$,
Output: a Boolean value

1: $reach \leftarrow$ `false`
2: $k \leftarrow 1$
3: **while** $k \leq N$ **do**
4: generate PWA system of $A^{\otimes k}$
5: compute X_k by (5.3)
6: **if** $X_k \cap Y_0 \neq \emptyset$ **then**
7: $reach \leftarrow$ `true`
8: **break**
9: $k \leftarrow k + 1$
10: **return** *reach*

Algorithm 5.3 RA of MPL systems (backward)

Inputs: $A \in \mathbb{R}_{\max}^{n\times n}$, initial and target sets $X_0, Y_0 \in \mathbb{R}^n$, $N \in \mathbb{N}$,
Output: a Boolean value

1: $reach \leftarrow$ `false`
2: generate PWA system of A
3: $k \leftarrow 1$
4: **while** $k \leq N$ **do**
5: compute Y_{-k} by (5.2)
6: **if** $Y_{-k} = \emptyset$ **then**
7: **break**
8: **if** $Y_{-k} \cap X_0 \neq \emptyset$ **then**
9: $reach \leftarrow$ `true`
10: **break**
11: $k \leftarrow k + 1$
12: **return** *reach*

Algorithm 5.4 RA of MPL systems (one-shot backward)

Inputs: $A \in \mathbb{R}_{\max}^{n\times n}$, initial and target sets $X_0, Y_0 \in \mathbb{R}^n$, $N \in \mathbb{N}$,
Output: a Boolean value

1: $reach \leftarrow$ `false`
2: $k \leftarrow 1$
3: **while** $k \leq N$ **do**
4: generate PWA system of $A^{\otimes k}$
5: compute Y_{-k} by (5.4)
6: **if** $Y_{-k} = \emptyset$ **then**
7: **break**
8: **if** $Y_{-k} \cap X_0 \neq \emptyset$ **then**
9: $reach \leftarrow$ `true`
10: **break**
11: $k \leftarrow k + 1$
12: **return** *reach*

There are a few elements contributing to the computational bottleneck (time and memory requirements) of the above algorithms. First of all, the worst-case complexity of generating the PWA system by Algorithm 3.1 is $\mathcal{O}(n^{n+3})$ where n is the dimension of the underlying MPL system. We recall that based on the benchmark in Table 3.1, Algorithm 3.1 fails to generate the PWA representation of 20-dimensional MPL systems within reasonable memory limit. Furthermore, the forward reachable sets X_k and backward reachable sets Y_{-k} are a union of finitely many DBMs. In the worst case, the number of DBMs grows exponentially with the time horizon. As shown in [6], the worst-case complexity to generate the sequential (resp. one-shot) forward reach sets up to bound N is $\mathcal{O}(\sum_{k=0}^{N-1} |X_k| \cdot n^{n+3})$ (resp. $\mathcal{O}((\lfloor \log_2 N \rfloor + |X_0|) \cdot n^{n+3})$), where $|X_k|$ represents the number of DBM in X_k. Similarly, the complexity for backward reach sets computations are $\mathcal{O}(\sum_{k=0}^{N-1} |Y_{-k}| \cdot n^{n+3})$ (for sequential) and $\mathcal{O}((\lfloor \log_2 N \rfloor + |Y_0|) \cdot n^{n+3})$ (for one-shot).

We now will discuss how to solve the unbounded RA of MPL systems. It is straightforward that Algorithms 5.1-5.4 are sound and complete since we can always terminate with the correct output (`true` or `false`): hence, Problem 5.1 is decidable. If the output for Problem 5.1 is `true` then so is for Problem 5.2. On the other hand, given a negative outcome from the algorithms, in general, we cannot conclude that Y_0 is not reachable from X_0 within time bounds greater than N. If the underlying MPL system (2.6) is irreducible and DBM in X_0 and Y_0 do not contain any inequality with a single variable (see Definition 2.12) then there exists a *completeness threshold* $N^* \in \mathbb{N}$ for Problem 5.2: if Y_0 is not reachable from X_0 up to bound N^* then Y_0 is surely also not reachable from X_0 within any larger bound $N > N^*$. Such completeness threshold is determined by the pair of transient and cyclicity.

Proposition 5.1 ([75])**.** Suppose we have an irreducible MPL system (2.6), an initial set X_0, and a target set Y_0. If DBM in X_0 and Y_0 do not contain any inequality with a single variable then Problem 5.2 is decidable with completeness threshold equals to $\mathsf{tr}(A) + \mathsf{cyc}(A) - 1$.

Proof. Since the underlying MPL system is irreducible, then there exist global transient $\mathsf{tr}(A) = l$ and cyclicity $\mathsf{cyc}(A) = c$. Suppose we have forward reach sets $X_0, X_1, \ldots$. By Proposition 2.2, for $k \geq l$ we have $A^{\otimes(k+c)} = (\lambda \times c) \otimes A^{\otimes k}$, which implies $\mathbf{x} \in X_k$ iff $(\lambda \times c) \otimes \mathbf{x} \in X_{k+c}$. Recall that the forward reach sets are in general unions of DBM. Furthermore, a DBM without a single-variable inequality is not affected by shifting operations: given a DBM D and $\alpha \in \mathbb{R}$, $\alpha \otimes D = \{\alpha \otimes \mathbf{x} \mid \mathbf{x} \in D\} = D$. Consequently, $X_{k+c} = X_k$ for $k \geq l$. From here, we can conclude that we only need to consider a bound before reaching the periodicity, i.e. $l + c - 1$. As a result, Problem 5.2 is dedicable. □

In this thesis, we show that Proposition 5.1 can be relaxed by allowing X_0 to have inequalities with a single variable. Furthermore, Problem 5.2 remains decidable for reducible (or, *unboundedly periodic*) MPL systems as long as the local transient $\mathsf{tr}(A, X_0)$ exists. As such, we may obtain a smaller completeness threshold from the local transient and cyclicity w.r.t. X_0.

Proposition 5.2. Suppose we have an MPL system (2.6), an initial set X_0, and a target set Y_0. If DBM in Y_0 do not contain any inequality with a single variable and $\mathsf{tr}(A, X_0)$ exists then Problem 5.2 is decidable with the corresponding completeness threshold equals to $\mathsf{tr}(A, X_0) + \mathsf{cyc}(A, X_0) - 1$

Proof. Let us denote $\mathsf{tr}(A, X_0) = l$ and $\mathsf{cyc}(A, X_0) = c$. For each $\mathbf{x}(0) \in X_0$ and $k \geq l$, we have $\mathbf{x}(k + c) = (\lambda \times c) \otimes \mathbf{x}(k)$ where λ is the max-plus eigenvalue of A. Suppose Y_0 is not reachable from X_0 up to bound $N = l + c - 1$: for each $\mathbf{x}(0) \in X_0$ and $\forall 1 \leq k \leq N$ we have $\mathbf{x}(k) \notin Y_0$. Since Y_0 does not contain any inequality with a single variable and $\mathbf{x}(k + c) = (\lambda \times c) \otimes \mathbf{x}(k)$ for $k \geq l$, it is straightforward that $\mathbf{x}(k + c) \notin Y_0$ for $k \geq l$. Hence, completeness threshold for Problem 5.2 is $l + c - 1$. Therefore, Problem 5.2 is indeed decidable. □

Example 5.1. Let us consider an MPL systems in (2.7). Suppose we define the initial and target sets respectively as $X_0 = \{\mathbf{x} \in \mathbb{R}^2 \mid x_1 - x_2 \geq 3\}$ and $Y_0 = \{\mathbf{x} \in \mathbb{R}^2 \mid x_1 - x_2 \geq 5\}$. Recall that the transient and cyclicity of (2.7) are $\mathsf{tr}(A) = \mathsf{cyc}(A) = 2$

and therefore the completeness threshold is $N^* = 3$. Leaving details aside, the forward reach sets are $X_1 = \{\mathbf{x} \in \mathbb{R}^2 \mid x_1 - x_2 = -1\}, X_2 = \{\mathbf{x} \in \mathbb{R}^2 \mid x_1 - x_2 = 0\}$, and $X_3 = \{\mathbf{x} \in \mathbb{R}^2 \mid x_1 - x_2 = 2\}$. Since $X_i \cap Y_0 = \emptyset$ for $i = 1, 2, 3$, we can conclude that Y_0 is not reachable from X. By backward reach set computation, we have $Y_1 = \emptyset$ which leads to the same conclusion. □

5.1.1 Extension to Other Models

This section describes how to apply Algorithms 5.1-5.4 to solve the reachability analysis of MiPL and IMPL systems.

Min-Plus Linear Systems

The extension of Algorithms 5.1-5.4 to MiPL systems is rather straightforward. First of all, the generation of the PWA system from an MiPL system (2.10) can be done similarly using Algorithm 3.5. Likewise, the computation of forward and backward reach sets (5.1)-(5.4) can be applied to MiPL systems due to Remark 2.3. As a consequence, Proposition 5.2 also holds for MiPL systems.

Corollary 5.1. Suppose we have an MiPL system (2.10), an initial set X_0, and a target set Y_0. If DBM in Y_0 do not contain any inequality with a single variable and $\mathsf{tr}_{\min}(B, X_0)$ exists then the unbounded RA (for MiPL systems) is decidable with completeness threshold equals to $\mathsf{tr}_{\min}(B, X_0) + \mathsf{cyc}_{\min}(B, X_0) - 1$. □

Interval Max-Plus Linear Systems

The explicit procedures to solve bounded reachability analysis for IMPL systems have been discussed in [26, 27]. As for MPL systems, they employ the computation of forward and backward reach sets from an IMPL system (2.14)

$$X_k = \mathtt{Im}(\mathbf{A}, X_{k-1}) = \{A \otimes \mathbf{x} \mid A \in \mathbf{A}, \mathbf{x} \in X_{k-1}\}, \tag{5.5}$$
$$Y_{-k} = \mathtt{Inv}(\mathbf{A}, Y_{-k+1}) = \{\mathbf{y} \in \mathbb{R}^n \mid \exists A \in \mathbf{A} \text{ such that } A \otimes \mathbf{y} \in Y_{-k+1}\}. \tag{5.6}$$

The sequential procedures to solve bounded RA of IMPL systems are similar to Algorithms 5.1-5.2 but with slight modifications. First, generating the PWA representations from an IMPL system can be done from its upper matrix $\overline{A}$. Second, the computation of reach sets (5.5)-(5.6) are implemented via Algorithms 2.7-2.8 (not Algorithms 2.5-2.6 that are used for computing (5.1)-(5.2)).

Remark 5.1. The one-shot procedures are also proposed in [26] by computing

$$W_k = \mathtt{Im}(\mathbf{A}^{\otimes k}, X_0) = \{A \otimes \mathbf{x} \mid A \in \mathbf{A}^{\otimes k}, \mathbf{x} \in X_0\}, \tag{5.7}$$
$$Z_{-k} = \mathtt{Inv}(\mathbf{A}^{\otimes k}, Y_0) = \{\mathbf{y} \in \mathbb{R}^n \mid \exists A \in \mathbf{A}^{\otimes k} \text{ such that } A \otimes \mathbf{y} \in Y_0\}. \tag{5.8}$$

Recall that the power of an interval matrix $\mathbf{A}$ is done by taking the power of the lower and upper matrices, i.e., $\mathbf{A}^{\otimes k} = [\underline{A}^{\otimes k}, \overline{A}^{\otimes k}]$. However, we argue that the resulting reach sets in (5.7)-(5.8) are not equal to their sequential counterparts (5.5)-(5.6). It is true that $X_k \subseteq W_k$ and $Y_{-k} \subseteq Z_{-k}$. However, the inclusion relations $W_k \subseteq X_k$

and $Z_{-k} \subseteq Y_{-k}$ in general do not hold. Given $A \in [\underline{A}^{\otimes k}, \overline{A}^{\otimes k}]$, it is not always the case that there exist $A_1, \ldots, A_k \in [\underline{A}, \overline{A}]$ such that $A_1 \otimes \ldots \otimes A_k = A$. The supporting example of this argument is presented in [76, Remark 2].

As for MPL and MiPL systems, the bounded RA for IMPL systems is decidable. However, the unbounded RA of IMPL systems remains undecidable since the orbits may never be periodic even though the global transient for both lower and upper matrices exist.

5.2 SMT-Based Reachability Analysis of Max-Plus Linear Systems

This section discusses the alternative procedures to solve Problem 5.1 by SMT solving. The main idea underpinning the novel methods is to transform the problem into an SMT instance. The following proposition shows that the dynamics of MPL systems (2.6) can be symbolically expressed as a QF-RDL formula.

Proposition 5.3 ([75]). Given real-valued variables $\mathtt{x}_1, \ldots, \mathtt{x}_n$ and real scalars $a_1, \ldots, a_n$, the equation $\mathtt{x}' = \bigoplus_{i=1}^{n} (\mathtt{x}_i \otimes a_i)$ is equivalent to

$$\left(\bigwedge_{i=1}^{n} (\mathtt{x}' - \mathtt{x}_i \geq a_i) \right) \wedge \left(\bigvee_{i=1}^{n} (\mathtt{x}' - \mathtt{x}_i = a_i) \right). \tag{5.9}$$

Proof. The QF-RDL formula (5.9) asserts that: 1) $\forall i \ \mathtt{x}' \geq \mathtt{x}_i + a_i$ and 2) $\exists i \ \mathtt{x}' = \mathtt{x}_i + a_i$. From both conditions, it is straigthforward to see that $\mathtt{x}'$ can be expressed as $\max\{\mathtt{x}_1 + a_1, \ldots, \mathtt{x}_n + a_n\}$. □

By Proposition 5.3, any MPL system in (2.6) can be expressed as a formula in QF-RDL as follows:

$$\bigwedge_{i=1}^{n} \left(\left(\bigwedge_{j \in \mathtt{fin}_i} \mathtt{x}_i^{(k)} - \mathtt{x}_j^{(k-1)} \geq A(i,j) \right) \wedge \left(\bigvee_{j \in \mathtt{fin}_i} \mathtt{x}_i^{(k)} - \mathtt{x}_j^{(k-1)} = A(i,j) \right) \right), \tag{5.10}$$

where $\mathtt{fin}_i$ is a set containing the indices of the finite elements of $A(i, \cdot)$. For the sake of simplicity, we denote (5.10) as $\mathtt{SymbMPL}(A, \mathcal{V}^{(k-1)}, \mathcal{V}^{(k)})$ where $\mathcal{V}^{(k)} = \{\mathtt{x}_1^{(k)}, \ldots, \mathtt{x}_n^{(k)}\}$ is the set of (symbolic) variables encompassing the states of the MPL system in (2.6) at time horizon k.

Consequently, the following QF-RDL formula

$$F = \bigwedge_{k=1}^{N} \mathtt{SymbMPL}(A, \mathcal{V}^{(k-1)}, \mathcal{V}^{(k)}) \tag{5.11}$$

corresponds to a *symbolic representation* of the orbits of MPL system (2.6) from time horizon 0 to k. Furthermore, notice that any DBM can be translated[8] into a

[8] A DBM without any single-variable inequality can be translated into a QF-RDL formula.

QF-LRA (which contains QF-RDL) formula, where Boolean connectives are exclusively conjunctions ($\wedge$): as such, the non-emptiness of a DBM is equivalent to the satisfiability of its corresponding QF-LRA formula.

It follows that the reachability of the target set Y_0 from the initial set X_0 up to bound N can be equivalently expressed as the satisfiability of

$$X^{(0)} \wedge F \wedge \bigvee_{k=1}^{N} Y^{(k)}, \tag{5.12}$$

where $X^{(0)}$ (resp. $Y^{(k)}$) is the QF-LRA representation for X_0 (resp. Y_0) over $\mathcal{V}^{(0)}$ (resp. $\mathcal{V}^{(k)}$). Furthermore, the one-shot approach to reachability analysis can be formulated from (5.11) as follows:

$$X^{(0)} \wedge \left(\bigvee_{k=1}^{N} \texttt{SymbMPL}(A^{\otimes k}, \mathcal{V}^{(0)}, \mathcal{V}^{(1)}) \right) \wedge Y^{(1)}. \tag{5.13}$$

Algorithms 5.5 and 5.6 illustrate the SMT-based adaptation of Algorithms 5.1 and 5.2, respectively. The function $\texttt{symb\_var}(k, n)$ generates a set of n real-valued variables for bound k while F is a last-in/first-out (LIFO) *program stack* containing all conjuncts in (5.12). The command `push_back` adds a formula into F from the back while `pop_back` removes the last one. For $i \geq 0$, $F[i]$ is the $(i+1)^{\text{th}}$ element of F (from the back).

Algorithm 5.5 SMT-RA of MPL systems (forward)

Inputs: $A \in \mathbb{R}_{\max}^{n \times n}$,
initial and target sets $X_0, Y_0 \in \mathbb{R}^n$,
$N \in \mathbb{N}$,
Output: a Boolean value

1: $X \leftarrow \texttt{to\_LRA}(X_0)$
2: $Y \leftarrow \texttt{to\_LRA}(Y_0)$
3: $\mathit{reach} \leftarrow \texttt{false}$
4: $n \leftarrow \textsc{row}(A)$ ▷ the number of rows of A
5: $\mathcal{V}^{(0)} \leftarrow \texttt{symb\_var}(0, n)$
6: $F \leftarrow \emptyset$ ▷ empty stack
7: $F.\texttt{push\_back}(X)$
8: $k \leftarrow 1$
9: **while** $k \leq N$ **do**
10: $\quad \mathcal{V}^{(k)} \leftarrow \texttt{symb\_var}(k, n)$
11: $\quad F.\texttt{push\_back}(\texttt{SymbMPL}(A, \mathcal{V}^{(k-1)}, \mathcal{V}^{(k)}))$
12: $\quad Y.\texttt{subs}(\mathcal{V}^{(k-1)}, \mathcal{V}^{(k)})$
13: $\quad F.\texttt{push\_back}(Y)$
14: $\quad$ **if** $\texttt{check}(\texttt{mk\_and}(F)) = \texttt{SAT}$ **then**
15: $\quad\quad \mathit{reach} \leftarrow \texttt{true}$
16: $\quad\quad$ **break**
17: $\quad F.\texttt{pop\_back}()$
18: $\quad k \leftarrow k + 1$
19: **return** *reach*

Algorithm 5.6 SMT-RA of MPL systems (one-shot forward)

Inputs: $A \in \mathbb{R}_{\max}^{n \times n}$,
initial and target sets $X_0, Y_0 \in \mathbb{R}^n$,
$N \in \mathbb{N}$,
Output: a Boolean value

1: $X \leftarrow \texttt{to\_LRA}(X_0)$
2: $Y \leftarrow \texttt{to\_LRA}(Y_0)$
3: $\mathit{reach} \leftarrow \texttt{false}$
4: $n \leftarrow \textsc{row}(A)$
5: $\mathcal{V}^{(0)} \leftarrow \texttt{symb\_var}(0, n)$
6: $\mathcal{V}^{(1)} \leftarrow \texttt{symb\_var}(1, n)$
7: $F \leftarrow \emptyset$ ▷ empty stack
8: $F.\texttt{push\_back}(X)$
9: $F.\texttt{push\_back}(\texttt{true})$
10: $Y.\texttt{subs}(\mathcal{V}^{(0)}, \mathcal{V}^{(1)})$
11: $F.\texttt{push\_back}(Y)$
12: $k \leftarrow 1$
13: **while** $k \leq N$ **do**
14: $\quad F[1] \leftarrow \texttt{SymbMPL}(A^{\otimes k}, \mathcal{V}^{(0)}, \mathcal{V}^{(1)})$
15: $\quad$ **if** $\texttt{check}(\texttt{mk\_and}(F)) = \texttt{SAT}$ **then**
16: $\quad\quad \mathit{reach} \leftarrow \texttt{true}$
17: $\quad\quad$ **break**
18: $\quad k \leftarrow k + 1$
19: **return** *reach*

At the start of both algorithms, X_0 and Y_0 are expressed as QF-LRA formulae over $\mathcal{V}^{(0)}$. The function $Y.\texttt{subs}(\mathcal{V}^{(k-1)}, \mathcal{V}^{(k)})$ substitutes each appearance of $\mathtt{x}_i^{(k-1)}$ in Y with $\mathtt{x}_i^{(k)}$. The non-emptiness checking of a union of DBM in line 6 of Algorithms

5.1-5.2 is now formulated as the satisfiability checking of a QF-LRA formula (line 14 of Algorithm 5.5 and line 15 of Algorithm 5.6, respectively), where $\texttt{mk\_and}(F)$ stands for $\wedge_{0\leq i<|F|}F[i]$. The `check` function is implemented by an SMT solver, where $\texttt{check}(\texttt{mk\_and}(F)) = \texttt{SAT}$ means that $\texttt{mk\_and}(F)$ is satisfiable.

In lines 11-13 of Algorithm 5.5, a QF-RDL formula for (5.10) and the target set over $\mathcal{V}^{(k)}$ are added to F for each iteration k. If the condition in line 14 is not fulfilled, then the last element of F (i.e., Y) is removed. For Algorithm 5.6, the number of elements in F remains three for each iteration. In line 9, we set a temporary element for $F[1]$, which will be changed at each iteration (line 14).

We now will describe the approach for SMT-based backward RA. For $k \geq 1$, we use $\mathcal{V}^{(-k)} = \{\mathtt{x}_1^{(-k)}, \ldots, \mathtt{x}_n^{(-k)}\}$ to represent the set of variables encompassing k^{th} backward states obtained from $\mathcal{V}^{(0)}$. The backward version of (5.12) is

$$Y^{(0)} \wedge \left(\bigwedge_{k=1}^{N} \texttt{SymbMPL}(A, \mathcal{V}^{(-k)}, \mathcal{V}^{(1-k)}) \right) \wedge \bigvee_{k=1}^{N} X^{(-k)}. \tag{5.14}$$

Similarly, the one-step version of (5.14) can be encoded as

$$Y^{(0)} \wedge \left(\bigvee_{k=1}^{N} \texttt{SymbMPL}(A^{\otimes k}, \mathcal{V}^{(-1)}, \mathcal{V}^{(0)}) \right) \wedge X^{(-1)}. \tag{5.15}$$

Algorithms 5.7 and 5.8 summarise the backward approach to solve RA via SMT-solving. Line 10 of Algorithm 5.7 and line 13 of Algorithm 5.8 are equivalent to the emptiness checking in line 6 of Algorithms 5.3-5.4.

Algorithm 5.7 SMT-RA of MPL systems (backward)

Inputs: $A \in \mathbb{R}_{\max}^{n\times n}$, initial and target sets $X_0, Y_0 \in \mathbb{R}^n$, $N \in \mathbb{N}$,
Output: a Boolean value

1: $reach \leftarrow \texttt{false}$
2: $n \leftarrow \textsc{Row}(A)$
3: $\mathcal{V}^{(0)} \leftarrow \texttt{symb\_var}(0, n)$
4: $F \leftarrow \emptyset$ ▷ empty stack
5: F.push_back(Y)
6: $k \leftarrow 1$
7: **while** $k \leq N$ **do**
8: $\quad \mathcal{V}^{(-k)} \leftarrow \texttt{symb\_var}(-k, n)$
9: $\quad F$.push_back(SymbMPL($A, \mathcal{V}^{(-k)}, \mathcal{V}^{(1-k)}$))
10: $\quad$ **if** $\texttt{mk\_and}(F) = \texttt{UNSAT}$ **then**
11: $\quad\quad$ **break**
12: $\quad X$.subs($\mathcal{V}^{(1-k)}, \mathcal{V}^{(-k)}$)
13: $\quad F$.push_back(X)
14: $\quad$ **if** $\texttt{mk\_and}(F) = \texttt{SAT}$ **then**
15: $\quad\quad reach \leftarrow \texttt{true}$
16: $\quad\quad$ **break**
17: $\quad F$.pop_back()
18: $\quad k \leftarrow k + 1$
19: **return** $reach$

Algorithm 5.8 SMT-RA of MPL systems (one-shot backward)

Inputs: $A \in \mathbb{R}_{\max}^{n\times n}$, initial and target sets $X_0, Y_0 \in \mathbb{R}^n$, $N \in \mathbb{N}$,
Output: a Boolean value

1: $reach \leftarrow \texttt{false}$
2: $n \leftarrow \textsc{Row}(A)$
3: $\mathcal{V}^{(0)} \leftarrow \texttt{symb\_var}(0, n)$
4: $\mathcal{V}^{(-1)} \leftarrow \texttt{symb\_var}(-1, n)$
5: $F \leftarrow \emptyset$ ▷ empty stack
6: F.push_back(Y)
7: F.push_back(true)
8: X.subs($\mathcal{V}^{(0)}, \mathcal{V}^{(-1)}$)
9: F.push_back(X)
10: $k \leftarrow 1$
11: **while** $k \leq N$ **do**
12: $\quad F[1] \leftarrow \texttt{SymbMPL}(A^{\otimes k}, \mathcal{V}^{(-1)}, \mathcal{V}^{(0)})$
13: $\quad$ **if** $\texttt{mk\_and}(F) = \texttt{UNSAT}$ **then**
14: $\quad\quad$ **break**
15: $\quad$ **if** $\texttt{mk\_and}(F) = \texttt{SAT}$ **then**
16: $\quad\quad reach \leftarrow \texttt{true}$
17: $\quad\quad$ **break**
18: $\quad k \leftarrow k + 1$
19: **return** $reach$

5.2.1 Extensions to Other Models

This subsection presents the SMT-based procedures to solve reachability analysis of MiPL and IMPL systems. First, we show that the dynamics of MiPL and IMPL systems can be expressed in QF-RDL and QF-LRA, respectively. Then, we provide the modifications for Algorithms 5.5-5.8.

Min-Plus Linear Systems

Proposition 5.4. Any MiPL system in (2.10) can be expressed as a formula in QF-RDL as follows:

$$\bigwedge_{i=1}^{n}\left(\left(\bigwedge_{j\in\mathtt{fin}_i}\mathtt{x}_i^{(k)}-\mathtt{x}_j^{(k-1)}\leq B(i,j)\right)\wedge\left(\bigvee_{j\in\mathtt{fin}_i}\mathtt{x}_i^{(k)}-\mathtt{x}_j^{(k-1)}=B(i,j)\right)\right), \tag{5.16}$$

where $\mathtt{fin}_i$ contains the indices of the finite elements (not equal to ξ) of $B(i,\cdot)$.

Proof. By (2.11), any MiPL $\mathbf{x}(k)=B\otimes'\mathbf{x}(k-1)$ can be rewritten as $-\mathbf{x}(k)=(-B)\otimes(-\mathbf{x}(k-1))$. Then, by Proposition 5.3 and (5.10), it can expressed as

$$\bigwedge_{i=1}^{n}\left(\left(\bigwedge_{j\in\mathtt{fin}_i}-\mathtt{x}_i^{(k)}-(-\mathtt{x}_j^{(k-1)})\geq -B(i,j)\right)\wedge\left(\bigvee_{j\in\mathtt{fin}_i}-\mathtt{x}_i^{(k)}-(-\mathtt{x}_j^{(k-1)})=-B(i,j)\right)\right).$$

The above expression is indeed equivalent to (5.16). □

For simplicity, we denote (5.16) by $\mathtt{SymbMiPL}(B,\mathcal{V}^{(k-1)},\mathcal{V}^{(k)})$. Suppose we want solve bounded RA of MiPL systems with inputs $B\in\mathbb{R}_{\min}^{n\times n}$, an initial set X_0, and a target set Y_0. For forward approaches, one only needs to change line 11 of Algorithm 5.5 with $F.\mathtt{push\_back}(\mathtt{SymbMiPL}(B,\mathcal{V}^{(k-1)},\mathcal{V}^{(k)}))$ and line 14 of Algorithm 5.6 with $F[1]\leftarrow\mathtt{SymbMiPL}(B^{\otimes' k},\mathcal{V}^{(0)},\mathcal{V}^{(1)})$. Likewise, for backward approaches, we only need to change line 9 of Algorithm 5.7 and line 12 of Algorithm 5.8 with similar modifications.

Interval Max-Plus Linear Systems

We now will discuss the procedures to solve reachability analysis for IMPL systems. Unlike MPL and MiPL systems, we cannot express the dynamics of IMPL systems into QF-RDL due to the uncertainty behaviour on the state matrices. Instead, it can be translated into a QF-LRA as shown in Proposition 5.5.

Proposition 5.5 ([76])**.** Any IMPL system in (2.14) can be expressed as a formula in QF-LRA as follows:

$$\begin{aligned}\bigwedge_{i=1}^{n}\Bigg(\Bigg(\bigwedge_{j\in\mathtt{fin}_i}\mathtt{x}_i^{(k)}-\mathtt{x}_j^{(k-1)}\geq\mathtt{a}_{ij}^{(k-1)}\Bigg)\wedge\Bigg(\bigvee_{j\in\mathtt{fin}_i}\mathtt{x}_i^{(k)}-\mathtt{x}_j^{(k-1)}=\mathtt{a}_{ij}^{(k-1)}\Bigg)\wedge\\ \Bigg(\bigwedge_{j\in\mathtt{fin}_i}(\mathtt{a}_{ij}^{(k-1)}\geq\underline{A}(i,j))\wedge(\mathtt{a}_{ij}^{(k-1)}\leq\overline{A}(i,j))\Bigg)\Bigg),\end{aligned} \tag{5.17}$$

where $\mathtt{a}_{ij}^{(k)}$ represents a symbolic variable for the element of the state matrix at the k-th step from (2.14), at row i and column j, and $\mathtt{fin}_i$ is a set containing the indices of the finite elements of $\overline{A}(i,\cdot)$.

Proof. For IMPL systems, we know that the state matrix is not fixed at each step k and bounded by the lower matrix $\underline{A}$ and upper matrix $\overline{A}$. Thus, formula (5.10) is modified into a QF-LRA formula (5.17). □

We denote the sets of symbolic variables for the $(k-1)$-th state matrix as $\mathcal{A}^{(k-1)} = \{\mathtt{a}_{ij}^{(k-1)} \mid 1 \leq i \leq n, j \in \mathtt{fin}_i\}$. For the next reference, we denote the first-two conjucts of (5.17) as $\mathtt{SymbIMPL}(\mathcal{A}^{(k-1)}, \mathcal{V}^{(k-1)}, \mathcal{V}^{(k)})$ and the last conjuct as $\mathtt{Mat}(\mathbf{A}, \mathcal{A}^{(k-1)})$. The formula $\mathtt{Mat}(\mathbf{A}, \mathcal{A}^{(k-1)})$ encodes the symbolic representation for the $(k-1)$-th state matrix of (2.14). Therefore, the forward reachability analysis of IMPL systems up to bound N can be expressed as a QF-LRA formula

$$X^{(0)} \wedge \left(\bigwedge_{k=1}^{N} \mathtt{SymbIMPL}(\mathcal{A}^{(k-1)}, \mathcal{V}^{(k-1)}, \mathcal{V}^{(k)}) \wedge \mathtt{Mat}(\mathbf{A}, \mathcal{A}^{(k-1)}) \right) \wedge \bigvee_{k=1}^{N} Y^{(k)}, \quad (5.18)$$

Similarly, the backward reachability of IMPL systems is expressed as

$$Y^{(0)} \wedge \left(\bigwedge_{k=1}^{N} \mathtt{SymbIMPL}(\mathcal{A}^{(1-k)}, \mathcal{V}^{(-k)}, \mathcal{V}^{(1-k)}) \wedge \mathtt{Mat}(\mathbf{A}, \mathcal{A}^{(1-k)}) \right) \wedge \bigvee_{k=1}^{N} X^{(-k)}, \quad (5.19)$$

Algorithm 5.9 SMT-RA of IMPL systems (forward)

Inputs: $\mathbf{A} = [\underline{A}, \overline{A}]$ where $\underline{A}, \overline{A} \in \mathbb{R}_{\max}^{n \times n}$, initial and target sets $X_0, Y_0 \in \mathbb{R}^n$, $N \in \mathbb{N}$,
Output: a Boolean value

$X \leftarrow \mathtt{to\_LRA}(X_0)$
$Y \leftarrow \mathtt{to\_LRA}(Y_0)$
$reach \leftarrow \mathtt{false}$
$\mathcal{V}^{(0)} \leftarrow \mathtt{symb\_var}(0, n)$
$F \leftarrow \emptyset$ ▷ empty stack
$F.\mathtt{push\_back}(X)$
$k \leftarrow 1$
while $k \leq N$ **do**
  $\mathcal{A}^{(k-1)} \leftarrow \mathtt{symb\_mat}(\overline{A}, k-1)$
  $\mathcal{V}^{(k)} \leftarrow \mathtt{symb\_var}(k, n)$
  $f \leftarrow \mathtt{SymbIMPL}(\mathcal{A}^{(k-1)}, \mathcal{V}^{(k-1)}, \mathcal{V}^{(k)})$
  $F.\mathtt{push\_back}(f)$
  $F.\mathtt{push\_back}(\mathtt{Mat}(\mathbf{A}, \mathcal{A}^{(k-1)}))$
  $Y.\mathtt{subs}(\mathcal{V}^{(k-1)}, \mathcal{V}^{(k)})$
  $F.\mathtt{push\_back}(Y)$
  if $\mathtt{check}(\mathtt{mk\_and}(F)) = \mathtt{SAT}$ **then**
    $reach \leftarrow \mathtt{true}$
    break
  $F.\mathtt{pop\_back}()$
  $k \leftarrow k + 1$
return *reach*

Algorithm 5.10 SMT-RA of IMPL systems (backward)

Inputs: $\mathbf{A} = [\underline{A}, \overline{A}]$ where $\underline{A}, \overline{A} \in \mathbb{R}_{\max}^{n \times n}$, initial and target sets $X_0, Y_0 \in \mathbb{R}^n$, $N \in \mathbb{N}$,
Output: a Boolean value

$X \leftarrow \mathtt{to\_LRA}(X_0)$
$Y \leftarrow \mathtt{to\_LRA}(Y_0)$
$reach \leftarrow \mathtt{false}$
$\mathcal{V}^{(0)} \leftarrow \mathtt{symb\_var}(0, n)$
$F \leftarrow \emptyset$ ▷ empty stack
$F.\mathtt{push\_back}(Y)$
$k \leftarrow 1$
while $k \leq N$ **do**
  $\mathcal{A}^{(1-k)} \leftarrow \mathtt{symb\_mat}(\overline{A}, 1-k)$
  $\mathcal{V}^{(-k)} \leftarrow \mathtt{symb\_var}(-k, n)$
  $f \leftarrow \mathtt{SymbIMPL}(\mathcal{A}^{(1-k)}, \mathcal{V}^{(-k)}, \mathcal{V}^{(1-k)})$
  $F.\mathtt{push\_back}(f)$
  $F.\mathtt{push\_back}(\mathtt{Mat}(\mathbf{A}, \mathcal{A}^{(1-k)}))$
  if $\mathtt{mk\_and}(F) = \mathtt{UNSAT}$ **then**
    break
  $X.\mathtt{subs}(\mathcal{V}^{(1-k)}, \mathcal{V}^{(-k)})$
  $F.\mathtt{push\_back}(X)$
  if $\mathtt{mk\_and}(F) = \mathtt{SAT}$ **then**
    $reach \leftarrow \mathtt{true}$
    break
  $F.\mathtt{pop\_back}()$
  $k \leftarrow k + 1$
return *reach*

Algorithms 5.9-5.10 show the procedure to solve bounded RA of IMPL systems. For each iteration, in line 9, we generate the symbolic representation of the state matrices.

5.2.2 Quantified Reachability Analysis

This subsection discusses other extensions of reachability analysis by allowing quantifies, $\exists$ or $\forall$, over initial sets and the sequence of state matrices. Notice that, Problem 5.1 can be considered as *existential* reachability analysis. Its *universal* version is

"for all $\mathbf{x}(0) \in X_0$ there exists $1 \leq k \leq N$ such that $\mathbf{x}(k) = A \otimes \mathbf{x}(k-1) \in Y_0$."

In general, for two different vectors $\mathbf{x}(0), \mathbf{y}(0) \in X_0$ which eventually reach a target set Y_0, it is possible that they "enter" Y_0 with different time horizons: $\mathbf{x}(k_1) \in Y_0$ and $\mathbf{y}(k_2) \in Y_0$ with $k_1 \neq k_2$.

Due to the duality between universal and existential quantifiers, we can solve the universal reachability analysis of MPL systems (also MiPL systems) by solving its negation. That is, there exists $\mathbf{x}(0) \in X_0$ such that its corresponding orbit $\mathbf{x}(0) \dots \mathbf{x}(N)$ eventually reach the complement set of Y_0, i.e., $\mathbb{R}^n \setminus Y_0$. The negation of the output for the former problem becomes the outcome for the latter problem. We argue that the universal reachability analysis is too complicated for explicit procedures in Section 5.2 since the data structure DBM cannot handle the complement operation. On the other hand, for SMT-based procedures, we can use a negation symbol $\neg$ to express the complement of a set.

The quantified reachability analysis for IMPL systems is more compelling than that of MPL and IMPL systems since we can also define quantifiers over state matrices in addition to a quantifier over the initial set. Consequently, reachability analysis is studied by modifying the forward SMT-based RA of IMPL systems (Algorithm 5.9). These extensions result in four possible new problems (Problem 5.3). Furthermore, they allow to perform *safety analysis* of IMPL systems w.r.t. uncertainty on the state matrices: for instance, the dynamics of an IMPL system up to bound N never reach the undesired conditions regardless of the sequence of state matrices $A_0, \dots, A_{N-1}$.

Problem 5.3 (Quantified Reachability Analysis)**.** Suppose we have an IMPL system (2.14), $X_0, Y_0 \subseteq \mathbb{R}^n$ respectively as the initial and target sets and a positive integer N, the bounded quantified reachability analysis refers to one of the following:

1. $\exists \mathbf{x}(0) \in X_0.\ \exists A_0, \dots, A_{N-1} \in [\underline{A}, \overline{A}].\ \exists 1 \leq k \leq N.\ \mathbf{x}(k) = A_{k-1} \otimes \dots \otimes A_0 \otimes \mathbf{x}(0) \in Y_0$,
2. $\forall \mathbf{x}(0) \in X_0.\ \exists A_0, \dots, A_{N-1} \in [\underline{A}, \overline{A}].\ \exists 1 \leq k \leq N.\ \mathbf{x}(k) = A_{k-1} \otimes \dots \otimes A_0 \otimes \mathbf{x}(0) \in Y_0$,
3. $\exists \mathbf{x}(0) \in X_0.\ \forall A_0, \dots, A_{N-1} \in [\underline{A}, \overline{A}].\ \exists 1 \leq k \leq N.\ \mathbf{x}(k) = A_{k-1} \otimes \dots \otimes A_0 \otimes \mathbf{x}(0) \in Y_0$,
4. $\forall \mathbf{x}(0) \in X_0.\ \forall A_0, \dots, A_{N-1} \in [\underline{A}, \overline{A}].\ \exists 1 \leq k \leq N.\ \mathbf{x}(k) = A_{k-1} \otimes \dots \otimes A_0 \otimes \mathbf{x}(0) \in Y_0$. □

Essentially, the above problems differ on how we quantify the initial vector $\mathbf{x}(0) \in X_0$ and the N state matrices $A_0, \ldots, A_{N-1}$ of (2.14). Notice that the existensial-bounded RA is equivalent to Problem 5.3.1 and can be considered as the weakest formulation. On the other contrary, Problem 5.3.4 assesses whether for all $\mathbf{x}(0) \in X_0$ and for all $A_0, \ldots, A_{N-1} \in [\underline{A}, \overline{A}]$ there exists $1 \leq k \leq N$ such that if we compute $\mathbf{x}(1), \ldots, \mathbf{x}(N)$ according to (2.14) then at least one of these vectors belongs to Y_0. It is evident that Problem 5.3.4 is the strictest one: if the answer for Problem 5.3.4 is `true`, then so are the others. The implication relation within Problems 5.3.1-5.3.4 is illustrated in Figure 5.1.

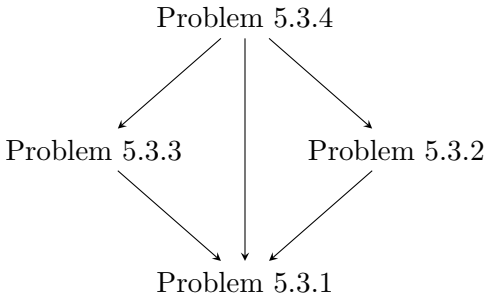


Figure 5.1: Logical relations amongst Problem 5.3. An arrow corresponds to a logical implication.

Remark 5.2. There are two additional possible orders of quantifiers for Problem 5.3, namely

$$\exists A_0, \ldots, A_{N-1} \in [\underline{A}, \overline{A}].\ \forall \mathbf{x}(0) \in X_0.\ \exists 1 \leq k \leq N.\\ \mathbf{x}(k) = A_{k-1} \otimes \ldots \otimes A_0 \otimes \mathbf{x}(0) \in Y_0,$$

and

$$\forall A_0, \ldots, A_{N-1} \in [\underline{A}, \overline{A}].\ \exists \mathbf{x}(0) \in X_0.\ \exists 1 \leq k \leq N.\\ \mathbf{x}(k) = A_{k-1} \otimes \ldots \otimes A_0 \otimes \mathbf{x}(0) \in Y_0.$$

This thesis does not discuss these problems, because it does not make practical sense to choose state matrices $A_0, \ldots, A_{N-1}$ prior to the initial vectors $\mathbf{x}(0)$. The backward formulation of Problems 5.3.1-5.3.4

$$Q_1(\mathbf{y} \in Y_0).\ Q_2(A_0, \ldots, A_{N-1} \in [\underline{A}, \overline{A}]).\ \exists \mathbf{x}(0) \in X_0.\\ \exists 1 \leq k \leq N.\ \mathbf{y} = A_{k-1} \otimes \ldots A_0 \otimes \mathbf{x}(0),$$

where the quantifiers $Q_1, Q_2 \in \{\exists, \forall\}$, is also not discussed in this work, because it is really unusual to quantify the target set for a reachability problem. □

Let us remark that Problems 5.3.2-5.3.4 (especially those with universal quantifiers) cannot be solved by standard approaches, namely by computing the reach sets. Indeed, if Problem 5.3.1 (which can be solved by modification of Algorithm 5.1) does not hold, then so is for other problems. If the output is `true` then there exists $1 \leq k \leq N$ such $X_k \cap Y_0 \neq \emptyset$. However, regardless the corresponding state

matrices $A_0, \ldots, A_{k-1}$, we cannot conclude whether the condition is satisfied by all vectors $\mathbf{x}(0) \in X_0$. It is also possible that two different vectors $\mathbf{x}(0), \mathbf{y}(0) \in X_0$, which eventually reach a target set Y_0, "enter" Y_0 with different sequence of state matrices. Therefore, the new problems can be uniquely addressed by the proposed SMT-based approaches.

Before describing the procedure to solve Problems 5.3.1-5.3.4, we will explain how to generate a quantified formula from quantifier-free ones. Given quantifier-free formulae F_1 and F_2 over variables $\mathtt{x}_1, \ldots, \mathtt{x}_n$, to express that all satisfying assigments of F_1 also satisfy F_2 we could write $\forall \mathtt{x}_1, \ldots, \mathtt{x}_n\ (F_1 \rightarrow F_2)$. On the other hand, to say that there exists one assignment for F_1 that also satisfies F_2, we can write $\exists \mathtt{x}_1, \ldots, \mathtt{x}_n\ (F_1 \wedge F_2)$.

Now we show how to express Problems 5.3.1-5.3.4 as LRA formulae with quantifiers. Recall that the quantifier-free expression for Problems 5.3.1-5.3.4 can be formulated as (5.18). The assertion "$\exists 1 \leq k \leq N$, such that $\mathbf{x}(k) \in Y$", presents in Problems 5.3.1-5.3.4, can be expressed as a sub-formula of (5.18), as

$$F_1 = \left(\bigwedge_{k=1}^{N} \mathtt{SymbIMPL}(\mathcal{A}^{(k-1)}, \mathcal{V}^{(k-1)}, \mathcal{V}^{(k)}) \right) \wedge \left(\bigvee_{k=1}^{N} Y^{(k)} \right). \tag{5.20}$$

Building on this, we can add quantifiers for the symbolic variables $\mathcal{V}^{(0)}, \ldots, \mathcal{V}^{(N)}$ and symbolic matrices $\mathcal{A}^{(0)}, \ldots, \mathcal{A}^{(N)}$.

The quantifier for $\mathcal{V}^{(0)}$ is restricted by the formula $X^{(0)}$, while the quantifiers for state matrices $\mathcal{A}^{(0)}, \ldots, \mathcal{A}^{(N)}$ are restricted by the formula $\mathtt{Mat}(\mathcal{A}^{(0)}) \wedge \ldots \wedge \mathtt{Mat}(\mathcal{A}^{(N-1)})$. For $k \geq 1$, the set of variables $\mathcal{V}^{(k)}$ is always preceded by quantifier $\exists$ because the vector $\mathbf{x}(k)$ is uniquely determined by the chosen initial vector $\mathbf{x}(0)$ and state matrices $A_0, \ldots, A_{k-1}$. Furthermore, the location for their quantifiers will be on the last ones. Thus, Problems 5.3.1 and 5.3.2 can be respectively formulated as

$$\exists \mathcal{V}^{(0)} \exists \mathcal{A}^{[N]} \exists \mathcal{V}^{[N]} \left(X^{(0)} \wedge \bigwedge_{k=0}^{N-1} \mathtt{Mat}(\mathbf{A}, \mathcal{A}^{(k)}) \wedge F_1 \right), \tag{5.21a}$$

$$\forall \mathcal{V}^{(0)} \exists \mathcal{A}^{[N]} \exists \mathcal{V}^{[N]} \left(X^{(0)} \rightarrow \left(\bigwedge_{k=0}^{N-1} \mathtt{Mat}(\mathbf{A}, \mathcal{A}^{(k)}) \wedge F_1 \right) \right), \tag{5.21b}$$

where $\exists \mathcal{A}^{[N]}$ and $\exists \mathcal{V}^{[N]}$ are respectively the abbreviation of $\exists \mathcal{A}^{(0)} \ldots \exists \mathcal{A}^{(N-1)}$ and $\exists \mathcal{V}^{(1)} \ldots \exists \mathcal{V}^{(N)}$.

For the remaining two Problems 5.3.3-5.3.4 where the state matrices are universally quantified, we express their negation instead. This will positively affect the performance of the procedure, as it is known that the fewer are the universal quantifiers in a quantified formula, the faster it is for an SMT-solver to check its satisfiability. For instance, the negation for Problem 5.3.3 is $\forall \mathbf{x}(0) \in X_0.\ \exists A_0, \ldots, A_{N-1} \in [\underline{A}, \overline{A}].\ \forall 1 \leq k \leq N.\ \mathbf{x}(k) = A_{k-1} \otimes \ldots \otimes A_0 \otimes \mathbf{x}(0) \notin Y$. The quantifier-free expression of this statement can be expressed as

$$F_2 = \left(\bigwedge_{k=1}^{N} \mathtt{SymbIMPL}(\mathcal{A}^{(k-1)}, \mathcal{V}^{(k-1)}, \mathcal{V}^{(k)}) \right) \wedge \left(\bigwedge_{k=1}^{N} \neg Y^{(k)} \right). \tag{5.22}$$

As a result, the negation of Problems 5.3.3 and 5.3.4 are respectively formulated as

$$\forall \mathcal{V}^{(0)} \exists \mathcal{A}^{[N]} \exists \mathcal{V}^{[N]} \left(X^{(0)} \rightarrow \left(\bigwedge_{k=0}^{N-1} \texttt{Mat}(\mathbf{A}, \mathcal{A}^{(k)}) \wedge F_2 \right) \right), \tag{5.23a}$$

$$\exists \mathcal{V}^{(0)} \exists \mathcal{A}^{[N]} \exists \mathcal{V}^{[N]} \left(X^{(0)} \wedge \bigwedge_{k=0}^{N-1} \texttt{Mat}(A^{(k)}) \wedge F_2 \right). \tag{5.23b}$$

Notice that, similar to (5.21b), there is only one universal quantifier in formula (5.23a). Similarly, the formula (5.23b) does not contain any universal quantifier: as in (5.21a).

Algorithm 5.11 Quantified RA of IMPL systems

Inputs: $\mathbf{A} = [\underline{A}, \overline{A}]$ where $\underline{A}, \overline{A} \in \mathbb{R}_{\max}^{n \times n}$,
initial and target sets $X_0, Y_0 \in \mathbb{R}^n$,
$N \in \mathbb{N}$,
$type \in \{1, 2, 3, 4\}$
Output: a Boolean value

1: $X \leftarrow \texttt{to\_LRA}(X_0)$
2: $Y \leftarrow \texttt{to\_LRA}(Y_0)$
3: $n \leftarrow \textsc{Row}(\underline{A})$ ▷ the number of rows of $\underline{A}$
4: $T, D, I, V, M \leftarrow \emptyset$ ▷ empty stacks
5: $\mathcal{V}^{(0)} \leftarrow \texttt{symb\_var}(0, n)$
6: **for** $1 \leq k \leq N$ **do**
7: $M.\texttt{push}(\mathcal{A}^{(k-1)})$ ▷ symbolic matrices
8: $\mathcal{V}^{(k)} \leftarrow \texttt{symb\_var}(k, n)$
9: $V.\texttt{push}(\mathcal{V}^{(k)})$ ▷ symbolic variables
10: $I.\texttt{push}(\texttt{Mat}(\mathcal{A}^{(k-1)}))$
11: $D.\texttt{push}(\texttt{SymbIMPL}(\mathcal{A}^{(k-1)}, \mathcal{V}^{(k-1)}, \mathcal{V}^{(k)}))$
12: $Y.\texttt{subs}(\mathcal{V}^{(k-1)}, \mathcal{V}^{(k)})$
13: $T.\texttt{push}(Y)$
14: $F \leftarrow \texttt{mk\_and}(D)$
15: **if** $(type = 1 \vee type = 2)$ **then**
16: $F \leftarrow F \wedge \texttt{mk\_or}(T)$ ▷ formula in (5.20)
17: **else if** $(type = 3 \vee type = 4)$ **then**
18: $F \leftarrow F \wedge \neg\texttt{mk\_or}(T)$ ▷ formula in (5.22)
19: **if** $(type = 1 \vee type = 4)$ **then**
20: $F \leftarrow \exists \mathcal{V}^{(0)}.\ \exists Mat.\ \exists Var.\ (X \wedge \texttt{mk\_and}(I) \wedge F)$
21: **else if** $(type = 2 \vee type = 3)$ **then**
22: $F \leftarrow \forall \mathcal{V}^{(0)}.\ \exists Mat.\ \exists Var.\ (X \rightarrow (\texttt{mk\_and}(I) \wedge F))$
23: $res \leftarrow \texttt{false}$
24: **if** $\texttt{check}(F) = \texttt{SAT}$ **then**
25: $res \leftarrow \texttt{true}$
26: **if** $(type = 1 \vee type = 2)$ **then**
27: **return** res
28: **else if** $(type = 3 \vee type = 4)$ **then**
29: **return** $\neg res$

Algorithm 5.11 summarises the steps to solve Problems 5.3.1-5.3.4. At the start, we generate empty stacks that will contain the formula encoding of the IMPL dynamics (D and I) and of the target set (T). Other stacks, V and M, correspond to the symbolic variables and matrices. In lines 7-13, we collect the symbolic variables and matrices in (5.17) and introduce the QF-LRA formulae for (5.17). Then, in

lines 15-18, a quantifier-free formula w.r.t. (5.20) or (5.22) is generated: the command `mk_or`(T) stands for $\vee_{f\in T} f$. The quantifier formula corresponding to one of four formulae in (5.21a)-(5.21b) or (5.23a)-(5.23b) is then generated in lines 19-22. Finally, we check the satisfaction of the resulting formula using an SMT solver. The result is expressed as a boolean variable res, which initialised as `false`. If the SMT solver reports "satisfiable" then res is changed to `true`.

The output of quantified reachability analysis depends on the type of the problem (represented by an integer $type \in \{1, 2, 3, 4\}$) and the value of res. If $1 \leq type \leq 2$, this corresponds to one of Problems 5.3.1 and 5.3.2, the value of res is the output for the quantified RA problem (line 27). Conversely, recalling that for Problems 5.3.3-5.3.4, the procedure in Algorithm 5.11 checks the satisfaction of their negation, if $3 \leq type \leq 4$, then the negation of res becomes the output instead (line 29).

The performance of Algorithm 5.11 crucially depends on the SMT-checking in line 24. As mentioned before, the more universal quantifiers in a (quantified) formula, the slower SMT solver to check its satisfiability. Thus, the running time of solving Problem 5.3.1 and 5.3.4 is potentially faster than those of Problems 5.3.2-5.3.3: this argument is confirmed in the experiments of the next section (see Table 5.6).

5.3 Computational Benchmarks

This section presents the performance comparison of the explicit and SMT-based procedures to solve the reachability analysis of MPL and IMPL systems. For each model, we apply specific experimental setups. We recall that the explicit algorithms that compute the reach sets are firstly introduced in [5,6] (for MPL systems) and [27] (for IMPL systems). The experiments are implemented on an Intel(R) Xeon(R) Gold 6248 CPU @ 2.50GHz with 120GB of RAM. For the SMT solver, we use Z3 [50].

5.3.1 Max-Plus Linear Systems

Experimental Setups In the benchmarks, we work with pairs (n, m) where $m \leq n$. For each dimension n i.e., number of continuous variables, we generate 50 irreducible matrices $A \in \mathbb{R}_{\max}^{n\times n}$ with m finite elements in each row, where their values are taken to be between 1 and 20. The locations of the finite elements are chosen randomly. We focus on irreducible matrices to ensure the output of the RA (`true` or `false`). The setting for initial set X_0 and target set Y_0 is determined as follows: $X_0 = \{\mathbf{x} \in \mathbb{R}^n \mid x_1 \geq \ldots \geq x_p, 0 < x_1 - x_p < 20\}$ and $Y_0 = \{\mathbf{x} \in \mathbb{R}^n \mid x_1 \leq \ldots \leq x_p\}$ where $p = 5$ if $n \leq 10$ and $p = \lceil \frac{n}{2} \rceil$ if $n > 10$. The latter setting is used to balance the output of RA of high-dimensional MPL systems, and is exclusively for SMT-based algorithms. We set 30 minutes as the `timeout` limit.

Results Table 5.1 (columns 2-5) shows the average and maximum running time of the reachability analysis via sequential approaches. The 6^{th} column reports the number of experiments (out of 50) with a `true` outcome, whilst the last one represents the average and maximum completeness threshold as obtained from the 50 experiments. As one can see in Table 5.1, the SMT-based algorithms are significantly

faster than those that explicitly compute the reach sets. Regarding the comparison between the forward and backward approaches (for both explicit and SMT-based algorithms), the latter seems to be faster. This output is likely due to the break condition in line 6 of Algorithm 5.3 and line 10 of Algorithm 5.7, which cause the backward algorithms to terminate earlier than the specified step bound whenever the RA problem yields `false`. It should also be noted that the completeness threshold also affects the overall running time. Moreover, the number of finite elements of the matrix heavily affects the runtime for the algorithms that explicitly compute the reach sets.

Table 5.1: Computational benchmark for sequential reachability analysis of MPL systems

(n, m)	Explicit		SMT		#`true`	N^*
	Alg. 5.1	Alg. 5.3	Alg. 5.5	Alg. 5.7		
$(5, 2)$	$\{0.01, 0.06\}$	$\{4.13, 205.66\}$	$\{0.05, 1.2\}$	$\{0.02, 0.42\}$	10	$\{12.08, 44\}$
$(6, 2)$	$\{0.07, 1.51\}$	$\{0.48, 22.95\}$	$\{0.05, 0.88\}$	$\{0.02, 0.02\}$	18	$\{11.42, 54\}$
$(7, 2)$	$\{13.61, 414.94\}$	$\{1.03, 48.35\}$	$\{0.91, 39.05\}$	$\{0.02, 0.02\}$	29	$\{18.48, 86\}$
$(8, 2)$	$\{12.79, 584.03\}$	$\{6.82, 162.05\}$	$\{0.02, 0.14\}$	$\{0.02, 0.02\}$	32	$\{14.88, 49\}$
$(9, 2)$	$\{24.2, 497.89\}$	$\{24.95, 420.08\}$	$\{0.02, 0.11\}$	$\{0.02, 0.02\}$	40	$\{19.98, 110\}$
$(10, 2)$	`timeout`(1)	`timeout`(3)	$\{1.17, 50.1\}$	$\{0.02, 0.03\}$	39	$\{18.22, 51\}$
$(5, 4)$	$\{0.03, 0.09\}$	$\{0.02, 0.36\}$	$\{0.05, 0.44\}$	$\{0.02, 0.03\}$	10	$\{7.74, 21\}$
$(6, 4)$	$\{2.16, 16.18\}$	$\{3.29, 66.22\}$	$\{0.2, 2.12\}$	$\{0.02, 0.03\}$	20	$\{10.34, 30\}$
$(7, 4)$	$\{6.37, 63.95\}$	`timeout`(1)	$\{0.13, 2.37\}$	$\{0.02, 0.05\}$	25	$\{9.26, 19\}$
$(8, 4)$	$\{40.2, 359.06\}$	$\{31.81, 294.22\}$	$\{2.06, 52.69\}$	$\{0.02, 0.02\}$	33	$\{14.44, 58\}$
$(9, 4)$	`timeout`(2)	`timeout`(1)	$\{4.36, 194.26\}$	$\{0.02, 0.03\}$	39	$\{16.02, 59\}$
$(10, 4)$	`timeout`(42)	`timeout`(33)	$\{0.6, 10.62\}$	$\{0.02, 0.03\}$	40	$\{13.24, 41\}$

Table 5.2: Computational benchmark for one-shot reachability analysis of MPL systems

(n, m)	Abstraction		SMT		#`true`	N^*
	Alg. 5.2	Alg. 5.4	Alg. 5.6	Alg. 5.8		
$(5, 2)$	$\{0.04, 0.2\}$	$\{0.04, 0.19\}$	$\{0.02, 0.05\}$	$\{0.01, 0.06\}$	10	$\{12.08, 44\}$
$(6, 2)$	$\{0.19, 1.82\}$	$\{0.18, 1.75\}$	$\{0.02, 0.1\}$	$\{0.01, 0.02\}$	18	$\{11.42, 54\}$
$(7, 2)$	$\{3.37, 84.74\}$	$\{2.25, 31.77\}$	$\{0.03, 0.2\}$	$\{0.01, 0.02\}$	29	$\{18.48, 86\}$
$(8, 2)$	$\{9.84, 104.67\}$	$\{7.5, 78.22\}$	$\{0.02, 0.11\}$	$\{0.01, 0.02\}$	32	$\{14.88, 49\}$
$(9, 2)$	$\{46.86, 627.5\}$	$\{36.3, 333.04\}$	$\{0.02, 0.09\}$	$\{0.01, 0.02\}$	40	$\{19.98, 110\}$
$(10, 2)$	`timeout`(9)	`timeout`(8)	$\{0.03, 0.22\}$	$\{0.01, 0.02\}$	39	$\{18.22, 51\}$
$(5, 4)$	$\{0.04, 0.12\}$	$\{0.04, 0.11\}$	$\{0.02, 0.03\}$	$\{0.01, 0.02\}$	10	$\{7.74, 21\}$
$(6, 4)$	$\{0.34, 1.24\}$	$\{0.33, 1.07\}$	$\{0.02, 0.05\}$	$\{0.01, 0.02\}$	20	$\{10.34, 30\}$
$(7, 4)$	$\{1.82, 5.21\}$	$\{1.92, 5.63\}$	$\{0.02, 0.05\}$	$\{0.01, 0.02\}$	25	$\{9.26, 19\}$
$(8, 4)$	$\{27.42, 106.19\}$	$\{25.21, 97.72\}$	$\{0.03, 0.19\}$	$\{0.01, 0.02\}$	33	$\{14.44, 58\}$
$(9, 4)$	$\{317.67, 1105.06\}$	$\{288.91, 1115.55\}$	$\{0.03, 0.24\}$	$\{0.01, 0.02\}$	39	$\{16.02, 59\}$
$(10, 4)$	`timeout`(36)	`timeout`(40)	$\{0.03, 0.16\}$	$\{0.01, 0.03\}$	40	$\{13.24, 41\}$

Table 5.2 reports the average and maximum running times obtained using one-shot approaches over the same tests of Table 5.1 (the last two columns are indeed equal). The one-shot strategy improves the running time over its sequential counterpart for the SMT-based algorithms. However, for the explicit algorithms, the improvement only presents in the experiments where the matrices have four finite elements in each row. Again, within the one-shot procedures, the SMT-based algorithms outperform those sequentially computing the reach sets.

The impressive (and almost constant) outcomes of the SMT-based algorithms (Algorithms 5.6 and 5.8) in Table 5.2 suggest to push their scalability to the limit. Hence, we provide a computational benchmark for high-dimensional MPL systems in Table 5.3. We focus the benchmark exclusively on one-shot algorithms, as we have seen that sequential algorithms are arguably slower. Similar to the results in Table 5.2, Table 5.3 shows that the performance of Algorithm 5.8 is better than that of Algorithm 5.6 up to the dimension of 140. Instead, for larger dimensions, the latter algorithm outperforms the former one. There are two possible reasons for this outcome. First, the larger proportion of `true` experiments: we argue that if the RA problem yields `true`, then Algorithm 5.6 (which performs SMT-checking once) is likely faster than Algorithm 5.8 (which uses SMT-checking twice for each iteration). Second, the smaller values of completeness thresholds also contribute to the relative speedup of Algorithm 5.6.

Table 5.3: Computational benchmark for SMT-based reachability analysis of high-dimensional MPL systems.

(n, m)	SMT		#`true`	N^*
	Alg. 5.6	Alg. 5.8		
$(20, 10)$	$\{0.28, 1.49\}$	$\{0.05, 0.15\}$	20	$\{18.9, 55\}$
$(30, 15)$	$\{1.39, 5.2\}$	$\{0.16, 0.44\}$	8	$\{19.64, 62\}$
$(40, 20)$	$\{3.61, 7.95\}$	$\{0.35, 0.98\}$	4	$\{19.88, 36\}$
$(50, 25)$	$\{7.75, 16.27\}$	$\{0.75, 2.98\}$	3	$\{21.9, 65\}$
$(60, 30)$	$\{13.91, 34.1\}$	$\{1.64, 6.34\}$	4	$\{21.66, 46\}$
$(70, 35)$	$\{19.84, 70.85\}$	$\{2.64, 11.42\}$	3	$\{17.6, 44\}$
$(80, 40)$	$\{28.02, 57.85\}$	$\{4.89, 19.25\}$	4	$\{15.5, 30\}$
$(90, 45)$	$\{36.2, 67.03\}$	$\{9.25, 31.04\}$	5	$\{13.94, 18\}$
$(100, 50)$	$\{47.48, 101.07\}$	$\{14.13, 55.36\}$	10	$\{12.7, 18\}$
$(110, 55)$	$\{70.3, 134.96\}$	$\{26.84, 83.76\}$	7	$\{11.74, 17\}$
$(120, 60)$	$\{84.5, 165.16\}$	$\{55.03, 198.72\}$	12	$\{10.86, 14\}$
$(130, 65)$	$\{104.88, 198.55\}$	$\{88.16, 373.6\}$	11	$\{10.14, 13\}$
$(140, 70)$	$\{155.11, 314.63\}$	$\{119.33, 444.74\}$	11	$\{9.84, 15\}$
$(150, 75)$	$\{164.29, 439.51\}$	$\{265.39, 1340.16\}$	20	$\{9.06, 11\}$
$(160, 80)$	$\{219.84, 479.46\}$	$\{257.51, 912.06\}$	15	$\{8.72, 12\}$
$(170, 85)$	$\{185.98, 563.7\}$	$\{471.88, 1344.89\}$	30	$\{8.22, 11\}$
$(180, 90)$	$\{218.67, 810.35\}$	`timeout`(1)	35	$\{7.94, 9\}$
$(190, 95)$	$\{368.03, 1037.91\}$	`timeout`(3)	30	$\{7.44, 9\}$
$(200, 100)$	$\{402.66, 1211.83\}$	`timeout`(19)	33	$\{7.32, 9\}$

5.3.2 Interval Max-Plus Linear Systems

Experimental Setups Again, we work with pairs (n, m) where $m \leq n$. For each dimension n, we generate 50 matrices $\underline{A}, \overline{A} \in \mathbb{R}_{\max}^{n \times n}$, with m finite elements in each row, where their values are taken to be between 1 and 20. We recall that the upper and lower matrices $\underline{A}, \overline{A}$ are generated under the conditions in Definition 2.10. The locations of the finite elements in each row of are chosen randomly. We select $X_0 = \{\mathbf{x} \in \mathbb{R}^n \mid 0 \leq x_1 \leq \ldots \leq x_p \leq 1\}$ and $Y = \{\mathbf{x} \in \mathbb{R}^n \mid x_1 > \ldots > x_p, x_1 - x_p > 20\}$ with $p = \lceil \frac{n}{2} \rceil$ as the set of initial and target sets, respectively. The experiments have been implemented to perform the reachability analysis over $N = 20$ steps. Again, we set 30 minutes as the `timeout` limit.

Results As depicted in Table 5.4, the SMT-based algorithms are much faster than the explicit ones to solve RA of IMPL systems. Interestingly, although most experiments result in `true`, the backward approach is much faster. This output is clearly shown for the explicit algorithms. On the other hand, due to the "almost constant" running times in the fourth and fifth columns of Table 5.4, we cannot conclude that Algorithm 5.10 is faster than Algorithm Algorithm 5.9: this suggests to challenge both approaches over IMPL systems with drastically larger dimensions.

Table 5.4: Computational benchmark for reachability analysis of IMPL systems.

(n, m)	Explicit		SMT		#`true`
	Alg. 5.1	Alg. 5.3	Alg. 5.9	Alg. 5.10	
$(5, 2)$	$\{0.68, 23.78\}$	$\{0.02, 0.7\}$	$\{0.0, 0.04\}$	$\{0.0, 0.07\}$	44
$(6, 2)$	$\{23.21, 1154.22\}$	$\{0.05, 0.51\}$	$\{0.01, 0.03\}$	$\{0.0, 0.01\}$	45
$(7, 2)$	`timeout`(1)	$\{0.35, 10.55\}$	$\{0.01, 0.12\}$	$\{0.01, 0.02\}$	47
$(8, 2)$	`timeout`(4)	$\{36.2, 1238.16\}$	$\{0.02, 0.15\}$	$\{0.01, 0.02\}$	42
$(5, 4)$	$\{38.23, 1707.6\}$	$\{0.11, 0.3\}$	$\{0.04, 0.18\}$	$\{0.01, 0.02\}$	32
$(6, 4)$	`timeout`(5)	$\{8.01, 10.92\}$	$\{0.04, 0.26\}$	$\{0.01, 0.02\}$	40
$(7, 4)$	`timeout`(4)	$\{29.58, 41.78\}$	$\{0.03, 0.93\}$	$\{0.01, 0.01\}$	45
$(8, 4)$	`timeout`(7)	`timeout`(1)	$\{0.02, 0.22\}$	$\{0.01, 0.04\}$	44

Table 5.5 presents a benchmark for SMT-based reachability analysis for high-dimensional IMPL systems. It is now clear that the backward approach is faster than the forward one. As the dimension increases, we see an increasing gap in the average and maximum running time: we then stop the experiments for the forward algorithm. Additionally, based on our experiments, the backward approach has an advantage when the experiment yields `false`, which is owed to the break condition in line 14 of Algorithm 5.10.

For the last benchmark, we present the average and maximum running time of Algorithm 5.11. For the sake of simplicity, we write P_i to refer Problem 5.3.i for $i \in \{1, 2, 3, 4\}$. Based on the table, it is clear that the universal quantifiers heavily burden the performance of Algorithm 5.11. Thanks to formula (5.23b), checking the satisfaction of Problem 5.3.1 is as fast as that of Problem 5.3.4. Notice that the number of experiments with `true` output for Problem 5.3.1 (the sixth column of

Table 5.6) is the same as that of quantifier-free RA (last column of Table Table 5.3). On the other hand, for Problem 5.3.4, there is no single experiment with `true` output because of its more restrictive requirements.

Table 5.5: Computational benchmark for SMT-based reachability analysis of high-dimensional IMPL systems.

(n, m)	SMT		#`true`
	Alg. 5.9	Alg. 5.10	
$(10, 5)$	$\{0.2, 2.47\}$	$\{0.01, 0.03\}$	40
$(20, 10)$	$\{50.37, 1455.93\}$	$\{0.08, 0.24\}$	37
$(30, 15)$	`timeout`(3)	$\{0.29, 2.75\}$	43
$(40, 20)$	`timeout`(10)	$\{0.53, 1.64\}$	40
$(50, 25)$	`timeout`(16)	$\{1.32, 4.67\}$	34
$(60, 30)$	-	$\{2.31, 9.68\}$	37
$(70, 35)$	-	$\{4.39, 23.69\}$	38
$(80, 40)$	-	$\{6.56, 28.04\}$	37
$(90, 45)$	-	$\{9.9, 19.99\}$	39
$(100, 50)$	-	$\{21.37, 160.26\}$	41
$(110, 55)$	-	$\{22.89, 112.95\}$	44
$(120, 60)$	-	$\{53.6, 486.7\}$	42
$(130, 65)$	-	$\{61.12, 375.64\}$	41
$(140, 70)$	-	`timeout`(1)	42
$(150, 75)$	-	`timeout`(2)	45

Table 5.6: Computational benchmark for quantified reachability analysis of IMPL systems.

(n, m)	running times				#`true`			
	P_1	P_2	P_3	P_4	P_1	P_2	P_3	P_4
$(5, 2)$	$\{0.11, 0.17\}$	$\{6.69, 252.54\}$	$\{2.37, 86.66\}$	$\{0.1, 0.27\}$	44	22	43	0
$(6, 2)$	$\{0.13, 0.26\}$	`timeout`(5)	$\{31.39, 523.08\}$	$\{0.12, 0.18\}$	45	28	27	0
$(7, 2)$	$\{0.19, 0.45\}$	`timeout`(6)	`timeout`(1)	$\{0.16, 0.23\}$	47	27	40	0
$(8, 2)$	$\{0.23, 0.92\}$	`timeout`(8)	`timeout`(10)	$\{0.21, 0.5\}$	42	20	14	0
$(5, 4)$	$\{0.44, 5.07\}$	$\{1.13, 8.2\}$	$\{2.74, 12.4\}$	$\{0.31, 1.86\}$	32	6	30	0
$(6, 4)$	$\{0.49, 7.91\}$	$\{5.04, 116.06\}$	$\{18.05, 122.66\}$	$\{0.28, 1.15\}$	40	7	16	0
$(7, 4)$	$\{1.42, 21.86\}$	$\{60.53, 811.39\}$	`timeout`(1)	$\{0.48, 2.35\}$	45	13	31	0
$(8, 4)$	$\{0.85, 5.45\}$	`timeout`(1)	`timeout`(5)	$\{0.5, 2.27\}$	44	15	10	0

5.4 Summary

In this chapter, we have discussed the reachability analysis of MPL systems. We have proposed the novel SMT-based procedures and compared their performance against the existing techniques that explicitly compute the reach sets. Based on the computational benchmarks, the performance of SMT-based algorithms is much faster and scalable, which allows performing reachability analysis of high-dimensional MPL

systems. Furthermore, it has been proven that the unbounded reachability analysis is a decidable problem for any MPL system as long as the corresponding transient exists. Furthermore, the resulting algorithms can be applied to MiPL and IMPL systems.

Chapter 6

Bounded Model Checking of Max-Plus Linear Systems

In this chapter, we develop a framework for bounded model checking of MPL systems. Given an MPL system with a set of initial states X, we check whether all underlying orbits satisfy a specification of interest. In this thesis, we express the specifications using the time-difference LTL (TDLTL) formula, which is defined over time-difference propositions (cf. Definition 6.1). Such propositions represent the delays between discrete-time events of MPL systems. Furthermore, we assume that the MPL systems are *periodic*. We propose two novel procedures to verify a TDLTL formula φ: 1) abstraction-based by generating the abstraction of MPL systems and 2) SMT-based by encoding the dynamics of MPL systems as well as the bounded counterexample of φ into QF-RDL formulae. We prove that for irreducible (*boundedly periodic*, in general) MPL systems, both procedures are sound and complete where the completeness threshold corresponds to the pair of transient and cyclicity. We also present the extensions of the procedures to MiPL and IMPL systems.

6.1 Related Work

The approach for formal verification for MPL systems has been firstly discussed in [2, 4]. Instead of verifying the specification on the original MPL system, it works on the abstract transition system generated from an MPL system. There are some limitations of this approach. First, the approach is not complete, especially when the specification is not satisfied on the abstract transition system. In general, one cannot conclude that the specification is also not valid on the concrete MPL system. The refinement procedure in [2, 7] may result in a refined abstraction that bisimulates the MPL system (see Theorem 3.1). However, the refinement procedure does not always terminate, especially for the abstract transition system generated from the high-dimensional MPL system. Second, the scope for the atomic propositions and specifications is limited. In most cases, the propositions and specifications are defined after generating the abstraction of an MPL system. Furthermore, it is assumed that the set of states (of MPL systems) satisfying each proposition is a DBM. Therefore, verifying specifications that involve disjunction is deemed hard because a union of DBM cannot be expressed as DBM.

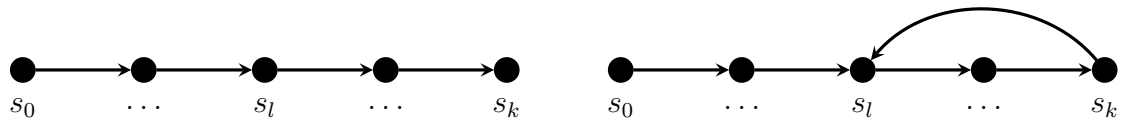
6.2 Preliminaries

This section elaborates on some preliminary notions such as Bounded Model Checking and time-difference LTL in MPL systems.

6.2.1 Bounded Model Checking

The term *model checking* refers to an algorithm for exploring the paths and traces of a transition system to determine whether it obeys a specification of its intended behaviour [36]. It works on paths (the sequence of states in a transition system) with unbounded length. Hence, it is often called as "unbounded model checking" technique. The common problem of unbounded model checking is that it suffers from the state-explosion problem since most of the techniques use Binary Decision Diagrams (BDD) [23] to represent the transition system symbolically. Therefore, a new type of model checking, bounded model checking (BMC) [17–19], is introduced to tackle this problem.

The basic idea of BMC is to search for a *bounded witness* of a temporal property: an infinite path of transition system in which property holds and which can be sufficiently represented by its finite prefix. Such a bounded witness can be represented as either no-loop or lasso-shape paths. An infinite path $\pi = s_0 s_1 \dots$ is in lasso-shape if there exist $k > l \geq 0$ such that $\pi[k+1+j] = \pi[l+j]$ for $j \geq 0$ [18,19]. This path can be expressed[9] as $\pi = (\pi_{\text{stem}})(\pi_{\text{loop}})^{\omega}$ where $\pi_{\text{stem}} = (s_0 \dots s_{l-1})$ and $\pi_{\text{loop}} = (s_l \dots s_k)$. This path is called as (k,l)-lasso in which the transition from s_k to s_l is called *back loop* (see Figure 6.1(b)). Although it is an infinite path, we can interpret it as a finite one (up to bound k). This type of paths can be used to represent the witness of invariant properties such as $\Box p$ and $\Diamond\Box p$. On the other hand, a finite path $\pi = s_0 \dots s_k$ is called no-loop if there is no back loop as in Figure 6.1(a). Indeed, this type of paths does not say anything about the infinite behaviour beyond state s_k. However, it can be used to represent the witness of liveness properties $\Diamond p$: if there exists $0 \leq i \leq k$ such that p holds on s_i then $\Diamond p$ hold for any infinite path with prefix π.



(a) no-loop path (b) (k,l)-lasso path

Figure 6.1: The two cases of bounded witness.

In the BMC algorithm, all possible k-length bounded witnesses of the negated specification, *bounded counterexample*, are encoded as Boolean satisfiability (SAT) problem. If the formula is satisfiable, then such a witness is present. On the other hand, if it is unsatisfiable, then the bound is increased by one. This process is repeated until a pre-known *completeness threshold* [38] is reached. Generally speaking,

[9] In the case that l equals 0, we write $\pi = (\pi_{\text{loop}})^{\omega}$ instead.

the completeness threshold is a bound k such that, if no counterexample with length k or less for an LTL formula is found, then the formula holds for all infinite paths in the transition system. For instance, the completeness threshold of invariant properties $\Box p$ where p is an atomic proposition is equal to the *diameter* (longest distance between any two states) of the transition system [17]. Likewise, the completeness threshold for liveness properties $\Diamond q$ is given by the *recurrent diameter* (the length of loop-free path) [39].

6.2.2 Time-Difference Linear Temporal Logic

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

Definition 6.1 ([77, Definition 11])**.** A time-difference proposition over a set of variables $\mathcal{V}^{(0)}$ is an atomic formula $p = \mathtt{x}_i^{(k)} - \mathtt{x}_j^{(l)} \sim \alpha$, 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 $\mathtt{x}_i - \mathtt{x}_j \sim \alpha$ instead. For $m \geq 0$, we write $p^{(m)} = \mathtt{x}_i^{(k+m)} - \mathtt{x}_j^{(l+m)} \sim \alpha$. □

Definition 6.2 ([77, Definition 12])**.** A TD formula for a vector of variables $\mathcal{V}^{(0)}$ is defined according to the following grammar

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

where $p = \mathtt{x}_i^{(k)} - \mathtt{x}_j^{(l)} \sim \alpha$ is a TD proposition over $\mathcal{V}^{(0)}$. We call a TD formula f initial if all TD propositions appearing in f are initial ones. □

Semantically, we interpret TD formulae on orbits[10]: given an orbit π and a TD formula f we say that $\pi \models f$ according to the following recursive rules.

$$\left.\begin{array}{ll} \pi \models \mathtt{true} & \\ \pi \models p = \mathtt{x}_i^{(k)} - \mathtt{x}_j^{(l)} \sim \alpha & \text{iff } \pi[k]_i \sim \pi[l]_j + \alpha \\ \pi \models \neg f_1 & \text{iff } \pi \not\models f_1, \\ \pi \models f_1 \wedge f_2 & \text{iff } \pi \models f_1 \wedge \pi \models f_2. \end{array}\right\} \tag{6.1}$$

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

$$\pi[0] \models f \text{ iff } \pi \models f, \tag{6.2}$$

or in general $\pi[i] \models f$ iff $\pi[i..] \models f$. Furthermore, two similar orbits π, σ yield the same satisfaction over a TD formula.

Proposition 6.1 ([77, Proposition 14])**.** Given a TD formula f and two similar orbits π, σ i.e., $\sigma[0] = \beta \otimes \pi[0]$ for $\beta \in \mathbb{R}$. We have $\pi \models f$ if and only if $\sigma \models f$.

[10] It should be noted that the supercripts in $p = \mathtt{x}_i^{(k)} - \mathtt{x}_j^{(l)} \sim \alpha$ simply represent the step-bound of the corresponding variables.

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 $p = \mathtt{x}_i^{(k)} - \mathtt{x}_j^{(l)} \sim \alpha$,

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

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

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 $\mathtt{get\_initial}(A, f)$ translation defined as follows. First, we translate all the TD propositions of f in the form of $\mathtt{x}_i^{(k)} - \mathtt{x}_j^{(l)} \sim \alpha$ into the following inequality

$$\bigoplus_{r=1}^{n} \left(\mathtt{x}_r + A^{\otimes k}(i,r)\right) \sim \bigoplus_{s=1}^{n} \left(\mathtt{x}_s + \alpha + A^{\otimes l}(j,s)\right), \tag{6.3}$$

then, we can translate f into an initial TD formula by applying the rewriting (4.9) in Proposition 4.3 for each inequality (6.3). We summarise the discussion in the following proposition and then present an example in Example 6.1.

Proposition 6.2 ([77, Proposition 13])**.** Given a TD formula f and $A \in \mathbb{R}_{\max}^{n\times n}$, let g be $\mathtt{get\_initial}(A, f)$. Then, for any orbit $\pi \in \mathsf{Orb}(A)$, $\pi \models f$ iff $\pi \models g$.

Proof. Due to Definition 6.2 and (6.1), it suffices to prove for a non-initial TD proposition $p = \mathtt{x}_i^{(k)} - \mathtt{x}_j^{(l)} \sim \alpha$. Notice that that p is equivalent to an inequality (6.3) which can be translated into an initial TD formula by Propositions 4.2 and 4.3. This completes the proof. □

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 $g = \mathtt{get\_initial}(A, f)$ by Proposition 6.2 may result in a larger formula, in terms of the number of its propositions. Notice that the number of propositions in (4.9) is at most $\lfloor \frac{n}{2} \rfloor \times (n - \lfloor \frac{n}{2} \rfloor)$. On the other hand, as demonstrated in Example 6.1, the advantage of such translation is that one can determine the satisfaction of g on the initial vector.

Example 6.1. Let us consider a TD formula $f = \mathtt{x}_1^{(1)} - \mathtt{x}_1^{(0)} \leq 5$ for an MPL system in Example 2.1. The formula represents "the delay between the second and first departures of trains on station S_1 is no greater than 5 time unit". It consists of one non-initial TD proposition. The translation of into initial TD formula is shown as follows.

$$\mathtt{x}_1^{(0)} - \mathtt{x}_1^{(1)} \geq -5 \;\Leftrightarrow\; \mathtt{x}_1 + 5 \geq \max\{\mathtt{x}_1 + 2, \mathtt{x}_2 + 5\} \;\Leftrightarrow\; \mathtt{x}_1 - \mathtt{x}_2 \geq 0,$$

Hence, we have $\texttt{get\_initial}(A, f) = \mathtt{x}_1 - \mathtt{x}_2 \geq 0$. Suppose we consider an orbit π with initial vector $\mathbf{x}(0) = [0\ 0]^\top$ as in Example 2.2,

$$\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}5\\3\end{bmatrix}, \begin{bmatrix}8\\8\end{bmatrix}, \begin{bmatrix}13\\11\end{bmatrix}, \ldots.$$

It is easy to verify that $\pi \models \mathtt{x}_1^{(1)} - \mathtt{x}_1^{(0)} \leq 5$ since $x_1(1) - x_1(0) = 5$. Similarly, $\pi \models \mathtt{x}_1 - \mathtt{x}_2 \geq 0$ since $x_1(0) - x_2(0) = 0$. □

We are now in the position to discuss TD specifications, which generalise TD formulae by adding temporal operators. Formally, they are defined as LTL formulae in positive normal form (Definition 3.7) defined over a set of initial TD propositions. We call this logic as Time-Difference Linear Temporal Logic (TDLTL) for the rest of this thesis.

Definition 6.3 (TDLTL [77])**.** A TDLTL formulae in defined according to the following grammar

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

where p is an initial TD proposition. □

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 6.2 and demonstrated in Example 6.1. Furthermore, any initial TD formula is a TDLTL formula (without any temporal operators). As described for TD formulae in (6.1), the semantics of TDLTL formulae are defined over orbits according to the following recursive rules:

$$\left.\begin{array}{ll}
\pi \models \texttt{true} & \text{for all } \pi \in \mathsf{Orb}(A), \\
\pi \not\models \texttt{false} & \text{for all } \pi \in \mathsf{Orb}(A), \\
\pi \models p = \mathtt{x}_i - \mathtt{x}_j \sim \alpha & \text{iff } \pi[0] \models p, \\
\pi \models \neg p & \text{iff } \pi \not\models p, \\
\pi \models \varphi_1 \wedge \varphi_2 & \text{iff } \pi \models \varphi_1 \wedge \pi \models \varphi_2, \\
\pi \models \varphi_1 \vee \varphi_2 & \text{iff } \pi \models \varphi_1 \vee \pi \models \varphi_2, \\
\pi \models \bigcirc\varphi & \text{iff } \pi[1..] \models \varphi, \\
\pi \models \varphi_1 \;\mathsf{U}\; \varphi_2 & \text{iff } \exists j \geq 0.\ \pi[j..] \models \varphi_2 \text{ and } \forall 0 \leq i < j.\ \pi[i..] \models \varphi_1, \\
\pi \models \varphi_1 \;\mathsf{R}\; \varphi_2 & \text{iff } \forall j \geq 0.\ \pi[j] \models \varphi_2 \text{ or} \\
 & \quad \exists i \geq 0.\ (\pi[i..] \models \varphi_1 \wedge \forall h \leq i.\ \pi[h..] \models \varphi_2) \\
\pi \models \Diamond\varphi & \text{iff } \exists j \geq 0.\ \pi[j..] \models \varphi, \\
\pi \models \Box\varphi & \text{iff } \forall j \geq 0.\ \pi[j..] \models \varphi.
\end{array}\right\} \quad (6.4)$$

Similar to (6.2), the semantics of TDLTL formulae over initial vectors in the case of an MPL system is as follows: $\pi[0] \models \varphi$ iff $\pi \models \varphi$. Given an MPL system (2.6), characterised by $A \in \mathbb{R}_{\max}^{n\times n}$, and a TDLTL formula φ, we say that $A \models \varphi$ if all the orbits of A satisfy φ (i.e., $\forall \pi \in \mathsf{Orb}(A).\ \pi \models \varphi$). Furthermore, if the dynamics of (2.6) are defined over a set of initial conditions $X \subseteq \mathbb{R}^n$,

$$\begin{array}{ll}
\mathsf{Orb}(A, X) \models \varphi & \text{iff } \forall \pi \in \mathsf{Orb}(A, X).\ \pi \models \varphi \\
 & \text{iff } \forall \mathbf{x}(0) \in X.\ \mathbf{x}(0) \models \varphi
\end{array} \quad (6.5)$$

6.3 Abstraction-Based Technique

This section describes the framework for formal verification of MPL systems by employing bounded model checking (BMC) and abstractions. We recall that the benefit of the abstraction technique is to replace the infinite and continuous state-space MPL system with a finite-state abstract transition system. Given an MPL system (2.6) with a predefined set of initial states $X \subseteq \mathbb{R}^n$ and an TDLTL formula φ, we present a method to check whether all orbits starting from X satisfy φ i.e., $\mathsf{Orb}(A, X) \models \varphi$. There are four main steps of the abstraction-based verification procedure.

First, we generate the abstract transition system that simulates the underlying MPL system. Second, starting with $k = 0$, we search for a counterexample of φ with length k. We use NuSMV 2.6.0 [33] via command `check_ltlspec_bmc_onepb` to apply BMC on the abstract transition system. It performs non-incremental BMC to find a counterexample with a specific bound k. If no such counterexample is present, the command is reapplied for bound $k + 1$. On the other hand, if a counterexample exists, we apply spurious checking of the counterexample (third step, cf. Subsection 5.3.2). In the case of a non-spurious counterexample, one can conclude that the specification is false. Otherwise, we refine the abstract transition system (fourth step, cf. Subsection 5.3.3) such that the counterexample is removed and then reapply the BMC command for bound k. Such a refinement procedure results in a finer transition system.

The above steps are repeated until either 1) we found a non-spurious counterexample, 2) the bound is equal to the completeness threshold (cf. Subsection 5.3.4), or 3) the problem is too hard to solve. The first outcome implies that the specification is false, while the second one indicates that the specification is valid. For the last outcome, we use a relatively large integer $pmax$ as the maximum number of abstract states allowed on the abstract transition system. If the number of abstract states exceeds $pmax$, we terminate the procedure.

6.3.1 Abstraction Procedure

We have discussed the tropical and predicate abstraction procedures for MPL systems in Chapter 3. In this subsection, we develop the combined procedure by considering the advantages and disadvantages of both methods. On the one hand, unlike in predicate abstraction, the resulting abstract states generated by tropical abstraction are already associated with affine dynamics (2.18). Furthermore, the predicate abstraction potentially produces many abstract states (as already shown in Table 3.2) if the state matrix of MPL systems has $m > 2$ finite elements in each row. On the other hand, the predicate abstraction technique is more fitted for formal verification procedure since it is *a specification-induced* method. We recall that in tropical abstraction, the choice of propositions depends on the abstract states (see Remark 3.1).

The combined abstraction technique is presented in Algorithm 6.1. First, we generate abstract states via Algorithm 3.1. Then, we partition the abstract states

according to the TD propositions (or predicates[11].) that appear on φ. If there are non-initial TD propositions in φ, one needs to transform it into an initial TD formula as demonstrated in Example 6.1. The algorithm yields a pair $\mathbf{R}$ and $\widehat{S}$. Each element of $\mathbf{R}$ consists of an abstract state (expressed as DBM) with the corresponding affine dynamic (expressed as a finite coefficient $\mathbf{g}$). Each element of $\widehat{S}$ represents the label of its corresponding abstract state. Having obtained the abstract states, one can generate the abstract transition system via *one-step reachability analysis* as described in Subsection 3.3.2.

Algorithm 6.1 Combined abstraction procedure of an MPL system

function MAXPLUS_COMB_ABS(A, φ)
  $\mathbf{R} \leftarrow$ MAXPLUS_TROP_ABS(A)
  $\widehat{S} \leftarrow \{\texttt{true}\}$
  $P \leftarrow$ GET_PRED(φ)
  for $p \in P$ **do**
    $\widehat{S} \leftarrow \bigcup_{\widehat{s}\in\widehat{S}}\{\widehat{s} \wedge \neg p\} \cup \bigcup_{\widehat{s}\in\widehat{S}}\{\widehat{s} \wedge p\}$
    $\mathbf{R}_{neg} \leftarrow \bigcup_{D\in\mathbf{R}}\{D \cap \texttt{DBM}(\neg p)\}$
    $\mathbf{R}_{pos} \leftarrow \bigcup_{D\in\mathbf{R}}\{D \cap \texttt{DBM}(p_i)\}$
    $\mathbf{R} \leftarrow \mathbf{R}_{neg} \cup \mathbf{R}_{pos}$
    for $j \in \{1, \ldots, |\mathbf{R}|\}$ **do**
      if $\mathbf{R}[j]$ is empty **then**
        remove $\mathbf{R}[j]$ from $\mathbf{R}$
        remove $\widehat{S}[j]$ from $\widehat{S}$
  return $(\mathbf{R}, \widehat{S})$

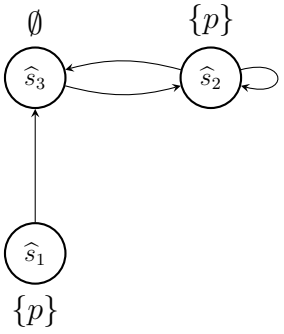
Remark 6.1. In addition to predicates from a TDLTL φ, one may need to generate predicates from the initial conditions $X \subseteq \mathbb{R}^n$ (expressed as a union of DBM). However, the more inequalities in X, the larger number of abstract states. This situation is again underlining the drawback of DBM as a data structure. In this thesis, the abstraction-based technique assumes that $X = \mathbb{R}^n$. By this assumption, we do not get additional predicates. Furthermore, the DBM representation of abstract states does not contain inequality with a single variable. □

To avoid ambiguity, for the rest of this chapter, we denote TS as the transition system associated with MPL system (Definition 3.4) while $\widehat{TS}$ is resulting abstract transition system. Likewise, $\pi = \mathbf{x}(0)\mathbf{x}(1)\ldots$ is an orbit (concrete path) over TS while $\widehat{\pi} = \widehat{s}_0\widehat{s}_2\ldots$ is an abstract path over $\widehat{TS}$. We also denote $\widehat{\varphi}$ as the abstracted version of a TDLDL formula φ.

Example 6.2. Suppose we have an MPL system (2.7) and an TDLTL formula $\varphi = \Diamond\Box(\mathtt{x}_1^{(1)} - \mathtt{x}_1^{(0)} \leq 5)$. The proposition of φ is non-initial. By Example 6.1, after translating the propositions into initial TD formula, it is equivalent to $\varphi = \Diamond\Box(\mathtt{x}_1 - \mathtt{x}_2 \geq 0)$. Hence, we obtain one predicates: $p \equiv \mathtt{x}_1 - \mathtt{x}_2 \geq 0$. From Example 3.2, the resulting abstract states via tropical abstraction are $\{\mathbf{x} \mid \mathbb{R}^2 \mid x_1 - x_2 \geq 3\}$, $\{\mathbf{x} \mid \mathbb{R}^2 \mid 0 \leq x_1 - x_2 < 3\}$, and $\{\mathbf{x} \mid \mathbb{R}^2 \mid x_1 - x_2 < 0\}$. The only abstract state where p does not hold is the last one.

[11]Simply put, a predicate is an abstracted representation of TD proposition.

Leaving the details aside, the resulting abstract transition is shown in Figure 6.2. Notice that, it is the same with the abstract transition system in Figure 3.1. The corresponding LTL formula for the abstract transition system is $\widehat{\varphi} = \Diamond\Box p$. It is straightforward that $\widehat{\varphi}$ is not satisfied. However, again, we cannot conclude that the original MPL system (2.6) does not satisfy φ. □



$$\texttt{DBM}(\widehat{s}_1) = \{\mathbf{x} \in \mathbb{R}^2 \mid x_1 - x_2 \geq 3\}$$
$$\texttt{DBM}(\widehat{s}_2) = \{\mathbf{x} \in \mathbb{R}^2 \mid 0 \leq x_1 - x_2 < 3\}$$
$$\texttt{DBM}(\widehat{s}_3) = \{\mathbf{x} \in \mathbb{R}^2 \mid x_1 - x_2 < 0\}$$

Figure 6.2: Abstract transition system from an MPL system in (2.7) and a TDLTL formula $\varphi = \Diamond\Box(\mathbf{x}_1^{(1)} - \mathbf{x}_1^{(0)} \leq 5)$. All abstract state are initial.

6.3.2 Spuriousness Checking Procedure

Given an abstract transition system $\widehat{TS}$ and an LTL formula $\widehat{\varphi}$, in the case that $\widehat{\varphi}$ does not hold in $\widehat{TS}$, it is possible that the bounded counterexample of $\widehat{\varphi}$ is spurious. Such a counterexample can be represented as a path $\widehat{\pi} = \widehat{s}_0 \dots \widehat{s}_k$ over $\widehat{TS}$ but has no corresponding (concrete) path on the concrete transition system. The spuriousness checking can be done via *forward-reachability analysis*. An abstract path $\widehat{\pi} = \widehat{s}_0 \dots \widehat{s}_k$ is not spurious if and only if there exists $\mathbf{x}(0) \in \texttt{DBM}(\widehat{s}_0)$ such that $\mathbf{x}(i) = A \otimes \mathbf{x}(i-1) \in \texttt{DBM}(\widehat{s}_i)$ for $1 \leq i \leq k$. This can be done by checking the emptiness of DBM $D_0, \dots, D_k$ where $D_0 = \texttt{DBM}(\widehat{s}_0)$ and $D_i = \mathsf{Im}(D_{i-1}) \cap \texttt{DBM}(\widehat{s}_i)$ for $1 \leq i \leq k$. The DBM $\mathsf{Im}(D_{i-1})$ is the image of D_{i-1} w.r.t the underlying MPL system. Since each abstract state has a corresponding affine dynamic (2.18), the image computation can be done by Algorithm 2.5. The spuriousness checking for no-loop paths is summarised in Algorithm 6.2. It also yields a set of DBM that will be used for refinement procedure on the next subsection if the corresponding path is spurious.

The spuriousness checking for lasso-shaped path $\widehat{\pi} = (\widehat{\pi}_{\text{stem}})(\widehat{\pi}_{\text{loop}})^\omega$ is implemented via Algorithm 6.3. We use periodicity checking to deal with the infinite suffix $(\widehat{\pi}_{\text{loop}})^\omega$. In lines 14-22, we check the spuriousness of $(\widehat{\pi}_{\text{stem}})(\widehat{\pi}_{\text{loop}})^{it}$ where $\widehat{\pi}_{\text{loop}}$ is repeated it times. If it is not spurious then we check the periodic behaviour of the sequence of DBM (line 25). We can conclude that $(\widehat{\pi}_{stem})(\widehat{\pi}_{loop})^\omega$ is not spurious if the the periodic behaviour is found. In case of an irreducible MPL system, by Proposition 2.2, such behaviour is guaranteed. On the other hand, if the periodic behaviour cannot be found after large number of iterations N then the algorithm is terminated with an "undecided" result.

Algorithm 6.2 Spuriousness checking of no-loop paths

Inputs: $A \in \mathbb{R}_{\max}^{n\times n}$,
an abstract path $\widehat{\pi}$
Outputs: a Boolean value,
a set of DBM

function IS_SPURIOUS($A, \widehat{\pi}$)
  $k \leftarrow \text{len}(\widehat{\pi})$ ▷ the length of $\widehat{\pi}$
  $D \leftarrow \texttt{DBM}(\widehat{\pi}[0])$
  $\mathbf{D} \leftarrow \{D\}$ ▷ D is the first DBM in $\mathbf{D}$
  $i \leftarrow 0$
  spurious $\leftarrow$ `false`
  while ($i \leq k-1$ and $\neg$*spurious*) **do**
    $S \leftarrow \textsc{Sign}(D)$ ▷ the sign matrix of D
    $\mathbf{g} \leftarrow \textsc{Af}(\widehat{\pi}[i])$ ▷ the affine dynamics for abstract state $\widehat{\pi}[i]$
    $D \leftarrow \textsc{Image\_DBM}(A, D, S, \mathbf{g}) \cap \texttt{DBM}(\widehat{\pi}[i+1])$
    if D is not empty **then**
      add D to $\mathbf{D}$
    else
      spurious $\leftarrow$ `true`
    $i \leftarrow i+1$
  return $\langle$*spurious*, $\mathbf{D}\rangle$

It is straightforward to see that the spuriousness checking for no-loop paths via Algorithm 6.2 is complete. However, this is not the case for Algorithm 6.3 since it may provide an undecided output. This incompleteness of Algorithm 6.3 is owing to non-periodic behaviour that could happen for reducible MPL systems. Conversely, Algorithm 6.3 is complete if the abstract transition system is generated from an irreducible MPL system. Proposition 6.3 relates the spuriousness of an abstract path, either a no-loop or lasso-shaped path, with the value of transient and cyclicity of an irreducible MPL system.

Proposition 6.3 ([74, Lemma 1])**.** Consider an irreducible MPL systems (2.6) with transient l and cyclicity c and the resulting abstract transition system $\widehat{TS}$ that simulates it. Suppose that $\widehat{\pi}$ is a finite path over $\widehat{TS}$.

i. If $\widehat{\pi}$ is a no-loop path with $\text{len}(\widehat{\pi}) \geq l + c$, then it is spurious.

ii. If $\widehat{\pi} = (\widehat{\pi}_{\text{stem}})(\widehat{\pi}_{\text{loop}})^{\omega}$ is a lasso-shaped path with $\text{len}(\widehat{\pi}_{\text{stem}}) + \text{len}(\widehat{\pi}_{\text{loop}}) \geq l + c$, then it is spurious.

Proof.

i. Let assume $\widehat{\pi} = \widehat{s}_0 \dots \widehat{s}_{l+c}$ is not spurious. Thus, there exists $\mathbf{x}(0) \in \texttt{DBM}(\widehat{s}_0)$ such that $\mathbf{x}(i+1) = A \otimes \mathbf{x}(i) = A^{\otimes i+1} \otimes \mathbf{x}(0) \in \texttt{DBM}(\widehat{s}_{i+1})$ for $0 \leq i \leq l+c$. By Proposition 2.2, we have $A^{\otimes l+c} = (\lambda \times c) \otimes A^{\otimes l}$ which implies $\mathbf{x}(l+c) = (\lambda \times c) \otimes \mathbf{x}(l)$. Since $\texttt{DBM}(\hat{s}_l)$ and $\texttt{DBM}(\widehat{s}_{l+c})$ do not contain a single-variable inequality (see Remark 6.1), we have $\texttt{DBM}(\hat{s}_l) = \texttt{DBM}(\widehat{s}_{l+c})$ and then $\widehat{s}_l = \widehat{s}_{l+c}$. This contradicts the fact that the abstract states in $\widehat{\pi}$ must be all different.

ii. Again by Proposition 2.2, for any initial vector $\mathbf{x}(0)$, we have $\mathbf{x}(i+c) = (\lambda \times c) \otimes \mathbf{x}(i)$ for $i \geq l$. This implies that, for any abstract path $\widehat{\pi} = \widehat{s}_0\widehat{s}_1 \dots$ we have $\widehat{s}_l = \widehat{s}_{l+c}$. Therefore, the maximum length for non-spurious lasso-shaped path is $l + c - 1$. □

Algorithm 6.3 Spuriousness checking of lasso-shaped paths

Inputs: $A \in \mathbb{R}_{\max}^{n\times n}$,
abstract paths $\widehat{\pi}_{\text{stem}}$ and $\widehat{\pi}_{\text{loop}}$,
positive integer N
Outputs: a Boolean value,
a set of DBM

```
1: function IS_SPURIOUS(A, π̂_stem, π̂_loop, N)
2:     ⟨spurious, D⟩ ← IS_SPURIOUS(A, π̂_stem)
3:     if spurious then                                   ▷ π̂_stem is already spurious
4:         go to line 31
5:     else
6:         l ← len(π̂_stem)                                ▷ the length of π̂_stem
7:         D ← D[l]                                       ▷ the last DBM in D
8:         m ← len(π̂_loop)                                ▷ the number of states in π̂_loop
9:         it ← 0                                         ▷ the number of iterations
10:        periodic ← false          ▷ Boolean variable to represent the periodic behaviour
11:        while (it ≤ N and ¬spurious and ¬periodic) do
12:            it ← it + 1
13:            i ← 0
14:            while (i ≤ m − 1 and ¬spurious) do
15:                S ← SIGN(D)
16:                g ← AF(π̂_loop[i])
17:                D ← IMAGE_DBM(A, D, S, g) ∩ DBM(π̂_loop[i + 1])
18:                if D is not empty then
19:                    add D to D
20:                else
21:                    spurious ← true
22:                i ← i + 1
23:            j ← |D|                                    ▷ the number of DBM in D,
24:            while (j − m > l and ¬periodic) do          notice that mod(|D|, m) = l
25:                if (D[j − m] = E) then
26:                    periodic ← true
27:                j ← j − m
28:    if (it > N and ¬spurious and ¬periodic) then
29:        print "undecided"
30:    else
31:        return ⟨spurious, D⟩
```

6.3.3 Refinement Procedure

Provided that the counterexample is spurious, it is necessary to refine the abstract transition. Instead of attempting to generate the abstract transition that bisimulates the concrete MPL system as described in Theorem 3.1, we refine the abstract transition based on a spurious counterexample using the concept of *lazy abstraction* [66]. This starts by finding a pivot state, namely a state in which the spuriousness of counterexample starts. Suppose $\widehat{\pi} = \widehat{s}_0 \dots \widehat{s}_k$ is a spurious counterexample. If π is a no-loop path then the pivot state can be found from the number of DBM in $\mathbf{D}$: $\widehat{s}_{|D|-1}$ is the pivot state. On the other hand, for a lasso-shaped path $\widehat{\pi} = (\widehat{\pi}_{\text{stem}})(\widehat{\pi}_{\text{loop}})^{\omega}$, the pivot state is $\widehat{s}_i$ where $i = |D|-1$, if $|D| < \text{len}(\widehat{\pi}_{stem})+1$ (the spuriousness is found in $\widehat{\pi}_{\text{stem}}$), otherwise $i = \text{len}(\widehat{\pi}_{stem})+1+m$ where $m = (|D|-\text{len}(\widehat{\pi}_{stem})-1)$ $\texttt{mod}$ $\text{len}(\widehat{\pi}_{loop})$.

After obtaining the pivot state, the next step is partitioning the pivot state according to the outgoing transitions. Due to the one-step reachability rule for gen-

erating abstract transition, it is impossible that there is only one outgoing from the pivot state. The steps to partition the abstract state with more than one outgoing transitions are described in Subsection 3.3.2. The labels and affine dynamics for the new abstract states are equal to those of the pivot state.

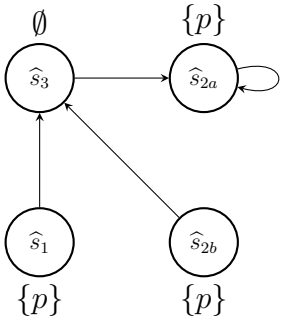
6.3.4 Upper Bound of the Completeness Threshold

For an irreducible MPL system (2.6) with transient $\mathsf{tr}(A) = l$ and cyclicity $\mathsf{cyc}(A) = c$, Proposition 6.3 suggests that it is sufficient to search a counterexample of $\widehat{\varphi}$ up to bound $l + c$ since any counterexample on abstract transition system $\widehat{TS}$ with length beyond $l + c$ is guaranteed to be spurious. Hence, we can conclude that the upper bound for completeness threshold to verify $\widehat{\varphi}$ over $\widehat{TS}$ is $l+c$. On the concrete transition system, this upper bound is unrelated to the TDLTL formula φ and solely depend on the periodic behaviour of the underlying MPL system.

Proposition 6.4 ([74, Lemma 2])**.** Consider an irreducible MPL system (2.6) with transient $\mathsf{tr}(A) = l$ and cyclicity $\mathsf{cyc}(A) = c$ and an TDLTL formula φ. Suppose $\widehat{TS}$ is the resulting abstract transition system with the corresponding abstracted LTL formula $\widehat{\varphi}$. If there is no counterexample of $\widehat{\varphi}$ with length of $l+c-1$ then φ is valid on the original MPL system.

Proof. By Proposition 6.3, any counterexample of $\widehat{\varphi}$ with length greater than $l+c-1$ (if any) is guaranteed to be spurious. As a result, if there is no counterexample of $\widehat{\varphi}$ up to length of $l + c - 1$, then φ is guaranteed to be valid on the original MPL system. □

Example 6.3. From the preceding abstract transition system in Figure 6.2, the LTL formula $\widehat{\varphi} = \Diamond\Box p$ has counterexample with length of two i.e., $\widehat{\pi} = \widehat{s}_1(\widehat{s}_3\widehat{s}_2)^\omega$. Without giving the details, $\widehat{\pi}$ is spurious and the corresponding pivot state is $\widehat{s}_2$. The resulting refined abstract transition system is depicted in Figure 6.3.



$$\mathtt{DBM}(\widehat{s}_1) = \{\mathbf{x} \in \mathbb{R}^2 \mid x_1 - x_2 \geq 3\}$$
$$\mathtt{DBM}(\widehat{s}_{2a}) = \{\mathbf{x} \in \mathbb{R}^2 \mid 0 \leq x_1 - x_2 \leq 2\}$$
$$\mathtt{DBM}(\widehat{s}_{2b}) = \{\mathbf{x} \in \mathbb{R}^2 \mid 2 < x_1 - x_2 < 3\}$$
$$\mathtt{DBM}(\widehat{s}_3) = \{\mathbf{x} \in \mathbb{R}^2 \mid x_1 - x_2 < 0\}$$

Figure 6.3: The resulting refinement of abstract transition system in Figure 6.2.

Notice that, the abstract state $\widehat{s}_2$ is partitioned into to $\widehat{s}_{2a}$ and $\widehat{s}_{2b}$. Coincidentally, all abstract states have only one outgoing transition. It is straightforward that $\widehat{\varphi}$ is satisfied by the refined abstract transition system. Hence, the original MPL system satisfies TDLTL formula $\varphi = \Diamond\Box(\mathbf{x}_1^{(1)} - \mathbf{x}_1^{(0)} \leq 5)$. □

Proposition 6.4 ensures that the completeness threshold to verify φ is not greater than the sum of the transient and cyclicity of the MPL systems. By Proposition 6.4, one could say that the BMC algorithm for irreducible MPL systems is complete for any TDLTL formula. However, this is not the case for reducible MPL systems, due to the incompleteness of Algorithm 6.3.

Figure 6.4 summarises the procedure to verify a TDLTL formula for an irreducible MPL system. A relatively large integer *pmax* is used as a termination condition: if the number of abstract state in $\widehat{TS}$ exceeds *pmax* the procedure is terminated condition with `unknown` outcome. We argue that the procedure can be applied for *boundedly periodic* MPL systems since the corresponding transient $\mathsf{tr}(A)$ exists. In fact, Propositions 6.3 and 6.4 also hold for *boundedly periodic* MPL systems.

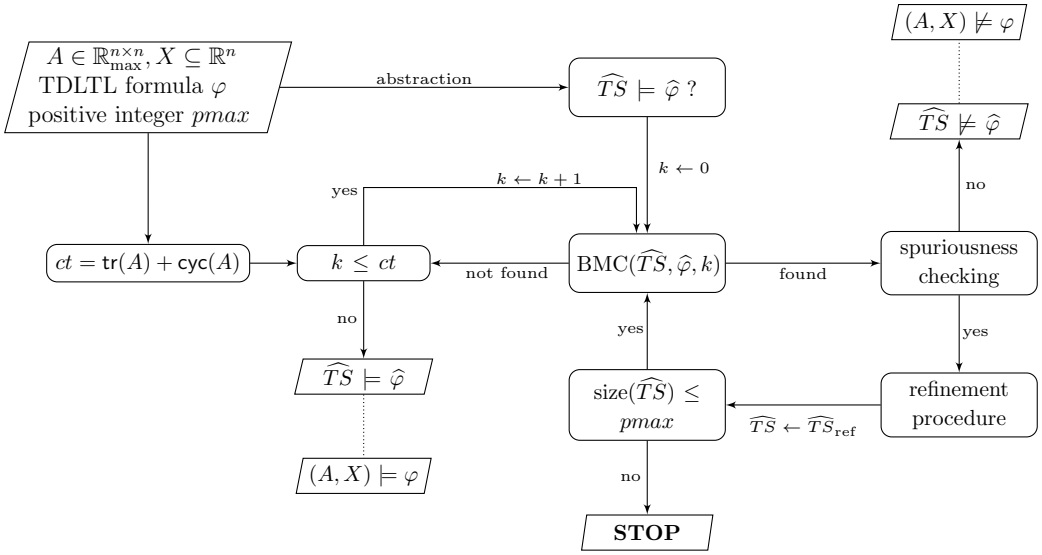


Figure 6.4: The framework of abstraction-based BMC for irreducible MPL systems. The trapezium represents input or output while the rounded rectangle represents procedure or computation. The solid line shows the initiation of steps while the dashed one shows an implication relation.

Remark 6.2. The abstraction-based bounded model checking can be developed similarly for MiPL systems since all the underlying steps are not restricted for MPL systems. First, the abstraction step can be applied to MiPL systems, as demonstrated in Subsection 3.5.1. Second, the spuriousness checking and refinement steps via DBM manipulation are also applicable to MiPL systems due to Remark 2.3. Third, Propositions 6.3-6.4 also hold for irreducible (or, *boundedly periodic* in general) MiPL systems since the computation of transient for min-plus matrix can be done by Algorithm 4.5. However, this technique cannot be implemented for IMPL systems since the refinement procedure for abstract transition system generated from an IMPL system is too complicated (see Remark 3.3). Furthermore, one cannot find the upper bound of the completeness threshold since the orbits of an IMPL system may never be periodic. □

6.4 SMT-Based Technique

Inspired by the procedures in Section 5.3 to solve reachability analysis of MPL systems, this section presents the SMT-based method to verify (6.5) where we build on the basic idea of BMC. However, instead of generating the abstraction of MPL systems, we encode the bounded counterexample of a TDLTL formula φ for each iteration. Due to the periodic behaviour of MPL systems, such a counterexample corresponds to an orbit with transient l and cyclicity c. This thesis introduces four variants of algorithms associated with two independent factors.

The first factor is the computation of the bound, that could be carried out either *incrementally* or *upfront*. If no counterexample with the current bound is found in the incremental algorithms, we increment the bound. Unlike the usual incremental BMC where the bound is increased by one, the algorithms increase the bound by finding the existence of another orbit with larger transient l' or cyclicity c'. This increment step is repeated as long as the bound does not exceed the completeness threshold given by Proposition 6.6. In the upfront algorithms, one only needs to encode the bounded counterexample of φ w.r.t. the pre-computing completeness threshold.

The other factor is the unrolling of the orbits of MPL systems, which can either *unrolled* or *initialised*. In the unrolled algorithms, we symbolically encode the orbit of MPL systems using (5.10). This encoding uses multiple sets of SMT variables up to the current bound. On the other hand, the initialised algorithms employ only one set of variables thanks to some algebraic properties of MPL systems and Proposition 6.2.

6.4.1 Encoding Bounded Counterexamples

This subsection describes the generation of an SMT instance corresponding to a bounded counterexample of the TDLTL formula φ. We first define the notion of *max-plus lasso* which incorporates the lasso path over MPL systems. Then, we show that the bounded counterexample of φ can be expressed as a TD formula.

An orbit $\pi = \mathbf{x}(0)\mathbf{x}(1)\ldots$ of (2.6) is called max-plus lasso if there exist $\delta \in \mathbb{R}$ and $k \geq l \geq 0$ such that $\mathbf{x}(k+1+j) = \delta \otimes \mathbf{x}(l+j)$ for $j \geq 0$. It is important to note that the definition of max-plus lasso is slightly different from the *canonical* lasso in Figure 6.1(b) which requires that the l-th and $(k+1)$-th states are the same. To avoid ambiguity, we will write (k, l, δ)-lasso to refer a max-plus lasso. The illustration for a (k, l, δ)-lasso is depicted in Figure 6.5.

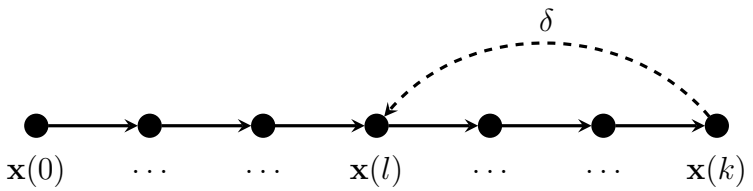


Figure 6.5: An illustration of a max-plus lasso. The dashed arrow represents the transition $\mathbf{x}(k+1) = \delta \otimes \mathbf{x}(l)$.

The following proposition shows the relation between a max-plus lasso and the existence of local transient and cyclicity.

Proposition 6.5 ([77, Proposition 6])**.** An orbit π w.r.t. $A \in \mathbb{R}_{\max}^{n\times n}$ 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 $j \geq 0$ where λ is the eigenvalue of A. As a result, π is a (k, l, δ)-lasso with $k = l + c - 1$ and $\delta = \lambda \times c$.
($\Rightarrow$) Suppose π is a (k, l, δ)-lasso: $\pi[k+1+j] = \delta \otimes \pi[l+j]$ for $j \geq 0$. Consequently, $\pi[j+k-l+1] = \delta \otimes \pi[j]$ for $j \geq l$ which shows that $\mathsf{tr}(A, \pi[0]) = l$ and $\mathsf{cyc}(A, \pi[0]) = k - l + 1$. □

Corollary 6.1 ([77, Corollary 7])**.** An MPL system (2.6) is periodic iff all orbits $\pi \in \mathsf{Orb}(A)$ are lasso.

Proof. Direct consequence of Proposition 6.5 and the notion of periodic MPL systems in Definition 2.8. □

Before describing the encoding for bounded counterexample of a TDLTL φ, we define the bounded version of (6.4) up to bound k for a (k, l, δ)-lasso $\pi \in \mathtt{Orb}(A)$ as follows:

$$
\left.\begin{array}{ll}
\pi \models_k \mathtt{true} & \\
\pi \not\models_k \mathtt{false} & \\
\pi \models_k p & \text{iff } \pi[0] \models p, \\
\pi \models_k \neg p & \text{iff } \pi \not\models_k p, \\
\pi \models_k \varphi_1 \wedge \varphi_2 & \text{iff } \pi \models_k \varphi_1 \wedge \pi \models_k \varphi_2, \\
\pi \models_k \varphi_1 \vee \varphi_2 & \text{iff } \pi \models_k \varphi_1 \vee \pi \models_k \varphi_2, \\
\pi \models_k \bigcirc\varphi & \text{iff } \pi[1..] \models_k \varphi, \\
\pi \models_k \varphi_1 \mathsf{U} \varphi_2 & \text{iff } \exists 0 \leq j \leq k.\ \pi[j..] \models_k \varphi_2 \text{ and} \\
 & \quad \forall 0 \leq i < j.\ \pi[i..] \models_k \varphi_1, \\
\pi \models_k \varphi_1 \mathsf{R} \varphi_2 & \text{iff } \forall 0 \leq j \leq k.\ \pi[j] \models_k \varphi_2 \text{ or} \\
 & \quad \exists 0 \leq i \leq k.\ (\pi[i..] \models_k \varphi_1 \wedge \forall h \leq i.\ \pi[h..] \models_k \varphi_2), \\
\pi \models_k \Diamond\varphi & \text{iff } \exists \leq j \leq k.\ \pi[j..] \models_k \varphi, \\
\pi \models_k \Box\varphi & \text{iff } \forall 0 \leq j \leq k.\ \pi[j..] \models_k \varphi.
\end{array}\right\} \quad (6.6)
$$

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

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. $\psi = \neg\varphi$. We recall that φ and ψ are assumed to be in positive normal form. The notation ${}_l[\psi]_k^m$ denotes the witness encoding of ψ (equivalently, the counterexample encoding of φ) at position $0 \leq m \leq k$ over a (k, l, δ)-lasso. Similar to the description in [17], the

encoding can be formulated as follows:

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

where $p = \mathtt{x}_i - \mathtt{x}_j \sim \alpha$ is an initial TD proposition. The final formula, which is satisfiable iff there exists a (k, l, δ)-lasso π such that $\pi \not\models_k \varphi$, is given by:

$$
\bigwedge_{i=0}^{k} \mathtt{SymbMPL}(A, \mathcal{V}^{(i)}, \mathcal{V}^{(i+1)}) \wedge {}_\iota[\psi]_k^0 \wedge \mathtt{Loop}(A, k+1, l, \delta), \tag{6.7}
$$

where $\mathtt{Loop}(A, k+1, l, \delta)$ is the looping constraint i.e., $\bigwedge_{i=1}^{n}(\mathtt{x}_i^{(k+1)} - \mathtt{x}_i^{(l)} = \delta)$. Recall that, the first conjunct of (6.7) represents the executions of (2.6) up to bound k.

By the same procedure used in Proposition 6.2, one can translate $\mathtt{Loop}(A, k+1, l, \delta) \wedge {}_\iota[\psi]_k^0$ 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). Therefire, the "initialised" version of (6.7) is

$$
\mathtt{get\_initial}\left(A, {}_\iota[\psi]_k^0 \wedge \mathtt{Loop}(A, k+1, l, \delta)\right). \tag{6.8}
$$

The first conjunct of (6.7) is not included because all TD propositions in (6.8) are initial ones. The number of TD propositions in (6.8) may be much larger compared to (6.7), especially for TDLTL formulae with multiple temporal operators. However, in some cases, it is possible that the former is simpler. For instance, considering $\psi = \Box p$, we have ${}_\iota[\psi]_k^0 = \bigwedge_{j=0}^{k} p^{(j)}$. If there exists a j such that the translation $p^{(j)}$ into an initial TD formula yields `false`, then the resulting encoding can be expressed as `false`. On the other hand, the encoding (6.8) has an advantage w.r.t. the number of variables: notice that (6.7) consists of variables from $\mathcal{V}^{(0)} \cup \ldots \cup \mathcal{V}^{(k+1)}$, whereas (6.8) involves $\mathcal{V}^{(0)}$ only.

6.4.2 Upper Bound of Completeness Threshold

As in the abstraction-based technique, the pair of transient and cyclicity determines the upper bound of completeness threshold to verify (6.5) via SMT. However, unlike

Proposition 6.4 which relates the global transient $\mathsf{tr}(A)$ and global cyclicity $\mathsf{cyc}(A)$, a smaller upper bound can be achieved from the transient and cyclicity w.r.t. the set of initial conditions X.

Proposition 6.6 ([77, Proposition 17])**.** Given an MPL system (2.6) with a set of initial conditions X and a TDLTL formula φ. If the transient $\mathsf{tr}(A, X)$ exists then the completeness threshold to verify $\mathsf{Orb}(A, X) \models \varphi$ is $\mathsf{tr}(A, X) + \mathsf{cyc}(A, X) - 1$.

Proof. By Corollary 6.1, 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 $(k, l, \lambda \times c)$-lasso where $k = l + c - 1$ and λ is the eiganvalue of A. Consequently, the upper bound of completeness threshold to verify (6.5) is given by the largest possible of such k i.e., $\mathsf{tr}(A, X) + \mathsf{cyc}(A, X) - 1$. □

We recall that the computation for $\mathsf{tr}(A, X)$ and $\mathsf{cyc}(A, X)$ can be implemented by Algorithm 4.3 for any set of initial conditions X that can be expressed as a QF-LRA formula. The upper bound in Proposition 6.6 is similar to the bound given by Proposition 5.2 since reachability analysis is a specific verification problem when the specifications only consist of a temporal operator $\Diamond$.

Example 6.4. Let us solve the verification problem presented in Example 6.2 using SMT. The underlying TDLTL formula is $\varphi = \Diamond\Box((\mathtt{x}_1 - \mathtt{x}_2 \geq 0) \wedge (\mathtt{x}_1 - \mathtt{x}_2 \geq -2))$. Hence, $\psi = \neg\varphi = \Box\Diamond((\mathtt{x}_2 - \mathtt{x}_1 > 0) \vee (\mathtt{x}_2 - \mathtt{x}_1 > -2))$. Since the MPL system (2.7) has eigenvalue $\lambda = 4$ as well as transient $l = \mathsf{tr}(A, \mathbb{R}^2) = 2$ and cyclicity $c = \mathsf{cyc}(A, \mathbb{R}^2) = 2$, we have $k = l + c - 1 = 3, \delta = \lambda \times c = 8$, and

$$
\begin{aligned}
{}_l[\psi]_k^0 &= \bigwedge_{j=0}^{3} \left(\bigvee_{i=\min\{j,l\}}^{3} (\mathtt{x}_2^{(i)} - \mathtt{x}_1^{(i)} > 0) \vee (\mathtt{x}_2^{(i)} - \mathtt{x}_1^{(i)} > -2) \right), \\
\mathtt{Loop}(A, k+1, l, \delta) &= \bigwedge_{i=1}^{2} (\mathtt{x}_i^{(4)} - \mathtt{x}_i^{(2)} = 8).
\end{aligned}
$$

Finally, the corresponding encoding (6.7) is given by $\bigwedge_{i=1}^{3} F_i \wedge {}_l[\psi]_k^0 \wedge \mathtt{Loop}(A, k+1, l, \delta)$ where

$$
\begin{aligned}
F_i = &(\mathtt{x}_1^{(i+1)} - \mathtt{x}_1^{(i)} \geq 2) \wedge (\mathtt{x}_1^{(i+1)} - \mathtt{x}_2^{(i)} \geq 5) \wedge \\
&((\mathtt{x}_1^{(i+1)} - \mathtt{x}_1^{(i)} = 2) \vee (\mathtt{x}_1^{(i+1)} - \mathtt{x}_2^{(i)} = 5)) \wedge \\
&(\mathtt{x}_2^{(i+1)} - \mathtt{x}_1^{(i)} \geq 3) \wedge (\mathtt{x}_2^{(i+1)} - \mathtt{x}_2^{(i)} \geq 3) \wedge \\
&((\mathtt{x}_2^{(i+1)} - \mathtt{x}_1^{(i)} = 3) \vee (\mathtt{x}_2^{(i+1)} - \mathtt{x}_2^{(i)} = 3)).
\end{aligned}
$$

By satisfiability checking using an SMT solver, the resulting encoding is unsatisfiable. Therefore, no counterexample for φ up to bound 3. This implies that φ is satisfied by an MPL system in (2.7). □

Proposition 6.6 suggests that the completeness to verify $\mathsf{Orb}(A, X) \models \varphi$ 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 an MPL system (2.6) is boundedly periodic, then the verification problem (6.5) is decidable. On the other hand, (6.5) is semi-decidable for unboundedly periodic MPL systems since the computation of transient for these models is semi-complete [75]. It is also shown in [75] that the computation of transient for unboundedly periodic MPL systems is semi-complete. We further summarise this discussion in the following corollary.

Corollary 6.2 ([77, Corollary 18])**.** Suppose we have an MPL system (2.6) with a set of initial conditions 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 6.6, verifying $\mathsf{Orb}(A, X) \models \varphi$ is decidable for boundedly MPL systems and semi-decidable for unboundedly periodic ones. □

In the following two Subsections, we present the SMT-based procedures, incremental and upfront, to verify (6.5). In the former ones, we incrementally search for a bounded counterexample of the TDLTL formula in the form of a max-plus lasso. Unlike the standard BMC technique, the bound may not be increased by one. Instead, the new bound is incremented w.r.t. the values of transient and cyclicity. In the latter procedures, we search a counterexample with the largest bound possible, which corresponds to the completeness threshold in [38]. In the same manner of Corollary 6.2, also discussed in [77], the proposed procedures in Subsection 6.4.3-6.4.4 are complete for boundedly periodic MPL systems and semi-complete for unboundedly periodic ones.

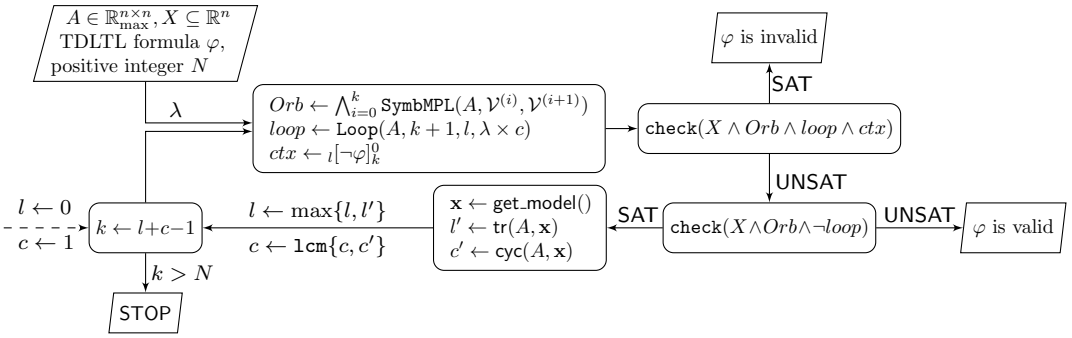
6.4.3 Incremental Approaches

Since the orbits of MPL system are potentially periodic with transient l and cyclicity c, we search a finite counterexample of a TDLTL formula φ with length $k = l + c - 1$ in a shape of max-plus lasso.We initially set $l = 0$ and $c = 1$, the smallest possible values. With these values, we generate the looping constraint $\mathtt{Loop}(A, k+1, l, \lambda \times c)$ and ${}_l[\neg\varphi]_k^0$, where λ is the max-plus eigenvalue of A. The formula $X \wedge F$, with F given by (6.7) or (6.8), corresponds to a counterexample of φ i.e., a $(k, l, \lambda \times c)$-lasso $\pi \in \mathsf{Orb}(A, X)$, such that $\pi \not\models \varphi$.

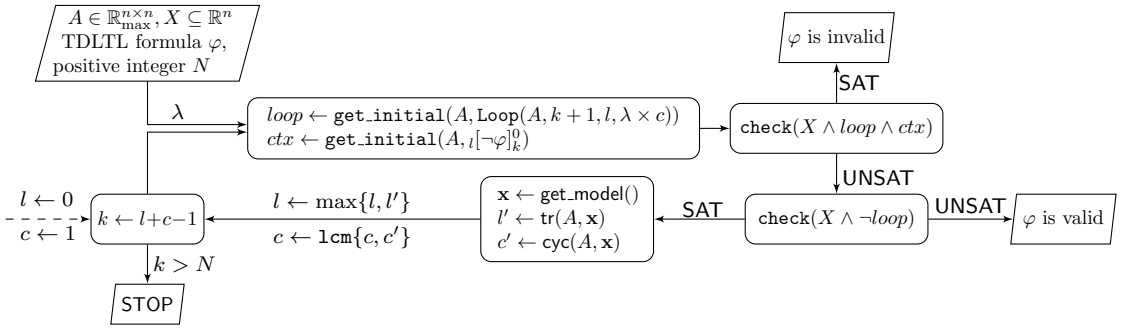
We use an SMT solver to check the satisfiability of $X \wedge F$. If the SMT solver reports `SAT`, then a counterexample is found. On the other hand, if the SMT solver reports `UNSAT`, we increment the bound. Instead of increasing the bound by one, we use SMT solver to check whether there exists an orbit with transient l' and cyclicity c' where either $l' > l$ or c' do not divide c. The value of $\max\{l, l'\} + \mathtt{lcm}\{c, c'\} - 1$ becomes the new bound. Notice that this increment of the bound is similarly used in Algorithm 4.3. 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 6.6. For the last outcome, we use a large integer as a termination condition. This undecided output could happen if the underlying MPL system is unboundedly periodic.

The incremental approaches are illustrated in Figure 6.6. We name the procedure that uses the encoding in (6.7) *unrolled-incremental*, since the first conjunct of (6.7) represents the execution of (2.6) up to bound k. On the other hand, the alternative procedure with the encoding in (6.8) is called *initialised-incremental*, due to the translation from Proposition 6.2.



(a) Unrolled-incremental procedure using the encoding in (6.7).



(b) Initialised-incremental procedure using the encoding in (6.8).

Figure 6.6: Incremental SMT-based bouded model checking of MPL systems. The function `check` is implemented with an SMT solver. The integer N represents the allowed maximum bound.

6.4.4 Upfront Approaches

Upfront approaches exploit the fact that the upper bound of the completeness threshold in Proposition 6.6 is unrelated to the TDLTL formula φ, and that it can be computed via SMT-based techniques, as in Algorithm 4.3. Hence, the upfront versions of the procedures in Figure 6.6 is obtained by generating (6.7) or (6.8) with $l = \mathsf{cyc}(A, X)$ 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 *initialised-upfront*, respectively.

6.4.5 Extensions to Other Models

The SMT-based bounded model checking can be implemented similarly for MiPL systems due to the duality relation with MPL systems. First of all, one can define the TD proposition and TD formula w.r.t. a min-plus matrix $B \in \mathbb{R}_{\min}^{n\times n}$ the similar way in Propositions 6.1-6.2. Second, the definition of TDLTL formula φ can be defined for an MiPL system accordingly. As such, the satisfaction relation is defined over (min-plus) orbit $\pi \in \mathsf{Orb}(B)$ as in (6.1) and (6.4).

Following the above descriptions, the min-plus version for encoding (6.7) is formulated as

$$\bigwedge_{i=0}^{k} \mathtt{SymbMiPL}(B, \mathcal{V}^{(i)}, \mathcal{V}^{(i+1)}) \wedge {}_{l}[\neg\varphi]_{k}^{0} \wedge \mathtt{Loop}(B, k+1, l, \delta). \tag{6.9}$$

Likewise, the initialised encoding is given by

$$\mathtt{get\_initial}\left(B, {}_{l}[\neg\varphi]_{k}^{0} \wedge \mathtt{Loop}(B, k+1, l, \delta)\right). \tag{6.10}$$

The only difference between (6.8) and (6.10) is that the former is governed by Proposition 4.3 while the latter is governed by Proposition 4.6. The encodings (6.9) and (6.10) are used for *unrolled-incremental* and *initialised-incremental* algorithms, respectively. Lastly, the *unrolled-upfront* and *initialised-upfront* algorithms can be implemented by computing the transient and cyclicity w.r.t. the set of initial conditions i.e., $\mathsf{tr}_{\min}(B, X)$ and $\mathsf{cyc}_{\min}(B, X)$. These two values determine the completeness threshold to verify the satisfaction of φ over MiPL systems (as in Proposition 6.6). Furthermore, they determine the decidability of verifying $\mathsf{Orb}(B, X) \models \varphi$ (as in Corollary 6.2).

Proposition 6.7. Given an MiPL system (2.10) with a set of initial conditions X and a TDLTL formula φ. If the transient $\mathsf{tr}_{\min}(B, X)$ exists then the completeness threshold to verify $\mathsf{Orb}(B, X) \models \varphi$ is given by $\mathsf{tr}_{\min}(B, X) + \mathsf{cyc}(B, X) - 1$. □

Corollary 6.3. Suppose we have an MiPL system (2.10) with a set of initial condition X and a TDLTL formula φ. If the (2.10) is boundedly periodic (resp. unboundedly periodic), then verifying $\mathsf{Orb}(B, X) \models \varphi$ is a decidable (resp. semi-decidable) problem. □

The extension to IMPL systems is slightly different since the corresponding orbits may be never periodic. Moreover, one cannot determine the completeness threshold as in Propositions 6.6-6.7. Consequently, the SMT-based bounded model checking for IMPL systems can only be done incrementally where we increment the bound by one. Unlike the algorithms for MPL and MiPL systems, we cannot associate the counterexample with an orbit π that has a specific pair of transient l and cyclicity c. Thus, the counterexample encoding for IMPL systems is expressed as follows

$$\left.\begin{aligned} &\left(\bigwedge_{i=0}^{k} \mathtt{SymbIMPL}(\mathcal{A}^{(i)}, \mathcal{V}^{(i)}, \mathcal{V}^{(i+1)}) \wedge \mathtt{Mat}(\mathbf{A}, \mathcal{A}^{(i)})\right) \wedge \\ &\qquad \left(\bigvee_{l=0}^{k} {}_{l}[\neg\varphi]_{k}^{0} \wedge \mathtt{Loop}(\mathbf{A}, k+1, l, \delta_l)\right). \end{aligned}\right\} \tag{6.11}$$

The first conjunct of (6.11) corresponds to the symbolic representation of IMPL orbits up to bound k while the second one encodes the counterexample of φ within bound k with different values of transient l. The looping constraints in (6.11) are associated with the unknown variable δ_l for each $0 \leq l \leq k$.

Remark 6.3. Unlike in MPL and MiPL systems, the initialised algorithms cannot be implemented to IMPL systems. The reason is that the image computation in IMPL systems is not uniquely defined.

The SMT-based BMC for IMPL systems is summarised in Figure 6.7. First, we initialise the bound to be $k = 0$. The bound will be incremented by one if the corresponding bounded counterexample does not exist (the SMT solver reports UNSAT). This process is repeated until either the SMT solver reports SAT or k exceeds the maximum bound N. The latter outcome only implies that the counterexample is not found until bound N. Consequently, we can conclude that the problem to verify a TDLTL formula φ for an IMPL system is undecidable.

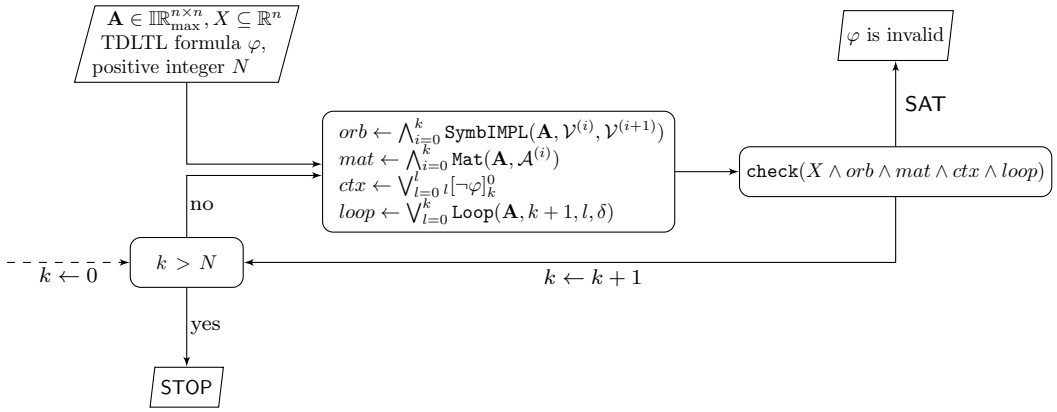


Figure 6.7: SMT-based bouded model checking of IMPL systems. The function `check` is implemented with an SMT solver. The integer N represents the allowed maximum bound.

6.5 Computational Benchmarks

This section presents the experimental evaluation for bounded model checking of MPL systems (and IMPL systems) over a random TDLTL formulae benchmark set. The experiments are implemented on an Intel(R) Xeon(R) Gold 6248 CPU @ 2.50GHz with 120GB of RAM.

6.5.1 Max-Plus Linear Systems

Before we present the setups for the experiments, we briefly describe an additional procedure that using a symbolic model checker nuXmv [28], a symbolic model checker for the analysis of synchronous, finite- and infinite-state systems,

Encoding in nuXmv The procedure encodes the maximization operation into SMV language, the input language of NuSMV and nuXmv. 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 (5.10). However, unlike the procedures in Figure 6.6, nuXmv requires that the lasso path is in the canonical form. This means that, in Figure 6.5, the states $\mathbf{x}(l)$ and $\mathbf{x}(k+1)$ must be equal. This condition can be achieved if the corresponding matrix has an eigenvalue equal to 0. It is straightforward that if matrix A in (2.6) has eigenvalue λ then $B = A \otimes (-\lambda)$ has eigenvalue 0.

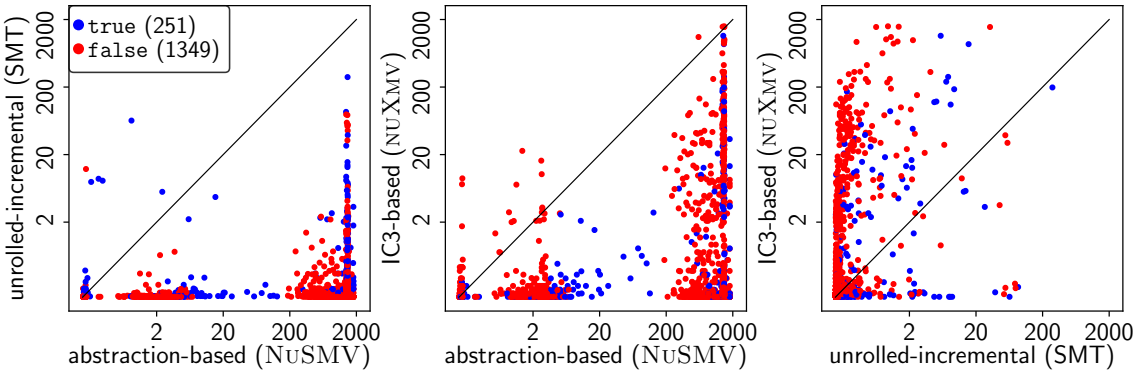
The procedure to verify (6.5) using nuXmv is as follows. First, we update the matrix by subtracting all elements with the corresponding eigenvalue. Then, we generate an SMV file from the matrix and the TDLTL specification[12]. Finally, we call nuXmv to verify the specification using a command `check_ltlspec_ic3` that implements the algorithm described in the [47]. 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[13] [34] underlying safety checker.

Experimental Setup For the experiments, we generate 20 irreducible matrices of dimension $n \in \{4, 6, 8, 10\} \cup \{12, 16, 20, 24, 28, 32, 36, 40\}$ 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 [56], which categorises them according to their size: the size of a TDLTL formula φ is intended as the number of operators and propositions in φ. For instance, the size of $p \wedge q$ and $p \,\mathsf{U}\, q$ is both three, while $\Box p$ has a size of two. Lastly, for each experiment, the set of initial conditions are $X = \mathbb{R}^n$. The experiments are implemented for each pair matrix and specification. Thus, there are $20 \times 20 \times 2$ experiments for each n (and for each algorithm). We set 30 minutes as `timeout` limit.

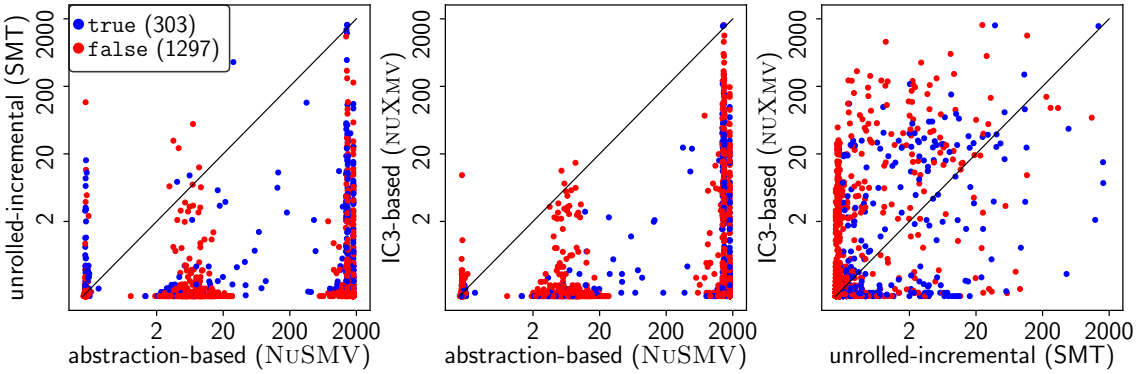
Results Figure 6.8 illustrates the performance comparison of the abstraction-based algorithm (Figure 6.4), unrolled-incremental algorithm (Figure 6.6(a)), and IC3-based technique. These three algorithms are similar because they unroll the dynamics of MPL systems up to the underlying step bound. The scattered plots in Figure 6.8 represent the running times for a pair of algorithms. As expected, the dimension of MPL systems heavily affects the runtime of the abstraction-based procedure: all experiments for 10×10 MPL systems yield `timeout` (see Table 6.2 in

[12] We present an example of the generation of the SMV file in the Appendix.
[13] IC3 stands for "Incremental Construction of Inductive Clauses for Indubitable Correctness"

Appendix). Those experiments may be stopped during the abstraction procedure (Subsection 6.3.1) and not performing the remaining ones (Subsections 6.3.2-6.3.3) at all. For this reason, we do not pursue abstraction-based experiments for higher dimensions. The plots 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 the IC3-based technique with SMT-based incremental (unrolled and initialised) algorithms.



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



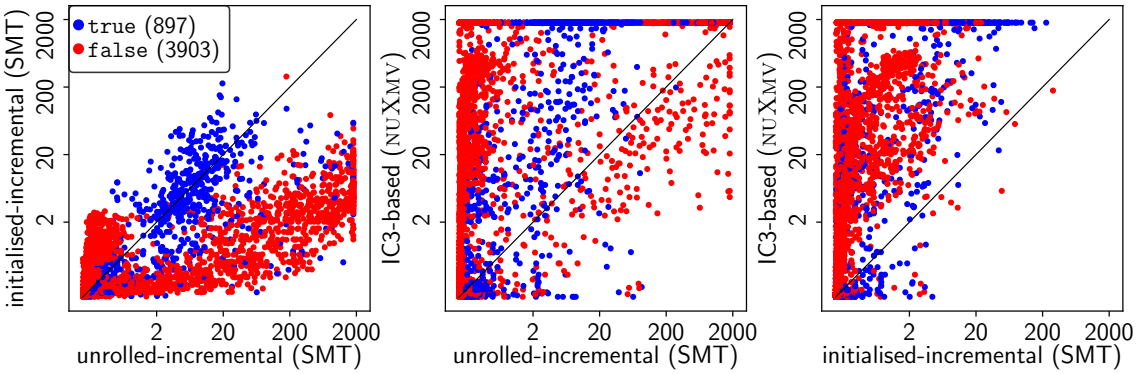
(b) The plots of experiments with TDLTL formulae of size 10.

Figure 6.8: The comparison of abstraction-based, unrolled-incremental and IC3-based algorithms for dimensions $n \in \{4, 6, 8, 10\}$. The plots are presented using a logarithmic scale, with running times in seconds.

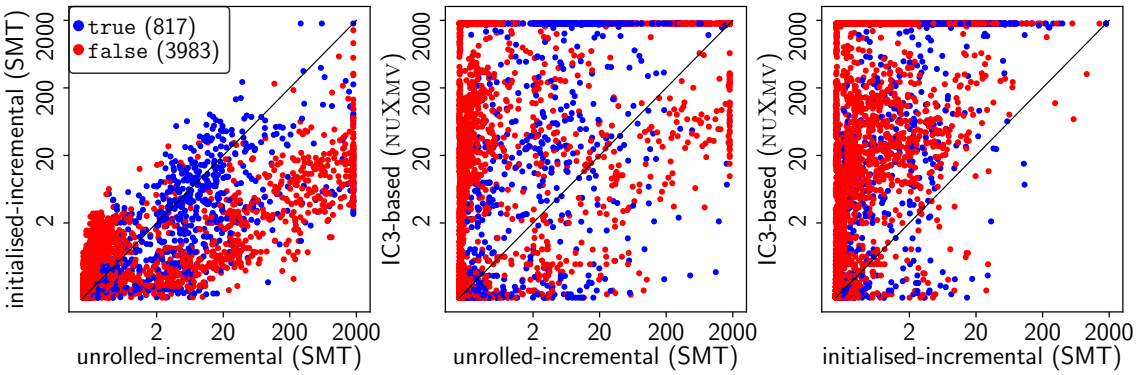
As depicted in Figure 6.9, the proposed algorithms outperform the procedure that employs IC3. We recall that, in incremental algorithms, the bound of the counterexample is not increased by one. Hence, they are indeed more effective to find a long counterexample. Furthermore, in each iteration, the counterexample search is implemented with a fixed loopback bound. Such a bound is related to the current value of transient l.

While the SMT-based incremental algorithms take a longer time to verify specifications with the 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 the IC3-based procedure: the number of experiments that yield `timeout` increases as the dimension grows. On the other hand, the performance of incremental algorithms

are affected by the upper bound of completeness threshold given by Proposition 6.6; 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 6.6(b), all TD propositions in (6.8) are initial ones. Therefore, only one set of variables appeared in (6.8). In comparison, there are $k+1$ sets of variables in (6.7) which correspond to the states of MPL system (2.6) from bound 0 until k.



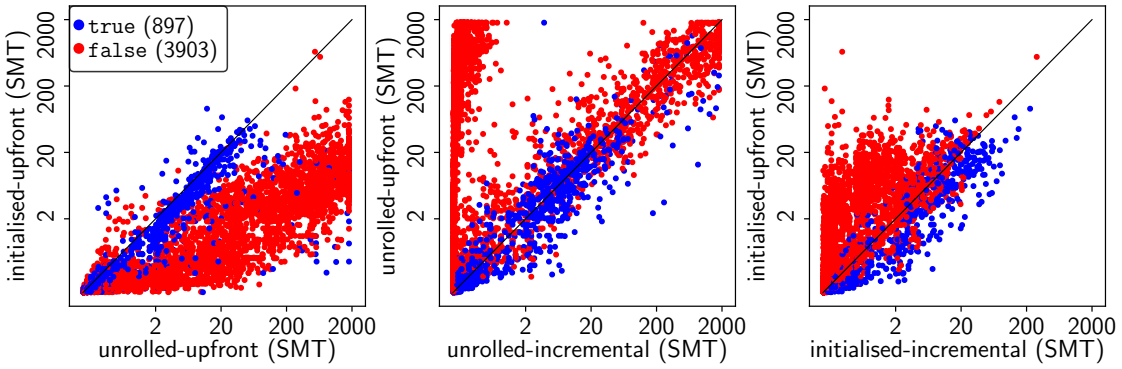
(a) The plots of experiments with TDLTL formulae of size 5.



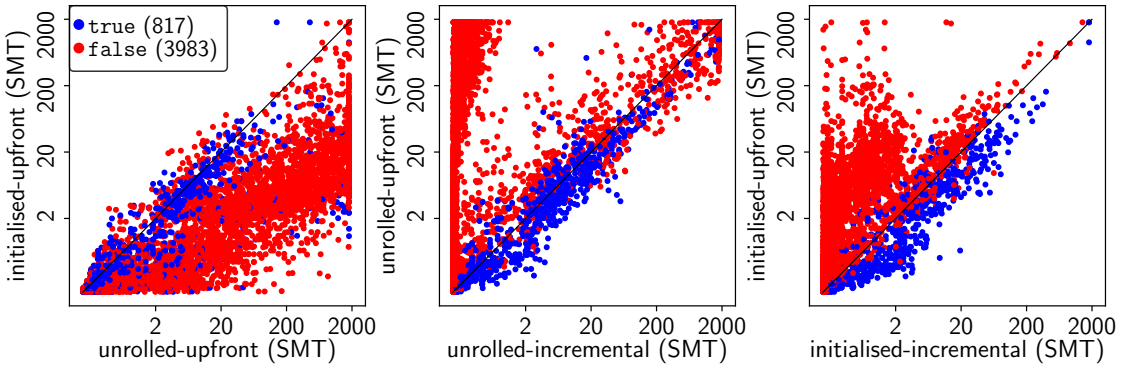
(b) The plots of experiments with TDLTL formulae of size 10.

Figure 6.9: The comparison of incremental algorithms (unrolled and initialised) and IC3-based technique. The plots are presented using a logarithmic scale, with running times in seconds.

We then compare the performance between incremental and upfront algorithms, which are shown in Figure 6.10. 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 situation happens because the counterexample may be found at a smaller bound than the completeness threshold given in Proposition 6.6. Furthermore, as in incremental approaches, the procedure that uses one set of variables (initialised-upfront) is faster than the other employing 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 6.10: The comparison of incremental and upfront algorithms. The plots are presented using a logarithmic scale, with running times in seconds.

6.5.2 Interval Max-Plus Linear Systems

Experimental Setups For the experiments, we generate 20 matrices $\underline{A}, \overline{A} \in \mathbb{R}_{\max}^{n \times n}$ where $n \in \{4, 6, 8, 10\} \cup \{12, 16, 20, 24, 28, 32, 36, 40\}$. For each matrix, there are $\frac{n}{2}$ finite elements as integers between 1 and 20. The locations of the finite elements are chosen randomly. We recall that the upper and lower matrices $\underline{A}, \overline{A}$ are generated under the conditions in Definition 2.10. Unlike the setup in Subsection 6.5.1, the matrices are not required to be irreducible. The remaining setups for initial conditions and the specifications are defined similarly as in Subsection 6.5.1. For each experiment, we run the procedure in Figure 6.7 for $N = 20$.

Results Table 6.1 summarises the performance of SMT-based bounded model checking over IMPL systems. In the table, we present the average and maximum runtime of experiments for each dimension of IMPL systems and the number of successful experiments that yield `true` and `false`. We recall that if the output is `false`, then a counterexample is found within bound N. On the other hand, when no counterexample is detected up to bound N, then the output is `true`: this does not imply, however, the lack of counterexample for the bound beyond N.

As depicted in Table 6.1, most experiments result in `false` which are much less contribute to the average and maximum running time: indeed, the experiments that yield `true` runs much longer. Furthermore, the table suggests that the size

of TDLTL formulae does not directly affect the running time. It also seems as if the dimension of IMPL systems does not influence the runtime. However, based on the experiments, it does. Moreover, the larger the dimension, the more `timeout` experiments.

Table 6.1: Computational benchmark for bounded model checking of IMPL systems.

n	Experiments with TDLTL formulae of size 5			Experiments with TDLTL formulae of size 10		
	runtime	#`false`	#`true`	runtime	#`false`	#`true`
4	$\{5.87, 256.95\}$	366	34	$\{2.99, 332.07\}$	395	5
6	$\{0.32, 60.62\}$	397	3	$\{4.84, 127.88\}$	384	16
8	$\{1.73, 47.89\}$	384	16	$\{5.11, 686.01\}$	389	11
10	$\{10.07, 252.2\}$	384	16	$\{2.9, 127.83\}$	391	9
12	$\{0.13, 5.07\}$	400	0	$\{0.08, 2.3\}$	400	0
16	$\{9.05, 1054.73\}$	391	9	$\{6.08, 840.9\}$	396	4
20	$\{18.65, 652.21\}$	382	18	$\{9.98, 1431.86\}$	399	1
24	$\{7.99, 340.51\}$	400	0	$\{7.29, 178.75\}$	400	0
28	`timeout`(18)	365	17	$\{25.64, 1319.72\}$	400	0
32	`timeout`(34)	366	0	`timeout`(24)	376	0
36	`timeout`(72)	314	14	`timeout`(41)	359	0
40	`timeout`(103)	297	0	`timeout`(47)	353	0

6.6 Summary

This chapter describes the procedures to verify specifications over MPL systems automatically. Such specifications are formally defined as Linear Temporal Logic (LTL) over time difference propositions. In the chapter, we propose abstraction- and SMT-based procedures grounded on bounded model checking techniques. While the abstraction-based algorithm is much less scalable, the SMT-based algorithms comfortably manage models with reasonable size. Furthermore, it has been shown that the bounded model checking over MPL systems is decidable whenever the corresponding transient (w.r.t. a set of initial conditions) exist. Finally, the extension of the procedures for MiPL and IMPL systems have also been discussed.

Appendix

The Generation of an SMV file from an MPL system and TDLTL specification

Let us consider a verification problem in Example 6.2. To verify the TDLTL formula $\varphi = \Diamond\Box((\mathtt{x}_1 - \mathtt{x}_2 \geq 0) \wedge (\mathtt{x}_1 - \mathtt{x}_2 \geq -2))$, one needs to write an SMV file as follows:

```
MODULE main
  VAR x_1 : real;
  VAR x_2 : real;

TRANS ((next(x_1) >= ((2.0 + x_1) - 4)) & (next(x_1) >= ((5.0 + x_2) - 4))) &
      ((next(x_1) = ((2.0 + x_1) - 4)) | (next(x_1) = ((5.0 + x_2) - 4)));
TRANS ((next(x_2) >= ((3.0 + x_1) - 4)) & (next(x_2) >= ((3.0 + x_2) - 4))) &
      ((next(x_2) = ((3.0 + x_1) - 4)) | (next(x_2) = ((3.0 + x_2) - 4)));

LTLSPEC F(G((x_1 - x_2 >= 0) & (x_1 - x_2 >= -2)));
```

We recall that the underlying MPL in (2.7) has a max-plus eigenvalue $\lambda = 4$.

The Runtime and Memory Consumption

The following tables present the algorithms' average and maximum running time (in second) and memory consumption (in MB).

Table 6.2: 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 6.3: 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 6.4: 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 6.5: 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 6.6: 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 6.7: 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)	$\{29.83, 47.21\}$

Chapter 7

Conclusions and Future Research

This thesis has discussed two novel procedures for finite abstractions of MPL, MiPL and IMPL systems. Furthermore, we have introduced a new SMT-based technique to compute the pair of transient and cyclicity for the models. Finally, we have applied these new techniques to solve relevant problems, such as reachability analysis and formal verification. In this chapter, we summarise our main contributions and formulate future research directions.

7.1 Conclusions

The main contributions of the thesis are:

- **Abstraction of MPL systems.** In Chapter 3, we have developed novel abstraction procedures applicable to MPL, MiPL and IMPL systems. The resulting finite abstraction has been proven to simulate the original model. We have discussed requirements under which the existence of an abstraction that bisimulates the original MPL or MiPL system is guaranteed.

- **SMT-based computation of transient of MPL systems.** Chapter 4 has discussed the computation of transient and cyclicity of MPL systems by utilising SMT solving. It has been proven that any inequality in max-plus algebra can be expressed as a QF-RDL formula. This translation has been applied to the incremental SMT-based procedure to obtain the pair of transient and cyclicity. Furthermore, we have shown that a similar procedure can be implemented on MiPL systems.

- **Reachabily analysis of MPL systems.** In Chapter 5, we have elaborated the procedures to solve reachability analysis over MPL systems. First, we have discussed the existing procedures that employ abstraction techniques and compute the reach sets. Then we have introduced the novel SMT-based procedures. We have proved that the reachability analysis is decidable whenever the transient exists. The extension of both procedures (abstraction- and SMT-based) for MiPL and IMPL systems have been thoroughly discussed. Furthermore, we have developed the SMT-based procedure to solve quantified

reachability analysis by allowing quantifiers for the set of initial conditions and state matrices.

- **Bounded model checking of MPL systems.** In Chapter 6, we have developed the framework to verify time-difference specifications over MPL systems. First, we introduced the notion of Time-Difference Linear Temporal Logic (TDLTL) to express the specifications of interest. Then we describe two novel procedures (abstraction- and SMT-based) to verify TDLTL specifications. While the former technique can only be applied for MPL systems with a relatively small dimension, the latter is much more scalable and faster. Further, we have shown that the SMT-based procedures are complete for any MPL system (with a set of initial conditions) that admits local transient.

7.2 Recommendations for Future Research

We discuss a few interesting topics that can be considered for future research.

- **Models.** In this thesis, we have applied the procedures in Chapters 3-6 for MPL, MiPL and IMPL systems. There are some other models that can be considered, such as switching MPL systems [90, 91], stochastic MPL systems [65], stochastic switching MPL systems [92], max-min-omega systems [81], and max-min-plus scaling systems [53]. We deem it worth looking at improving and extending the procedures in this thesis to those models.

  Currently, the proposed procedures in the thesis are guaranteed to be complete if the models (for MPL and MiPL systems) are eventually periodic with a specific pair of transient and cyclicity. We are interested in examining the completeness of the procedures for non-periodic models.

- **Specifications.** In Subsection 6.2.2, we have introduced TDLTL specifications, LTL formulae defined over time-difference propositions. Considering other logic formulations such as Computational Tree Logic (CTL) [11, Definition 6.1] and CTL* [11, Definition 6.80] is possible extension to our results. The specifications expresed as CTL or CTL* formulae can be applied for IMPL systems whose the dynamics are not deterministic.

  Similarly, we are interested in extending the probabilistic specification expressed in Probabilistic CTL (PCTL) [11, Definition 10.36] and PCTL* which can be used for MPL systems with stochastic behaviour.

- **Tools.** We are interested in implementing the resulting procedures (e.g., reachability and formal verification) in this thesis into a tool/software. The existing MATLAB tool named VeriSiMPL can be found in [3, 7]. The new tool will be developed in C++ and will be linked with two libraries: 1) Armadillo [83] to enable matrix operations on max-plus algebra (including DBM manipulations) and 2) an SMT solver Z3 [50]. As a matter of fact, many MATLAB functions are converted in Armadillo.

We also plan to cover the classical problems over MPL, MiPL and IMPL systems such as eigenproblem and solving linear equations for the tool.

A prototype of this tool has participated in the 4th International Competition on Verifying Continuous and Hybrid Systems (ARCH-COMP'20). The technical report covering the results of this competition can be found in [24].

- **Applications.** We have presented several examples (e.g., Example 5.1 and Example 6.2) in reachability analysis and bounded model checking of two-dimensional MPL systems. We are interested in applying the resulting procedures in this thesis to large-scale applications such as railway networks [64,89].

Bibliography

[1] A. Abate, A. Cimatti, A. Micheli, and M. S. Mufid. Computation of the transient in max-plus linear systems via SMT-solving. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 161–177. Springer, 2020.

[2] D. Adzkiya. *Finite Abstractions of Max-Plus-Linear Systems: Theory and Algorithms*. PhD thesis, Delft University of Technology, 2014.

[3] D. Adzkiya and A. Abate. VeriSiMPL: Verification via biSimulations of MPL models. In K. Joshi, M. Siegle, M. Stoelinga, and P. D'Argenio, editors, *Proc. 10th International Conference on Quantitative Evaluation of Systems (QEST'13)*, volume 8054 of *Lecture Notes in Computer Science*, pages 253–256, Hiedelberg, 2013. Springer.

[4] D. Adzkiya, B. De Schutter, and A. Abate. Finite abstractions of max-plus-linear systems. *IEEE Transactions on Automatic Control*, 58(12):3039–3053, 2013.

[5] D. Adzkiya, B. De Schutter, and A. Abate. Backward reachability of autonomous max-plus-linear systems. *IFAC Proceedings Volumes*, 47(2):117–122, 2014.

[6] D. Adzkiya, B. De Schutter, and A. Abate. Forward reachability computation for autonomous max-plus-linear systems. In E. Abraham and K. 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.

[7] D. Adzkiya, Y. Zhang, and A. Abate. VeriSiMPL 2: An open-source software for the verification of max-plus-linear systems. *Discrete Event Dynamic Systems*, 26(1):109–145, 2016.

[8] M. Alirezaei, T. J. van den Boom, and R. 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.

[9] S. Amari, I. Demongodin, and J. J. Loiseau. Control of linear min-plus systems under temporal constraints. In *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC'05. 44th IEEE Conference on*, pages 7738–7743. IEEE, 2005.

[10] F. Baccelli, G. Cohen, G. J. Olsder, and J.-P. Quadrat. *Synchronization and linearity: an algebra for discrete event systems*. John Wiley & Sons Ltd, 1992.

[11] C. Baier and J.-P. Katoen. *Principles of model checking*. MIT Press, 2008.

[12] T. Ball, R. Majumdar, T. Millstein, and S. K. Rajamani. Automatic predicate abstraction of c programs. In *In Proc. Programming Language Design and Implementation 2001 (PLDI'01)*, volume 36, pages 203–213. ACM, 2001.

[13] C. Barrett, L. De Moura, and A. Stump. SMT-COMP: Satisfiability modulo theories competition. In K. Etessami and S. K. Rajamani, editors, *Intl. Conf. on Computer Aided Verification* (CAV'05), volume 3576 of *LNCS*, pages 20–23. Springer, 2005.

[14] C. Barrett, A. Stump, and C. Tinelli. The satisfiability modulo theories library, 2010. `http://smtlib.cs.uiowa.edu`.

[15] C. Barrett and C. Tinelli. Satisfiability modulo theories. In *Handbook of Model Checking*, pages 305–343. Springer, 2018.

[16] A. Bemporad, G. Ferrari-Trecate, and M. Morari. Observability and controllability of piecewise affine and hybrid systems. *IEEE transactions on automatic control*, 45(10):1864–1876, 2000.

[17] A. Biere, A. Cimatti, E. Clarke, and Y. 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.

[18] A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, Y. Zhu, et al. Bounded model checking. *Advances in Computers*, 58(11):117–148, 2003.

[19] A. Biere, K. Heljanko, T. Junttila, T. Latvala, and V. Schuppan. Linear encodings of bounded LTL model checking. *Logical Methods in Computer Science*, 2(5):1–64, 2006.

[20] R. Boukra, S. Lahaye, and J.-L. Boimond. New representations for (max,+) automata with applications to performance evaluation and control of discrete event systems. *Discrete Event Dynamic Systems*, 25(1-2):295–322, 2015.

[21] J.-L. Bouquard, C. 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.

[22] C. A. Brackley, D. S. Broomhead, M. C. Romano, and M. Thiel. A max-plus model of ribosome dynamics during mRNA translation. *Journal of Theoretical Biology*, 303:128–140, 2012.

[23] R. E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys (CSUR)*, 24(3):293–318, 1992.

[24] L. Bu, A. Abate, D. Adzkiya, M. S. Mufid, R. Ray, Y. Wu, and E. Zaffanella. ARCH-COMP20 category report: Hybrid systems with piecewise constant dynamics and bounded model checking. In G. Frehse and M. Althoff, editors, *The 7th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH20)*, volume 74 of *EPiC Series in Computing*, pages 1–15. EasyChair, 2020.

[25] P. Butkovič, H. Schneider, et al. Generators, extremals and bases of max cones. *Linear algebra and its applications*, 421(2-3):394–406, 2007.

[26] R. M. F. Cândido. *Reachability Analysis of Uncertain Max Plus Linear Systems*. PhD thesis, Universitè d'Angers, 2017.

[27] R. M. F. Cândido, L. Hardouin, M. Lhommeau, and R. S. Mendes. Conditional reachability of uncertain max plus linear systems. *Automatica*, 94:426–435, 2018.

[28] R. Cavada, A. Cimatti, M. Dorigatti, A. Griggio, A. Mariotti, A. Micheli, S. Mover, M. Roveri, and S. Tonetta. The nuxmv symbolic model checker. In *International Conference on Computer Aided Verification*, pages 334–342. Springer, 2014.

[29] K. Cechlárová. Eigenvectors of interval matrices over max–plus algebra. *Discrete applied mathematics*, 150(1-3):2–15, 2005.

[30] S. Chaki, E. Clarke, A. Groce, and O. Strichman. Predicate abstraction with minimum predicates. In *Proc. Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME 2003)*, volume 2860 of *Lecture Notes in Computer Science*, pages 19–34. Springer, 2003.

[31] T. M. Chan. More algorithms for all-pairs shortest paths in weighted graphs. *SIAM Journal on Computing*, 39(5):2075–2089, 2010.

[32] B. Charron-Bost, M. Függer, and T. Nowak. New transience bounds for max-plus linear systems. *Discrete Applied Mathematics*, 219:83–99, 2017.

[33] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV 2: An opensource tool for symbolic model checking. In *Proc. of International Conference on Computer Aided Verification (CAV'02)*, pages 359–364. Springer, 2002.

[34] A. Cimatti, A. Griggio, S. Mover, and S. Tonetta. IC3 modulo theories via implicit predicate abstraction. In E. Ábrahám and K. 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.

[35] A. Cimatti, A. Griggio, B. J. Schaafsma, and R. Sebastiani. The MATHSAT5 SMT solver. In N. Piterman and S. A. Smolka, editors, *Intl. Conf. on Tools and Algorithms for the Construction and Analysis of Systems* (TACAS'13), volume 7795 of *LNCS*, pages 93–107. Springer, 2013.

[36] E. Clarke, A. Biere, R. Raimi, and Y. Zhu. Bounded model checking using satisfiability solving. *Formal methods in system design*, 19(1):7–34, 2001.

[37] E. Clarke, O. Grumberg, M. Talupur, and D. Wang. Making predicate abstraction efficient. In *Proc. International Conference on Computer Aided Verification 2003 (CAV'03)*, pages 126–140. Springer, 2003.

[38] E. Clarke, D. Kroening, J. Ouaknine, and O. Strichman. Completeness and complexity of bounded model checking. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*, pages 85–96. Springer, 2004.

[39] E. Clarke, D. Kroening, J. Ouaknine, and O. Strichman. Completeness and complexity of bounded model checking. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*, pages 85–96. Springer, 2004.

[40] E. Clarke, D. Kroening, N. Sharygina, and K. Yorav. Predicate abstraction of ANSI-C programs using SAT. *Formal Methods in System Design*, 25(2-3):105–127, 2004.

[41] E. Clarke, D. Kroening, N. Sharygina, and K. Yorav. SATABS: SAT-based predicate abstraction for ANSI-C. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 570–574. Springer, 2005.

[42] E. Clarke, M. Talupur, H. Veith, and D. Wang. SAT based predicate abstraction for hardware verification. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 78–92. Springer, 2003.

[43] E. M. Clarke, O. Grumberg, and D. E. Long. Model checking and abstraction. *ACM transactions on Programming Languages and Systems (TOPLAS)*, 16(5):1512–1542, 1994.

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

[45] S. Cotton, E. Asarin, O. Maler, and P. Niebert. Some progress in satisfiability checking for difference logic. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, pages 263–276. Springer, 2004.

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

[47] J. Daniel, A. Cimatti, A. Griggio, S. Tonetta, and S. Mover. Infinite-state liveness-to-safety via implicit abstraction and well-founded relations. In S. Chaudhuri and A. 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.

[48] S. Das, D. L. Dill, and S. Park. Experience with predicate abstraction. In *Proc. International Conference on Computer Aided Verification (CAV'99)*, pages 160–171. Springer, 1999.

[49] L. Daviaud, P. Guillon, and G. Merlet. Comparison of max-plus automata and joint spectral radius of tropical matrices. In K. G. Larsen, H. L. Bodlaender, and J.-F. Raskin, editors, *Proc. 42nd International Symposium on Mathematical Foundations of Computer Science (MFCS 2017)*, volume 83 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 19:1–19:14, Dagstuhl, 2017.

[50] L. De Moura and N. Bjørner. Z3: An efficient smt solver. In C. R. Ramakrishnan and J. 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.

[51] B. De Schutter. Upper bounds for the index of cyclicity of a matrix. Technical report, Tech. rep. 98-32, ESATSISTA, KU Leuven, Leuven, Belgium, 1999.

[52] B. 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.

[53] B. De Schutter and T. Van den Boom. Model predictive control for max-min-plus-scaling systems. In *Proceedings of the 2001 American Control Conference*, volume 1, pages 319–324. IEEE, 2001.

[54] B. De Schutter and T. van den Boom. Max-plus algebra and max-plus linear discrete event systems: An introduction. In *Proc. 9th International Workshop on Discrete Event Systems (WODES 2008)*, pages 36–42, May 2008.

[55] D. L. Dill. Timing assumptions and verification of finite-state concurrent systems. In J. Sifakis, editor, *Proc. International Conference on Computer Aided Verification*, volume 407 of *Lecture Notes in Computer Science*, pages 197–212, Hiedelberg, 1989. Springer.

[56] A. Duret-Lutz, A. Lewkowicz, A. Fauchille, T. Michaud, E. Renault, and L. 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.

[57] B. Dutertre. Yices 2.2. In *Intl. Conf. on Computer Aided Verification* (CAV'14), volume 8559 of *LNCS*, pages 737–744, 2014.

[58] K. Fahim, J. van der Woude, et al. On a generalization of power algorithms over max-plus algebra. *Discrete Event Dynamic Systems*, 27(1):181–203, 2017.

[59] C. Flanagan and S. Qadeer. Predicate abstraction for software verification. In *Proc. 29th Principles of Programming Languages (POPL'02)*, volume 37, pages 191–202. ACM, 2002.

[60] R. W. Floyd. Algorithm 97: shortest path. *Communications of the ACM*, 5(6):345, 1962.

[61] M. Gavalec, J. Plávka, and D. Ponce. Tolerance types of interval eigenvectors in max-plus algebra. *Information Sciences*, 367:14–27, 2016.

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

[63] W. Heemels, B. De Schutter, and A. Bemporad. Equivalence of hybrid dynamical models. *Automatica*, 37(7):1085–1091, July 2001.

[64] B. Heidergott, G. J. Olsder, and J. 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.

[65] B. F. Heidergott. *Max-plus linear stochastic systems and perturbation analysis*, volume 15. Springer Science & Business Media, 2006.

[66] T. A. Henzinger, R. Jhala, R. Majumdar, and G. Sutre. Lazy abstraction. In *Proc. of the ACM Symposium on Principles of Programming Languages (POPL'02)*, pages 58–70, 2002.

[67] A. Imaev and R. P. Judd. Hierarchial modeling of manufacturing systems using max-plus algebra. In *Proc. American Control Conference, 2008*, pages 471–476, June 2008.

[68] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In R. Majumdar and V. Kunčak, editors, *Intl. Conf. on Computer Aided Verification* (CAV'17), volume 10426 of *LNCS*, pages 97–117. Springer, 2017.

[69] Y. Li, A. Albarghouthi, Z. Kincaid, A. Gurfinkel, and M. Chechik. Symbolic optimization with smt solvers. In *ACM SIGPLAN Notices*, volume 49, pages 607–618. ACM, 2014.

[70] P. A. Lotito, E. M. Mancinelli, and J.-P. Quadrat. A min-plus derivation of the fundamental car-traffic law. *IEEE Transactions on Automatic Control*, 50(5):699–705, 2005.

[71] G. Merlet, T. Nowak, and S. Sergeev. Weak CSR expansions and transience bounds in max-plus algebra. *Linear Algebra and its Applications*, 461:163–199, 2014.

[72] M. Molnárová. Generalized matrix period in max-plus algebra. *Linear algebra and its applications*, 404:345–366, 2005.

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

[74] M. S. Mufid, D. Adzkiya, and A. 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.

[75] M. S. Mufid, D. Adzkiya, and A. Abate. Symbolic reachability analysis of high dimensional max-plus linear systems. *IFAC-PapersOnLine*, 53(4):459–465, 2020. 15th IFAC Workshop on Discrete Event Systems WODES 2020 - Rio de Janeiro, Brazil, 11-13 November 2020.

[76] M. S. Mufid, D. Adzkiya, and A. Abate. SMT-based reachability analysis of high dimensional interval max-plus linear systems. *IEEE Transactions on Automatic Control*, 2021.

[77] M. S. Mufid, A. Micheli, A. Abate, and A. Cimatti. SMT-Based Model Checking of Max-Plus Linear Systems. In S. Haddad and D. Varacca, editors, *32nd International Conference on Concurrency Theory (CONCUR 2021)*, volume 203 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 22:1–22:20, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[78] H. Myšková. Interval max-plus systems of linear equations. *Linear Algebra and its Applications*, 437(8):1992–2000, 2012.

[79] H. Myšková. Interval max-plus matrix equations. *Linear Algebra and its Applications*, 492:111–127, 2016.

[80] T. Nowak and B. Charron-Bost. An overview of transience bounds in max-plus algebra. *Tropical and Idempotent Mathematics and Applications*, 616:277–289, 2014.

[81] E. L. Patel. *Maxmin-plus models of asynchronous computation.* PhD thesis, University of Manchester, 2012.

[82] M. Péron and N. Halbwachs. An abstract domain extending difference-bound matrices with disequality constraints. In *Proc. 8th International Conference on Verification, Model-checking, and Abstract Intepretation (VMCAI'07)*, volume 43497 of *Lecture Notes in Computer Science*, pages 268–282. Springer, January 2007.

[83] C. Sanderson and R. Curtin. Armadillo: a template-based c++ library for linear algebra. *Journal of Open Source Software*, 1(2):26, 2016.

[84] S. Sergeev. Max-plus definite matrix closures and their eigenspaces. *Linear algebra and its applications*, 421(2-3):182–201, 2007.

[85] Y. Shang, L. Hardouin, M. Lhommeau, and C. A. Maia. Robust controllers in disturbance decoupling of uncertain max-plus linear systems: an application to a high throughput screening system for drug discovery. In *International Workshop on Discrete Event Systems* (WODES'16), pages 404–409. IEEE, 2016.

[86] M. R. Shoaei, L. Kovács, and B. Lennartson. Supervisory control of discrete-event systems via IC3. In *Haifa Verification Conference*, pages 252–266. Springer, 2014.

[87] E. Sontag. Nonlinear regulation: The piecewise linear approach. *IEEE Transactions on automatic control*, 26(2):346–358, 1981.

[88] G. Soto Y Koelemeijer. *On the behaviour of classes of min-max-plus systems.* PhD thesis, Delft University of Technology, 2003.

[89] Subiono and J. van der Woude. Power algorithms for (max,+)-and bipartite (min, max,+)-systems. *Discrete Event Dynamic Systems*, 10(4):369–389, 2000.

[90] T. van den Boom and B. De Schutter. Model predictive control for switching max-plus-linear systems with random and deterministic switching. *IFAC Proceedings Volumes*, 41(2):7660–7665, 2008.

[91] T. J. van den Boom and B. De Schutter. Modeling and control of switching max-plus-linear systems with random and deterministic switching. *Discrete Event Dynamic Systems*, 22(3):293–332, 2012.

[92] S. Van Loenhout, T. Van Den Boom, S. Farahani, and B. De Schutter. Model predictive control for stochastic switching max-plus-linear systems. *IFAC Proceedings Volumes*, 45(29):79–84, 2012.

[93] M. Witczak, P. Majdzik, R. Stetter, and G. Bocewicz. Interval max-plus fault-tolerant control under resource conflicts and redundancies: application to the seat assembly. *International Journal of Control*, pages 1–13, 2019.

[94] Y. Zhang, S. Blusseau, S. Velasco-Forero, I. Bloch, and J. Angulo. Max-plus operators applied to filter selection and model pruning in neural networks. In *International Symposium on Mathematical Morphology and Its Applications to Signal and Image Processing*, pages 310–322. Springer, 2019.

Glossary

List of Symbols and Notations

Below follows a list of the most frequently used symbols and notations in this thesis.

A	state matrix of MPL systems
A_{ext}	extension of state matrix w.r.t MPL systems
$\mathbf{A}$	interval matrix of IMPL systems
$\underline{A}, \overline{A}$	lower and upper matrix of IMPL systems resp.
$\mathcal{AP}$	set of atomic propositions in transition systems
B	state matrix of MiPL systems
$c, \mathsf{cyc}(\cdot), \mathsf{cyc}(\cdot,\cdot)$	cyclicity in MPL systems
$\mathsf{cone}(\cdot), \mathsf{rcone}(\cdot)$	max-plus cone
$\mathsf{cone}_{\min}(\cdot), \mathsf{rcone}_{\min}(\cdot)$	min-plus cone
$\mathsf{cyc}_{\min}(\cdot)$	cyclicity in MiPL systems
c	max-plus conjugate operator
D, S	pair of matrices representing DBM
$\texttt{DBM}(\cdot)$	DBM associated with an atomic proposition and abstract state
$E(\cdot)$	max-plus eigenspace
$\mathsf{EqFunc}(\cdot,\cdot)$	"equality function" formula
f	abstraction function
$\mathbf{g}$	finite coefficient
$\mathcal{G}(\cdot)$	precedence graph in MPL and IMPL systems
$\mathcal{G}_{\min}(\cdot)$	precedence graph in MiPL systems
I, I_f	initial states of concrete and abstract transition systems resp.
$\mathsf{Im}(\cdot,\cdot)$	image operator w.r.t. models
I_n	identity matrix in max-plus algebra
$\mathsf{Inv}(\cdot,\cdot)$	inverse operator w.r.t.models
k	discrete-event counter
$l, \mathsf{tr}(\cdot), \mathsf{tr}(\cdot,\cdot)$	transient of MPL systems
$\texttt{left}(\cdot)$	left index of a predicate
$L(\cdot), L_f(\cdot)$	labeling function of concrete and abstract transition systems resp.

$\mathrm{len}(\cdot)$	length of path in transition systems
$\mathtt{Mat}(\cdot,\cdot)$	SMT formula w.r.t. the state matrix of IMPL systems
n	dimension of the state space
N	event horizon
$\mathbb{N}$	set of natural numbers, i.e. $\{1, 2, 3, \dots\}$
$\mathcal{O}$	big O notation
$\mathsf{Orb}(\cdot), \mathsf{Orb}(\cdot,\cdot)$	set of orbits w.r.t. models
$p, p_1, p_2, \dots$	atomic propositions (predicates)
P	set of predicates
$Paths(\cdot)$	set of all paths in transition systems
$Post(\cdot)$	operator yielding the direct successors in transition systems
$\mathtt{right}(\cdot)$	right index of a predicate
R	region of PWA systems w.r.t MPL and MiPL systems
$\overline{\mathsf{R}}$	region of PWA systems w.r.t IMPL systems
$\mathcal{R}$	binary relation
$s, \widehat{s}$	concrete and abstract states of transition systems resp.
$S, \hat{S}$	state space of concrete and abstract transition systems resp.
$\mathcal{S}_{\cdot}$	set of states corresponding to a transient
$\mathcal{S}_{\cdot,\cdot}$	set of states corresponding to a pair of transient and cyclicity
$\mathtt{SymbMPL}(\cdot,\cdot,\cdot)$	SMT formula w.r.t. MPL systesm
$\mathtt{SymbIMPL}(\cdot,\cdot,\cdot)$	SMT formula w.r.t. IMPL systems
$\mathsf{tr}_{\min}(\cdot)$	transient in MiPL systems
$Traces(\cdot)$	set of traces of transition systems
TS, TS_f	concrete and abstract transition systems resp.
$\mathbf{x} = [x_1 \ \dots \ x_n]^\top$	current state vector w.r.t. models
$\mathbf{x}' = [x'_1 \ \dots \ x'_n]^\top$	next state vector w.r.t. models
$\mathtt{x}_1, \mathtt{x}_2, \dots$	real-valued SMT variables
X	set of initial states w.r.t. models
X_0	initial set
$X_1, X_2, X_3, \dots$	(forward) reach sets
Y_0	target set
$Y_{-1}, Y_{-2}, Y_{-3}, \dots$	backward reach sets
α	scalar
β	scalar
χ	cycle-time vector
λ	max-plus eigenvalue
$\lambda_{\min}$	min-plus eigenvalue
π	orbit of the models
π, σ	path of concrete transition systems
$\widehat{\pi}$	path of abstract transition systems

ε	zero (or neutral) element of the max-plus algebraic addition
$\varphi, \varphi_1, \varphi_2, \ldots$	LTL and TDLTL formula
ξ	zero (or neutral) element of the min-plus algebraic addition
$\top$	transpose operator
$\emptyset$	empty set
$\cap, \cup$	intersection and union operator of sets resp.
$\subset, \subseteq$	subset relation
$\supseteq$	superset relation
$\setminus$	setminus operator
$2^{\cdot}$	power-set operator
$\oplus, \otimes$	max-plus algebraic addition and multiplication resp.
$\oplus', \otimes'$	min-plus algebraic addition and multiplication resp.
$\sim$	$\{>, \geq\}$
$\longrightarrow$	transition relation
$\leftarrow$	assignment operator in algorithms
$\lvert \cdot \rvert$	number of elements of a set
$[\cdot, \cdot]$	interval in max-plus algebra
$\neg, \wedge, \vee$	Boolean operators: not, and, or
$\bigcirc, \mathsf{U}, \mathsf{R}, \Box, \Diamond$	temporal operators: next, until, release, always, eventually
$\models$	satisfaction relation
$\binom{\cdot}{\cdot}$	binomial operator
$\exists, \forall$	existential and universal quantifiers resp.

List of Abbreviations

The following abbreviations are used in this thesis:

BMC	Bounded Model Checking
CTL	Computation Tree Logic
DBM	Difference Bound Matrices
IC3	Incremental Construction of Inductive Clauses for Indubitable Correctness
IMPL	Interval Max-Plus Linear
LRA	Linear Real Arithmetic
LTL	Linear Temporal Logic
MiPL	Min Plus Linear
MPL	Max Plus Linear
PWA	Piecewise Affine
QF	Quantifier Free
RA	Reachability Analysis
RDL	Real Difference Logic
SMT	Satisfiability Modulo Theory
TD	Time-Difference
TDLTL	Time-Difference Linear Temporal Logic