Automatica 171 (2025) 111896

Contents lists available at ScienceDirect

Automatica

journal homepage: www.elsevier.com/locate/automatica

Polynomial logical zonotope: A set representation for reachability analysis of logical systems☆

Amr Alanwar [a,b,*], Frank J. Jiang [c], Karl H. Johansson [c]

[a] Technical University of Munich, Germany
[b] Constructor University, Germany
[c] KTH Royal Institute of Technology, Sweden

ARTICLE INFO

Article history:
Received 21 June 2023
Received in revised form 3 May 2024
Accepted 2 June 2024
Available online 13 September 2024

Keywords:
Polynomial logical zonotope
Logical zonotope
Reachability analysis
Security

ABSTRACT

In this paper, we introduce a set representation called polynomial logical zonotopes for performing exact and computationally efficient reachability analysis on logical systems. We prove that through this polynomial-like construction, we are able to perform all of the fundamental logical operations (XOR, NOT, XNOR, AND, NAND, OR, NOR) between sets of points exactly in a reduced space, i.e., generator space with reduced complexity. Polynomial logical zonotopes are a generalization of logical zonotopes, which are able to represent up to 2^{γ} binary vectors using only γ generators. Due to their construction, logical zonotopes are only able to support exact computations of some logical operations (XOR, NOT, XNOR), while other operations (AND, NAND, OR, NOR) result in over-approximations in the generator space. In order to perform all fundamental logical operations exactly, we formulate a generalization of logical zonotopes that is constructed by dependent generators and exponent matrices. While we are able to perform all of the logical operations exactly, this comes with a slight increase in computational complexity compared to logical zonotopes. To illustrate and showcase the computational benefits of polynomial logical zonotopes, we present the results of performing reachability analysis on two use cases: (1) safety verification of an intersection crossing protocol and (2) reachability analysis on a high-dimensional Boolean function. Moreover, to highlight the extensibility of logical zonotopes, we include an additional use case where we perform a computationally tractable exhaustive search for the key of a linear feedback shift register.

© 2024 The Author(s). Published by Elsevier Ltd. This is an open access article under the CC BY-NC license (http://creativecommons.org/licenses/by-nc/4.0/).

1. Introduction

For several decades, logical systems have been used to model complex behaviors in numerous applications. By modeling a system as a collection of logical functions operating in a binary vector space, we can design models that consist of relatively simple dynamics but still capture a complex system's behavior at a sufficient level of abstraction. Some popular approaches to modeling logical systems are finite automatons, Petri nets, and Boolean Networks (BNs). Notably, logical systems have been used to successfully model the behavior of physical systems such as gene regulatory networks (Akutsu, Miyano, & Kuhara, 1999; Shmulevich, Dougherty, Kim, & Zhang, 2002) and robotics (Roli, Manfroni, Pinciroli, & Birattari, 2011; Thunberg, Ögren, & Hu, 2011). One of the most common types of logical systems, discrete-event systems, are used for analysis in a variety of applications such as communication systems (Cassandras & Lafortune, 2008), manufacturing systems (Schuh, Zgorzelski, & Lunze, 2015), and transportation systems (Dallal, Colombo, Del Vecchio, & Lafortune, 2017; Giua & Seatzu, 2008).

1.1. Motivation

Although logical systems have been successfully used to model complex systems with relatively simple dynamics, analyzing these models can still pose computational challenges. Notably, for logical systems defined over Boolean vector spaces with n bits, analysis that needs to search or propagate the logical dynamics exhaustively may have exponential computational complexities in n; in other words, they suffer from the so-called "curse of dimensionality".

In the control community, reachability analysis is an essential form of analysis that often suffers from the curse of dimensionality. Reachability analysis allows us to formally verify the

☆ The material in this paper was partially presented at The 62nd IEEE Conference on Decision and Control (CDC), December 13–15, 2023, Marina Bay Sands, Singapore. This paper was recommended for publication in revised form by Associate Editor Paolo Frasca under the direction of Editor Sophie Tarbouriech.

* Corresponding author at: Technical University of Munich, Germany.
E-mail addresses: alanwar@tum.de (A. Alanwar), frankji@kth.se (F.J. Jiang), kallej@kth.se (K.H. Johansson).

https://doi.org/10.1016/j.automatica.2024.111896
0005-1098/© 2024 The Author(s). Published by Elsevier Ltd. This is an open access article under the CC BY-NC license (http://creativecommons.org/licenses/by-nc/4.0/).

behavior of logical systems and provide guarantees that, for example, the system will not enter into undesired states. One of the primary challenges of reachability analysis is the need to exhaustively explore the system's state space, which grows exponentially with the number of variables. To avoid exponential computational complexity, many reachability analysis algorithms leverage representations such as Binary Decision Diagram (BDD). However, due to drawbacks that we will discuss in the next section, we propose a new family of representations we call logical zonotopes.

We are inspired by the role zonotopes and polynomial zonotopes defined over real vector spaces already play in the reachability analysis of dynamical systems (Althoff, 2010; Girard, 2005; Kochdumper & Althoff, 2020). Classical zonotopes are constructed by taking the Minkowski sum of a real vector center and a combination of real vector generators. Through this construction, a set of infinite real vectors can be represented by a finite number of generators. Then, by leveraging the fact that the Minkowski sum of two classical zonotopes can be computed through the concatenation of their generators, researchers have formulated computationally efficient approaches to reachability analysis (Althoff, 2010). In binary vector spaces, logical zonotopes are able to represent up to 2^{γ} binary vectors using only γ generators. In previous work, we showed that we could apply logical operations to two logical zonotopes and compute either exact or over-approximated solutions (Alanwar, Jiang, Amin, & Johansson, 2023). In this paper, we address cases where we need only exact solutions. We will show that by adding polynomial terms into the construction of logical zonotopes, we formulate polynomial logical zonotopes that support exact computations of all fundamental logical operations (XOR, NOT, XNOR, AND, NAND, OR, NOR).

1.2. Related work

As we mentioned in the previous section, many reachability analysis algorithms avoid exponential computational complexity by leveraging BDDs. Given a proper variable ordering, BDDs can evaluate Boolean functions with linear complexity in the number of variables (Hu, 1996). Due to this benefit, BDDs are widely used for verifying real-world hardware systems (Bingham, 2015) and discrete event systems (Byrod, Lennartson, Vahidi, & Akesson, 2006). While BDDs play a crucial role in verification, they have well-known drawbacks, such as requiring an externally supplied variable ordering, since, in many applications, automatically finding an optimal variable ordering is an NP-complete problem (Byrod et al., 2006; Cabodi, Camurati, Lavagno, & Quer, 1997). Due to these drawbacks, BDDs are difficult to apply to a general class of logical systems.

Outside of BDDs, there are also approaches to reachability analysis for logical systems modeled as BNs, or Boolean Control Networks (BCNs) for systems with control inputs, that rely on the semi-tensor product (Li & Tang, 2017). However, these approaches are point-wise and work with an explicit representation of sets where all set members are explicitly enumerated, leading to reachability analysis with computational complexities that are exponential in the dimension of the system's state space. Furthermore, the structure matrix used in semi-tensor product-based approaches also grows exponentially in size with respect to the number of states and inputs, making it even more challenging to apply to high-dimensional logical systems (Leifeld, Zhang, & Zhang, 2019).

Additionally, there is also a body of work proposing new representations with similar constructions to classical zonotopes. For example, researchers propose constrained zonotopes (Scott, Raimondo, Marseglia, & Braatz, 2016), which enable richer analysis on systems defined over real vector spaces. However, since this representation is defined over real vector spaces, it is not directly applicable to logical systems. To generalize representations with zonotope-like constructions, Combastel proposed a formulation for functional sets with typed symbols (Combastel, 2022). Under Combastel's formulation, logical zonotopes are functional sets with Boolean symbols. In this work, we overview the use of polynomial logical zonotopes for the analysis of logical systems and compare their computational performance with logical zonotopes, BCNs, and BDDs in three different use cases.

1.3. Contributions

The main contribution of this work is the detailed introduction of polynomial logical zonotopes and their use in reachability analysis. We are able to perform all logical operations between sets of points in a reduced space, i.e., the generator space of polynomial logical zonotopes. Extending our previous work (Alanwar et al., 2023), in this paper, we present the formulation of polynomial logical zonotopes and discuss the trade-off of using them instead of their simpler representation: logical zonotopes. Furthermore, in computations where a polynomial logical zonotope appears repeatedly, each instance is treated independently within the framework of Minkowski logical operations. This occurrence, commonly referred to in the literature as the "dependency problem", serves as a motivation for us to address this challenge. Inspired by the work in Combastel and Zolghadri (2020), Kochdumper and Althoff (2020), we provide an approach to resolve the dependency problem and facilitate exact reachability analysis. Explicitly, the contributions of this work are summarized by the following contributions.

(1) We present our formulation of polynomial logical zonotopes as a generalization of logical zonotopes.
(2) We propose performing the logical operations between sets of points over a reduced space, i.e., the generator space of polynomial logical zonotopes in exact reachability analysis.
(3) We illustrate the computational benefits of polynomial logical zonotopes and logical zonotopes in three different applications.

To recreate our results, readers can use our publicly available logical zonotope library, which contains code for working with both logical zonotopes and polynomial logical zonotopes.[1]

1.4. Organization

The remainder of the paper is organized as follows. In Section 2, we introduce the notation and preliminary definitions we will use throughout this work. In Section 3, we formulate polynomial logical zonotopes and detail the different operations they support. Section 4 compares polynomial logical zonotopes with logical zonotopes and discusses the trade-off between accuracy and computational complexity. In Section 5, we evaluate the application of polynomial logical zonotopes for verifying intersection-crossing protocols, performing reachability analysis on a high-dimensional Boolean function, and the key discovery of a linear-feedback shift register (LFSR). Finally, in Section 6, we conclude the work with a discussion about the potential of both representations and future work.

2. Problem statement and preliminaries

In this section, we introduce details about the notation used throughout this work, the problem statement, and the preliminary definitions for logical systems, and reachability analysis.

[1] https://github.com/aalanwar/Logical-Zonotope

2.1. Notation

The set of natural and real numbers are denoted by $\mathbb{N}$ and $\mathbb{R}$, respectively. We denote the binary set $\{0, 1\}$ by $\mathbb{B}$. The XOR, NOT, OR, and AND operations are denoted by $\oplus, \neg, \vee$, and $\wedge$, respectively. Throughout the rest of the work, with a slight abuse of notation, we omit the $\wedge$ from $a \wedge b$ and write ab instead. The NAND, NOR, and XNOR are denoted by $\barwedge, \barvee$, and $\odot$, respectively. Later, we use the same notation for both the classical and Minkowski logical operators, as it will be clear when the operation is taken between sets or individual vectors. Like the classical AND operator, we will also omit the Minkowski AND to simplify the presentation. The $\prod_{k=1}^{p} \alpha_k$ denotes the ANDing of $\alpha_k, \forall k = 1, \ldots, p$. We use the notation $\overline{\oplus}$ for the exact XOR between sets, and similarly, for other exact logical operations, we add a bar on top of the notation. Furthermore, $\mathbb{B}^{n\times m}$ denotes the binary $n \times m$ set. Matrices are denoted by uppercase letters, e.g., $G \in \mathbb{B}^{n\times k}$, and sets by uppercase calligraphic letters, e.g., $\mathcal{Z} \subset \mathbb{B}^n$. Vectors and scalars are denoted by lowercase letters, e.g., $b \in \mathbb{B}^n$. The identity matrix of size $n \times n$ is denoted I_n. We denote the Kronecker product by $\otimes$. $x \in \mathbb{B}^n$ is an $n \times 1$ binary vector. Given a matrix $A \in \mathbb{B}^{n\times m}$, $A_{(i,\cdot)}$ represents the ith row, $A_{(\cdot,j)}$ the jth column, and $A_{(i,j)}$ the jth entry of row i. Given a discrete set $\mathcal{H} \in \{\cdot\}^n$, $|\mathcal{H}| = n$ denotes the cardinality of the set. The $\max(v_1, v_2)$ of two vectors is a vector while taking the maximum value row-wise. A matrix of ones (resp. zeros) with a size of $n \times m$ is denoted by $1_{n\times m}$ (resp. $0_{n\times m}$).

2.2. Problem statement

For this work, we consider a system with a logical function $f : \mathbb{B}^{n_x} \times \mathbb{B}^{n_u} \to \mathbb{B}^{n_x}$:

$$x(k+1) = f\big(x(k), u(k)\big) \tag{1}$$

where $x(k) \in \mathbb{B}^{n_x}$ is the state and $u(k) \in \mathbb{B}^{n_u}$ is the control input. The logical function f can consist of any combination of $\oplus, \neg, \vee, \barwedge, \barvee, \odot$, and $\wedge$. We define the reachable sets of system (1) by the following definition.

Definition 1 (*Exact Reachable Set*)**.** Given a set of initial states $\mathcal{X}_0 \subset \mathbb{B}^{n_x}$ and a set of possible inputs $\mathcal{U}_k \subset \mathbb{B}^{n_u}$, the exact reachable set $\mathcal{R}_N$ of (1) after N steps is

$$\mathcal{R}_N = \Big\{x(N) \in \mathbb{B}^{n_x} \;\Big|\; \forall k \in \{0, \ldots, N-1\} : \\ x(k+1) = f\big(x(k), u(k)\big),\; x(0) \in \mathcal{X}_0, u(k) \in \mathcal{U}_k\Big\}.$$

We aim to compute the exact forward reachable sets of systems defined by (1) using our new set representation, polynomial logical zonotope, as a generalization of a logical zonotope.

2.3. Preliminaries

We will represent sets of states and inputs for (1) using logical zonotopes and polynomial logical zonotopes. As will be shown, polynomial logical zonotopes are constructed using a Minkowski XOR operation, which we define as follows.

Definition 2 (*Minkowski XOR*)**.** Given two sets $\mathcal{L}_1$ and $\mathcal{L}_2$ of binary vectors, the Minkowski XOR is defined between every two points in the two sets as

$$\mathcal{L}_1 \oplus \mathcal{L}_2 = \{z_1 \oplus z_2 | z_1 \in \mathcal{L}_1, z_2 \in \mathcal{L}_2\}. \tag{2}$$

Similarly, we define the Minkowski NOT, OR, and AND operations as follows.

$$\neg\mathcal{L}_1 = \{\neg z_1 | z_1 \in \mathcal{L}_1\}, \tag{3}$$

$$\mathcal{L}_1 \vee \mathcal{L}_2 = \{z_1 \vee z_2 | z_1 \in \mathcal{L}_1, z_2 \in \mathcal{L}_2\}, \tag{4}$$

$$\mathcal{L}_1\mathcal{L}_2 = \{z_1 z_2 | z_1 \in \mathcal{L}_1, z_2 \in \mathcal{L}_2\}. \tag{5}$$

Then, to contextualize the introduction of polynomial logical zonotopes, we introduce a definition of the previously developed logical zonotope set representation. Inspired by the classical zonotopic set representation, which is defined in real vector space (Kühn, 1998), in previous work (Alanwar et al., 2023), we proposed logical zonotopes as a computationally efficient set representation for binary vectors. A logical zonotope is defined as follows.

Definition 3 (*Logical Zonotope (Alanwar et al., 2023)*)**.** Given a point $c \in \mathbb{B}^n$ and $\gamma \in \mathbb{N}$ generator vectors in a generator matrix $G=[g_1, \ldots, g_\gamma] \in \mathbb{B}^{n\times\gamma}$, a logical zonotope is defined as

$$\mathcal{L} = \left\{x \in \mathbb{B}^n \;\middle|\; x = c \oplus \bigoplus_{i=1}^{\gamma} g_i\beta_i,\; \beta \in \{0, 1\}^\gamma\right\}.$$

We use the shorthand notation $\mathcal{L} = \langle c, G\rangle$ for a logical zonotope.

Now, starting in the next section, we will extend the formulation of logical zonotopes to develop polynomial logical zonotopes and discuss the implications of the new representation.

3. Polynomial logical zonotopes

In this section, we present the formulation of polynomial logical zonotopes. Logical zonotopes, a special case of polynomial logical zonotopes, can only support exact computations for operators XOR, NOT, and XNOR and over-approximated computations for AND, NAND, OR, and NOR in the generator space. Since in certain applications, one might need to perform an exact analysis instead of an over-approximated one, we extend the formulation of logical zonotopes with additional constructions that enable exact computations for all of the fundamental logical operations in the generator space. In the next subsections, we introduce this new construction and how to compute the fundamental logical operations exactly.

3.1. Set representation

The polynomial logical zonotope is defined as follows.

Definition 4 (*Polynomial Logical Zonotope*)**.** Given a point $c \in \mathbb{B}^n$, a dependent generator matrix $G = \big[g_1, \ldots, g_h\big] \in \mathbb{B}^{n\times h}$, identifier $id \in \mathbb{N}^{1\times p}$ for identifying the dependent factors $\alpha_1, \ldots, \alpha_p$, and an exponent matrix $E \in \mathbb{B}^{p\times h}$, a polynomial logical zonotope is defined as

$$\mathcal{P} = \left\{x \in \mathbb{B}^n \;\middle|\; x = c \oplus \bigoplus_{i=1}^{h}\Big(\prod_{k=1}^{p} \alpha_k^{E(k,i)}\Big)g_i,\; \alpha \in \{0, 1\}^p\right\}.$$

We use the shorthand notation $\mathcal{P} = \langle c, G, E, id\rangle$ for a polynomial logical zonotope.

Interestingly, polynomial logical zonotopes can be viewed as functional sets with Boolean symbols (Combastel, 2022). We give the following example to illustrate the proposed set representation and its associated binary points.

Example 1. Consider the following polynomial logical zonotope

$$\bar{\mathcal{P}}_1 = \left\langle \begin{bmatrix}0\\1\\0\end{bmatrix}, \begin{bmatrix}0 & 1\\1 & 1\\1 & 1\end{bmatrix}, \begin{bmatrix}1 & 1\\0 & 1\end{bmatrix}, \begin{bmatrix}1 & 2\end{bmatrix}\right\rangle. \tag{6}$$

This is interpreted as the following set

$$\bar{\mathcal{P}}_1 = \left\{\begin{bmatrix}0\\1\\0\end{bmatrix} \oplus \begin{bmatrix}0\\1\\1\end{bmatrix}\alpha_1 \oplus \begin{bmatrix}1\\1\\1\end{bmatrix}\alpha_1\alpha_2 \;\middle|\; \alpha_1, \alpha_2 \in \{0, 1\}\right\}.$$

By considering all possible binary combination of α_1 and α_2, we get the following set of points

$$\left\{\begin{bmatrix}0\\0\\1\end{bmatrix},\begin{bmatrix}0\\1\\0\end{bmatrix},\begin{bmatrix}1\\1\\0\end{bmatrix}\right\}.$$

where the identifier vector $id=\begin{bmatrix}1 & 2\end{bmatrix}$ stores the identifier 1 for the dependent factor α_1 and the identifier 2 for the dependent factor α_2.

Next, we borrow the operator `mergeID` from Kochdumper and Althoff (2020) that is necessary in order to build a common representation of exponent matrices and fully exploit the dependencies between identical dependent factors.

Proposition 1 (*Merge ID Kochdumper & Althoff, 2020, Prob 1*). *Given two polynomial logical zonotopes, $\bar{\mathcal{P}}_1=\langle c_1,G_1,\bar{E}_1,\bar{id}_1\rangle$ and $\bar{\mathcal{P}}_2=\langle c_2,G_2,\bar{E}_2,\bar{id}_2\rangle$,* `mergeID` *returns two adjusted polynomial logical zonotopes $\mathcal{P}_1$ and $\mathcal{P}_2$ with identical identifier vectors that are equivalent to $\mathcal{P}_1$ and $\mathcal{P}_2$, and has a complexity of $\mathcal{O}(p_1p_2)$:*

$$\texttt{mergeID}(\bar{\mathcal{P}}_1,\bar{\mathcal{P}}_2)=\Big(\underbrace{\langle c_1,G_1,E_1,id\rangle}_{\mathcal{P}_1},\underbrace{\langle c_2,G_2,E_2,id\rangle}_{\mathcal{P}_2}\Big)$$

with $id=\begin{bmatrix}\bar{id}_1 & \bar{id}_{2(\mathcal{H})}\end{bmatrix},\quad \mathcal{H}=\left\{i\mid \bar{id}_{2(i)}\notin \bar{id}_1\right\},$

$$E_1=\begin{bmatrix}\bar{E}_1\\ 0_{|\mathcal{H}|\times h_1}\end{bmatrix}\in\mathbb{B}^{a\times h_1},$$

$$E_{2(i,\cdot)}=\begin{cases}\bar{E}_{2(j,\cdot)}, & \text{if } \exists j\ id_{(i)}=\bar{id}_{2(j)}\\ 0_{1\times h_2}, & \text{otherwise}\end{cases}\qquad i=1\dots a,$$

where $a=|\mathcal{H}|+p_1$.

Next, we show an example on the operator `mergeID`.

Example 2. Consider the following polynomial logical zonotope

$$\bar{\mathcal{P}}_2=\left\langle\begin{bmatrix}1\\0\\0\end{bmatrix},\begin{bmatrix}1&0\\0&1\\1&0\end{bmatrix},\begin{bmatrix}0&1\\1&1\end{bmatrix},\begin{bmatrix}1&3\end{bmatrix}\right\rangle.$$

This is interpreted as the following set

$$\bar{\mathcal{P}}_2=\left\{\begin{bmatrix}1\\0\\0\end{bmatrix}\oplus\begin{bmatrix}1\\0\\1\end{bmatrix}\alpha_2\oplus\begin{bmatrix}0\\1\\0\end{bmatrix}\alpha_1\alpha_2\,\middle|\,\alpha_1,\alpha_2\in\{0,1\}\right\}.$$

where the identifier vector $id=\begin{bmatrix}1 & 3\end{bmatrix}$ stores the identifier 1 for the dependent factor α_1 and the identifier 3 for the dependent factor α_2. If we apply the operator $\texttt{mergeID}(\bar{\mathcal{P}}_1,\bar{\mathcal{P}}_2)$ where $\bar{\mathcal{P}}_1$ is defined in (6), we get the following sets with common identifiers.

$$\mathcal{P}_1=\left\langle\begin{bmatrix}0\\1\\0\end{bmatrix},\begin{bmatrix}0&1\\1&1\\1&1\end{bmatrix},\begin{bmatrix}1&1\\0&1\\0&0\end{bmatrix},\begin{bmatrix}1&2&3\end{bmatrix}\right\rangle$$

$$\mathcal{P}_2=\left\langle\begin{bmatrix}1\\0\\0\end{bmatrix},\begin{bmatrix}1&0\\0&1\\1&0\end{bmatrix},\begin{bmatrix}0&1\\0&0\\1&1\end{bmatrix},\begin{bmatrix}1&2&3\end{bmatrix}\right\rangle$$

We consider as well an operator $\texttt{uniqueID}(p)$ for generating unique p ids out of longer repeated ids. This has a complexity of $\mathcal{O}(p)$.

Next, we provide the Minkowski logical operations using polynomial logical zonotopes.

3.2. Minkowski logical operations

We propose to perform the following Minkowski logical operations in the generator space of polynomial logical zonotopes. Polynomial logical zonotopes are closed under all logical operations.

3.2.1. Minkowski XOR ($\oplus$)

We start with the Minkowski XOR over the generator space of polynomial logical zonotope as follows.

Lemma 1. *Given two polynomial logical zonotopes $\mathcal{P}_1=\langle c_1,G_1,E_1,id_1\rangle$ and $\mathcal{P}_2=\langle c_2,G_2,E_2,id_2\rangle$, the Minkowski XOR is computed as:*

$$\mathcal{P}_1\oplus\mathcal{P}_2=\left\langle c_1\oplus c_2,\begin{bmatrix}G_1,G_2\end{bmatrix},\begin{bmatrix}E_1&0\\0&E_2\end{bmatrix},\texttt{uniqueID}(p_1+p_2)\right\rangle. \tag{7}$$

Proof. Let us denote the right-hand side of (7) by $\mathcal{P}_\oplus$. We aim to prove that $\mathcal{P}_1\oplus\mathcal{P}_2\subseteq\mathcal{P}_\oplus$ and $\mathcal{P}_\oplus\subseteq\mathcal{P}_1\oplus\mathcal{P}_2$. Choose any $z_1\in\mathcal{P}_1$ and $z_2\in\mathcal{P}_2$

$$\exists\hat{\alpha}_1: z_1=c_1\oplus\bigoplus_{i=1}^{h_1}\Big(\prod_{k=1}^{p_1}\hat{\alpha}_{1,k}^{E_{1,(k,i)}}\Big)g_{1,i}\,,$$

$$\exists\hat{\alpha}_2: z_2=c_2\oplus\bigoplus_{i=1}^{h_2}\Big(\prod_{k=1}^{p_2}\hat{\alpha}_{2,k}^{E_{2,(k,i)}}\Big)g_{2,i}\,.$$

Let $\hat{\alpha}_{\oplus,1:p_\oplus}=\begin{bmatrix}\hat{\alpha}_{1,1:p_1}\,,\hat{\alpha}_{2,1:p_2}\end{bmatrix}$ with $p_\oplus=p_1+p_2$. Given that XOR is an associative and commutative gate, we have the following:

$$z_1\oplus z_2=c_\oplus\oplus\bigoplus_{i=1}^{h_\oplus}\Big(\prod_{k=1}^{p_\oplus}\hat{\alpha}_{\oplus,k}^{E_{\oplus,(k,i)}}\Big)g_{\oplus,i}\,,$$

where $c_\oplus=c_1\oplus c_2$, $G_\oplus=\begin{bmatrix}G_1\,,\ G_2\end{bmatrix}$ with $G_\oplus=\begin{bmatrix}g_{\oplus,1},\dots,g_{\oplus,q_\oplus}\end{bmatrix}$, and $E_\oplus=\begin{bmatrix}E_1&0\\0&E_2\end{bmatrix}$. Thus, $z_1\oplus z_2\in\mathcal{P}_\oplus$ and therefore $\mathcal{P}_1\oplus\mathcal{P}_2\subseteq\mathcal{P}_\oplus$. Conversely, let $z_\oplus\in\mathcal{P}_\oplus$, then

$$\exists\hat{\alpha}_\oplus: z_\oplus=c_\oplus\oplus\bigoplus_{i=1}^{h_\oplus}\Big(\prod_{k=1}^{p_\oplus}\hat{\alpha}_{\oplus,k}^{E_{\oplus,(k,i)}}\Big)g_{\oplus,i}\,.$$

Partitioning $\hat{\alpha}_{\oplus,1:p_\oplus}=\begin{bmatrix}\hat{\alpha}_{1,1:p_1}\,,\ \hat{\alpha}_{2,1:p_2}\end{bmatrix}$, it follows that there exist $z_1\in\mathcal{P}_1$ and $z_2\in\mathcal{P}_2$ such that $z_\oplus=z_1\oplus z_2$. Therefore, $z_\oplus\in\mathcal{P}_1\oplus\mathcal{P}_2$ and $\mathcal{P}_\oplus\subseteq\mathcal{P}_1\oplus\mathcal{P}_2$.

3.2.2. Minkowski AND

Different from logical zonotopes, Minkowski AND can be performed exactly using polynomial logical zonotopes.

Lemma 2. *Given two polynomial logical zonotopes $\mathcal{P}_1=\langle c_1,G_1,E_1,id_1\rangle$ and $\mathcal{P}_2=\langle c_2,G_2,E_2,id_2\rangle$, the Minkowski AND is computed exactly and leads to $\mathcal{P}_\wedge=\langle c_\wedge,G_\wedge,E_\wedge,id_\wedge\rangle$ where:*

$$c_\wedge=c_1c_2, \tag{8}$$

$$G_\wedge=\Big[c_1g_{2,1},\dots,c_1g_{2,h_2},c_2g_{1,1},\dots,c_2g_{1,h_1},\ g_{1,1}g_{2,1},g_{1,1}g_{2,2},\dots,g_{1,h_1}g_{2,h_2}\Big], \tag{9}$$

$$E_\wedge=\Bigg[\begin{bmatrix}0_{p_1\times1}\\E_{2,(.,1)}\end{bmatrix},\dots,\begin{bmatrix}0_{p_1\times1}\\E_{2,(.,h_2)}\end{bmatrix},\begin{bmatrix}E_{1,(.,1)}\\0_{p_2\times1}\end{bmatrix},\dots,\begin{bmatrix}E_{1,(.,h_1)}\\0_{p_2\times1}\end{bmatrix},\ \begin{bmatrix}E_{1,(.,1)}\\E_{2,(.,1)}\end{bmatrix},\begin{bmatrix}E_{1,(.,1)}\\E_{2,(.,2)}\end{bmatrix},\dots,\begin{bmatrix}E_{1,(.,h_1)}\\E_{2,(.,h_2)}\end{bmatrix}\Bigg],$$

$$id_\wedge=\texttt{uniqueID}(p_1+p_2+p_1p_2). \tag{10}$$

Proof. We aim to prove that $\mathcal{P}_1\mathcal{P}_2\subseteq\mathcal{P}_\wedge$ and $\mathcal{P}_\wedge\subseteq\mathcal{P}_1\mathcal{P}_2$. Choose $z_1\in\mathcal{P}_1$ and $z_2\in\mathcal{P}_2$. Then, we have

$$\exists\hat{\alpha}_1: z_1=c_1\oplus\bigoplus_{i=1}^{h_1}\Big(\prod_{k=1}^{p_1}\hat{\alpha}_{1,k}^{E_{1,(k,i)}}\Big)g_{1,i}\,, \tag{11}$$

$$\exists\hat{\alpha}_2: z_2=c_2\oplus\bigoplus_{i=1}^{h_2}\Big(\prod_{k=1}^{p_2}\hat{\alpha}_{2,k}^{E_{2,(k,i)}}\Big)g_{2,i}\,. \tag{12}$$

ANDing (11) and (12) results in

$$\begin{aligned} z_1 z_2 =& c_1 c_2 \oplus \left(\bigoplus_{i=1}^{h_2} \left(\prod_{k=1}^{p_2} \hat{\alpha}_{2,k}^{E_{2,(k,i)}} \right) g_{2,i} c_1 \right) \\ & \oplus \left(\bigoplus_{i=1}^{h_1} \left(\prod_{k=1}^{p_1} \hat{\alpha}_{1,k}^{E_{1,(k,i)}} \right) g_{1,i} c_2 \right) \\ & \oplus \left(\bigoplus_{i=1_1,i_2=1}^{h_1,h_2} \left(\prod_{k_1=1}^{p_1} \hat{\alpha}_{1,k_1}^{E_{1,(k_1,i_1)}} \right) g_{1,i_1} \left(\prod_{k_2=1}^{p_2} \hat{\alpha}_{2,k_2}^{E_{2,(k_2,i_2)}} \right) g_{2,i_2} \right). \end{aligned} \tag{13}$$

Concatenating the factors in

$$\hat{\alpha}_\wedge = \left[\hat{\alpha}_{1,1:p_1},\ \hat{\alpha}_{2,1:p_2},\ \hat{\alpha}_{1,1}\hat{\alpha}_{2,1}, \ldots, \hat{\alpha}_{1,p_1}\hat{\alpha}_{2,p_2} \right] \tag{14}$$

results in having $E_\wedge$ and $G_\wedge$. Thus, $z_1 z_2 \in \mathcal{P}_\wedge$ and therefore $\mathcal{P}_1\mathcal{P}_2 \subseteq \mathcal{P}_\wedge$. Conversely, let $z_\wedge \in \mathcal{P}_\wedge$, then

$$\exists \hat{\alpha}_\wedge : z_\wedge = c_\wedge \oplus \bigoplus_{i=1}^{h_\wedge} \left(\prod_{k=1}^{p_\wedge} \hat{\alpha}_{\wedge,k}^{E_{\wedge,(k,i)}} \right) g_{\wedge,i}\,.$$

Partitioning $\hat{\alpha}_\wedge = \left[\hat{\alpha}_{1,1:p_1},\ \hat{\alpha}_{2,1:p_2},\ \hat{\alpha}_{1,1}\hat{\alpha}_{2,1}, \ldots, \hat{\alpha}_{1,p_1}\hat{\alpha}_{2,p_2} \right]$, it follows that there exist $z_1 \in \mathcal{P}_1$ and $z_2 \in \mathcal{P}_2$ such that $z_\wedge = z_1 z_2$. Therefore, $z_\wedge \in \mathcal{P}_1\mathcal{P}_2$ and thus $\mathcal{P}_\wedge \subseteq \mathcal{P}_1\mathcal{P}_2$.

3.2.3. Minkowski NOT ($\neg$), XNOR ($\odot$), NAND ($\barwedge$), OR ($\vee$), NOR ($\barvee$)

Based on the operations presented so far, we can compute the Minkowski NOT and XNOR as follows:

$$\neg\mathcal{P} = \left\langle c \oplus 1_{n\times 1}, G, E, id \right\rangle, \tag{15}$$

$$\mathcal{P}_1 \odot \mathcal{P}_2 = \neg(\mathcal{P}_1 \oplus \mathcal{P}_2). \tag{16}$$

Using Minkowski NOT and AND operations, we can compute the Minkowski NAND ($\barwedge$). We can also compute Minkowski OR ($\vee$), and NOR ($\barvee$) exactly using the Minkowski NAND:

$$\mathcal{P}_1 \barwedge \mathcal{P}_2 = \neg(\mathcal{P}_1\mathcal{P}_2), \tag{17}$$

$$\mathcal{P}_1 \vee \mathcal{P}_2 = (\neg\mathcal{P}_1) \barwedge (\neg\mathcal{P}_2), \tag{18}$$

$$\mathcal{P}_1 \barvee \mathcal{P}_2 = \neg(\mathcal{P}_1 \vee \mathcal{P}_2). \tag{19}$$

3.3. Exact logical operations

If a polynomial logical zonotope occurs several times in a calculation, each occurrence is taken independently during the Minkowski logical operations. This phenomenon is known in the literature as the "dependency problem". This motivates us to introduce an id for each factor inspired by Combastel and Zolghadri (2020), Kochdumper and Althoff (2020) to solve the dependency problem and provide an exact reachability analysis.

We present the following logical operations that take into account the dependency between variables. We start by the exact XOR ($\overline{\oplus}$) over the generator space of the polynomial logical zonotopes. We execute the exact logical operations after executing the operator `mergeID` on the two input polynomial logical zonotopes.

3.3.1. Exact XOR ($\overline{\oplus}$)

The exact XOR is performed as follows.

Lemma 3. *Given two polynomial logical zonotopes $\mathcal{P}_1 = \langle c_1, G_1, E_1, id\rangle$ and $\mathcal{P}_2 = \langle c_2, G_2, E_2, id\rangle$ with a common identifier vector id, the exact XOR is computed as:*

$$\mathcal{P}_1 \overline{\oplus} \mathcal{P}_2 = \left\langle c_1 \oplus c_2, \left[G_1, G_2\right], \left[E_1, E_2\right], id \right\rangle. \tag{20}$$

Proof. The proof is the same as Minkowski XOR while utilizing the merged identifiers instead of assigning unique identifiers.

In order to highlight the importance of the exact logical operation, we present the following motivating example of the dependency problem in set-based theory.

Example 3. Consider the following polynomial logical zonotope $\mathcal{P}_3 = \langle 0, 1, 1, 1\rangle$, which encapsulates the points 0 and 1. The Minkowski XOR of

$$\mathcal{P}_3 \oplus \mathcal{P}_3 = \left\langle 0, \left[1, 1\right], \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \left[1, 2\right] \right\rangle,$$

which, if evaluated, results in the following set of points $\{0, 1\}$. On the other hand, the exact XOR results in the following

$$\mathcal{P}_3 \overline{\oplus} \mathcal{P}_3 = \left\langle 0, 0, 1, 1 \right\rangle,$$

which, if evaluated, results in the following set $\{0\}$. This aligns with the expected results of XORing a variable with itself. The exact XOR comes with a solution for the dependency problem.

Next, we consider the exact ANDing after performing the operator `mergeID`.

3.3.2. Exact AND ($\overline{\wedge}$)

Lemma 4. *Given two polynomial logical zonotopes $\mathcal{P}_1 = \langle c_1, G_1, E_1, id\rangle$ and $\mathcal{P}_2 = \langle c_2, G_2, E_2, id\rangle$ with a common identifier vector id, the exact ANDing leads to $\mathcal{P}_{\bar{\wedge}} = \langle c_{\bar{\wedge}}, G_{\bar{\wedge}}, E_{\bar{\wedge}}, id\rangle$ where:*

$$c_{\bar{\wedge}} = c_1 c_2, \tag{21}$$

$$\begin{aligned} G_{\bar{\wedge}} =& \Big[c_1 g_{2,1}, \ldots, c_1 g_{2,h_2}, c_2 g_{1,1}, \ldots, c_2 g_{1,h_1}, \\ & g_{1,1} g_{2,1}, g_{1,1} g_{2,2}, \ldots, g_{1,h_1} g_{2,h_2} \Big], \end{aligned} \tag{22}$$

$$\begin{aligned} E_{\bar{\wedge}} =& \Big[E_{2,(.,1)}, \ldots, E_{2,(.,h_2)}, E_{1,(.,1)}, \ldots, E_{1,(.,h_1)}, \\ & max\big(E_{1,(.,1)}, E_{2,(.,1)}\big), max\big(E_{1,(.,1)}, E_{2,(.,2)}\big), \ldots, \\ & max\big(E_{1,(.,h_1)}, E_{2,(.,h_2)}\big) \Big], \end{aligned} \tag{23}$$

with a row-wise max.

Proof. The proof is the same as Minkowski AND while utilizing the merged identifiers instead of assigning unique identifiers.

The Minkowski NOT is the same as the exact NOT, as there is no generator involved in the operation. Given that we are able to perform the exact AND and NOT operations, we will be able to perform exact NAND and, thus, all the exact logical operations.

3.4. Containment and generators simplification

In certain scenarios, we might need to find a polynomial logical zonotope that contains at least the given binary vectors. This is especially relevant at the beginning of any set-based analysis, where one usually starts with an initial set of binary vectors from which a polynomial logical zonotope should be computed. One way to do that is as follows.

Lemma 5. *Given a list $\mathcal{S} = \{s_1, \ldots, s_p\}$ of p binary vectors in $\mathbb{B}^n$, the polynomial logical zonotope $\mathcal{P} = \langle c, G, E, id\rangle$ with $s_i \in \mathcal{P}, \forall i = \{1, \ldots, p\}$, is given by*

$$c = s_1, \tag{24}$$

$$g_{i-1} = s_i \oplus c, \ \ \forall i = \{2, \ldots, p\}, \tag{25}$$

$$G = [g_1, \ldots, g_{p-1}], \tag{26}$$

$$E = I_p, \tag{27}$$

$$id = \texttt{uniqueID}(p). \tag{28}$$

Algorithm 1 Function `simplify` to decrease the number of generators of a polynomial logical zonotope.

Input: A polynomial logical zonotope $\mathcal{P} = \langle c, G, E, id \rangle$ with large number of h dependent generators and p dependent factors
Output: A polynomial logical zonotope $\mathcal{P}_s = \langle c_s, G_s, E_s, id_s \rangle$ with $h_s \leq h$ dependent generators and $p_s \leq p$ dependent factors

$c_s = c$
$\mathcal{S} = \texttt{evaluate}(\mathcal{P})$ // list of all binary vectors in $\mathcal{P}$
$G_s = G, E_s = E, id_s = id$
for $i = 1 : h$ **do**
  $\mathcal{S}_s = \texttt{evaluate}(\mathcal{P} \setminus g_i)$ // evaluate $\mathcal{P}$ without g_i
  if $\texttt{isequal}(\mathcal{S}, \mathcal{S}_s)$ **then**
    $G_s = \texttt{remove}(G_s, g_i)$
    $(E_s, id_s) = \texttt{remove}\big((E_s, id_s), E_{(.,i)}\big)$ // remove the corresponding column from E_s and the unused identifiers
  end if
end for
$\mathcal{P}_s = \langle c_s, G_s, E_s, id_s \rangle$

Proof. By considering the truth table of all values of α, we can find that the evaluation of $\mathcal{P}$ results in $c = s_1$ at one point and $g_{i-1} \oplus c = s_i \oplus c \oplus c = s_i$, $\forall i = \{2, \ldots, p\}$, at other points.

The drawback of the technique proposed in Lemma 5 is that it results in generators with a count equal to the number of points minus one. Thus, after finding a polynomial logical zonotope containing the given binary vectors, reducing the number of generators without sacrificing any unique binary vector would be helpful. The simplification can be made with a small modification of Alanwar et al. (2023, Algorithm 1), where the dependent generators are checked for possible exclusion. This is shown in Algorithm 1 in which we simply check the generators and corresponding exponents; however, with additional computational complexity, it is possible to also check the dependent factors for exclusion. While this approach to simplifying the polynomial logical zonotopes is straightforward, it can scale exponentially in complexity with the number of generators. An important future work will be to develop a simplification approach for polynomial logical zonotopes that leverages approximations for better computational complexities.

3.5. Reachability analysis

We utilize the polynomial logical zonotope to have an exact reachable region of (1), as defined in Definition 1 and presented in Algorithm 2, as follows.

Theorem 1. *Given a logical function $f : \mathbb{B}^{n_x} \times \mathbb{B}^{n_u} \to \mathbb{B}^{n_x}$ in (1) and starting from initial polynomial logical zonotope $\mathcal{R}_0 \subset \mathbb{B}^{n_x}$, where $x(0) \in \mathcal{R}_0$, with input polynomial logical zonotope $\mathcal{U}_k \subset \mathbb{B}^{n_u}$, then the exact reachable region computed as*

$$\mathcal{R}_{k+1} = f\big(\mathcal{R}_k, \mathcal{U}_k\big) \tag{29}$$

in the generator space of polynomial logical zonotopes.

Proof. The logical function consists in general of XOR and NOT operations and any logical operations constructed from the NAND. $\forall x(k) \in \mathcal{R}_k$ and $u(k) \in \mathcal{U}_k$, we are able to compute Minkowski XOR and NOT exactly using Lemma 1 and (15), Minkowski NAND using (17). Moreover, we are able to compute the exact XOR and AND operations by Lemmas 3 and 4, respectively, and thus the exact NAND operation based on NOT and exact AND operations. Given that the NAND is a universal gate, we can perform all the remaining logical operations in the generator space.

Algorithm 2 Reachability analysis for N-steps.

Input: A logical function f, an initial set of points $\mathcal{S}_0$, a set of control input points $\mathcal{S}_{u,k}$, $\forall k = 1, \ldots, N$
Output: A reachable polynomial logical zonotope $\mathcal{R}_N$ at the N-th step

1: $\bar{\mathcal{R}}_0 = \texttt{enclosePoints}(\mathcal{S}_0)$ // Enclose the set of points with a polynomial logical zonotope using Lemma 5
2: $\mathcal{R}_0 = \texttt{simplify}(\bar{\mathcal{R}}_0)$ // Reduce the number of generators using Algorithm 1
3: $\bar{\mathcal{U}}_k = \texttt{enclosePoints}(\mathcal{S}_{u,k})$, $\forall k = 0, \ldots, N-1$
4: $\mathcal{U}_k = \texttt{simplify}(\bar{\mathcal{U}}_k)$, $\forall k = 0, \ldots, N-1$
5: **for** $k = 0 : N-1$ **do**
6: $\mathcal{R}_{k+1} = f(\mathcal{R}_k, \mathcal{U}_k)$ // Apply Minkowski and exact logical operations
7: **end for**

In Algorithm 2, we overview an algorithm based on Theorem 1 for N-step reachability analysis using polynomial logical zonotopes. First, in line 1, we use Lemma 5 to convert the initial set of points $\mathcal{S}_0$ to get an initial polynomial logical zonotope $\bar{\mathcal{R}}_0$ which is further simplified to $\mathcal{R}_0$ as explained in Section 3.4. We apply the same steps to get the input polynomial logical zonotope $\mathcal{U}_k$. Then, we iterate N times to find the Nth-step reachable set as a polynomial logical zonotope.

4. Comparison with logical zonotopes

In this section, we compare polynomial logical zonotopes with their special case, logical zonotopes. We start by reminding the reader of the application of Minkowski XOR, NOT, XNOR, AND, NAND, OR, NOR, and reachability analysis to logical zonotopes. Then, we discuss the computational complexity of both polynomial logical zonotopes and logical zonotopes to highlight the trade-off between the full and simpler representation.

4.1. Minkowski logical operations with logical zonotopes

In this section, we briefly introduce the application of each fundamental logical operation on logical zonotopes. We separate them into two groups: (1) the Minkowski operations that yield exact solutions and (2) the Minkowski operations that yield over-approximations. For proofs and more details, we refer readers to Alanwar et al. (2023).

Logical zonotopes can support exact solutions for Minkowski XOR, NOT, and XNOR operations. For logical zonotopes, $\mathcal{L}_1 = \langle c_1, G_1 \rangle$ and $\mathcal{L}_2 = \langle c_2, G_2 \rangle$, the Minkowski XOR, NOT, and XNOR are computed as follows.

$$\mathcal{L}_1 \oplus \mathcal{L}_2 = \Big\langle c_1 \oplus c_2, \big[G_1, G_2\big] \Big\rangle, \tag{30}$$

$$\neg \mathcal{L}_1 = \Big\langle c_1 \oplus 1_{n\times 1}, G_1 \Big\rangle, \tag{31}$$

$$\mathcal{L}_1 \odot \mathcal{L}_2 = \neg(\mathcal{L}_1 \oplus \mathcal{L}_2). \tag{32}$$

However, due to the limitations of their construction, logical zonotopes can only support over-approximated solutions for Minkowski AND, NAND, OR, and NOR as follows.

$$\mathcal{L}_1 \mathcal{L}_2 \subseteq \big\langle c_1 c_2, G_\wedge \big\rangle, \tag{33}$$

$$G_\wedge = \Big[\, c_1 g_{2,1}, \ldots, c_1 g_{2,\gamma_2}, c_2 g_{1,1}, \ldots, c_2 g_{1,\gamma_1}, \\ g_{1,1} g_{2,1}, g_{1,1} g_{2,2}, \ldots, g_{1,\gamma_1} g_{2,\gamma_2} \,\Big],$$

$$\mathcal{L}_1 \nwedge \mathcal{L}_2 = \neg(\mathcal{L}_1 \mathcal{L}_2), \tag{34}$$

Table 1
We list out the Minkowski logical operation computational complexity for polynomial logical zonotopes and logical zonotopes. Note, for logical zonotopes, the Minkowski AND, NAND, OR, and NOR computations yield over-approximated instead of exact solutions.

Operation	Polynomial logical zonotope	Logical zonotope (Alanwar et al., 2023)
Minkowski NOT	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Minkowski XOR, XNOR	$\mathcal{O}(n+p_1+p_2)$	$\mathcal{O}(n)$
Minkowski AND, NAND, OR, NOR	$\mathcal{O}(nh_1h_2+p_1p_2)$	$\mathcal{O}(nh_1h_2)$
Exact XOR, XNOR	$\mathcal{O}(n+p_1p_2)$	–
Exact AND, NAND, OR, NOR	$\mathcal{O}(nh_1h_2+p_1p_2)$	–

$$\mathcal{L}_1 \vee \mathcal{L}_2 = (\neg\mathcal{L}_1) \not\curlywedge (\neg\mathcal{L}_2), \tag{35}$$

$$\mathcal{L}_1 \not\curlyvee \mathcal{L}_2 = \neg(\mathcal{L}_1 \vee \mathcal{L}_2). \tag{36}$$

The term over-approximation in binary sets with $\mathcal{L}_1 \subseteq \mathcal{L}_2$ means that $\mathcal{L}_2$ contains at least all the binary vectors contained in $\mathcal{L}_1$.

4.2. Reachability analysis with logical zonotopes

While sometimes we need to compute these operations exactly, there are other cases where over-approximating the solutions to these operations is acceptable. For example, when computing the reachable sets of a logical system to check if the system will reach unsafe sets, it is still possible to formulate safety guarantees based on over-approximated reachable sets. In these cases, the simpler logical zonotopes may be a better choice than polynomial logical zonotopes, as some operations have a greater computational complexity when applied to polynomial logical zonotopes. The following theorem, which is presented and proven in our previous work (Alanwar et al., 2023), summarizes the reachability analysis of logical zonotopes.

Theorem 2. *Given a logical function $f : \mathbb{B}^{n_x} \times \mathbb{B}^{n_u} \to \mathbb{B}^{n_x}$ in (1) and starting from initial logical zonotope $\mathcal{R}_0 \subset \mathbb{B}^{n_x}$, where $x(0) \in \mathcal{R}_0$, with input logical zonotope $\mathcal{U}_k \subset \mathbb{B}^{n_u}$, then the reachable region $\hat{\mathcal{R}}_{k+1}$ over-approximates the exact reachable set $\mathcal{R}_{k+1}$. $\hat{\mathcal{R}}_{k+1}$ is computed in the generator space of logical zonotopes.*

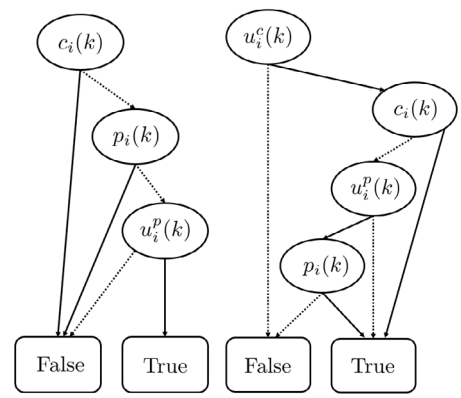
$$\hat{\mathcal{R}}_{k+1} = f\left(\hat{\mathcal{R}}_k, \mathcal{U}_k\right) \supseteq \mathcal{R}_{k+1}. \tag{37}$$

In the next section, we will discuss the computational trade-offs between polynomial logical and logical zonotopes in more detail.

4.3. Computational complexity trade-offs

To compare the computational complexity of the presented logical operations, let two polynomial logical zonotopes be defined as $\mathcal{P}_1 = \langle c_1, G_1, E_1, id_1\rangle$ and $\mathcal{P}_2 = \langle c_2, G_2, E_2, id_2\rangle$, where $c_1, c_2 \in \mathbb{B}^n$, $G_1 \in \mathbb{B}^{n\times h_1}$, $E_1 \in \mathbb{B}^{p_1\times h_1}$, $G_2 \in \mathbb{B}^{n\times h_2}$, and $E_2 \in \mathbb{B}^{p_2\times h_2}$. Then, let two logical zonotopes be defined as $\mathcal{L}_1 = \langle c_1, G_1\rangle$ and $L_2 = \langle c_2, G_2\rangle$.

We start with the computational complexity of the Minkowski logical operations. The computational complexity of NOT scales linearly with the dimension of the binary space, or $\mathcal{O}(n)$ for both logical and polynomial logical zonotopes. For applying XOR and XNOR to logical zonotopes $\mathcal{L}_1$ and $\mathcal{L}_2$, the complexity is also linear in the dimension of the binary space. However, for applying XOR and XNOR to polynomial logical zonotopes, in addition to scaling linearly with the dimension of the binary space, the operations also scale linearly with the sum of the number of dependent factors of $\mathcal{P}_1$ and $\mathcal{P}_2$ due to the `uniqueID` operation in the XOR, giving both operations a complexity of $\mathcal{O}(n+p_1+p_2)$. Similarly, for the AND, NAND, OR, NOR operations, the application of the operations to $\mathcal{L}_1$ and $\mathcal{L}_2$ has a complexity dominated by the AND operation of $\mathcal{O}(nh_1h_2)$, while the application of the operations to $\mathcal{P}_1$ and $\mathcal{P}_2$ has a complexity of $\mathcal{O}(nh_1h_2 + p_1p_1)$ due to the additional `uniqueID` operation. For the exact XOR, XNOR, AND, NAND, OR, and NOR operations, the complexity is determined using the same arguments as the Minkowski ones, however, with the complexity of the `mergeID` operation, which has a complexity of $\mathcal{O}(p_1p_2)$, instead of the `uniqueID` operations. The exact logical operations cannot be applied to $\mathcal{L}_1$ and $\mathcal{L}_2$ due to the absence of identifiers.



(a) $p_i(k+1)$ (b) $c_i(k+1)$

Fig. 1. Reduced BDDs for the intersection crossing example.

Currently, one computational challenge when using polynomial logical zonotopes is handling simplifications in cases where the number of generators grows quickly. The simplify algorithm outlined in Algorithm 1 depends heavily on the `evaluate` function, which builds a list of all the binary vectors contained within a polynomial logical zonotope. Simple implementations of `evaluate` can have complexities that grow exponentially with the number of dependent factors in the polynomial logical zonotope. As we show in the following section, there are many cases and applications that do not prohibitively suffer from this computational challenge. However, for the general application of polynomial logical zonotopes, a simplification algorithm with low computational complexity is an important future work.

In summary, we can see that the increase in complexity for applying logical operators primarily stems from the management of dependent factors. As shown in Table 1, when computing the Minkowski logical operations XOR, XNOR, AND, NAND, OR, and NOR for polynomial logical zonotopes, there is an increase in computational complexity. Moreover, when exact XOR, XNOR, AND, NAND, OR, NOR operations are necessary, polynomial logical zonotopes need to be used. However, in cases where over-approximations for AND, NAND, OR, NOR can be tolerated, then logical zonotopes without the exponent matrices and identifiers can be used to lower computational complexity. In the next section, we illustrate some of these computational trade-offs between logical zonotope and polynomial logical zonotope-based reachability analysis and search algorithms in our case studies.

5. Case studies

To illustrate the use of operating over the generators' space of polynomial logical zonotopes and logical zonotopes, we present three different use cases. We first formulate an intersection crossing problem, where we compare the computational complexity of BDDs, BCN-based semi-tensor products, and logical zonotopes when verifying the safety of four vehicle's intersection crossing protocol. Then, we demonstrate the use of polynomial logical zonotopes for conducting reachability analysis on a high-dimensional Boolean function. In addition to the two reachability analysis use cases, we also include a use case showing how logical zonotopes can drastically improve the complexity of exhaustively searching for the key of an LFSR. All of the experiments are performed on a processor 11th Generation Intel(R) Core(TM) i7-1185G7 with 16.0 GB RAM.

To compare with BCN-based semi-tensor product approaches, we use the classical definition for semi-tensor products (Cheng, Qi, & Xue, 2007). Explicitly, given two matrices $M \in \mathbb{B}^{m\times n}$ and $N \in \mathbb{B}^{p\times q}$, the semi-tensor product, denoted by $\ltimes$, is computed as:

$$M \ltimes N = (M \otimes I_{s_1})(N \otimes I_{s_2}),$$

with s as the least common multiple of n and p, $s_1 = s/n$, and $s_2 = s/p$. Note that we can apply a semi-tensor product to logical zonotopes as explained in Alanwar et al. (2023).

5.1. Safety verification of an intersection crossing protocol

We extend the provided example in Alanwar et al. (2023) to compare against polynomial logical zonotopes. More specifically, we consider an intersection where four vehicles need to pass through the intersection, while avoiding collision. For comparison, we encode their respective crossing protocols as logical functions and verify the safety of their protocols through reachability analysis using BDDs, a BCN semi-tensor product-based approach, logical zonotopes, and polynomial logical zonotopes. We denote whether vehicle i is passing the intersection or not at time k by $p_i(k)$. Then, we denote whether vehicle i came first or not at time k by $c_i(k)$. The control inputs $u_i^p(k)$ and $u_i^c(k)$ denote the decision of vehicle i to pass or to come first at time k, respectively.

For each vehicle $i = 1, \dots, 4$, the intersection passing protocol is represented as follows.

$$p_i(k+1) = u_i^p(k)\neg p_i(k)\neg c_i(k). \tag{38}$$

Then, the logic behind coming first for each vehicle $i = 1, \dots, 4$ is written as follows.

$$c_i(k+1) = \neg p_i(k+1)(u_i^c(k) \vee (\neg p_i(k)p_i(k+1))). \tag{39}$$

To perform reachability analysis, we initialize the crossing problem with the following conditions: $p_1(0) = 1$, $p_2(0) \in \{0, 1\}$, $p_3(0) = 0$, $p_4(0) \in \{0, 1\}$, $c_1(0) = 1$, $c_2(0) \in \{0, 1\}$, $c_3(0) = 0$, $c_4(0) \in \{0, 1\}$. To verify the passing protocol is always safe, under any decision made by each vehicle, we perform reachability analysis under the following uncertain control inputs: $u_1^p(k) \in \{0, 1\}$, $u_2^p(k) = 0$, $u_3^p(k) \in \{0, 1\}$, $u_4^p(k) = 0$, $u_1^c(k) \in \{0, 1\}$, $u_2^c(k) \in \{0, 1\}$, $u_3^c(k) \in \{0, 1\}$, and $u_4^c(k) \in \{0, 1\}$, $k = 0, \dots, N-1$.

We construct BDDs for each formula and execute the reduced form of the BDDs with uncertainty which is illustrated in Fig. 1. For the semi-tensor product-based approach with BCNs, we write state $x(k)=(\ltimes_{i=1}^4 p_i(k)) \ltimes (\ltimes_{i=1}^4 c_i(k))$. We write input $u(k) = (\ltimes_{i=1}^4 u_i^p(k)) \ltimes (\ltimes_{i=1}^4 u_i^c(k))$. The structure matrix L, which encodes (38)–(39), is a $2^8 \times 2^{16}$ matrix where 8 is the number of the states and 16 is the number of states and inputs. We perform reachability analysis for the BCN using $x(k+1) = L \ltimes u(k) \ltimes x(k)$ for all possible combinations. For reachability analysis with logical zonotopes and polynomial logical zonotopes, we represent each variable in (38)–(39) with a logical zonotope and polynomial logical zonotopes. We first compute the initial zonotope $\hat{\mathcal{R}}_0$ using Lemma 5 which contains the initial and certain states. Then, using Theorems 1 and 2, we compute the next reachable sets as polynomial logical zonotopes and logical zonotopes. The execution time of the three approaches is presented in seconds in Table 2. We note that reachability analysis using logical zonotopes provides the best run-time compared to other techniques. Moreover, as the reachability analysis's time horizon increases, its run-time with logical zonotopes increases slower than the other two methods. The polynomial logical zonotopes provide exact reachability compared to logical zonotopes at the cost of slightly worse execution. The term $p_i(k+1)$ appears twice in (39), which requires the exact logical operations to take care of the dependency between the terms for all iterations. On the other hand, we did not find a way to carry the dependency from one iteration k to the next one $k+1$ using BDD and BCN. This is the reason for having 13 points for polynomial logical zonotopes in comparison to 14 for BDD and BCN with $N = 5$. Notably, the impact of this dependency issue did not manifest in the subsequent steps of our example for BDD and BCN.

Table 2
Execution Time (seconds) and number of points in each set (size) for verifying an intersection crossing protocol.

Steps N	Zonotope		Poly. Zonotope		BDD		BCN	
	Time	Size	Time	Size	Time	Size	Time	Size
5	0.05	16	0.15	13	1.17	14	3.40	14
10	0.06	16	0.18	14	3.32	14	7.75	14
50	0.15	16	0.25	14	19.87	14	48.40	14
100	0.26	16	0.45	14	39.78	14	104.91	14
1000	1.84	16	2.84	14	406.60	14	1142.10	14

Table 3
Execution Time (seconds) for reachability analysis of a high-dimensional Boolean function.

Steps N	Zonotope		Poly. Zonotope		BDD	
	Time	Size	Time	Size	Time	Size
2	0.04	768	0.05	211	0.34	211
3	0.05	896	0.06	580	1.86×10^5	580
4	0.06	896	0.07	580	2.44×10^6 [a]	–
5	0.07	896	0.56	580	$> 10^6$ [a]	–

[a] Estimated execution times.

5.2. Reachability analysis on a high-dimensional boolean function

We consider the following Boolean functions with $B_i \in \mathbb{B}^{10}$ and $U_i \in \mathbb{B}^{10}$, $i = 1, 2, 3$.

$$B_1(k+1) = U_1(k) \vee (B_2(k) \odot B_1(k)), \tag{40}$$

$$B_2(k+1) = B_2(k) \odot (B_1(k) \wedge U_2(k)), \tag{41}$$

$$B_3(k+1) = B_3(k) \nwarrow (U_2(k) \odot U_3(k)). \tag{42}$$

For our reachability analysis, we initially assign sets of two possible values to $B_1(0), B_2(0)$, and $B_3(0)$. Then, we compare the execution time of reachability analysis starting from this initial condition using BDDs and logical zonotopes. We do not compare with the semi-tensor product-based approach in this example since the size structure matrix is intractable for high-dimensional systems. For the supplied variable ordering, the reachability analysis with $N = 4$ and $N = 5$ using BDDs was not completed in a reasonable amount of time, so we instead used the average

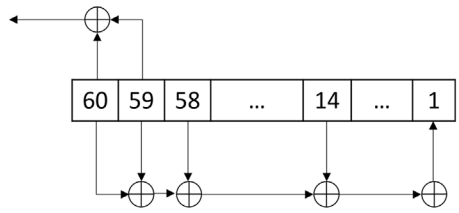


Fig. 2. LFSR A.

execution time for one iteration and multiplied that time to get the total time for the reachability analysis. The results are shown in Table 3. The logical zonotopes provide a huge over-approximation in high-dimensional systems. On the other hand, polynomial logical zonotopes provide exact reachability analysis with a low execution time. Conducting reachability analysis over a high number of steps necessitates an effective reduction function to manage the number of generators, thereby improving execution time. Developing such a function will be addressed in future work.

It is worth emphasizing that the execution times listed for polynomial logical zonotopes and logical zonotopes in Tables 2 and 3 do not include the time spent calculating the set size. To determine the size of each set, we convert them into discrete points by considering all possible combinations of the parameters α and β.

5.3. Exhaustive search for the key of an LFSR

In this use case, we revisit the case presented in Alanwar et al. (2023) and showcase a practical application where logical zonotopes can be leveraged to decrease computational complexity while maintaining exact results. In particular, we use logical zonotopes to reduce the search space when looking for the key of an LFSR.

LFSRs are used intensively in many stream ciphers in order to generate pseudo-random longer keys from the input key. For simplicity we consider 60-bits LFSR A initialized with the input key K_A with length l_k. The operations on the bit level are shown in Fig. 2, where

$$A[1] = A[60] \oplus A[59] \oplus A[58] \oplus A[14],$$
$$\text{output} = A[60] \oplus A[59].$$

Each bit i of the output of the LFSR is XORed with the message $m_A[i]$ to obtain one bit of the ciphertext $c_A[i]$.

Now consider that we aim to obtain the input key K_A using exhaustive search by trying out 2^{l_k} key values that can generate the cipher c_A from m_A with worst-case complexity $\mathcal{O}(2^{l_k})$ where $l_k = 60$ is the key length. Instead, we propose to use logical zonotopes in Algorithm 3 to decrease the complexity of the search algorithm. Given that the XOR is exact already using logical zonotopes, we did not need to move forward with polynomial logical zonotopes. We start by defining a logical zonotope $\mathcal{L}_B$, which contains 0 and 1 in line 1. Initially, we assign a logical zonotope to each bit of LFSR A in line 4 except the first two bits. Then, we set the first two bits of LFSR A to one of the 2^2 options of comb list in line 7. Then, we call the LFSR with the assigned key bits to get a list of logical zonotopes $\mathcal{G}_A$ with misuse of notations. The pseudo-random output of logical zonotopes $\mathcal{G}_A$ is XORed with the message m_A to get a list of ciphertext logical zonotopes $\mathcal{C}_A$. If any cipher of the list c_A is not included in the corresponding logical zonotope $\mathcal{C}_A$, then the assigned two digits in line 7 are wrong, and we do not need to continue finding values for the remaining bits of LFSR A. We note that the `contains` function in lines 10 and 17 is implemented in the points domain by converting the logical zonotope into points and checking the containment. After finding the correct two bits with $c_A \in \mathcal{C}_A$, we continue by assigning a zero to bit by bit in line 14. Then we generate the pseudo-random numbers $\mathcal{G}_A$ and XORed it with the m_A to get the list of cipher logical zonotopes $\mathcal{C}_A$. The ciphers' logical zonotopes $\mathcal{C}_A$ are checked to contain the list of ciphers c_A and assign $\mathcal{K}_A$ in line 18, accordingly. We measured the execution time of Algorithm 3 with different key sizes in comparison to the execution time of traditional search in Table 4. To compute the execution time of the traditional search, we multiply the number of iterations by the average execution time of a single iteration.

Table 4
Execution Time (seconds) of exhaustive key search.

Key size	Algorithm 3	Traditional search
30	1.97	1.18×10^{6} [a]
60	4.76	1.26×10^{15} [a]
120	7.95	1.46×10^{33} [a]

[a] Estimated execution times.

Algorithm 3 Exhaustive search for LFSR key using logical zonotopes.

Input: A sequence of messages m_A and its ciphertexts c_A with length l_m
Output: The used key $\mathcal{K}_A$ with length l_k in encrypting m_A

1: $\mathcal{L}_B$ =`enclosePoints`([0 1]) // enclose the points 0 and 1 by a logical zonotope
2: comb = {00, 01, 10, 11}
3: **for** $i = 3 : l_k$ **do**
4: $\quad \mathcal{K}_A[i] = \mathcal{L}_B$ // assign the logical zonotope $\mathcal{L}_B$ to the key bits
5: **end for**
6: **for** $i = 1 : 4$ **do**
7: $\quad \mathcal{K}_A[1:2] = \text{comb}[i]$
8: $\quad \mathcal{G}_A = \text{LFSR}(\mathcal{K}_A)$ // generate pseudo-random numbers from the key $\mathcal{K}_A$
9: $\quad \mathcal{C}_A = \mathcal{G}_A \oplus m_A$
10: $\quad$ **if** $\neg$`contains`$(\mathcal{C}_A, c_A)$ **then**
11: $\quad\quad$ `continue`; // continue if $c_A \notin \mathcal{C}_A$
12: $\quad$ **end if**
13: $\quad$ **for** $j = 3 : l_k$ **do**
14: $\quad\quad \mathcal{K}_A[j] = 0.$
15: $\quad\quad \mathcal{G}_A = \text{LFSR}(\mathcal{K}_A)$
16: $\quad\quad \mathcal{C}_A = \mathcal{G}_A \oplus m_A$
17: $\quad\quad$ **if** $\neg$`contains`$(\mathcal{C}_A, c_A)$ **then**
18: $\quad\quad\quad \mathcal{K}_A[i] = 1$ // assign if $c_A \notin \mathcal{C}_A$
19: $\quad\quad$ **end if**
20: $\quad$ **end for**
21: $\quad$ **if** `isequal`$(\mathcal{K}_A \oplus m_A, c_A)$ **then**
22: $\quad\quad$ `return` $\mathcal{K}_A$
23: $\quad$ **end if**
24: **end for**

6. Conclusion

In this work, we propose the use of a generalization of logical zonotopes called polynomial logical zonotopes for reachability analysis on logical systems. Polynomial logical zonotopes are constructed with additional dependent generators and exponent matrices, which allow for the exact computation of the logical operations AND, NAND, OR, and NOR. In two different use cases, we show that polynomial logical zonotopes can be used for computationally efficient reachability analysis. Then, to illustrate the extensibility of logical zonotopes, we use them to reduce

the computational complexity of exhaustive searches on logical systems. Moreover, we detail the trade-off between computational complexity and precision when using polynomial logical zonotopes or logical zonotopes in a computation. In future work, we will continue exploring the practical application of polynomial logical zonotopes and investigate new approaches for generator reduction. Furthermore, we will explore not only new use cases for polynomial logical zonotope-based reachability analysis but also other forms of analysis that benefit from the representation, such as search algorithms.

Acknowledgments

This paper has received funding from Knut and Alice Wallenberg Foundation Wallenberg Scholar Grant, the Swedish Research Council Distinguished Professor Grant 2017-01078, and the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

References

Akutsu, Tatsuya, Miyano, Satoru, & Kuhara, Satoru (1999). Identification of genetic networks from a small number of gene expression patterns under the Boolean network model. In *Biocomputing'99* (pp. 17–28). World Scientific.

Alanwar, Amr, Jiang, Frank J., Amin, Samy, & Johansson, Karl H. (2023). Logical zonotopes: A set representation for the formal verification of Boolean functions. In *62nd IEEE conference on decision and control* (pp. 60–66).

Althoff, Matthias (2010). *Reachability analysis and its application to the safety assessment of autonomous cars* (Ph.D. thesis), Technische Universität München.

Bingham, Jesse (2015). Universal boolean functional vectors. In *Formal methods in computer-aided design* (pp. 25–32). IEEE.

Byrod, Martin, Lennartson, Bengt, Vahidi, Arash, & Akesson, Knut (2006). Efficient reachability analysis on modular discrete-event systems using binary decision diagrams. In *2006 8th international workshop on discrete event systems* (pp. 288–293). IEEE.

Cabodi, Gianpiero, Camurati, Paolo, Lavagno, Luciano, & Quer, Stefano (1997). Disjunctive partitioning and partial iterative squaring: An effective approach for symbolic traversal of large circuits. In *Proceedings of the 34th annual design automation conference* (pp. 728–733).

Cassandras, Christos G., & Lafortune, Stéphane (2008). *Introduction to discrete event systems*. Springer.

Cheng, Daizhan, Qi, Hongsheng, & Xue, Ancheng (2007). A survey on semi-tensor product of matrices. *Journal of Systems Science and Complexity*, *20*(2), 304–322.

Combastel, Christophe (2022). Functional sets with typed symbols: Mixed zonotopes and polynotopes for hybrid nonlinear reachability and filtering. *Automatica*, *143*, Article 110457.

Combastel, Christophe, & Zolghadri, Ali (2020). A distributed Kalman filter with symbolic zonotopes and unique symbols provider for robust state estimation in CPS. *International Journal of Control*, *93*(11), 2596–2612.

Dallal, Eric, Colombo, Alessandro, Del Vecchio, Domitilla, & Lafortune, Stéphane (2017). Supervisory control for collision avoidance in vehicular networks using discrete event abstractions. *Discrete Event Dynamic Systems: Theory and Applications*, *27*(1), 1–44.

Girard, Antoine (2005). Reachability of uncertain linear systems using zonotopes. In Manfred Morari, & Lothar Thiele (Eds.), *Hybrid systems: Computation and control* (pp. 291–305). Berlin, Heidelberg: Springer Berlin Heidelberg.

Giua, Alessandro, & Seatzu, Carla (2008). Modeling and supervisory control of railway networks using Petri nets. *Transactions on Automation Science and Engineering*, *5*(3), 431–445.

Hu, Alan John (1996). *Techniques for efficient formal verification using binary decision diagrams*. stanford university.

Kochdumper, Niklas, & Althoff, Matthias (2020). Sparse polynomial zonotopes: A novel set representation for reachability analysis. *Transactions on Automatic Control*, *66*(9), 4043–4058.

Kühn, Wolfgang (1998). Rigorously computed orbits of dynamical systems without the wrapping effect. *Computing*, *61*(1), 47–67.

Leifeld, Thomas, Zhang, Zhihua, & Zhang, Ping (2019). Overview and comparison of approaches towards an algebraic description of discrete event systems. *Annual Reviews in Control*, *48*, 80–88.

Li, Fangfei, & Tang, Yang (2017). Robust reachability of Boolean control networks. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, *14*(3), 740–745.

Roli, Andrea, Manfroni, Mattia, Pinciroli, Carlo, & Birattari, Mauro (2011). On the design of Boolean network robots. In *European conference on the applications of evolutionary computation* (pp. 43–52).

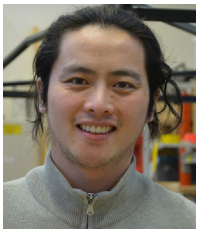
Schuh, Melanie, Zgorzelski, Markus, & Lunze, Jan (2015). Experimental evaluation of an active fault–tolerant control method. *Control Engineering Practice*, *43*, 1–11.

Scott, Joseph K., Raimondo, Davide M., Marseglia, Giuseppe Roberto, & Braatz, Richard D. (2016). Constrained zonotopes: A new tool for set-based estimation and fault detection. *Automatica*, *69*, 126–136.

Shmulevich, Ilya, Dougherty, Edward R., Kim, Seungchan, & Zhang, Wei (2002). Probabilistic Boolean networks: A rule-based uncertainty model for gene regulatory networks. *Bioinformatics*, *18*(2), 261–274.

Thunberg, Johan, Ögren, Petter, & Hu, Xiaoming (2011). A Boolean control network approach to pursuit evasion problems in polygonal environments. In *International conference on robotics and automation* (pp. 4506–4511). IEEE.

Amr Alanwar is an assistant professor at Technical University of Munich, Germany, and an adjunct assistant professor at Constructor University, Germany. He received an M.Sc. in Computer Engineering from Ain Shams University, Cairo, Egypt, in 2013 and a Ph.D. in Computer Science from the Technical University of Munich in 2020. He was a postdoctoral researcher at KTH Royal Institute of Technology. He was also a research assistant at the University of California, Los Angeles. He received the Best Demonstration Paper Award at the 16th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN/CPSWeek 2017) and was a finalist in the Qualcomm Innovation Fellowship for two years in a row.



Frank J. Jiang is a doctoral student with the School of Electrical Engineering and Computer Science at KTH Royal Institute of Technology in Sweden. He received his B.S. degree in Electrical Engineering and Computer Science from the University of California, Berkeley, in 2016 and his M.S. degree in Systems, Control, and Robotics from the KTH Royal Institute of Technology in 2019. His research interests are in formal verification, machine learning, and control, and their applications in robotics and intelligent transportation systems. He received the Best Student Paper Award at the 2020 IFAC Conference on Cyber–Physical–Human Systems (CPHS 2020).

Karl H. Johansson is the Director of Digital Futures and a professor at the School of Electrical Engineering and Computer Science at KTH Royal Institute of Technology in Sweden. He received an M.Sc. degree in Electrical Engineering and a Ph.D. in Automatic Control from Lund University. He has held visiting positions at UC Berkeley, California Institute of Technology, Nanyang Technological University, Institute of Advanced Studies Hong Kong University of Science and Technology, Norwegian University of Science and Technology, and Zhejiang University. At KTH he directed the ACCESS Linnaeus Centre 2009–2016 and the Strategic Research Area ICT TNG 2013–2020. His research interests are in networked control systems and cyber–physical systems with applications in transportation, energy, and automation networks, areas in which he has co-authored more than 800 journal and conference papers and supervised almost 100 postdocs and Ph.D. students. He has co-authored and edited 4 books, 33 book chapters, and 7 patents. He is President of the European Control Association and a member of the IFAC Council and has served on the IEEE Control Systems Society Board of Governors and the Swedish Scientific Council for Natural Sciences and Engineering Sciences. He is a Fellow of the IEEE and the Royal Swedish Academy of Engineering Sciences.