Research Article
Proving Simulink Block Diagrams Correct via Refinement

Wei Zhang,1 Quan Sun,2 Chao Wang,1 and Zhiming Liu1,3

1College of Computer and Information Science, Southwest University, Chongqing 400715, China
2College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, China
3School of Software, Northwestern Polytechnical University, Xi’an 710129, China

Correspondence should be addressed to Quan Sun; quansun@nuaa.edu.cn and Zhiming Liu; zliu@nwpu.edu.cn

Received 22 June 2022; Accepted 30 July 2022; Published 19 September 2022

Academic Editor: Chenglu Jin

Copyright © 2022 Wei Zhang et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Simulink is a well-known block diagram-based tool for modular design and multidomain simulation of Cyber-Physical Systems (CPS). However, the simulation by Simulink cannot completely cover the state space or behavior of a target system, which would not ensure the correctness of the developed block diagrams in Simulink. In this work, we present a contract-based method, which supports compositional reasoning and refinement, for proving the correctness of Simulink block diagrams with discrete-time and continuous-time dynamic behavior. We use the assume-guarantee contract as a specification language. The Simulink block diagrams are correct in the sense that if the block diagrams satisfy the formal specifications of the system being modeled. To prove the correctness of a block diagram, we first define semantics for Simulink block diagrams. We study three composition operators, i.e., serial, parallel, and algebraic loop-free feedback with multistep delays. We present a satisfaction relation between the block diagram and contract and present a refinement relation between the contracts. We prove that if the Simulink block diagram satisfies the composition contract and the composition contract refines the system specifications, the block diagram is correct relative to the system specifications. Furthermore, we demonstrate the effectiveness of our method via a real-world case study originating from the control system of a reservoir. Our method can also provide an idea to verify whether the designed CPS is planted with a logic bomb by attackers.

1. Introduction

Cyber-Physical Systems (CPS) are engineering systems where functionalities emerge from the network interaction of physical and computational processes [1]. Designing CPS correctly and efficiently is a critical challenge for computer science and industry. Model-based design (MBD) [2] provides virtual system integration and a visual approach to develop models for CPS. Bugs in the model can be identified and corrected at an early stage of the design process when no hardware is available. Such a method is considered as an effective solution to design CPS correctly.

Simulink [3] is a graphical modeling language for model-based design (MBD). Currently, Simulink greatly appeals to CPS engineers since it captures the dynamic behavior of the modeled system. A Simulink block diagram consists of blocks connected via wires. The blocks (from Simulink library, a set of predefined blocks that can assemble block diagrams of systems with drag-and-drop mouse operations) represent different parts of a system being modeled, and wires indicate the communication between the blocks. The blocks have input and output ports that receive the input signals and send the output signals. The signals are the functions of time: continuous-time or discrete-time. Hence, the Simulink block diagrams can be classified based on the time: contain only discrete-time blocks, continuous-time blocks, or a mixture of discrete-time and continuous-time blocks. Our work focuses on the Simulink block diagrams containing only discrete-time blocks and continuous-time blocks, which we call discrete-time Simulink block diagrams or continuous-time Simulink block diagrams.

Simulink supports the design, modeling, simulation, and test of CPS. The test for Simulink block diagrams is based on numerical simulation. One of the drawbacks of numerical simulation is that it does not completely cover a target system’s state space or behavior. In addition, a logic bomb [4] maliciously inserted into Simulink by attackers can persistently change the behavior. In safety-critical systems, an
error could lead to incorrect analysis results and thus result in property damage, even significant injury or death. Formal methods can rigorously prove that all possible behaviors satisfy a specific formal specification, thus ensuring correctness. By “correctness,” we mean that all possible behaviors of the Simulink block diagram satisfy the given formal specification of a system to be modeled.

A number of methods have been reported in the literature. To the best of our knowledge, some existing solutions only focus on Simulink block diagrams with discrete-time behaviors, e.g., [5–11]. A common approach for tackling continuous-time Simulink block diagrams is to discretize the continuous-time dynamical behavior [12, 13]. However, the discretization of continuous systems reduces the accuracy of the verification of continuous dynamics. The contract supports compositional reasoning and refinement, enabling hierarchical design and verification of complex systems by decomposed system-level specification into the block-level specification to provide implementations correctly. Based on this advantage, we employ contract as formal specification to specify the observable trajectory of discrete-time and continuous-time blocks and present a contract-based refinement technique for proving Simulink block diagrams’ correctness.

To prove the correctness of Simulink block diagrams, we first define formal semantics for Simulink block diagrams. We consider the blocks in the library to be units and call them elementary blocks. The elementary block expresses the time-dependent (continuous-time or discrete-time) relationships between the inputs, internal states, and outputs. Thus, we define the elementary block as a dynamic system that can model both continuous-time and discrete-time blocks. Based on this definition, we define the observable trajectory for the blocks, i.e., the evolution of the value of input-output variables over time. We formulate the wire as unilaterral connection (i.e., the relations of output and input between connected blocks) for communication. Then, to construct the Simulink block diagrams, we define three basic composition operators, namely, serial, parallel, and algebraic loop-free feedback composition. Moreover, we study the algebraic loop-free feedback composition containing multistep delays. Similar works [13, 14] only considered the ones with unit delay.

We then present a contract-based refinement technique to prove the correctness of the Simulink block diagrams, as shown in Figure 1. Our purpose is to prove that the block diagram satisfies the system specification. To this end, we introduce a mid-level called composition contract (i.e., the composition of contracts corresponding to the blocks that construct the Simulink block diagrams) between low-level block diagrams and high-level system specifications for composition. The approach is divided into two stages. Firstly, we define the satisfaction relation that relates block to contract and verify that the satisfaction relation is preserved by composition, i.e., if blocks satisfy their contract, respectively, their composition satisfies the contract composition. Secondly, we define the refinement relation between contracts and prove that the composition of contracts refines the system specifications. The block diagram is correct as long as we prove that the block diagram satisfies the composition contract and the composition contract refines the system specification to imply that the block diagram satisfies the system specification.

1.1. Contributions. Contributions of this paper are summarized as follows:

(i) We define the formal semantics for Simulink block diagrams from the viewpoint of dynamic systems to precisely express the trajectory of the Simulink block diagrams with discrete-time and continuous-time

(ii) Under the semantics, we propose a contract-based refinement technique mentioned above for proving the correctness of the Simulink block diagrams with discrete-time and continuous-time blocks

(iii) We demonstrate the effectiveness of our method through a case study of the control system of a reservoir that is modelled with Simulink block diagrams

1.2. Organization. The remainder of this paper is organized as follows. Section 2 reviews the related works. Section 3 introduces the notations and notions used in our work. Section 4 presents the semantics of the Simulink block diagrams. Section 5 proposes the contract-based refinement method for proving the correctness of Simulink block diagrams. Section 6 demonstrates the effectiveness of our method with a case study. Section 7 concludes the paper and proposes future works.

2. Related Works

In recent years, there are a range of methods to analyze and verify Simulink.

There exist some works that translate Simulink into other formal modeling languages. For example, Tripakis et al. [5] translated the Simulink block diagrams to a synchronous dataflow language, Lustre. Since the Lustre has a discrete-time semantics, the work [5] only handled discrete-time Simulink block diagrams. In [6], Cavalcanti et al. presented a semantics for discrete-time Simulink blocks diagrams called Circus. The work [6] was based on existing tools that generate CSP and Z specifications from discrete-time block diagrams. It only translated discrete-time Simulink blocks diagrams, Simulink block diagrams with continuous-time were not considered.

Chen and Dong [15, 16] presented the method to automatically transformed Simulink diagrams with discrete-time and continuous-time into Timed Interval Calculus (TIC) models. This method applied the Prototype Verification System (PVS) to validate that TIC fulfils requirements. These works were the first attempt to model Simulink block diagrams with continuous-time. The work [17] presented an operational semantics for Simulink’s simulation engine that formally defines the numerical simulation result, including discrete-time and continuous-time. Zou et al. [18]
automatically translated Simulink block diagrams into Hybrid Communicating Sequential Processes (HCSP) and showed how the translated HCSP models are verified using the Hybrid Hoare Logic Verifier. In [19], it showed how different Simulink blocks can be expressed in the synchronous language Ze'lus, which extends a language Lustre with ODEs and zero-crossing events. The main difference between the articles [15–19] and our work is intentions. Our main goal here is not to translate the Simulink block diagrams to other formalisms, nor to define the semantics of Simulink’s engine, but to directly define the trajectory of the Simulink block diagrams from the point of view of dynamic systems and provide a compositional and refinement technology to prove the correctness of Simulink block diagrams with discrete-time and continuous-time. On the other hand, we have three basic composition operators, i.e., serial, parallel, and algebraic loop-free feedback composition with multistep delays, which more facilitate the composition in construction. However, the work [18] could verify hybrid system, which is not considered in our work.

The contract-based approaches for verifying the correctness of Simulink block diagrams were also widely studied in the literature. Bostro et al. [7] showed definitions of contract and refinement using the action systems for Simulink models, while refinement provides a framework for reasoning about implementation correctness. However, this work only focuses on discrete-time Simulink block diagrams. Ye et al. [10, 11] defined a theoretical reasoning framework for Simulink block diagrams using Unifying Theories of Programming (UTP). The main idea of these papers is to translate each block or subsystem to a design, and the hierarchical connections of blocks are mapped to a variety of compositions of designs, and verify some properties. However, these papers only handled discrete-time Simulink block diagrams. In our work, we provide a compositional and refinement technology to prove the correctness of Simulink block diagrams with discrete-time and continuous-time. Dragomir et al. [13] recently presented a Refinement Calculus of Reactive Systems (RCRS) toolset for compositional formal modeling and reasoning about discrete and continuous reaction systems. RCRS is a discrete-time framework. The continuous systems can be modeled by discretizing time. However, the discretization of continuous systems reduces the accuracy of the verification of continuous dynamics. Our approach differs because we can directly represent and theoretically verify the correctness of discrete-time and continuous-time Simulink block diagrams. Moreover, we study the algebraic loop-free feedback composition with multistep delays.

3. Preliminary

In this section, we introduce some notations and notions that will be used in our work. We denote the set of natural numbers by \( \mathcal{N} \), i.e., \( \{0, 1, 2, \cdots\} \), the set of positive integers by \( \mathcal{N}^+ \), i.e., \( \{1, 2, \cdots\} \), the set of positive real numbers \( \mathbb{R}^+ \), and the set of nonnegative real numbers by \( \mathbb{R}_0^+ \). We denote the set of integers between 1 and n by \( [n] = \{1, \cdots, n\} \subset \mathcal{N}^+ \). We denote vectors by bold fonts, and their components are

![Figure 1: An overview of contract-based correctness proof methods for Simulink block diagrams, where the dashed line and \( \models \) : satisfaction; \( \vdash \) : refinement; \( \otimes \) : composition.](image-url)
4. Simulink Block Diagrams and Semantics

In this section, we present the semantics of Simulink block diagrams. We will start by giving a brief introduction to Simulink block diagrams, and we highlight the features relevant to our work. For more details, we refer the reader to [3].

4.1. Introduction of Simulink Block Diagrams. A Simulink block diagram is a graphical representation of a dynamic system. The Simulink block diagrams are composed of blocks and wires. The blocks can be either elementary blocks provided by the Simulink library or composition blocks made up of elementary blocks or other composition blocks. An example that implements the relationship between the vehicle’s power, resistance, and speed is shown in Figure 2. It consists of Constant, Subsystem, and Scope, where Constant and Scope are elementary blocks, and the composition block Subsystem comprises four elementary blocks Gain1, Subtract, Integrator, and Gain2.

In its most general form, the elementary block has Inputports and Outputports that receive the input signals and send the output signals. For some special blocks, the absence of Inputports or Outputports is also allowed. We will explain the details later. An elementary block is either stateful or stateless. We say a block is stateful if the output of this block depends on its inputs and internal states (i.e., memory). We say a block is stateless if the output depends only on its inputs. Wires transmit signals in the direction indicated by the arrow. It must transmit signals from Outputports of one block to Inputports of another block in terms of its sample times. An exception to this is that one wire can be drawn from another. This sends the original signal to two (or more) target blocks. Wire communication is instant. That is, when a block outputs the value to a wire, all blocks connected to that wire will see the new value simultaneously.

To improve the modeling capabilities of Simulink, each elementary block contains some user-tunable parameters. One of the significant parameters is sample time that indicates the rate at which the block executes in simulation. According to the sample time, the blocks are divided into two main categories: continuous and discrete blocks. The sample time comprises two parameters: sample time period $\tau$ and initial time offset $\theta$. For the continuous blocks, the sample time period $\tau = 0$. For the discrete blocks, the sample time period $\tau$ is always greater than zero and less than the simulation end time and $\theta$ less than or equal to $\tau$. Since the default value of the initial offset is 0, unless otherwise mentioned, we let the initial offset $\theta$ be 0. As an example, suppose that the time unit is seconds, let the sample time period of a block be 0.02 s, and then the block updates methods (update, derivative or output) each 0.02 s. The Simulink block diagrams can be single rate where all blocks run with the same period or multirate where blocks run on different periods. This work considers single rate discrete-time Simulink block diagrams and continuous-time Simulink block diagrams with $\tau = 0$.

There are three basic composition operators in Simulink: (i) Serial composition is the composition that the output of the source block is connected to the input of the target block. (ii) Parallel composition is that two blocks are “stacked on top of each other” without any wires between the two blocks. (iii) Feedback composition is that the output of a block connects to one of its inputs. Other forms of composition can be assembled from these three basic composition operations and wires.

4.2. The Semantics of Simulink Block Diagrams. In this subsection, we formally define the semantics for Simulink block diagrams. We focus on the semantics of elementary block, composition semantics for composition operators, and the semantics of communication between blocks.
As introduced in 4.1, an elementary block has Inputs that receive the input signals, internal state, and Outputs that send the output signals. It describes a mathematical relationship between inputs, outputs, and internal states to capture dynamic behavior. We define an elementary block as a dynamical system.

**Definition 1 (An elementary block).** An elementary block $B$ is a tuple $(x, s, y, \varphi, f)$, where

(i) $x : T \rightarrow X$ is an input signal, $x(t_0) \in \mathcal{X}$ is the initial value of the input signal

(ii) $y : T \rightarrow Y$ is an output signal, $y(t) \in \mathcal{Y}$, $y(t_0)$ is the initial value of the output signal

(iii) $s : T \rightarrow S$ is an internal state signal, $s(t) \in \mathcal{S}$, $s(t_0)$ is the initial value of the internal state signal

(iv) $\varphi$ is the transition function of internal state

(v) $f$ is the output function, i.e., $y(t) = f(x(t), s(t))$

The above definition can express both discrete-time blocks and continuous-time blocks. We say that a block is continuous if it operates on continuous-time signals. Every block must define its output function and may define initialize, update, or derivative function to realize the corresponding function. For the discrete-time blocks, we denote $\varphi = \varphi_u$, i.e., $s(t_{k+1}) = \varphi_u(x(t_k), s(t_k))$, which refers to the update function of internal state. For the continuous-time blocks, we denote $\varphi = \varphi_d$, i.e., $s(t) = \varphi_d(x(t), s(t))$, which refers to the derivative function of internal state. The input or internal state of a block can be empty, respectively. In that case, we denote the input or internal state by $x = 0$ or $s = 0$ and use a symbol “...” to denote the transition function of the internal state. If a block has no internal state, we say the block is stateless.

Some elementary blocks mentioned in our article are shown in Table 1. Three simple examples are shown below.

**Example 1 (Unidelay).** An example of a stateful discrete-time elementary block is the Unidelay block. The Unidelay expression states that the current output value of this block is equal to the value of the current internal state, and the value of the next internal state is equal to the current input value. It can be represented as $B = (x_0, s_0, y_0, s_1(t_{k+1}) = x_1(t_k), y_1(t_k) = s_0(t_k))$, where $t_k \in T$.

**Example 2 (Multistep delays).** A block with multistep delays is denoted as $B = (x, y, \varphi, f)$, where $y(t_{nk}) = s(t_{nk})$, and $s(t_{nk+1}) = x(t_{nk})$, where $n$ means that the block outputs the input of this block after $n$ sample periods, $n \in \mathcal{N}^+$, $k \in \mathcal{N}$.

**Example 3 (Integrator).** An example of a stateful continuous-time elementary block is the Integrator. The block models the relations, $s(t) = x(t)$ with $y(t) = s(t)$. The Integrator can be represented as $B = (x, y_1, y_2, s_1(t) = x_1(t), y_1(t) = s_1(t))$, where $t \in \mathcal{T}$.

We use the observable trajectory to model the evolution of the input-output variables.

**Definition 2 (The observable trajectory of the discrete-time elementary block).** Let $B = (x, y, \varphi_u, f)$ be a discrete-time elementary block. An observable trajectory $Tr$ of $B$ is a set of $(x, y) : T \rightarrow X \times Y$ of $\exists y(t_k) = f(s(t_k), x(t_k)) / s(t_{k+1}) = \varphi_u(x(t_k), s(t_k)), k \in \mathcal{N}$, where $x : T \rightarrow X \subseteq \mathcal{X}$ is the input trajectory and $y : T \rightarrow Y \subseteq \mathcal{Y}$ is the output trajectory.

**Definition 3 (The observable trajectory of the continuous-time elementary block).** Let $B = (x, y, \varphi_d, f)$ be a continuous-time elementary block. An observable trajectory $Tr$ of $B$ is a set of $(x, y) : T \rightarrow X \times Y$ of $\exists y(t) = f(s(t), x(t))$ and $s(t_k) = \varphi_d(x(t), s(t_k)))$, where $x : T \rightarrow X \subseteq \mathcal{X}$ is the input trajectory and $y : T \rightarrow Y \subseteq \mathcal{Y}$ is the output trajectory.

The wires connect some Outputs of a block to some Inputs of other block for communicating. For any blocks, $B_1 = (x_1, s_1, y_1, \varphi_1, f_1)$ and $B_2 = (x_2, s_2, y_2, \varphi_2, f_2)$, where $x_1 = (x_{1,1}, \cdots, x_{1,n})$, $y_2 = (y_{1,1}, \cdots, y_{1,m})$, $x_1 = (x_{2,1}, \cdots, x_{2,n})$, and $y_2 = (y_{2,1}, \cdots, y_{2,m})$. We model the wires between $B_1$ and $B_2$ as a relation, called unilateral connection. A unilateral connection is a set of variables pairs $(y_{1,j}, x_{2,i})$, for $j \in [m]$ and $i \in [n]$, where the former of each pair is the output variable that comes from $B_1$, the latter of each pair is the input variable that comes from $B_2$. We define the unilateral connection as follows:

**Definition 4 (Unilateral connection).** Given two blocks $B_1 = (x_1, s_1, y_1, \varphi_1, f_1)$ and $B_2 = (x_2, s_2, y_2, \varphi_2, f_2)$, we define a unilateral connection from $B_1$ to $B_2$ (and vice versa) as a relation $\rho = \{(y_{1,i}, x_{2,i}) | j \in [m], i \in [n] \}$ satisfying that:

(i) $y_{1,j}$ is the $j$-th component of $y_1$, for all $j \in [m]$

(ii) $x_{2,i}$ is the $i$-th component of $x_2$, for all $i \in [n]$

(iii) $y_{1,i}(t) = x_{2,i}(t)$, for all $t \in T$

Note that not all the blocks can be connected. Given a source block $B_1 = (x_1, s_1, y_1, \varphi_1, f_1)$ and a target block $B_2 = (x_2, s_2, y_2, \varphi_2, f_2)$, if we connect the output variables of the source block with input variables of the target block, variable names and their types (the sets of values that a variable can take) impose some constraints: First, the names of the input and output variables should not conflict. Second, the types should match and $Y_1 \subseteq X_2$, where $Y_1$ is the type of $y_1$ and $X_2$ is the type of $x_2$. Third, one output port can connect to many input ports, but an input port can connect to at most one output ports.

For readability, we use $\{\rho_1, \cdots, \rho_n\} \subseteq \rho$ to denote a set of specific unilateral connection (determined by a relation as defined in Definition 4). Note that, since $\rho = \{(y_{1,i}, x_{2,i}) | j \in [m], i \in [n] \}$
A Simulink block diagram is a composition block constructed by elementary blocks according to composition operators. We will define the semantics of three basic composition operators in the following, namely, serial, parallel, and algebraic loop-free feedback composition with multistep delays. We first consider the serial composition.

We can compose blocks $B_1$ and $B_2$ to form a serial composition when there is a unilateral connection between $B_1$ and $B_2$, and the unilateral connection satisfies the connection rules, as shown in Figure 3(a).

**Definition 5** (Serial composition). Given blocks $B_1 = (x_1, s_1, y_1, \phi_1, f_1)$ and $B_2 = (x_2, s_2, y_2, \phi_2, f_2)$, a serial composition $B_1 \circ B_2$ of $B_1$ and $B_2$ is defined as a block $(x, s, y, f, \phi)$ if there exists a unilateral connection and

\[
\begin{align*}
x &:= x_1, \\
s &:= (s_1, s_2), \\
y &:= y_2, \\
\phi &:= (\phi_1(x_1, s_1), \phi_2(x_2, s_2)), \\
f &:= (f_1 ; \rho ; f_2)(x, s) = f_2(\rho(f_1(x_1, s_1)), s_2).
\end{align*}
\]

The definition of the serial composition of elementary blocks coincides with the definition of an elementary block (i.e., Definition 1), and therefore a composition can be considered a block.

**Example 4** (Serial composition). In Figure 3(b), the stateless block Gain takes input $x_1$, computes $k \in \mathbb{R}$ times of $x_1$, and returns $y_1$ as output, where $x_1 : \mathbb{R} \mapsto \mathbb{R}$ and $y_1 : \mathbb{R} \mapsto \mathbb{R}$. 

<table>
<thead>
<tr>
<th>Library</th>
<th>Description</th>
<th>Elementary block</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td>Constant value</td>
<td>$\begin{array}{c} C \ \uparrow \end{array}$</td>
<td>$B = (0, 0, y, -y(t) = c)$</td>
</tr>
<tr>
<td>Sinks</td>
<td>Display System output</td>
<td>$\times \nearrow \begin{array}{c} x \ \downarrow \end{array}$</td>
<td>$B = (x, 0, y, -y(t) = x(t))$</td>
</tr>
<tr>
<td>Signal routing</td>
<td>Switch Conditional statement</td>
<td>$\times \nearrow \begin{array}{c} x_1 \ \uparrow \downarrow \leftarrow \rightarrow \rightarrow \ x_2 \nearrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \right)</td>
<td></td>
</tr>
</tbody>
</table>
Another stateless block Sum has the input \( x_2 = (x_{2,1}, x_{2,2}) \) and output \( y_2(t) = x_{2,1}(t) + x_{2,2}(t) \), where \( x_2 : \mathcal{R} \rightarrow \mathcal{R} \times \mathcal{R} \) and \( y_2 : \mathcal{R} \rightarrow \mathcal{R}, \forall t \in \mathcal{R} \). Let \( \rho \) be the unilateral connection between Gain and Sum. Then, we can express these blocks as follows:

\[
\begin{align*}
\text{Gain} &:= (x_1, 0, y_1) = \text{gain}(x_1(t), 0), \\
\rho &:= \{(y_1, x_{2,1}), (y_1, x_{2,2})\}, \\
\text{Sum} &:= (x_2, 0, y_2) = \text{add}(x_2(t), 0),
\end{align*}
\]

where \( y_2(t) = \text{add}(x_2(t), 0) = x_{2,1}(t) + x_{2,2}(t) \), \( y_1(t) = \text{gain}(x_1(t), 0) = k \cdot x_1(t) \), \( B_1(t) = x_{2,1}(t) \), and \( y_2(t) = x_{2,2}(t) \). According to Definition 5, we compute the output function of the serial composition as follows:

\[
\begin{align*}
\rho &= \{(y_1, x_{2,1}), (y_1, x_{2,2})\}, \\
\text{Sum} &= \{(x_2, 0, y_2) = \text{add}(x_2(t), 0)\}, \\
\text{Gain} &= \{(x_1, 0, y_1) = \text{gain}(x_1(t), 0)\}, \\
\text{Sum} &= \{(x_2, 0, y_2) = \text{add}(x_2(t), 0)\}.
\end{align*}
\]

Therefore, we have \( \text{Gain} \circ \text{Adder} = (x_1, 0, y_2, y_2(t)) = 2k \cdot x_1(t) \).

The serial composition of blocks satisfies associative laws.

**Lemma 6** (Associativity). Given blocks \( B_1, B_2, \) and \( B_3 \), we have \( B_1; B_2; B_3 = B_1; (B_2; B_3) \).

**Proof.** It is easy to verify using Definition 5. Thus, if we compose multiple blocks in serial, we can first compose two blocks, compose the result with a third one, \( \cdots \), and rearranging the brackets in the expression does not change the result as long as the block’s position remains the same. \( \square \)

Parallel composition is a particular case of composition with connection, where the unilateral connection between \( B_1 \) and \( B_2 \) is an empty set. We define parallel composition as follows:

**Definition 7** (Parallel composition). Given blocks \( B_1 = (x_1, s_1, y_1, \phi_1, f_1) \) and \( B_2 = (x_2, s_2, y_2, \phi_2, f_2) \), we define the parallel composition \( B_1 \parallel B_2 \) of \( B_1 \) and \( B_2 \) as a block \( (x, s, y, f, \phi) \),

where

\[
\begin{align*}
x &:= (x_1, x_2), \\
s &:= (s_1, s_2), \\
y &:= (y_1, y_2), \\
f &:= (f_1 \parallel f_2)((x_1, s_1), (x_2, s_2)) \\
\phi &:= \{(\phi_1 \parallel \phi_2)((x_1, s_1), (x_2, s_2))\}.
\end{align*}
\]

**Example 5** (Parallel composition). Consider the parallel composition shown in Figure 4(a). The block Divide models the relation \( y_1(t) = 1/x_1(t) \), where \( x_1 \) is the input signal and requires \( x_1(t) \neq 0 \), \( y_1 \) is the output signal, and \( t \) represents time. The Sine Wave block models the relation \( y_2(t) = x_2(t) \), where \( x_2 \) is the input variable and \( y_2 \) is the output variable. Therefore, the Divide and Sine Wave can be represented as

\[
\text{Divide} := \left(x_1, 0, y_1, y_1(t) = \frac{1}{x_1(t)}\right),
\]

\[
\text{Sine Wave} := (x_1, 0, y_2, y_2(t) = x_2(t)).
\]

Following Definition 7, we write the parallel composition as follows:

\[
\text{Divide} \parallel \text{Sine Wave} := \left(x_1, 0, (y_1, y_2), \begin{cases} \left(y_1(t) = \frac{1}{x_1(t)}\right), & y_1(t) \neq 0 \\ \left(y_2(t) = x_2(t), & y_2(t) \neq 0 \end{cases}\right)\right).
\]

Note that Definition 5 defines the case that all the Outputs of \( B_1 \) match the Inputs of \( B_2 \). However, not all Outputs of \( B_1 \) match the Inputs of \( B_2 \) and not all Outputs of \( B_2 \) are connected to all the Inputs of \( B_2 \). As an example, we consider Figure 4(b). To handle this composition, we introduce a particular elementary block \( \text{Id} \) representing its output is identical to its input. We model the wires that connect the \( y_{1,1} \) and \( x_{2,2} \) as \( \text{Id}_1 \) and \( \text{Id}_2 \), respectively. We then denote the composition by \( B = (B_1 \parallel \text{Id}_1) ; \rho ; (B_1 \parallel \text{Id}_2) \), where \( \rho = \{(y_{1,1}, x_{1,1}), (y_{1,2}, x_{2,1}), (y_{2,1}, x_{1,2}), (y_{2,2}, x_{2,2})\} \).
The parallel composition of blocks satisfies commutative law and associative law.

**Lemma 8.** Given blocks $B_1$, $B_2$ and $B_3$, we have

(i) $B_1 || B_2 = B_2 || B_1$

(ii) $(B_1 || B_2) || B_3 = B_1 || (B_2 || B_3)$

**Proof.** Straightforward from the Definition 7.

We will next define the algebraic loop-free feedback composition with multistep delays. Before defining that, we first introduce the algebraic loop feedback to aid this definition. In Simulink, an algebraic loop occurs when an input port of a direct feedthrough block is driven by the output of the same block in the same time step. The direct feedthrough means that a stateless block computes its output only depending on the input value at the current time. In mathematics, an algebraic loop can be expressed as the algebraic equation $f(x, y) = 0$, where $x$ is the input variable and $y$ is the algebraic variable. Simulink solves the algebraic equation for $y$ at each time instant.

A simple example of an algebraic loop is the feedback that represents in Figure 5(a). The feedback is a Subtraction block with input $x = (x_{1,1}, x_{1,2})$ and an output $y$. The first element $x_{1,1}$ of input is used to communicate with the environment. The second element $x_{1,2}$ is used for feedback. The output $y$ is split into two equal signals: one is for output, and another is to feed the output back into input $x_{1,2}$. Then, this loop implies that the output of the Subtraction block is an algebraic variable $y$ that is constrained to equal the first input $x_{1,1}$ minus $x_{1,2}$, i.e., $x_{1,1}(t) - x_{1,2}(t) = y(t)$. Let $x_{1,2}(t) = y(t)$ and $x_{1,1}(t) = u(t)$, we have $u(t) = y(t) = y(t)$, and then the solution of this loop is $y(t) = u(t)/2$.

However, the algebraic loop has inherent difficulties in solving: (1) while Simulink solver solves the algebraic loop, the simulation can execute slowly. (2) Some algebraic loops have no solution (an example shown in Figure 5(b), the expression of this loop is $u(t) + y(t) = y(t)$, etc. These problems lead to algebraic loops that are undesirable. To remove the algebraic loop, according to Simulink, we connect $y$ to $x_{1,2}$ by stateful blocks (such as Delay, Memory, or Integrator) in this feedback to break the algebraic loop. We refer to this kind of acyclic structure as algebraic loop-free feedback, as shown in Figure 6. The work [14] defined the feedback with Unitdelay. We will handle the case of multistep delays.

**Definition 9** (Algebraic loop-free feedback composition with multistep delays). Let blocks $B_1 = (x_1, s_1, y_1, \phi_1, f_1)$ be feedback with an algebraic loop and $B_2 = (x_2, s_2, y_2, \phi_2, f_2)$ be a multistep delays block. Let $\rho_1 = (y_2, x_{1,2})$ and $\rho_2 = (y, x_2)$ be the unilateral connections between $B_1$ and $B_2$. We denote by $B_1 \otimes \rho_1 \otimes B_2$ the algebraic loop-free feedback composition of $B_1$ and $B_2$ with multistep delays and define $B_1 \otimes \rho_1 \otimes B_2 = (x, s, y, \phi, f)$, where

\[
\begin{align*}
  x &= x_{1,1}, \\
  s &= (s_1, s_2), \\
  y &= y_1, \\
  \phi &= (\phi_1(\|\phi_2\|)(x_1, s_1), (x_2, s_2)) \\
  &= \phi_1(x_1, s_1), \phi_2(x_2, s_2), \\
  f &= \begin{cases} 
    y(t_0) & t = t_0, \\
    y(t_{nk}) & t = t_{nk}, n, k \in \mathbb{N}^+.
  \end{cases}
\end{align*}
\]

In the definition above, the calculation process starts with the multistep delays block. When $t = t_0$, let the initial state $s(t_0) = (0, s_2(t_0))$. The output’s initial value of this algebraic loop-free feedback $y(t_0) = f_1((x_{1,1}(t_0), x_{1,2}(t_0)), s(t_0))$. There exists $\rho_1$, s.t. $x_{1,2}(t_0) = \rho_1(y_2(t_0))$, when $y_2(t_0) = f_2(x_2(t_0), s_2(t_0))$. Given a delay step $n \in \mathbb{N}$, when $t = t_{nk}$, where $k \in \mathbb{N}$, the output $y(t_{nk}) = f_1((x_{1,1}(t_{nk}), x_{1,2}(t_{nk})), s(t_{nk}))$, where the internal state $s(t_{nk}) = (0, s_2(t_{nk}))$, and $x_{1,2}(t_{nk}) = \rho_1(y_2(}
there exists $\rho_2$, s.t. $x_2(t_n(k)) = \rho_2 y(t_n(k-1))$ is the input of $B_2$, and $s_2(t_n(k)) = \phi_2(x_2(t_n(k-1)), s_2(t_n(k-1)))$.

We next give an example to illustrate removing an algebraic loop by introducing Unitdelay, i.e., let the delay step $n = 1$.

**Example 6** (An algebraic loop-free feedback composition with Unitdelay). In Figure 6(a), the sum block $\text{Sum} = \{(x_{1,1}, x_{1,2}), 0, y_1, y_1(t) = x_{1,1}(t) + x_{1,2}(t)\}$ is feedback with an algebraic loop. We connect $y$ to $x_{1,2}$ in this feedback by the Unitdelay to break the algebraic loop. The Unitdelay block $B_u = (x_u, s_u, y_u, s_u(t_{k+1}) = x_u(t_k), y_u(t_k) = s_u(t_k))$, where $t_k \in T$. Let $\rho_1 = (y_u, x_{1,2})$ and $\rho_2 = (y, x_u)$, the $\rho_1$ and $\rho_2$ satisfy the connection rules, and then an algebraic loop-free feedback composition with Unitdelay is a block $B_1 \otimes B_2 = (x, s, y, \varphi, f)$, where $x = x_{1,1}, s = (0, s_u), y = y$, and the output function

$$y(t) = \begin{cases} y(t_0) & t = t_0, \\ y(t_k) & t = t_k, k \in \mathbb{N}^+. \end{cases}$$

When $t = t_0$, let $s_u(t_0)$ be the initial value of the internal state of the Unitdelay. We have $y_u(t_0) = s_u(t_0)$ and $x_{1,2}(t_0) = y_u(t_0)$. Hence, we get the initial output value of this composition $y(t_0) = x_{1,1}(t_0) + s_u(t_0)$.

When $t = t_k, k \in \mathbb{N}^+$, we first consider Unitdelay. Because of $x_u(t_k) = y(t_k), s_u(t_{k+1}) = x_u(t_k), y_u(t_k) = s_u(t_k)$, and $x_{1,2}(t_k) = y_u(t_k)$, we have $y(t_k) = x_{1,1}(t_k) + y(t_{k-1})$.

Next, we give an example about algebraic loop-free feedback composition with Integrator.

**Example 7** (An algebraic loop-free feedback composition with Integrator). An example of continuous time algebraic loop-free feedback composition with Integrator is shown in Figure 6(b). As previously mentioned, the sum block $\text{Sum} = \{(x_{1,1}, x_{1,2}), 0, y_1, y_1(t) = x_{1,1}(t) + x_{1,2}(t)\}$ is feedback with an algebraic loop. The Integrator can be represented as $B_1 = (x_I, s_I, \dot{s}_I, \dot{s}_I(t) = x_I(t), y_I(t) = s_I(t))$, where $t \in \mathbb{R}^+$. In this composition, there exist unilateral connections $\rho_1 = (y_I, x_{1,2})$ and $\rho_2 = (y, x_I)$, and the unilateral connections satisfy the connection rules, and then an algebraic loop-free feedback composition with Integrator is a block $B_1 \otimes B_2 = (x, s, y, \varphi, f)$, where $x = x_{1,1}, s = (0, s_I)$. Let $s_I(0)$ be the initial value of the internal state of the Integrator. We have $y_I(0) = s_I(0)$, and there is a $\rho_1$, s.t. $x_{1,2}(0) = y_I(0)$. Hence, we get $y(0) = x_{1,1}(0) + x_{1,2}(0) = x_{1,1}(0) + y_I(0)$.

When $t \in \mathbb{R}^+$, similarly, we first consider Integrator. Because there is a $\rho_2$, s.t. $x_I(t) = y(t)$, and $\dot{s}_I(t) = x_I(t), y_I(t)$
= s_1(t). There exists \( \rho_1 \), s.t. \( x_{1,2}(t) = y_f(t) \), we get

\[
y(t) = x_{1,1}(t) + x_{1,2}(t) = x_{1,1}(t) + \left( y_f(0) + \int_0^t x_f(\delta)d\delta \right)
\]

5. A Contract-Based Refinement Approach

The correctness refers to the trajectories of Simulink block diagrams that should satisfy the requirement specifications of the system. Since contracts are centered around trajectories, they are expressive and versatile enough to specify discrete-time and continuous-time blocks. We use the contract as a specification language to formalize and prove system requirements and specify the trajectories of blocks. In this section, we present a generic contract-based method for proving the correctness of Simulink block diagrams with discrete-time and continuous-time blocks.

Because the requirement specifications of a system are informal, we first formalize system requirement specifications into system contract specifications. We construct the Simulink block diagram according to the contract specification. We aim to prove that the Simulink block diagrams satisfy the system contract specification. To this end, the approach is divided into two steps. First, we modularly verify that Simulink block diagrams satisfy the composition contract (i.e., the composition of contracts corresponding to the blocks that construct the Simulink block diagrams). We associate a contract for every elementary block as a specification. After that, we define the satisfaction relation that relates block to contract and verify that the satisfaction relation is preserved by composition. That is, if blocks satisfy their contract, respectively, then the composition of blocks satisfies the composition of contracts. Second, we define the refinement relation between contracts and verify that the composition of contracts refines the system contract specifications. If the block diagram satisfies the composition contracts and the composition contracts refine the system specification, then the block diagram also satisfies the system specification. Hence, the Simulink block diagram is correct with respect to the system specification.

We further elaborate on the concepts and properties mentioned above in the following. We first define the contract specification for the block.

**Definition 10 (Contract).** A contract is a tuple \( C = (x, y, \phi_a, \phi_g) \), where

(i) \( x, y \) are the input vectors and output vectors, respectively

(ii) \( \phi_a \subseteq X^T \) represents the assumption for the input trajectories of the block, where \( X \subseteq \mathbb{R}^m \)

(iii) \( \phi_g \subseteq (X \times Y)^T \) represents the guarantee for the input-output trajectories of the block, where \( Y \subseteq \mathbb{R}^m \)

The contract specifies the expected trajectory for each elementary block. An example of a contract for a block Gain is shown below.

**Example 8 (Contract for Gain).** As an example, we consider the Gain block that represented in Example 4. A contract specification for Gain is a tuple: \( C = (x, y, \phi_a, \phi_g) \), where

\[
\phi_a := \{ x \in \mathbb{R}^T | \forall t \in T, x(t) \in \mathbb{R} \},
\]

\[
\phi_g := \{ (x, y) \in (\mathbb{R} \times \mathbb{R})^T | \forall t \in T, y(t) = k \cdot x(t) \}.
\]

Next, we will define the satisfaction relation, which relates a block to a contract by determining when a given block’s trajectories satisfy the specified specification. To define satisfaction relation, we first define a projection that expresses the input trajectory to aid that definition. A projection of a set \( Tr \subseteq (X \times Y)^T \) into \( X \subset X^T \) is defined as the set \( Tr|_X = \{ (x, y) \in X^T | y \in Tr \} \), where \( T \) can be either the discrete-time domain or the continuous-time domain. We are now in a position to define satisfaction relation.

**Definition 11 (Satisfaction).** Let \( B = (x, s, y, \phi, f) \) be a block and \( C = (x, y, \phi_a, \phi_g) \) be a contract. We say \( B \) satisfies \( C \), denoted by \( B \models C \), if \( Tr|_X \subseteq \phi_a \) and \( Tr \subseteq \phi_g \).

We say that \( B \) is a correct implementation of \( C \) if \( B \models C \).

We have defined the notion of contract and satisfaction relation. The primary task of the proof method is to state that the satisfaction relation is preserved by composition. In the following, we study the composition operators of contracts according to the composition operators of blocks, i.e., serial, parallel, and algebraic loop-free feedback with multistep delays. We first define the serial composition of contracts.

**Definition 12 (The serial composition of contracts).** Let \( C_i = (x_i, y, \phi_{a_i}, \phi_{g_i}) \) be contracts for \( i = 1, 2 \). The serial composition of \( C_1 \) and \( C_2 \), written \( C_1 ; C_2 \), is a contract \( (x, y, \phi_{a'}, \phi_{g'}) \), where

\[
x = x_1,
\]

\[
y = y_2,
\]

\[
\phi_{a'} = \{ x_1 \in X^T | x_1 \in \phi_{a_1} \land (\exists y_1)(\{ x_1, y_1 \in \phi_{a_1} \land y_1 \in \phi_{g_1} \}) \},
\]

\[
\phi_{g'} = \{ (x_1, y_2) \in (X \times Y)^T | (\exists x_2)(\{ x_1, y_2 \in \phi_{g_2} \land (x_2, y_2) \in \phi_{g_2} \}) \},
\]

\[
\phi_p = \land_{(x_i, y_i)} y_{i+1}(t) = x_{i+1}(t), \forall t \in T.
\]

The serial composition preserves the satisfaction relation. That is, if blocks satisfy their contracts, respectively, then the composition of blocks satisfies the composition of contracts.
Theorem 13 (Serial composition preserves satisfaction). If \( B_1 \models C_1 \) and \( B_2 \models C_2 \), then \( B_1 \hat{;} B_2 \models C_1 \hat{;} C_2 \).

Proof. We show a proof here for blocks with continuous-time trajectories, and the proof for blocks with discrete-time trajectories is similar. Let \( B_1 = (x_1, y_1, \phi_1, f_1) \) and \( B_2 = (x_2, y_2, \phi_2, f_2) \) be continuous-time blocks. The observable trajectories of \( B_1 \) and \( B_2 \) are denoted as \( \tau_{B_1} = \{(x_1(t), y_1(t)) : T \rightarrow X \times Y \} \) and \( \tau_{B_2} = \{(x_2(t), y_2(t)) : T \rightarrow X \times Y \} \), respectively, where \( x_1, x_2 \subseteq \mathcal{R}^m, y_1, y_2 \subseteq \mathcal{R}^m, n, m \in \mathcal{N}^+ \). The projection of \( \tau_{B_1} \) onto \( \mathcal{X}_1 \) is \( \tau_{B_1|\mathcal{X}_1} = \{x_1 | \exists y_1 (x_1, y_1) \in \tau_{B_1}\} \). The projection of \( \tau_{B_2|\mathcal{X}_1} \) onto \( \mathcal{X}_2 \) is \( \tau_{B_2|\mathcal{X}_1} = \{x_2 | \exists y_2 (x_2, y_2) \in \tau_{B_2}\} \).

According to Definition 5, we write \( B_1 \hat{;} B_2 = (x_1, y_1, \phi_1, f_1, y_2, \phi_2, f_2) \) and \( \phi = \phi_2 \alpha \phi \). Thus, \( \tau_{B_1 \hat{;} B_2|\mathcal{X}_1} \subseteq \mathcal{X}_1 \). Therefore, \( \tau_{B_1 \hat{;} B_2|\mathcal{X}_1} \subseteq \mathcal{X}_1 \) implies \( \tau_{B_1 \hat{;} B_2 \models C_1 \hat{;} C_2} \).

Theorem 15 (Parallel composition preserves satisfaction). If \( B_1 \models C_1 \) and \( B_2 \models C_2 \), then \( B_1 \mid B_2 \models C_1 \mid C_2 \).

Proof. We show here a proof for blocks with continuous-time trajectories, and the proof for blocks with discrete-time trajectories is similar. Let \( B_1 = (x_1, y_1, \phi_1, f_1) \) and \( B_2 = (x_2, y_2, \phi_2, f_2) \) be continuous-time blocks with continuous-time trajectories. The observable trajectories of \( B_1 \) and \( B_2 \) are denoted as \( \tau_{B_1} = \{(x_1(t), y_1(t)) : T \rightarrow X \times Y \} \) and \( \tau_{B_2} = \{(x_2(t), y_2(t)) : T \rightarrow X \times Y \} \), respectively, where \( x_1, x_2 \subseteq \mathcal{R}^m, y_1, y_2 \subseteq \mathcal{R}^m, n, m \in \mathcal{N}^+ \). The projection of \( \tau_{B_1} \) onto \( \mathcal{X}_1 \) is \( \tau_{B_1|\mathcal{X}_1} = \{x_1 | \exists y_1 (x_1, y_1) \in \tau_{B_1}\} \). The projection of \( \tau_{B_2} \) onto \( \mathcal{X}_2 \) is \( \tau_{B_2|\mathcal{X}_1} = \{x_2 | \exists y_2 (x_2, y_2) \in \tau_{B_2}\} \).

According to Definition 12, we have \( \tau_{B_1 \mid B_2|\mathcal{X}_1} \subseteq \mathcal{X}_1 \). Thus, \( \tau_{B_1 \mid B_2 \models C_1 \mid C_2} \). Therefore, \( \tau_{B_1 \mid B_2 \models C_1 \hat{;} C_2} \).

Definition 14 (The parallel composition of contracts). Given contracts \( C_1 = (x_1, y_1, \phi_1, f_1) \) and \( C_2 = (x_2, y_2, \phi_2, f_2) \), we define the parallel composition of contracts as \( C_1 \parallel C_2 = (x, y, \phi, f) \), where \( x = (x_1, x_2), y = (y_1, y_2) \), and

\[
\phi = \{x \in X^T | x 
\phi \mid X_2 \in \phi_2^\phi \},
\phi_\parallel = \{x, y \in (X \times Y)^T | (x_1, y_1) \in \phi_1^\phi \land (x_2, y_2) \in \phi_2^\phi \}.
\]

The parallel composition also preserves the satisfaction relation. That is, if blocks satisfy their contracts, respectively, then the parallel composition of blocks satisfies the parallel composition of contracts.
\( \phi_{g} \) be the contracts. The algebraic loop-free feedback composition of \( C_{1} \) and \( C_{2} \) be defined as \( C_{1} \otimes_{f} C_{2} = (x, y, \phi_{a}, \phi_{g}) \),

\[
\phi_{a} := \{ x_{1,1} \in X^{T} \mid x_{1,1,} \in 1_{f}^{a} \land \left( (\exists y_{1})(x_{1}, y_{1}) \in 1_{g}^{a} \land \phi_{p} \rightarrow y_{1} \in 1_{g}^{g} \right) \land \left( (\exists y_{2})(x_{2}, y_{2}) \in 1_{g}^{g} \land \phi_{p} \rightarrow y_{2} \in 1_{a}^{a} \right) \},
\]

\[
\phi_{g} := \{ (x_{1,1}, y_{1}) \in (X \times Y)^{T} \mid (\exists x_{1,2}, \exists x_{2}) \left( (x_{1,1}, y_{1}) \in 1_{g}^{a} \land \phi_{p} \land (x_{2}, y_{2}) \in 1_{g}^{g} \right) \},
\]

\[
\phi_{p} := \phi_{1} \land \phi_{2}.
\]

where  \( \phi_{p_{1}} \) and \( \phi_{p_{2}} \) are the contracts. The algebraic loop-free feedback composition also preserves the satisfaction relation.

**Theorem 18** (Algebraic loop-free feedback composition preserves satisfaction). Let \( B_{1} \) be the feedback with an algebraic loop and \( B_{2} \) be a block with multistep delays. Let \( C_{1} = (x_{1,1}, y_{1,1}) \) and \( C_{2} = (x_{2,1}, y_{2,1}) \) be the contracts. If \( B_{1} \sim B_{2} \),\( B_{3} \subseteq C_{1} \) and \( B_{3} \subseteq C_{2} \), then \( B_{1} \otimes_{f} B_{2} \subseteq C_{1} \otimes_{f} C_{2} \).

**Proof.** It easily follows from Theorem 13 to Theorem 15. \( \square \)

We next turn to the refinement relation between the contract. We follow a standard notion inspired by [20].

**Definition 19** (Refinement of contracts). Let \( C_{1} = (x, y, \phi_{a}^{1}, \phi_{g}^{1}) \) and \( C_{2} = (x, y, \phi_{a}^{2}, \phi_{g}^{2}) \) be two contracts. We say \( C_{2} \) refines \( C_{1} \), denoted by \( C_{2} \preceq C_{1} \), if \( \phi_{a}^{2} \subseteq \phi_{a}^{1} \) and \( \phi_{g}^{2} \subseteq \phi_{g}^{1} \).

Refinement relaxes assumptions and reinforces guarantees, therefore, strengthening the contract. Obviously, the following refinement rule holds.

Let \( C_{1} = (x, y, \phi_{a}^{1}, \phi_{g}^{1}) \) and \( C_{2} = (x, y, \phi_{a}^{2}, \phi_{g}^{2}) \) be two contracts. Then

(i) \( C_{1} \preceq C_{1} \) for \( i = 1, 2 \)

(ii) If \( \phi_{a}^{1} \subseteq \phi_{a}^{2} \) and \( \phi_{a}^{2} = \phi_{g}^{1} \), then \( C_{2} \preceq C_{1} \)

(iii) If \( \phi_{g}^{1} \subseteq \phi_{g}^{2} \) and \( \phi_{a}^{1} = \phi_{a}^{2} \), then \( C_{2} \preceq C_{1} \)

The refinement implies that every Simulink block diagram satisfies the \( C_{2} \) also satisfies the \( C_{1} \). This gives us the following property of correctness.

Theorem 21 (Correctness). Let \( B = (x, y, \phi_{a}, \phi_{f}) \) be a block. Let \( C_{1} = (x, y, \phi_{a}^{1}, \phi_{g}^{1}) \) and \( C_{2} = (x, y, \phi_{a}^{2}, \phi_{g}^{2}) \) be contracts. The \( B \) is said to be correct for \( C_{2} \) if \( \{ B \models C_{1} \land C_{2} \} \Rightarrow B_{C_{2}} \).

**Proof.** Suppose \( C_{1} \models C_{2} \), in terms of Definition 19, then \( \phi_{a}^{2} \subseteq \phi_{a}^{1} \) and \( \phi_{g}^{2} \subseteq \phi_{g}^{1} \). Suppose \( B_{C_{2}} \), then, for all \( x \in T_{r} \subseteq \phi_{a}^{1} \) and \( T_{r} \subseteq \phi_{g}^{1} \). Hence, if \( x \in T_{r} \subseteq \phi_{a}^{2} \), then \( B_{C_{2}} \).

Theorem 21 is essential in our contract theory since it relates the Simulink block diagram (composition block) to a system contract specification. That is, if a Simulink block diagram satisfies the composition contract, and the composition contract refines the system contract, then it also satisfies the system contract. This ensures that the Simulink block diagram is correct.

**6. Case Study**

We have discussed the proof approach in the previous section, and in this section, the approach that we proposed is being implemented through a real-world case study.

**6.1. Problem Statement.** We examine a case, a safety-critical water level control system of reservoir, to illustrate how to verify that the controller model of a reservoir originating from the farmland irrigation satisfy the safety requirements.

This reservoir mainly is used for irrigation and considers the comprehensive utilization of flood control and aquaculture. An inlet water pipe and a spillway exist on the top and bottom of the dam, respectively. The maximum dam height is 40m, the maximum is 30m, and the minimum water level is 10m. The management facilities of the reservoir are very backward. There are no special management agencies to observe and safety check the water level and no timely flood control.

We will use Simulink to model the water level control system of the reservoir and simulate the water level trajectories to monitor the water level control operation. Then we adopt the method that we presented to verify the Simulink block diagram’s correctness for ensuring the water level’s safety according to the reservoir’s actual situation. The water level controls the opening or closing of the valves. That is, the water level is neither higher than the highest water level,
which causes the dam to be overloaded, nor can the aquaculture industry and farmland irrigation be affected by the water level being lower than the lowest water level.

6.2. Architectural Overview of Control System. The control system of water level consists of the following two major components: Sensor and Plant.

(i) **Sensor.** The Sensor is a water level sensor. Inputs about the water level of the reservoir from its corresponding level sensor, and we suppose that the Sensor works properly

(ii) **Plant.** The Plant is made up of four principal components, as shown in Figure 7(a): Reservoir, Valves, Controller, and Water inlet and outlet pipe

(a) **Reservoir.** The Reservoir has parameters giving the maximum water lever \( L_2 \), the minimum water lever \( L_1 \), and the water level of the reservoir \( h \geq 0 \)

(b) **Water valves.** The Reservoir has two water valves. The inlet valve \( V_2 \) is at the top of the Reservoir, and the outlet valve \( V_1 \) is at the bottom. We suppose that the valves are either fully opened or fully closed immediately

(c) **Controller.** The Controller is able to read the current water level provided by the sensor, and the output is the command signal *open* or *closed*. The controller’s target is to keep the water level between the minimum water lever \( L_1 \) and the maximum water lever \( L_2 \)

(d) **Water inlet and outlet.** The inlet pipe \( F_{in} \geq 0 \) and \( F_{out} \geq 0 \), and \( F_{in} \neq F_{out} \).

As the system is a closed-loop system, the controller must work with the reservoir. In Figure 8(a), a system including both controller and plant is given. We only focus on the controller.

6.3. The Safety Requirements of Water Level. The safety requirements for water level \( h \) in the controller are given below, and the purpose is to avoid the water empty and filling of reservoir.

Requirement 1 When the \( h \) of reservoir is above \( L_1 \), the \( V_1 \) is opened, and when the \( h \) of reservoir is below \( L_1 \), the \( V_1 \) is closed.

Requirement 2 When the \( h \) of reservoir is above \( L_2 \), the \( V_2 \) is closed, and when the \( h \) of reservoir is below \( L_2 \), the \( V_2 \) is opened.

Driven by requirement, we now refine system-level requirements into block-level implementations. When the system is refined, subsystem details are added. We first decompose the controller into three different controllers: *Decision controller*, *Controller_1* controller, and *Controller_2*.
controller. The block diagrams of these controllers can be depicted graphically as in Figure 8(b). In the block diagram, the Decision decides that the water level is high or low. The Controller1 computes whether the Valve1 should be open, while the Controller2 computes if the Valve2 should be open.

For the Decision, if the water level is above the maximum water level, the controller has two controllers. The block diagrams of these controllers can be denoted as $C_{\text{Controller1}}$. Otherwise, the Valve2 should be open(1). For the Valve1, if the water level is below the minimum water level, then Valve1 should be close(0). Otherwise, then Valve1 should be open(1).

We use the contract as a formal specification language. We suppose $L_1 = 10m$, $L_2 = 30m$, and $0m \leq h(t) \leq 40m$. The system-level safety requirement specification 1 is denoted as $C_{\text{req1}} = \{ h, y_{x_1}, \phi_{\text{req1}}, \phi_{g_{\text{req1}}} \}$.

\[
\phi_{\text{req1}} = \{ h \in R^T \mid \forall t \in T, 0 \leq h(t) \leq 40 \},
\]
\[
\phi_{g_{\text{req1}}} = \{ (h, y_{x_1}) \in (R \times R)^T \mid \forall t \in T, y_{x_1} = \begin{cases} \text{open(1)}, & h(t) > 10, \\ \text{close(0)}, & 0 \leq h(t) < 10. \end{cases} \}
\]

The system-level safety requirement specification 2 is denoted as $C_{\text{req2}} = \{ h, y_{x_2}, \phi_{\text{req2}}, \phi_{g_{\text{req2}}} \}$, where

\[
\phi_{\text{req2}} = \{ h \in R^T \mid \forall t \in T, 0 \leq h(t) \leq 40 \},
\]
\[
\phi_{g_{\text{req2}}} = \{ (h, y_{x_2}) \in (R \times R)^T \mid \forall t \in T, y_{x_2} = \begin{cases} \text{close(0)}, & h(t) \geq 30, \\ \text{open(1)}, & 0 \leq h(t) < 30. \end{cases} \}
\]

6.4. Verification of Correctness for the Controller System

We construct a Simulink block diagram for the control system according to requirements shown in Figure 7(b). Following the previous technical route, the verification method applied in this case consists of two stages.

Stage 1: verifying the Simulink block diagram satisfies the corresponding composition of contract.

The Decision controller has two controllers: Decision1 and Decision2. The Decision1 determines whether the water level is higher than the highest water level. The composition of Simulink block diagrams of Decision2 is denoted as $\otimes B_{d_2} = \langle B_{c_2}, B_{w_{12}} \rangle$; $\rho_1 : B_{w_{12}}$, where $\rho_1 = \{ (y_{c_1}, sw_{11}), (y_{c_1}, sw_{13}) \}$. We denote the $h$ by $sw_{12}$, and then $\otimes B_{d_2} = \{ (sw_{12}, 0, y_{d_2}, f_{d_2} (sw_{12}(t), 0)) \}$, where,

\[
f_{d_2} (sw_{12}, 0) = \begin{cases} 1(\text{high}) & sw_{12}(t) \geq 30, \\ 0(\text{low}) & sw_{12}(t) < 30. \end{cases}
\]

Therefore, the observable trajectories of $\otimes B_{d_2}$ are denoted as $\text{Tr}_{d_2} = \{ (sw_{12}, y_{d_2}) ; T \to R \times R \forall t \in T, y_{d_2}(t) = f_{d_2} (sw_{12}(t), 0) \}$, where

\[
y_{d_2}(t) = \begin{cases} 1(\text{high}) & sw_{12}(t) \geq 30, \\ 0(\text{low}) & sw_{12}(t) < 30. \end{cases}
\]

The main task in the next step is to apply the composition rules to calculate the composition contract and prove that the composition of blocks satisfies the composition of the corresponding contracts. We first specify a contract for each elementary block following Table 1. Then we apply the composition rules to calculate the composition contract and prove that the composition of blocks satisfies the composition contracts.

The composition contract of the Decision2 is stated as $\otimes C_{d_2} = \{ (sw_{12}, y_{d_2}, \phi_{d_2}, \phi_{g_{d_2}}) \}$, where

\[
\phi_{d_2} = \{ sw_{12} \in R^T \mid \forall t \in T, (\exists y_{B_{c_2}}) : (0, y_{B_{c_2}}) \in \phi_{g_{d_2}} \land \phi_{d_2} \to y_{B_{c_2}} \in \phi_{w_{12}} \},
\]
\[
\phi_{g_{d_2}} = \{ (0, y_{d_2}) \in (R \times R)^T \mid \forall t \in T, y_{d_2}(t) = f_{w_{12}} (sw_{12}(t), 0) \},
\]}
\[
\phi_{d_1} = \{ y_{c_1}(t) = sw_{11}(t) \land y_{c_1}(t) = sw_{13}(t) \},
\]

Since

\[
\phi_{g_{d_1}} = \{ (0, y_{B_{c_1}}) \in (R^T) \mid \forall t \in T, y_{c_1}(t) = 1 \land y_{c_1}(t) = 0 \},
\]
\[
\phi_{g_{d_2}} = \{ (sw_{12}, y_{d_2}) \in (R \times R)^T \mid \forall t \in T, y_{d_2}(t) = f_{w_{12}} (sw_{12}(t), 0) \},
\]

where

\[
y_{d_2}(t) = \begin{cases} sw_{11} & sw_{12} \geq 30, \\ sw_{13} & sw_{12} < 30. \end{cases}
\]

We use quantifier elimination law to simplify the expression. Therefore, the composition contract $(\otimes C_{d_2})$ of the $D$
cision$_1$ controller was written in the following form.

$$
\phi_d^1 = \{ s \mu_{12} \in \mathcal{R}^T | \forall t \in T, s \mu_{12}(t) \in \mathcal{R} \},
$$

$$
\phi_g^1 = \left\{ \left( s \mu_{12}, y_{d1} \right) \in (\mathcal{R} \times \mathcal{R})^T | \forall t \in T, y_{d1}(t) = \begin{cases} \text{high}(1) & s \mu_{12}(t) \geq 0.3, \\ \text{low}(0) & s \mu_{12}(t) < 0.3. \end{cases} \right\}.
$$

(9)

Similarly, the Decision$_1$ determines whether the water level is below the lowest water level. The composition of Simulink block diagrams of Decision$_1$ is denoted as $\otimes B_{d1} = (B_{c1} \parallel B_{c2}) ; B_{s2}$, where $B_{s2} = \{ (y_{c1}, s'w_{11}), (y_{c1}, s'w_{13}) \}$. Let $B_{c1} = B_{c2} \parallel B_{c3}$ and $y_{B1} = (y_{c3}, y_{c4})$. We denote the Simulink block diagrams of Decision$_1$ by $\otimes B_{d1} = (s'w_{12}, 0, y_{d1}, s'f_{d1}(s'w_{12}(t), 0))$, where

$$
f_{d1}(s'w_{12}(t), 0) = \begin{cases} 0(\text{low}) & s'w_{12}(t) \geq 10, \\ 1(\text{low}) & s'w_{12}(t) < 10. \end{cases}
$$

(10)

The observable trajectories of $\otimes B_{d1}$ is denoted as $Tr_{1} = \{(s'w_{12}, y_{d1}); T \mapsto \mathcal{R} \times \mathcal{R} | \forall t \in T, y_{d1}(t) = f_{d1}(s'w_{12}(t), 0) \}$, where

$$
y_{d1}(t) = \begin{cases} 0(\text{low}) & s'w_{12}(t) \geq 10, \\ 1(\text{low}) & s'w_{12}(t) < 10. \end{cases}
$$

(11)

The composition contract of Decision$_1$ controller for $\otimes B_{d1}$ is stated as $\otimes C_{d1} = (s'w_{12}, y_{d1}, \phi_a^1, \phi_g^1)$, where

$$
\phi_{a1} = \{ s'w_{12} \in \mathcal{R}^T | \forall t \in T, s'w_{12}(t) \in \mathcal{R} \},
$$

$$
\phi_{g1} = \left\{ \left( s'w_{12}, y_{d1} \right) \in (\mathcal{R} \times \mathcal{R})^T | \forall t \in T, y_{d1}(t) = \begin{cases} \text{low}(1) & s'w_{12}(t) \geq 10, \\ \text{low}(0) & s'w_{12}(t) < 10. \end{cases} \right\}.
$$

(12)

Since variables $s \mu_{12}$ and $s'w_{12}$ are used to read the values of water level, so $s \mu_{12}(t) = h(t)$ and $s'w_{12}(t) = h(t)$. According to Definition 11, $\forall t \in T$, we have $\{ s \mu_{12} \in \mathcal{R}^T | s \mu_{12}(t) \geq 0 \} \subseteq \phi_{a1}^1$, $Tr_{3} \subseteq \phi_{g1}^1$, and $\{ s'w_{12} \in \mathcal{R}^T | s'w_{12}(t) \geq 0 \} \subseteq \phi_{a1}^1$, $Tr_{1} \subseteq \phi_{g1}^1$. Hence, $\otimes B_{d1} \equiv \otimes C_{d1}$ and $\otimes B_{d1} \equiv \otimes C_{d1}$.

We next turn to research the controller Controller$_1$. It can be represented as a composition block $\otimes B_{r1} = (B_{v1} \parallel B_{c}) ; B_{s2}$, where $B_{s2} = \{ (y_{v1}, s'w_{23}), (y_{v1}, s'w_{21}) \}$. Let $B_{v1} = (s'w_{22}, 0, y_{r1}, -y_{r1}, f_{v1}(s'w_{22}(t), 0))$, where

$$
f_{v1}(s'w_{22}(t), 0) = \begin{cases} 0(\text{close}) & s'w_{22}(t) > 0, \\ 1(\text{open}) & s'w_{22}(t) \leq 0. \end{cases}
$$

(13)

Therefore, the observable trajectories of $\otimes B_{v1}$ is denoted as $Tr_{v1} = \{(s'w_{22}, y_{v1}); T \mapsto \mathcal{R} \times \mathcal{R} | \forall t \in T, y_{v1}(t) = f_{v1}(s'w_{22}(t), 0) \}$, where

$$
y_{v1}(t) = \begin{cases} 0(\text{close}) & s'w_{22}(t) > 0, \\ 1(\text{open}) & s'w_{22}(t) \leq 0. \end{cases}
$$

(14)

The composition contract of Controller$_1$ is stated as $\otimes C_{v1} = (s'w_{22}, y_{v1}, \phi_{a1}^1, \phi_{g1}^1)$, where

$$
\phi_{a1}^1 = \{ s'w_{22} \in \mathcal{R}^T | \forall t \in T, s'w_{22}(t) \in \mathcal{R} \},
$$

$$
\phi_{g1}^1 = \left\{ \left( s'w_{22}, y_{v1} \right) \in (\mathcal{R} \times \mathcal{R})^T | \forall t \in T, y_{v1}(t) = \begin{cases} 0(\text{close}) & s'w_{22}(t) > 0, \\ 1(\text{open}) & s'w_{22}(t) \leq 0. \end{cases} \right\}.
$$

(15)

According to Definition 11, $\forall t \in T$, we have $\{ s'w_{22} : T \mapsto \mathcal{R} | s'w_{22}(t) \in \mathcal{R} \} \subseteq \phi_{a1}^1$, and $Tr_{1} \subseteq \phi_{g1}^1$. Hence, $\otimes B_{v1} \equiv \otimes C_{v1}$.

For requirement 1 Next, we will construct the Decision$_1$ controller and Controller$_1$ controller for verifying requirement 1. We denote the composition of Decision$_1$ and Controller$_1$ by $\otimes B_{1}$. We denote the $h$ by $s'w_{12}$. We write $\otimes B_{1} = (s'w_{12}, 0, y_{r1}, -y_{r1}, f_{1}(s'w_{12}(t), 0))$, where

$$
f_{1}(s'w_{12}(t), 0) = \begin{cases} 0(\text{close}) & s'w_{12}(t) > 0, \\ 1(\text{open}) & s'w_{12}(t) \leq 0. \end{cases}
$$

(16)

The observable trajectories of $\otimes B_{1}$ is denoted as $Tr_{1} = \{(s'w_{12}, y_{r1}); T \mapsto \mathcal{R} \times \mathcal{R} | \forall t \in T, f_{1}(s'w_{12}(t), 0) \}$, where

$$
y_{1}(t) = \begin{cases} 0(\text{close}) & s'w_{12}(t) > 0, \\ 0(\text{open}) & s'w_{12}(t) < 10. \end{cases}
$$

(17)

The composition contract of $\otimes B_{1}$ is defined as $\otimes C_{1} = (s'w_{12}, y_{r1}, \phi_{a1}^1, \phi_{g1}^1)$, and

$$
\phi_{a1}^1 = \{ s'w_{12} \in \mathcal{R}^T | \forall t \in T, s'w_{12}(t) \in \mathcal{R} \},
$$

$$
\phi_{g1}^1 = \left\{ \left( s'w_{12}, y_{r1} \right) \in (\mathcal{R} \times \mathcal{R})^T | \forall t \in T, y_{r1}(t) = \begin{cases} 0(\text{close}) & s'w_{12}(t) > 10, \\ 0(\text{close}) & s'w_{12}(t) < 10. \end{cases} \right\}.
$$

(18)

$\forall t \in T$, we have $\{ s'w_{12}(t) : T \mapsto \mathcal{R} | s'w_{12}(t) \in \mathcal{R} \} \subseteq \phi_{a1}^1$, and $Tr_{1} \subseteq \phi_{g1}^1$. Hence, $B_{1} \equiv C_{1}$. 
Stage 2: correctness verification. Our primary goal in this step is to verify that the Simulink block diagram \((\otimes B_{r_2})\) is correct for system contract \(C_{req1}\). In terms of Definition 19, we get \(\phi_a^{req1} \subseteq \phi_a^{lm1}\) and \(\phi_g^{lm1} \subseteq \phi_g^{req1}\), then, \(C_{lm1} \models C_{req1}\). According to Theorem 21, we have \(B_{lm1} \vdash C_{lm1} \land C_{lm1} \models C_{req1} \Rightarrow B_{lm1} \vdash C_{req1}\). For requirement 2

We next turn to verify the correctness of the Simulink block diagrams for requirement 2. Similar to the method above, the Controller_2 controller can be represented as a composition block \(\otimes B_{r_2} = (B_y \parallel B_{r_2}); \rho_2; B_{sw2}\), where \(\rho_2 = \{ (y_v, sw_{23}), (y_v, sw_{21}) \}\). Then \(\otimes B_{r_2} = (sw_{22}, 0, y_v, y_v, \phi_y, \phi_v) = f_{r_2}(sw_{22}(t), 0)\), where

\[
y_v(t) = \begin{cases} (0, close), & sw_{22}(t) > 0 \\ (1, open), & sw_{22}(t) \leq 0 \end{cases}
\]

Therefore, the observable trajectories of the subsystem \(\otimes B_{r_2}\) is denoted as \(T_{r_2} \{ (sw_{22}, y_v); T \mapsto R \times R | \forall t \in T, y_v(t) = f_{r_2}(sw_{22}(t), 0) \}\), where

\[
y_v(t) = \begin{cases} (0, close), & sw_{22}(t) > 0 \\ (1, open), & sw_{22}(t) \leq 0 \end{cases}
\]

Stage 1: verifying the Simulink block diagram satisfies the corresponding composition of contract.

The composition contract of Controller_2 is defined as \(\otimes C_{r_2} = (sw_{22}, y_v, \phi_y, \phi_v)\), where

\[
\phi_y = (sw_{22} \in \mathcal{R} | \forall t \in T, sw_{22}(t) \in \mathcal{R})
\]

Next, we will compose the Decision_1 and Controller_2 to verify requirement 2. Let \(\otimes B_{lm2} = \otimes B_{d_2}; \otimes B_{r_2} = (sw_{12}, 0, y_v, y_v, \phi_y, \phi_v) = f_{r_2}(sw_{12}(t), 0)\), where

\[
f_{lm2}(sw_{12}(t), 0) = \begin{cases} (0, close), & sw_{12}(t) \geq 30 \\ (1, open), & sw_{12}(t) < 30 \end{cases}
\]

The observable trajectories of \(\otimes B_{lm2}\) are denoted as \(T_{lm2} \{ (sw_{12}, y_v); T \mapsto R \times R | \forall t \in T, y_{lm2}(t) = f_{lm2}(sw_{12}(t), 0) \}\), where

\[
y_{lm2}(t) = \begin{cases} (0, close), & sw_{12}(t) \geq 30 \\ (1, open), & sw_{12}(t) < 30 \end{cases}
\]

The composition contract is defined as \(\otimes C_{lm2} = (sw_{22}, y_v, \phi_y^{lm2}, \phi_v^{lm2})\), and

\[
\phi_y^{lm2} = \{ sw_{12} \in \mathcal{R} | \forall t \in T, sw_{12}(t) \in \mathcal{R} \}, \phi_v^{lm2} = \{ (sw_{12}, y_v); (sw_{12}, y_v) = f_{lm2}(sw_{12}(t), 0) \}
\]

Stage 2: correctness verification. Our main goal in this step is to verify that the Simulink block diagram \((\otimes B_{a})\) is a correct implementation of system contract \(C_{req2}\). For \(T \in T\), we have \(\{ sw_{12}(t); T \mapsto \mathcal{R} | sw_{12}(t) \in \mathcal{R} \} \subseteq \phi_a^{lm2}\), and \(T_{lm2} \subseteq \phi_g^{lm2}\). Hence, \(B_{lm2} \vdash C_{lm2}\). In terms of Definition 19, we get \(\phi_{req2} \subseteq \phi_a^{lm2}, \phi_g^{lm2} \subseteq \phi_g^{req2}\). Then, \(C_{lm2} \vdash C_{req2}\). According to Theorem 21, we have \(B_{lm2} \vdash C_{lm2} \land C_{lm2} \models C_{req2} \Rightarrow B_{lm2} \vdash C_{req2}\).

In the example above, when an attacker adds a malicious logic bomb to Simulink, it can maliciously manipulate the input and output behaviors of any block in the block diagrams and the water level values read from the Sensor. This may lead to the block diagram’s behavior not satisfying the system’s formal specification. Considering from this perspective, our approach can identify and verify whether the designed CPS is planted with the logic bomb.

7. Conclusion and Future Work

In this paper, we presented a method to prove the correctness of Simulink block diagrams with discrete-time or continuous-time blocks using contract. This approach addressed the problem of proving that the system formal specifications are satisfied by composition and refinement of trajectories. We showed the usability of our proposals via a use case (as an example) which models the control system of a reservoir. Our method can improve the reliability of Simulink and reduce the development costs by performing early safety verification on verification of the target system.

In this work, we made the first step towards the contract-based verification of cyber-physical models in Simulink. Future work includes extending the work along several dimensions. First, this work considers single rate Simulink block diagrams. We will extend this work to consider the multiple sample times (multirate systems) diagrams. Another nontrivial extension involves applying our idea to prove the correctness of Simulink block diagrams mixture of discrete-time and continuous-time blocks. More challenging would be to develop a contract-based refinement approach to handle Statetow. Second, this work only considers the blocks whose inputs-outputs can be explicitly represented through the mathematical relation. It is interesting to prove the correctness of the Simulink block diagrams, whose input-output behavior be expressed implicitly by a mathematical relation. Third, we plan to automatically verify
the correctness of the Simulink block diagrams based on our method.

**Data Availability**

No data were used to support this study.

**Conflicts of Interest**

The authors declare that there is no conflict of interest regarding the publication of this article.

**Acknowledgments**

This work was funded in part by the Chongqing Graduate Research and Innovation Project (grant No. CYB20098), the National Natural Science Foundation of China (Nos. 62032019, 61732019, 61672435, and 6200298), and the Capacity Development Grant of Southwest University (SWU116007). We are grateful for the valuable comments from Prof. Xinxin Liu and Prof. Zheng Yang on the earlier versions of this paper. We thank the anonymous referees whose suggestions improved the presentation.

**References**


