International Journal on Software Tools for Technology Transfer (2018) 20:615–643
DOI 10.1007/s10009-018-0502-9

Tactical Contract Composition for Hybrid System Component Verification

Andreas Müller[1], **Stefan Mitsch**[2], **Werner Retschitzegger**[1], **Wieland Schwinger**[1], **André Platzer**[2*]

[1] Department of Cooperative Information Systems
Johannes Kepler University, Altenbergerstr. 69, 4040 Linz, Austria
e-mail: {andreas.mueller,wieland.schwinger,werner.retschitzegger}@jku.at

[2] Computer Science Department
Carnegie Mellon University, Pittsburgh PA 15213, USA
e-mail: {smitsch,aplatzer}@cs.cmu.edu

Published online: 14 August 2018

Abstract We present an approach for hybrid systems that combines the advantages of component-based modeling (e.g., reduced model complexity) with the advantages of formal verification (e.g., guaranteed contract compliance). Component-based modeling can be used to split large models into multiple component models with local responsibilities to reduce modeling complexity. Yet, this only helps the analysis if verification proceeds one component at a time. In order to benefit from the decomposition of a system into components for both modeling and verification purposes, we prove that the safety of compatible components implies safety of the composed system. We implement our composition theorem as a tactic in the KeYmaera X theorem prover, allowing automatic generation of a KeYmaera X proof for the composite system from proofs for the components without soundness-critical changes to KeYmaera X.

Our approach supports component contracts (i. e., input assumptions and output guarantees for each component) that characterize the magnitude and rate of change of values exchanged between components. These contracts can take into account what has changed between two components in a given amount of time since the last exchange of information.

Key words: component-based development – hybrid systems – component-based verification

1 Introduction

Cyber-physical systems (CPS) feature discrete dynamics (e. g., autopilots in airplanes, controllers in self-driving cars) as well as continuous dynamics (e. g., motion of airplanes or cars) and are increasingly used in safety-critical areas. Models of such CPS (i. e., hybrid system models, e. g., hybrid automata [13], hybrid processes [6], hybrid programs [32]) are used to capture properties of these CPS as a basis to analyze their behavior and ensure safe operation with formal verification methods.

However, as the complexity of these systems increases, monolithic models and analysis techniques become unnecessarily challenging. As already established for discrete software, decomposition into subsystems with contracts is essential in taming the complexity of larger systems. We, thus, explore compositional modeling and verification techniques for hybrid systems that conclude safety of the entire system from separate isolated safety arguments about its subsystems and their interaction with the environment.

As a basis for our approach, we use differential dynamic logic dL [29,33,36], which is a hybrid systems specification and verification logic that is already compositional for each of its operators. Reasoning in dL splits models along the dL operators into smaller pieces. In this article we add a notion of components with interfaces and explain how to *make hybrid system theorem proving modular on a component level.* This achieves another layer of compositionality of larger granularity. We exploit the special structure of components and their contracts to compose *verified components* and their *safety proofs* to a verified CPS. Under certain precisely formalized compatibility conditions on how components are connected, we ensure that their compositions directly inherit safety from the safety of the components.

Component-based hybrid systems verification is challenging because both local component behavior (e. g., decisions and motion of a robot) and inherently global phenomena (e. g., time) co-occur, as components can interact virtually (e. g., robots communicate) and physically (e. g., a robot manipulates an object), and because their

[*] This material is based on research sponsored by DARPA under agreement DARPA FA8750-12-2-0291, AFOSR FA9550-16-1-0288, and by the Austrian Science Fund (FWF) P28187-N31.

Table 1: Hybrid Programs (HPs)

Statement	Meaning
$\alpha; \beta$	Sequentially composes β after α
$\alpha \cup \beta$	Executes either α or β, nondeterministically
α^*	Repeats α zero or more times
$x := \theta$	Assigns value of term θ to x
$x := *$	Assigns an arbitrary real value to x
$x' = \theta \,\&\, Q$	Continuous evolution[1]
$?Q$	Aborts run if formula Q is not true

interaction is subject to communication delays, measurement uncertainty, and actuation disturbance.

The key step is to represent the abstract behavior of a component in its interfaces phrased as contracts on the input assumptions and output guarantees. In addition to the precise contracts describing computations of discrete programs, continuous time and continuous dynamics in hybrid systems call for techniques to faithfully characterize the discrete-time observations that other components make about continuous phenomena. Our contracts, therefore, emphasize the externally observable nature of component behavior: they specify the *magnitude of change* between two observations (e. g., current speed is at most twice the previously observed speed) and also capture the *rate of change* (e. g., current speed is at most previous speed increased by accelerating for some time). Such contracts abstract the hybrid (continuous-time) behavior of one component to discrete-time observations available to other components. The isolated hybrid behavior of a component in question is, thus, analyzed with respect to a hybrid model in the own component but simpler discrete-time abstractions for all other components in the system. This reduction is important to ensure that not all details of all behaviors of all components need to be understood at the same time.

This article extends our previous work [25,27] with improved techniques for system composition: we exploit information encapsulation [28] to define parallel composition in an *associative* manner, introduce proof tactics to *automatically check assumptions and generate safety proofs* from component proofs, and *reflect practical considerations of composition* properly in proof obligations (e. g., observation with sensors, and glue code to bridge slight discrepancies between component interfaces).

2 Preliminaries: Differential Dynamic Logic

Syntax. For specifying and verifying correctness statements about hybrid systems, we use *differential dynamic logic* (dL) [29,33,36], which supports *hybrid programs* as a program notation for hybrid systems. The syntax and informal semantics of hybrid programs is summarized in Table 1. The sequential composition $\alpha; \beta$ expresses that β starts after α finishes. The nondeterministic choice $\alpha \cup \beta$ follows either α or β. The nondeterministic repetition operator α^* repeats α zero or more times. Discrete assignment $x := \theta$ instantaneously assigns the value of the term θ to the variable x, while $x := *$ assigns an arbitrary value to x. The ODE $\{x' = \theta \;\&\; Q\}$ describes a continuous evolution of x, where x' denotes derivation with respect to time within the evolution domain Q. The test $?Q$ checks that a condition expressed by property Q holds, and aborts if it does not. A typical pattern $x := *;\; ?a \le x \le b$, which involves assignment and tests, is to limit the assignment of arbitrary values to known bounds. Other control flow statements can be expressed with these primitives (e. g., if (Q) α else β can be expressed as $?Q; \alpha \cup ?\neg Q; \beta$) [30]. A no-operation statement `skip` is the test $?true$ that always holds.

For example, a time-triggered program

$$(y := *;\; ?y \le z;\; t := 0;\; \{x' = y, t' = 1 \;\&\; t \le 10\})^* \quad (1)$$

picks any real value for y that does not exceed z, resets time t to zero, and then in the ODE continuously evolves the value of x according to the fixed slope y while simultaneously increasing the value of t with constant slope 1. The ODE stops nondeterministically at any time, but at the latest before $t \le 10$ becomes false; then the program repeats by the * operator.

Semantics. The semantics of dL [29,33,36] is a Kripke semantics in which the states of the Kripke model are the states of the hybrid system. Let $\mathbb{R}$ denote the set of real numbers and $\mathcal{V}$ denote the set of variables. A state is a map $\nu : \mathcal{V} \to \mathbb{R}$ assigning a real value $\nu(x)$ to each variable $x \in \mathcal{V}$. We write $\nu \models \phi$ if formula ϕ is true at state ν. The real value of term θ at state ν is denoted $\nu[\![\theta]\!]$. The semantics of a hybrid program α is a relation $[\![\alpha]\!]$ between initial and final states. For example $\nu \models [\alpha]\phi$ iff $\omega \models \phi$ for all $(\nu, \omega) \in [\![\alpha]\!]$. We write $\alpha \equiv \beta$ to mean $[\![\alpha]\!] = [\![\beta]\!]$. For details on the semantics of hybrid programs see [29,33,36] and Appendix A.

Safety Properties. To specify safety properties about hybrid programs, dL provides modal operator $[\alpha]$. When ϕ is a dL formula describing a state and α is a hybrid program, then the dL formula $[\alpha]\phi$ expresses that all states reachable by α satisfy ϕ. The set of dL formulas relevant in this article is generated by the following EBNF grammar (θ_1, θ_2 are arithmetic expressions in $+, -, \cdot, /$ over the reals and where $\sim \in \{<, \le, =, \ge, >\}$):

$$\phi ::= \theta_1 \sim \theta_2 \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \phi \to \psi \mid \phi \leftrightarrow \psi \mid \\ \forall x\, \phi \mid \exists x\, \phi \mid [\alpha]\phi$$

For example, $\phi \to [\alpha]\psi$ says that formula ψ holds in all states reachable by program α from starting states

[1] A continuous evolution along the differential equation system $x' = \theta$ for an arbitrary real duration within the region described by formula Q.

that satisfy formula ϕ. Proofs for properties of nondeterministic repetitions often use *invariants*, representing properties that hold before and after each repetition. Even though there is no unified approach for invariant generation, if a safety property including a nondeterministic repetition is valid, an invariant exists [32].

In component compatibility it will be important to keep track of which component reads or changes which variables. FV(·) is used as an operator on terms, formulas and hybrid programs returning the free variables, i. e., the ones that are read, whereas BV(·) is an operator returning the bound variables, i. e., those written in assignments or ODEs [36]. For example, the free variables in program (1) are $\{t, x, y, z\}$, whereas the bound variables are $\{t, x, y\}$ since z is not written. Similarly, $\mathrm{V}(\cdot) = \mathrm{FV}(\cdot) \cup \mathrm{BV}(\cdot)$ returns all variables occurring in terms, formulas and hybrid programs, whether read or written. In definitions and formulas, we use dL to denote the set of all dL formulas, and *HP* to denote the set of all hybrid programs. We use "$\mapsto$" to define functions. $f = (a \mapsto b)$ means that the (partial) function f maps argument a to result b and is solely defined for a.

Program Independence. The order of independent programs without information flow between them is irrelevant, see Lemma 1. This insight will become important when establishing commutativity and associativity of our component composition operators, and in KeYmaera X in general when using lemmas to close proof obligations whose shape slightly differs syntactically from the lemma conclusion.

Lemma 1 (Program independence)**.** *Let ψ be a* dL *formula and α, β be independent hybrid programs without information flow between them, i. e., $BV(\alpha) \cap V(\beta) = \emptyset$ and $BV(\beta) \cap V(\alpha) = \emptyset$. Then this* dL *formula is valid:*

$$[\alpha; \beta]\psi \leftrightarrow [\beta; \alpha]\psi$$

Proof. Follows directly from the semantics of dL, see Appendix C. □

3 Component-based Modeling

In this section, we describe the fundamentals and steps of component-based modeling for component-based verification of CPS.

3.1 Components and Interfaces

We adopt common component notions (e. g., [12,44]) that consider a component as a superposition of a *behavior model* and an *interaction model* as illustrated schematically in Fig. 1: the behavior model describes the dynamic behavior of the component, while the interaction model defines the component ports and their properties that determine how one component interacts with others.

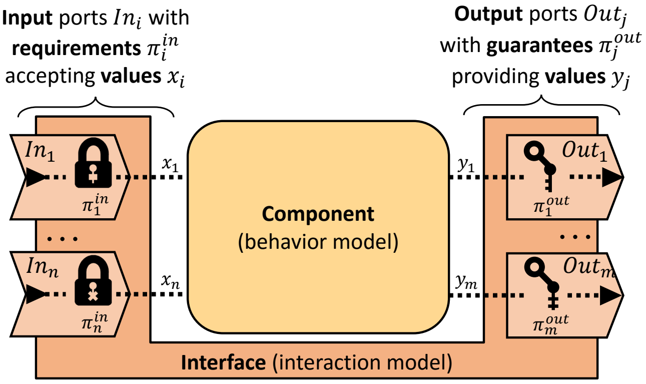
Input ports In_i with
requirements π_i^{in}
accepting **values** x_i
Output ports Out_j
with **guarantees** π_j^{out}
providing **values** y_j
In_1
π_1^{in}
x_1
In_n
π_n^{in}
x_n
Component
(behavior model)
y_1
y_m
Out_1
π_1^{out}
Out_m
π_m^{out}
Interface (interaction model)

Figure 1: Structure of a component and its interface: The internal behavior of a *component* is encapsulated by the *interface*, which specifies interaction with the component through (arbitrarily many) input ports In_i and output ports Out_j. The ports transfer external values x_i to the encapsulated component and emit values y_j from it. If two ports should be connected, the key (i. e., output guarantee π_j^{out}) must fit into the respective lock (i. e., input assumption π_i^{in}).

In the context of CPS, *components* consist of their discrete computations controlling continuous physical dynamics, which together represent the component's *internal behavior*. *Interfaces* describe a component's *externally observable behavior* and interaction capabilities. Interaction between components occurs through their interfaces: input ports receive external input from other components, output ports pass on component output to other components. Discrete time information sharing through ports requires that the involved components agree on the values allowed for transfer. Interface *contracts* specify input requirements and provide output guarantees with logical formulas characterizing the properties of values accepted on input ports and provided through output ports.

Ports can be vector-valued, i. e., transfer values of multiple variables at once, which are characterized through a common input requirement or output guarantee that allows relating their values (e. g., $a + b \leq 1$).

3.2 Composition and Compatibility

Components do not share variables. Each component has a set of local variables, which is exclusively accessed in this component's internal behavior and through ports. They may, however, share global unmodifiable constants, for example system parameters such as a legal maximum speed.[2]

[2] A set of globally shared constants is useful as a modeling construct. For implementation, global constants can be realized, e. g., through shared memory or simply as local constants with uniform values across components.

To build hybrid system models from components, a *composition operation* connects *compatible* components through their ports. A composition operation can be formalized as an binary operation, taking two components with their interfaces and additional information about their interactions, and returning a composed component and interface [12]. In order to provably derive system properties from component guarantees, we need a formal model of components, composition, and their meaning, which we obtain from the semantics of dL by specifying a *syntactic composition operation* on a syntactic component notion in hybrid programs. We use the operators of hybrid programs, to define a composition operation for components: on the level of operations, hybrid programs can be composed sequentially by using the sequential composition operator ";". Parallel composition is supported for differential equations in hybrid programs, but gives rise to issues of interleaving and synchronization [5] in the discrete fragment. Since components do not share variables and interaction between components occurs after controller execution through ports as synchronization points, controllers are independent by Lemma 1 and, therefore, can be arranged in any sequential order. Thus, we need no interleaving of the internal discrete computations, but it suffices to choose a single sequential composition of controllers. Only the externally observable continuous physical dynamics of multiple components evolves in parallel. As a result, parallel component composition (see Def. 6 later) arranges controllers sequentially in arbitrary order, followed by parallel composition of continuous dynamics in differential equations, followed by communication programs transferring values between connected ports (i. e., communication takes place at a discrete point in time).

Ports are connected by composing components, which requires that their contracts are *compatible* since safety can only follow compositionally if all components are connected in ways that do not violate their assumptions about one another. An input port and an output port are compatible, if the output port's guarantees imply the input port's assumptions and the dimensions of vector-valued ports match. Fig. 2 shows compatible and non-compatible ports when composing two components through their interfaces. If the output guarantees π^{out} are at least as strong as the input assumptions π^{in}, i. e., the implication $\pi^{out} \rightarrow \pi^{in}$ is valid, then a value emitted from the output port will always fulfill the input assumption of the input port and can thus be received on the input port. For example, an output port that guarantees $\pi^{out} \equiv x \leq 5$ can be connected to an input port requiring $\pi^{in} \equiv x \leq 10$, but not to an input port requiring $x \leq 2$ because that requirement is not guaranteed to be met when $x \leq 5$.

Not all ports of a component need to be connected to the ports of a single other component. Ports may remain unconnected after composition and might be connected to other components later on.

Formalizing Contracts. A contract is a formally verified agreement about the behavior of a component, including its input and output behavior as described by its interface. It specifies the assumptions under which a component may be used, as well as the guarantees it warrants under such correct use [3,42]. We formalize contracts in dL with safety properties of the form

$$\phi \rightarrow [\alpha]\psi$$

where ϕ is an initial state description, α is a hybrid program of either a single component or a system that is already composed of other components, and ψ is a postcondition that must hold after all runs of the hybrid program α.

In a monolithic system, interactions between subsystems are baked into the model itself. When building systems from components, however, the isolated components and their contracts abstract from interaction. Thus, the local component contracts must include interaction properties that capture assumptions and guarantees about the communication and interaction with other components. For example, a contract might restrict a vehicle's movement to prevent it from moving too fast, or specify the acceptable degree of input sensor uncertainty. The following contract restricts the distance between position x of a vehicle moving with maximum speed S for a duration of t and its previous position x^-:

$$\begin{aligned} & S \geq 0 \land x = x^- \land t = 0 \rightarrow \\ & \qquad [x^- := x; \{x' = S, t' = 1\}] \left| x - x^- \right| \leq t \cdot S \end{aligned}$$

In hybrid system verification, whose complexity heavily depends on the dimension of the analyzed system and the fidelity of differential equation models, it is beneficial to reduce complexity by abstracting from the internal behavior of components [26], see Fig. 3.

Global contracts restrict values to *globally known (symbolic) regions.* For instance, a robot might be confined in a known, fixed area, e. g., the robot's position x must always be in a fixed range R describing a room (e. g., $-R \leq x \leq R$, where $R \in \mathbb{R}$ is a fixed global design constant).

Change contracts restrict the *magnitude of change* regardless of how much time passed between measurements (e. g., relate a previously communicated value with the current value). For instance, a robot might guarantee to stay close to its previous position x^- (e. g., $-R \leq x - x^- \leq R$).

Rate contracts restrict the *rate of change* by keeping track of time. For instance, a robot may guarantee to change its position according to its speed s, so the current position x and the previous position x^- are related by duration t (e. g., $-s \cdot t \leq x - x^- \leq s \cdot t$).

Communication and interaction between components can be subject to information loss and delay [26], see Fig. 4.

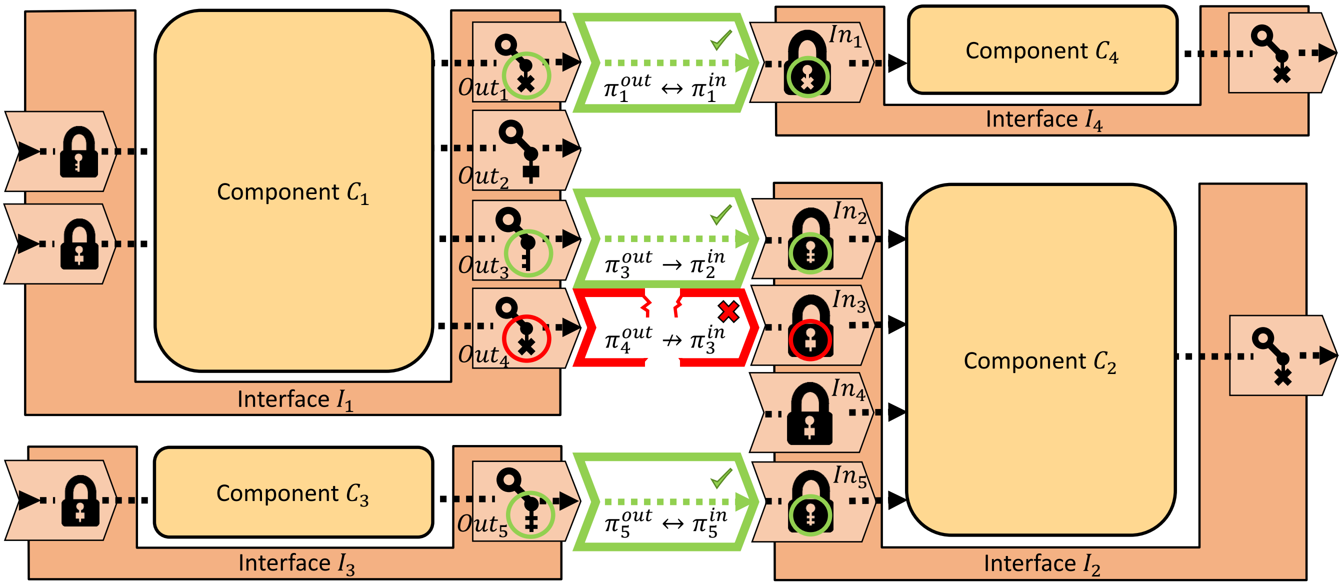

Figure 2: Composition: The output port Out_3 in the interface I_1 of component C_1 is compatible with the input port In_2 in the interface I_2 of component C_2, since $\pi_3^{out} \rightarrow \pi_2^{in}$, even though it is not a perfect match. Ports Out_4 and In_3 are not compatible and thus, a connection is not allowed between these ports. Output port Out_2 and input port In_4 are not connected yet and remain open. Input port In_5 of I_2 can be connected to a different component's interface I_3, as long as the respective ports In_5 and Out_5 are compatible. Similarly, output port Out_1 of I_1 can be connected to a different component's interface I_4, as long as the respective ports Out_1 and In_1 are compatible.

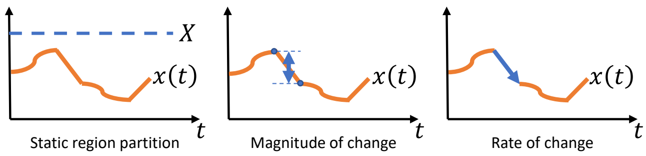

Figure 3: Abstractions from continuous dynamics with contracts [26].

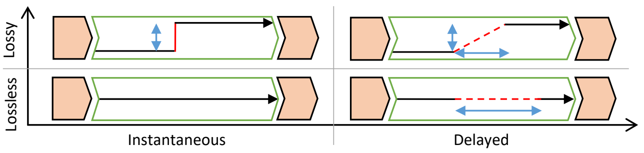

Figure 4: Dimensions of Communication: Communication can be subject to information loss and delay [26].

Lossy communication is used when real-world environments ports are subject to information loss or sensor uncertainty and thus, provide slightly off approximations of the actual values. Even though the actual error might be unknown, maximum error bounds might be available (e. g., according to a sensor specification). Lossy communication, for instance modeled as $\lambda := *;\ ?\,|\lambda| \leq \Lambda;\ \hat{x} := x + \lambda$, uses a nondeterministically chosen error value λ ($\lambda := *$ nondeterministically assigns any real value to λ), bounded by the maximum error Λ (the test $?\,|\lambda| \leq \Lambda$ ensures that the value of λ is between $-\Lambda$ and Λ), which distorts the communicated value $\hat{x}$.

Delayed communication results in an accumulated error, e. g., when a distance sensor in a car reports slightly outdated distances the error to the true distance grows with speed and measurement delay.

Instantaneous, lossless communication is often used as a first approximation of sensing and communication and can be modeled by a direct assignment of variables $\hat{x} := x$, i. e., the true position x is passed on to the measured position $\hat{x}$. If the system is not safe under perfect knowledge it is never safe.

In this article, we define change and rate contracts supporting generic (e. g., lossless and lossy) communication according to these categories.

3.3 Compose Verified Components to Verified Systems

We adapt the steps of our prior decomposition-focused approach for component-based verification with global contracts [25] to system composition from verified components, see Fig. 5. After the appropriate components and interfaces are identified (1), system initial conditions and safety property can be derived from the respective initial conditions and guarantees of the components (2). Identification of suitable components, interfaces and local safety properties is a crucial design task; automa-

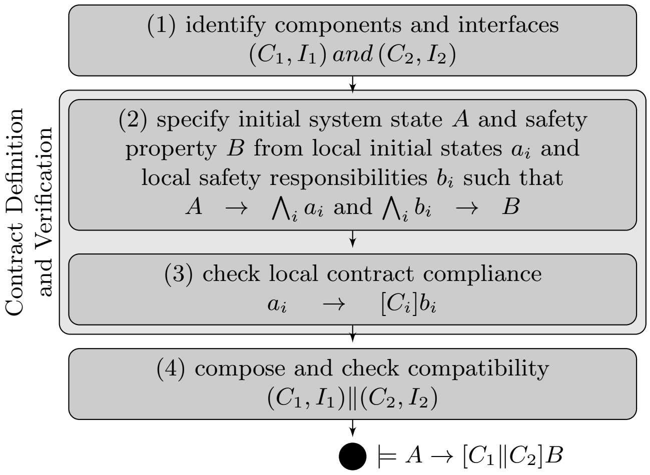
(1) identify components and interfaces
$(C_1, I_1)\, and\, (C_2, I_2)$
Contract Definition and Verification
(2) specify initial system state A and safety property B from local initial states a_i and local safety responsibilities b_i such that
$A \rightarrow \bigwedge_i a_i$ and $\bigwedge_i b_i \rightarrow B$
(3) check local contract compliance
$a_i \rightarrow [C_i]b_i$
(4) compose and check compatibility
$(C_1, I_1) \| (C_2, I_2)$
$\models A \rightarrow [C_1 \| C_2]B$

Figure 5: Verified system composition from verified components

tion support for it is not our focus here. Each interface comes with a contract and a contract compliance proof (3) witnessing that the component alone complies with its interface contract. Finally, the system safety proof is constructed from the individual contract compliance proofs; it also discharges the compatibility proof obligations generated upon composition (4). In Section 5 we present a tactic to construct such a proof automatically.

The main result of this process is that the component proofs—performed for compatible components in isolation—transfer to safety of an arbitrarily large system built by instantiating and composing these components. This enables the safe composition of safe components, where compatible proofs will be constructed (by a tactic) as evidence for the composition meeting the required compatibility conditions.

4 Hybrid Components with Change and Rate Contracts

In this section, we formalize the notion of *components* as hybrid programs and define their *interfaces* as dL formulas, which identify assumptions about component inputs and guarantees about component outputs phrased in terms of magnitude and rate of change. We define what it means for a component to *comply with its contract* by a dL formula expressing local safety responsibilities and compliance with its interface. We also define the *compatibility of component connections* rigorously as dL formulas. These notions make it possible to give meaning to and prove rigorously what safety responsibility and compatibility of a component really means and rigorously prove safety of the composition.

The main result of this section is a proof showing that contract compliance of components and connection compatibility ensure system safety: Users only provide a *specification* of components, interfaces, and how the components are connected, and verify *proof obligations* about individual component contract compliance and compatibility; safety of the whole system follows automatically from these component verification results.

4.1 Running Example: Tele-Operated Robot with Collision Avoidance

To illustrate the concepts, we use a running example of a tele-operated robot with collision avoidance inspired by [21], see Fig. 6. The overall system objective is to keep the robot from actively colliding with an obstacle. The system consists of three components:

1. The *remote control* (RC) component occasionally issues a new speed advisory d on its output port.
2. The *obstacle* component moves with arbitrary speed s_o limited to at most S and sends its current position p_o on its single output port. Obstacles include both stationary elements (e. g., a wall with $S = 0$) or moving entities (e. g., a person).
3. The *robot* component reads speed advice from the remote control component on input port $\hat{d}$ and follows that speed advice if the obstacle position measured on input port $\hat{p}_o$ is at a safe distance.

The system safety property (no collision while driving) can be expressed as $\psi_{sys}^{safe} \equiv s_r > 0 \rightarrow p_o \neq p_r$. Two consecutive speed advisories from the RC should require a speed change of at most D (i. e., $|d - d^-| \leq D$). The RC issues speed advice to the robot, but has no physical dynamics. The obstacle chooses a new nonnegative speed but at most S and moves according to its plant. The robot measures the obstacle's position. If the distance is safe, the robot chooses the speed suggested by the RC; otherwise, the robot stops.

Formal definitions of these three components, their interfaces, and their contracts, will be introduced step by step as a running example along the definitions in subsequent sections.

4.2 Specification: Components and Interfaces

Components and interfaces specify what a component assumes about the magnitude and rate of change at each of its inputs, and what it guarantees about the magnitude and rate of change of its outputs. To make such conditions expressible, every component will use additional port memory variables to store both the current and the previous value communicated along a port. These variables can be used to model jumps in discrete control, and for discrete-time measurements of continuous physical behavior.

Naming conventions. We use variable names x to refer to the internal state and output port of a component (if it has x as output), x^- or explicitly $\text{old}(x)$ to refer to

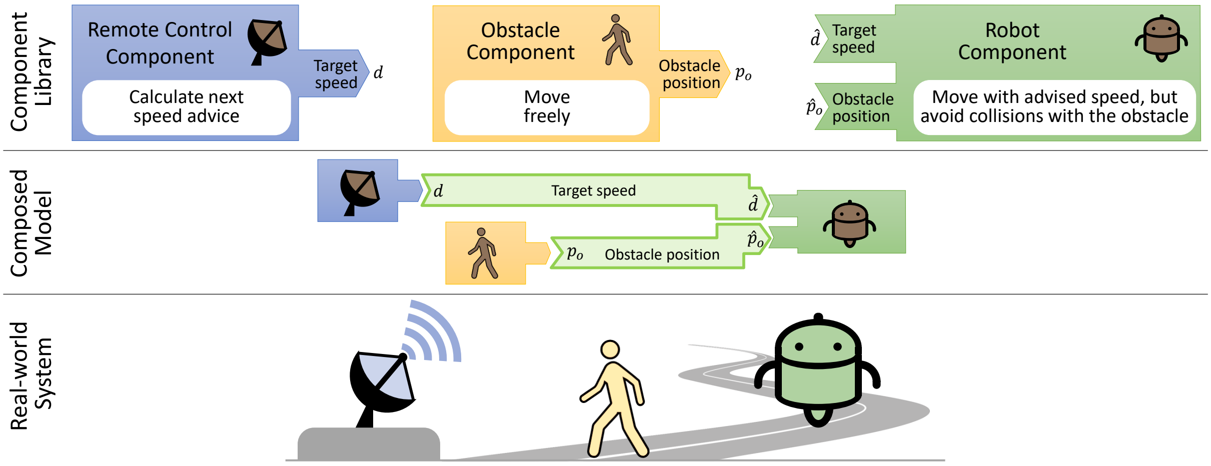

Figure 6: Running Example: Robot receives speed advice and obstacle position, and has to avoid crashes.

the previous value received last before the current value x. We use $\hat{x}$ to disambiguate an input port that reads x from another component.

4.2.1 Components

Components consist of discrete control computations and a continuous plant, as will be defined in Def. 1. The control computations and plant are composed with inputs and outputs to a hybrid program describing the entire component behavior later in Def. 5. To build systems with arbitrarily many components by nested binary composition, we compose components hierarchically from sub-components, so components include glue code for the internally connected ports of sub-components.

Definition 1 (Component). A component

$$\mathrm{C} = (\mathit{ctrl}, \mathit{plant}, \mathit{cp})$$

consists of the following:

- *ctrl* are the discrete computations of the component without differential equations,
- *plant* is a differential equation with an evolution domain constraint Q ($x_1' = \theta_1, \dots, x_n' = \theta_n \& Q$) for $n \in \mathbb{N}$,
- *cp* is the internal glue code connecting ports of nested sub-components,
- $V(C) \stackrel{\text{def}}{\equiv} V(\mathit{ctrl}) \cup V(\mathit{plant}) \cup V(\mathit{cp})$,
- $\mathrm{BV}(C) \stackrel{\text{def}}{\equiv} \mathrm{BV}(\mathit{ctrl}) \cup \mathrm{BV}(\mathit{plant}) \cup \mathrm{BV}(\mathit{cp})$, and
- $\mathrm{FV}(C) \stackrel{\text{def}}{\equiv} \mathrm{FV}(\mathit{ctrl}) \cup \mathrm{FV}(\mathit{plant}) \cup \mathrm{FV}(\mathit{cp})$.

The content of *cp* depends on the type of composition used for the sub-components (e. g., lossless composition, lossy composition) and will be detailed in Section 4.4.2. For example, base components have $\mathit{cp} \equiv \texttt{skip}$ (statement of no effect), whereas components with lossless instantaneous composition have a list of assignments $\hat{x} := x$ from output port x to the input port $\hat{x}$ that it is connected to (e. g., $\hat{d} := d; \hat{p}_o := p_o$ for Fig. 6). The variables of a component are the variables of its controller, plant, and all its sub-components. In order to get components that can be analyzed in isolation and arranged in arbitrary sequential order by Lemma 1, *components cannot share variables and must communicate solely through ports.* Otherwise, they break component abstraction. To make up for this restriction, global shared constants (read-only and thus not used for communication purposes) are included for convenience to share common knowledge about system parameters among all components in a single place.

Definition 2 (Global constants). Global constants V^{global} shared among components C_i are read-only, i. e., $V^{global} \cap \mathrm{BV}(C_i) = \emptyset$. No other variables are shared, i. e., $V(\mathrm{C}_i) \cap V(\mathrm{C}_j) \subseteq V^{global}$ for components $\mathrm{C}_i \neq \mathrm{C}_j$.

Assumptions about system parameters are available to any component and invariant throughout the system execution, since they mention only global constants.

4.2.2 Example: Components

Consider the robot collision avoidance system. Its global variables $V^{global} = \{S, D\}$ are the maximum obstacle speed S and the maximum difference D between two speed advisories, since they are not changed by any component. Both are non-negative ($S \geq 0 \wedge D \geq 0$).

Example 1 RC Component

$$\mathrm{C}_{rc} = (\underbrace{d := *;\ ?\left|d - d^-\right| \leq D}_{\mathit{ctrl}_{rc}},\ \underbrace{\texttt{skip}}_{\mathit{plant}_{rc}},\ \underbrace{\texttt{skip}}_{\mathit{cp}_{rc}})$$

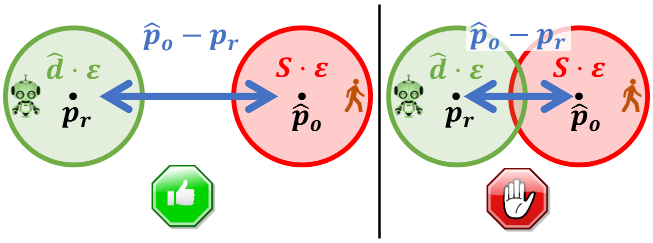

Figure 7: The robot only accepts speed advice if it is safe: The left green circle represents the area that the robot might reach until the next controller run (i. e., within ε time units) with the received speed advice $\hat{d}$. The right red circle represents the area that the obstacle might reach during the same interval ε with maximum speed S.

Example 1 describes the RC component. Its controller $ctrl_{rc}$ picks a new speed advice and ensures that it is not too far from the previous speed advice to avoid sudden spikes in speed. Since the RC is an atomic component without physical dynamics, $plant_{rc}$ and cp_{rc} are empty.

The obstacle component C_o in Example 2 moves with arbitrary but limited speed (systems with infinite speed obstacles are inherently unsafe), so the obstacle controller $ctrl_o$ chooses any new non-negative speed s_o limited by the maximum speed S. The obstacle plant adapts the obstacle position according to the chosen speed (i. e., the obstacle moves along ODE $p_o' = s_o$). The internally connected ports cp_o are empty, since the obstacle is an atomic component.

Example 2 Obstacle Component

$$\mathrm{C}_o = (\underbrace{s_o := *;\ ?\big(0 \le s_o \le S\big)}_{ctrl_o},\ \underbrace{p_o' = s_o}_{plant_o},\ \underbrace{\texttt{skip}}_{cp_o})$$

The robot component C_r in Example 3 should follow speed advice from the RC and measures the position of the obstacle to avoid collisions. It has a control cycle time of ε, which restricts how long the robot can drive without receiving control input. This ensures that the robot's controller runs regularly. The robot controller first chooses a new speed. If the obstacle is far enough away, i. e., the distance $(\hat{p}_o - p_r)$ between obstacle and robot is greater than the maximum distance $(S \cdot \varepsilon)$ that the obstacle can move, plus the maximum distance $(\hat{d} \cdot \varepsilon)$ the robot itself can move with the new desired speed, the robot follows the speed advice of the RC, see (2), (5) and Fig. 7. Otherwise, the robot stops to avoid imminent collision, as indicated by overlapping areas of motion in Fig. 7. The robot's plant (3) adapts the robot's position according to the chosen speed (i. e., the robot moves). The robot does not have internal connections, so $cp_r \equiv$ `skip` (4).

Example 3 Robot Component

$$\mathrm{C}_r = (ctrl_r, plant_r, cp_r)$$

$$ctrl_r \equiv \text{if}\,(\text{far})\ s_r := \hat{d}\ \text{else}\ s_r := 0 \tag{2}$$

$$plant_r \equiv p_r' = s_r\ \&\ t - t^- \le \varepsilon \tag{3}$$

$$cp_r \equiv \texttt{skip} \tag{4}$$

$$\text{far} \equiv \hat{p}_o - p_r > (\hat{d} + S) \cdot \varepsilon \tag{5}$$

4.2.3 Time and Rate of Change

In a combined ODE $p_r' = s_r, p_o' = s_o$ both objects move for the same duration. But the point of components is to decompose for the sake of reducing complexity, at which point the now separate ODEs $p_r' = s_r$ and $p_o' = s_o$ in the respective components loose synchronization in time.

From the viewpoint of a single component, all other plants reduce to discrete abstractions through input assumptions on ports, which is an important step to reduce verification complexity. These input assumptions are phrased in terms of worst-case behavior (e. g., from the viewpoint of the robot, the obstacle may "jump" at most distance $S \cdot \varepsilon$ between measurements because it lost a precise model of obstacle motion). If the robot's ODE (nondeterministically) runs for a shorter amount of time, the measurements and the continuous behavior of the robot drift as robot and obstacle think they move for different durations. To prevent this, we introduce *rate contracts* as a way of ensuring that changes are consistent with the actual time that passes in a component.

To unify the timing for all components of a system, we introduce a globally synchronized time t to measure the duration $t - t^-$ of plant executions. Both t and t^- are special global variables, which cannot be changed by the user, see Def. 3, but only change at designated locations in the composition infrastructure as we will see later.

Definition 3 (System time). System time t changes with constant rate $t' = 1$ from plant start time t^- to measure plant duration $t - t^-$ and can be read but not written by components C_i, i. e., $\{t, t^-\} \cap BV(\mathrm{C}_i) = \emptyset$.

System time enables interfaces to specify the assumed rate of change on input ports and guaranteed rate of change on output ports.

4.2.4 Interfaces

An interface defines how a component may interact with other components through its ports, what assumptions the component makes about its inputs, and what guarantees it provides for its outputs, see Def. 4. It defines what other components can rely on when using its outputs but also lists requirements on inputs. Similar to combining controllers in arbitrary sequential order, we want to read from input ports in any arbitrary order. Hence, input assumptions are local to their port, i. e., no input formula can mention input variables of other ports. To support change and rate contracts, we introduce a port memory to recall previous port values in input assumptions and output guarantees. We want to guarantee safety for recursive components in a uniform way with safety for systems composed from multiple components, and, therefore, prevent leaking information outside their official interfaces by requiring that input formulas do not mention output variables.

Definition 4 (Interface). An *interface* I for a component C is a tuple

$$\mathrm{I} = \left(V^{in}, \pi^{in}, V^{out}, \pi^{out}, V^{-}\right)$$

with

- read-only input port variables $V^{in} \subseteq V(\mathrm{C})$ with $V^{in} \cap \mathrm{BV}(\mathrm{C}) = \emptyset$ and disjoint writable output port variables $V^{out} \subseteq V(\mathrm{C})$, i. e., $V^{in} \cap V^{out} = \emptyset$,
- satisfiable input assumptions $\pi^{in} : V^{in} \to \mathsf{dL}$ disjoint across ports: $V(\pi^{in}(v)) \subseteq \left(V(\mathrm{C}) \setminus \left(V^{in} \cup V^{out}\right)\right) \cup \{v\}$ for all $v \in V^{in}$,
- output guarantees $\pi^{out} : V^{out} \to \mathsf{dL}$,
- $V^{-} \subseteq V(\mathrm{C})$ with $V^{-} \cap \mathrm{BV}(\mathrm{C}) = \emptyset$ are read-only port memory variables storing the previous values of some ports, disjoint from other interface variables $V^{-} \cap (V^{in} \cup V^{out}) = \emptyset$; we use the notational convention x^{-} to refer to the port memory of variable x or explicitly $\mathrm{old}(x)$.

The definition is accordingly for vector-valued ports that share multiple variables along a single port, provided that each variable is part of at most one vectorial port for proper data abstraction.

4.2.5 Example: Interfaces

We continue the remote-controlled robot example from Fig. 6 by defining interfaces for the three components: the RC interface I_{rc}, the obstacle interface I_o, and the robot interface I_r. Recall that the robot's collision avoidance also assumes the remote control to not request sudden speed changes and that the obstacle does not move too fast.

Example 4 Remote Control Interface

$$\mathrm{I}_{rc} = \left(V^{in}_{rc}, \pi^{in}_{rc}, V^{out}_{rc}, \pi^{out}_{rc}, V^{-}_{rc}\right)$$

$$V^{in}_{rc} = \{\} \tag{6}$$

$$\pi^{in}_{rc} = () \tag{7}$$

$$V^{out}_{rc} = \{d\} \tag{8}$$

$$\pi^{out}_{rc} = \left(d \mapsto \left|d - d^{-}\right| \leq D\right) \tag{9}$$

$$V^{-}_{rc} = \{d^{-}\} \tag{10}$$

Example 5 Obstacle Interface

$$\mathrm{I}_{o} = \left(V^{in}_{o}, \pi^{in}_{o}, V^{out}_{o}, \pi^{out}_{o}, V^{-}_{o}\right)$$

$$V^{in}_{o} = \{\} \tag{11}$$

$$\pi^{in}_{o} = () \tag{12}$$

$$V^{out}_{o} = \{p_o\} \tag{13}$$

$$\pi^{out}_{o} = \left(p_o \mapsto \left|p_o - p_o^{-}\right| \leq S \cdot \left(t - t^{-}\right)\right) \tag{14}$$

$$V^{-}_{o} = \{p_o^{-}\} \tag{15}$$

The RC interface in Example 4 has no input ports, so V^{in}_{rc} (6) and π^{in}_{rc} (7) are empty. The single output port d (8) provides the current speed advice, which, by output guarantee π^{out}_{rc} (9), is never further away than D from the previous advice d^{-} (10).

The obstacle interface in Example 5 has no input ports, see (11)–(12). The single output port provides the current obstacle position p_o (13), where output guarantee π^{out}_{o} (14) restricts the position to an interval of size $S \cdot (t - t^{-})$ centered at the obstacle's previous position p_o^{-} (15). This captures the rate of change between the previous value p_o^{-} and the current value p_o, tied together by plant duration $t - t^{-}$.

The robot interface in Example 6 specifies two input ports (16)-(17). On input port $\hat{d}$ it receives a speed advice, which is assumed to be close to the previous value $\hat{d}^{-}$, so describes the magnitude of change in speed advice. On input port $\hat{p}_o$ it receives the obstacle's current position, which is assumed to be close to the obstacle's previous position $\hat{p}_o^{-}$. This input assumption describes the rate of change of the obstacle position. Thus, a global contract as in [25] would not suffice here. The robot has no output ports, see (18)-(19). The previous values $\hat{p}_o^{-}$ and $\hat{d}^{-}$ (20) of both input ports are stored for access in the contract.

Example 6 Robot Interface

$$\mathrm{I}_{r} = \left(V^{in}_{r}, \pi^{in}_{r}, V^{out}_{r}, \pi^{out}_{r}, V^{-}_{r}\right)$$

$$V^{in}_{r} = \{\hat{p}_o, \hat{d}\} \tag{16}$$

$$\begin{aligned}\pi^{in}_{r} = \Big(&\hat{p}_o \mapsto \left|\hat{p}_o - \hat{p}_o^{-}\right| \leq S \cdot \left(t - t^{-}\right), \\ &\hat{d} \mapsto \left|\hat{d} - \hat{d}^{-}\right| \leq D\Big)\end{aligned} \tag{17}$$

$$V^{out}_{r} = \{\} \tag{18}$$

$$\pi^{out}_{r} = () \tag{19}$$

$$V^{-}_{r} = \{\hat{p}_o^{-}, \hat{d}^{-}\} \tag{20}$$

In summary, the remote control is responsible for asking only manageable speed changes and the obstacle is responsible for changing its position not too suddenly. The robot will rely on these guarantees to ensure in turn that it does not drive too close to the obstacle. Interfaces are a powerful tool to separate responsibilities. Next we guarantee that components also deliver on these promises.

4.3 Proof Obligations: Change and Rate Contract

Contract compliance ties together components and interfaces by showing that a component guarantees the output changes that its interface specifies under the input assumptions made in the interface. Contract compliance additionally establishes a component's responsibilities with respect to how it contributes to system safety.

For example, the safety responsibilities of a robot might require that the robot will not drive too close to the last measurement of the position of the obstacle. Together with the obstacle's output guarantee of not moving too far from its previous position, the local safety responsibilities imply a system-wide safety property (e. g., robot and obstacle will not collide), since we know that a measurement previously reflected the real position. The subtle but important consequence of our composition and contract compliance notion is that the components are *locally responsible with respect to their inputs* (e. g., the robot is locally responsible with respect to position measurements), but *system safety follows for the true values* (e. g., true robot and obstacle position do not coincide).

In order to make guarantees about the behavior of a composed system, we use the system time t to measure the duration $(t - t^-)$ between controller runs in rate contract compliance proof obligations.

Definition 5 (Contract compliance). Let C be a component with its interface I (Def. 4), including output guarantees $\Pi^{out} \equiv \bigwedge_{v \in V^{out}} \pi^{out}(v)$. Let formula ϕ describe initial states and formula ψ^{safe} local safety responsibilities of C, both over the component variables $V(\mathrm{C})$. Formula Ω with $\mathrm{V}(\Omega) \subseteq V^{global}$ specifies facts about design parameters of the system. We abbreviate (vectorial) nondeterministic assignments to input ports satisfying input assumptions $\pi^{in}(v)$

$$\mathit{in} \overset{\text{def}}{\equiv} \big(v := *; ?\pi^{in}(v)\big) \text{ sequentially for all } v \in V^{in} \quad ,$$

and (vectorial) assignments storing previous values of port variables in port memory:

$$\Delta \overset{\text{def}}{\equiv} v^- := v \text{ sequentially for all } v^- \in V^- \quad .$$

*Change contract compliance CCC(*C, I*)* of C with I is defined as validity of the dL formula:

$$\mathit{CCC}(\mathrm{C}, \mathrm{I}) \overset{\text{def}}{\equiv}$$
$$(\Omega \wedge \phi) \rightarrow [(\Delta; \mathit{ctrl}; \mathit{plant}; \mathit{in}; \mathit{cp})^*]\big(\psi^{safe} \wedge \Pi^{out}\big) \quad .$$

*Rate contract compliance RCC(*C, I*)* is defined as validity of the dL formula:

$$\mathit{RCC}(\mathrm{C}, \mathrm{I}) \overset{\text{def}}{\equiv} \big(t = t^- \wedge \Omega \wedge \phi\big) \rightarrow$$
$$\big[\big(\Delta; \mathit{ctrl}; t^- := t;$$
$$\{t' = 1, \mathit{plant}\}; \mathit{in}; \mathit{cp}\big)^*\big] \big(\psi^{safe} \wedge \Pi^{out}\big) \quad .$$

Contract compliance can be verified using KeYmaera X [10].

The order of the assignments in both *in* and Δ is irrelevant because the assignments are over disjoint variables and $\pi^{in}(v)$ are local to their port per Def. 4. The variables v^- can be used in a component's *ctrl* and *plant* to access the initial values of ports, e. g., while the variable $v_i \in V^{in}$ holds the newly transmitted value of a port, v_i^- can be used to access its previous value.

In this notion of contracts, input ports are read at the end of the component, after the run of *plant*. While reading from input ports at the beginning of a component's loop body (i. e., before the controller runs, e. g., as in [25]) may seem intuitive, it would require severe restrictions to a component's plant in order to make inputs and plant agree on duration. Thus, we prepare the next loop iteration at the end of the loop body (i. e., after *plant*), so that the actual plant duration can be considered for computing the next input values.

4.3.1 Example: Contract Compliance

We continue the collision avoidance example by proving contract compliance for the remote control, obstacle, and robot component. The remote control from Fig. 6 issues speed advice in a purely discrete manner and therefore a change contract according to Def. 5 is sufficient to relate the current advice d to the previous advice d^-. The precondition (21) for the RC bootstraps the output port's previous value d^- from the current speed advice d. Here, ψ_{rc} comprises only the output guarantees (22) of the RC, since the RC has no local safety responsibilities. In summary, the RC guarantees that consecutive speed advice are at most D apart. The bounds on $D \geq 0$ are specified globally for all components.

$$\phi_{rc} \equiv d = d^- \tag{21}$$
$$\psi_{rc} \equiv \big|d - d^-\big| \leq D \tag{22}$$

The resulting *change contract* per Def. 5 for the RC was verified using KeYmaera X:

$$(D \geq 0 \wedge \phi_{rc}) \rightarrow [(\overbrace{d^- := d;}^{\Delta_{rc}} \overbrace{d := *; ?\,\big|d - d^-\big| \leq D;}^{\mathit{ctrl}_{rc}} \overbrace{\texttt{skip}}^{\mathit{plant}_{rc}};$$
$$\underbrace{\texttt{skip}}_{\mathit{in}_{rc}}; \underbrace{\texttt{skip}}_{\mathit{cp}_{rc}})^*]\psi_{rc}$$

We thus know that the component is safe and complies with its interface. Compared to contracts with fixed ranges as in approaches [4,25], we do not have to assume a global limit for speed advice d, but consider the previous advice d^- as a reference value when calculating the next speed advice.

Obstacles move and, therefore, obstacle positions p_o are related by how much time passes in the obstacle's ODE $plant_o \equiv p_o' = s_o$. Hence, we follow Def. 5 to abstract the obstacle's motion to its rate of change in position (24). The precondition ϕ_o (23) for the obstacle bootstraps the output port's previous value p_o^- from the position p_o and initializes the obstacle speed to 0, which is also subject to a maximum speed system parameter $S \geq 0$. Formula ψ_o (24) gives only the output port guarantees of the obstacle, since our liberal notion of obstacles should not assume obstacles to cooperate for safety. Such an abstraction can be found by solving the plant ODE or from differential invariants [34].

$$\phi_o \equiv p_o = p_o^- \wedge s_o = 0 \tag{23}$$

$$\psi_o \equiv \left|p_o - p_o^-\right| \leq S \cdot \left(t - t^-\right) \tag{24}$$

The resulting *rate contract* per Def. 5 for the obstacle was verified using KeYmaera X:

$$\begin{aligned}&(t = t^- \wedge S \geq 0 \wedge \phi_o) \rightarrow\\ &\quad [\overbrace{(p_o^- := p_o}^{\Delta_o}; \overbrace{s_o := *; ?(0 \leq s_o \leq S)}^{ctrl_o};\\ &\quad\quad t^- := t; \{t' = 1, \underbrace{p_o' = s_o}_{plant_o}\}; \underbrace{\texttt{skip}}_{in_o}; \underbrace{\texttt{skip}}_{cp_o})^*]\psi_o\end{aligned}$$

The proof guarantees that the obstacle moves at most distance $S \cdot (t - t^-)$ between measurements p_o and p_o^- taken $(t - t^-)$ apart.

Finally, we turn to the rate contract of the robot. The precondition (25) bootstraps the input ports' previous values $\hat{p}_o^-$ and $\hat{d}^-$ from $\hat{p}_o$ and $\hat{d}$, initializes the robot's speed to 0 and ensures a positive control cycle time (i. e. maximum *plant* runtime ε). The robot also gets to assume the system parameter bounds $S \geq 0$ and $D \geq 0$. The robot guarantees ψ_r (26) that its own position and the measured obstacle position never coincide, unless the robot is stopped.

$$\phi_r \equiv \hat{p}_o = \hat{p}_o^- \wedge \hat{d} = \hat{d}^- \wedge s_r = 0 \wedge \varepsilon > 0 \tag{25}$$

$$\psi_r \equiv s_r > 0 \rightarrow \hat{p}_o \neq p_r \tag{26}$$

The resulting *rate contract* per Def. 5 for the robot was verified using KeYmaera X:

$$\begin{aligned}&(t = t^- \wedge S \geq 0 \wedge D \geq 0 \wedge \phi_r) \rightarrow [\overbrace{(\hat{p}_o^- := \hat{p}_o; \hat{d}^- := \hat{d}}^{\Delta_r};\\ &\overbrace{\text{if (far) } s_r := \hat{d} \text{ else } s_r := 0}^{ctrl_r}; t^- := t; \{t' = 1, \overbrace{p_r' = s_r}^{plant_r}\};\\ &\underbrace{\hat{p}_o := *; ?\pi_r^{in}(\hat{p}_o); \hat{d} := *; ?\pi_r^{in}(\hat{d})}_{in_r}; \underbrace{\texttt{skip}}_{cp_r})^*]\psi_r\end{aligned}$$

The proof guarantees that the robot does not drive too close to the measured obstacle position.

4.4 Proof Obligations: Compatible Composition

From components with verified contract compliance, we now compose systems such that the safety of the composed system can be guaranteed from the safety results about the individual components. Not all naive compositions of components would be safe. But we show that those that respect the interface compatibilities are.

4.4.1 Parallel Composition of Components

Parallel composition of components requires parallel composition of their controllers as well as of their plants, and connections between their ports, see Section 3.2. Unlike plants, which are ODEs and have a native parallel composition operator in dL, truly parallel composition of controllers would require enumerating all possible interleavings of controller statements and result in a potentially vast proof effort. Instead, we rely on the strict variable separation between components to introduce a quasi-parallel composition: the discrete *ctrl* computations of the components are executed sequentially, while the continuous *plant* dynamics run in parallel. Which exact sequential execution order of *ctrl* blocks is chosen is irrelevant by Lemma 1, since the *ctrl* computations of different components are independent according to Def. 2 (i. e., programs having disjoint free and bound variables) and the communication between components happens *after* all their combined *ctrl* computations. Similarly, the internally connected ports *cp* of the components are independent and thus composed sequentially in any order.

Such a definition is natural in dL, since time only passes during continuous evolution in hybrid programs, while the discrete actions of a program do not consume time and, thus, happen instantaneously at a single real point in time, but in a specific order.

Fundamental ingredients for parallel composition of two components are their connections that specify how output values from one component are passed on to inputs of the other component.

The connections $\mathcal{X}$ are parametric in a communication program *con* that defines how values are passed between connected ports. For example, an instantaneous, lossless communication can be modeled with a deterministic assignment $x := \mathcal{X}(x)$ directly transferring the value of the source output port $\mathcal{X}(x)$ to its connected input port x. Parallel composition uses connections $\mathcal{X}$ to wire components and merge unconnected ports into a composed interface. Connections must satisfy the following conditions.

Remark 1 (Connections). Connections $\mathcal{X}$

$$\mathcal{X} : \left(V_1^{in} \cup V_2^{in}\right) \rightharpoonup \left(V_1^{out} \cup V_2^{out}\right) \ ,$$

provided $\mathcal{X}(v) \notin V_i^{out}$ for all $v \in V_i^{in}$, are specified with a partial (i. e., not every input must be mapped), injective (i. e., every output is only mapped to at most one input) function, connecting some inputs to some outputs, with domain $\mathcal{I}^{\mathcal{X}} = \{x \in V_i^{in} \mid \mathcal{X}(x) \text{ is defined}\}$ and image $\mathcal{O}^{\mathcal{X}} = \{y \in V_i^{out} \mid y = \mathcal{X}(x) \text{ for some } x \in V_j^{in}\}$. The connection program $con : \mathcal{I}^{\mathcal{X}} \to HP$ models the connection using a discrete program without ODEs per connected input port to read values from the connected source output port, such that

- each $con(v)$ sets only its input port, for all $v \in \mathcal{I}^{\mathcal{X}}$ and all C_i: $\mathrm{BV}(con(v)) \cap (V(\mathrm{C}_i) \cup V^{global}) = \{v\}$,
- each $con(v)$ only reads the connected ports and global variables: $(V(con(v)) \cap V(C_i)) \subseteq (V^{global} \cup \{v, \mathcal{X}(v)\})$ for all $v \in \mathcal{I}^{\mathcal{X}}$ and C_i ,
- connections bind disjoint inputs, for all $v_k \neq v_l \in \mathcal{I}^{\mathcal{X}}$: $\mathrm{BV}(con(v_k)) \cap \mathrm{BV}(con(v_l)) = \emptyset$.

Any communication program that satisfies the conditions in Remark 1 can be used for parallel composition per Def. 6 below.

Definition 6 (Parallel composition). Let

$$\mathrm{C}_i = (ctrl_i, plant_i, cp_i) \text{ for } i \in \{1, 2\}$$

denote two components with their interfaces

$$\mathrm{I}_i = \left(V_i^{in}, \pi_i^{in}, V_i^{out}, \pi_i^{out}, V_i^{-}\right) \quad ,$$

sharing only V^{global} and system time: $V(\mathrm{C}_1) \cap V(\mathrm{C}_2) \subseteq V^{global} \cup \{t, t^-\}$. The composition $(\mathrm{C}_1, \mathrm{I}_1) \underset{\mathcal{X}, con}{\|} (\mathrm{C}_2, \mathrm{I}_2)$ of two components and their interfaces according to connections $\mathcal{X}$ and communication programs con is defined as:

- sequential controllers $ctrl \equiv ctrl_1; ctrl_2$,
- parallel plants inside both evolution domains

$$\begin{aligned} plant \equiv\, & \overbrace{x_1^{(1)\prime} = \theta_1^{(1)}, \ldots, x_1^{(k)\prime} = \theta_1^{(k)}}^{\text{component } \mathrm{C}_1}, \ldots, \\ & \underbrace{x_2^{(1)\prime} = \theta_2^{(1)}, \ldots, x_2^{(m)\prime} = \theta_2^{(m)}}_{\text{component } \mathrm{C}_2} \;\&\; Q_1 \wedge Q_2 \quad , \end{aligned}$$

- connected ports $cp_1; cp_2$ are extended with new connections $con(v_k), \ldots, con(v_l)$ for $\{v_k, \ldots, v_l\} = \mathcal{I}^{\mathcal{X}}$

$$cp \overset{\text{def}}{\equiv} cp_1; cp_2; \underbrace{con(v_k); \ldots; con(v_l)}_{\text{newly connected inputs}} \quad ,$$

- previous values $V^- \overset{\text{def}}{\equiv} V_1^- \cup V_2^-$ are merged,
- unconnected inputs $V^{in} = \left(V_1^{in} \cup V_2^{in}\right) \setminus \mathcal{I}^{\mathcal{X}}$ and unconnected outputs $V^{out} = (V_1^{out} \cup V_2^{out}) \setminus \mathcal{O}^{\mathcal{X}}$ are merged and their assumptions/guarantees preserved

$$\pi^{in}(v) \equiv \begin{cases} \pi_1^{in}(v) & \text{if } v \in V_1^{in} \setminus \mathcal{I}^{\mathcal{X}} \\ \pi_2^{in}(v) & \text{if } v \in V_2^{in} \setminus \mathcal{I}^{\mathcal{X}} \end{cases}$$

$$\pi^{out}(v) \equiv \begin{cases} \pi_1^{out}(v) & \text{if } v \in V_1^{out} \setminus \mathcal{O}^{\mathcal{X}} \\ \pi_2^{out}(v) & \text{if } v \in V_2^{out} \setminus \mathcal{O}^{\mathcal{X}} \end{cases} \quad .$$

Note that by moving connected ports $x \in \mathcal{I}^{\mathcal{X}}$ from the composed V^{in} into the connected ports cp, the communication programs con replace the nondeterministic assignments to open inputs of Def. 5. The order of con is irrelevant because their bound variables are disjoint per Remark 1. A communication program con may introduce and bind new local variables, as long as they are not part of any other component. However, con cannot use differential equations, as time passes only in the plants of components. Merged π^{in} and π^{out} remain disjoint since V_i^-, V_i^{in} and V_i^{out} are disjoint between components by Def. 2. It follows that the set of variables of the composed component $V(\mathrm{C})$ is the union of both variable sets, i. e., $V(\mathrm{C}) = V(\mathrm{C}_1) \cup V(\mathrm{C}_2)$.

The user provides component specifications $(\mathrm{C}_i, \mathrm{I}_i)$, a communication function con to transfer values between connected ports, and connections $\mathcal{X}$ that define which output is connected to which input. The composed system of parallel components is defined syntactically in Def. 6.

Remark 2. Since $V^- = V_1^- \cup V_2^-$, the current and previous values of ports can still be used internally in the composed system, even when the ports are no longer exposed through its external interface.

Associativity and Commutativity. The above composition operation is commutative and associative, and can, thus, be lifted to any number of components.

Proposition 1 (Parallel composition is commutative). *Let C_i for $i \in \{1, 2\}$ be components with interfaces I_i, and let $\mathcal{X} : \mathcal{I} \to \mathcal{O}$ be connections with $\mathcal{O} \subseteq V_1^{\text{out}}$, $\mathcal{I} \subseteq V_2^{\text{in}}$. Then*

$$(C_1, I_1) \underset{\mathcal{X}, \text{con}}{\|} (C_2, I_2) \equiv (C_2, I_2) \underset{\mathcal{X}, \text{con}}{\|} (C_1, I_1)$$

Proof. We have to show that $ctrl$, $plant$ and cp, as well as port memory variables V^-, unconnected inputs (i. e., input ports V^{in} and input assumptions π^{in}), and unconnected outputs (i. e., output ports V^{out} and output guarantees π^{out}) are equal on both sides.

- Controllers $ctrl_1; ctrl_2 \equiv ctrl_2; ctrl_1$ are commutative because sequential composition “;” of *independent* (i. e., those that do not bind any variables the other component reads) hybrid programs is commutative by Lemma 1.
- $plant_1, plant_2 \equiv plant_2, plant_1$ because composition “,” of differential equations is commutative.
- For port connections $cp_1; cp_2; con \equiv cp_2; cp_1; con$, commutativity also follows by Lemma 1.
- Unions of variable sets are commutative by commutativity of set union $\cup$: previous values are merged $V^- \overset{\text{def}}{\equiv} V_1^- \cup V_2^-$.
- Sets of unconnected input ports are merged and commutative by commutativity of set union $\cup$, that is,

$V^{in} = \left(V_1^{in} \cup V_2^{in}\right) \setminus \mathcal{I}^{\mathcal{X}}$. Input assumptions are preserved and the order of merging is irrelevant.

$$\pi^{in}(v) \equiv \begin{cases} \pi_1^{in}(v) & \text{if } v \in V_1^{in} \setminus \mathcal{I}^{\mathcal{X}} \\ \pi_2^{in}(v) & \text{if } v \in V_2^{in} \setminus \mathcal{I}^{\mathcal{X}} \end{cases} .$$

Similarly for unconnected output ports. □

Proposition 2 (Parallel composition is associative). *Let C_i for $i \in \{1,2,3\}$ be components with interfaces I_i, and let $\mathcal{X} : \mathcal{I}^{\mathcal{X}} \mapsto \mathcal{O}^{\mathcal{X}}$ and $\mathcal{Y} : \mathcal{I}^{\mathcal{Y}} \mapsto \mathcal{O}^{\mathcal{Y}}$ be connections with $\mathcal{O}^{\mathcal{X}} \subseteq V_1^{\text{out}}$, $\mathcal{I}^{\mathcal{X}} \subseteq V_2^{\text{in}}$, $\mathcal{O}^{\mathcal{Y}} \subseteq V_2^{\text{out}}$ and $\mathcal{I}^{\mathcal{Y}} \subseteq V_3^{\text{in}}$. Then*

$$\left((C_1, I_1) \underset{\mathcal{X},\text{con}_{\mathcal{X}}}{\|} (C_2, I_2)\right) \underset{\mathcal{Y},\text{con}_{\mathcal{Y}}}{\|} (C_3, I_3) \equiv (C_1, I_1) \underset{\mathcal{X},\text{con}_{\mathcal{X}}}{\|} \left((C_2, I_2) \underset{\mathcal{Y},\text{con}_{\mathcal{Y}}}{\|} (C_3, I_3)\right)$$

Proof. We have to show that *ctrl*, *plant* and *cp*, port memory variables V^-, unconnected inputs (i. e., input ports V^{in} and input assumptions π^{in}), and unconnected outputs (i. e., output ports V^{out} and output guarantees π^{out}) are equivalent on both sides and satisfy Def. 6.

- $(ctrl_1; ctrl_2)\,; ctrl_3 \equiv ctrl_1; (ctrl_2; ctrl_3)$ because sequential composition ";" of hybrid programs is associative.
- $(plant_1, plant_2), plant_3 \equiv plant_1, (plant_2, plant_3)$ because composition "," of differential equations is associative.
- For port connections *cp*, we have to show that

  $$(cp_1; cp_2; con_{\mathcal{X}})\,; cp_3; con_{\mathcal{Y}} \equiv$$
  $$cp_1; (cp_2; cp_3; con_{\mathcal{Y}})\,; con_{\mathcal{X}}$$

  where

  $$con_{\mathcal{X}} \equiv con(v_k), \ldots, con(v_l) \text{ for } \{v_k, \ldots, v_l\} = \mathcal{I}^{\mathcal{X}}$$
  $$con_{\mathcal{Y}} \equiv con(v_r), \ldots, con(v_s) \text{ for } \{v_r, \ldots, v_s\} = \mathcal{I}^{\mathcal{Y}}$$

  represent the new port connections. Sequential composition ";" is associative, and since all communication programs *con* bind disjoint variables and do not read variables bound in other *con* (see Def. 6), their order is irrelevant and can be commuted by Lemma 1.
- Unions of variable sets are associative by associativity of set union $\cup$: $V^- \overset{\text{def}}{=} (V_1^- \cup V_2^-) \cup V_3^- = V_1^- \cup (V_2^- \cup V_3^-)$.
- Sets of unconnected input ports are merged and associative by associativity of set union $\cup$: $V^{in} = (V_1^{in} \cup V_2^{in} \cup V_3^{in}) \setminus \left(\mathcal{I}^{\mathcal{X}} \cup \mathcal{I}^{\mathcal{Y}}\right)$. Input assumptions are preserved and the order of merging is irrelevant.

  $$\pi^{in}(v) \equiv \begin{cases} \pi_1^{in}(v) & \text{if } v \in V_1^{in} \setminus \left(\mathcal{I}^{\mathcal{X}} \cup \mathcal{I}^{\mathcal{Y}}\right) \\ \pi_2^{in}(v) & \text{if } v \in V_2^{in} \setminus \left(\mathcal{I}^{\mathcal{X}} \cup \mathcal{I}^{\mathcal{Y}}\right) \\ \pi_3^{in}(v) & \text{if } v \in V_3^{in} \setminus \left(\mathcal{I}^{\mathcal{X}} \cup \mathcal{I}^{\mathcal{Y}}\right) \end{cases}$$

  Similarly for unconnected output ports. □

4.4.2 Communication

The composition operation in Def. 6 can be parametrized with communication programs *con* satisfying Remark 1. In this section, we formalize lossless and lossy communication (recall Section 3.2) as introduced in [26], and a unit conversion communication program.

Instantaneous, lossless communication. Instantaneous, lossless interaction between components is a useful model for direct communication between components and a first approximation for sensor measurements. Instantaneous, lossless communication $con_{ll}(v_i)$ can be modeled with a deterministic assignment from output port $\mathcal{X}(v_i)$ ($\mathcal{X}$ as in Def. 6) to the connected input port v_i as follows:

$$con_{ll}(v_i) \equiv v_i := \mathcal{X}(v_i) \ .$$

Instantaneous, lossy communication. Lossy communication can be used when transmission of exact values cannot be guaranteed, e. g., to model sensor uncertainty. We model lossy communication $con_{ly}(v_i)$ with a deterministic assignment distorted with a bounded error as follows:

$$con_{ly}(v_i) \equiv \lambda := *; ?\,|\lambda| \le \Lambda; v_i := \mathcal{X}(v_i) + \lambda \ .$$

The variable $\Lambda \in V^{global}$ is an error bound on the actual error λ, which can vary with every transmission. While the error variable λ must be local to each pair of connected ports, the same error bound Λ can be shared between multiple connections. Lossy communication with $\Lambda = 0$ is equivalent to lossless communication.

Unit conversion. Communication programs act as glue code between components that can perform computations on the transferred values. A typical example is conversion of units. Consider a sensor measuring the distance to an obstacle in feet, whereas a control component may perform computations in meters internally. A *unit conversion program* $con_{uc}(v_i)$ transforms the distance information and thus allows connecting ports without changing the original components, e. g.,

$$con_{uc}(v_i) \equiv v_i := \mathcal{X}(v_i) \cdot U \ ,$$

where U is a constant factor used for unit conversion (e. g., $U = 0.3048$ for conversion from feet to meters).

4.4.3 Example: Parallel Composition

Returning to our running example of Fig. 6, after contract compliance is checked for each component separately, we compose the components to form the overall collision avoidance system: the remote control gives speed advice to the robot, which measures obstacle positions to decide whether to follow the advice or stop to avoid collision with the obstacle. The connections $\mathcal{X}$ and $\mathcal{Y}$ connect the output ports of the RC and the obstacle with the respective input ports of the robot: the robot measures the obstacle position $\hat{p}_o$ from the true position p_o and receives the speed advice $\hat{d}$ from the true d (27).

$$\mathcal{X} = \big(\hat{p}_o \mapsto p_o\big) \enspace , \quad \mathcal{Y} = \big(\hat{d} \mapsto d\big) \tag{27}$$

The component C_{sys} in (28) and interface I_{sys} in (29) result from parallel composition of the RC, the robot, and the obstacle, using the connection mapping (27).

$$\begin{aligned}\mathrm{C}_{sys} = \Big(&\overbrace{(ctrl_{rc};\, ctrl_r;\, ctrl_o)}^{ctrl_{sys}}, \overbrace{(plant_r, plant_o)}^{plant_{sys}},\\ &\underbrace{\Big(con_{ll}(\hat{p}_o);\, con_{ll}(\hat{d})\Big)}_{cp_{sys}}\Big)\end{aligned} \tag{28}$$

$$\mathrm{I}_{sys} = \big(\underbrace{\{\}}_{V^{in}}, \underbrace{()}_{\pi^{in}}, \underbrace{\{\}}_{V^{out}}, \underbrace{()}_{\pi^{out}}, \underbrace{\{p_o^-, d^-, \hat{p}_o^-, \hat{d}^-\}}_{V^-}\big) \tag{29}$$

The robot's input ports are connected to the RC's and obstacle's output ports. How values are transmitted between robot, obstacle and RC is specified by $con_{ll}(\hat{p}_o)$ and $con_{ll}(\hat{d})$:

$$con_{ll}(\hat{p}_o) \equiv \hat{p}_o := p_o$$
$$con_{ll}(\hat{d}) \equiv \hat{d} := d$$

Here, we connect two pairs of ports, transferring (i) the obstacle position from the obstacle to the robot, and (ii) the speed advice from the RC to the robot. The robot might measure the position of the obstacle using a sensor, which is subject to sensor uncertainty. This can be modeled using lossy communication as follows (Λ_p represents the maximum measurement error, according to the sensor's specification):

$$con_{ly}(\hat{p}_o) \equiv \lambda_p := *;\ ?\,|\lambda_p| \le \Lambda_p;\ \hat{p}_o := p_o + \lambda_p$$

4.4.4 Connection Compatibility

During composition, the tests guarding the input ports of an interface are replaced with a hybrid program modeling the port connections of the components. That is only correct if the respective output guarantees and input assumptions match. Hence, in addition to contract compliance, users have to show connection compatibility.

Compatibility links the output guarantees of an output port to the input assumptions of its connected input port via a specific communication program. We summarize the behavior of the communication program *con* as a communication guarantee as follows.

Definition 7 (Communication guarantee). Let port connection *con* satisfy Remark 1 and transfer values from output port v_{out} to input port v_{in}. We say connection *con* provides a communication guarantee $\zeta(v_{in}, v_{out})$ if the following dL formulas are valid:

$$[con(v_{in})]\zeta(v_{in}, v_{out}) \tag{30}$$
$$\langle con(v_{in})\rangle true \enspace . \tag{31}$$

Each communication program *con* requires a suitable communication guarantee $\zeta(v_{in}, v_{out})$ and user-provided proofs of (30) and (31). The communication guarantee can for instance be derived from the communication program using ModelPlex [22]. The communication guarantees of lossless and lossy communication are straightforward. Lossless communication directly assigns values, so the communication guarantee (32) unambiguously characterizes the communication program by ensuring that the values of the connected ports are equal.

$$\zeta_{ll}(v_{in}, v_{out}) \equiv v_{in} = v_{out} \tag{32}$$

Lossy communication allows for a measurement error, which is reflected in the communication guarantee (33).

$$\zeta_{ly}(v_{in}, v_{out}) \equiv |\lambda| \le \Lambda \wedge v_{in} = v_{out} + \lambda \tag{33}$$

Definition 8 (Compatible connection). A parallel composition $((\mathrm{C}_1, \mathrm{I}_1) \| (\mathrm{C}_2, \mathrm{I}_2))_{\mathcal{X}}$ is *compatible* iff dL formula

$$\begin{aligned} CPO(\mathcal{X}) \overset{\text{def}}{\equiv}\ & \zeta(\mathrm{old}(v), \mathrm{old}(\mathcal{X}(v))) \rightarrow \\ & [con(v)]\big(\pi_j^{out}(\mathcal{X}(v)) \rightarrow \pi_i^{in}(v)\big)\end{aligned}$$

is valid over (vectorial) equalities and assignments for input ports $v \in \mathcal{I}^{\mathcal{X}}$. Formula $\zeta(\mathrm{old}(v), \mathrm{old}(\mathcal{X}(v))$ is the communication guarantee. Facts about global constants V^{global} can be used in the proof. We call *CPO($\mathcal{X}$)* the *compatibility proof obligation* for the connection $\mathcal{X}$ between interfaces I_1 and I_2 and say the interfaces I_1 and I_2 are *compatible* with respect to $\mathcal{X}$ if *CPO($\mathcal{X}$)* is valid.

Compatibility ensures that the output guarantees are strong enough to satisfy the input assumptions of connected ports under a certain communication program and its communication guarantee. This is important to preserve the input assumptions in the composed system, which lacks the explicit tests of the isolated components. To achieve local compatibility checks for pairs of connected ports, instead of global checks over entire component models, Def. 4 restricts input assumptions to only mention variables of the associated ports. Note that even though Def. 4 does not restrict output guarantees, compatibility proofs will only succeed if output guarantee also only mention variables of the associated ports.

4.4.5 Example: Compatibility

In our example, we have to ensure compatibility of the components with respect to $\mathcal{X}$ and $\mathcal{Y}$ (27). Since we have two connected ports, we discharge two compatibility proof obligations (34) and (35), one for each port.

$$CPO(\mathcal{Y}) \equiv \big((d^- = \hat{d}^-) \wedge (S \geq 0 \wedge D \geq 0)\big) \rightarrow \\ [\hat{d} := d]\Big(\left|d - d^-\right| \leq D \rightarrow \left|\hat{d} - \hat{d}^-\right| \leq D\Big) \tag{34}$$

$$CPO(\mathcal{X}) \equiv \big(\,(p_o^- = \hat{p}_o^-) \wedge (S \geq 0 \wedge D \geq 0)\big) \rightarrow \\ [\hat{p}_o := p_o]\Big(\left|p_o - p_o^-\right| \leq S \cdot (t - t^-) \rightarrow \\ \left|\hat{p}_o - \hat{p}_o^-\right| \leq S \cdot (t - t^-)\,\Big) \tag{35}$$

Formulas (34) and (35) can be proved automatically using KeYmaera X, i. e., connections $\mathcal{X}$ and $\mathcal{Y}$ are compatible.

Compatibility for lossy communication. With lossy communication between robot and obstacle, we can no longer verify the compatibility proof obligation, because the input port presently requires exact measurements, see (17). Incompatibility indicates that either the robot or the obstacle interface made incompatible assumptions about its environment and requires change. In (36) below, we opt for changing the robot to allow sensor uncertainty in its input assumptions, which in turn requires change to the robot controller to re-establish contract compliance. Formula (36) uses the communication invariant and the communication program for lossy composition, and additionally changes the robot's input assumption to consider the possible loss of precision.

$$CPO(\mathcal{X}) \equiv \boldsymbol{|\lambda_p| \leq \Lambda_p \wedge p_o^- = \hat{p}_o^- + \lambda_p} \wedge S \geq 0 \wedge D \geq 0 \\ \rightarrow [\boldsymbol{\lambda_p := *;\ ?\,|\lambda_p| \leq \Lambda_p;\ \hat{p}_o := p_o + \lambda_p}] \\ \Big(\big(\left|p_o - p_o^-\right| \leq S \cdot (t - t^-) \rightarrow \\ \left|\hat{p}_o - \hat{p}_o^-\right| \leq S \cdot (t - t^-) \boldsymbol{+ 2\Lambda_p}\,\big)\Big) \tag{36}$$

4.5 Transferring Component Safety to System Safety

From contract compliance and compatibility proofs, Theorem 1 below transfers the local safety responsibilities in component contracts to safety of the composed system. As a result, showing safety of the composed system no longer requires a monolithic proof, but is inferred from local component and compatibility proofs. The proof of Theorem 1 can be found in Section 5 as part of our implementation.

Theorem 1 (Composition contract retention)**.** *Let C_1 and C_2 be components with interfaces I_1 and I_2 that are rate contract compliant per Def. 5 and* compatible *with respect to $\mathcal{X}$ per Def. 8. Assume the communication guarantee ζ per Def. 7 holds initially to start from consistent connected ports. Then the parallel composition $(C_1, I_1) \|_{\mathcal{X}} (C_2, I_2)$ satisfies the following contract with Ω specifying global system parameters over V^{global} and* in*,* cp*,* ctrl*, and* plant *according to Def. 6:*

$$\vDash (t = t^- \wedge \Omega \wedge \phi_1 \wedge \phi_2 \wedge \zeta) \rightarrow \\ [(\Delta; \text{ctrl}; t^- := t; \{t' = 1, \text{plant}\}\,; \text{in}; \text{cp})^*] \\ \big(\psi_1^{\text{safe}} \wedge \Pi_1^{\text{out}} \wedge \psi_2^{\text{safe}} \wedge \Pi_2^{\text{out}}\big)\ .$$

Theorem 1 provides strong safety guarantees about the whole system from local component and compatibility proofs, but requires that the assumptions made in Def. 5, Def. 6, Def. 7, and Def. 8 are carefully checked on every use. The tactic in Section 5 will do so automatically from the axioms of dL.

Remark 3. Because of the precondition ζ and because *cp* is executed after every execution of the main loop per Def. 5, we know that the values of connected input and output ports behave as indicated by their communication guarantee, as one would expect. This is useful to deduce system safety properties about true values from guarantees about measured values. With lossless communication, for instance, the local safety responsibility of the robot $|p_r - \hat{p}_o| > 0$ phrased over measured obstacle position $\hat{p}_o$ guarantees safety over true obstacle positions $|p_r - p_o| > 0$.

5 Proof Automation

Proof automation for Theorem 1 can be achieved in the hybrid systems theorem prover KeYmaera X [10] in different ways: (i) Theorem 1 could be added directly to the prover core as a new proof rule, which is efficient but requires a complicated soundness-critical algorithm that checks all its nontrivial side conditions and, thereby, significantly increases the complexity of the algorithms that are responsible for the correctness of the verification results; (ii) as a proof tactic *outside* the small soundness-critical core to automatically derive a proof for each composite system instance from individual component and compatibility proofs, with all side conditions verified in the core for free as part of the proof construction. We follow the tactic-based approach since it preserves soundness and is able to handle user-defined component behavior.

The main idea behind the proof construction tactic, as illustrated for the robot and remote control in Fig. 8, is to adapt the program shape of the composed system to match the shape of its components, so that component proofs fill in most proof obligations directly. The proof reuse mechanism of KeYmaera X closes proof obligations from lemmas whose conclusion is syntactically equal to the open proof obligation. This requires additional systematic proof steps to adapt the shape of an open proof obligation to exactly the shape of the lemma conclusion. Therefore, the tactic adapts the shape of the system contract per Def. 6 to the shapes of component and compatibility proofs with the following steps:

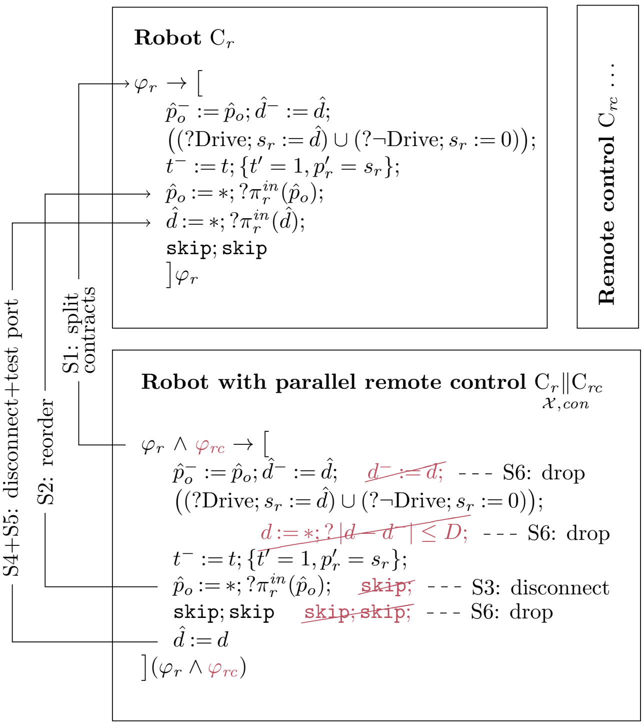
Robot C_r
$\varphi_r \rightarrow [$
$\hat{p}_o^- := \hat{p}_o; \hat{d}^- := \hat{d};$
$\big((?\text{Drive}; s_r := \hat{d}) \cup (?\neg\text{Drive}; s_r := 0)\big);$
$t^- := t; \{t' = 1, p_r' = s_r\};$
$\hat{p}_o := *; ?\pi_r^{in}(\hat{p}_o);$
$\hat{d} := *; ?\pi_r^{in}(\hat{d});$
`skip; skip`
$]\varphi_r$
Remote control C_{rc} ...
S4+S5: disconnect+test port
S2: reorder
S1: split contracts
Robot with parallel remote control $\mathrm{C}_r \underset{\mathcal{X},con}{\|} \mathrm{C}_{rc}$
$\varphi_r \wedge \varphi_{rc} \rightarrow [$
$\hat{p}_o^- := \hat{p}_o; \hat{d}^- := \hat{d};$ ~~$d^- := d;$~~ --- S6: drop
$\big((?\text{Drive}; s_r := \hat{d}) \cup (?\neg\text{Drive}; s_r := 0)\big);$
~~$d := *; ?\,|d - d^-| \leq D;$~~ --- S6: drop
$t^- := t; \{t' = 1, p_r' = s_r\};$
$\hat{p}_o := *; ?\pi_r^{in}(\hat{p}_o);$ ~~`skip;`~~ --- S3: disconnect
`skip; skip` ~~`skip; skip;`~~ --- S6: drop
$\hat{d} := d$
$](\varphi_r \wedge \varphi_{rc})$

Figure 8: Proof sketch to decompose the composition of robot and remote control into isolated components

S1 splits the proof along component contracts (proves that the composed system preserves the component contracts)
S2 reorders communication programs to match the order in the corresponding component (Lemma 1)
S3 disconnects outputs by dropping all communication programs that are irrelevant for the current contract (Lemma 2)
S4 re-introduces tests for input assumptions after communication programs to prepare disconnecting inputs (Lemmas 5 and 6)
S5 disconnects inputs by replacing communication programs with nondeterministic assignments to resemble port behavior of unconnected components (Lemma 4)
S6 drops plants and controllers that are irrelevant for the current contract (Lemmas 2 and 3)

The lemmas and tactic details in the following subsections illustrate the details of the system proof construction and also serve as an example of proof reuse in KeYmaera X.

5.1 Automation for Program Shape Adaptation

In this section, we introduce lemmas for program shape adaptation that are used in proving system safety from component and compatibility proofs following the above proof sketch. The proofs for Lemmas 2, 3, 5, and 6 follow our prior work [25]. A detailed proof for the newly introduced Lemma 1 can be found in Appendix C, and helpful implementation Corollaries in Appendix D. Throughout the section, we use the proof rules and axioms listed in Appendix B.

Drop Control. We use Lemma 2 below to simplify programs to only the relevant statements that influence the safety property.

Lemma 2 (Drop control)**.** *Let A be a* dL *formula and α, β be hybrid programs. Program β has no influence over A, i. e., $FV(A) \cap BV(\beta) = \emptyset$ and there is no information flow from β to α, i. e., $FV(\alpha) \cap BV(\beta) = \emptyset$. Then these formulas are valid:*

$$[\alpha]A \rightarrow [\beta][\alpha]A \text{ and } [\alpha]A \rightarrow [\alpha][\beta]A$$

Since Lemma 2 makes crucial assumptions about the intersection of free and bound variables of program constants α and β, it is not expressible as an axiom in KeYmaera X. However, when implemented as a tactic that operates on concrete programs α and β, their free and bound variables can be computed (e. g., the HP $x := y$ has bound variable x and free variable y) and the assumptions checked. Uniform substitution [36] in the KeYmaera X kernel will fail the tactic if it operates on programs that violate the assumptions.

Drop Plant. Lemma 3 simplifies systems of ODEs to only those differential equations that are relevant for the safety property.

Lemma 3 (Drop plant)**.** *Let θ and η be terms possibly mentioning x and y, respectively, where x and y are vectors of disjoint variables. Let A be a* dL *formula over x and H, Q be formulas over x and y, respectively. Then*

$$[\{x' = \theta \ \& \ H\}]A \rightarrow [\{x' = \theta, y' = \eta \ \& \ H \wedge Q\}]A$$

is valid.

Similarly to Lemma 2, we implement Lemma 3 as a tactic that operates on concrete programs and relies on uniform substitution for soundness.

Nondeterministic Program Overapproximation. Instead of proving a safety property about a hybrid program, we can replace it with a proof for nondeterministic assignments to the variables bound in the program. This is useful for generalizing proofs: instead of proving a property about a specific program, we prove this property about a more abstract family of programs, which makes the proof reusable. Lemma 4 will be used to drop connections by replacing communication programs with nondeterministic assignments that represent unconnected input ports.

Lemma 4 (Overapproximate program)**.** *Let $x = BV(\alpha)$ be the bound variables of program α, and A be a* dL *formula potentially mentioning x. Then this is valid:*

$$[x := *]A \rightarrow [\alpha]A$$

$$
\to\text{r}\,\dfrac{\text{ind}\,\dfrac{{}^{(40)}\dfrac{*}{t = t^- \wedge \phi_3 \vdash \varphi_3} \qquad \dfrac{\text{② (Fig. 10)}}{\varphi_3 \vdash [\Delta_3; ctrl_3; t^- := t; \{t' = 1, plant_3\ \}\,; in_3; cp_3]\,\varphi_3} \qquad {}^{(42)}\dfrac{*}{\varphi_3 \vdash \left(\psi_3^{safe} \wedge \Pi_3^{out}\right)}}{t = t^- \wedge \phi_3 \vdash \left[(\Delta_3; ctrl_3; t^- := t; \{t' = 1, plant_3\}\,; in_3; cp_3)^*\right]\left(\psi_3^{safe} \wedge \Pi_3^{out}\right)}}{\vdash t = t^- \wedge \phi_3 \to \left[(\Delta_3; ctrl_3; t^- := t; \{t' = 1, plant_3\}\,; in_3; cp_3)^*\right]\left(\psi_3^{safe} \wedge \Pi_3^{out}\right)}
$$

Figure 9: Loop induction on the system contract using the composite loop invariant.

$$
\text{def}\,\dfrac{\wedge\text{l}\,\dfrac{[;]\,\dfrac{[]\wedge,\wedge\text{r}\,\dfrac{\dfrac{\text{④ (Fig. 11)}}{\varphi_1, \varphi_2, \zeta \vdash [\ldots]\varphi_1} \qquad \dfrac{\text{similar to ④ (Fig. 11)}}{\varphi_1, \varphi_2, \zeta \vdash [\ldots]\varphi_2} \qquad \text{def}\,\dfrac{(30)\,\dfrac{\text{V}\,\dfrac{*}{\varphi_1, \varphi_2, \zeta \vdash [\ldots][in_3; cp_1; cp_2]true}}{\varphi_1, \varphi_2, \zeta \vdash [\ldots][in_3; cp_1; cp_2; con]\zeta}}{\varphi_1, \varphi_2, \zeta \vdash [\ldots][in_3; cp_3]\zeta}}{\varphi_1, \varphi_2, \zeta \vdash [\Delta_3][ctrl_3][t^- := t][\{t' = 1, plant_3\}][in_3; cp_3]\,(\varphi_1 \wedge \varphi_2 \wedge \zeta)}}{\varphi_1, \varphi_2, \zeta \vdash [\Delta_3; ctrl_3; t^- := t; \{t' = 1, plant_3\}\,; in_3; cp_3]\,(\varphi_1 \wedge \varphi_2 \wedge \zeta)}}{\varphi_1 \wedge \varphi_2 \wedge \zeta \vdash [\Delta_3; ctrl_3; t^- := t; \{t' = 1, plant_3\}\,; in_3; cp_3]\,(\varphi_1 \wedge \varphi_2 \wedge \zeta)}}{\dfrac{\varphi_3 \vdash [\Delta_3; ctrl_3; t^- := t; \{t' = 1, plant_3\}\,; in_3; cp_3]\varphi_3}{\text{② (Fig. 9) continued}}}
$$

Figure 10: Prove component loop invariants and communication guarantee separately

Test Introduction. Program overapproximation with Lemma 4 discards all knowledge about the computations of the abstracted programs. Lemma 5 allows summarizing the relevant characteristics of these computations as tests before overapproximation.

Lemma 5 (Introduce test)**.** *Let A and F be formulas, and α be a hybrid program. Then this is valid:*

$$[\alpha]F \to \big([\alpha; ?F]A \leftrightarrow [\alpha]A\big)$$

Test Weakening. The following Lemma 6 allows us to weaken test conditions that are unnecessarily strong.

Lemma 6 (Weaken test)**.** *Let A, F, and G be formulas. Then this is valid:*

$$\big((F \to G) \wedge [?G]A\big) \to [?F]A$$

5.2 Automation for System Safety Proofs

Now that we have created the necessary prerequisites in the form of provably correct program shape adaptations, we implement the proof sketch as a KeYmaera X tactic, which automatically derives a system safety proof from component and compatibility proofs.

Users provide component proofs that witness contract compliance, i. e., formula (37) is valid (accordingly for component C$_2$).

$$
\begin{aligned}
&RCC(C_1, I_1) \overset{\text{Def. 5}}{\equiv} t = t^- \wedge \phi_1 \to [(\Delta_1; ctrl_1; t^- := t; \\
&\quad \{t' = 1, plant_1\}\,; in_1; cp_1)^*]\left(\psi_1^{safe} \wedge \Pi_1^{out}\right) \qquad (37)
\end{aligned}
$$

Users also provide compatibility proofs that witness compatible connections, i. e., formulas (38) (one for each connection) are valid.

$$
\begin{aligned}
CPO(\mathcal{X}) &\overset{\text{Def. 8}}{\equiv} \Big(\zeta(\text{old}(v), \text{old}(\mathcal{X}(v))) \wedge \Omega\Big) \to \\
&\qquad [con(v)]\big(\pi_j^{out}(\mathcal{X}(v)) \to \pi_i^{in}(v)\big)
\end{aligned} \qquad (38)
$$

In summary, from component proofs (37) and compatibility proofs (38), we prove system safety (39) of the parallel composition $(C_3, I_3) \overset{\text{def}}{\equiv} (C_1, I_1) \underset{\mathcal{X}, con}{\|} (C_2, I_2)$, that is:

$$
\begin{aligned}
&RCC(C_3, I_3) \overset{\text{Def. 5}}{\equiv} t = t^- \wedge \phi_3 \to [\big(\Delta_3; ctrl_3; t^- := t; \\
&\quad \{t' = 1, plant_3\}\,; in_3; cp_3\big)^*]\left(\psi_3^{safe} \wedge \Pi_3^{out}\right) \qquad (39)
\end{aligned}
$$

where the items of the parallel composition follow from Def. 6 and invariants, input assumptions, and output guarantees are read off the component proofs:

$$
\begin{aligned}
ctrl_3 &\overset{\text{def}}{\equiv} ctrl_1; ctrl_2 \\
plant_3 &\overset{\text{def}}{\equiv} plant_1, plant_2 \\
\phi_3 &\overset{\text{def}}{\equiv} \phi_1 \wedge \phi_2 \wedge \zeta \\
\psi_3^{safe} &\overset{\text{def}}{\equiv} \psi_1^{safe} \wedge \psi_2^{safe} \\
\Pi_3^{out} &\overset{\text{def}}{\equiv} \Pi_1^{out} \wedge \Pi_2^{out}
\end{aligned}
$$

For space reasons, we elide facts about global constants. They are invariant throughout the proof, so available everywhere.

The tactic has to verify that the contract (39) of the parallel composition $RCC(C_3, I_3)$ is valid. We know that

$$\begin{array}{lrl}
 & & \ast \\ \hline
(41) & \varphi_1 & \vdash [\Delta_1][\mathit{ctrl}_1][t^- := t][\{t' = 1, \mathit{plant}_1\}][\mathit{in}_1][\mathit{cp}_1]\varphi_1 \\ \hline
\mathrm{Wl} & \varphi_1, \varphi_2 & \vdash [\Delta_1][\mathit{ctrl}_1][t^- := t][\{t' = 1, \mathit{plant}_1\}][\mathit{in}_1][\mathit{cp}_1]\varphi_1 \\ \hline
\text{14.,L. 2} & \varphi_1, \varphi_2 & \vdash [\Delta_1; \Delta_2][\mathit{ctrl}_1; \mathit{ctrl}_2][t^- := t][\{t' = 1, \mathit{plant}_1\}][\mathit{in}_1][\mathit{cp}_1]\varphi_1 \\ \hline
\text{13.,def,L. 1} & \varphi_1, \varphi_2 & \vdash [\Delta_3][\mathit{ctrl}_1; \mathit{ctrl}_2][t^- := t][\{t' = 1, \mathit{plant}_1\}][\mathit{in}_1][\mathit{cp}_1]\varphi_1 \\ \hline
\text{12.,L. 3} & \varphi_1, \varphi_2 & \vdash [\Delta_3][\mathit{ctrl}_1; \mathit{ctrl}_2][t^- := t][\{t' = 1, \mathit{plant}_1, \mathit{plant}_2\}][\mathit{in}_1][\mathit{cp}_1]\varphi_1 \\ \hline
\text{def} & \varphi_1, \varphi_2 & \vdash [\Delta_3][\mathit{ctrl}_1; \mathit{ctrl}_2][t^- := t][\{t' = 1, \mathit{plant}_3\}][\mathit{in}_1][\mathit{cp}_1]\varphi_1 \\ \hline
\text{11.} & \varphi_1, \varphi_2 & \vdash [\Delta_3][\mathit{ctrl}_1; \mathit{ctrl}_2][t^- := t][\{t' = 1, \mathit{plant}_3\}][\mathit{in}_1^*][v_j := \ast][?\pi_1^{\mathit{in}}(v_j)][\mathit{con}^*][\mathit{cp}_1]\varphi_1 \\ \hline
\text{10.,L. 4} & \varphi_1, \varphi_2 & \vdash [\Delta_3][\mathit{ctrl}_1; \mathit{ctrl}_2][t^- := t][\{t' = 1, \mathit{plant}_3\}][\mathit{in}_1^*][\mathit{con}(v_j)][?\pi_1^{\mathit{in}}(v_j)][\mathit{con}^*][\mathit{cp}_1]\varphi_1 \quad \ldots\text{⑦ (Fig. 13)} \\ \hline
\text{9.,L. 6} & \varphi_1, \varphi_2, \zeta & \vdash [\Delta_3][\mathit{ctrl}_1; \mathit{ctrl}_2][t^- := t][\{t' = 1, \mathit{plant}_3\}][\mathit{in}_1^*][\mathit{con}(v_j)][?\pi_2^{\mathit{out}}(\mathcal{X}(v_j))][\mathit{con}^*][\mathit{cp}_1]\varphi_1 \\ \hline
\text{8.,L. 1} & \varphi_1, \varphi_2, \zeta & \vdash [\Delta_3][\mathit{ctrl}_1; \mathit{ctrl}_2][t^- := t][\{t' = 1, \mathit{plant}_3\}][?\pi_2^{\mathit{out}}(\mathcal{X}(v_j))][\mathit{in}_1^*][\mathit{con}(v_j)][\mathit{con}^*][\mathit{cp}_1]\varphi_1 \\ \hline
\text{7.,L. 5} & \varphi_1, \varphi_2, \zeta, F_2^{\mathit{out}} & \vdash [\Delta_3][\mathit{ctrl}_1; \mathit{ctrl}_2][t^- := t][\{t' = 1, \mathit{plant}_3\}][\mathit{in}_1^*][\mathit{con}(v_j)][\mathit{con}^*][\mathit{cp}_1]\varphi_1 \\ \hline
\text{6.,def} & \varphi_1, \varphi_2, \zeta, F_2^{\mathit{out}} & \vdash [\Delta_3][\mathit{ctrl}_1; \mathit{ctrl}_2][t^- := t][\{t' = 1, \mathit{plant}_3\}][\mathit{in}_1^*; \mathit{con}; \mathit{cp}_1]\varphi_1 \\ \hline
\text{5.,L. 2} & \varphi_1, \varphi_2, \zeta, F_2^{\mathit{out}} & \vdash [\Delta_3][\mathit{ctrl}_1; \mathit{ctrl}_2][t^- := t][\{t' = 1, \mathit{plant}_3\}][\mathit{in}_1^*; \mathit{con}; \mathit{in}_2^*; \mathit{cp}_1]\varphi_1 \\ \hline
\text{4.,L. 2} & \varphi_1, \varphi_2, \zeta, F_2^{\mathit{out}} & \vdash [\Delta_3][\mathit{ctrl}_1; \mathit{ctrl}_2][t^- := t][\{t' = 1, \mathit{plant}_3\}][\mathit{in}_1^*; \mathit{con}; \mathit{in}_2^*; \mathit{cp}_1; \mathit{cp}_2]\varphi_1 \\ \hline
\text{3.,L. 1} & \varphi_1, \varphi_2, \zeta, F_2^{\mathit{out}} & \vdash [\Delta_3][\mathit{ctrl}_1; \mathit{ctrl}_2][t^- := t][\{t' = 1, \mathit{plant}_3\}][\mathit{in}_1^*; \mathit{in}_2^*; \mathit{cp}_1; \mathit{cp}_2; \mathit{con}]\varphi_1 \\ \hline
\text{2.,def,L. 1} & \varphi_1, \varphi_2, \zeta, F_2^{\mathit{out}} & \vdash [\Delta_3][\mathit{ctrl}_1; \mathit{ctrl}_2][t^- := t][\{t' = 1, \mathit{plant}_3\}][\mathit{in}_3; \mathit{cp}_3]\varphi_1 \quad \ldots\text{⑥ (Fig. 12)} \\ \hline
\text{1.,cut} & \varphi_1, \varphi_2, \zeta & \vdash [\Delta_3][\mathit{ctrl}_1; \mathit{ctrl}_2][t^- := t][\{t' = 1, \mathit{plant}_3\}][\mathit{in}_3; \mathit{cp}_3]\varphi_1 \\ \hline
 & & \text{④ (Fig. 10) continued}
\end{array}$$

Figure 11: Disassemble system into components: adapt program order and disconnect ports

formula (37) is valid, hence we can read off invariant φ_1 from the contract compliance proof of component C_1 (accordingly for component C_2) such that:

$$\models t = t^- \land \phi_1 \to \varphi_1 \tag{40}$$

$$\models \varphi_1 \to [\Delta_1; \mathit{ctrl}_1; t^- := t; \{t' = 1, \mathit{plant}_1\}\,; \mathit{in}_1; \mathit{cp}_1]\varphi_1 \tag{41}$$

$$\models \varphi_1 \to \left(\psi_1^{\mathit{safe}} \land \Pi_1^{\mathit{out}}\right) \tag{42}$$

Formula φ_1 is an inductive loop invariant for the component C_1, so $\mathrm{FV}(\varphi_1) \subseteq V(\mathrm{C}_1) \cup V^{\mathit{global}} \cup \{t, t^-\}$.

The system proof construction in Fig. 9 is a sequent proof: horizontal lines separate the premises of a proof step from its conclusion; from proven premises (above the line), the axioms and proof rules that are annotated to the left of the horizontal lines then justify the conclusion (below the line).

The proof uses loop induction for the system contract, using $\varphi_3 \equiv \varphi_1 \land \varphi_2 \land \zeta$ as a loop invariant, i. e., the conjunction of the two component loop invariants φ_1 and φ_2, and the communication guarantee ζ.

The tactic transforms each branch individually until we get formulas that correspond to (40), (41) and (42). To prove the induction base case and use case, the tactic applies a series of simple propositional steps (see Fig. 17 and Fig. 18 in Appendix C for details) to transform each branch until it can be closed from formulas (40), (42) (hold correspondingly for φ_2 of the component C_2), or from the communication guarantee ζ.

The induction step follows the steps S1–S6 of the proof sketch and is described in detail below.

S1: Split along contracts. In the induction step Fig. 10, after a few simple structural steps the tactic proves invariance of φ_1, φ_2, and ζ separately on three branches: branch ④ to prove invariance of φ_1 and accordingly for φ_2, as well as preservation of the communication guarantee ζ. For communication guarantees we know that $[\mathit{con}]\zeta$ holds by Def. 7 (30), which together with $\mathit{cp}_3 \equiv \mathit{cp}_1; \mathit{cp}_2; \mathit{con}$ per Def. 6 closes branch $[\ldots][\mathit{in}_3; \mathit{cp}_3]\zeta$ immediately.

The remaining goal of the tactic is to transform ④ until it matches the shape of formula (41) to close the system proof from the component proofs. The proof steps are summarized in Fig. 11. The tactic explanation below follows the proof sketch and uses the same step numbers as in the sequent proof in Fig. 11. Large parts of the tactic operate on the connections without touching port memory, controllers, or plant; these passive parts of the proof are grayed out.

We abbreviate

$$\begin{aligned} F_2^{\mathit{out}} \overset{\text{def}}{\equiv} \varphi_2 \to & [\Delta_3][\mathit{ctrl}_1; \mathit{ctrl}_2][t^- := t] \\ & [\{t' = 1, \mathit{plant}_1, \mathit{plant}_2\}]\Pi_2^{\mathit{out}} \ . \end{aligned}$$

1. We simplify the side condition proof of Lemma 5 by an initial cut to provide F_2^{out} as an assumption that will be used throughout the proof. The side condition itself is verified in ⑥ (Fig. 12) from the component induction step (41) of component C_2, as described below.
   A Plant plant_1, port memory Δ_1, and control ctrl_1 of component C_1 are irrelevant in the side condition since their bound variables do not overlap

$$
\text{Wl,Wr}\;\dfrac{\text{def}\;\dfrac{\text{A,L. 3}\;\dfrac{\text{def}\;\dfrac{\text{A,L. 2}\;\dfrac{\text{A,L. 2}\;\dfrac{\text{B,MR}\;\dfrac{\langle\cdot\rangle,\neg\mathrm{l}\;\dfrac{\text{C,V}\;\dfrac{\neg\mathrm{r}\;\dfrac{\text{id}\;\dfrac{*}{\mathit{\Pi}_2^{out} \vdash \mathit{\Pi}_2^{out}}}{\vdash \neg\mathit{\Pi}_2^{out}, \mathit{\Pi}_2^{out}}}{\vdash [in_2; cp_2]\neg\mathit{\Pi}_2^{out}, \mathit{\Pi}_2^{out}}}{\langle in_2; cp_2\rangle\mathit{\Pi}_2^{out} \vdash \mathit{\Pi}_2^{out}} \quad \text{D,}[\cdot]{\to}\langle\cdot\rangle\;\dfrac{\text{E,(42)}\;\dfrac{[;]\;\dfrac{(41)\;\dfrac{*}{\vdash \varphi_2 \to [\mathit{\Delta}_2; ctrl_2; t^- := t; \{t' = 1, plant_2\}\,; in_2; cp_2]\varphi_2}}{\vdash \varphi_2 \to [\mathit{\Delta}_2][ctrl_2][t^- := t][\{t' = 1, plant_2\}][in_2; cp_2]\varphi_2}}{\vdash \varphi_2 \to [\mathit{\Delta}_2][ctrl_2][t^- := t][\{t' = 1, plant_2\}][in_2; cp_2]\mathit{\Pi}_2^{out}}}{\vdash \varphi_2 \to [\mathit{\Delta}_2][ctrl_2][t^- := t][\{t' = 1, plant_2\}]\langle in_2; cp_2\rangle\mathit{\Pi}_2^{out}}}{\vdash \varphi_2 \to [\mathit{\Delta}_2][ctrl_2][t^- := t][\{t' = 1, plant_2\}]\mathit{\Pi}_2^{out}}}{\vdash \varphi_2 \to [\mathit{\Delta}_1; \mathit{\Delta}_2][ctrl_2][t^- := t][\{t' = 1, plant_2\}]\mathit{\Pi}_2^{out}}}{\vdash \varphi_2 \to [\mathit{\Delta}_1; \mathit{\Delta}_2][ctrl_1; ctrl_2][t^- := t][\{t' = 1, plant_2\}]\mathit{\Pi}_2^{out}}}{\vdash \varphi_2 \to [\mathit{\Delta}_3][ctrl_1; ctrl_2][t^- := t][\{t' = 1, plant_2\}]\mathit{\Pi}_2^{out}}}{\vdash \varphi_2 \to [\mathit{\Delta}_3][ctrl_1; ctrl_2][t^- := t][\{t' = 1, plant_1, plant_2\}]\mathit{\Pi}_2^{out}}}{\vdash F_2^{out}}}{\varphi_1, \varphi_2, \zeta \vdash [\mathit{\Delta}_3][ctrl_1; ctrl_2][t^- := t][\{t' = 1, plant_1, plant_2\}]\varphi_1, F_2^{out}}
$$

⑥ (Fig. 11) continued

Figure 12: Justify F_2^{out}.

the free variables of F_2^{out} and are dropped using Lemmas 2 and 3.

B With rule MR we generalize the postcondition to introduce inputs and connections $\langle in_2; cp_2\rangle\mathit{\Pi}_2^{out}$ of component C_2 to adapt towards the shape of formula (41).

C We use $\langle\cdot\rangle$ to turn the assumption $\langle in_2; cp_2\rangle\mathit{\Pi}_2^{out}$ into a proof obligation $[in_2; cp_2]\neg\mathit{\Pi}_2^{out}$ whose dynamics can then be abstracted with V, because $\mathrm{FV}(\mathit{\Pi}_2^{out}) \cap \mathrm{BV}(in_2; cp_2) = \emptyset$.

D We know $\langle in_2; cp_2\rangle true$ since all input assumptions π_2^{in} in in_2 are satisfiable per Def. 4 and all the connection programs $con(v_{in})$ in $ports_2$ can run ($\models \langle con(v_{in})\rangle true$ per Def. 7). Using derived axiom $[\cdot]{\to}\langle\cdot\rangle$ (i. e., $\langle\alpha\rangle true \to [\alpha]P \to \langle\alpha\rangle P$), this turns the liveness obligation $\langle in_2; cp_2\rangle\mathit{\Pi}_2^{out}$ into a safety obligation $[in_2; cp_2]\mathit{\Pi}_2^{out}$.

E The use case proof of (42) of component C_2 allows strengthening $\mathit{\Pi}_2^{out}$ to φ_2 by rule MR, which then concludes the side condition proof from the induction step proof (41) of component C_2. □

S2: Reorder ports and communication programs. This step adapts the order of ports and communication programs so that subsequent steps meet their requirements on bound and free variables.

2. We use Lemma 1 to reorder the subprograms within in_3 and cp_3 such that the input ports of C_1 precede the ones of C_2. The KeYmaera X prover kernel will fail Lemma 1 if the conditions on variable binding in open ports as well as communication programs are violated, i. e., if variables other than the respective input ports are modified.
3. Next, we use Lemma 1 to move the external connections con directly after the open input ports in_1^* in preparation of disconnecting inputs, so that later external communication programs can be moved easily into the unconnected input ports in_1^* to rebuild the isolated in_1.

S3: Disconnect outputs. Next, we work towards isolating C_1: we disconnect the outputs of component C_1 from the inputs of component C_2 and drop all open inputs of component C_2.

4. Lemma 2 removes all internal connections cp_2 of component C_2, where $\alpha \overset{\text{def}}{\equiv} cp_1$, $\beta \overset{\text{def}}{\equiv} cp_2$ and $A \overset{\text{def}}{\equiv} \varphi_1$. Lemma 2 is applicable, because there is no information flow from program β to α ($\mathrm{FV}(\alpha) \cap \mathrm{BV}(\beta) = \emptyset$) and formula A is not influenced by program β ($\mathrm{FV}(A) \cap \mathrm{BV}(\beta) = \emptyset$). We know that there is no information flow from β to α since $\mathrm{FV}(cp_1) \subseteq V(\mathrm{C}_1)$ are disjoint from $\mathrm{BV}(cp_2) \cap (V(\mathrm{C}_1) \cup V(\mathrm{C}_2)) \subseteq V_2^{in}$, and therefore in turn also disjoint from $\mathrm{BV}(cp_2) \cap (V(\mathrm{C}_1) \cup V(\mathrm{C}_2)) \subseteq V(\mathrm{C}_2) \setminus \big(V^{global} \cup \{t, t^-\}\big)$). Formula A is not influenced by program β since $\mathrm{FV}(\varphi_1) \subseteq V(\mathrm{C}_1)$ and $V(\mathrm{C}_1) \cap V(\mathrm{C}_2) \setminus \big(V^{global} \cup \{t, t^-\}\big) = \emptyset$ by Def. 2.
5. Similarly, Lemma 2 removes unconnected ports in_2^* of component C_2, where $\alpha \overset{\text{def}}{\equiv} in_1^*$, $\beta \overset{\text{def}}{\equiv} in_2^*$ and $A \overset{\text{def}}{\equiv} [cp_1]\varphi_1$. Again, there is no flow from β to α ($\mathrm{FV}(\alpha) \cap \mathrm{BV}(\beta) = \emptyset$) since $\mathrm{FV}(in_1^*) \subseteq V(\mathrm{C}_1)$ and $\mathrm{BV}(in_2^*) \subseteq V_2^{in}$ (and thus $\mathrm{BV}(cp_2) \subseteq V(\mathrm{C}_2) \setminus \big(V^{global} \cup \{t, t^-\}\big)$) are disjoint. Furthermore, property A is not influenced by program β ($\mathrm{FV}(A) \cap \mathrm{BV}(\beta) = \emptyset$) since $\mathrm{FV}([cp_1]\varphi_1) \cap (V(\mathrm{C}_1) \cup V(\mathrm{C}_2)) \subseteq V(\mathrm{C}_1)$ and $V(\mathrm{C}_1) \cap V(\mathrm{C}_2) \subseteq \big(V^{global} \cup \{t, t^-\}\big)$.

S4: Re-introduce input assumptions. Next, we prepare for disconnecting inputs by introducing tests that check the input assumptions guaranteed by communication programs. Steps 6–9 are repeated for each communication program $con(v_j)$.

$$\begin{array}{l|l}
 & \hspace{6em} * \\ \hline
\wedge\text{l,id} & \varphi_1, \varphi_2, \bigwedge_{v \in \mathcal{I}^{\mathcal{X}}} \zeta(v, \mathcal{X}(v)) \vdash \zeta(v_j, \mathcal{X}(v_j)) \\ \hline
\text{V} & \varphi_1, \varphi_2, \bigwedge_{v \in \mathcal{I}^{\mathcal{X}}} \zeta(v, \mathcal{X}(v)) \vdash [\ldots; \cancel{\text{old}(\mathcal{X}(v_j)) := \mathcal{X}(v_j)}; \ldots; \cancel{\text{old}(v_j) := v_j}; \ldots]\zeta(v_j, \mathcal{X}(v_j)) \\ \hline
\text{C,[:=]} & \varphi_1, \varphi_2, \bigwedge_{v \in \mathcal{I}^{\mathcal{X}}} \zeta(v, \mathcal{X}(v)) \vdash [\ldots; \text{old}(\mathcal{X}(v_j)) := \mathcal{X}(v_j); \ldots; \text{old}(v_j) := v_j; \ldots]\zeta(\text{old}(v_j), \text{old}(\mathcal{X}(v_j))) \\ \hline
\text{def} & \varphi_1, \varphi_2, \zeta \vdash [\Delta_1; \Delta_2]\zeta(\text{old}(v_j), \text{old}(\mathcal{X}(v_j))) \\ \hline
\text{def} & \varphi_1, \varphi_2, \zeta \vdash [\Delta_3]\zeta(\text{old}(v_j), \text{old}(\mathcal{X}(v_j))) \\ \hline
\text{B,L. 2} & \varphi_1, \varphi_2, \zeta \vdash [\Delta_3][\mathit{ctrl}_1; \mathit{ctrl}_2][t^- := t][\{t' = 1, \mathit{plant}_3\}][\mathit{in}_1^*]\zeta(\text{old}(v_j), \text{old}(\mathcal{X}(v_j))) \\ \hline
\text{A,(46)} & \varphi_1, \varphi_2, \zeta \vdash [\Delta_3][\mathit{ctrl}_1; \mathit{ctrl}_2][t^- := t][\{t' = 1, \mathit{plant}_3\}][\mathit{in}_1^*][\mathit{con}(v_j)]\left(\pi_2^{\mathit{out}}(\mathcal{X}(v_j)) \to \pi_1^{\mathit{in}}(v_j)\right) \\ \hline
 & \text{⑦ (Fig. 11) continued}
\end{array}$$

Figure 13: Use connection compatibility to verify the side condition of Lemma 6.

6. The tactic extracts the leftmost communication program $\mathit{con}(v_j)$ from con. The program con^* denotes the remaining communication programs.
7. Lemma 5 uses F_2^{out} to insert a test $?\pi_2^{\mathit{out}}(\mathcal{X}(v_j))$.
8. Lemma 1 then sorts the new test $?\pi_2^{\mathit{out}}(\mathcal{X}(v_j))$ after the communication program $\mathit{con}(v_j)$. The lemma is applicable because in_1^* and $\mathit{con}(v_j)$ write only input variables from C_1, while $\pi_2^{\mathit{out}}(\mathcal{X}(v_j))$ reads only variables from C_2, so altogether $\text{BV}(\mathit{in}_1^*; \mathit{con}(v_j)) \cap \text{FV}(\pi_2^{\mathit{out}}(\mathcal{X}(v_j))) = \emptyset$.
9. Lemma 6 then relaxes the test of output guarantees to the potentially weaker input assumptions $?\pi_1^{\mathit{in}}(v_j)$ of the communication program, which will allow us to later disconnect the port. The condition of Lemma 6 that $F \to G$ is valid is justified from the compatibility proof (38) by the steps in Fig. 13.
   A The compatibility proof for (38) simplifies compatibility $\pi_2^{\mathit{out}}(\mathcal{X}(v_j)) \to \pi_1^{\mathit{in}}(v_j)$ of communication program $\mathit{con}(v_j)$ to the communication guarantee $\zeta(\text{old}(v_j), \text{old}(\mathcal{X}(v_j)))$.
   B The communication guarantee is now phrased over initial $\text{old}(\mathcal{X}(v_j)) \in V_2^-$ and $\text{old}(v_j) \in V_1^-$ that are written neither in controllers nor plants, since Def. 4 requires that port memory storage V^- is not modified $(\text{BV}(\mathit{ctrl}) \cup \text{BV}(\mathit{plant}) \cup \text{BV}(\mathit{cp})) \cap V^- = \emptyset$. Therefore, Lemma 2 allows us to drop all controllers and plants except the port memory Δ_3.
   C The condition closes from straightforward assignments by axiom [:=] of the port memory Δ_3 and V. □

S5: Disconnect inputs. This step disconnects the component inputs by replacing communication programs with a nondeterministic overapproximation.

10. Now that the input assumptions $?\pi_1^{\mathit{in}}(v_j)$ are in place, Lemma 4 disconnects the input by replacing the communication program $\mathit{con}(v_j)$ with a nondeterministic assignment $v_j := *$.
11. The input with its assumptions is now appended to in_1^*. We repeat steps 10 and 11 for every connected port v_j. Afterwards, we use Lemma 1 to adjust the order of input assignments until we get the shape $\mathit{in}_1; \mathit{cp}_1$.

S6: Drop plants and controllers. Now that the component C_1 is disconnected, we drop all other components.

12. Lemma 3 drops the plant plant_2 of component C_2, since it no longer influences component C_1, i. e., $\text{FV}([\mathit{in}_1][\mathit{cp}_1]\varphi_1) \cap \text{BV}(\mathit{plant}_2) = \emptyset$.
13. Lemma 1 sorts the port memories Δ_3 such that memories Δ_1 of C_1 precede the memories Δ_2 of C_2.
14. Finally, Lemma 2 removes the controller ctrl_2 and port memory Δ_2 of component C_2 to get the shape (41), which concludes the induction step of component C_1.

The tactic for the induction step of component C_2 works in a similar manner, using φ_2 in place of φ_1. □

Throughout the proof, the assignments in vector-valued ports are kept together; the dimension of a vector-valued port indicates how many programs to move simultaneously.

6 Case Studies

To evaluate our approach[3], we use the running example of a remote-controlled robot (RC robot) and revisit prior case studies on the European Train Control System (ETCS) [37], two-component robot collision avoidance (Robix) [21], and adaptive cruise control (LLC) [17], see Fig. 14. In ETCS, a radio-block controller (RBC) communicates speed limits to a train, i.e., it requires the train to have at most speed d after some point m. The RBC vector-valued change contract relates distances m, m^- and demanded speeds d, d^- in input assumptions/output guarantees of the form $d \geq 0 \wedge (d^-)^2 - d^2 \leq 2b(m - m^-) \wedge \mathit{state} = \mathit{drive}$, thus avoiding physically impossible maneuvers.

[3] Implementation and full models available online at `http://www.cs.cmu.edu/~smitsch/resource/components`

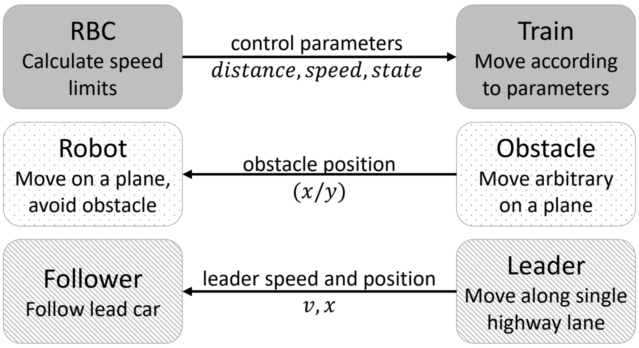

Figure 14: Case Studies: Components and communication

In *Robix*, a robot measures the position of a moving obstacle with a maximum speed S. The obstacle guarantees to not move further than $S \cdot (t - t^-)$ in either axis between measurements, using a rate contract.

In *LLC*, a follower car measures both speed v_l and position x_l of a leader car, with maximum acceleration A and braking capabilities B. Hence, we use a vector-valued port rate contract with properties of the form $2 \cdot (x_l - x_l^-) \geq v_l + v_l^- \cdot t \wedge 0 \leq v_l \wedge -B \cdot t \leq v_l - v_l^- \leq A \cdot t$ tying together speed change and position progress.

6.1 Results

Table 2 summarizes the experimental results of the component-based approach in comparison to monolithic models in terms of duration and degree of proof automation. The column *Contract* describes the kind of contract used in the case study (i. e., vector-valued ports, rate contract or change contract), as well as whether or not the models use non-linear differential equations. The column *Automation* indicates fully automated proofs with checkmarks; it indicates the number of built-in tactics composed to form a proof script when user input is required. The column *Duration* compares the proof duration, using Z3 [23] as a back-end decision procedure to discharge arithmetic. Note that the runtime of proofs where user input was required is highly dependent on the used proof script. For comparable numbers, all proof scripts were created by a single user in a similar style. The column *Sum* sums up the proof durations for the components (columns C_1 and C_2) and Theorem 1 (column *Th. 1*, i. e., checking compatibility and the execution of our composition proof). Checking the composition proof is fully automated, following the proof steps of Theorem 1. All measurements were conducted on an Intel i7-6700HQ CPU@2.6 GHz with 16GB memory.

In summary, we observe that component-based verification uses less tedious interactive proving (see Robix and LLC) and may even improve proof checking performance to some extent, since it can help reduce the combinatorial explosion that may occur in monolithic models when nondeterministic choices are composed sequentially (see ETCS).

6.2 Discussion

Tele-operated Robot (RC Robot). The tele-operated robot, which was used as a running example throughout this article, uses rate contracts and was—due to its trivial models—solved automatically. The proof for the monolithic system took more than 10 times longer than for the component-based version using our approach.

European Train Control System (ETCS). The ETCS used a vector-valued change contract, which was verified automatically using KeYmaera X. Even though the train component C_2 took much longer than the controller component C_1, the whole proof duration of our component-based approach is almost 20 times shorter than for the monolithic approach.

Robot Collision Avoidance (Robix). The case study used rate contracts and required the use of non-linear ODEs to model the robot's motion. Here both versions—the component-based version (i. e., the robot component) and the monolithic version—required manual guidance to complete the proof. While the proofs for the obstacle component and the theorem (including side conditions) finished automatically, the proof for the robot used 31 manual tactic applications. The monolithic proof used more than three times as many manual steps. Besides reducing tedious manual work, the decomposition effect is also reflected in reduced proof checking duration.

Adaptive Cruise Control (LLC). LLC used a vector-valued rate contract, where neither the follower component nor the monolithic system were verified automatically. The number of manual proof steps for the monolithic approach is reduced to less than 50% in the component-based approach. Note that the proof duration for the component-based case is about 30% higher than for the monolithic case. This is caused by several factors: (i) As the monolithic model is rather small, there is no significant dimension reduction when using the component-based case; on the contrary, our approach introduces additional variables (e. g., plant start time). (ii) The monolithic approach uses tricks to reduce the variable number, which are not applicable in the component-based case.

Summary. In summary, the results indicate that our approach verification can lead to performance improvements and smaller user-provided proof scripts. As long as proof automation of KeYmaera X was able to verify the contracts, our component-based approach outperformed the monolithic approach, see RC Robot and

Table 2: Experimental results for case studies

		Contract				**Automation**				**Duration** $[s]$				
		Multi	Change	Rate	Non-linear	C_1	C_2	Th. 1	Mono-lithic	C_1	C_2	Th. 1	Sum	Mono-lithic
RC Robot				✓		✓	✓	✓	✓	32	101	56	**189**	**1934**
ETCS	[37]	✓	✓			✓	✓	✓	✓	127	608	179	**873**	**15306**
Robix	[21]			✓	✓	(31)	✓	✓	(96)	469	117	132	**718**	**902**
LLC	[17]	✓		✓		✓	(50)	✓	(131)	135	351	267	**753**	**568**

ETCS. As soon as manual proof steps (i. e., manual application of tactics) are required, the proof duration of the component-based approach and the monolithic approach are almost equal. However, the number of manual steps needed to verify the contracts is reduced when using our component-based approach.

7 Related Work

We group related work into hybrid automata, hybrid process algebras, and hybrid programs.

Hybrid Automata and Assume-Guarantee Reasoning. Parallel composition of hybrid automata leads to an exponential product automaton because the associated verification procedure for its safety properties is not compositional [1]. Thus, for a hybrid automaton it is not sufficient to establish a property about its parts in order to establish a property about the automaton. We, instead, decompose *verification* into local proofs and get system safety automatically. Hybrid I/O automata [19] extend hybrid automata with a notion of external behavior. The associated implementation relation (i. e., if automaton A implements automaton B, properties verified for B also hold for A) is respected by their composition operation in the sense that if A_1 implements A_2, then the composition of A_1 and B implements the composition of A_2 and B. Similarly, approximate bi-simulation allows abstraction of models and is compositional for a synchronous composition operator [11]. Hybrid (I/O) automata are mainly verified using reachability analysis.

Therefore, techniques to prevent state-space explosion are needed, such as assume-guarantee reasoning (AGR, e. g., [4,9,14]), which was developed to decompose a verification task into subtasks. Timed transition systems are used to approximate a component's behavior by discretization [9]. These abstractions are then used in place of the more complicated automata to verify refinement properties, but the implementation is limited to linear hybrid automata. In analogy, we discretize plants to rate contracts; however, in our approach, contracts completely replace components and do not need to retain simplified transition systems.

A similar AGR rule is presented in [14], where the approximation drops continuous behaviors of single components entirely. As a result, the approach only works when the continuous behavior is irrelevant to the verified property, which rarely happens in CPS. Our change and rate contracts still preserve knowledge about continuous behavior.

The AGR approach of [4] uses contracts consisting of input assumptions and output guarantees to verify properties about single components: a component is an abstraction of another component if it has a stricter contract. The approach is restricted to constant intervals, i. e., static global contracts as in [25].

In [7], a component-based design framework for controllers of hybrid systems with linear dynamics based on hybrid automata is presented. It focuses on checking interconnections of components: alarms propagated by an out-port must be handled by the connected in-ports. We, too, check component compatibility, but for contracts, and focus on transferring proofs from components to the system level. We provide parallel composition, while [7] uses sequential composition.

The compositional verification approach in [2] bases on linear hybrid automata using invariants to over-approximate component behavior and interactions. However, interactions between components are restricted to synchronization (i. e., no variable state can be transferred between components).

In summary, aforementioned approaches are limited to linear dynamics [7] or linear hybrid automata [2], use global contracts [4], focus on sequential composition [7] or rely on reachability analysis, over-approximation and model checking [4,9,14]. We, in contrast, focus on *theorem proving* in dL, using change and rate contracts and handle *non-linear dynamics* and parallel composition. Most crucially, we transfer local safety responsibilities from components to safety of whole systems using tactics without needing soundness-critical prover extensions, whereas related approaches require safety-critical extensions and work on the complementary question of property transfer between different levels of abstraction [4,9,14] (which dL handles by refinement [16]).

Hybrid Process Algebras. Hybrid process algebras are compositional modeling formalisms for the description of behavior and interaction of processes, based on algebraic equations. Examples are Hybrid χ [39], HyPA [6] or the Φ-Calculus [40]. Although the modeling is compositional, for verification purposes the models are again analyzed using simulation or reachability analysis in a non-compositional fashion (e. g., Hybrid χ using PHAVer [45], HyPA using HyTech [20], Φ-Calculus using SPHIN [43]), while we focus on exploiting compositionality in the proof.

Hybrid Programs. Quantified hybrid programs enable a compositional verification of hybrid systems with an arbitrary number of components [31] of a homogeneous structure (e. g., many cars, or many robots). They were used to split monolithic hybrid program models into smaller parts to show that adaptive cruise control prevents collisions for an arbitrary number of cars on a highway [17]. We focus on components of different shapes. Similarly, the approach in [24] presents a component-based approach limited to traffic flow and global contracts.

Our approach extends prior work [25], which was restricted to contracts over constant ranges. Such global contracts are well-suited for certain use cases, where the change of a port's value does not matter for safety, such as the traffic flow models of [24]. However, for systems such as the remote-controlled robot obstacle avoidance from our running example (cf. Section 4.1), which require knowledge about the change of certain values, global contracts only work for considerably more conservative models (e. g., robot and obstacle must stay in fixed globally known regions, since the obstacle's last position is unknown). Change and rate contracts allow more liberal component interaction.

Focusing on architectural properties, [41] proposes a component-based modeling approach for hybrid-systems. Although they do not transfer verification results from components to composites, their component definitions have inspired our own definitions.

Alternative parallel composition operations for dL [18] use nondeterministic choices between all controllers and the parallel plants, which is a good fit if all control operations are nondeterministic. Such a composition operation gets commutativity and associativity for free from the underlying dL operator for nondeterministic choice, but requires careful user modeling to respect variable restrictions and preserve controller executability after composition, which we get by design of the composition operation. Furthermore, their approach aims at composition of contracts with restrictions on variables, but does not consider compatibility of ports. We, on the other hand, use interfaces to explicitly designate inputs and outputs together with their behavior, which facilitates reuse of components and allows a notion of compatibility. Most crucially, for safety verification [18] introduces new proof rules for the parallel composition operator, which must be trusted for soundness. Our tactic to produce system safety proofs, in contrast, is a syntactic decomposition on the shape of programs and therefore inherits soundness for free from the KeYmaera X prover kernel.

8 Conclusion

Component-based modeling makes sense for complicated systems, but only really pays off if accompanied by component-based verification. Just as component-based modeling splits big systems into smaller components, component-based verification splits monolithic system verification into proofs about components with local responsibilities. This is especially useful if the safety of the composed system follows directly from the safety of the individual components, which is what we identify corresponding sufficient conditions for in this article. The dL basis that we use already provides compositionality for each of its operators, but we now add compositionality at the larger granularity of components.

Our component-based verification leverages component contracts. Change contracts relate a port's previous value to its current value (i. e., the change since the last port transmission), while rate contracts additionally relate to the time passed between measurements. Rate contracts allow the verification of a broader range of systems, but need more time variables. As the number of variables can be crucial for formal verification, change contracts are favorable if applicable.

The safety properties of components that are described by component contracts and verified using KeYmaera X transfer to the composed system without reverification of the entire system. We have shown the applicability of our approach on a running example and three existing case studies, which furthermore demonstrated the potential reduction of verification effort compared to proving monolithic models. We implemented our approach as a KeYmaera X tactic that automatically verifies composite systems from verified component contracts *without increasing the trusted prover core.*

Our experiments have demonstrated an impact on either the time it took KeYmaera X's proof automation to find a proof, or the size of the user-supplied tactic. We used existing case studies for monolithic systems as a basis. The biggest remaining litmus test for component-based verification is the empirical question of whether it is easier or harder for people to *start* with a component-based design compared to a monolithic model.

References

1. Alur, R., Courcoubetis, C., Henzinger, T.A., Ho, P.H.: Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In: Grossman,

R.L., Nerode, A., Ravn, A.P., Rischel, H. (eds.) Hybrid Systems. pp. 209–229. Lecture Notes in Computer Science, Springer (1993)

2. Aştefanoaei, L., Bensalem, S., Bozga, M.: A compositional approach to the verification of hybrid systems. In: Ábrahám, E., Bonsangue, M., Johnsen, B.E. (eds.) Theory and Practice of Formal Methods, vol. 9660, pp. 88–103. Springer (2016)
3. Benveniste, A., Caillaud, B., Ferrari, A., Mangeruca, L., Passerone, R., Sofronis, C.: Multiple viewpoint contract-based specification and design. In: Boer, F.S.d., Bonsangue, M.M., Graf, S., Roever, W.P.d. (eds.) Formal Methods for Components and Objects, 6th International Symposium. pp. 200–225. Lecture Notes in Computer Science, Springer (2007)
4. Benvenuti, L., Bresolin, D., Collins, P., Ferrari, A., Geretti, L., Villa, T.: Assume–guarantee verification of nonlinear hybrid systems with Ariadne. Int. J. Robust Nonlinear Control 24(4), 699–724 (2014)
5. Bornot, S., Sifakis, J.: On the composition of hybrid systems. In: Henzinger, T.A., Sastry, S. (eds.) Hybrid Systems: Computation and Control, First International Workshop, Proceedings. pp. 49–63. Lecture Notes in Computer Science, Springer (1998)
6. Cuijpers, P.J.L., Reniers, M.A.: Hybrid process algebra. J. Log. Algebr. Program. 62(2), 191–245 (2005)
7. Damm, W., Dierks, H., Oehlerking, J., Pnueli, A.: Towards component based design of hybrid systems: Safety and stability. In: Manna, Z., Peled, D.A. (eds.) Time for Verification, Essays in Memory of Amir Pnueli, Lecture Notes in Computer Science, vol. 6200, pp. 96–143. Springer (2010)
8. Felty, A., Middeldorp, A. (eds.): International Conference on Automated Deduction, CADE'15, Berlin, Germany, Proceedings, LNCS, vol. 9195. Springer, Berlin (2015)
9. Frehse, G., Zhi Han, Krogh, B.: Assume-guarantee reasoning for hybrid I/O-automata by over-approximation of continuous interaction. In: 43rd IEEE Conference on Decision and Control, CDC. vol. 1, pp. 479–484 Vol.1 (2004)
10. Fulton, N., Mitsch, S., Quesel, J.D., Völp, M., Platzer, A.: KeYmaera X: An axiomatic tactical theorem prover for hybrid systems. In: Felty and Middeldorp [8], pp. 527–538
11. Girard, A., Pappas, G.J.: Approximation metrics for discrete and continuous systems. IEEE Trans. Automat. Contr. 52(5), 782–798 (2007)
12. Gößler, G., Sifakis, J.: Composition for component-based modeling. Sci. Comput. Program. 55(1-3), 161–183 (2005)
13. Henzinger, T.A.: The theory of hybrid automata. In: Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science. pp. 278–292. IEEE Computer Society (1996)
14. Henzinger, T.A., Minea, M., Prabhu, V.S.: Assume-guarantee reasoning for hierarchical hybrid systems. In: Di Benedetto, Maria Domenica, Sangiovanni-Vincentelli, A.L. (eds.) Hybrid Systems: Computation and Control, 4th International Workshop, Proceedings. Lecture Notes in Computer Science, vol. 2034, pp. 275–290. Springer (2001)
15. Logic in Computer Science (LICS), 2012 27th Annual IEEE Symposium on. IEEE, Los Alamitos (2012)
16. Loos, S.M., Platzer, A.: Differential refinement logic. In: Grohe, M., Koskinen, E., Shankar, N. (eds.) LICS. pp. 505–514. ACM, New York (2016)
17. Loos, S.M., Platzer, A., Nistor, L.: Adaptive cruise control: Hybrid, distributed, and now formally verified. In: Butler, M., Schulte, W. (eds.) FM. LNCS, vol. 6664, pp. 42–56. Springer, Berlin (2011)
18. Lunel, S., Boyer, B., Talpin, J.P.: Compositional proofs in differential dynamic logic dL. In: 17th International Conference on Application of Concurrency to System Design, Proceedings. pp. 19–28. IEEE Computer Society (2017)
19. Lynch, N.A., Segala, R., Vaandrager, F.W.: Hybrid I/O automata. Inf. Comput. 185(1), 105–157 (2003)
20. Man, K.L., Reniers, M.A., Cuijpers, P.J.L.: Case studies in the hybrid process algebra Hypa. Int. J. Software Eng. Knowl. Eng. 15(2), 299–306 (2005)
21. Mitsch, S., Ghorbal, K., Vogelbacher, D., Platzer, A.: Formal verification of obstacle avoidance and navigation of ground robots. I. J. Robotics Res. 36(12), 1312–1340 (2017)
22. Mitsch, S., Platzer, A.: ModelPlex: Verified runtime validation of verified cyber-physical system models. Form. Methods Syst. Des. 49(1-2), 33–74 (2016), special issue of selected papers from RV'14
23. Moura, L.M.d., Bjørner, N.: Z3: An efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, Proceedings. pp. 337–340. Lecture Notes in Computer Science, Springer (2008)
24. Müller, A., Mitsch, S., Platzer, A.: Verified traffic networks: Component-based verification of cyber-physical flow systems. In: 18th International Conference on Intelligent Transportation Systems. pp. 757–764 (2015)
25. Müller, A., Mitsch, S., Retschitzegger, W., Schwinger, W., Platzer, A.: A component-based approach to hybrid systems safety verification. In: Abraham, E., Huisman, M. (eds.) IFM. LNCS, vol. 9681, pp. 441–456. Springer (2016)
26. Müller, A., Mitsch, S., Retschitzegger, W., Schwinger, W., Platzer, A.: A benchmark for component-based hybrid systems safety verification. In: Frehse, G., Althoff, M. (eds.) 4th International Workshop on Applied Verification of Continuous and Hybrid Systems. EPiC Series in Computing, vol. 48, pp. 65–74. EasyChair (2017)
27. Müller, A., Mitsch, S., Retschitzegger, W., Schwinger, W., Platzer, A.: Change and delay contracts for hybrid system component verification. In: Huisman, M., Rubin, J. (eds.) FASE. LNCS, vol. 10202, pp. 134–151. Springer (2017)
28. Parnas, D.L.: On the criteria to be used in decomposing systems into modules. Commun. ACM 15(12), 1053–1058 (1972)
29. Platzer, A.: Differential dynamic logic for hybrid systems. J. Autom. Reas. 41(2), 143–189 (2008)
30. Platzer, A.: Differential-algebraic dynamic logic for differential-algebraic programs. J. Log. Comput. 20(1), 309–352 (2010)
31. Platzer, A.: A complete axiomatization of quantified differential dynamic logic for distributed hybrid systems. Log. Meth. Comput. Sci. 8(4:17), 1–44 (2012), special issue for selected papers from CSL'10
32. Platzer, A.: The complete proof theory of hybrid systems. In: LICS [15], pp. 541–550

33. Platzer, A.: Logics of dynamical systems. In: LICS [15], pp. 13–24
34. Platzer, A.: The structure of differential invariants and differential cut elimination. Log. Meth. Comput. Sci. 8(4:16), 1–38 (2012)
35. Platzer, A.: A uniform substitution calculus for differential dynamic logic. In: Felty and Middeldorp [8], pp. 467–481
36. Platzer, A.: A complete uniform substitution calculus for differential dynamic logic. J. Autom. Reas. 59(2), 219–265 (2017)
37. Platzer, A., Quesel, J.D.: European Train Control System: A case study in formal verification. In: Breitman, K., Cavalcanti, A. (eds.) ICFEM. LNCS, vol. 5885, pp. 246–265. Springer, Berlin (2009)
38. Platzer, A., Tan, Y.K.: Differential equation axiomatization: The impressive power of differential ghosts. In: Dawar, A., Grädel, E. (eds.) LICS. pp. 819–828. ACM, New York (2018)
39. Ramon R. H. Schiffelers, D. A. van Beek, Man, K.L., Reniers, M.A., Rooda, J.E.: Formal semantics of Hybrid Chi. In: Larsen, K.G., Niebert, P. (eds.) Formal Modeling and Analysis of Timed Systems: First International Workshop. pp. 151–165. Lecture Notes in Computer Science, Springer (2003)
40. Rounds, W.C., Song, H.: The Phi-Calculus: A language for distributed control of reconfigurable embedded systems. In: Maler, O., Pnueli, A. (eds.) 6th International Workshop on Hybrid Systems: Computation and Control. pp. 435–449. Lecture Notes in Computer Science, Springer (2003)
41. Ruchkin, I., Schmerl, B., Garlan, D.: Architectural abstractions for hybrid programs. In: Kruchten, P., Becker, S., Schneider, J. (eds.) Proceedings of the 18th International Symposium on Component-Based Software Engineering. pp. 65–74. CBSE'15, ACM (2015)
42. Schreiter, L., Bresolin, D., Capiluppi, M., Raczkowsky, J., Fiorini, P., Wörn, H.: Application of contract-based verification techniques for hybrid automata to surgical robotic systems. In: European Control Conference, ECC 2014. pp. 2310–2315. IEEE (2014)
43. Song, H., Compton, K.J., Rounds, W.C.: SPHIN: A model checker for reconfigurable hybrid systems based on SPIN. Electr. Notes Theor. Comput. Sci. 145, 167–183 (2006)
44. UML Revision Task Force: OMG unified modeling language specification, version 2.5: OMG document number: formal/15-03-01, `http://www.omg.org/spec/UML/2.5/`
45. Xinyu, C., Huiqun, Y., Xin, X.: Verification of Hybrid Chi model for cyber-physical systems using PHAVer. In: Barolli, L., You, I., Xhafa, F., Leu, F.Y., Chen, H.C. (eds.) Seventh International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing. pp. 122–128. IEEE Computer Society (2013)

A Formal Semantics

The semantics of hybrid programs α is expressed as a transition relation between states (Def. 9). A differential equation $x' = \theta \,\&\, Q$ can transition between any pair of states connected by a continuous flow φ that respects the differential equations and evolution domain. The set of all states is denoted by $\mathrm{Sta}(\mathcal{V})$. We write $\varphi \models x' = \theta \,\&\, Q$ to mean that φ is a flow of the differential equation $x' = \theta$ contained within the region Q, see [33,36,29] for full details.

Definition 9 (Transition semantics of hybrid programs). The transition relation $[\![\alpha]\!]$ specifies which states ω are reachable from a state ν by operations of α. It is defined as follows:

1. $(\nu, \omega) \in [\![x := \theta]\!]$ iff $\omega(x) = \nu[\![\theta]\!]$, and for all other-variables $z \neq x$, $\omega(z) = \nu(z)$
2. $(\nu, \omega) \in [\![x := *]\!]$ iff $\omega(z) = \nu(z)$ for all variables $z \neq x$
3. $(\nu, \omega) \in [\![?Q]\!]$ iff $\nu = \omega$ and $\nu \models Q$
4. $(\nu, \omega) \in [\![x' = \theta \,\&\, Q]\!]$ iff exists solution $\varphi{:}[0, r] \to \mathrm{Sta}(\mathcal{V})$ for $r \geq 0$ with $\varphi(0) = \nu$, $\varphi(r) = \omega$, and $\varphi \models x' = \theta \,\&\, Q$
5. $[\![\alpha \cup \beta]\!] = [\![\alpha]\!] \cup [\![\beta]\!]$
6. $[\![\alpha; \beta]\!] = \{(\nu, \omega) : (\nu, \mu) \in [\![\alpha]\!], (\mu, \omega) \in [\![\beta]\!],\ \text{exists } \mu\}$
7. $[\![\alpha^*]\!] = [\![\alpha]\!]^*$, the transitive, reflexive closure of $[\![\alpha]\!]$

Definition 10 (Interpretation of dL formulas). Truth of dL formula ϕ in state ν, written $\nu \models \phi$, is defined as follows:

1. $\nu \models \theta_1 \sim \theta_2$ iff $\nu[\![\theta_1]\!] \sim \nu[\![\theta_2]\!]$ for $\sim \in \{=, \leq, <, \geq, >\}$
2. $\nu \models \phi \wedge \psi$ iff $\nu \models \phi$ and $\nu \models \psi$, so on for $\neg, \vee, \to, \leftrightarrow$
3. $\nu \models \forall x\, \phi$ iff $\omega \models \phi$ for all states ω that agree with ν except for the value (in $\mathbb{R}$) of x
4. $\nu \models \exists x\, \phi$ iff $\omega \models \phi$ for some state ω that agrees with ν except for the value (in $\mathbb{R}$) of x
5. $\nu \models [\alpha]\phi$ iff $\omega \models \phi$ for all ω with $(\nu, \omega) \in [\![\alpha]\!]$
6. $\nu \models \langle\alpha\rangle\phi$ iff $\omega \models \phi$ for some ω with $(\nu, \omega) \in [\![\alpha]\!]$

We denote *validity* as $\vDash \phi$, i.e., $\nu \models \phi$ for all states ν.

B Proof Rules

Throughout the article, we use the dL proof rules and axioms listed in Fig. 16, for more details see [36]. Note that casing of formula names is significant: in the nondeterministic choice axiom $[a \cup b]P \leftrightarrow [a]P \wedge [b]P$ the formula P is allowed to mention any variable (even bound variables of programs a and b), whereas in the V axiom $p \to [a]p$ the free variables of formula p must be disjoint from the bound variables of program α. In the nondeterministic assignment axiom $[x := *]p(x) \leftrightarrow \forall x\, p(x)$ the formula $p(x)$ means that x may occur free in $p(x)$ even though it is bound in program $x := *$. Throughout our proofs, we use congruence reasoning with CE [36] to apply lemmas in the context of other formulas.

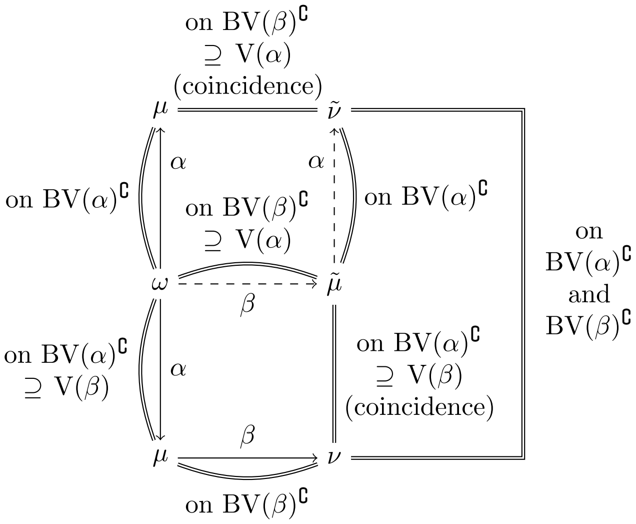
on $\mathrm{BV}(\beta)^{\complement}$
$\supseteq \mathrm{V}(\alpha)$
(coincidence)
μ
$\tilde{\nu}$
α
α
on $\mathrm{BV}(\alpha)^{\complement}$
on $\mathrm{BV}(\beta)^{\complement}$
$\supseteq \mathrm{V}(\alpha)$
on $\mathrm{BV}(\alpha)^{\complement}$
on
$\mathrm{BV}(\alpha)^{\complement}$
and
$\mathrm{BV}(\beta)^{\complement}$
ω
$\tilde{\mu}$
β
on $\mathrm{BV}(\alpha)^{\complement}$
$\supseteq \mathrm{V}(\beta)$
α
on $\mathrm{BV}(\alpha)^{\complement}$
$\supseteq \mathrm{V}(\beta)$
(coincidence)
β
μ
ν
on $\mathrm{BV}(\beta)^{\complement}$

Figure 15: Proof sketch of Lemma 1

C Proofs

Proof of Lemma 1. This proof uses the reachability semantics of dL and shows $[\alpha; \beta]\psi \leftrightarrow [\beta; \alpha]\psi$ by duality from $\langle\alpha; \beta\rangle\psi \leftrightarrow \langle\beta; \alpha\rangle\psi$.

$\langle\alpha; \beta\rangle\psi \to \langle\beta; \alpha\rangle\psi$ Assume $(\omega, \mu) \in [\![\alpha]\!]$ and $(\mu, \nu) \in [\![\beta]\!]$, i.e., $\omega \xrightarrow{\alpha} \mu \xrightarrow{\beta} \nu$. We have to show that there exists $\tilde{\nu} = \nu$ with $(\omega, \tilde{\mu}) \in [\![\beta]\!]$ and $(\tilde{\mu}, \tilde{\nu}) \in [\![\alpha]\!]$, i.e., $\omega \xrightarrow{\beta} \tilde{\mu} \xrightarrow{\alpha} \tilde{\nu}$. Fig. 15 illustrates the proof steps. Note that $\mathrm{BV}(\alpha)^{\complement} \supseteq \mathrm{V}(\beta)$ since $\mathrm{BV}(\alpha) \cap \mathrm{V}(\beta) = \emptyset$, and $\mathrm{BV}(\beta)^{\complement} \supseteq \mathrm{V}(\alpha)$ since $\mathrm{BV}(\beta) \cap \mathrm{V}(\alpha) = \emptyset$.

Since $(\mu, \nu) \in [\![\beta]\!]$ by assumption and $\omega = \mu$ on $\mathrm{BV}(\alpha)^{\complement} \supseteq \mathrm{V}(\beta)$ by the bound effect lemma (cf. [35, Lem. 9]), there exists $(\omega, \tilde{\mu}) \in [\![\beta]\!]$ such that $\tilde{\mu} = \nu$ on $\mathrm{BV}(\beta)$ by the coincidence lemma ([35, Lem. 12]) and $\tilde{\mu} = \omega$ on $\mathrm{BV}(\beta)^{\complement}$ by the bound effect lemma.

Since $(\omega, \mu) \in [\![\alpha]\!]$ by assumption, there exists $(\tilde{\mu}, \tilde{\nu}) \in [\![\alpha]\!]$ such that $\tilde{\nu} = \mu$ on $\mathrm{BV}(\beta)^{\complement} \supseteq \mathrm{V}(\alpha)$ by the coincidence lemma.

Now $\tilde{\nu} = \tilde{\mu} = \nu$ on $\mathrm{BV}(\alpha)^{\complement}$ and also $\tilde{\nu} = \mu = \nu$ on $\mathrm{BV}(\beta)^{\complement}$ and hence we conclude $\tilde{\nu} = \nu$.

$\langle\beta; \alpha\rangle\psi \to \langle\alpha; \beta\rangle\psi$ Follows accordingly.

$[\alpha; \beta]\psi \leftrightarrow [\beta; \alpha]\psi$ follows from $\langle\alpha; \beta\rangle\psi \leftrightarrow \langle\beta; \alpha\rangle\psi$ by $\neg[a]\neg P \leftrightarrow \langle a\rangle P$. □

Proof of Lemma 2. We first show that $[\alpha]A \to [\beta][\alpha]A$ follows immediately from V (i.e., $\phi \to [\gamma]\phi$, if $\mathrm{FV}(\phi) \cap \mathrm{BV}(\gamma) = \emptyset$) with $\phi = [\alpha]A$ and $\gamma = \beta$, since we know that $\mathrm{FV}(A) \cap \mathrm{BV}(\beta) = \emptyset$ and $\mathrm{FV}(\alpha) \cap \mathrm{BV}(\beta) = \emptyset$.

$$
\dfrac{\dfrac{\dfrac{*}{[\alpha]A \vdash [\alpha]A}\,\text{id}}{[\alpha]A \vdash [\beta][\alpha]A}\,\text{V}}{\vdash [\alpha]A \to [\beta][\alpha]A}\,{\to}\text{r}
$$

([;])	$[a;b]P \leftrightarrow [a][b]P$	($\leftrightarrow$r)	$\dfrac{\Gamma,\phi\vdash\psi,\Delta \quad \Gamma,\psi\vdash\phi,\Delta}{\Gamma\vdash\phi\leftrightarrow\psi,\Delta}$	(ind)	$\dfrac{\Gamma\vdash\phi,\Delta \quad \phi\vdash[\alpha]\phi \quad \phi\vdash\psi}{\Gamma\vdash[\alpha^*]\psi,\Delta}$
([$\cup$])	$[a\cup b]P \leftrightarrow [a]P\wedge[b]P$	($\rightarrow$r)	$\dfrac{\Gamma,\phi\vdash\psi,\Delta}{\Gamma\vdash\phi\rightarrow\psi,\Delta}$	(cut)	$\dfrac{\Gamma\vdash C,\Delta \quad \Gamma,C\vdash\Delta}{\Gamma\vdash\Delta}$
([:=])	$[x:=e]p(x)\leftrightarrow p(e)$	($\rightarrow$l)	$\dfrac{\Gamma\vdash\phi,\Delta \quad \Gamma,\psi\vdash\Delta}{\Gamma,\phi\rightarrow\psi\vdash\Delta}$	(id)	$\dfrac{}{\Gamma,P\vdash P,\Delta}$
([?])	$[?Q]P\leftrightarrow(Q\rightarrow P)$	($\wedge$r)	$\dfrac{\Gamma\vdash\phi,\Delta \quad \Gamma\vdash\psi,\Delta}{\Gamma\vdash\phi\wedge\psi,\Delta}$	(M[])	$\dfrac{Q\vdash P}{[a]Q\vdash[a]P}$
([:*])	$[x:=*]p(x)\leftrightarrow\forall x\,p(x)$	($\wedge$l)	$\dfrac{\Gamma,\phi,\psi\vdash\Delta}{\Gamma,\phi\wedge\psi\vdash\Delta}$	(MR)	$\dfrac{\Gamma\vdash[a]Q,\Delta \quad Q\vdash P}{\Gamma\vdash[a]P,\Delta}$
($\langle\cdot\rangle$)	$\neg[a]\neg P\leftrightarrow\langle a\rangle P$	($\neg$r)	$\dfrac{\Gamma,\phi\vdash\Delta}{\Gamma\vdash\neg\phi,\Delta}$	(CER)	$\dfrac{\Gamma\vdash C(Q),\Delta \quad P\leftrightarrow Q}{\Gamma\vdash C(P),\Delta}$
(V)	$p\rightarrow[a]p$	($\neg$l)	$\dfrac{\Gamma\vdash\phi,\Delta}{\Gamma,\neg\phi\vdash\Delta}$	(CEL)	$\dfrac{\Gamma,C(Q)\vdash\Delta \quad P\leftrightarrow Q}{\Gamma,C(P)\vdash\Delta}$
(K)	$[a](P\rightarrow Q)\rightarrow([a]P\rightarrow[a]Q)$	(Wr)	$\dfrac{\Gamma\vdash\Delta}{\Gamma\vdash\phi,\Delta}$	(CE)	$\dfrac{p(\bar{x})\leftrightarrow q(\bar{x})}{C(p(\bar{x}))\leftrightarrow C(q(\bar{x}))}$
(MP)	$P\wedge(P\rightarrow Q)\rightarrow Q$	(Wl)	$\dfrac{\Gamma\vdash\Delta}{\Gamma,\phi\vdash\Delta}$	(GVR)	$\dfrac{\Gamma_{\text{const}}\vdash P,\Delta_{\text{const}}}{\Gamma\vdash[a]P,\Delta}$
([$\cdot$]$\rightarrow\langle\cdot\rangle$)	$\langle a\rangle\top\rightarrow([a]P\rightarrow\langle a\rangle P)$				
([]$\wedge$)	$[a](P\wedge Q)\leftrightarrow[a]P\wedge[a]Q$				
($\forall$i)	$(\forall x\,p(x))\rightarrow p(e)$				

$$\text{(DR)}\quad [x'=f(x)\&q(x)]r(x)\rightarrow\big([x'=f(x)\&r(x)]p(x)\rightarrow[x'=f(x)\&q(x)]p(x)\big)$$

$$\text{(DG)}\quad [x'=f(x)\&q(x)]p(x)\leftrightarrow\forall y\,[x'=f(x),y'=a(x)y+b(x)\&q(x)]p(x)$$

Figure 16: Proof Rules, see [36]

Formula $[\alpha]A\rightarrow[\alpha][\beta]A$ follows in a similar manner from monotonicity M[] and V since $\text{FV}(A)\cap\text{BV}(\beta)=\emptyset$.

$$\to\text{r}\,\dfrac{\text{M[]}\,\dfrac{\text{V}\,\dfrac{\text{id}\,\dfrac{*}{A\vdash A}}{A\vdash[\beta]A}}{[\alpha]A\vdash[\alpha][\beta]A}}{\vdash[\alpha]A\rightarrow[\alpha][\beta]A}$$

□

Proof of Lemma 3. The first step uses differential refinement (DR) [38] to remove Q from the evolution domain. Then the tactic checks each differential equation individually and drops the ones in $y'=\eta$ one by one, by introducing a universal quantifier (∀i) for the respective y and applying the differential ghost axiom[4] in the unusual inverse direction (DG). This step by step removal is necessary until vectorial $x'=\theta$ are introduced into KeYmaera X.

$$\text{DR}\,\dfrac{\forall\text{i}\,\dfrac{\text{DG}\,\dfrac{\text{id}\,\dfrac{*}{[\{x'=\theta\ \&\ H\}]A\vdash[\{x'=\theta\ \&\ H\}]A}}{[\{x'=\theta\ \&\ H\}]A\vdash\forall y\,[\{x'=\theta,y'=\eta\ \&\ H\}]A}}{[\{x'=\theta\ \&\ H\}]A\vdash[\{x'=\theta,y'=\eta\ \&\ H\}]A}}{[\{x'=\theta\ \&\ H\}]A\vdash[\{x'=\theta,y'=\eta\ \&\ H\wedge Q\}]A}$$

The DR axiom requires the verification of an additional side condition

$$\vdash[\{x'=\theta,y'=\eta\ \&\ H\wedge Q\}]H$$

which closes immediately, since H holds throughout the continuous evolution and thus also after the evolution has stopped. □

Proof of Lemma 4. The proof rule GVR abstracts a hybrid program by universally quantifying all bound variables.

$$\to\text{r}\,\dfrac{[{:}*]\,\dfrac{\text{GVR}\,\dfrac{\forall\text{i}\,\dfrac{\text{id}\,\dfrac{*}{A(x)\vdash A(x)}}{\forall x\,A(x)\vdash A(x)}}{\forall x\,A(x)\vdash[\alpha]A(x)}}{[x:=*]A(x)\vdash[\alpha]A(x)}}{\vdash[x:=*]A(x)\rightarrow[\alpha]A(x)}$$

□

Proof of Lemma 5. We prove both directions of the equivalence independently.

$$\to\text{r}\,\dfrac{\leftrightarrow\text{r}\,\dfrac{\wedge\text{r}\,\dfrac{\dots④ \qquad \dots⑤}{[\alpha]F\vdash([\alpha;?F]A\rightarrow[\alpha]A)\wedge([\alpha]A\rightarrow[\alpha;?F]A)}}{[\alpha]F\vdash[\alpha;?F]A\leftrightarrow[\alpha]A}}{\vdash[\alpha]F\rightarrow([\alpha;?F]A\leftrightarrow[\alpha]A)}$$

The direction $[\alpha;?F]A\rightarrow[\alpha]A$ follows from K, monotonicity M[], and modus ponens MP.

[4] Axiom DG is usually used to augment an ODE for the sake of proving invariants with additional differential equations whose solutions exist long enough. We use it to drop differential equations.

$$\begin{array}{rc}
 & * \\ \cline{2-2}
\text{MP} & F, F \to A \vdash A \\ \cline{2-2}
\to\text{r} & F \vdash (F \to A) \to A \\ \cline{2-2}
\text{M[]} & [\alpha]F \vdash [\alpha]\left((F \to A) \to A\right) \\ \cline{2-2}
\text{K} & [\alpha]F \vdash [\alpha]\left(F \to A\right) \to [\alpha]A \\ \cline{2-2}
[?] & [\alpha]F \vdash [\alpha][?F]A \to [\alpha]A \\ \cline{2-2}
[;] & [\alpha]F \vdash [\alpha; ?F]A \to [\alpha]A \\ \cline{2-2}
 & \text{④ continued}
\end{array}$$

The direction $[\alpha]A \to [\alpha; ?F]A$ takes similar steps.

$$\begin{array}{rc}
 & * \\ \cline{2-2}
\text{id} & F, A \vdash A \\ \cline{2-2}
\to\text{r} & F \vdash A \to (F \to A) \\ \cline{2-2}
\text{M[]} & [\alpha]F \vdash [\alpha]\left(A \to (F \to A)\right) \\ \cline{2-2}
\text{K} & [\alpha]F \vdash [\alpha]A \to [\alpha]\left(F \to A\right) \\ \cline{2-2}
[?] & [\alpha]F \vdash [\alpha]A \to [\alpha][?F]A \\ \cline{2-2}
[;] & [\alpha]F \vdash [\alpha]A \to [\alpha; ?F]A \\ \cline{2-2}
 & \text{⑤ continued}
\end{array}$$

□

Proof of Lemma 6. The lemma is verified as a derived axiom in KeYmaera X. Since the formulas F, G and A, and the program constant α are not restricted to specific variables, the tactic simplifies to propositional transitivity.

$$\begin{array}{rc}
 & * \\ \cline{2-2}
 & F \to G, G \to A \vdash F \to A \\ \cline{2-2}
[?] & F \to G, [?G]A \vdash [?F]A \\ \cline{2-2}
\wedge\text{l} & (F \to G) \wedge [?G]A \vdash [?F]A \\ \cline{2-2}
\to\text{r} & \vdash \left((F \to G) \wedge [?G]A\right) \to [?F]A
\end{array}$$

□

D Corollaries for Implementation Purposes

For tactic implementation purposes, the following corollaries to Lemma 1 and Lemma 6 are useful since they are implementable as derived axioms in KeYmaera X and therefore faster to use than their more general tactics counterparts.

Corollary 1 (Reorder Specific Programs). *Let x, y be variables, s, t be terms not mentioning x, y, and F, G be* dL *formulas, $A(x, y)$ a* dL *formula that is allowed to mention x, y free, and p be a* dL *formula not mentioning x, y free. Then the following formulas are valid:*

$$[x := s; y := t]A(x, y) \leftrightarrow [y := t; x := s]A(x, y) \tag{43}$$

$$[x := *; y := *]A(x, y) \leftrightarrow [y := *; x := *]A(x, y) \tag{44}$$

$$[x := *; y := t]A(x, y) \leftrightarrow [y := t; x := *]A(x, y) \tag{45}$$

$$[x := *; ?p]A(x) \leftrightarrow [?p; x := *]A(x) \tag{46}$$

$$[x := s; ?p]A(x) \leftrightarrow [?p; x := s]A(x) \tag{47}$$

$$[?F; ?G]A \leftrightarrow [?G; ?F]A \tag{48}$$

Proof. These formulas can be proved as derived axioms in KeYmaera X. For instance, (43) below is proved using the tactic below:

$$\begin{array}{rc}
 & * \\ \cline{2-2}
 & A(s, t) \vdash A(s, t) \\ \cline{2-2}
[;],[:=],[:=] & [x := s; y := t]A(x, y) \vdash [y := t; x := s]A(x, y) \\ \cline{2-2}
\to\text{r} & \vdash (43)
\end{array}$$

The tactics for all other formulas work accordingly, i. e., perform all assignments, tests and nondeterministic assignments with according instantiations of the resulting all-quantifiers. □

Corollary 2 (Weaken Test in Context). *Let A, F and G be arbitrary* dL *formulas and let α be an arbitrary program. Then this is valid:*

$$\Big(\big([\alpha][?G]A\big) \wedge \big([\alpha]\left(F \to G\right)\big)\Big) \to [\alpha][?F]A \tag{49}$$

Corollary 2 is a consequence of Lemma 6 and allows weakening of a test preceded by an arbitrary program.

Proof. To verify Corollary 2 as derived axioms in KeYmaera X, the tactic uses $[]\wedge$ in the inverse direction followed by M[] to remove the enclosing program α, which then allows application of Lemma 6.

$$\begin{array}{rc}
 & * \\ \cline{2-2}
\text{L. 6} & \left([?G]A\right) \wedge (F \to G) \vdash [?F]A \\ \cline{2-2}
\text{M[]} & [\alpha]\left([?G]A \wedge (F \to G)\right) \vdash [\alpha][?F]A \\ \cline{2-2}
[]\wedge & [\alpha][?G]A \wedge [\alpha]\left(F \to G\right) \vdash [\alpha][?F]A \\ \cline{2-2}
\to\text{r} & \vdash (49)
\end{array}$$

□

$$
\dfrac{
\dfrac{
\text{def}\;\dfrac{
\wedge\text{l},\wedge\text{r}\;\dfrac{
\text{Wl}\;\dfrac{(40)\;\dfrac{*}{t = t^-, \phi_1 \vdash \varphi_1}}{t = t^-, \phi_1, \phi_2, \zeta \vdash \varphi_1}
\qquad
\text{Wl}\;\dfrac{(40)\;\dfrac{*}{t = t^-, \phi_2 \vdash \varphi_2}}{t = t^-, \phi_1, \phi_2, \zeta \vdash \varphi_2}
\qquad
\text{Wl}\;\dfrac{\text{id}\;\dfrac{*}{\zeta \vdash \zeta}}{t = t^-, \phi_1, \phi_2, \zeta \vdash \zeta}
}{t = t^- \wedge \phi_1 \wedge \phi_2 \wedge \zeta \vdash \varphi_1 \wedge \varphi_2 \wedge \zeta}
}{t = t^- \wedge \phi_3 \vdash \varphi_3}
}{}
}{\text{Base case Fig. 9 continued}}
$$

Figure 17: Proof steps: Verify induction base case

$$
\dfrac{
\text{def}\;\dfrac{
\wedge\text{l}\;\dfrac{
\wedge\text{r}\;\dfrac{
\text{Wl}\;\dfrac{(42)\;\dfrac{*}{\varphi_1 \vdash \psi_1^{safe} \wedge \mathit{\Pi}_1^{out}}}{\varphi_1, \varphi_2, \zeta \vdash \psi_1^{safe} \wedge \mathit{\Pi}_1^{out}}
\qquad
\text{Wl}\;\dfrac{(42)\;\dfrac{*}{\varphi_2 \vdash \psi_2^{safe} \wedge \mathit{\Pi}_2^{out}}}{\varphi_1, \varphi_2, \zeta \vdash \psi_2^{safe} \wedge \mathit{\Pi}_2^{out}}
}{\varphi_1, \varphi_2, \zeta \vdash \left(\psi_1^{safe} \mathit{\Pi}_1^{out}\right) \wedge \left(\psi_2^{safe} \wedge \mathit{\Pi}_2^{out}\right)}
}{\varphi_1 \wedge \varphi_2 \wedge \zeta \vdash \left(\psi_1^{safe} \wedge \mathit{\Pi}_1^{out}\right) \wedge \left(\psi_2^{safe} \wedge \mathit{\Pi}_2^{out}\right)}
}{\varphi_3 \vdash \left(\psi_3^{safe} \wedge \mathit{\Pi}_3^{out}\right)}
}{\text{Use case Fig. 9 continued}}
$$

Figure 18: Proof steps: Verify induction use case