Change and Delay Contracts for Hybrid System Component Verification

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

February 2017
CMU-CS-17-100
JKU-CIS-2017-01

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

[1]School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA
[2]Department of Cooperative Information Systems, Johannes Kepler University, Linz, Austria

A conference version of this report has appeared at FASE 2017 [17].
Andreas Müller, Stefan Mitsch, Werner Retschitzegger, Wieland Schwinger and André Platzer. Change and Delay Contracts for Hybrid System Component Verification. In, 20th International Conference on Fundamental Approaches to Software Engineering, FASE, Uppsala, Sweden, Proceedings, LNCS. Springer, 2017.

This material is based on research sponsored by DARPA under agreement DARPA FA8750-12-2-0291, and by the Austrian Science Fund (FWF) P28187-N31.

Keywords: component-based development, hybrid systems, formal verification

Abstract

In this paper, we present reasoning techniques for a component-based modeling and verification approach for hybrid systems comprising discrete dynamics as well as continuous dynamics, in which the components have local responsibilities. Our approach supports component contracts (i. e., input assumptions and output guarantees of interfaces) that are more general than previous component-based hybrid systems verification techniques in the following ways: We introduce *change contracts*, which characterize the magnitude of change to describe how current values exchanged between components along ports relate to previous values. We also introduce *delay contracts*, which characterize the rate of change to describe how change is related to the duration between value exchanges. Together, these contracts can take into account what has changed between two components in a given amount of time since the last exchange of information. Most crucially, we prove that the safety of compatible components implies safety of the composed system. The proof steps of the theorem are also implemented as a tactic in KeYmaera X, allowing automatic generation of a KeYmaera X proof for the composite system from proofs of the concrete components.

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 [8], hybrid programs [22]) 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.

Since complex systems are typically composed of multiple subsystems and interact with other systems in their environment, it stands to reason to apply *component-based modeling* and split the analysis into isolated questions about subsystems and their interaction. However, approaches supporting component-based *verification* of hybrid system models are rare and differ strongly in the supported class of problems (cf. Section 5). Component-based techniques for hybrid (I/O) automata are based on *assume-guarantee reasoning* (AGR) [3, 6, 9] and focus on reachability analysis. Complementarily, hybrid systems theorem proving provides proofs, which are naturally compositional [21] (improves modularity) and support nonlinear dynamics, but so far limit components [15, 16] to contracts over constant ranges (e. g., speed of a robot is non-negative and at most 10). Such contracts require substantial static independence of components, which does not fit to dynamic interactions frequently found in CPS. For example, one might show that a robot in the kitchen will not collide with obstacles in the physically separated back yard, but nothing can be said about what happens when both occupy the same parts of the space at different times to be safe. We, thus, extend CPS contracts [15, 16] to consider change of values and timing.

In this paper, we introduce a component-based modeling and verification approach, which improves over previous approaches in the following critical ways. We introduce *change contracts* to specify the *magnitude of change* of a variable between two states (e. g., current speed is at most twice the previous speed). We further add *delay contracts* to capture the *rate of change* between the states (e. g., current speed is at most previous speed increased by accelerating for some time ε). Together, change and delay contracts make the hybrid (continuous-time) behavior of a component available as a discrete-time measurement abstraction in other components. That way, the joint hybrid behavior of a system of components simplifies to analyzing each component separately for safety and for satisfying its contracts, together with checks of compatibility and local side conditions. The isolated hybrid behavior of a component in question is, thus, analyzed with respect to simpler discrete-time abstractions of all other components in the system. We prove that this makes safety proofs about components transfer to the joint hybrid behavior of the composed system built from these compatible components. Moreover, we automate constructing the safety proof of the joint hybrid behavior from component proofs with a proof tactic in KeYmaera X [7]. This enables a small-core implementation [4] of the theory we develop here.

2 Preliminaries: Differential Dynamic Logic

For specifying and verifying correctness statements about hybrid systems, we use *differential dynamic logic* (dL) [21, 24], which supports *hybrid programs* as a program notation for hybrid systems, according to the following EBNF grammar:

$$\alpha ::= \alpha;\beta \mid \alpha \cup \beta \mid \alpha^* \mid x := \theta \mid x := * \mid (x_1' = \theta_1, \ldots, x_n' = \theta_n \,\&\, H) \mid ?H$$

For details on the formal semantics of hybrid programs see [21, 24]. The sequential composition $\alpha;\beta$ expresses that β starts after α finishes. The non-deterministic choice $\alpha \cup \beta$ follows either α or β. The non-deterministic 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 $(x' = \theta \,\&\, H)$ describes a continuous evolution of x (x' denotes derivation with respect to time) within the evolution domain H. The test $?H$ checks that a condition expressed by property H holds, and aborts if it does not. A typical pattern $x := *; ?a \leq x \leq b$, 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 [19]. For example, in this paper, we represent a no-operation statement $\texttt{skip} \equiv ?true$ as a test that always holds.

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 paper is generated by the following EBNF grammar (where $\sim \in \{<, \leq, =, \geq, >\}$ and θ_1, θ_2 are arithmetic expressions in $+, -, \cdot, /$ over the reals):

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

Proof for properties containing non-deterministic repetitions often use *invariants*, representing a property that holds before and after each repetition. Even though there is no unified approach for invariant generation, if a safety property including a non-deterministic repetition is valid, an invariant exists.

We use V to denote a set of variables. $FV(.)$ is used as an operator on terms, formulas and hybrid programs returning the free variables, whereas $BV(.)$ is an operator returning the bound variables.[1] Similarly, $V(.) = FV(.) \cup BV(.)$ returns all variables occurring in terms, formulas and hybrid programs. We use dL in definitions and formulas to denote the set of all dL formulas. 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.

3 Hybrid Components with Change and Delay Contracts

In this section, we extend *components* (defined as hybrid programs) and their *interfaces* [16] with time and delay concepts. Interfaces identify assumptions about component inputs and guarantees

[1] *Bound variables* of a hybrid program are all those that may potentially be written to, while *free variables* are all those that may potentially be read [24].

about component outputs. We define what it means for a component to *comply with its contract* by a dℒ formula expressing safe behavior and compliance with its interface. And we define the *compatibility of component connections* rigorously as dℒ formulas as well. The main result of this paper is a proof showing that the safety properties of components transfer to a composed system, given proofs of contract compliance, compatibility and satisfaction of local side conditions. Users only provide a *specification* of components, interfaces, and how the components are connected, and show *proof obligations* about component contract compliance, compatibility and local side conditions; system safety follows automatically.

3.1 Running Example: Tele-Operated Robot with Collision Avoidance

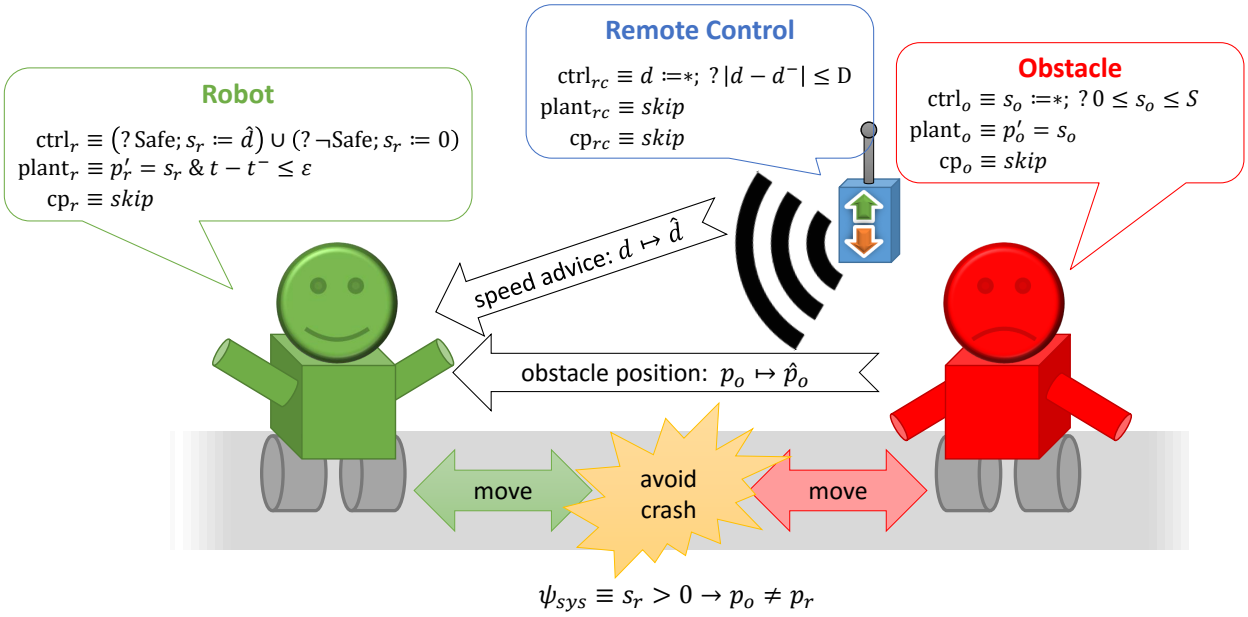


Figure 1: Running Example: Robot receives speed advice and obstacle position, and has to avoid crashes.

To illustrate the concepts, we use a running example of a tele-operated robot with collision avoidance inspired by [10, 13], see Fig. 1. The system consists of three components, cf. Fig. 1:

1. The *remote control* component periodically 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 provides its current position on its single output port p_o. Obstacles include both stationary elements (e. g., a wall) or a moving entities (e. g., a person).
3. The *robot* component reads speed advice on input port $\hat{d}$ and follows the speed advice, if the obstacle position measured on input port $\hat{p}_o$ is at a safe distance.

Two consecutive speed advisories from the RC should be at most D apart (i. e., $|d - d^{-}| \leq D$). The RC issues speed advice to the robot, but has no physical part. The obstacle chooses a new non-negative 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.

The overall system target is to keep the robot from actively colliding with the obstacle, i. e., the robot's and the obstacle's position must not coincide when the robot is driving. Formally, this property can be written as $\psi_{sys}^{safe} \equiv s_r > 0 \rightarrow p_o \neq p_r$. Formal definitions of these three components, their interfaces, and the respective contracts, will be introduced step-by-step along the definitions in subsequent sections.

3.2 Specification: Components and Interfaces

Change components and interfaces specify what a component assumes about the magnitude of change at each of its inputs, and what it guarantees about the magnitude of change on its outputs. To make such conditions expressible, every component will use additional variables to store both the current and the previous value communicated along a port. These so-called Δ-ports can be used to model jumps in discrete control, and for measurement of physical behavior if the rate of change is irrelevant.

3.2.1 Components

Components may consist of a discrete control part and a continuous plant, cf. Def. 1. However, Def. 1 does not prescribe how control and plant are composed; the composition to a hybrid program is specified later in Def. 5. We allow components to be hierarchically composed from sub-components, so components list the internally connected ports of sub-components.

Definition 1 (Component)**.** *A component* $C = (\text{ctrl}, \text{plant}, \text{cp})$ *is defined as*

- ctrl *is the discrete part without differential equations,*
- plant *is a differential equation* $(x_1' = \theta_1, \ldots, x_n' = \theta_n \& H)$ *for* $n \in \mathbb{N}$*,*
- cp *are deterministic assignments connecting ports of sub-components, and*
- $V(C_i) \stackrel{def}{=} V(\text{ctrl}) \cup V(\text{plant}) \cup V(\text{cp})$*, correspondingly for* $BV(C_i)$.

If a component is atomic, i. e., not composed from sub-components, the port connections *cp* are empty (`skip` statement of no effect). The variables of a component are the variables of its controller, plant, and all its sub-components. We aim at components that can be analyzed in isolation and that communicate solely through ports. Global shared constants (read-only and thus not used for communication purposes) are included for convenience to share common knowledge for all components in a single place.

Definition 2 (Variable Restrictions)**.** *A system of components* $C_1, ..., C_n$ *is well-defined if*

- *global variables* V^{global} *are read-only and shared by all components:* $V^{\text{global}} \cap BV(C_i) = \emptyset$*,*
- $\forall i \neq j \,.\, V(C_i) \cap V(C_j) \subseteq V^{\text{global}}$ *such that no variable of other components can be read or written.*

3.2.2 Example: Components

Consider the robot collision avoidance system. Its global variables are the maximum obstacle speed S and the maximum difference D between two speed advisories, i. e., $V^{global} = \{S, D\}$. They can neither be bound in control nor plant of any component, cf. Def. 2.

Example 1 describes the RC component (2). Its controller C_{rc} picks a new speed advisory and ensures that it is not too far from the previous speed advice. Since the RC is an atomic component without any physical characteristics, *plant* and *cp* remain empty, cf. (3)-(4).

Example 1 RC Component

$$\mathrm{C}_{rc} = (\mathit{ctrl}_{rc}, \mathit{plant}_{rc}, \mathit{cp}_{rc}) \tag{1}$$
$$\mathit{ctrl}_{rc} \equiv d := *; ?\left|d - d^{-}\right| \leq D; \tag{2}$$
$$\mathit{plant}_{rc} \equiv \texttt{skip} \tag{3}$$
$$\mathit{cp}_{rc} \equiv \texttt{skip} \tag{4}$$

The obstacle component C_o (cf. Example 2) moves with an arbitrary but limited speed. Thus, the obstacle controller chooses a new non-negative speed s_o limited by the maximum speed S, cf. (6). The obstacle plant adapts the obstacle position according to the chosen speed (i. e., the obstacle moves), cf. (7). The internally connected ports cp_o are empty, since the obstacle is an atomic component, cf. (8).

Example 2 Obstacle Component

$$\mathrm{C}_o = (\mathit{ctrl}_o, \mathit{plant}_o, \mathit{cp}_o) \tag{5}$$
$$\mathit{ctrl}_o \equiv s_o := *; ?\big(0 \leq s_o \leq S\big); \tag{6}$$
$$\mathit{plant}_o \equiv p_o' = s_o \tag{7}$$
$$\mathit{cp}_o \equiv \texttt{skip} \tag{8}$$

The robot component C_r (cf. Example 3) should follow speed advice from the RC and measures the position of the obstacle to avoid collisions.

For the robot, ε is the maximum time that the plant can run. 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 between obstacle and robot (i. e., $\hat{p}_o - p_r$) is greater than the maximum distance that the obstacle can move (i. e., $\hat{d} \cdot \varepsilon$), plus the maximum distance the robot itself can move (i. e., $s_r \cdot \varepsilon$), the robot follows the speed advice of the RC, cf. (10). Otherwise, it stops, cf. (11). The robot's plant (12) adapts the robot's position according to the chosen speed (i. e., the robot moves). The internally connected ports cp_r (13) are empty since the robot is an atomic component.

Example 3 Robot Component

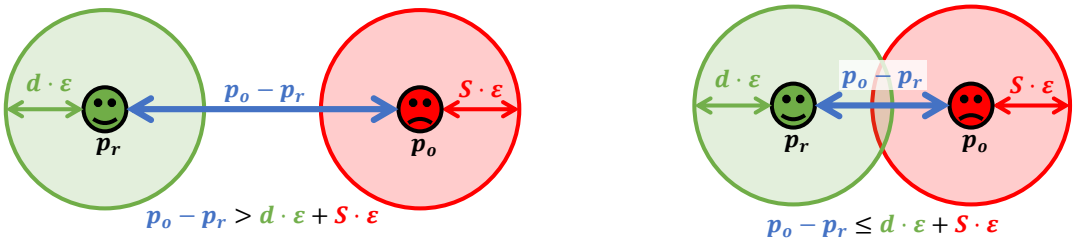
$$\mathrm{C}_r = (ctrl_r, plant_r, cp_r) \tag{9}$$

$$ctrl_r \equiv ?\left(\hat{p}_o - p_r > \left(\hat{d} + S\right) \cdot \varepsilon\right); s_r := \hat{d}; \cup \tag{10}$$

$$?\left(\hat{p}_o - p_r \leq \left(\hat{d} + S\right) \cdot \varepsilon\right); s_r := 0; \tag{11}$$

$$plant_r \equiv p_r' = s_r \tag{12}$$

$$cp_r \equiv \texttt{skip} \tag{13}$$



(a) The robot is safe to move as the areas do not overlap.

(b) The robot stops to avoid imminent collision as indicated by overlapping areas of motion.

Figure 2: The robot only accepts the speed suggestion if it is safe: The green circle represents the area that the robot might reach until the next controller run (i. e., within ε time units) with the suggested speed d. The red circle represents the area that the obstacle might reach during the same interval ε with maximum speed S.

3.2.3 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. 3.

Definition 3 (Admissible Interface)**.** *An* admissible interface I^Δ *for a component* C *is a tuple* $I^\Delta = \left(V^{\text{in}}, \pi^{\text{in}}, V^{\text{out}}, \pi^{\text{out}}, V^\Delta, V^-, pre\right)$ *with*

- $V^{\text{in}} \subseteq V(C)$ *and* $V^{\text{out}} \subseteq V(C)$ *are disjoint sets of input- and output variables,*
- $V^{\text{in}} \cap BV(C) = \emptyset$*, i. e., input variables may not be bound in the component,*
- $\pi^{\text{in}} : V^{\text{in}} \to \mathsf{dL}$ *is a function specifying exactly one formula per input variable (i. e., input port), representing input requirements and assumptions,*
- $\pi^{\text{out}} : V^{\text{out}} \to \mathsf{dL}$ *specifies output guarantees for output ports,*
- $\forall v \in V^{\text{in}} : V(\pi^{\text{in}}(v)) \subseteq \left(V(C) \setminus \left(V^{\text{in}} \cup V^{\text{out}}\right)\right) \cup \{v\}$ *such that input formulas are local to their port,*
- $V^\Delta = V^{\Delta+} \cup V^{\Delta i} \subseteq V(C)$ *is a set of* Δ*-port variables of unconnected public* $V^{\Delta+} \subseteq V^{\text{in}} \cup V^{\text{out}}$*, and connected private* $V^{\Delta i}$*, with* $V^{\Delta i} \cap \left(V^{\text{in}} \cup V^{\text{out}}\right) = \emptyset$*, so* $V^{\Delta+} \cap V^{\Delta i} = \emptyset$*,*

- $V^- \subseteq V(\boldsymbol{C})$ *with* $V^- \cap BV(C) = \emptyset$ *is a read-only set of variables storing the previous values of* Δ*-ports, disjoint from other interface variables* $V^- \cap (V^{\text{in}} \cup V^{\text{out}} \cup V^{\Delta}) = \emptyset$*,*
- *pre* $: V^{\Delta} \to V^-$ *is a bijective function, assigning one variable to each* Δ*-port to store its previous value.*

The definition is accordingly for vector-valued ports that share multiple variables along a single port, provided that each variable is part of exactly one vectorial port. This leads to *multi-ports*, which transfer the values of multiple variables, but have a single joint output guarantee/input assumption over the variables in the multi-port vector. Input assumptions are local to their port, i. e., no input formula can mention other input variables (which lets us reshuffle port ordering) nor any output variables (which prevents cyclic port definitions). Not all ports of a component need to be connected to other components; unconnected ports simply remain input/output ports of the resulting composite system.

3.2.4 Example: Interfaces

Considering our example, we have to define admissible interfaces for the three components, i. e. the RC interface I_{rc}^{Δ}, the obstacle interface I_{o}^{Δ} and the robot interface I_{r}^{Δ}. We start with the RC interface.

The RC controller interface in Example 4 has no input ports, so V_{rc}^{in} and π_{rc}^{in} are empty, cf. (15)-(16). The single output port d provides the speed advice, which is guaranteed to be no further than D from the previous advice, cf. (17)-(18). The output property π_{rc}^{out} limits the magnitude of change between the previous advice d^- and the current advice d, so the port d is a port with initial value d^-, cf. (19)-(21).

Example 4 Remote Control Interface

$$\mathrm{I}_{rc}^{\Delta} = \left(V_{rc}^{in}, \pi_{rc}^{in}, V_{rc}^{out}, \pi_{rc}^{out}, V_{rc}^{\Delta}, V_{rc}^{-}, \mathrm{pre}_{rc}\right) \tag{14}$$
$$V_{rc}^{in} = \{\} \tag{15}$$
$$\pi_{rc}^{in} = () \tag{16}$$
$$V_{rc}^{out} = \{d\} \tag{17}$$
$$\pi_{rc}^{out} = \left(d \mapsto \left|d - d^-\right| \leq D\right) \tag{18}$$
$$V_{rc}^{\Delta} = \{d\} \tag{19}$$
$$V_{rc}^{-} = \{d^-\} \tag{20}$$
$$\mathrm{pre}_{rc} = \left(d \mapsto d^-\right) \tag{21}$$

3.3 Specification: Time and Delay

In a monolithic hybrid program with a combined plant for all components, time passes synchronously for all components and their ODEs evolve for the same amount of time. When split into

separate components, the ODEs are split into separate plants too, thereby losing the connection of evolving for identical amounts of time. From the viewpoint of a single component, other plants reduce to discrete abstractions through input assumptions on Δ-ports. 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). The robot's ODE, however, still runs for some arbitrary time, which makes the measurements and the continuous behavior of the robot drift (i. e., robot and obstacle appear to move for different durations). To address this issue, we introduce *delay* as a way of ensuring that the changes are consistent with the time that passes in a component.

To unify the timing for all components of a system, we introduce a globally synchronized time t and a global variable t^- to store the time before each run of plant. Both are special global variables, which cannot be bound by the user, but only on designated locations specified through the contract, cf. Def. 4.

Definition 4 (Time)**.** *Let $C_i,\ i \in \mathbb{N}$ be any number of components with variables according to Def. 2. When working with* delay contracts*, we assume*

- *the* global system time t *changes with constant rate $t' = 1$,*
- *t^- is the* initial plant time *at the start of the current plant run,*
- *$\{t, t^-\} \cap BV(C_i) = \emptyset$, thus clocks t, t^- are not written by a component.*

3.3.1 Example: Interfaces and Time

Let us continue our running example with the obstacle's and the robot's interfaces, which use the introduced global time in some of their input and output properties.

The obstacle interface (cf. Example 5) has no input ports, cf. (23)-(24). The single output port provides the current obstacle position, which is guaranteed to be in an interval of size $S \cdot (t - t^-)$ centered at the obstacle's previous position, cf. (25)-(26). The output property (27) of p_o captures the rate of change between the previous value p_o^- and the current value p_o by considering global time t. The initial value of p_o is kept in p_o^-, cf. (28)-(29).

Example 5 Obstacle Interface

$$\mathrm{I}_o^\Delta = \left(V_o^{in}, \pi_o^{in}, V_o^{out}, \pi_o^{out}, V_o^\Delta, V_o^-, \mathrm{pre}_o\right) \tag{22}$$
$$V_o^{in} = \{\} \tag{23}$$
$$\pi_o^{in} = () \tag{24}$$
$$V_o^{out} = \{p_o\} \tag{25}$$
$$\pi_o^{out} = \left(p_o \mapsto \left|p_o - p_o^-\right| \leq S \cdot \left(t - t^-\right)\right) \tag{26}$$
$$V_o^\Delta = \{p_o\} \tag{27}$$
$$V_o^- = \{p_o^-\} \tag{28}$$
$$\mathrm{pre}_o = \left(p_o \mapsto p_o^-\right) \tag{29}$$

The robot interface (cf. Example 6) contains two input ports, cf. (31)-(32). On input port $\hat{p}_o$ it receives the obstacle's current position, which is guaranteed to be within the area around the obstacle's previous position. This property describes the rate of change of the obstacle position, instead of a fixed global region. On input port $\hat{d}$ it receives a speed advice, which is guaranteed to be close to the previous value. This property describes the magnitude of change in speed advuce. Thus, a global contract as in [16] would not suffice here. The robot has no output ports, cf. (33)-(34). Both input port are delta ports, cf. (35)-(37).

Example 6 Robot Interface

$$\mathrm{I}_r^{\Delta} = \left(V_r^{in}, \pi_r^{in}, V_r^{out}, \pi_r^{out}, V_r^{\Delta}, V_r^{-}, \mathrm{pre}_r\right) \tag{30}$$

$$V_r^{in} = \{\hat{p}_o, \hat{d}\} \tag{31}$$

$$\pi_r^{in} = \left(\hat{p}_o \mapsto \left|\hat{p}_o - \hat{p}_o^{-}\right| \leq S \cdot \left(t - t^{-}\right), \hat{d} \mapsto \left|\hat{d} - \hat{d}^{-}\right| \leq D\right) \tag{32}$$

$$V_r^{out} = \{\} \tag{33}$$

$$\pi_r^{out} = () \tag{34}$$

$$V_r^{\Delta} = \{\hat{p}_o, \hat{d}\} \tag{35}$$

$$V_r^{-} = \{\hat{p}_o^{-}, \hat{d}^{-}\} \tag{36}$$

$$\mathrm{pre}_r = \left(\hat{p}_o \mapsto p_o^{-}, \hat{d} \mapsto \hat{d}^{-}\right) \tag{37}$$

3.4 Proof Obligations: Change and Delay 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 further shows a local safety property, which describes the component's desired safe states. For example, a safety property of a robot might require that the robot will not drive too close to the last measured position of the obstacle. Together with the obstacle's output guarantee of not moving too far from its previous position, the local safety property implies a system-wide safety property (e. g., robot and obstacle will not collide), since we know that a measurement previously reflected the real position. Contract compliance can be verified using KeYmaera X [7].

In order to make guarantees about the behavior of a composed system we use the synchronized system time t to measure the duration $(t - t^{-})$ between controller runs in delay contract compliance proof obligations, cf. Def. 5.

Definition 5 (Contract Compliance)**.** *Let C be a component with its admissible interface I^{Δ} (cf. Def. 3). Let formula ϕ describe initial states of C and formula ψ^{safe} the safe states, both over the component variables $V(C)$. The output guarantees $\Pi^{\text{out}} \equiv \bigwedge_{v \in V^{\text{out}}} \pi^{\text{out}}(v)$ extend safety to*

$\psi^{\text{safe}} \land \Pi^{\text{out}}$. Change contract compliance $\text{CC}(C, I^\Delta)$ *of C with* I^Δ *is defined as the* dL *formula:*

$$\text{CC}(C, I^\Delta) \overset{def}{\equiv} \phi \to [(\Delta; \text{ctrl}; \text{plant}; \text{in}; \text{cp})^*]\big(\psi^{\text{safe}} \land \Pi^{\text{out}}\big)$$

and delay contract compliance $\text{DC}(C, I^\Delta)$ *is defined as the* dL *formula:*

$$\text{DC}(C, I^\Delta) \overset{def}{\equiv} t = t^- \land \phi \to [\big(\Delta; \text{ctrl}; t^- := t; (t' = 1, \text{plant})\,; \text{in}; \text{cp}\big)^*]\big(\psi^{\text{safe}} \land \Pi^{\text{out}}\big)$$

where

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

are (vectorial) assignments to input ports satisfying input assumptions $\pi^{\text{in}}(v)$ *and* Δ *are (vectorial) assignments storing previous values of* Δ*-port variables:*

$$\Delta \overset{def}{\equiv} \textit{pre}(v) := v \textit{ for all } v \in V^\Delta \ .$$

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, cf. Def. 3. The function pre can be used throughout the component to read the initial value of a Δ-port. Since $\text{pre}(v) \in V^-$ for all $v \in V^\Delta$, Def. 3 and Def. 5 require that the resulting initial variable is not bound anywhere outside Δ.

This notion of contracts crucially changes compared to [16] with respect to where ports are read and how change is modeled: reading from input ports at the beginning of a component's loop body (i. e., before the controller runs) as in [16] may seem intuitive, but it would require severe restrictions to a component's plant in order to make inputs and plant agree on duration. Instead, we prepare the next loop iteration at the end of the loop body (i. e., after *plant*), so that actual plant duration can be considered for computing the next input values.

3.4.1 Example: Change Contract

We continue the collision avoidance system with a change contract (40) according to Def. 5 for the RC from Fig. 1, since the output property relates the current value d to the previous value d^-. Delay is irrelevant, because the RC has no physical part (i. e., $\textit{plant}_{rc} \equiv \texttt{skip}$). The precondition for the RC specifies the bound for the global variable D and bootstraps the output port's previous value d^- from the current demanded speed d, cf. (38). Here, ψ_{rc} comprises only the output port guarantees of the RC, since the RC has no additional safety property, cf. (39). Consequently, the RC guarantees that consecutive speed advisories are at most D apart.

$$\phi_{rc} \equiv D \geq 0 \land d = d^- \tag{38}$$

$$\psi_{rc} \equiv \big|d - d^-\big| \leq D \tag{39}$$

The resulting *change contract* per Def. 5 for the RC was verified using KeYmaera X, cf. (40). We thus know that the component is safe and complies with its interface. Compared to contracts

with fixed ranges as in approaches [3, 16], we do not have to assume a global limit for demanded speeds d, but consider the previous advice d^- as a reference value when calculating the next speed advice.

$$\phi_{rc} \to [(\underbrace{d^- := d}_{\Delta_{rc}}; \underbrace{d := *; ?\left|d - d^-\right| \leq D}_{ctrl_{rc}}; \underbrace{\texttt{skip}}_{plant_{rc}}; \underbrace{\texttt{skip}}_{in_{rc}}; \underbrace{\texttt{skip}}_{cp_{rc}})^*] \underbrace{\left(\left|d - d^-\right| \leq D\right)}_{\Pi^{out}_{rc}} \tag{40}$$

3.4.2 Example: Delay Contract

Change in obstacle position depends on speed and on how much time passed because obstacle positions are related by how much time passes in the ODE $plant_o \equiv p_o' = s_o$. Hence, we follow Def. 5 to specify the obstacle delay contract (43). The precondition for the obstacle specifies the bound for the global variable S, bootstraps the output port's previous value p_o^- from the position p_o and initializes the obstacle speed to 0, cf. (41). Here, ψ_o comprises only the output port guarantees of the obstacle, since our liberal notion of obstacles should not assume obstacles to cooperate for safety, cf. (42). Such an abstraction can be found by solving the plant ODE or from differential invariants [23].

$$\phi_o \equiv S \geq 0 \land p_o = p_o^- \land s_o = 0 \tag{41}$$

$$\psi_o \equiv \left|p_o - p_o^-\right| \leq S \cdot \left(t - t^-\right) \tag{42}$$

The resulting *delay contract* per Def. 5 for the obstacle was verified using KeYmaera X, cf. (43).

$$\begin{aligned} t = t^- \land \phi_o \to [(\overbrace{p_o^- := p_o}^{\Delta_o}; \overbrace{s_o := *; ?(0 \leq s_o \leq S)}^{ctrl_o}; t^- := t; \{t' = 1, \overbrace{p_o' = s_o}^{plant_o}\}; \\ \underbrace{\texttt{skip}}_{in_o}; \underbrace{\texttt{skip}}_{cp_o})*] \underbrace{\left(\left|p_o - p_o^-\right| \leq S \cdot \left(t - t^-\right)\right)}_{\Pi^{out}_o} \end{aligned} \tag{43}$$

Finally, we turn to the delay contract for the robot, according to Def. 5. The precondition specifies the bound for the global variables S and D, 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. max *plant* runtime ε), cf. (44). The safety property for the robot states that the robot's position and the obstacle's position must never coincide, unless the robot is stopped. Since the robot hast no output ports it suffices to verify the safety property ψ_r^{safe}, cf. (45).

$$\phi_r \equiv S \geq 0 \land D \geq 0 \land \hat{p}_o = \hat{p}_o^- \land \hat{d} = \hat{d}^- \land s_r = 0 \land \varepsilon > 0 \tag{44}$$

$$\psi_r \equiv s_r > 0 \to \hat{p}_o \neq p_r \tag{45}$$

The resulting *delay contract* per Def. 5 for the robot was verified using KeYmaera X, see (46).

$$t = t^- \wedge \phi_r \rightarrow [\overbrace{(\hat{p}_o^- := \hat{p}_o; \hat{d}^- := \hat{d}}^{\Delta_r}; \overbrace{?\text{Safe}; s_r := \hat{d} \cup ?\neg\text{Safe}; s_r := 0}^{ctrl_r}; t^- := t; \{t' = 1, \underbrace{p_r' = s_r}_{plant_r}\}; \underbrace{\hat{p}_o := *; ?\pi_r^{in}(\hat{p}_o); \hat{d} := *; ?\pi_r^{in}(\hat{d})}_{in_r}; \underbrace{\texttt{skip}}_{cp_r})*] \underbrace{(s_r > 0 \rightarrow \hat{p}_o \neq p_r)}_{\psi_r} \tag{46}$$

3.5 Proof Obligations: Compatible Parallel Composition

From components with verified contract compliance, we now compose systems in away, that provides safety guarantees about them, without redoing system proofs.

3.5.1 Parallel Composition

For this, Def. 6 introduces a quasi-parallel composition, where the discrete *ctrl* parts of the components are executed sequentially in any order, while the continuous *plant* parts run in parallel. The connected ports *cp* of all components are composed sequentially in any order, since the order of independent deterministic assignments (i. e., assignments having disjoint free and bound variables) is irrelevant. 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. The actual execution order of independent components in a real system is unknown, which we model with a non-deterministic choice between all possible controller execution orders. Values can be exchanged between components using Δ-ports; all other variables are internal to a single component, except global variables, which can be read everywhere, but never bound, and system time t, t^-, which can be read everywhere, but only bound at specific locations fixed by the delay contract, cf. Def. 5. Δ-ports store their previous values in the composite component, regardless if connected or not. For all connected ports, Def. 6 replaces the non-deterministic assignments to open inputs (cf. *in*) with a deterministic assignment from the connected port (cf. *cp*). This represents an instantaneous and lossless interaction between components.

Definition 6 (Parallel Composition)**.** *Let* $C_i = (\text{ctrl}_i, \text{plant}_i, \text{cp}_i)$ *denote components with their corresponding admissible interfaces*

$$I_i^\Delta = \left(V_i^{\text{in}}, \pi_i^{\text{in}}, V_i^{\text{out}}, \pi_i^{\text{out}}, V_i^\Delta, V_i^-, pre_i\right) \text{ for } i \in \{1, \ldots, n\} \ ,$$

sharing only V^{global} *and global times such that* $V(C_i) \cap V(C_j) \subseteq V^{\text{global}} \cup \{t, t^-\}$ *for* $i \neq j$*. Let further*

$$\mathcal{X} : \left(\bigcup_{1 \leq j \leq n} V_j^{\text{in}}\right) \rightharpoonup \left(\bigcup_{1 \leq i \leq n} V_i^{\text{out}}\right) \ , \text{ provided } \mathcal{X}(v) \notin V_j^{\text{out}} \ , \text{ for all } v \in V_j^{\text{in}}$$

be 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 $\mathcal{I}^{\mathcal{X}} = \{x \in V^{\text{in}} \mid \mathcal{X}(x) \text{ is defined}\}$ *and image* $\mathcal{O}^{\mathcal{X}} = \{y \in V^{\text{out}} \mid y = \mathcal{X}(x) \text{ for some } x \in V^{\text{in}}\}$*. The composition of*

n components and their interfaces $(C, I^\Delta) \stackrel{def}{\equiv} \big((C_1, I_1^\Delta) \| \ldots \| (C_n, I_n^\Delta)\big)_{\mathcal{X}}$ according to $\mathcal{X}$ is defined as:

- *controllers are executed in non-deterministic order of all the $n!$ possible permutations of $\{1, \ldots, n\}$,*

$$\begin{aligned} \text{ctrl} \equiv (\text{ctrl}_1; \text{ctrl}_2; \ldots; \text{ctrl}_n) \cup (\text{ctrl}_2; \text{ctrl}_1; \ldots; \text{ctrl}_n) \\ \cup \ldots \cup (\text{ctrl}_n; \ldots; \text{ctrl}_2; \text{ctrl}_1) \end{aligned}$$

- *plants are executed in parallel, with evolution domain $H \equiv \bigwedge_{i \in \{1,\ldots,n\}} H_i$*
  $\text{plant} \equiv \underbrace{x_1^{(1)\prime} = \theta_1^{(1)}, \ldots, x_1^{(k)\prime} = \theta_1^{(k)}}_{\textit{component } C_1}, \ldots, \underbrace{x_n^{(1)\prime} = \theta_n^{(1)}, \ldots, x_n^{(m)\prime} = \theta_n^{(m)}}_{\textit{component } C_n} \;\&\; H \;\;,$
- *port assignments are extended with connections for some $\{v_j, \ldots, v_r\} = \mathcal{I}^{\mathcal{X}}$*

$$\text{cp} \stackrel{def}{\equiv} \underbrace{\text{cp}_1; \text{cp}_2; \ldots; \text{cp}_n}_{\textit{components' cp}}; \underbrace{v_j := \mathcal{X}(v_j); \ldots; v_r := \mathcal{X}(v_r)}_{\textit{connected inputs}} \;\;,$$

- *previous values $V^- \stackrel{def}{=} \bigcup_{1 \le i \le n} V_i^-$ are merged; connected ports become private $V^{\Delta i} \stackrel{def}{=} \big(\bigcup_{1 \le i \le n} V_i^{\Delta i}\big) \cup \mathcal{I}^{\mathcal{X}} \cup \mathcal{O}^{\mathcal{X}}$; unconnected ports remain public $V^{\Delta +} \stackrel{def}{=} \big(\bigcup_{1 \le i \le n} V_i^{\Delta +}\big) \setminus (\mathcal{I}^{\mathcal{X}} \cup \mathcal{O}^{\mathcal{X}})$,*
- *pre_i are combined such that $\text{pre}(v) \equiv \text{pre}_i(v)$ if $v \in V_i^\Delta$ for all $i \in \{1, \ldots, n\}$,*
- *unconnected inputs $V^{\text{in}} = \big(\bigcup_{1 \le i \le n} V_i^{\text{in}}\big) \setminus \mathcal{I}^{\mathcal{X}}$ and unconnected outputs $V^{\text{out}} = \big(\bigcup_{1 \le i \le n} V_i^{\text{out}}\big) \setminus \mathcal{O}^{\mathcal{X}}$ are merged and their requirements preserved*

$$\begin{aligned} \pi^{\text{in}}(v) &\equiv \pi_i^{\text{in}}(v) \textit{ if } v \in V_i^{\text{in}} \setminus \mathcal{I}^{\mathcal{X}} \textit{ for all } i \in \{1, \ldots, n\} \\ \pi^{\text{out}}(v) &\equiv \pi_i^{\text{out}}(v) \textit{ if } v \in V_i^{\text{out}} \setminus \mathcal{O}^{\mathcal{X}} \textit{ for all } i \in \{1, \ldots, n\} \;\;. \end{aligned}$$

Remark 1. *The order of port assignments is irrelevant because all sets of variables are disjoint and a port can only be either an input port or output port, cf. Def. 1 and Def. 3, and thus the assignments share no variables. This also entails that the merged pre, π^{in} and π^{out} are well-defined since V_i^Δ, V_i^{in}, respectively V_i^{out}, are disjoint between components by Def. 2.*

Remark 2. *The user provides component specifications (C_i, I_i^Δ) and a mapping function $\mathcal{X}$, defining which output is connected to which input. The composed system of parallel components can be derived automatically from Def. 6.*

Remark 3. *It follows that the set of variables of the composite component $V(C)$ is the union of all involved components' variable sets $V(C_i)$, i. e., $V(C) = \bigcup_{1 \le i \le n} V(C_i)$. The set of global variables V^{global} contains all global variables in the system (i. e., in all components) and thus, its contents do not change.*

Remark 4. *Since $V^\Delta = \bigcup_{1 \le i \le n} V_i^\Delta$, this definition implies that internally connected Δ-ports $V^{\Delta i}$ of sub-components, as well as the previous values $V^{\Delta +}$ for all open Δ-ports are still stored. As a result, the current and previous values of Δ-ports can still be used internally in the composite, even when the ports are no longer exposed through the external interface of the composed system.*

3.5.2 Example: Parallel Composition

Returning to our running example of Fig. 1, after every component's contract was verified separately, we have to compose the components to form the overall collision avoidance system. The mapping function connects the output ports of the RC and the obstacle with the respective input ports of the robot, cf. (47).

$$\mathcal{X} = \big(\hat{p}_o \mapsto p_o, \hat{d} \mapsto d\big) \tag{47}$$

The component C_{sys} in (48) and interface $\mathrm{I}^{\Delta}_{sys}$ in (49) result from parallel composition of the RC, the robot, and the obstacle, using the introduced mapping function.

$$\mathrm{C}_{sys} = (\underbrace{(\mathit{ctrl}_{rc}; \mathit{ctrl}_r; \mathit{ctrl}_o \cup \mathit{ctrl}_o; \ldots)}_{\mathit{ctrl}_{sys}}, \underbrace{(\mathit{plant}_r, \mathit{plant}_o)}_{\mathit{plant}_{sys}}, \underbrace{\hat{p}_o := p_o; \hat{d} := d}_{\mathit{cp}_{sys}}) \tag{48}$$

$$\mathrm{I}^{\Delta}_{sys} = \big(\underbrace{\{\}}_{V^{in}}, \underbrace{()}_{\pi^{in}}, \underbrace{\{\}}_{V^{out}}, \underbrace{()}_{\pi^{out}}, \underbrace{\{p_o, d, \hat{p}_o, \hat{d}\}}_{V^{\Delta}}, \underbrace{\{p_o^-, d^-, \hat{p}_o^-, \hat{d}^-\}}_{V^-}, \underbrace{\big(p_o \mapsto p_o^-, \ldots\big)}_{\text{pre}}\big) \tag{49}$$

The robot's input ports are connected to the RC's and obstacle's output ports.

3.5.3 Compatibility

During composition, the tests guarding the input ports of an interface are replaced with a deterministic assignment modeling the port connection of the components, which is only safe if the respective output guarantees and input assumptions match. Hence, in addition to contract compliance, users have to show compatibility of components as defined in Def. 7.

Definition 7 (Compatible Composite)**.** *A composite of n components with admissible interfaces $\big((C_1, I_1^{\Delta}) \| \ldots \| (C_n, I_n^{\Delta})\big)_{\mathcal{X}}$ is a* compatible composite *iff* dL *formula*

$$\mathrm{CPO}(I_i^{\Delta}) \overset{def}{\equiv} \big(\mathit{pre}(\mathcal{X}(v)) = \mathit{pre}(v)\big) \rightarrow [v := \mathcal{X}(v)](\pi_j^{\text{out}}(\mathcal{X}(v)) \rightarrow \pi_i^{\text{in}}(v))$$

is valid over (vectorial) equalities and assignments for input ports $v \in \mathcal{I}^{\mathcal{X}} \cap V_i^{\text{in}}$ from I_i^{Δ} connected to $\mathcal{X}(v) \in \mathcal{O}^{\mathcal{X}} \cap V_j^{\text{out}}$ from I_j^{Δ}. We call CPO(I_i^{Δ}) *the* compatibility proof obligation *for the interfaces I_i^{Δ} and say the interfaces I_i^{Δ} are* compatible *(with respect to $\mathcal{X}$) if* CPO(I_i^{Δ}) *is valid for all i.*

Components are compatible if the output properties imply the input properties of connected ports. Compatibility guarantees that handing an output port's value over to the connected input port ensures that the input port's input assumption π^{in} holds, which is no longer checked explicitly by a test, so $\pi_j^{out}(\mathcal{X}(v)) \rightarrow \pi_i^{in}(v)$. To achieve local compatibility checks for pairs of connected ports, instead of global checks over entire component models, Def. 3 restricts input assumptions to only mention variables of the associated ports. Note that even though Def. 3 does not restrict output guarantees, in order to show compatibility each output guarantee should also only mention variables of the associated ports.

3.5.4 Example: Compatibility

In our example, we have to ensure compatibility (cf. Def. 7) of the components with respect to $\mathcal{X}$. Since we have two connected ports, we discharge two compatibility proof obligations, one for each port, cf. (50)-(51).

$$\mathit{CPO}(\mathrm{I}_{rc}^{\Delta}) \equiv \left(d^- = \hat{d}^-\right) \rightarrow \left([\hat{d} := d]\left(\left|d - d^-\right| \leq D \rightarrow \left|\hat{d} - \hat{d}^-\right| \leq D\right)\right) \tag{50}$$

$$\mathit{CPO}(\mathrm{I}_{o}^{\Delta}) \equiv (p_o = \hat{p}_o) \rightarrow \Big([\hat{p}_o := p_o] \left(\left|p_o - p_o^-\right| \leq S \cdot \left(t - t^-\right) \rightarrow \left|\hat{p}_o - \hat{p}_o^-\right| \leq S \cdot \left(t - t^-\right)\right)\Big) \tag{51}$$

Formulas (50)-(51) can be proved automatically using KeYmaera X, so the three interfaces I_{rc}^{Δ}, I_{o}^{Δ} and I_{r}^{Δ} are compatible.

3.6 Transferring Local Component Safety to System Safety

From contract compliance and compatibility proofs, Theorem 1 below transfers the safety properties in component contracts to safety of the composed system. As a result, for showing safety of the monolithic system, we no longer need a (probably huge) monolithic proof. The proof of Theorem 1 can be found in Appendix A.

Theorem 1 (Composition Retains Contracts)**.** *Let C_1 and C_2 be components with admissible interfaces I_1^{Δ} and I_2^{Δ} that are* delay contract compliant *(cf. Def. 5) and* compatible *with respect to $\mathcal{X}$ (cf. Def. 7). Initially, assume $\phi^p \overset{def}{\equiv} \bigwedge_{v \in \mathcal{I}^{\mathcal{X}}} \mathcal{X}(v) = v$ to bootstrap connected ports. Then, if the side condition* (52) *holds (φ_i is the loop invariant used to prove the component's contract)*

$$\models \varphi_i \rightarrow [\Delta_i][\mathrm{ctrl}_i][t^- := t][(t' = 1, \mathrm{plant}_i)]\Pi_i^{\mathrm{out}} \tag{52}$$

for all components C_i, the parallel composition $(C, I^{\Delta}) = \left((C_1, I_1^{\Delta}) \| (C_2, I_2^{\Delta})\right)_{\mathcal{X}}$ then satisfies the contract (53) *with* in, cp, ctrl, *and* plant *according to Def. 6:*

$$\models \left(t = t^- \wedge \phi_1 \wedge \phi_2 \wedge \phi^p\right) \rightarrow [(\Delta; \mathrm{ctrl}; t^- := t; (t' = 1, \mathrm{plant}); \mathrm{in}; \mathrm{cp})^*] \left(\psi_1^{\mathrm{safe}} \wedge \Pi_1^{\mathrm{out}} \wedge \psi_2^{\mathrm{safe}} \wedge \Pi_2^{\mathrm{out}}\right) \quad . \tag{53}$$

The composite contract's precondition ϕ^p ensures that the values of connected ports are consistent initially. Side condition (52) shows that a component already produces the correct output from just its *ctrl* and *plant*; preparing the port inputs for the next loop iteration does not change the current output.

Remark 5. *The side condition* (52) *is trivially true for components without output ports, since $\Pi_i^{\mathrm{out}} \equiv \mathit{true}$. This side condition can—often automatically—be verified using KeYmaera X. For atomic components without input ports, the proof of* (52) *automatically follows from the contract proof, since* in; cp *is empty.*

Remark 6. *Because of the precondition ϕ^p and because* cp *is executed after every execution of the main loop (cf. Def. 5), we know that the values of connected input and output ports coincide in the safety property, as one would expect. Thus, for instance, if the local safety property of a single component mentions an input port (e. g., $\psi_1^{\text{safe}} \equiv |p_r - \hat{p}_o| > 0$), we can replace the input port with the original value as provided by the output port for the composite safety property (e. g., $\psi^{\text{safe}} \equiv |p_r - \hat{p}_o| > 0 \equiv |p_r - p_o| > 0$).*

Remark 7. *Theorem 1 easily extends to n components (cf. proof sketch in [17]) and also holds for change contracts. A change port cannot be attached to a delay port and vice versa.*

3.6.1 Example: Transferring Component Safety to System Safety

Example 3.5.4 proved that the remote control RC, the robot, and the obstacle component are compatible. The remaining proof below discharges the three side conditions from Theorem 1—one for each component.

$$SC(\mathrm{C}_{rc}, \mathrm{I}_{rc}^{\Delta}) \equiv \varphi_{rc} \to [\Delta_{rc}][\mathit{ctrl}_{rc}][t^- := t][\{t' = 1, \mathit{plant}_{rc}\}] \left|d - d^-\right| \leq D \tag{54}$$

$$SC(\mathrm{C}_{o}, \mathrm{I}_{o}^{\Delta}) \equiv \varphi_{o} \to [\Delta_{o}][\mathit{ctrl}_{o}][t^- := t][\{t' = 1, \mathit{plant}_{o}\}] \left|p_o - p_o^-\right| \leq S \cdot \left(t - t^-\right) \tag{55}$$

$$SC(\mathrm{C}_{r}, \mathrm{I}_{r}^{\Delta}) \equiv \varphi_{r} \to [\Delta_{r}][\mathit{ctrl}_{r}][t^- := t][\{t' = 1, \mathit{plant}_{r}\}]\mathit{true} \tag{56}$$

The side condition for the robot is trivially true, since the robot component has no output ports and thus we have to verify that true holds, cf. (56). For the other side conditions we need the respective invariants used to verify each component's contract.

$$\varphi_{rc} \equiv D \geq 0 \land \left|d - d^-\right| \leq D \tag{57}$$

$$\varphi_{o} \equiv S \geq 0 \land 0 \leq s_o \leq S \land \left|p_o - p_o^-\right| \leq S \cdot \left(t - t^-\right) \tag{58}$$

The invariant of the RC component preserves the bound for the global variable D and ensures that the target speed is always in bounds, cf. (57). The invariant of the obstacle component also preserves the bound for the respective global variable S, restricts the obstacle speed to valid values and ensures that the obstacle position stays within bounds, cf. (58). Using these invariants, (54)-(55) can be verified automatically using KeYmaera X. Note, that in this case the side conditions also follow from Remark 5, since the RC component and the obstacle are atomic component without input ports.

Finally, we have verified all component contracts as well as the compatibility proof obligations and the side conditions. Theorem 1 then guarantees system safety $\psi_{sys}^{safe} \equiv s_r > 0 \to p_o \neq p_r$ from the robot's safety property $\psi_r^{safe} \equiv s_r > 0 \to \hat{p}_o \neq p_r$ together with the port connection $\hat{p}_o = p_o$ having the connected ports coincide (cf. Remark 6).

3.6.2 Automation

We implemented the proof steps of Theorem 1 as a KeYmaera X tactic, which automatically reduces a system safety proof to separate proofs about components[5]. This gave us the best of the two

worlds: the flexibility of reasoning with components that our Theorem 1 provides, together with the soundness guarantees we inherit from KeYmaera X, which derives proofs by uniform substitution from axioms [24]. This is to be contrasted with the significant soundness-critical changes we would have to do if we were to add Theorem 1 as a built-in rule into the KeYmaera X prover core. Uniform substitution guarantees, e.g., that the subtle conditions on how and where input and output variables can be read or written in components are checked correctly.

So far the theorem has only be implemented for the composition of two components. Thus, in order to apply the tactic to our running example, we merged the RC component and the obstacle component into a single component, which is possible since the RC and the obstacle are utterly independent.

4 Case Studies

To evaluate our approach[2], we use the running example of a remote-controlled robot (RC robot) and revisit prior case studies on the European Train Control System (i. e., ETCS) [25], two-component robot collision avoidance (i. e., Robix) [13], and adaptive cruise control (i. e., LLC) [10]. 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 multi-port change contract relates distances m, m^- and demanded speeds d, d^- in input assumptions/output guarantees of the form $d \geq 0 \land (d^-)^2 - d^2 \leq 2b(m - m^-) \land \mathit{state} = \mathit{drive}$, thus avoiding physically impossible maneuvers.

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 delay 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 multi-port delay contract with properties of the form $2 \cdot (x_l - x_l^-) \geq v_l + v_l^- \cdot t \land 0 \leq v_l \land -B \cdot t \leq v_l - v_l^- \leq A \cdot t$ tying together speed change and position progress.

Table 1 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., multiport, delay 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 [14] as a back-end decision procedure to discharge arithmetic. 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, condition (52) 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, the results indicate that our approach verification leads to performance improvements

[2]Implementation and full models available online at
`http://www.cs.cmu.edu/~smitsch/resource/fase17`

Table 1: Experimental results for case studies

	Contract				Automation				Duration $[s]$				
	Multi	Change	Delay	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 [25]	✓	✓			✓	✓	✓	✓	127	608	179	**873**	**15306**
Robix [13]			✓	✓	(31)	✓	✓	(96)	469	117	132	**718**	**902**
LLC [10]	✓		✓		✓	(50)	✓	(131)	135	351	267	**753**	**568**

and smaller user-provided proof scripts.

5 Related Work

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

Hybrid Automata and Assume-Guarantee Reasoning. Hybrid automata [1] can be composed in parallel. However, the associated verification procedure (i. e., verify that a formula holds throughout all runs of the automaton) is not compositional, but requires verification of the exponential product automaton [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 [11] 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. Hybrid (I/O) automata are mainly verified using reachability analysis. Therefore, techniques to prevent state-space explosion are needed, like assume-guarantee reasoning (AGR, e. g., [3, 6, 9]), which was developed to decompose a verification task into subtasks. In [6], timed transition systems are used to approximate a component's behavior by discretization. These abstractions are then used in place of the more complicated automata to verify refinement properties. The implementation of their approach is limited to linear hybrid automata. In analogy, we discretize plants to delay 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 [9], 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 delay contracts still preserve knowledge about continuous behavior. The AGR approach of [3] 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 [16].

In [5], 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 [5] 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 [5] or even linear hybrid automata [2], use global contracts [3], focus on sequential composition [5] or rely on reachability analysis, over-approximation and model checking [3, 6, 9]. We, in contrast, focus on *theorem proving* in $\mathsf{d}\mathcal{L}$, using change and delay contracts and handle *non-linear dynamics* and parallel composition. Most crucially, we focus on transfer of safety properties from components to composites, while related approaches are focused on property transfer between different levels of abstraction [3, 6, 9].

Hybrid process algebras are compositional modeling formalisms for the description of behavior and interaction of processes, based on algebraic equations. Examples are Hybrid χ [26], HyPA [18] or the Φ-Calculus [27]. 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 [29], HyPA using HyTech [12], Φ-Calculus using SPHIN [28]), 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 [20], if they all have the same 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 [10]. We focus on different components. Similarly, the approach in [15] presents a component-based approach limited to traffic flow and global contracts.

Our approach extends [16], 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 [15]. However, for systems such as the remote-controlled robot obstacle avoidance from our running example (cf. Section 3.2), 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). Contracts with change and delay allow more liberal component interaction.

6 Conclusion and Future Work

Component-based modeling and verification for hybrid systems splits monolithic system verification into proofs about components with local responsibilities. It reduces verification effort compared to proving monolithic models, while change and delay contracts preserve crucial properties about component behavior to allow liberal component interaction.

Change contracts relate a port's previous value to its current value (i. e., the change since the last port transmission), while delay contracts additionally relate to the delay between measurements. Properties of components, described by component contracts and verified using KeYmaera X, transfer to composed systems of multiple compatible components without re-verification 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. We implemented our approach as a KeYmaera X tactic, which automatically verifies composite systems based on components with verified contracts without increasing the trusted prover core.

For future work, we plan to (i) introduce further composition operations (e. g., error-prone transmission), and (ii) provide support for system decomposition by discovery of output properties (i. e., find abstraction for port behavior).

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. LNCS, vol. 736, pp. 209–229. 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] Benvenuti, L., Bresolin, D., Collins, P., Ferrari, A., Geretti, L., Villa, T.: Assume–guarantee verification of nonlinear hybrid systems withAriadne. International Journal of Robust and Nonlinear Control 24(4), 699–724 (2014)

[4] Bohrer, B., Rahli, V., Vukotic, I., Völp, M., Platzer, A.: Formally verified differential dynamic logic. In: Bertot, Y., Vafeiadis, V. (eds.) Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs. pp. 208–221. ACM (2017)

[5] 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, LNCS, vol. 6200, pp. 96–143. Springer (2010)

[6] Frehse, G., Zhi Han, Krogh, B.: Assume-guarantee reasoning for hybrid I/O-automata by overapproximation of continuous interaction. In: 43rd IEEE Conference on Decision and Control, CDC. vol. 1, pp. 479–484 (2004)

[7] Fulton, N., Mitsch, S., Quesel, J.D., Völp, M., Platzer, A.: KeYmaera X: An axiomatic tactical theorem prover for hybrid systems. In: Felty, A.P., Middeldorp, A. (eds.) 25th International Conference on Automated Deduction, Proceedings. LNCS, vol. 9195, pp. 527–538. Springer (2015)

[8] 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)

[9] 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. LNCS, vol. 2034, pp. 275–290. Springer (2001)

[10] Loos, S.M., Platzer, A., Nistor, L.: Adaptive cruise control: Hybrid, distributed, and now formally verified. In: Butler, M.J., Schulte, W. (eds.) 17th International Symposium on Formal Methods. LNCS, vol. 6664, pp. 42–56. Springer (2011)

[11] Lynch, N.A., Segala, R., Vaandrager, F.W.: Hybrid I/O automata. Inf. Comput. 185(1), 105–157 (2003)

[12] Man, K.L., Reniers, M.A., Cuijpers, P.J.L.: Case studies in the hybrid process algebra Hypa. International Journal of Software Engineering and Knowledge Engineering 15(2), 299–306 (2005)

[13] Mitsch, S., Ghorbal, K., Platzer, A.: On provably safe obstacle avoidance for autonomous robotic ground vehicles. In: Newman, P., Fox, D., Hsu, D. (eds.) Robotics: Science and Systems IX (2013)

[14] 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. LNCS, vol. 4963, pp. 337–340. Springer (2008)

[15] 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)

[16] Müller, A., Mitsch, S., Retschitzegger, W., Schwinger, W., Platzer, A.: A component-based approach to hybrid systems safety verification. In: Ábrahám, E., Huisman, M. (eds.) Integrated Formal Methods - 12th International Conference, Proceedings. LNCS, vol. 9681, pp. 441–456. Springer (2016)

[17] Müller, A., Mitsch, S., Retschitzegger, W., Schwinger, W., Platzer, A.: Change and delay contracts for hybrid system component verification. Tech. Rep. CMU-CS-17-100, Carnegie Mellon (2017)

[18] Pieter J. L. Cuijpers, Reniers, M.A.: Hybrid process algebra. J. Log. Algebr. Program. 62(2), 191–245 (2005)

[19] Platzer, A.: Differential-algebraic dynamic logic for differential-algebraic programs. J. Log. Comput. 20(1), 309–352 (2010)

[20] Platzer, A.: Quantified differential dynamic logic for distributed hybrid systems. In: Dawar, A., Veith, H. (eds.) Computer Science Logic 24th International Workshop, 19th Annual Conference of the EACSL Proceedings. LNCS, vol. 6247, pp. 469–483. Springer (2010)

[21] Platzer, A.: The complete proof theory of hybrid systems. In: Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science. pp. 541–550. IEEE Computer Society (2012)

[22] Platzer, A.: Logics of dynamical systems science. In: Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science. pp. 13–24. IEEE Computer Society (2012)

[23] Platzer, A.: The structure of differential invariants and differential cut elimination. Logical Methods in Computer Science 8(4), 1–38 (2012)

[24] Platzer, A.: A complete uniform substitution calculus for differential dynamic logic. J. Autom. Reas. pp. 1–47 (2016)

[25] Platzer, A., Quesel, J.D.: European train control system: A case study in formal verification. In: Breitman, K.K., Cavalcanti, A. (eds.) Formal Methods and Software Engineering. LNCS, vol. 5885, pp. 246–265. Springer (2009)

[26] 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: 1st International Workshop. LNCS, vol. 2791, pp. 151–165. Springer (2003)

[27] 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. LNCS, vol. 2623, pp. 435–449. Springer (2003)

[28] 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)

[29] 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.) 7th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing. pp. 122–128. IEEE Computer Society (2013)

$$([;]) \quad [\alpha][\beta]\phi \leftrightarrow [\alpha;\beta]\phi \qquad (\rightarrow \text{r}) \quad \frac{\Gamma, \phi \vdash \psi, \Delta}{\Gamma \vdash \phi \rightarrow \psi, \Delta}$$

$$([\cup]) \quad [\alpha]\phi \wedge [\beta]\phi \leftrightarrow [\alpha \cup \beta]\phi \qquad (\rightarrow \text{l}) \quad \frac{\Gamma \vdash \phi, \Delta \quad \Gamma, \psi \vdash \Delta}{\Gamma, \phi \rightarrow \psi \vdash \Delta}$$

$$([:=]) \quad [x := e]\phi(x) \leftrightarrow \phi(e)$$

$$([?]) \quad H \rightarrow \psi \leftrightarrow [?H]\psi \qquad (\wedge \text{r}) \quad \frac{\Gamma \vdash \phi, \Delta \quad \Gamma \vdash \psi, \Delta}{\Gamma \vdash \phi \wedge \psi, \Delta}$$

$$(\text{Wr}) \quad \frac{\Gamma \vdash \Delta}{\Gamma \vdash \phi, \Delta} \qquad (\wedge \text{l}) \quad \frac{\Gamma, \phi, \psi \vdash \Delta}{\Gamma, \phi \wedge \psi \vdash \Delta}$$

$$(\text{Wl}) \quad \frac{\Gamma \vdash \Delta}{\Gamma, \phi \vdash \Delta} \qquad (\forall \text{l}) \quad \frac{\Gamma, \phi(X) \vdash \Delta}{\Gamma, \forall x\, \phi(x) \vdash \Delta}$$

$$(cut) \quad \frac{\Gamma \vdash \phi, \Delta \;\; \Gamma, \phi \vdash \Delta}{\Gamma \vdash \Delta} \qquad (\forall \text{r}) \quad \frac{\Gamma \vdash \phi(s(X_1, .., X_n)), \Delta}{\Gamma \vdash \forall x\, \phi(x), \Delta}\,{}^{1}$$

$$([:*]) \quad \frac{\forall X\, [x := X]\phi}{[x := *]\phi} \qquad ([]gen) \quad \frac{\Gamma \vdash [\alpha]\phi, \Delta \quad \phi \vdash \psi}{\Gamma \vdash [\alpha]\psi, \Delta}$$

$$([]\ \text{M}) \quad \frac{\phi \vdash \psi}{[\alpha]\phi \vdash [\alpha]\psi} \qquad (ind) \quad \frac{\Gamma \vdash \phi, \Delta \quad \phi \vdash [\alpha]\phi \quad \phi \vdash \psi}{\Gamma \vdash [\alpha^*]\psi, \Delta}$$

$$(\text{CER}) \quad \frac{\Gamma \vdash C\left(Q\right), \Delta \quad P \leftrightarrow Q}{\Gamma \vdash C\left(P\right), \Delta} \qquad (\text{CE}) \quad \frac{p\left(\bar{x}\right) \leftrightarrow q\left(\bar{x}\right)}{C\left(p\left(\bar{x}\right)\right) \leftrightarrow C\left(q\left(\bar{x}\right)\right)}$$

$$(\text{CEL}) \quad \frac{\Gamma, C\left(Q\right) \vdash \Delta \quad P \leftrightarrow Q}{\Gamma, C\left(P\right) \vdash \Delta}$$

[1] s is a new (Skolem) function symbol and $X_1, .., X_n$ are all free logical variables of $\forall x\, \phi(x)$.

Figure 3: Proof Rules, see [24]

A Proof of Theorem 1

This section presents the proof of Theorem 1. Throughout this section, we use the proof rules and axioms listed in Fig. 3, for more details see [24]. The axioms for the transformation of hybrid programs can be applied using the CER and CEL proof rules, which allow the application of logical equivalences in any context. The usage of CER and CEL will not be explicitly mentioned throughout the paper.

A.1 Proof Sketch

The proof for Theorem 1 follows the proof sketch of [16]. The main idea is to match the behavior and properties of the composite with the behavior of its components, so that component proofs fill in most proof obligations. We

1. split the proof along component contracts (prove that the composite preserves the component

contracts)

2. reorder port assignments to match the order in the respective component (Lemma 4)
3. drop port assignments and control statements, which are irrelevant for the current contract (Lemma 1)
4. re-introduce (idle) tests for deterministic port assignments (Lemma 5, Lemma 6 and Corollary 1)
5. replace deterministic with non-deterministic assignments to resemble port behavior of unconnected components (Lemma 3)
6. drop plant behavior that is irrelevant for the current contract (Lemma 2)

These steps are implemented as a KeYmaera X tactic, which can be used to check Theorem 1 for components with verified contracts.

A.2 Lemmas and Implementation

We first repeat and adapt the lemmas [16, Lemma 1–Lemma 6] here for easy reference, before we proceed with the proof of Theorem 1. Additionally, we provide insights on how these lemmas were implemented in KeYmaera X.

Lemma 1 (Drop Control)**.** *Let A be an arbitrary dL formula and α, β be hybrid programs, with $FV(A) \cap BV(\beta) = \emptyset$ and $FV(\alpha) \cap BV(\beta) = \emptyset$. Then*

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

are valid.

Proof of Lemma 1, cf. [16, Lemma 1]. Since the lemma bases on assumptions about the intersection of free and bound variables of program constants α and β, it is not expressible in KeYmaera X yet. However, when implemented as a tactic that operates on concrete programs α and β, their free and bound variables can be computed (e. g., $\alpha \equiv x := y$ has bound variable x and free variable y) and the assumptions checked; uniform substitution in the KeYmaera X kernel will fail the tactic if it operates on programs that violate the assumptions.

Lemma 2 (Drop Plant)**.** *Let $\theta_1(x_1)$ and $\theta_2(x_2)$ be terms possibly containing x_1 (respectively x_2), where x_1 and x_2 are vectors with $x_1 \neq x_2$. Let t be a variable with $t \notin x_1$ and $t \notin x_2$. Let $A(x_1)$ be an arbitrary dL formula over x_1 and $H_1(x_1), H_2(x_2)$ be predicates over x_1 (respectively x_2). Then*

$$[(t' = 1, x_1' = \theta_1(x_1) \& H_1(x_1))]A(x_1) \to$$
$$[(t' = 1, x_1' = \theta_1(x_1), x_2' = \theta_2(x_2) \& H_1(x_1) \land H_2(x_2))]A(x_1)$$

is valid.

Proof of Lemma 2, cf. [16, Lemma 2]. Similarly to Lemma 1, we implement the lemma as a tactic that operates on concrete programs and relies on uniform substitution for soundness. That way, differential equations $x' = \theta$ can be dropped one by one from the ODE system, instead of introducing vectorial $\vec{x}' = \vec{\theta}$ into the prover kernel.

Lemma 3 (Overapproximate Assignment)**.** *Let $A(x)$ be an arbitrary dL formula, θ be a term, and x be a variable. Then*

$$[x := *]A(x) \rightarrow [x := \theta]A(x)$$

is valid.

Proof of Lemma 3, cf. [16, Lemma 3]. We proved this lemma as a derived axiom and it can thus be used as a fact in subsequent KeYmaera X proofs. The proof in KeYmaera X first performs the deterministic and the non-deterministic assignment and then instantiates the resulting all-quantifier with θ.

Lemma 4 (Reorder Programs)**.** *Let x, y, a, b be variables, A, F, G be arbitrary dL formulas and $B(a)$ be a dL formula allowed to mention a bound variable a free in B. Then*

$$[x := a; y := b]A \leftrightarrow [y := b; x := a]A \tag{59}$$
$$[x := *; y := *]A \leftrightarrow [y := *; x := *]A \tag{60}$$
$$[x := *; y := b]A \leftrightarrow [y := b; x := *]A \tag{61}$$
$$[x := *; ?B(a)]A \leftrightarrow [?B(a); x := *]A \tag{62}$$
$$[x := a; ?B(a)]A \leftrightarrow [?B(a); x := a]A \tag{63}$$
$$[?F; ?G]A \leftrightarrow [?G; ?F]A \tag{64}$$

are valid.

Lemma 4. The proofs follow from the definition of (non-deterministic) assignments and the axiom for tests. We start with proofing (59)–(61). We only show one direction of the equivalence–the other one follows accordingly.

$$\dfrac{\dfrac{*}{A(a,b) \vdash A(a,b)}}{\dfrac{[x := a; y := b]A(x,y) \vdash [y := b; x := a]A(x,y)}{\vdash (59)}\ {}^{[;],[:=],[:=]}_{\rightarrow \mathrm{r}}}$$

$$\dfrac{\dfrac{*}{[x := *]A(x,b) \vdash [x := *]A(x,b)}}{\dfrac{[x := *; y := b]A(x,y) \vdash [y := b; x := *]A(x,y)}{\vdash (60)}\ {}^{[;],[:=]}_{\rightarrow \mathrm{r}}}$$

$$\dfrac{\dfrac{*}{\forall x\,.\,\forall y\,.\,A(x,y) \vdash \forall x\,.\,\forall y\,.\,A(x,y)}}{\dfrac{[x := *; y := *]A(x,y) \vdash [y := *; x := *]A(x,y)}{\vdash (61)}\ {}^{[;],[:*]}_{\rightarrow \mathrm{r}}}$$

Next, we show formulas (62)–(63). Again we just show one direction of the equivalence–the second direction follows accordingly.

$$\dfrac{\dfrac{\dfrac{*}{[?B(a)]A(a,y) \vdash [?B(a)]A(a,y)}}{[x := a; ?B(a)]A(x,y) \vdash [?B(a); x := a]A(x,y)}\,{}^{[:=],[;]}}{\vdash (62)}\,{}^{\to \mathrm{r}}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{*}{B(a) \vdash A(x_1,y), B(a)} \quad \dfrac{*}{A(x_1,y), B(a) \vdash A(x_1,y)}}{([?B(a)]A(x_1,y)), B(a) \vdash A(x_1,y)}\,{}^{[?],\to \mathrm{l}}}{\forall x\ .\ ([?B(a)]A(x,y)), B(a) \vdash A(x_1,y)}\,{}^{\forall \mathrm{l}}}{\forall x\ .\ ([?B(a)]A(x,y)), B(a) \vdash \forall x\ .\ A(x,y)}\,{}^{\forall \mathrm{r}}}{\forall x\ .\ ([?B(a)]A(x,y)) \vdash [?B(a)]\,(\forall x\ .\ A(x,y))}\,{}^{[?],\to \mathrm{r}}}{[x := *; ?B(a)]A(x,y) \vdash [?B(a); x := *]A(x,y)}\,{}^{[;],[:*]}}{\vdash (63)}\,{}^{\to \mathrm{r}}$$

Finally, we show (64). First, we apply the tests and then use the equivalence $\big(a \to (b \to c)\big) \leftrightarrow \big(b \to (a \to c)\big)$ (i. e., step *impl*).

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{*}{F \to (G \to A) \vdash F \to (G \to A)}}{F \to (G \to A) \vdash G \to (F \to A)}\,{}^{\mathrm{impl}}}{F \to ([?G]A) \vdash [?G]\,(F \to A)}\,{}^{[?]}}{[?F; ?G]A \vdash [?G; ?F]A}\,{}^{[?],[;]}}{\vdash (64)}\,{}^{\to \mathrm{r}}$$

□

Lemma 5 (Introduce Test)**.** *Let A be an arbitrary dℒ formula, α be a hybrid program and F be a formula. Then*

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

Proof of Lemma 5, cf. [16, Lemma 5]. We proved this lemma as a derived axiom and it can thus be used as a fact in subsequent KeYmaera X proofs.

Lemma 6 (Weaken Test)**.** *Let A be an arbitrary dℒ formula and F and G be formulas. Then*

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

Proof of Lemma 6, cf. [16, Lemma 6].

Corollary 1 (Weaken Test – Context)**.** *Let A, F and G be arbitrary dL formulas and let α be an arbitrary program. Then*

$$\big(([\alpha][?G]A) \land ([\alpha]\,(F \to G))\big) \to [\alpha][?F]A$$

Corollary 1. After removing α we can again use Lemma 6.

$$\dfrac{\dfrac{\dfrac{\dfrac{*}{([?G]A) \land (F \to G) \vdash [?F]A}\text{L. 6}}{\big([\alpha]\,([?G]A \land (F \to G))\big) \vdash [\alpha][?F]A}\text{[] M}}{\big(([\alpha][?G]A) \land ([\alpha]\,(F \to G))\big) \vdash [\alpha][?F]A}\text{[] split inv}}{\vdash \big(([\alpha][?G]A) \land ([\alpha]\,(F \to G))\big) \to [\alpha][?F]A}\to\text{r}$$

□

Both, Lemma 6 and Corollary 1, could be verified as derived axioms in KeYmaera X. Since the formulas F, G and A, and the program constant α are not restricted to reasoning about any number of specific variables, Lemma 6 can be verified by applying the implications and tests, which leads to the same formulas in antecedent and succedent. For Corollary 1, we used box monotonicity to remove the context (i. e., program α), which then allows application of Lemma 6.

Note, that in the proof of Theorem 1 we will use these lemmas in the context of other logical and modal formulas. In the corresponding proof steps, we implicitly assume that the lemma consequence is cut into the context, and then the cut is shown using the appropriate choice from axioms K, G, and CE [24] to unpeel the context and use the lemma top level.

A.3 Proof for Two Components

From these lemmas, next we prove that the composition of two safe components with interfaces result in a safe composed system, given proven contract compliance and compatibility of the components.

Theorem 1 (Composition Retains Contracts)**.** *Let C_1 and C_2 be components with admissible interfaces I_1^Δ and I_2^Δ that are* delay contract compliant *(cf. Def. 5) and* compatible *with respect to $\mathcal{X}$ (cf. Def. 7). Initially, assume $\phi^p \overset{def}{\equiv} \bigwedge_{v\in\mathcal{I}^\mathcal{X}} \mathcal{X}(v) = v$ to bootstrap connected ports. Then, if the side condition* (52) *holds (φ_i is the loop invariant used to prove the component's contract)*

$$\models \varphi_i \to [\Delta_i][\text{ctrl}_i][t^- := t][(t' = 1, \text{plant}_i)]\Pi_i^{\text{out}} \tag{52}$$

for all components C_i, the parallel composition $(C, I^\Delta) = \big((C_1, I_1^\Delta)\|(C_2, I_2^\Delta)\big)_\mathcal{X}$ then satisfies the contract (53) *with* in, cp, ctrl, *and* plant *according to Def. 6:*

$$\begin{aligned}\models \big(t = t^- \land \phi_1 \land \phi_2 \land \phi^p\big) \to [(\Delta; \text{ctrl}; t^- := t; (t' = 1, \text{plant});\\ \text{in}; \text{cp})^*]\,\big(\psi_1^{\text{safe}} \land \Pi_1^{\text{out}} \land \psi_2^{\text{safe}} \land \Pi_2^{\text{out}}\big)\quad .\end{aligned} \tag{53}$$

Theorem 1. For space reasons let

$$
\begin{aligned}
(C_3, I_3^\delta) &\overset{\text{def}}{\equiv} \left((C_1, I_1^\delta) \| (C_2, I_2^\delta)\right)_{\mathcal{X}} \\
\mathit{ctrl}_3 &\overset{\text{def}}{\equiv} (\mathit{ctrl}_1; \mathit{ctrl}_2) \cup (\mathit{ctrl}_2; \mathit{ctrl}_1) \\
\mathit{plant}_3 &\overset{\text{def}}{\equiv} \mathit{plant}_1, \mathit{plant}_2 \\
\phi_3 &\overset{\text{def}}{\equiv} \phi_1 \wedge \phi_2 \wedge \phi^p \\
\psi_3^{\mathit{safe}} &\overset{\text{def}}{\equiv} \psi_1^{\mathit{safe}} \wedge \psi_2^{\mathit{safe}} \\
\Pi_3^{\mathit{out}} &\overset{\text{def}}{\equiv} \left(\bigwedge_{v \in V^{\mathit{out}}} \pi_1^{\mathit{out}}(v) \right) \wedge \left(\bigwedge_{v \in V^{\mathit{out}}} \pi_2^{\mathit{out}}(v) \right)
\end{aligned}
$$

We know that

$$
\mathit{DC}(C_1^\delta, I_1^\delta) \overset{\text{Def. 5}}{\equiv} t = t^- \wedge \phi_1 \rightarrow
$$

$$
\left[\left(\Delta_1; \mathit{ctrl}_1; t^- := t; (t' = 1, \mathit{plant}_1)\,; \mathit{in}_1; \mathit{cp}_1\right)^*\right] \left(\psi_1^{\mathit{safe}} \wedge \bigwedge_{v \in V_1^{\mathit{out}}} \pi_1^{\mathit{out}}(v) \right) \tag{65}
$$

$$
\mathit{DC}(C_2^\delta, I_2^\delta) \overset{\text{Def. 5}}{\equiv} t = t^- \wedge \phi_2 \rightarrow
$$

$$
\left[\left(\Delta_2; \mathit{ctrl}_2; t^- := t; (t' = 1, \mathit{plant}_2)\,; \mathit{in}_2; \mathit{cp}_2\right)^*\right] \left(\psi_2^{\mathit{safe}} \wedge \bigwedge_{v \in V_2^{\mathit{out}}} \pi_2^{\mathit{out}}(v) \right) \tag{66}
$$

$$
\mathit{DC}(C_3^\delta, I_3^\delta) \overset{\text{Def. 5}}{\equiv} t = t^- \wedge \phi_3 \rightarrow
$$

$$
\left[\left(\Delta_3; \mathit{ctrl}_3; t^- := t; (t' = 1, \mathit{plant}_3)\,; \mathit{in}_3; \mathit{cp}_3\right)^*\right] \left(\psi_3^{\mathit{safe}} \wedge \Pi_3^{\mathit{out}} \right) \tag{67}
$$

$$
\mathit{CPO}(I_1^\delta) \overset{\text{Def. 7}}{\equiv} \text{pre}(\mathcal{X}(v)) = \text{pre}(v) \rightarrow [v := \mathcal{X}(v)] \left(\pi_1^{\mathit{out}}(\mathcal{X}(v)) \rightarrow \pi_2^{\mathit{in}}(v) \right) \text{ and} \tag{68}
$$

$$
\mathit{CPO}(I_2^\delta) \overset{\text{Def. 7}}{\equiv} \text{pre}(\mathcal{X}(v)) = \text{pre}(v) \rightarrow [v := \mathcal{X}(v)] \left(\pi_2^{\mathit{out}}(\mathcal{X}(v)) \rightarrow \pi_1^{\mathit{in}}(v) \right) \text{ for all } v \in \mathcal{I}^{\mathcal{X}} \cap V_{1,2}^{\mathit{in}}\ . \tag{69}
$$

We have to show that the contract (67) of the parallel composition $\mathit{DC}(C_3^\delta, I_3^\delta)$ is valid. We know that formulas (65) and (66) are valid, hence there exist invariants φ_1 and φ_2 such that:

$$
t = t^- \wedge \phi_1 \rightarrow \varphi_1 \tag{70}
$$

$$
\varphi_1 \rightarrow [\Delta_1; \mathit{ctrl}_1; t^- := t; (t' = 1, \mathit{plant}_1)\,; \mathit{in}_1; \mathit{cp}_1] \varphi_1 \tag{71}
$$

$$
\varphi_1 \rightarrow \left(\psi_2^{\mathit{safe}} \wedge \bigwedge_{v \in V^{\mathit{out}}} \pi_1^{\mathit{out}}(v) \right) \tag{72}
$$

If (65) and (66) were verified using loop induction, the invariants are even known. Note, that φ_1 is an inductive loop invariant for the first component, so it can be phrased such that $FV(\varphi_1) \subseteq V(\mathrm{C}_1) \cup V^{\mathit{global}} \cup \{t, t^-\}$, accordingly for φ_2. Formulas (70)–(72) are phrased accordingly for component C_2.

By *imply-right* and *loop induction*, where we choose $\varphi_3 = \varphi_1 \wedge \varphi_2 \wedge \phi^p$ (i. e., the loop invariant for the proof is the conjunction of the two invariants known to exist from the independent proofs, plus the knowledge that the values of connected ports are equal), we get

$$\dfrac{\dfrac{t = t^- \wedge \phi_3 \vdash \varphi_3 \qquad \varphi_3 \vdash [\Delta_3; \mathit{ctrl}_3; t^- := t; (t' = 1, \mathit{plant}_3)\,; \mathit{in}_3; \mathit{cp}_3]\,\varphi_3 \quad \varphi_3 \vdash \left(\psi_3^{\mathit{safe}} \wedge \Pi_3^{\mathit{out}}\right)}{t = t^- \wedge \phi_3 \vdash \left[(\Delta_3; \mathit{ctrl}_3; t^- := t; (t' = 1, \mathit{plant}_3)\,; \mathit{in}_3; \mathit{cp}_3)^*\right] \left(\psi_3^{\mathit{safe}} \wedge \Pi_3^{\mathit{out}}\right)}\,\mathit{ind}}{\vdash t = t^- \wedge \phi_3 \rightarrow \left[(\Delta_3; \mathit{ctrl}_3; t^- := t; (t' = 1, \mathit{plant}_3)\,; \mathit{in}_3; \mathit{cp}_3)^*\right] \left(\psi_3^{\mathit{safe}} \wedge \Pi_3^{\mathit{out}}\right)}\,{\rightarrow}\mathrm{r}$$

We will transform the three resulting branches until we get formulas that correspond to (70), (71) and (72). To prove the induction base case and the use case, we use the loop invariants φ_1 and φ_2, for which (70) and (72) hold.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{*}{t = t^-, \phi_1 \vdash \varphi_1}(70)}{t = t^-, \phi_1, \phi_2, \phi^p \vdash \varphi_1}\mathrm{Wl}}{t = t^- \wedge \phi_1 \wedge \phi_2 \wedge \phi^p \vdash \varphi_1}\wedge\mathrm{l} \qquad \dfrac{\dfrac{\dfrac{*}{t = t^-, \phi_2 \vdash \varphi_2}(70)}{t = t^-, \phi_1, \phi_2, \phi^p \vdash \varphi_2}\mathrm{Wl}}{t = t^- \wedge \phi_1 \wedge \phi_2 \wedge \phi^p \vdash \varphi_2}\wedge\mathrm{l} \qquad \dfrac{\dfrac{*}{\phi^p \vdash \phi^p}}{t = t^-, \phi_1, \phi_2, \phi^p \vdash \phi^p}\mathrm{Wl}}{t = t^- \wedge \phi_1 \wedge \phi_2 \wedge \phi^p \vdash \phi^p}\wedge\mathrm{l}}{t = t^- \wedge \phi_1 \wedge \phi_2 \wedge \phi^p \vdash \varphi_1 \wedge \varphi_2 \wedge \phi^p}\wedge\mathrm{r}}{t = t^- \wedge \phi_3 \vdash \varphi_3}\,\mathrm{def}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{*}{\varphi_1 \vdash \psi_1^{\mathit{safe}} \wedge \bigwedge_{v \in V_1^{\mathit{out}}} \pi^{\mathit{out}}(v)}(72)}{\varphi_1, \varphi_2, \phi^p \vdash \psi_1^{\mathit{safe}} \wedge \bigwedge_{v \in V_1^{\mathit{out}}} \pi^{\mathit{out}}(v)}\mathrm{Wl} \qquad \dfrac{\dfrac{*}{\varphi_2 \vdash \psi_2^{\mathit{safe}} \wedge \bigwedge_{v \in V_2^{\mathit{out}}} \pi^{\mathit{out}}(v)}(72)}{\varphi_1, \varphi_2, \phi^p \vdash \psi_2^{\mathit{safe}} \wedge \bigwedge_{v \in V_2^{\mathit{out}}} \pi^{\mathit{out}}(v)}\mathrm{Wl}}{\varphi_1, \varphi_2, \phi^p \vdash \left(\psi_1^{\mathit{safe}} \wedge \bigwedge_{v \in V_1^{\mathit{out}}} \pi^{\mathit{out}}(v)\right) \wedge \left(\psi_2^{\mathit{safe}} \wedge \bigwedge_{v \in V_2^{\mathit{out}}} \pi^{\mathit{out}}(v)\right)}\wedge\mathrm{r}}{\varphi_1 \wedge \varphi_2 \wedge \phi^p \vdash \left(\psi_1^{\mathit{safe}} \wedge \bigwedge_{v \in V_1^{\mathit{out}}} \pi^{\mathit{out}}(v)\right) \wedge \left(\psi_2^{\mathit{safe}} \wedge \bigwedge_{v \in V_2^{\mathit{out}}} \pi^{\mathit{out}}(v)\right)}\wedge\mathrm{l}}{\varphi_3 \vdash \left(\psi_3^{\mathit{safe}} \wedge \Pi_3^{\mathit{out}}\right)}\,\mathrm{def}$$

Invariance of $\phi^p \equiv \bigwedge_{v \in \mathcal{I}^{\mathcal{X}}} \mathcal{X}(v) = v$ follows immediately, since connected ports are always assigned in cp_i (i. e., $v = \mathcal{X}(v)$).

The implementation of the tactic for the base case and use case is fairly straightforward. First apply the structural proof rules (i. e., $\wedge$ and weakening), then apply the proofs of the single components.

It remains to show the induction step, which we do by proving invariance of φ_1 and φ_2 separately.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dots ① \qquad \dots ②}{\varphi_1, \varphi_2, \phi^p \vdash [\Delta_3][\mathit{ctrl}_3][t^- := t][(t' = 1, \mathit{plant}_3)][\mathit{in}_3; \mathit{cp}_3]\varphi_1}\,[]\mathrm{split},\wedge\mathrm{r}}{\varphi_1, \varphi_2, \phi^p \vdash [\Delta_3][\mathit{ctrl}_3][t^- := t][(t' = 1, \mathit{plant}_3)][\mathit{in}_3; \mathit{cp}_3]\,(\varphi_1 \wedge \varphi_2)}\,[;]}{\varphi_1, \varphi_2, \phi^p \vdash [\Delta_3; \mathit{ctrl}_3; t^- := t; (t' = 1, \mathit{plant}_3)\,; \mathit{in}_3; \mathit{cp}_3]\,(\varphi_1 \wedge \varphi_2)}\wedge\mathrm{l}}{\varphi_1 \wedge \varphi_2 \wedge \phi^p \vdash [\Delta_3; \mathit{ctrl}_3; t^- := t; (t' = 1, \mathit{plant}_3)\,; \mathit{in}_3; \mathit{cp}_3]\,(\varphi_1 \wedge \varphi_2)}\,\mathrm{def}}{\varphi_3 \vdash [\Delta_3; \mathit{ctrl}_3; t^- := t; (t' = 1, \mathit{plant}_3)\,; \mathit{in}_3; \mathit{cp}_3]\varphi_3}$$

We have to prove that both, φ_1 (i. e., ①) and φ_2 (i. e., ②) hold. We illustrate the strategy only for branch ①, because branch ② follows in a similar manner.

We apply the proof rule for non-deterministic choice and get two branches.

$$\dfrac{\dfrac{\dfrac{\dots ③}{\varphi_1, \varphi_2, \phi^p \vdash [\Delta_3][ctrl_1; ctrl_2][t^- := t][(t' = 1, plant_3)][in_3; cp_3]\varphi_1} \dots ④}{\varphi_1, \varphi_2, \phi^p \vdash [\Delta_3][((ctrl_1; ctrl_2) \cup \dots)][t^- := t][(t' = 1, plant_3)][in_3; cp_3]\varphi_1}\,[\cup],[]\text{split},\wedge\text{r}}{\varphi_1, \varphi_2, \phi^p \vdash [\Delta_3][ctrl_3][t^- := t][(t' = 1, plant_3)][in_3; cp_3]\varphi_1}\,\text{def}$$

① continued

We will now transform ③ until we get (71). First, we remove control $ctrl_2$, and reorder in_3 so that we can then remove the assignments in_2. We reintroduce tests and turn the deterministic assignments of connected ports into non-deterministic ones until they behave like non-connected inputs, and finally get (71). The detailed proof steps are explained below and can be cross-referenced using enumeration and the step number in the sequent proof.

For space reasons we define

$$F_2^{out} \overset{\text{def}}{\equiv} \varphi_2 \rightarrow [\Delta_3][ctrl_1; ctrl_2][t^- := t][(t' = 1, plant_1, plant_2)]\Pi_2^{out} \ .$$

	$*$
(71)	$\varphi_1 \vdash [\Delta_1][ctrl_1][t^- := t][(t' = 1, plant_1)][in_1][cp_1]\varphi_1$
Wl	$\varphi_1, \varphi_2 \vdash [\Delta_1][ctrl_1][t^- := t][(t' = 1, plant_1)][in_1][cp_1]\varphi_1$
cf. 13.,L. 1	$\varphi_1, \varphi_2 \vdash [\Delta_1; \Delta_2][ctrl_1][t^- := t][(t' = 1, plant_1)][in_1][cp_1]\varphi_1$
cf. 13.,L. 1	$\varphi_1, \varphi_2 \vdash [\Delta_1; \Delta_2][ctrl_1; ctrl_2][t^- := t][(t' = 1, plant_1)][in_1][cp_1]\varphi_1$
cf. 12.,def,L. 4	$\varphi_1, \varphi_2 \vdash [\Delta_3][ctrl_1; ctrl_2][t^- := t][(t' = 1, plant_1)][in_1][cp_1]\varphi_1$
cf. 11.,L. 2	$\varphi_1, \varphi_2 \vdash [\Delta_3][ctrl_1; ctrl_2][t^- := t][(t' = 1, plant_1, plant_2)][in_1][cp_1]\varphi_1$
cf. 10.,def	$\varphi_1, \varphi_2 \vdash [\Delta_3][ctrl_1; ctrl_2][t^- := t][(t' = 1, plant_3)][in_1^*][v_j := *][?\pi_1^{in}(v_j)][cp_1^*]\varphi_1$
cf. 9.,L. 3	$\varphi_1, \varphi_2 \vdash [\Delta_3][ctrl_1; ctrl_2][t^- := t][(t' = 1, plant_3)][in_1^*][v_j := \mathcal{X}(v_j)][?\pi_1^{in}(v_j)][cp_1^*]\varphi_1$
cf. 8.,C. 1	$\varphi_1, \varphi_2, \phi^p \vdash [\Delta_3][ctrl_1; ctrl_2][t^- := t][(t' = 1, plant_3)][in_1^*][v_j := \mathcal{X}(v_j)][?\pi_2^{out}(\mathcal{X}(v_j))][cp_1^*]\varphi_1$
cf. 7.,L. 4	$\varphi_1, \varphi_2, \phi^p \vdash [\Delta_3][ctrl_1; ctrl_2][t^- := t][(t' = 1, plant_3)][?\pi_2^{out}(\mathcal{X}(v_j))][in_1^*][v_j := \mathcal{X}(v_j)][cp_1^*]\varphi_1$
cf. 6.,def	$\varphi_1, \varphi_2, \phi^p \vdash [\Delta_3][ctrl_1; ctrl_2][t^- := t][(t' = 1, plant_3)][?\pi_2^{out}(\mathcal{X}(v_j))][in_1^*; cp_1]\varphi_1$
cf. 5.,L. 5	$\varphi_1, \varphi_2, \phi^p, F_2^{out} \vdash [\Delta_3][ctrl_1; ctrl_2][t^- := t][(t' = 1, plant_3)][in_1^*; cp_1]\varphi_1$
cf. 4.,L. 1	$\varphi_1, \varphi_2, \phi^p, F_2^{out} \vdash [\Delta_3][ctrl_1; ctrl_2][t^- := t][(t' = 1, plant_3)][in_1^*; in_2^*; cp_1]\varphi_1$
cf. 3.,L. 1	$\varphi_1, \varphi_2, \phi^p, F_2^{out} \vdash [\Delta_3][ctrl_1; ctrl_2][t^- := t][(t' = 1, plant_3)][in_1^*; in_2^*; cp_1; cp_2]\varphi_1$
cf. 2.,def,L. 4	$\varphi_1, \varphi_2, \phi^p, F_2^{out} \vdash [\Delta_3][ctrl_1; ctrl_2][t^- := t][(t' = 1, plant_3)][in_3; cp_3]\varphi_1$ … ⑥
cf. 1.,*cut*	$\varphi_1, \varphi_2, \phi^p \vdash [\Delta_3][ctrl_1; ctrl_2][t^- := t][(t' = 1, plant_3)][in_3; cp_3]\varphi_1$

③ continued

Up to this point the proof of the induction step could almost be implemented one-to-one as a tactic. However, the following proof steps require detailed knowledge of the exact structure of the respective formulas, which complicates automatic verification. The dropping of a specific part requires the dismembering of sequential compositions to separate boxes, followed by a reassembling of the remaining boxes to sequential compositions after the removal. Especially rotation of tests and assignments in steps 7 and 10 require a great number of decompositions and compositions, and multiple applications of the correct part of Lemma 4.

In detail, we applied the following lemmas:

1. First, we cut in F_2^{out}, which we will need later throughout the proof, to verify the side condition of Lemma 5. The side condition of the cut is verified in ⑥.

2. We use Lemma 4 to reorder the assignments in in_3 and cp_2 in a way that the assignments of C_1 precede the ones of C_2. Note, that in_3 contains only the non-connected ports of C_1 and C_2, while the connected ports are still in cp_3. Hence, we use in_1^* and in_2^* to denote that these assignments are not the full in_1 and in_2.

3. We apply Lemma 1 to remove cp_2, with $\alpha \overset{\text{def}}{\equiv} cp_1$, $\beta \overset{\text{def}}{\equiv} cp_2$ and $A \overset{\text{def}}{\equiv} \varphi_1$, so $FV(\alpha) \cap BV(\beta) = \emptyset$ since $FV(cp_1) \subseteq V(\mathrm{C}_1)$ and $BV(cp_2) \subseteq V_2^{in}$ (and thus $BV(cp_2) \subseteq V(\mathrm{C}_2) \setminus \left(V^{global} \cup \{t, t^-\}\right)$) are disjoint. Further $FV(A) \cap BV(\beta) = \emptyset$ since $FV(\varphi_1) \subseteq V(\mathrm{C}_1)$ and $V(\mathrm{C}_1) \cap V(\mathrm{C}_2) \setminus \left(V^{global} \cup \{t, t^-\}\right) = \emptyset$.

4. We apply Lemma 1 to remove in_2^*, with $\alpha \overset{\text{def}}{\equiv} in_1^*$, $\beta \overset{\text{def}}{\equiv} in_2^*$ and $A \overset{\text{def}}{\equiv} [cp_1]\varphi_1$, so $FV(\alpha) \cap BV(\beta) = \emptyset$ since $FV(in_1^*) \subseteq V(\mathrm{C}_1)$ and $BV(cp_2) \subseteq V_2^{in}$ (and thus $BV(cp_2) \subseteq V(\mathrm{C}_2) \setminus \left(V^{global} \cup \{t, t^-\}\right)$) are disjoint. Further, $FV(A) \cap BV(\beta) = \emptyset$ since $FV([cp_1]\varphi_1) \subseteq V(\mathrm{C}_1)$ and $V(\mathrm{C}_1) \cap V(\mathrm{C}_2) \setminus \left(V^{global} \cup \{t, t^-\}\right) = \emptyset$.

5. By application of Lemma 5, we can insert a test for $\pi_2^{out}(\mathcal{X}(v_j))$, for the leftmost deterministic assignment in cp_1 without changing the behavior (side condition follows immediately from F_2^{out}). Additionally, we hide F_2^{out} for space reasons.

6. We then split cp_1 into two parts: the leftmost assignment and cp_1^*, which represents the remaining port assignments.

7. We change the order of programs to move the introduced test for $\pi_2^{out}(\mathcal{X}(v_j))$ after the leftmost deterministic assignment. This is possible because in_1^* and cp_1 bind only input variables from C_1, while $\pi_2^{out}(\mathcal{X}(v_j))$ only reasons about variables from C_2, so $BV(in_1^*; cp_1) \cap FV(\pi_2^{out}(\mathcal{X}(v_j))) = \emptyset$ (cf. Lemma 4).

8. We then relax the test to $\pi_{v_j}^{in}$ using compatibility as follows.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\varphi_1, \varphi_2, \bigwedge_{v \in \mathcal{I}^{\mathcal{X}}} \mathcal{X}(v) = v \vdash [\ldots; \mathrm{pre}(\mathcal{X}(v_j)) := \mathcal{X}(v_j); \ldots; \mathrm{pre}(v_j) := v_j; \ldots]^* (\mathrm{pre}(\mathcal{X}(v_j)) = \mathrm{pre}(v_j))}\ \text{def}}{\varphi_1, \varphi_2, \phi^p \vdash [\Delta_1; \Delta_2] (\mathrm{pre}(\mathcal{X}(v_j)) = \mathrm{pre}(v_j))}\ \text{def}}{\varphi_1, \varphi_2, \phi^p \vdash [\Delta_3] (\mathrm{pre}(\mathcal{X}(v_j)) = \mathrm{pre}(v_j))}\ \text{L. 1}}{\varphi_1, \varphi_2, \phi^p \vdash [\Delta_3][ctrl_1; ctrl_2][t^- := t][(t' = 1, plant_3)][in_1^*] (\mathrm{pre}(\mathcal{X}(v_j)) = \mathrm{pre}(v_j))}\ (68)$$

$$\varphi_1, \varphi_2, \phi^p \vdash [\Delta_3][ctrl_1; ctrl_2][t^- := t][(t' = 1, plant_3)][in_1^*][v_j := \mathcal{X}(v_j)] (\pi_2^{out}(\mathcal{X}(v_j)) \to \pi_1^{in}(v_j))$$

Step L. 1 is justified from $\mathrm{pre}(\mathcal{X}(v_j)) \in V_2^-$ and $\mathrm{pre}(v_j) \in V_1^-$ with Def. 3 specifying that $(BV(ctrl) \cup BV(plant) \cup BV(cp)) \cap V^- = \emptyset$, and all variables except global ones being disjoint across components. We then expand $\Delta_1; \Delta_2$ according to Def. 5, so we get $\mathrm{pre}(\mathcal{X}(v_j)) := \mathcal{X}(v_j)$ and $\mathrm{pre}(v_j) := v_j$, among further assignments to previous values per $\mathcal{X}$. This concludes the step, since ϕ^p holds initially and thus $\mathcal{X}(v_j) = v_j$.

9. By Lemma 3 we can replace the deterministic port assignment with a non-deterministic assignment.

10. After repeating steps 8–9 once for every connected port v_j and after multiple applications of Lemma 4 (to change the order of the assignments) get $\mathit{in}_1; \mathit{cp}_1$.

11. Using Lemma 2, we can remove the plant of component two (i. e., plant_2), because $BV(\mathit{plant}_2) \cap FV([\mathit{in}_1][\mathit{cp}_1]\varphi_1) = \emptyset$.

12. We use Lemma 4 to reorder assignments in Δ_3 in a way that assignments to previous variables from the first component (i. e., Δ_1) precede the ones to previous variables from the second component (i. e., Δ_2).

13. By multiple applications of Lemma 1, we can remove all remaining control parts of component 2 until we get (71) (for the second component).

For the proof to close in the implementation, we have to ensure that the resulting formula has the exact same structure as (71), which again requires a larger number of decomposition and composition steps.

If we use multi-ports, the test for the multi-port has to be moved at the respective position and the assignments to all of the port's variables have to be kept together.

The side condition of the cut from step 1 is verified below.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{*}{\varphi_2 \vdash [\Delta_2][\mathit{ctrl}_2][t^- := t][(t' = 1, \mathit{plant}_2)]\Pi_2^{\mathit{out}}}\,\text{cf. 3.}}{\varphi_2 \vdash [\Delta_1; \Delta_2][\mathit{ctrl}_2][t^- := t][(t' = 1, \mathit{plant}_2)]\Pi_2^{\mathit{out}}}\,\text{cf. 2.,L. 1}}{\varphi_2 \vdash [\Delta_1; \Delta_2][\mathit{ctrl}_1; \mathit{ctrl}_2][t^- := t][(t' = 1, \mathit{plant}_2)]\Pi_2^{\mathit{out}}}\,\text{cf. 2.,L. 1}}{\varphi_2 \vdash [\Delta_3][\mathit{ctrl}_1; \mathit{ctrl}_2][t^- := t][(t' = 1, \mathit{plant}_2)]\Pi_2^{\mathit{out}}}\,\text{def}}{\varphi_2 \vdash [\Delta_3][\mathit{ctrl}_1; \mathit{ctrl}_2][t^- := t][(t' = 1, \mathit{plant}_1, \mathit{plant}_2)]\Pi_2^{\mathit{out}}}\,\text{cf. 1.,L. 2}}{\vdash \varphi_2 \to [\Delta_3][\mathit{ctrl}_1; \mathit{ctrl}_2][t^- := t][(t' = 1, \mathit{plant}_1, \mathit{plant}_2)]\Pi_2^{\mathit{out}}}\,{\to}\text{r}$$

⑥: Side Condition, cut

1. We apply Lemma 2 to get rid of plant_1.

2. By multiple applications of Lemma 1 we can remove Δ_1 and ctrl_1.

3. The proof closes, since we know that the side condition of Theorem 1 holds, cf. (52).

The proof for ④ works almost alike, where the only difference is the order of the control parts. Thus we can apply the same proof steps as above, except step 13, where we have to use the second part of Lemma 1.

The proof for ② follows accordingly, using φ_2 in place of φ_1. Thus, we conclude that $\mathit{DC}(C_3^\delta, I_3^\delta)$ is valid. □

A.4 Proof Sketch for n Components.

So far, we proved Theorem 1 for two components. Next, we sketch how the proof can be extended to n components. In order to generalize the proof to n interfaces, we have to consider n change contracts, one for each component with its interface $(\mathrm{C}_i, \mathrm{I}_i^{\Delta})$ (for $i \in \{0, \ldots, n\}$).

$$\mathit{DC}(C_i^{\delta}, I_i^{\delta}) \equiv t = t^{-} \wedge \phi_i \rightarrow [\big(\Delta_i; \mathit{ctrl}_i; t^{-} := t; (t' = 1, \mathit{plant}_i)\big)^*][\mathit{in}_i; \mathit{cp}_i]\psi_i \tag{73}$$

Again, we assume that formula (73) was proven for all i, hence there exist invariants φ_i (cf. (70)-(72)). We still need to verify $\mathit{DC}(C^{\delta}, I^{\delta})$, except that *plant* now executes all n plants in parallel, *cp* contains all old port assignments of all components and *ctrl* contains all $n!$ permutations of all control parts, i. e.,

$$\begin{aligned} \mathit{ctrl} \equiv\ & (\mathit{ctrl}_1; \mathit{ctrl}_2; ...; \mathit{ctrl}_n) \cup \\ & (\mathit{ctrl}_2; \mathit{ctrl}_1; ...; \mathit{ctrl}_n) \cup \\ & ... \\ & (\mathit{ctrl}_n; ...; \mathit{ctrl}_2, \mathit{ctrl}_1) \end{aligned}$$

and

$$\mathit{cp} \equiv \mathit{cp}_1; \mathit{cp}_2; ...; \mathit{cp}_n$$

The order of port assignments is irrelevant because of Lemma 4, since all port assignments are independent.

We define our invariant φ as the conjunction of all φ_i and ϕ^p. The base case and use case follow immediately from the loop invariants φ_i, analogous two the two-component case. It remains to the induction step, which can be traced back to the steps carried out to prove the two-component version.

We consider one example branch, where we need to show that φ_2 holds after each of its runs:

$$\bigwedge_i \varphi_i, \phi^p \vdash [\Delta_3; (\mathit{ctrl}_1; \mathit{ctrl}_2; ...; \mathit{ctrl}_n)\,; (t' = 1, \mathit{plant}_1)][\mathit{in}_3; \mathit{cp}_3]\varphi_2$$

We ultimately have to reduce the branch to the loop induction step of component C_2 (cf. (71)). Thus, we have to transform the connected ports back to unconnected ones and remove the unnecessary parts. First, we remove all *in* and *cp* parts, which do not belong to C_2. Steps 5 to eleven work alike. Then, the plants can be removed, using Lemma 2. The unnecessary control parts can be removed using Lemma 1. In a similar way, all other branches can be proved.