A Component-based Approach to Hybrid Systems Safety Verification

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

June 2016
CMU-CS-16-100
JKU-CIS-2016-01

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 iFM 2016 [22].
Andreas Müller, Stefan Mitsch, Werner Retschitzegger, Wieland Schwinger and André Platzer. A Component-based Approach to Hybrid Systems Safety Verification. In Erika Abraham, Marieke Huisman, editors, 12th International Conference on integrated Formal Methods, iFM, Reykjavik, Island, Proceedings, LNCS. Springer, 2016.

Work partly funded by BMVIT grant FFG BRIDGE 838526, OeAD Marietta Blau grant ICM-2014-08600, FWF P28187-N31, and ERC PIOF-GA-2012-328378.

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

Abstract

We study a component-based approach to simplify the challenges of verifying large-scale hybrid systems. Component-based modeling can be used to split large models into partial models to reduce modeling complexity. Yet, verification results also need to transfer from components to composites. In this paper, we propose a component-based hybrid system verification approach that combines the advantages of component-based modeling (e. g., reduced model complexity) with the advantages of formal verification (e. g., guaranteed contract compliance). Our strategy is to decompose the system into components, verify their local safety individually and compose them to form an overall system that provably satisfies a global contract, without proving the whole system. We introduce the necessary formalism to define the structure and behavior of components and a technique how to compose components such that safety properties provably emerge from component safety.

1 Introduction

The hybrid dynamics of computation and physics in safety-critical cyber-physical systems (CPS), such as driver assistance systems, self-driving cars, autonomous robots, and airplanes, are almost impossible to get right without proper formal analysis. To enable this analysis, CPS are modeled using so called *hybrid system models*. At larger scales of realistic hybrid system models, formal verification of monolithic models becomes quite challenging. Therefore, component-based modeling approaches split large models into partial models, i. e., co-existing or interacting components (e. g., multiple airplanes in a collision avoidance maneuver). Even though this can lead to component-based models with improved structure and reduced modeling complexity, component verification results do not always transfer to composite systems without appropriate care.

This paper generalizes our previous work [21], which was limited to traffic flow models (i. e., port conditions limited to maximum values, contracts limited to load restrictions, components limited to interfaces and predefined behavior), to a more generic approach to *make hybrid system theorem proving modular on a component level*. The approach exploits component contracts to compose *verified components* and their *safety proofs* to a verified CPS. Differential dynamic logic $\mathsf{d}\mathcal{L}$ [24, 25], the hybrid systems specification and verification logic we are working with, is already compositional for each of its operators and, thus, a helpful basis for our approach. Reasoning in $\mathsf{d}\mathcal{L}$ splits models along the $\mathsf{d}\mathcal{L}$ operators into smaller pieces. In this paper, we build compositionality for a notion of components and interfaces on top of $\mathsf{d}\mathcal{L}$. We focus on modeling a system in terms of components that each capture only a part of the system's behavior (as opposed to monolithic models) and a way to compose components by connecting their interfaces (as opposed to basic program composition with $\mathsf{d}\mathcal{L}$ operators). Component-based hybrid systems verification is challenging because both local component behavior (e. g., decisions and motion of a robot) and inherently global phenomena (e. g., time) co-occur, because components can interact virtually (e. g., robots communicate) and physically (e. g., a robot manipulates an object), and because their interaction is subject to communication delays, measurement uncertainty, and actuation disturbance. Typically, our components are open systems [12], which are described and verified in isolation from other components, separated by interfaces with assumptions about the environment that provide guarantees about the behavior of components. If needed, they can be turned into a closed system [12] by including a model of a specific environment.

This paper focuses on (i) lossless and instantaneous interaction between components (allows uncertainty and delay in dedicated "ether" components, e. g., sense the speed of a car precisely without measurement error), (ii) components without physical entanglement (allows separated continuous dynamics, e. g., robots drive on their own, but do not push each other), and (iii) components without synchronized communication (parallel composition of continuous dynamics, simplification to any sequential interleaving for discrete dynamics, e. g., robots can sense their environment, but not negotiate with each other).

With this focus in mind, we define the structure and behavior of a notion of components and a technique how to compose components such that safety properties about the whole system emerge from component safety proofs (e. g., robots will not collide when staying in disjoint spatial regions). We illustrate our approach with a vehicle cruise control case study.

2 Preliminaries

2.1 Differential Dynamic Logic

For specifying and verifying correctness statements about hybrid systems, we use *differential dynamic logic* (dL) [24, 25], which supports *hybrid programs* as a program notation for hybrid systems. dL models can be verified using KeYmaera X [9], which is open source and has been applied for verification of several case studies.[1] The syntax of hybrid programs is generated by the following EBNF grammar:

$$\alpha ::= \alpha;\beta \mid \alpha \cup \beta \mid \alpha^* \mid x := \theta \mid x := * \mid \{x_1' = \theta_1, \ldots, x_n' = \theta_n \,\&\, H\} \mid ?\phi \ .$$

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. $\{x' = \theta \,\&\, H\}$ describes a continuous evolution of x (x' denotes derivation with respect to time) within the evolution domain H. The test $?\phi$ checks that a condition expressed by ϕ 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.

To specify safety properties about hybrid programs, dL provides a 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 for 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):

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

We use dL in definitions and formulas to denote the the set of all dL formulas.

2.2 Notation: Variables

In dL (and thus throughout the paper) all variables are real-valued. We use V to denote a set of variables. $FV(.)$ is used as an operator on terms, formulas and hybrid programs returning only their free variables, whereas $BV(.)$ is an operator returning only their bound variables.[2] Similarly, $V(.) = FV(.) \cup BV(.)$ returns all variables (free as well as bound).

2.3 Notation: Indices

Throughout this paper, subscript indices represent enumerations (e. g., x_i). Superscript indices are used to further specify the kinds of items described by the respective variables (e. g., v^{out} represents an *output* variable). If needed, a double (super- and subscript) one-letter index is used for double numeration (e. g., x_i^j represents element j of the vector x_i).

[1] cf. `http://symbolaris.com/info/KeYmaera.html`

[2] *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 [26].

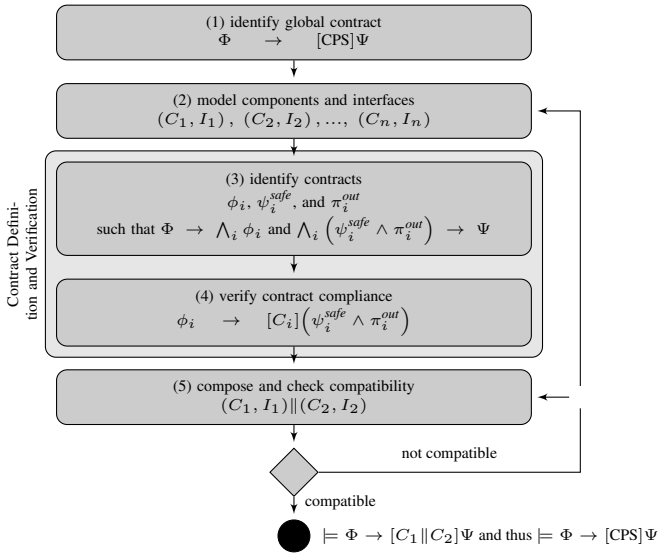


Figure 1: Steps for component-based modeling and verification

2.4 Notation: Functions

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 Modeling and Verification Steps

In this section we present the modeling and verification steps in our component-based verification approach (cf. Fig. 1).

> To illustrate the steps, we will use an example of a vehicle cruise control system, which consists of an actuator component adapting the vehicle speed according to a target speed chosen by a cruise control component. The vehicle moves *continuously*, while the control behavior is described by a *discrete* control part (e. g., choose speed and acceleration). The goal is to keep the actual speed in some range $[0, S]$, where S denotes a maximum speed. Note that we model components fully symbolically, which means that each component represents actually a family of concrete components.

The approach consists of the following steps:

(1) identify global contract: Before decomposing the system, it is important to learn what properties the system as a whole should fulfill (e. g., supported by domain experts). The global contract specifies the initial state of the whole system (Φ, e. g., initially the speed is 0) as well as its overall safety property (Ψ, e. g., the speed will stay in the desired range).

(2) model components and interfaces: Find recurring parts or natural splitting points for implementations (e. g., we split our cruise control system in a cruise controller and an actuator). The number of different components should be kept small, so that the verification effort remains low; still, there have to be sufficiently many components that can be instantiated to assemble the system. Modeling components and their interfaces is a manual effort (e. g., by modeling experts). A component has a behavior, while its interface defines public input ports and output ports, see Def. 2 and Def. 3 later.

(3) identify contracts: For each component and its interface, we identify initial states ϕ_i (e. g., initial target speed is 0), a safety property ψ_i^{safe} (e. g., actual speed does not exceed S), as well as an output contract π_i^{out} (e. g., target speed is always in the desired range), see Def. 4 later. These properties have to be chosen such that the global contract follows by *refinement* (cf. Def. 8) or *dominance* [5]: $\Phi \rightarrow \bigwedge_i \phi_i$ and $\bigwedge_i(\psi_i^{safe} \wedge \pi_i^{out}) \rightarrow \Psi$.

(4) verify contract compliance: Verify that components satisfy their contracts formally, in our case (hybrid programs and $\mathsf{d}\mathcal{L}$), with KeYmaera X.

(5) compose and check compatibility: Construct the system by connecting component ports to compose verified components in parallel, see Def. 5 later. Any component can be instantiated multiple times in the whole system (e. g., instantiate maximum speed parameters of a cruise control with actual values; connect the controller with the actuator). In order to transfer proofs about components to a global system proof, the compatibility of the components must be checked (see Theorem 1 in Section 4.2, which is proved under these compatibility assumptions). Intuitively, the *compatibility check* ensures that the values *provided* for symbolic parameters of an output port of one component instance are *compatible* with the values *required* on a connected input port of the next instance, see Def. 6 later (e. g., the controller cannot demand target speeds outside the target range).

The main result of this process is that the component safety proofs—done for compatible components in isolation—transfer to an arbitrarily large system built by instantiating these components (cf. Theorem 1).

4 Component-based Modeling

In this section we introduce essential modeling idioms and definitions for the presented steps. Section 4.1 introduces components (cf. step (2)) and their contracts (cf. step (3)). Similarly, Bauer et al. [4] show how a contract framework can be built generically. Section 4.2 introduces composition (cf. step (5)) and ensures that the local properties transfer to the overall system. Finally, Section 4.3 discusses the plausibility of composites and introduces the notion of refinement (cf. step (3)).

4.1 Components and Contracts

Components can observe a shared global state, and modify their internal state.

Definition 1 (Global Variables)**.** *The global variables* V^{global} *are a set of variables shared by all components. It contains the variable* t*, which represents the system time, is initially set to* 0*, and increases linearly with rate* 1*. None of the global variables can ever be bound in any part of any component.*

In the following paragraphs, we define components, which have a behavior (e. g., how a cruise controller chooses a target speed), and interfaces, which consist of input ports (e. g., the current actual speed received by cruise control, will be provided by the actuator) and output ports (e. g., the new target speed as provided by cruise control to the actuator). We define the behavior of a component in the canonical order of a control part followed by a plant, which enables the definition of a structured composition operation for components and interfaces.

Definition 2 (Component)**.** *A component* C *is defined as a tuple*

$$C = (\text{ctrl}, \text{plant}) \text{ , } \textit{where}$$

- ctrl *is the discrete control part of a hybrid program (HP) and does not contain continuous parts (i. e., differential equations), and*
- plant *is the continuous part of the form* $\{x_1' = \theta_1, ..., x_n' = \theta_n \& H\}$ *for* $n \in \mathbb{N}$ *i. e., an ordinary differential equation with evolution domain constraint* H*.*

The interface of a component consists of input and output ports (i. e., V^{in} and V^{out}), which can have contracts (i. e., π^{in} and π^{out}, e. g., value range for the target speed).

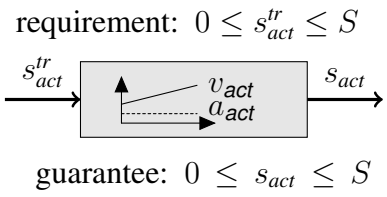
Definition 3 (Interface)**.** *An interface* I *is defined as a tuple*

$$I = \left(V^{\text{in}}, \pi^{\text{in}}, V^{\text{out}}, \pi^{\text{out}}\right) \text{ , } \textit{where}$$

- V^{in} *is a set of input variables,* V^{out} *is a set of output variables,*
- $\pi^{\text{in}} : V^{\text{in}} \to \mathsf{dL}$ *specifies an input predicate (*dL *represents the set of all logical formulas) representing input requirements and assumptions, exactly one per input variable (i. e., input port), accordingly for* $\pi^{\text{out}} : V^{\text{out}} \to \mathsf{dL}$*,*
- $\forall v \in V^{\text{in}} \ : \ V(\pi^{\text{in}}(v)) \subseteq \left(\text{V} \setminus V^{\text{in}}\right) \cup \{v\}$*, i. e., no input predicate can mention other input variables, which lets us reshuffle port ordering.*

An interface I *is called* admissible *for a component* C*, if* $(BV(\text{ctrl}) \cup BV(\text{plant})) \cap V^{\text{in}} = \emptyset$*, i. e., none of the input variables are bound in* ctrl *or* plant*.*

Consider our running example of the vehicle cruise control, where the actuator component chooses the acceleration according to a target speed (cf. Fig. 2). As illustrated in Fig. 2a, the component has a single input port to receive a target speed and a single output port to provide the current speed.



(a) Actuator illustration

$$ctrl_{act} \equiv a_{act} := \frac{s^{tr}_{act} - s_{act}}{\varepsilon};\ t^0_{act} := t \quad (1)$$

$$plant_{act} \equiv \{s'_{act} = a_{act}\ \&\ t - t^0_{act} \leq \varepsilon\} \quad (2)$$

$$\pi^{in}_{act}(s^{tr}_{act}) \equiv 0 \leq s^{tr}_{act} \leq S \quad (3)$$

$$\pi^{out}_{act}(s_{act}) \equiv 0 \leq s_{act} \leq S \quad (4)$$

(b) Formal component/interface

Figure 2: Actuator component/interface example (C_{act}, I_{act})

Fig. 2b describes this component and interface formally: The actuator receives a target speed between 0 and S on its single input port s^{tr}_{act}, cf. (3). It is a time-triggered controller with sampling period ε. The controller chooses the acceleration of the vehicle such that it will not exceed the target speed until the next run and stores the current system time, cf. (1). The plant adapts the speed accordingly and runs for at most ε time to enforce the sampling period, cf. (2). The single output port yields the resulting actual speed, which still has to be in range between 0 and S, cf. (4).

The *contract* links the component and its admissible interface, and includes information about the components initial- and target states, cf. Def. 4.

Definition 4 (Contract)**.** *Let C be a component, I be an admissible interface for C, and ϕ be a formula over the component's variables* V*, which determines the component's initial state. Let ψ^{safe} be a predicate over the component's variables* V*, i. e., a property describing the desirable target system state (i. e., a* safety property*). We define $\psi \overset{def}{\equiv} \psi^{\text{safe}} \wedge \Pi^{\text{out}}$, where $\Pi^{\text{out}} \equiv \bigwedge_{v \in V^{\text{out}}} \pi^{\text{out}}(v)$ is the conjunction of all output guarantees. The* contract *of a component C with its interface I is defined as*

$$\text{Cont}(C, I) \ \equiv\ t = 0 \wedge \phi \rightarrow [(\text{in}; \text{ctrl}; \{t' = 1, \text{plant}\})^*]\psi$$

with input in $\overset{def}{\equiv} \big(v_1 := *; ?\pi^{\text{in}}(v_1)\big); ...; \big(v_r := *; ?\pi^{\text{in}}(v_r)\big)$ *for all $v_i \in V^{\text{in}}$* .

As the input predicates are not allowed to mention other inputs, the order of inputs in *in* is irrelevant. We call a component with an admissible interface that provably satisfies its contract to be *contract compliant*. This means, if started in a state satisfying ϕ, the component only reaches states that satisfy safety ψ^{safe} and all output guarantees π^{out} when all inputs satisfy π^{in}.

In our running example of Fig. 2, the actuator component has an output guarantee $\pi^{out} \equiv (0 \leq s_{act} \leq S)$ (i. e., the speed must always be in range), and when starting from the initial conditions $\phi \equiv (s_{act} = 0 \wedge \varepsilon > 0 \wedge S > 0)$ (i. e., vehicle initially stopped) it can provably guarantee safety[3] $\psi^{safe} \equiv 0 \leq s_{act} \leq S$.

4.2 Composition of Components

Now that we have defined the structure and behavior of single components and their interfaces, we specify how to compose a number of those components by defining a syntactic composition operator for components. Differential dynamic logic follows the common assumption in hybrid systems that discrete actions do not consume time, i. e., multiple discrete actions of a program can happen instantaneously at the same real point in time. Time only passes during continuous evolution measured through t' in *plant*. Hence, if we disallow direct interaction between the controllers of components,[4] we can compose the discrete *ctrl* of multiple components in parallel by executing them sequentially in any order, while keeping their plants truly parallel through $\{x_1' = \theta_1, \ldots, x_n' = \theta_n \,\&\, H\}$. Interaction between components is then possible by observing plant output.

Such interaction, which exchanges information between components, will be defined by connecting ports when composing components through their interfaces. The port connections are represented by a mapping function $\mathcal{X}$, which assigns an output port to an input port for some number of input ports. In this paper, we focus on instantaneous lossless interaction, where the input variable v instantaneously takes on the value of the output port it is connected to, cf. $v := \mathcal{X}(v)$ in Def. 5. Other interaction patterns can be modeled by adapting Def. 5. For example, measurement with sensor uncertainty Δ is $v := *;\ ?\,(\mathcal{X}(v) - \Delta \leq v \leq \mathcal{X}(v) + \Delta)$, which yields a modified compatibility check.

As we do not require all ports to be connected, the mapping function is a partial function. Ports which are not connected become ports of the composite, while ports which are connected become internal variables.

Definition 5 (Parallel Composition)**.** *Let C_i denote one of n components*

$$C_i = (\text{ctrl}_i, \text{plant}_i) \text{ for } i \in \{1, ..., n\}$$

with their corresponding admissible interfaces

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

where $\left(V_i^{\text{in}} \cup V_i^{\text{out}} \cup \text{V}(\text{ctrl}_i) \cup \text{V}(\text{plant}_i)\right) \cap \left(V_j^{\text{in}} \cup V_j^{\text{out}} \cup \text{V}(\text{ctrl}_j) \cup \text{V}(\text{plant}_j)\right) \subseteq V^{\text{global}}$ for $i \neq j$, i. e., only variables in V^{global} are shared between components, and let

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

be a partial (i. e., not every input must be mapped), injective (i. e., every output is only mapped to one input) function, connecting inputs to outputs. We define $\mathcal{I}^{\mathcal{X}}$ as the domain of $\mathcal{X}$ (i. e., all variables $x \in V^{\text{in}}$ such that $\mathcal{X}(x)$ is defined) and $\mathcal{O}^{\mathcal{X}}$ as the image of $\mathcal{X}$ (i. e., all variables $y \in V^{\text{out}}$ such that $y = \mathcal{X}(x)$ holds for some $x \in V^{\text{in}}$).

$$(C, I) \overset{def}{\equiv} ((C_1, I_1) \| ... \| (C_n, I_n))_{\mathcal{X}}$$

is defined as the composite of n components and their interfaces (with respect to $\mathcal{X}$), where

[4] Def. 5 restricts how variables between components can be shared.

- *the sensing for non-connected inputs remains unchanged*

$$\text{in} \equiv \underbrace{v_k := *; ?\pi^{\text{in}}(v_k); \ldots; v_s := *; ?\pi^{\text{in}}(v_s)}_{\textit{open inputs}} \textit{ for } \{v_k, \ldots, v_s\} = V^{\text{in}} \setminus \mathcal{I}^{\mathcal{X}}$$

- *the order in which the control parts (and the respective port mappings) are executed is chosen non-deterministically (considering all the $n!$ possible permutations of $\{1, ..., n\}$), so that connected ports become internal behavior of the composite component*

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

with $\text{ports}_i \overset{\textit{def}}{\equiv} \underbrace{v_j := \mathcal{X}(v_j); \ldots; v_r := \mathcal{X}(v_r)}_{\textit{connected inputs}} \textit{ for } \{v_j, \ldots, v_r\} = \mathcal{I}^{\mathcal{X}} \cap V_i^{\text{in}}$,

- *continuous parts are executed in parallel, staying inside* all *evolution domains*

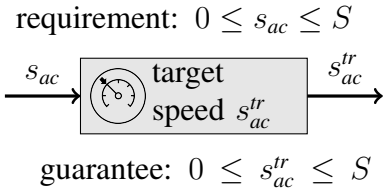
$$\text{plant} \equiv \Big\{ \underbrace{x_1^{(1)\prime} = \theta_1^{(1)}, \ldots, x_1^{(k)\prime} = \theta_1^{(k)}}_{\textit{component } C_1}, \ldots, \underbrace{x_n^{(1)\prime} = \theta_n^{(1)}, \ldots, x_n^{(m)\prime} = \theta_n^{(m)}}_{\textit{component } C_n} \ \& \ H_1 \wedge \ldots \wedge H_n \Big\} \ ,$$

- *the respective sets of variables are merged, so $V^{\text{in}} = \bigcup_{1 \le i \le n} V_i^{\text{in}} \setminus \mathcal{I}^{\mathcal{X}}$, $V^{\text{out}} = \bigcup_{1 \le i \le n} V_i^{\text{out}} \setminus \mathcal{O}^{\mathcal{X}}$, i. e., ports not connected within the composite component remain input and output variables of the resulting interface,*

- *input port requirements of all interfaces are preserved, except for connected inputs, i. e., $\pi^{\text{in}} : V^{\text{in}} \to \mathsf{d}\mathcal{L}$ becomes $\pi^{\text{in}}(v)$, accordingly for $\pi^{\text{out}}(v)$:*

$$\pi^{\text{in}}(v) \equiv \begin{cases} \pi_1^{\text{in}}(v) & \textit{if } v \in V_1^{\text{in}} \setminus \mathcal{I}^{\mathcal{X}} \\ \ldots & \\ \pi_n^{\text{in}}(v) & \textit{if } v \in V_n^{\text{in}} \setminus \mathcal{I}^{\mathcal{X}} \end{cases} \qquad \pi^{\text{out}}(v) \equiv \begin{cases} \pi_1^{\text{out}}(v) & \textit{if } v \in V_1^{\text{out}} \setminus \mathcal{O}^{\mathcal{X}} \\ \ldots & \\ \pi_n^{\text{out}}(v) & \textit{if } v \in V_n^{\text{out}} \setminus \mathcal{O}^{\mathcal{X}} \end{cases} .$$

To demonstrate parallel composition in our running example, we first introduce a cruise controller component (cf. Fig. 3). The cruise control selects a target speed from the interval, but keeps the difference between the current (received) speed and the chosen target speed below δ_S (cf. (5)–(6)). That way, the acceleration set by the actuator component is bounded by δ_S/ε (i. e., the vehicle does not accelerate too fiercely). We connect this cruise controller component to the actuator component (cf. Fig. 2), as illustrated in Fig. 4.



(a) Cruise controller illustration

$$\mathit{ctrl}_{ac} \equiv s^{tr}_{ac} := *; \tag{5}$$

$$?\big(0 \leq s^{tr}_{ac} \leq S \wedge |s^{tr}_{ac} - s_{ac}| \leq \delta_S\big) \tag{6}$$

$$\pi^{in}_{ac}(s_{ac}) \equiv 0 \leq s_{ac} \leq S \tag{7}$$

$$\pi^{out}_{ac}(s^{tr}_{ac}) \equiv 0 \leq s^{tr}_{ac} \leq S \tag{8}$$

(b) Formal component/interface

Figure 3: Cruise controller component/interface example (C_{ac}, I_{ac})

Remark 1. *Note that verifying the hybrid program for a composite according to Def. 5 would require a proof of each of the $n!$ branches of* ctrl *individually, as they all differ slightly. For a large number of components, this entails a huge proof effort. Previous non-component-based case studies (e. g., [15, 18, 19]), therefore, chose only one specific ordering. Our component-based approach verifies all possible orderings at once, because the permutations are all proven correct as part of proving Theorem 1 below in this paper.*

Remark 2. *This definition of parallel composition uses a conjunction of all evolution domains, which resembles synchronization on the most restrictive component (i. e., as soon as the first and most restrictive condition is no longer fulfilled all plants have to stop and hand over to* ctrl*). A more liberal component might be forced to execute its control part because the evolution domain of a more restrictive component did no longer hold. For example a component increasing a counter on every run of its control is then forced to count although its own evolution domain might have allowed it to postpone control. If this is undesired, a component's control can be defined as* $\text{ctrl}_i \cup ?\mathit{true}$*, which would allow the component to skip when forced to run its control part.*

Remark 3. *We define this composition operation for any number of components, since it is* not associative*, because the composition of three components results in $3! = 6$ possible execution orders, whereas composing two components and adding a third yields only $2! + 2! = 4$ of the possible 6 execution orders.*

Note that Def. 5 replaces the non-deterministic input guarded by a test from Def. 2 with a deterministic assignment that represents instantaneous and lossless interaction between components (i. e., *ports*$_i$), as illustrated in Fig. 4. Hence, the respective output guarantees and input assumptions must match.

For instance, a cruise controller component providing speeds $0 \leq s^{tr}_{ac} \leq 70$ is compatible with an actuator demanding $0 \leq s^{tr}_{act} \leq 100$, but a controller component providing speeds $0 \leq s^{tr}_{ac} \leq 100$ is not compatible with an actuator demanding $0 \leq s^{tr}_{act} \leq 70$, since the controller component might provide a speed s^{tr}_{ac} which is outside the validity interval of the actuator (i. e., $s^{tr}_{ac} = 100$ is allowed, but $s^{tr}_{act} = 100$ is not).

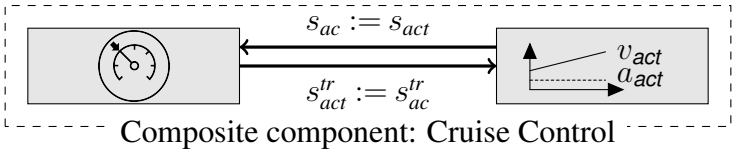


Figure 4: Cruise control composed of a cruise controller and an actuator by Def. 5. The port connections $\mathcal{X} = \{(s_{ac} \mapsto s_{act}), (s^{tr}_{act} \mapsto s^{tr}_{ac})\}$ replace the input port $s^{tr}_{act} := *; ?(0 \leq s^{tr}_{act} \leq S)$ with an internal port assignment $s^{tr}_{act} := s^{tr}_{ac}$, provided the compatibility check $[s^{tr}_{act} := s^{tr}_{ac}]\,(\pi^{out}_{ac}(s^{tr}_{ac}) \rightarrow \pi^{in}_{act}(s^{tr}_{act}))$ succeeds, cf. Def. 6, and accordingly for the second port.

Definition 6 (Compatible Composite)**.** *The composite of n components with interfaces $((C_1, I_1)\|...\|(C_n, I_n))_{\mathcal{X}}$ is a* compatible composite *iff*

$$\text{CPO}(I_i) \equiv [v := \mathcal{X}(v)]\left(\pi^{\text{out}}_j(\mathcal{X}(v)) \rightarrow \pi^{\text{in}}_i(v)\right)$$

is valid for all input ports $v \in \mathcal{I}^{\mathcal{X}} \cap V^{\text{in}}_i$, for all interfaces I_i and where I_j is the interface containing the port that is connected to the input port v of I_i. We call CPO(C_i) *the* compatibility proof obligation *for the interfaces I_i and say the interfaces I_i are* compatible *(with respect to $\mathcal{X}$) if* CPO(I_i) *holds.*

In other words, $((C_1, I_1)\|...\|(C_n, I_n))_{\mathcal{X}}$ is a *compatible* composite if all internal port connections are appropriate, i. e., if the guarantee of the output port implies the requirements of the respective input port to which it is connected.

Now that we have defined components and interfaces, their contracts, and how to compose them to form larger composites, we prove that the contracts of single components transfer to composites if compatible.

Theorem 1 (Composition Retains Contracts)**.** *Let C_1 and C_2 be components with admissible interfaces I_1 and I_2 that are* contract compliant *(i. e., their contracts are valid)*

$$\models t = 0 \wedge \phi_1 \rightarrow [(\text{in}_1; \text{ctrl}_1; \{t' = 1, \text{plant}_1\})^*]\,(\psi_1) \text{ and} \tag{9}$$
$$\models t = 0 \wedge \phi_2 \rightarrow [(\text{in}_2; \text{ctrl}_2; \{t' = 1, \text{plant}_2\})^*]\,(\psi_2) \tag{10}$$

and compatible *with respect to $\mathcal{X}$ (i. e., compatibility proof obligations are valid)*

$$\text{for all input ports } v \in \mathcal{I}^{\mathcal{X}} \cap V^{\text{in}}_2 \;:\; \models [v := \mathcal{X}(v)]\left(\pi^{\text{out}}_1(\mathcal{X}(v)) \rightarrow \pi^{\text{in}}_2(v)\right) \text{ and} \tag{11}$$
$$\text{for all input ports } v \in \mathcal{I}^{\mathcal{X}} \cap V^{\text{in}}_1 \;:\; \models [v := \mathcal{X}(v)]\left(\pi^{\text{out}}_2(\mathcal{X}(v)) \rightarrow \pi^{\text{in}}_1(v)\right) \;. \tag{12}$$

Then, the parallel composition $C_3, I_3 = ((C_1, I_1)\|(C_2, I_2))_{\mathcal{X}}$ satisfies the contract

$$\models t = 0 \wedge (\phi_1 \wedge \phi_2) \rightarrow [(\text{in}_3; \text{ctrl}_3; \{t' = 1, \text{plant}_3\})^*](\psi_1 \wedge \psi_2) \tag{13}$$

with in_3*,* ctrl_3*, and* plant_3 *according to Def. 5.*

The proof for Theorem 1 can be found in Appendix A. □

This central theorem (along with a generalization to n components, cf. Appendix A) allows us to infer how properties from single components transfer to their composition. As such, it suffices to prove the properties for the components and conclude that a similar property holds for the composite, without explicitly having to verify it. The composite contract states that, considering both pre-conditions hold (i. e., $\phi_1 \wedge \phi_2$), all states reached by the parallel execution of the components, both post-conditions hold (i. e., $\psi_1 \wedge \psi_2$).

In order to ensure that the result of our parallel composition is again a component with an interface by Def. 2 and Def. 3, we have to compare the resulting construct to the definition. In particular, we have to ensure the following properties:

- $(\mathrm{V}^{in} \cup \mathrm{V}^{global}) \cap (BV(\mathit{ctrl}) \cup BV(\mathit{plant})) = \emptyset$:
  We know from the definition of *ctrl* in Def. 5 that

  $$BV(\mathit{ctrl}) = \bigcup_{1 \leq i \leq n} (BV(\mathit{ctrl}_i) \cup BV(\mathit{ports}_i)) \quad .$$

  Since C_i/I_i are components/interfaces, $\left(\mathrm{V}_i^{in} \cup \mathrm{V}_i^{global}\right) \cap BV(\mathit{ctrl}_i) = \emptyset$ holds. Furthermore, since $\mathrm{V}^{in} \subseteq \bigcup_{1 \leq i \leq n} \mathrm{V}_i^{in}$, we know that $(\mathrm{V}^{in} \cup \mathrm{V}^{global}) \cap \bigcup_{1 \leq i \leq n} BV(\mathit{ctrl}_i) = \emptyset$.
  From $BV(\mathit{ports}_i) \subseteq \mathcal{I}^{\mathcal{X}}$ for all i, $\mathrm{V}^{in} \cap \mathcal{I}^{\mathcal{X}} = \emptyset$, and $BV(\mathit{ports}_i) \subseteq \mathrm{V}_i^{in}$, $\mathrm{V}_i^{in} \cap \mathrm{V}^{global} = \emptyset$ for all i (by Def. 2), and $\mathrm{V}^{in} \subseteq \bigcup_{1 \leq i \leq n} \mathrm{V}_i^{in}$ by Def. 5), we further get that $(\mathrm{V}^{in} \cup \mathrm{V}^{global}) \cap \bigcup_{1 \leq i \leq n} BV(\mathit{ports}_i) = \emptyset$. Hence, $\left(\mathrm{V}^{in} \cup \mathrm{V}^{global}\right) \cap BV\left(\mathit{ctrl}\right) = \emptyset$.

  Since $BV(\mathit{plant}) = \bigcup_{1 \leq i \leq n} BV(\mathit{plant}_i)$ and since C_i are components we know $(\mathrm{V}_i^{in} \cup \mathrm{V}^{global}) \cap BV(\mathit{plant}_i) = \emptyset$. Hence, also $(\mathrm{V}^{in} \cup \mathrm{V}^{global}) \cap BV(\mathit{plant}) = \emptyset$ and thus we conclude $(\mathrm{V}^{in} \cup \mathrm{V}^{global}) \cap (BV(\mathit{ctrl}) \cup BV(\mathit{plant})) = \emptyset$.

- Global variables must not be bound:
  Follows immediately, since global variables cannot be bound anywhere.

- $\forall v \in \mathrm{V}^{in} \ : \ V(\pi^{in}(v)) \subseteq \left(V \setminus \mathrm{V}^{in}\right) \cup \{v\}$:
  Since the definition $\pi^{in}(v)$ is defined as $\pi_i^{in}(v)$ for $v \in \mathrm{V}_i^{in}$, and since C_i/I_i are components/interfaces, this condition transfers to the composite. □

As a result, composite components can be used as components in yet another composition.

4.3 Plausibility and Refinement

Although, through Theorem 1, safety of the composite is ensured, the contract *Cont* might be vacuously true, if the precondition ϕ is not satisfiable. Even if that is avoided for single components, their composition (conjunction) could still not be satisfiable, because they might share global variables.

For example, assume two components with their interface $A = (C_A, I_A)$ and $B = (C_B, I_B)$ that have neither input- nor output-variables, but share a single global variable $\mathrm{V}_1^{global} = \mathrm{V}_2^{global} = \{T\}$.

Both control parts and both plants are empty. The contract of A is $\phi_A \equiv t = 0 \wedge T > 0$ and $\psi_A^{safe} \equiv T > 0$. The contract of B is $\phi_B \equiv t = 0 \wedge T < 0$ and $\psi_B^{safe} \equiv T < 0$. Each contract is valid, as $(t = 0 \wedge T > 0) \rightarrow [?true]T > 0$ and $(t = 0 \wedge T < 0) \rightarrow [?true]T < 0$ always hold. However, even though their composition is again a safe component according to Theorem 1, the resulting contract is vacuously true: $(t = 0 \wedge T > 0 \wedge T < 0) \rightarrow [?true]\,(T > 0 \wedge T < 0)$.

Definition 7 (Plausible Composite)**.** *Let C_i be* compatible *components, with admissible interfaces I_i. We call the parallel composition $(\|_i\,(C_i, I_i))_{\mathcal{X}}$ of these components a* plausible composite*, iff $\bigwedge_i \phi_i$ is satisfiable. $(\|_i\,(C_i, I_i))_{\mathcal{X}}$ is a* plausible *composite if the pre-condition of the resulting contract remains satisfiable (assuming that its components were satisfiable).*

In step 3 of Section 3, we use the notion of *refinement* (or *dominance*), which is inspired by the work of Benvenuti et al. [5], who defined *dominance checking*. A contract refines another contract, if under weaker assumptions the component promises stronger guarantees. Similar notions of refinement can be found in the literature (e. g., [3],[20]).

Definition 8 (Contractual Refinement)**.** *Let (C, I) be a component with its interface and let* $\text{Cont}_1(C, I)$ *and* $\text{Cont}_2(C, I)$ *be contracts, with pre-conditions ϕ_1 and ϕ_2 and post-conditions ψ_1 and ψ_2. We say* Cont_1 *refines* Cont_2*, iff $(\phi_1 \rightarrow \phi_2) \wedge (\psi_2 \rightarrow \psi_1)$.*

If *Cont*$_1$ refines *Cont*$_2$, we know from a proof of *Cont*$_2$ that *Cont*$_1$ must hold as well, see Theorem 2.

Theorem 2 (Refinement Theorem)**.** *Let (C, I) be a component with its interface, with contracts* $\text{Cont}_1(C, I)$ *and* $\text{Cont}_2(C, I)$*. If* Cont_1 *is valid and* Cont_2 *refines* Cont_1 *under these contracts, then* Cont_2 *is valid.*

The proof for Theorem 2 can be found in Appendix A. □

5 Case Study: Vehicle Cruise Control

To illustrate our approach, we used a running example of a simple *vehicle cruise control system*. The overall system requirement was to keep the speed s_{act} in a desired range $[0, S]$ at all times, i. e., $0 \leq s_{act} \leq S \rightarrow [CruiseControl]0 \leq s_{act} \leq S$. We split the system into two components, cf. Fig. 5: an actuator component adapts speed according to a target speed s_{act}^{tr} provided by an automated cruise controller component as s_{ac}^{tr}. If the automated cruise controller component (Fig. 3) provides a valid target speed to the actuator (i. e., $0 \leq s_{act}^{tr} \leq S$), the actuator component (Fig. 2) ensures to keep the actual speed s_{ac} in the desired range (i. e., $0 \leq s_{act} \leq S$), thus ensuring the overall system property. Additionally, the actuator provides the current speed on an output port that is read by the controller, acting as a feedback loop.

The detailed components and their interfaces according to Def. 2 and Def. 3 are listed for easy reference in Model 1 and Model 2.

In its control part, the actuator component first sets a new acceleration. It chooses the target acceleration in a way that guarantees that the target speed is not exceeded after ε time units. Then

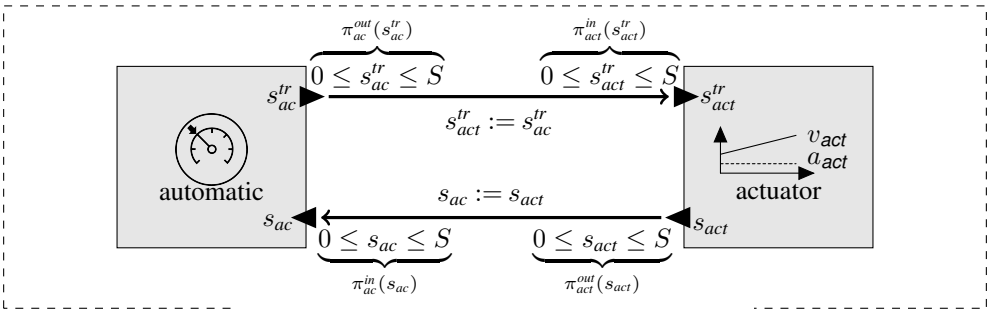


Figure 5: Cruise control composed of the automatic cruise controller and an actuator. Port conditions of connected ports must be compatible.

it stores the current time, which is necessary to ensure a plant runtime of at most ε time units, cf. (14). In the plant the speed is changed according to the selected acceleration and the evolution domain ensures the maximum runtime of ε, cf. (15). The actuator component has one input port s^{tr}_{act}, on which it receives a positive target speed that is bound by the maximum speed S, cf. (16). Furthermore, it has one output port s_{act}, which provides the actual current speed, which is guaranteed to be in the interval $0 \leq s_{act} \leq S$, cf. (17).

The automated cruise controller component's control part first chooses an arbitrary target speed. Then it checks that the selected speed is within the interval $0 \leq s^{tr}_{ac} \leq S$ and not further from the current speed than δ_S, thus ensuring a smooth acceleration, cf. (18). In other words, this automated cruise controller implementation selects target speeds at random from an interval around the current speed. The interval bound δ_S influences how drastically the target speed differs from the current speed. The component's plant is empty. The automated cruise controller component has one input port s_{ac}, on which it receives the current target speed, which is assumed to be non-negative and bound by S, cf. (20). Furthermore, it has one output port s^{tr}_{ac}, on which it outputs the chosen target speed, which is guaranteed to be within the interval $0 \leq s^{tr}_{ac} \leq S$, cf. (21).

Model 1 Component and Interface of the actuator component

$$C_{act} = (\mathit{ctrl}_{act}, \mathit{plant}_{act})$$

$$\mathit{ctrl}_{act} \equiv a_{act} := \frac{s^{tr}_{act} - s_{act}}{\varepsilon}; t^0_{act} := t \tag{14}$$

$$\mathit{plant}_{act} \equiv \{s'_{act} = a_{act} \;\&\; t - t^0_{act} \leq \varepsilon\} \tag{15}$$

$$I_{act} = \left(\mathrm{V}^{in}_{act}, \pi^{in}_{act}, \mathrm{V}^{out}_{act}, \pi^{out}_{act}\right)$$

$$\mathrm{V}^{in}_{act} = \{s^{tr}_{act}\},\ \pi^{in}_{act}(s^{tr}_{act}) \equiv 0 \leq s^{tr}_{act} \leq S \tag{16}$$

$$\mathrm{V}^{out}_{act} = \{s_{act}\},\ \pi^{out}_{act}(s_{act}) \equiv 0 \leq s_{act} \leq S \tag{17}$$

Model 2 Component and Interface of the automated cruise controller component

$$C_{ac} = (\mathit{ctrl}_{ac}, \mathit{plant}_{ac})$$
$$\mathit{ctrl}_{ac} \equiv s_{ac}^{tr} := *; ?\big(0 \leq s_{ac}^{tr} \leq S \land |s_{ac}^{tr} - s_{ac}| \leq \delta_S\big) \tag{18}$$
$$\mathit{plant}_{ac} \equiv \{\} \tag{19}$$

$$I_{ac} = \big(\mathrm{V}_{ac}^{in}, \pi_{ac}^{in}, \mathrm{V}_{ac}^{out}, \pi_{ac}^{out}\big)$$
$$\mathrm{V}_{ac}^{in} = \{s_{ac}\},\ \pi_{ac}^{in}(s_{ac}) \equiv 0 \leq s_{ac} \leq S \tag{20}$$
$$\mathrm{V}_{ac}^{out} = \{s_{ac}^{tr}\},\ \pi_{ac}^{out}(s_{ac}^{tr}) \equiv 0 \leq s_{ac}^{tr} \leq S \tag{21}$$

Following Def. 4, we derive contracts for each component, which consists of initial conditions ϕ, cf. (22)–(23), safety conditions ψ^{safe}, cf. (24), and the output port conditions, cf. (17) and (21). Initially, maximum speed $S > 0$ and cycle time $\varepsilon > 0$ must be known. Additionally, the automated cruise controller initializes $s_{ac}^{tr} = 0$ and $\delta_S > 0$, cf. (22). The actuator restricts the initial speed to $0 \leq s_{act} \leq S$, cf. (23). Since the automatic cruise controller component has no additional safety property, the sole safety property ψ_{act}^{safe} restricts speed of the actuator component to the interval $0 \leq s_{act} \leq S$, cf. (24).

$$\phi_{act} \equiv 0 \leq s_{act} \leq S \land \varepsilon > 0 \land S > 0 \tag{22}$$
$$\phi_{ac} \equiv s_{ac}^{tr} = 0 \land \varepsilon > 0 \land S > 0 \land \delta_S > 0 \tag{23}$$
$$\psi_{act}^{safe} \equiv 0 \leq s_{act} \leq S \tag{24}$$

The set of global variables follows accordingly (cf. Def. 1): $\mathrm{V}^{global} = \{\varepsilon, S, t\}$.

After verifying[5] both contracts *Cont*(C_{ac}, I_{ac}) and *Cont*(C_{act}, I_{act}), we want to compose the components to get the overall system, using the mapping function $\mathcal{X} = (s_{ac} \mapsto s_{act}, s_{act}^{tr} \mapsto s_{ac}^{tr})$ (cf. Fig. 4). Therefore, we have to check the compatibility proof obligations for both connected ports. As we connect the output port of the automated cruise controller providing the target speed, to the respective input port of the actuator component, we have to verify *CPO*(I_{act}, I_{ac}), cf. (26). Since we furthermore connect the actual speed as provided by the actuator to the respective input port of the automated cruise controller, we furthermore have to verify *CPO*(I_{ac}, I_{act}), cf. (25). Then the overall system property directly follows from the contract of the actuator component.

$$\mathit{CPO}(I_{ac}, I_{act}) \equiv [s_{ac} := s_{act}]\left(\pi_{act}^{out}(s_{act}) \rightarrow \pi_{ac}^{in}(s_{ac})\right) \equiv [s_{ac} := s_{act}]\,(0 \leq s_{act} \leq S \rightarrow 0 \leq s_{ac} \leq S) \tag{25}$$

$$\mathit{CPO}(I_{act}, I_{ac}) \equiv [s_{act}^{tr} := s_{ac}^{tr}]\left(\pi_{ac}^{out}(s_{ac}^{tr}) \rightarrow \pi_{act}^{in}(s_{act}^{tr})\right) \equiv [s_{act}^{tr} := s_{ac}^{tr}]\,(0 \leq s_{ac}^{tr} \leq S \rightarrow 0 \leq s_{act}^{tr} \leq S) \tag{26}$$

[5] All proofs were done in KeYmaera X [9].

Splitting a system into components reduces the model complexity considerably, since a component needs to know neither about the differential equation systems of other components, nor about their control choices. In combined models, we have to analyze all the possible permutations of control choices, while in the component-based approach, by Theorem 1 we can guarantee correctness for all possible sequential orderings, without the proof effort entailed by listing them explicitly. The first part of Table 1 (i. e., "Automated") compares the proof effort of the component based version. Although, the proof effort is relatively low, the combined number of proof steps is evidently smaller when using our approach, rather than a monolithic model.

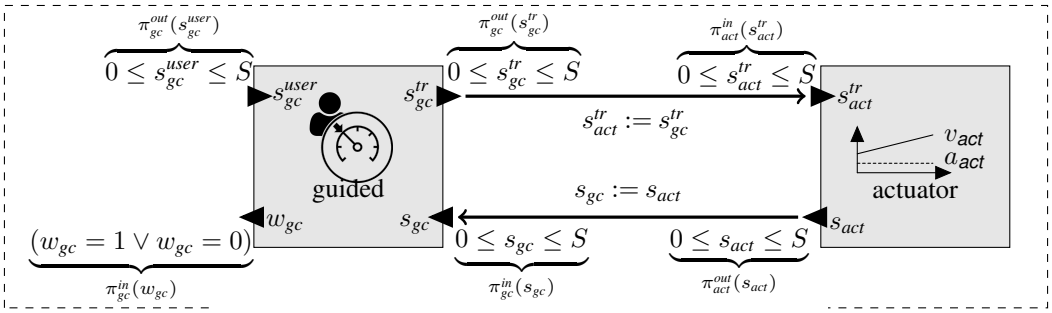


Figure 6: Cruise control composed of the guided cruise controller and an actuator. Port conditions of connected ports must be compatible. Not-connected ports remain open.

The benefit of component-based verification becomes even larger when replacing components in a system, cf. Fig. 6. For example, we can easily replace the automated cruise controller from Model 2 with a more sophisticated *guided cruise controller*, cf. Model 3. The guided controller receives a user speed s^{user}_{gc} on an additional input port, which represents a user suggestion for the new target speed, cf. (31). In order to keep the acceleration at a smooth level, the guided cruise controller checks if the user chosen speed is too far from the current speed and uses an additional output port to issues warnings if not, cf. (32). More precisely, if this user speed is close enough to the current actual speed, it is chosen as target speed and no warning is issued (i. e., $w = 0$), cf. (29). Otherwise, if the user speed is too high or low, the target speed is modified and a warning is issued (i. e., $w = 1$). If the speed is to high, the guided controller uses the current actual speed increased by δ_S as target speed, cf. (27), and if the speed is to low, the guided controller uses the current actual speed decreased by δ_S as target speed, cf. (28).

The guided cruise controller—like the automated cruise controller—has an empty *plant*, cf. (30).

Again we derive a contract for the component, following Def. 4, which consists of an initial condition ϕ, cf. (33), and the output port conditions, cf. (32). Again the controller has no additional safety condition. Initially, maximum speed $S > 0$ and cycle time $\varepsilon > 0$ must be known. Additionally, the guided cruise controller initializes $s^{tr}_{gc} = 0$ and $\delta_S > 0$. Furthermore, it restricts the initial actual speed to $0 \le s_{gc} \le S$, the user speed $s^{user}_{gc} = s_{gc}$ and the state of the warning flag $w_{gc} = 0$, cf. (33).

$$\phi_{gc} \equiv s^{tr}_{gc} = 0 \wedge \varepsilon > 0 \wedge S > 0 \wedge \delta_S > 0 \wedge 0 \le s_{gc} \le S \wedge s^{user}_{gc} = s_{gc} \tag{33}$$

Model 3 Component and Interface of the guided cruise controller component

$$C_{gc} = \left(\mathit{ctrl}_{gc}, \mathit{plant}_{gc}\right)$$

$$\mathit{ctrl}_{gc} \equiv \Big(\Big(?\left(s_{gc}^{\mathit{user}} - s_{gc} \geq \delta_S\right); s_{gc}^{\mathit{tr}} := s_{gc} + \delta_S; w_{gc} := 1\Big) \cup \tag{27}$$

$$\Big(?\left(s_{gc} - s_{gc}^{\mathit{user}} \geq \delta_S\right); s^{\mathit{tr}} := s_{gc} - \delta_S; w_{gc} := 1\Big) \cup \tag{28}$$

$$\Big(?\left(\left|s_{gc}^{\mathit{user}} - s_{gc}\right| < \delta_S\right); s_{gc}^{\mathit{tr}} := s_{gc}^{\mathit{user}}; w := 0\Big)\Big) \tag{29}$$

$$\mathit{plant}_{gc} \equiv \{\} \tag{30}$$

$$I_{gc} = \left(\mathrm{V}_{gc}^{\mathit{in}}, \pi_{gc}^{\mathit{in}}, \mathrm{V}_{gc}^{\mathit{out}}, \pi_{gc}^{\mathit{out}}\right)$$

$$\mathrm{V}_{gc}^{\mathit{in}} \equiv \{s_{gc}, s_{gc}^{\mathit{user}}\},\ \pi_{gc}^{\mathit{in}} \equiv \{(s_{gc} \mapsto 0 \leq s_{gc} \leq S),\ (s_{gc}^{\mathit{user}} \mapsto 0 \leq s_{gc}^{\mathit{user}} \leq S)\} \tag{31}$$

$$\mathrm{V}_{gc}^{\mathit{out}} \equiv \{s_{gc}^{\mathit{tr}}, w_{gc}\},\ \pi_{gc}^{\mathit{out}} \equiv \{(s_{gc}^{\mathit{tr}} \mapsto 0 \leq s_{gc}^{\mathit{tr}} \leq S), (w_{gc} \mapsto (w_{gc} = 1 \vee w_{gc} = 0)\} \tag{32}$$

Again, the set of global variables follows accordingly (cf. Def. 1): $\mathrm{V}^{\mathit{global}} = \{\varepsilon, S, \delta_S, t\}$.

After verifying the user guided cruise control component, we only have to re-check the compatibility proof obligations. In a monolithic model, in contrast, the whole system including the actuator component must be re-verified. In this case, the advantage of our approach is even more obvious, as the second part of Table 1 (cf. "Guided") shows.

Note, that in the monolithic system using the guided cruise controller two ports remain unconnected (i. e., the input port providing the desired speed and the output port issuing warnings), which can, for instance, be connected to a user-interface, where a driver chooses a speed and receives a warning if the difference is too high. Despite those unconnected ports, the model can be verified—a suitable user-interface component can be connected later on, if the respective port conditions are fulfilled.

6 Related Work

CPS Verification. Hybrid automata [2] are popular for modeling CPS, and mainly verified using reachability analysis. Unlike hybrid programs, hybrid automata are not compositional, i. e., for a hybrid automaton it is not sufficient to establish a property about its parts in order to establish a property about the automaton. Techniques such as assume-guarantee reasoning or hybrid I/O automata [16], which are an extension of hybrid automata with input- and output-ports, address this issue. Our approach here shares some of the goals with hybrid I/O automata and also uses I/O ports. But we target compositional reasoning for *hybrid programs*, where the execution order of statements is relevant, so that our approach defines how parallel composition results in interleaving of hybrid programs. Furthermore, we define compositional modeling for hybrid programs such that *theorem proving* of the entire system is reduced to proving properties about the components and *simple composition checks*. Hybrid process algebras (e. g., Hybrid χ [27], HyPA [23]) are specifically developed as compositional modeling formalisms to describe behavior and interaction of processes using algebraic equations. For verification purposes by simulation or reachability

Table 1: Case study summary

	Model		Proof		
	Description	Dim.	Steps	Branches	Time [s]
Automated	Actuator component	5	47	4	0.19
	Automated cruise controller component	3	55	4	0.12
	Compatibility proof obligation *CPO*(I_{ac}, I_{act})	2	3	1	0.02
	Compatibility proof obligation *CPO*(I_{act}, I_{ac})	2	3	1	0.02
	Sum Component Effort		**108**	**10**	**0.35**
	Monolithic system $(act; ac) \cup (ac; act)$	7	170	11	0.83
Guided	Actuator component (proof reused)		–	–	–
	Guided cruise controller component	4	435	17	1.00
	Compatibility proof obligation *CPO*(I_{gc}, I_{act})	2	3	1	0.02
	Compatibility proof obligation *CPO*(I_{act}, I_{gc})	2	3	1	0.02
	Sum Component Effort		**441**	**19**	**1.04**
	Monolithic system $(act; gc) \cup (gc; act)$	9	955	53	7.71

analysis, translations from Hybrid χ into hybrid automata and timed automata exist, so even though modeling is compositional, verification still falls back to monolithic analysis. We, in contrast, focus on exploiting compositionality in the proof.

Component-based CPS Modeling. Damm et al. [6] present a component-based design framework for controllers of hybrid systems with a focus on reaction times. The framework checks connections when interconnecting components: alarms propagated by an out-port must be handled by the connected in-ports. We, too, check component compatibility, but for contracts, and we focus on transferring proofs from components to the system level.

Focusing on architectural properties, Ruchkin et al. [29] propose a component-based modeling approach for hybrid-systems. Although they do not transfer verification results from components to composites, their definitions have been an inspiration for our notion of components. Ringert et al. [28] model CPS as Component and Connector (C&C) architectures using automata to describe solely the discrete behavior. They verify the translated models of single components, but do not provide guarantees about verified compositions.

Interface algebras (cf. [1, 10]) are formalisms that separate component-based models into interface models and component models. Similar to our approach, the component model describes what a component does, while the interface model defines how the component can be used. It is often distinguished between interfaces with and without state, where stateful interfaces are usually viewed as concurrent games. Our approach is similar to a stateless interface algebra [1]. Similarly, Bauer et al. [4] show how a contract framework can be built generically. While useful for inspiration, these approaches focus on modeling aspects and do not consider verification.

Verification. Madl et al. [17] model real-time event-driven systems. Their models can be transformed to UPPAAL (cf. [13]) timed automata, restricting the continuous part of their models to

time instead of arbitrary physical behavior (e. g., movement). Moreover, their analysis targets the *entire composition* of timed automata, thus defeating the advantages of components for verification.

A field closely related to component-based verification is assume-guarantee reasoning (AGR, e. g., [8, 11]), which was originally developed as a device to counteract the state explosion problem in model checking by decomposing a verification task into subtasks. In AGR, individual components are analyzed together with *assumptions* about their context and *guarantees* about their behavior (i. e., a component's contract). AGR rules need to exercise care for circularity in the sense that the approaches verify one component in the context of the other and vice-versa, like Frehse et al. [8] (using Hybrid Labeled Transition Systems as abstraction for Hybrid I/O-Automata) and Henzinger et al. [11] (using hierarchical hybrid systems based on hybrid automata). However, existing approaches are often limited to linear dynamics, cannot handle continuity or use corresponding reachability analysis or model checking techniques. In $\mathsf{d}\mathcal{L}$, in contrast, we can handle non-linear dynamics and focus on theorem proving.

In summary, only few component-based approaches handle generic CPS with both discrete and continuous aspects (e. g., [6, 17, 29]), but those do not yet focus on the impact on formal verification. Related techniques for CPS and hybrid systems verification focus mainly on timed automata, hybrid process algebras, and hybrid automata with linear dynamics or end up in monolithical verification.

7 Conclusion and Future Work

We presented an approach for component-based modeling and verification of CPS that (i) splits a CPS into components, (ii) verifies a contract for each of these components and (iii) composes component instances in a way that transfers the component contracts to a composite contract. Our approach makes hybrid system verification more modular at the scale of components, and combines the advantages of component-based modeling approaches (e. g., well structured models, reduced model complexity, simplified model evolution) with the advantages of formal verification (e. g., guaranteed contract compliance).

Currently, our approach is limited to global properties that are stated relative to the initial system state. Port conditions are only allowed to mention global variables and the port variable itself, which prevents conditions on the change of a port since the last measurement (e. g., how far has a vehicle moved since the beginning vs. how far has it moved since the last measurement). This restriction can be removed with ports that remember their previous value and relate measurements over time. Additionally, we plan to (i) introduce further composition operations (e. g., sensing with measurement errors), (ii) provide further component extensions (e. g., multi-cast ports), and (iii) provide tool support to instantiate and compose components, and to generate their hybrid programs.

References

[1] de Alfaro, L., Henzinger, T.A.: Interface theories for component-based design. In: Henzinger, T.A., Kirsch, C.M. (eds.) Embedded Software, First Int. Workshop, EMSOFT 2001, Oct., 8-10, 2001, Proc. LNCS, vol. 2211, pp. 148–165. Springer (2001)

[2] Alur, R., Courcoubetis, C., Henzinger, T.A., Ho, P.: 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 (1992)

[3] Alur, R., Grosu, R., Lee, I., Sokolsky, O.: Compositional refinement for hierarchical hybrid systems. In: Di Benedetto, Maria Domenica, Sangiovanni-Vincentelli, A.L. (eds.) Hybrid Systems: Computation and Control, 4th International Workshop, Proceedings. Lecture Notes in Computer Science, vol. 2034, pp. 33–48. Springer (2001), http://dx.doi.org/10.1007/3-540-45351-2_7

[4] Bauer, S.S., David, A., Hennicker, R., Larsen, K.G., Legay, A., Nyman, U., Wasowski, A.: Moving from specifications to contracts in component-based design. In: de Lara, J., Zisman, A. (eds.) Fundamental Approaches to Software Engineering (FASE), Mar. 24 - Apr. 1, 2012. Proc. LNCS, vol. 7212, pp. 43–58. Springer (2012)

[5] Benvenuti, L., Bresolin, D., Collins, P., Ferrari, A., Geretti, L., Villa, T.: Assume-guarantee verification of nonlinear hybrid systems with Ariadne. Int. Journal of Robust and Nonlinear Control 24(4), 699–724 (2014)

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

[7] Felty, A.P., Middeldorp, A. (eds.): Automated Deduction - CADE-25 - 25th Int. Conf. on Autom. Deduction, Aug. 1-7, 2015, Proc., LNCS, vol. 9195. Springer (2015)

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

[9] Fulton, N., Mitsch, S., Quesel, J., Völp, M., Platzer, A.: KeYmaera X: an axiomatic tactical theorem prover for hybrid systems. In: Felty and Middeldorp [7], pp. 527–538

[10] Graf, S., Passerone, R., Quinton, S.: Contract-based reasoning for component systems with rich interactions. In: Sangiovanni-Vincentelli, A., Zeng, H., Di Natale, M., Marwedel, P. (eds.) Embedded Sys. Dev., vol. 20, pp. 139–154. Springer (2014)

[11] Henzinger, T.A., Minea, M., Prabhu, V.S.: Assume-guarantee reasoning for hierarchical hybrid systems. In: Benedetto, M.D.D., Sangiovanni-Vincentelli, A.L. (eds.) Hybrid Systems:

Computation and Control, 4th Int. Workshop, HSCC 2001, Mar. 28-30, 2001, Proc. LNCS, vol. 2034, pp. 275–290. Springer (2001)

[12] Kurki-Suonio, R.: Component and interface refinement in closed-system specifications. In: Wing, J.M., Woodcock, J., Davies, J. (eds.) FM'99 - Formal Methods, Sept. 20-24, 1999, Proc. LNCS, vol. 1708, pp. 134–154. Springer (1999)

[13] Larsen, K.G., Pettersson, P., Yi, W.: UPPAAL in a nutshell. STTT 1(1-2), 134–152 (1997)

[14] Loos, S.M., Platzer, A.: Differential refinement logic. In: LICS. ACM (2016)

[15] Loos, S.M., Platzer, A., Nistor, L.: Adaptive cruise control: Hybrid, distributed, and now formally verified. In: Butler, M., Schulte, W. (eds.) FM'11 - Formal Methods. LNCS, vol. 6664, pp. 42–56. Springer (2011)

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

[17] Madl, G., Abdelwahed, S., Karsai, G.: Automatic verification of component-based real-time CORBA applications. In: Proc. of the 25th IEEE Real-Time Systems Symp. (RTSS), 5-8 Dec. 2004. pp. 231–240. IEEE Computer Society (2004)

[18] 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, Technische Universität Berlin, June 24-28, 2013 (2013)

[19] Mitsch, S., Loos, S.M., Platzer, A.: Towards formal verification of freeway traffic control. In: ICCPS. pp. 171–180. IEEE/ACM (2012)

[20] Mitsch, S., Quesel, J.D., Platzer, A.: Refactoring, refinement, and reasoning - a logical characterization for hybrid systems. In: Jones, C.B., Pihlajasaari, P., Sun, J. (eds.) FM 2014: Formal Methods - 19th International Symposium, Singapore, May 12-16, 2014. Proceedings. LNCS, vol. 8442, pp. 481–496. Springer (2014), http://dx.doi.org/10.1007/978-3-319-06410-9_33

[21] Müller, A., Mitsch, S., Platzer, A.: Verified traffic networks: Component-based verification of cyber-physical flow systems. In: 18th IEEE Intelligent Transportation Systems Conf. (ITSC). pp. 757–764. IEEE (2015)

[22] 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. Lecture Notes in Computer Science, vol. 9681, pp. 441–456. Springer (2016), http://dx.doi.org/10.1007/978-3-319-33693-0_28

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

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

[25] Platzer, A.: The complete proof theory of hybrid systems. In: Proc. of the 27th Annual IEEE Symp. on Logic in Computer Science, LICS 2012, June 25-28, 2012. pp. 541–550. IEEE Computer Society (2012)

[26] Platzer, A.: A uniform substitution calculus for differential dynamic logic. In: Felty and Middeldorp [7], pp. 467–481

[27] 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. LNCS, vol. 2791, pp. 151–165. Springer (2003)

[28] Ringert, J.O., Rumpe, B., Wortmann, A.: From software architecture structure and behavior modeling to implementations of cyber-physical systems. In: Wagner, S., Lichter, H. (eds.) Software Engineering 2013 - Workshopband, 26. Feb. - 1. Mar. 2013. LNI, vol. 215, pp. 155–170. GI (2013)

[29] Ruchkin, I., Schmerl, B.R., Garlan, D.: Architectural abstractions for hybrid programs. In: Kruchten, P., Becker, S., Schneider, J. (eds.) Proc. of the 18th Int. ACM SIGSOFT Symp. on Component-Based Software Engineering, CBSE 2015, May 4-8, 2015. pp. 65–74. ACM (2015)

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

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

$$([:=]) \quad \frac{\phi^{\theta_1 \ldots \theta_n}_{x_1 \ldots x_n}}{[x_1 := \theta_1, \ldots, x_n := \theta_n]\phi} \qquad (\wedge\text{r}) \quad \frac{\Gamma \vdash \phi, \Delta \quad \Gamma \vdash \psi, \Delta}{\Gamma \vdash \phi \wedge \psi, \Delta}$$

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

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

$$(\text{Wl}) \quad \frac{\Gamma \vdash \Delta}{\Gamma, \phi \vdash \Delta} \qquad (\forall\text{r}) \quad \frac{\Gamma \vdash \phi(s(X_1, \ldots, X_n)), \Delta}{\Gamma \vdash \forall x\, \phi(x), \Delta}\ {}_1$$

$$(cut) \quad \frac{\Gamma \vdash \phi, \Delta \;\; \Gamma, \phi \vdash \Delta}{\Gamma \vdash \Delta} \qquad ([]gen) \quad \frac{\Gamma \vdash [\alpha]\phi, \Delta \quad \phi \vdash \psi}{\Gamma \vdash [\alpha]\psi, \Delta}$$

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

$$([]\ \text{M}) \quad \frac{\phi \vdash \psi}{[\alpha]\phi \vdash [\alpha]\psi} \qquad (DI) \quad \frac{\Gamma, H \vdash F, \Delta \quad \vdash (H \rightarrow F'^{\theta_1 \ldots \theta_n}_{x'_1 \ldots x'_n})}{\Gamma \vdash [x'_1 = \theta_1, \ldots, x'_n = \theta_n \,\&\, H]F, \Delta}$$

$$(\text{CE}) \quad \frac{p(\bar{x}) \leftrightarrow q(\bar{x})}{C(p(\bar{x})) \leftrightarrow C(q(\bar{x}))}$$

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

Figure 7: Proof Rules

A Proofs

Throughout the proofs we will use the proof rules listed in Fig. 7.

A.1 Proof of Theorem 1 – Composition Retains Contracts

The proof for Theorem 1 follows the proof sketch below. 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.

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

2. drop plant behavior that is irrelevant for the current contract (Lemma 2)

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

We first introduce lemmas for these steps before proceeding with a proof of Theorem 1. Lemma 1 allows us to drop irrelevant control parts from a dL formula.

Lemma 1 (Drop Program)**.** *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*

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

are valid.

Lemma 1. We first show that $[\alpha]A \rightarrow [\beta][\alpha]A$. This follows immediately from the V-axiom of dL (i. e., $\phi \rightarrow [\gamma]\phi$, if $FV(\phi) \cap BV(\gamma) = \emptyset$) with $\phi = [\alpha]A$ and $\gamma = \beta$, since we know that $FV(A) \cap BV(\beta) = \emptyset$ and $FV(\alpha) \cap BV(\beta) = \emptyset$.

$$\dfrac{\dfrac{\dfrac{*}{[\alpha]A \vdash [\alpha]A}}{[\alpha]A \vdash [\beta][\alpha]A}\,\text{V (since } FV([\alpha]A) \cap BV(\beta) = \emptyset)}{\vdash [\alpha]A \rightarrow [\beta][\alpha]A}\,\rightarrow\text{r}$$

The proof of $[\alpha]A \rightarrow [\alpha][\beta]A$ uses monotonicity and the V axiom.

$$\dfrac{\dfrac{\dfrac{\dfrac{*}{A \vdash A}}{A \vdash [\beta]A}\,\text{V (since } FV(A) \cap BV(\beta) = \emptyset)}{[\alpha]A \vdash [\alpha][\beta]A}\,[]\,\text{M}}{\vdash [\alpha]A \rightarrow [\alpha][\beta]A}\,\rightarrow\text{r}$$

□

Lemma 2 allows us to drop irrelevant parts from the plant of a hybrid program.

Lemma 2 (Drop Plant)**.** *Let $\theta_1(x_1)$ and $\theta_2(x_2)$ be terms possibly containing x_1 (or x_2, respectively), where x_1 and x_2 are vectors with $V(x_1) \cap V(x_2) = \emptyset$. 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 (or x_2, respectively). Then*

$$[\{t' = 1, x_1' = \theta_1(x_1) \& H_1(x_1)\}]A(x_1) \rightarrow$$
$$[\{t' = 1, x_1' = \theta_1(x_1), x_2' = \theta_2(x_2) \& H_1(x_1) \wedge H_2(x_2)\}]A(x_1)$$

is valid.

Lemma 2. The proof uses one side of the differential ghost axiom (DG) and differential cut (DC). First we use DC in the unusual inverse direction to get rid of $H_2(x_2)$. The precondition for the differential cut,

$$[\{t' = 1, x_1' = \theta_1(x_1), x_2' = \theta_2(x_2) \& H_1(x_1) \wedge H_2(x_2)\}] H_2(x_2)$$

follows immediately, since an evolution domain always holds, i. e., we know that

$$[\{t' = 1, x_1' = \theta_1(x_1), x_2' = \theta_2(x_2) \& H_1(x_1) \wedge H_2(x_2)\}] A(x_1)$$

and thus, $H_2(x_2)$ obviously holds after the continuous evolution.

Next we add a universal quantifier using $\forall$-eliminate[6]—$(\forall x \,.\, p(\bar{x})) \rightarrow p(\bar{x})$—and then we apply DG. Note, that DG is phrased in terms of an existentially quantified variable y, which occurs in a linear ordinary differential equation (ODE). However, in the direction that we need (see step DG), the axiom is stronger, i. e., $[x' = f(x) \& q(x)] p(x) \rightarrow \forall y [x' = f(x), y' = \eta \& q(x)] p(x)$, and uses a universal quantifier and an arbitrary ODE [26].

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{*}{[\{t' = 1, x_1' = \theta_1(x_1) \& H_1(x_1)\}] A(x_1) \vdash [\{t' = 1, x_1' = \theta_1(x_1) \& H_1(x_1)\}] A(x_1)}}{[\{t' = 1, x_1' = \theta_1(x_1) \& H_1(x_1)\}] A(x_1) \vdash \forall x_2 \,.\, [\{t' = 1, x_1' = \theta_1(x_1), x_2' = \theta_2(x_2) \& H_1(x_1)\}] A(x_1)}\,\text{DG}}{[\{t' = 1, x_1' = \theta_1(x_1) \& H_1(x_1)\}] A(x_1) \vdash [\{t' = 1, x_1' = \theta_1(x_1), x_2' = \theta_2(x_2) \& H_1(x_1)\}] A(x_1)}\,\forall i}{[\{t' = 1, x_1' = \theta_1(x_1) \& H_1(x_1)\}] A(x_1) \vdash [\{t' = 1, x_1' = \theta_1(x_1), x_2' = \theta_2(x_2) \& H_1(x_1) \wedge H_2(x_2)\}] A(x_1)}\,\text{DC}$$

□

The next lemma states that instead of proving a safety property about a deterministic assignment, we can replace it with a proof for a non-deterministic assignment (cf. [14]).

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.

Lemma 3. For the proof, we expand the definition of non-deterministic assignments and use universal instantiation (i. e., if the formula is true for all x it is also true for θ).

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{*}{A(\theta) \vdash A(\theta)}}{\forall x \,.\, A(x) \vdash A(\theta)}\,\forall \text{l}}{\forall x \,.\, A(x) \vdash [x := \theta] A(x)}\,[:=]}{[x := *] A(x) \vdash [x := \theta] A(x)}\,[:*]}{\vdash [x := *] A(x) \rightarrow [x := \theta] A(x)}\,\rightarrow \text{r}$$

□

[6] $\bar{x}$ is the vector of all relevant variables, cf. [26].

The next lemma allows us to reorder assignments, which do not depend on each other.

Lemma 4 (Reorder Assignment)**.** *Let $A(x,y)$ be an arbitrary dL formula and let x, y be different variables and $F(x), G(y)$ be predicates over x, respectively y. Furthermore, let $y \notin V(F(x))$ and $x \notin V(G(y))$. Then*

$$[x := *; ?F(x); y := *; ?G(y)]A(x,y) \leftrightarrow [y := *; ?G(y); x := *; ?F(x)]A(x,y)$$

is valid.

Lemma 4. The proof follows from the definition of non-deterministic assignments and the axiom for tests. The step *all distribute* follows, as we know that $y \notin V(F(x))$ and $x \notin V(G(y))$. The step *impl* uses the equivalence $\big(a \to (b \to c)\big) \leftrightarrow \big(b \to (a \to c)\big)$. We have to show both directions of the equivalence.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\forall x\,.\,\forall y\,.\,(F(x) \to (G(y) \to A(x,y))) \vdash \forall x\,.\,\forall y\,.\,(F(x) \to (G(y) \to A(x,y)))}\,*}{\forall x\,.\,\forall y\,.\,(F(x) \to (G(y) \to A(x,y))) \vdash \forall x\,.\,\forall y\,.\,(G(y) \to (F(x) \to A(x,y)))}\,\text{impl}}{\forall x\,.\,(F(x) \to \forall y\,.\,(G(y) \to A(x,y))) \vdash \forall y\,.\,(G(y) \to \forall x\,(F(x) \to A(x,y)))}\,\text{all distr}}{[x := *][?F(x)][y := *][?G(y)]A(x,y) \vdash [y := *][?G(y)][x := *][?F(x)]A(x,y)}\,[:*],[:=],[?]}{[x := *; ?F(x); y := *; ?G(y)]A(x,y) \vdash [y := *; ?G(y); x := *; ?F(x)]A(x,y)}\,[;]$$

The second direction works accordingly. □

The next lemma allows the addition of tests, which are known to be true.

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

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

Lemma 5. For the proof, we first apply proof rules for implication, before we split the equivalence into two implications, which we verify one after the other.

$$\dfrac{\dfrac{\dfrac{\dots ① \qquad \dots ②}{[\alpha]F \vdash ([\alpha; ?F]A \to [\alpha]A) \wedge ([\alpha]A \to [\alpha; ?F]A)}\,\wedge\text{r}}{[\alpha]F \vdash [\alpha; ?F]A \leftrightarrow [\alpha]A}\,\text{equiv}}{\vdash [\alpha]F \to ([\alpha; ?F]A \leftrightarrow [\alpha]A)}\,\to\text{r}$$

To prove ① we use the *K-axiom* and modus ponens (MP).

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{F, F \to A \vdash A}\,*}{F \vdash (F \to A) \to A}\,\text{MP}}{[\alpha]F \vdash [\alpha]\,((F \to A) \to A)}\,\to\text{r}}{[\alpha]F \vdash [\alpha]\,(F \to A) \to [\alpha]A}\,[]\,\text{M}}{[\alpha]F \vdash [\alpha][?F]A \to [\alpha]A}\,\text{K}}{[\alpha]F \vdash [\alpha; ?F]A \to [\alpha]A}\,[?]$$

① continued

Similarly, for ②. Since $(A \to (F \to A)) \equiv true$, this equivalence also holds in the context of $[\alpha]$ (cf. CE).

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{*}{F \vdash true}}{[\alpha]F \vdash [\alpha]true}\,[]\,\text{M}}{[\alpha]F \vdash [\alpha]\,(A \to (F \to A))}\,\text{CE}}{[\alpha]F \vdash [\alpha]A \to [\alpha]\,(F \to A)}\,\text{K}}{[\alpha]F \vdash [\alpha]A \to [\alpha][?F]A}\,[?]}{[\alpha]F \vdash [\alpha]A \to [\alpha;?F]A}\,[;]}{\text{② continued}}$$

□

The final lemma allows us to weaken a test if we know that the old test implies the new one.

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

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

Lemma 6. The proof uses transitivity of logical implication.

$$\dfrac{\dfrac{\dfrac{\dfrac{*}{G \to A, F \to G \vdash F \to A}\,\text{trans}}{[?G]A, F \to G \vdash [?F]A}\,[?]}{([?G]A) \land (F \to G) \vdash [?F]A}\,\land\text{l}}{\vdash (([?G]A) \land (F \to G)) \to [?F]A}\,\to\text{r}$$

□

Corollary 1 is a variation of Lemma 6, including an assignment before the tests.

Corollary 1 (Weaken Test – Assignment)**.** *Let A be an arbitrary dℒ formula, x and y variables and $F(x)$ and $G(y)$ be formulas. Then*

$$(([y := x][?G(y)]A(x,y)) \land ([y := x]\,(F(x) \to G(y)))) \to [y := x][?F(x)]A(x,y)$$

Corollary 1. After applying the assignments we can again use Lemma 6.

$$\dfrac{\dfrac{\dfrac{*}{[?G(x)]A(x,x), (F(x) \to G(x)) \vdash [?F(x)]A(x,x)}\,\text{Lemma 6}}{[y := x][?G(y)]A(x,y), [y := x]\,(F(x) \to G(y)) \vdash [y := x][?F(x)]A(x,y)}\,[:=]}{([y := x][?G(y)]A(x,y)) \land ([y := x]\,(F(x) \to G(y))) \vdash [y := x][?F(x)]A(x,y)}\,\land\text{l}$$

□

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/proof rules K, G, and CE [26] to unpeel the context and use the lemma top level.

Now we finally have all we need to prove that two safe components, which communicate using ports, result in another safe component upon composition.

Theorem 1 (Composition Retains Contracts)**.** *Let C_1 and C_2 be components with admissible interfaces I_1 and I_2 that are* contract compliant *(i. e., their contracts are valid)*

$$\models t = 0 \land \phi_1 \rightarrow [(\text{in}_1; \text{ctrl}_1; \{t' = 1, \text{plant}_1\})^*]\,(\psi_1) \text{ and} \tag{9}$$
$$\models t = 0 \land \phi_2 \rightarrow [(\text{in}_2; \text{ctrl}_2; \{t' = 1, \text{plant}_2\})^*]\,(\psi_2) \tag{10}$$

and compatible *with respect to $\mathcal{X}$ (i. e., compatibility proof obligations are valid)*

$$\text{for all input ports } v \in \mathcal{I}^{\mathcal{X}} \cap V_2^{\text{in}} \;:\; \models [v := \mathcal{X}(v)]\left(\pi_1^{\text{out}}(\mathcal{X}(v)) \rightarrow \pi_2^{\text{in}}(v)\right) \text{ and} \tag{11}$$
$$\text{for all input ports } v \in \mathcal{I}^{\mathcal{X}} \cap V_1^{\text{in}} \;:\; \models [v := \mathcal{X}(v)]\left(\pi_2^{\text{out}}(\mathcal{X}(v)) \rightarrow \pi_1^{\text{in}}(v)\right) \;. \tag{12}$$

Then, the parallel composition $C_3, I_3 = ((C_1, I_1) \| (C_2, I_2))_{\mathcal{X}}$ satisfies the contract

$$\models t = 0 \land (\phi_1 \land \phi_2) \rightarrow [(\text{in}_3; \text{ctrl}_3; \{t' = 1, \text{plant}_3\})^*](\psi_1 \land \psi_2) \tag{13}$$

with in_3*,* ctrl_3*, and* plant_3 *according to Def. 5.*

Proof of Theorem 1. For space reasons let

$$\begin{aligned}
(C_3, I_3) &\overset{\text{def}}{\equiv} ((C_1, I_1) \| (C_2, I_2))_{\mathcal{X}} \\
\mathit{ctrl}_3 &\overset{\text{def}}{\equiv} (\mathit{ports}_1; \mathit{ctrl}_1; \mathit{ports}_2; \mathit{ctrl}_2) \cup (\mathit{ports}_2; \mathit{ctrl}_2; \mathit{ports}_1; \mathit{ctrl}_1) \\
\mathit{plant}_3 &\overset{\text{def}}{\equiv} \mathit{plant}_1, \mathit{plant}_2 \\
\phi_3 &\overset{\text{def}}{\equiv} \phi_1 \land \phi_2 \\
\psi_3^{\mathit{safe}} &\overset{\text{def}}{\equiv} \psi_1^{\mathit{safe}} \land \psi_2^{\mathit{safe}} \\
\pi_3^{\mathit{out}} &\overset{\text{def}}{\equiv} \left(\bigwedge_{v \in \mathrm{V}^{\mathit{out}}} \pi_1^{\mathit{out}}(v)\right) \land \left(\bigwedge_{v \in \mathrm{V}^{\mathit{out}}} \pi_2^{\mathit{out}}(v)\right)
\end{aligned}$$

We know that

$$\mathit{Cont}(C_1, I_1) \overset{\text{Def. 4}}{\equiv} t = 0 \land \phi_1 \to [(\mathit{in}_1; \mathit{ctrl}_1; (t' = 1, \mathit{plant}_1))^*] \left(\psi_1^{\mathit{safe}} \land \bigwedge_{v \in \mathrm{V}_1^{\mathit{out}}} \pi_1^{\mathit{out}}(v) \right)$$

$$\mathit{Cont}(C_2, I_2) \overset{\text{Def. 4}}{\equiv} t = 0 \land \phi_2 \to [(\mathit{in}_2; \mathit{ctrl}_2; (t' = 1, \mathit{plant}_2))^*] \left(\psi_2^{\mathit{safe}} \land \bigwedge_{v \in \mathrm{V}_2^{\mathit{out}}} \pi_2^{\mathit{out}}(v) \right)$$

$$\mathit{Cont}(C_3, I_3) \overset{\text{Def. 4}}{\equiv} t = 0 \land \phi_3 \to [(\mathit{in}_3; \mathit{ctrl}_3; (t' = 1, \mathit{plant}_3))^*] \left(\psi_3^{\mathit{safe}} \land \pi_3^{\mathit{out}} \right)$$

$$\mathit{CPO}(I_1) \overset{\text{Def. 6}}{\equiv} [v := \mathcal{X}(v)] \left(\pi_1^{\mathit{out}}(\mathcal{X}(v)) \to \pi_2^{\mathit{in}}(v) \right) \text{ and}$$

$$\mathit{CPO}(I_2) \overset{\text{Def. 6}}{\equiv} [v := \mathcal{X}(v)] \left(\pi_2^{\mathit{out}}(\mathcal{X}(v)) \to \pi_1^{\mathit{in}}(v) \right) \text{ for all } v \in \mathcal{I}^{\mathcal{X}} \cap \mathrm{V}_{1,2}^{\mathit{in}} \ .$$

We have to show that the contract of the parallel composition $\mathit{Cont}(C_3, I_3)$ is valid, i. e.,

$$\phi_3 \to \left[(\mathit{in}_3; \mathit{ctrl}_3; \{t' = 1, \mathit{plant}_3\})^* \right] \psi_3 \tag{34}$$

Assume that formulas (9) and (10) are valid, hence there exist invariants φ_1 and φ_2 [25] (if (9) and (10) were verified using loop induction, the invariants are even known), such that:

$$t = 0 \land \phi_1 \to \varphi_1 \qquad t = 0 \land \phi_2 \to \varphi_2 \tag{35}$$

$$\varphi_1 \to [\mathit{in}_1; \mathit{ctrl}_1; \{t' = 1, \mathit{plant}_1\}]\varphi_1 \qquad \varphi_2 \to [\mathit{in}_2; \mathit{ctrl}_2; \{t' = 1, \mathit{plant}_2\}]\varphi_2 \tag{36}$$

$$\varphi_1 \to \left(\psi_1^{\mathit{safe}} \land \bigwedge_{v \in \mathrm{V}^{\mathit{out}}} \pi_1^{\mathit{out}}(v) \right) \qquad \varphi_2 \to \left(\psi_2^{\mathit{safe}} \land \bigwedge_{v \in \mathrm{V}^{\mathit{out}}} \pi_2^{\mathit{out}}(v) \right) \tag{37}$$

By *imply-right* and *loop induction*, where we choose $\varphi_3 = \varphi_1 \land \varphi_2$ (i. e., the loop invariant for the proof is the conjunction of the two invariants known to exist from the independent proofs), we get

$$\dfrac{\dfrac{\dfrac{\dots ①}{t = 0 \land \phi_3 \vdash \varphi_3} \quad \dfrac{\dots ②}{\varphi_3 \vdash [\mathit{in}_3; \mathit{ctrl}_3; \{t' = 1, \mathit{plant}_3\}]\varphi_3} \quad \dfrac{\dots ③}{\varphi_3 \vdash \left(\psi_3^{\mathit{safe}} \land \Pi_3^{\mathit{out}} \right)}}{t = 0 \land \phi_3 \vdash [(\mathit{in}_3; \mathit{ctrl}_3; \{t' = 1, \mathit{plant}_3\})^*] \left(\psi_3^{\mathit{safe}} \land \Pi_3^{\mathit{out}} \right)}\, ind}{\vdash t = 0 \land \phi_3 \to [(\mathit{in}_3; \mathit{ctrl}_3; \{t' = 1, \mathit{plant}_3\})^*] \left(\psi_3^{\mathit{safe}} \land \Pi_3^{\mathit{out}} \right)}\, \to\text{r}$$

We will transform the three resulting branches until we get formulas that correspond to (35), (36) and (37). To prove the induction base case and the use case, we use the loop invariants φ_1 and φ_2, for which (35) and (37) hold.

$$\dfrac{\dfrac{\dfrac{\dfrac{*}{(34)\ t = 0 \land \phi_1 \vdash \varphi_1}}{\text{Wl}\ t = 0 \land \phi_1, \phi_2 \vdash \varphi_1}}{\land\text{l}\ t = 0 \land \phi_1 \land \phi_2 \vdash \varphi_1} \quad \dfrac{\dfrac{\dfrac{*}{(34)\ t = 0 \land \phi_2 \vdash \varphi_2}}{\text{Wl}\ t = 0 \land \phi_1, \phi_2 \vdash \varphi_2}}{\land\text{l}\ t = 0 \land \phi_1 \land \phi_2 \vdash \varphi_2}}{\land\text{r}\ t = 0 \land \phi_1 \land \phi_2 \vdash \varphi_1 \land \varphi_2}$$

$$\text{def}\ \dfrac{}{①\ \text{continued}}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{*}{\varphi_1 \vdash \psi_1^{safe} \wedge \bigwedge_{v\in \mathrm{V}_1^{out}} \pi^{out}(v)}(36)}{\varphi_1, \varphi_2 \vdash \psi_1^{safe} \wedge \bigwedge_{v\in \mathrm{V}_1^{out}} \pi^{out}(v)}\mathrm{Wl} \quad \dfrac{\dfrac{*}{\varphi_2 \vdash \psi_2^{safe} \wedge \bigwedge_{v\in \mathrm{V}_2^{out}} \pi^{out}(v)}(36)}{\varphi_1, \varphi_2 \vdash \psi_2^{safe} \wedge \bigwedge_{v\in \mathrm{V}_2^{out}} \pi^{out}(v)}\mathrm{Wl}}{\varphi_1, \varphi_2 \vdash \left(\psi_1^{safe} \wedge \bigwedge_{v\in \mathrm{V}_1^{out}} \pi^{out}(v)\right) \wedge \left(\psi_2^{safe} \wedge \bigwedge_{v\in \mathrm{V}_2^{out}} \pi^{out}(v)\right)}\wedge\mathrm{r}}{\varphi_1 \wedge \varphi_2 \vdash \left(\psi_1^{safe} \wedge \bigwedge_{v\in \mathrm{V}_1^{out}} \pi^{out}(v)\right) \wedge \left(\psi_2^{safe} \wedge \bigwedge_{v\in \mathrm{V}_2^{out}} \pi^{out}(v)\right)}\wedge\mathrm{l}}{\text{③ continued}}\mathrm{def}$$

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

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dots④}{\varphi_1, \varphi_2 \vdash [in_3; ctrl_3][\{t' = 1, plant_3\}]\varphi_1} \quad \dots⑤}{\varphi_1, \varphi_2 \vdash [in_3; ctrl_3][\{t' = 1, plant_3\}]\,(\varphi_1 \wedge \varphi_2)}[]\wedge,[;]}{\varphi_1, \varphi_2 \vdash [in_3; ctrl_3; \{t' = 1, plant_3\}]\,(\varphi_1 \wedge \varphi_2)}[;]}{\varphi_1 \wedge \varphi_2 \vdash [in_3; ctrl_3; \{t' = 1, plant_3\}]\,(\varphi_1 \wedge \varphi_2)}\wedge\mathrm{l}}{\varphi_3 \vdash [in_3; ctrl_3; \{t' = 1, plant_3\}]\,\varphi_3}\mathrm{def}}{\text{② continued}}$$

We have to prove that both, φ_1 (i. e., ④) and φ_2 (i. e., ⑤) hold. We illustrate the strategy only for branch ④, because branch ⑤ follows in a similar manner.

First we apply Lemma 2, which is possible because we limit the scopes of our plant to the variables of φ_1, which is permitted as the state of all other variables does not influence the truth value of our formula (by coincidence lemma, cf. [26, Lemma 2]). Then we apply the proof rule for non-deterministic choice and get two branches.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dots⑥}{\varphi_1, \varphi_2 \vdash [in_3; ports_1; ctrl_1; ports_2; ctrl_2][\{t' = 1, plant_1\}]\varphi_1} \quad \dots⑦}{\varphi_1, \varphi_2 \vdash [in_3; ((ports_1; ctrl_1; ports_2; ctrl_2) \cup \dots)][\{t' = 1, plant_1\}]\varphi_1}[\cup],\wedge\mathrm{r}}{\varphi_1, \varphi_2 \vdash [in_3; ctrl_3][\{t' = 1, plant_1\}]\varphi_1}\mathrm{def}}{\varphi_1, \varphi_2 \vdash [in_3; ctrl_3][\{t' = 1, plant_1, plant_2\}]\varphi_1}\text{Lemma 2}}{\varphi_1, \varphi_2 \vdash [in_3; ctrl_3][\{t' = 1, plant_3\}]\varphi_1}\mathrm{def}}{\text{④ continued}}$$

We will now transform ⑥ until we get (36). 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 (36). The detailed proof steps are explained below and can be cross-referenced

using enumeration and the step number in the sequent proof.

	$*$
(35)	$\varphi_1 \vdash [\mathit{in}_1][\mathit{ctrl}_1][\{t' = 1, \mathit{plant}_1\}]\varphi_1$
Wl	$\varphi_1, \varphi_2 \vdash [\mathit{in}_1][\mathit{ctrl}_1][\{t' = 1, \mathit{plant}_1\}]\varphi_1$
cf. 8.,Lemma 4,def	$\varphi_1, \varphi_2 \vdash [\mathit{in}_1^\dagger; (v_j := *; ?\pi^{\mathit{in}}(v_j))][\mathit{ctrl}_1][\{t' = 1, \mathit{plant}_1\}]\varphi_1$
cf. 7.,Lemma 3	$\varphi_1, \varphi_2 \vdash [\mathit{in}_1^\dagger; (v_j := \mathcal{X}(v_j); ?\pi^{\mathit{in}}(v_j))][\mathit{ctrl}_1][\{t' = 1, \mathit{plant}_1\}]\varphi_1$
cf. 6.,Cor. 1	$\varphi_1, \varphi_2 \vdash [\mathit{in}_1^\dagger; (v_j := \mathcal{X}(v_j); ?\pi^{\mathit{out}}(\mathcal{X}(v_j)))][\mathit{ctrl}_1][\{t' = 1, \mathit{plant}_1\}]\varphi_1$
cf. 5.,Lemma 6	$\varphi_1, \varphi_2 \vdash [\mathit{in}_1^\dagger; (v_j := \mathcal{X}(v_j); ?\varphi_2)][\mathit{ctrl}_1][\{t' = 1, \mathit{plant}_1\}]\varphi_1 \quad \cdots ⑧$
cf. 4.,Lemma 5	$\varphi_1, \varphi_2 \vdash [\mathit{in}_1^\dagger; (v_j := \mathcal{X}(v_j))][\mathit{ctrl}_1][\{t' = 1, \mathit{plant}_1\}]\varphi_1$
def	$\varphi_1, \varphi_2 \vdash [\mathit{in}_1^\dagger][\mathit{ports}_1][\mathit{ctrl}_1][\{t' = 1, \mathit{plant}_1\}]\varphi_1$
cf. 3.,Lemma 1	$\varphi_1, \varphi_2 \vdash [\mathit{in}_1^\dagger][\mathit{in}_2^\dagger][\mathit{ports}_1][\mathit{ctrl}_1][\{t' = 1, \mathit{plant}_1\}]\varphi_1$
[;]	$\varphi_1, \varphi_2 \vdash [\mathit{in}_1^\dagger; \mathit{in}_2^\dagger][\mathit{ports}_1][\mathit{ctrl}_1][\{t' = 1, \mathit{plant}_1\}]\varphi_1$
cf. 2.,Lemma 4	$\varphi_1, \varphi_2 \vdash [\mathit{in}_3][\mathit{ports}_1][\mathit{ctrl}_1][\{t' = 1, \mathit{plant}_1\}]\varphi_1$
cf. 1.,Lemma 1	$\varphi_1, \varphi_2 \vdash [\mathit{in}_3][\mathit{ports}_1][\mathit{ctrl}_1][\mathit{ports}_2; \mathit{ctrl}_2][\{t' = 1, \mathit{plant}_1\}]\varphi_1$
[;]	$\varphi_1, \varphi_2 \vdash [\mathit{in}_3; \mathit{ports}_1; \mathit{ctrl}_1; \mathit{ports}_2; \mathit{ctrl}_2][\{t' = 1, \mathit{plant}_1\}]\varphi_1$

⑥ continued

In detail, we applied the following lemmas:

1. We apply Lemma 1 to get rid of $\mathit{ports}_2; \mathit{ctrl}_2$, with $\alpha \overset{\text{def}}{\equiv} \mathit{ctrl}_1$, $\beta \overset{\text{def}}{\equiv} \mathit{ports}_2; \mathit{ctrl}_2$ and $A \overset{\text{def}}{\equiv} [\{t' = 1, \mathit{plant}_1\}]\varphi_1$, since $BV(\mathit{ports}_2) \subseteq \mathrm{V}_2^{\mathit{in}}$, $BV(\mathit{ctrl}_2) \subseteq V_2 \setminus \left(\mathrm{V}_2^{\mathit{global}} \cup \mathrm{V}_2^{\mathit{in}} \cup \{t\}\right)$ (and thus $BV(\mathit{ports}_2) \cup BV(\mathit{ctrl}_2) \subseteq V_2 \setminus \left(\mathrm{V}^{\mathit{global}} \cup \{t\}\right)$), $FV(\mathit{ctrl}_1) \subseteq V_1$, $FV(\varphi_1) \subseteq V_1$ and $V_1 \cap V_2 \setminus \left(\mathrm{V}^{\mathit{global}} \cup \{t\}\right) = \emptyset$, and thus $FV(\alpha) \cap BV(\beta) = \emptyset$ and $FV(A) \cap BV(\beta) = \emptyset$.

2. We use Lemma 4 to reorder the assignments in in_3 in a way that the assignments of C_1 precede the ones of C_2. Note, that these assignments are only the non-connected ports of C_1 and C_2, while the connected ports are still in ports_1. Hence, we use $\mathit{in}_1^\dagger$ and $\mathit{in}_2^\dagger$ to denote that these assignments are not the full in_1 and in_2.

3. Using Lemma 1 we can remove the assignments from component C_2, with $\alpha \overset{\text{def}}{\equiv} \mathit{in}_1^\dagger$, $\beta \overset{\text{def}}{\equiv} \mathit{in}_2^\dagger$ and $A \overset{\text{def}}{\equiv} [\mathit{ports}_1][\mathit{ctrl}_1][\{t' = 1, \mathit{plant}_1\}]\varphi_1$.

4. By multiple applications of Lemma 5 (i. e., once for each connected port v_j), starting from the rightmost deterministic assignment, we can insert tests for φ_2 after each deterministic assignment in ports_1 without changing the behavior (side conditions verified in ⑧).

5. We can then relax all the tests for φ_2 using Lemma 6 (i. e., once for each connected port v_j) since $\varphi_2 \to \pi_2^{\mathit{out}}(\mathcal{X}(v_j))$ (cf. (37)).

6. We then relax the tests to $\pi_{v_j}^{\mathit{in}}$ since we know $[v_j := \mathcal{X}(v_j)]\pi_2^{\mathit{out}}(\mathcal{X}(v_j)) \to \pi^{\mathit{in}}(v_j)$ (cf. (9)–(10)), and because all these test are preceded by deterministic assignments $v_j := \mathcal{X}(v_j)$ in ports_1 (cf. Corollary 1).

7. By multiple applications of Lemma 3 we can then replace the deterministic port assignments with non-deterministic assignments (i. e., once for each connected port v_j).

8. Finally, by multiple applications of Lemma 4 we change the order of the assignments v_j and the assignments in $in_1^\dagger$ and transform them until we get in_1.

It remains to prove the side condition for Lemma 5, with $\alpha \overset{\text{def}}{\equiv} in_1^\dagger$ and $F \overset{\text{def}}{\equiv} \varphi_2$ (cf. ⑧).

$$\text{Lemma 1}\dfrac{\dfrac{*}{\varphi_2 \vdash \varphi_2}}{\varphi_2 \vdash [in_1^\dagger]\varphi_2}$$

⑧: Side Condition, Lemma 5

The proof for ⑦ works similarly. We first split in_3, apply Lemma 5, as the resulting side condition still holds (cf. ⑨) and apply Lemma 3 before Lemma 1, so that its preconditions are fulfilled. With $\alpha \overset{\text{def}}{\equiv} \big(v_j := *; ?\pi^{in}(v_j); \ldots\big); ctrl_1$, $\beta \overset{\text{def}}{\equiv} ports_2; ctrl_2$ and $A \overset{\text{def}}{\equiv} \varphi_j$, we can apply Lemma 1, since $BV(ports_2) \subseteq \mathrm{V}_2^{in}$, $BV(ctrl_2) \subseteq V_2 \setminus \Big(\mathrm{V}_2^{global} \cup \mathrm{V}_2^{in} \cup \{t\}\Big)$ (which furthermore means that $BV(ports_2) \cup BV(ctrl_2) \subseteq V_2 \setminus \big(\mathrm{V}^{global} \cup \{t\}\big)$), $FV(ctrl_1) \subseteq V_1$, $FV\big(v_j := *; ?\pi^{in}(v_1); \ldots\big) \subseteq V_1$, $FV(\varphi_1) \subseteq V_1$ and $V_1 \cap V_2 \setminus \big(\mathrm{V}^{global} \cup \{t\}\big) = \emptyset$, and thus $FV(\alpha) \cap BV(\beta) = \emptyset$ and $FV(A) \cap BV(\beta) = \emptyset$. Finally, we apply Lemma 1 again to remove $in_2^\dagger$ and are done after reordering.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{*}{\varphi_1 \vdash [in_1; ctrl_1][\{t' = 1, plant_1\}]\varphi_1}(35)}{\varphi_1, \varphi_2 \vdash [in_1; ctrl_1][\{t' = 1, plant_1\}]\varphi_1}\text{Wl}}{\varphi_1, \varphi_2 \vdash [in_1^\dagger][(v_j := *; ?\pi^{in}(v_j)); ctrl_1][\{t' = 1, plant_1\}]\varphi_1}\text{Lemma 4,def}}{\varphi_1, \varphi_2 \vdash [in_1^\dagger][in_2^\dagger][(v_j := *; ?\pi^{in}(v_j)); ctrl_1][\{t' = 1, plant_1\}]\varphi_1}\text{Lemma 1}}{\varphi_1, \varphi_2 \vdash [in_1^\dagger][in_2^\dagger][ports_2; ctrl_2][(v_j := *; ?\pi^{in}(v_j)); ctrl_1][\{t' = 1, plant_1\}]\varphi_1}\text{Lemma 1}}{\varphi_1, \varphi_2 \vdash [in_1^\dagger][in_2^\dagger][ports_2; ctrl_2][(v_j := \mathcal{X}(v_j); ?\pi^{in}(v_j)); ctrl_1][\{t' = 1, plant_1\}]\varphi_1}\text{Lemma 3}}{\varphi_1, \varphi_2 \vdash [in_1^\dagger][in_2^\dagger][ports_2; ctrl_2][(v_j := \mathcal{X}(v_j); ?\varphi_2); ctrl_1][\{t' = 1, plant_1\}]\varphi_1}\text{Lemma 6}\quad \cdots ⑨}{\varphi_1, \varphi_2 \vdash [in_1^\dagger][in_2^\dagger][ports_2; ctrl_2][(v_j := \mathcal{X}(v_j)); ctrl_1][\{t' = 1, plant_1\}]\varphi_1}\text{Lemma 5}}{\varphi_1, \varphi_2 \vdash [in_3][ports_2; ctrl_2][ports_1; ctrl_1][\{t' = 1, plant_1\}]\varphi_1}\text{def}}{\varphi_1, \varphi_2 \vdash [in_3; ports_2; ctrl_2; ports_1; ctrl_1][\{t' = 1, plant_1\}]\varphi_1}[;]$$

⑦ continued

The side condition still holds, since from (36) we also know $\varphi_2 \to [in_2; ctrl_2]\varphi_2$ by reflexivity of $\{t' = 1, plant_2\}$.[7] We first use Lemma 1 to remove the unused input ports $in_1^\dagger$, introduce the test through Lemma 5, and weaken the test using Corollary 1. Finally, we change the deterministic assignments to non-deterministic ones (cf. Lemma 3) and get (36) by reordering assignments (cf. Lemma 4). The side condition for this application of Lemma 5 follows immediately (cf. ⑧, with $in_2^\dagger$ and φ_1).

[7] In dℒ, continuous evolution is reflexive since differential equations can evolve for time 0.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{*}{\varphi_1, \varphi_2 \vdash [\mathit{in}_2; \mathit{ctrl}_2]\varphi_2}}{\varphi_1, \varphi_2 \vdash [\mathit{in}_2^\dagger][(v_k := *; ?\pi^{\mathit{in}}(v_k))\,; \mathit{ctrl}_2]\varphi_2}\,\text{Lemma 4}}{\varphi_1, \varphi_2 \vdash [\mathit{in}_2^\dagger][(v_k := \mathcal{X}(v_k); ?\pi^{\mathit{in}}(v_k))\,; \mathit{ctrl}_2]\varphi_2}\,\text{Lemma 3}}{\varphi_1, \varphi_2 \vdash [\mathit{in}_2^\dagger][(v_k := \mathcal{X}(v_k); ?\varphi_1)\,; \mathit{ctrl}_2]\varphi_2}\,\text{Lemma 6}}{\varphi_1, \varphi_2 \vdash [\mathit{in}_2^\dagger][(v_k := \mathcal{X}(v_k))\,; \mathit{ctrl}_2]\varphi_2}\,\text{Lemma 5}}{\varphi_1, \varphi_2 \vdash [\mathit{in}_2^\dagger][\mathit{ports}_2; \mathit{ctrl}_2]\varphi_2}\,\text{def}}{\varphi_1, \varphi_2 \vdash [\mathit{in}_1^\dagger][\mathit{in}_2^\dagger][\mathit{ports}_2; \mathit{ctrl}_2]\varphi_2}\,\text{Lemma 1}$$

⑨: Side Condition, Lemma 5

The proof for ⑤ follows accordingly, using φ_2 in place of φ_1. Thus, we conclude $\phi_3 \rightarrow [(\mathit{in}_3; \mathit{ctrl}_3; \{t' = 1, \mathit{plant}_3\})^*]\,\psi_3$, i. e. (34) is valid. ents □

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 components, we have to consider n contracts, one for each component with its interface (C_i, I_i) (for $i \in \{0, \ldots, n\}$).

$$\mathit{Cont}(C_i, I_i) \equiv \phi_i \rightarrow [(\mathit{in}_i; \mathit{ctrl}_i; (t' = 1, \mathit{plant}_i))^*]\psi_i \tag{38}$$

Again, we assume that formula (38) was proven for all i, hence there exist invariants φ_i (cf. (35)-(37)). We still need to verify (34), except that *plant* now executes all n plants in parallel and *ctrl* contains all $n!$ permutations of all control parts, i. e.,

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

We define our invariant φ as the conjunction of all φ_1. The base case and use case follow immediately from the loop invariants φ_i. 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:

$$\bigwedge_i \varphi_i \vdash [\mathit{in}][(\mathit{ports}_1; \mathit{ctrl}_1; \mathit{ports}_2; \mathit{ctrl}_2; ...; \mathit{ports}_n; \mathit{ctrl}_n)\,; \{t' = 1, \mathit{plant}_1\}]\varphi_2$$

We ultimately have to reduce the branch to the loop induction step of component C_2 (cf. (36)). Thus, we have to remove the unnecessary parts. The plants can be removed, using Lemma 2, similar to ①. The unnecessary control parts right of ctrl_2 can be removed similar to ③ and the unnecessary control parts left of ctrl_2 can be removed similar to ④. These steps can be repeated until we reduced the formula to (36). In a similar way, all other branches can be proved.

A.2 Proof of Theorem 2 – Refinement Theorem

Theorem 2 (Refinement Theorem)**.** *Let* (C, I) *be a component with its interface, with contracts* $\text{Cont}_1(C, I)$ *and* $\text{Cont}_2(C, I)$*. If* Cont_1 *is valid and* Cont_2 *refines* Cont_1 *under these contracts, then* Cont_2 *is valid.*

Proof of Theorem 2. We know that

$$\models \mathit{Cont}_1(C, I), \text{ i. e., } \models \left(t = 0 \land \phi_1 \to [(\mathit{in}; \mathit{ctrl}; \{t' = 1, \mathit{plant}\})^*]\psi_1\right) \tag{39}$$

$$\models \phi_2 \to \phi_1 \tag{40}$$

$$\models \psi_1 \to \psi_2 \tag{41}$$

and need to prove

$$\models \mathit{Cont}(C, I), \text{ i. e., } \models \left(t = 0 \land \phi_2 \to [(\mathit{in}; \mathit{ctrl}; \{t' = 1, \mathit{plant}\})^*]\psi_2\right) \tag{42}$$

Out of space reasons we omit $t = 0$ and assume it is a part of ϕ_i and define

$$\mathit{HP}(C) = [(\mathit{in}; \mathit{ctrl}; \{t' = 1, \mathit{plant}\})^*]$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{*}{\psi_1 \vdash \psi_2}\,(8)}{[\mathit{HP}(C)]\psi_1 \vdash [\mathit{HP}(C)]\psi_2}\,[]\,\text{M}}{\phi_1, [\mathit{HP}(C)]\psi_1 \vdash [\mathit{HP}(C)]\psi_2}\,\text{Wl} \qquad \dfrac{*}{\phi_1 \vdash [\mathit{HP}(C)]\psi_1}\,(6)}{\phi_1 \vdash [\mathit{HP}(C)]\psi_2, [\mathit{HP}(C)]\psi_1}\,\text{Wr}}{\phi_1 \vdash [\mathit{HP}(C)]\psi_2}\,\mathit{cut} \qquad \dfrac{*}{\phi_2 \vdash \phi_1}\,(7)}{\phi_2, \phi_1 \vdash [\mathit{HP}(C)]\psi_2}\,\text{Wl} \qquad \dfrac{\phi_2 \vdash \phi_1}{\phi_2 \vdash [\mathit{HP}(C)]\psi_2, \phi_1}\,\text{Wr}}{\phi_2 \vdash [\mathit{HP}(C)]\psi_2}\,\mathit{cut}$$

$$\dfrac{\phi_2 \vdash [\mathit{HP}(C)]\psi_2}{\vdash \phi_2 \to [\mathit{HP}(C)]\psi_2}\,\to\text{r}$$

□