Change and Delay Contracts for Hybrid System Component Verification

Andreas Müller[1(✉)], Stefan Mitsch[2], Werner Retschitzegger[1], Wieland Schwinger[1], and André Platzer[2]

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

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

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

Keywords: Component-based development · Hybrid systems · Formal verification

1 Introduction

Cyber-physical systems (CPS) feature discrete dynamics (e.g., autopilots in airplanes, controllers in self-driving cars) as well as continuous dynamics (e.g., motion of airplanes or cars) and are increasingly used in safety-critical areas. Models of such CPS (i.e., hybrid system models, e.g., hybrid automata [8], hybrid programs [23]) are used to capture properties of these CPS as a basis to analyze their behavior and ensure safe operation with formal verification methods.

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

M. Huisman and J. Rubin (Eds.): FASE 2017, LNCS 10202, pp. 134–151, 2017.
DOI: 10.1007/978-3-662-54494-5_8

However, as the complexity of these systems increases, monolithic models and analysis techniques become unnecessarily challenging.

Since complex systems are typically composed of multiple subsystems and interact with other systems in their environment, it stands to reason to apply *component-based modeling* and split the analysis into isolated questions about subsystems and their interaction. However, approaches supporting component-based *verification* of hybrid system models are rare and differ strongly in the supported class of problems (cf. Sect. 5). Component-based techniques for hybrid (I/O) automata are based on *assume-guarantee reasoning* (AGR) [3,6,9] and focus on reachability analysis. Complementarily, hybrid systems theorem proving provides proofs, which are naturally compositional [22] (improves modularity) and support nonlinear dynamics, but so far limit components [15,16] to contracts over constant ranges (e.g., speed of a robot is non-negative and at most 10). Such contracts require substantial static independence of components, which does not fit to dynamic interactions frequently found in CPS. For example, one might show that a robot in the kitchen will not collide with obstacles in the physically separated back yard, but nothing can be said about what happens when both occupy the same parts of the space at different times to be safe. We, thus, extend CPS contracts [15,16] to consider change of values and timing.

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

2 Preliminaries: Differential Dynamic Logic

For specifying and verifying correctness statements about hybrid systems, we use *differential dynamic logic* ($\mathsf{d}\mathcal{L}$) [19,21], which supports *hybrid programs* as a program notation for hybrid systems, according to the following EBNF grammar:

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

For details on the formal semantics of hybrid programs see [19,21]. The sequential composition $\alpha;\beta$ expresses that β starts after α finishes. The non-deterministic choice $\alpha \cup \beta$ follows either α or β. The non-deterministic repetition operator α^* repeats α zero or more times. Discrete assignment $x := \theta$ instantaneously assigns the value of the term θ to the variable x, while $x := *$ assigns an arbitrary value to x. The ODE $(x' = \theta \;\&\; H)$ describes a continuous evolution of x (x' denotes derivation with respect to time) within the evolution domain H. The test $?H$ checks that a condition expressed by property H holds, and aborts if it does not. A typical pattern $x := *; ?a \leq x \leq b$, which involves assignment and tests, is to limit the assignment of arbitrary values to known bounds. Other control flow statements can be expressed with these primitives [19].

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

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

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

We use V to denote a set of variables. $FV(.)$ is used as an operator on terms, formulas and hybrid programs returning the free variables, whereas $BV(.)$ is an operator returning the bound variables.[1] Similarly, $V(.) = FV(.) \cup BV(.)$ returns all variables occurring in terms, formulas and hybrid programs. We use d$\mathcal{L}$ in definitions and formulas to denote the set of all d$\mathcal{L}$ formulas. We use "$\mapsto$" to define functions. $f = (a \mapsto b)$ means that the (partial) function f maps argument a to result b and is solely defined for a.

3 Hybrid Components with Change and Delay Contracts

In this section, we extend *components* (defined as hybrid programs) and their *interfaces* [16] with time and delay concepts. Interfaces identify assumptions about component inputs and guarantees about component outputs. We define what it means for a component to *comply with its contract* by a d$\mathcal{L}$ formula expressing safe behavior and compliance with its interface. And we define the *compatibility of component connections* rigorously as d$\mathcal{L}$ formulas as well. The main result of this paper is a proof showing that the safety properties of components transfer to a composed system, given proofs of contract compliance, compatibility and satisfaction of local side conditions. Users only provide a *specification* of components, interfaces, and how the components are connected, and

[1] *Bound variables* of a hybrid program are all those that may potentially be written to, while *free variables* are all those that may potentially be read [25].

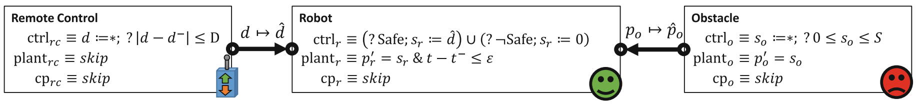
Remote Control
$\text{ctrl}_{rc} \equiv d :=*;\ ?\,|d - d^-| \leq D$
$\text{plant}_{rc} \equiv skip$
$\text{cp}_{rc} \equiv skip$
$d \mapsto \hat{d}$
Robot
$\text{ctrl}_r \equiv (?\,\text{Safe}; s_r := \hat{d}) \cup (?\,\neg\text{Safe}; s_r := 0)$
$\text{plant}_r \equiv p_r' = s_r \,\&\, t - t^- \leq \varepsilon$
$\text{cp}_r \equiv skip$
$p_o \mapsto \hat{p}_o$
Obstacle
$\text{ctrl}_o \equiv s_o :=*;\ ?\,0 \leq s_o \leq S$
$\text{plant}_o \equiv p_o' = s_o$
$\text{cp}_o \equiv skip$

Fig. 1. Components for a collision avoidance system with remote control

show *proof obligations* about component contract compliance, compatibility and local side conditions; system safety follows automatically.

To illustrate the concepts, we use a running example of a tele-operated robot with collision avoidance inspired by [10, 13], see Fig. 1: a *robot* reads speed advice d at least every ε time units from a *remote control* (RC), but automatically overrides the advice to avoid collisions with an *obstacle* that moves with an arbitrary but bounded speed $0 \leq s_o \leq S$ (e.g., a moving person). Two consecutive speed advisories from the RC should be at most D apart ($|d - d^-| \leq D$). The RC issues speed advice to the robot, but has no physical part. The obstacle chooses a new non-negative speed but at most S and moves according to the ODE $p_o' = s_o$. The robot measures the obstacle's position. If the distance is safe, the robot chooses the speed suggested by the RC; otherwise, the robot stops.

The RC satisfies a change contract (two consecutive speed advisories are not too far apart), while the obstacle and the robot satisfy delay contracts (their positions change according to speed and how much time passed). Formal definitions of these three components, their interfaces, and the respective contracts, will be introduced step-by-step along the definitions in subsequent sections. A comprehensive explanation of the running example can be found in [17].

3.1 Specification: Components and Interfaces

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

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

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

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

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

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

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

Consider the robot collision avoidance system. Its global variables are the maximum obstacle speed S and the maximum difference D between two speed advisories, i.e., $V^{global} = \{S, D\}$. They can neither be bound in control nor plant, cf. Definition 2. The RC component's controller picks a new demanded speed d that is not too far from the previous demanded speed d^-. Since it is not composed from sub-components the cp_{rc} part is `skip` in Fig. 1. The obstacle's controller chooses any speed s_o that does not exceed the maximum speed S. The plant ODE $p_o' = s_o$ describes how the position of the obstacle changes according to its speed. Since the obstacle is an atomic component, cp_o is `skip`, cf. Fig. 1.

An interface defines how a component may interact with other components through its ports, what assumptions the component makes about its inputs, and what guarantees it provides for its outputs, see Definition 3.

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

- *$V^{\text{in}} \subseteq V(C)$ and $V^{\text{out}} \subseteq V(C)$ are disjoint sets of input- and output variables,*
- *$V^{\text{in}} \cap BV(C) = \emptyset$, i.e., input variables may not be bound in the component,*
- *$\pi^{\text{in}} : V^{\text{in}} \to \mathsf{d}\mathcal{L}$ is a function specifying exactly one formula per input variable (i.e., input port), representing input requirements and assumptions,*
- *$\pi^{\text{out}} : V^{\text{out}} \to \mathsf{d}\mathcal{L}$ specifies output guarantees for output ports,*
- *$\forall v \in V^{\text{in}} \;:\; V(\pi^{\text{in}}(v)) \subseteq \left(V(C) \setminus \left(V^{\text{in}} \cup V^{\text{out}}\right)\right) \cup \{v\}$ such that input formulas are local to their port,*
- *$V^{\Delta} = V^{\Delta+} \cup V^{\Delta i} \subseteq V(C)$ is a set of Δ-port variables of unconnected public $V^{\Delta+} \subseteq V^{\text{in}} \cup V^{\text{out}}$, and connected private $V^{\Delta i}$, with $V^{\Delta i} \cap \left(V^{\text{in}} \cup V^{\text{out}}\right) = \emptyset$, so $V^{\Delta+} \cap V^{\Delta i} = \emptyset$,*
- *$V^{-} \subseteq V(C)$ with $V^{-} \cap BV(C) = \emptyset$ is a read-only set of variables storing the previous values of Δ-ports, disjoint from other interface variables $V^{-} \cap (V^{\text{in}} \cup V^{\text{out}} \cup V^{\Delta}) = \emptyset$,*
- *$pre : V^{\Delta} \to V^{-}$ is a bijective function, assigning one variable to each Δ-port to store its previous value.*

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

Considering our example, the RC interface I^{Δ}_{rc} from (1) contains one output port d in V^{out}, where the robot can retrieve the demanded speed. The RC guarantees that the new demanded speed d will not deviate too far from the previous speed d^-, so $|d - d^-| \leq D$. Since the current and the previous demanded speed are related, d is a Δ-port in V^{Δ} with its previous value d^- in V^- per pre.

$$\mathrm{I}^{\Delta}_{rc} = \Big(\underbrace{\{\}}_{V^{in}}, \underbrace{()}_{\pi^{in}}, \underbrace{\{d\}}_{V^{out}}, \underbrace{\big(d \mapsto \left|d - d^-\right| \leq D\big)}_{\pi^{out}}, \underbrace{\{d\}}_{V^{\Delta}}, \underbrace{\{d^-\}}_{V^-}, \underbrace{\big(d \mapsto d^-\big)}_{\text{pre}}\Big) \tag{1}$$

3.2 Specification: Time and Delay

In a monolithic hybrid program with a combined plant for all components, time passes synchronously for all components and their ODEs evolve for the same amount of time. When split into separate components, the ODEs are split into separate plants too, thereby losing the connection of evolving for identical amounts of time. From the viewpoint of a single component, other plants reduce to discrete abstractions through input assumptions on Δ-ports. These input assumptions are phrased in terms of worst-case behavior (e.g., from the viewpoint of the robot, the obstacle may jump at most distance $S \cdot \varepsilon$ between measurements because it lost a precise model). The robot's ODE, however, still runs for some arbitrary time, which makes the measurements and the continuous behavior of the robot drift (i.e., robot and obstacle appear to move for different durations). To address this issue, we introduce *delay* as a way of ensuring that the changes are consistent with the time that passes in a component.

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

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

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

Let us continue our running example with the obstacle's interface, which has one output port (providing the current obstacle position) that uses the introduced global time in its output property to restrict the obstacle's position relative to its previous position and maximum speed, i.e.,

$$\mathrm{I}_o^{\Delta} = \Big(\underbrace{\{\}}_{V^{in}}, \underbrace{()}_{\pi^{in}}, \underbrace{\{p_o\}}_{V^{out}}, \underbrace{\big(p_o \mapsto |p_o - p_o^-| \leq S \cdot (t - t^-)\big)}_{\pi^{out}}, \underbrace{\{p_o\}}_{V^{\Delta}}, \underbrace{\{p_o^-\}}_{V^-}, \underbrace{\big(p_o \mapsto p_o^-\big)}_{\text{pre}}\Big).$$

3.3 Proof Obligations: Change and Delay Contract

Contract compliance ties together components and interfaces by showing that a component guarantees the output changes that its interface specifies under the input assumptions made in the interface. Contract compliance further shows a local safety property, which describes the component's desired safe states. For example, a safety property of a robot might require that the robot will not drive too close to the last measured position of the obstacle. Together with the obstacle's output guarantee of not moving too far from its previous position, the local safety property implies a system-wide safety property (e.g., robot and obstacle will not collide), since we know that a measurement previously reflected the real position. Contract compliance can be verified using KeYmaera X [7].

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

Definition 5 (Contract Compliance). *Let C be a component with its admissible interface I^{Δ} (cf. Definition 3). Let formula ϕ describe initial states of C and formula ψ^{safe} the safe states, both over the component variables $V(C)$. The output guarantees $\Pi^{\text{out}} \equiv \bigwedge_{v \in V^{\text{out}}} \pi^{\text{out}}(v)$ extend safety to $\psi^{\text{safe}} \wedge \Pi^{\text{out}}$.* Change contract compliance $\mathrm{CC}(C, I^{\Delta})$ *of C with I^{Δ} is defined as the* d$\mathcal{L}$ *formula:*

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

and delay contract compliance $\mathrm{DC}(C, I^{\Delta})$ *is defined as the* d$\mathcal{L}$ *formula:*

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

where

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

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

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

The order of the assignments in both *in* and Δ is irrelevant because the assignments are over disjoint variables and $\pi^{in}(v)$ are local to their port, cf. Definition 3. The function pre can be used throughout the component to read the initial value of a Δ-port. Since $\mathrm{pre}(v) \in V^-$ for all $v \in V^{\Delta}$, Definitions 3 and 5 require that the resulting initial variable is not bound anywhere outside Δ.

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

Example: Change Contract. We continue the collision avoidance system with a change contract (2) according to Definition 5 for the RC from Fig. 1. The difference between speed advisories should be non-negative, whereas the output port's previous value d^- is bootstrapped from the current demanded speed d to guarantee contract compliance even without component execution, since non-deterministic repetitions can execute 0 times, so $\phi_{rc} \equiv D \geq 0 \land d = d^-$. The RC guarantees that consecutive speed advisories are at most D apart, see Π_{rc}^{out} ($\psi_{rc}^{safe} \equiv \top$).

$$\phi_{rc} \rightarrow [(\underbrace{d^- := d}_{\Delta_{rc}}; \underbrace{d := *; ?\left|d - d^-\right| \leq D}_{ctrl_{rc}}; \underbrace{\texttt{skip}}_{plant_{rc}}; \underbrace{\texttt{skip}}_{in_{rc}}; \underbrace{\texttt{skip}}_{cp_{rc}})^*] \underbrace{\left(\left|d - d^-\right| \leq D\right)}_{\Pi_{rc}^{out}} \tag{2}$$

We verified the RC contract using KeYmaera X and thus know that the component is safe and complies with its interface. Compared to contracts with fixed ranges as in approaches [3,16], we do not have to assume a global limit for demanded speeds d, but consider the previous advice d^- as a reference value when calculating the next speed advice.

Example: Delay Contract. Change in obstacle position depends on speed and on how much time passed. Hence, we follow Definition 5 to specify the obstacle delay contract (3). For simplicity, assume that maximum speed S is non-negative and the obstacle stopped initially. As before, the previous position p_o^- is bootstrapped from the current position p_o, so $\phi_o \equiv S \geq 0 \land p_o = p_o^- \land s_o = 0$. We model an adversarial obstacle, which does not care about safety. Thus, the output property only guarantees that positions change at most by $S \cdot (t - t^-)$, which is a discrete abstraction of the obstacle's physical movement. Such an abstraction can be found by solving the plant ODE or from differential invariants [24]. Again, we verified the obstacle's contract compliance using KeYmaera X.

$$\begin{aligned} t = t^- \land \phi_o \rightarrow [(\overbrace{p_o^- := p_o}^{\Delta_o}; \overbrace{s_o := *; ?(0 \leq s_o \leq S)}^{ctrl_o}; t^- := t; \{t' = 1, \overbrace{p_o' = s_o}^{plant_o}\}; \\ \underbrace{\texttt{skip}}_{in_o}; \underbrace{\texttt{skip}}_{cp_o})*] \underbrace{\left(\left|p_o - p_o^-\right| \leq S \cdot \left(t - t^-\right)\right)}_{\Pi_o^{out}} \end{aligned} \tag{3}$$

3.4 Proof Obligations: Compatible Parallel Composition

From components with verified contract compliance, we now compose systems and provide safety guarantees about them, without redoing system proofs. For this, Definition 6 introduces a quasi-parallel composition, where the discrete *ctrl* parts of the components are executed sequentially in any order, while the continuous *plant* parts run in parallel. The connected ports *cp* of all components are composed sequentially in any order, since the order of independent deterministic assignments (i.e., assignments having disjoint free and bound variables) is irrelevant. Such a definition is natural in $\mathsf{d}\mathcal{L}$, since time only passes during continuous evolution in hybrid programs, while the discrete actions of a program do not consume time and thus happen instantaneously at a single real point in time, but in a specific order. The actual execution order of independent components in a real system is unknown, which we model with a non-deterministic choice between all possible controller execution orders. Values can be exchanged between components using Δ-ports; all other variables are internal to a single component, except global variables, which can be read everywhere, but never bound, and system time t, t^-, which can be read everywhere, but only bound at specific locations fixed by the delay contract, cf. Definition 5. Δ-ports store their previous values in the composite component, regardless if connected or not. For all connected ports, Definition 6 replaces the non-deterministic assignments to open inputs (cf. *in*) with a deterministic assignment from the connected port (cf. *cp*). This represents an instantaneous and lossless interaction between components.

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

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

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

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

be a partial (i.e., not every input must be mapped), injective (i.e., every output is only mapped to at most one input) function, connecting some inputs to some outputs, with domain $\mathcal{I}^{\mathcal{X}} = \{x \in V^{\mathrm{in}} \mid \mathcal{X}(x) \text{ is defined}\}$ *and image* $\mathcal{O}^{\mathcal{X}} = \{y \in V^{\mathrm{out}} \mid y = \mathcal{X}(x) \text{ for some } x \in V^{\mathrm{in}}\}$*. The composition of* n *components and their interfaces* $(C, I^\Delta) \overset{def}{\equiv} \left((C_1, I_1^\Delta)\|\dots\|(C_n, I_n^\Delta)\right)_{\mathcal{X}}$ *according to* $\mathcal{X}$ *is defined as:*

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

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

– *plants are executed in parallel, with evolution domain* $H \equiv \bigwedge_{i\in\{1,\dots,n\}} H_i$

$$\mathrm{plant} \equiv \underbrace{x_1^{(1)\prime} = \theta_1^{(1)}, \dots, x_1^{(k)\prime} = \theta_1^{(k)}}_{component\ C_1}, \dots, \underbrace{x_n^{(1)\prime} = \theta_n^{(1)}, \dots, x_n^{(m)\prime} = \theta_n^{(m)}}_{component\ C_n} \ \&\ H,$$

– *port assignments are extended with connections for some* $\{v_j, \ldots, v_r\} = \mathcal{I}^{\mathcal{X}}$

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

– *previous values* $V^- \overset{def}{\equiv} \bigcup_{1\le i\le n} V_i^-$ *are merged; connected ports become private* $V^{\Delta i} \overset{def}{\equiv} \left(\bigcup_{1\le i\le n} V_i^{\Delta i}\right) \cup \mathcal{I}^{\mathcal{X}} \cup \mathcal{O}^{\mathcal{X}}$*; unconnected ports remain public* $V^{\Delta+} \overset{def}{\equiv} \left(\bigcup_{1\le i\le n} V_i^{\Delta+}\right) \setminus (\mathcal{I}^{\mathcal{X}} \cup \mathcal{O}^{\mathcal{X}})$,

– pre_i *are combined such that* $\mathit{pre}(v) \equiv \mathit{pre}_i(v)$ *if* $v \in V_i^{\Delta}$ *for all* $i \in \{1, \ldots, n\}$,

– *unconnected inputs* $V^{\mathrm{in}} = \left(\bigcup_{1\le i\le n} V_i^{\mathrm{in}}\right) \setminus \mathcal{I}^{\mathcal{X}}$ *and unconnected outputs* $V^{\mathrm{out}} = \left(\bigcup_{1\le i\le n} V_i^{\mathrm{out}}\right) \setminus \mathcal{O}^{\mathcal{X}}$ *are merged and their requirements preserved*

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

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

The user provides component specifications $(\mathrm{C}_i, \mathrm{I}_i^{\Delta})$ and a mapping function $\mathcal{X}$, defining which output is connected to which input. The composed system of parallel components can be derived automatically from Definition 6. It follows that the set of variables of the composite component $V(C)$ is the union of all involved components' variable sets $V(\mathrm{C}_i)$, i.e., $V(\mathrm{C}) = \bigcup_{1\le i\le n} V(\mathrm{C}_i)$. The set of global variables V^{global} contains all global variables in the system (i.e., in all components) and thus, its contents do not change. Since $V^{\Delta} = \bigcup_{1\le i\le n} V_i^{\Delta}$, this definition implies that internally connected Δ-ports $V^{\Delta i}$ of sub-components, as well as the previous values $V^{\Delta+}$ for all open Δ-ports are still stored. As a result, the current and previous values of Δ-ports can still be used internally in the composite, even when the ports are no longer exposed through the external interface of the composed system.

Returning to our running example of Fig. 1 the component C_{sys} in (4) and interface $\mathrm{I}_{sys}^{\Delta}$ in (5) result from parallel composition of the RC, the robot, and the obstacle. The robot controller follows the speed advice received on input port $\hat{d}$ if the robot is at a safe distance from the obstacle position measured with input port $\hat{p}_o$; otherwise it stops. The robot plant changes the robot's position according to its speed, where the controller executes at least every ε time-units. The robot's input ports are connected to the RC's and obstacle's output ports.[2]

$$\mathrm{C}_{sys} = (\underbrace{(\mathit{ctrl}_{rc}; \mathit{ctrl}_r; \mathit{ctrl}_o \cup \mathit{ctrl}_o; \ldots)}_{\mathit{ctrl}_{sys}}, \underbrace{(\mathit{plant}_r, \mathit{plant}_o)}_{\mathit{plant}_{sys}}, \underbrace{\hat{p}_o := p_o; \hat{d} := d}_{\mathit{cp}_{sys}}) \tag{4}$$

[2] For the detailed robot component C_r and interface I_r^{Δ}, see [17].

$$\mathrm{I}^{\Delta}_{sys} = \Big(\underbrace{\{\}}_{V^{in}}, \underbrace{()}_{\pi^{in}}, \underbrace{\{\}}_{V^{out}}, \underbrace{()}_{\pi^{out}}, \underbrace{\{p_o, d, \hat{p}_o, \hat{d}\}}_{V^{\Delta}}, \underbrace{\{p_o^-, d^-, \hat{p}_o^-, \hat{d}^-\}}_{V^-}, \underbrace{(p_o \mapsto p_o^-, ...)}_{\text{pre}}\Big) \quad (5)$$

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

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

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

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

Components are compatible if the output properties imply the input properties of connected ports. Compatibility guarantees that handing an output port's value over to the connected input port ensures that the input port's input assumption π^{in} holds, which is no longer checked explicitly by a test, so $\pi_j^{out}(\mathcal{X}(v)) \rightarrow \pi_i^{in}(v)$. To achieve local compatibility checks for pairs of connected ports, instead of global checks over entire component models, we restrict output guarantees, respectively input assumptions to the associated output ports, respectively input ports (cf. Definition 3). In our example, the robot and the obstacle, respectively the RC are compatible, as witnessed by proofs of $CPO(\mathrm{I}_{rc}^{\Delta})$ and $CPO(\mathrm{I}_o^{\Delta})$, cf. [17].

3.5 Transferring Local Component Safety to System Safety

After verifying contract compliance and compatibility proof obligations, Theorem 1 below ensures that the safety properties in component contracts imply safety of the composed system. Thus, to ensure a safety property of the monolithic system, we no longer need a (probably huge) monolithic proof, but can apply Theorem 1 (proof available in [17]).

Theorem 1 (Composition Retains Contracts). *Let* C_1 *and* C_2 *be components with admissible interfaces* I_1^{Δ} *and* I_2^{Δ} *that are* delay contract compliant *(cf. Definition 5) and* compatible *with respect to* $\mathcal{X}$ *(cf. Definition 7). Initially, assume* $\phi^p \overset{def}{\equiv} \bigwedge_{v \in \mathcal{I}^{\mathcal{X}}} \mathcal{X}(v) = v$ *to bootstrap connected ports. Then, if the side condition (6) holds (*φ_i *is the loop invariant used to prove the component's contract)*

$$\models \varphi_i \rightarrow [\Delta_i][\mathrm{ctrl}_i][t^- := t][(t' = 1, \mathrm{plant}_i)]\Pi_i^{\text{out}} \quad (6)$$

for all components C_i, the parallel composition $(C, I^{\Delta}) = \big((C_1, I_1^{\Delta}) \| (C_2, I_2^{\Delta})\big)_{\mathcal{X}}$ *then satisfies the contract* (7) *with* in*,* cp*,* ctrl*, and* plant *according to Definition 6:*

$$\models \big(t = t^- \wedge \phi_1 \wedge \phi_2 \wedge \phi^p\big) \rightarrow [(\Delta; \mathrm{ctrl}; t^- := t; (t' = 1, \mathrm{plant}); \mathrm{in}; \mathrm{cp})^*] \big(\psi_1^{\mathrm{safe}} \wedge \Pi_1^{\mathrm{out}} \wedge \psi_2^{\mathrm{safe}} \wedge \Pi_2^{\mathrm{out}}\big) . \tag{7}$$

The composite contract's precondition ϕ^p ensures that the values of connected ports are consistent initially. Side condition (6) shows that a component already produces the correct output from just its *ctrl* and *plant*; preparing the port inputs for the next loop iteration does not change the current output.

The side condition (6) is trivially true for components without output ports, since $\Pi_i^{out} \equiv true$. For atomic components without input ports, the proof of (6) automatically follows from the contract proof, since *in*; *cp* is empty. Because of the precondition ϕ^p and because *cp* is executed after every execution of the main loop (cf. Definition 5), we know that the values of connected input and output ports coincide in the safety property, as one would expect. Thus, for instance, if the local safety property of a single component mentions an input port (e.g., $\psi_1^{safe} \equiv |p_r - \hat{p}_o| > 0$, we can replace the input port with the original value as provided by the output port for the composite safety property (e.g., $\psi^{safe} \equiv |p_r - \hat{p}_o| > 0 \equiv |p_r - p_o| > 0$). Theorem 1 easily extends to n components (cf. proof sketch in [17]) and also holds for change contracts. A change port cannot be attached to a delay port and vice versa.

Going back to our example, the overall system property of our collision avoidance system follows from Theorem 1, given the local safety property of the robot, the change contract compliance of the RC, the delay contract compliance of the obstacle, and the compatibility of the connections. Since we verified all component contracts as well as the compatibility proof obligations and since the components with output ports are atomic and have no input ports (i.e., the side condition holds), safety of the collision avoidance system follows.

Automation. We implemented the proof steps of Theorem 1 as a KeYmaera X tactic, which automatically reduces a system safety proof to separate proofs about components[3]. This gave us the best of the two worlds: the flexibility of reasoning with components that our Theorem 1 provides, together with the soundness guarantees we inherit from KeYmaera X, which derives proofs by uniform substitution from axioms [25]. This is to be contrasted with the significant soundness-critical changes we would have to do if we were to add Theorem 1 as a built-in rule into the KeYmaera X prover core. Uniform substitution guarantees, e.g., that the subtle conditions on how and where input and output variables can be read or written in components are checked correctly.

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

4 Case Studies

To evaluate our approach (See footnote 3), we use the running example of a remote-controlled robot (RC robot) and revisit prior case studies on the European Train Control System (i.e., ETCS) [26], two-component robot collision avoidance (i.e., Robix) [13], and adaptive cruise control (i.e., LLC) [10]. In *ETCS*, a radio-block controller (RBC) communicates speed limits to a train, i.e., it requires the train to have at most speed d after some point m. The RBC multi-port change contract relates distances m, m^- and demanded speeds d, d^- in input assumptions/output guarantees of the form $d \geq 0 \wedge (d^-)^2 - d^2 \leq 2b(m - m^-) \wedge state = drive$, thus avoiding physically impossible maneuvers.

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

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

Table 1 summarizes the experimental results of the component-based approach in comparison to monolithic models in terms of duration and degree of proof automation. The column *Contract* describes the kind of contract used in the case study (i.e., multiport, delay contract or change contract), as well as whether or not the models use non-linear differential equations. The column *Automation* indicates fully automated proofs with checkmarks; it indicates the number of built-in tactics composed to form a proof script when user input is required. The column *Duration* compares the proof duration, using Z3 [14] as a back-end decision procedure to discharge arithmetic. The column *Sum* sums up the proof durations for the components (columns C_1 and C_2) and Theorem 1 (column *Th. 1*, i.e., checking compatibility, condition (6) and the execution of our composition proof). Checking the composition proof is fully automated, following the proof steps of Theorem 1.

All measurements were conducted on an Intel i7-6700HQ CPU@2.6 GHz with 16 GB memory. In summary, the results indicate that our approach verification leads to performance improvements and smaller user-provided proof scripts.

Table 1. Experimental results for case studies

	Contract				Automation				Duration [s]				
	Multi	Change	Delay	Non-linear	C_1	C_2	Th. 1	Mono-lithic	C_1	C_2	Th. 1	Sum	Mono-lithic
RC robot			✓		✓	✓	✓	✓	32	101	56	**189**	**1934**
ETCS [26]	✓	✓			✓	✓	✓	✓	127	608	179	**873**	**15306**
Robix [13]			✓	✓	(31)	✓	✓	(96)	469	117	132	**718**	**902**
LLC [10]	✓		✓		✓	(50)	✓	(131)	135	351	267	**753**	**568**

5 Related Work

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

Hybrid Automata and Assume-Guarantee Reasoning. Hybrid automata [1] can be composed in parallel. However, the associated verification procedure (i.e., verify that a formula holds throughout all runs of the automaton) is not compositional, but requires verification of the exponential product automaton [1]. Thus, for a hybrid automaton it is not sufficient to establish a property about its parts in order to establish a property about the automaton. We, instead, decompose *verification* into local proofs and get system safety automatically. Hybrid I/O automata [11] extend hybrid automata with a notion of external behavior. The associated implementation relation (i.e., if automaton A implements automaton B, properties verified for B also hold for A) is respected by their composition operation in the sense that if A_1 implements A_2, then the composition of A_1 and B implements the composition of A_2 and B. Hybrid (I/O) automata are mainly verified using reachability analysis. Therefore, techniques to prevent state-space explosion are needed, like assume-guarantee reasoning (AGR, e.g., [3,6,9]), which was developed to decompose a verification task into subtasks. In [6], timed transition systems are used to approximate a component's behavior by discretization. These abstractions are then used in place of the more complicated automata to verify refinement properties. The implementation of their approach is limited to linear hybrid automata. In analogy, we discretize plants to delay contracts; however, in our approach, contracts completely replace components and do not need to retain simplified transition systems. A similar AGR rule is presented in [9], where the approximation drops continuous behaviors of single components entirely. As a result, the approach only works when the continuous behavior is irrelevant to the verified property, which rarely happens in CPS. Our change and delay contracts still preserve knowledge about continuous behavior. The AGR approach of [3] uses contracts consisting of input assumptions and output guarantees to verify properties about single components: a component is an abstraction of another component if it has a stricter contract. The approach is restricted to constant intervals, i.e., static global contracts as in [16].

In [5], a component-based design framework for controllers of hybrid systems with linear dynamics based on hybrid automata is presented. It focuses on checking interconnections of components: alarms propagated by an out-port must be handled by the connected in-ports. We, too, check component compatibility, but for contracts, and focus on transferring proofs from components to the system level. We provide parallel composition, while [5] uses sequential composition. The compositional verification approach in [2] bases on linear hybrid automata using invariants to over-approximate component behavior and interactions. However, interactions between components are restricted to synchronization. (i.e., no variable state can be transferred between components).

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

Hybrid process algebras are compositional modeling formalisms for the description of behavior and interaction of processes, based on algebraic equations. Examples are Hybrid χ [27], HyPA [18] or the Φ-Calculus [28]. Although the modeling is compositional, for verification purposes the models are again analyzed using simulation or reachability analysis in a non-compositional fashion (e.g., Hybrid χ using PHAVer [30], HyPA using HyTech [12], Φ-Calculus using SPHIN [29]), while we focus on exploiting compositionality in the proof.

Hybrid Programs. Quantified hybrid programs enable a compositional verification of hybrid systems with an arbitrary number of components [20], if they all have the same structure (e.g., many cars, or many robots). They were used to split monolithic hybrid program models into smaller parts to show that adaptive cruise control prevents collisions for an arbitrary number of cars on a highway [10]. We focus on different components. Similarly, the approach in [15] presents a component-based approach limited to traffic flow and global contracts.

Our approach extends [16], which was restricted to contracts over constant ranges. Such global contracts are well-suited for certain use cases, where the change of a port's value does not matter for safety, such as the traffic flow models of [15]. However, for systems such as the remote-controlled robot obstacle avoidance from our running example (cf. Sect. 3.1), which require knowledge about the change of certain values, global contracts only work for considerably more conservative models (e.g., robot and obstacle must stay in fixed globally known regions, since the obstacle's last position is unknown). Contracts with change and delay allow more liberal component interaction.

6 Conclusion and Future Work

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

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

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

References

1. Alur, R., Courcoubetis, C., Henzinger, T.A., Ho, P.-H.: Hybrid automata: an algorithmic approach to the specification and verification of hybrid systems. In: Grossman, R.L., Nerode, A., Ravn, A.P., Rischel, H. (eds.) HS 1991-1992. LNCS, vol. 736, pp. 209–229. Springer, Heidelberg (1993). doi:10.1007/3-540-57318-6_30
2. Aştefănoaei, L., Bensalem, S., Bozga, M.: A compositional approach to the verification of hybrid systems. In: Ábrahám, E., Bonsangue, M., Johnsen, E.B. (eds.) Theory and Practice of Formal Methods. LNCS, vol. 9660, pp. 88–103. Springer, Cham (2016). doi:10.1007/978-3-319-30734-3_8
3. Benvenuti, L., Bresolin, D., Collins, P., Ferrari, A., Geretti, L., Villa, T.: Assume-guarantee verification of nonlinear hybrid systems with Ariadne. Int. J. Robust Nonlinear Control **24**(4), 699–724 (2014)
4. Bohrer, B., Rahli, V., Vukotic, I., Völp, M., Platzer, A.: Formally verified differential dynamic logic. In: Bertot, Y., Vafeiadis, V. (eds.) Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs, pp. 208–221. ACM (2017)
5. Damm, W., Dierks, H., Oehlerking, J., Pnueli, A.: Towards component based design of hybrid systems: safety and stability. In: Manna, Z., Peled, D.A. (eds.) Time for Verification. LNCS, vol. 6200, pp. 96–143. Springer, Heidelberg (2010). doi:10.1007/978-3-642-13754-9_6
6. Frehse, G., Han, Z., Krogh, B.: Assume-guarantee reasoning for hybrid I/O-automata by over-approximation of continuous interaction. In: 43rd IEEE Conference on Decision and Control, CDC, vol. 1, pp. 479–484 (2004)
7. Fulton, N., Mitsch, S., Quesel, J.-D., Völp, M., Platzer, A.: KeYmaera X: an axiomatic tactical theorem prover for hybrid systems. In: Felty, A.P., Middeldorp, A. (eds.) CADE 2015. LNCS (LNAI), vol. 9195, pp. 527–538. Springer, Cham (2015). doi:10.1007/978-3-319-21401-6_36
8. Henzinger, T.A.: The theory of hybrid automata. In: Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science, pp. 278–292. IEEE Computer Society (1996)
9. Henzinger, T.A., Minea, M., Prabhu, V.: Assume-guarantee reasoning for hierarchical hybrid systems. In: Di Benedetto, M.D., Sangiovanni-Vincentelli, A. (eds.) HSCC 2001. LNCS, vol. 2034, pp. 275–290. Springer, Heidelberg (2001). doi:10.1007/3-540-45351-2_24
10. Loos, S.M., Platzer, A., Nistor, L.: Adaptive cruise control: hybrid, distributed, and now formally verified. In: Butler, M., Schulte, W. (eds.) FM 2011. LNCS, vol. 6664, pp. 42–56. Springer, Heidelberg (2011). doi:10.1007/978-3-642-21437-0_6

11. Lynch, N.A., Segala, R., Vaandrager, F.W.: Hybrid I/O automata. Inf. Comput. **185**(1), 105–157 (2003)
12. Man, K.L., Reniers, M.A., Cuijpers, P.J.L.: Case studies in the hybrid process algebra Hypa. Int. J. Softw. Eng. Knowl. Eng. **15**(2), 299–306 (2005)
13. Mitsch, S., Ghorbal, K., Platzer, A.: On provably safe obstacle avoidance for autonomous robotic ground vehicles. In: Newman, P., Fox, D., Hsu, D. (eds.) Robotics: Science and Systems IX (2013)
14. de Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008). doi:10.1007/978-3-540-78800-3_24
15. Müller, A., Mitsch, S., Platzer, A.: Verified traffic networks: component-based verification of cyber-physical flow systems. In: 18th International Conference on Intelligent Transportation Systems, pp. 757–764 (2015)
16. Müller, A., Mitsch, S., Retschitzegger, W., Schwinger, W., Platzer, A.: A component-based approach to hybrid systems safety verification. In: Ábrahám, E., Huisman, M. (eds.) IFM 2016. LNCS, vol. 9681, pp. 441–456. Springer, Cham (2016). doi:10.1007/978-3-319-33693-0_28
17. Müller, A., Mitsch, S., Retschitzegger, W., Schwinger, W., Platzer, A.: Change and delay contracts for hybrid system component verification. Technical report CMU-CS-17-100, Carnegie Mellon (2017)
18. Cuijpers, P.J.L., Reniers, M.A.: Hybrid process algebra. J. Log. Algebr. Program. **62**(2), 191–245 (2005)
19. Platzer, A.: Differential-algebraic dynamic logic for differential-algebraic programs. J. Log. Comput. **20**(1), 309–352 (2010)
20. Platzer, A.: Quantified differential dynamic logic for distributed hybrid systems. In: Dawar, A., Veith, H. (eds.) CSL 2010. LNCS, vol. 6247, pp. 469–483. Springer, Heidelberg (2010). doi:10.1007/978-3-642-15205-4_36
21. Platzer, A.: A complete axiomatization of quantified differential dynamic logic for distributed hybrid systems. Logical Methods Comput. Sci. **8**(4), 1–44 (2012)
22. Platzer, A.: The complete proof theory of hybrid systems. In: Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, pp. 541–550. IEEE Computer Society (2012)
23. Platzer, A.: Logics of dynamical systems science. In: Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, pp. 13–24. IEEE Computer Society (2012)
24. Platzer, A.: The structure of differential invariants and differential cut elimination. Logical Methods Comput. Sci. **8**(4), 1–38 (2012)
25. Platzer, A.: A complete uniform substitution calculus for differential dynamic logic. J. Autom. Reas. 1–47 (2016). doi:10.1007/s10817-016-9385-1
26. Platzer, A., Quesel, J.-D.: European train control system: a case study in formal verification. In: Breitman, K., Cavalcanti, A. (eds.) ICFEM 2009. LNCS, vol. 5885, pp. 246–265. Springer, Heidelberg (2009). doi:10.1007/978-3-642-10373-5_13
27. Schiffelers, R.R.H., van Beek, D.A., Man, K.L., Reniers, M.A., Rooda, J.E.: Formal semantics of hybrid Chi. In: Larsen, K.G., Niebert, P. (eds.) FORMATS 2003. LNCS, vol. 2791, pp. 151–165. Springer, Heidelberg (2004). doi:10.1007/978-3-540-40903-8_12
28. Rounds, W.C., Song, H.: The Ö-calculus: a language for distributed control of reconfigurable embedded systems. In: Maler, O., Pnueli, A. (eds.) HSCC 2003. LNCS, vol. 2623, pp. 435–449. Springer, Heidelberg (2003). doi:10.1007/3-540-36580-X_32

29. Song, H., Compton, K.J., Rounds, W.C.: SPHIN: a model checker for reconfigurable hybrid systems based on SPIN. Electr. Notes Theor. Comput. Sci. **145**, 167–183 (2006)
30. Xinyu, C., Huiqun, Y., Xin, X.: Verification of hybrid Chi model for cyber-physical systems using PHAVer. In: Barolli, L., You, I., Xhafa, F., Leu, F.Y., Chen, H.C. (eds.) 7th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, pp. 122–128. IEEE Computer Society (2013)