Compositional proofs in differential dynamic logic dL

Simon Lunel, Benoît Boyer, Jean-Pierre Talpin

Compositional proofs in differential dynamic logic $d\mathcal{L}$

Simon Lunel and Benoît Boyer
Mitsubishi Electric R&D Centre Europe
1 allée de Beaulieu, CS 10806
35708 Rennes CEDEX 7, France
Email: s.lunel@fr.merce.mee.com

Jean-Pierre Talpin
Inria, Centre de recherche Rennes - Bretagne - Atlantique
Campus universitaire de Beaulieu
35042 Rennes Cedex, FRANCE
Email: jean-pierre.talpin@inria.fr

***Abstract*—Modularity and composability are essential properties to facilitate and scale the design of cyber-physical systems from the specification of hybrid, discrete and continuous, components. Modularity is essential to break down a system model into comprehensible and manageable component specifications. Composability is essential to design a system from component models while preserving their verified properties, expressed as assume-guarantee contracts.**

In this paper, we address the specification of hybrid system using Platzer's differential dynamic logic ($d\mathcal{L}$). Our contribution is threefold: (1) We define a new composition operator in $d\mathcal{L}$ and prove that it is associative and commutative (AC). Prior notions of composition in $d\mathcal{L}$ were not associative. (2) We provide a theorem which characterizes necessary conditions to automate the proof that composed components satisfy the composition of their individual contracts, enabling modular and compositional verification. (3) We case-study our AC composition operator by considering the modular and detailed specification of a cruise controller in KeYmaera X, the latest implementation of $d\mathcal{L}$, to demonstrate the proof automation capability of our contribution and exemplify a compositional design methodology.

1. Introduction

The problem of verifying the safety properties of an hybrid system is complex. It subsumes the problem of verifying that of its discrete control program, which is itself undecidable. It is hence desirable to seek abstraction and modularity in order to gain tractability and scalability. An approach to tame system verification complexity is to break down the system model into sub-systems and components: to conceive each functional part of the system separately and relate them using contracts expressing logical assumptions and guarantees between a component and its environment. To support a compositional design method, it is desirable to carry out the proof of composed components automatically from the composition of contracts that they were proved to individually satisfy. This can be done by defining a composition operator $\otimes$ in a manner algebraically suitable to the logic under consideration.

In this paper, we consider differential dynamic logic, $d\mathcal{L}$, proposed by Platzer et al. [1] to specify and verify hybrid system models. $d\mathcal{L}$ is an extension of Pratt's dynamic logic [2] (DL) with ordinary differential equations (ODEs): DL allows specifying the behavior of discrete programs and express their safety and liveness properties, while $d\mathcal{L}$ additionally allows to define time-continuous variables. It also provides an extension of a sequent calculus to prove properties about them. The sequent calculus of $d\mathcal{L}$ provides an important theoretical contribution to the field of hybrid system design, as it enable automated proof reasoning to verify large hybrid system specifications, as we will attempt to demonstrate in this paper.

1.1. Related work

Despite the inherent expressive capability of $d\mathcal{L}$ and its sequent calculus, some additional features need to be introduced in order to address the modular verification of large hybrid systems.

To prove that a system in $d\mathcal{L}$ satisfies some safety or liveness property, one straightforward approach is to express it as a formula, and then try to prove this formula correct. For a large system, the deductive process involved in the automation of such a proof leads to the generation of proof obligations, which can potentially be numerous, larger, and complex (to correlate with the initial problem), gradually making the ultimate system proof a lot more complex. This necessary deductive and proof automation process can hence possibly limit scalability of $d\mathcal{L}$ as a logical method to prove hybrid systems, and refrain non-experienced user from using it.

To alleviate this issue, a recent paper by Müller et al. [3] introduces a notion of composition in $d\mathcal{L}$. The approach proposed in that paper consist of breaking down a system model into independent functional parts or components, to prove the properties of these components, expressed by the mean of contract, and compose the components to obtain the system. However, the composition operator proposed in [3] is not associative. As a consequence, one has to define the whole system at once, as a structured composition of components, and then prove each of its components' properties.

In this paper, we contribute to $d\mathcal{L}$ with a composition operator that additionally enjoys associativity. This property permit a more flexible design, and coupled with our theorem that we can under some conditions automate the proof of a composed system from the proof of the components, permit a more scalable proof method.

In turn, our composition operator is largely inspired by Actions Systems [4]. Action Systems theory has also inspired Event-B to design and prove systems by refinement (which was our initial aim). Ronkko et al. [5], [6] have already proposed hybrid extensions to the framework of Action Systems and introduced Hybrid Action Systems. Hybrid Action Systems provide an AC composition operator which greatly inspired our contribution. By combining AC-composition with $d\mathcal{L}$, we aim to take advantage of the best of both theories which, furthermore, is a promising prospect toward a refinement-based design methodology.

It is also worth mentioning recent developments on Event-B [7] to define the so-called Hybrid Event-B. This formalism is also greatly inspired by hybrid action systems, and follows the spirit of the B method in the sense that it is founded on set theory, which seems not algebraically expressive or structured enough to scale large systems.

1.2. Contributions

Our work has been conducted, specified and proven using the interactive theorem prover KeYmaera X, which implements the $d\mathcal{L}$ logic. In this paper, we address the specification of hybrid system in Platzer's differential dynamic logic and define a new composition operator that we prove associative and commutative (AC). Then, we provide a theorem which characterizes the necessary conditions to automate the proof that composed components satisfy the composition of their contracts.

To exemplify and test-case our result, we have defined a detailed, modular, specification of a cruise controller [3] in $d\mathcal{L}$, which we use to demonstrate the proof automation capability of our contribution and exemplify a compositional design methodology.

2. Differential Dynamic Logic ($d\mathcal{L}$)

This section briefly introduces the *differential dynamic logic* ($d\mathcal{L}$), defined by A. Platzer [1]. As in DL, system properties in $d\mathcal{L}$ are expressed in classical logic, but they are interpreted over the reals, due to the continuous nature of systems under consideration. Additionally, $d\mathcal{L}$ supports a proof system, which is implemented in the interactive theorem prover Keymaera X.

In $d\mathcal{L}$, the notation $\dot{X} = \theta \,\&\, H$ denotes an ODE where $\dot{X} = \theta$ is a system of n equations of the form $\dot{x_1} = \theta_1$, $\ldots$, $\dot{x_n} = \theta_n$ and H is the characterization of the domain of the ODE. Derivative variables are identified using the convention of Physics, *i.e.* $\dot{x_i}$, in order to emphasize that derivation is done with respect to time, denoted by the reserved variable t. The body θ_i of each equation is a real arithmetic term. The domain of the ODE restricts the validity domain of the ODE solutions, which is $\mathbb{R}^n$ at most. The semantic of an ODE is given by his solution $f\colon [0, r] \to \mathbb{R}$. The states reachable by an ODE are the $f(r)$, denoted now by f_r, for any r.

Example 1. *The ODE $\dot{t} = 1 \,\&\, t \geqslant 0$ describes how the time t evolves. The domain restriction $t \geqslant 0$ ensures that we don't consider negative values for the time.*

Definition 1 (Syntax of $d\mathcal{L}$)**.**

$$
\begin{array}{lll}
\alpha, \beta & ::= x := \theta \mid ?\varphi \mid \alpha; \beta \mid \alpha \cup \beta \mid \alpha^* \mid \dot{x} = \theta \,\&\, H & \textit{(specs)} \\
\theta & ::= x \mid 0 \mid 1 \mid \theta + \theta \mid \theta - \theta \mid \theta \times \theta \mid \theta \div \theta & \textit{(terms)} \\
\varphi, \psi & ::= \theta_1 \sim \theta_2 \mid \bot \mid \varphi \to \psi \mid \forall x \varphi \mid [\alpha]\varphi & \textit{(formulas)} \\
 & \textit{where} \;\; \sim \in \{<, \leqslant, =, \geqslant, >\} &
\end{array}
$$

Hybrid program specifications, Def. 1, consist of assignments and tests inductively combined using sequences, non deterministic choices and an arbitrary but finite number of iterations (α^*), for the *discrete* part, and an ODE, for the *continuous* part. Terms are interpreted using real arithmetic. In a formula, the modal operator $[\alpha]\varphi$ means φ holds after any valid execution of α. The other logical connectives have the usual encoding (*e.g.* $\neg\varphi \equiv \varphi \to \bot$).

A *state* ν is a mapping from variables to $\mathbb{R}^n$. We denote the value of a term θ in ν by $[\![\theta]\!]_\nu$. The set of states is noted $\mathcal{S}$. The semantic of hybrid programs is a semantic of *reachability*, Def. 2.

Definition 2 (Semantic of hybrid programs)**.** *Assuming α, β are hybrid programs, $\rho_\nu(\alpha)$ inductively defines the set of the reachable states from ν by α:*

$$
\begin{array}{ll}
\rho_\nu(x := \theta) & = \{\omega \mid \omega = \nu \textit{ except that } [\![x]\!]_\omega = [\![\theta]\!]_\nu\} \\
\rho_\nu(?\varphi) & = \{\nu \mid \nu \models \varphi\} \\
\rho_\nu(\dot{x} = \theta_x \,\&\, H) & = \{f(r) \mid f_r(0) = \nu \textit{ and } f_r(t) \models \dot{x} = \theta \\
 & \qquad \textit{and } f_r(t) \models H \textit{ for any duration } r\} \\
\rho_\nu(\alpha \cup \beta) & = \rho_\nu(\alpha) \cup \rho_\nu(\beta) \\
\rho_\nu(\alpha; \beta) & = \bigcup_{\omega \in \rho_\nu(\alpha)} \rho_\omega(\beta) \\
\rho(\alpha^*) & = \bigcup_{n \in \mathbb{N}} \rho(\alpha^n) \textit{ with } \alpha^0 =?\top \\
 & \qquad \textit{and } \; \alpha^{n+1} = \alpha^n; \alpha
\end{array}
$$

The run of an hybrid program may modify some variables and may only read some other variables. This distinction is useful to characterize the interaction of a running program with its context. The notion of *free and bound variables* of an hybrid program are introduced for this purpose: let $Var(\alpha)$ be the set of all variables that occur in the hybrid program α. Bound variables can be updated by α whereas free variables are read at most.

Definition 3 (Free and Bound Variables)**.** *The set $BV(\alpha)$ of any hybrid program α, defines are the variables that may be updated by assignements (e.g. $x := 10$) or ODEs (e.g. $\dot{x} = 3$). The free variables, $FV(\alpha) = Var(\alpha) \backslash BV(\alpha)$, are the variables read by α, but never modified by α.*

The behavior of a cyber-physical system can be modeled using this language and its properties are expressed as formulas.

Definition 4 (Semantic of formulas)**.** *The satisfiability of formulas is provided for any state ν:*

$$\begin{array}{lll} \nu \models \theta_1 \sim \theta_2 & \Leftrightarrow & [\![\theta_1]\!]_\nu \sim [\![\theta_2]\!]_\nu, \ \sim \in \{<, \leqslant, =, \geqslant, >\} \\ \nu \models \varphi \rightarrow \psi & \Leftrightarrow & \nu \models \varphi \Rightarrow \nu \models \psi \\ \nu \models \forall x\ \varphi & \Leftrightarrow & \omega \models \varphi, \forall \omega |\ \forall z, z \neq x \Rightarrow [\![z]\!]_\omega = [\![z]\!]_\nu \\ \nu \models [\alpha]\varphi & \Leftrightarrow & \forall \omega \in \rho_\nu(\alpha),\ \omega \models \varphi \\ \nu \not\models \bot & & \end{array}$$

The logic $d\mathcal{L}$ enjoys a proof calculus [1], [8], [9]. Its base rules are these of a classical first-order sequent calculus. A second set of rules is dedicated to the proof of goals of the form $[\alpha]\phi$. They support straightforward reasoning on any hybrid program α by deconstruction. The last set of rules is dedicated to iteration and differential equations.

3. Design framework for the specification of component

This section defines a notion of component together with an operator of composition, along with the proofs of some properties of it. To gain in clarity, we partition specifications α into their discrete and continuous parts $\mathbf{disc}_\alpha$ and $\mathbf{cont}_\alpha$.

Definition 5. *The general form of a component α consists of its partition into a continuous part $\boldsymbol{cont}_\alpha$, of the form $?\psi_X; \dot{X} = \theta_X \ \& \ H_X$, and the discrete part $\boldsymbol{disc}_\alpha$.*

Notice that the discrete part of a system $\mathbf{disc}_\alpha$ is itself defined by the union of discrete, functional, components. We additionally define the *inputs* and *outputs* of a component. The free (the used) variables $FV(\alpha)$ of a component, defined in the previous section, are regarded as its inputs. Conversely, the bound (the defined) variables $BV(\alpha)$ of a component are interpreted as its outputs and internal variables.

Example 2. *We model the discrete controller of a cruise control system that is responsible for delivering a targeted speed to the vehicle engine (e.g. a car).*

$$(s_{ctrl} := *; ?(0 \leqslant s_{ctrl} \leqslant S \wedge |s_{ctrl} - s_{tach}| \leqslant \delta))^*$$

It chooses an arbitrary value for s_{ctrl}, the targeted speed, and checks if it is in the desired range $[0, S]$, where S is the speed limit. It additionally checks that the difference between the speed measured by the tachymeter, s_{tach}, and the target speed, s_{ctrl}, is not too high, in order to ensure that the acceleration set by the throttle is not too brutal and/or within the capabilities of the engine. Next, we model the continuous acceleration of the engine by a guarded derivative:

$$?(a_{thro} = \frac{s_{ctrl} - s_{tach}}{\varepsilon} \wedge s_{eng} = s_{tach}); \dot{s_{eng}} = a_{thro}$$

The guard checks that the acceleration a_{thro} given by the throttle conforms to limits and that the value of the speed measured by the tachymeter, s_{tach}, is equal to the actual value of speed of the engine, s_{eng}. The guarded action is a differential equation describing the evolution of speed by a function of the acceleration over time.

We define the composition $\oplus$ of two continuous components.

Definition 6. *Let $\alpha \triangleq \dot{X} = ?\psi_X; \theta_X \ \& \ H_X$ and $\beta \triangleq \dot{Y} = ?\psi_Y; \theta_Y \ \& \ H_Y$ be two continuous components (ODEs) such that they do not share any bound variables ($\dot{X} \cap \dot{Y} = \varnothing$) :*

$$\alpha \oplus \beta \triangleq ?\psi_X \wedge \psi_Y; \dot{X}, \dot{Y} = \theta_X, \theta_Y \ \& \ (H_X \wedge H_Y)$$

To illustrate, assume that we want to set a time limit to our vehicle engine component.

Example 3. *To set a time limit, we need to make time explicit in our system, by using the variable t, and the representation of the passing of time is achieved by introducing the ODE $\dot{t} = 1$. We add the formula $t \leqslant \varepsilon$ in the evolution domain $\dot{t} = 1 \ \& \ t \leqslant \varepsilon$, where ε is the desired limit. By composing this proposition with our example, we obtain the timed model:*

$$?\psi; \dot{s_{eng}} = a_{thro}, \dot{t} = 1 \ \& \ (t \leqslant \varepsilon)$$

where ψ is $a_{thro} = \frac{s_{ctrl} - s_{tach}}{\varepsilon} \wedge s_{eng} = s_{tach}$.

Our notion of composition between two ODEs differs from the one of Ronkko et al. [5] in that we just restrict the behaviors of components on the intersection of their domains, andd we require for a separation between their bounded variables.

Given that slight refinement, it is nonetheless possible to prove that composition is associative and commutative.

Property 1 (Commutativity of $\oplus$)**.** $\forall \alpha, \beta, \quad \alpha \oplus \beta = \beta \oplus \alpha.$

Property 2 (Associativity of $\oplus$)**.**

$$\forall \alpha, \beta, \gamma, \quad (\alpha \oplus \beta) \oplus \gamma = \alpha \oplus (\beta \oplus \gamma)$$

We define the composition of two components in two cases: the composition of two continuous components and between hybrid ones (discrete and/or continuous).

Definition 7. *Let α and β be two components. If α and β are continuous components, then:*

$$\alpha \otimes \beta \triangleq \alpha \oplus \beta$$

If α and β are of the general form, then:

$$\alpha \otimes \beta \triangleq ((\boldsymbol{disc}_\alpha \cup \boldsymbol{disc}_\beta) \cup (\boldsymbol{cont}_\alpha \oplus \boldsymbol{cont}_\beta))^*$$

We exemplify the composition of general component, with both discrete and continuous part, by considering the composition of the previous engine with a tachymeter. The role of a tachymeter is to be the interface between the physical world and the discrete world. In our model, it just consist of $(s_{tach} := s_{eng})^*$, i.e. it measures repeatedly the speed value.

Example 4. *The composition of the engine and the tachymeter is:*

$$Engine \otimes Tachymeter \triangleq ((s_{tach} := s_{eng}) \cup (?a_{thro} = \tfrac{s_{ctrl} - s_{tach}}{\varepsilon} \wedge s_{eng} = s_{tach}; \dot{s_{eng}} = a_{thro}))^*$$

Property 3 (Commutativity of $\otimes$)**.**

$$\forall \alpha, \beta, \quad \alpha \otimes \beta = \beta \otimes \alpha$$

Property 4 (Associativity of $\otimes$)**.**

$$\forall \alpha, \beta, \gamma, \quad (\alpha \otimes \beta) \otimes \gamma = \alpha \otimes (\beta \otimes \gamma)$$

We have presented a definition of component in $d\mathcal{L}$ as a syntactical composition operator along with its algebraic properties. We now have a way to model systems and compose them together. The next step addresses ways to express properties on such systems and prove them.

4. Composition theorem

We consider a notion of *contract* to associate component specifications α with a pair of formulas (A, G). A, the assumption, defines valid conditions for executing α. G, the guarantee, describes the properties ensured by the execution of α when A is satisfied. In $d\mathcal{L}$, the satisfiability of a contract (A, G) by a component α is expressed as $A \rightarrow [\alpha]G$. In the rest of the paper, we will use a calculus on logically equivalent sequents $A \vdash [\alpha]G$. To achieve proofs of such sequents, we need to consider the whole context of α by considering its initial state and defining its parameters: we build a logical context Γ which includes the assumption A, the formulas for the initialization and parameters of α. This leads us to the notation $\Gamma \vdash [\alpha]\varphi$ where $A \subseteq \Gamma$ and $G \subseteq \varphi$.

Example 5. *The engine contract is based on the assumption* $A_{eng} : 0 \leqslant s_{tach} \leqslant S \wedge 0 \leqslant s_{ctrl} \leqslant S$*. Moreover, the engine is parametrized by* $\varepsilon > 0$ *and* $S > 0$*. Those additional hypotheses are generally required to complete proofs:* $\Gamma \triangleq A_{eng} \wedge \varepsilon > 0 \wedge S > 0$.

To illustrate our approach, let us consider two components α and β, and two proofs P_α and P_β for the sequents $\Gamma_1 \vdash [\alpha]\varphi_1$ and $\Gamma_2 \vdash [\beta]\varphi_2$, respectively. Our objective is to automatically derive a proof for the sequent $\Gamma_1, \Gamma_2 \vdash [\alpha \otimes \beta]\varphi_1 \wedge \varphi_2$.

Whereas this is generally not possible, automatic derivation of a proof requires α and β not to share any bound variables. Having $BV(\alpha) \cap BV(\beta) = \varnothing$ means that α and β cannot modify the same variable. Otherwise, the execution of $\alpha \otimes \beta$ may make no sense and the sequent $\Gamma_1, \Gamma_2 \vdash [\alpha \otimes \beta]\varphi_1 \wedge \varphi_2$ may also be invalid.

Example 6 (Invalid property composition)**.** *Let* $\alpha = (x := 10)^*$ *and* $\beta = (x := 100)^*$ *be two hybrid programs. We can easily prove* $[\alpha](x \leqslant 15)$ *and* $[\beta](x \leqslant 100)$*, but not* $([\alpha \otimes \beta])(x \leqslant 15 \wedge x \leqslant 100)$*. As soon as the last loop iteration assigns* 100 *to* x*, the left-handside* $(x \leqslant 15)$ *of the formula does not hold anymore.*

This example provides us with a first necessary condition toward automated derivation of proof trees for the composition of components. Another condition is that variables occurring in φ_1 should not be bound in β as well. Indeed, this would mean for φ_1 to depend on the behavior of β, which doesn't seem to be a good design either. Last, we will make use of skolemization of formulas. That notion will be needed in the sequent calculus and used as a logical abstraction to ensure soundness of logical rules.

Definition 8. *We denote by* $\forall^\alpha \varphi$ *the* skolemization *of a formula with respect to the bound variables of its hybrid program* α*. (In the paper, we will use the shorter notation* φ^α *where unambiguous).*

Example 7. *Let* $\alpha \triangleq Engine \otimes Tachymeter$*. The skolemization by* α *of the formula* φ *is* $\varphi^\alpha = \forall s_{tach}\, a_{thro}, \varphi$.

Lemma 1 (Separation)**.** *Let* α *be a component and* φ *a formula such that* $BV(\alpha) \cap Var(\varphi) = \varnothing$*. Then the skolemization* φ^α *is equivalent to* φ*, i.e.* $\varphi^\alpha \leftrightarrow \varphi$.

Proof. Given a proof of φ^α, we can obtain a proof of φ. The reverse direction is a bit more difficult, the idea is that skolemized variables are not present in φ: the universal quantification is thus irrelevant. □

Based on the above, we can now present an important step regarding the composition of purely continuous programs.

Theorem 1. *Let* α *and* β *be two continuous components and assume that we have two proof trees of* $\Gamma_1 \vdash [\alpha]\varphi_1$ *and* $\Gamma_2 \vdash [\beta]\varphi_2$ *respectively. Furthermore, assume that*

(a) $BV(\alpha) \cap BV(\beta) = \varnothing$*,*
(b_1) $BV(\alpha) \cap Var(\varphi_2) = \varnothing$ *and*
(b_2) $BV(\beta) \cap Var(\varphi_1) = \varnothing$*.*

Then it exists a proof tree of $\Gamma_1, \Gamma_2 \vdash [\alpha \oplus \beta](\varphi_1 \wedge \varphi_2)$.

Theorem 1 makes intensive use of the separation between bound and output variables. The first assumption (a) assumes components to have separate internal variables and requires them to define disjoint output variables (i.e. unique definitions), which essentially amounts to good modeling practice. The second assumption (b_1) requires the safety property φ_1 to guard the behavior of the system α, i.e. its outputs, and of course not βs, it hence seems natural to require its separation with φ_2. The third equation (b_2) defines a symmetric assumption of separation between φ_1 and β.

Proof. Let α and β be two continuous components, of the form $\alpha \triangleq \dot{X} = \theta_X \,\&\, H_X$ and $\beta \triangleq \dot{Y} = \theta_Y \,\&\, H_Y$, and assume that we have a proof tree P_α of $\Gamma_1 \vdash [\alpha]\varphi_1$ and a proof tree P_β of $\Gamma_2 \vdash [\beta]\varphi_2$. The composition is still a continuous component and we want a proof tree of $\Gamma_1, \Gamma_2 \vdash [\alpha \otimes \beta](\varphi_1 \wedge \varphi_2)$ using the proofs tree P_α and P_β.

To ensure that such a tree exists, we inspect all the rules that can be applied in P_α to prove the sequent $\Gamma_1 \vdash [\alpha]\varphi_1$ (resp. for the proof tree P_β of $\Gamma_2 \vdash [\beta]\varphi_2$) and for each particular association, we provide a proof of $\Gamma_1, \Gamma_2 \vdash [\alpha \otimes \beta](\varphi_1 \wedge \varphi_2)$.

We will abbreviate sequents of the form $\Gamma \vdash [?\psi_X; \dot{X} = \theta_X]\varphi$ by $\Gamma, \psi_X \vdash [\dot{X} = \theta_X]\varphi$. Indeed, we can exchange between the first and second form by successive application of rules $(;)$, $(?)$ and $\rightarrow_r$.

$$\dfrac{\dfrac{P^{H_X}}{\Gamma_1,\Gamma_2,H_X,H_Y\vdash\varphi_1}\quad\dfrac{P^{H_Y}}{\Gamma_1,\Gamma_2,H_X,H_Y\vdash\varphi_2}\,\wedge_r}{\Gamma_1,\Gamma_2,H_X,H_Y\vdash\varphi_1\wedge\varphi_2}\qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{P'_\alpha}{\Gamma_1,\Gamma_2,H_X^\alpha,H_Y^\beta\vdash(\varphi_1'^\alpha)^{\theta_X}_{\dot X}}\quad\dfrac{P'_\beta}{\Gamma_1,\Gamma_2,H_X^\alpha,H_Y^\beta\vdash(\varphi_2'^\beta)^{\theta_Y}_{\dot Y}}\,\wedge_r}{\Gamma_1,\Gamma_2,H_X^\alpha,H_Y^\beta\vdash(\varphi_1'^\alpha)^{\theta_X}_{\dot X}\wedge(\varphi_2'^\beta)^{\theta_Y}_{\dot Y}}\,\text{(ii)}}{\Gamma_1,\Gamma_2,H_X^\alpha,H_Y^\beta\vdash(\varphi_1'^\alpha\wedge\varphi_2'^\beta)^{\theta_X\ \theta_Y}_{\dot X\ \dot Y}}\,{\to_r},{\wedge_l}}{\Gamma_1,\Gamma_2\vdash H_X^\alpha\wedge H_Y^\beta\to(\varphi_1'^\alpha\wedge\varphi_2'^\beta)^{\theta_X\ \theta_Y}_{\dot X\ \dot Y}}\,\text{(i)}}{\Gamma_1,\Gamma_2\vdash(H_X\wedge H_Y\to(\varphi_1'\wedge\varphi_2')^{\theta_X\ \theta_Y}_{\dot X\ \dot Y})^{\alpha\otimes\beta}}$$

$$\dfrac{\Gamma_1,\Gamma_2\vdash(H_X\wedge H_Y\to(\varphi_1'\wedge\varphi_2')^{\theta_X\ \theta_Y}_{\dot X\ \dot Y})^{\alpha\otimes\beta}}{\Gamma_1,\Gamma_2\vdash[\dot X,\dot Y=\theta_X,\theta_Y\,\&\,H_X\wedge H_Y](\varphi_1\wedge\varphi_2)}\,\text{(DI)}$$

Figure 1. Proof tree for the (DI) case

We can apply the following rules to the sequent $\Gamma_1 \vdash [\dot{X} = \theta_X]\varphi_1$: differential invariant (DI), ODE solution (ODESolve), differential weakening (DW), differential cut (DC), differential auxiliaries (DA), cut (Cut) and generalization (Gen). The rules differential cut, differential auxiliaries, cut and generalization do not decompose the ODE since we still have to prove $[\alpha]\varphi_1$ in one of the premises. For example, the rule cut applied to $\Gamma_1 \vdash [\alpha]\varphi_1$ leads to:

$$\dfrac{\Gamma_1,\Gamma_2\vdash C\qquad \Gamma_1,C\vdash[\dot X=\theta_X\,\&\,H_X]\varphi_1}{\Gamma_1\vdash[\dot X=\theta_X\,\&\,H_X]\varphi_1}\,\text{(Cut)}$$

The next goal is $[\dot{X} = \theta_X \,\&\, H_X]\varphi_1$. If one of these rules that does not decompose the ODE is applied in the proof tree P_α or P_β, we will apply it also in the general proof tree.

This careful inspection shows that there are only three rules to consider: (DI), (ODESolve) and (DW). We first treat the case for which $\Gamma_1 \vdash [\alpha]\varphi_1$ and $\Gamma_2 \vdash [\beta]\varphi_2$ are proved by the rule (DI). For the two other rules, (ODESolve) and (DW), we show how to build the proof tree using (DI) rule too.

(rule DI). If $\Gamma_1 \vdash [\alpha]\varphi_1$ and $\Gamma_2 \vdash [\beta]\varphi_2$ are proved by using rule (DI), then we have a proof tree of this form for P_α:

$$\dfrac{\dfrac{P^{H_X}}{\Gamma_1,H_X\vdash\varphi_1}\quad\dfrac{\dfrac{P'_\alpha}{\Gamma_1,H_X^\alpha\vdash(\varphi_1'^\alpha)^{\theta_X}_{\dot X}}}{\Gamma_1\vdash H_X^\alpha\to(\varphi_1'^\alpha)^{\theta_X}_{\dot X}}\,{\to_r}}{\Gamma_1\vdash[\dot X=\theta_X\,\&\,H_X]\varphi_1}\,\text{(DI)}$$

The notation $\varphi^{\theta_X}_{\dot X}$ stands for the formula φ where all the occurrences of $\dot{X}$ are replaced by θ_X. The left branch, P^{H_X}, corresponds to a proof that φ_1 holds at the initialisation. The right branch P'_α denotes the induction step. We have a similar proof tree for $\Gamma_2 \vdash [\beta]\varphi_2$. By reusing the branches P^{H_X}(resp. P^{H_Y}) and P'_α(resp. P'_β), we achieve the proof tree in Figure 1 for $\Gamma_1,\Gamma_2 \vdash [\dot{X},\dot{Y} = \theta_X,\theta_Y \& (H_X,H_Y)](\varphi_1 \wedge \varphi_2)$.

We close the branches P^{H_X} and P^{H_Y} in Figure 1, because we know that we already have some proofs for $\Gamma_1, H_X \vdash \varphi_1$ and $\Gamma_2, H_Y \vdash \varphi_2$. Similarly, we are able to complete the proofs of P'_α and P'_β.

Lemma 1 justifies the step (i). Indeed, the conditions (b1) and (b2) of Theorem 1 allows to apply the lemma and so ensures the soundness of this step. For step (ii), one has to remember that $\dot{X}$ does not occur in β (resp. $\dot{Y}$ does not occur in α), thanks to the condition (a). Then, we restrict the substitution of $\dot{X}$ by θ_X to the sole formula $\varphi_1'^\alpha$.

(rule ODESolve). We consider the case where $\Gamma_1 \vdash [\dot{X} = \theta_X \,\&\, H_X]\varphi_1$ is proved by the use of the rule (ODESolve), which means that we find an explicit solution w.r.t. time to the ODE and we replace each occurrence of X in φ_1 by this solution. We show that, given a proof using the rule (ODESolve), we can derive a proof using the differential invariant rule (DI). This leads us back to the previous situation. We do the proof with only one variable, X. It is easy to generalize it to a system of ODEs. Our reasoning is inspired from [10, p.247]. By hypothesis, we have the following rule for (ODESolve):

$$\dfrac{\Gamma_1\vdash\forall t\geqslant 0(\forall 0\leqslant\tilde t\leqslant t,(H_X)^{y(\tilde t)}_X)\to(\varphi_1)^{y(t)}_X}{\Gamma_1\vdash[\dot X=\theta_X\,\&\,H_X]\varphi_1}$$

$y(t)$ is the solution of the ODE $\dot{X} = \theta_X$. Let us introduce a fresh variable t to stand for time in the ODE. It will be evaluated after the rule (DI) by using the differential auxiliaries rule (DA).

$$\dfrac{\Gamma_1\vdash[\dot X=\theta_X,\dot t=1\,\&\,H_X]\varphi_1}{\Gamma_1\vdash[\dot X=\theta_X\,\&\,H_X]\varphi_1}\,\text{(DA)}$$

Also, the solution of the ODE shall contain an occurrence of the initial value. To remember it, we use the auxiliary variable rule (IA).

$$\dfrac{\Gamma_1\vdash[X_0:=X][\dot X=\theta_X,\dot t=1\,\&\,H_X]\varphi_1}{\Gamma_1\vdash[\dot X=\theta_X,\dot t=1\,\&\,H_X]\varphi_1}\,\text{(IA)}$$

The proof tree of $\Gamma_1 \vdash [\alpha]\varphi_1$ using the rule (DI) is in Figure 2. To introduce the solution as an invariant, we use the generalization rule (Gen).

$$\dfrac{\dfrac{}{\Gamma_1, H_X \vdash X = y(0)}\,\text{ax} \qquad \dfrac{\dfrac{\dfrac{}{\Gamma_1, H_X \vdash \theta_X = \theta_X}\,\text{ax}}{\Gamma_1, H_X \vdash (X = y(t))^{\theta_X\ 1}_{\dot{X}\ \dot{t}}}\,\text{(i)}}{\Gamma_1 \vdash H_X \to (X = y(t))^{\theta_X\ 1}_{\dot{X}\ \dot{t}}}\,{\to_r}}{\Gamma_1 \vdash \forall X, H_X \to (X = y(t))^{\theta_X\ 1}_{\dot{X}\ \dot{t}}}\,\forall_r}{\Gamma_1 \vdash [\dot{X} = \theta_X, \dot{t} = 1 \,\&\, H_X]X = y(t)}\,\text{(DI)} \qquad \dfrac{\dfrac{\dfrac{\Pi_1}{\Gamma_1, X^0 = y(t^0) \vdash \varphi_1^0}}{\Gamma_1 \vdash X^0 = y(t^0) \to \varphi_1^0}\,{\to_r}}{\Gamma_1 \vdash \forall X \forall t, X = y(t) \to \varphi_1}\,\forall_r$$
$$\Gamma_1 \vdash [\dot{X} = \theta_X, \dot{t} = 1 \,\&\, H_X]\varphi_1\,\text{(Gen)}$$

Figure 2. Proof tree for the (ODESolve) case

By hypothesis, we assume a proof P of $\vdash \varphi_X^{y(t^0)}$, which is derived from the goal $\forall t \geqslant 0(\forall 0 \leqslant \tilde{t} \leqslant t, (H_X)_X^{y(\tilde{t})}) \to (\varphi_1)_X^{y(t)}$ by a straightforward application of the rules $\forall_r$ and $\to_r$. Here, t^0, X^0 and φ^0 are fresh variables consequently introduced by the rule $\forall_r$. for the branch Π_1, we apply the same rules than for P. There is no matter with the logical connectives, since we build proof tree having a logical structure which is similar to between the assumed proof tree. However, arithmetic over reals may be more technical. Let's assume that we have proved $y(t^0) \leqslant \theta_X$ in P. We need to prove $X^0 \leqslant \theta$ in the proof of φ_1^0 (resulting from the rule $\forall_r$ in the most-right branch in Figure 2). Fortunately, we know that $X^0 = y(t)$ by definition: the conclusion is immediate.

For the left premise of rule (DI), we should remember that it is here to prove the initial condition: the property at time $t = 0$, i.e. to ensure that the initials conditions prove the (inductive) property. Since the value of X is exactly $y(0)$, we conclude with the axiom rule. For the right premise, we perform the substitution and the step (i) is justified by the same reasoning as for (DI).

(rule DW). We still need to prove the differential weakening case. The proofs are of the following form:

$$\dfrac{\dfrac{P^\alpha}{\Gamma_1 \vdash (H_X \to \varphi_1)^\alpha}}{\Gamma_1 \vdash [\dot{X} = \theta_X \,\&\, H_X]\varphi_1}\,\text{(DW)}$$

This rule is used when the evolution domain characterization is sufficient to deduce the goal.

If $\Gamma_1 \vdash [\alpha]\varphi_1$ and $\Gamma_2 \vdash [\beta]\varphi_2$ are both proved by the mean of differential weakening, then the proof of the composition is the following:

$$\dfrac{\dfrac{\dfrac{P^\alpha}{\Gamma_1, \Gamma_2, H_X^\alpha, H_Y^\beta \vdash \varphi_1^\alpha} \qquad \dfrac{P^\beta}{\Gamma_1, \Gamma_2, H_X^\alpha, H_Y^\beta \vdash \varphi_2^\beta}\,{\wedge_r}}{\Gamma_1, \Gamma_2 \vdash H_X^\alpha \wedge H_Y^\beta \to \varphi_1^\alpha \wedge \varphi_2^\beta}\,\text{(i)}}{\Gamma_1, \Gamma_2 \vdash (H_X \wedge H_Y \to \varphi_1 \wedge \varphi_2)^{\alpha \otimes \beta}}\,\text{(DW)}$$
$$\Gamma_1, \Gamma_2 \vdash [\alpha \oplus \beta]\varphi_1 \wedge \varphi_2$$

It is a very simple case, where the evolution domains are sufficient to prove the safety properties. The step (i) is the same as when we have proved the theorem for the rule (DI).

Now, let's assume that $\Gamma_1 \vdash [\alpha]\varphi_1$ have been proved with the rule (DI) and $\Gamma_2 \vdash [\beta]\varphi_2$ with the rule (DW). The idea is to add φ_1 into the evolution domain of $\alpha \otimes \beta$ using the differential cut rule (DC).

$$\dfrac{\dfrac{}{\Gamma \vdash [\ldots \,\&\, H]\varphi_1}\,\text{(DI)} \qquad \dfrac{}{\Gamma \vdash [\ldots \,\&\, H \wedge \varphi_1]\varphi_1 \wedge \varphi_2}\,\text{(DW)}}{\Gamma_1, \Gamma_2 \vdash [\ldots \,\&\, H_X \wedge H_Y]\varphi_1 \wedge \varphi_2}\,\text{(DC)}$$

Thanks to the condition (b2), the component β cannot impact the validity of φ_1: the first premise amounts to prove the sequent $\Gamma_1 \vdash [\alpha]\varphi_1$. The second premise is proved by using the rule (DW).

□

We have proved our theorem only for the continuous case. The next result extends it to the general form of hybrid discrete/continuous specifications.

Theorem 2. *Let us assume that we have a proof tree of* $\Gamma_1 \vdash [\alpha]\varphi_1$ *and* $\Gamma_2 \vdash [\beta]\varphi_2$*, for* α *and* β *components of the general form. Furthermore, assume that*

(a) $BV(\alpha) \cap BV(\beta) = \varnothing$,
(b_1) $BV(\alpha) \cap Var(\varphi_2) = \varnothing$,
(b_2) $BV(\beta) \cap Var(\varphi_1) = \varnothing$.

Then we have an automatic derivation of $\Gamma_1, \Gamma_2 \vdash [\alpha \otimes \beta](\varphi_1 \wedge \varphi_2)$.

Because Theorem 2 is applied in the conditions than Theorem 1, it relies on the same assumptions.

Proof. We have proved our result for the case of the composition of two continuous component thanks to Theorem 1. Next, we focus the general case including discrete behaviors.

Let's assume $\alpha \doteq [(\alpha_0 \cup \ldots \cup \alpha_n \cup \textbf{cont}_\alpha)^*]$ and $\beta \doteq [(\beta_0 \cup \ldots \cup \beta_m \cup \textbf{cont}_\beta)^*]$ are two hybrid programs respecting the conditions above. Let $\alpha \otimes \beta$ be their composition. As in Theorem 1, giving some proofs for the sequents $\Gamma_1 \vdash [\alpha]\varphi_1$ and $\Gamma_2 \vdash [\beta]\varphi_2$, we want to prove the sequent for the system obtained by composition, i.e. to prove $\Gamma_1, \Gamma_2 \vdash [\alpha \otimes \beta]\varphi_1 \wedge \varphi_2$.

First, if we have proved $\Gamma_1 \vdash [(\alpha_0 \cup \ldots \cup \alpha_n \cup \textbf{cont}_\alpha)^*]\varphi_1$, we also have a proof of $[\alpha_i]\varphi_1, i \in \{0, n\}$. Indeed, we just look up in the proof tree of $\Gamma_1 \vdash [\alpha]\varphi_1$ until the application of the $(\cup)$ rule with the goal $[\alpha_i]\varphi_1$ in premise.

$$\dfrac{\dfrac{\Pi_3}{\Gamma \vdash \varphi_1 \wedge \varphi_2} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\Pi_4}{\Gamma, \varphi_1^\alpha, \varphi_2^\beta \vdash [\alpha_0]\varphi_1^\alpha} \qquad \dfrac{\dfrac{\Gamma, \varphi_1^\alpha, \varphi_2^\beta, (\varphi_1^\alpha)^{\alpha_0} \vdash (\varphi_1^\alpha)^{\alpha_0} \wedge (\varphi_2^\beta)^{\alpha_0}}{\Gamma, \varphi_1^\alpha, \varphi_2^\beta \vdash (\varphi_1^\alpha)^{\alpha_0} \to (\varphi_1^\alpha)^{\alpha_0} \wedge (\varphi_2^\beta)^{\alpha_0}}\,\forall_r}{\Gamma, \varphi_1^\alpha, \varphi_2^\beta \vdash \forall^{\alpha_0}(\varphi_1^\alpha) \to (\varphi_1^\alpha) \wedge (\varphi_2^\beta)}\,\text{(Gen)}}{\Gamma, \varphi_1^\alpha, \varphi_2^\beta \vdash [\alpha_0]\varphi_1^\alpha \wedge \varphi_2^\beta}\,\Pi_5}{\Gamma, \varphi_1^\alpha, \varphi_2^\beta \vdash [\_]\varphi_1^\alpha \wedge \varphi_2^\beta}\,(\cup)}{\Gamma \vdash (\varphi_1 \wedge \varphi_2)^{\alpha\otimes\beta} \to [\_](\varphi_1 \wedge \varphi_2)^{\alpha\otimes\beta}}\,\to_r}{\Gamma \vdash \forall^{\alpha\otimes\beta}(\varphi_1 \wedge \varphi_2 \to [\_]\varphi_1 \wedge \varphi_2)}\,\forall_r}{\Gamma \vdash [(\alpha_0 \cup \ldots \cup \alpha_n \cup \beta_0 \cup \ldots \cup \beta_m \cup \mathbf{cont}_{\alpha\otimes\beta})^*]\varphi_1 \wedge \varphi_2}\,\text{(Ind)}$$

Figure 3. Proof tree for the general form

To close the branch Π_3 in Figure 3, we need a proof of both $\Gamma_1 \vdash \varphi_1$ and $\Gamma_2 \vdash \varphi_2$. When we look at the proof of $\Gamma_1 \vdash [\alpha]\varphi_1$, it has the following form:

$$\dfrac{\dfrac{\vdots}{\Gamma_1 \vdash \varphi_1} \qquad \ldots}{\Gamma_1 \vdash [\alpha]\varphi_1}\,\text{(Ind)}$$

We already have a proof of $\Gamma_1 \vdash \varphi_1$. We do exactly the same for $\Gamma_2 \vdash \varphi_2$ which concludes this part.

The last point closes the branch Π_4. From this proof, we show how to extract a proof tree for each discrete subpart in the proof tree for a general component. By repeating the discrete subpart isolation (using the rule $(\cup)$) followed the extraction approach (same as for Π_4), we easily complete the proof for the whole discrete part. □

5. Methodology for a use case

We have a notion of component and contract. We also have a syntactic composition operator and a theorem which give us a proof of the contract of a composition given the proofs of the contracts of each components, under some conditions. To illustrate our theory and to present our methodology, we studied a cruise controller system.

5.1. Presentation of the cruise controller

This example is inspired from [3]. We present it following the A-D (Analogic-Digital) paradigm of the control theory, widely used in industrial contexts [11]. This representation of the system is easily achieved thanks to the associativity of our composition operator.

The typical modeling of system in control theory is based on the analogic digital paradigm. The general scheme is presented in Figure 4. It is composed of a D-A (digital-to-analog) component, called actuator, and an A-D (analog-to-digital) component, called sensor, which are interfaces between the physical world and the digital world. The control part monitors the system and decides actions to perform, according to the data collected from the physical (continuous) part by the sensor.

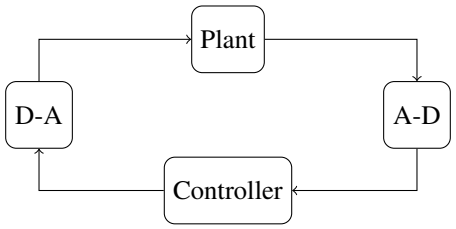


Figure 4. A-D paradigm

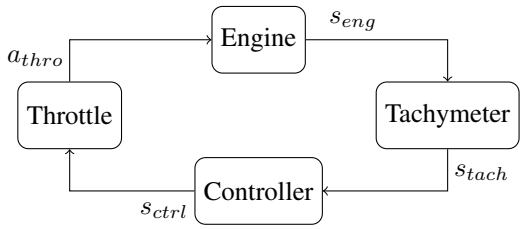


Figure 5. Cruise controller system

Figure 5 shows how we instantiate the paradigm for a cruise controller system. It regulates the speed of a vehicle as the following: the vehicle's speed s_{eng} obeys to the differential equation $\dot{s_{eng}} = a_{thro}$, where a_{thro} is the acceleration. The tachymeter measures the actual speed at least every ε units of time and saves it as s_{tach}. Then, the controller chooses the speed order, s_{ctrl}, *i.e.* the maximum speed reachable by the vehicle before the next sampling. The controller also checks that the vehicle does not overcome the maximum speed allowed, S. According to the speed order, the throttle modifies the acceleration that the vehicle applies during the next period.

Whereas our model perfectly fits the modeling scheme of the control theory, directly applying our composition operator with these components does not result the model as we would expect. For instance, we implicitly suppose that the controller executes after the sensor such that the controller takes decision once the tachymeter measured the speed. We need to add sequentialization in the model, whereas our composition operator allows the execution of the controller in parallel of the tachymeter. To solve this

problem, we present a way to synchronize components using a notification mechanism.

5.2. Sequencing using notifications

The cruise controller is executed during the time period ε. The model execution is based on the infinite repetition of this execution cycle. In order to define a temporal reference, we introduce the variable t denoting the time elapsed in the current cycle. Intuitively, t can evolve in the time interval $[0, \varepsilon]$.

For sequencing every component of the cruise system, we need to identify the time when every component has been executed. This time is saved into a fresh variable. For instance, once the tachymeter has been executed, we set the variable t_{tach} to the current time t. This is done by appending the statement $t_{tach} := t$ to the specification of the tachymeter: $(s_{tach} := s_{eng}; t_{tach} := t)^*$. Similarly, we introduce the variables t_{ctrl} and t_{thro} for the controller and the throttle respectively.

We have a mechanism to notify when a component was executed. Then, we need a mechanism to trigger the next executable component according to the sequencing. For instance, the controller is the next component to execute after the tachymeter. We prefix its specification by a guard checking that the tachymeter has just been executed, *i.e.* $t_{tach} = t$. We obtain the following specification for the controller: $(?t_{tach} = t; s_{ctrl} := *; ?(0 \leqslant s_{ctrl} \leqslant S \wedge |s_{ctrl} - s_{tach}| \leqslant \delta); t_{ctrl} := t)^*$. Because the reachability semantics of hybrid programs in $d\mathcal{L}$, the guard blocks the execution of the controller until it is is satisfied. Similarly, the throttle is executed, once both tachymeter and controller are executed. So the throttle specification is prefixed by the guard $?(t = t_{tach} \wedge t = t_{ctrl})$.

To ensure the interleaving between the continuous part and the digital part, we introduce a guard of the form $?(t > t_{tach} + \varepsilon_0)$ where ε_0 is the minimal execution time of the cruise controller, which cannot be instantaneous. As consequence, the vehicle engine is free to run at least during ε_0 without any interaction from the cruise controller. This guard is added to the starting component of system: the sensor specification is finally $(?(t > t_{tach} + \varepsilon_0); s_{tach} := s_{eng}; t_{tach} := t)^*$.

Following this approach we are now able to provide sequencing of our components that is compatible with our composition framework.

5.3. The contracts for components and composition

We associate contracts to components and induce the contracts resulting for their composition.

The engine contract.

$$C_{eng} = \begin{cases} A_{eng}: & 0 \leqslant s_{tach} \leqslant S \wedge 0 \leqslant s_{ctrl} \leqslant S \\ G_{eng}: & 0 \leqslant s_{eng} \leqslant S \end{cases}$$

To obtain the $d\mathcal{L}$ sequent representing this component, we have to add the characterization of parameters like $S > 0$, $0 < \varepsilon$. The behavior of engine is specified as:

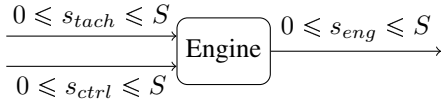
$$\begin{aligned} \text{Engine} \triangleq ?(a_{thro} = \tfrac{s_{ctrl} - s_{tach}}{\varepsilon} \wedge s_{eng} = s_{tach}); \\ \dot{s_{eng}} = a_{thro} \,\&\, (t - t_{tach} \leqslant \varepsilon \\ \wedge\ t - t_{ctrl} \leqslant \varepsilon \\ \wedge\ t - t_{thro} \leqslant \varepsilon) \end{aligned}$$

We obtain the following sequent:

$$\begin{aligned} & 0 \leqslant s_{tach} \leqslant S \wedge 0 \leqslant s_{ctrl} \leqslant S \\ & \wedge\ t = t_{tach} \wedge t = t_{ctrl} \wedge t = t_{thro} \\ & \wedge\ \varepsilon > 0 \wedge S > 0 \wedge s_{eng} = s_{tach} \end{aligned} \vdash [\text{Engine}] G_{eng}$$

The proof is automatically done thanks to the prover KeYmaera X. We give a possible sketch to manually prove it in KeYmaera X. We first apply the rule $(?)$ thus, the rule $(;)$ to treat the guard. We have to reason on the ODE. We notice it is a linear ODE; providing a solution is easy. We put this solution $(s_{eng} = t * a_{thro})$ in the ODE evolution domain using the differential cut rule. We apply a differential weakening rule, *i.e.* that the formulas in the evolution domain are sufficient to prove the safety conditions. At this point, it remains one goal which is a formula of the real arithmetic. It can be discharged by automatic reasoning.

We provide below a representation of the contracts on inputs and outputs.



The tachymeter contract.

$$C_{tach} = \begin{cases} A_{tach}: & 0 \leqslant s_{eng} \leqslant S \\ G_{tach}: & 0 \leqslant s_{tach} \leqslant S \end{cases}$$

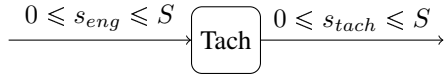
The contract is associated to the component tachymeter specified as the following:

$$\text{Tach} = (?t > t_{tach} + \varepsilon_0; s_{tach} := s_{eng}; t_{tach} := t)^*$$

By combining the specification and its contract, we build the corresponding sequent:

$$A_{tach} \wedge \varepsilon_0 > 0 \wedge S > 0 \wedge s_{tach} = 0 \vdash [\text{Tach}] G_{tach}$$

KeYmaera X is also able to automatically prove this sequent. The proof is indeed straightforward where the formula $0 \leqslant s_{tach} \leqslant S$ can be directly used as a loop invariant for the rule (Ind). After applying the corresponding hybrid programs rules, it leads to a rewriting of s_{tach} by s_{eng} in the $0 \leqslant s_{tach} \leqslant S$ which becomes the hypothesis provided by A_{tach}.



By composing the engine and the tachymeter, we obtain the following component:

$$\text{Engine} \otimes \text{Tach} \triangleq (\mathbf{disc}_{\text{Tach}} \cup \mathbf{cont}_{\text{Engine}})^*$$

The composition contract. The composition result, summarized in 6, can be considered as new component which can be composed with other component, the controller and the throttle.

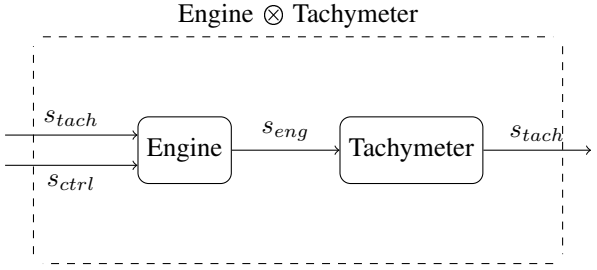


Figure 6. Composition of the engine and the tachymeter

The contract of the resulting component is the following

$$C_{\otimes} \left\{ \begin{array}{ll} A: & A_{eng} \wedge A_{tach} \\ G: & G_{eng} \wedge G_{tach} \end{array} \right.$$

Similarly, by composing it successively with the controller and the throttle, we are able to produce the contract for the complete system because their respective contracts are also compatible with Theorem 2. Providing that controller and throttle are respectively mapped to the contracts

$$C_{Controller} \left\{ \begin{array}{ll} A_{ctrl}: & 0 \leqslant s_{tach} \leqslant S \\ G_{ctrl}: & 0 \leqslant s_{ctrl} \leqslant S \end{array} \right.$$

and

$$C_{Throttle} \left\{ \begin{array}{ll} A_{thro}: & \top \\ G_{thro}: & \top \end{array} \right.$$

the whole contract is:

$$C_{Cruise} \left\{ \begin{array}{ll} A: & A_{eng} \wedge A_{tach} \wedge A_{ctrl} \wedge A_{thro} \\ G: & G_{eng} \wedge G_{tach} \wedge G_{ctrl} \wedge G_{thro} \end{array} \right.$$

and there is a proof of it thank to Theorem 2.

6. Conclusion, future work

We have presented a syntactic composition operator for differential dynamic logic $d\mathcal{L}$. We have proved associativity and commutativity properties about this composition operator together with a theorem on proof compositionality by means of component contracts. To finish, we have exemplified our methodology with the detailed case study of a cruise controller system. This allowed us to evaluate our theoretical expectations in a practical context.

We believe that the composition operator we propose for $d\mathcal{L}$ offers promising applications. Additionally, we already identified some possible ways to relax some of the syntactic limitations of Theorem 2 to make it of more versatile usability. By evaluating our framework on more complex use cases, we identified some additional leverages to tackle. They are essentially related to explicitly consider time in component interactions. Integrating the execution cycle issued from the control theory will strongly increase the application impact of Theorem 2: it will permit to prove contracts that are more intimately related to the timed behavioral properties of components.

From a theoretical point of view, design by composition is hard to scale up, because each composition step build new contracts by merging contracts of the sub-components: for large-systems the contracts computed at top level may hence become huge. Refinement and abstraction mechanisms will be essential tools to tackle those issues, filtering relevant properties from lower-level to higher-level contracts. They will bring more flexibility in the design phase and hopefully reduce the effort needed to prove system.

Future Work

To establish our results, we navigate between the theory and the practice thanks to our use case. It permit to raise a lot of troubles, but also to ask interesting questions. Some of them are still not solved, and are good lines for future research.

- We plan to inquire a system of contracts on top of $d\mathcal{L}$ logic.
- We need a way to perform refinement between systems, i.e. a vertical mechanism in our design methodologies. It will bring more flexibility in the design phase, and hopefully also in the proof phase. It should be natural to have such mechanism since our original idea is taken from Action Systems.

References

[1] A. Platzer, "Differential dynamic logic for hybrid systems." *J. Autom. Reas.*, vol. 41, no. 2, pp. 143–189, 2008.

[2] V. R. Pratt, "Dynamic logic," *Studies in Logic and the Foundations of Mathematics*, vol. 104, pp. 251–261, 1982.

[3] A. Müller, S. Mitsch, W. Retschitzegger, W. Schwinger, and A. Platzer, "A component-based approach to hybrid systems safety verification," in *IFM*, ser. LNCS, E. Abraham and M. Huisman, Eds., vol. 9681. Springer, 2016, pp. 441–456.

[4] R.-J. Back and J. Wright, *Refinement calculus: a systematic introduction*. Springer Science & Business Media, 1998.

[5] M. Rönkkö and A. P. Ravn, "Action systems with continuous behaviour," in *International Hybrid Systems Workshop*. Springer, 1997, pp. 304–323.

[6] M. Rönkkö, A. P. Ravn, and K. Sere, "Hybrid action systems," *Theoretical Computer Science*, vol. 290, no. 1, pp. 937–973, 2003.

[7] R. Banach, M. Butler, S. Qin, N. Verma, and H. Zhu, "Core hybrid event-b i: single hybrid event-b machines," *Science of Computer Programming*, vol. 105, pp. 92–123, 2015.

[8] A. Platzer, *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*. Heidelberg: Springer, 2010.

[9] ——, "Cheat sheet of rules in keymaera," http://symbolaris.com/info/KeYmaera-cheat.pdf.

[10] ——, "Foundations of cyber-physical systems," Lecture Notes 15-424/624, Carnegie Mellon University, 2014. [Online]. Available: http://symbolaris.com/course/fcps14/fcps14.pdf

[11] K. J. Åström and B. Wittenmark, *Computer-controlled systems: theory and design*. Courier Corporation, 2013.

[12] A. Müller, S. Mitsch, and A. Platzer, "Verified traffic networks: component-based verification of cyber-physical flow systems," in *2015 IEEE 18th International Conference on Intelligent Transportation Systems*. IEEE, 2015, pp. 757–764.

[13] S. M. Loos and A. Platzer, "Differential refinement logic," in *LICS*. ACM, 2016.

[14] A. Platzer and J.-D. Quesel, "European Train Control System: A case study in formal verification," in *ICFEM*, ser. LNCS, K. Breitman and A. Cavalcanti, Eds., vol. 5885. Springer, 2009, pp. 246–265.

[15] A. Platzer, "Logics of dynamical systems," in *LICS*. IEEE, 2012, pp. 13–24.

[16] ——, "The complete proof theory of hybrid systems," in *LICS*. IEEE, 2012, pp. 541–550.

[17] S. Mitsch, J.-D. Quesel, and A. Platzer, "Refactoring, refinement, and reasoning," in *International Symposium on Formal Methods*. Springer, 2014, pp. 481–496.

[18] N. Fulton, S. Mitsch, J.-D. Quesel, M. Völp, and A. Platzer, "Keymaera x: An axiomatic tactical theorem prover for hybrid systems," in *International Conference on Automated Deduction*. Springer, 2015, pp. 527–538.

[19] M. Rönkkö and K. Sere, "Refinement and continuous behaviour," in *International Workshop on Hybrid Systems: Computation and Control*. Springer, 1999, pp. 223–237.

[20] A. Sangiovanni-Vincentelli, W. Damm, and R. Passerone, "Taming dr. frankenstein: Contract-based design for cyber-physical systems," *European journal of control*, vol. 18, no. 3, pp. 217–238, 2012.

[21] R. Back and J. von Wright, *Trace refinement of action systems*. Springer, 1994.

[22] R.-J. Back, L. Petre, and I. Porres, "Continuous action systems as a model for hybrid systems," *Nord. J. Comput.*, vol. 8, no. 1, pp. 2–21, 2001.

[23] M. Rönkkö and X. Li, "Linear hybrid action systems," *Nordic Journal of Computing*, vol. 8, no. 1, pp. 159–177, 2001.

[24] A. Fehnker and F. Ivančić, "Benchmarks for hybrid systems verification," in *Hybrid Systems: Computation and Control*. Springer, 2004, pp. 326–341.

[25] A. Benveniste, D. Nickovic, and T. Henzinger, "Compositional Contract Abstraction for System Design," INRIA, Research Report, 2014. [Online]. Available: https://hal.inria.fr/hal-00938854

[26] I. Ruchkin, D. De Niz, S. Chaki, and D. Garlan, "Contract-based integration of cyber-physical analyses," in *Embedded Software (EMSOFT), 2014 International Conference on*. IEEE, 2014, pp. 1–10.

[27] S. Gao, S. Kong, and E. M. Clarke, "dreal: An smt solver for nonlinear theories over the reals," in *Automated Deduction–CADE-24*. Springer, 2013, pp. 208–214.

[28] D. Ricketts, G. Malecha, M. M. Alvarez, V. Gowda, and S. Lerner, "Towards verification of hybrid systems in a foundational proof assistant," in *Formal Methods and Models for Codesign (MEMOCODE), 2015 ACM/IEEE International Conference on*. IEEE, 2015, pp. 248–257.

[29] J.-R. Abrial, W. Su, and H. Zhu, "Formalizing hybrid systems with event-b," in *International Conference on Abstract State Machines, Alloy, B, VDM, and Z*. Springer, 2012, pp. 178–193.

[30] L. Lamport, "Hybrid systems in tla+," in *Hybrid Systems*. Springer, 1993, pp. 77–102.