AVACS – Automatic Verification and Analysis of Complex Systems

REPORTS

of SFB/TR 14 AVACS

Editors: Board of SFB/TR 14 AVACS

Differential Dynamic Logic for Verifying Parametric Hybrid Systems

by
André Platzer

AVACS Technical Report No. 15
May 2007
ISSN: 1860-9821

Publisher: Sonderforschungsbereich/Transregio 14 AVACS
(Automatic Verification and Analysis of Complex Systems)
Editors: Bernd Becker, Werner Damm, Martin Fränzle, Ernst-Rüdiger Olderog,
Andreas Podelski, Reinhard Wilhelm
ATRs (AVACS Technical Reports) are freely downloadable from `www.avacs.org`

Author(s) contact: André Platzer (`andre.platzer@informatik.uni-oldenburg.de`).

Differential Dynamic Logic for Verifying Parametric Hybrid Systems*

André Platzer

University of Oldenburg, Department of Computing Science, Germany
Carnegie Mellon University, Computer Science Department, Pittsburgh, PA
platzer@informatik.uni-oldenburg.de

Abstract. We introduce a first-order dynamic logic for reasoning about systems with discrete and continuous state transitions, and we present a sequent calculus for this logic. As a uniform model, our logic supports hybrid programs with discrete and differential actions. For handling real arithmetic during proofs, we lift quantifier elimination to dynamic logic. To obtain a modular combination, we use side deductions for verifying interacting dynamics. With this, our logic supports deductive verification of hybrid systems with symbolic parameters and first-order definable flows. Using our calculus, we prove a parametric inductive safety constraint for speed supervision in a train control system.

Keywords: dynamic logic, sequent calculus, verification of parametric hybrid systems, quantifier elimination

1 Introduction

Frequently, correctness of a real-time or hybrid system [16] depends on the choice of parameters [9, 12, 25]. Such parameters naturally arise from the degrees of freedom of how a part of the system can be instantiated or how a controller can respond to input. They include both external system parameters like the braking force of a train, and control parameters of internal choice like when to start braking before approaching an open gate or a preceding train [9].

Symbolic parameters occurring in system dynamics raise a couple of challenges. Even simple parametric flows and guards are *non-linear*: With parameter b, the flow constraint $2x + by \geq 0$ is an algebraic inequality but not linear. For this reason, we cannot use approaches with linear arithmetic like the model checkers HyTech [1], or PHAVer [14]. More generally, Davoren and Nerode [11] argue that, unlike deductive methods, model checking [1, 7, 14, 16] does not support free parameters. Furthermore, correctness of parametric systems typically depends on a constraint on the free parameters, which is not always known a

* This research was supported by a fellowship of the German Academic Exchange Service (DAAD) and by the German Research Council (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS, see www.avacs.org). A shorter version [21] of this report appeared at Tableaux 2007.

priori but needs to be identified during the analysis. It is, however, quite complicated to synthesise such general symbolic constraints from the concrete values of a counterexample trace produced by a model checker. Even without parameters, there are limitations of non-symbolic techniques for handling continuous flows [25].

To overcome these issues, we propose a fully symbolic technique following a deductive approach. We introduce a logic for verifying hybrid systems with parameters. As a basis we use first-order logic, which has widely proven its power and flexibility in handling symbolic parameters as logical variables. For reasoning about state transitions, our logic further extends dynamic logic.

First-order dynamic logic (DL) [15] is a successful approach for reasoning about (discrete) state changes [3, 4, 15, 18]. Like model checking, first-order DL can analyse the behaviour of operational system models [22, 23]. Yet, DL calculi accept parameters: they verify systems by deductive proof rather than a more enumerative and graph-theoretic analysis of the (abstract) state space as in model checking [7]. In addition, operational models are internalised, and DL is closed under logical operators. Thus, DL can analyse the relationship between multiple systems [22, 24], which is useful for compositional verification.

Since hybrid systems are subject to both continuous evolution and discrete state change, we add continuous state changes to discrete DL. As a uniform model for hybrid systems, our logic introduces hybrid programs with discrete assignments and differential actions. The resulting first-order dynamic logic is called differential dynamic logic ($\mathsf{d}\mathcal{L}$). It has been motivated and proposed (without a formal introduction) in our preliminary work [22].

In [23, 24], we have introduced logics that extend the basic ideas of [22] into different directions. In [24], we have presented a logic with nominals to investigate compositionality. In [23], we have introduced a temporal logic and a calculus that reduces temporal statements to non-temporal formulas. The calculi of [23, 24] reduce their respective extensions to $\mathsf{d}\mathcal{L}$. In this paper, we give a proof system for $\mathsf{d}\mathcal{L}$ itself, which can be combined with the extensions in [23, 24].

We show how interacting discrete and continuous dynamics can be verified *constructively* by integrating quantifier elimination into our calculus to handle the resulting arithmetic. Moreover, this combination is modular in the sense that we directly combine quantifier elimination and dynamic logic side by side. Using side deductions, we achieve such a modular combination even though both quantifiers and dynamic modalities interact by affecting the values of variables.

As an important modelling characteristic of hybrid systems, we generalise differential actions to differential equations that can be restricted to first-order invariant regions. Moreover, we show how induction can be integrated.

The first contribution of this paper is a formal introduction of our logic $\mathsf{d}\mathcal{L}$. The main contribution is a *full* calculus for verifying *interacting* discrete and continuous dynamics including the resulting arithmetic. For this, we present a modular combination of quantifier elimination with a sequent calculus. Using our calculus, we prove safety of speed supervision in train control [9, 12] and synthesise the required parametric induction invariant.

Hybrid Systems. The behaviour of safety-critical systems typically depends on both the state of a discrete controller and continuous physical quantities. Hybrid systems are mathematical models for dynamic systems with interacting discrete and continuous behaviour [11,16]. Their behaviour combines continuous evolution (called *flow*) characterised by differential equations and discrete jumps.

Dynamic Logic. The principle of dynamic logic is to combine system operations and correctness statements about system states within a single specification language (see [15] for a general introduction for discrete systems). By permitting system operations α as actions of a labelled multi-modal logic, dynamic logic provides formulas of the form $[\alpha]\phi$ and $\langle\alpha\rangle\phi$, where $[\alpha]\phi$ expresses that all (terminating) runs of system α lead to states in which condition ϕ holds. Likewise, $\langle\alpha\rangle\phi$ expresses that there is at least one (terminating) run of α after which ϕ holds. In $\mathsf{d}\mathcal{L}$, hybrid programs play the role of α. In particular, $\mathsf{d}\mathcal{L}$ extends discrete dynamic logic [15] such that α can display continuous evolution.

Related Work. Several approaches [2, 13, 19, 20] use quantifier elimination [8] in first-order real arithmetic for model checking hybrid automata. Thus, we use the same decision procedure as a basis for handling arithmetic of non-linear flows. We generalise these results to a deductive calculus for improved handling of free parameters. As a more uniform model that is amenable to compositional symbolic processing by calculi, we use hybrid programs rather than automata.

Zhou et al. [28] extended duration calculus with mathematical expressions in derivatives of state variables. They use a multitude of rules and an oracle that requires external mathematical reasoning about derivatives and continuity.

Rönkkö et al. [26] presented a guarded command language with differential relations and gave a semantics in higher-order logic with built-in derivatives. Without providing a means for verification of this higher-order logic, the approach is still limited to providing a notational variant of classical mathematics.

Rounds [27] defined a semantics in set theory of a "spatial" logic for a hybrid π-calculus. Without giving a calculus for that logic, this approach is not suitable for verification yet. Further, the automatic proving potential is limited by the large number of very expressive operators in this formalism.

Boulton et al. [6] introduced a Hoare calculus for gain and phase shift properties of a special case of block diagrams. It requires manual reasoning about complicated expressions with square roots, rational and trigonometric functions. The authors do not give a soundness result or formal semantics.

Davoren and Nerode [10, 11] extended the propositional modal μ-calculus with a semantics in hybrid systems and examine topological aspects. They provided Hilbert-style calculi to prove formulas that are valid for all hybrid systems simultaneously. Thus, only limited information can be obtained about a particular system: In propositional modal logics, system behaviour needs to be axiomatised in terms of abstract actions a, b, c of *unknown effect*, see, e.g. [22].

The strength of our logic primarily is that it is a *first-order* dynamic logic: It handles actual operational models of hybrid systems like $x := x + y; \dot{x} = 2y$

rather than abstract propositional actions of unknown effect. Further, we provide a calculus for actually verifying hybrid programs with free parameters and first-order definable flows, i.e., flows that are definable in first-order arithmetic. For verifying the coordination level of train dynamics, which we use as an example, first-order definable flows are sufficient [9]. First-order approximations of more general flows can be used according to [2, 25].

Structure of this Paper. After introducing syntax and semantics of the differential logic $\mathsf{d}\mathcal{L}$ in Section 2, we introduce a sequent calculus in Section 3, which can be used for verifying parametric hybrid systems in $\mathsf{d}\mathcal{L}$, and prove soundness. In Section 4, we prove an inductive safety property in train control using the $\mathsf{d}\mathcal{L}$ calculus. Finally, we draw conclusions and discuss future work in Section 5.

2 Syntax and Semantics of Differential Logic

As a uniform model for hybrid systems, our logic introduces hybrid programs, which generalise real-time programs [17] to hybrid change. They are more amenable to step-wise symbolic processing by calculus rules than graph structures of automata. Since hybrid automata [1] can be embedded (see appendix B), there is no loss of expressivity. Hybrid programs have a simple compositional semantics. With this basis, we can construct a compositional calculus that reduces verification of system properties to proving properties of its parts.

The differential logic $\mathsf{d}\mathcal{L}$ is a dynamic logic for reasoning about programs with three basic characteristics to meet the requirements of hybrid systems:

Discrete jumps. Projections in state space are represented as instantaneous assignments of values to state variables. With this, mode switches like $\mathsf{mode} := 4$ or $\mathsf{signal} := 1$ can be expressed with discrete jumps, as well as resets $z := 0$ or adjustments of control variables like $z := z - 2$.

Continuous evolution. Continuous variation in system dynamics is represented with differential equations as evolution constraints. For example, the evolution of a train with constant braking can be expressed with a system action for the differential equation $\ddot{z} = -b$ with second time-derivative $\ddot{z}$ of z. Similarly, the effect of $\ddot{z} = -b \,\&\, z \geq 5$ is an evolution that is restricted to always remain within the region $z \geq 5$, i.e., to stop braking at the latest when $z < 5$. Such an evolution can stop at any time within $z \geq 5$, it can even continue with transient grazing along the border $z = 5$, but it is not allowed to proceed when it enters $z < 5$.

Regular combinations. Discrete and continuous transitions can be combined to form *hybrid programs* using regular expression operators $(\cup, ^*, ;)$ as structured behaviour of hybrid systems. For example, $\mathsf{mode} := 4 \cup \ddot{z} = -b$ describes a train controller that can either choose to switch its state to an alert mode (4) or initiate braking by the differential equation $\ddot{z} = -b$, by a nondeterministic choice $(\cup)$. In conjunction with other regular combinations, control constraints can be expressed using conditions like $?z \geq 9$ as guards for the system state.

2.1 Syntax of dℒ

Terms and Formulas. The formulas of dℒ are built over a finite set V of real-valued variables and a fixed signature Σ of function and predicate symbols. For simplicity, the signature Σ is assumed to contain exclusively the usual function and predicate symbols for real arithmetic, such as $0, 1, +, \cdot, =, \leq, <, \geq, >$.

The set $\mathrm{Trm}(V)$ of *terms* is defined as in classical first-order logic yielding polynomial expressions. The set $\mathrm{Fml}(V)$ of *formulas* of dℒ is defined as common in first-order dynamic logic [15]. That is, they are built using propositional connectives $\wedge, \vee, \rightarrow, \leftrightarrow, \neg$ and quantifiers $\forall, \exists$ (first-order part). In addition, if ϕ is a dℒ formula and α a hybrid program, then $[\alpha]\phi, \langle\alpha\rangle\phi$ are formulas (dynamic part). Refer to Appendix A for a detailed formal definition of the syntax of dℒ.

Hybrid Programs. In dℒ, elementary discrete jumps and continuous evolutions interact using regular control structure to form hybrid programs.

Definition 2.1 (Hybrid programs). *The set* $\mathrm{HP}(V)$ *of* hybrid programs *is inductively defined as the smallest set such that:*

- *If* $x \in V$ *and* $\theta \in \mathrm{Trm}(V)$*, then* $(x := \theta) \in \mathrm{HP}(V)$.
- *If* $x \in V$*,* $\theta \in \mathrm{Trm}(V)$*, and* $\chi \in \mathrm{Fml}(V)$ *further is a quantifier-free first-order formula, then* $(\dot{x} = \theta \,\&\, \chi) \in \mathrm{HP}(V)$.
- *If* $\chi \in \mathrm{Fml}(V)$ *is a quantifier-free first-order formula, then* $(?\chi) \in \mathrm{HP}(V)$.
- *If* $\alpha, \gamma \in \mathrm{HP}(V)$ *then* $(\alpha; \gamma) \in \mathrm{HP}(V)$.
- *If* $\alpha, \gamma \in \mathrm{HP}(V)$ *then* $(\alpha \cup \gamma) \in \mathrm{HP}(V)$.
- *If* $\alpha \in \mathrm{HP}(V)$ *then* $(\alpha^*) \in \mathrm{HP}(V)$.

The effect of $x := \theta$ is an instantaneous discrete jump in state space. That of $\dot{x} = \theta \,\&\, \chi$ is an ongoing continuous evolution controlled by the differential equation $\dot{x} = \theta$ while remaining within the region described by χ. The evolution is allowed to stop at any point in χ. It is, however, obliged to stop before it leaves χ. For unrestricted evolution, we abbreviate $\dot{x} = \theta \,\&\, \mathit{true}$ by $\dot{x} = \theta$. The dℒ semantics allows differential equations with arbitrary terms θ and arbitrary occurrences of x or other variables in θ. As, e.g., in [13,20], however, our calculus assumes that $\dot{x} = \theta$ has a first-order definable flow or approximation. See [2,25] for flow approximation techniques. Extensions of dℒ and its calculus to systems of differential equations and higher-order derivatives are accordingly.

The $?\phi$ action is used to define conditions. Its semantics is that of a no-op if ϕ is true in the current state, and that of a failure divergence blocking all further evolution, otherwise. The non-deterministic choice $\alpha \cup \gamma$, sequential composition $\alpha; \gamma$ and non-deterministic repetition α^* of hybrid programs are as usual. They can be combined with $?\phi$ to form more complicated control structures, see [15].

In dℒ, there is no need to distinguish between discrete and continuous variables or between system parameters and state variables, as they share the same uniform semantics. For instance, $\exists z\, [\dot{z} = -z] z \leq 5$ expresses that there is a choice of the initial value for z (which could be a parameter) such that after all evolutions of z along $\dot{z} = -z$, the outcome of the state variable z will be at most 5. For pragmatic reasons, an informal distinction can improve readability. Formal distinctions of quantified variables and state variables [4] carry over to dℒ.

2.2 Semantics

Hybrid systems evolve along a piecewise continuous trajectory in n-dimensional space as time passes (see Fig. 1 for a possible evolution with one system variable x over time t). The discontinuities are caused by discrete jumps in the state space while the segments of continuous evolution are governed by differential equations. Concerning semantics of hybrid system models, there is a variety of slightly different formalisations. Since the interplay of discrete change with continuous evolution raises peculiar subtleties, we start with a motivation that highlights the advantages of our choice of semantics for dL.

Motivation. Consider the scenario in Fig. 1. The semantics has to restrict the behaviour of the hybrid system during the continuous evolution phase, e.g., on the interval $[1, 2]$, to respect the differential equation $\dot{x} = f(x)$. Yet, the discrete jump at time 2 will necessarily lead to a discontinuity in the overall system trajectory. Now, an overall system trajectory function g (where $g(t)$ records the value of x at time t) can only assume a *single* value at time 2, say $g(2) = 0.6$. Hence, the evolution of g will only be continuous on the inner interval $(1, 2)$. Still, the evolution along $\dot{x} = f(x)$ has to be constrained to possess a *left*-continuous continuation at time 2 towards a projected value of 1, although this value will never be assumed by g. This complicates the well-posed definition of semantics on the basis of an overall system trajectory. Note that leaving out this condition of left-continuity would lead to a total transition relation with *all* states being reachable, which, of course, would not reflect the proper system behaviour either.

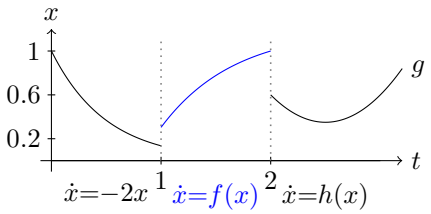


Fig. 1. Discontinuous hybrid trajectory

In contrast to this, the dL semantics inflates points in time with instantaneous discrete progression by associating an *individual* trajectory for each continuous evolution or instant jump phase. Hence, the trajectories remain continuous within each differential evolution phase, with discontinuities isolated purely in discrete jump transitions. Thereby, the dL semantics directly traces the succession of values assumed during the hybrid evolution, even if they belong to states which occur without model time passing in between. In addition to the fact that those so-called *super-dense* time effects naturally occur at mode switches between differential evolutions, they are necessary for joint mode switches of several system variables at once, like in $x := 3; y := 5$. We argue that the dL semantics is much simpler to define than for approaches with a global overall system trajectory as, for example, in [9].

Formal Semantics. The interpretations of dL consist of states (worlds) that are first-order structures over the reals, with state variables progressing along a

sequence of states. A potential behaviour of a hybrid system corresponds to a sequence of states that contain the observable values of system variables during its hybrid evolution. More precisely, the semantics of a single (compound or elementary) system action is captured by the state transitions that are possible using this action. For discrete jumps α this transition relation $\rho(\alpha)$ holds for pairs of states that satisfy the jump constraint. In case of continuous evolutions, the transition relation holds for pairs of states that can be interconnected by a continuous system flow that respects the differential equation, thereby hiding the intermediate flow details from the logic. To retain a manageable logic and calculus, it is important to hide as much as possible of the branching factor of continuous evolution from the logic. Since function and predicate symbols are interpreted as usual for real arithmetic, we omit first-order structures from the notation and focus on states, i.e, assignments of variables with real values.

Definition 2.2 (State). *A state is a map $\nu : V \to \mathbb{R}$; the set of all states is denoted by* $\mathrm{Sta}(V)$. *An* interpretation *is a non-empty set of states that is closed under hybrid program operations (see Def. 2.4).*

Further, we use $\nu[x \mapsto d]$ to denote the *semantic modification* of a state ν that is identical to ν except for the interpretation of the symbol x, which is $d \in \mathbb{R}$. With the exception of continuous evolution, the semantics, $\rho(\alpha)$, of hybrid program α as a state transition structure in $\mathsf{d}\mathcal{L}$ is as customary in dynamic logic (Def. 2.4).

Definition 2.3 (Valuation of terms and formulas). *For terms and formulas, the* valuation $val(\nu, \cdot)$ *with respect to state ν is defined as usual for first-order modal logic (e.g. [15]), i.e., using the following definitions for modal operators(see Appendix A for a detailed formal definition):*

1. $val(\nu, [\alpha]\phi) = true :\Longleftrightarrow val(\omega, \phi) = true$ *for all* ω *with* $(\nu, \omega) \in \rho(\alpha)$
2. $val(\nu, \langle\alpha\rangle\phi) = true :\Longleftrightarrow val(\omega, \phi) = true$ *for some* ω *with* $(\nu, \omega) \in \rho(\alpha)$

Definition 2.4 (Semantics of hybrid programs). *The* valuation, $\rho(\alpha)$, of a hybrid program α, *is a* transition relation *on states. It specifies which state ω is reachable from a state ν by operations of the hybrid system α and is defined as:*

1. $(\nu, \omega) \in \rho(x := \theta) :\Longleftrightarrow \omega = \nu[x \mapsto val(\nu, \theta)]$
2. $(\nu, \omega) \in \rho(\dot{x} = \theta \,\&\, \chi) :\Longleftrightarrow$ *there is a function* $f : [0, r] \to \mathrm{Sta}(V)$ *with* $r \geq 0$ *such that* $f(0) = \nu, f(r) = \omega$, *and* $val(f(\zeta), x)$ *is continuous in* ζ *on* $[0, r]$ *and has a derivative of value* $val(f(\zeta), \theta)$ *at each time* $\zeta \in (0, r)$. *For* $y \neq x$ *and* $\zeta \in [0, r]$, $val(f(\zeta), y) = val(\nu, y)$. *Further,* $val(f(\zeta), \chi) = true$ *for each* $\zeta \in (0, r)$. *Systems of differential equations are defined accordingly.*
3. $\rho(?\chi) = \{(\nu, \nu) \;:\; val(\nu, \chi) = true\}$
4. $\rho(\alpha; \gamma) = \rho(\alpha) \circ \rho(\gamma) = \{(\nu, \omega) : (\nu, z) \in \rho(\alpha), (z, \omega) \in \rho(\gamma)$ *for some state* $z\}$
5. $\rho(\alpha \cup \gamma) = \rho(\alpha) \cup \rho(\gamma)$
6. $(\nu, \omega) \in \rho(\alpha^*)$ *iff there are* $n \in \mathbb{N}$ *and* $\nu{=}\nu_0, \ldots, \nu_n{=}\omega$ *with* $(\nu_i, \nu_{i+1}) \in \rho(\alpha)$ *for* $0 \leq i < n$.

For the semantics of differential equations, derivatives are well-defined on $(0, r)$ as $\mathrm{Sta}(V)$ is isomorphic to a finite dimensional real space when V is finite.

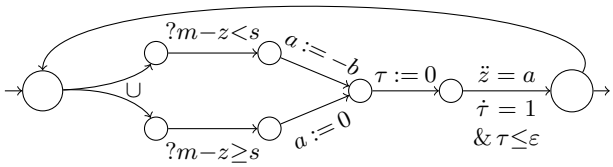


Fig. 2. Transition structure of speed supervision.

2.3 Speed Supervision in Train Control

In the European Train Control System (ETCS) [9, 12], trains are only allowed to move within their current movement authority block (MA). When their MA is not extended before reaching its end, trains always have to stop within the MA because there can be open gates or other trains beyond. Here, we identify a single component which is most responsible for the hybrid characteristics of safe driving: speed supervision locally controls the movement of a train such that it always remains within its MA. Depending on the current driving situation, the speed supervision determines a safety envelope s around the train, within which driving is safe, and adjusts its acceleration a in accordance with s (called *correction* in [9]). In the course of this paper, we derive a constraint on s that guarantees safe driving. To simplify the presentation, we assume, as in [9], that the train controller only chooses between braking and keeping speed. Constraints for positive acceleration can be derived accordingly.

Of course, a safe operation of ETCS also depends on other aspects like a disjoint partitioning of the track into MA, appropriate computation of the safe rear end of trains, or proper functioning of gates [9]. But these more static properties are much easier to show when the most important hybrid train dynamics have been captured in a reliable operation of the speed supervision.

We assume that an MA has been granted up to track position m and the train is located at position z, heading with initial speed v towards m. In this situation, $\mathsf{d}\mathcal{L}$ can analyse the following safety property of speed supervision:

$$\begin{aligned} \psi \;&\rightarrow\; [(\mathit{corr}; \mathit{drive})^*]\, z \leq m \qquad (1)\\ \text{where } \mathit{corr} \;&\equiv\; (?m - z < s; a := -b) \cup (?m - z \geq s; a := 0)\\ \mathit{drive} \;&\equiv\; \tau := 0; (\dot{z} = v, \dot{v} = a, \dot{\tau} = 1 \,\&\, v \geq 0 \wedge \tau \leq \varepsilon) \;. \end{aligned}$$

It expresses that a train will *always* remain within its MA m, assuming a constraint ψ for the parameters. In *corr*, the train corrects its acceleration or brakes with force b (as a failsafe recovery manoeuvre [9]) on the basis of the remaining distance $(m-z)$. Then, the train continues moving according to *drive*. There, the position z of the train evolves according to the system $\dot{z} = v, \dot{v} = a$ (i.e., $\ddot{z} = a$). The evolution stops when the speed v drops below zero (or earlier). Simultaneously, clock τ measures the duration of the current *drive* phase before the controllers react to situation changes (we model this to bridge the gap of continuous-time models and discrete-time control design). Clock τ is reset to zero

when entering *drive*, constantly evolves along $\dot{\tau} = 1$, and is bound by the invariant region $\tau \leq \varepsilon$. The effect is that a *drive* phase is interrupted for reassessing the driving situation after at most ε seconds, and the *corr*; *drive* loop repeats. The corresponding transition structure $\rho((corr; drive)^*)$ is depicted in Fig. 2.

Instead of manually choosing specific values for the free parameters as in [9, 12], we will use our calculus to synthesise constraints on the relationship of parameters that are required for a safe operation of train control.

3 A Verification Calculus for Differential Logic

In this section, we introduce a sequent calculus for verifying hybrid systems in $\mathsf{d}\mathcal{L}$. With the basic idea being to perform a symbolic evaluation, hybrid programs are successively transformed into logical formulas describing their effects.

For propositional logic, standard rules P1–P9 are listed in Fig. 3. The other rules transform hybrid programs into simpler logical formulas, thereby relating the meaning of programs and formulas. Rules D1–D7 are as in discrete dynamic logic [4, 15]. D8 uses generalised substitutions [4] for handling discrete change. Unlike in uninterpreted first-order logic [15], quantifiers are dealt with using quantifier elimination [8] over the reals (D9–D12) in a way that is compatible with dynamic modalities. D13–D14 handle continuous evolutions given a first-order definable flow y_x for $\dot{x} = \theta$. In combination with D9–D12, they fully encapsulate the handling of differential equations within hybrid systems. D15 is an induction schema with inductive invariant p.

3.1 Rules of the Calculus

A *sequent* is of the form $\Gamma \vdash \Delta$, where Γ and Δ are finite sets of formulas. Its semantics is that of the formula $\bigwedge_{\phi\in\Gamma} \phi \rightarrow \bigvee_{\psi\in\Delta} \psi$. Sequents will be treated as an abbreviation. In the following, an *update* $\mathcal{U}$ simply is a list of discrete assignments of the form $x := \theta$ (see [4] for techniques dealing with a single parallel update rather than a list, which can be combined with our calculus). Rules are applicable anywhere in the sequent, within any context Γ, Δ and update $\langle\mathcal{U}\rangle$:

Definition 3.1 (Provability, derivability). *A formula ψ is* provable *from a set Φ of formulas, denoted by $\Phi \vdash_{\mathsf{d}\mathcal{L}} \psi$ iff there is a finite set $\Phi_0 \subseteq \Phi$ for which the sequent $\Phi_0 \vdash \psi$ is derivable. In turn, a sequent of the form $\Gamma, \langle\mathcal{U}\rangle\Phi \vdash \langle\mathcal{U}\rangle\Psi, \Delta$ (for some update $\mathcal{U}$, including the empty update, and finite sets Γ, Δ of context formulas) is* derivable *iff there is an instance*

$$\frac{\Phi_1 \vdash \Psi_1 \qquad \dots \qquad \Phi_n \vdash \Psi_n}{\Phi \vdash \Psi}$$

of a rule schema of the $\mathsf{d}\mathcal{L}$ calculus in Fig. 3 such that for each $1 \leq i \leq n$

$$\Gamma, \langle\mathcal{U}\rangle\Phi_i \vdash \langle\mathcal{U}\rangle\Psi_i, \Delta$$

(P1) $\dfrac{\vdash \phi}{\neg\phi \vdash}$ (P4) $\dfrac{\phi, \psi \vdash}{\phi \wedge \psi \vdash}$ (P7) $\dfrac{\phi \vdash \qquad \psi \vdash}{\phi \vee \psi \vdash}$

(P2) $\dfrac{\phi \vdash}{\vdash \neg\phi}$ (P5) $\dfrac{\vdash \phi \qquad \vdash \psi}{\vdash \phi \wedge \psi}$ (P8) $\dfrac{\vdash \phi, \psi}{\vdash \phi \vee \psi}$

(P3) $\dfrac{\phi \vdash \psi}{\vdash \phi \to \psi}$ (P6) $\dfrac{\vdash \phi \qquad \psi \vdash}{\phi \to \psi \vdash}$ (P9) $\dfrac{}{\phi \vdash \phi}$

(D1) $\dfrac{\phi \wedge \psi}{\langle ?\phi \rangle \psi}$ (D5) $\dfrac{\phi \vee \langle \alpha ; \alpha^* \rangle \phi}{\langle \alpha^* \rangle \phi}$ (D9) $\dfrac{\mathsf{qelim}(\exists x \bigwedge_i (\Gamma_i \vdash \Delta_i))}{\Gamma \vdash \Delta, \exists x\, \phi}$

(D2) $\dfrac{\phi \to \psi}{[?\phi]\psi}$ (D6) $\dfrac{\phi \wedge [\alpha ; \alpha^*] \phi}{[\alpha^*]\phi}$ (D10) $\dfrac{\mathsf{qelim}(\forall x \bigwedge_i (\Gamma_i \vdash \Delta_i))}{\Gamma, \exists x\, \phi \vdash \Delta}$

(D3) $\dfrac{\langle \alpha \rangle \phi \vee \langle \gamma \rangle \phi}{\langle \alpha \cup \gamma \rangle \phi}$ (D7) $\dfrac{[\![\alpha]\!][\![\gamma]\!]\phi}{[\![\alpha ; \gamma]\!]\phi}$ (D11) $\dfrac{\mathsf{qelim}(\forall x \bigwedge_i (\Gamma_i \vdash \Delta_i))}{\Gamma \vdash \Delta, \forall x\, \phi}$

(D4) $\dfrac{[\alpha]\phi \wedge [\gamma]\phi}{[\alpha \cup \gamma]\phi}$ (D8) $\dfrac{\phi_x^\theta}{[\![x := \theta]\!]\phi}$ (D12) $\dfrac{\mathsf{qelim}(\exists x \bigwedge_i (\Gamma_i \vdash \Delta_i))}{\Gamma, \forall x\, \phi \vdash \Delta}$

(D13) $\dfrac{\exists t{\geq}0\, (\bar{\chi} \wedge \langle x := y_x(t) \rangle \phi)}{\langle \dot{x} = \theta \,\&\, \chi \rangle \phi}$ (D14) $\dfrac{\forall t{\geq}0\, (\bar{\chi} \to [x := y_x(t)] \phi)}{[\dot{x} = \theta \,\&\, \chi] \phi}$

(D15) $\dfrac{\vdash p \qquad \vdash [\alpha^*](p \to [\alpha]p)}{\vdash [\alpha^*]p}$

Rule D8 is only applicable if the substitution of x by θ in ϕ_x^θ introduces no new bindings. In D13–D14, t and $\tilde{t}$ are fresh variables, and y_v is the solution of the initial value problem $(\dot{x} = \theta, x(0) = v)$. Additionally, $\bar{\chi}$ is an abbreviation for $\forall 0{<}\tilde{t}{<}t\, \langle x := y_x(\tilde{t}) \rangle \chi$; it simplifies to *true* if χ equals *true*. In D9–D12, x does not occur in Γ, Δ. Further, the $\Gamma_i \vdash \Delta_i$ are obtained from the resulting sub-goals of a side deduction, see ($\star$) in Fig. 4. The side deduction is started from the goal $\Gamma \vdash \Delta, \phi$ at the bottom (or $\Gamma, \phi \vdash \Delta$ for D10 and D12). In the resulting sub-goals $\Gamma_i \vdash \Delta_i$, variable x is assumed to occur in first-order formulas only, as quantifier elimination (qelim) is then applicable.

Fig. 3. Rule schemata of the $\mathsf{d}\mathcal{L}$ verification calculus.

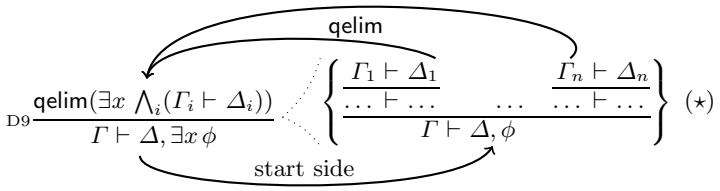


Fig. 4. Side deduction for quantifier elimination rules.

is derivable. Moreover, the symmetric schemata D1–D14 can be applied on either side of the sequent (again in context Γ, Δ and update $\langle\mathcal{U}\rangle$). In D7 and D8, the schematic modality $\langle\!|\cdot|\!\rangle$ can further be instantiated with both $[\cdot]$ and $\langle\cdot\rangle$. The same modality instance has to be chosen within a single schema instantiation, though.

As usual in sequent calculus—although the direction of entailment is from premisses (above rule bar) to conclusion (below)—the order of reasoning is *goal-directed*: Rules are applied in tableau-style, that is, starting from the desired conclusion at the bottom (goal) to the premisses (sub-goals). In the sequel we illustrate the new $\mathsf{d}\mathcal{L}$ rules.

Discrete jumps. For handling discrete change, rule D8 uses generalised substitutions [4]. The result of applying to ϕ the substitution that replaces x by θ is defined as usual [15]; it is denoted by ϕ_x^θ. Rule D8 is not applicable when the substitution introduces new bindings. For this, the definition of a "bound occurrence of a variable y" is amended to include the scope of $y := \theta$ and $\dot{y} = \theta$, because both change the value of y.

Continuous evolution. D13–D14 require solving a symbolic initial value problem. We assume that the differential equations have first-order definable unique flows. See [2, 25] for first-order approximation techniques of more general flows.

Real arithmetic. Rules D9–D12 constitute a purely modular interface to mathematical reasoning. They can use any theory that admits quantifier elimination and has a decidable ground theory (e.g., [8]):

Definition 3.2 (Quantifier elimination). *A first-order theory admits* quantifier elimination *if to each formula ϕ, a quantifier-free formula $\mathsf{qelim}(\phi)$ can be effectively associated that is equivalent (i.e., $\phi \leftrightarrow \mathsf{qelim}(\phi)$ is valid) and has no other free variables. The operation* qelim *is further assumed to evaluate ground formulas (i.e., without variables), yielding a decision procedure for this theory.*

Integrating quantifier elimination to deal with statements about real quantities is quite challenging in the presence of modalities that influence the value of variables and terms. Even more so, the effect of a modality depends on the solutions of the differential equations contained therein. For instance, it is hard to know in advance, which first-order constraints need to be solved by qelim in $\exists x\,[\ddot{x} = -b; x := 2x]x \geq 5$. To find out, the way how x evolves from $\exists x$ to $x \geq 5$ along the system dynamics needs to be taken into account. Hence, our calculus first unveils the first-order constraints on x before applying qelim. To achieve this in a concise way, we use side deductions.

The effect of a side deduction is as follows. First, the $\mathsf{d}\mathcal{L}$ calculus discovers all relevant first-order constraints from modal formulas using a side deduction in $\mathsf{d}\mathcal{L}$. Secondly, these constraints are equivalently reduced using qelim and the main proof continues. For instance, an application of D9 to a sequent $\Gamma \vdash \Delta, \exists x\,\phi$ starts a side deduction (marked ($\star$) in Fig. 4) with the goal $\Gamma \vdash \Delta, \phi$ at the bottom. This side deduction is carried out in the $\mathsf{d}\mathcal{L}$ calculus until x no longer

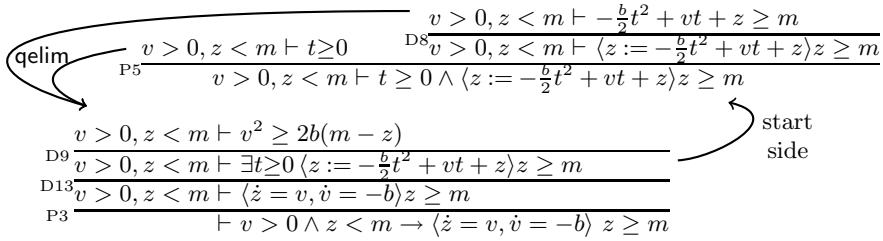


Fig. 5. Analyse MA-violation in braking mode using side deductions.

occurs within modal formulas of the remaining open branches $\Gamma_i \vdash \Delta_i$ of $(\star)$. Once all occurrences of x are in first-order formulas, the resulting sub-goals $\Gamma_i \vdash \Delta_i$ of $(\star)$ are copied back to the main proof and qelim is applied (which determines the resulting sub-goal of rule D9 on the upper left side of Fig. 4). The remaining modal formulas not containing x can be considered as atoms for this purpose as they do not impose constraints on x.

For implementations in a theorem prover, a careful analysis shows that side deductions can also be performed within the original proof, but the corresponding calculus rules, which keep track of the (lost) quantifier nesting, are quite technical and side deductions lead to a cleaner proof structure. Observe that reintegrating the open branches $\Gamma_i \vdash \Delta_i$ into the main proof corresponds to discharging multiple sub-goals simultaneously. Modifying tableau procedures to remove multiple open branches at once is not difficult. It works similarly to simultaneous closing substitutions in all branches of free variable tableaux, except that quantifier elimination can produce more than one instantiation.

3.2 Modular Combination by Side Deduction

To illustrate how our calculus combines arithmetic with dynamic reasoning using side deductions, we give an example. Because we will need a similar result when verifying speed supervision, we consider braking of trains. The deduction in Fig. 5 can be used to analyse whether a train could violate its MA although it brakes. As the prover will discover, the answer depends on the initial velocity v.

Rules D9 and D13 are implemented using Mathematica. Applying D9 in the main proof triggers a side deduction. The conjunction of the open proof goals in the side deduction can be handled by quantifier elimination and simplification in Mathematica, yielding the resulting premiss for D9 in the main proof:

$$\begin{aligned} &\mathsf{qelim}(\exists t\,\big((v>0\wedge z<m\rightarrow t\geq 0)\wedge(v>0\wedge z<m\rightarrow -\frac{b}{2}t^2+vt+z\geq m)\big)) \\ \equiv\ & v>0\wedge z<m\ \rightarrow\ v^2\geq 2b(m-z)\ . \end{aligned}$$

The open branch of the main proof reveals the speed limit and can be used to synthesise a corresponding parametric constraint. When $v^2 \geq 2b(m-z)$ holds

initially, then the MA will be violated despite braking. Similarly, $v^2 \leq 2b(m-z)$ guarantees that the MA can be respected by appropriate braking.

3.3 Soundness and Incompleteness

In this section we prove that verification with the $\mathsf{d}\mathcal{L}$ calculus always produces correct results about system safety, i.e., the $\mathsf{d}\mathcal{L}$ calculus is sound.

Theorem 3.1 (Soundness). *The $\mathsf{d}\mathcal{L}$ calculus is* sound*, i.e., derivable formulas are* valid *(true in all states of all interpretations).*

This theorem is a direct consequence of an even stronger result of soundness localised with respect to a single state (instead of requiring the premiss to be true in all states). The primary challenges within this proof are continuous evolutions and the interaction of quantifier elimination with sequent calculus.

Proposition 3.1 (Local soundness). *All $\mathsf{d}\mathcal{L}$ rules in Fig. 3 are* locally sound*: for all states ν, the conclusion is true in ν when all premisses are true in ν.*

Proof. Most rules can be proven as in [4]. The special cases for $\mathsf{d}\mathcal{L}$ are:

- Rule D9 is locally sound: Let ν be a state in which the premiss is true, i.e.,

$$\nu \models \mathsf{qelim}(\exists x \bigwedge_i (\Gamma_i \vdash \Delta_i)) \ .$$

  We have to show that the conclusion is true in this state. Using that quantifier elimination yields an equivalence, we see that ν also satisfies $\exists x \bigwedge_i (\Gamma_i \vdash \Delta_i)$ prior to the quantifier elimination. Hence, for some $d \in \mathbb{R}$ we obtain:

$$\nu[x \mapsto d] \models \bigwedge_i (\Gamma_i \vdash \Delta_i) \ .$$

  As $(\star)$ in Fig. 4 is inductively shown to be locally sound, we can conclude that $\nu[x \mapsto d] \models (\Gamma \vdash \Delta, \phi)$. Therefore, $\nu \models \exists x\, (\Gamma \vdash \Delta, \phi)$. Now the conjecture can be obtained using standard reasoning with quantifiers and the absence of x in Γ, Δ by rewriting the conclusion with local equivalences:

$$\exists x\, (\Gamma \vdash \Delta, \phi) \ \equiv \ \exists x\, (\neg\Gamma \vee \Delta \vee \phi) \ \equiv \ \neg\Gamma \vee \Delta \vee \exists x\, \phi \ \equiv \ \Gamma \vdash \Delta, \exists x\, \phi \tag{2}$$

  The soundness proof for D11 is similar since (2) holds for any quantifier. The proofs of D10 and D12 can be derived using duality of quantifiers.
- Rule D13 is locally sound: Assuming the premiss is true in some state ν, we have to show that there is a state ω with $\omega \models \phi$ such that $(\nu, \omega) \in \rho(\dot{x} = \theta)$. By premise, there is a real value $e \geq 0$ such that when we abbreviate $\nu[t \mapsto e]$ by $\tilde{\nu}$, we have $\tilde{\nu} \models \langle x := y_x(t) \rangle \phi$. Let ω_e be such that $(\tilde{\nu}, \omega_e) \in \rho(x := y_x(t))$, thus $\omega_e \models \phi$. Then, it only remains to show $(\tilde{\nu}, \omega_e) \in \rho(\dot{x} = \theta)$. This, in turn, is shown using the function $e \mapsto \omega_e$, which yields continuity and a solution of the initial value problem by the corresponding properties of y_x. Since $\rho(\dot{x} = \theta)$ and ϕ do not depend on the fresh variable t, the same reasoning holds for ν in place of $\tilde{\nu}$. The invariant χ can be shown accordingly.

$$\text{(G1)}\ \frac{\phi \vdash \psi}{\langle\!\langle\alpha\rangle\!\rangle\phi \vdash \langle\!\langle\alpha\rangle\!\rangle\psi} \qquad \text{(G2)}\ \frac{\Gamma \vdash \langle\!\langle\mathcal{U}\rangle\!\rangle p, \Delta \qquad p \vdash [\alpha]p \qquad p \vdash \phi}{\Gamma \vdash \langle\!\langle\mathcal{U}\rangle\!\rangle[\alpha^*]\phi, \Delta}$$

Fig. 6. Global and derived dℒ rules.

Now we show that unlike first-order *real* arithmetic, dℒ is undecidable. We show that both unbounded repetition in the discrete fragment and unbounded evolution in the continuous fragment cause incompleteness and undecidability.

Theorem 3.2 (Incompleteness). *Both the discrete fragment and the continuous fragment of* dℒ *are inherently* incomplete*, i.e., there is no sound and complete effective calculus. Hence, valid* dℒ *formulas are not always provable.*

Proof. We prove that natural numbers are definable amongst the real numbers of dℒ interpretations in both fragments. Then these fragments extend first-order *integer* arithmetic such that the incompleteness theorem of Gödel applies. Natural numbers are definable in the discrete fragment without continuous evolutions $\dot{x} = \theta$ using repetitive additions as usual:

$$nat(n) \;\leftrightarrow\; \langle x := 0; (x := x + 1)^* \rangle\, x = n \;\; .$$

In the continuous fragment without $\{^*, :=\}$, natural numbers are definable as:

$$nat(n) \;\leftrightarrow\; \exists s\, \exists c\, (s = 0 \wedge c = 1 \wedge \langle \dot{s} = c, \dot{c} = -s, \dot{\tau} = 1 \rangle (s = 0 \wedge \tau = n)) \;\; .$$

These ODEs have *sin* and *cos* as unique solutions for s and c, respectively. Their zeros, detected by τ, characterise an isomorphic copy of natural numbers, scaled by π. The initial values for s and c prevent the trivial solution identical to 0.

4 Verifying Speed Supervision in Train Control

Finding Inductive Candidates. We want to prove the safety statement (1) of Section 2.3. Using parametric extraction techniques, we identify both the requirement ψ for safe driving and the induction hypothesis p that is required for the proof. An unwinding of the loop in (1) by D6 can be used to extract a candidate for a parametric inductive hypothesis (similar to the proof in Section 3.2). It expresses that there is sufficient braking distance $(m - z)$ at current speed v:

$$p \;\equiv\; v^2 \leq 2b(m - z) \wedge b > 0 \wedge \varepsilon > 0 \;\; .$$

Inductive Verification. The generalisation rule G1 in Fig. 6 can be used to derive the variant G2 of the induction rule D15: With G1, G2 can be derived from D15 by discharging $[\alpha^*]$ in the premiss of D15 and strengthening ϕ to p.

Applying G2 to (1), the premiss $p \vdash z \leq m$ holds as $0 \leq v^2 \leq 2b(m - z)$ and $b > 0$. The induction start $\psi \vdash p$ of G2 will be examined after the full proof has been given, since we want to identify the prerequisite ψ for safe driving

by proof analysis. For proving the induction step $p \vdash [corr; drive]p$, we remove condition $m - z < s$ from *corr*, because it is not used in the proof (as braking remains safe with respect to $z \leq m$). Here, we abbreviate the side deductions of D11 as they do not branch but only apply P3,D8. The induction step of G2 can be proven in $\mathsf{d}\mathcal{L}$ (D11 and D14 are implemented in Mathematica):

$$\dfrac{\dfrac{\dfrac{}{p \vdash [a := -b][drive]p}\,(\ldots) \qquad \dfrac{\dfrac{}{p, m{-}z{\geq}s \vdash [a := 0][drive]p}\,(\ldots)}{p \vdash [?m{-}z{\geq}s; a := 0][drive]p}\,\text{D2,P3}}{p \vdash [corr][drive]p}\,\text{D4,P5}}{p \vdash [corr; drive]p}\,\text{D7}$$

Here, the invariant evolution conditions are convex, hence $\bar{\chi}$ can be simplified to $\langle x := y_x(t)\rangle\chi$ to save space. Further, we leave out conditions which are unnecessary for closing the proof: In the left branch, the constrained evolution of τ is irrelevant and will be left out. The left branch closes as follows (marked as $*$):

$$\dfrac{\dfrac{\dfrac{*}{p \vdash \forall t{\geq}0\,([v := -bt + v]v \geq 0 \rightarrow [z := -\frac{b}{2}t^2 + vt + z; v := -bt + v]p)}\,\text{D11,P3,D8}}{p \vdash [\dot{z} = v, \dot{v} = -b \,\&\, v \geq 0]p}\,\text{D14}}{p \vdash [a := -b][drive]p}\,\text{D8}$$

The right branch does not need condition $v \geq 0$, because v does not decrease:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{p, m{-}z{\geq}s \vdash v^2 \leq 2b(m - \varepsilon v - z)}\,(\ldots)}{p, m{-}z{\geq}s \vdash \forall t{\geq}0\,([\tau := t]\tau \leq \varepsilon \rightarrow [z := vt + z]p)}\,\text{D11,P3,D8}}{p, m{-}z{\geq}s \vdash [\tau := 0]\forall t{\geq}0\,([\tau := t + \tau]\tau \leq \varepsilon \rightarrow [z := vt + z]p)}\,\text{D8}}{p, m{-}z{\geq}s \vdash [\tau := 0][\dot{z} = v, \dot{v} = 0, \dot{\tau} = 1 \,\&\, \tau \leq \varepsilon]p}\,\text{D14}}{p, m{-}z{\geq}s \vdash [a := 0][\tau := 0][\dot{z} = v, \dot{v} = a, \dot{\tau} = 1 \,\&\, \tau \leq \varepsilon]p}\,\text{D8}}{p, m{-}z{\geq}s \vdash [a := 0][drive]p}\,\text{D7}$$

Augmenting Inductive Candidates. The right branch only closes when p guarantees the succedent $v^2 \leq 2b(m - \varepsilon v - z)$, i.e., that there will still be sufficient braking distance even after keeping the speed for up to ε seconds. As $m - z \geq s$, this succedent is implied by $v^2 \leq 2b(s - \varepsilon v)$, which can be discovered automatically by quantifier elimination. In fact, using $p \wedge v^2 \leq 2b(s - \varepsilon v)$ in place of p makes the argument inductive, and the whole proof of the safety statement (1) closes when the same formula is chosen for ψ. Here, no conjunct of ψ can be removed without leaving the proof open due to a counterexample.

From $v^2 \leq 2b(s - \varepsilon v)$, we can also derive $s \geq \varepsilon v + \frac{v^2}{2b}$ as an equivalent yet constructive constraint. From the above proof, we can further conclude that speed supervision remains safe even when s is chosen *adaptively* in accordance with this constraint at the beginning of *corr* in response to speed changes. This permits safe behaviour

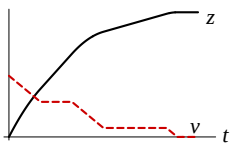


Fig. 7. Correcting.

as complex as that in Fig. 7. Similar correctness constraints can be derived when the train is allowed to increase its speed if $m - z \geq s$.

In this example, we can see the effect of the $\mathsf{d}\mathcal{L}$ calculus. It takes a specification of a hybrid system and successively identifies the arithmetic constraints which need to be investigated for proving correctness. These constraints can then be handled in a purely modular way by D9-D12 and side deductions.

5 Conclusions and Future Work

We have introduced a first-order dynamic logic with interacting discrete jumps and continuous evolutions along differential equations. For this differential logic, $\mathsf{d}\mathcal{L}$, we have presented a calculus for verifying parametric hybrid systems.

Our sequent calculus for $\mathsf{d}\mathcal{L}$ is based on a classical calculus for discrete dynamic logic [15]. In order to handle continuous evolution, we combine quantifier elimination with the calculus in a modular and constructive way. Our calculus handles first-order definable flows for differential equations. It combines non-invasively with deductive verification systems for dynamic logic. Further, it has a modular interface to combine arithmetic with dynamic reasoning in the presence of state change. We demonstrate that our calculus can verify safety in a parametric train control scenario. Meanwhile, this case study has also been verified in an interactive theorem prover based on the KeY system [3].

We currently extend our partial implementation of the $\mathsf{d}\mathcal{L}$ calculus. Moreover, dynamic logic supports reasoning about dynamic reconfiguration of system structure [4], which we want to extend to hybrid systems. Finally, our future ambition is to analyse the quotient of reasoning about hybrid systems modulo differential equation solving and inductive first-order system invariants.

Acknowledgements. I would like to thank the anonymous referees for their insightful comments and Ernst-Rüdiger Olderog and his group for their remarks.

References

1. R. Alur, T. A. Henzinger, and P.-H. Ho. Automatic symbolic verification of embedded systems. *IEEE Trans. Software Eng.*, 22(3):181–201, 1996.
2. H. Anai and V. Weispfenning. Reach set computations using real quantifier elimination. In M. D. D. Benedetto and A. L. Sangiovanni-Vincentelli, editors, *HSCC*, volume 2034 of *LNCS*, pages 63–76. Springer, 2001.
3. B. Beckert, R. Hähnle, and P. H. Schmitt, editors. *Verification of Object-Oriented Software: The KeY Approach*, volume 4334 of *LNCS*. Springer-Verlag, 2007.
4. B. Beckert and A. Platzer. Dynamic logic with non-rigid functions: A basis for object-oriented program verification. In U. Furbach and N. Shankar, editors, *IJCAR*, volume 4130 of *LNCS*, pages 266–280. Springer, 2006.
5. A. Bemporad, A. Bicchi, and G. Buttazzo, editors. *Hybrid Systems: Computation and Control, 10th International Conference, HSCC 2007, Pisa, Italy, Proceedings*, volume 4416 of *LNCS*. Springer, 2007.

6. R. J. Boulton, R. Hardy, and U. Martin. A Hoare logic for single-input single-output continuous-time control systems. In O. Maler and A. Pnueli, editors, *HSCC*, volume 2623 of *LNCS*, pages 113–125. Springer, 2003.
7. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.
8. G. E. Collins and H. Hong. Partial cylindrical algebraic decomposition for quantifier elimination. *J. Symb. Comput.*, 12(3):299–328, 1991.
9. W. Damm, H. Hungar, and E.-R. Olderog. On the verification of cooperating traffic agents. In F. S. de Boer, M. M. Bonsangue, S. Graf, and W. P. de Roever, editors, *FMCO*, volume 3188 of *LNCS*, pages 77–110. Springer, 2003.
10. J. M. Davoren. On hybrid systems and the modal μ-calculus. In P. J. Antsaklis, W. Kohn, M. D. Lemmon, A. Nerode, and S. Sastry, editors, *Hybrid Systems*, volume 1567 of *LNCS*, pages 38–69. Springer, 1997.
11. J. M. Davoren and A. Nerode. Logics for hybrid systems. *Proceedings of the IEEE*, 88(7):985–1010, July 2000.
12. J. Faber and R. Meyer. Model checking data-dependent real-time properties of the European Train Control System. In *FMCAD*, pages 76–77. IEEE Computer Society, 2006.
13. M. Fränzle. Analysis of hybrid systems. In J. Flum and M. Rodríguez-Artalejo, editors, *CSL*, volume 1683 of *LNCS*, pages 126–140. Springer, 1999.
14. G. Frehse. PHAVer: Algorithmic verification of hybrid systems past HyTech. In M. Morari and L. Thiele, editors, *HSCC*, volume 3414 of *LNCS*, pages 258–273. Springer, 2005.
15. D. Harel, D. Kozen, and J. Tiuryn. *Dynamic logic*. MIT Press, 2000.
16. T. A. Henzinger. The theory of hybrid automata. In *LICS*, pages 278–292. IEEE Computer Society, 1996.
17. T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. In *LICS*, pages 394–406. IEEE Computer Society, 1992.
18. D. Hutter, B. Langenstein, C. Sengler, J. H. Siekmann, W. Stephan, and A. Wolpers. Deduction in the verification support environment (VSE). In M.-C. Gaudel and J. Woodcock, editors, *FME*, volume 1051 of *LNCS*. Springer, 1996.
19. G. Lafferriere, G. J. Pappas, and S. Yovine. A new class of decidable hybrid systems. In F. W. Vaandrager and J. H. van Schuppen, editors, *HSCC*, volume 1569 of *LNCS*, pages 137–151. Springer, 1999.
20. C. Piazza, M. Antoniotti, V. Mysore, A. Policriti, F. Winkler, and B. Mishra. Algorithmic algebraic model checking I. In K. Etessami and S. K. Rajamani, editors, *CAV*, volume 3576 of *LNCS*, pages 5–19. Springer, 2005.
21. A. Platzer. Differential dynamic logic for verifying parametric hybrid systems. In N. Olivetti, editor, *TABLEAUX*, LNCS. Springer, 2007.
22. A. Platzer. Differential logic for reasoning about hybrid systems. In Bemporad et al. [5], pages 746–749.
23. A. Platzer. A temporal dynamic logic for verifying hybrid system invariants. In S. Artemov and A. Nerode, editors, *Logical Foundations of Computer Science, International Symposium, LFCS 2007, New York, USA, Proceedings*, volume 4514 of *LNCS*, pages 457–471. Springer, 2007.
24. A. Platzer. Towards a hybrid dynamic logic for hybrid dynamic systems. In P. Blackburn, T. Bolander, T. Braüner, V. de Paiva, and J. Villadsen, editors, *Proc., LICS International Workshop on Hybrid Logic, 2006, Seattle*, ENTCS, 2007.
25. A. Platzer and E. M. Clarke. The image computation problem in hybrid systems model checking. In Bemporad et al. [5], pages 473–486.
26. M. Rönkkö, A. P. Ravn, and K. Sere. Hybrid action systems. *Theor. Comput. Sci.*, 290(1):937–973, 2003.

27. W. C. Rounds. A spatial logic for the hybrid π-calculus. In R. Alur and G. J. Pappas, editors, *HSCC*, volume 2993 of *LNCS*, pages 508–522. Springer, 2004.
28. C. Zhou, A. P. Ravn, and M. R. Hansen. An extended duration calculus for hybrid real-time systems. In R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, editors, *Hybrid Systems*, volume 736 of *LNCS*, pages 36–59. Springer, 1992.

A Formal Syntax and Semantics of $\mathbf{d\mathcal{L}}$

A.1 Syntax

The sets $\mathrm{Trm}(V)$ of terms, $\mathrm{Fml}(V)$ of formulas, and $\mathrm{HP}(V)$ of hybrid programs are simultaneously inductively defined in Definitions A.1, A.2 and 2.1, respectively.

Definition A.1 (Terms). $\mathrm{Trm}(V)$ *is the set of all* terms*, which is the smallest set such that:*

- *If $x \in V$ is a variable, then $x \in \mathrm{Trm}(V)$.*
- *If $f \in \Sigma$ is a function symbol and, for $1 \leq i \leq n$, $\theta_i \in \mathrm{Trm}(V)$, then $f(\theta_1, \ldots, \theta_n) \in \mathrm{Trm}(V)$.*

Definition A.2 (Formulas). *The set* $\mathrm{Fml}(V)$ *of* formulas *is the smallest set with:*

- *If $p \in \Sigma$ is a predicate symbol and $\theta_i \in \mathrm{Trm}(V)$, then $p(\theta_1, \ldots, \theta_n) \in \mathrm{Fml}(V)$.*
- *If $\phi, \psi \in \mathrm{Fml}(V)$, then $\neg\phi, (\phi \wedge \psi), (\phi \vee \psi), (\phi \rightarrow \psi) \in \mathrm{Fml}(V)$.*
- *If $\phi \in \mathrm{Fml}(V)$ and $x \in V$, then $\forall x\, \phi, \exists x\, \phi \in \mathrm{Fml}(V)$.*
- *If $\phi \in \mathrm{Fml}(V)$ and $\alpha \in \mathrm{HP}(V)$, then $[\alpha]\phi, \langle\alpha\rangle\phi \in \mathrm{Fml}(V)$.*

A.2 Semantics

The valuation $val(\nu, \cdot)$ of terms and formulas, and the semantics $\rho(\alpha)$ of hybrid programs are simultaneously inductively defined in Definitions A.3, A.4 and 2.4, respectively.

Definition A.3 (Valuation of Terms). *The* valuation of terms *with respect to state ν is defined by:*

1. *$val(\nu, x) = \nu(x)$ if x is a variable*
2. *$val(\nu, f(\theta_1, \ldots, \theta_n)) = f^\ell\big(val(\nu, \theta_1), \ldots, val(\nu, \theta_n)\big)$, where f^ℓ is the operation associated to f.*

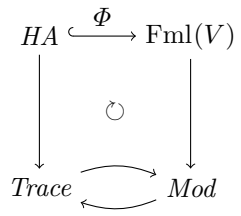
Definition A.4 (Valuation of Formulas). *The* valuation of formulas *with respect to ν is defined by:*

1. *$val(\nu, p(\theta_1, \ldots, \theta_n)) = p^\ell\big(val(\nu, \theta_1), \ldots, val(\nu, \theta_n)\big)$, where p^ℓ is the relation associated to p*
2. *$val(\nu, \phi \wedge \psi)$ is defined as usual, the same holds for $\neg, \vee, \rightarrow, \leftrightarrow$*
3. *$val(\nu, \forall x\, \phi) = true :\Longleftrightarrow val(\nu[x \mapsto d], \phi) = true$ for all $d \in \mathbb{R}$*
4. *$val(\nu, \exists x\, \phi) = true :\Longleftrightarrow val(\nu[x \mapsto d], \phi) = true$ for some $d \in \mathbb{R}$*
5. *$val(\nu, [\alpha]\phi) = true :\Longleftrightarrow val(\omega, \phi) = true$ for all ω with $(\nu, \omega) \in \rho(\alpha)$*
6. *$val(\nu, \langle\alpha\rangle\phi) = true :\Longleftrightarrow val(\omega, \phi) = true$ for some ω with $(\nu, \omega) \in \rho(\alpha)$*

B Embedding Linear Hybrid Automata

In this section we show that hybrid automata [16] can be embedded faithfully into $\mathsf{d}\mathcal{L}$.

Proposition B.1 (Hybrid Automata Embedding). *There is an effective mapping Φ from linear hybrid automata to formulas of $\mathsf{d}\mathcal{L}$ such that the following diagram commutes (Trace is the set of hybrid automata traces, while Mod is the set of satisfying models for the respective formulas):*



Proof. Without loss of generality we can assume the absence of (*convex*) invariants since they can be deferred to transition guards for reachability investigations. Moreover, by using instant transitions to subsequent initial locations, we can assume the presence of a single initial location s_0 without loss of generality. Hence, assume a linear hybrid automaton with a set of discrete locations S, an initial location s_0, jump constraints $\mathsf{jump}(e)$ relating the prestate variables x and poststate variables x^+ for each edge e, continuous evolutions $\dot{x} = c$ (see below for differential inclusions).

The linear hybrid automaton can reach a state satisfying condition ϕ if and only if the following $\mathsf{d}\mathcal{L}$ formula is satisfiable:

$$\begin{array}{l}\langle s := s_0; \\ \big(?s = s_1; (x^+ := *; ?\mathsf{jump}(e); x := x^+); s := s_2 \\ \cup\, ?s = s_1; \dot{x} = c \\ \cup\, \ldots \\ \big)^*\rangle\ \phi\ .\end{array}$$

The system actions in this formula are subject to a choice for each edge e from some state s_1 to some state s_2 (accordingly for more locations). The random assignment, in turn, can be axiomatised in $\mathsf{d}\mathcal{L}$ as follows:

$$\langle x := *\rangle\phi \ \leftrightarrow\ \exists y\, \langle x := y\rangle\phi$$

Further, notice that differential inequalities are expressible with differential equations in $\mathsf{d}\mathcal{L}$ provided, for instance, $\theta = c \in \mathbb{R}$ is constant:

$$[\dot{x} \leq \theta]\phi \ \equiv\ [z := x; \dot{z} = \theta]\forall x\, (x \leq z \to \phi) \ \equiv\ \forall x\, \big(\langle z := x; \dot{z} = \theta\rangle x \leq z \to \phi\big)$$

Similarly, disjunctions in flow constraints can be expressed as $\dot{x} = c_1 \cup \dot{x} = c_2$ rather than $\dot{x} = c$. Conjunctions lead to systems of differential equations, whereas negations as in $[\dot{x} \neq \theta]\phi \ \equiv\ \forall x\, \phi$ are pointless from a reachability point of view.