Quantified Differential Dynamic Logic for Distributed Hybrid Systems*

André Platzer

Carnegie Mellon University, Computer Science Department, Pittsburgh, PA, USA
aplatzer@cs.cmu.edu

Abstract. We address a fundamental mismatch between the combinations of dynamics that occur in complex physical systems and the limited kinds of dynamics supported in analysis. Modern applications combine communication, computation, and control. They may even form dynamic networks, where neither structure nor dimension stay the same while the system follows mixed discrete and continuous dynamics.
We provide the logical foundations for closing this analytic gap. We develop a system model for distributed hybrid systems that combines quantified differential equations with quantified assignments and dynamic dimensionality-changes. We introduce a dynamic logic for verifying distributed hybrid systems and present a proof calculus for it. We prove that this calculus is a sound and complete axiomatization of the behavior of distributed hybrid systems relative to quantified differential equations. In our calculus we have proven collision freedom in distributed car control even when new cars may appear dynamically on the road.

1 Introduction

Many safety-critical computers are embedded in cyber-physical systems like cars [1] or aircraft [2]. How do we know that their designs will work as intended? Ensuring correct functioning of cyber-physical systems is among the most challenging and most important problems in computer science, mathematics, and engineering. But the ability to analyze and understand global system behavior is the key to designing smart and reliable control.

Today, there is a fundamental mismatch between the actual combinations of dynamics that occur in applications and the restricted kinds of dynamics supported in analysis. Safety-critical systems in automotive, aviation, railway, and power grids combine *communication, computation, and control*. Combining computation and control leads to *hybrid systems* [3], whose behavior involves both discrete and continuous dynamics originating, e.g., from discrete control decisions and differential equations of movement. Combining communication and computation leads to *distributed systems* [4], whose dynamics are discrete transitions of system parts that communicate with each other. They may form *dynamic*

* This material is based upon work supported by the National Science Foundation under Grant Nos. CNS-0926181 and CNS-0931985, by the NASA grant NNG-05GF84H, and by the ONR award N00014-10-1-0188.

A. Dawar and H. Veith (Eds.): CSL 2010, LNCS 6247, pp. 469–483, 2010.

distributed systems, where the structure of the system is not fixed but evolves over time and agents may appear or disappear during the system evolution.

Combinations of all three aspects (communication, computation, and control) are used in sophisticated applications, e.g., cooperative distributed car control [1]. Neither structure nor dimension stay the same, because new cars can appear on the street or leave it; see Fig. 1. These systems are *(dynamic) distributed hybrid systems*. They cannot be considered just as a distributed system (because, e.g., the continuous evolution of positions and velocities matters for collision freedom in car control) nor just as a hybrid system (because the evolving system structure and appearance of new agents can make an otherwise collision-free system unsafe). It is generally impossible to split the analysis of distributed hybrid systems soundly into an analysis of a distributed system (without continuous movement) and an analysis of a hybrid system (without structural changes or appearance), because all kinds of dynamics interact. Just like hybrid systems that generally cannot be analyzed from a purely discrete or a purely continuous perspective [3, 5].

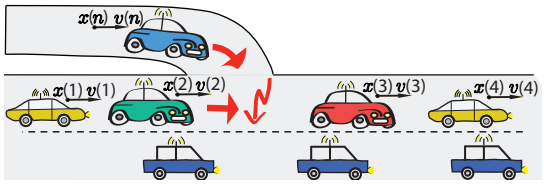

Fig. 1. Distributed car control.

Distributed hybrid systems have been considered to varying degrees in modeling languages [6–9]. In order to build these systems, however, scientists and engineers also need analytic tools to understand and predict their behavior. But formal verification and proof techniques do not yet support the required combination of dynamical effects—which is not surprising given the numerous sources of undecidability for distributed hybrid systems verification.

In this paper, we provide the logical foundations to close this fundamental analytic gap. We develop *quantified hybrid programs* (QHPs) as a model for distributed hybrid systems, which combine dynamical effects from multiple sources: *discrete transitions, continuous evolution, dimension changes, and structural dynamics*. In order to account for changes in the dimension and for co-evolution of an unbounded and evolving number of participants, we generalize the notion of states from assignments for primitive system variables to full first-order structures. Function term $x(i)$ may denote the position of car i of type C, $f(i)$ could be the car registered by communication as the car following car i, and the term $d(i, f(i))$ could denote the minimum safety distance negotiated between car i and its follower. The values of all these terms may evolve *for all* i as time progresses according to interacting laws of discrete and continuous dynamics. They are also affected by changing the system dimension as new cars appear, disappear, or by reconfiguring the system structure dynamically. The defining characteristic of QHPs is that they allow *quantified hybrid dynamics* in which variables like i that occur in function arguments of the system dynamics are quantified over, such that the system co-evolves, e.g., *for all* cars i of type C.

There is a crucial difference between a primitive system variable x and a first-order function term $x(i)$, where i is quantified over. Hybrid dynamics of primitive system variables can model, say, 5 cars (putting scalability issues aside), but not n cars and not systems with a varying number of cars. With first-order function symbols $x(i)$ and hybrid dynamics quantifying over all cars i, a QHP can represent *any* number of cars at once and even (dis)appearance of cars.

Verification of distributed hybrid systems is challenging, because they have three independent sources of undecidability: discrete dynamics, continuous dynamics, and structural/dimensional dynamics. As an analysis tool for distributed hybrid systems, we introduce a specification and verification logic for QHPs that we call *quantified differential dynamic logic* (Qd$\mathcal{L}$). Qd$\mathcal{L}$ provides dynamic logic [10] modal operators $[\alpha]$ and $\langle\alpha\rangle$ that refer to the states reachable by QHP α and can be placed in front of any formula. Formula $[\alpha]\phi$ expresses that all states reachable by system α satisfy formula ϕ, while $\langle\alpha\rangle\phi$ expresses that there is at least one reachable state satisfying ϕ. These modalities can express necessary or possible properties of the transition behavior of α. With its ability to verify (dynamic) distributed hybrid systems and quantified dynamics, Qd$\mathcal{L}$ is a major extension of prior work for static hybrid systems [5, 11] or programs [12, 13].

Our primary contributions are:

- We introduce a *system model and semantics* that succinctly captures the logical quintessence of (dynamic) distributed hybrid systems with joint discrete, continuous, structural, and dimension-changing dynamics.
- We introduce a *specification/verification logic* for distributed hybrid systems.
- We present a *proof calculus* for this logic, which, to the best of our knowledge, is the *first verification approach* that can handle distributed hybrid systems with their hybrid dynamics and unbounded (and evolving) dimensions.
- We prove that this compositional calculus is a *sound and complete axiomatization* relative to differential equations.
- We have used our proof calculus to verify *collision freedom in a distributed car control system*, where new cars may appear dynamically on the road.

This work constitutes the logical foundation for analysis of distributed hybrid systems. Since distributed hybrid control is the key to control numerous advanced systems, analytic approaches have significant potential for applications.

Our verification approach for distributed hybrid systems is a fundamental extension compared to previous approaches. In much the same way as first-order logic increases the expressive power over propositional logic (quantifiers and function symbols are required to express properties of unbounded structures), Qd$\mathcal{L}$ increases the expressive power over its predecessors (because first-order functions and quantifiers in the dynamics of QHPs are required to characterize systems with unbounded and changing dimensions).

2 Related Work

Multi-party distributed control has been suggested for car control [1] and air traffic control [2]. Due to limits in verification technology, no formal analysis of

the distributed hybrid dynamics has been possible for these systems yet. Analysis results include discrete message handling [1] or collision avoidance for two participants [2]. In distributed car control and air traffic control systems, appearance of new participants is a major unsolved challenge for formal verification.

The importance of understanding dynamic / reconfigurable distributed hybrid systems was recognized in modeling languages SHIFT [6] and R-Charon [8]. They focused on simulation / compilation [6] or the development of a semantics [8], so that no verification is possible yet. For stochastic simulation see [9], where soundness has not been proven, because ensuring coverage is difficult.

For distributed hybrid systems, even giving a formal semantics is very challenging [14, 7, 8, 15]! Zhou et al. [14] gave a semantics for a hybrid version of CSP in the Extended Duration Calculus. Rounds [7] gave a semantics in a rich set theory for a spatial logic for a hybrid version of the π-calculus. In the hybrid π-calculus, processes interact with a continuously changing environment, but cannot themselves evolve continuously, which would be crucial to capture the physical movement of traffic agents. From the semantics alone, no verification is possible in these approaches, except perhaps by manual semantic reasoning.

Other process-algebraic approaches, like χ [15], have been developed for modeling and simulation. Verification is still limited to small fragments that can be translated directly to other verification tools like PHAVer or UPPAAL, which have fixed dimensions and restricted dynamics (no distributed hybrid systems).

Our approach is completely different. It is based on first-order structures and dynamic logic. We focus on developing a logic that supports distributed hybrid dynamics and is amenable to automated theorem proving in the logic itself.

Our previous work and other verification approaches for static hybrid systems cannot verify distributed hybrid systems. Distributed hybrid systems may have an unbounded and changing number of components/participants, which cannot be represented with any fixed number of dimensions of the state space. In distributed car control, for instance, there is no prior limit on the number of cars on the street. Even when there is a limit, explicit replication of the system, say, 100 times, does not yield a scalable verification approach.

Approaches for distributed systems [4] do not cover hybrid systems, because the addition of differential equations to distributed systems is even more challenging than the addition of differential equations to discrete dynamics.

3 Syntax of QdL

As a formal logic for specifying and verifying correctness properties of distributed hybrid systems, we introduce *quantified differential dynamic logic* (QdL). QdL combines dynamic logic for reasoning about system runs [10] with many-sorted first-order logic for reasoning about all ($\forall i:C\ \phi$) or some ($\exists i:C\ \phi$) objects of a sort C, e.g., the sort of all cars. The most important defining characteristic of QdL is that its system model of *quantified hybrid programs* (QHP) supports quantified operations that affect *all* objects of a sort C at once. If C is the sort of cars, QHP $\forall i:C\ a(i) := a(i)+1$ increases the respective accelerations $a(i)$ of

all cars i at once. QHP $\forall i : C\ v(i)' = a(i)$ represents a continuous evolution of the respective velocities $v(i)$ of *all cars* at once according to their acceleration. Quantified assignments and quantified differential equation systems are crucial for representing distributed hybrid systems where an unbounded number of objects co-evolve simultaneously. Note that we use the same quantifier notation $\forall i : C$ for quantified operations in programs and for logical formulas.

We model the appearance of new participants in the distributed hybrid system, e.g., new cars entering the road, by a program $n := \mathsf{new}\, C$. It creates a new object of type C, thereby extending the range of subsequent quantified assignments or differential equations ranging over created objects of type C. With quantifiers and function terms, new can be handled in a modular way (Section 5).

3.1 Quantified Differential Dynamic Logic

Sorts Qd$\mathcal{L}$ supports a (finite) number of object sorts, e.g., the sort of all cars. For continuous quantities of distributed hybrid systems like positions or velocities, we add the sort $\mathbb{R}$ for real numbers. See previous work [12] for subtyping of sorts.

Terms Qd$\mathcal{L}$ terms are built from a set of (sorted) function/variable symbols as in many-sorted first-order logic. Unlike in first-order logic, the interpretation of function symbols can change by running QHPs. Even objects may appear or disappear while running QHPs. We use function symbol $\exists(\cdot)$ to distinguish between objects i that actually exist and those that have not been created yet, depending on the value of $\exists(i)$, which may change its interpretation. We use $0, 1, +, -, \cdot$ with the usual notation and fixed semantics for real arithmetic. For $n \geq 0$ we abbreviate $f(s_1, \ldots, s_n)$ by $f(\boldsymbol{s})$ using vectorial notation and we use $\boldsymbol{s} = \boldsymbol{t}$ for element-wise equality.

Formulas The formulas of Qd$\mathcal{L}$ are defined as in first-order dynamic logic plus many-sorted first-order logic by the following grammar (ϕ, ψ are formulas, θ_1, θ_2 are terms of the same sort, i is a variable of sort C, and α is a QHP):

$$\phi, \psi \ ::= \ \theta_1 = \theta_2 \mid \theta_1 \geq \theta_2 \mid \neg\phi \mid \phi \wedge \psi \mid \forall i : C\ \phi \mid \exists i : C\ \phi \mid [\alpha]\phi \mid \langle\alpha\rangle\phi$$

We use standard abbreviations to define $\leq, >, <, \vee, \rightarrow$. Sorts $C \neq \mathbb{R}$ have no ordering and only $\theta_1 = \theta_2$ is allowed. For sort $\mathbb{R}$, we abbreviate $\forall x : \mathbb{R}\ \phi$ by $\forall x\, \phi$. In the following, all formulas and terms have to be well-typed. Qd$\mathcal{L}$ formula $[\alpha]\phi$ expresses that *all states* reachable by QHP α satisfy formula ϕ. Likewise, $\langle\alpha\rangle\phi$ expresses that *there is at least one state* reachable by α for which ϕ holds.

For short notation, we allow conditional terms of the form $\mathsf{if}\, \phi\, \mathsf{then}\, \theta_1\, \mathsf{else}\, \theta_2\, \mathsf{fi}$ (where θ_1 and θ_2 have the same sort). This term evaluates to θ_1 if the formula ϕ is true and to θ_2 otherwise. We consider formulas with conditional terms as abbreviations, e.g., $\psi(\mathsf{if}\, \phi\, \mathsf{then}\, \theta_1\, \mathsf{else}\, \theta_2\, \mathsf{fi})$ for $(\phi \rightarrow \psi(\theta_1)) \wedge (\neg\phi \rightarrow \psi(\theta_2))$.

Example A major challenge in distributed car control systems [1] is that they do not follow fixed, static setups. Instead, new situations can arise dynamically that change structure and dimension of the system whenever new cars appear on the road from on-ramps or leave it; see Fig. 1. As a running example, we model a *distributed car control system DCCS*. First, we consider $\mathtt{Qd}\mathcal{L}$ properties.

If i is a term of type C (for cars), let $x(i)$ denote the position of car i, $v(i)$ its current velocity, and $a(i)$ its current acceleration. A state is collision-free if all cars are at different positions, i.e., $\forall i{\neq}j : C\ x(i){\neq}x(j)$. The following $\mathtt{Qd}\mathcal{L}$ formula expresses that the system *DCCS* controls cars collision-free:

$$(\forall i,j : C\ \mathcal{M}(i,j)) \rightarrow [DCCS]\ \forall i{\neq}j : C\ x(i){\neq}x(j) \tag{1}$$

It says that *DCCS* controlled cars are always in a collision-free state (postcondition), provided that *DCCS* starts in a state satisfying $\mathcal{M}(i,j)$ for all cars i,j (precondition). Formula $\mathcal{M}(i,j)$ characterizes a simple compatibility condition: for different cars $i \neq j$, the car that is further down the road (i.e., with greater position) neither moves slower nor accelerates slower than the other car, i.e.:

$$\begin{aligned}\mathcal{M}(i,j)\ \equiv\ i \neq j \rightarrow \big((x(i) < x(j) \land v(i) \leq v(j) \land a(i) \leq a(j))\\ \lor\ (x(i) > x(j) \land v(i) \geq v(j) \land a(i) \geq a(j))\big)\end{aligned} \tag{2}$$

3.2 Quantified Hybrid Programs

As a system model for distributed hybrid systems, we introduce *quantified hybrid programs* (QHP). These are regular programs from dynamic logic [10] to which we add quantified assignments and quantified differential equation systems for *distributed* hybrid dynamics. From these, QHPs are built like a Kleene algebra with tests [16]. QHPs are defined by the following grammar (α, β are QHPs, θ a term, i a variable of sort C, f is a function symbol, $\boldsymbol{s}$ is a vector of terms with sorts compatible to f, and χ is a formula of first-order logic):

$$\alpha,\beta\ ::=\ \forall i : C\ f(\boldsymbol{s}) := \theta \mid \forall i : C\ f(\boldsymbol{s})' = \theta\,\&\,\chi \mid ?\chi \mid \alpha \cup \beta \mid \alpha;\beta \mid \alpha^*$$

Quantified State Change The effect of *quantified assignment* $\forall i : C\ f(\boldsymbol{s}) := \theta$ is an instantaneous discrete jump assigning θ to $f(\boldsymbol{s})$ simultaneously for all objects i of sort C. The effect of *quantified differential equation* $\forall i : C\ f(\boldsymbol{s})' = \theta\,\&\,\chi$ is a continuous evolution where, for all objects i of sort C, all differential equations $f(\boldsymbol{s})' = \theta$ hold and formula χ holds throughout the evolution (the state remains in the region described by χ). The dynamics of QHPs changes the interpretation of terms over time: $f(\boldsymbol{s})'$ is intended to denote the derivative of the interpretation of the term $f(\boldsymbol{s})$ over time during continuous evolution, not the derivative of $f(\boldsymbol{s})$ by its argument $\boldsymbol{s}$. For $f(\boldsymbol{s})'$ to be defined, we assume f is an $\mathbb{R}$-valued function symbol. For simplicity, we assume that f does not occur in $\boldsymbol{s}$. In most quantified assignments/differential equations $\boldsymbol{s}$ is just i. Time itself is implicit. If a clock variable t is needed in a QHP, it can be axiomatized by $t' = 1$, which is equivalent to $\forall i : C\ t' = 1$ where i does not occur in t. For such *vacuous quantification* (i does not occur anywhere), we may omit $\forall i : C$ from assignments and differential equations. Similarly, we may omit vectors $\boldsymbol{s}$ of length 0.

Regular Programs The effect of *test* $?\chi$ is a *skip* (i.e., no change) if formula χ is true in the current state and *abort* (blocking the system run by a failed assertion), otherwise. *Nondeterministic choice* $\alpha \cup \beta$ is for alternatives in the behavior of the distributed hybrid system. In the *sequential composition* $\alpha; \beta$, QHP β starts after α finishes (β never starts if α continues indefinitely). *Nondeterministic repetition* α^* repeats α an arbitrary number of times, possibly zero times.

QHPs (with their semantics and our proof rules) can be extended to systems of quantified differential equations, simultaneous assignments to multiple functions f, g, or statements with multiple quantifiers ($\forall i : C \ \forall j : D \ \ldots$). To simplify notation, we do not focus on these cases, which are vectorial extensions [5, 12].

Example Continuous movement of position $x(i)$ of car i with acceleration $a(i)$ is expressed by differential equation $x(i)'' = a(i)$, which corresponds to the first-order differential equation system $x(i)' = v(i), v(i)' = a(i)$ with velocity $v(i)$. Simultaneous movement of all cars with their respective accelerations $a(i)$ is expressed by the QHP $\forall i : C \ (x(i)'' = a(i))$ where quantifier $\forall i : C$ ranges over all cars, such that all cars co-evolve at the same time.

In addition to continuous dynamics, cars have discrete control. In the following QHP, discrete and continuous dynamics interact (repeatedly by the *):

$$\big(\forall i : C \ (a(i) := \mathsf{if}\, \forall j : C \ \mathit{far}(i,j)\, \mathsf{then}\, a\, \mathsf{else}\, -b\, \mathsf{fi});\ \ \forall i : C \ (x(i)'' = a(i))\big)^*$$

First, all cars i control their acceleration $a(i)$. Each car i chooses maximum acceleration $a \geq 0$ for $a(i)$ if its distance to all other cars j is far enough (some condition $\mathit{far}(i,j)$). Otherwise, i chooses full braking $-b < 0$. After all accelerations have been set, all cars move continuously along $\forall i : C \ (x(i)'' = a(i))$. Accelerations may change repeatedly, because the repetition operator * can repeat the QHP when the continuous evolution stops at any time.

4 Semantics of Qd$\mathcal{L}$

The Qd$\mathcal{L}$ semantics is a *constant domain Kripke semantics [17] with first-order structures as states* that associate total functions of appropriate type with function symbols. In constant domain, all states share the same domain for quantifiers. In particular, we choose to represent object creation not by changing the domain of states, but by changing the interpretation of the createdness flag $\mathsf{E}(i)$ of the object denoted by i. With $\mathsf{E}(i)$, object creation is definable (Section 5).

States A *state* σ associates an infinite set $\sigma(C)$ of objects with each sort C, and it associates a function $\sigma(f)$ of appropriate type with each function symbol f, including $\mathsf{E}(\cdot)$. For simplicity, σ also associates a value $\sigma(i)$ of appropriate type with each variable i. The domain of $\mathbb{R}$ and the interpretation of $0, 1, +, -, \cdot$ is that of real arithmetic. We assume constant domain for each sort C: all states σ, τ share the same infinite domains $\sigma(C) = \tau(C)$. Sorts $C \neq D$ are disjoint: $\sigma(C) \cap \sigma(D) = \emptyset$. The set of all states is denoted by $\mathcal{S}$. The state σ_i^e agrees with σ except for the interpretation of variable i, which is changed to e.

Formulas We use $\sigma[\![\theta]\!]$ to denote the value of term θ at state σ. Especially, $\sigma_i^e[\![\theta]\!]$ denotes the value of θ in state σ_i^e, i.e., in state σ with i interpreted as e. Further, $\rho(\alpha)$ denotes the state transition relation of QHP α as defined below. The *interpretation* $\sigma \models \phi$ of $\mathsf{Qd}\mathcal{L}$ formula ϕ with respect to state σ is defined as:

1. $\sigma \models (\theta_1 = \theta_2)$ iff $\sigma[\![\theta_1]\!] = \sigma[\![\theta_2]\!]$; accordingly for $\geq$.
2. $\sigma \models \phi \wedge \psi$ iff $\sigma \models \phi$ and $\sigma \models \psi$; accordingly for $\neg$.
3. $\sigma \models \forall i{:}C\ \phi$ iff $\sigma_i^e \models \phi$ for all objects $e \in \sigma(C)$.
4. $\sigma \models \exists i{:}C\ \phi$ iff $\sigma_i^e \models \phi$ for some object $e \in \sigma(C)$.
5. $\sigma \models [\alpha]\phi$ iff $\tau \models \phi$ for all states τ with $(\sigma, \tau) \in \rho(\alpha)$.
6. $\sigma \models \langle\alpha\rangle\phi$ iff $\tau \models \phi$ for some τ with $(\sigma, \tau) \in \rho(\alpha)$.

Programs The *transition relation,* $\rho(\alpha) \subseteq \mathcal{S} \times \mathcal{S}$, of QHP α specifies which state $\tau \in \mathcal{S}$ is reachable from $\sigma \in \mathcal{S}$ by running QHP α. It is defined inductively:

1. $(\sigma, \tau) \in \rho(\forall i{:}C\ f(\boldsymbol{s}) := \theta)$ iff state τ is identical to σ except that at each position $\boldsymbol{o}$ of f: if $\sigma_i^e[\![\boldsymbol{s}]\!] = \boldsymbol{o}$ for some object $e \in \sigma(C)$, then $\tau(f)\big(\sigma_i^e[\![\boldsymbol{s}]\!]\big) = \sigma_i^e[\![\theta]\!]$. If there are multiple objects e giving the same position $\sigma_i^e[\![\boldsymbol{s}]\!] = \boldsymbol{o}$, then all of the resulting states τ are reachable.
2. $(\sigma, \tau) \in \rho(\forall i{:}C\ f(\boldsymbol{s})' = \theta\,\&\,\chi)$ iff, there is a function $\varphi{:}[0, r] \to \mathcal{S}$ for some $r \geq 0$ with $\varphi(0) = \sigma$ and $\varphi(r) = \tau$ satisfying the following conditions. At each time $t \in [0, r]$, state $\varphi(t)$ is identical to σ, except that at each position $\boldsymbol{o}$ of f: if $\sigma_i^e[\![\boldsymbol{s}]\!] = \boldsymbol{o}$ for some object $e \in \sigma(C)$, then, at each time $\zeta \in [0, r]$:
   - The differential equations hold and derivatives exist (trivial for $r = 0$):
   $$\frac{\mathsf{d}\,(\varphi(t)_i^e[\![f(\boldsymbol{s})]\!])}{\mathsf{d}t}(\zeta) = (\varphi(\zeta)_i^e[\![\theta]\!])$$
   - The evolution domain is respected: $\varphi(\zeta)_i^e \models \chi$.

   If there are multiple objects e giving the same position $\sigma_i^e[\![\boldsymbol{s}]\!] = \boldsymbol{o}$, then all of the resulting states τ are reachable.
3. $\rho(?\chi) = \{(\sigma, \sigma)\ :\ \sigma \models \chi\}$
4. $\rho(\alpha \cup \beta) = \rho(\alpha) \cup \rho(\beta)$
5. $\rho(\alpha; \beta) = \{(\sigma, \tau)\ :\ (\sigma, z) \in \rho(\alpha) \text{ and } (z, \tau) \in \rho(\beta) \text{ for a state } z\}$
6. $(\sigma, \tau) \in \rho(\alpha^*)$ iff there is an $n \in \mathbb{N}$ with $n \geq 0$ and there are states $\sigma = \sigma_0, \ldots, \sigma_n = \tau$ such that $(\sigma_i, \sigma_{i+1}) \in \rho(\alpha)$ for all $0 \leq i < n$.

The semantics is *explicit change*: nothing changes unless an assignment or differential equation specifies how. In cases 1–2, only f changes and only at positions of the form $\sigma_i^e[\![\boldsymbol{s}]\!]$ for some interpretation $e \in \sigma(C)$ of i. If there are multiple such e that affect the same position $\boldsymbol{o}$, any of those changes can take effect by a nondeterministic choice. QHP $\forall i{:}C\ x := a(i)$ may change x to *any* $a(i)$. Hence, $[\forall i{:}C\ x := a(i)]\phi(x) \equiv \forall i{:}C\ \phi(a(i))$ and $\langle\forall i{:}C\ x := a(i)\rangle\phi(x) \equiv \exists i{:}C\ \phi(a(i))$. Similarly, x can evolve along $\forall i{:}C\ x' = a(i)$ with any of the slopes $a(i)$. But evolutions cannot start with slope $a(c)$ and then switch to a different slope $a(d)$ later. Any choice for i is possible but i remains unchanged during each evolution.

We call a quantified assignment $\forall i:C\ f(\boldsymbol{s}):=\theta$ or a quantified differential equation $\forall i:C\ f(\boldsymbol{s})'=\theta\,\&\,\chi$ *injective* iff there is at most one e satisfying cases 1–2. We call quantified assignments and quantified differential equations *schematic* iff $\boldsymbol{s}$ is i (thus injective) and the only arguments to function symbols in θ are i. Schematic quantified differential equations like $\forall i:C\ f(i)'=a(i)\,\&\,\chi$ are very common, because distributed hybrid systems often have a family of similar differential equations replicated for multiple participants i. Their synchronization typically comes from discrete communication on top of their continuous dynamics, less often from complicated, physically coupled differential equations.

5 Actual Existence and Object Creation

Actual Existence For the Qd$\mathcal{L}$ semantics, we chose constant domain semantics, i.e., all states share the same domains. Thus quantifiers range over all possible objects (*possibilist quantification*) not just over active existing objects (*actualist quantification* in varying domains) [17]. In order to distinguish between *actual objects* that exist in a state, because they have already been created and can now actively take part in its evolution, versus *possible objects* that still passively await creation, we use function symbol $\mathsf{E}(\cdot)$. Symbol $\mathsf{E}(\cdot)$ is similar to existence predicates in first-order modal logic [17], but its value can be assigned to in QHPs.

Object Creation For term i of type $C\neq\mathbb{R}$, $\mathsf{E}(i)=1$ represents that the object denoted by i has been created and actually exists. We use $\mathsf{E}(i)=0$ to represent that i has not been created. Object creation amounts to changing the interpretation of $\mathsf{E}(i)$. For an object denoted by i that has not been created ($\mathsf{E}(i)=0$), object creation corresponds to the state change caused by assignment $\mathsf{E}(i):=1$. With quantified assignments and function symbols, *object creation* is definable:

$$n:=\mathsf{new}\,C\ \equiv\ (\forall j:C\ n:=j);\ ?(\mathsf{E}(n)=0);\ \mathsf{E}(n):=1$$

It assigns an arbitrary j of type C to n that did not exist before ($\mathsf{E}(n)=0$) and adjusts existence ($\mathsf{E}(n):=1$). *Disappearance* of object i corresponds to $\mathsf{E}(i):=0$. Our choice avoids semantic subtleties of varying domains about the meaning of free variables denoting non-existent objects as in free logics [17]. Denotation is standard. Terms may just denote objects that have not been activated yet. This is useful to initialize new objects (e.g., $x(n):=8$) before activation ($\mathsf{E}(n):=1$).

Actualist Quantifiers We define abbreviations for *actualist quantifiers* in formulas / quantified assignments / quantified differential equations that range only over previously *created objects*, similar to relativization in modal logic [17]:

$$\begin{aligned}
\forall i:C!\ \phi &\equiv \forall i:C\ (\mathsf{E}(i)=1\rightarrow\phi)\\
\exists i:C!\ \phi &\equiv \exists i:C\ (\mathsf{E}(i)=1\wedge\phi)\\
\forall i:C!\ f(\boldsymbol{s}):=\theta &\equiv \forall i:C\ f(\boldsymbol{s}):=(\mathsf{if}\,\mathsf{E}(i)=1\,\mathsf{then}\,\theta\,\mathsf{else}\,f(\boldsymbol{s})\,\mathsf{fi})\\
\forall i:C!\ f(\boldsymbol{s})'=\theta &\equiv \forall i:C\ f(\boldsymbol{s})'=(\mathsf{if}\,\mathsf{E}(i)=1\,\mathsf{then}\,\theta\,\mathsf{else}\,0\,\mathsf{fi})\ \equiv\ \forall i:C\ f(\boldsymbol{s})'=\mathsf{E}(i)\theta
\end{aligned}$$

The last 2 cases define quantified state change for actually existing objects using conditional terms that choose effect θ if $\mathsf{E}(i)=1$ and choose no effect (retaining the old value $f(\boldsymbol{s})$ or evolving with slope 0) if $\mathsf{E}(i)=0$. Notation $C!$ signifies that the quantifier domain is restricted to actually existing objects of type C.

We generally assume that QHPs involve only quantified assignments / differential equations that are restricted to created objects, because real systems only affect objects that are physically present, not those that will be created later. We still treat actualist quantification over $C!$ as a defined notion, in order to simplify the semantics and proof calculus by separating object creation from quantified state change rules in a modular way. If only finitely many objects have been created in the initial state (say 0), then it is easy to see that only finitely many new objects will be created with finitely many such QHP transitions, because each quantified state change for $C!$ only ranges over a finite domain then. We thus assume $\mathsf{E}(\cdot)$ to have *(unbounded but) finite support*, i.e., each state only has a finite number of positions i at which $\mathsf{E}(i)=1$. This makes sense in practice, because there is a varying but still finite numbers of participants (e.g., cars).

Example In order to restrict the dynamics and properties in the car control examples of Section 3 to created and physically present cars, we simply replace each occurrence of $\forall i:C$ with $\forall i:C!$. A challenging feature of distributed car control, however, is that new cars may appear dynamically from on-ramps (Fig. 1) changing the set of active objects. To model this, we consider the following QHP:

$$DCCS \;\equiv\; (n := \mathsf{new}\, C;\, (?\forall i:C!\ \mathcal{M}(i,n));\, \forall i:C!\ (x(i)'' = a(i)))^* \qquad (3)$$

It creates a new car n at an arbitrary position $x(n)$ satisfying compatibility condition $\mathcal{M}(i,n)$ with respect to all other created cars i. Hence *DCCS* allows new cars to appear, but not drop right out of the sky in front of a fast car or run at Mach 8 only 10ft away. When cars appear into the horizon from on-ramps, this condition captures that a car is only allowed to join the lane ("appear" into the model world) if it cannot cause a crash with other existing cars (Fig. 1). Unboundedly many cars may appear during the operation of *DCCS* and change the system dimension arbitrarily, because of the repetition operator *.

DCCS is simple but shows how properties of distributed hybrid systems can be expressed in $\mathsf{Qd}\mathcal{L}$. Structural dynamics corresponds to assignments to function terms. Say, $f(i)$ is the car registered by communication as the car following car i. Then a term $d(i,f(i))$, which denotes the minimum safety distance negotiated between car i and its follower, is a crucial part of the system dynamics. Restructuring the system in response to lane change corresponds to assigning a new value to $f(i)$, which impacts the value of $d(i,f(i))$ in the system dynamics.

6 Proof Calculus

In Fig. 2, we present a proof calculus for $\mathsf{Qd}\mathcal{L}$ formulas. We use the sequent notation informally for a systematic proof structure. With finite sets of formulas

for the *antecedent* Γ and *succedent* Δ, *sequent* $\Gamma\rightarrow\Delta$ is an abbreviation for the formula $\bigwedge_{\phi\in\Gamma}\phi \rightarrow \bigvee_{\psi\in\Delta}\psi$. The calculus uses standard proof rules for propositional logic with cut rule (not shown). The proof rules are used backwards from the *conclusion* (goal below horizontal bar) to the *premisse*s (subgoals above bar).

In the calculus, we use substitutions that take effect within formulas and programs (defined as usual). Only admissible substitutions are applicable, which is crucial for soundness. An application of a substitution σ is *admissible* if no replaced term θ occurs in the scope of a quantifier or modality binding a symbol in θ or in its replacement $\sigma\theta$. A modality *binds* a symbol f iff it contains an assignment to f (like $\forall i:C\ f(\boldsymbol{s}):=\theta$) or a differential equation containing a $f(\boldsymbol{s})'$ (like $\forall i:C\ f(\boldsymbol{s})'=\theta$). The substitutions in Fig. 2 that insert a term θ into $\phi(\theta)$ also have to be admissible for the proof rules to be applicable.

Regular Rules The next proof rules axiomatize sequential composition ([;],⟨;⟩), nondeterministic choice ([∪],⟨∪⟩), and test ([?],⟨?⟩) of regular programs as in dynamic logic [10]. Like other rules in Fig. 2, these rules do not contain sequent symbol $\rightarrow$, i.e., they can be applied to any subformula. These rules represent (directed) equivalences: conclusion and premiss are equivalent.

Quantified Differential Equations Rules [′],⟨′⟩ handle continuous evolutions for quantified differential equations with first-order definable solutions. Given a solution for the quantified differential equation system with symbolic initial values $f(\boldsymbol{s})$, continuous evolution along differential equations can be replaced with a quantified assignment $\forall i:C\ \mathcal{S}(t)$ corresponding to the solution (footnote 1 in Fig. 2), and an additional quantifier for evolution time t. In [′], postcondition ϕ needs to hold *for all* evolution durations t. In ⟨′⟩, it needs to hold after *some* duration t. The constraint on χ restricts the continuous evolution to remain in the evolution domain region χ at all intermediate times $\tilde{t}\leq t$.

For schematic cases like $\forall i:C\ f(i)'=a(i)$, first-order definable solutions can be obtained by adding argument i to first-order definable solutions of the deparametrized version $f'=a$. We only present proof rules for first-order definable solutions of quantified differential equations here. See [11] for other proof rules.

Quantified Assignments Rules [:=],⟨:=⟩ handle quantified assignments (both are equivalent for the injective case, i.e., a match for at most one i). Their effect depends on whether the quantified assignment $\forall i:C\ f(\boldsymbol{s}):=\theta$ *matches* $f(\boldsymbol{u})$, i.e., there is a choice for i such that $f(\boldsymbol{u})$ is affected by the assignment, because $\boldsymbol{u}$ is of the form $\boldsymbol{s}$ for some i. If it matches, the premiss uses the term θ assigned to $f(\boldsymbol{s})$ instead of $f(\boldsymbol{u})$, either for all possible $i:C$ that match $f(\boldsymbol{u})$ in case of [:=], or for some of those $i:C$ in case of ⟨:=⟩. Otherwise, the occurrence of f in $\phi(f(\boldsymbol{u}))$ will be left unchanged. Rules [:=],⟨:=⟩ make a case distinction on matching by if-then-else. In all cases, the original quantified assignment $\forall i:C\ f(\boldsymbol{s}):=\theta$, which we abbreviate by $\mathcal{A}$, will be applied to $\boldsymbol{u}$ in the premiss, because the value of argument $\boldsymbol{u}$ may also be affected by $\mathcal{A}$, recursively. Rule *skip* characterizes

$$([;])\ \frac{[\alpha][\beta]\phi}{[\alpha;\beta]\phi} \qquad ([\cup])\ \frac{[\alpha]\phi \wedge [\beta]\phi}{[\alpha\cup\beta]\phi} \qquad ([?])\ \frac{\chi\to\psi}{[?\chi]\psi}$$

$$(\langle;\rangle)\ \frac{\langle\alpha\rangle\langle\beta\rangle\phi}{\langle\alpha;\beta\rangle\phi} \qquad (\langle\cup\rangle)\ \frac{\langle\alpha\rangle\phi \vee \langle\beta\rangle\phi}{\langle\alpha\cup\beta\rangle\phi} \qquad (\langle?\rangle)\ \frac{\chi\wedge\psi}{\langle?\chi\rangle\psi}$$

$$([']) \ \frac{\forall t{\geq}0\,\big((\forall 0{\leq}\tilde{t}{\leq}t\,[\forall i:C\ \mathcal{S}(\tilde{t})]\chi)\to[\forall i:C\ \mathcal{S}(t)]\phi\big)}{[\forall i:C\ f(\boldsymbol{s})'=\theta\,\&\,\chi]\phi}\ {}^{1}$$

$$(\langle'\rangle)\ \frac{\exists t{\geq}0\,\big((\forall 0{\leq}\tilde{t}{\leq}t\,\langle\forall i:C\ \mathcal{S}(\tilde{t})\rangle\chi)\wedge\langle\forall i:C\ \mathcal{S}(t)\rangle\phi\big)}{\langle\forall i:C\ f(\boldsymbol{s})'=\theta\,\&\,\chi\rangle\phi}\ {}^{1}$$

$$([:=])\ \frac{\mathsf{if}\,\exists i:C\ \boldsymbol{s}=[\mathcal{A}]\boldsymbol{u}\,\mathsf{then}\,\forall i:C\ (\boldsymbol{s}=[\mathcal{A}]\boldsymbol{u}\to\phi(\theta))\,\mathsf{else}\,\phi(f([\mathcal{A}]\boldsymbol{u}))\,\mathsf{fi}}{\phi([\forall i:C\ f(\boldsymbol{s}):=\theta]f(\boldsymbol{u}))}\ {}^{2}$$

$$(\langle:=\rangle)\ \frac{\mathsf{if}\,\exists i:C\ \boldsymbol{s}=\langle\mathcal{A}\rangle\boldsymbol{u}\,\mathsf{then}\,\exists i:C\ (\boldsymbol{s}=\langle\mathcal{A}\rangle\boldsymbol{u}\wedge\phi(\theta))\,\mathsf{else}\,\phi(f(\langle\mathcal{A}\rangle\boldsymbol{u}))\,\mathsf{fi}}{\phi(\langle\forall i:C\ f(\boldsymbol{s}):=\theta\rangle f(\boldsymbol{u}))}\ {}^{2}$$

$$(skip)\ \frac{\Upsilon([\forall i:C\ f(\boldsymbol{s}):=\theta]\boldsymbol{u})}{[\forall i:C\ f(\boldsymbol{s}):=\theta]\,\Upsilon(\boldsymbol{u})}\ {}^{3} \qquad ([:*])\ \frac{\forall j:C\ \phi(\theta)}{[\forall j:C\ n:=\theta]\phi(n)} \qquad (\langle:*\rangle)\ \frac{\exists j:C\ \phi(\theta)}{\langle\forall j:C\ n:=\theta\rangle\phi(n)}$$

$$(ex)\ \frac{true}{\exists n:C\ \mathrm{E}(n)=0}$$

$$(\exists\mathrm{r})\ \frac{\Gamma\to\phi(\theta),\exists x\,\phi(x),\Delta}{\Gamma\to\exists x\,\phi(x),\Delta}\ {}^{4} \qquad (\forall\mathrm{r})\ \frac{\Gamma\to\phi(f(X_1,..,X_n)),\Delta}{\Gamma\to\forall x\,\phi(x),\Delta}\ {}^{5}$$

$$(\forall\mathrm{l})\ \frac{\Gamma,\phi(\theta),\forall x\,\phi(x)\to\Delta}{\Gamma,\forall x\,\phi(x)\to\Delta}\ {}^{4} \qquad (\exists\mathrm{l})\ \frac{\Gamma,\phi(f(X_1,..,X_n))\to\Delta}{\Gamma,\exists x\,\phi(x)\to\Delta}\ {}^{5}$$

$$(\mathrm{i}\forall)\ \frac{\mathrm{QE}(\forall X,Y\,(\mathsf{if}\,\boldsymbol{s}=\boldsymbol{t}\,\mathsf{then}\,\Phi(X)\to\Psi(X)\,\mathsf{else}\,\Phi(X)\to\Psi(Y)\,\mathsf{fi}))}{\Phi(f(\boldsymbol{s}))\to\Psi(f(\boldsymbol{t}))}\ {}^{6}$$

$$(\mathrm{i}\exists)\ \frac{\mathrm{QE}(\exists X\ \bigwedge_i(\Phi_i\to\Psi_i))}{\Phi_1\to\Psi_1\quad\ldots\quad\Phi_n\to\Psi_n}\ {}^{7}$$

$$([]gen)\ \frac{\phi\to\psi}{\Gamma,[\alpha]\phi\to[\alpha]\psi,\Delta} \qquad (\langle\rangle gen)\ \frac{\phi\to\psi}{\Gamma,\langle\alpha\rangle\phi\to\langle\alpha\rangle\psi,\Delta} \qquad (ind)\ \frac{\phi\to[\alpha]\phi}{\Gamma,\phi\to[\alpha^*]\phi,\Delta}$$

$$(con)\ \frac{v>0\wedge\varphi(v)\to\langle\alpha\rangle\varphi(v-1)}{\Gamma,\exists v\,\varphi(v)\to\langle\alpha^*\rangle\exists v{\leq}0\,\varphi(v),\Delta}\ {}^{8}$$

[1] $t,\tilde{t}$ are new variables, $\forall i:C\ \mathcal{S}(t)$ is the quantified assignment $\forall i:C\ f(\boldsymbol{s}):=y_{\boldsymbol{s}}(t)$ with solutions $y_{\boldsymbol{s}}(t)$ of the (injective) differential equations and $f(\boldsymbol{s})$ as initial values.

[2] Occurrence $f(\boldsymbol{u})$ in $\phi(f(\boldsymbol{u}))$ is not in scope of a modality (admissible substitution) and we abbreviate assignment $\forall i:C\ f(\boldsymbol{s}):=\theta$ by $\mathcal{A}$, which is assumed to be injective.

[3] $f\neq\Upsilon$ and the quantified assignment $\forall i:C\ f(\boldsymbol{s}):=\theta$ is injective. The same rule applies for $\langle\forall i:C\ f(\boldsymbol{s}):=\theta\rangle$ instead of $[\forall i:C\ f(\boldsymbol{s}):=\theta]$.

[4] θ is an arbitrary term, often a new logical variable.

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

[6] X,Y are new variables of sort $\mathbb{R}$. QE needs to be applicable in the premiss.

[7] Among all open branches, the free (existential) logical variable X of sort $\mathbb{R}$ only occurs in the branches $\Phi_i\to\Psi_i$. QE needs to be defined for the formula in the premiss, especially, no Skolem dependencies on X occur.

[8] logical variable v does not occur in α.

Fig. 2. Rule schemata of the proof calculus for quantified differential dynamic logic.

that quantified assignments to f have no effect on all other operators $\Upsilon \neq f$ (including other function symbols, $\wedge$, if then else fi), so that only argument $\boldsymbol{u}$ is affected by prefixing $\mathcal{A}$ but Υ remains unchanged.

Rules $[:=]$,$\langle:=\rangle$,*skip* also apply for assignments without quantifiers, which correspond to vacuous quantification $\forall i : C$ where i does not occur anywhere. Rules $[:*]$,$\langle:*\rangle$ reduce nondeterministic assignments to universal or existential quantification. For nondeterministic differential equations, see [11].

Object Creation Given our definition of new C as a QHP from Section 5, object creation can be proven by the other proof rules in Fig. 2. In addition, axiom *ex* expresses that, for sort $C \neq \mathbb{R}$, there always is a new object n that has not been created yet ($\mathrm{E}(n) = 0$), because domains are infinite.

Quantifiers For quantifiers, we cannot just use standard rules [18], because these are for uninterpreted first-order logic and work by instantiating quantifiers, eagerly as in ground tableaux or lazily by unification as in free variable tableaux [18]. Qd$\mathcal{L}$ is based on first-order logic interpreted over the reals [19]. A formula like $\exists a : \mathbb{R}\ \forall x : \mathbb{R}\ (x^2 + a > 0)$ cannot be proven by instantiating quantifiers but is still valid for reals. Unfortunately, the decision procedure for real arithmetic, *quantifier elimination* (QE) in the theory of real-closed fields [19], cannot be applied to formulas with modalities either, because these are quantified reachability statements. Even in discrete dynamic logic, quantifiers plus modalities make validity Π^1_1-complete [10]. Also QE cannot handle sorts $C \neq \mathbb{R}$.

Instead, our Qd$\mathcal{L}$ proof rules combine quantifier handling of many-sorted logic based on instantiation with theory reasoning by QE for the theory of reals. Figure 2 shows rules for quantifiers that combine with decision procedures for real-closed fields. Classical instantiation is sound for sort $\mathbb{R}$, just incomplete.

Rules $\exists$r and $\forall$l instantiate with arbitrary terms θ, including a new free variable X, where $\exists$r and $\forall$l become the usual γ-rules [18, 17]. Rules $\forall$r and $\exists$l correspond to the δ-rule [18]. As in our previous work [5], rules i$\forall$ and i$\exists$ reintroduce and eliminate quantifiers over $\mathbb{R}$ once QE is applicable, as the remaining constraints are first-order in the respective variables. Unlike in previous work, however, functions and different argument vectors can occur in Qd$\mathcal{L}$. If the argument vectors $\boldsymbol{s}$ and $\boldsymbol{t}$ in i$\forall$ have the same value, the same variable X can be reintroduced for $f(\boldsymbol{s})$ and $f(\boldsymbol{t})$, otherwise different variables $X \neq Y$ have to be used. Rule i$\exists$ merges all proof branches containing (existential) variable X, because X has to satisfy all branches simultaneously. It thus has multiple conclusions. See [5] for merging and for lifting QE to the presence of function symbols, including formulas that result from the base theory by substitution.

Global Rules The rules in the last block depend on the truth of their premisses in all states reachable by α, thus the context Γ, Δ cannot be used in the premiss. Rules $[]gen$,$\langle\rangle gen$ are Gödel generalization rules and *ind* is an induction schema for loops with *inductive invariant* ϕ [10]. Similarly, *con* generalizes Harel's convergence rule [10] to the hybrid case with decreasing *variant* φ [5].

7 Soundness and Completeness

The verification problem for distributed hybrid systems has *three independent sources* of undecidability. Thus, no verification technique can be effective. Hence, QdL cannot be effectively axiomatizable. Both its discrete and its continuous fragments alone are subject to Gödel's incompleteness theorem [5]. The fragment with only structural and dimension-changing dynamics is not effective either, because it can encode two-counter machines. The standard way to show adequacy of proof calculi for problems that are not effective is to prove completeness relative to an oracle for handling a fragment of the logic. Unlike in Cook/Harel relative completeness for discrete programs [10], however, QdL cannot be complete relative to the fragment of the data logic ($\mathbb{R}$), because real arithmetic is decidable. Instead, we prove that our QdL calculus is a complete axiomatization relative to an oracle for the fragment of QdL that has only quantified differential equations in modalities. We replace rules $[']$,$\langle' \rangle$ with an oracle and show that the QdL calculus would be complete if only we had complete replacements for $[']$,$\langle' \rangle$. The calculus completely lifts *any* approximation of this oracle to the full QdL!

Theorem 1 (Axiomatization). *The calculus in Fig. 2 is a sound and complete axiomatization of* QdL *relative to quantified differential equations; see [20].*

This shows that properties of distributed hybrid systems can be proven to exactly the same extent to which properties of quantified differential equations can be proven. Proof-theoretically, the QdL calculus completely lifts verification techniques for quantified continuous dynamics to distributed hybrid dynamics.

8 Distributed Car Control Verification

With the QdL calculus and the compatibility condition $\mathcal{M}(i,j)$ from eqn. (2), we can easily prove collision freedom in the distributed car control system (3):

$$(\forall i,j : C!\ \mathcal{M}(i,j)) \rightarrow \\ [(n := \mathsf{new}\, C; ?\forall i : C!\ \mathcal{M}(i,n); \forall i : C!\ (x(i)'' = a(i)))^*]\ \forall i{\neq}j : C!\ x(i){\neq}x(j)$$

See [20] for a formal QdL proof of this QdL formula, which proves collision freedom despite dynamic appearance of new cars, following the pattern of (1).

9 Conclusions

We have introduced a system model and semantics for dynamic distributed hybrid systems together with a compositional verification logic and proof calculus. We believe this is the *first formal verification approach for distributed hybrid dynamics*, where structure and dimension of the system can evolve jointly with the discrete and continuous dynamics. We have proven our calculus to be a *sound and complete axiomatization* relative to quantified differential equations.

Our calculus proves collision avoidance in distributed car control with dynamic appearance of new cars on the road, which is out of scope for other approaches.

Future work includes modular concurrency in distributed hybrid systems, which is already challenging in discrete programs.

Acknowledgments I want to thank Frank Pfenning for his helpful comments.

References

1. Hsu, A., Eskafi, F., Sachs, S., Varaiya, P.: Design of platoon maneuver protocols for IVHS. PATH Research Report UCB-ITS-PRR-91-6, UC Berkeley (1991)
2. Dowek, G., Muñoz, C., Carreño, V.A.: Provably safe coordinated strategy for distributed conflict resolution. In: AIAA Proceedings, AIAA-2005-6047. (2005)
3. Henzinger, T.A.: The theory of hybrid automata. In: LICS, IEEE (1996) 278–292
4. Attie, P.C., Lynch, N.A.: Dynamic input/output automata: A formal model for dynamic systems. In Larsen, K.G., Nielsen, M., eds.: CONCUR. Volume 2154 of LNCS., Springer (2001) 137–151
5. Platzer, A.: Differential dynamic logic for hybrid systems. J Autom Reas **41**(2) (2008) 143–189
6. Deshpande, A., Göllü, A., Varaiya, P.: SHIFT: A formalism and a programming language for dynamic networks of hybrid automata. In: Hybrid Systems. Volume 1273 of LNCS., Springer (1996) 113–133
7. Rounds, W.C.: A spatial logic for the hybrid π-calculus. In Alur, R., Pappas, G.J., eds.: HSCC. Volume 2993 of LNCS., Springer (2004) 508–522
8. Kratz, F., Sokolsky, O., Pappas, G.J., Lee, I.: R-Charon, a modeling language for reconfigurable hybrid systems. [21] 392–406
9. Meseguer, J., Sharykin, R.: Specification and analysis of distributed object-based stochastic hybrid systems. [21] 460–475
10. Harel, D., Kozen, D., Tiuryn, J.: Dynamic logic. MIT Press, Cambridge (2000)
11. Platzer, A.: Differential-algebraic dynamic logic for differential-algebraic programs. J. Log. Comput. **20**(1) (2010) 309–352 DOI 10.1093/logcom/exn070.
12. Beckert, B., Platzer, A.: Dynamic logic with non-rigid functions: A basis for object-oriented program verification. In Furbach, U., Shankar, N., eds.: IJCAR. Volume 4130 of LNCS., Springer (2006) 266–280
13. Rümmer, P.: Sequential, parallel, and quantified updates of first-order structures. In Hermann, M., Voronkov, A., eds.: LPAR. Volume 4246 of LNCS., Springer (2006) 422–436
14. Chaochen, Z., Ji, W., Ravn, A.P.: A formal description of hybrid systems. In Alur, R., Henzinger, T.A., Sontag, E.D., eds.: Hybrid Systems. Volume 1066 of LNCS., Springer (1995) 511–530
15. van Beek, D.A., Man, K.L., Reniers, M.A., Rooda, J.E., Schiffelers, R.R.H.: Syntax and consistent equation semantics of hybrid Chi. J. Log. Algebr. Program. **68**(1-2) (2006) 129–210
16. Kozen, D.: Kleene algebra with tests. ACM TOPLAS **19**(3) (1997) 427–443
17. Fitting, M., Mendelsohn, R.L.: First-Order Modal Logic. Kluwer (1999)
18. Fitting, M.: First-Order Logic and Automated Theorem Proving. Springer (1996)
19. Collins, G.E., Hong, H.: Partial cylindrical algebraic decomposition for quantifier elimination. J. Symb. Comput. **12**(3) (1991) 299–328
20. Platzer, A.: Quantified differential dynamic logic for distributed hybrid systems. Technical Report CMU-CS-10-126, SCS, Carnegie Mellon University (2010)
21. Hespanha, J.P., Tiwari, A., eds.: HSCC. Volume 3927 of LNCS., Springer (2006)