Verifying Hybrid Systems with Modal Kleene Algebra

Jonathan Julián Huerta y Munive and Georg Struth

Department of Computer Science, University of Sheffield
{jjhuertaymunive1,g.struth}@sheffield.ac.uk

Abstract. Modal Kleene algebras have been used for building verification components for imperative programs with Isabelle/HOL. We integrate this approach with recent Isabelle components for ordinary differential equations to build two verification components for hybrid systems, where continuous phase space dynamics complement the discrete dynamics on program stores. We demonstrate their feasibility by deriving the most important domain-specific rules of differential dynamic logic and discussing four simple examples.

1 Introduction

Hybrid systems integrate continuous dynamics and discrete control. Their verification is increasingly important, yet notoriously difficult: applications range from the control of chemical plants, finance and traffic systems to the coordination of autonomous vehicles or robots, the optimisation of mechanical systems and biosystems engineering [1, 29]. Mathematically, hybrid systems verification requires integrating dynamical systems, often modelled by ordinary differential equations (ODEs), into formalisms for their description and analysis, such as automata [15] or modal logics [9, 28]. A prominent approach is differential dynamic logic $\mathsf{d}\mathcal{L}$ [28], an extension of dynamic logic [14] to hybrid programs with domain-specific inference rules for ODEs and their solutions. It is well supported by the KeYmaera X tool [10] and has already proved its worth in numerous case studies [21, 24, 28].

Mathematical components for dynamical systems have recently been formalised in proof assistants such as Isabelle/HOL [25]. An impressive theory stack—from topology and measure theory to analysis [17] and ODEs [18, 20, 19]—is ready for use. In real-word software verification, such proof assistants play an important role already [6, 23, 22], and verification components based on modal Kleene algebra (MKA) [7, 11, 12, 13], which subsumes dynamic logic, exist in Isabelle as well. Yet an integration of the two theory stacks for $\mathsf{d}\mathcal{L}$-style hybrid program verification is missing.

Such an integration seems natural and interesting for several reasons. Firstly, the resulting components for $\mathsf{d}\mathcal{L}$-style hybrid program verification would be correct by construction relative to the proof assistant's small trustworthy core. The existence and uniqueness of solutions to ODEs, for instance, which is required in the $\mathsf{d}\mathcal{L}$ semantics, could be formally certified, in particular when these are computed by external tools. Secondly, such components could be engineered and used in open modular ways; limited only by the expressivity of the proof assistant's logic, the versatility of its mathematical components and the power of its provers and solvers. Specific components for particular classes of ODEs or decidable assertion languages, for instance for real-closed fields, could be developed and verified within the general framework. Finally, the integration would benefit from active communities that are driving the development of $\mathsf{d}\mathcal{L}$ and a wide range of mathematical components forward.

Our main contribution is the proof of concept for this integration: a formal reconstruction of $\mathsf{d}\mathcal{L}$ within MKA and Isabelle/HOL, resulting in prototypical $\mathsf{d}\mathcal{L}$-based verification components for this general purpose proof assistant.

More detailed technical contributions are: (i) a relational flow model of MKA, suitable for hybrid systems, and its formalisation in Isabelle; (ii) a predicate transformer semantics for hybrid programs using this model, and a simple generic $\mathsf{d}\mathcal{L}$-style Isabelle verification component based on equational reasoning with weakest liberal preconditions; (iii) an alternative MKA-based verification component using $\mathsf{d}\mathcal{L}$-style differential invariants instead of flows of ODEs; (iv) the derivation of domain-specific $\mathsf{d}\mathcal{L}$-style inference rules for these components within the relational flow semantics.

This is our approach in a nutshell: MKA provides variants of the laws of propositional dynamic logic for reasoning equationally about while programs without assignments, hence an algebraic predicate transformer semantics. Extant MKA-based verification components [12] capture the program store dynamics of assignments within the relational program store model of MKA. In Isabelle, a modular instantiation of MKA to this concrete semantics is supported by type polymorphism. Hence one can simply replace such discrete program stores with dynamical ones for hybrid programs; essentially with phase spaces. One can use evolution statements, which describe vector fields or systems of ODEs, in addition to assignment statements to model hybrid programs. The relational hybrid program semantics then uses flows (solutions of ODEs, provided they exist) for "updating" phase spaces together with update functions for assignments. This sets up an algebraic variant of $\mathsf{d}\mathcal{L}$ as an instance of MKA.

Our first Isabelle verification component asks users—and ultimately external solvers—to supply flows for ODEs and certify them internally. MKA and Isabelle's ODE components are then used for manipulating predicate transformers over phase spaces and verifying correctness specifications for hybrid programs by equational reasoning. Alternatively to manipulating explicit solutions, users can provide invariants for ODEs in verification proofs, as with $\mathsf{d}\mathcal{L}$, in our second verification component. Both are shown at work on a series of simple examples.

Our Isabelle components form a main contribution of this paper. They confirm the feasibility of the algebraic approach to $\mathsf{d}\mathcal{L}$. The main statements in this article have been formalised (cf. Appendix C). The complete formalisation, including a PDF proof document, is available online[1].

[1] `https://github.com/yonoteam/CPSVerification`

2 Modal Kleene Algebra

A *Kleene algebra* is a structure $(K, +, \cdot, 0, 1, ^*)$ such that $(K, +, \cdot, 0, 1)$ is a *dioid*, a semiring in which $+$ is idempotent. The Kleene star $^* : K \to K$ satisfies the unfold and induction axioms

$$1 + \alpha \cdot \alpha^* \leq \alpha^*, \qquad \gamma + \alpha \cdot \beta \leq \beta \to \alpha^* \cdot \gamma \leq \beta$$

and their opposites, where the order of multiplication is swapped. These axioms use the semilattice order, which is given by $\alpha \leq \beta \leftrightarrow \alpha + \beta = \beta$ on any dioid. The class of Kleene algebras is closed under opposition. The complete axioms for the algebras discussed in this article are listed in Appendix A.

Elements of K can represent programs: $+$ models nondeterministic choice, $\cdot$ sequential composition and * finite iteration; 0 models the abortive and 1 the ineffective program. Formally, programs are interpreted as binary relations over a state space S. In fact, the set $\mathcal{P}(S \times S)$ of all binary relations over S forms a *relation Kleene algebra* under $\cup$, relational composition ;, the empty relation $\emptyset$, the identity relation Id_S and the reflexive-transitive closure operation *. More realistic program semantics require modelling tests and assertions as well.

A *modal Kleene algebra* (MKA) [7] is a Kleene algebra K expanded by an *antidomain operation* $ad : K \to K$ and an *antirange operation* $ar : K \to K$. Antidomain is axiomatised by

$$ad\,\alpha \cdot \alpha = 0, \qquad ad\,\alpha + ad^2\,\alpha = 1, \qquad ad\,(\alpha \cdot \beta) \leq ad\,(\alpha \cdot ad^2\,\beta).$$

Antirange is antidomain in the opposite Kleene algebra. The operations interact via the axioms $ad^2 \circ ar^2 = ar^2$ and $ar^2 \circ ad^2 = ad^2$.

Intuitively, $ad\,\alpha$ models those states from which program α cannot be executed; the domain $d\,\alpha = ad^2\,\alpha$ of α models those states from which it can be. Dually, $ar\,\alpha$ models those states in which α cannot terminate; the range $r\,\alpha = ar^2\,\alpha$ models those states in which it can. The set $K_{ad} = \{\alpha \in K \mid ad\,\alpha = \alpha\}$ forms a boolean subalgebra of K bounded by 0 and 1. In K_{ad}, which is the same as K_{ar}, $+$ corresponds to join and $\cdot$ to meet; ad and ar correspond to complementation. Elements $p, q, r \in K_{ad}$ can therefore model tests and assertions. In particular, $p \cdot \alpha$ and $\alpha \cdot p$ model the domain and range restriction of α to states satisfying p. In relation MKAs, $ad\,R = \{(s, s) \mid \neg\exists s'.\ (s, s') \in R\}$. Hence $ad\,R \subseteq Id_S$ for all R. The class MKA is once again closed under opposition.

3 Modal Kleene Algebra and Dynamic Logic

On the one hand, MKA yields an algebraic semantics for simple while languages:

$$\alpha; \beta = \alpha \cdot \beta, \qquad \textbf{if}\ p\ \textbf{then}\ \alpha\ \textbf{else}\ \beta = p \cdot \alpha + \bar{p} \cdot \beta, \qquad \textbf{while}\ p\ \textbf{do}\ \alpha = (p \cdot \alpha)^* \cdot \bar{p},$$

writing $\bar{p}$ instead of $ad\,p$. On the other hand, the forward modal operators of propositional dynamic logic (PDL) can be defined as

$$|\alpha\rangle p = d\,(\alpha \cdot p), \qquad |\alpha]p = ad\,(\alpha \cdot ad\,p);$$

backward ones are obtained by opposition: $\langle\alpha|p = r\,(\alpha\cdot p)$, and $[\alpha|p = ar\,(\alpha\cdot ar\,p)$. This is consistent with the relational (Kripke) semantics of PDL [7, 12], where $|\alpha\rangle p$ models those states from which executing α may lead into states where p holds; and $|\alpha]p$ describes those states from which α must lead to states satisfying p. In particular, algebraic variants of the PDL axioms are derivable as theorems of MKA, so that the latter can be seen as an algebraic relative of the former.

It is well known from PDL that $|\alpha] : K_{ad} \to K_{ad}$ is a backward conjunctive predicate transformer on K_{ad}. In fact, $|\alpha]q$ models the weakest liberal precondition (wlp) of program α and postcondition q. The other modalities yield dual transformers that are less interesting for this article.

Predicate transformers are useful for specifying program correctness conditions and for verification condition generation. The identity

$$p \leq |\alpha]q$$

encodes the standard partial correctness specification for programs: if program α is executed from states where precondition p holds, and if it terminates, then postcondition q holds in the states where it does. One can thus calculate $|\alpha]q$ backwards over the program structure from q and check that the result is above p. Calculating $|\alpha]q$ for straight-line programs is completely equational; yet loops require invariants. To this end we use while loops with annotations: **while** p **inv** i **do** α = **while** p **do** α and calculate wlps recursively over the program structure as follows [12]. For all $p, q, i, t \in K_{ad}$ and $x, y \in K$,

$$\begin{gathered} |\alpha;\beta]q = |\alpha]|\beta]q, \\ |\mathbf{if}\ p\ \mathbf{then}\ \alpha\ \mathbf{else}\ \beta]q = (\bar{p} + |\alpha]q)(p + |\beta]q), \\ p \leq i \wedge i\bar{t} \leq q \wedge it \leq |\alpha]i \ \to\ p \leq |\mathbf{while}\ t\ \mathbf{inv}\ i\ \mathbf{do}\ \alpha]q. \end{gathered}$$

As binary relations form MKAs, the approach is consistent with the relational program semantics and the predicate transformer semantics obtained from it.

4 Integrating Discrete Program State Dynamics

Two important ingredients for program semantics and verification are still missing: a concrete relational semantics of the program store and program assignments, and rules for calculating wlps for these basic commands. To prepare for hybrid programs (see Appendix A for a syntax) we discuss stores and assignments in terms of discrete dynamical systems over state spaces.

Formally, a *dynamical system* [3, 30] is an *action* of a monoid $(M, \star, e)$ on a set or state space S, that is, a monoid morphism $\varphi : M \to S \to S$ into the *transformation monoid* $(S^S, \circ, id_S)$ on the function space S^S. Thus $\varphi\,(m \star n) = (\varphi\, m) \circ (\varphi\, n)$ and $\varphi\, e = id_S$. The first action axiom captures the inherent determinism of dynamical systems. Conversely, each transformation monoid $(S^S, \circ, id_S)$ determines a monoid action in which the action $\varphi : S^S \to S \to S$ is function application.

States of simple while programs are functions $s : V \to E$ from program variables in V to values in E. State spaces for such discrete dynamical systems are function spaces E^V. An update function $f_a : V \to (E^V \to E) \to E^V \to E^V$ for assignment commands can be defined as $f_a\, v\, e\, s = s[v \mapsto (e\, s)]$, where $f[a \mapsto b]$ updates function $f : A \to B$ in argument $a \in A$ by value $b \in B$ ($f[a \mapsto b]\, x$ is b if $x = a$ and $f\, x$ otherwise). The "expression" $e : E^V \to E$ is evaluated in state s to $e\, s$. Yet with a shallow embedding of data domains in Isabelle, e is, by its type, a function. The functions $f_a\, v\, e$ generate a transformation monoid, hence a monoid action of type $(E^V)^{(E^V)} \to E^V \to E^V$ on $(E^V)^{(E^V)}$. They also connect the concrete program state semantics with the wlp semantics used for verification condition generation.

The relational semantics of assignment statements can hence be defined as

$$(v := e) = \{(s, f_a\, v\, e\, s) \mid s \in E^V\}, \tag{1}$$

and the wlp for assignments be derived in the relational program semantics as

$$|v := e]\lceil Q\rceil = \lceil \lambda s.\ Q(s[v \mapsto (e\, s)])\rceil,$$

where $\lceil - \rceil$ embeds predicates into relations: $\lceil P\rceil = \{(s, s) \mid P\, s\}$.

Adding this wlp law for assignments to those for the program structure suffices for generating data-level verification conditions for while programs. The development also yields a hybrid encoding of dynamic logic, where the propositional part is captured algebraically and the rest in set theory.

We have already implemented verification components based on MKA in Isabelle/HOL [12, 13]. Apart from the uses outlined, they also support verification based on Hoare logic, verification of simple recursive programs, verification in the context of total correctness, verification by symbolic execution, as well as program construction, transformation and refinement.

5 Integrating Continuous Program State Dynamics

Hybrid programs require continuous dynamics as well as discrete ones. Actions φ are now of type $\mathbb{R} \to S \to S$. They are usually flows of systems of differential equations [3, 30] over a phase space S. In general, for each $s \in S$, the functions $\varphi_s : \mathbb{R} \to S$ given by $\varphi_s = \lambda t.\ \varphi\, t\, s$ are maximal smooth integral curves generated by vector fields f that assign tangent vectors to points of some manifold S, varying smoothly from point to point in t, and starting in (or passing through) point s. The tangent vector of $\varphi_s\, t$ at any point in S must then coincide with the value of f at that point. The set $\gamma\, s = \{\varphi\, t\, s \mid t \geq 0\}$ is the (positive) *orbit* of φ through s. We write γ^φ to indicate the dependency of orbits on flows.

Example 5.1 (Vector field of fluid). Velocity vectors describe the motion of a fluid in an (open) subset of $\mathbb{R}^3$. They form a vector field that associates a velocity with each particle at each point of space. The integral curves are the trajectories, orbits or evolutions of fluid particles through time, and $\varphi_s\, t$ describes the movement of a particle from initial position $s \in \mathbb{R}^3$ at time $t = 0$. □

Typical phase spaces for continuous dynamical systems are open subsets of $\mathbb{R}^n$. System models are typically differential equations. Our approach is currently limited to autonomous systems of ordinary differential equations (ODEs) [30]

$$x'\,t = f\,(x\,t),$$

where $x : \mathbb{R} \to X$, for any open $X \subseteq \mathbb{R}^n$, and Lipschitz continuous vector fields $f \in C^k(X, \mathbb{R}^n)$ for some $k \geq 1$. The Picard-Lindelöf Theorem guarantees that such systems have unique solutions for initial value problems $x'\,t = f\,(x\,t)$, $x\,t_0 = x_0$ within certain closed intervals $I(x_0)$ [30]. These solutions are integral curves $\varphi_{x_0}\,t$. We currently allow only functions for which $I(x_0) = \mathbb{R}$. The general case is left for future work. It is already supported by Isabelle.

Hybrid programs, by contrast, require phase spaces $\mathbb{R}^V$, where V is a countable set of variables [28]. This leads to some subtleties in our model, which are due to Isabelle's strongly typed setting. As hybrid programs change only finitely many variables, there is always a subspace of $\mathbb{R}^V$ isomorphic to some $\mathbb{R}^n$ on which the program acts and where existence and uniqueness theorems apply, and an orthogonal subspace, where nothing changes.

We are now ready to define an analogue of the assignment semantics from Section 4 for hybrid programs and derive an equation for the corresponding wlp operator. First we replace assignment statements in the definiendum of (1) by *evolution statements* for vector fields of the form

$$x_1' = f_1, \dots, x_n' = f_n,$$

or, more briefly, $x' = f$, where $x_i' = f_i$ is formally a pair $(x_i, f_i) : V \times (\mathbb{R}^V \to \mathbb{R})$. These, and the usual assignments, form the basic commands of hybrid programs. By analogy, we wish to replace the definiens of (1) by the flow $\varphi : \mathbb{R} \to \mathbb{R}^V \to \mathbb{R}^V$ for $x' = f$. Yet this requires checking its existence and uniqueness: a functional dependency of φ on f and x. For $X = \{x_1, \dots, x_n\}$, we define the predicates

$$\begin{aligned}
\mathsf{Act}_1\,\varphi\,s &\Leftrightarrow \forall t_1, t_2 \geq 0.\ \varphi\,(t_1 + t_2)\,s = \varphi\,t_1\,(\varphi\,t_2\,s),\\
\mathsf{Act}_2\,\varphi\,s &\Leftrightarrow \lambda x \in V.\ \varphi\,0\,s\ x = s\ x,\\
\mathsf{Diff}\,\varphi\,x\,f\,s &\Leftrightarrow \forall t \geq 0, 1 \leq i \leq n.\ (\lambda\,\tau.\,\varphi\,\tau\,s\,x_i)'\,t = f_i\,(\varphi\,t\,s),\\
\mathsf{Id}\,\varphi\,s &\Leftrightarrow \forall t \geq 0, y \in V \setminus X.\ \varphi\,t\,s\,y = s\,y,\\
\mathsf{Flow}_\exists\,\varphi\,x\,f\,s &\Leftrightarrow \mathsf{Act}_1\,\varphi\,s \wedge \mathsf{Act}_2\,\varphi\,s \wedge \mathsf{Diff}\,\varphi\,x\,f\,s \wedge \mathsf{Id}\,\varphi\,s,\\
\mathsf{Flow}\,\varphi\,x\,f\,s &\Leftrightarrow \mathsf{Flow}_\exists\,\varphi\,x\,f\,s \wedge \forall \psi.\ \mathsf{Flow}_\exists\,\psi\,x\,f\,s \to \varphi = \psi.
\end{aligned}$$

Act_1 and Act_2 check that φ is actually a flow; in addition, Act_2 checks that the flow solves the initial value problem for $x' = f$ at point s. Diff checks that the coordinates of the derivative of φ are equal to f_i, hence that the tangent vectors of the flow are equal to the vector field f at each point $\varphi\,s\,t$ along an integral curve. These three conditions are standard for dynamical systems [3, 30]. The fourth condition, Id, is particular to hybrid programs with countably many program variables. It guarantees that all variables that do not occur primed in $x' = f$ remain unchanged during the evolution. Finally, verifying uniqueness in practice requires checking that f satisfies conditions such as those in Picard-Lindelöf's theorem. In fact, the resulting uniqueness proof subsumes Act_1.

Next we define the relational flow semantics of evolution statements as

$$(\forall s \in \mathbb{R}^V.\ \mathsf{Flow}\,\varphi\,x\,f\,s) \to (x' = f) = \{(s, \varphi\,t\,s) \mid s \in \mathbb{R}^V \wedge t \geq 0\}, \qquad (2)$$

by analogy with (1). Each initial state s is thus related to its orbit $\gamma^\varphi\,s$. The wlp for evolution statements can now be calculated in this semantics.

Proposition 5.2. *Let $\varphi : \mathbb{R} \to \mathbb{R}^V \to \mathbb{R}^V$, $Q : \mathbb{R}^V \to \mathbb{B}$, $f_i : \mathbb{R}^V \to \mathbb{R}$ and $x_i \in V$. Then*

$$(\forall s \in \mathbb{R}^V.\ \mathsf{Flow}\,\varphi\,x\,f\,s) \ \to\ |x' = f]\lceil Q\rceil = \lceil \lambda s.\ \forall t \geq 0.\ Q\,(\varphi\,t\,s)\rceil.$$

Alternatively, we can write $\lceil \lambda s.\ \forall t \geq 0.\ Q\,s[x_1 \mapsto \varphi_s\,t\,x_1, \dots, x_n \mapsto \varphi_s\,t\,x_n]\rceil$ for the right-hand side of the wlp-identity, where $f[a_1 \mapsto b_1, \dots, a_m \mapsto b_m]$ indicates simultaneous function update. By definition, and consistently with $\mathsf{d}\mathcal{L}$, postconditions thus hold along the orbit of each particular state s, including s itself. Proposition 5.2 implies a generic inference rule for proving correctness specifications of evolution statements in the style of Hoare logic.

Lemma 5.3. *For $\varphi : \mathbb{R} \to \mathbb{R}^V \to \mathbb{R}^V$, $P, Q : \mathbb{R}^V \to \mathbb{B}$, $f_i : \mathbb{R}^V \to \mathbb{R}$ and $x_i \in V$,*

$$\frac{\forall s.\ \mathsf{Flow}\,\varphi\,x\,f\,s \qquad \forall s.\ P\,s \to \forall t \geq 0.\ Q\,(\varphi_s\,t)}{\lceil P\rceil \subseteq |x' = f]\lceil Q\rceil}\ \mathit{solve}$$

This rule can be used freely with the Hoare logic within MKA for hybrid program verification in the relational flow semantics. Proposition 5.2, by contrast, augments the predicate transformer laws from Section 3. The set-up of these rules requires users to supply flows for evolution statements and then certify that these actually solve the corresponding ODEs. See Section 9 for examples.

Example 5.4 (Motion with constant velocity). The evolution statement $x' = \lambda\,s.\ s\,v$ represents the ODE $x' = v$, where v is a constant, and notation is overloaded. For each initial value x_0, the flow $\varphi_{x_0}\,t = v \cdot t + x_0$ solves this ODE for all $t \geq 0$ in the phase space $\mathbb{R}$. With hybrid programs, we consider the phase space $\mathbb{R}^V$, where only the variable x changes. The relational flow semantics for evolution statements therefore requires, for all $t \geq 0$ and all $s \in \mathbb{R}^V$, the flow

$$\varphi_s\,t\,y = \begin{cases} (s\,v) \cdot t + (s\,x), & \text{if } y = x, \\ s\,y, & \text{if } y \in V \setminus \{x\}. \end{cases}$$

For postcondition $\lambda s.\ s\,x > 1$ we can then calculate

$$\begin{aligned} |x' = \lambda s.\ s\,v]\lceil \lambda s.\ s\,x > 1\rceil &= \lceil \lambda s.\ \forall t \geq 0.\ (\lambda s.\ s\,x > 1)\,s[x \mapsto (s\,v) \cdot t + (s\,x)]\rceil \\ &= \lceil \lambda s.\ \forall t \geq 0.\ (s\,v) \cdot t + (s\,x) > 1\rceil. \end{aligned}$$

Hence the wlp holds precisely of those states $s \in \mathbb{R}^V$ where $(s\,v) \cdot t + (s\,x) > 1$ for all $t \geq 0$. An Isabelle verification is presented in Section 9. □

6 Guarded Evolutions

Applications of dynamical systems and hybrid programs often depend on boundary conditions and similar constraints. These are called *guards* in $\mathsf{d}\mathcal{L}$, and they are restricted to first-order formulas of real arithmetic. A flow is supposed to satisfy a guard during an entire evolution, just like a postcondition. With Isabelle, we support arbitrary higher-order formulas or predicates as guards.

A *guarded evolution statement* [28] with guard G has the form

$$x_1' = f_1, \dots, x_n' = f_n \,\&\, G, \qquad \text{or more briefly,} \quad x' = f \,\&\, G.$$

Its relational flow semantics is captured by the flow predicate

$$\mathsf{Flow}_G\, \varphi\, x\, f\, G\, s \Leftrightarrow \mathsf{Flow}\, \varphi\, x\, f\, s \wedge \forall t \geq 0.\ G\, (\varphi\, t\, s).$$

and (2) with Flow_G in place of Flow. We can proceed like in Section 5.

Proposition 6.1. *Let* $\varphi : \mathbb{R} \to \mathbb{R}^V \to \mathbb{R}^V$, $G, Q : \mathbb{R}^V \to \mathbb{B}$, $f_i : \mathbb{R}^V \to \mathbb{R}$ *and* $x_i \in V$. *Then*

$$(\forall s \in \mathbb{R}^V.\mathsf{Flow}_G\, \varphi\, x\, f\, G\, s) \to |x' = f \,\&\, G]\lceil Q\rceil = \lceil \lambda s. \forall t \geq 0. G\, (\varphi_s\, t) \to Q\, (\varphi_s\, t)\rceil.$$

Lemma 6.2. *Let* $\varphi : \mathbb{R} \to \mathbb{R}^V \to \mathbb{R}^V$, $P, G, Q : \mathbb{R}^V \to \mathbb{B}$, $f_i : \mathbb{R}^V \to \mathbb{R}$ *and* $x_i \in V$, *then*

$$\frac{\forall s.\ \mathsf{Flow}_G\, \varphi_s\, x\, f\, G\, s \qquad \forall s \forall t \geq 0.\ P\, s \wedge G\, (\varphi_s\, t) \to Q\, (\varphi_s\, t)}{\lceil P\rceil \subseteq |x' = f \,\&\, G]\lceil Q\rceil}\ \mathit{solve}_G$$

Finally, we can derive another $\mathsf{d}\mathcal{L}$-style inference rule for guards.

Lemma 6.3. *The* differential weakening *rule is derivable for* $P, G, Q : \mathbb{R}^V \to \mathbb{B}$, $f_i : \mathbb{R}^V \to \mathbb{R}$ *and* $x_i \in V$.

$$\frac{\lceil G\rceil \subseteq \lceil Q\rceil}{\lceil P\rceil \subseteq |x' = f \,\&\, G]\lceil Q\rceil}\ dW$$

Soundness of (dW) is rather trivial: $|x' = f \,\&\, G]\lceil G\rceil = \mathit{Id}$ by evaluating the wlp. Thus $\lceil P\rceil \subseteq \mathit{Id} = |x' = f \,\&\, G]\lceil G\rceil \subseteq |x' = f \,\&\, G]\lceil Q\rceil$ by isotonicity of boxes in MKA and the assumption $\lceil G\rceil \subseteq \lceil Q\rceil$.

An example verification of a guarded evolution is presented in Section 9.

7 Differential Invariants

In the theory of dynamical systems, an *invariant set* for a flow φ is a set $X \subseteq \mathbb{R}^n$ such that $\gamma^\varphi\, x \subseteq X$ for all $x \in X$ [30]. Alternatively, such sets can be described as *differential invariants* of the action φ of the Lie group $\mathbb{R}$ on $\mathbb{R}^n$ [26], which is beyond the scope of this article. Intuitively, if an invariant X holds at the initial point x_0 of φ_{x_0}, then it holds at all other points of the orbit through x_0. In particular, all orbits or unions of orbits are invariants. Invariants can sometimes be used instead of flows to solve differential equations. It is straightforward to transfer this to hybrid program verification. A *differential invariant* [27, 28] for $x' = f \,\&\, G$ is a predicate I that satisfies the partial correctness specification

$$\lceil I \rceil \subseteq |x'{=}f \,\&\, G] \lceil I \rceil.$$

If I holds before the evolution, then it remains true during and after it. Due to the intentional limitations of solving differential equations with $\mathsf{d}\mathcal{L}$, differential invariants form an integral part of this logic. The general approach is mathematically involved. We merely illustrate it by a classical example (cf. [27]).

Example 7.1 (Circular movement of particle). The ODEs

$$x' = y, \qquad y' = -x$$

can be solved by linear combinations of trigonometric functions. Alternatively, all orbits are "governed" by the separable differential equation

$$\frac{dy}{dx} = \frac{y'}{x'} = -\frac{x}{y},$$

obtained by parametric derivation. Rewriting it as $ydy{+}xdx = 0$ and integrating both sides yields $I = x^2 + y^2 = r^2$ for some constant $r > 0$. This invariant describes the circular orbits—the *phase portrait*—of the ODEs. Checking that I is indeed a differential invariant in the sense of $\mathsf{d}\mathcal{L}$ requires reverting this calculation in the specification statement $\lceil I \rceil \subseteq |x' = y, y' = -x] \lceil I \rceil$. It can be rewritten as $\lceil I \rceil \subseteq |x' = y, y' = -x] \lceil 2xx' + 2yy' = 0 \rceil$ by using the well known fact that two continuous differentiable functions differ by a constant if and only if they have the same derivatives, thus putting the above differential equation in place of the invariant. Substituting $(2xx' + 2yy')[y/x', -x/y'] = 2xy - 2yx$ in the postcondition then yields $\lceil I \rceil \subseteq |x' = y, y' = -x] \lceil \lambda s.\ \mathit{True} \rceil$, which holds irrespective of $|x' = y, y' = -x]$. Evaluating this wlp is therefore unnecessary for proving the correctness statement. □

One of the most intriguing features of $\mathsf{d}\mathcal{L}$ is its supports for such calculations in a proof-theoretic setting. This is achieved through four domain-specific inference rules for differential invariants, which may circumvent solving ODEs: a differential invariant, a differential cut, a differential ghost and sometines a differential effect rule. We derive the first two in our setting (we currently do not see any need for the other ones in our setting).

We extend our language to support substitutions of primed variables, as in Example 7.1. We use the language of *differential rings*: rings R equipped with an additive group morphism $(-)' : R \to R$ satisfying the Leibniz rule $(\theta \cdot \eta)' = \theta' \cdot \eta + \eta \cdot \theta'$. We use constant symbols and primed variables, thus require $c' = 0$. We also extend $\mathbb{R}^V$ to $\mathbb{R}^{V \cup V'}$, such that $V \cap V' = \emptyset$ and $x \in V \Leftrightarrow x' \in V'$.

Within this approach, we can easily extend flows to $\mathbb{R}^{V \cup V'}$ and the flow predicate to $\mathsf{Flow}_{GI}\, \varphi\, x\, f\, G\, s \Leftrightarrow \mathsf{Flow}_G\, \varphi\, x\, f\, G\, s \wedge \mathsf{Prime}_1\, \varphi\, s \wedge \mathsf{Prime}_2\, \varphi\, s$. Here,

$\mathsf{Prime}_1\, \varphi\, s \Leftrightarrow \forall t \geq 0, 1 \leq i \leq n.\ \varphi\, t\, s\, x_i' = f_i\, (\varphi\, t\, s)$ and furthermore $\mathsf{Prime}_2\, \varphi\, s \Leftrightarrow \forall t \geq 0, z \in V \setminus X.\ \varphi\, t\, s\, z' = 0$ impose "solution conditions" for primed variables on φ. The relational flow semantics of evolution statements in (2) can then be extended to $\mathbb{R}^{V \cup V'}$ and precondition Flow_{GI}; wlps for these statements can be calculated in a straightforward extension of Proposition 6.1.

Proposition 7.2. *Let* $\varphi : \mathbb{R} \to \mathbb{R}^{V \cup V'} \to \mathbb{R}^{V \cup V'}$*,* $G, Q : \mathbb{R}^{V \cup V'} \to \mathbb{B}$*,* $f_i : \mathbb{R}^{V \cup V'} \to \mathbb{R}$*,* $x_i \in V$ *and* $z_i \in V \setminus X$*. Then* $\forall s \in \mathbb{R}^V.\ \mathsf{Flow}_G\, \varphi\, x\, f\, G\, s$ *implies*

$$\begin{aligned}&|x' = f \,\&\, G]\lceil Q \rceil\\ &\quad = \lceil \lambda s.\ \forall t \geq 0.\ G\,(\varphi_s\, t) \to Q\,(s[x \mapsto \varphi_s\, t\, x, x' \mapsto f\,(\varphi_s\, t), z' \mapsto 0])\rceil.\end{aligned}$$

Once again, we suppress indices to simplify notation. Rules for solving guarded evolution statements and the differential weakening rule (dW) can readily be extended to $\mathbb{R}^{V \cup V'}$, too. Yet $(solve_G)$ is not particularly interesting in this context: the whole purpose of invariants is to avoid this rule!

To derive the specific invariant rules of $\mathsf{d}\mathcal{L}$ we follow $\mathsf{d}\mathcal{L}$ in extending $(-)'$ from terms to atomic predicates $=$ and $\leq$ and to positive formulas over the language of differential rings:

$$(\theta = \eta)' = (\theta' = \eta'), \qquad (\theta < \eta)' = (\theta \leq \eta)' = (\theta' \leq \eta'),$$
$$(\phi \wedge \psi)' = (\phi \vee \psi)' = \phi' \wedge \psi'.$$

These derivatives are symbolic; they apply to syntactic objects, which explains the change in notation. We have created Isabelle datatypes for this syntax and use an interpretation function $[\![-]\!]$ to link it with the functions and predicates of the relational semantics.

Lemma 7.3. *The* differential invariant *rule is derivable, for* $G, I : \mathbb{R}^{V \cup V'} \to \mathbb{B}$*,* $f_i : \mathbb{R}^{V \cup V'} \to \mathbb{R}$*,* $x_i \in V$*,* $[\![\phi]\!] = I$*,* $[\![\theta]\!] = f$ *and where no variable in* ϕ *is primed.*

$$\frac{\forall s.\, G\, s \wedge (\forall z \neq x_i.\, s\, z' = 0) \to [\![\phi'[\theta/x']]\!]\, s}{\lceil I \rceil \subseteq |x'{=}f \,\&\, G]\lceil I \rceil}\ dI$$

Intuitively, (dI)—also called *differential induction* rule—embodies the substitutions from Example 7.1. The proof uses properties of substitutions and the mean value theorem.

Example 7.4 (Circular motion cont.). Consider again the ODEs $x' = y$ and $y' = -x$ from Example 7.1 and let $G = \lambda s.\ \mathit{True}$. Applying (dI) backwards to the specification statement in Example 7.1 yields the hypothesis

$$\forall s.\, \forall z \notin \{x, y\}.\ s\, z' = 0 \to [\![(x^2 + y^2 = r^2)'[y/x', -x/y']]\!]\, s.$$

Symbolic differentiation and substitution application confirm its validity. □

The rule (dI) can of course handle inequalities, conjunctions, disjunctions and quantified formulas as well. Its conditions on z and z', however, are not needed in its $\mathsf{d}\mathcal{L}$ counterpart. In addition, we can derive the following variant.

Corollary 7.5. *For* $P, G, I, Q : \mathbb{R}^{V \cup V'} \to \mathbb{B}$*,* $f_i : \mathbb{R}^{V \cup V'} \to \mathbb{R}$ *and* $x_i \in V$*,*

$$\frac{\lceil P \rceil \subseteq \lceil I \rceil \qquad \forall s.\, G\, s \wedge (\forall z \neq x_i.\, s\, z' = 0) \to [\![\phi'[\theta/x']]\!]\, s \qquad \lceil I \rceil \subseteq \lceil Q \rceil}{\lceil P \rceil \subseteq |x'{=}f \,\&\, G]\lceil Q \rceil}\ dI_2$$

where again $[\![\phi]\!] = I, [\![\theta]\!] = f$ *and no variable in* ϕ *is primed.*

This rule is designed to be combined with (dW) and the final $\mathsf{d}\mathcal{L}$ rule we derive.

Lemma 7.6. *The* differential cut *rule is derivable for* $P, G, I, Q : \mathbb{R}^{V \cup V'} \to \mathbb{B}$*,* $f_i : \mathbb{R}^{V \cup V'} \to \mathbb{R}$ *and* $x_i \in V$*:*

$$\frac{\lceil P \rceil \subseteq |x'{=}f \,\&\, G]\lceil I \rceil \qquad \lceil P \rceil \subseteq |x'{=}f \,\&\, (\lambda\, s. G\, s \wedge I\, s)]\lceil Q \rceil}{\lceil P \rceil \subseteq |x'{=}f \,\&\, G]\lceil Q \rceil}\ dC$$

This rule is essential for introducing differential invariants into proofs. An example for its use in combination with (dI_2) and (dW) can be found in Section 9.

Soundness of (dC) can be explained as follows: By the right premise, G, I and Q hold along the evolution $x' = f$ on part of the phase space. By the left premise, G and I holds along $x' = f$ anyway. Hence G and Q hold along $x' = f$.

This completes the derivation of the inference rules $\mathsf{d}\mathcal{L}$ in Isabelle/HOL—except for the differential effect and ghost rule. They allow the verification of hybrid programs with a general purpose proof assistant in the style of $\mathsf{d}\mathcal{L}$. A recent soundness proof of a uniform substitution calculus for $\mathsf{d}\mathcal{L}$ with Isabelle [5] includes a proof term checker for this specific calculus and hence KeYmaera X, but not a prima facie verification component.

8 Isabelle Components

Our verification components for hybrid programs in Isabelle are based on the integration of two theory stacks from the *Archive of Formal Proofs*: those for Kleene algebras, MKA and the corresponding verification components [2, 11, 13], and those related to analysis and ODEs [19]. The second stack, in particular, is very deep. It contains mathematical components for topology, measure theory, analysis and differential equations. Theory engineering and theorem proving at this theory depth can certainly be challenging, but our overall experience was very positive. Our new verification components on top of these stacks with >120 mathematical components currently consist of >1500 lines of Isabelle code with >100 lemmas. The online Isabelle proof document consists of 36 pages in PDF format. Proofs are usually highly interactive; they relate mainly to set-theoretic and higher-order properties of the relational flow model. At the moment, many of them are rather long and complex. Scope for simplifying models and proofs remains and our components should be considered as prototypes.

Our previous MKA verification components provide syntactic illusions for program specifications in the relational program store model. Specification statements can be written as *PRE* ⟨*precondition*⟩ ⟨*program*⟩ *POST* ⟨*postcondition*⟩.

The ODE components contain useful functions and definitions for vector fields and flows, for instance $((\textit{flow solves-ode vector-field})\ \textit{interval ran})$. This statement expresses that a given flow is a solution of a vector field where, in our setting, *interval* is $\mathbb{R}$ and *ran* an open subset of $\mathbb{R}^n$.

We encode program variables as strings. Hence phase spaces $\mathbb{R}^V$ have type *real store*, which is a synonym of *string* $\Rightarrow$ *real.* The relational flow semantics of evolution statements (2) is formalised as

$$\textit{ODEsystem xfList with } G = \{(s,\ F\ t) \mid s\ t\ F.\ 0 \leq t \wedge \textit{solvesStoreIVP F xfList s G}\},$$

where *solvesStoreIVP F xfList s G* is the Isabelle-analogue of $\mathsf{Flow}_{GI}\, \varphi\, x\, f\, s$ and the term *xfList* :: (*string* $\times$ ((*string* $\Rightarrow$ *real*) $\Rightarrow$ *real*)) *list* that of the system represented by x and f. This deviates from our mathematical presentation. The $\mathsf{Flow}_\exists$ constraint is included into the definiens, instead of acting as a hypothesis. In addition, uniqueness checks are currently postponed to inference rules. This is suitable for Isabelle as the resulting definition is total. The semantics evaluates to $\emptyset$ if the flow is not a solution to the corresponding evolution statement.

Our formalisation includes guards as well as primed variables, that is, the semantics and $(\textit{solves}_{GI})$ rule uses the predicate Flow_{GI} defined in Section 7. This rule requires that users supply a flow. We require that its input has type (*real* $\Rightarrow$ *real store* $\Rightarrow$ *real*) *list* to make it easier for users to supply a specification with λ-abstractions (see Section 9). It is then transformed into a semantics with update function *(s,t,xfList,uInput)* $\mapsto$ *sol s*[*xfList*$\leftarrow$*uInput*] *t*, where

$$\textit{sol } s[\textit{xfList}\leftarrow\textit{uInput}]\ t = \begin{cases} \upsilon_i\, t\, s, & \text{if } w = x_i, \\ (\lambda\, \tau.\, \upsilon_i\, \tau\, s)'\, t, & \text{if } w = x_i', \\ 0, & \text{if } w \in V' \setminus X', \\ s\, w, & \text{if } w \in V \setminus X, \end{cases}$$

and *uInput* is the list of user-inputs $\upsilon_1, \ldots, \upsilon_n$. Proof obligations for

$$\mathsf{Flow}_{GI}\, (\lambda t\, s.\textit{sol } s[\textit{xfList}\leftarrow\textit{uInput}]\ t)\, x\, f\, s,$$

where we use mixed notation, are then generated automatically:

1. $G\, (\textit{sol } s[\textit{xfList}\leftarrow\textit{uInput}]\ t)$,
2. $P\, s \rightarrow Q\, (\textit{sol } s[\textit{xfList}\leftarrow\textit{uInput}]\ t)$,
3. $(\lambda\, \tau.\, (\textit{sol } s[\textit{xfList}\leftarrow\textit{uInput}]\ \tau)\ x_i)' = (\lambda\, \tau.\, f_i\, (\textit{sol } s[\textit{xfList}\leftarrow\textit{uInput}]\ \tau))$ in the interval $(0, t)$,
4. $(\lambda\, \tau.\, f\, (\textit{sol } s[\textit{xfList}\leftarrow\textit{uInput}]\ \tau))$ is continuous in $[0, t]$,
5. $\forall \varphi.\, \mathsf{Solves}_{GI}\, \varphi\, x\, f\, s \rightarrow \forall \tau \in [0, t].\, f_i\, (\varphi\, \tau) = f_i\, (\textit{sol } s[\textit{xfList}\leftarrow\textit{uInput}]\ \tau)$.

Condition (1) requires that the guard holds along the flow; (2) that the postcondition holds along the flow whenever the initial state satisfies the precondition; (3) that the derivative of the flow is equal to the vector field; (4) that the flow is continuous; and (5)that flows are uniquely defined. There are some additional simple conditions that specify well formedness of the verification statement, that the f_i do not include primed variables or that the lists *xfList* and *uInput* have the same length.

9 Verification Examples

The four simple examples in this section are adapted from the literature on KeYmaera X [29, 28]. Their main purpose is to illustrate the main rules of $\mathsf{d}\mathcal{L}$; thus demonstrating the feasibility of the approach. MKA rules for the control flow are already well tested [12].

Motion with constant velocity. First we consider the evolution of an object x that moves with constant positive velocity v away from another point y.

```
lemma PRE (λ s. s ''y'' < s ''x''  ∧ s ''v'' > 0)
      (ODEsystem [(''x'',(λ s. s ''v''))] with (λ s. True))
      POST (λ s. (s ''y'' < s ''x''))
  apply(rule-tac uInput=[λ t s. s ''v'' · t + s ''x''] in dSolve-toSolveUBC)
  prefer 11 subgoal by(simp add: wp-trafo vdiff-def add-strict-increasing2)
  apply(simp-all add: vdiff-def varDiffs-def, clarify)
  apply(rule-tac f'1=λt. st ''v'' and g'1=λt. 0 in derivative-intros(173))
  apply(rule-tac f'1=λt.0 and g'1=λt.1 in derivative-intros(176))
    subgoal by (auto intro: derivative-intros)
    subgoal by(clarify, rule continuous-intros)
  by(simp add: solvesStoreIVP-def vdiff-def varDiffs-def)
```

The double quotes indicate Isabelle strings; they represent program variables. The postcondition describes that the object x is always above y. The first line of the proof supplies the solution $\lambda t\ s.\ s\ ''v'' \cdot t + s\ ''x''$ to the evolution statement. It could have been computed by a computer algebra system or a similar solver. The second line solves the wlp by invoking Isabelle's simplifier. The remaining lines prove that Flow holds, that is, the conditions (1)-(5) in the previous section.

System where the guard implies the postcondition. Next we consider a system that is initially at position $x = 0$ with velocity $x' = x + 1$ and satisfies postcondition $x \geq 0$. It can be verified without providing a flow or using a differential invariant, because the guard and the postcondition are equal. Our first proof works directly with the wlp for guarded evolutions in Proposition 6.1.

```
lemma PRE (λ s. s ''x'' = 0)
      (ODEsystem [(''x'',(λ s. s ''x'' + 1))] with (λ s. s ''x'' ≥ 0))
      POST (λ s. s ''x'' ≥ 0)
  apply(clarify, simp add: p2r-def)
  apply(simp add: rel-ad-def rel-antidomain-kleene- algebra.addual.ars-r- def)
  apply(simp add: rel-antidomain-kleene- algebra.fbox-def)
  apply(simp add: relcomp-def rel-ad-def guarDiffEqtn-def)
  by(simp add: solvesStoreIVP-def, auto)
```

The lemmas mentioned in the apply statements show that Isabelle pulls together facts about MKAs, ODEs and the relational flow semantics. A fully automated proof can, however, be obtained simply by applying (dW) and then simplifying: **using** *dWeakening* **by** *simp*.

Circular motion. Next we formalise the manual proof from Example 7.1 to illustrate the use of differential invariants in Section 7.

lemma *PRE* (λs. (s ''x'') · (s ''x'') + (s ''y'') · (s ''y'') − (s ''r'') · (s ''r'') = 0)
  (*ODEsystem* [(''x'',(λs. s ''y'')),(''y'',(λs. − s ''x''))] *with G*)
  POST (λs. (s ''x'') · (s ''x'') + (s ''y'') · (s ''y'') − (s ''r'') · (s ''r'') = 0)
 apply(*rule-tac* η= ''x'' ⊗ ''x'' ⊕ ''y'' ⊗ ''y'' ⊖ ''r'' ⊗ ''r''
  and *uInput*=[''y'', - ''x''] **in** *dInvForTrms*)
 apply(*simp-all add*: *vdiff-def varDiffs-def*)
 by(*clarsimp*, *erule-tac x*=''r'' **in** *allE, simp*)

In the first line of the proof, we supply the invariant η in the differential ring syntax developed, as explained in Section 8. We need to supply invariants as terms, in this example as $x^2+y^2-r^2$. It is translated into $x^2+y^2=r^2$ internally. Moreover, a term for the ODE system has to be given. This, however, should be automated, as this information has already been provided in another syntax in the *ODEsystem* declaration. The syntax with $\oplus$, $\ominus$ and $\odot$ simplifies the one used in our Isabelle theories. Isabelle's simplifiers can then deal with the remaining proof obligations, using some domain-specific definitions for differential rings. The guard G in the program is generic; it plays no role in the proof.

Single-hop ball. Finally, we use invariants in a two-modes example of a particle falling from an initial height H to the ground, and then changing its velocity upwardly by a factor of $0 \leq c \leq 1$ by using an assignment statement.

lemma *single-hop-ball*:
PRE (λ s. 0 ≤ s ''x'' ∧ s ''x'' = H ∧ s ''v'' = 0 ∧ s ''g'' > 0 ∧ 1 ≥ c ∧ c ≥ 0)
(((*ODEsystem* [(''x'', λ s. s ''v''),(''v'',λ s. − s ''g'')] *with* (λ s. 0 ≤ s ''x'')));
(*IF* (λ s. s ''x'' = 0) *THEN* (''v'' ::= (λ s. − c · s ''v'')) *ELSE* (''v'' ::= (λ s. s ''v'')) *FI*)) *POST* (λ s. 0 ≤ s ''x'' ∧ s ''x'' ≤ H)
apply(*simp add*: *d-p2r*, *subgoal-tac rdom* ⌈λs. 0 ≤ s ''x'' ∧ s ''x'' = H ∧ s ''v'' = 0 ∧ 0 < s ''g'' ∧ c ≤ 1 ∧ 0 ≤ c⌉ ⊆ *wp* (*ODEsystem* [(''x'', λs. s ''v''), (''v'', λs. − s ''g'')] *with* (λs. 0 ≤ s ''x'')) ⌈*inf* (*sup* (− (λs. s ''x'' = 0)) (λs. 0 ≤ s ''x'' ∧ s ''x'' ≤ H)) (*sup* (λs. s ''x'' = 0) (λs. 0 ≤ s ''x'' ∧ s ''x'' ≤ H))⌉)
apply(*simp add*: *d-p2r*, *rule-tac C* = λ s. s ''g'' > 0 **in** *dCut*)
apply(*rule-tac* φ = (t_C 0) ≺ (t_V ''g'') **and** *uInput*=[t_V ''v'', ⊖ t_V ''g'']**in** *dInvFinal*)
apply(*simp-all add*: *vdiff-def varDiffs-def*, *clarify*, *erule-tac x*=''g'' **in** *allE*, *simp*)
apply(*rule-tac C* =λ s. s ''v'' ≤ 0 **in** *dCut*)
apply(*rule-tac* φ = (t_V ''v'') ⪯ (t_C 0) **and** *uInput*=[t_V ''v'', ⊖ t_V ''g''] **in** *dInvFinal*)
apply(*simp-all add*: *vdiff-def varDiffs-def*)
apply(*rule-tac C* = λ s. s ''x'' ≤ H **in** *dCut*)
apply(*rule-tac* φ = (t_V ''x'') ⪯ (t_C H) **and** *uInput*=[t_V ''v'', ⊖ t_V ''g'']**in** *dInvFinal*)
apply(*simp-all add*: *varDiffs-def vdiff-def*) **using** *dWeakening* **by** *simp*

The Isabelle syntax in our theory files is simplified by using $\preceq$. The proof consists of two blocks, in which differential invariants are provided through (dC) and then discharged with (dI_2). Using semantic notation, the invariants are $g > 0$, $v \leq 0$, and $x \leq H$. The proof is then finished by (dW).

10 Conclusion

We have constructed generic modular verification components à la $\mathsf{d}\mathcal{L}$ for hybrid programs, based on MKA, and formalised with Isabelle. These include a relational flow semantics for hybrid programs for MKA, generic rules for solving and reasoning about ODEs in predicate transformer semantics, and soundness proofs for the most relevant inference rules of $\mathsf{d}\mathcal{L}$ relative to this semantics.

Our soundness proofs complement those in [5], which focus on a uniform substitution calculus within a proof theory for $\mathsf{d}\mathcal{L}$. We work entirely within the semantics of MKA and, by and large, use store updates in place of substitutions.

Our general goals and motivations differ from those of $\mathsf{d}\mathcal{L}$ and KeYmaera X as well. There, a main concern is scalable hybrid program verification in practice. This is achieved via significant restrictions on the ODEs admitted and the assertion language used. We aim at an open experimental platform for mathematically simple, light-weight and effective integrations of continuous dynamics and discrete control into proof assistants, beyond $\mathsf{d}\mathcal{L}$. MKA could, for instance, be reduced to Kleene algebra with tests over a relational flow semantics—and dynamic logic thus be reduced to Hoare logic. This semantics could also be combined with a duration calculus, for which we already have Isabelle support [8] or with other algebraic semantics [16]. Another upshot of MKA is its close relationship to refinement calculi based on predicate transformers [4]. Domain-specific refinement rules for hybrid systems seem another avenue worth exploring.

Our verification components are still prototypes, and there is much scope for making the development both more abstract and more efficient. In particular, more refined domains and intervals of existence for flows are needed. Even the simple examples in Section 9 currently require a significant amount of user interaction. Better simplification mechanisms and Isabelle tactics should be developed before more large scale case studies are undertaken.

Acknowledgements. This work was funded by a CONACYT scholarship. We are grateful to André Platzer for sharing his wisdom on $\mathsf{d}\mathcal{L}$, and to Achim Brucker, Michael Herzberg, Andreas Lochbihler and Makarius Wenzel for Isabelle advice.

References

[1] R. Alur. Formal verification of hybrid systems. In *EMSOFT 2011*, pages 273–278. ACM, 2011.
[2] A. Armstrong, G. Struth, and T. Weber. Kleene algebra. *Archive of Formal Proofs*, 2013.
[3] V. I. Arnol'd. *Ordinary Differential Equations.* Springer, 1992.
[4] R. Back and J. von Wright. *Refinement Calculus—A Systematic Introduction.* Springer, 1998.
[5] B. Bohrer, V. Rahli, I. Vukotic, M. Völp, and A. Platzer. Formally verified differential dynamic logic. In *CPP 2017*, pages 208–221. ACM, 2017.
[6] A. Chlipala. *Certified Programming with Dependent Types—A Pragmatic Introduction to the Coq Proof Assistant.* MIT Press, 2013.

[7] J. Desharnais and G. Struth. Internal axioms for domain semirings. *Science of Computer Programming*, 76(3):181–203, 2011.

[8] B. Dongol, I. J. Hayes, and G. Struth. Relational convolution, generalised modalities and incidence algebras. *CoRR*, abs/1702.04603, 2017.

[9] G. E. Fainekos, H. Kress-Gazit, and G. J. Pappas. Hybrid controllers for path planning: A temporal logic approach. In *IEEE Conference on Decision and Control*, pages 4885–4890, 2005.

[10] N. Fulton, S. Mitsch, J. Quesel, M. Völp, and A. Platzer. KeYmaera X: an axiomatic tactical theorem prover for hybrid systems. In *CADE-25*, volume 9195 of *LNCS*, pages 527–538. Springer, 2015.

[11] V. B. F. Gomes, W. Guttman, P. Höfner, G. Struth, and T. Weber. Kleene algebra with domain. *Archive of Formal Proofs*, 2016.

[12] V. B. F. Gomes and G. Struth. Modal Kleene algebra applied to program correctness. In *FM 2016*, volume 9995 of *LNCS*, pages 310–325, 2016.

[13] V. B. F. Gomes and G. Struth. Program construction and verification components based on Kleene algebra. *Archive of Formal Proofs*, 2016.

[14] D. Harel, D. Kozen, and J. Tiuryn. *Dynamic Logic.* MIT Press, 2000.

[15] T. A. Henzinger. The theory of hybrid automata. In *LICS 1996*, pages 278–292. IEEE Computer Society, 1996.

[16] P. Höfner and B. Möller. An algebra of hybrid systems. *J. Logic and Algebraic Programming*, 78(2):74–97, 2009.

[17] J. Hölzl, F. Immler, and B. Huffman. Type classes and filters for mathematical analysis in Isabelle/HOL. In *ITP 2013*, volume 7998 of *LNCS*, pages 279–294. Springer, 2013.

[18] F. Immler and J. Hölzl. Numerical analysis of ordinary differential equations in Isabelle/HOL. In *ITP 2012*, volume 7406 of *LNCS*, pages 377–392. Springer, 2012.

[19] F. Immler and J. Hölzl. Ordinary differential equations. *Archive of Formal Proofs*, 2012.

[20] F. Immler and C. Traut. The flow of ODEs. In *ITP 2016*, volume 9807 of *LNCS*, pages 184–199. Springer, 2016.

[21] J. Jeannin, K. Ghorbal, Y. Kouskoulas, A. Schmidt, R. Gardner, S. Mitsch, and A. Platzer. A formally verified hybrid system for safe advisories in the next-generation airborne collision avoidance system. *STTT*, 19(6):717–741, 2017.

[22] G. Klein, J. Andronick, K. Elphinstone, T. C. Murray, T. Sewell, R. Kolanski, and G. Heiser. Comprehensive formal verification of an OS microkernel. *ACM Trans. Comput. Syst.*, 32(1):2:1–2:70, 2014.

[23] X. Leroy. Formal verification of a realistic compiler. *CACM*, 52(7):107–115, 2009.

[24] S. M. Loos, A. Platzer, and L. Nistor. Adaptive cruise control: Hybrid, distributed, and now formally verified. In *FM 2011*, volume 6664 of *LNCS*, pages 42–56. Springer, 2011.

[25] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL—A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[26] P. J. Olver. *Applications of Lie Groups to Differential Equations.* Springer, 1986.

[27] A. Platzer. The structure of differential invariants and differential cut elimination. *LMCS*, 8(4), 2008.

[28] A. Platzer. *Logical Analysis of Hybrid Systems.* Springer, 2010.

[29] J. Quesel, S. Mitsch, S. M. Loos, N. Arechiga, and A. Platzer. How to model and prove hybrid systems with KeYmaera: a tutorial on safety. *STTT*, 18(1):67–91, 2016.

[30] G. Teschl. *Ordinary Differential Equations and Dynamical Systems.* AMS, 2012.

A Axioms and Definitions

Semiring A *semiring* is a structure $(S, +, \cdot, 0, 1)$ that, for all $\alpha, \beta, \gamma \in S$ satisfies

$$\alpha \cdot (\beta \cdot \gamma) = (\alpha \cdot \beta) \cdot \gamma, \qquad 1 \cdot \alpha = \alpha, \qquad \alpha \cdot 1 = \alpha,$$
$$\alpha + (\beta + \gamma) = (\alpha + \beta) + \gamma, \qquad \alpha + \beta = \beta + \alpha, \qquad \alpha + 0 = \alpha, \qquad 0 + \alpha = \alpha,$$
$$\alpha \cdot (\beta + \gamma) = \alpha \cdot \beta + \alpha \cdot \gamma, \qquad (\alpha + \beta) \cdot \gamma = \alpha \cdot \gamma + \beta \cdot \gamma,$$
$$0 \cdot \alpha = 0, \qquad \alpha \cdot 0 = 0.$$

Dioid A *dioid* is a semiring S that satisfies $\alpha + \alpha = \alpha$ for all $x \in S$.

Kleene algebra A *Kleene algebra* is a structure $(K, +, \cdot, 0, 1, ^*)$ such that $(K, +, \cdot, 0, 1)$ is a dioid and, for all $x, y, z \in K$,

$$1 + \alpha \cdot \alpha^* \leq \alpha^*, \qquad \gamma + \alpha \cdot \beta \leq \beta \to \alpha^* \cdot \gamma \leq \beta,$$
$$1 + \alpha^* \cdot \alpha \leq \alpha^*, \qquad \gamma + \beta \cdot \alpha \leq \beta \to \gamma \cdot \alpha^* \leq \beta.$$

Modal Kleene Algebra A *modal Kleene algebra* is a tuple $(K, +, \cdot, 0, 1, ^*, ad, ar)$ such that $(K, +, \cdot, 0, 1, ^*)$ is a Kleene algebra and, for all $x, y \in K$,

$$ad\, \alpha \cdot \alpha = 0, \qquad ad\, \alpha + ad^2\, \alpha = 1, \qquad ad\,(\alpha \cdot \beta) \leq ad\,(\alpha \cdot ad^2\, \beta),$$
$$\alpha \cdot (ar\, \alpha) = 0, \qquad ar\, \alpha + ar^2\, \alpha = 1, \qquad ar\,(\alpha \cdot \beta) \leq ar\,(ar^2\, \alpha \cdot \beta),$$
$$ad^2\,(ar^2\, \alpha) = ar^2\, \alpha, \qquad ar^2\,(ad^2\, \alpha) = ad^2\, \alpha.$$

Syntax of Differential Rings $\mathcal{R} ::= 0 \mid 1 \mid x \mid c \mid \mathcal{R} + \mathcal{R} \mid \mathcal{R} - \mathcal{R} \mid \mathcal{R} \cdot \mathcal{R} \mid \mathcal{R}'$, where x is drawn from a set of variables and c from a set of constants.

d$\mathcal{L}$-Style Syntax of Hybrid Programs

$$\mathcal{C} ::= x := e \mid x' = f \;\&\; G \mid \mathcal{C};\mathcal{C} \mid \textbf{if } P \textbf{ then } \mathcal{C} \textbf{ else } \mathcal{C} \mid \textbf{while } P \textbf{ inv } I \textbf{ do } \mathcal{C},$$

where x is a variable, e an expression, f a vector field expression, G a guard and P a test. However we do not work with an explicit syntax for hybrid programs in Isabelle. The entire development is within the relational flow semantics. We are only creating syntactic illusions within the semantics.

B Background on Differential Equations

Ordinary Differential Equations A system of n ordinary differential equations (ODEs) can be described by a vector field $f \in C(X, \mathbb{R}^n)$, where X is an open subset of $\mathbb{R}^{n+1}$. Its solutions are integral curves $x : I \to \mathbb{R}^n$ that is, differentiable functions that satisfy $x'\, t = f\,(t, x\, t)$, where $t \in I \subseteq \pi_1[X]$ (and π_1 is the standard first projection map). An initial value problem for this system of ODEs is specified with an initial condition $x\ t_0 = x_0$ where $x_0 \in X$ and $t_0 \in I$.

Autonomous Systems of ODEs The vector fields described above are time dependent. Time independent vector fields satisfy $f \in C(X, \mathbb{R}^n)$ with $X \subseteq R^n$. The system of ODEs is then called *autonomous* [30] and its solutions satisfy the equation $x'\ t = f\ (x\ t)$ for $t \in I \subseteq \mathbb{R}$.

Lipschitz Continuity A function $f : V \to W$ between normed vector spaces V and W is Lipschitz continuous on $X \subseteq V$ if there is a constant $k \geq 0$ such that $||f\,x - f\,y|| \leq k||x - y||$ holds for all $x, y \in X$. The function f is a contraction if it is Lipschitz continuous with $k < 1$.

Solutions to ODEs: Existence and Uniqueness A Banach space is a complete normed vector space, that is, all Cauchy sequences converge. Let $B \neq \emptyset$ be a closed subset of a Banach space. Then, by Banach's fixpoint theorem, every contraction $f : B \to B$ has a unique fixpoint. This theorem is essential for proving the Picard-Lindelöf Theorem, which guarantees unique solutions for initial value problems. We state a special case for autonomous systems of ODEs.

Theorem B.1 (Picard-Lindelöf Theorem). *For every Lipschitz continuous vector field $f : X \subseteq \mathbb{R}^n \to \mathbb{R}^n$ and $x_0 \in X$ there exists a unique integral curve $x \in C^1(I(x_0))$ that satisfies the initial value problem for the autonomous system $x'\ t = f\,(x\,t)$ and $x\,t_0 = x_0$. $I(x_0)$ is the maximal interval of existence for x around t_0.*

Flow of an autonomous system of ODEs Let $f : X \subseteq \mathbb{R}^n \to X$ be a Lipschitz continuous vector field that admits, for each $x \in X$, a unique integral curve $\phi_x \in C^1(I(x))$ such that $\phi_x\ 0 = x$ by Picard-Lindelöf's theorem. The local flow $\varphi : T \to X \to X$ for f is defined by $\varphi\ t\ x = \phi_x\ t$, where $T = \bigcap_{x \in X} I(x)$.

For $\mathbb{R} = T$, the group action equations $\varphi\ 0 = id$ and $\varphi\ (s+t) = \varphi\ s \circ \varphi\ t$ are immediately derivable from this definition. The flow is thus indeed an action of the additive group $\mathbb{R}$ on X, and hence a dynamical system.

C Cross-References to Isabelle Proofs

The following table links the results in this article with facts from the Isabelle repository. The repository contains a readme file with further instructions.

Result in article	Result in Isabelle theories
Definition of evolution statements	subsumed by *guarDiffEqtn*
Definition of Flow	*solvesStoreIVP* (no uniqueness)
Proposition 5.2	subsumed by *dS*
Lemma 5.3	subsumed by *dSolve*
Definition of guarded evolution statement	*guarDiffEqtn*
Proposition 6.1	*dS*
Lemma 6.2	subsumed by *dSolve*, (Flow_{GI} in Isabelle syntax)
Lemma 6.3	subsumed by *dWeakening*
Definition of differential ring language	**datatype** *trms*, **primrec** *rdiff*
Lemma 7.2	implied by *prelim-dSolve*
Rules for solving ODEs in $\mathbb{R}^{V \cup V'}$	*dSolve*
Differential weakening rule for $\mathbb{R}^{V \cup V'}$	*dWeakening*
Lemma 7.3	*dInv*
Corollary 7.5	*dInvFinal*
Lemma 7.6	*dCut*