Uniform Substitution for Differential Refinement Logic*

Enguerrand Prebet and André Platzer✉

Karlsruhe Institute of Technology, Karlsruhe, Germany
{enguerrand.prebet,platzer}@kit.edu

Abstract. This paper introduces a uniform substitution calculus for differential refinement logic dRL. The logic dRL extends the differential dynamic logic dL such that one can simultaneously reason about properties of and relations between hybrid systems. Refinements are useful e.g. for simplifying proofs by relating a concrete hybrid system to an abstract one from which the property can be proved more easily. Uniform substitution is the key to parsimonious prover microkernels. It enables the verbatim use of single axiom formulas instead of axiom schemata with soundness-critical side conditions scattered across the proof calculus. The uniform substitution rule can then be used to instantiate all axioms soundly. Access to differential variables in dRL enables more control over the notion of refinement, which is shown to be decidable on a fragment of hybrid programs.

Keywords: Uniform substitution · Differential dynamic logic · Refinement · Hybrid systems

1 Introduction

Hybrid systems modeled by joint discrete dynamics and continuous dynamics are important and subtle systems in need of sound proofs [26] on account of their important applications [15,20,16,22,32]. Since such systems are important to get right, hybrid systems verification techniques themselves should be sound. Uniform substitution [24,25,27,28], originally phrased by Church for first-order logic [10, §35,40], has been identified as the key technique reducing the soundness-critical core to a prover microkernel and is behind the KeYmaera X prover [14].

This paper designs a corresponding uniform substitution proof calculus for differential refinement logic (dRL) [19]. The logic dRL is unique in its capabilities of proving simultaneous hybrid systems properties and hybrid systems refinement relations. This ability of dRL has been shown to be beneficial for establishing refinement relations of system implementations to verification abstractions and for relating time-triggered implementation models to event-triggered verification models [18]. The latter relation overcomes a stark divide in embedded

* Funding has been provided by an Alexander von Humboldt Professorship and the pilot program Core Informatics (KiKIT) of the Helmholtz Association (HGF).

C. Benzmüller et al. (Eds.): IJCAR 2024, LNAI 14740, pp. 196–215, 2024.
DOI: 10.1007/978-3-031-63501-4_11

system design principles while combining ease of verification with ease of implementation in ways that neither design paradigm alone supports. But such proving power only helps practical system verification if the theoretical proof calculi are implemented in a sound way and, in fact, dRL has not yet been implemented at all. Such an implementation is significantly simplified and significantly easier to get sound by identifying a uniform substitution calculus, which has no axiom schemata with their usual side conditions (and the algorithms implementing them) but merely a finite list of concrete dRL formulas as axioms. Reasoning directly with these concrete formulas also makes the proofs easier as the conditions are checked only when uniform substitution is used. This means that a direct consequence of the axioms could have more admissible substitution instances than the axioms themselves, whereas with schemata, the side conditions would pile up and not generalize as well. Other beneficial side effects include the fact that dRL now acquires a Hilbert-style proof calculus that is significantly more flexible and also more modular than dRL's previous sequent calculus.

Challenges include the fact that uniform substitution calculi for hybrid systems give a differential-form semantics to differentials and differential symbols [25], which is critical to obtain logic-based decision procedures for differential equation invariants [30], but also renders some sequent calculus proof rules of dRL unsound due to the resulting finer-grained view on differential equations. The flip side is that this finer view distinguishes widely different classes of differential equations better, thereby making it easier to tell apart different differential equations that merely coincide on the overall reachable set while having different temporal behavior. This difference is exploited here to obtain a decidability result for refinement for a fragment of hybrid systems. Other challenges to overcome are the unexpected definition of free variables of refinements, which are required for soundness. The core of the resulting calculus has been implemented in KeYmaera X[1], extending the prover microkernel in 4 hours of work with about 300 lines of code, mostly spent on writing down all the new axioms.

2 Related Work

Hybrid programs in dRL form a Kleene algebra with tests [17]. Program equivalence for Kleene algebra with tests is known to be decidable for abstract atomic programs. Refinement $\alpha \leq \beta$ can be recovered and defined as $\alpha \cup \beta = \beta$, but that duplicates reasoning about β. Certain classes of hypotheses can be added to the theory, e.g. Hoare-like triples $?p;\alpha;?\neg q = ?\mathit{false}$, without breaking the decidability [11]. This however does not extend when limited commutativity is allowed, which arises even in the discrete fragment: $(x := 2; y := 3) = (y := 3; x := 2)$ but $(x := 2; x := 3) \neq (x := 3; x := 2)$. KAT with only discrete assignments has been studied as Schematic KAT [4]. dRL can derive the axioms of Schematic KAT, but also allows reasoning with continuous dynamics and differential equations.

The Event-B method [1] is a formalism for reasoning about discrete models where the primary mechanism is refinement to check the conformance between

[1] https://github.com/LS-Lab/KeYmaeraX-release/tree/dRL

abstract models and more detailed ones. Multiple different formalisms have been proposed. Hybrid Event-B [2,6,5] is an extension with tool support [8] for hybrid systems with events corresponding to discrete and continuous evolutions. These continuous steps are however abstracted by the invariants they are assumed to satisfy. Event-B can also be extended with theories [9]. By adding some axioms about differential equations, it allows refinement reasoning with some continuous dynamics [12,3]. In contrast, dRL captures the continuous dynamics directly and proves the invariants as a consequence of the continuous dynamics.

Uniform substitution was proposed by Alonzo Church for first-order logic to capture axioms instead of axiom schemata [10, §35,40]. Modern uniform substitution originated for dL to support hybrid systems theorem proving in simple ways [25], extended to hybrid games in differential game logic dGL [27], and to communicating parallel programs dL_{CHP} [7]. This work is complementing the approach by adding refinement reasoning in a uniform substitution calculus for hybrid systems. Developing uniform substitution calculi are key to the design of small soundness-critical prover microkernels such as KeYmaera X [14].

3 Differential Refinement Logic dRL

Differential refinement logic dRL [19] extends the differential dynamic logic dL for hybrid systems [23] with a first-class refinement operator $\leq$ on hybrid systems. This section presents *differential-form* dRL, which prepares dRL for the features needed for dL's uniform substitution axiomatization, most notably the inclusion of differential terms alongside function symbols, predicate symbols, and program constant symbols, but also the requisite inclusion of differential variable symbols. Differential terms $(\theta)'$ are the fundamental logical device with which to enable sound [25] and complete [29,30] reasoning about differential equations.

3.1 Syntax

This section defines the syntax of the differential refinement logic dRL. The set of all variables is $\mathcal{V}$. To each variable $x \in \mathcal{V}$ is associated a *differential symbol* x' which is also in $\mathcal{V}$. Its purpose is to use x' to refer to the time-derivative of variable x during a differential equation, but also to cleverly relay that information to surrounding formulas in a sound way [25]. It is this (crucial) presence of differential symbols, that gives differential-form dRL a refined notion of refinement, especially of differential equations, compared to its sequent calculus predecessor [19].

Definition 1 (Terms). Terms *are defined by the grammar below where* $x \in \mathcal{V}$ *is a variable,* f *is a function symbol of arity* n *and* $\theta, \eta, \theta_1, \ldots, \theta_n$ *are terms:*

$$\theta, \eta ::= x \mid f(\theta_1, \ldots, \theta_n) \mid \theta + \eta \mid \theta \cdot \eta \mid (\theta)'$$

Terms have the usual arithmetic operations and function symbols. They also have differentials of terms $(\theta)'$ which describe how the value of θ changes locally

depending on the values of the differential symbols associated to the variables of θ.

Definition 2 (Formulas). Formulas *are defined by the grammar below where* $\theta, \eta, \theta_1, \ldots, \theta_n$ *are terms,* p *is a predicate symbol of arity* n, ϕ, ψ *are formulas and* α, β *are hybrid programs (Def. 3):*

$$\phi, \psi ::= \theta \leq \eta \mid p(\theta_1, \ldots, \theta_n) \mid \neg\phi \mid \phi \wedge \psi \mid \forall x\, \phi \mid [\alpha]\phi \mid \alpha \leq \beta$$

In addition to the operators of first-order logic of real arithmetic, formulas also contain the dL modality $[\alpha]\phi$ which expresses that the formula ϕ holds after all possible runs of the hybrid program α. dRL extends dL with the refinement operator $\alpha \leq \beta$ which expresses that α refines β as β has more behaviors than α: it is true in a state ν if all states reachable by hybrid program α from ν can be reached by hybrid program β. The program equivalence $\alpha = \beta$ is shorthand for $\alpha \leq \beta \wedge \beta \leq \alpha$. This will be made explicit by axiom (=) in Section 5.

Note the fundamental difference between dRL modal formula $[\alpha]\phi$, which expresses that all runs of hybrid program α satisfy dRL formula ϕ, compared to the dRL refinement formula $\alpha \leq \beta$, which expresses that all runs of hybrid program α are also runs of hybrid program β. Both dRL formulas refer to the runs of a hybrid program α, but only the former states a property of the (final) states reached, while only the latter relates the overall transition behavior of hybrid program α to that of another program. Just like $[\alpha]\phi$, formula $\alpha \leq \beta$ is a dRL formula and not just a judgment, so it can be true in some states and false in others. This makes it possible to easily express conditional refinement as $\phi \rightarrow \alpha \leq \beta$ meaning that if ϕ is true initially, then α refines β. The logic dRL is closed under all operators. For example the dRL formula $[\alpha]\beta \leq \gamma$ expresses that after all runs of α it is the case that all runs of β are also runs of γ. Just like in an ordinary implication, $\phi \rightarrow \alpha \leq \beta$ says nothing about what happens when the initial state does not satisfy ϕ. Just like ordinary dynamic logic modalities, $[\alpha]\beta \leq \gamma$ says nothing about what happens before program α ran. Indeed, this extended capabilities that dRL is closed under all operators will add to its expressibility and the eloquence of its uniform substitution proof calculus.

Definition 3 (Hybrid Programs). Hybrid programs *are defined by the grammar below where* x *is a variable,* θ *is a term,* a *is a program constant,* ψ *is a differential-free formula and* α, β *are hybrid programs:*

$$\alpha, \beta ::= a \mid ?\psi \mid x := \theta \mid x := * \mid x' = \theta \,\&\, \psi \mid \alpha \cup \beta \mid \alpha; \beta \mid \alpha^*$$

The *test* $?\psi$ behaves like a skip if the formula ψ is true in the current state and blocks the system otherwise. The *assignment* $x := \theta$ instantaneously updates the value of the variable x to the value of the term θ. The *nondeterministic assignment* $x := *$ updates the value of the variable x to an arbitrary value. The *differential equation* $x' = \theta \,\&\, \psi$ behaves like a continuous evolution where both the differential equation $x' = \theta$ and the evolution domain ψ holds. The

nondeterministic choice $\alpha \cup \beta$ can behave like either α or β. The *sequence* $\alpha;\beta$ behaves like α followed by β. The *nondeterministic repetition* α^* behaves like α repeated an arbitrary natural number of times.

Example 1 (Modelling safe breaking). Let us consider a car that needs to stop before a wall at distance m. It starts from a safe position and can accelerate with acceleration A if some safety condition $\text{safe}_T(x)$ is true or brake with braking force B. The controller is run at most every T seconds. Proving its safety can be achieved by proving the following dRL formula:

$$A \geq 0 \land B > 0 \land x + v^2/2B \leq m \rightarrow [car_T]x \leq m$$
$$car_T ::= (a := -B \cup ?\text{safe}_T(x); a := A); t_0 := t; x' = v, v' = a, t' = 1 \,\&\, t - t_0 \leq T$$

Such system, called *time-triggered*, can be refined to a *event-triggered* system where the controller is sure to run before a critical event, leaving the domain $E(x)$, occurs. Event-triggered systems are easier to verify but less realistic. With dRL and the axiom ($[\leq]$) below, the time-triggered system can be proved safe by proving the safety of the event-triggered system and the refinement between the two systems:

$$A \geq 0 \land B \geq 0 \land x + v^2/2B \leq m \rightarrow car_T \leq car_E \land [car_E]x \leq m$$
$$car_E ::= (a := -B \cup ?\text{safe}_E(x); a := A); t_0 := t; x' = v, v' = a, t' = 1 \,\&\, E(x)$$

3.2 Semantics

A state ν is a mapping $\mathcal{V} \to \mathbb{R}$. The state ν_x^r agrees with the state ν except for the variable x whose value is $r \in \mathbb{R}$. State ω is a *U-variation* of ν if ω and ν are equal on the complement $U^{\complement}$ of that set of variables U. For instance, ν_x^r is an $\{x\}$-variation of ν. The set of all states is $\mathcal{S}$. The interpretation of a function symbol of arity n in *interpretation* I is a smooth function $I(f) : \mathbb{R}^n \to \mathbb{R}$.

Definition 4 (Term semantics). *The* semantics of a term θ *in interpretation* I *and state* ν *is its value* $I\nu[\![\theta]\!] \in \mathbb{R}$ *and is defined as follows:*

1. $I\nu[\![x]\!] = \nu(x)$
2. $I\nu[\![f(\theta_1, \ldots, \theta_n)]\!] = I(f)(I\nu[\![\theta_1]\!], \ldots, I\nu[\![\theta_n]\!])$
3. $I\nu[\![\theta + \eta]\!] = I\nu[\![\theta]\!] + I\nu[\![\eta]\!]$
4. $I\nu[\![\theta \cdot \eta]\!] = I\nu[\![\theta]\!] \cdot I\nu[\![\eta]\!]$
5. $I\nu[\![(\theta)']\!] = \sum\limits_{x \in \mathcal{V}} \nu(x') \frac{\partial I[\![\theta]\!]}{\partial x}(\nu) = \sum\limits_{x \in \mathcal{V}} \nu(x') \frac{\partial I\nu[\![\theta]\!]}{\partial x}$

The partial derivative $\frac{\partial I\nu[\![\theta]\!]}{\partial x}$ corresponds to the derivative of the one-dimensional function $X \mapsto I\nu_x^X[\![\theta]\!]$ at $X = \nu(x)$. Since $I\nu[\![\theta]\!]$ denotes a smooth function, the derivative always exists.

Since hybrid programs appear in formulas and vice versa, the interpretation of hybrid programs and formulas is defined by simultaneous induction. The interpretation of a predicate symbol of arity n in interpretation I is an n-ary relation $I(p) \subseteq \mathbb{R}^n$. The interpretation of a program constant symbol a in interpretation I is a state-transition relation $I(a) \subseteq \mathcal{S} \times \mathcal{S}$ where $(\nu, \omega) \in I[\![a]\!]$ iff the program constant a can reach the state ω starting from the state ν.

Definition 5 (dRL semantics). *The* semantics of a formula ϕ *for an interpretation* I *is the subset* $I[\![\phi]\!] \subseteq \mathcal{S}$ *of states in which* ϕ *is true and defined as:*

1. $\nu \in I[\![\theta \leq \eta]\!]$ *iff* $I\nu[\![\theta]\!] \leq I\nu[\![\eta]\!]$
2. $\nu \in I[\![p(\theta_1, \ldots, \theta_n)]\!]$ *iff* $(I\nu[\![\theta_1]\!], \ldots, I\nu[\![\theta_n]\!]) \in I(p)$
3. $\nu \in I[\![\neg\phi]\!]$ *iff* $\nu \notin I[\![\phi]\!]$
4. $\nu \in I[\![\phi \wedge \psi]\!]$ *iff* $\nu \in I[\![\phi]\!]$ *and* $\nu \in I[\![\psi]\!]$
5. $\nu \in I[\![\forall x\, \phi]\!]$ *iff* $\nu^r_x \in I[\![\phi]\!]$ *for all* $r \in \mathbb{R}$
6. $\nu \in I[\![[\alpha]\phi]\!]$ *iff* $\omega \in I[\![\phi]\!]$ *for all* $(\nu, \omega) \in I[\![\alpha]\!]$
7. $\nu \in I[\![\alpha \leq \beta]\!]$ *iff* $(\nu, \omega) \in I[\![\beta]\!]$ *for all* $(\nu, \omega) \in I[\![\alpha]\!]$

A formula ϕ is *valid in* I if $I[\![\phi]\!] = \mathcal{S}$. A formula ϕ is *valid* if it is valid in all interpretations.

Definition 6 (Transition semantics of programs). *The* semantics of a hybrid program α *for an interpretation* I *is the transition relation* $I[\![\alpha]\!] \subseteq \mathcal{S} \times \mathcal{S}$ *and is defined as follows:*

1. $I[\![a]\!] = I(a)$
2. $I[\![?\psi]\!] = \{(\nu, \nu) \;:\; \nu \in I[\![\psi]\!]\}$
3. $I[\![x := *]\!] = \{(\nu, \nu^r_x) \;:\; \text{for all } r \in \mathbb{R}\}$
4. $I[\![x := \theta]\!] = \{(\nu, \nu^r_x) \;:\; r = I\nu[\![\theta]\!]\}$
5. $I[\![x' = \theta \,\&\, \psi]\!] = \{(\nu, \omega) \;:\; \varphi(0) \text{ is a } \{x'\}\text{-variation of } \nu \text{ and } \omega = \varphi(r)$ *for some function* $\varphi : [0, r] \to \mathcal{S}$ *where* $\varphi(t)$ *is a* $\{x, x'\}$*-variation of* ν, *satisfies* $\varphi(t) \in I[\![x' = \theta \wedge \psi]\!]$ *and* $\frac{\mathsf{d}I\varphi(t)[\![x]\!]}{\mathsf{d}t} = I\varphi(t)[\![\theta]\!]$ *for all* $t \in [0, r]\}$
6. $I[\![\alpha \cup \beta]\!] = I[\![\alpha]\!] \cup I[\![\beta]\!]$
7. $I[\![\alpha; \beta]\!] = I[\![\alpha]\!] \circ I[\![\beta]\!] = \{(\nu, \omega) : (\nu, \mu) \in I[\![\alpha]\!], (\mu, \omega) \in I[\![\beta]\!] \text{ for some } \mu \in \mathcal{S}\}$
8. $I[\![\alpha^*]\!] = \bigcup_{n \in \mathbb{N}} I[\![\alpha^n]\!]$

Most importantly, $\alpha \leq \beta$ is true in a state ν iff all states ω reachable from ν by running program α are also reachable by running β from ν.

The transition for a differential equation $x' = \theta \,\&\, \psi$ synchronizes the differential symbol x' with the current time-derivative of x, i.e. θ, and then evolves the system continuously along the solution φ of the differential equation $x' = \theta$ within the domain ψ. Differential equations are the only hybrid programs that intrinsically relate variables with their associated differential symbol.

As differential equations effectively *change* the value of differential symbols, this is taken into account in the semantics of refinements. The differential equations $x' = 1$ and $x' = 2$ are *not* equivalent: although both can reach the same values for x, their respective end states will always have a different value for x'. This behavior differs from the original semantics of dRL [19]. Intuitively, this notion of refinement corresponds to assuming that differential equations evolve with a global time $t' = 1$. Other extensions of dL like dL_{CHP} [7] already assume the presence of such global time. This property allows to express refinements of differential equations as a dL formula as shown in the axiom (ODE) below.

3.3 Static Semantics

Uniform substitution relies on the notions of free and bound variables to prevent any unsound substitution attempts. Static semantics gives a definition for free and bound variables of terms, formulas and hybrid programs based on their (dynamic) semantics, which can be defined as in dL [25]:

Definition 7 (Static semantics). *The* static semantics *defines the free variables* $\mathsf{FV}(\theta)$, $\mathsf{FV}(\phi)$ *and* $\mathsf{FV}(\alpha)$*, which are the variables whose values the expression depends on, and the bound variables* $\mathsf{BV}(\alpha)$*, which are the variables whose values may change during the execution of* α*. They are defined formally as follows:*

$$\begin{aligned}
\mathsf{FV}(\theta) &= \{x \in \mathcal{V} \ : \ \exists I, \nu, \tilde{\nu} \ a \ \{x\}\text{-variation of } \nu \text{ such that } I\nu[\![\theta]\!] \neq I\tilde{\nu}[\![\theta]\!]\} \\
\mathsf{FV}(\phi) &= \{x \in \mathcal{V} \ : \ \exists I, \nu, \tilde{\nu} \ a \ \{x\}\text{-variation of } \nu \text{ such that } \nu \in I[\![\phi]\!] \not\ni \tilde{\nu}\} \\
\mathsf{FV}(\alpha) &= \{x \in \mathcal{V} \ : \ \exists I, \nu, \omega, \tilde{\nu} \ a \ \{x\}\text{-variation of } \nu \text{ such that } (\nu, \omega) \in I[\![\alpha]\!] \\
&\qquad \text{and } \forall \tilde{\omega} \ \{x\}\text{-variation of } \omega \text{ such that } (\tilde{\nu}, \tilde{\omega}) \notin I[\![\alpha]\!]\} \\
\mathsf{BV}(\alpha) &= \{x \in \mathcal{V} \ : \ \exists I, \nu, \omega \text{ such that } (\nu, \omega) \in I[\![\alpha]\!] \text{ and } \nu(x) \neq \omega(x)\}
\end{aligned}$$

Free and bounds variables are the only information needed about the logic to ensure that the result of uniform substitution is only defined when sound. The coincidence lemmas [25] show that the truth-values of formulas only depend on their free variables and the interpretation of the symbols appearing in them (similarly for terms and hybrid programs). The set of function, predicate, and program symbols appearing in a formula, term or hybrid program is denoted $\Sigma(\cdot)$.

Lemma 1 (Coincidence for terms [25]). *The set* $\mathsf{FV}(\theta)$ *is the smallest set with the coincidence property for* θ*: If* $\nu = \tilde{\nu}$ *on* $V \supseteq \mathsf{FV}(\theta)$ *and* $I = J$ *on* $\Sigma(\theta)$*, then* $I\nu[\![\theta]\!] = J\tilde{\nu}[\![\theta]\!]$*.*

Lemma 2 (Coincidence for formulas [25]). *The set* $\mathsf{FV}(\phi)$ *is the smallest set with the coincidence property for* ϕ*: If* $\nu = \tilde{\nu}$ *on* $V \supseteq \mathsf{FV}(\phi)$ *and* $I = J$ *on* $\Sigma(\phi)$*, then* $\nu \in I[\![\phi]\!]$ *iff* $\tilde{\nu} \in J[\![\phi]\!]$*.*

Lemma 3 (Coincidence for hybrid programs [25]). *The set* $\mathsf{FV}(\alpha)$ *is the smallest set with the coincidence property for* α*: If* $\nu = \tilde{\nu}$ *on* $V \supseteq \mathsf{FV}(\alpha)$ *and* $I = J$ *on* $\Sigma(\alpha)$*, then* $(\nu, \omega) \in I[\![\alpha]\!]$ *implies* $(\tilde{\nu}, \tilde{\omega}) \in J[\![\alpha]\!]$ *for some* $\tilde{\omega}$ *with* $\omega = \tilde{\omega}$ *on* V*.*

The proof [25] requires a mutual induction on the structure of the formula and hybrid program to show that $I[\![\phi]\!] = J[\![\phi]\!]$ and $I[\![\alpha]\!] = J[\![\alpha]\!]$ which extends to the refinement case. The rest is done by induction on the set of variables S where the states ν and $\tilde{\nu}$ can differ.

Lemma 4 (Bound effect [25]). *The set* $\mathsf{BV}(\alpha)$ *is the smallest set with the bound effect property for* α*: If* $(\nu, \omega) \in I[\![\alpha]\!]$*, then* $\nu = \omega$ *on* $\mathsf{BV}(\alpha)^{\complement}$*.*

These sets are the smallest sets with the coincidence property, which means that all conservative extensions of these sets can also be used soundly. We define $\mathrm{FV}(\theta), \mathrm{FV}(\phi), \mathrm{FV}(\alpha)$ and $\mathrm{BV}(\alpha)$ as such overapproximations that can be computed syntactically. Computing the free variables for a formula $[\alpha]\phi$ requires the *must-bound variables* of the hybrid program α, written $\mathrm{MBV}(\alpha)$. They represent the variables that will be written in all executions of α. These sets are given in [31] and are constructed in a standard way [25], except for the new refinement operator.

Since the behavior of hybrid program α and β only depends on their respective free variables (Lemma 3), it would be tempting to define $\mathrm{FV}(\alpha \leq \beta) = \mathrm{FV}(\alpha) \cup \mathrm{FV}(\beta)$ stating that the refinement depends on the variables for which either program depends on. Somewhat surprisingly, this would be unsound for reasons that truly touch on the nature of refinement. Take the refinement formula $?\mathit{true} \leq x := 1$ and a state ν with $\nu(x) = 0$. Then $\nu \notin I[\![?\mathit{true} \leq x := 1]\!]$. However if the initial value of x is 1, then the refinement holds: $\nu^1_x \in I[\![?\mathit{true} \leq x := 1]\!]$, because the assignment $x := 1$ has no effect. In fact $\mathsf{FV}(?\mathit{true} \leq x := 1) = \{x\}$ even though $\mathsf{FV}(?\mathit{true}) = \mathsf{FV}(x := 1) = \emptyset$. To obtain a sound definition of $\mathrm{FV}(\alpha \leq \beta)$, one needs to take into account the variables that may be written in one program, $\mathrm{BV}(\alpha) \cup \mathrm{BV}(\beta)$, but that can also remain unmodified (which makes them depend on their initial values), so not in $\mathrm{MBV}(\alpha) \cap \mathrm{MBV}(\beta)$. Hence, the (syntactic) free variables of a refinement are defined as follows:

$$\mathrm{FV}(\alpha \leq \beta) = \mathrm{FV}(\alpha) \cup \mathrm{FV}(\beta) \cup ((\mathrm{BV}(\alpha) \cup \mathrm{BV}(\beta)) \setminus (\mathrm{MBV}(\alpha) \cap \mathrm{MBV}(\beta)))$$

With this definition for refinements as the only but notable outlier to an otherwise standard definition of the syntatic computations for a static semantics [25], the static semantics $\mathrm{FV}(\phi)$ etc. can be proved to be sound overapproximations of the static semantics $\mathsf{FV}(\phi)$ from Def. 7 and thereby enjoy the coincidence lemmas 1–3 and the bound effect lemma 4, respectively.

Lemma 5 (Soundness of static semantics). *For all terms θ, formulas ϕ and hybrid programs α:*

$$FV(\theta) \supseteq \mathsf{FV}(\theta) \quad FV(\phi) \supseteq \mathsf{FV}(\phi) \quad FV(\alpha) \supseteq \mathsf{FV}(\alpha) \quad BV(\alpha) \supseteq \mathsf{BV}(\alpha)$$

The proof of $\mathrm{FV}(\cdot) \supseteq \mathsf{FV}(\cdot)$ for formulas and hybrid programs is the only case affected by the addition of refinement operators compared to prior proofs [25, Lem. 17]. It is proved by induction on the structure of the formulas and hybrid programs. For hybrid programs, the property shown for $\mathrm{FV}(\alpha)$ is stronger than the coincidence property from Lemma 3, enforcing $\omega = \tilde{\omega}$ on $V \cup \mathrm{MBV}(\alpha)$ rather than V.

For the case of the refinement operator $\alpha \leq \beta$, the main insight is visible when proving that $\tilde{\nu} \in J[\![\alpha \leq \beta]\!]$ implies $\nu \in I[\![\alpha \leq \beta]\!]$ with $\nu = \tilde{\nu}$ on V and $I = J$ on $\Sigma(\alpha \leq \beta)$. For any $(\nu, \omega) \in I[\![\alpha]\!]$, we have $(\tilde{\nu}, \tilde{\omega}) \in J[\![\alpha]\!]$, $(\tilde{\nu}, \tilde{\omega}) \in J[\![\beta]\!]$ and $(\nu, \mu) \in I[\![\beta]\!]$ for some states $\tilde{\omega}, \mu$ by repeated use of the induction hypothesis and the definition of refinement. Both the induction hypothesis and Lemma 4 give us information on $\tilde{\omega}$ and μ. As $V \supseteq \mathrm{FV}(\alpha \leq \beta)$, the definition of $\mathrm{FV}(\alpha \leq \beta)$ is crucial for ensuring that this knowledge is enough to fully determine $\tilde{\omega}$ and μ from ν, ω and $\tilde{\nu}$, and then that $\omega = \mu$.

4 Uniform Substitution

A *uniform substitution* σ is a mapping from terms of the form $f(\cdot)$ to terms $\sigma(f(\cdot))$, from formulas of the form $p(\cdot)$ to formulas $\sigma(p(\cdot))$, and from program constants a to hybrid programs $\sigma(a)$. The reserved 0-ary function symbol $\cdot$ marks the position where the argument, e.g. θ in $p(\theta)$, will be substituted in the resulting expression. Soundness of such substitutions requires that the substitution does not introduce new free variables in a context where they are bound [10].

$$
\begin{array}{rll}
\sigma^U(x) = \sigma(x) & & \text{for } x \in \mathcal{V} \\
\sigma^U(f(\theta)) = \{\cdot \mapsto \sigma^U\theta\}^{\emptyset}\sigma(f(\cdot)) & & \text{if } \mathrm{FV}(\sigma(f(\cdot))) \cap U = \emptyset \\
\sigma^U(\theta + \eta) = \sigma^U\theta + \sigma^U\eta & & \\
\sigma^U(\theta \cdot \eta) = \sigma^U\theta \cdot \sigma^U\eta & & \\
\sigma^U((\theta)') = (\sigma^{\mathcal{V}}\theta)' & & \\
\hline
\sigma^U(p(\theta)) = \{\cdot \mapsto \sigma^U\theta\}^{\emptyset}\sigma(p(\cdot)) & & \text{if } \mathrm{FV}(\sigma(p(\cdot))) \cap U = \emptyset \\
\sigma^U(\neg\phi) = \neg\sigma^U\phi & & \\
\sigma^U(\phi \wedge \psi) = \sigma^U\phi \wedge \sigma^U\psi & & \\
\sigma^U(\forall x\, \phi) = \forall x\, \sigma^{U\cup\{x\}}\phi & & \\
\sigma^U([\alpha]\phi) = [\sigma^U_V\alpha]\sigma^V\phi & & \\
\sigma^U(\alpha \leq \beta) = \sigma^U_V\alpha \leq \sigma^U_W\beta & & \\
\hline
\sigma^U_{U\cup \mathrm{BV}(\sigma(a))}(a) = \sigma(a) & & \\
\sigma^U_U(?\phi) = ?\sigma^U\phi & & \\
\sigma^U_{U\cup\{x\}}(x := \theta) = x := \sigma^U\theta & & \\
\sigma^U_{U\cup\{x\}}(x := *) = x := * & & \\
\sigma^U_{U\cup\{x,x'\}}(x' = \theta\,\&\,\phi) = x' = \sigma^{U\cup\{x,x'\}}\theta\,\&\,\sigma^{U\cup\{x,x'\}}\phi & & \\
\sigma^U_{V\cup W}(\alpha \cup \beta) = \sigma^U_V\alpha \cup \sigma^U_W\beta & & \\
\sigma^U_W(\alpha;\beta) = \sigma^U_V\alpha;\sigma^V_W\beta & & \\
\sigma^U_V(\alpha^*) = (\sigma^V_V\alpha)^* & & \text{where } \sigma^U_V\alpha \text{ is defined}
\end{array}
$$

Fig. 1: Recursive application of uniform substitution with input taboos $U \subseteq \mathcal{V}$

Fig. 1 defines the result $\sigma^U\phi$ of applying a uniform substitution σ with taboo set $U \subseteq \mathcal{V}$ to a formula ϕ (or term θ, or hybrid programs α respectively) [28]. For hybrid programs α, the substitution result $\sigma^U_V\alpha$ for input taboo $U \subseteq \mathcal{V}$ also outputs a taboo set $V \subseteq \mathcal{V}$, written in subscript notation, that will be tabooed after program α. Taboos U, V are sets of variables that cannot be substituted in free during the application of the substitution, because they have been bound

within the context and, thus, potentially changed their meaning compared to the original substitution σ. The difference is that the input U is already taboo when the substitution σ is applied to α while V is the new output taboo after α. Finally, $\sigma(\phi)$ is short for $\sigma^{\emptyset}\phi$ started without initial taboos. The key advantage to working with uniform substitution applications with taboo passing is that they enable an efficient one-pass substitution [28] compared to the classical Church-style uniform substitution application mechanism that checks admissibility at every binding operator along the way [25]. One-pass uniform substitution postpones admissibility checks till the actual substitutions of function and predicate symbols according to explicit taboos carried around.

Despite the surprising definition of the free variables of a refinement, defining uniform substitution for the refinement case is standard, the input taboo U is given to both programs except that their output taboos V, W are discarded:

$$\sigma^U(\alpha \leq \beta) = \sigma^U_V\alpha \leq \sigma^U_W\beta$$

The reason is two-fold:

1. Unlike quantifiers and modalities, refinements do not subsequently bind any variables.
2. The free variables of a refinement introduced by a substitution can only be introduced free in the programs, and thus checking these against the input taboo set U is sufficient.

This last statement is a consequence of $\mathrm{BV}(\sigma\alpha) \subseteq \mathrm{BV}(\alpha)$ and $\mathrm{MBV}(\sigma\alpha) \supseteq \mathrm{MBV}(\alpha)$, which is proved by a direct induction.

4.1 Uniform Substitutions and Adjoint Interpretations

The proof of the soundness of uniform substitution follows the same structure as the proof of the uniform substitution lemma for dGL [28] but adapted to hybrid programs instead of hybrid games and generalized to the presence of refinements. The output taboo V of a uniform substitution $\sigma^U_V\alpha$ will include the original taboo set U and all variables bound in the program α.

Lemma 6 (Taboo set computation [28]). *If $\sigma^U_V\alpha$ is defined, then $V \supseteq U \cup \mathsf{BV}(\sigma^U_V\alpha)$.*

Whereas uniform substitutions are syntactic transformations on expressions, their semantic counterparts are semantic transformations on interpretations. The two are related by Lemmas 7 and 8. Let $I^d_{\cdot}$ denote the interpretation that agrees with interpretation I except for the constant function symbol $\cdot$ which is interpreted as the constant $d \in \mathbb{R}$.

Definition 8 (Adjoint interpretation). *For an interpretation I and a state ω, the* adjoint interpretation $\sigma^*_\omega I$ *modifies the interpretation of each function*

symbol $f \in \sigma$, predicate symbol $p \in \sigma$ and program constant $a \in \sigma$ as follows:

$$\sigma^*_\omega I(f) : \mathbb{R} \to \mathbb{R}; d \mapsto I^d_\bullet \omega[\![\sigma f(\cdot)]\!]$$
$$\sigma^*_\omega I(p) = \{d \in \mathbb{R} \ : \ \omega \in I^d_\bullet[\![\sigma p(\cdot)]\!]\}$$
$$\sigma^*_\omega I(a) = I[\![\sigma a]\!]$$

Lemma 7 (Uniform substitution for terms [28]). *The uniform substitution σ for taboo $U \subseteq \mathcal{V}$ and its adjoint interpretation $\sigma^*_\omega I$ for I, ω have the same semantics on U-variations ν of ω for all* terms *θ:*

$$I\nu[\![\sigma^U \theta]\!] = \sigma^*_\omega I\nu[\![\theta]\!]$$

Lemma 8 (Uniform substitution for formulas, programs). *Uniform substitution σ for taboo $U \subseteq \mathcal{V}$ and its adjoint interpretation $\sigma^*_\omega I$ for I, ω have the same semantics on U-variations ν of ω for all* formulas *ϕ and* hybrid programs *α:*

$$\textit{for all } U\textit{-variations } \nu \textit{ of } \omega : \nu \in I[\![\sigma^U \phi]\!] \textit{ iff } \nu \in \sigma^*_\omega I[\![\phi]\!]$$

$$\textit{for all states } \mu \textit{ and all } U\textit{-variations } \nu \textit{ of } \omega : (\nu, \mu) \in I[\![\sigma^U_V \alpha]\!] \textit{ iff } (\nu, \mu) \in \sigma^*_\omega I[\![\alpha]\!]$$

The proof is done by simultaneous induction on the structure of σ, α and ϕ for all U, ν, ω and μ [31]. The use of U-variations is critical when the induction hypothesis needs to be used in a state other than ν, e.g. for quantifiers and modalities. Without considering the extension of the refinement operator, this result was previously proved in a weaker form ($U = \emptyset$) for dL [25] or for more complex semantics like hybrid games [28].

4.2 Soundness of Uniform Substitution

Lemma 8 is essentially all that is required to ensure the sound application of uniform substitution. First, uniform substitution can be used to have a sound instantiation of the axioms, using the uniform substitution rule (US). A proof rule is *sound* if the validity of the premises implies the validity of the conclusion.

Theorem 1 (Soundness of uniform substitution [28]). *The proof rule (US) is sound.*

$$\text{(US)} \ \frac{\phi}{\sigma(\phi)}$$

Uniform substitution can also be used on rules or whole inferences, as long as they are *locally sound*, i.e. the conclusion is valid in any interpretation where the premises are valid. Locally sound inferences are also sound.

Theorem 2 (Soundness of uniform substitution for rules [28]). *All locally sound inferences remain locally sound when substituted with a uniform substitution σ with taboo set $\mathcal{V}$.*

$\frac{\phi_1 \quad \dots \quad \phi_n}{\psi}$ *locally sound implies* $\frac{\sigma^{\mathcal{V}}\phi_1 \quad \dots \quad \sigma^{\mathcal{V}}\phi_n}{\sigma^{\mathcal{V}}\psi}$ *locally sound.*

5 Proof Calculus

Most notably, uniform substitution makes it possible to use concrete dRL formulas as axioms instead of axiom schemata that accept infinitely many formulas as axioms. Axioms are finite syntactic objects, and are thus easy to implement, while axiom schemata are ultimately algorithms accepting certain formulas as input while rejecting others [25]. Figure 2 lists the axioms of dRL. dRL also satisfies the axioms of KAT [17], Schematic KAT [4] and the axioms of dL [31]. Some axioms use the reverse implication $\phi \leftarrow \psi$ instead of $\psi \rightarrow \phi$ for emphasis.

In the axiom ($[\leq]$), $\bar{x}$ stands for the (finite) vector of all relevant variables (alternative treatments [25,28] of $p(\bar{x})$ use quantifier symbols or additional program constants instead, but are not necessary for this paper). This characteristic axiom of dRL expresses that if formula $p(\bar{x})$ holds after all runs of hybrid program b, then it also holds after any refinement a. Thus, as long as a proof of the refinement is given, it is possible to replace hybrid programs inside modalities. In general, axioms are meant to be applied to the axiom key (marked blue).

Refinement is transitive ($\leq_t$), allowing the introduction of intermediate refinements c similar to the role that cuts play in first-order logic.

Axioms ($\cup_l$) and ($\cup_r$) decompose the choice operator using logical connectives. As the choice $a \cup b$ can behave like either subprograms, whenever it refines a program c, both a and b must refine c. Axiom ($\cup_r$) is not an equivalence though. $a \leq b \vee a \leq c$ says that for each initial state, one of the two refinement holds. However, when a is nondeterministic, and so can have multiple end states for one initial state, it may not be the case despite the left-hand side being true.

Axiom (;) helps proving a refinement between two sequences of programs ($a; b \leq c; d$) by proving the refinement of the first programs ($a \leq c$) and the refinement of the second programs, but only after all executions of a ($[a]b \leq d$). Axioms ($?_{\text{det}}$) and ($:=_{\text{det}}$) are particular cases of the axiom (;) where the implication can be strengthened to an equivalence. As such, the implication from right to left is not required for both axioms [31].

Axioms (loop_l), (loop_r) and (unloop) are used to prove refinements of loops. The first two state that if adding a program before or after only leads to less executions, then adding an unbounded number of executions, i.e. a loop, will also lead to less executions. The axiom (unloop) is useful for comparing two loops, as it allows to reduce the problem to comparing the loop bodies. Both axioms (loop_l) and (unloop) need a box modality when proving the refinement of the loop body, as the refinement must be proved after any number of iterations of a.

The axiom (ODE) describes how to prove refinements between differential equations. A refinement $x' = f(x) \,\&\, p(x) \leq x' = g(x) \,\&\, q(x)$ is true iff throughout the execution of the former ODE, it always satisfies the latter differential equation and evolution domain. Along with the axioms ($\text{DW}_=$) and ($\text{DE}_=$), these axioms subsume differential cut (DC), differential weakening (DW) and differential effect (DE) from dL [31]. The equivalence in the axiom (ODE) effectively means that refinements of differential equations can *always* be reduced to standard dL formulas, which is essential to our decidability result.

$$(\leq_t)\ a \leq b \leftarrow a \leq c \wedge c \leq b \qquad (\cup_l)\ a \cup b \leq c \leftrightarrow a \leq c \wedge b \leq c$$

$$(=)\ a = b \leftrightarrow a \leq b \wedge b \leq a \qquad (\cup_r)\ a \leq b \cup c \leftarrow a \leq b \vee a \leq c$$

$$([\leq])\ a \leq b \rightarrow ([a]p(\bar{x}) \leftarrow [b]p(\bar{x})) \qquad (;)\ a;b \leq c;d \leftarrow a \leq c \wedge [a]b \leq d$$

$$(?)\ (?p \leq ?q) \leftrightarrow (p \rightarrow q) \qquad (\text{loop}_l)\ a^*;b \leq b \leftarrow [a^*]a;b \leq b$$

$$(:=)\ x := f\ =\ x := *; ?x = f \qquad (\text{loop}_r)\ a;b^* \leq a \leftarrow a;b \leq a$$

$$(?_{\text{det}})\ ?p;a \leq ?p;b \leftrightarrow [?p]a \leq b \qquad (\text{unloop})\ a^* \leq b^* \leftarrow [a^*](a \leq b)$$

$$(\text{stutter})\ x := x\ =\ ?\mathit{true} \qquad (:=_{\text{det}})\ x := f;a \leq x := f;b \leftrightarrow [x := f]a \leq b$$

$$(:*_{\text{merge}})\ x := *; ?p(x); x := *\ =\ x := *; ?\exists y\, p(y)$$

$$(:=*_{\text{merge}})\ x := *; ?p(x); x := f(x)\ =\ x := *; ?\exists y\, (p(y) \wedge x = f(y))$$

$$(\text{ODE})\ x' = f(x)\,\&\,p(x) \leq x' = g(x)\,\&\,q(x) \leftrightarrow [x' = f(x)\,\&\,p(x)](x' = g(x) \wedge q(x))$$

$$(\text{DW}_=)\ x' = f(x)\,\&\,p(x)\ =\ ?p(x); x' = f(x)\,\&\,p(x); ?p(x)$$

$$(\text{DE}_=)\ x' = f(x)\,\&\,p(x)\ =\ x' = f(x)\,\&\,p(x); x' := f(x)$$

$$(\text{DX})\ x' := f(x); ?p(x)\ \leq\ x' = f(x)\,\&\,p(x)$$

$$(\text{ODE}_{\text{idemp}})\ x' = f(x)\,\&\,p(x); x' = f(x)\,\&\,p(x)\ =\ x' = f(x)\,\&\,p(x)$$

Fig. 2: Axioms of dRL

The axiom (DX) states that a differential equation always has a solution for the interval $[0, 0]$. In that case, the execution succeeds only if the domain holds, and the correct value $f(x)$ is assigned to the differential variable x'. The axiom $(\text{ODE}_{\text{idemp}})$ states that following the same differential equation twice in a row is equivalent to following it only once, because the concatenation of solutions of the same differential equation is still a solution of the same differential equation.

Compared to the original sequent calculus for dRL [19], the proof rule schemata matching infinitely many instances are now replaced by a *finite* number of axioms that are concrete dRL formulas rather than standing for infinitely many instances. The infinitely many possible instances can then be recovered soundly using the uniform substitution rule (US). Because of this two-step mechanism, reasoning with the axioms can be done without considering the possible instantiations. Take for instance the sound equivalence $x := f; x := * = x := *$. The proof can be done by transitivity $(\leq_t)$ with $x := *; ?x = f; x := *$ as intermediate step [31]. But the same proof cannot be done by replacing f by any term θ: the intermediate program is not always equivalent to the other two (e.g. for $\theta = x + 1$). On the other hand, by proving the equivalence for f and then using rule (US), the equivalence can be proved for all terms θ.

The dRL axioms are also more modular than its cast-in-stone sequent calculus rules. For instance, with rule (G) and axiom (K), any implication $\phi \to \psi$, e.g. $(\cup_r)$, can be used to prove $[a]\phi \to [a]\psi$. This would not fit the shape of the corresponding sequent rule, which requires ψ at the top level. The lack of differential symbols in the original sequent calculus [19] changes the soundness of some rules: the match direction field rule (MDF) would allow rescaling the right-hand side of a differential equation, which is unsound here as it would change the resulting differential symbols. Conversely, only the reverse implication of the axiom (ODE) would be sound in the original calculus, again for lack of differential symbols. The dRL axioms are proved sound [31]:

Theorem 3 (Soundness of dRL axioms). *All axioms of* dRL *are sound.*

6 Decidability of Refinement for a Fragment of dRL

This section identifies a subset of hybrid programs for which the refinement problem is decidable. It is focused on concrete programs, i.e. programs without function symbols, predicate symbols or program constants. They have the following high-level structure: $(ctrl; plant)^*$ where a discrete, loop-free program $ctrl$, modelling a controller that sets some parameters $\bar{u}$, then a continuous program $plant$ that describes the dynamics of the variables $\bar{y}$ according to the choice of the parameters $\bar{u}$. These steps are then repeated nondeterministically. The continuous variables $\bar{y}$ (and by extension $\bar{y}'$) are expected to be distinct from the discrete variables $\bar{u}$ and also contain a global clock t which follows the differential equation $t' = 1$. The presence of the clock t is not needed for comparing the differential equations, but to distinguish between discrete executions and hybrid executions.

For two such programs, $(ctrl_a; plant_a)^*$ and $(ctrl_b; plant_b)^*$, a canonical proof of the refinement has the following shape (omitting uses of MP for brevity):

$$\text{unloop}\dfrac{\text{G}\dfrac{;\dfrac{\dfrac{\cdots}{ctrl_a \leq ctrl_b} \quad \dfrac{\cdots}{[ctrl_a](plant_a \leq plant_b)}}{ctrl_a; plant_a \leq ctrl_b; plant_b}}{[(ctrl_a; plant_a)^*](ctrl_a; plant_a \leq ctrl_b; plant_b)}}{(ctrl_a; plant_a)^* \leq (ctrl_b; plant_b)^*}$$

This means that proving the refinement of the whole programs is reduced to proving the refinement of the controllers, $ctrl_a \leq ctrl_b$ and the refinement of the plants after all $ctrl_a$ executions, $[ctrl_a](plant_a \leq plant_b)$. With our restrictions on the controllers, the first refinement is always decidable.

Lemma 9. *For concrete, discrete and loop-free controllers $ctrl_a$ and $ctrl_b$, the validity of $ctrl_a \leq ctrl_b$ is decidable by* dRL *proof.*

Given a controller $ctrl_a$, it is possible to synthesize a first-order formula $\phi_a(x, x^+)$ that characterizes the behavior of $ctrl_a$, where x (resp. x^+) corresponds to

the variables after (resp. before) the controller [21]. Using the dRL axioms, $ctrl_a \leq ctrl_b$ is provable from $\phi_a(x, x^+) \to \phi_b(x, x^+)$. The validity of the latter is decidable as it is first-order real arithmetic [33]. The full proof is in [31].

The second refinement, $[ctrl_a](plant_a \leq plant_b)$, is more complex. Let us write the two plants as $plant_a \equiv \bar{y}' = p(\bar{y}, \bar{u}) \,\&\, Q$ and $plant_b \equiv \bar{y}' = q(\bar{y}, \bar{u}) \,\&\, R$ for some polynomials $p(\bar{y}, \bar{u}), q(\bar{y}, \bar{u})$ and formulas Q, R. The axiom ODE entails that we must prove $[ctrl_a][plant_a](p(\bar{y}, \bar{u}) = q(\bar{y}, \bar{u}) \wedge R)$, which no longer contains any refinement. For the decidability result (Theorem 4) to hold, we require that the validity of this formula is decidable.

There are two cases which always ensure this. First, if the differential equation $plant_a$ admits a solution expressible in dRL (e.g. a polynomial), then using standard dL reasoning, the formula can be reduced to a first-order formula and thus its validity can be decided. The differential equation from Example 1, $x' = v, v' = a$, is such a case.

The second case is when domain R is algebraic, i.e. of the form $\bigwedge_i \bigvee_j p_{ij}(x) = 0$ for some polynomial p_{ij} and Q, the domain of $plant_a$, is a semialgebraic set [30].

The remaining question is now to show that the approach presented above is complete, meaning it always succeeds when the refinement holds. The only additional constraint we require is that the controller $ctrl_b$ is idempotent.

Definition 9 (Idempotent controller). *A controller ctrl is* idempotent *if it satisfies* $ctrl; ctrl = ctrl$.

An idempotent controller cannot reach more states by executing multiple times without any continuous dynamics happening. Pure reactive controllers, i.e. controllers for which the parameters' values only depend on the values of the continuous variables, are always idempotent. This is the case for the controllers in Example 1: $x := -B \cup ?\text{safe}_T(x); x := A$. On the other hand, counting the number of times the controller has been executed would not be idempotent.

Lemma 10. *This derived rule is invertible, if* $ctrl_b$ *is idempotent.*

$$\frac{ctrl_a; plant_a \leq ctrl_b; plant_b}{(ctrl_a; plant_a)^* \leq (ctrl_b; plant_b)^*}$$

The derivation of the rule is given in the canonical proof. The converse, that the conclusion implies the premise, is more involved [31]. Proving $ctrl_a; plant_a \leq (ctrl_b; plant_b)^*$ from $(ctrl_a; plant_a)^* \leq (ctrl_b; plant_b)^*$ is done by unfolding the loop on the left. To get rid of the loop on the right, we use the fact that $ctrl_b$ is idempotent. It means that if the global time is not modified, then we can assume without loss of generality that the controller (and thus also the plant) is executed only once. The case when the global time is modified additionally considers the value of the derivative to ensure that there is an execution of the right program that does not require looping.

With the above lemma, we can now state the decidability result.

Theorem 4 (Decidability of refinement for idempotent controllers). *For concrete hybrid programs $ctrl_a;plant_a$ and $ctrl_b;plant_b$ discrete loop-free $ctrl_a, ctrl_b$ and with $plant_a \equiv \bar{y}' = p(\bar{y},\bar{u})\,\&\,Q$ and $plant_b \equiv \bar{y}' = q(\bar{y},\bar{u})\,\&\,R$, if $ctrl_b$ is idempotent, and the validity of $[ctrl_a][plant_a](p(\bar{y},\bar{u}) = q(\bar{y},\bar{u}) \wedge R)$ is decidable, then the validity of $(ctrl_a;plant_a)^* \leq (ctrl_b;plant_b)^*$ is also decidable.*

In particular, the theorem applies to the event-triggered model and the time-triggered model templates used to show how to prove that the latter refines the former [19]. Indeed, their controller template is loop-free and idempotent and the differential equation are assumed to be solvable. Theorem 4 strengthens their result by showing the completeness of the approach.

7 Conclusion

This paper introduced a uniform substitution proof calculus for differential refinement logic dRL. This yields a parsimonious prover microkernel for hybrid systems verification that simultaneously works for properties of and relations between hybrid systems. The handling of refinement relations between hybrid systems is subtle even only in its static semantics, which makes the correctness proofs of this paper particularly interesting. The uniform substitution is one-pass [28] giving it respectable performance advantages compared to Church-style uniform substitutions. While the joint presence of differential equations reasoning and refinement reasoning causes challenges, a resulting benefit besides soundness is that a finer notion of differential equation refinement is obtained with logical decidability properties on a fragment of hybrid systems refinements.

Future work involves improving the implementation of the uniform substitution calculus in KeYmaera X. Although the prover microkernel was straightforward following the uniform substitution process and list of dRL's uniform substitution axioms, the prover would benefit from quality of life features, e.g. using the axioms to rewrite on subprograms, and an implementation of the refinement decision algorithm for the decidable fragment. Another axis of research is to combine refinements with hybrid games, with a proper semantics and adapt the new axioms of dRL to games, some of which would not be sound as is.

References

1. Abrial, J.: Modeling in Event-B - System and Software Engineering. Cambridge University Press (2010). doi: 10.1017/CBO9781139195881
2. Abrial, J., Su, W., Zhu, H.: Formalizing hybrid systems with Event-B. In: Derrick, J., Fitzgerald, J.S., Gnesi, S., Khurshid, S., Leuschel, M., Reeves, S., Riccobene, E. (eds.) Abstract State Machines, Alloy, B, VDM, and Z - Third International Conference, ABZ 2012, Pisa, Italy, June 18-21, 2012. Proceedings. LNCS, vol. 7316, pp. 178–193. Springer (2012). doi: 10.1007/978-3-642-30885-7_13
3. Afendi, M., Laleau, R., Mammar, A.: Modelling hybrid programs with Event-B. In: Raschke, A., Méry, D., Houdek, F. (eds.) Rigorous State-Based Methods - 7th International Conference, ABZ 2020, Ulm, Germany, May 27-29, 2020,

Proceedings. LNCS, vol. 12071, pp. 139–154. Springer (2020). doi: 10.1007/978-3-030-48077-6_10
4. Angus, A., Kozen, D.: Kleene algebra with tests and program schematology. Tech. Rep. TR2001-1844, Cornell University, USA (July 2001), https://ecommons.cornell.edu/items/b376f0d0-ca43-4896-b8b3-4c8a31e24ab1
5. Banach, R., Butler, M.J., Qin, S., Verma, N., Zhu, H.: Core hybrid Event-B I: single hybrid Event-B machines. Sci. Comput. Program. **105**, 92–123 (2015). doi: 10.1016/J.SCICO.2015.02.003
6. Banach, R., Zhu, H., Su, W., Huang, R.: Continuous KAOS, ASM, and formal control system design across the continuous/discrete modeling interface: a simple train stopping application. Formal Aspects Comput. **26**(2), 319–366 (2014). doi: 10.1007/S00165-012-0263-2
7. Brieger, M., Mitsch, S., Platzer, A.: Uniform substitution for dynamic logic with communicating hybrid programs. In: Pientka, B., Tinelli, C. (eds.) CADE. LNCS, vol. 14132, pp. 96–115. Springer (2023). doi: 10.1007/978-3-031-38499-8_6
8. Butler, M.J., Abrial, J., Banach, R.: Modelling and refining hybrid systems in Event-B and Rodin. In: Petre, L., Sekerinski, E. (eds.) From Action Systems to Distributed Systems - The Refinement Approach, pp. 29–42. Chapman and Hall/CRC (2016). doi: 10.1201/B20053-5
9. Butler, M.J., Maamria, I.: Practical theory extension in Event-B. In: Liu, Z., Woodcock, J., Zhu, H. (eds.) Theories of Programming and Formal Methods - Essays Dedicated to Jifeng He on the Occasion of His 70th Birthday. LNCS, vol. 8051, pp. 67–81. Springer (2013). doi: 10.1007/978-3-642-39698-4_5
10. Church, A.: Introduction to Mathematical Logic. Princeton University Press, Princeton, NJ (1956)
11. Doumane, A., Kuperberg, D., Pous, D., Pradic, P.: Kleene algebra with hypotheses. In: Bojanczyk, M., Simpson, A. (eds.) Foundations of Software Science and Computation Structures - 22nd International Conference, FOSSACS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings. LNCS, vol. 11425, pp. 207–223. Springer (2019). doi: 10.1007/978-3-030-17127-8_12
12. Dupont, G.: Correct-by-Construction Design of Hybrid Systems Based on Refinement and Proof. (Conception correcte par construction de systèmes hybrides basée sur le raffinement et la preuve). Ph.D. thesis, National Polytechnic Institute of Toulouse, France (2021), https://tel.archives-ouvertes.fr/tel-04165215
13. Felty, A., Middeldorp, A. (eds.): International Conference on Automated Deduction, CADE-25, Berlin, Germany, Proceedings, LNCS, vol. 9195. Springer, Berlin (2015). doi: 10.1007/978-3-319-21401-6
14. Fulton, N., Mitsch, S., Quesel, J.D., Völp, M., Platzer, A.: KeYmaera X: An axiomatic tactical theorem prover for hybrid systems. In: Felty and Middeldorp [13], pp. 527–538. doi: 10.1007/978-3-319-21401-6_36
15. Jeannin, J., Ghorbal, K., Kouskoulas, Y., Schmidt, A., Gardner, R., Mitsch, S., Platzer, A.: A formally verified hybrid system for safe advisories in the next-generation airborne collision avoidance system. STTT **19**(6), 717–741 (2017). doi: 10.1007/s10009-016-0434-1
16. Kabra, A., Mitsch, S., Platzer, A.: Verified train controllers for the Federal Railroad Administration train kinematics model: Balancing competing brake and track forces. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **41**(11), 4409–4420 (2022). doi: 10.1109/TCAD.2022.3197690
17. Kozen, D.: Kleene algebra with tests. ACM Trans. Program. Lang. Syst. **19**(3), 427–443 (1997). doi: 10.1145/256167.256195

18. Loos, S.M.: Differential Refinement Logic. Ph.D. thesis, Computer Science Department, School of Computer Science, Carnegie Mellon University (2016), http://reports-archive.adm.cs.cmu.edu/anon/2015/CMU-CS-15-144.pdf
19. Loos, S.M., Platzer, A.: Differential refinement logic. In: Grohe, M., Koskinen, E., Shankar, N. (eds.) LICS. pp. 505–514. ACM, New York (2016). doi: 10.1145/2933575.2934555
20. Mitsch, S., Ghorbal, K., Vogelbacher, D., Platzer, A.: Formal verification of obstacle avoidance and navigation of ground robots. I. J. Robotics Res. **36**(12), 1312–1340 (2017). doi: 10.1177/0278364917733549
21. Mitsch, S., Platzer, A.: ModelPlex: Verified runtime validation of verified cyber-physical system models. Form. Methods Syst. Des. **49**(1-2), 33–74 (2016). doi: 10.1007/s10703-016-0241-z, special issue of selected papers from RV'14
22. Pereira, A., Baumann, M., Gerstner, J., Althoff, M.: Improving efficiency of human-robot coexistence while guaranteeing safety: Theory and user study. IEEE Trans Autom. Sci. Eng. **20**(4), 2706–2719 (2023)
23. Platzer, A.: Differential dynamic logic for hybrid systems. J. Autom. Reas. **41**(2), 143–189 (2008). doi: 10.1007/s10817-008-9103-8
24. Platzer, A.: A uniform substitution calculus for differential dynamic logic. In: Felty and Middeldorp [13], pp. 467–481. doi: 10.1007/978-3-319-21401-6_32
25. Platzer, A.: A complete uniform substitution calculus for differential dynamic logic. J. Autom. Reas. **59**(2), 219–265 (2017). doi: 10.1007/s10817-016-9385-1
26. Platzer, A.: Logical Foundations of Cyber-Physical Systems. Springer (2018). doi: 10.1007/978-3-319-63588-0
27. Platzer, A.: Uniform substitution for differential game logic. In: Galmiche, D., Schulz, S., Sebastiani, R. (eds.) IJCAR. LNCS, vol. 10900, pp. 211–227. Springer (2018). doi: 10.1007/978-3-319-94205-6_15
28. Platzer, A.: Uniform substitution at one fell swoop. In: Fontaine, P. (ed.) CADE. LNCS, vol. 11716, pp. 425–441. Springer (2019). doi: 10.1007/978-3-030-29436-6_25
29. Platzer, A., Tan, Y.K.: Differential equation axiomatization: The impressive power of differential ghosts. In: Dawar, A., Grädel, E. (eds.) LICS. pp. 819–828. ACM, New York (2018). doi: 10.1145/3209108.3209147
30. Platzer, A., Tan, Y.K.: Differential equation invariance axiomatization. J. ACM **67**(1), 6:1–6:66 (2020). doi: 10.1145/3380825
31. Prebet, E., Platzer, A.: Uniform substitution for differential refinement logic (2024). doi: 10.48550/arXiv.2404.16734
32. Stauner, T., Müller, O., Fuchs, M.: Using HYTECH to verify an automative control system. In: HART. LNCS, vol. 1201, pp. 139–153. Springer (1997). doi: 10.1007/BFB0014722
33. Tarski, A., McKinsey, J.C.C.: A Decision Method for Elementary Algebra and Geometry. University of California Press, Berkeley (1951). doi: 10.1525/9780520348097

A Additional dRL Axioms

([?]) $[?q]p \leftrightarrow (q \rightarrow p)$

([:=]) $[x := f]p(x) \leftrightarrow p(f)$

([:*]) $[x := *]p(x) \leftrightarrow \forall x\, p(x)$

([∪]) $[a \cup b]p(\bar{x}) \leftrightarrow [a]p(\bar{x}) \wedge [b]p(\bar{x})$

*([*]) $[a^*]p(\bar{x}) \leftrightarrow p(\bar{x}) \wedge [a][a^*]p(\bar{x})$

(K) $[a](p(\bar{x}) \rightarrow q(\bar{x})) \rightarrow ([a]p(\bar{x}) \rightarrow [a]q(\bar{x}))$

(I) $[a^*]p(\bar{x}) \leftrightarrow p(\bar{x}) \wedge [a^*](p(\bar{x}) \rightarrow [a]p(\bar{x}))$

([;]) $[a; b]p(\bar{x}) \leftrightarrow [a][b]p(\bar{x})$

(V) $p \rightarrow [a]p$

(G) $\dfrac{p(\bar{x})}{[a]p(\bar{x})}$

(∀) $\dfrac{p(x)}{\forall x\, p(x)}$

(MP) $\dfrac{p \rightarrow q \quad p}{q}$

*(DE) $[x' = f(x) \,\&\, q(x)]p(\bar{x}) \leftrightarrow [x' = f(x) \,\&\, q(x)][x' := f(x)]p(\bar{x})$

*(DW) $[x' = f(x) \,\&\, q(x)]p(x) \leftrightarrow [x' = f(x) \,\&\, q(x)](q(x) \rightarrow p(x))$

(DI) $\big([x' = f(x) \,\&\, q(x)]p(x) \leftrightarrow [?q(x)]p(x)\big) \leftarrow (q(x) \rightarrow [x' = f(x) \,\&\, q(x)](p(x))')$

*(DC) $\big([x' = f(x) \,\&\, q(x)]p(x) \leftrightarrow [x' = f(x) \,\&\, q(x) \wedge r(x)]p(x)\big) \leftarrow [x' = f(x) \,\&\, q(x)]r(x)$

(DG) $[x' = f(x) \,\&\, q(x)]p(x) \leftrightarrow \exists y\, [x' = f(x), y' = a(x) \cdot y + b(x) \,\&\, q(x)]p(x)$

(DS) $[x' = f \,\&\, q(x)]p(x) \leftrightarrow \forall t{\geq}0\, \big((\forall 0{\leq}s{\leq}t\, q(x + fs)) \rightarrow [x := x + ft]p(x)\big)$

(a) Axioms of dL

($\leq_{\text{refl}}$) $a \leq a$

($;_{\text{id-l}}$) $?true; a = a$

(dist-l) $(a; b) \cup (a; c) = a; (b \cup c)$

(annih-l) $?false; a = ?false$

(unfold-l) $?true \cup (a; a^*) = a^*$

($\cup_{\text{id}}$) $a \cup ?false = a$

($;_{\text{assoc}}$) $a; (b; c) = (a; b); c$

($;_{\text{id-r}}$) $a; ?true = a$

(annih-r) $a; ?false = ?false$

(unfold-r) $?true \cup (a^*; a) = a^*$

(b) Axioms of KAT

($:*_{\text{comm}}$) $x := *; y := * = y := *; x := *$

($:*_{\text{test}}$) $x := *; ?p = ?p; x := *$

(c) Axioms of SKAT with nondeterministic assignment

Fig. 3: Additional axioms of dRL

Axioms of dL also include the differential axioms, e.g. $(x)' = x'$ [25], to reason on terms, which are omitted as it is not the main focus of this paper. Axioms preceded by a star can be derived from other axioms [31].