A Deductive Refinement Calculus for Differential-Algebraic Programs

Jonathan Hellwig[1], Long Qian[2(✉)], and André Platzer[1]

[1] Karlsruhe Institute of Technology, Karlsruhe, Germany
{jonathan.hellwig,platzer}@kit.edu
[2] Carnegie Mellon University, Pittsburgh, USA
longq@andrew.cmu.edu

Abstract. This paper presents *differential-algebraic refinement logic* (dARL) with which one can deductively verify both properties and relations of *differential-algebraic programs* (DAPs) that extend hybrid dynamical systems with differential-algebraic equations (DAEs). A refinement calculus is introduced that enables the sound comparison of trajectories of differential-algebraic equations, crucially utilizing a novel trace-based semantics. This enables the incremental verification/simplification of complicated DAEs, while ensuring correctness at each step by the soundness of the calculus. The calculus is shown to certify index reductions of DAEs, providing trustworthy syntactic proofs of correctness at each step of the reduction.

Keywords: differential dynamic logic · differential-algebraic equations · index reduction

1 Introduction

A large class of physical systems ranging from mechanical multi-body systems to electrical circuits are naturally modeled by a combination of algebraic and differential constraints. For instance, in electrical circuit modeling, *Kirchhoff's laws* govern how current and voltage behave across connections via a set of algebraic equations. In contrast, charge accumulation in capacitors is governed by differential equations. Systems characterized by these types of equations are formally known as *differential-algebraic equations* (DAEs). Beyond their grounding in physical laws, these equations exhibit a notable feature: they facilitate compositional modeling. Subsystems may be defined independently and then coupled by algebraic constraints. For this reason, DAEs are the theoretical foundation of popular modeling frameworks like Modelica, which feature large libraries of reusable components. However, this modeling flexibility comes at a cost: DAEs contain implicit constraints that complicate both analytical and numerical treatments. Whilst techniques exist to transform DAEs into a representation that is more amenable to analysis, safety-critical applications raise critical questions.

© The Author(s) 2026
A. Biere et al. (Eds.): IJCAR 2026, LNAI 16689, pp. 269–285, 2026.
https://doi.org/10.1007/978-3-032-32592-1_16

How can we be sure that the transformed system is equivalent to the original? Furthermore, how can we guarantee that it satisfies its safety specifications?

Differential dynamic logic (d$\mathcal{L}$) [12,14,16,17] provides a partial answer to these questions. Its deductive framework enables reasoning about complex dynamical systems, extending dynamic logic to verify correctness properties of hybrid programs. However, d$\mathcal{L}$ is built on top of explicit *ordinary differential equations* (ODEs). Because DAEs define their dynamics implicitly rather than explicitly, they require a shift towards new fundamental reasoning principles.

A foundational step towards this end, was the introduction of *Differential-algebraic programs* (DAPs) [13] extending d$\mathcal{L}$ to differential-algebraic equations. Nevertheless, the initial DAP framework was originally restricted to a special normal form [13]. A recent extension generalized DAPs to arbitrary semi-algebraic constraints, but crucially under the strong assumption that solutions are real-analytic [6]. This reliance on analyticity is a significant limitation, as it excludes a range of practically relevant non-smooth models.

However, even if the question of regularity were to be addressed, DAEs pose an additional challenge. In practice, analysis of DAEs heavily relies on *index reduction*, a technique to reveal the DAE's implicit constraints by differentiation. Unfortunately, many algorithmic implementations lack a rigorous foundation. As a result, unverified index-reduction may introduce spurious solutions or discard valid ones. Ensuring that the index-reduced model has the same solution set requires refinement reasoning. While prior work on DAPs addresses DAE verification, it lacks the formal machinery to support refinement. Conversely, refinement calculi for hybrid systems have been developed [10,20], enabling formal comparison and simplification of hybrid programs, but only for ODEs.

Contributions. This paper introduces *differential-algebraic refinement logic* (dARL), addressing the limitations outlined above through the following contributions. First, we introduce a trace-based semantics for differential-algebraic programs grounded in continuously differentiable functions. Compared with the real-analytic semantics of prior work, this semantic foundation admits a larger class of practically relevant systems, including dynamics involving min and $|\cdot|$. Second, we develop a sound refinement calculus for DAPs. This calculus enables relational reasoning between DAPs. In particular, building on this calculus we present a method to algorithmically verify index reductions.

2 Differential-Algebraic Refinement Logic

This section presents the syntax and semantics of differential-algebraic refinement logic (dARL). The logic combines core ideas from differential refinement logic [10,20] and differential-algebraic dynamic logic [6,13]. We recall the basic syntax and semantics of differential dynamic logic and extend them with definitions for differential-algebraic programs and refinement, which together provide the foundation for refinement principles introduced in Sect. 3.

2.1 Syntax

This subsection introduces the syntax of dARL. The syntax follows largely that of differential dynamic logic [12,14,16,17], and is recalled here for completeness, with the addition of refinements and differential-algebraic programs. We begin with the construction of terms. Let $\mathbb{V}$ denote the set of variables. Each variable $x \in \mathbb{V}$ has an associated *differential variable* denoted by x'. Differential variables are independent of the base variables; in particular, x' is not an abbreviation for the derivative of x, but a distinct variable. For terms, we adopt the standard term language of differential dynamic logic, including *(syntactic) differentials* [16,17], defined formally below.

Definition 1. (Term Syntax). *Let $x \in \mathbb{V}$ be a variable and $c \in \mathbb{Q}$ be a rational constant. The language of terms e, g is defined by the following grammar:*

$$e, g ::= c \mid x \mid e + g \mid e \cdot g \mid (e)'$$

The term language extends real arithmetic with differentials. Intuitively, the differential $(e)'$ captures the local change of the term e under infinitesimal changes of its variables.

Formulas and differential-algebraic programs are defined by simultaneous induction, because programs can occur within formulas and vice versa.

Definition 2. (Formula Syntax). *Let e, g be terms, $\mathbf{x} = (x_1, \ldots, x_n)$ be a finite tuple of variables and α, β be differential-algebraic programs. The language of dARL formulas P, Q is defined by the following grammar:*

$$P, Q ::= e \leq g \mid \neg P \mid P \wedge Q \mid \forall x\, P \mid [\alpha]P \mid \alpha \leq_{\mathbf{x}} \beta$$

As dARL is an extension of the first-order logic of real arithmetic, the interpretations/constructions of first-order formulas are standard. The *modal formula* $[\alpha]P$ is inherited from $\mathsf{d}\mathcal{L}$ and classical dynamic logic of discrete programs, expressing the property that P holds after *every run* of the (differential-algebraic) program α. The *(partial) refinement formula* $\alpha \leq_{\mathbf{x}} \beta$ extends the existing refinement notion [10] and is crucial to the axiomatic framework presented in this work, expressing the property that *every possible evolution* of the variables $\mathbf{x}$ attained by executing the program α can also be achieved by somehow executing the program β. Intuitively, this implies that the possible behaviors of $\mathbf{x}$ under α is a subset of the possible behaviors of $\mathbf{x}$ under β, thus α is said to be a *(partial) refinement* of β with respect to $\mathbf{x}$. Mutual refinement of programs concerning variables $\mathbf{x}$ is denoted by $\alpha =_{\mathbf{x}} \beta$, abbreviating $\alpha \leq_{\mathbf{x}} \beta \wedge \beta \leq_{\mathbf{x}} \alpha$.

The following definition introduces the syntax of *differential-algebraic programs* (DAPs).

Definition 3. (Program Syntax). *Let x be a variable, e be a term and let Q, F be formulas of real arithmetic. The language of* differential-algebraic programs *α, β is defined by the following grammar:*

$$\alpha, \beta ::= x := e \mid ?Q \mid \{\mathbf{x} : F\} \mid \alpha \cup \beta \mid \alpha;\, \beta \mid (\alpha)^*$$

Programs in dARL model hybrid systems by combining discrete dynamics with continuous evolution. In contrast to $\mathsf{d}\mathcal{L}$, which restricts continuous evolution of $\mathbf{x}$ to *ordinary differential equations*, dARL introduces *differential-algebraic programs* (DAP) which permit evolution governed by arbitrary real-arithmetic formulas over $\mathbf{x}$ and $\mathbf{x}'$. This generalization allows modeling of disturbances and implicit dynamics. An important special case of differential-algebraic programs are *differential-algebraic equations*, which consist of a set of algebraic equations $e_1(\mathbf{x}, \mathbf{x}') = 0, \ldots, e_n(\mathbf{x}, \mathbf{x}') = 0$. Unlike ODEs, such equations are in general not explicitly solvable for $\mathbf{x}'$ and need not admit locally unique solutions.

The remaining program constructors characterize discrete program structure and control flow. Intuitively, programs $x := e$ and $?Q$ are discrete and represent *discrete assignments* and *tests*. Programs $\alpha \cup \beta$, $\alpha; \beta$, $(\alpha)^*$ are program combinators that represent *nondeterministic choice* (either one of α, β can be executed), *sequential composition* (α is executed and then β is executed) and *(unbounded) repetition* (α can be executed any finite number of times) respectively.

Notation. To simplify the presentation, we adopt a vector and matrix notation. We denote vectors of variables by $\mathbf{x} = (x_1, \ldots, x_n)^\top$, vectors of terms by $\mathbf{e} = (e_1, \ldots, e_n)^\top$, and matrices of terms by $\mathbf{A} = (a_{ij})$. The matrix-vector product $\mathbf{Ae}$ is an abbreviation for a term vector whose i-th component is the i-ith coordinate $(\mathbf{Ae})_i = \sum_{j=1} a_{ij} e_j$ of the matrix product $\mathbf{Ae}$. Relational comparison for vectors $\mathbf{e}$ and $\mathbf{g}$ of the form $\mathbf{e} \leq \mathbf{g}$ abbreviate the conjunction $\bigwedge_{i=1}^{n} e_i \leq g_i$ We also write $\mathbf{x}^\bullet \equiv (\mathbf{x}, \mathbf{x}')$ for the tuple of state and differential variables.

Expressiveness of Differential-Algebraic Programs. Explicit ordinary differential equations with domain constraints, as used in $\mathsf{d}\mathcal{L}$, are subsumed as the following special-case DAP:

$$\{\mathbf{x} : \mathbf{x}' = \mathbf{f}(\mathbf{x}) \wedge Q\}$$

DAPs also allow a much broader class of systems, including, for example, linear differential-algebraic equations

$$\{\mathbf{x} : \mathbf{A}\mathbf{x}' = \mathbf{b}\},$$

which cannot in general be reduced to ordinary differential equations when the matrix $\mathbf{A}$ is not invertible. More generally, DAPs admit arbitrary semi-algebraic constraints over state and differential variables, for example

$$\{\mathbf{x} : \mathbf{g}(\mathbf{x}, \mathbf{x}') \geq 0 \wedge \mathbf{h}(\mathbf{x}, \mathbf{x}') = 0 \wedge \mathbf{r}(\mathbf{x}, \mathbf{x}') > 0\}.$$

2.2 Semantics

Prior work on DAPs defines the semantics of programs via state transition relations [6,12], representing executions via initial and final state pairs. Such semantics are insufficient in our setting, where reasoning about intermediate states is

essential, rather than just their end states. We therefore adopt a trace-based semantics, interpreting programs as sets of continuous functions from time to states. This perspective aligns naturally with our focus of continuous dynamics in this work.

A *state* $\omega : \mathbb{V} \mapsto \mathbb{R}$ is a mapping from variables to real values. In particular, states also assign values to differential variables. We write $\mathbb{S}$ for the set of all states and $\omega(x)$ for the value of variable x in state ω. The semantics of terms are standard [12,15]. For a real value $r \in \mathbb{R}$, we use the notation $(\omega)^r_x$ to denote the state obtained by updating ω at the variable x with the value r.

Definition 4. *The* semantics *of a term e is mapping $[\![e]\!] : \mathbb{S} \to \mathbb{R}$ defined inductively:*

1. $\omega[\![x]\!] = \omega(x)$,
2. $\omega[\![e + g]\!] = \omega[\![e]\!] + \omega[\![g]\!]$,
3. $\omega[\![e \cdot g]\!] = \omega[\![e]\!] \cdot \omega[\![g]\!]$,
4. $\omega[\![(e)']\!] = \sum_{v \in \mathbb{V}} \omega(v') \frac{\partial}{\partial r}(\omega)^r_v [\![e]\!]$.

Definitions (1) - (3) are standard, the valuation of differential term $\omega[\![(e)']\!]$ evaluates the differential of $(e)'$ using the chain rule, giving $e' = \sum_{v \in \mathbb{V}} v' \frac{\partial e}{\partial v}$, and evaluating the expression at the state $\omega \in \mathbb{S}$ yields definition (4). For notational convenience, we extend the valuation componentwise to vectors and matrices of terms:

$$\omega[\![\mathbf{e}]\!] = (\omega[\![e_i]\!])_i, \quad \omega[\![\mathbf{A}]\!] = (\omega[\![a_{ij}]\!])_{ij}.$$

We now define the core semantic objects of our programs: A *trace* φ of duration $T_\varphi \geq 0$ is a mapping $\varphi : [0, T_\varphi] \to \mathbb{S}$. We denote the set of traces by $\mathcal{T} = \bigcup_{T \geq 0} \mathbb{S}^{[0,T]}$. The valuation of a variable vector $\mathbf{x}$ at time t is understood pointwise as $\varphi(t)(\mathbf{x}) = (\varphi(t)(x_1), \ldots, \varphi(t)(x_n))^\top$.

A key assumption in earlier works axiomatizing continuous evolutions is that such traces are real-analytic functions in the time variable t [6,19], which makes it impossible to switch between disjuncts with different differential-algebraic equations. This work considers a much more general collection of traces by relaxing such regularity requirements.

Definition 5. *A trace $\varphi \in \mathcal{T}$ is called* $\mathbf{x}$-regular*, if*

1. $t \mapsto \varphi(t)(\mathbf{x})$ *is of class* $C^1([0, T_\varphi], \mathbb{R}^n)$,
2. $\frac{d}{dt}\varphi(t)(\mathbf{x}) = \varphi(t)(\mathbf{x}')$ *for all* $t \in [0, T_\varphi]$,
3. *for all* $y \notin \mathbf{x} \cup \mathbf{x}'$ *and all* $t \in [0, T_\varphi]$, *we have* $\varphi(t)(y) = \varphi(0)(y)$.

The set of $\mathbf{x}$*-regular traces is denoted by* $\mathcal{C}^1_{\mathbf{x}}$.

Remark 1. The derivatives at the boundary points $\{0, T_\varphi\}$ are defined as the one-sided limits from within the interval $[0, T_\varphi]$, and the duration zero trace with $T_\varphi = 0, \varphi(0)(\mathbf{x}) = \mathbf{x}$ is understood to be $\mathbf{x}$-regular.

The first property restricts the regularity of traces to the class of C^1 functions. Definition 5 uses a closed interval, which means it excludes traces whose first derivative blows up as it approaches the boundary. The second property requires *differential consistency*: the value of each differential variable must be the time derivative of its corresponding state variable. The third property constrains the bound variables of DAPs: only the variables in $\mathbf{x} \cup \mathbf{x}'$ may change along the trace, while all others must remain constant.

A central feature of $\mathsf{d}\mathcal{L}$ is its integration of both discrete and continuous programs, which can be combined using sequential composition. While sequential composition for relational semantics can be defined easily via the composition of reachability relations, a trace-based approach requires more careful treatment to ensure a well-defined (single-valued) trace is obtained rather than a general relation. To this end, we define the *concatenation operator* $\cdot : \mathcal{T} \times \mathcal{T} \to \mathcal{T}$ such that for $\varphi, \psi \in \mathcal{T}$, the concatenated trace $\varphi \cdot \psi$ is defined on $[0, T_\varphi + T_\psi]$ as

$$(\varphi \cdot \psi)(t) = \begin{cases} \varphi(t) & \text{for } t \in [0, T_\varphi), \\ \psi(t - T_\varphi) & \text{for } t \in [T_\varphi, T_\varphi + T_\psi]. \end{cases}$$

Note that this operation is defined for any two traces. If the transition states do not match, the trace exhibits an instantaneous jump at T_φ. Consequently, the concatenation of continuous traces is, in general, only a right-continuous trace. For the discrete program constructs, we define the point trace at state ω, denoted δ_ω. It is the instantaneous trace with duration $T_{\delta_\omega} = 0$ and $\delta_\omega(0) = \omega$.

The preceding definitions provide the necessary foundation to characterize the semantics of programs:

Definition 6. (Program Semantics). *The* semantics *of a program α is a mapping $[\![\alpha]\!] : \mathbb{S} \to 2^{\mathcal{T}}$, defined inductively:*

1. $[\![x := e]\!] = \omega \mapsto \{\delta_{(\omega)^{[\![e]\!]}_x}\}$
2. $[\![?Q]\!] = \omega \mapsto \{\delta_\omega \mid \omega \in [\![Q]\!]\}$,
3. $[\![\{\mathbf{x} : F\}]\!] = \omega \mapsto \{\varphi \in \mathcal{C}^1_{\mathbf{x}} \mid \varphi(0) = \omega, \varphi([0, T_\varphi]) \subseteq [\![F]\!]\}$,
4. $[\![\alpha;\ \beta]\!] = \omega \mapsto \{\varphi \cdot \psi \mid \varphi \in [\![\alpha]\!](\omega),\ \psi \in [\![\beta]\!](\varphi(T_\varphi))\}$,
5. $[\![\alpha \cup \beta]\!] = \omega \mapsto [\![\alpha]\!](\omega) \cup [\![\beta]\!](\omega)$,
6. $[\![(\alpha)^*]\!] = \omega \mapsto \bigcup_{n=0}^{\infty} [\![\alpha^n]\!](\omega)$,

where $\alpha^0 \equiv ?\top$ and $\alpha^{n+1} \equiv \alpha; \alpha^n$.

The semantics of programs is defined as a mapping from initial states to sets of traces. For a given state $\omega \in \mathbb{S}$, the notation $[\![\alpha]\!](\omega)$ denotes the set of traces originating in ω. This formulation is necessary because continuous and discrete programs treat initial states differently: continuous programs evolve from the initial state, while discrete assignment instantaneously transition to a new state. An important property of the semantics of DAPs is that traces respect the entire initial state, including values of differential variables $\mathbf{x}'$. This is crucial for ensuring the soundness of our proof calculus.

We say two traces φ and ψ *coincide* on a set of variables V, denoted by the *coincidence relation* $\psi \sim_V \varphi$, if they have the same duration $T_\varphi = T_\psi$ and agree

on the valuation of all variables (and their differential variables) in V for all $t \in [0, T_\varphi]$.

Definition 7. (Formula Semantics). *The* semantics *of a formula P is a set of states $[\![P]\!] \subseteq \mathbb{S}$ defined inductively:*

1. $[\![e \leq g]\!] = \{\omega \in \mathbb{S} \mid \omega[\![e]\!] \leq \omega[\![g]\!]\}$,
2. $[\![P \wedge Q]\!] = [\![P]\!] \cap [\![Q]\!]$,
3. $[\![\neg P]\!] = \mathbb{S} \setminus [\![P]\!]$,
4. $[\![\forall x\, P]\!] = \{\omega \in \mathbb{S} \mid \forall r \in \mathbb{R} : (\omega)^r_x \in [\![P]\!]\}$,
5. $[\![[\alpha]P]\!] = \{\omega \in \mathbb{S} \mid \forall \varphi \in [\![\alpha]\!](\omega) : \varphi(T_\varphi) \in [\![P]\!]\}$,
6. $[\![\alpha \leq_{\mathbf{x}} \beta]\!] = \{\omega \in \mathbb{S} \mid \forall \varphi \in [\![\alpha]\!](\omega)\ \exists \psi \in [\![\beta]\!](\omega) : \psi \sim_{\mathbf{x}} \varphi\}$.

The last clause defines the semantics of the (partial) refinement formula. A state ω satisfies $\alpha \leq_{\mathbf{x}} \beta$, if for every trace φ of program α starting in state ω, there exists a corresponding trace ψ in program β, such that φ and ψ coincide on the variables $\mathbf{x}$.

Although this work focuses on the continuous fragment of programs, redefining the semantics of $\mathsf{d}\mathcal{L}$ requires ensuring compatibility with its original interpretation. The following proposition establishes that this is indeed the case.

Proposition 1. *On the syntactic fragment of $\mathsf{d}\mathcal{L}$, the trace-based semantics and relational semantics are equivalent with respect to reachability: for any program α and a pair of states $(\omega, \nu) \in \mathbb{S} \times \mathbb{S}$, we have $(\omega, \nu) \in [\![\alpha]\!]_{\mathsf{d}\mathcal{L}}$, if and only if, there exists a trace $\varphi \in [\![\alpha]\!](\omega)$ with terminal state $\varphi(T_\varphi) = \nu$.*

Proof. See [7] Appendix A.1.

This proposition establishes that dARL is an *extension* of $\mathsf{d}\mathcal{L}$. In particular, all axioms for discrete programs remain valid.

3 Axioms and Rules of dARL

This section develops a sound proof calculus for dARL. This calculus combines the axiomatic treatment of differential terms with additional principles for refinements.

Syntactic Side Conditions. Syntactic side conditions are essential for the soundness of our proof calculus. We use standard definitions [16] and write $\mathrm{FV}(P)$ for the set of (syntactically) free variables of formula P and $\mathrm{BV}(\alpha)$ for the set of (syntactically) bound variables of program α. For DAPs, we set $\mathrm{BV}(\{\mathbf{x} : F\}) = \mathbf{x} \cup \mathbf{x}'$. The following lemmas formally relate syntactically free and bound variables to our trace-based semantics.

Lemma 1. (Coincidence). *Let P be a formula and $\omega, \nu \in \mathbb{S}$ be states. If $\omega \sim_V \nu$ for some variable set $\mathrm{FV}(P) \subseteq V$, then $\omega \in [\![P]\!]$, if and only if, $\nu \in [\![P]\!]$.*

Lemma 2. (Bound Effect). *For any program α and trace $\varphi \in [\![\alpha]\!](\omega)$, we have $\omega(v) = \varphi(T_\varphi)(v)$ for all $v \in \mathrm{BV}(\alpha)^{\complement}$.*

The coincidence lemma states that if two states agree on a superset of a formula's free variables, then they agree on whether the formula is satisfied. The bound effect lemma expresses that a program execution can only modify the values of its bound variables, leaving all other variables unchanged along the trace.

Differential Term Axioms. Because our calculus is based on C^1 traces, rather than real-analytic ones, some of our reasoning principles require additional regularity assumptions. To state these assumptions syntactically, we define syntactic Jacobians of term vectors.

Definition 8. (Syntactic Partial Derivative and Jacobian). *Let $x, y \in \mathbb{V}$ be variables and $c \in \mathbb{Q}$ a rational constant. For terms e, g the syntactic partial derivative $\frac{\partial(e)}{\partial x}$ is defined by*

$$\frac{\partial(e+g)}{\partial x} \equiv \frac{\partial(e)}{\partial x} + \frac{\partial(g)}{\partial x} \qquad \frac{\partial(c)}{\partial x} \equiv 0$$
$$\frac{\partial(e \cdot g)}{\partial x} \equiv \frac{\partial(e)}{\partial x} g + e \frac{\partial(g)}{\partial x} \qquad \frac{\partial(y)}{\partial x} \equiv \begin{cases} 1 & y = x \\ 0 & y \neq x \end{cases}$$

Let $\mathbf{x} = (x_1, \ldots, x_n)$ be a tuple of variables and let $\mathbf{e} = (e_1, \ldots, e_m)$ be a vector of terms. The syntactic Jacobian *of $\mathbf{e}$ with respect to $\mathbf{x}$ is the $m \times n$ matrix $\mathsf{J}_{\mathbf{x}}\mathbf{e}$ defined by:*

$$\mathsf{J}_{\mathbf{x}}\mathbf{e} \equiv \left[\frac{\partial e_i}{\partial x_j}\right]_{i,j}$$

The axiom schema J expresses the relation between syntactic differentials and syntactic Jacobians. It is only valid if the term vector $\mathbf{e}$ constrains at most the variables $\mathbf{x}$.

$$\mathsf{J} \quad (\mathbf{e})' = \mathbf{J}_{\mathbf{x}}\mathbf{e} \cdot \mathbf{x}' \quad \mathrm{FV}(\mathbf{e}) \subseteq \mathbf{x}$$

Refinement Axioms. The following two *refinement axioms* presented below are generalizations of the $[\leq]$ and $\leq_t$ axioms [20] to the partial refinement case:

$$[\leq_{\mathbf{x}}] \quad \alpha \leq_{\mathbf{x}} \beta \to ([\beta]P \to [\alpha]P) \quad \mathrm{FV}(P) \cap \mathsf{BV}(\alpha, \beta) \subseteq \mathbf{x}$$
$$\leq^t_{\mathbf{x}} \quad \alpha \leq_{\mathbf{x}} \beta \wedge \beta \leq_{\mathbf{x}} \gamma \to \alpha \leq_{\mathbf{x}} \gamma$$

The $[\leq_{\mathbf{x}}]$ axiom builds a bridge between box and refinement formulas. It is only valid under the stated side condition, which requires that any free variable of P outside $\mathbf{x}$ may not be modified by either of the two programs. Axiom $\leq^t_{\mathbf{x}}$ states the usual transitivity property of refinement.

Box Axioms. The following axioms express basic reasoning principles for box modalities of DAEs. Although they follow from elementary properties of $\mathbf{x}$-

regular traces, these axioms are sufficiently expressive to derive non-trivial properties of DAPs (c.f. Sect. 4). We write $\preceq$ to denote either $<$ or $\leq$:

$$\begin{array}{lll} \mathsf{DW} & [\{\mathbf{x} : F\}]F & \\ \mathsf{DI}_{\preceq} & e \preceq 0 \rightarrow [\{\mathbf{x} : (e)' \leq 0\}]e \leq 0 & \mathsf{FV}(e) \subseteq \mathbf{x} \\ \mathsf{DHC} & (e)' = 0 \rightarrow [\{\mathbf{x} : e = 0\}](e)' = 0 & \mathsf{FV}(e) \subseteq \mathbf{x} \end{array}$$

The *differential weakening* axiom DW axiomatizes the simple property that if F is a constraint for the system $\{\mathbf{x} : F\}$, then F must be satisfied along all traces. Axiom $\mathsf{DI}_{\preceq}$ is the *differential induction* axiom, expressing the fact that if a differential-free term e is initially (negative) non-positive and along all possible traces its differential is non-positive, then e always remains (negative) non-positive. Finally, axiom DHC is the *differential hidden constraint* axiom. It can be viewed as the converse of the $\mathsf{DI}_{\preceq}$ axiom: if a differential-free term e is identically zero on the entire time horizon, then necessarily its differential $(e)'$ must also be zero. The role of axiom DHC is discussed in Sect. 4.

Ghost Axioms. The *ghost axioms* provide reasoning principles for extending DAPs with additional variables while preserving refinement. Under our C^1 trace-based semantics, such extensions require regularity assumptions to ensure that the added variables admit a solution on the same time interval of existence as the original trace. To state these conditions, we first define a syntactic determinant.

Definition 9. (Syntactic Determinant). *Let $\mathbf{A} = (a_{ij})$ be an $n \times n$ matrix whose entries are terms. We define the term $\mathsf{det}(\mathbf{A})$ by recursion on n:*

$$\mathsf{det}([a_{11}]) \equiv a_{11}, \qquad \mathsf{det}(\mathbf{A}) \equiv \sum_{j=1}^{n} (-1)^{1+j} a_{1j}\, \mathsf{det}(\mathbf{A}_{1j}) \quad (n > 1),$$

where $\mathbf{A}_{1j}$ is obtained from $\mathbf{A}$ by deleting row 1 and column j.

Using this notation, we state the two extension principles:

$$\begin{array}{lll} \mathsf{AG} & [\{\mathbf{x} : F\}](\mathbf{e} = \mathbf{0} \wedge \det \mathbf{J}_{\mathbf{x}'}\mathbf{e} \neq 0) \rightarrow & z, z' \notin \{\mathbf{x} : F\}, g \\ & \quad \exists \mathbf{z} \exists \mathbf{z}' \, (\{\mathbf{x} : F\} \leq_{\mathbf{x}\bullet} \{\mathbf{x}, \mathbf{z} : F \wedge \mathbf{z} = \mathbf{g}\}) & \\ \mathsf{DG} & [\{\mathbf{x} : F\}] \det \mathbf{A} \neq 0 \rightarrow & z, z' \notin \{\mathbf{x} : F\}, \mathbf{A}, \mathbf{B}, \mathbf{c} \\ & \quad \forall \mathbf{z} \exists \mathbf{z}' \, (\{\mathbf{x} : F\} \leq_{\mathbf{x}\bullet} \{\mathbf{x}, \mathbf{z} : F \wedge \mathbf{A}\mathbf{z}' = \mathbf{B}\mathbf{z} + \mathbf{c}\}) & \end{array}$$

Axiom AG is the *algebraic ghost* axiom, axiomatizing the fact that if the Jacobian associated to an algebraic equation $\mathbf{e} = \mathbf{0}$ is invertible, then the system is actually explicit and the differentials $\mathbf{x}'$ can be solved *explicitly* from the equation $\mathbf{e} = \mathbf{0}$. This allows for the introduction of new variables without losing regularity. In particular, one can introduce constrained variables $\mathbf{z} = \mathbf{x}'$ (using $\mathbf{g} \equiv \mathbf{x}'$) and manipulate $\mathbf{z}$ as a differential-free variable. Axiom DG (*differential ghost*) states that (differential) variables governed by linear DAEs can be soundly

added to the system, *provided that* $\mathbf{A}$ is invertible along all traces of the original DAE $\{\mathbf{x} : F\}$. In that case, the linear DAE is equivalent to a linear ODE obtained by solving for the differential variables $\mathbf{z}'$ via $\mathbf{A}^{-1}$.

The following axioms provide reasoning principles to analyze subsystems of DAEs.

$$\begin{aligned}
&\text{DC} && [\{\mathbf{x} : F\}]R \to \{\mathbf{x} : F\} \leq_{\mathbf{x}\bullet} \{\mathbf{x} : R\} \\
&\text{DR} && \forall \mathbf{z} \forall \mathbf{z}' \left(\{\mathbf{x}, \mathbf{z} : R \wedge F\} \leq_{\mathbf{x}\bullet} \{\mathbf{x} : F\}\right) && \mathbf{z}, \mathbf{z}' \notin F \\
&\text{DM} && \{\mathbf{x} : F\} \leq_{\mathbf{x}\bullet} \{\mathbf{x} : G\} \to \{\mathbf{x} : F \wedge R\} \leq_{\mathbf{x}\bullet} \{\mathbf{x} : G \wedge R\} \\
&\text{DP} && [\{\mathbf{x} : \exists \mathbf{y}' \exists \mathbf{y} F\}]F \to [\{\mathbf{x}, \mathbf{y} : \exists \mathbf{y}' \exists \mathbf{y} G\}]G \to \\
& && \quad \{\mathbf{x} : F\} \leq_{\mathbf{x}\bullet} \{\mathbf{x} : G\} \to \{\mathbf{x}, \mathbf{y} : F\} \leq_{(\mathbf{x},\mathbf{y})\bullet} \{\mathbf{x}, \mathbf{y} : G\}
\end{aligned}$$

Axiom DC is the *differential cut* axiom, expressing that if a formula R is always satisfied along a DAE $\{\mathbf{x} : F\}$, then R can be soundly added as a constraint. Axiom DR is the *differential refinement* axiom, axiomatizing the fact that additional constraints only restrict the system. Intuitively, the *differential monotonicity* axiom DM expresses a monotonicity property of differential programs. If $\{\mathbf{x} : F\} \leq_{\mathbf{x}} \{\mathbf{x} : G\}$ holds, then imposing an additional constraint R on both systems preserves the refinement relation. The DP Axiom (*differential projection*) states that if F and G are semantically independent of $\mathbf{y}'$ and $\mathbf{y}$ along all traces, then it is sound to prove the refinement for the full system $(\mathbf{x}, \mathbf{y})$ by proving it on the $\mathbf{x}$-subsystem.

Theorem 1. (Soundness). *The dARL proof calculus is sound.*

Proof. See [7] Appendix A.2.

Derived Rules. An important feature of axiomatic, logical approaches to the verification of dynamical systems is the ability to *derive* sophisticated proof rules from a small set of sound axioms by clever deductions. Such derived axioms are then guaranteed to be sound by the soundness of the base axioms (Theorem 1). The following theorem presents the list of derived axioms established, allowing for a symbolic, deductive treatment of index reduction in Sect. 4.

Corollary 1. (Derived Rules). *Let* $\mathbf{e}$ *be a differential-free term. The following are derived rules of dARL:*

$$\text{dW}\ \frac{\Gamma \vdash P}{\Gamma \vdash [\{\mathbf{x} : F\}]P, \Delta} \qquad \text{dA}\ \frac{\Gamma \vdash R \qquad \Gamma \vdash \{\mathbf{x} : R\} \leq_{\mathbf{x}\bullet} \{\mathbf{x} : G\}, \Delta}{\Gamma \vdash \{\mathbf{x} : F\} \leq_{\mathbf{x}\bullet} \{\mathbf{x} : G\}, \Delta}$$

$$\text{dI}\ \frac{\Gamma \vdash \mathbf{e} \preceq \mathbf{0}, \Delta \qquad \Gamma \vdash [\{\mathbf{x} : F\}](\mathbf{e})' \leq \mathbf{0}, \Delta}{\Gamma \vdash \{\mathbf{x} : F\} =_{\mathbf{x}\bullet} \{\mathbf{x} : F \wedge \mathbf{e} \preceq \mathbf{0}\}, \Delta}\ \mathrm{FV}(\mathbf{e}) \subseteq \mathbf{x}$$

$$\text{dHC}\ \frac{\Gamma \vdash (\mathbf{e})' = \mathbf{0}, \Delta \qquad \Gamma \vdash [\{\mathbf{x} : F\}]\mathbf{e} = \mathbf{0}, \Delta}{\Gamma \vdash \{\mathbf{x} : F\} =_{\mathbf{x}\bullet} \{\mathbf{x} : F \wedge (\mathbf{e})' = \mathbf{0}\}, \Delta}\ \mathrm{FV}(\mathbf{e}) \subseteq \mathbf{x}$$

Proof. See [7] Appendix A.3.

- Proof rule dW is the derived version of axiom DW.
- Proof rule dA is the *differential advance* rule, where a constraint F can be replaced by R provided that R is more permissive.
- Proof rule dI is the *differential induction* rule for DAPs, the derived version of axiom $\mathrm{DI}_{\preceq}$.
- Proof rule dHC is the *differential hidden constraint* rule for equality constraints of the form $\mathbf{e} = \mathbf{0}$, the derived version of DHC.

The following example shows how dARL can be used for safety verification via refinements.

Example 1. Consider the system $\{x, y : x' = -y \wedge xy = 1\}$ with variables $\mathbf{x} = (x, y)$ and suppose we wish to prove the property that starting from $x = y = 1, x' = -1, y' = 1$, the positivity condition $y > 0$ always holds. That is, the goal is to prove the following sequent

$$x = y = 1, x' = -1, y' = 1 \vdash [\{x, y : x' = -y \wedge xy = 1\}]y > 0$$

First apply axiom DG with $\mathbf{A} = \mathbf{I}$ being the identity matrix to soundly introduce the differential ghost $z' = -y^2 z/2$ with initial condition $z^2 y = 1$ (sound as $y = 1$ initially). This application yields the refinement relation

$$\{x, y : x' = -y \wedge xy = 1\} \leq_{(x,y)\bullet} \{x, y, z : x' = -y \wedge xy = 1 \wedge z' = -y^2 z/2\}$$

As the desired safety condition is $y > 0$, an application of the refinement axiom $[\leq_{\mathbf{x}}]$ reduces the problem to proving

$$x = y = 1, x' = -1, y' = 1, z^2 y = 1 \vdash [\{x, y, z : x' = -y \wedge xy = 1 \wedge z' = -y^2 z/2\}]y > 0$$

Further note that it suffices to prove the safety property of $z^2 y = 1$, as it then follows by arithmetic reasoning that $y > 0$. To establish this, an application[1] of axiom $\mathrm{DI}_{\preceq}$ reduces the proof obligation to establishing the validity of $(z^2 y)' = 0$ along the DAE. By the syntactic definition of differentials, this evaluates to

$$(z^2 y)' = (z^2)' y + z^2 (y)' = 2z' zy + z^2 y' = -y^3 z^2 + z^2 y'$$

By axioms DW and rule dA, we may apply arithmetic reasoning and conclude that since $xy = 1$ holds, $x \neq 0$ and $(z^2 y)' = 0 \leftrightarrow x \cdot (z^2 y)' = 0$. The latter expression evaluates to (using $xy = 1$)

$$x \cdot (-y^3 z^2 + z^2 y') = -xy^3 z^2 + xz^2 y' = -y^2 z^2 + xz^2 y'$$

The proof is now seemingly stuck, as no further progress can be made without explicit information on y'. Nonetheless, axiom DHC can be applied on the algebraic equality $xy = 1$ to derive *implicit* constraints on y', this gives:

$$\vdash (xy)' = 0 \rightarrow [\{x, y : x' = -y \wedge xy = 1\}](xy)' = 0$$

[1] Technically we use the equality version, which is equivalent to applying $\mathrm{DI}_{\preceq}$ twice with different signs.

Direct computations give $(xy)' = x'y + xy'$, which is indeed equivalent to 0 at the initial conditions $x = y = 1, x' = -1, y' = 1$. Thus, axiom DHC derives that this is also true along the dynamics, simplifying with $xy = 1$ gives

$$x'y + xy' = 0, xy = 1, x' = -y \vdash xy' = y^2$$

deriving

$$x = y = 1, x' = -1, y' = 1 \vdash [\{x, y : x' = -y \wedge xy = 1\}]xy' = y^2$$

Finally, combining this with our earlier differential computation (axiom DC) proves

$$(z^2y)' = -y^2z^2 + xz^2y', xy' = y^2 \vdash (z^2y)' = -y^2z^2 + y^2z^2 = 0$$

Thus, this shows that $z^2y = 1$ is an invariant along the DAE, thereby provably establishing the safety property $y > 0$, as desired.

4 DAE Index Reduction

DAEs contain algebraic equations that may impose additional tangency conditions not syntactically present in the original set of constraints. These *hidden constraints* are the reason why DAEs require additional preprocessing steps before differential invariant reasoning becomes applicable. The following example illustrates this issue.

Example 2. Consider the term vector

$$\mathbf{f} \equiv \begin{pmatrix} x' + x - z \\ y' + y + z \\ x - y \end{pmatrix}$$

which defines the DAP $\{x, y, z : \mathbf{f} = \mathbf{0}\}$. Let $V \equiv x^2 + y^2 + z^2$. Suppose we want to prove that the sublevel set $V \leq r$ is an invariant of the system: every solution initially satisfying this bound continues to satisfy it.

A direct application of rule dI reduces the proof to two obligations: $V \leq r$ must hold initially, and all solutions must satisfy $(V)' \leq 0$.

$$\mathrm{dI}\,\frac{\mathbb{R}\,\dfrac{*}{V \leq r \vdash V \leq r} \quad V \leq r \vdash [\{x, y, z : \mathbf{f} = \mathbf{0}\}](V)' \leq 0}{V \leq r \vdash x, y, z : \mathbf{f} = \mathbf{0}\} =_{\mathbf{x}\bullet} \{x, y, z : \mathbf{f} = \mathbf{0} \wedge V \leq r\}}$$

Now, the second obligation further simplifies using differential weakening and real arithmetic:

$$\mathrm{dW},\,\mathbb{R},\,\mathrm{J}\,\frac{\mathbb{R}\,\dfrac{x' = -x + z, y' = -y - z, x = y \vdash -4x^2 + 2zz' \leq 0}{x' = -x + z, y' = -y - z, x = y \vdash 2xx' + 2yy' + 2zz' \leq 0}}{V \leq r \vdash [\{x, y, z : \mathbf{f} = \mathbf{0}\}](V)' \leq 0}$$

However, the remaining goal is not derivable from the given assumptions: after simplifying using the system's equations, we obtain

$$-4x^2 + 2zz' \leq 0.$$

The term $2zz'$ cannot be controlled, because both z and z' are unconstrained. This raises the natural question of whether there are hidden constraints among the DAE variables that make the invariant proof succeed.

In the numerical DAE literature, the systematic exposure of hidden constraints was presented by Gear [5]. A DAE is classified by its *differentiation index*, which is the minimum number of times the system's constraints need to be differentiated until all relevant differential variables are uniquely determined. The process of identifying and adding differentiated constraints to the system is called *index reduction*. Efficient implementations of index reduction (often based on structural index reduction algorithms like Pantelides' [11]) already exist and scale to large equation systems.

It makes sense to exploit such procedures as automation for discovering hidden constraints. In dARL, however, their output cannot be used directly: only transformations which *provably* preserve the behavior of the original system are admissible. Naive index reduction produces a system that is incomparable to the original, i.e., neither is a refinement of the other. The following common mistakes break the refinement relation:

1. Ignoring required initial consistency conditions,
2. Differentiating constraints without verifying required regularity conditions, which may exclude valid C^1 solutions,
3. Using unsound algebraic simplifications.

We abstract away from the implementation details of any particular index reduction procedure and isolate the steps relevant for proof reconstruction. In the following, let Γ denote a placeholder context to collect the assumptions on the initial state. Suppose we have a DAP of the following form

$$\{\mathbf{x} : \mathbf{f} = \mathbf{0} \wedge Q\}.$$

Starting from $\mathbf{f}_0 \equiv \mathbf{f}$, an index reduction procedure produces a sequence of term vectors $\mathbf{f}_0, \mathbf{f}_1, \ldots, \mathbf{f}_m$ by repeatedly exposing hidden constraints. At each iteration $k = 0, \ldots, m-1$, the relevant soundness-critical steps are as follows:

1. Select a differential-free term $\mathbf{r}_k$ such that $\mathrm{FV}(\mathbf{r}_k) \subseteq \mathbf{x}$ and

$$\mathbf{f}_k = \mathbf{0}, Q \vdash \mathbf{r}_k = 0$$

2. Augment the system with the differential of $\mathbf{r}_k$:

$$\mathbf{f}_{k+1} \equiv \mathbf{f}_k \cup (\mathbf{r}_k)'.$$

The first step identifies an algebraic consequence of the system that contains no differential variables. The second step augments the system with the differential of the chosen term. We illustrate this procedure on the running example.

Example 3. Let $\mathbf{f}_0 \equiv \mathbf{f}$. For the first iteration, we select $\mathbf{r}_0 \equiv x - y$, which is immediate from $\mathbf{f}_0 = \mathbf{0}$. Adding its derivative is valid under the obligation $\Gamma \vdash x' = y'$. The hidden constraint step is constructed as follows:

$$\text{dHC}\ \frac{\text{J, }\mathbb{R}\ \dfrac{\Gamma \vdash x' = y'}{\Gamma \vdash (x-y)' = 0} \quad \text{dW}\ \dfrac{\mathbb{R}\ \dfrac{}{\mathbf{f}_0 = \mathbf{0} \vdash x - y = 0}^{*}}{\Gamma \vdash [\{x,y,z : \mathbf{f}_0 = \mathbf{0}\}]x - y = 0}}{\Gamma \vdash \{x,y,z : \mathbf{f}_0 = \mathbf{0}\} =_{\mathbf{x}\bullet} \{x,y,z : \mathbf{f}_0 = \mathbf{0} \wedge (x-y)' = 0\}}$$

Now, set $\mathbf{f}_1 \equiv \mathbf{f}_0 \cup (x-y)'$. To proceed, we identify a new differential-free consequence of the augmented system. Using the equations in $\mathbf{f}_0 = \mathbf{0}$, the newly added constraint simplifies to

$$(x-y)' = (-x+z) - (-y-z) = -x + y + 2z = 0.$$

Together with $x = y$, this entails $z = 0$. Thus, for the next iteration we select $\mathbf{r}_1 \equiv z$. Again, adding its derivative requires the obligation $\Gamma \vdash z' = 0$. Applying the DHC rule, we have:

$$\text{dHC}\ \frac{\text{J, }\mathbb{R}\ \dfrac{\Gamma \vdash z' = 0}{\Gamma \vdash (z)' = 0} \quad \text{dW}\ \dfrac{\mathbb{R}\ \dfrac{}{\mathbf{f}_1 = \mathbf{0} \vdash z = 0}^{*}}{\Gamma \vdash [\{x,y,z : \mathbf{f}_1 = \mathbf{0}\}]z = 0}}{\Gamma \vdash \{x,y,z : \mathbf{f}_1 = \mathbf{0}\} =_{\mathbf{x}\bullet} \{x,y,z : \mathbf{f}_1 = \mathbf{0} \wedge (z)' = 0\}}$$

Finally, set $\mathbf{f}_2 \equiv \mathbf{f}_1 \cup (z)'$. Combining the two derivations, the $\leq^t_{\mathbf{x}}$ axiom yields:

$$\frac{\Gamma \vdash x' = y' \quad \Gamma \vdash z' = 0}{\Gamma \vdash \{x,y,z : \mathbf{f}_0 = \mathbf{0}\} =_{\mathbf{x}\bullet} \{x,y,z : \mathbf{f}_2 = \mathbf{0}\}}$$

Thus, the reduction from $\mathbf{f}_0$ to $\mathbf{f}_2$ is valid under any context Γ that proves $x' = y'$ and $z' = 0$. With these hidden constraints exposed, we can now prove that $V \leq r$ is an invariant. Notice that the proof relies on two conditions: proving initial consistency, and verifying that the proposed constraint is a valid logical consequence of the current system.

The following derived rule of dARL certifies *any* proposed index reduction as a mutual refinement:

Theorem 2. (Index Reduction). *Let $\mathbf{r}_0, \ldots, \mathbf{r}_m$ be a sequence of differential-free term vectors. Then, the following is a derivable proof rule of dARL:*

$$\text{IR}\ \frac{\Gamma \vdash \bigwedge_{k=0}^{m} (\mathbf{r}_k)' = \mathbf{0} \quad \vdash \bigwedge_{k=0}^{m} (\mathbf{f}_k = \mathbf{0} \wedge Q \rightarrow \mathbf{r}_k = \mathbf{0})}{\Gamma \vdash \{\mathbf{x} : \mathbf{f}_{m+1} = \mathbf{0} \wedge Q\} =_{\mathbf{x}\bullet} \{\mathbf{x} : \mathbf{f}_0 = \mathbf{0} \wedge Q\}}\ \mathrm{FV}(\mathbf{r}_1, \ldots, \mathbf{r}_m) \subseteq \mathbf{x}$$

Proof. See [7] Appendix A.4.

The first premise enforces initial consistency; the second requires that each proposed constraint is an algebraic consequence of the current system. The condition that $\mathrm{FV}(\mathbf{r}_1, \ldots, \mathbf{r}_m) \subseteq \mathbf{x}$ ensures that differentiating does not inadvertently assume higher-order regularity. Together, these conditions guarantee that any index reduction preserves the behavior of the original system avoiding the pitfalls stated above.

5 Related Work

Deductive Verification of Differential Equations. The deductive verification of differential equations aims to axiomatize all necessary principles using a small set of core axioms, and deriving other desired properties (e.g. safety, liveness) with accompanying trustworthy syntactic proofs using the sound axioms. Such approaches include differential dynamic logic ($\mathsf{d}\mathcal{L}$) and its variants [6,10,12,13], some of which can be proof-checked in KeYmaera X [4]. Sophisticated proof-rules and completeness results have also been established in the case for ordinary differential equations [18,19]. Axiomatic approaches to DAEs are relatively less explored. Earlier works [6,13] considered a direct extension of the reachability-based semantics in $\mathsf{d}\mathcal{L}$ to DAEs, and therefore do not provide a refinement-calculus capable of directly expressing index reductions. Earlier work [6] also assumes traces of DAEs to be real-analytic, which is a much stronger condition than the C^1 regularity imposed in this article.

Numerical and Structural Index Reduction. Classical index reduction techniques such as those by Gear [5], Pantelides [11], and Pryce [21] address the consistent initialization of DAEs for numerical simulation. In contrast, our goal is to reason about parametric and underspecified models within our axiomatic framework via symbolic invariant reasoning. By revealing the hidden constraints of a DAE, we can reason about safety entirely without numerical integration.

Differential Algebra. Differential algebra offers a formal treatment of differential equations through the study of differential ideals [9,22]. Later, these ideas were operationalized into effective algorithms [3,8], providing a rigorous foundation for symbolic index reduction. However, these approaches are purely structural and do not account for the semantics of continuous traces. Consequently, they lack the logical framework to reason about reachability or safety properties. We address this by integrating index reduction into a deductive proof calculus.

Formal Semantics of Modeling Languages. The work of Benveniste *et al.* [1,2] most closely aligns with our framework in terms of modeling power, as they formally address the interplay between DAEs and discrete mode transitions. However, their primary concern is the correctness of compilation and execution for DAE-based simulation software. By contrast, we go beyond execution and provide an axiomatic framework for deductive verification.

6 Conclusion

This paper presents an axiomatic framework dARL for the deductive verification of general DAEs. By leveraging a novel trace-based semantics, dARL supports sound comparison of DAEs through refinements, enabling the incremental verification of complicated DAEs via iterative simplifications. In particular, index reductions of DAEs can be completely internalized in dARL, with supporting syntactic proofs of correctness at each step of the reduction. For future work, it would be interesting to implement a proof-checker for dARL.

Acknowledgments. This work has been supported by an Alexander von Humboldt Professorship and the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - SFB 1608 - 501798263.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Benveniste, A., Caillaud, B., Elmqvist, H., Ghorbal, K., Otter, M., Pouzet, M.: Structural analysis of multi-mode DAE systems. In: Frehse, G., Mitra, S. (eds.) Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control, HSCC 2017, Pittsburgh, PA, USA, April 18-20, 2017, pp. 253–263. ACM (2017). https://doi.org/10.1145/3049797.3049806
2. Benveniste, A., Caillaud, B., Malandain, M.: The mathematical foundations of physical systems modeling languages. Annu. Rev. Control. **50**, 72–118 (2020). https://doi.org/10.1016/J.ARCONTROL.2020.08.001
3. Boulier, F., Lazard, D., Ollivier, F., Petitot, M.: Representation for the radical of a finitely generated differential ideal. In: Levelt, A.H.M. (ed.) Proceedings of the 1995 International Symposium on Symbolic and Algebraic Computation, ISSAC '95, Montreal, Canada, July 10-12, 1995, pp. 158–166. ACM (1995). https://doi.org/10.1145/220346.220367
4. Fulton, N., et al.: KeYmaera X: An axiomatic tactical theorem prover for hybrid systems, LNCS, vol. 9195, pp. 527–538. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21401-6_36
5. Gear, C.W.: Differential-algebraic equation index transformations. SIAM J. Sci. Stat. Comput. **9**(1), 39–47 (1988). https://doi.org/10.1137/0909004
6. Hellwig, J., Platzer, A.: A real-analytic approach to differential-algebraic dynamic logic. In: Barrett, C.W., Waldmann, U. (eds.) Automated Deduction - CADE 30 - 30th International Conference on Automated Deduction, Stuttgart, Germany, July 28-31, 2025, Proceedings. Lecture Notes in Computer Science, vol. 15943, pp. 696–714. Springer (2025). https://doi.org/10.1007/978-3-031-99984-0_36
7. Hellwig, J., Qian, L., Platzer, A.: A deductive refinement calculus for differential-algebraic programs (2026). https://doi.org/10.48550/arXiv.2605.10188
8. Hubert, E.: Notes on triangular sets and triangulation decomposition algorithms II: differential systems. In: Winkler, F., Langer, U. (eds.) Symbolic and Numerical Scientific Computation, Second International Conference, SNSC 2001, Hagenberg, Austria, September 10-11, 2001, Revised Papers. Lecture Notes in Computer Science, vol. 2630, pp. 40–87. Springer (2001). https://doi.org/10.1007/3-540-45084-X_2

9. Kolchin, E.R.: Differential Algebra and Algebraic Groups, vol. 54. Academic press (1973)
10. Loos, S.M., Platzer, A.: Differential refinement logic. In: Grohe, M., Koskinen, E., Shankar, N. (eds.) Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016, pp. 505–514. ACM (2016). https://doi.org/10.1145/2933575.2934555
11. Pantelides, C.C.: The consistent initialization of differential-algebraic systems. SIAM J. Sci. Stat. Comput. **9**(2), 213–231 (1988). https://doi.org/10.1137/0909014
12. Platzer, A.: Differential dynamic logic for hybrid systems. J. Autom. Reason. **41**(2), 143–189 (2008). https://doi.org/10.1007/S10817-008-9103-8
13. Platzer, A.: Differential-algebraic dynamic logic for differential-algebraic programs. J. Log. Comput. **20**(1), 309–352 (2010). https://doi.org/10.1093/logcom/exn070
14. Platzer, A.: Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14509-4
15. Platzer, A.: The complete proof theory of hybrid systems. In: Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012, pp. 541–550. IEEE Computer Society (2012). https://doi.org/10.1109/LICS.2012.64
16. Platzer, A.: A complete uniform substitution calculus for differential dynamic logic. J. Autom. Reason. **59**(2), 219–265 (2017). https://doi.org/10.1007/S10817-016-9385-1
17. Platzer, A.: Logical Foundations of Cyber-Physical Systems. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-63588-0
18. Platzer, A., Qian, L.: Axiomatization of compact initial value problems: open properties. J. ACM **72**(6), 1–51 (2025). https://doi.org/10.1145/3763228
19. Platzer, A., Tan, Y.K.: Differential equation invariance axiomatization. J. ACM **67**(1), 6:1–6:66 (2020). https://doi.org/10.1145/3380825
20. Prebet, E., Platzer, A.: Uniform substitution for differential refinement logic. In: Benzmüller, C., Heule, M.J., Schmidt, R.A. (eds.) IJCAR. LNCS, vol. 14740, pp. 196–215. Springer (2024). https://doi.org/10.1007/978-3-031-63501-4_11
21. Pryce, J.: A simple structural analysis method for DAEs. BIT Numer. Math. **41**, 364–394 (2001). https://doi.org/10.1023/A:1021998624799
22. Ritt, J.F.: Differential algebra, vol. 33. American Mathematical Soc. (1950)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

