Algebra, coalgebra, and minimization in polynomial differential equations[⋆]

Michele Boreale
Università di Firenze

Abstract. We consider reasoning and minimization in systems of polynomial ordinary differential equations (ODEs). The ring of multivariate polynomials is employed as a syntax for denoting system behaviours. We endow polynomials with a transition system structure based on the concept of *Lie derivative*, thus inducing a notion of $\mathcal{L}$*-bisimulation*. Two states (variables) are proven $\mathcal{L}$-bisimilar if and only if they correspond to the same solution in the ODEs system. We then characterize $\mathcal{L}$-bisimilarity algebraically, in terms of certain ideals in the polynomial ring that are invariant under Lie-derivation. This characterization allows us to develop a complete algorithm, based on building an ascending chain of ideals, for computing the largest $\mathcal{L}$-bisimulation containing all valid identities that are instances of a user-specified template. A specific largest $\mathcal{L}$-bisimulation can be used to build a reduced system of ODEs, equivalent to the original one, but *minimal* among all those obtainable by linear aggregation of the original equations.

Keywords: Ordinary Differential Equations, Bisimulation, Minimization, Gröbner Bases.

1 Introduction

The past few years have witnessed a surge of interest in computational models based on ordinary differential equations (ODEs), ranging from continuous-time Markov chains (e.g. [3]), to process description languages oriented to bio-chemical systems (e.g. [33,4,10]), to deterministic approximations of stochastic systems (e.g. [15,32]), and to hybrid systems (e.g. [30,28,23]).

From a computational point of view, our motivations to study ODEs arises from the following problems.

1. *Reasoning*: provide methods to automatically prove and discover identities involving the system variables.

[⋆] Author's address: Michele Boreale, Università di Firenze, Dipartimento di Statistica, Informatica, Applicazioni (DiSIA) "G. Parenti", Viale Morgagni 65, I-50134 Firenze, Italy. E-mail: `michele.boreale@unifi.it`.

2. *Reduction*: provide methods to automatically reduce, and possibly minimize, the number of variables and equations of a system, in such a way that the reduced system retains all the relevant information of the original one.

Reasoning may help an expert (a chemist, a biologist, an engineer) to prove or to disprove certain system properties, even before actually solving, simulating or realizing the system. Often, the identities of interest take the form of *conservation laws*. For instance, chemical reactions often enjoy a mass conservation law, stating that the sum of the concentrations of two or more chemical species, is a constant. More generally, one would like tools to automatically *discover* all laws of a given form. Pragmatically, before actually solving or simulating a given system, it can be critical being able to reduce the system to a size that can be handled by a solver or a simulator.

Our goal is showing that these issues can be dealt with by a mix of algebraic and coalgebraic techniques. We will consider *initial value* problems, specified by a system of ODEs of the form $\dot{x}_i = f_i(x_1, ..., x_N)$, for $i = 1, ..., N$, plus initial conditions. The functions f_is are called *drifts*; here we will focus on the case where the drifts are multivariate polynomials in the variables $x_1, .., x_N$. Practically, the majority of functions found in applications is in, or can be encoded into this format (possibly under restrictions on the initial conditions), including exponential, trigonometric, logarithmic and rational functions.

A more detailed account of our work follows. We introduce the ring of multivariate polynomials as a syntax for denoting the *behaviours* induced by the given initial value problem (Section 2). In other words, a behaviour is any polynomial combination of the individual components $x_i(t)$ $(i = 1, .., N)$ of the (unique) system solution. We then endow the polynomial ring with a transition system, based on a purely syntactic notion of *Lie derivative* (Section 3). This structure naturally induces a notion of bisimulation over polynomials, $\mathcal{L}$-*bisimilarity*, that is in agreement with the underlying ODE' s. In particular, any two variables x_i and x_j are $\mathcal{L}$-bisimilar if and only the corresponding solutions are the same, $x_i(t) = x_j(t)$ (this generalizes to polynomial behaviours as expected). This way, one can prove identities between two behaviours, for instance conservation laws, by exhibiting bisimulations containing the given pair (in [8] we show how to enhance the resulting proof technique by introducing a polynomial version of the *up to* technique of [27]). In order to turn this method into a fully automated proof procedure, we first characterize $\mathcal{L}$-bisimulation algebraically, in terms of certain *ideals* in the polynomial ring that are invariant under Lie-derivation (Section 4). This characterization leads to an algorithm that, given a user-specified template, returns the set of all its instances that are valid identities in the system (Section 5). One may use this algorithm, for instance, to discover all the conservation laws of the system involving terms up to a given degree. The algorithm implies building an ascending chain of ideals until stabilization, and relies on a few basic concepts from Algebraic Geometry, notably Gröbner bases [17]. The output of the algorithm is in turn essential to build a reduced system of ODEs, equivalent to the original one, but featuring a *minimal* number of equations and variables, in the class of systems that can be obtained by linear aggregation from the original one (Section 6). We then illustrate the results of some simple experiments we have conducted using a prototype implementation (in Python) of our algorithms (Section 7). Our approach is mostly related to some recent work on equivalences for ODEs by Cardelli et al. [11] and to work in the area of hybrid systems. We discuss this and other related work, as well as some possible directions for future work, in the concluding section (Section 8). Due to space limitations, most proofs and examples are omitted from the present version; they can be found in a technical report available online [8].

2 Preliminaries

Let us fix an integer $N \geq 1$ and a set of N distinct variables $x_1, ..., x_N$. We will denote by $\mathbf{x}$ the column[1] vector $(x_1, ..., x_N)^T$. We let $\mathbb{R}[\mathbf{x}]$ denote the set of multivariate polynomials in the variables $x_1, ..., x_N$ with coefficients in $\mathbb{R}$, and let p, q range over it. Here we regard polynomials as syntactic objects. Given an integer $d \geq 0$, by $\mathbb{R}_d[\mathbf{x}]$ we denote the set of polynomials of degree $\leq d$. As an example, $p = 2xy^2 + (1/5)wz + yz + 1$ is a polynomial of degree $\deg(p) = 3$, that is $p \in \mathbb{R}_3[x, y, z, w]$, with monomials xy^2, wz, yz and 1. Depending on the context, with a slight abuse of notation it may be convenient to let a polynomial denote the induced function $\mathbb{R}^N \to \mathbb{R}$, defined as expected. In particular, x_i can be seen as denoting the projection on the i-th coordinate.

A (polynomial) *vector field* is a vector of N polynomials, $F = (f_1, ..., f_N)^T$, seen as a function $F : \mathbb{R}^N \to \mathbb{R}^N$. A vector field F and an initial condition $\mathbf{x}_0 \in \mathbb{R}^N$ together define an *initial value problem* $\mathbf{\Phi} = (F, \mathbf{x}_0)$, often written in the following form

$$\mathbf{\Phi} : \begin{cases} \dot{\mathbf{x}}(t) = F(\mathbf{x}(t)) \\ \mathbf{x}(0) = \mathbf{x}_0 \,. \end{cases} \tag{1}$$

The functions f_i in F are called *drifts* in this context. A *solution* to this problem is a differentiable function $\mathbf{x}(t) : D \to \mathbb{R}^N$, for some nonempty open interval $D \subseteq \mathbb{R}$ containing 0, which fulfills the above two equations, that is: $\frac{d}{dt}\mathbf{x}(t) = F(\mathbf{x}(t))$ for each $t \in D$ and $\mathbf{x}(0) = \mathbf{x}_0$. By the Picard-Lindelöf theorem [2], there exists a nonempty open interval D containing 0, over which there is a *unique* solution, say $\mathbf{x}(t) = (x_1(t), ..., x_N(t))^T$, to the problem. In our case, as F is infinitely often differentiable, the solution is seen to be *analytic* in D: each $x_i(t)$ admits a Taylor series expansion in a neighborhood of 0. For definiteness, we will take the domain of definition D of $\mathbf{x}(t)$ to be the largest symmetric open interval where the Taylor expansion from 0 of each of the $x_i(t)$ converges (possibly $D = \mathbb{R}$). The resulting vector function of t, $\mathbf{x}(t)$, is called the *time trajectory* of the system.

[1] *Vector* means column vector, unless otherwise specified.

Given a differentiable function $g : E \to \mathbb{R}$, for some open set $E \subseteq \mathbb{R}^N$, the *Lie derivative of g along F* is the function $E \to \mathbb{R}$ defined as

$$\mathcal{L}_F(g) \overset{\triangle}{=} \langle \nabla g, F \rangle = \sum_{i=1}^{N} (\frac{\partial g}{\partial x_i} \cdot f_i)\,.$$

The Lie derivative of the sum $h+g$ and product $h \cdot g$ functions obey the familiar rules

$$\mathcal{L}_F(h+g) = \mathcal{L}_F(h) + \mathcal{L}_F(g) \tag{2}$$

$$\mathcal{L}_F(h \cdot g) = h \cdot \mathcal{L}_F(g) + \mathcal{L}_F(h) \cdot g\,. \tag{3}$$

Note that $\mathcal{L}_F(x_i) = f_i$. Moreover if $p \in \mathbb{R}_d[\mathbf{x}]$ then $\mathcal{L}_F(p) \in \mathbb{R}_{d+d'}[\mathbf{x}]$, for some integer $d' \geq 0$ that depends on d and on F. This allows us to view the Lie derivative of polynomials along a polynomial field F as a purely syntactic mechanism, that is as a function $\mathcal{L}_F : \mathbb{R}[\mathbf{x}] \to \mathbb{R}[\mathbf{x}]$ that does not assume anything about the solution of (1). Informally, we can view p as a program, and taking Lie derivative of p can be interpreted as unfolding the definitions of the variables x_i's, according to the equations in (1) and to the formal rules for product and sum derivation, (2) and (3). We will pursue this view systematically in Section 3.

Example 1. Consider $N = 4$, $\mathbf{x} = (x, y, z, w)^T$ and the set of polynomials $\mathbb{R}[\mathbf{x}]$. The vector field $F = (xz + z, yw + z, z, w)^T$ and the initial condition $\mathbf{x}_0 = (0, 0, 1, 1)^T$ together define an initial value problem (with no particular physical meaning) $\mathbf{\Phi} = (F, \mathbf{x}_0)$. This problem can be equivalently written in the form

$$\begin{cases} \dot{x}(t) = x(t)z(t) + z(t) \\ \dot{y}(t) = y(t)w(t) + z(t) \\ \dot{z}(t) = z(t) \\ \dot{w}(t) = w(t) \\ \mathbf{x}(0) = \mathbf{x}_0 \ = \ (0,0,1,1)^T\,. \end{cases} \tag{4}$$

As an example of Lie derivative, if $p = 2xy^2 + wz$, we have $\mathcal{L}_F(p) = 4wxy^2 + 2wz + 2xy^2z + 4xyz + 2y^2z$.

The connection between time trajectories, polynomials and Lie derivatives can be summed up as follows. For any polynomial $p \in \mathbb{R}[\mathbf{x}]$, the function $p(\mathbf{x}(t)) : D \to \mathbb{R}$, obtained by composing p as a function with the time trajectory $\mathbf{x}(t)$, is analytic: we let $p(t)$ denote the extension of this function over the largest symmetric open interval of convergence (possibly coinciding with $\mathbb{R}$) of its Taylor expansion from 0. We will call $p(t)$ the *polynomial behaviour induced by p and by* the initial value problem (1). The connection between Lie derivatives of p along F and the initial value problem (1) is given by the following equations, which can be readily checked. Here and in the sequel, we let $p(\mathbf{x}_0)$ denote the real number obtained by evaluating p at $\mathbf{x}_0$.

$$p(t)_{|t=0} = p(\mathbf{x}_0) \tag{5}$$

$$\frac{d}{dt}p(t) = (\mathcal{L}_F(p))(t)\,. \tag{6}$$

More generally, defining inductively $\mathcal{L}_F^{(0)}(p) \stackrel{\triangle}{=} p$ and $\mathcal{L}_F^{(j+1)}(p) \stackrel{\triangle}{=} \mathcal{L}_F(\mathcal{L}_F^{j}(p))$, we have the following equation for the j-th derivative of $p(t)$ $(j = 0, 1, ...)$

$$\frac{d^j}{dt^j} p(t) = (\mathcal{L}_F^{(j)}(p))(t)\,. \tag{7}$$

In the sequel, we shall often abbreviate $\mathcal{L}_F^{(j)}(p)$ as $p^{(j)}$, and shall omit the subscript $_F$ from $\mathcal{L}_F$ when clear from the context.

3 Coalgebraic semantics of polynomial ODEs

In this section we show how to endow the polynomial ring with a transition relation structure, hence giving rise to coalgebra. Bisimilarity in this coalgebra will correspond to equality between polynomial behaviours.

We recall that a *(Moore) coalgebra* (see e.g. [26]) with outputs in a set O is a triple $C = (S, \delta, o)$ where S is a set of *states*, $\delta : S \to S$ is a *transition* function, and $o : S \to O$ is an *output* function. A *bisimulation* in C is a binary relation $R \subseteq S \times S$ such that whenever $s\,R\,t$ then: (a) $o(s) = o(t)$, and (b) $\delta(s)\,R\,\delta(t)$. It is an (easy) consequence of the general theory of bisimulation that a largest bisimulation over S, called bisimilarity and denoted by $\sim$, exists, is the union of all bisimulation relations, and is an equivalence relation over S.

Given an initial value problem $\mathbf{\Phi} = (F, \mathbf{x}_0)$ of the form (1), the triple

$$C_{\mathbf{\Phi}} \stackrel{\triangle}{=} (\mathbb{R}[\mathbf{x}], \mathcal{L}_F, o)$$

forms a coalgebra with outputs in $\mathbb{R}$, where: (1) $\mathbb{R}[\mathbf{x}]$ is the set of states; (2) $\mathcal{L}_F$ acts as the transition function; and (3) o defined as $o(p) \stackrel{\triangle}{=} p(\mathbf{x}_0)$ is the output function. Note that this definition of coalgebra is merely syntactic, and does not presuppose anything about the solution of the given initial value problem. When the standard definition of bisimulation over coalgebras is instantiated to $C_{\mathbf{\Phi}}$, it yields the following.

Definition 1 ($\mathcal{L}$-bisimulation $\sim_{\mathbf{\Phi}}$). *Let $\mathbf{\Phi}$ be an initial value problem. A binary relation $R \subseteq \mathbb{R}[\mathbf{x}] \times \mathbb{R}[\mathbf{x}]$ is a* $\mathcal{L}$-bisimulation *if, whenever $p\,R\,q$ then: (a) $p(\mathbf{x}_0) = q(\mathbf{x}_0)$, and (b) $\mathcal{L}(p)\,R\,\mathcal{L}(q)$. The largest $\mathcal{L}$-bisimulation over $\mathbb{R}[\mathbf{x}]$ is denoted by $\sim_{\mathbf{\Phi}}$.*

We now introduce a new coalgebra with outputs in $\mathbb{R}$. Let $\mathcal{A}$ denote the family of real valued functions f such that f is analytic at 0 and f's domain of definition coincides with the open interval of convergence of its Taylor series (nonempty, centered at 0, possibly coinciding with $\mathbb{R}$)[2]. We define the coalgebra of analytic functions as

$$C_{\text{an}} \stackrel{\triangle}{=} (\mathcal{A}, (\cdot)', o_{\text{an}})$$

[2] Equivalently, $\mathcal{A}$ is the set of power series $f(t) = \sum_{j \geq 0} a_j t^j$ with a positive radius of convergence.

where $(f)' = \frac{df}{dt}$ is the standard derivative, and $o_{\mathrm{fin}}(f) \overset{\triangle}{=} f(0)$ is the output function. We recall that a *morphism* μ between two coalgebras with outputs in the same set, $\mu : C_1 \to C_2$, is a function from states to states that preserves transitions $(\mu(\delta_1(s)) = \delta_2(\mu(s)))$ and outputs $(o_1(s) = o_2(\mu(s)))$. It is a standard (and easy) result of coalgebra that a morphism maps bisimilar states into bisimilar states: $s \sim_1 s'$ implies $\mu(s) \sim_2 \mu(s')$.

The coalgebra C_{an} has a special status, in that, given any coalgebra C with outputs in $\mathbb{R}$, *if* there is a morphism from C to C_{an}, this morphism is guaranteed to be *unique*[3]. For our purposes, it is enough to focus on $C = C_{\mathbf{\Phi}}$. We define the function $\mu : \mathbb{R}[\mathbf{x}] \to \mathcal{A}$ as

$$\mu(p) \overset{\triangle}{=} p(t)\,.$$

Theorem 1 (coinduction). *μ is the unique morphism from $C_{\mathbf{\Phi}}$ to C_{an}. Moreover, the following* coinduction *principle is valid: $p \sim_{\mathbf{\Phi}} q$ in $C_{\mathbf{\Phi}}$ if and only if $p(t) = q(t)$ in $\mathcal{A}$.*

Theorem 1 permits proving polynomial relations among components $x_i(t)$ of $\mathbf{x}(t)$, say that $p(t) = q(t)$, by coinduction, that is, by exhibiting a suitable $\mathcal{L}$-bisimulation relating the polynomials p and q.

Example 2. For $N = 2$, consider the vector field $F = (x_2, -x_1)^T$ with the initial value $\mathbf{x}_0 = (0, 1)^T$. The binary relation $R \subseteq \mathbb{R}[x_1, x_2] \times \mathbb{R}[x_1, x_2]$ defined thus

$$R = \{\, (0,0),\ (x_1^2 + x_2^2, 1)\,\}$$

is easily checked to be an $\mathcal{L}$-bisimulation. Thus we have proved the polynomial relation $x_1^2(t) + x_2^2(t) = 1$. Note that the unique solution to the given initial value problem is the pair of functions $\mathbf{x}(t) = (\sin(t), \cos(t))^T$. This way we have proven the familiar trigonometric identity $\sin(t)^2 + \cos(t)^2 = 1$.

This proof method can be greatly enhanced by a so called *$\mathcal{L}$-bisimulation up to* technique, in the spirit of [27]. See [8].

4 Algebraic characterization of $\mathcal{L}$-bisimilarity

We first review the notion of polynomial ideal from Algebraic Geometry, referring the reader to e.g. [17] for a comprehensive treatment. A set of polynomials $I \subseteq \mathbb{R}[\mathbf{x}]$ is an *ideal* if: (1) $0 \in I$, (2) I is closed under sum $+$, (3) I is absorbing under product $\cdot$, that is $p \in I$ implies $h \cdot p \in I$ for each $h \in \mathbb{R}[\mathbf{x}]$. Given a set of polynomials S, the ideal generated by S, denoted by $\langle\ S\ \rangle$, is defined as

$$\left\{ \sum_{j=1}^{m} h_j p_j \ :\ m \geq 0,\ h_j \in \mathbb{R}[\mathbf{x}] \text{ and } p_j \in S, \text{ for } j = 1, ..., m \right\}. \tag{8}$$

[3] Existence of a morphism is not guaranteed, though. In this sense, C_{an} is not *final*.

The polynomial coefficients h_j in the above definition are called *multipliers*. It can be proven that $\langle\, S \,\rangle$ is the smallest ideal containing S, which implies that $\langle\, \langle\, S \,\rangle \,\rangle = \langle\, S \,\rangle$. Any set S such that $\langle\, S \,\rangle = I$ is called a *basis* of I. Every ideal in the polynomial ring $\mathbb{R}[\mathbf{x}]$ is finitely generated, that is has a finite basis (an instance of Hilbert's basis theorem).

$\mathcal{L}$-bisimulations can be connected to certain types of ideals. This connection relies on Lie derivatives. First, we define the Lie derivative of any set $S \subseteq \mathbb{R}[\mathbf{x}]$ as follows

$$\mathcal{L}(S) \stackrel{\triangle}{=} \{\mathcal{L}(p) : p \in S\}\,.$$

We say that S is a *pre-fixpoint* of $\mathcal{L}$ if $\mathcal{L}(S) \subseteq S$. $\mathcal{L}$-bisimulations can be characterized as particular pre-fixpoints of $\mathcal{L}$ that are also ideals, called *invariants*.

Definition 2 (invariant ideals). *Let $\mathbf{\Phi} = (F, \mathbf{x}_0)$. An ideal I is a $\mathbf{\Phi}$-*invariant *if: (a) $p(\mathbf{x}_0) = 0$ for each $p \in I$, and (b) I is a pre-fixpoint of $\mathcal{L}_F$.*

We will drop the $\mathbf{\Phi}$- from $\mathbf{\Phi}$-*invariant* whenever this is clear from the context. The following definition and lemma provide the link between invariants and $\mathcal{L}$-bisimulation.

Definition 3 (kernel). *The* kernel *of a binary relation R is* $\ker(R) \stackrel{\triangle}{=} \{p - q : p\,R\,q\}$.

Lemma 1. *Let R be a binary relation. If R is an $\mathcal{L}$-bisimulation then $\langle\, \ker(R) \,\rangle$ is an invariant. Conversely, given an invariant I, then $R = \{(p, q) : p - q \in I\}$ is an $\mathcal{L}$-bisimulation.*

Consequently, proving that $p \sim_{\mathbf{\Phi}} q$ is equivalent to exhibiting an invariant I such that $p - q \in I$. A more general problem than equivalence checking is finding *all* valid polynomial equations of a given form. We will illustrate an algorithm to this purpose in the next section.

The following result sums up the different characterization of $\mathcal{L}$-bisimilarity $\sim_{\mathbf{\Phi}}$. In what follows, we will denote the constant zero function in $\mathcal{A}$ simply by 0 and consider the following set of polynomials.

$$\mathcal{Z}_{\mathbf{\Phi}} \stackrel{\triangle}{=} \{p : p(t) \text{ is identically } 0\,\}\,.$$

The following result also proves that $\mathcal{Z}_{\mathbf{\Phi}}$ is the largest $\mathbf{\Phi}$-invariant.

Theorem 2 ($\mathcal{L}$-bisimilarity via ideals). *For any pair of polynomials p and q: $p \sim_{\mathbf{\Phi}} q$ iff $p - q \in \ker(\sim_{\mathbf{\Phi}}) = \mathcal{Z}_{\mathbf{\Phi}} \stackrel{(1)}{=} \left\{p : p^{(j)}(\mathbf{x}_0) = 0 \text{ for each } j \geq 0\right\} = \bigcup\{I : I \text{ is a } \mathbf{\Phi}\text{-invariant}\,\}$.*

5 Computing invariants

By Theorem 2, proving $p \sim_{\mathbf{\Phi}} q$ means finding an invariant I such that $p - q \in I \subseteq \mathcal{Z}_{\mathbf{\Phi}}$. More generally, we focus here on the problem of finding invariants that include a user-specified set of polynomials. In the sequel, we will make use of the following two basic facts about ideals, for whose proof we refer the reader to [17].

1. Any infinite ascending chain of ideals in a polynomial ring, $I_0 \subseteq I_1 \subseteq \cdots$, stabilizes at some finite k. That is, there is $k \geq 0$ such that $I_k = I_{k+j}$ for each $j \geq 0$.
2. The *ideal membership problem*, that is, deciding whether $p \in I$, given p and a finite set of S of *generators* (such that $I = \langle\ S\ \rangle$), is decidable (provided the coefficients used in p and in S can be finitely represented). The ideal membership will be further discussed later on in the section.

The main idea is introduced by the naive algorithm presented below.

A naive algorithm Suppose we want to decide whether $p \in \mathcal{Z}_{\mathbf{\Phi}}$. It is quite easy to devise an algorithm that computes the smallest invariant containing p, or returns 'no' in case no such invariant exists, i.e. in case $p \notin \mathcal{Z}_{\mathbf{\Phi}}$. Consider the successive Lie derivatives of p, $p^{(j)} = \mathcal{L}^{(j)}(p)$ for $j = 0, 1, ...$ For each $j \geq 0$, let $I_j \stackrel{\triangle}{=} \langle\ \{p^{(0)}, ..., p^{(j)}\}\ \rangle$. Let m be the least integer such that either

$$\text{(a) } p^{(m)}(\mathbf{x}_0) \neq 0, \qquad \text{or} \qquad \text{(b) } I_m = I_{m+1}.$$

If (a) occurs, then $p \notin \mathcal{Z}_{\mathbf{\Phi}}$, so we return 'no' (Theorem 2(1)); if (b) occurs, then I_m is the least invariant containing p. Note that the integer m is well defined: $I_0 \subseteq I_1 \subseteq I_2 \subseteq \cdots$ forms an infinite ascending chain of ideals, which must stabilize in a finite number of steps (fact 1 at the beginning of the section).

Checking condition (b) amounts to deciding if $p^{(m+1)} \in I_m$. This is an instance of the ideal membership problem, which can be solved effectively. Generally speaking, given a polynomial p and finite set of polynomials S, deciding the ideal membership $p \in I = \langle\ S\ \rangle$ can be accomplished by first transforming S into a *Gröbner basis* G for I (via, e.g. the Buchberger's algorithm), then computing r, the *residual* of p modulo G (via a sort generalised division of p by G): one has that $p \in I$ if and only if $r = 0$ (again, this procedure can be carried out effectively only if the coefficients involved in p and S are finitely representable; in practice, one often confines to rational coefficients). We refer the reader to [17] for further details on the ideal membership problem. Known procedures to compute Gröbner bases have exponential worst-case time complexity, although may perform reasonably well in some concrete cases. One should in any case invoke such procedures parsimoniously.

Let us now introduce a more general version of the naive algorithm, which will be also able to deal with (infinite) *sets* of user-specified polynomials. First, we need to introduce the concept of template.

Templates Polynomial templates have been introduced by Sankaranarayanan, Sipma and Manna in [28] as a means to compactly specify sets of polynomials. Fix a tuple of $n \geq 1$ of distinct *parameters*, say $\mathbf{a} = (a_1, ..., a_n)$, disjoint from $\mathbf{x}$. Let $Lin(\mathbf{a})$, ranged over by ℓ, be the set of *linear expressions* with coefficients in $\mathbb{R}$ and variables in $\mathbf{a}$; e.g. $\ell = 5a_1 + 42a_2 - 3a_3$ is one such expression[4]. A *template* is a polynomial in $Lin(\mathbf{a})[\mathbf{x}]$, that is, a polynomial with linear expressions as coefficients; we let π range over templates. For example, the following is a template: $\pi = (5a_1 + (3/4)a_3)xy^2 + (7a_1 + (1/5)a_2)xz + (a_2 + 42a_3)$. Given a vector $v = (r_1, ..., r_n)^T \in \mathbb{R}^n$, we will let $\ell[v] \in \mathbb{R}$ denote the result of replacing each parameter a_i with r_i, and evaluating the resulting expression; we will let $\pi[v] \in \mathbb{R}[\mathbf{x}]$ denote the polynomial obtained by replacing each ℓ with $\ell[v]$ in π. Given a set $S \subseteq \mathbb{R}^n$, we let $\pi[S]$ denote the set $\{\pi[v] : v \in S\} \subseteq \mathbb{R}[\mathbf{x}]$.

The (formal) Lie derivative of π is defined as expected, once linear expressions are treated as constants; note that $\mathcal{L}(\pi)$ is still a template. It is easy to see that the following property is true: for each π and v, one has $\mathcal{L}(\pi[v]) = \mathcal{L}(\pi)[v]$. This property extends as expected to the j-th Lie derivative ($j \geq 0$)

$$\mathcal{L}^{(j)}(\pi[v]) = \mathcal{L}^{(j)}(\pi)[v]\,. \tag{9}$$

A double chain algorithm We present an algorithm that, given a template π with n parameters, returns a pair (V, J), where $V \subseteq \mathbb{R}^n$ is such that $\pi[\mathbb{R}^n] \cap \mathcal{Z}_{\mathbf{\Phi}} = \pi[V]$, and J is the smallest invariant that includes $\pi[V]$, possibly $J = \{0\}$. We first give a purely mathematical description of the algorithm, postponing its effective representation to the next subsection. The algorithm is based on building two chains of sets, a descending chain of vector spaces and an (eventually) ascending chain of ideals. The ideal chain is used to detect the stabilization of the sequence. For each $i \geq 0$, consider the sets

$$V_i \stackrel{\triangle}{=} \{v \in \mathbb{R}^n \,:\, \pi^{(j)}[v](\mathbf{x}_0) = 0 \text{ for } j = 0, ..., i\,\} \tag{10}$$

$$J_i \stackrel{\triangle}{=} \langle\ \bigcup_{j=1}^{i} \pi^{(j)}[V_i]\ \rangle\,. \tag{11}$$

It is easy to check that each $V_i \subseteq \mathbb{R}^n$ is a vector space over $\mathbb{R}$ of dimension $\leq n$. Now let $m \geq 0$ be the least integer such that the following conditions are *both* true:

$$V_{m+1} = V_m \tag{12}$$

$$J_{m+1} = J_m\,. \tag{13}$$

The algorithm returns (V_m, J_m). Note that the integer m is well defined: indeed, $V_0 \supseteq V_1 \supseteq \cdots$ forms an infinite descending chain of finite-dimensional vector spaces, which must stabilize in finitely many steps. In other words, we can consider the least m' such that $V_{m'} = V_{m'+k}$ for each $k \geq 1$. Then $J_{m'} \subseteq J_{m'+1} \subseteq \cdots$

[4] Differently from Sankaranarayanan et al. we do not allow linear expressions with a constant term, such as $2 + 5a_1 + 42a_2 - 3a_3$. This minor syntactic restriction does not practically affect the expressiveness of the resulting polynomial templates.

forms an infinite ascending chain of ideals, which must stabilize at some $m \geq m'$. Therefore there must be some index m such that (12) and (13) are both satisfied, and we choose the least such m.

The next theorem states the correctness and relative completeness of this abstract algorithm. Informally, the algorithm will output the largest space V_m such that $\pi[V_m] \subseteq \mathcal{Z}_{\mathbf{\Phi}}$ and the smallest invariant J_m witnessing this inclusion. Note that, while typically the user will be interested in $\pi[V_m]$, J_m as well may contain useful information, such as higher order, nonlinear conservation laws. We need a technical lemma.

Lemma 2. *Let V_m, J_m be the sets returned by the algorithm. Then for each $j \geq 1$, one has $V_m = V_{m+j}$ and $J_m = J_{m+j}$.*

Theorem 3 (correctness and relative completeness). *Let V_m, J_m be the sets returned by the algorithm for a polynomial template π.*

(a) $\pi[V_m] = \mathcal{Z}_{\mathbf{\Phi}} \cap \pi[\mathbb{R}^n]$;
(b) J_m is the smallest invariant containing $\pi[V_m]$.

Proof. Concerning part (a), we first note that $\pi[v] \in \mathcal{Z}_{\mathbf{\Phi}} \cap \pi[\mathbb{R}^n]$ means $(\pi[v])^{(j)}(\mathbf{x}_0) = \pi^{(j)}[v](\mathbf{x}_0) = 0$ for each $j \geq 0$ (Theorem 2(1)), which, by definition, implies $v \in V_j$ for each $j \geq 0$, hence $v \in V_m$. Conversely, if $v \in V_m = V_{m+1} = V_{m+2} = \cdots$ (here we are using Lemma 2), then by definition $(\pi[v])^{(j)}(\mathbf{x}_0) = \pi^{(j)}[v](\mathbf{x}_0) = 0$ for each $j \geq 0$, which implies that $\pi[v] \in \mathcal{Z}_{\mathbf{\Phi}}$ (again Theorem 2(1)). Note that in proving both inclusions we have used property (9).

Concerning part (b), it is enough to prove that: (1) J_m is an invariant, (2) $J_m \supseteq \mathcal{Z}_{\mathbf{\Phi}} \cap \pi[\mathbb{R}^n]$, and (3) for any invariant I such that $\mathcal{Z}_{\mathbf{\Phi}} \cap \pi[\mathbb{R}^n] \subseteq I$, we have that $J_m \subseteq I$. We first prove (1), that J_m is an invariant. Indeed, for each $v \in V_m$ and each $j = 0, ..., m-1$, we have $\mathcal{L}(\pi^{(j)}[v]) = \pi^{(j+1)}[v] \in J_m$ by definition, while for $j = m$, since $v \in V_m = V_{m+1}$, we have $\mathcal{L}(\pi^{(m)}[v]) = \pi^{(m+1)}[v] \in J_{m+1} = J_m$ (note that in both cases we have used property (9)). Concerning (2), note that $J_m \supseteq \pi[V_m] = \mathcal{Z}_{\mathbf{\Phi}} \cap \pi[\mathbb{R}^n]$ by virtue of part (a). Concerning (3), consider any invariant $I \supseteq \mathcal{Z}_{\mathbf{\Phi}} \cap \pi[\mathbb{R}^n]$. We show by induction on $j = 0, 1, ...$ that for each $v \in V_m$, $\pi^{(j)}[v] \in I$; this will imply the wanted statement. Indeed, $\pi^{(0)}[v] = \pi[v] \in \mathcal{Z}_{\mathbf{\Phi}} \cap \pi[\mathbb{R}^n]$, as $\pi[V_m] \subseteq \mathcal{Z}_{\mathbf{\Phi}}$ by (a). Assuming now that $\pi^{(j)}[v] \in I$, by invariance of I we have $\pi^{(j+1)}[v] = \mathcal{L}(\pi^{(j)}[v]) \in I$ (again, we have used here property (9)).

According to Theorem 3(a), given a template π and $v \in \mathbb{R}^n$, checking if $\pi[v] \in \pi[V_m]$ is equivalent to checking if $v \in V_m$, which can be effectively done knowing a basis B_m of V_m. We show how to effectively compute such a basis in the following.

Effective representation For $i = 0, 1, ...,$ we have to give effective ways to: (i) represent the sets V_i, J_i in (10) and (11); and, (ii) check the termination conditions (12) and (13). It is quite easy to address (i) and (ii) in the case of the vector spaces V_i. For each i, consider the linear expression $\pi^{(i)}(\mathbf{x}_0)$. By factoring out the parameters $a_1, ..., a_n$ in this expression, we can write, for a suitable (row) vector of coefficients $t_i = (t_{i1}, ..., t_{in}) \in \mathbb{R}^{1\times n}$: $\pi^{(i)}(\mathbf{x}_0) = t_{i1} \cdot a_1 + \cdots + t_{in} \cdot a_n$ The condition on $v \in \mathbb{R}^n$, $\pi^{(i)}[v](\mathbf{x}_0) = 0$, can then be translated into the linear constraint on v

$$t_i \cdot v = 0\,. \tag{14}$$

Letting $T_i \in \mathbb{R}^{i\times n}$ denote the matrix obtained by stacking the rows $t_1, ..., t_i$ on above the other, we see that V_i is the right null space of T_i. That is (here, $\mathbf{0}_i$ denotes the null vector in $\mathbb{R}^i$): $V_i = \{v \in \mathbb{R}^n \,:\, T_i v = \mathbf{0}_i\,\}$. Checking whether $V_i = V_{i+1}$ or not amounts then to checking whether the vector t_{i+1} is or not linearly dependent from the rows in T_i, which can be accomplished by standard and efficient linear algebraic techniques. In practice, the linear constraints (14) can be resolved and propagated via parameter elimination[5] incrementally, as soon as they are generated following computation of the derivatives $\pi^{(i)}$. Concerning the representation of the ideals J_i, we will use the following lemma[6].

Lemma 3. *Let $V \subseteq \mathbb{R}^n$ be a vector space with B as a basis, and $\pi_1, ..., \pi_k$ be templates. Then $\langle\ \cup_{j=1}^k \pi_j[V]\ \rangle = \langle\ \cup_{j=1}^k \pi_j[B]\ \rangle$.*

Now let B_i be a finite basis of V_i, which can be easily built from the matrix T_i. By the previous lemma, $\cup_{j=1}^i \pi^{(j)}[B_i]$ is a *finite* set of generators for J_i: this solves the representation problem. Concerning the termination condition, we note that, after checking that actually $V_i = V_{i+1}$, checking $J_i = J_{i+1}$ reduces to checking that $\pi^{(i+1)}[B_i] \subseteq \langle\ \cup_{j=1}^i \pi^{(j)}[B_i]\ \rangle\ =\ J_i\ \ (*)$.

To check this inclusion, one can apply standard computer algebra techniques. For example, one can check if $\pi^{(i+1)}[b] \in J_i$ for each $b \in B_i$, thus solving $|B_i|$ ideal membership problems, for one and the same ideal J_i. As already discussed, this presupposes the computation of a Gröbner basis for J_i, a potentially expensive operation. One advantage of the above algorithm, over methods proposed in program analysis with termination conditions based on testing ideal membership (e.g. [25]), is that $(*)$ is not checked at every iteration, but only when $V_{i+1} = V_i$ (the latter a relatively inexpensive check).

Example 3. Consider the initial value problem of Example 1 and the template $\pi = a_1x + a_2y + a_3z + a_4w$. We run the double chain algorithm with this system and template as inputs. In what follows, $v = (v_1, v_2, v_3, v_4)^T$ will denote a generic vector in $\mathbb{R}^4$. Recall that $\mathbf{x} = (x, y, z, w)^T$ and $\mathbf{x}_0 = (0, 0, 1, 1)^T$.

[5] E.g., if for $\pi = a_1x + a_2y + a_3x + a_4w$ and $\mathbf{x}_0 = (0, 0, 1, 1)^T$, $\pi[v](\mathbf{x}_0) = 0$ is resolved by the substitution $[a_3 \mapsto -a_4]$.

[6] The restriction that linear expressions in templates do not contain constant terms is crucial here.

– For each $v \in \mathbb{R}^4$: $\pi^{(0)}[v](\mathbf{x}_0) = (v_1x + v_2y + v_3z + v_4w)(\mathbf{x}_0) = 0$ if and only if $v \in V_0 \triangleq \{v : v_3 = -v_4\}$.
– For each $v \in V_0$: $\pi^{(1)}[v](\mathbf{x}_0) = (v_1xz + v_1z + v_2wy + v_2z + v_4w - v_4z)(\mathbf{x}_0) = 0$ if and only if $v \in V_1 \triangleq \{v \in V_0 : v_1 = -v_2\}$.
– For each $v \in V_1$: $\pi^{(2)}[v](\mathbf{x}_0) = (v_2w^2y + v_2wy + v_2wz - v_2xz^2 - v_2xz - v_2z^2 + v_4w - v_4z)(\mathbf{x}_0) = 0$ if and only if $v \in V_2 \triangleq V_1$.
Being $V_2 = V_1$, we also check if $J_2 = J_1$. A basis of V_1 is $B_1 = \{b_1, b_2\}$ with $b_1 = (-1, 1, 0, 0)^T$ and $b_2 = (0, 0, -1, 1)^T$. According to $(*)$, we have therefore to check if, for $\ell = 1, 2$: $\pi^{(2)}[b_\ell] \in J_1 \triangleq \left\langle \{\pi^{(0)}[b_1], \pi^{(0)}[b_2], \pi^{(1)}[b_1], \pi^{(1)}[b_2]\} \right\rangle$. With the help of a computer algebra system, one computes a Gröbner basis for J_1 as $G_1 = \{x - y, z - w\}$. Then one can reduce $\pi^{(2)}[b_1] = w^2y + wy + wz - xz^2 - xz - z^2$ modulo G_1 and obtain $\pi^{(2)}[b_1] = h_1(x - y) + h_2(z - w)$, with $h_1 = -z^2 - z$ and $h_2 = -wy - yz - y - z$, thus proving that $\pi^{(2)}[b_1] \in J_1$. One proves similarly that $\pi^{(2)}[b_2] \in J_1$. This shows that $J_2 = J_1$.

Hence the algorithm terminates with $m = 1$ and returns (V_1, J_1), or, concretely, (B_1, G_1). In particular, $x - y \in \mathcal{Z}_{\mathbf{\Phi}}$, or equivalently $x(t) = y(t)$. Similarly for $z - w$.

Remark 1 (linear systems). When the system of ODEs is linear, that is when the drifts f_i are linear functions of the x_i's, stabilization of the chain of vector spaces can be detected without resorting to ideals. The resulting single chain algorithm essentially boils down to the 'refinement' algorithm of [6, Th.2]. See [8] for details.

6 Minimization

We present a method for reducing the size of the an initial value problem. The basic idea is projecting the original system onto a suitably chosen subspace of $\mathbb{R}^N$. Consider the subspace $W \subseteq \mathbb{R}^N$ of all vectors that are orthogonal to $\mathbf{x}(t)$ for each t in the domain of definition D of the time trajectory, that is $W = \{v \in \mathbb{R}^N : \langle v\,, \mathbf{x}(t)\rangle = 0 \text{ for each } t \in D\}$. It is not difficult to prove (see [8]) that $W = V_m^\perp$, where V_m is the subspace returned by the double chain algorithm when fed with the linear template $\pi = \sum_{i=1}^N a_ix_i$. Let B the $N \times l$ ($l \leq m+1, N$) matrix whose columns are the vectors of an orthonormal basis of W. Then, for each t, $B^T\mathbf{x}(t)$ are the coordinates of $\mathbf{x}(t)$ in the subspace W w.r.t. the chosen basis. Consider now the following (reduced) initial value problem $\mathbf{\Psi}$, in the new variables $\mathbf{y} = (y_1, ..., y_l)^T$, obtained by projecting the original problem $\mathbf{\Phi}$ onto W. In [8], we prove the following result. In essence, all information about $\mathbf{\Phi}$ can be recovered exactly from the reduced $\mathbf{\Psi}$, which is the best possible linear reduction of $\mathbf{\Phi}$.

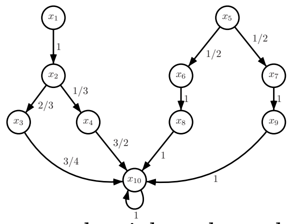
$$\mathbf{\Psi} : \begin{cases} \dot{\mathbf{y}}(t) = B^TF\left(B\mathbf{y}(t)\right) \\ \mathbf{y}(0) = B^T\mathbf{x}(0)\,. \end{cases} \tag{15}$$

Theorem 4 (minimal exact reduction). *Let $\mathbf{y}(t)$ be the unique analytic solution of the (reduced) problem* (15). *Then,* $\mathbf{x}(t) = B\mathbf{y}(t)$. *Moreover, suppose for* *some $N \times k$ matrix C and vector function $\mathbf{z}(t)$, we have $\mathbf{x}(t) = C\mathbf{z}(t)$, for each $t \in D$. Then $k \geq l$.*

7 Examples

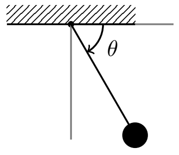
We have put a proof-of-concept implementation[7] of our algorithms at work on a few simple examples taken from the literature. We illustrate below two cases.

Example 1: linearity and weighted automata The purpose of this example is to argue that, when the transition structure induced by the Lie-derivatives is considered, $\mathcal{L}$-bisimilarity is fundamentally a linear-time semantics. We first introduce *weighted automata*, a more compact[8] way of representing the transition structure of the coalgebra $C_{\mathbf{\Phi}}$.



A (finite or infinite) weighted automaton is like an ordinary automaton, save that both states and transitions are equipped with *weights* from $\mathbb{R}$. Given F and $\mathbf{x}_0$, we can build a weighted automaton with monomials as states, weighted transitions given by the rule $\alpha \xrightarrow{\lambda} \beta$ iff $\mathcal{L}_F(\alpha) = \lambda\beta + q$ for some polynomial q and real $\lambda \neq 0$, and where each state α is assigned weight $\alpha(\mathbf{x}_0)$. As an example, consider the weighted automaton on the right, where the state weights (not displayed) are 1 for x_{10}, and 0 for any other state. This automaton is generated (and in fact codes up) a system of ODEs with ten variables, where $\dot{x}_1 = x_2$, $\dot{x}_2 = (2/3)x_3 + (1/3)x_4$ etc., with the initial condition as specified by the state weights ($x_1(0) = 0$ etc.). The standard linear-time semantics of weighted automata (see [26,5] and references therein) is in full agreement with $\mathcal{L}$-bisimilarity [8]. As a consequence, in our example we have for instance that $x_1(t) = x_5(t)$. In fact, when invoked with this system and $\pi = \sum_{i=1}^{10} a_i x_i$ as inputs, the double chain algorithm terminates at $m = 2$ (in about 0.3 s; being this a linear system, Gröbner bases are never actually needed), returning $\pi[V_2] = \big(a_1(x_6 - x_7) + a_2(x_8 - x_9) + a_3(x_6 - x_2) + a_4(x_5 - x_1) + a_5(\frac{3}{2}x_8 - x_4) + a_6(\frac{3}{4}x_8 - x_3)\big)[\mathbb{R}^6]$. This implies the expected $x_1 = x_5$, as well as other equivalences, and a 60% reduction in the minimal system.

Example 2: nonlinear conservation laws The law of the simple pendulum is $\frac{d^2}{dt^2}\theta = \frac{g}{\ell}\cos\theta$, where θ is the angle from the roof to the rod measured clockwise, ℓ is the length of the rod and g is gravity acceleration (see picture on the right). If we assume the initial condition $\theta(0) = 0$, this can be translated



[7] Python code available at `http://local.disia.unifi.it/boreale/papers/DoubleChain.py`. Reported execution times relative to the pypy interpreter under Windows 8 on a core i5 machine.

[8] At least for linear vector fields, the resulting weighted automaton is finite.

into the polynomial initial value problem below, where $\mathbf{x} = (\theta, \omega, x, y)^T$. The meaning of the variables is $\omega = \dot{\theta}$, $x = \cos\theta$ and $y = \sin\theta$. We assume for simplicity $\ell = 1$ and $g = 9$.

$$\begin{cases} \dot{\theta} = \omega \\ \dot{\omega} = \frac{g}{\ell} x \\ \dot{x} = -y\omega \\ \dot{y} = x\omega \\ \mathbf{x}(0) = (0, 0, \ell, 0)^T . \end{cases}$$

For this system, the double chain algorithm reports that there is no nontrivial linear conservation law (after $m = 6$ iterations and about 0.3 s). We then ask the algorithm to find all the conservation laws of order two, that is we use the template (α ranges over monomials) $\pi = \sum_{\alpha_i \,:\, \deg(\alpha_i) \leq 2} a_i \alpha_i$ as input. The algorithm terminates after $m = 16$ iterations (in about 7 s). The invariant J_{16} contains all the wanted conservation laws. The returned Gröbner basis for it is $G = \{x^2 + y^2 - 1, \omega^2 - 18y\}$. The first term here just expresses the trigonometric identity $(\cos\theta)^2 + (\sin\theta)^2 = 1$. Recalling that the (tangential) speed of the bob is $v = \ell\dot{\theta} = \ell\omega$, and that its vertical distance from the roof is $h = \ell \sin\theta = \ell y$, we see that the second term, considering our numerical values for ℓ, g, is equivalent to the equation $\frac{1}{2}v^2 = gh$, which, when multiplied by the mass m of the bob, yields the law of conservation of energy $\frac{1}{2}mv^2 = mgh$ (acquired kinetic energy = lost potential energy).

8 Future and related work

We briefly outline future work and related work below, referring the reader to [8] for a more comprehensive discussion.

Directions for future work Scalability is an issue, as already for simple systems the Gröbner basis construction involved in the main algorithm can be computationally quite demanding. Further experimentation, relying on a well-engineered implementation of the method, and considering sizeable case studies, is called for in order to assess this aspect. Approximate reductions in the sense of System Theory [1] are also worth investigating.

Related work Bisimulations for weighted automata are related to our approach, because, as argued in subsection 7, Lie-derivation can be naturally represented by such an automaton. Algorithms for computing largest bisimulations on *finite* weighted automata have been studied by Boreale et al. [6,5]. A crucial ingredient in these algorithms is the representation of bisimulations as finite-dimensional vector spaces. Approximate version of this technique have also been recently considered in relation to Markov chains [7]. As discussed in Remark 1, in the case of linear systems, the algorithm in the present paper reduces to that of [6,5]. Algebraically, moving from linear to polynomial systems corresponds to moving from vector spaces to ideals, hence from linear bases to Gröbner bases. From the point of view automata, this step leads to considering infinite weighted automata. In this respect, the present work may be also be related to the automata-theoretic treatment of linear ODEs by Fliess and Reutenauer [18].

Although there exists a rich literature dealing with linear aggregation of systems of ODEs (e.g. [1,20,31,22]), we are not aware of fully automated approaches to minimization (Theorem 4), with the notable exception of a series of recent works by Cardelli and collaborators [11,12,13]. Mostly related to ours is [11]. There, for an extension of the polynomial ODE format called IDOL, the authors introduce two flavours of *differential equivalence*, called *Forward* (FDE) and *Backward* (BDE). They provide a symbolic, SMT-based partition refining algorithms to compute the largest equivalence of each type. While FDE is unrelated with our equivalence, BDE can be compared directly to our $\mathcal{L}$-bisimulation. An important difference is that BDE is stricter than necessary, as it may tell apart variables that have the same solution. This is not the case with $\mathcal{L}$-bisimilarity, which is, in this respect, correct and complete. An important consequence of this difference is that the quotient system produced by BDE is not minimal, whereas that produced by $\mathcal{L}$-bisimulation is in a precise sense. For example, BDE finds no reductions at all in the linear system[9] of Section 7, whereas $\mathcal{L}$-bisimulation, as seen, leads to a 60% reduction. Finally, the approach of [11] and ours rely on two quite different algorithmic decision techniques, SMT and Gröbner bases, both of which have exponential worst-case complexity. As shown by the experiments reported in [11], in practice BDE and FDE have proven quite effective at system reduction. At the moment, we lack similar experimental evidence for $\mathcal{L}$-bisimilarity.

The seminal paper of Sankaranarayanan, Sipma and Manna [28] introduced polynomial ideals to find invariants of hybrid systems. Indeed, the study of the safety of hybrid systems can be shown to reduce constructively to the problem of generating invariants for their differential equations [24]. The results in [28] have been subsequently refined and simplified by Sankaranarayanan using *pseudoideals* [29], which enable the discovery of polynomial invariants of a special form. Other authors have adapted this approach to the case of imperative programs, see e.g. [9,21,25] and references therein. Reduction and minimization seem to be not a concern in this field.

Still in the field of formal verification of hybrid systems, mostly related to ours is Ghorbal and Platzer's recent work on polynomial invariants [19]. One one hand, they characterize algebraically invariant regions of vector fields – as opposed to initial value problems, as we do. On the other hand, they offer sufficient conditions under which the trajectory induced by a specific initial value satisfies all instances of a polynomial template (cf. [19, Prop.3]). The latter result compares with ours, but the resulting method appears to be not (relatively) complete in the sense of our double chain algorithm. Moreover, the computational prerequisites of [19] (symbolic linear programming, exponential size matrices, symbolic root extraction) are very different from ours, and much more demanding. Again, minimization is not addressed.

Acknowledgments The author has benefited from stimulating discussions with Mirco Tribastone.

[9] Checked with the Erode tool by the same authors [14].

References

1. A.C. Antoulas. *Approximation of Large-scale Dynamical Systems.* SIAM, 2005.
2. V.I. Arnold. *Ordinary Differential Equations.* The MIT Press, ISBN 0-262-51018-9, 1978.
3. M. Bernardo. A survey of Markovian behavioral equivalences. In *Formal Methods for Performance Evaluation*, vol. 4486 of LNCS, pages 180-219. Springer, 2007.
4. M. L. Blinov, J. R. Faeder, B. Goldstein, and W. S. Hlavacek. BioNet-Gen: software for rule-based modeling of signal transduction based on the interactions of molecular domains. *Bioinformatics*, 20(17): 3289-3291, 2004.
5. F. Bonchi, M.M. Bonsangue, M. Boreale, J.J.M.M. Rutten, and A. Silva. A coalgebraic perspective on linear weighted automata. *Inf. Comput.* 211: 77-105, 2012.
6. M. Boreale. Weighted Bisimulation in Linear Algebraic Form. *Proc. of CONCUR 2009*, LNCS 5710, pp. 163-177, Springer, 2009.
7. M. Boreale. Analysis of Probabilistic Systems via Generating Functions and Padé Approximation. *ICALP 2015* (2) 2015: 82-94, LNCS 9135, Springer, 2015. Extended version available as DiSIA working paper 2016/10, `http://local.disia.unifi.it/wp_disia/2016/wp_disia_2016_10.pdf`.
8. M. Boreale. Algebra, coalgebra, and minimization in polynomial differential equations. DiSIA working paper, 2017/01, `http://local.disia.unifi.it/wp_disia/2017/wp_disia_2017_01.pdf`.
9. D. Cachera, Th. Jensen, A. Jobi, and F. Kirchner. Inference of Polynomial Invariants for Imperative Programs: A Farewell to Gröbner Bases. *SAS 2012*, LNCS 7460: 58-74, Springer, 2012.
10. L. Cardelli. On process rate semantics. *Theoretical Computer Science*, 391(3):190-215, 2008.
11. L. Cardelli, M. Tribastone, M. Tschaikowski, and A. Vandin. Symbolic Computation of Differential Equivalences, *POPL 2016*.
12. L. Cardelli, M. Tribastone, M. Tschaikowski, and A. Vandin. Efficient Syntax-driven Lumping of Differential Equations, *TACAS 2016*.
13. L. Cardelli, M. Tribastone, M. Tschaikowski, and A. Vandin. Comparing Chemical Reaction Networks: A Categorical and Algorithmic Perspective, *LICS 2016* (to appear).
14. L. Cardelli, M. Tribastone, M. Tschaikowski, and A. Vandin. ERODE: Evaluation and Reduction of Ordinary Differential Equations. Available from `http://sysma.imtlucca.it/tools/erode/`.
15. F. Ciocchetta and J. Hillston. Bio-PEPA:A framework for the modelling and analysis of biological systems. *Theoretical Computer Science*, 410 (33-34):3065-3084, 2009
16. M. Colón. Polynomial approximations of the relational semantics of imperative programs. *Science of Computer Programming* 64: 76-9, 2007.
17. D. Cox, J. Little, and D. O'Shea. *Ideals, Varieties, and Algorithms An Introduction to Computational Algebraic Geometry and Commutative Algebra.* Undergraduate Texts in Mathematics, Springer, 2007.
18. M. Fliess, and C. Reutenauer. Theorie de Picard-Vessiot des Systèmes Reguliers. *Colloque Nat. CNRS-RCP567, Belle-ile sept. 1982*, in *Outils et Modèles Mathématiques pour l'Automatique l'Analyse des systèmes et le tratement du signal.* CNRS, 1983.
19. K. Ghorbal, A. Platzer. Characterizing Algebraic Invariants by Differential Radical Invariants. *TACAS 2014*: 279-294, 2014. Extended version available from `http://reports-archive.adm.cs.cmu.edu/anon/2013/CMU-CS-13-129.pdf`.

20. G. Li, H. Rabitz, and J. Tóth. A general analysis of exact nonlinear lumping in chemical kinetics. *Chemical Engineering Science* 49 (3), 343-361, 1994.
21. M. Müller-Olm and H. Seidl. Computing polynomial program invariants. *Information Processing Letters* 91(5), 233-244, 2004.
22. M.S. Okino and M.L. Mavrovouniotis. Simplification of mathematical models of chemical reaction systems. *Chemical Reviews*, 2(98):391-408, 1998.
23. A. Platzer. Differential dynamic logic for hybrid systems. *J. Autom. Reasoning* 41(2), 143-189, 2008.
24. A. Platzer. Logics of dynamical systems. In *LICS 2012*: 13-24, IEEE, 2012.
25. E. Rodríguez-Carbonell and D. Kapur. Generating all polynomial invariants in simple loops. *Journal of Symbolic Computation* 42(4), 443-476, 2007.
26. J.J.M.M. Rutten. Behavioural differential equations: a coinductive calculus of streams, automata, and power series. *Theoretical Computer Science*, 308(1–3): 1–53, 2003.
27. D. Sangiorgi. Beyond Bisimulation: The "up-to" Techniques. *FMCO 2005*: 161-171, 2005.
28. S. Sankaranarayanan, H. Sipma, and Z. Manna. Non-linear loop invariant generation using Gröbner bases. *POPL 2004*.
29. S. Sankaranarayanan. Automatic invariant generation for hybrid systems using ideal fixed points. *HSCC 2010*: 221-230, 2010.
30. A. Tiwari. Approximate reachability for linear systems. *HSCC 2003*: 514-525, 2003.
31. J. Tóth, G. Li, H. Rabitz, and A. S. Tomlin. The effect of lumping and expanding on kinetic differential equations. *SIAM Journal on Applied Mathematics*, 57(6):1531-1556, 1997.
32. M. Tribastone, S. Gilmore, and J. Hillston. Scalable differential analysis of process algebra models. *IEEE Trans. Software Eng.*, 38(1):205-219, 2012.
33. E. O. Voit. Biochemical systems theory: A review. *ISRN Biomathematics*, 2013:53, 2013.