Journal of Symbolic Computation 47 (2012) 1233–1266

Contents lists available at SciVerse ScienceDirect

Journal of Symbolic Computation

journal homepage: www.elsevier.com/locate/jsc

Algorithmic Thomas decomposition of algebraic and differential systems

Thomas Bächler, Vladimir Gerdt [1], Markus Lange-Hegermann [2], Daniel Robertz

Lehrstuhl B für Mathematik, RWTH-Aachen University, Templergraben 64, 52062 Aachen, Germany

ARTICLE INFO

Article history:
Received 24 March 2011
Accepted 15 July 2011
Available online 24 December 2011

Keywords:
Disjoint triangular decomposition
Simple systems
Polynomial systems
Differential systems
Involutivity

ABSTRACT

In this paper, we consider systems of algebraic and non-linear partial differential equations and inequations. We decompose these systems into so-called simple subsystems and thereby partition the set of solutions. For algebraic systems, simplicity means triangularity, square-freeness and non-vanishing initials. Differential simplicity extends algebraic simplicity with involutivity. We build upon the constructive ideas of J. M. Thomas and develop them into a new algorithm for disjoint decomposition. The present paper is a revised version of Bächler et al. (2010) and includes the proofs of correctness and termination of our decomposition algorithm. In addition, we illustrate the algorithm with further instructive examples and describe its Maple implementation together with an experimental comparison to some other triangular decomposition algorithms.

© 2011 Elsevier Ltd. All rights reserved.

1. Introduction

Nowadays, triangular decomposition algorithms, which go back to the characteristic set method of Ritt (1950) and Wu (2000), have become powerful tools for investigating and solving systems of multivariate polynomial equations. In many cases these methods are computationally more efficient than those based on the construction of Gröbner bases. For an overview of triangular decomposition methods for polynomial and differential–polynomial systems we refer the reader to the tutorial papers by Hubert (2003a,b) and to the bibliographical references therein.

E-mail addresses: thomas@momo.math.rwth-aachen.de (T. Bächler), gerdt@jinr.ru (V. Gerdt), markus@momo.math.rwth-aachen.de (M. Lange-Hegermann), daniel@momo.math.rwth-aachen.de (D. Robertz).

[1] Permanent address: Joint Institute for Nuclear Research, Dubna, Russia.

[2] Tel.: +49 241 80 94538; fax: +49 241 80 92502.

0747-7171/$ – see front matter © 2011 Elsevier Ltd. All rights reserved.

doi:10.1016/j.jsc.2011.12.043

Among numerous triangular decompositions, the Thomas one stands out. It was suggested by the American mathematician Thomas (1937, 1962) and decomposes a finite system of polynomial equations and/or inequations into finitely many triangular subsystems, which he called *simple*. The Thomas decomposition splits a given quasi-affine variety into a finite number of quasi-affine varieties defined by simple systems. Unlike other decomposition algorithms, the Thomas decomposition always yields a *disjoint* decomposition of the solution set.

Wang was the first to design and implement an algorithm that constructs the Thomas decomposition (cf. Wang (1998, 2001) and Li and Wang (1999)). For polynomial systems he implemented his algorithm in Maple (cf. Wang (2004)) as part of the software package ϵpsilon (cf. Wang (2003)), which also contains implementations of a number of other triangular decomposition algorithms. Delliére (2000) has shown that the "dynamic constructible closure" introduced in the thesis by Gómez Diaz (1994) can be modeled using simple systems. Nonetheless, according to the remark after Delliére (2000, Thm. 5.2), simple systems are more general.

Every simple system is a regular system and its equations form a regular chain. The RegularChains package (cf. Lemaire et al. (2005)) includes procedures for decomposing the solution set of the input by means of regular chains (if the input only consists of equations) or regular systems. However, the Thomas decomposition differs noticeably from this decomposition, since the Thomas decomposition is finer and demands disjointness of the solution set. For a detailed description of algorithms related to regular chains, we refer the reader to Moreno Maza (1999).

The disjointness of the Thomas decomposition combined with the properties of the simple systems provides a useful platform for counting solutions of polynomial systems. In fact, the Thomas decomposition is the only known method for computing the *counting polynomial* introduced by Plesken (2009a).

During his research on triangular decomposition, Thomas was motivated by the Riquier–Janet theory (cf. Riquier (1910) and Janet (1929)), extending it to *non-linear systems of partial differential equations*. For this purpose he developed a theory of (Thomas) monomials, which generate an involutive monomial division nowadays called Thomas division (cf. Gerdt and Blinkov (1998a)). He gave a recipe for decomposing a non-linear differential system into algebraically simple and passive subsystems (cf. Thomas (1937)). A modified version of the differential Thomas decomposition was considered by Gerdt (2008) with its link to the theory of involutive bases (cf. Gerdt and Blinkov (1998a), Gerdt (1999, 2005) and Seiler (2010)). In this decomposition, the output systems are Janet-involutive in accordance to the involutivity criterion from Gerdt (2008) and hence they are coherent. For a linear differential system it is a Janet basis of the corresponding differential ideal, as computed by the Maple package Janet (cf. Blinkov et al. (2003)).

The differential Thomas decomposition differs from that computed by the Rosenfeld–Gröbner algorithm (cf. Boulier et al. (2009) and Boulier et al. (1995)). The latter decomposition forms a basis of the diffalg, DifferentialAlgebra and BLAD packages (cf. Boulier and Hubert (1996–2004) and Boulier (2004–2009)). Experimentally, we found that these three packages are optimized and well-suited for ordinary differential equations. Furthermore, ϵpsilon also allows one to treat ordinary differential systems. Bouziane et al. (2001) mentions another implementation not available to the authors. However, all of these methods give a zero decomposition, which, unlike the Thomas decomposition, is not necessarily disjoint.

In the present paper we present a new algorithmic version of the Thomas decomposition for polynomial and (ordinary and partial) differential systems. In this unified algorithm, only two changes to the algebraic version are necessary to adapt it for the treatment of differential systems. We briefly describe our implementation of this algorithm in Maple.

This paper is organized as follows. In Section 2, we present the algebraic part of our algorithm for the Thomas decomposition with its main objects defined in Section 2.1. In Section 2.2, we describe the main algorithm and its subalgorithms, and then give the correctness and termination proof. Decomposition of differential systems is considered in Section 3. Here, we briefly introduce some basic notions and concepts from differential algebra (Section 3.1) and from the theory of involutive bases specific to Janet division (Section 3.2). In Section 3.3, we present our version of the differential pseudo-reduction and, building upon it, the definition of differential simple systems. Section 3.4 contains a description of the differential Thomas decomposition algorithm and the proof of its correctness and termination. Some implementation issues are discussed in Section 4, and then we give a comparison of our implementation to some other implementations of triangular decompositions with the help of benchmarks.

2. Algebraic systems

This section introduces the concepts of simple systems and the Thomas decomposition for algebraic systems. These concepts are based on properties of the set of solutions of a system. We conclude the section with an algorithm for constructing a Thomas decomposition.

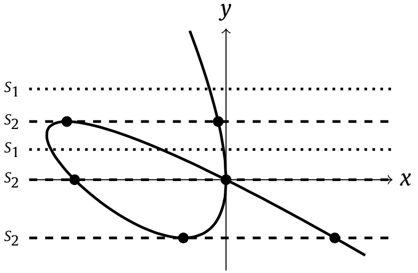
Example 2.1. We give an easy example of a Thomas decomposition. Consider the equation

$$p = \underline{x}^3 + (3y+1)x^2 + (3y^2+2y)x + y^3 = 0.$$

A Thomas decomposition of $\{p = 0\}$ is given by

$$S_1 := \{\underline{x}^3 + (3y+1)x^2 + (3y^2+2y)x + y^3 = 0,\quad 27\underline{y}^3 - 4y \neq 0\}$$

$$S_2 := \{6\underline{x}^2 + (-27y^2+12y+6)x - 3y^2 + 2y = 0,\quad 27\underline{y}^3 - 4y = 0\}$$



The picture shows the solutions of $\{p = 0\}$ in the real affine plane. The cardinality of the fibers of the projection onto the y-component depends on y. However, if we consider all solutions in the complex plane, this cardinality is constant within each system, i.e., 3 and 2 in S_1 and S_2, respectively. This property is formalized in the definition of simple systems.

2.1. Preliminaries

Let F be a computable field of characteristic 0 and $R := F[x_1, \ldots, x_n]$ be the polynomial ring in n variables. A total order $<$ on $\{1, x_1, \ldots, x_n\}$ with $1 < x_i$ for all i is called a **ranking**. From now on, unless otherwise noted, we assume that $i < j$ implies $x_i < x_j$. The indeterminate x is called the **leader** of $p \in R$ if x is the $<$-largest variable occurring in p.[3] In this case we write $\mathrm{ld}(p) = x$. If $p \in F$, we define $\mathrm{ld}(p) = 1$. The degree of p in $\mathrm{ld}(p)$ is called the **main degree** of p ($\mathrm{mdeg}(p)$) and the leading coefficient $\mathrm{init}(p) \in F[y \mid y < \mathrm{ld}(p)]$ of $\mathrm{ld}(p)^{\mathrm{mdeg}(p)}$ in p is called the **initial** of p.

For $\mathbf{a} \in \overline{F}^n$, where $\overline{F}$ denotes the algebraic closure of F, define the following evaluation homomorphism:

$$\phi_{\mathbf{a}} : F[x_1, \ldots, x_n] \to \overline{F} : x_i \mapsto a_i.$$

For $\mathbf{a} \in \overline{F}^i$, $k - 1 \leq i \leq n$, define:

$$\phi_{<x_k,\mathbf{a}} : F[x_1, \ldots, x_n] \to \overline{F}[x_k, \ldots, x_n] : \begin{cases} x_i \mapsto a_i, & i < k \\ x_i \mapsto x_i, & \text{otherwise.} \end{cases}$$

Given a polynomial $p \in R$, the symbols $p_=$ and $p_{\neq}$ denote the equation $p = 0$ and inequation $p \neq 0$, respectively. A finite set of equations and inequations is called an **(algebraic) system** over R. Abusing notation, we sometimes treat $p_=$ or $p_{\neq}$ as the underlying polynomial p. A **solution** of $p_=$ or $p_{\neq}$ is a tuple $\mathbf{a} \in \overline{F}^n$ with $\phi_{\mathbf{a}}(p) = 0$ or $\phi_{\mathbf{a}}(p) \neq 0$, respectively. We call $\mathbf{a} \in \overline{F}^n$ a solution of a system S if it is a solution of each element in S. The set of all solutions of S is denoted by $\mathfrak{Sol}(S)$.

[3] In the context of triangular decompositions, the leader is usually called the **main variable**. The term “leader” is used in Thomas (1937) and was later adopted in differential algebra.

The subsets of all equations $p_= \in S$ and all inequations $p_{\neq} \in S$ are denoted by $S^=$ and $S^{\neq}$, respectively. Define $S_x := \{p \in S \mid \operatorname{ld}(p) = x\}$. In a situation where it is clear that $|S_x| = 1$, we also write S_x to denote the unique element of S_x. The subset $S_{<x} := \{p \in S \mid \operatorname{ld}(p) < x\}$ is a system over $F[y \mid y < x]$.

The Thomas approach uses the homomorphisms $\phi_{<x,\mathbf{a}}$ to treat each polynomial $p \in S_x$ as the family of *univariate* polynomials $\phi_{<x,\mathbf{a}}(p) \in \overline{F}[x]$ for $\mathbf{a} \in \mathfrak{Sol}(S_{<x})$. This idea forms the basis of our central object, the **simple system**:

Definition 2.2 (*Simple Systems*)**.** Let S be a system.

1. S is **triangular** if $|S_{x_i}| \leq 1 \; \forall \; 1 \leq i \leq n$ and $S \cap \{c_=, c_{\neq} \mid c \in F\} = \emptyset$.
2. S has **non-vanishing initials** if $\phi_{\mathbf{a}}(\operatorname{init}(p)) \neq 0 \; \forall \; \mathbf{a} \in \mathfrak{Sol}(S_{<x_i})$ and $p \in S_{x_i}$ for $1 \leq i \leq n$.
3. S is **square-free**[4] if the univariate polynomial $\phi_{<x_i,\mathbf{a}}(p) \in \overline{F}[x_i]$ is square-free $\forall \mathbf{a} \in \mathfrak{Sol}(S_{<x_i})$ and $p \in S_{x_i}$ for $1 \leq i \leq n$.
4. S is called **simple** if it is *triangular*, has *non-vanishing initials* and is *square-free*.

Property (2) and (3) are characterized via solutions of lower-ranking equations and inequations. However, the Thomas decomposition algorithm does not calculate roots of polynomials. Instead, it uses polynomial equations and inequations to *partition* the set of solutions of the lower-ranking system to ensure the above properties.

Remark 2.3. Every simple system has a solution. In particular, if $\mathbf{b} \in \mathfrak{Sol}(S_{<x})$ and S_x is not empty, then $\phi_{<x,\mathbf{b}}(S_x)$ is a univariate polynomial with exactly $\operatorname{mdeg}(S_x)$ *distinct* roots. When S_x is an equation, each solution $\mathbf{b} \in \mathfrak{Sol}(S_{<x})$ extends to a solution $(\mathbf{b}, a) \in \mathfrak{Sol}(S_{\leq x})$ with $\operatorname{mdeg}(S_x)$ possible choices $a \in \overline{F}$. Otherwise, all but finitely many $a \in \overline{F}$ yield a solution $(\mathbf{b}, a) \in \mathfrak{Sol}(S_{\leq x})$, because an inequation S_x excludes $\operatorname{mdeg}(S_x)$ different a, and $S_x = \emptyset$ imposes no restriction on a.

Conversely, if $(a_1, \ldots, a_n) \in \mathfrak{Sol}(S)$ where S is a system over $F[x_1, \ldots, x_n]$ with $x_1 < \cdots < x_n$, then $(a_1, \ldots, a_i) \in \mathfrak{Sol}(S_{\leq x_i})$.

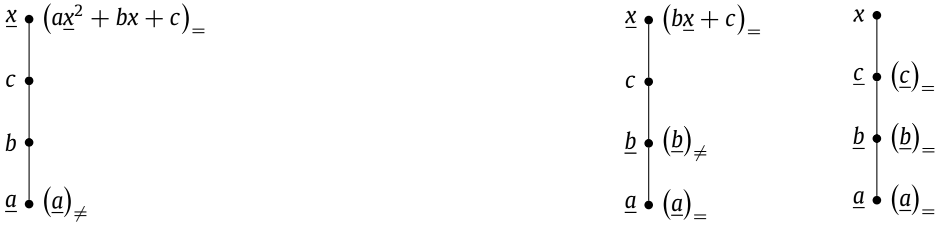
To transform a system into a simple system, it is in general necessary to partition the set of solutions. This leads to a so-called decomposition into simple systems.

Definition 2.4. A family $(S_i)_{i=1}^m$ is called a **decomposition** of S if $\mathfrak{Sol}(S) = \bigcup_{i=1}^m \mathfrak{Sol}(S_i)$. A decomposition is called **disjoint** if $\mathfrak{Sol}(S_i) \cap \mathfrak{Sol}(S_j) = \emptyset \; \forall \; i \neq j$. A *disjoint* decomposition of a system into *simple systems* is called an **(algebraic)** Thomas **decomposition**.

For any algebraic system S, there exists a Thomas decomposition (cf. Thomas (1937, 1962) and Wang (1998)). The algorithm presented in the following section provides another proof of this fact.

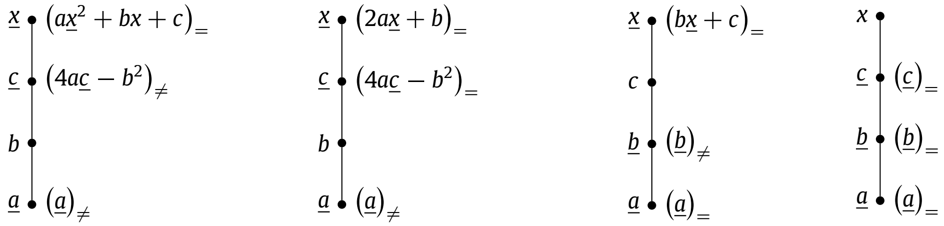
Example 2.5. We compute a Thomas decomposition of $\left\{\left(p := a\underline{x}^2 + bx + c\right)_=\right\} \subseteq \mathbb{Q}[a, b, c, x]$ with respect to $a < b < c < x$. We highlight the highest power of the leader by underlining it.

First, we ensure that the initial $\operatorname{init}(p)$ of p is not zero. Therefore, we insert $(\operatorname{init}(p))_{\neq} = \left(\underline{a}\right)_{\neq}$ into the system. Since we restricted the solution set of this system, we also have to consider the system $\left\{p_=, \left(\underline{a}\right)_=\right\}$, which simplifies to $\left\{\left(b\underline{x} + c\right)_=, \left(\underline{a}\right)_=\right\}$. Similarly, we add $\left(\underline{b}\right)_{\neq}$ to ensure $\operatorname{init}(b\underline{x} + c) \neq 0$ and get the special case system $\left\{\left(\underline{c}\right)_=, \left(\underline{b}\right)_=, \left(\underline{a}\right)_=\right\}$. Up to this point, we have three systems, where the second and third ones are easily checked to be simple:


$\underline{x}$ $\left(a\underline{x}^2 + bx + c\right)_=$
c
b
$\underline{a}$ $\left(\underline{a}\right)_{\neq}$
$\underline{x}$ $\left(b\underline{x} + c\right)_=$
c
$\underline{b}$ $\left(\underline{b}\right)_{\neq}$
$\underline{a}$ $\left(\underline{a}\right)_=$
x
$\underline{c}$ $\left(\underline{c}\right)_=$
$\underline{b}$ $\left(\underline{b}\right)_=$
$\underline{a}$ $\left(\underline{a}\right)_=$

[4] Square-freeness has an important side-effect in the differential case. A square-free polynomial and its separant have no common roots. Thus, the separants do not vanish on solutions of the lower-ranking subsystems.

Second, we ensure that p is square-free by the insertion of $\left(4a\underline{c} - b^2\right)_{\neq}$ into the first system. Again, we also need to consider the system $\{(p)_{=}, \left(4a\underline{c} - b^2\right)_{=}, \left(\underline{a}\right)_{\neq}\}$. As p is a square in this system, we can replace it by its square-free part $2a\underline{x} + b$. Now, all systems are easily verified to be simple and we obtain the following Thomas decomposition:



$\underline{x}$ $\left(a\underline{x}^2 + bx + c\right)_{=}$
$\underline{c}$ $\left(4a\underline{c} - b^2\right)_{\neq}$
b
$\underline{a}$ $\left(\underline{a}\right)_{\neq}$
$\underline{x}$ $\left(2a\underline{x} + b\right)_{=}$
$\underline{c}$ $\left(4a\underline{c} - b^2\right)_{=}$
b
$\underline{a}$ $\left(\underline{a}\right)_{\neq}$
$\underline{x}$ $\left(b\underline{x} + c\right)_{=}$
c
$\underline{b}$ $\left(\underline{b}\right)_{\neq}$
$\underline{a}$ $\left(\underline{a}\right)_{=}$
x
$\underline{c}$ $\left(\underline{c}\right)_{=}$
$\underline{b}$ $\left(\underline{b}\right)_{=}$
$\underline{a}$ $\left(\underline{a}\right)_{=}$

2.2. Algebraic Thomas decomposition

This section presents our main algorithm for algebraic systems and its subalgorithms. The algorithm represents each system as a pair consisting of a candidate simple system and a queue of unprocessed equations and inequations.[5] In each step, the algorithm chooses a suitable polynomial from the queue, pseudo-reduces it and afterwards combines it with the polynomial from the candidate simple system having the same leader. In this process, the algorithm may split the system, i.e., add a new polynomial into the queue as an inequation and at the same time create a new subsystem with the same polynomial added to the queue as an equation. In this way, we ensure that no solutions are lost and the solution sets are disjoint. The algorithm considers a system inconsistent and discards it when an equation of the form $c_{=}$ with $c \in F \setminus \{0\}$ or the inequation $0_{\neq}$ is produced.

We consider a system S as a pair of sets (S_T, S_Q), where S_T represents the candidate simple system and S_Q is the queue. We require S_T to be triangular and thus $(S_T)_x$ denotes the unique equation or inequation of leader x in S_T, if any. Moreover, S_T must fulfill a weaker form of the other two simplicity conditions; in particular, in conditions Definition 2.2(2) and (3), the tuple $\mathbf{a}$ can be a solution of $(S_T)_{<x} \cup (S_Q)_{<x}$ instead of just $(S_T)_{<x}$. Obviously, $S_Q = \emptyset$ implies simplicity of S.

From now on, let prem be a **pseudo-remainder algorithm**[6] in R and pquo the corresponding **pseudo-quotient algorithm**. To be precise, if $p, q \in R$ with $\operatorname{ld}(p) = \operatorname{ld}(q) = x$, then

$$m \cdot p = \mathsf{pquo}(p, q, x) \cdot q + \mathsf{prem}(p, q, x) \tag{1}$$

holds, where $\deg_x(q) > \deg_x(\mathsf{prem}(p, q, x))$, $\operatorname{ld}(m) < x$ and $m \mid \operatorname{init}(q)^k$ for some $k \in \mathbb{Z}_{\geq 0}$. Note that $\phi_{\mathbf{a}}(\operatorname{init}(p)) \neq 0$ and $\phi_{\mathbf{a}}(\operatorname{init}(q)) \neq 0$ imply $\phi_{\mathbf{a}}(\mathsf{pquo}(p, q, x)) \neq 0$ and $\phi_{\mathbf{a}}(m) \neq 0$.

The following algorithm employs prem to reduce a polynomial modulo S_T:

Algorithm 2.6 (Reduce).
Input: A system S, a polynomial $p \in R$.
Output: A polynomial q with $\phi_{\mathbf{a}}(p) = 0$ if and only if $\phi_{\mathbf{a}}(q) = 0$ for each $\mathbf{a} \in \mathfrak{Sol}(S)$.
Algorithm:

```
$x \leftarrow \operatorname{ld}(p)$; $q \leftarrow p$
while $x > 1$ and $(S_T)_x$ is an equation and $\operatorname{mdeg}(q) \geq \operatorname{mdeg}((S_T)_x)$ do
    $q \leftarrow \mathsf{prem}(q, (S_T)_x, x)$
    $x \leftarrow \operatorname{ld}(q)$
end while
if $x > 1$ and Reduce$(S, \operatorname{init}(q)) = 0$ then
```

[5] This approach has been adapted from Gerdt and Blinkov (1998b), where T was an intermediate Janet basis and Q a queue of new prolongations to be checked. A similar approach was later used for triangular decompositions in Moreno Maza (1999).

[6] In our context prem does not necessarily have to be the classical pseudo-remainder, but any sparse pseudo-remainder with property (1) will suffice.

```
    return Reduce($S$, $q - \mathrm{init}(q)x^{\mathrm{mdeg}(q)}$)
else
    return $q$
end if
```

Proof (*Correctness*)**.** There exist $m \in R\setminus\{0\}$ with $\mathrm{ld}(m) < \mathrm{ld}(p)$ and $\phi_{\mathbf{a}}(m) \neq 0$ for all $\mathbf{a} \in \mathfrak{Sol}(S_{\leq \mathrm{ld}(p)})$ such that

$$\mathsf{Reduce}(S, p) = mp - \sum_{y \leq \mathrm{ld}(p)} c_y \cdot (S_T)_y$$

with $c_y \in R$ and $\mathrm{ld}(c_y) \leq \mathrm{ld}(p)$ if $(S_T)_y$ is an equation and $c_y = 0$ otherwise. This implies

$$\phi_{\mathbf{a}}(\mathsf{Reduce}(S, p)) = \underbrace{\phi_{\mathbf{a}}(m)}_{\neq 0} \phi_{\mathbf{a}}(p) - \sum_{y \leq x} \phi_{\mathbf{a}}(c_y) \underbrace{\phi_{\mathbf{a}}((S_T)_y)}_{=0}$$

and therefore $\phi_{\mathbf{a}}(p) = 0$ if and only if $\phi_{\mathbf{a}}(\mathsf{Reduce}(S, p)) = 0$. □

Note that this algorithm only uses the equation part of the triangular system in S, i.e. $S_T^{=}$.

A polynomial p **reduces to** q **modulo** S_T if $\mathsf{Reduce}(S, p) = q$. A polynomial is **reduced modulo** S_T if it reduces to itself.

The Reduce algorithm differs slightly from the classical $\mathsf{prem}(p, S_T^{=})$ as defined in Aubry et al. (1999). While $\mathsf{prem}(p, S_T^{=})$ fully reduces p modulo all variables, $\mathsf{Reduce}(S, p)$ only reduces modulo the leader and ensures that the initial of the reduced form does not vanish. Performing $\mathsf{Reduce}(S, p)$ in combination with a full coefficient reduction (see also Section 4.2) is the same as computing $\mathsf{prem}(p, S_T^{=})$. It is therefore possible to replace $\mathsf{Reduce}(S, p)$ with $\mathsf{prem}(p, S_T^{=})$ in the following algorithms. Our approach adds some flexibility, as we can choose to omit a full reduction in an implementation. In particular, if a polynomial does not reduce to zero, we can determine that without performing a full prem reduction. We apply this multiple times in our implementation, most prominently in Algorithm 2.18. However, if a polynomial reduces to zero, Reduce has no advantage over prem.

Later, we will use the following facts about the Reduce algorithm.

Remark 2.7. Let $q = \mathsf{Reduce}(S, p) \neq 0$.

1. If $S_{\mathrm{ld}(q)}$ is an equation, then $\mathrm{mdeg}(q) < \mathrm{mdeg}(S_{\mathrm{ld}(q)})$.
2. $\mathsf{Reduce}(S, \mathrm{init}(\mathsf{Reduce}(S, p))) \neq 0$.
3. $\mathrm{ld}(q) \leq \mathrm{ld}(p)$ and if $\mathrm{ld}(q) = \mathrm{ld}(p)$, then $\mathrm{mdeg}(q) \leq \mathrm{mdeg}(p)$.

The result of the Reduce algorithm does not need to be a canonical normal form; however, the algorithm recognizes polynomials that vanish on all solutions:

Corollary 2.8. *Let* $p \in R$ *with* $\mathrm{ld}(p) = x$. $\mathsf{Reduce}(S, p) = 0$ *implies* $\phi_{\mathbf{a}}(p) = 0\ \forall\, \mathbf{a} \in \mathfrak{Sol}(S_{\leq x})$.

Proof. For all $\mathbf{a} \in \mathfrak{Sol}(S_{\leq x})$, it holds that $\phi_{\mathbf{a}}(p) = 0$ if and only if $\phi_{\mathbf{a}}(\mathsf{Reduce}(S, p)) = 0$. The statement follows from $\phi_{\mathbf{a}}(\mathsf{Reduce}(S, p)) = \phi_{\mathbf{a}}(0) = 0$. □

The converse of this corollary does not hold in general. Thus, we provide two weaker statements in the following remark.

Remark 2.9. Let p and x as in Corollary 2.8.

1. If $(S_Q)_{\leq x} = \emptyset$, i.e., $S_{\leq x} = (S_T)_{\leq x}$ is simple, then $\mathsf{Reduce}(S, p) \neq 0$ implies $\exists\, \mathbf{a} \in \mathfrak{Sol}(S_{\leq x})$ such that $\phi_{\mathbf{a}}(p) \neq 0$.
2. If $(S_Q)_{<x}^{=} = \emptyset$ and $\mathsf{Reduce}(S, p) \neq 0$ hold, then either $\mathfrak{Sol}(S_{<x}) = \emptyset$ or $\exists\, \mathbf{a} \in \mathfrak{Sol}(S_{<x} \cup \{(S_T)_x\})$ such that $\phi_{\mathbf{a}}(p) \neq 0$.

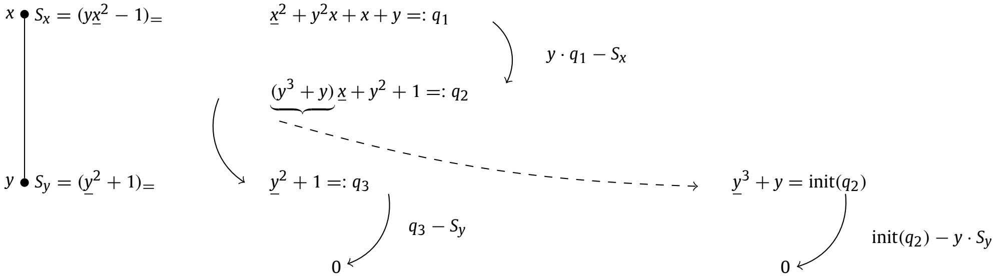
Proof. We only prove the second part, as the first part easily follows.

Let $(S_Q)^{=}_{<x} = \emptyset$, $\mathsf{Reduce}(S, p) \neq 0$ and $|\mathfrak{Sol}(S_{<x})| > 0$. First, as $\mathrm{ld}((S_T)_x) = x$ and $\mathrm{mdeg}((S_T)_x) > 0$, for each $\mathbf{a} \in \mathfrak{Sol}(S_{<x})$, the univariate polynomial $\phi_{<x,\mathbf{a}}((S_T)_x) \in \overline{F}[x]$ has positive degree. Thus $|\mathfrak{Sol}(S_{<x} \cup \{(S_T)_x\})| > 0$.

Let $\phi_{\mathbf{a}}(p) = 0\ \forall\, \mathbf{a} \in \mathfrak{Sol}(S_{<x} \cup \{(S_T)_x\})$ (*). Then $(S_T)_x$ is an equation and $\deg_x(p) \geq \deg_x((S_T)_x)$ and therefore $p \neq \mathsf{Reduce}(S, p)$. In fact, (*) further implies $\mathrm{ld}(\mathsf{Reduce}(S, p)) < x$, as otherwise $\deg_x(\mathsf{Reduce}(S, p)) \geq \deg_x((S_T)_x)$ would hold. By repeating the previous arguments, we can inductively conclude that $\mathrm{ld}(\mathsf{Reduce}(S, p)) = 1$. As $\phi_{\mathbf{a}}(p) = 0$, we conclude that $\mathsf{Reduce}(S, p) = 0$, a contradiction. □

The first part of this remark in conjunction with Corollary 2.8 implies Wang (1998, Thm. 4).

Example 2.10. Reduce $q_1 := x^2 + y^2x + x + y$ modulo the simple system on the left.


x
$S_x = (y\underline{x}^2 - 1)_=$
y
$S_y = (\underline{y}^2 + 1)_=$
$\underline{x}^2 + y^2x + x + y =: q_1$
$y \cdot q_1 - S_x$
$\underbrace{(y^3 + y)}\underline{x} + y^2 + 1 =: q_2$
$\underline{y}^2 + 1 =: q_3$
$\underline{y}^3 + y = \mathrm{init}(q_2)$
$q_3 - S_y$
$\mathrm{init}(q_2) - y \cdot S_y$
0
0

In the first reduction step, q_1 is pseudo-reduced modulo S_x. The result q_2 still has leader x, but a main degree smaller than S_x. We determine that the initial of q_2 reduces to 0 and remove the highest power of x from q_2. The resulting polynomial q_3 now pseudo-reduces to 0 modulo S_y, i.e. $\mathsf{Reduce}(\{S_x, S_y\}, q_1) = 0$.

Now, we examine all splitting methods needed during the algorithm. We will use the following one-liner as a subalgorithm for the splitting subalgorithms.

Algorithm 2.11 (Split). *Input:* A system S, a polynomial $p \in R$.
Output: The disjoint decomposition $\left(S \cup \left\{p_{\neq}\right\}, S \cup \{p_=\}\right)$ of S.
Algorithm:

```
return $\left((S_T, S_Q \cup \{p_{\neq}\}), (S_T, S_Q \cup \{p_=\})\right)$
```

For a better understanding of the following splitting subalgorithms we first need to explain how they are applied in the main algorithm. Each step of the algorithm treats a system S as follows. An equation or inequation q is chosen and removed from the queue S_Q. Then we reduce q modulo S_T. For the simplicity properties to hold w.r.t. q it is necessary to add inequations to S. To accomplish this, we pass S together with q to the splitting subalgorithms. Each such subalgorithm returns two systems. The first system S_1 contains an additional inequation. The second system S_2 contains a complementary equation, q is added back into the queue of S_2, and S_2 is put aside for later treatment. In each case $(S_1 \cup \{q\}, S_2)$ is a disjoint decomposition of the original system $S \cup \{q\}$. Then S_1 and q may be subjected to further splitting algorithms and eventually q is added into the candidate simple system.

The first splitting algorithm that we consider is $\mathsf{InitSplit}$, which is concerned with property (2.2)(2).

Algorithm 2.12 ($\mathsf{InitSplit}$). *Input:* A system S, an equation or inequation q with $\mathrm{ld}(q) = x$.
Output: Two systems S_1 and S_2, where $(S_1 \cup \{q\}, S_2)$ is a disjoint decomposition of $S \cup \{q\}$. Moreover, $\phi_{\mathbf{a}}(\mathrm{init}(q)) \neq 0$ holds for all $\mathbf{a} \in \mathfrak{Sol}(S_1)$ and $\phi_{\mathbf{a}}(\mathrm{init}(q)) = 0$ for all $\mathbf{a} \in \mathfrak{Sol}(S_2)$.
Algorithm:

```
$(S_1, S_2) \leftarrow \mathsf{Split}(S, \mathrm{init}(q))$
$(S_2)_Q \leftarrow (S_2)_Q \cup \{q\}$
return $(S_1, S_2)$
```

For the further splitting algorithms, we need some preparation. In Definition 2.2 we consider a multivariate polynomial p as the family of univariate polynomials $\phi_{<\mathrm{ld}(p),\mathbf{a}}(p)$. For ensuring triangularity and square-freeness, we have to compute the gcd (greatest common divisor) of two polynomials, which in general depends on $\mathbf{a}$. Subresultants provide a generalization of the Euclidean algorithm and enable us to take the tuple $\mathbf{a}$ into account.

Definition 2.13. Let $p, q \in R$ with $\mathrm{ld}(p) = \mathrm{ld}(q) = x$, $\deg_x(p) = d_p > \deg_x(q) = d_q$. We denote by $\mathrm{PRS}(p, q, x)$ the **subresultant polynomial remainder sequence** (see Habicht (1948), Mishra (1993, Chap. 7) and Yap (2000, Chap. 3)) of p and q w.r.t. x, and by $\mathrm{PRS}_i(p, q, x)$, $i < d_q$, the *regular* polynomial of degree i in $\mathrm{PRS}(p, q, x)$ if it exists, denoting this by 0 otherwise. Furthermore, $\mathrm{PRS}_{d_p}(p, q, x) := p$, $\mathrm{PRS}_{d_q}(p, q, x) := q$ and $\mathrm{PRS}_i(p, q, x) := 0$, $d_q < i < d_p$.

Define $\mathrm{res}_i(p, q, x) := \mathrm{init}\,(\mathrm{PRS}_i\,(p, q, x))$ for $0 < i < d_p$, $\mathrm{res}_{d_p}(p, q, x) := 1$ and $\mathrm{res}_0(p, q, x) := \mathrm{PRS}_0\,(p, q, x)$. Note that $\mathrm{res}_0(p, q, x)$ is the usual resultant.[7]

The initials of the subresultants provide conditions for determining the degrees of all possible gcds. Using these conditions, we describe the splittings necessary to determine degrees of polynomials within one system.

Definition 2.14. Let S be a system and $p_1, p_2 \in R$ with $\mathrm{ld}(p_1) = \mathrm{ld}(p_2) = x$. If $|\mathfrak{Sol}(S_{<x})| > 0$, we call

$$i := \min\left\{i \in \mathbb{Z}_{\geq 0} \mid \exists\, \mathbf{a} \in \mathfrak{Sol}(S_{<x}) \text{ such that } \deg_x(\gcd(\phi_{<x,\mathbf{a}}(p_1), \phi_{<x,\mathbf{a}}(p_2))) = i\right\}$$

the **fiber cardinality** of p_1 and p_2 w.r.t. S. Moreover, if $(S_Q)^{=}_{<x} = \emptyset$, then

$$i' := \min\{i \in \mathbb{Z}_{\geq 0} \mid \mathsf{Reduce}(S_T, \mathrm{res}_j(p_1, p_2, x)) = 0\ \forall\, j < i \text{ and } \mathsf{Reduce}(S_T, \mathrm{res}_i(p_1, p_2, x)) \neq 0\}$$

is the **quasi-fiber cardinality** of p_1 and p_2 w.r.t. S. A disjoint decomposition (S_1, S_2) of S such that

1. $\deg_x(\gcd(\phi_{<x,\mathbf{a}}(p_1), \phi_{<x,\mathbf{a}}(p_2))) = i\ \forall\, \mathbf{a} \in \mathfrak{Sol}\,((S_1)_{<x})$ and
2. $\deg_x(\gcd(\phi_{<x,\mathbf{a}}(p_1), \phi_{<x,\mathbf{a}}(p_2))) > i\ \forall\, \mathbf{a} \in \mathfrak{Sol}\,((S_2)_{<x})$

is called ith **fibration split** of p_1 and p_2 w.r.t. S. A polynomial $r \in R$ with $\mathrm{ld}(r) = x$ such that $\deg_x(r) = i$ and

$$\phi_{<x,\mathbf{a}}(r) \sim \gcd(\phi_{<x,\mathbf{a}}(p_1), \phi_{<x,\mathbf{a}}(p_2))\ \forall\, \mathbf{a} \in \mathfrak{Sol}\,((S_1)_{<x})$$

is called ith **conditional greatest common divisor** of p_1 and p_2 w.r.t. S, where $p \sim q$ if and only if $p \in (\overline{F} \setminus \{0\})q$. Furthermore, $q \in R$ with $\mathrm{ld}(q) = x$ and $\deg_x(q) = \deg_x(p_1) - i$ such that

$$\phi_{<x,\mathbf{a}}(q) \sim \frac{\phi_{<x,\mathbf{a}}(p_1)}{\gcd(\phi_{<x,\mathbf{a}}(p_1), \phi_{<x,\mathbf{a}}(p_2))}\ \forall\, \mathbf{a} \in \mathfrak{Sol}\,((S_1)_{<x})$$

is called ith **conditional quotient** of p_1 by p_2 w.r.t. S. By replacing $\phi_{<x,\mathbf{a}}(p_2)$ in the above definition with $\frac{\partial}{\partial x}(\phi_{<x,\mathbf{a}}(p_1))$, we get an ith **square-free split** and an ith **conditional square-free part** of p_1 w.r.t. S.

Example 2.15. Consider the system $S := \{(\underline{x}^3 + y)_=\}$ and the polynomial $q := \underline{x}^2 + x + y + 1$ with $y < x$. Compute $\mathrm{res}_0(S_x, q, x) = \underline{y}^3 + 7y^2 + 5y + 1$, $\mathrm{res}_1(S_x, q, x) = -\underline{y}$ and $\mathrm{res}_2(S_x, q, x) = 1$. The fiber cardinality of S_x and q w.r.t. S is 0. The zeroth fibration split is given by $S_1 := S \cup \{(\mathrm{res}_0(S_x, q, x))_{\neq}\}$ and $S_2 := S \cup \{(\mathrm{res}_0(S_x, q, x))_=\}$. The fiber cardinality w.r.t. S_2 is 1. The first fibration split is given by $S_{2,1} := S_2 \cup \{(-\underline{y})_{\neq}\}$ and $S_{2,2} := S \cup \{(-\underline{y})_=\}$. Note that in this case $\mathfrak{Sol}(S_{2,1}) = \mathfrak{Sol}(S_2)$ and $\mathfrak{Sol}(S_{2,2}) = \emptyset$. The zeroth conditional quotient of S_x and q is S_x. The first conditional gcd and first conditional quotient are $-y\underline{x} + 2y + 1$ and $y^2\underline{x}^2 + (2y^2 + y)x + 4y^2 + 4y + 1$, respectively.

[7] These definitions are slightly different from the ones cited in the literature (Mishra, 1993, Chap. 7, Yap, 2000, Chap. 3), since we only use the regular subresultants. However, it is easy to see that all theorems from Mishra (1993, Chap. 7) that we refer to still hold for $i < d_q$.

It is in general hardly possible to compute the fiber cardinality directly. However, in the case where the quasi-fiber cardinality is strictly smaller than the fiber cardinality, the corresponding fibration split will lead to one inconsistent system, and one where the quasi-fiber cardinality is increased.

Lemma 2.16. *Let $|\mathfrak{Sol}(S_{<x})| > 0$ and $(S_Q)^{=}_{<x} = \emptyset$. For p_1, p_2 as in Definition 2.14 with $\phi_{\mathbf{a}}(\mathrm{init}(p_1)) \neq 0 \; \forall \, \mathbf{a} \in \mathfrak{Sol}(S_{<x})$ and $\mathrm{mdeg}(p_1) > \mathrm{mdeg}(p_2)$, let i be the fiber cardinality of p_1 and p_2 w.r.t. S and i' the corresponding quasi-fiber cardinality. Then*

$$i' \le i$$

where the equality holds if and only if $\left|\mathfrak{Sol}\left(S_{<x} \cup \{\mathrm{res}_{i'}(p_1, p_2, x)_{\neq}\}\right)\right| > 0$.

Proof. Let $\mathbf{a} \in \mathfrak{Sol}(S_{<x})$, $\mathrm{mdeg}(p_1) > \mathrm{mdeg}(p_2)$, $d_{p_1} := \deg_x(p_1) = \deg_x(\phi_{<x,\mathbf{a}}(p_1))$, $d_{p_2} := \deg_x(p_2)$ and $d_{p_2,\mathbf{a}} := \deg_x(\phi_{<x,\mathbf{a}}(p_2))$. If $i < \max(d_{p_1}, d_{p_2,\mathbf{a}}) - 1 = d_{p_1} - 1$, then Mishra (1993, Thm. 7.8.1) implies

$$\phi_{<x,\mathbf{a}}(\mathrm{PRS}_i(p_1, p_2, x)) \sim \mathrm{PRS}_i(\phi_{<x,\mathbf{a}}(p_1), \phi_{<x,\mathbf{a}}(p_2), x) \tag{2}$$

and

$$\phi_{\mathbf{a}}(\mathrm{res}_i(p_1, p_2, x)) = 0 \iff \mathrm{res}_i(\phi_{<x,\mathbf{a}}(p_1), \phi_{<x,\mathbf{a}}(p_2), x) = 0. \tag{3}$$

Conditions (2) and (3) by definition also hold for the trivial cases $d_{p_2} \le i \le d_{p_1}$.

For all indices $j < i'$, Corollary 2.8 and the fact that $\mathsf{Reduce}(S_T, \mathrm{res}_j(p_1, p_2, x)) = 0$ imply that $\phi_{\mathbf{a}}(\mathrm{res}_j(p_1, p_2, x)) = 0$. By (2) and (3), $\mathrm{res}_j(\phi_{<x,\mathbf{a}}(p_1), \phi_{<x,\mathbf{a}}(p_2), x) = 0$ follows. We apply Mishra (1993, Thm. 7.10.5) successively and get $\mathrm{PRS}_j(\phi_{<x,\mathbf{a}}(p_1), \phi_{<x,\mathbf{a}}(p_2), x) = 0$. Thus,

$$\deg_x(\gcd(\phi_{<x,\mathbf{a}}(p_1), \phi_{<x,\mathbf{a}}(p_2))) \ge i' \tag{4}$$

holds. This implies that $i' \le i$.

Equality in (4) holds if and only if there exists $\mathbf{a} \in \mathfrak{Sol}(S_{<x})$ such that $\phi_{\mathbf{a}}(\mathrm{res}_{i'}(p_1, p_2, x)) \neq 0$. Therefore, $i = i'$ if and only if $\left|\mathfrak{Sol}\left(S_{<x} \cup \{\mathrm{res}_{i'}(p_1, p_2, x)_{\neq}\}\right)\right| > 0$. □

The above lemma does not apply if the two polynomials have the same degree. In this case, both polynomials must have non-vanishing initials, as shown in the following corollary.

Corollary 2.17. *Let $|\mathfrak{Sol}(S_{<x})| > 0$ and $(S_Q)^{=}_{<x} = \emptyset$. For polynomials p_1, p_2 as in Definition 2.14 with $\phi_{\mathbf{a}}(\mathrm{init}(p_1)) \neq 0$ and $\phi_{\mathbf{a}}(\mathrm{init}(p_2)) \neq 0 \; \forall \, \mathbf{a} \in \mathfrak{Sol}(S_{<x})$, let i be the fiber cardinality of p_1 and p_2 w.r.t. S and i' the quasi-fiber cardinality of p_1 and $\mathsf{prem}(p_2, p_1, x)$ w.r.t. S. Then*

$$i' \le i$$

with equality if and only if $\left|\mathfrak{Sol}\left(S_{<x} \cup \{\mathrm{res}_{i'}(p_1, \mathsf{prem}(p_2, p_1, x), x)_{\neq}\}\right)\right| > 0$.

Proof. Let $\mathbf{a} \in \mathfrak{Sol}(S_{<x})$. By the assumption on the initials, Mishra (1993, Cor. 7.5.6) implies $\phi_{<x,\mathbf{a}}(\mathsf{prem}(p_2, p_1, x)) = \mathsf{prem}(\phi_{<x,\mathbf{a}}(p_2), \phi_{<x,\mathbf{a}}(p_1), x)$. The univariate polynomials $\phi_{<x,\mathbf{a}}(p_1)$ and $\phi_{<x,\mathbf{a}}(p_2)$ have the same gcd as $\phi_{<x,\mathbf{a}}(p_1)$ and $\mathsf{prem}(\phi_{<x,\mathbf{a}}(p_2), \phi_{<x,\mathbf{a}}(p_1), x)$. We can therefore replace p_2 with $\mathsf{prem}(p_2, p_1, x)$ in Lemma 2.16. □

The following algorithm computes the quasi-fiber cardinality of two polynomials.

Algorithm 2.18 ($\mathsf{ResSplit}$). *Input:* A system S with $(S_Q)^{=}_{<x} = \emptyset$, two polynomials $p, q \in R$ with $\mathrm{ld}(p) = \mathrm{ld}(q) = x$, $\mathrm{mdeg}(p) > \mathrm{mdeg}(q)$ and $\phi_{\mathbf{a}}(\mathrm{init}(p)) \neq 0$ for all $\mathbf{a} \in \mathfrak{Sol}(S_{<x})$.
Output: The quasi-fiber cardinality i of p and q w.r.t. S and an ith fibration split (S_1, S_2) of p and q w.r.t. S.
Algorithm:

```
$i \leftarrow \min\{i \in \mathbb{Z}_{\ge 0} \mid \mathsf{Reduce}(S_T, \mathrm{res}_j(p, q, x)) = 0 \; \forall j < i \text{ and } \mathsf{Reduce}(S_T, \mathrm{res}_i(p, q, x)) \neq 0\}$
return $(i, S_1, S_2) := (i, \mathsf{Split}(S, \mathrm{res}_i(p, q, x)))$
```

Proof (*Correctness*)**.** Assume that $|\mathfrak{Sol}((S_l)_{<x})| > 0, l = 1, 2$, as the statement is trivial otherwise.

Let $\mathbf{a} \in \mathfrak{Sol}((S_1)_{<x})$. The polynomial $g := \mathrm{PRS}_i(\phi_{<x,\mathbf{a}}(p), \phi_{<x,\mathbf{a}}(q), x)$ is not identically zero, due to $(\mathrm{init}(g))_{\neq} = (\mathrm{res}_i(p, q, x))_{\neq} \in (S_1)_Q$. The degree of g is i and $g \sim \gcd(\phi_{<x,\mathbf{a}}(p), \phi_{<x,\mathbf{a}}(q))$, as discussed in the proof of Lemma 2.16.

Let $\mathbf{a} \in \mathfrak{Sol}((S_2)_{<x})$. Mishra (1993, Thm. 7.10.5) and $(\mathrm{init}(g))_{=} = (\mathrm{res}_i(p, q, x))_{=} \in (S_2)_Q$ imply $g \equiv 0$. Therefore, $\deg_x(\gcd(\phi_{<x,\mathbf{a}}(p), \phi_{<x,\mathbf{a}}(q))) > i$. □

We apply the fiber cardinality and fibration split to compute a greatest common divisor of a polynomial in S_T and another polynomial.

Algorithm 2.19 (ResSplitGCD)**.** *Input:* A system S with $(S_Q)^{=}_{<x} = \emptyset$, where $(S_T)_x$ is an equation, and an equation $q_{=}$ with $\mathrm{ld}(q) = x$. Furthermore, $\mathrm{mdeg}(q) < \mathrm{mdeg}((S_T)_x)$.
Output: Two systems S_1 and S_2 and an equation $\widetilde{q}_{=}$ such that:

(a) $S_2 = \widetilde{S_2} \cup \{q\}$ where $\left(S_1, \widetilde{S_2}\right)$ is an ith fibration split of $(S_T)_x$ and q w.r.t. S,
(b) $\widetilde{q}$ is an ith conditional gcd of $(S_T)_x$ and q w.r.t. S,

where i is the quasi-fiber cardinality of p and q w.r.t. S.
Algorithm:

```
1: $(i, S_1, S_2) \leftarrow$ ResSplit $(S, (S_T)_x, q)$
2: $(S_2)_Q \leftarrow (S_2)_Q \cup \{q\}$
3: return $S_1, S_2, \mathrm{PRS}_i((S_T)_x, q, x)_{=}$
```

Proof (*Correctness*)**.** Property (a) follows from Algorithm 2.18 and line 2. Property (b) was already shown in the correctness proof of Algorithm 2.18. □

Note that $i > 0$ is required in this case, as $i = 0$ would yield an inconsistency. Therefore, before calling ResSplitGCD, we will always ensure this condition in the main algorithm by incorporating the resultant of two equations into the system.

The following algorithm is similar. But instead of the gcd, it returns the first input polynomial divided by the gcd.

Algorithm 2.20 (ResSplitDivide)**.** *Input:* A system S with $(S_Q)^{=}_{<x} = \emptyset$ and two polynomials p, q with $\mathrm{ld}(p) = \mathrm{ld}(q) = x$ and $\phi_{\mathbf{a}}(\mathrm{init}(p)) \neq 0$ for all $\mathbf{a} \in \mathfrak{Sol}(S_{<x})$. Furthermore, if $\mathrm{mdeg}(p) \leq \mathrm{mdeg}(q)$ then $\phi_{\mathbf{a}}(\mathrm{init}(q)) \neq 0$.
Output: Two systems S_1 and S_2 and a polynomial $\widetilde{p}$ such that:

(a) $S_2 = \widetilde{S_2} \cup \{q\}$ where $\left(S_1, \widetilde{S_2}\right)$ is an ith fibration split of p and q' w.r.t. S,
(b) $\widetilde{p}$ is an ith conditional quotient of p by q' w.r.t. S,

where i is the quasi-fiber cardinality of p and q' w.r.t. S, with $q' = q$ for $\mathrm{mdeg}(p) > \mathrm{mdeg}(q)$ and $q' = \mathrm{prem}(q, p, x)$ otherwise.
Algorithm:

```
if $\mathrm{mdeg}(p) \leq \mathrm{mdeg}(q)$ then
    return ResSplitDivide$(S, p, \mathrm{prem}(q, p, x))$
else
    $(i, S_1, S_2) \leftarrow$ ResSplit $(S, p, q)$
    if $i > 0$ then
        $\widetilde{p} \leftarrow \mathrm{pquo}(p, \mathrm{PRS}_i(p, \mathrm{prem}(q, p, x), x), x)$
    else
        $\widetilde{p} \leftarrow p$
    end if
    $(S_2)_Q \leftarrow (S_2)_Q \cup \{q\}$
    return $S_1, S_2, \widetilde{p}$
end if
```

Proof (*Correctness*)**.** According to Corollary 2.17, we can without loss of generality assume that $\operatorname{mdeg}(p) > \operatorname{mdeg}(q)$.

Property (a) follows from Algorithm 2.18 and line 10. For all $\mathbf{a} \in \mathfrak{Sol}(S_1)$, the following holds: If $i = 0$, then $\deg_x(\gcd(\phi_{<x,\mathbf{a}}(p), \phi_{<x,\mathbf{a}}(q'))) = 0$ and thus $\phi_{<x,\mathbf{a}}(p)$ shares no roots with $\phi_{<x,\mathbf{a}}(q')$. Now let $i > 0$. Formula (1) implies

$$m \cdot p = \widetilde{p} \cdot \mathrm{PRS}_i\left(p, q', x\right) + \mathsf{prem}\left(p, \mathrm{PRS}_i\left(p, q', x\right), x\right) .$$

Due to Mishra (1993, Cor. 7.5.6) and (2), (3) there exist $k_1, k_2 \in F \setminus \{0\}$ such that

$$\begin{aligned}\underbrace{\phi_{\mathbf{a}}(m)}_{\neq 0} \cdot \phi_{<x,\mathbf{a}}(p) &= \phi_{<x,\mathbf{a}}(\widetilde{p}) \cdot \phi_{<x,\mathbf{a}}\left(\mathrm{PRS}_i\left(p, q, x\right)\right) + \phi_{<x,\mathbf{a}}\left(\mathsf{prem}(p, \mathrm{PRS}_i\left(p, q, x\right), x)\right) \\ &= \phi_{<x,\mathbf{a}}(\widetilde{p}) \cdot k_1 \,\mathrm{PRS}_i\left(\phi_{<x,\mathbf{a}}(p), \phi_{<x,\mathbf{a}}(q), x\right) \\ &\quad + k_2 \mathsf{prem}(\phi_{<x,\mathbf{a}}(p), \underbrace{\mathrm{PRS}_i\left(\phi_{<x,\mathbf{a}}(p), \phi_{<x,\mathbf{a}}(q), x\right)}_{\text{divides } \phi_{<x,\mathbf{a}}(p)}, x) \\ &= \phi_{<x,\mathbf{a}}(\widetilde{p}) \cdot k_1 \gcd(\phi_{<x,\mathbf{a}}(p), \phi_{<x,\mathbf{a}}(q)) + 0.\end{aligned}$$

Thus, we obtain property (b) from

$$\phi_{<x,\mathbf{a}}(\widetilde{p}) \sim \frac{\phi_{<x,\mathbf{a}}(p)}{\gcd(\phi_{<x,\mathbf{a}}(p), \phi_{<x,\mathbf{a}}(q))}$$

and $\deg_x(\phi_{<x,\mathbf{a}}(\widetilde{p})) = \deg_x(\phi_{<x,\mathbf{a}}(p)) - \deg_x(\gcd(\phi_{<x,\mathbf{a}}(p), \phi_{<x,\mathbf{a}}(q))) = \deg_x(p) - i.$ □

Applying the last algorithm to a polynomial p and $\frac{\partial}{\partial \operatorname{ld}(p)} p$ yields an algorithm for making p square-free. We present it separately for better readability of the main algorithm.

Algorithm 2.21 (ResSplitSquareFree)**.** *Input:* A system S with $(S_Q)^{=}_{<x} = \emptyset$ and a polynomial p with $\operatorname{ld}(p) = x$ and $\phi_{\mathbf{a}}(\operatorname{init}(p)) \neq 0$ for all $\mathbf{a} \in \mathfrak{Sol}(S_{<x})$.
Output: Two systems S_1 and S_2 and a polynomial r such that:

(a) $S_2 = \widetilde{S_2} \cup \{p\}$ where $\left(S_1, \widetilde{S_2}\right)$ is an ith square-free split of p w.r.t. S,
(b) r is an ith conditional square-free part of p w.r.t. S,

where i is the quasi-fiber cardinality of p and $\frac{\partial}{\partial x} p$ w.r.t. S.
Algorithm:

```
(i, S_1, S_2) ← ResSplit $\left(S, p, \frac{\partial}{\partial x} p\right)$
if $i > 0$ then
    $r$ ← pquo $\left(p, \mathrm{PRS}_i\left(p, \frac{\partial}{\partial x} p, x\right), x\right)$
else
    $r \leftarrow p$
end if
$(S_2)_Q \leftarrow (S_2)_Q \cup \{p\}$
return $S_1, S_2, r$
```

Proof (*Correctness*)**.** Since $\phi_{<x,\mathbf{a}}(\frac{\partial}{\partial x} p) = \frac{\partial}{\partial x} \phi_{<x,\mathbf{a}}(p)$, an ith square-free split of p is an ith fibration split of p and $\frac{\partial}{\partial x} p$. The rest follows from the proof of Algorithm 2.20. □

In all ResSplit-based algorithms, $(S_Q)^{=}_{<x} = \emptyset$ is required. This ensures that all equations with a leader smaller than x can be used for reduction modulo S_T. The order in which polynomials are treated by the main algorithm must therefore be restricted.

Definition 2.22 (Select)**.** Let $\mathbb{P}_{\mathrm{finite}}(M)$ be the set of all finite subsets of a set M. A **selection strategy** is a map

$$\begin{aligned}\mathsf{Select} : \mathbb{P}_{\mathrm{finite}}(\{p_=, p_{\neq} \mid p \in R\}) &\longrightarrow \{p_=, p_{\neq} \mid p \in R\} : \\ Q &\longmapsto q \in Q\end{aligned}$$

with the following properties:

1. If $\mathsf{Select}(Q) = q_=$ is an equation, then $Q^{=}_{<\mathrm{ld}(q)} = \emptyset$.
2. If $\mathsf{Select}(Q) = q_{\neq}$ is an inequation, then $Q^{=}_{\leq\mathrm{ld}(q)} = \emptyset$.

We demonstrate that these conditions are necessary for termination of our approach, by giving an example where we violate them.

Example 2.23. Consider $R := F[a, x]$ with $a < x$ and the system S with $S_T := \emptyset$ and $S_Q := \left\{(x^2 - a)_=\right\}$. To insert $(x^2 - a)_=$ into S_T, we need to apply the $\mathsf{ResSplitSquareFree}$ algorithm: We calculate $\mathrm{res}_0(x^2 - a, 2x, x) = -4a$, $\mathrm{res}_1(x^2 - a, 2x, x) = 2$ and $\mathrm{res}_2(x^2 - a, 2x, x) = 1$ according to Definition 2.13. The quasi-fiber cardinality is 0 and we get the two new systems S_1, S_2 with

$$(S_1)_T = \{(x^2 - a)_=\}, (S_1)_Q = \{(-4a)_{\neq}\} \quad \text{and} \quad (S_2)_T = \emptyset, (S_2)_Q = \{(x^2 - a)_=, (-4a)_=\}.$$

We now consider what happens with S_2: If we select $(x^2 - a)_=$ as the next equation to be treated, in violation of the properties in Definition 2.22, $\mathsf{ResSplitSquareFree}$ will split up S_2 into $S_{2,1}, S_{2,2}$ with

$$(S_{2,1})_T = \{(x^2 - a)_=\}, (S_{2,1})_Q = \{(-4a)_{\neq}, (-4a)_=\}$$

and

$$(S_{2,2})_T = \emptyset, (S_{2,2})_Q = \{(x^2 - a)_=, (-4a)_=, (-4a)_=\}.$$

As $S_2 = S_{2,2}$, this will lead to an endless loop.

The following trivial algorithm inserts a new equation into S_T. It will be replaced with a different algorithm in Section 3 when the differential Thomas decomposition is considered.

Algorithm 2.24 ($\mathsf{InsertEquation}$). *Input:* A system S and an equation $r_=$ with $\mathrm{ld}(r) = x$ satisfying $\phi_{\mathbf{a}}(\mathrm{init}(r)) \neq 0$ and $\phi_{<x,\mathbf{a}}(r)$ square-free for all $\mathbf{a} \in \mathfrak{Sol}(S_{<x})$.
Output: A system S where $r_=$ is inserted into S_T.
Algorithm:

```
if $(S_T)_x$ is not empty then
    $S_T \leftarrow (S_T \setminus \{(S_T)_x\})$
end if
$S_T \leftarrow S_T \cup \{r_=\}$
return $S$
```

Now we present the main algorithm. The general structure is as follows: in each iteration, a system S is selected from a list P of unfinished systems. An equation or inequation q is chosen from the queue S_Q according to the selection strategy. Then q is reduced modulo S_T and incorporated into the candidate simple system S_T with the splitting algorithms as described above. In doing so, the algorithm may add new systems S_i to P. As soon as the algorithm produces a system containing an equation $c_=$ for $c \in F \setminus \{0\}$ or the inequation $0_{\neq}$, this system is discarded.

Algorithm 2.25 ($\mathsf{Decompose}$). The algorithm is printed on page 1245.

We demonstrate the algorithm with a simple example. Note that we will omit systems which are obviously inconsistent.

Example 2.26. Let $S = (S_T, S_Q) := (\emptyset, \{(x^2 + x + 1)_=, (x + a)_{\neq}\})$ with $a < x$. According to Select, $q := (x^2 + x + 1)_=$ is chosen. As $\mathrm{init}(q) = 1$ and $\mathrm{res}_0(q, \frac{\partial}{\partial x} q, x) = 1$, the original system S is replaced by $\left(\{(x^2 + x + 1)_=\}, \{(x + a)_{\neq}\}\right)$.

Now, $q := (x + a)_{\neq}$ is selected and $\mathsf{ResSplitDivide}(S, (S_T)_x, q)$ computes $\mathrm{res}_0((S_T)_x, q, x) = \mathsf{prem}((S_T)_x, q, x) = a^2 - a + 1$, $\mathrm{res}_1((S_T)_x, q, x) = \mathrm{init}(q) = 1$, and $\mathrm{res}_2((S_T)_x, q, x) = 1$. As S_T contains no equation of leader a, none of these polynomials can be reduced. Then, we decompose S into

$$S := (\underbrace{\{(x^2 + x + 1)_=, (a^2 - a + 1)_{\neq}\}}_{=S_T}, \underbrace{\{\}}_{=S_Q}),$$

Algorithm 2.25 (Decompose)

Input: A system S' with $(S')_T = \emptyset$.
Output: A Thomas decomposition of S'.
Algorithm:

```
P ← {S'}; Result ← ∅
while |P| > 0 do
  Choose S ∈ P; P ← P \ {S}
  if |S_Q| = 0 then
    Result ← Result ∪ {S}
  else
    q ← Select(S_Q); S_Q ← S_Q \ {q}
    q ← Reduce(q, S_T); x ← ld(q)
    if q ∉ {0_≠, c_= | c ∈ F \ {0}} then
      if x ≠ 1 then
        if q is an equation then
          if (S_T)_x is an equation then
            if Reduce(S_T, res_0((S_T)_x, q, x)) = 0 then
              (S, S_1, p) ← ResSplitGCD(S, q); P ← P ∪ {S_1}
              S ← InsertEquation(S, p_=)
            else
              S_Q ← S_Q ∪ {q_=, res_0((S_T)_x, q, x)_=}
            end if
          else
            if (S_T)_x is an inequation^a then
              S_Q ← S_Q ∪ {(S_T)_x}; S_T ← S_T \ {(S_T)_x}
            end if
            (S, S_2) ← InitSplit(S, q); P ← P ∪ {S_2}
            (S, S_3, p) ← ResSplitSquareFree (S, q); P ← P ∪ {S_3}
            S ← InsertEquation(S, p_=)
          end if
        else if q is an inequation then
          if (S_T)_x is an equation then
            (S, S_4, p) ← ResSplitDivide (S, (S_T)_x, q); P ← P ∪ {S_4}
            S ← InsertEquation(S, p_=)
          else
            (S, S_5) ← InitSplit(S, q); P ← P ∪ {S_5}
            (S, S_6, p) ← ResSplitSquareFree (S, q); P ← P ∪ {S_6}
            if (S_T)_x is an inequation then
              (S, S_7, r) ← ResSplitDivide (S, (S_T)_x, p); P ← P ∪ {S_7}
              (S_T)_x ← (r · p)_≠
            else if (S_T)_x is empty then
              (S_T)_x ← p_≠
            end if
          end if
        end if
      end if
      P ← P ∪ {S}
    end if
  end if
end while
return Result
```

[a] Remember that $(S_T)_x$ might be empty, and thus neither an equation nor an inequation.

which is already simple, and

$$S_1 := (\underbrace{\{(x^2+x+1)_=\}}_{=(S_1)_T}, \underbrace{\{(x+a)_{\neq}, (a^2-a+1)_=\}}_{=(S_1)_Q}).$$

We replace S_1 by

$$S_1 := \left(\{(x^2+x+1)_=, (a^2-a+1)_=\}, \{(x+a)_{\neq}\}\right)$$

and apply $\mathsf{ResSplitDivide}(S_1, ((S_1)_T)_x, q)$ to S_1 again. This time, $\mathsf{Reduce}((S_1)_T, a^2-a+1) = 0$ holds and S_1 is replaced with

$$S_1 := (\{\underbrace{(x-a+1)_=}_{\mathsf{pquo}(x^2+x+1, x+a, x)}, (a^2-a+1)_=\}, \{1_{\neq}\}).$$

Finally, a Thomas decomposition of S is

$$\left(\{(x^2+x+1)_=, (a^2-a+1)_{\neq}\}, \{(x-a+1)_=, (a^2-a+1)_=\}\right).$$

Proof (*Correctness*)**.** First, note that it is easily verified that the input specifications of all subalgorithms are fulfilled (in particular, for lines 14 and 29, cf. Remark 2.7(1)).

The correctness of the $\mathsf{Decompose}$ algorithm is proved by verifying two loop invariants:

1. $P \cup \mathit{Result}$ is a disjoint decomposition of the input S'.
2. For all systems $S \in P \cup \mathit{Result}$, S_T is triangular and
   (a) $\phi_{<x,\mathbf{a}}(p)$ is square-free and
   (b) $\phi_{\mathbf{a}}(\mathrm{init}(p)) \neq 0$
   for all $p \in S_T$ with $\mathrm{ld}(p) = x$ and all $\mathbf{a} \in \mathfrak{Sol}((S_T)_{<x} \cup (S_Q)_{<x})$.

We begin with proving the first loop invariant. Assume that $P \cup \mathit{Result}$ is a disjoint decomposition of S' at the beginning of the main loop. It suffices to show that all systems that we add to P or Result add up to a disjoint decomposition of the system S that is chosen in line 3. If $S_Q = \emptyset$ holds in line 4, the algorithm just moves S from P to Result.

In line 17, adding $\mathrm{res}_0((S_T)_x, q, x)_=$ to S does not change the solutions of S, as for each $\mathbf{a} \in \overline{F}^n$, $\phi_{<x,\mathbf{a}}((S_T)_x) = 0$ and $\phi_{<x,\mathbf{a}}(q) = 0$ imply $\phi_{\mathbf{a}}(\mathrm{res}_0((S_T)_x, q, x)) = 0$ (cf. Mishra (1993, Lemma 7.2.3)).

Note now that if (S, S_i) is the output of any of the $\mathsf{ResSplitGcd}$, $\mathsf{InitSplit}$, $\mathsf{ResSplitSquareFree}$ and $\mathsf{ResSplitDivide}$ algorithms, then $(S \cup \{q\}, S_i)$ is a disjoint decomposition of $S_0 \cup \{q\}$, where S_0 is the input of the respective algorithm. It remains to be shown that the actions performed in lines 15, 25, 30, 36 and 38 are equivalent to putting q back into the system S.

Let $\mathbf{a} \in \mathfrak{Sol}(S_{<x})$. In the context of line 15, Algorithm 2.19 guarantees

$$\phi_{<x,\mathbf{a}}(p) = 0 \Longleftrightarrow \phi_{<x,\mathbf{a}}((S_T)_x) = 0 \quad \text{and} \quad \phi_{<x,\mathbf{a}}(q) = 0.$$

In the context of line 30, Algorithm 2.20 ensures that

$$\phi_{<x,\mathbf{a}}(p) = 0 \Longleftrightarrow \phi_{<x,\mathbf{a}}((S_T)_x) = 0 \quad \text{and} \quad \phi_{<x,\mathbf{a}}(q) \neq 0.$$

In lines 25, 36 and 38, p has the same solutions as q, due to Algorithm 2.21 and

$$\phi_{<x,\mathbf{a}}(p) \sim \frac{\phi_{<x,\mathbf{a}}(q)}{\gcd\left(\phi_{<x,\mathbf{a}}(q), \phi_{<x,\mathbf{a}}\left(\frac{\partial}{\partial x} q\right)\right)} = \frac{\phi_{<x,\mathbf{a}}(q)}{\gcd\left(\phi_{<x,\mathbf{a}}(q), \frac{\partial}{\partial x}\phi_{<x,\mathbf{a}}(q)\right)}.$$

In addition, in line 36,

$$\phi_{<x,\mathbf{a}}(r) \sim \frac{\phi_{<x,\mathbf{a}}((S_T)_x)}{\gcd(\phi_{<x,\mathbf{a}}((S_T)_x), \phi_{<x,\mathbf{a}}(p))} \Longrightarrow \phi_{<x,\mathbf{a}}(r \cdot p) \sim \mathrm{lcm}(\phi_{<x,\mathbf{a}}((S_T)_x), \phi_{<x,\mathbf{a}}(p)).$$

This concludes the proof of the first loop invariant.

Now, we prove the second loop invariant. At the beginning, the loop invariant holds because $S'_T = \emptyset$ holds for the input system S'. Assume that the second loop invariant holds at the beginning of the main loop.

One easily checks that all steps in the algorithm allow only one polynomial $(S_T)_x$ in S_T for each leader x; thus triangularity obviously holds.

We show that all polynomials added to S_T have non-zero initial and are square-free. For $\mathfrak{Sol}(S_{<x}) = \emptyset$, the statement is trivially true. So, let $\mathbf{a} \in \mathfrak{Sol}(S_{<x})$.

For the equation $p_=$ added as the conditional gcd of $(S_T)_x$ and q in line 15, it holds that $\phi_{<x,\mathbf{a}}(p)$ is a divisor of $\phi_{<x,\mathbf{a}}((S_T)_x)$. As $\phi_{<x,\mathbf{a}}((S_T)_x)$ is square-free by assumption, so is $\phi_{<x,\mathbf{a}}(p)$. The inequation added to S in ResSplitGCD is by Definition 2.13 the initial of $p_=$.

The equation $p_=$ inserted into S_T in line 25 and the inequation $p_{\neq}$ inserted in line 38 are square-free due to Algorithm 2.21 and their initials are non-zero as p is either identical to q, or it is a pseudo-quotient of q by $\mathrm{PRS}_i\left(q, \frac{\partial}{\partial x}q, x\right)$ for some $i > 0$. On the one hand, if p equals q, the call of InitSplit for q ensures a non-zero initial for p. On the other hand, the polynomial $\mathrm{PRS}_i\left(q, \frac{\partial}{\partial x}q, x\right)$ has initial $\mathrm{res}_i\left(q, \frac{\partial}{\partial x}q, x\right)$, which is added as an inequation by ResSplitSquareFree. This implies that the initial of the pseudo-quotient is also non-zero.

The equation $p_=$ that replaces the old equation $(S_T)_x$ in line 30 is the quotient of $(S_T)_x$ by an inequation. It is square-free, because $\phi_{<x,\mathbf{a}}(p)$ is a divisor of $\phi_{<x,\mathbf{a}}((S_T)_x)$, which is square-free by assumption. Again, p is either identical to $(S_T)_x$ or a pseudo-quotient of $(S_T)_x$ by $\mathrm{PRS}_i\left((S_T)_x, q, x\right)$ for some $i > 0$ and, using the same arguments as in the last paragraph, the initial of p does not vanish.

Finally, consider the inequation $(r \cdot p)_{\neq}$ added in line 36 as a least common multiple of $((S_T)_x)_{\neq}$ and $p_{\neq}$. The inequation $\phi_{<x,\mathbf{a}}(p)$ is square-free and has non-vanishing initial for the same reasons as before. Due to $\phi_{<x,\mathbf{a}}(r) \sim \frac{\phi_{<x,\mathbf{a}}((S_T)_x)}{\gcd(\phi_{<x,\mathbf{a}}((S_T)_x), \phi_{<x,\mathbf{a}}(p))}$, the polynomials $\phi_{<x,\mathbf{a}}(r)$ and $\phi_{<x,\mathbf{a}}(p)$ have no common divisors. As $\phi_{<x,\mathbf{a}}(r)$ divides $\phi_{<x,\mathbf{a}}((S_T)_x)$, using the same arguments as before, $\phi_{<x,\mathbf{a}}(r)$ is square-free and has a non-vanishing initial. This completes the proof of the second loop invariant.

It is obvious that a system S with $S_Q = \emptyset$ for which these loop invariants hold is simple. Thus the algorithm returns the correct result if it terminates. □

We now start showing termination. The system S chosen from P is treated in one of three ways: it is either discarded, added to *Result*, or replaced in P by at least one new system. To show that P is empty after finitely many iterations, we define an order on the systems and show that it is well-founded. Afterwards we prove termination by detailing that the algorithm produces descending chains of systems.

Definition 2.27. For transitive and asymmetric[8] partial orders $<_i$ for $i = 1, \ldots, m$, we define the **composite order** " $<$ " $:= [<_1, \ldots, <_m]$ as follows: $a < b$ if and only if there exists $i \in \{1, \ldots, m\}$ such that $a <_i b$ and neither $a <_j b$ nor $b <_j a$ for $j < i$. The composite order is clearly transitive and asymmetric. An order $<$ is called **well-founded** if each $<$-descending chain becomes stationary.

The following trivial statement will be used repeatedly:

Remark 2.28. If each $<_i$ is well-founded, then so is the composite order $<$, using the notation from Definition 2.27.

Now we define the orders and show their well-foundedness:

Definition and Remark 2.29. Define $\prec$ as the composite order $[\prec_1, \prec_2, \prec_3, \prec_4]$ of the four orders defined below. It is well-founded since the $\prec_i$ are.

1. For $i = 1, \ldots, n$ define $\prec_{1,x_i}$ by $S \prec_{1,x_i} S'$ if and only if $\mathrm{mdeg}\left((S_T)^{=}_{x_i}\right) < \mathrm{mdeg}\left((S'_T)^{=}_{x_i}\right)$, with $\mathrm{mdeg}\left((S_T)^{=}_{x_i}\right) := \infty$ if $(S_T)^{=}_{x_i}$ is empty. Define the composite order $\prec_1$ as $[\prec_{1,x_1}, \ldots, \prec_{1,x_n}]$. Since

[8] A relation $\prec$ is asymmetric if $S \prec S'$ implies $S' \not\prec S$ for all S, S'. Asymmetry implies irreflexivity.

degrees can only decrease finitely many times, the orders $\prec_{1,x_i}$ are clearly well-founded and, thus, $\prec_1$ is.

2. Define the map μ from the set of all systems over R to $\{1, x_1, \ldots, x_n, x_\infty\}$, where $\mu(S)$ is minimal such that there exists an equation $p \in (S_Q)^{=}_{\mu(S)}$ with $\mathsf{Reduce}(S_T, p) \neq 0$, or $\mu(S) = x_\infty$ if no such equation exists. Then, $S \prec_2 S'$ if and only if $\mu(S) < \mu(S')$ with $1 < x_i$ and $x_i < x_\infty$ for $i \in \{1, \ldots, n\}$. The order $\prec_2$ is well-founded since $<$ is well-founded on the finite set $\{1, x_1, \ldots, x_n, x_\infty\}$.
3. $S \prec_3 S'$ if and only if there is $p_{\neq} \in R^{\neq}$ and a finite (possibly empty) set $L \subset R^{\neq}$ with $\mathrm{ld}(q) < \mathrm{ld}(p)\ \forall\, q \in L$ such that $S_Q \uplus \{p_{\neq}\} = S'_Q \uplus L$ holds. We show well-foundedness by induction on the highest leader x appearing in $(S_Q)^{\neq}$: for $x = 1$ we can only make a system S $\prec_3$-smaller by removing one of the finitely many inequations in $(S_Q)^{\neq}$. Now assume that the statement is true for all indeterminates $y < x$. By the induction hypothesis we can only $\prec_3$-decrease S finitely many times without changing $(S_Q)^{\neq}_x$. To further $\prec_3$-decrease S, we have to remove an inequation from $(S_Q)^{\neq}_x$. As $(S_Q)^{\neq}_x$ is finite, this process can only be repeated finitely many times until $(S_Q)^{\neq}_x = \emptyset$. Now, the highest leader appearing in $(S_Q)^{\neq}$ is smaller than x and by the induction hypothesis, the statement is proved.
4. $S \prec_4 S'$ if and only if $|S_Q| < |S'_Q|$.

Proof (*Termination*)**.** We will tacitly use the fact that reduction never makes polynomials bigger in the sense of Remark 2.7(3).

We denote the system chosen from P in line 3 by $\widehat{S}$ and the system added to P in line 43 by S. We prove that the systems $S, S_1, \ldots, S_7$ generated from $\widehat{S}$ are $\prec$-smaller than $\widehat{S}$. For $i = 1, \ldots, 4$ we will use the notation $S \overset{\nsucc}{\nprec}_i S'$ if neither $S \prec_i S'$ nor $S' \prec_i S$ holds.

For $j = 1, \ldots, 7$, $((S_j)_T)^{=} = (\widehat{S_T})^{=}$ and thus $S_j \overset{\nsucc}{\nprec}_1 \widehat{S}$. The properties of Select in Definition 2.22 directly require that there is no equation in $(\widehat{S_Q})^{=}$ with a leader smaller than x. However, the equation added to the system S_j returned from $\mathsf{InitSplit}$ Algorithm 2.12 is the initial of q, which has a leader smaller than x and does not reduce to 0 (cf. Remark 2.7(2)). Furthermore, the equations added in one of the subalgorithms based on $\mathsf{ResSplit}$ Algorithm 2.18 have a leader smaller than x and do not reduce to 0. In each case $S_j \prec_2 \widehat{S}$ is proved.

It remains to show $S \prec \widehat{S}$. If q is reduced to $0_{=}$, then it is omitted from S_Q and so $S \prec_4 \widehat{S}$. As the system is otherwise unchanged, $S \overset{\nsucc}{\nprec}_i \widehat{S}$, $i = 1, 2, 3$, and therefore $S \prec \widehat{S}$ holds. If q is reduced to $c_{\neq}$ for some $c \in F \setminus \{0\}$, then $S \prec_3 \widehat{S}$ and $S \overset{\nsucc}{\nprec}_i \widehat{S}$, $i = 1, 2$, since the only change was the removal of an inequation from S_Q. Otherwise, exactly one of the following cases will occur:

Lines 14–15 set $(S_T)_x$ to $p_{=}$ of smaller degree than $(\widehat{S_T})_x$ and 20–25 add $(S_T)_x$ as a new equation. In both cases we get $S \prec_1 \widehat{S}$.

In line 17, $S_T = \widehat{S}_T$ implies $S \overset{\nsucc}{\nprec}_1 \widehat{S}$. The polynomial q is chosen according to Select (cf. Definition 2.22(1)), which implies $(\widehat{S_Q})^{=}_{<x} = \emptyset$ and $(S_Q)^{=}_{<x} = \{\mathrm{res}_0((S_T)_x, q, x)_{=}\}$. Line 13 ensures that $\mathsf{Reduce}(S, \mathrm{res}_0((S_T)_x, q, x)) \neq 0$ and, thus, $S \prec_2 \widehat{S}$ follows.

Consider lines 29–30. If the degree of $(S_T)_x$ is smaller than the degree of $(\widehat{S_T})_x$, then $S \prec_1 \widehat{S}$. If the degree does not change, we have $S \overset{\nsucc}{\nprec}_1 \widehat{S}$ and $(S_Q)^{=} = (\widehat{S_Q})^{=}$ guarantees $S \overset{\nsucc}{\nprec}_2 \widehat{S}$. However, q is removed from S_Q and replaced by an inequation of smaller leader, which implies $S \prec_3 \widehat{S}$.

In 31–39, obviously $S \overset{\nsucc}{\nprec}_i \widehat{S}$, $i = 1, 2$. As before, q is removed from S_Q and replaced by an inequation of smaller leader, which once more implies $S \prec_3 \widehat{S}$. □

2.3. *Notes on applications of simple systems*

In this subsection, we briefly present some examples where simple systems are necessary and any weaker decomposition into triangular systems is not sufficient.

The properties of simple systems (cf. Definition 2.2) correspond exactly to the following fibration structure on the solution sets (cf. Plesken (2009a)). Let S be a simple system and $\Pi_i : \overline{F}^i \to \overline{F}^{i-1} : (a_1, \dots, a_i) \mapsto (a_1, \dots, a_{i-1})$. Furthermore, for any solution $\mathbf{a} \in \mathfrak{Sol}(S_{\leq x_i})$, let $s_{i,\mathbf{a}} = {\Pi_i}^{-1}(\{\Pi_i(\mathbf{a})\})$. Then, if S_{x_i} is an equation, $|s_{i,\mathbf{a}}| = \operatorname{mdeg}(S_{x_i})$ holds. If S_{x_i} is an inequation, then $s_{i,\mathbf{a}} = \overline{F} \setminus \tilde{s}_{i,\mathbf{a}}$ with $|\tilde{s}_{i,\mathbf{a}}| = \operatorname{mdeg}(S_{x_i})$. If S_{x_i} is empty, then $s_{i,\mathbf{a}} = \overline{F}$. The cardinalities of $s_{i,\mathbf{a}}$ or $\tilde{s}_{i,\mathbf{a}}$ are constant for each i, i.e. independent of the choice of the solution $\mathbf{a} \in \mathfrak{Sol}(S_{\leq x_i})$ (cf. Remark 2.3). We can examine solution sets of arbitrary systems by decomposing them disjointly into simple systems. Further analysis of this fibration structure, especially in the context of algebraic varieties, is a topic of future research.

We already saw such a fibration structure in Example 2.1. In this case, other triangular decompositions like a decomposition into regular chains would have only resulted in a single system consisting of the polynomial p from the input.

A special case occurs when all polynomials in the input and output can be factored into linear polynomials. If we compute the counting polynomial as introduced by Plesken (2009a) (which requires the disjointness of the decomposition and the fibration structure), we can substitute the cardinality of a finite field F (of sufficiently large characteristic) into the counting polynomial of a Thomas decomposition computed over $\mathbb{Q}$. This yields the exact number of distinct solutions over F. For example, the counting polynomial of a Thomas decomposition of $\{\det(M)_{\neq}\}$ for a generic $n \times n$ matrix $M = (x_{ij})_{1 \leq i,j \leq n}$ yields the well-known formula for the cardinality of $\mathrm{GL}_n(F)$ for any finite field F. Furthermore, we can automatically reproduce the results in Plesken (1982, Ex. V.4), where pairs of matrices (A, B) with given ranks of A, B, and $A + B$ are counted.

Plesken (2009b) gave another example concerning the Gauss–Bruhat decomposition and the LU decomposition. The cells of these decompositions of M as above can be identified with certain simple systems in the Thomas decomposition of $\{\det(M)_{\neq}\}$ for suitable rankings on the x_{ij}.

We clearly see that simple systems are necessary for these applications to expose the aforementioned fibration structure and count solutions. A disjoint decomposition into triangular systems with weaker properties does not suffice.

3. Differential Thomas decomposition

The differential Thomas decomposition is concerned with manipulations of polynomial differential equations and inequations. The basic idea for our construction of this decomposition is twofold. On the one hand, a combinatorial calculus developed by Janet finds unique reductors and all integrability conditions by completing systems to involution. On the other hand, the algebraic Thomas decomposition makes the necessary splits for regularity of initials and ensures disjointness of the solution sets.

Initially, we recall some basic definitions from differential algebra. Then, we summarize the Janet division and its relevance. Its combinatorics leads us to substitute the algebraic algorithm InsertEquation by its differential analog. Afterwards, we review a differential generalization of the algebraic reduction algorithm and present the algorithm Reduce utilized for differential reduction. Replacing the insertion and reduction from the previous section with these differential counterparts yields the differential Thomas decomposition algorithm.

3.1. Preliminaries from differential algebra

Let $\Delta = \{\partial_1, \dots, \partial_n\}$ be a non-empty set of derivations and F be a **Δ-ring**. This means that any $\partial_j \in \Delta$ is a linear operator $\partial_j : F \to F$ which satisfies the Leibniz rule. Given a **differential indeterminate** u, the **polynomial Δ-ring** $F\{u\} := F\left[\, u_{\mathbf{i}} \mid \mathbf{i} \in \mathbb{Z}^n_{\geq 0} \,\right]$ is defined as the polynomial ring infinitely generated by the algebraically independent set $\langle u \rangle_\Delta := \{u_{\mathbf{i}} \mid \mathbf{i} \in \mathbb{Z}^n_{\geq 0}\}$. The operation of $\partial_j \in \Delta$ on $\langle u \rangle_\Delta$ is defined by $\partial_j u_{\mathbf{i}} = u_{\mathbf{i}+e_j}$ and this operation extends linearly and via the Leibniz rule to $F\{u\}$. Let $U = \{u^{(1)}, \dots, u^{(m)}\}$ be a set of differential indeterminates. The multivariate polynomial Δ-ring is given by $F\{U\} := F\{u^{(1)}\} \dots \{u^{(m)}\}$. Its generators, the elements of $\langle U \rangle_\Delta :=$

$\left\{u_{\mathbf{i}}^{(j)} \mid \mathbf{i} \in \mathbb{Z}_{\geq 0}^{n}, j \in \{1, \dots, m\}\right\}$, are called **differential variables**. From now on let F be a computable Δ**-field** of characteristic zero.

The differential structure of F uniquely extends to the differential structure of its algebraic closure $\overline{F}$ (Kolchin, 1973, Section II.2, Lemma 1). Let $E := \bigoplus_{j=1}^{m} \overline{F}[[z_1, \dots, z_n]]$ where $\overline{F}[[z_1, \dots, z_n]]$ denotes the ring of formal power series in $z_1, \dots, z_n$. Then E is isomorphic to $\overline{F}^{\langle U \rangle_{\Delta}}$ via

$$\alpha : \bigoplus_{j=1}^{m} \overline{F}[[z_1, \dots, z_n]] \to \overline{F}^{\langle U \rangle_{\Delta}} : \left(\sum_{\mathbf{i} \in \mathbb{Z}_{\geq 0}^{n}} a_{\mathbf{i}}^{(1)} \frac{z^{\mathbf{i}}}{\mathbf{i}!}, \dots, \sum_{\mathbf{i} \in \mathbb{Z}_{\geq 0}^{n}} a_{\mathbf{i}}^{(m)} \frac{z^{\mathbf{i}}}{\mathbf{i}!} \right) \mapsto \left(u_{\mathbf{i}}^{(j)} \mapsto a_{\mathbf{i}}^{(j)} \right)$$

where $z^{\mathbf{i}} := z_1^{i_1} \cdot \ldots \cdot z_n^{i_n}$ and $\mathbf{i}! := i_1! \cdot \ldots \cdot i_n!$.

We define solutions in E, consistent with the algebraic case: for $e \in E$, let

$$\phi_e : F\{U\} \to \overline{F} : u_{\mathbf{i}}^{(j)} \mapsto \alpha(e)(u_{\mathbf{i}}^{(j)})$$

be the F-algebra homomorphism evaluating the differential variables at e. A **differential equation** or **inequation** for m functions $U = \{u^{(1)}, \dots, u^{(m)}\}$ in n indeterminates is an element $p \in F\{U\}$, written as $p_{=}$ or $p_{\neq}$, respectively. A **solution** of $p_{=}$ or $p_{\neq}$ is an $e \in E$ with $\phi_e(\langle p \rangle_{\Delta}) = \{0\}$ or $\phi_e(\langle p \rangle_{\Delta}) \neq \{0\}$, respectively. Here, $\langle p \rangle_{\Delta}$ denotes the differential ideal in $F\{U\}$ generated by p. Furthermore, $e \in E$ is called a solution of a set P of equations and inequations if it is a solution of each element in P. The set of solutions of P is denoted by $\mathfrak{Sol}(P) := \mathfrak{Sol}_E(P) \subseteq E$.

In differential algebra one usually considers solutions in a universal Δ-field, while we consider power series solutions. As the universal differential field we can take the universal closure $\widehat{F}$ of $\overline{F}$. There is a strong link between these two concepts. On the one hand, Seidenberg (1958, 1969) has shown that every finitely differentially generated differential field is differentially isomorphic to a differential field of meromorphic functions in n variables. On the other hand, $\overline{F}[[z_1, \dots, z_n]] \hookrightarrow \overline{F}((z_1, \dots, z_n)) \hookrightarrow \widehat{F}$. Here, the first map is the natural embedding into the quotient field and the second is an embedding given by the definition of the universal Δ-field (Kolchin, 1973, Sections II.2 and III.7), as $\overline{F}((z_1, \dots, z_n))$ is a finitely generated Δ-field extension of $\overline{F}$. Thus, any power series solution can be considered as a solution in the universal differential field.

A finite set of equations and inequations is called a **(differential) system** over $F\{U\}$. We will be using the same notation for systems as in the algebraic Thomas decomposition introduced in Sections 2.1 and 2.2; in particular a system S is represented by a pair (S_T, S_Q). However, the candidate simple system S_T will also reflect a differential structure based on the combinatorics from the following section.

3.2. Janet *division*

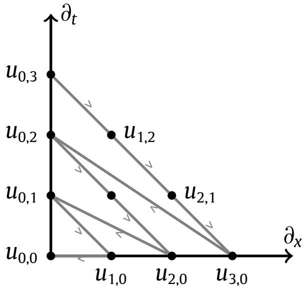
In this subsection we will focus on a combinatorial approach called Janet division (cf. Gerdt and Blinkov (1998a)). It manages the infinite set of differential variables and guarantees inclusion of all integrability conditions in a differential system. For this purpose, it partitions the set of differential variables into "free" variables and finitely many "cones" of dependent variables. We present an algorithm for inserting new equations into an existing set of equations and adjusting this cone decomposition accordingly. An overview of modern development on Janet division can be found in Gerdt (2005) and Seiler (2010) and the original ideas were formulated by Janet (1929).

A (differential) **ranking** $<$ is defined as a total order on the differential variables and 1 with $1 < u \, \forall \, u \in U$, such that

1. $u < \partial_j u$ and
2. $u < v$ implies $\partial_j u < \partial_j v$

for all $u, v \in \langle U \rangle_{\Delta}$, $\partial_j \in \Delta$. From now on let $<$ be an arbitrary and fixed differential ranking. For any finite set of differential variables, a differential ranking induces a ranking as defined for the algebraic case in Section 2.1. Thereby, in accordance with the algebraic part, define the largest differential variable $\mathrm{ld}(p)$ appearing in a differential polynomial $p \in F\{U\}$ as the **leader**, which is set to 1 for $p \in F$. Furthermore, define $\mathrm{mdeg}(p)$ and $\mathrm{init}(p)$ as the degree in the leader and the coefficient of $\mathrm{ld}(p)^{\mathrm{mdeg}(p)}$, respectively.

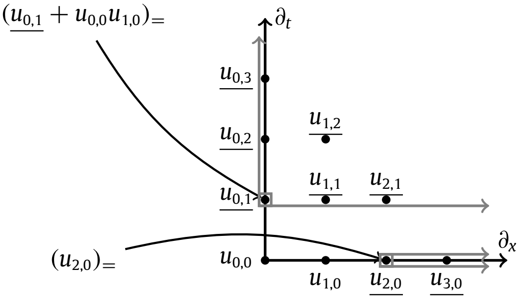
Example 3.1. Consider two derivations $\Delta = \{\partial_x, \partial_t\}$ and one differential indeterminate u.



In this setting, any partial differential equation with constant coefficients in one dependent variable and two independent variables can be represented as a differential polynomial in $\mathbb{C}\{u\}$.

The ranking $<$ is defined by $u_{i_1,i_2} < u_{j_1,j_2}$ if and only if either $i_1 + i_2 < j_1 + j_2$ or $i_1 + i_2 = j_1 + j_2$ and $i_2 < j_2$ holds. Thus, the smallest differential variables are $u_{0,0} < u_{1,0} < u_{0,1} < u_{2,0} < u_{1,1} < u_{0,2} < u_{3,0}$. Considering the set of differential variables as a grid in the first quadrant of a plane, the picture on the left illustrates this ranking.

Consider $(\underline{u_{0,1}} + u_{0,0}u_{1,0})_=$ representing the inviscid Burgers' equation $\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = 0$. As in the algebraic part, we indicate an equation in the picture by attaching it to its leader. However, contrary to the case for the algebraic part, a differential equation affects not only its leader, but also the derivatives of its leader. This is because property (2) of a differential ranking implies $\partial\, \mathrm{ld}(p) = \mathrm{ld}(\partial p)\ \forall \partial \in \Delta, p \in F\{U\}$. For example $\partial_t(\underline{u_{0,1}} + u_{0,0}u_{1,0}) = \underline{u_{0,2}} + u_{0,1}u_{1,0} + u_{0,0}u_{1,1}$. In the diagram we illustrate this by drawing a cone with apex $u_{0,1}$.



Assume that we are only interested in solutions of the inviscid Burgers' equation which are linear in x. So, we add the second equation $(\underline{u_{2,0}})_=$ to our system. This second equation also affects the derivatives of its leader. In particular, $(\underline{u_{0,1}} + u_{0,0}u_{1,0})_=$ and $(\underline{u_{2,0}})_=$ both affect the differential variable $u_{2,1}$ and its derivatives. This contradicts the triangularity of the system. According to the involutive approach as suggested by Janet, we do not allow certain equations to be derived using certain partial derivations. In this example, we allow $(\underline{u_{2,0}})_=$ to be derived only using ∂_x. In the diagram we illustrate this by drawing a (degenerate) cone with apex $u_{2,0}$ in direction of ∂_x. Thus, the differential consequence $(\partial_t \underline{u_{2,0}})_=$ is not yet considered and, so, we have to add it as a separate equation for further treatment.

A set W of differential variables is **closed** under the action of $\Delta' \subseteq \Delta$ if $\partial_i w \in W$ for all $\partial_i \in \Delta'$ and $w \in W$. The smallest set containing a differential variable w, which is closed under Δ', is called a **cone** and denoted by $\langle w\rangle_{\Delta'}$. In this case, we call the elements of Δ' **reductive derivations**.[9] The Δ'-closed set generated by a set W of differential variables is defined as

$$\langle W\rangle_{\Delta'} := \bigcap_{\substack{W_i \supseteq W \\ W_i\ \Delta'\text{-closed}}} W_i \subseteq \langle U\rangle_{\Delta}.$$

For a finite set $W = \{w_1, \ldots, w_r\}$, the Janet **division** algorithmically assigns reductive derivations to the elements of W such that the cones generated by the $w \in W$ are disjoint (cf. Gerdt et al. (2001)

[9] In Gerdt (1999) and Seiler (2010, Chap. 7) the reductive derivations are called multiplicative variables and in Bächler et al. (2010) they are called admissible derivations.

for a fast algorithm). We call these derivations Janet-reductive. The derivation $\partial_l \in \Delta$ is assigned to the cone generated by $w = u^{(j)}_{\mathbf{i}} \in W$ as a reductive derivation if and only if

$$\mathbf{i}_l = \max\left\{\mathbf{i}'_l \mid u^{(j)}_{\mathbf{i}'} \in W, \mathbf{i}'_k = \mathbf{i}_k \text{ for all } 1 \le k < l\right\}$$

holds (cf. Gerdt (2005, Ex. 3.1)). We remark that j is fixed in this definition, i.e., when constructing cones we only take into account other differential variables belonging to the same differential indeterminate. Furthermore, the assignment of reductive derivations to $w \in W$ in general depends on the whole set W. The reductive derivations assigned to w are denoted by $\Delta_W(w) \subseteq \Delta$ and we call the cone $\langle w \rangle_{\Delta_W(w)}$ the Janet **cone** of w with respect to W. This construction ensures disjointness of cones but not necessarily that the union of cones equals $\langle W \rangle_\Delta$. The problem is circumvented by enriching W to its Janet **completion** $\widetilde{W} \supseteq W$. This completion $\widetilde{W}$ is successively created by adding any

$$\tilde{w} = \partial_i w_j \notin \biguplus_{w \in \widetilde{W}} \langle w \rangle_{\Delta_{\widetilde{W}}(w)}$$

to $\widetilde{W}$, where $w_j \in \widetilde{W}$ and $\partial_i \in \Delta \setminus \Delta_{\widetilde{W}}(w_j)$. This leads to the disjoint Janet **decomposition**

$$\langle W \rangle_\Delta = \biguplus_{w \in \widetilde{W}} \langle w \rangle_{\Delta_{\widetilde{W}}(w)}$$

that algorithmically separates a Δ-closed set $\langle W \rangle_\Delta$ into finitely many cones $\langle w \rangle_{\Delta_{\widetilde{W}}(w)}$. For details see Gerdt (2005, Def. 3.4) and Gerdt and Blinkov (1998a, Cor. 4.11).

We extend the Janet decomposition from differential variables to differential polynomials according to their leaders. To be precise, $\Delta_T(q) := \Delta_{\mathrm{ld}(T)}(\mathrm{ld}(q))$ for finite $T \subset F\{U\}$ and $q \in T$. We call a derivative of an equation by a finite (possibly empty) sequence of derivations a **prolongation**. If all these derivations are reductive, the derivative is called **reductive prolongation** of q with respect to T. Otherwise it is called **non-reductive prolongation**.

A differential polynomial $p \in F\{U\}$ is called **reducible** modulo $q \in F\{U\}$ if there exists $\mathbf{i} \in \mathbb{Z}^n_{\ge 0}$ such that $\partial_1^{\mathbf{i}_1} \cdot \ldots \cdot \partial_n^{\mathbf{i}_n} \mathrm{ld}(q) = \mathrm{ld}(\partial_1^{\mathbf{i}_1} \cdot \ldots \cdot \partial_n^{\mathbf{i}_n} q) = \mathrm{ld}(p)$ and $\mathrm{mdeg}(\partial_1^{\mathbf{i}_1} \cdot \ldots \cdot \partial_n^{\mathbf{i}_n} q) \le \mathrm{mdeg}(p)$. For $\mathbf{i} \ne (0, \ldots, 0)$ the condition on the main degree always holds. We now restrict ourselves to reductive prolongations: for a finite set $T \subset F\{U\}$, we call a differential polynomial $p \in F\{U\}$ Janet-**reducible** modulo $q \in T$ w.r.t. T if p is reducible modulo q and $\partial_1^{\mathbf{i}_1} \cdot \ldots \cdot \partial_n^{\mathbf{i}_n} q$ is a reductive prolongation of q w.r.t. T, with $\mathbf{i} \in \mathbb{Z}^n_{\ge 0}$ from the reducibility conditions. We also say that p is Janet-**reducible** modulo T if there is a $q \in T$ such that p is Janet-reducible modulo q w.r.t. T.

A set of differential variables $T \subset \langle U \rangle_\Delta$ is called **minimal** if for any set $S \subset \langle U \rangle_\Delta$ with

$$\biguplus_{t \in T} \langle t \rangle_{\Delta_T(t)} = \biguplus_{s \in S} \langle s \rangle_{\Delta_S(s)}$$

the condition $T \subseteq S$ holds Gerdt and Blinkov (1998b, Def. 4.2). We call a set of differential polynomials minimal if the corresponding set of leaders is minimal.

At each step of the algorithm we assign reductive derivations to the equations in $(S_T)^=$. When an equation p is not reducible modulo $(S_T)^=$, it is added to $(S_T)^=$. Then, we remove all polynomials from S_T that have a leader which is a derivative of $\mathrm{ld}(p)$. This will later ensure minimality. In addition, when adding a new equation to $(S_T)^=$, all non-reductive prolongations are put into the queue. This is formalized in the following algorithm.

Algorithm 3.2 (InsertEquation).
Input: A system S' and a polynomial $p_= \in F\{U\}$ not reducible modulo $(S'_T)^=$.
Output: A system S where $(S_T)^= \subseteq (S'_T)^= \cup \{p_=\}$ is maximal, satisfying

$$(\mathrm{ld}(S_T) \setminus \{\mathrm{ld}(p)\}) \cap \langle \mathrm{ld}(p) \rangle_\Delta = \emptyset,$$
$$S_Q = S'_Q \cup (S'_T \setminus S_T) \cup \{(\partial_i q)_= \mid q \in (S_T)^=, \partial_i \notin \Delta_{((S_T)^=)}(q)\}.$$

Algorithm:

```
$S \leftarrow S'$
$S_T \leftarrow S_T \cup \{p_=\}$
for $q \in S_T \setminus \{p\}$ do
    if $\operatorname{ld}(q) \in \langle \operatorname{ld}(p) \rangle_\Delta$ then
        $S_Q \leftarrow S_Q \cup \{q\}$
        $S_T \leftarrow S_T \setminus \{q\}$
    end if
end for
Reassign reductive derivations to $(S_T)^=$
$S_Q \leftarrow S_Q \cup \{(\partial_i q)_= \mid q \in (S_T)^=, \partial_i \notin \Delta_{((S_T)^=)}(q)\}$
return $S$
```

Correctness and termination are obvious. We remark that a non-reductive prolongation might be added to S_Q several times. An implementation should remember which prolongations have been added before to avoid redundant computations.

3.3. *Differential simple systems*

In this subsection, we extend the algebraic reduction algorithm to its differential counterpart. Finally, we can define differential simple systems at the end of this subsection.

The Janet partition of the dependent differential variables into cones provides a mechanism for finding the unique reductor for the differential reduction quickly (cf. Gerdt et al. (2001)). We prolong this reductor and afterwards apply a pseudo-reduction algorithm.

For a valid pseudo-reduction, we need to ensure that the initials (and the initials of the prolongations) of the equations are non-zero. Let $r \in F\{U\}$ with $x = \operatorname{ld}(r)$ and define the **separant** $\operatorname{sep}(r) := \frac{\partial r}{\partial x}$. One easily checks that the initial of any non-trivial prolongation of r is $\operatorname{sep}(r)$ and the separant of any square-free equation r is non-zero (cf. Kolchin (1973, Section I.8, Lemma 5) or Hubert (2003b, Section3.1)). So, by making sure that the equations have non-vanishing initials and are square-free, as in the algebraic case, we ensure that we can reduce modulo all prolongations of r. This provides the correctness of the following reduction algorithm. [10]

Algorithm 3.3 (Reduce)**.**
Input: A differential system S and a polynomial $p \in F\{U\}$.
Output: A polynomial q that is not Janet-reducible modulo S_T with $\phi_e(p) = 0$ if and only if $\phi_e(q) = 0$ for each $e \in \mathfrak{Sol}(S)$.
Algorithm:

```
$x \leftarrow \operatorname{ld}(p)$
while exists $q_= \in (S_T)^=$ and $\mathbf{i} \in \mathbb{Z}^n_{\geq 0}$ with $\mathbf{i}_j = 0$ for $\partial_j \notin \Delta_{(S_T)^=}(q)$ such that $\partial_1^{\mathbf{i}_1} \cdot \ldots \cdot \partial_n^{\mathbf{i}_n} \operatorname{ld}(q) = \operatorname{ld}(p)$ and $\operatorname{mdeg}(\partial_1^{\mathbf{i}_1} \cdot \ldots \cdot \partial_n^{\mathbf{i}_n} p) \geq \operatorname{mdeg}(q)$ hold do
    $p \leftarrow \mathsf{prem}(p, \partial_1^{\mathbf{i}_1} \cdot \ldots \cdot \partial_n^{\mathbf{i}_n} q, x)$
    $x \leftarrow \operatorname{ld}(p)$
end while
if Reduce$(S, \operatorname{init}(p)) = 0$ then
    return Reduce$(S, p - \operatorname{init}(p) x^{\operatorname{mdeg}(p)})$
else
```

[10] In differential algebra, one usually distinguishes between a (full) differential reduction as used here and a partial (differential) reduction. Partial reduction only employs *proper* derivations of equations for reduction (cf. Kolchin (1973, Section I.9) or Hubert (2003b, Section 3.2)). This is useful for the separation of differential and algebraic parts of the algorithm and for the use of the Rosenfeld Lemma (cf. Rosenfeld (1959)), which is the theoretical basis for the Rosenfeld–Gröbner algorithm (cf. Boulier et al. (2009, 1995) and Hubert (2003b).)

```
    return $p$
end if
```

A polynomial $p \in F\{U\}$ **reduces to** q **modulo** S_T if $\mathsf{Reduce}(S, p) = q$. A polynomial $p \in F\{U\}$ is called **reduced**[11] **modulo** S_T if it reduces to itself. The properties of the algebraic reduction algorithm from Remark 2.7 also apply for this reduction algorithm.

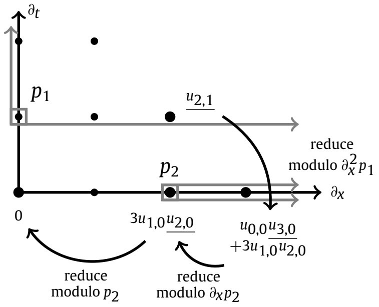
Termination of the reduction algorithm is provided by the DICKSON lemma (cf. Cox et al. (1992, Chap. 2, Thm. 5) or Kolchin (1973, Section 0.17, Lemma 15)), which states that the ranking $<$ is well-founded on the set of leaders, i.e., a strictly $<$-descending chain of leaders is finite.

Example 3.4. We continue Example 3.1 and take care of the differential consequence $(\underline{u_{2,1}})_=$.

We reduce $(\underline{u_{2,1}})_=$ modulo the system S with

$$S_T := \left\{ p_1 := (\underline{u_{0,1}} + u_{0,0}u_{1,0})_=, p_2 := (\underline{u_{2,0}})_= \right\}.$$

First, we observe that $\mathrm{ld}(\underline{u_{2,1}}) = u_{2,1}$ is in the cone generated by $\mathrm{ld}(p_1)$ and $\mathrm{ld}(\partial_x^2 p_1) = \mathrm{ld}(\underline{u_{2,1}})$. Thus, we reduce $(\underline{u_{2,1}})$ modulo $\partial_x^2 p_1$ and the pseudo-reduction yields $u_{0,0}\underline{u_{3,0}} + 3u_{1,0}u_{2,0}$. Second, we reduce $u_{0,0}\underline{u_{3,0}} + 3u_{1,0}u_{2,0}$ modulo $\partial_x p_2$, because $u_{3,0}$ lies in the cone generated by $(\underline{u_{2,0}})_=$. This results in $3u_{1,0}\underline{u_{2,0}}$ and a third reduction step modulo p_2 produces zero. As a result, the only differential consequence is already implied by the system. In this desirable situation, there are no further integrability conditions, which motivates the definition of involutivity below.


∂_t
p_1
$\underline{u_{2,1}}$
reduce modulo $\partial_x^2 p_1$
p_2
∂_x
0
$3u_{1,0}\underline{u_{2,0}}$
$u_{0,0}\underline{u_{3,0}}$ $+3u_{1,0}u_{2,0}$
reduce modulo p_2
reduce modulo $\partial_x p_2$

Now, we define differential simple systems. We demand algebraic simplicity, involutivity of differential equations as seen in the previous Example 3.4, and some minimality conditions.

Definition 3.5 (*Differential Simple Systems*)**.** A differential system S is (JANET-)**involutive** if all non-reductive prolongations of $(S_T)^=$ reduce to zero modulo $(S_T)^=$.
A system S is called **differentially simple** or **simple** if

1. S is algebraically simple (in the finitely many differential variables that appear in it),
2. S is involutive,
3. $S^=$ is minimal,
4. no inequation in $S^{\neq}$ is reducible modulo $S^=$.

A disjoint decomposition of a system into differentially simple subsystems is called a **(differential)** THOMAS **decomposition**.

As in the algebraic case, every simple system has a solution in E.

3.4. The differential decomposition algorithm

The differential THOMAS decomposition algorithm is a modification of the algebraic THOMAS decomposition algorithm. We have already introduced the new algorithms InsertEquation Algorithm 3.2 for adding new equations into the systems and Reduce Algorithm 3.3 for reduction, that can replace their counterparts in the algebraic algorithm. This subsection provides the necessary correctness and termination proofs for the modified algorithm. It then demonstrates this algorithm with examples.

[11] There is a fine difference between not being reducible and being reduced. In the case of not being reducible the initial of a polynomial can still reduce to zero and iteratively the entire polynomial.

Algorithm 3.6 (DifferentialDecompose)**.**
Input: A differential system S' with $(S')_T = \emptyset$.
Output: A differential Thomas decomposition of S'.
Algorithm: The algorithm is obtained by replacing the two subalgorithms InsertEquation and Reduce in Algorithm 2.25 with their differential counterparts Algorithms 3.2 and 3.3, respectively.

Proof (*Correctness*)**.** The correctness proof of the algebraic decomposition Algorithm 2.25 also holds verbatim for the differential case. Therefore, we do not need to show that the output is algebraically simple. We will prove three loop invariants for any system $S \in P \cup \mathit{Result}$:

1. $(S_T)^=$ is minimal.
2. No inequation in $(S_T)^{\neq}$ is Janet-reducible modulo S_T.
3. Let r be any non-reductive prolongation of $(S_T)^=$. Then r reduces to zero by using both conventional differential reductions[12] of $(S_Q)^=$ *and* reductions modulo reductive prolongations of $(S_T)^=$.

The first loop invariant is a purely combinatorial matter, which is proved by Gerdt (2002) for an algorithm using exactly the same combinatorial approach.

Proving the second loop invariant is equally simple. On the one hand, a newly added inequation q in S_T is not Janet-reducible modulo $(S_T)^=$, since algorithm Reduce Algorithm 3.3 is applied to it before insertion. On the other hand, algorithm InsertEquation Algorithm 3.2 removes all inequations from S_T which are divisible by a newly added equation and places them into S_Q.

The third loop invariant clearly holds at the beginning of the algorithm, because S_T is empty.

We claim that reduction of an equation $q_= \in S_Q$ by $(S_T)^=$ in line 8 of Algorithm 2.25 does not affect the loop invariant, i.e. any non-reductive prolongation r reducing to zero beforehand reduces to zero afterwards. We prove this claim by performing a single reduction step on q, which generalizes by an easy induction. Let $q' := \mathrm{prem}(q, p, x) = m \cdot q - \mathrm{pquo}(q, p, x) \cdot p$ be a pseudo-remainder identity (see (1) on page 1237) reducing q to q' modulo p. Then a pseudo-remainder identity $\mathrm{prem}(r, q, x) = m' \cdot r - \mathrm{pquo}(r, q, x) \cdot q$ describing a reduction of r modulo q may simply be rewritten as the iterated identity

$$\underbrace{m \cdot \mathrm{prem}(r, q, x)}_{\mathrm{prem}(\mathrm{prem}(r,p,x),q',x)} = m \cdot m' \cdot r - \mathrm{pquo}(r, q, x) \cdot q' - \mathrm{pquo}(r, q, x) \cdot \mathrm{pquo}(q, p, x) \cdot p.$$

Using the Leibniz rule the same holds for reduction modulo partial derivatives of q. This holds especially for an equation $q_= \in S_Q$ reducing to 0 modulo $(S_T)^=$ in line 8, which can be removed from S_Q without violating the loop invariant.

Now, we consider line 25, where InsertEquation inserts the square-free part $p_=$ of $q_=$ into S_T and show that this does not violate the third loop invariant. First, the non-reductive prolongations in $\{(\partial_i r)_= \mid r \in (S_T)^= \partial_i \notin \Delta_{((S_T)^=)}(r)\}$ are added to S_Q as equations. Thus, any of these reduce to 0 modulo $(S_Q)^=$. Second, moving equations from S_T back into S_Q in InsertEquation does not change the loop invariant either, because their reductive prolongations can still be used for reduction afterwards. Third, every non-reductive prolongation that reduced to zero using $q_= \in (S_Q)^=$ still reduces to zero after InsertEquation. This holds for two reasons. On the one hand, everything that reduces to zero modulo $q_=$ also reduces to zero modulo $p_=$. Write $m \cdot q = p \cdot q_1$ with $\mathrm{ld}(m) < x$ and $\phi_{\mathbf{a}}(m) \neq 0\ \forall \mathbf{a} \in \mathfrak{Sol}(S_{<\mathrm{ld}(q)})$. Then p algebraically pseudo-reduces q to zero. Any derivative ∂q of q is reduced to zero modulo $p_=$ and $(\partial p)_=$, since $\partial(m \cdot q) = (\partial p) \cdot q_1 + p \cdot (\partial q_1)$ for any $\partial \in \Delta$. Inductively, the same holds for repeated derivatives of $q_=$. Therefore, $p_=$ implies all constraints given by $q_=$. On the other hand, all reduction steps modulo $p_=$ are either Janet-reductions modulo $p_=$ w.r.t. S_T or differential reductions modulo non-reductive prolongations of $p_=$. The latter equations have been added to S_Q.

[12] That is, modulo any prolongation.

When computing the gcd of two equations in line 14, the gcd of q and $(S_T)_x$ will be inserted into S_T and reduces everything to zero that both q and $(S_T)_x$ did. As above, the non-reductive prolongations are covered by inserting them into S_Q and the reductive prolongations are implied.

Dividing an equation $(S_T)_x$ by an inequation $q_{\neq}$ in lines 29 and 30 also influences $(S_T)^{=}$. The new equation $p_{=}$, being a divisor of $(S_T)_x$, reduces everything to zero that $(S_T)_x$ and its non-reductive prolongations did by the same arguments as before.

This proves the third loop invariant. When the algorithm terminates, S_Q is empty and thus all non-reductive prolongations from $(S_T)^{=}$ Janet-reduce to zero modulo $(S_T)^{=}$. The system is therefore involutive.

Furthermore, the first loop invariant implies minimality and the second loop invariant implies that no inequation is reducible by an equation, since for an involutive set, reducibility is equivalent to Janet-reducibility. □

Our main tool for proving the termination of the algorithm is using six orders on differential systems. These are similar to the four orders used to show the termination of the algebraic decomposition algorithm. We use the Dickson lemma as the main tool for showing the well-foundedness of these orders.

Definition and Remark 3.7. Define the orders $\prec_{1a}$, $\prec_{1b}$, $\prec_{1c}$, $\prec_2$, $\prec_3$, and $\prec_4$ as follows.

$\prec_{1a}$: For $V \subseteq \langle U \rangle_\Delta$ there is a unique minimal set $\nu(V) \subseteq V$ with $V \subseteq \langle \nu(V) \rangle_\Delta$ (Cox et al., 1992, Chap. 2, Section 4, exercises 7 and 8), called the **canonical differential generators** of V. For a system S, define $\nu(S)$ as $\nu(\mathrm{ld}((S_T)^{=}))$. For systems S, S' we define $S \prec_{1a} S'$ if and only if $\min_<(\nu(S) \setminus \nu(S')) < \min_<(\nu(S') \setminus \nu(S))$. The empty set is assumed to have x_∞ as its minimum, which is $<$-larger than all differential variables. By the Dickson lemma, $\prec_{1a}$ is well-founded.

$\prec_{1b}$: For systems S, S' define $S \prec_{1b} S'$ if and only if $S \not\preceq_{1a} S'$ and $\min_< \left(\mathrm{ld}((S_T)^{=}) \setminus \mathrm{ld}((S'_T)^{=})\right) < \min_< \left(\mathrm{ld}((S'_T)^{=}) \setminus \mathrm{ld}((S_T)^{=})\right)$. Minimality of $(S_T)^{=}$ at each step of the algorithm and the constructivity property of the Janet division (Gerdt and Blinkov, 1998a, Prop. 4.13) imply well-foundedness of $\prec_{1b}$ (Gerdt and Blinkov, 1998a, Thm. 4.14).

$\prec_{1c}$: For systems S and S' with $S \not\preceq_{1a} S'$ and $S \not\preceq_{1b} S'$, we have that $(S_T)^{=}$ and $(S'_T)^{=}$ have the same leaders $x_1, \ldots, x_l$. Define $S \prec_{1c,x_k} S'$ if and only if $\mathrm{mdeg}((S_T)^{=}_{x_i}) < \mathrm{mdeg}((S'_T)^{=}_{x_i})$. This order is clearly well-founded. For these systems define $S \prec_{1c} S'$ as $[\prec_{1c,x_1}, \ldots, \prec_{1c,x_l}]$, which is again well-founded as a composite order.

$\prec_2$: This is defined identically to the algebraic $\prec_2$. We remark that in this case the set of possible leaders is $\{1\} \cup \langle U \rangle_\Delta$. To show well-foundedness of the differential order $\prec_2$ we use that $<$ is well-founded on the set of leaders as implied by the Dickson lemma. In this way, $<$ is extended to a well-founded order on $\{1, x_\infty\} \cup \langle U \rangle_\Delta$ with $1 < y$ and $y < x_\infty$ for all $y \in \langle U \rangle_\Delta$.

$\prec_3$: This is verbatim the same condition and proof of well-foundedness as in the algebraic case. However, in the latter proof, we use a Noetherian induction (Bourbaki, 1968, III.6.5, Prop. 7) instead of an ordinary induction.

$\prec_4$: This is identical to the algebraic case.

Remark 2.28 provides the well-foundedness of the composite order $\prec := [\prec_{1a}, \prec_{1b}, \prec_{1c}, \prec_2, \prec_3, \prec_4]$.

Proof (*Termination*)**.** We prove termination the same way as in the algebraic case. All arguments where systems get $\prec_2$-, $\prec_3$-, or $\prec_4$-smaller apply verbatim here.

In the algebraic case a system $\prec_1$-decreases if and only if either an equation is added to S_T or the degree of an equation in S_T is decreased. We adapt this argument to the differential case: On the one hand, inserting a new equation with a leader that is not yet present in $\mathrm{ld}((S_T)^{=})$ decreases either $\prec_{1a}$ or $\prec_{1b}$. On the other hand, if an equation in $(S_T)^{=}$ is replaced by one with the same leader and lower degree, the system $\prec_{1c}$-decreases.

Thus, like in the algebraic termination proof, we have a strictly decreasing chain of systems and, thus, termination is proved. □

In the following examples, we use jet notation for differential polynomials, e.g., $u_{x,x,y} := u_{2,1}$ in the case $\Delta = \{\partial_x, \partial_y\}$ and $U = \{u\}$.

We give an example taken from Buium and Cassidy (1999, pp. 597–600):

Example 3.8 (*Cole–Hopf Transformation*)**.** For $F := \mathbb{R}(x,t)$, $\Delta = \{\frac{\partial}{\partial x}, \frac{\partial}{\partial t}\}$, and $U = \{\eta, \zeta\}$ consider the heat equation $h = (\eta_t + \eta_{xx})_=$ and the Burgers' equation $b = (\zeta_t + \zeta_{xx} + 2\zeta_x \cdot \zeta)_=$.

First, we claim that any power series solution for the heat equation with a non-zero constant term can be transformed to a solution of the Burgers' equation using the Cole–Hopf transformation $\lambda : \eta \mapsto \frac{\eta_x}{\eta}$. A differential Thomas decomposition for an orderly ranking with $\zeta_x > \eta_t$ of

$$\{h_=, \underbrace{(\eta \cdot \zeta - \eta_x)_=}_{\Leftrightarrow \zeta = \lambda(\eta)}, \eta_{\neq}\}$$

consists of the single system

$$S = \{(\underline{\eta_x} - \eta \cdot \zeta)_=, (\eta \cdot \underline{\zeta_x} + \eta_t + \eta \cdot \zeta^2)_=, \underline{\eta}_{\neq}\}$$

and one checks that $\mathsf{Reduce}(S, b) = 0$ holds. This implies that λ maps any non-zero solution of the heat equation to a solution of the Burgers' equation.

In addition we claim that λ is surjective. For the proof we choose an elimination ranking (cf. Hubert (2003b, Section 8.1) or Boulier (2007)) with $\eta \gg \zeta$, i.e., $\eta_{\mathbf{i}} > \zeta_{\mathbf{j}}$ for all $\mathbf{i}, \mathbf{j} \in \mathbb{Z}_{\geq 0}$. We compute a differential Thomas decomposition of $\{h_=, b_=, (\eta \cdot \zeta - \eta_x)_=, \eta_{\neq}\}$. It consists of the single system

$$S = \{(\underline{\eta_x} - \eta \cdot \zeta)_=, (\eta \cdot \zeta_x + \underline{\eta_t} + \eta \cdot \zeta^2)_=, b_=, \underline{\zeta}_{\neq}\}.$$

The elimination order guarantees that the only constraint for ζ is the Burgers' equation $b_=$. As S is simple, for any solution $f \in \mathfrak{Sol}(b_=)$ there exists a solution $(g, f) \in \mathfrak{Sol}(S)$ (cf. Remark 2.3), implying that λ is surjective.

Elements of the Δ-field F are not subjected to splittings and assumed to be non-zero. However, we are able to model the elements of F as differential indeterminates. For example for $F = \mathbb{C}(x)$ with $\Delta = \{\frac{\partial}{\partial x}\}$, we can study a differential polynomial ring over $\mathbb{C}\{X\}$ instead and replace x by X in all equations and inequations. We subject X to the relation $\frac{\partial}{\partial x} X = 1$ for X "generic" or $(\frac{\partial}{\partial x} X - 1) \cdot \frac{\partial}{\partial x} X = 0$ if we allow specialization of X. These two cases are considered in Examples 3.9 and 3.10, respectively, and will be subjects of further study.

Example 3.9. For $F := \mathbb{C}(x)$, $\Delta = \{\frac{\partial}{\partial x}, \frac{\partial}{\partial t}\}$ and $U = \{u\}$ consider the special case

$$(u_t - u_{xx} - x \cdot u_x - u)_= \tag{5}$$

of the Fokker–Planck equation. We add an auxiliary differential indeterminate X to U and instead examine the equation

$$(u_t - u_{xx} - X \cdot u_x - u)_= , \qquad (X_x - 1)_= , \qquad (X_t)_= \tag{6}$$

in the Δ-ring $\mathbb{C}\{X, u\}$. An elimination ranking $X \gg u$ splits the system (6) into two simple systems:

(i) $\big(u_x \cdot (-\underline{u_{xxx}} + u_{xt} - 2u_x) - u_{xx} \cdot (u_t - u_{xx} - u)\big)_= ,$
$\big(u_x \cdot (-\underline{u_{xxt}} + u_{tt} - u_t) - u_{xt} \cdot (u_t - u_{xx} - u)\big)_= ,$
$\big(u_t - u_{xx} - \underline{X} \cdot u_x - u\big)_= , \big(\underline{u_x}\big)_{\neq}$

(ii) $\big(\underline{u_x}\big)_= , \big(\underline{u_t} - u\big)_= , \big(\underline{X_x} - 1\big)_= , \big(\underline{X_t}\big)_=$

where, due to the ranking, the first two equations in (i) generate $(F\{u\}[\Delta] \cdot (u_t - u_{xx} - x \cdot u_x - u)) \cap \mathbb{C}\{u\}$, i.e., they have constant coefficients. These two equations are the derivatives of $\frac{u_t - u_{xx} - u}{u_x} - x$, which is clearly equivalent to (5) in the case $u_x \neq 0$.

The next example sketches an approach for treating equations with variable coefficients and finding submanifolds where solutions behave differently.

Example 3.10. For $F := \mathbb{C}(x, y)$, $\Delta = \{\frac{\partial}{\partial x}, \frac{\partial}{\partial y}\}$ and $U = \{u\}$ consider

$$(xy - 1) \cdot u(x, y) = 0$$

and determine solutions on $\mathbb{C}^2$ and its submanifolds. A differential Thomas decomposition over $F\{u\}$ simply reproduces this equation, because $(xy - 1) \in F \setminus \{0\}$. However, we can model a search for solutions on submanifolds by adding two differential indeterminates X and Y to U and consider the equations. In order to allow splitting the manifold $\mathbb{C}^2$, we add two differential indeterminates X and Y to U which model the Δ-field elements x and y. Thus, we have to consider the additional equations $(X_x \cdot (X_x - 1))_=$, $(X_y)_=$, $(Y_y \cdot (Y_y - 1))_=$, $(Y_x)_=$ together with the modified equation $((XY - 1) \cdot u)_=$. A differential Thomas decomposition with $X, Y \ll u$ yields three systems:

(i) $(XY - 1)_=, (X_x)_=, (X_y)_=, (X)_{\neq}$,
(ii) $(u)_=, (Y_x)_=, (Y_y \cdot (Y_y - 1))_=, (X_x \cdot (X_x - 1))_=, (X_y)_=, (X)_{\neq}, (XY - 1)_{\neq}$,
(iii) $(u)_=, (Y_x)_=, (Y_y \cdot (Y_y - 1))_=, (X)_=$.

System (i) allows an arbitrary function u on the submanifold $M \subset \mathbb{C}^2$ defined by $xy - 1 = 0$ as a solution. The other systems (ii) and (iii) determine $u \equiv 0$ as the only solution on $\mathbb{C}^2 \setminus M$.

4. Implementation

In this section, we describe our implementation of the decomposition algorithm. First, we list some other implementations of triangular decomposition algorithms. Second, we give some typical optimizations for making the computations feasible. Third, we describe our implementation in Maple. Fourth, we give benchmarks in order to get a more detailed and practical comparison between different decomposition algorithms.

4.1. Implementations of similar decomposition algorithms

The RegularChains package of Lemaire et al. (2005) is shipped with recent versions of Maple. It contains the Triangularize command, which implements a decomposition of an algebraic variety given by a set of equations by means of regular chains. If the input also contains inequations, the resulting decomposition is represented by regular systems instead. It is possible to make these decompositions disjoint using the MakePairwiseDisjoint command.

The ϵpsilon package of Wang (2003) implements different kinds of triangular decompositions in Maple. It is the only software package besides our own that implements the algebraic Thomas decomposition. It closely resembles the approach that Thomas (1937, 1962) suggested, i.e., polynomials of higher leader are considered first. All polynomials with the same leader are combined into one common consequence, resulting in new conditions of lower leader. These are not taken into account right away and will be treated in later steps. Contrary to our the case in approach, one cannot reduce modulo an *unfinished* system. Therefore, one needs extra inconsistency checks to avoid spending too much time on computations with inconsistent systems. ϵpsilon implements such checks in order to achieve good performance.

The Maple packages diffalg of Boulier and Hubert (1996–2004) and DifferentialAlgebra of Boulier and Cheb-Terrab deal with ordinary and partial differential equations as described by Boulier et al. (2009). They compute a radical decomposition of a differential ideal, i.e., a description of the vanishing ideal of the Kolchin closure (Kolchin, 1973, Section IV.1) of the set of solutions. Computation of integrability conditions is driven by the reduction of Δ-polynomials (Rosenfeld, 1959, Section 2), which are analogons of s-polynomials in differential algebra. Just like in RegularChains, this approach usually does not give disjoint solution sets, although in principle disjointness might be achieved. The diffalg package has been superseded by DifferentialAlgebra in Maple 14. DifferentialAlgebra is based on the BLAD libraries of Boulier (2004–2009) which have been designed as a set of stand-alone C libraries with an emphasis on usability for non-mathematicians and extensive documentation.

4.2. Algorithmic optimizations

In this subsection, we describe algorithmic optimizations helpful for a reasonably fast implementation of the Decompose algorithm.

In our algorithm, pseudo-remainder sequences for the same pairs of polynomials are usually needed several times in different branches. As these calculations are expensive in general, our implementation always keeps the results in memory and reuses them when the same pseudo-remainder sequence is requested again to *avoid repeated computations*.

Coefficient growth is a common problem in elimination. Polynomials should be represented as compactly as possible. Once we know that the initial of a polynomial is non-zero, the *content* of a polynomial (in the univariate sense) is non-zero, too. Thus, every time an initial is added to the system as an inequation, we can divide the polynomial by its content. Additionally, the multivariate content, which is an element of F, can be removed.

The reduction Algorithms 2.6 and 3.3 do not recognize that non-leading coefficients are zero. However, we can reduce the coefficients modulo the polynomials of lower leader, in addition to reduction of the polynomial itself. Thereby, in some cases the sizes of coefficients decrease, and in other cases they increase. The latter outcome is partly due to multiplying the whole polynomials with the initials of the reductors. Finding a good heuristic for this *coefficient reduction* is crucial for efficiency.

Factorization of a polynomial improves computation time in many cases. More precisely, the system $S \uplus \{(p \cdot q)_=\}$ decomposes disjointly into $(S \cup \{p_=\}, S \cup \{p_{\neq}, q_=\})$ and the system $S \cup \{(p \cdot q)_{\neq}\}$ is equivalent to $S \cup \{p_{\neq}, q_{\neq}\}$. In most cases, the computation of two smaller problems resulting from a factorization is cheaper than the computation of the big, original problem. This idea extends to factorizations over an extension of the base field: Let $Y_i := \left\{x_j \mid x_j < x_i, (S_T)^{=}_{x_j} \neq \emptyset\right\}$ and $Z_i := \left\{x_j \mid x_j < x_i, (S_T)^{=}_{x_j} = \emptyset\right\}$. Assume that $(S_T)^{=}_{x_i}$ is irreducible over the field $F_i := F(Z_i)[Y_i]/\langle (S_T)^{=}_{<x_i} \rangle$ for all $i \in \{1, \ldots, n\}$, where $\langle (S_T)^{=}_{<x_i} \rangle$ is the ideal generated by $(S_T)^{=}_{<x_i}$ in the polynomial ring $F(Z_i)[Y_i]$. Factorization over F_n instead of F may split the polynomial into more factors, but it is not clear whether this improves the runtime. Preliminary tests show that factorization over F should be preferred for $F = \mathbb{Q}$.

In the algebraic algorithm, polynomials need not be square-free when they are inserted into the candidate simple system. Efficiency can sometimes be improved by postponing the computation of the square-free split as long as possible. However, this is not possible for the differential case. Differential polynomials need to be made square-free to ensure that their separant is non-zero, i.e. non-trivial prolongations have a non-zero initial.

In the differential case, application of *criteria* can decrease computation time by avoiding useless reductions of non-reductive prolongations. The Janet combinatorial approach already avoids many reductions of Δ-polynomials, as used in other approaches (see Gerdt and Yanovich (2006)). In addition, we use the involutive criteria 2–4 (cf. Gerdt and Blinkov (1998a), Gerdt (2005) and Apel and Hemmecke (2005)), which together are equivalent to the chain criterion. Applicability of this criterion in the non-linear differential case was shown in Boulier et al. (2009, Section 4, Prop. 5).

The axioms of a selection strategy (see Definition 2.22) already strongly limit the choice for the polynomial considered in the current step. However, the remaining freedom is another important aspect for the speed of an actual implementation. We will describe different selection strategies in Section 4.3 and compare them in the benchmarks.

As described up to now, the algorithm often keeps on computing with inconsistent systems. We want to optimize the algorithm in order to detect the inconsistencies as early as possible. This allows the algorithm to discard inconsistent systems as early as possible. One of the problems is the selection strategies postponing the costly treatment of inequations. A test for detecting whether inequations in S_Q reduce to zero is comparably cheap.

Another possible improvement is parallelization, since the main loop in the Decompose Algorithm 2.25 can naturally be used in parallel for different systems.

4.3. Selection strategies

We consider our two main approaches to selection strategies (see Definition 2.22).

1. The "equations first" strategies: Select only chooses an inequation if Q does not contain any equations. Among the equations or inequations, it prefers the ones with the smallest leader.
2. The "leader first" strategies: Select always chooses an equation or inequation with the smallest leader occurring in Q. If there are both equations and inequations with that leader, it chooses an equation.

In both approaches, if the above criteria do not yield a unique choice, we compare the leader of the initial and choose the smaller one. We apply the last test recursively to the initial of the initial and so on. At this point, it is still possible that we fail to make a unique selection. However, these cases are rare and there does not seem to be a considerable performance advantage for any choice. Therefore, it suffices to make an arbitrary (but preferably unique) choice.

In our experimental observation "leader first" strategies usually produce decompositions with fewer systems, while "equations first" strategies are more efficient (cf. Section 4.5).

4.4. Implementation in MAPLE

Both the algebraic and the differential case of the THOMAS decomposition algorithm have been implemented in the MAPLE computer algebra system. Packages can be downloaded from our web page (Bächler and Lange-Hegermann, 2008–2012); documentation and example worksheets are available there.

The main reason for choosing MAPLE for the implementation is the collection of solvers for polynomial equations, ODEs, and PDEs already present. Furthermore, fast algorithms exist for polynomial factorization over finitely generated field extensions of $\mathbb{Q}$ and for gcd computation.

The AlgebraicThomas package includes procedures for computing a THOMAS decomposition, reducing polynomials modulo simple systems and computing counting polynomials (cf. Plesken (2009a)). Furthermore, it can represent the complement and intersection of solution sets as decompositions into simple systems. Finally, a comprehensive THOMAS decomposition can be computed; this topic will be discussed in a later publication.

Example 4.1. We demonstrate how to use the AlgebraicThomas package by computing a decomposition of the system in example Example 2.5.

```
>  with(AlgebraicThomas):
>  p := a*x^2 + b*x + c;
```

$$p := x^2\, a + x\, b + c$$

```
>  S := AlgebraicThomasDecomposition([p], [x,c,b,a]);
```

$$S := [[x^2\, a + x\, b + c = 0,\ 4\, c\, a - b^2 \neq 0,\ a \neq 0],\ [2\, x\, a + b = 0,\ 4\, c\, a - b^2 = 0,\ a \neq 0],\ [x\, b + c = 0,\ b \neq 0,\ a = 0],\ [c = 0,\ b = 0,\ a = 0]]$$

Information about the leader and main degree can optionally be included in the output.

```
>  map(printSystem, S, ["PT", "LR"]);
```

$$[[[x^2\, a + x\, b + c = 0,\ x^2],\ [4\, c\, a - b^2 \neq 0,\ c],\ [a \neq 0,\ a]],$$
$$[[2\, x\, a + b = 0,\ x],\ [4\, c\, a - b^2 = 0,\ c],\ [a \neq 0,\ a]],$$
$$[[x\, b + c = 0,\ x],\ [b \neq 0,\ b],\ [a = 0,\ a]],\ [[c = 0,\ c],\ [b = 0,\ b],\ [a = 0,\ a]]]$$

It is possible to include inequations in the input to exclude some degenerate cases:

```
>  q := a<>0;
```

$$q := a \neq 0$$

```
>  T := AlgebraicThomasDecomposition([p, q], [x,c,b,a]);
```

$$T := [[x^2\,a + x\,b + c = 0,\ 4\,c\,a - b^2 \neq 0,\ a \neq 0],\ [2\,x\,a + b = 0,\ 4\,c\,a - b^2 = 0,\ a \neq 0]]$$

Features for the differential package DifferentialThomas include arbitrary differential rankings, using special functions implemented in Maple as differential field elements, computation of power series solutions, and a direct connection to the solvers of Maple for differential equations.

Example 4.2. We treat the following control theoretic example taken from Diop (1992).

```
>  with(DifferentialThomas):
>  ComputeRanking([t],[x2,x1,y,u],"EliminateFunction");
```

This creates the differential polynomial ring $\mathbb{Q}\{x^{(2)}, x^{(1)}, y, u\}$ for $\Delta = \{\frac{\partial}{\partial t}\}$. Here u indicates the input, $x^{(1)}$ and $x^{(2)}$ the state, and y the output of the system. The chosen ranking "$<$" is the elimination ranking with $x^{(2)} \gg x^{(1)} \gg y \gg u$, i.e., $x^{(2)}_{\mathbf{i}} > x^{(1)}_{\mathbf{j}} > y_{\mathbf{k}} > u_{\mathbf{l}}$ for all $\mathbf{i}, \mathbf{j}, \mathbf{k}, \mathbf{l} \in \mathbb{Z}_{\geq 0}$.

```
>  L:=[x1[1]-u[0]*x2[0],x2[1]-x1[0]-u[0]*x2[0],y[0]-x1[0]]:
```

We follow Diop (1992, Ex. 1) and compute the external trajectories of a differential ideal generated by L, i.e. intersect this differential ideal with $\mathbb{Q}\{y, u\}$.

```
>  res:=DifferentialThomasDecomposition(L,[]);
```

$$res := [DifferentialSystem,\ DifferentialSystem]$$

We show the equations and inequations of the differential systems not involving $x^{(1)}$ or $x^{(2)}$. The chosen ranking guarantees that the systems shown determine the external trajectories of the system:

```
>  PrettyPrintDifferentialSystem(res[1]):remove(a->has(a,[x1,x2]),%);
```

$$[-\mathrm{u}(t)\,(\tfrac{d^2}{dt^2}\,\mathrm{y}(t)) + (\tfrac{d}{dt}\,\mathrm{y}(t))\,\mathrm{u}(t)^2 + (\tfrac{d}{dt}\,\mathrm{y}(t))\,(\tfrac{d}{dt}\,\mathrm{u}(t)) + \mathrm{y}(t)\,\mathrm{u}(t)^2 = 0,\ \mathrm{u}(t) \neq 0]$$

```
>  PrettyPrintDifferentialSystem(res[2]):remove(a->has(a,[x1,x2]),%);
```

$$[\tfrac{d}{dt}\,\mathrm{y}(t) = 0,\ \mathrm{u}(t) = 0]$$

These systems, having disjoint solution sets, are identical to the ones found in Diop (1992).

4.5. Benchmarks

In this subsection, we compare our two Maple packages to the other implementations of triangular decompositions mentioned in Section 4.1 using benchmarks. Not all of the implementations compute equivalent results. This should be considered when comparing the timings. We omitted examples where all tested systems took less than one second to complete the computation or could not be computed by any software package.

All benchmarks have been performed with Linux x86-64 running on a third-generation Opteron, 2.3 GHz. The time limit has been set to 3 h and available memory is limited to 4 GB. All times are given in seconds. The polynomial multiplication in Maple 14 benefits from a new parallel implementation (cf. Monagan and Pearce (2009)). Nonetheless, we state the total CPU time in our benchmarks, as returned by the Maple `time` command.

By default, both of our Maple packages behave as follows:

- Polynomials are factorized over $\mathbb{Q}$.
- The content of polynomials is removed.
- The selection strategy is an "equations first" strategy, as described in Section 4.3.
- After reducing a polynomial, we always reduce its coefficients fully.
- Inequations in S_Q are reduced for early inconsistency checks.

See Section 4.2 for details.

Table 1
Comparison of algebraic decompositions 1: `polsys50` from Wang (2003).

Name	RC1	RC2	RC3	DW1	DW2	AT1	AT2	AT3	AT4
1	3.5	3.7	4.3	0.4	1.0	3.0	1.1	1.8	1.4
2	7.4	6.7	7.5	7.6	8.4	7.1	169.7	95.8	6.6
3	>3 h	>3 h	>4 GB	985.7	1344.6	7538.0	>4 GB	>4 GB	194.6
4	>4 GB	>4 GB	>4 GB	>4 GB	>4 GB	0.2	>4 GB	>4 GB	32.1
6	0.4	0.4	47.2	0.1	0.2	0.2	0.1	0.2	0.1
7	>3 h	>3 h	>3 h	7352.6	>3 h	>4 GB	>4 GB	>4 GB	>4 GB
12	0.5	0.6	0.5	0.3	0.4	0.4	0.6	1.1	0.4
14	0.5	2.3	0.6	>3 h	>4 GB	1.5	1.6	1.4	2.5
16	0.9	0.9	1.0	1.4	1.5	1.8	5.6	>3 h	2.2
17	6.5	6.4	13.0	4.7	6.3	75.5	12 076.5	>3 h	12.6
18	0.3	0.3	3.7	0.1	0.1	0.1	0.1	0.1	0.1
19	419.9	452.9	>4 GB	0.4	0.6	0.4	5842.5	0.4	0.3
21	1.6	1.9	2.1	86.6	>4 GB	4.5	>3 h	4.4	112.8
22	0.6	0.6	0.6	1.2	1.6	1.5	2.9	32.4	2.0
23	0.4	0.7	0.4	0.1	>4 GB	29.5	>3 h	>4 GB	29.0
24	1.2	1.1	1.3	1.3	2.6	1.0	2.0	4.5	1.6
25	1.2	8.5	1.6	>3 h	>4 GB	>4 GB	>3 h	>3 h	>3 h
29	0.3	0.5	0.4	0.3	0.3	0.3	55.2	0.3	0.3
30	>4 GB	>4 GB	>4 GB	>4 GB	>3 h	45.3	42.9	40.8	>4 GB
31	>4 GB	>4 GB	>4 GB	>4 GB	>3 h	>3 h	>4 GB	>3 h	>3 h
33	3.4	3.6	3.2	1.3	1.3	3.5	66.9	15.2	1.1
34	911.5	916.9	926.5	>3 h	>4 GB	>4 GB	>4 GB	>3 h	>4 GB
35	1.5	1.5	1.6	1.2	1.3	1.7	4.9	7.3	0.5
39	0.6	0.7	0.8	1.2	1.9	0.6	1.0	8.0	0.5
41	1.5	1.5	1.6	1.5	1.7	7.0	1.4	0.6	114.5
43	0.7	0.7	0.7	3.1	4.4	0.2	1.0	0.2	0.2
44	24.5	17.2	24.1	3.4	4.2	1.2	1.7	0.8	>4 GB
47	1.3	1.7	1.4	2.8	6.6	13.0	>3 h	11.1	92.4
49	0.3	0.3	0.3	610.2	32.1	0.5	>3 h	0.5	0.5

4.5.1. Algebraic systems

For testing the AlgebraicThomas package, we used two sets of examples, namely, the test examples from the `polsys50` file in Wang's ϵpsilon package (Wang, 2003) printed in Table 1 and the examples from Chen et al. (2007) as shown in Table 2.

In contrast to the case for Algorithm 2.25, the implementation in the AlgebraicThomas package inserts equations or inequations into S_T without making them square-free first. It delays this computation as long as possible, sometimes until the end of the decomposition. This avoids some expensive and unnecessary discriminant computations entirely.

We compared AlgebraicThomas with the RegularChains package from Maple 14 and ϵpsilon. We also tested the AlgebraicThomas and RegularChains packages in different configurations. The timings of the following procedures are being compared:

- (RC1) `RegularChains[Triangularize]`.
- (RC2) `RegularChains[Triangularize]` with the `'output'='lazard'` option set.
- (RC3) `RegularChains[Triangularize]` with the `'radical'='yes'` option set.
- (DW1) `epsilon[RegSer]`.
- (DW2) `sisys[simser]`.
- (AT1) `AlgebraicThomasDecomposition`.
- (AT2) `AlgebraicThomasDecomposition` with factorization disabled.
- (AT3) `AlgebraicThomasDecomposition` with a "leader first" selection strategy (cf. Section 4.3).
- (AT4) `AlgebraicThomasDecomposition` with coefficient reduction disabled.

We compare Tables 1 and 2 within our own implementation. We observe that (AT2) is much slower than (AT1) and, thus, conclude that factorization is vital for making many computations feasible. In a

Table 2
Comparison of algebraic decompositions 2: test examples from Chen et al. (2007).

Name	RC1	RC2	RC3	DW1	DW2	AT1	AT2	AT3	AT4
AlkashiSinus	0.6	0.6	0.8	0.1	7.1	5.7	2.6	6.5	3.6
Bronstein	0.4	0.5	0.5	0.2	0.4	0.3	0.4	1.1	0.4
Cheaters-homotopy-easy	0.7	>3 h	532.5	>4 GB	>4 GB	>3 h	>4 GB	>3 h	>3 h
Cheaters-homotopy-hard	0.7	>3 h	559.8	>4 GB	>4 GB	>3 h	>4 GB	>3 h	>3 h
Gerdt	1.4	1.4	1.4	2.0	2.2	8.1	3.2	0.5	1532.1
Hereman-2	0.8	1.0	0.8	0.3	0.4	0.3	1.2	0.3	0.5
Hereman-8-8	26.9	31.6	208.3	>3 h	>3 h	>3 h	>3 h	>3 h	>4 GB
KdV	722.2	707.1	725.7	>3 h	>3 h	>3 h	>3 h	>3 h	>3 h
Lanconelli	0.4	0.6	0.4	0.2	0.4	0.4	1.3	0.3	0.3
Lazard-ascm2001	1.2	17.5	1.4	>3 h	error	>4 GB	>4 GB	>3 h	>3 h
Leykin-1	5.6	8.0	5.8	>3 h	>3 h	2.3	>3 h	6.0	1.4
Maclane	2.4	6.7	2.6	3576.5	>4 GB	7.4	17.4	13.1	7.0
MontesS10	0.5	1.0	0.6	0.4	17.7	2.0	2.2	2.3	1.5
MontesS11	0.2	0.5	0.2	0.2	>3 h	23.5	>3 h	21.9	12.4
MontesS12	0.5	3.2	0.5	1.6	>3 h	9.4	31.0	13.6	115.4
MontesS13	0.3	0.5	0.3	0.2	0.5	0.8	1.8	1.1	0.9
MontesS14	0.7	1.3	0.8	>3 h	>3 h	6.0	>4 GB	14.5	12.1
MontesS15	1.2	1.8	1.3	0.6	8.2	4.8	3.8	6.7	3.3
MontesS16	4.3	3.4	4.2	1.5	1.5	2.5	2.2	3.2	1.5
MontesS7	0.4	0.5	0.4	0.2	0.5	0.7	0.6	2.5	1.2
Neural	0.5	0.7	0.6	>3 h	>4 GB	1.4	3050.9	1.7	1.2
Pavelle	1.1	15.9	1.4	>3 h	>4 GB	>3 h	>3 h	>3 h	>3 h
Wang93	1.2	1.3	1.2	1.6	3.4	4.9	>3 h	3.4	6.5
genLinSyst-3-2	0.3	1.1	0.3	0.2	0.2	0.3	0.2	0.3	0.2
genLinSyst-3-3	0.3	4.5	0.4	1.2	1.2	6.0	2.5	4.8	1.1

few examples, we see the relative advantage of the default selection strategy compared to the one used in (AT3). Generally speaking, disabling coefficient reduction increases computation time for (AT4), but there are some strong counterexamples to this observation. This indicates that different strategies for coefficient reduction, as seen in (AT1) and (AT4), should be investigated further.

The programs `sisys[simser]` (DW2) and `AlgebraicThomasDecomposition` (AT1-4) are the only ones that compute a Thomas decomposition. All test examples that could be computed by (DW2) could also be computed by (AT1). However, there are some examples that `RegularChains` (RC1) or `epsilon` (DW1) could treat, but we could not decompose into simple systems. Moreover, the test examples indicate that (RC1) is in general faster than (AT1) in the positive-dimensional case. Our evaluation suggests that this is due to the strict square-free property of simple systems. In the zero-dimensional case, however, the situation is less clear, since there are examples where (RC1) is faster than (AT1) and vice versa.

4.5.2. Differential systems

We compared DifferentialThomas with the packages diffalg and DifferentialAlgebra, see Tables 3 and 4. Finding a suitable set of benchmark examples for the differential case was more difficult. We are not aware of any sets of standard benchmarks. Thus, we used a collection of examples which we came across in our work. These examples are published on our homepage (Bächler and Lange-Hegermann, 2008–2012).

The timings of the following procedures are compared:

- (DA) `DifferentialAlgebra[Rosenfeld_Groebner]`.
- (da) `diffalg[Rosenfeld_Groebner]`.[13]
- (DT1) `DifferentialThomasDecomposition`.
- (DT2) `DifferentialThomasDecomposition` with factorization disabled.

[13] with `_env_diffalg_uses_DifferentialAlgebra:=false`

Table 3
Benchmarks for ODE systems.

Name	DA	da	DT1	DT2	DT3	DT4
Diffalg4	2.9	2.9	852.5	>3 h	8932.4	36.0
LLG3	0.5	>3 h	5.4	5.6	4.4	4.9
LLG4	0.3	19.1	2.6	37.4	20.3	4.0
ODE1	2.4	3.7	0.6	0.3	0.6	0.8
ODE6	2.3	1.5	0.8	1.2	0.6	0.8
ODE7	>3 h	>3 h	3.2	60.8	47.2	5.4
kepler vs newton	0.7	0.9	1.4	2.8	0.8	2.0
keppler1	0.1	0.2	1.1	0.6	0.8	1.1
keppler2	0.1	0.1	1.3	1.5	1.1	0.8
keppler3	0.1	0.2	1.1	0.6	0.7	1.1
murray1	0.1	0.4	1.9	2.4	1.7	2.2
murray2	0.1	0.1	0.7	1.4	0.8	0.6

Table 4
Benchmarks for PDE systems.

Name	DA	da	DT1	DT2	DT3	DT4
Cyclic 5 variant1	>3 h	2.9	1.3	1.5	1.4	1.2
Cyclic 5 variant2	>3 h	>3 h	2.3	2.6	0.7	2.3
Diffalg2	0.5	0.3	1.4	1.5	41.5	1.2
Diffalg3	0.2	0.5	0.9	0.6	1.6	0.7
Ibragimov 2, 17.9c	>3 h	>3 h	18.4	>3 h	40.8	12.3
PDE6	>4 GB	>4 GB	11.1	23.0	>4 GB	16.5
PDE7	>4 GB	116.7	91.3	83.3	>4 GB	41.4
PDE8	>4 GB	>4 GB	6.8	>4 GB	14.2	7.5
Riquier 1b	0.1	0.1	1.9	1.9	1.9	2.0
Riquier 3a	0.3	0.2	0.8	1.1	0.5	0.6
Riquier 3b	0.6	0.6	1.4	1.7	1.4	1.2
boulier	>3 h	546.4	2.5	1690.9	2.6	1.5
cyclic 6	>4 GB	571.9	160.7	159.8	349.6	154.1
noon6	>4 GB	72.6	40.6	36.8	63.7	31.0

- (DT3) `DifferentialThomasDecomposition` with a "leader first" selection strategy (cf. Section 4.3)
- (DT4) `DifferentialThomasDecomposition` with coefficient reduction disabled.

We want to mention one further example not included in the benchmark tables. It is the test example 5 of `diffalg`. None of the packages in their default setting could compute this example. Still, `diffalg` and `DifferentialAlgebra` were able to do so instantaneously by a change of ordering strategy.

The comparison between (DT1), (DT2) and (DT3) is similar to the algebraic case. In particular, factorization should be enabled and the default selection strategy should be preferred. In contrast to the algebraic implementation, the comparison of (DT1) and (DT4) is less conclusive.

All test examples which could be computed by `DifferentialAlgebra` or `diffalg` could also be computed by our default strategy (DT1). For ODEs, the three packages show similar timings, but for PDEs, `DifferentialThomasDecomposition` appears to be faster. This might be explained by the involutive approach, which we utilize to make the subsystems coherent. A similar result can be found for the GINV project (cf. Blinkov et al. (2010a,b)).

Acknowledgements

The contents of this paper profited very much from numerous useful comments and remarks by Wilhelm Plesken. The authors thank him as well as Dongming Wang, François Boulier and

François Lemaire for fruitful discussions. The second author (V.P.G.) acknowledges the Deutsche Forschungsgemeinschaft for the financial support that made his stay in Aachen possible. The results presented were obtained during his visits. The contributions of the first and third authors (T.B. and M.L.-H.) were partly supported by Schwerpunkt SPP 1489 of the Deutsche Forschungsgemeinschaft. Finally, our gratitude goes to the anonymous referees for valuable comments, for pointing out informative references and for taking the time to verify the results of our benchmarks.

References

Apel, J., Hemmecke, R., 2005. Detecting unnecessary reductions in an involutive basis computation. J. Symbolic Comput. 40 (4–5), 1131–1149. URL http://dx.doi.org/10.1016/j.jsc.2004.04.004.

Aubry, P., Lazard, D., Moreno Maza, M., 1999. On the theories of triangular sets. J. Symbolic Comput. 28 (1–2), 105–124. Polynomial elimination—algorithms and applications. URL http://dx.doi.org/10.1006/jsco.1999.0269.

Bächler, T., Gerdt, V., Lange-Hegermann, M., Robertz, D., 2010. Thomas decomposition of algebraic and differential systems. In: Computer Algebra in Scientific Computing. Tsakhkadzor, Armenia, pp. 31–54.

Bächler, T., Lange-Hegermann, M., 2008–2012. AlgebraicThomas and DifferentialThomas: Thomas Decomposition for algebraic and differential systems (http://wwwb.math.rwth-aachen.de/thomasdecomposition/).

Blinkov, Y.A., Cid, C.F., Gerdt, V.P., Plesken, W., Robertz, D., 2003. The MAPLE package Janet : I. Polynomial systems. II. Linear partial differential equations. In: Proc. 6th Int. Workshop on Computer Algebra in Scientific Computing, Passau, Germany. pp. 31–40 and 41–54 (http://wwwb.math.rwth-aachen.de/Janet).

Blinkov, Y.A., Gerdt, V.P., Robertz, D., 2010a. GINV-project. Gröbner bases constructed by involutive algorithms. http://invo.jinr.ru/ginv/index.html.

Blinkov, Y.A., Gerdt, V.P., Robertz, D., 2010b. GINV-project benchmarks. Standard benchmarks between Gröbner bases systems. http://cag.jinr.ru/wiki/Benchmarks.

Boulier, F., 2004–2009. BLAD: Bibliothéques Lilloises d'Algébre Différentielle (http://www.lifl.fr/~boulier/BLAD/).

Boulier, F., 2007. Differential elimination and biological modelling. In: Gröbner Bases in Symbolic Analysis. In: Radon Ser. Comput. Appl. Math., vol. 2. Walter de Gruyter, Berlin, pp. 109–137.

Boulier, F., Hubert, E., 1996–2004. DIFFALG: description, help pages and examples of use. Symbolic Computation Group, University of Waterloo, Ontario, Canada (http://www-sop.inria.fr/members/Evelyne.Hubert/diffalg/).

Boulier, F., Lazard, D., Ollivier, F., Petitot, M., 1995. Representation for the radical of a finitely generated differential ideal. In: ISSAC'95: Proceedings of the 1995 International Symposium on Symbolic and Algebraic Computation. ACM Press, New York, NY, USA, pp. 158–166. http://hal.archives-ouvertes.fr/hal-00138020.

Boulier, F., Lazard, D., Ollivier, F., Petitot, M., 2009. Computing representations for radicals of finitely generated differential ideals. Appl. Algebra Engrg. Comm. Comput. 20 (1), 73–121. URL http://dx.doi.org/10.1007/s00200-009-0091-7.

Bourbaki, N., 1968. Elements of mathematics. In: Theory of Sets. Translated from the French. Hermann, Publishers in Arts and Science, Paris.

Bouziane, D., Kandri Rody, A., Maârouf, H., 2001. Unmixed-dimensional decomposition of a finitely generated perfect differential ideal. J. Symbolic Comput. 31 (6), 631–649. URL http://dx.doi.org/10.1006/jsco.1999.1562.

Buium, A., Cassidy, P.J., 1999. Differential algebraic geometry and differential algebraic groups: from algebraic differential equations to diophantine geometry. Kolchin (1999), pp. 567–636.

Chen, C., Golubitsky, O., Lemaire, F., Moreno Maza, M., Pan, W., 2007. Comprehensive triangular decomposition. In: Ganzha, V.G., Mayr, E.W., Vorozhtsov, E.V. (Eds.), CASC. In: Lecture Notes in Computer Science, vol. 4770. Springer, pp. 73–101.

Cox, D., Little, J., O'Shea, D., 1992. An introduction to computational algebraic geometry and commutative algebra. In: Ideals, Varieties, and Algorithms. In: Undergraduate Texts in Mathematics, Springer-Verlag, New York.

Delliére, S., 2000. D.M. Wang simple systems and dynamic constructible closure. Rapport de Recherche No. 2000–16 de l'Université de Limoges. URL http://www.unilim.fr/laco/rapports/2000/R2000_16.pdf.

Diop, S., 1992. On universal observability. In: Proc. 31st Conference on Decision and Control, Tucson, Arizona.

Gerdt, V.P., 1999. Completion of linear differential systems to involution. In: Computer Algebra in Scientific Computing—CASC'99, Munich. Springer, Berlin, pp. 115–137.

Gerdt, V.P., 2002. On an algorithmic optimization in the computation of involutive bases. Program. Comput. Softw. 28 (2), 62–65. URL http://dx.doi.org/10.1023/A:1014816631983.

Gerdt, V.P., 2005. Involutive algorithms for computing Gröbner bases. In: Computational Commutative and Non-commutative Algebraic Geometry. In: NATO Sci. Ser. III Comput. Syst. Sci., vol. 196. IOS, Amsterdam, pp. 199–225.

Gerdt, V.P., 2008. On decomposition of algebraic PDE systems into simple subsystems. Acta Appl. Math. 101 (1–3), 39–51. URL http://dx.doi.org/10.1007/s10440-008-9202-x.

Gerdt, V.P., Blinkov, Y.A., 1998a. Involutive bases of polynomial ideals. Math. Comput. Simulation 45 (5–6), 519–541. Simplification of systems of algebraic and differential equations with applications.

Gerdt, V.P., Blinkov, Y.A., 1998b. Minimal involutive bases. Math. Comput. Simulation 45 (5–6), 543–560. Simplification of systems of algebraic and differential equations with applications.

Gerdt, V.P., Yanovich, D.A., 2006. Investigation of the effectiveness of involutive criteria for computing polynomial Janet bases. Program. Comput. Softw. 32 (3), 134–138. URL http://dx.doi.org/10.1134/S0361768806030030.

Gerdt, V.P., Yanovich, D.A., Blinkov, Y.A., 2001. Fast search for the Janet divisor. Program. Comput. Softw. 27 (1), 22–24. URL http://dx.doi.org/10.1023/A:1007130618376.

Gómez Diaz, T., 1994. Quelques applications de l'évaluation dynamique. Ph.D. Thesis, Univ. de Limoges, Limoges, France. http://www-igm.univ-mlv.fr/~teresa/dynev/.

Habicht, W., 1948. Eine Verallgemeinerung des Sturmschen Wurzelzählverfahrens. Comment. Math. Helv. 21, 99–116.

Hubert, E., 2003a. Notes on triangular sets and triangulation–decomposition algorithms. I. Polynomial systems. In: Symbolic and numerical scientific computation (Hagenberg, 2001). In: Lecture Notes in Comput. Sci., vol. 2630. Springer, Berlin, pp. 1–39. URL http://dx.doi.org/10.1007/3-540-45084-X_1.

Hubert, E., 2003b. Notes on triangular sets and triangulation–decomposition algorithms. II. Differential systems. In: Symbolic and numerical scientific computation (Hagenberg, 2001). In: Lecture Notes in Comput. Sci., vol. 2630. Springer, Berlin, pp. 40–87. URL http://dx.doi.org/10.1007/3-540-45084-X_2.

Janet, M., 1929. Leçons sur les systèmes d'équations aux dérivées partielles. In: Cahiers Scientifiques IV. Gauthiers-Villars, Paris.

Kolchin, E.R., 1973. Differential Algebra and Algebraic Groups. In: Pure and Applied Mathematics, vol. 54. Academic Press, New York.

Kolchin, E.R., 1999. Selected Works of Ellis Kolchin with Commentary. American Mathematical Society, Providence, RI, Commentaries by Armand Borel, Michael F. Singer, Bruno Poizat, Alexandru Buium and Phyllis J. Cassidy, edited and with a preface by Hyman Bass, Buium and Cassidy.

Lemaire, F., Moreno Maza, M., Xie, Y., 2005. The RegularChains library in Maple. SIGSAM Bull. 39 (3), 96–97.

Li, Z., Wang, D., 1999. Coherent, regular and simple systems in zero decompositions of partial differential systems. Syst. Sci. Math. Sci. 12, 43–60.

Mishra, B., 1993. Algorithmic Algebra. In: Texts and Monographs in Computer Science, Springer-Verlag, New York.

Monagan, M., Pearce, R., 2009. Parallel sparse polynomial multiplication using heaps. In: ISSAC'09: Proceedings of the 2009 International Symposium on Symbolic and Algebraic Computation. ACM, New York, NY, USA, pp. 263–270.

Moreno Maza, M., 1999. On triangular decompositions of algebraic varieties. Tech. Rep., presented at the MEGA-2000 Conference.

Plesken, W., 1982. Counting with groups and rings. J. Reine Angew. Math. 334, 40–68. URL http://dx.doi.org/10.1515/crll.1982.334.40.

Plesken, W., 2009a. Counting solutions of polynomial systems via iterated fibrations. Arch. Math. (Basel) 92 (1), 44–56. URL http://dx.doi.org/10.1007/s00013-008-2785-7.

Plesken, W., 2009b. Gauss–Bruhat decomposition as an example of Thomas decomposition. Arch. Math. (Basel) 92 (2), 111–118. URL http://dx.doi.org/10.1007/s00013-008-2786-6.

Riquier, C., 1910. Les systèmes d'équations aux dérivées partielles. Gauthiers-Villars, Paris.

Ritt, J.F., 1950. Differential Algebra. In: American Mathematical Society Colloquium Publications, vol. XXXIII. American Mathematical Society, New York, NY.

Rosenfeld, A., 1959. Specializations in differential algebra. Trans. Amer. Math. Soc. 90, 394–407.

Seidenberg, A., 1958. Abstract differential algebra and the analytic case. Proc. Amer. Math. Soc. 9, 159–164.

Seidenberg, A., 1969. Abstract differential algebra and the analytic case II. Proc. Amer. Math. Soc. 23, 689–691.

Seiler, W.M., 2010. The formal theory of differential equations and its applications in computer algebra. In: Involution. In: Algorithms and Computation in Mathematics, vol. 24. Springer-Verlag, Berlin, URL http://dx.doi.org/10.1007/978-3-642-01287-7.

Thomas, J.M., 1937. Differential Systems. In: AMS Colloquium Publications, vol. XXI.

Thomas, J.M., 1962. Systems and Roots. The William Byrd Press, Inc., Richmond Virginia.

Wang, D., 1998. Decomposing polynomial systems into simple systems. J. Symbolic Comput. 25 (3), 295–314. URL http://dx.doi.org/10.1006/jsco.1997.0177.

Wang, D., 2001. Elimination Methods. In: Texts and Monographs in Symbolic Computation, Springer-Verlag, Vienna.

Wang, D., 2003. ϵpsilon: description, help pages and examples of use. (http://www-spiral.lip6.fr/~wang/epsilon/).

Wang, D., 2004. Elimination Practice. Imperial College Press, London, Software tools and applications, With 1 CD-ROM (UNIX/LINUX, Windows).

Wu, W.-T., 2000. Mechanical geometry theorem-proving, mechanical geometry problem-solving and polynomial equations-solving. In: Mathematics Mechanization. In: Mathematics and its Applications, vol. 489. Kluwer Academic Publishers Group, Dordrecht.

Yap, C.K., 2000. Fundamental Problems of Algorithmic Algebra. Oxford University Press, New York.