An Automated Confluence Proof for an Infinite Rewrite System via a Gröbner Basis Computation

Loredana Tec*
Research Institute for Symbolic Computation, Johannes Kepler University,
Castle of Hagenberg, Austria 4032
ltec@risc.uni-linz.ac.at

1 Introduction

In this paper we present an *automated proof* for the confluence of a rewrite system for integro-differential operators (given in Table 1). We also outline a generic prototype implementation of the integro-differential polynomials—the key tool for this proof—realized using the Theorema system. With its generic functor mechanism—detailed in Section 2—we are able to provide a formalization of the theory of integro-differential algebras. This formalization via functors also allows us to compute in such domains, and in this way to explore the new theory by computational experiments.

Integro-differential operators (introduced in [21]) are useful for solving linear boundary problems [25] given by a differential equation with a symbolic right-hand side along with boundary conditions. Boundary problems are of major importance in applications but they are usually treated numerically. A symbolic method for solving such problems was introduced in [20] for the constant coefficients case and extended to a general differential algebra setting in [21].

A very simple example of a boundary problem is the following: Given $f \in C^\infty[0,1]$, find $g \in C^\infty[0,1]$ such that

$$\boxed{\begin{aligned} &u'' = f, \\ &u(0) = u(1) = 0. \end{aligned}}$$

More generally, we consider linear boundary problems of the following type: Given $f \in \mathscr{G}$, find $u \in \mathscr{F}$ such that

$$\boxed{\begin{aligned} &Tu = f, \\ &\beta_1(u) = \cdots = \beta_n(u) = 0, \end{aligned}} \tag{1}$$

where

$$T = \partial^n + c_{n-1}\partial^{n-1} + \cdots + c_0$$

is a monic (i.e. having the leading coefficient 1) differential operator of order n and ∂ denotes the derivation. For the spaces $\mathscr{G}, \mathscr{F}$ we could for example choose $\mathscr{F} = \mathscr{G} = C^\infty[a,b]$ as real or complex vector spaces. The boundary conditions $\beta_1, \ldots, \beta_n$ can be arbitrary functionals; in particular, point evaluations of derivatives, but also more general boundary conditions of the form

$$\beta(u) = \sum_{i=0}^{n-1} a_i u^{(i)}(a) + b_i u^{(i)}(b) + \int_a^b v(\xi)\, u(\xi)\, d\xi$$

with $v \in \mathscr{F}$, called "Stieltjes boundary conditions".

A boundary problem is called *regular* if for each f there exists a unique solution u satisfying (1). For such problems there exists a unique linear operator $G\colon \mathscr{G} \to \mathscr{F}$ called "Green's operator" mapping

*Recipient of a DOC-fFORTE-fellowship of the Austrian Academy of Sciences

f to its unique solution $u = Gf$. The Green's operator for the above example is the noncommutative polynomial

$$XAX + XBX - AX - XB,$$

in the indeterminates $A = \int_0^x$, $B = \int_x^1$ and X, where multiplication corresponds to composition of operators.

The importance of the integro-differential operators consists in the fact that they can express both the problem statement (differential equation and boundary conditions) and its solution operator (Green's operator). The rewrite system used for their construction describes the interaction between differentiation and integration, by capturing the basic axioms that involve these operations.

To build up the algebra of integro-differential operators, we first consider the noncommutative polynomials in the corresponding operators and then we factor out the system of rewrite rules given in Table 1. The goal of the paper is to give an automated proof that this is a Noetherian and confluent rewrite system (see Theorem 1 in Section 4). There is a close relationship between rewriting and Gröbner bases, where the analogue of the Knuth-Bendix procedure—detailed in [2, 28]—is Buchberger's algorithm. The goal of the former is to transform a set of equations (over terms) into a Noetherian and confluent term rewrite system. In fact, we prove the confluence of the rewrite system by equivalently proving that it corresponds to a Gröbner basis in this noncommutative algebra. *Gröbner bases* were introduced in [5, 6] and have became a crucial tool in computer algebra, specifically for solving several algebraic problems concerning ideals [8], see for instance [13, 14] for the commutative setting, and [4, 12, 17] for the noncommutative setting. A detailed presentation of the proof is given in [24]; see also [26] for a summary.

2 The Theorema System

Theorema is a system designed as an integrated environment for doing mathematics, in particular proving, solving, and computing in various domains of mathematics [10]. Implemented on top of the computer algebra system Mathematica, its core language is higher-order predicate logic and contains a natural programming language such that algorithms can be coded and verified in a unified formal frame.

In this logic-internal programming language, *functors* are a powerful tool for building up hierarchical domains in mathematics in a modular and generic way that unites elegance and formal clarity. They were introduced and first implemented in Theorema by Bruno Buchberger.

The notion of functor in Theorema is akin to functors in ML, not to be confused with the functors of category theory. From a computational point of view, a Theorema functor is a higher-order function that produces a new domain from given domains, where each domain is considered as a bundle of operations (including carrier predicates). Operations in the new domain are defined in terms of operations in the underlying domains. Apart from this computational aspect, functors also have an important reasoning aspect—a functor transports properties of the input domains to properties of the output domain, typical examples being the various "preservation theorems" in mathematics: "If R is a ring, then $R[x]$ is also a ring". This means the functor $R \mapsto R[x]$ preserves the property of being a ring, in other words: it goes from the "category of rings" to itself. In this context, a category is simply a collection of domains characterized by a common property (a higher-order predicate on domains). See below for an example of a functor named `LexWords`. It takes a linearly ordered alphabet `L` as input domain and builds the word monoid over this alphabet:

$$
\begin{array}{l}
\textbf{Definition}\left[\text{"Word Monoid"}, \textbf{any}[L],\right. \\
\quad \textbf{LexWords}[L] = \textbf{Functor}\left[W, \textbf{any}[v, w, \xi, \eta, \bar{\xi}, \bar{\eta}],\right. \\
\qquad \frac{\mathbb{s} = \langle\rangle}{} \\
\qquad \underset{W}{\in}[w] \Leftrightarrow \bigwedge \left\{ \begin{array}{l} \textbf{is-tuple}[w] \\ \underset{i=1,..,|w|}{\forall} \underset{L}{\in}[w_i] \end{array} \right. \\
\qquad \underset{W}{\square} = \langle\rangle \\
\qquad v \underset{W}{*} w = v \asymp w \\
\qquad \left(\langle \eta, \bar{\eta} \rangle \underset{W}{>} \langle\rangle\right) \Leftrightarrow \textbf{True} \\
\qquad \left(\langle\rangle \underset{W}{>} \langle \bar{\eta} \rangle\right) \Leftrightarrow \textbf{False} \\
\qquad \left(\langle \eta, \bar{\eta} \rangle \underset{W}{>} \langle \xi, \bar{\xi} \rangle\right) \Leftrightarrow \bigvee \left\{ \begin{array}{l} \eta \underset{L}{>} \xi \\ (\eta = \xi) \bigwedge \langle \bar{\eta} \rangle \underset{W}{>} \langle \bar{\xi} \rangle \end{array} \right.
\end{array}
\right]\right]
$$

Here $\bar{\xi}$, $\bar{\eta}$ are sequence variables, i.e. they can be instantiated with finite sequences of terms. The new domain `W` has the following operations: `W`$[\in]$ denotes the carrier predicate, the neutral element is given by `W`$[\square]$, the multiplication `W`$[*]$ is defined as concatenation, and `W`$[>]$ defines the lexicographic ordering on `W`.

$$
\begin{array}{l}
\textbf{Definition}\left[\text{"Free Module"}, \textbf{any}[K, W],\right. \\
\quad \textbf{FreeModule}[K, W] = \textbf{Functor}\left[V, \textbf{any}[c, d, x, y, \xi, \eta, A, \bar{x}, \bar{y}],\right. \\
\qquad \frac{\mathbb{s} = \langle\rangle}{} \\
\qquad \underset{V}{\in}[x] \Leftrightarrow \left(\bigwedge \left\{ \begin{array}{l} \textbf{is-tuple}[x] \\ \underset{i=1,..,|x|}{\forall} \left(\bigwedge \left\{ \begin{array}{l} \textbf{is-tuple}[x_i] \\ |x_i| = 2 \\ \underset{V}{\textbf{is-coeff}}[(x_i)_1] \\ \underset{V}{\textbf{is-bvec}}[(x_i)_2] \end{array} \right. \right) \\ \underset{i=1,..,|x|-1}{\forall} (x_i)_2 \underset{W}{>} (x_{i+1})_2 \end{array} \right. \right) \\
\qquad \frac{\underset{V}{\textbf{is-bvec}}[\xi] \Leftrightarrow \underset{W}{\in}[\xi]}{} \\
\qquad \underset{V}{\textbf{is-coeff}}[c] \Leftrightarrow \left(\underset{K}{\in}[c] \bigwedge c \neq \underset{K}{0}\right) \\
\qquad \ldots \\
\qquad \frac{\underset{V}{0} = \langle\rangle}{} \\
\qquad \langle\rangle \underset{V}{+} y = y \\
\qquad x \underset{V}{+} \langle\rangle = x \\
\qquad \langle\langle c, \xi\rangle, \bar{x}\rangle \underset{V}{+} \langle\langle d, \eta\rangle, \bar{y}\rangle = \left\{ \begin{array}{lll} \langle c, \xi\rangle \frown \left(\langle \bar{x}\rangle \underset{V}{+} \langle\langle d, \eta\rangle, \bar{y}\rangle\right) & \Leftarrow & \xi \underset{W}{>} \eta \\ \langle d, \eta\rangle \frown \left(\langle\langle c, \xi\rangle, \bar{x}\rangle \underset{V}{+} \langle \bar{y}\rangle\right) & \Leftarrow & \eta \underset{W}{>} \xi \\ \left\langle c \underset{K}{+} d, \xi\right\rangle \frown \left(\langle \bar{x}\rangle \underset{V}{+} \langle \bar{y}\rangle\right) & \Leftarrow & (\xi = \eta) \bigwedge c \underset{K}{+} d \neq \underset{K}{0} \\ \langle \bar{x}\rangle \underset{V}{+} \langle \bar{y}\rangle & \Leftarrow & \textbf{otherwise} \end{array} \right. \\
\qquad \underset{V}{-}\langle\rangle = \langle\rangle \\
\qquad \frac{\underset{V}{-}\langle\langle c, \xi\rangle, \bar{x}\rangle = \left\langle \underset{K}{-} c, \xi\right\rangle \frown \left(\underset{V}{-}\langle \bar{x}\rangle\right)}{} \\
\qquad x \underset{V}{-} y = x \underset{V}{+} \left(\underset{V}{-} y\right) \\
\qquad \underset{V}{0} \cdot y \text{(* scalar multiplication *)} = \langle\rangle \\
\qquad \frac{x \underset{V}{\cdot} 0 = \langle\rangle}{} \\
\qquad \ldots \\
\qquad c \underset{V}{\cdot} \langle\langle d, \eta\rangle, \bar{y}\rangle = \left\langle c \underset{K}{*} d, \eta\right\rangle \frown c \underset{V}{\cdot} \langle \bar{y}\rangle \\
\qquad \langle\langle c, \xi\rangle, \bar{x}\rangle \underset{V}{\cdot} d = \left\langle c \underset{K}{*} d, \xi\right\rangle \frown \langle \bar{x}\rangle \underset{V}{\cdot} d
\end{array}
\right]\right]
$$

In the following, we illustrate one way of *building up polynomials* in Theorema starting from the base categories of fields with ordering and ordered monoids. Via the functor `FreeModule`, we construct first the free vector space `V` over a field `K` generated by the set of words in an ordered monoid `W`. The elements of `V` are described by `V`$[\in]$ as lists of pairs, each pair containing one (non-zero) coefficient from `K` and one basis vector from `W`, where the basis vectors are ordered according to the ordering on `W`. The

operations of addition, subtraction and scalar multiplication are defined recursively, using the operations on `K` and `W`. By the `MonoidAlgebra` functor we extend this domain, introducing a multiplication using the corresponding operations in `K` and `W`:

```
MonoidAlgebra[K, W] = where[V = FreeModule[K, W],
  Functor[P, any[c, d, f, g, ξ, η, m̄, n̄],
    𝕤 = ⟨⟩
    ...(* linear operations from V *)
    (* multiplication *)
    ⟨⟩ *_P g = ⟨⟩
    f *_P ⟨⟩ = ⟨⟩
    ⟨⟨c, ξ⟩, m̄⟩ *_P ⟨⟨d, η⟩, n̄⟩ = ⟨⟨c *_K d, ξ *_W η⟩⟩ +_P ⟨⟨c, ξ⟩⟩ *_P ⟨n̄⟩ +_P ⟨m̄⟩ *_P ⟨⟨d, η⟩, n̄⟩
```

The new domain inherits the structure on the elements of `V`.

The main advantage of the above construction is that it is fully *generic*: Not only can it be instantiated for different coefficient rings (or fields) and different sets of indeterminates, it comprises also the commutative and noncommutative case (where `W` is instantiated respectively by a commutative and noncommutative monoid).

3 Differential Algebras, Operators and Polynomials

We first recall the structures of differential operators and polynomials; see for example [27, 15] for more details. Let $(\mathscr{F},\partial)$ be a commutative differential algebra over a field K, so $\partial\colon \mathscr{F}\to\mathscr{F}$ is a K-linear map satisfying the Leibniz rule

$$\partial(fg) = g\partial(f) + f\partial(g).$$

In the following we will use the notation f' for $\partial(f)$.

The differential operators are usually defined over a differential algebra $(\mathscr{F},\partial)$ directly in terms of normal forms

$$\sum_{i=0}^{n} f_i\partial^i,$$

with $f_i\in\mathscr{F}$ and multiplication coming from $f\partial = \partial f - f'$. An example of a differential operator over $(K[x],\partial)$ is given by

$$(3x^2+5x+4)\partial^8 + 2x^3\partial^5 + (4x+7)\partial.$$

But alternatively, one can construct them via the K-algebra generated by the symbol ∂ and the functions $f\in\mathscr{F}$, factoring out the rules

$$\partial f \to \partial\cdot f + f\partial \quad\text{and}\quad fg\to f\cdot g \tag{2}$$

derived from the Leibniz rule, where $\cdot$ denotes the action. This rewrite system certainly terminates, since in the last rule the length is decreasing and for the first rule we choose $\partial > f$. In order to prove confluence, we have to consider the minimal overlap ∂fg of the two rules (2); this can be reduced in two ways as:

$$\begin{array}{ccc} & \partial fg & \\ \swarrow & & \searrow \\ (\partial\cdot f)g - (f\partial)g & & \partial(f\cdot g) \end{array}$$

with the S-polynomial

$$\mathrm{Spol}(fg - f\cdot g, \partial f - \partial\cdot f - f\partial) = \partial(fg) - (\partial f)g = (fg)\partial + (fg)' - f(\partial g) - f'g = (fg)' - f'g - fg',$$

which is 0 for all f and g because of the Leibniz rule. To prove this automatically, one would have to compute with generic functions f and g. The appropriate algebraic setting for this is the algebra of differential polynomials.

The differential polynomials, denoted by $\mathscr{F}\{u\}$, are usually defined as the ring in infinitely many indeterminates $u, u', u'', \ldots$, and derivation via the Leibniz rule, a typical example being

$$(3x^3 + 5x)\, u^{(1,0,3,2)} + 7\partial(x^5)u^{(2,0,1)} + 2xu^{(1,1)}$$

when $\mathscr{F} = K[x]$. Here we use the multi-index notation $u^{(1,0,3,2)} = u(u'')^3(u''')^2$.

They can be constructed, alternatively, by adjoining an indeterminate u to a differential algebra $(\mathscr{F}, \partial)$: consider all terms that one can build up using the indeterminate u, coefficients from $\mathscr{F}$ and the operations $+, \cdot, \partial$. By applying the Leibniz rule and the axioms of K, one obtains K-linear combinations of terms of the form

$$f \prod_{i=0}^{\infty} u_i^{\beta_i} \tag{3}$$

as canonical forms for differential polynomials, where $f \in \mathscr{F}$, u_n is short for $\partial^n(u)$, and only finitely many $\beta_i \in \mathbb{N}$ are nonzero. This is a special case of the general construction of polynomials in universal algebra. For a comprehensive treatment of this notion of polynomials for arbitrary algebras we refer to [16]; useful surveys can be found in [1, 11].

4 Integro-Differential Operators and Polynomials

In this section we present two algebraic structures introduced in [21] for treating differential and integral operators simultaneously, as extensions of the previously outlined differential structures.

An *integro-differential algebra* $(\mathscr{F}, \partial, \int)$ is a differential algebra $(\mathscr{F}, \partial)$ with a linear operation $\int : \mathscr{F} \to \mathscr{F}$ such that $(\int f)' = f$, and the following axiom holds:

$$(\int f')(\int g') + \int (fg)' = (\int f')g + f(\int g').$$

A standard example is $\mathscr{F} = C^\infty(\mathbb{R})$, where ∂ is the usual derivation and $\int$ the integral operator

$$\textstyle\int_a^x : f \mapsto \int_a^x f(\xi)\, d\xi$$

for $a \in \mathbb{R}$. The first axiom corresponds to the Fundamental Theorem of Calculus, while the second axiom captures integration by parts. In any integro-differential algebra, one can define an *evaluation* $E = 1 - \int \partial$, which corresponds to $f(a)$ in the above example. See [21] for further examples and [19] for a short summary.

Translating the axioms for integro-differential algebras and some of their consequences, one obtains the rewrite rules of Table 1 for the corresponding operators. For instance, from $(\int f)' = f$ one can easily obtain the rule $\partial \int \to 1$ as:

$$(\partial \int)(f) = \partial(\int f) = f.$$

Also the Leibniz rule translates immediately into a rule for the corresponding operators:

$$\partial f \to \partial \cdot f + f\partial.$$

Introduced in [21] as a generalization of "Green's polynomials" [20], the *integro-differential operators* are an algebraic analogue of differential, integral and boundary operators in the context of linear ordinary

$fg \to f \cdot g$	$\partial f \to \partial \cdot f + f\partial$	$\int f \int \to (\int \cdot f)\int - \int(\int \cdot f)$
$\varphi\psi \to \psi$	$\partial\varphi \to 0$	$\int f\partial \to f - \int(\partial \cdot f) - (\mathtt{E} \cdot f)\,\mathtt{E}$
$\varphi f \to (\varphi \cdot f)\,\varphi$	$\partial\int \to 1$	$\int f\varphi \to (\int \cdot f)\,\varphi$

Table 1: Rewrite System for $\mathscr{F}[\partial, \int]$

differential equations (LODEs). Methods for solving and factoring boundary problems are described in [18, 22], both in a differential algebra and in an abstract setting. See also [23] for an overview of operations on linear boundary problems in a symbolic framework.

The integro-differential operators are realized by a suitable quotient of noncommutative polynomials over a given integro-differential algebra. They are built as an instance of the monoid algebra [3] for the word monoid over the infinite alphabet consisting of the letters ∂ and $\int$ along with all basis elements $x^n e^{\lambda x}$ $(n \in \mathbb{N}, \lambda \in \mathbb{C})$ of the exponential polynomials and all point evaluations corresponding to characters denoted by φ and ψ, including the evaluation $E = 1 - \int\partial$. Then the nine parametrized rewrite rules (see Table 1) are factored out. As mentioned in the Introduction, our goal is to provide an automated proof of the following theorem:

Theorem 1. *The above rewrite system is noetherian and confluent.*

The proof of confluence is given in Section 5. Noetherianity is ensured by choosing a semigroup ordering $>$ that is compatible with the reduction rules: we choose $\partial > f$ for all $f \in \mathscr{F}$ and extend this to words by the graded lexicographic construction. The ordering is compatible with the rewrite system since all rules reduce the word length except for the Leibniz rule, which is compatible because $\partial > f$.

In analogy to the algebra of differential polynomials (see Section 3), the algebra of *integro-differential polynomials* is realized as a special case of the general construction of polynomials in universal algebra, by adjoining one indeterminate function to an integro-differential algebra. The integro-differential polynomials model nonlinear differential and integral operators with an indeterminate u. They can also be seen as polynomial functions (of x) involving an "unknown" function (namely u). A typical integro-differential polynomial is given by

$$4u^2 \int u'^3 + \int (x^6 u u''^5 \int (x^2 e^{4x} u^3 u'^2 \int u^4)).$$

Every integro-differential polynomial can be represented as K-linear combinations of terms of the form

$$f u(0)^\alpha u^\beta \int f_1 u^{\gamma_1} \int \cdots \int f_n u^{\gamma_n}, \tag{4}$$

where $f, f_1, \ldots, f_n \in \mathscr{F}$, and each multi-index as well as n may be zero. Note that these are not yet canonical forms, since for example terms like

$$f \int \lambda g u \quad \text{and} \quad \lambda f \int g u$$

with $f, g \in \mathscr{F}$ and $\lambda \in K$ represent the same polynomial. The crucial fact for computing with integro-differential polynomials is expressed in the following theorem.

Theorem 2. *The integro-differential polynomials $\mathscr{F}\{u\}$ constitute an integro-differential algebra with an algorithmic canonical simplifier.*

The implementation of integro-differential polynomials in Theorema is realized using the functor hierarchy presented in Section 3. The multi-index representation u^β for terms of the form (3) is realized

by the monoid $\mathbb{N}^*$ of natural tuples with finitely many nonzero entries, generated by a functor named `TuplesMonoid`. The nested integrals $\int f_1 u^{\gamma_1} \int \cdots \int f^n u^{\gamma_n}$ are represented as lists of pairs of the form $\langle f_k, \gamma_k \rangle$, with $f_k \in \mathscr{F}$ and $\gamma_k \in \mathbb{N}^*$. The terms of the form (4) are then constructed via a cartesian product of monoids as follows:

```
Definition["Term Monoid for IDP", any[ℱ, N],
 TermMonoid[ℱ, N] = TuplesMonoid[N] × TuplesMonoid[N] × TuplesMonoid[ℱ × TuplesMonoid[N]]]
```

Using this construction, the integro-differential polynomials are built up by the `FreeModule`$[\mathscr{F},\mathtt{B}]$ functor that constructs the $\mathscr{F}$-module with basis `B`. It is instantiated with $\mathscr{F}$ being a given integro-differential algebra and `B` the term monoid just described. We will equip this domain with the operations of multiplication, differentiation and integration, using a functor named `IntDiffPol`$[\mathscr{F},\mathtt{K}]$. We have also implemented a canonical simplifier that identifies different expressions denoting the same integro-differential polynomial.

5 A Proof of Confluence

We show the confluence of the rewrite system of Table 1 by equivalently showing that the polynomials given by the difference between the left-hand and right-hand side of these rules form a noncommutative Gröbner basis in the underlying integro-differential algebra. Every such rule is infinitely parametrized, thus the set of polynomials generates an infinite noncommutative polynomial ideal. Consequently, we need an algorithmic way to handle parametrized polynomial reduction and S-polynomials. For this purpose we make use of a noncommutative adaption of reduction rings (rings with so-called *reduction multipliers* in the sense of [7, 9]).

In order to prove that this set of noncommutative polynomials represents a Gröbner basis for the infinite polynomial ideal they generate, we have to show that all the corresponding S-polynomials reduce to 0. A suitable domain where these S-polynomials may be computed can be realized using the algebra of integro-differential polynomials in two unknown functions. Hence, for the S-polynomial computations we use the functor

$$\mathtt{IntDiffPolys}[\mathtt{IntDiffPolys}[\mathscr{F},\mathtt{K}],\mathtt{K}]$$

of integro-differential operators over the algebra of integro-differential polynomials in two indeterminate functions u and v, that produces a domain denoted below by $\mathbb{G}$. The proof reduces to computations in this domain where all the 72 parametrized S-polynomials are reduced to 0. The corresponding Theorema command is given below.

```
TS_In[201]:= GB_𝔊[⟨-u⟨1⟩ v⟨1⟩ + u⟨1⟩ · v⟨1⟩, -"E" + "E" "E", "E" u⟨1⟩ - u[0]⟨1⟩ "E",
                  "∂" u⟨1⟩ - u⟨0,1⟩ - u⟨1⟩ "∂", "∂" "E", -1 + "∂" "∫", "∫" u⟨1⟩ "∫" + "∫" "∫" u⟨1⟩ - "∫" u⟨1⟩ "∫",
                  "∫" u⟨0,1⟩ + "∫" u⟨1⟩ "∂" - u⟨1⟩ + u[0]⟨1⟩ "E", "∫" u⟨1⟩ "E" - "∫" u⟨1⟩ "E"⟩]
TS_Out[201]=
⟨-u⟨1⟩ v⟨1⟩ + u⟨1⟩  v⟨1⟩, -E + E E, E u⟨1⟩  - u[0]⟨1⟩  E, ∂ u⟨1⟩  - u⟨0,1⟩  - u⟨1⟩  ∂, ∂ E, -1 + ∂ ∫,
 ∫ u⟨1⟩  ∫ + ∫ ∫ u⟨1⟩  - ∫ u⟨1⟩  ∫, ∫ u⟨0,1⟩  + ∫ u⟨1⟩  ∂ - u⟨1⟩  + u[0]⟨1⟩  E, ∫ u⟨1⟩  E - ∫ u⟨1⟩  E⟩
```

Here $\mathtt{Gb}_{\mathbb{G}}$ is the function that computes a Gröbner basis in the domain $\mathbb{G}$; the computation takes approximately 1min. Since the input remains unchanged, we conclude that the rewrite rules of Table 1 correspond to a Gröbner basis. Thus, the corresponding rewrite system is confluent.

6 Conclusion

We provide a new automated proof for the confluence of an infinite rewrite system, parametrized over integro-differential algebras. For this purpose we use a generic implementation—with the functor lan-

guage of the Theorema system—of the integro-differential structures introduced in [21]. This approach could also be generalized to deal with other parametrized rewrite systems for algebraic structures with linear operators satisfying certain axioms. Moreover, the functor approach allows us to explore and do computational experiments within such new domains.

References

[1] E. Aichinger and G. F. Pilz. A survey on polynomials and polynomial and compatible functions. In *Proceedings of the Third International Algebra Conference (Tainan, 2002)*, pages 1–16, Dordrecht, 2003. Kluwer Acad. Publ.

[2] F. Baader and T. Nipkow. *Term rewriting and all that*. Cambridge University Press, Cambridge, 1998.

[3] T. Becker and V. Weispfenning. *Gröbner bases*, volume 141 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 1993. A computational approach to commutative algebra, In cooperation with Heinz Kredel.

[4] G. M. Bergman. The diamond lemma for ring theory. *Adv. in Math.*, 29(2):178–218, 1978.

[5] B. Buchberger. *An algorithm for finding the bases elements of the residue class ring modulo a zero dimensional polynomial ideal (German)*. PhD thesis, Univ. of Innsbruck, 1965. English translation J. Symbolic Comput. **41**(3-4) (2006) 475–511.

[6] B. Buchberger. Ein algorithmisches Kriterium für die Lösbarkeit eines algebraischen Gleichungssystems. *Aequationes Math.*, 4:374–383, 1970.

[7] B. Buchberger. A critical-pair/completion algorithm for finitely generated ideals in rings. In *Logic and machines: decision problems and complexity (Münster, 1983)*, volume 171 of *Lecture Notes in Comput. Sci.*, pages 137–161. Springer, Berlin, 1984.

[8] B. Buchberger. Introduction to Gröbner bases. In B. Buchberger and F. Winkler, editors, *Gröbner bases and applications*. Cambridge Univ. Press, 1998.

[9] B. Buchberger. Groebner rings and modules. In S. Maruster, B. Buchberger, V. Negru, and T. Jebelean, editors, *Proceedings of SYNASC 2001*, pages 22–25, 2001.

[10] B. Buchberger, A. Craciun, T. Jebelean, L. Kovacs, T. Kutsia, K. Nakagawa, F. Piroi, N. Popov, J. Robu, M. Rosenkranz, and W. Windsteiger. Theorema: Towards computer-aided mathematical theory exploration. *J. Appl. Log.*, 4(4):359–652, 2006.

[11] B. Buchberger and R. Loos. Algebraic simplification. In *Computer algebra*, pages 11–43. Springer, Vienna, 1983.

[12] J. Bueso, J. Gómez-Torrecillas, and A. Verschoren. *Algorithmic methods in non-commutative algebra*, volume 17 of *Mathematical Modelling: Theory and Applications*. Kluwer Academic Publishers, Dordrecht, 2003. Applications to quantum groups.

[13] D. Cox, J. Little, and D. O'Shea. *Ideals, varieties, and algorithms*. Undergraduate Texts in Mathematics. Springer, New York, third edition, 2007. An introduction to computational algebraic geometry and commutative algebra.

[14] G.-M. Greuel and G. Pfister. *A* **Singular** *introduction to commutative algebra*. Springer, Berlin, extended edition, 2008. With contributions by Olaf Bachmann, Christoph Lossen and Hans Schönemann, With 1 CD-ROM (Windows, Macintosh and UNIX).

[15] E. Kolchin. *Differential algebra and algebraic groups*, volume 54 of *Pure and Applied Mathematics*. Academic Press, New York-London, 1973.

[16] H. Lausch and W. Nöbauer. *Algebra of polynomials*, volume 5 of *North-Holland Mathematical Library*. North-Holland Publishing Co., Amsterdam, 1973.

[17] H. Li. *Noncommutative Gröbner bases and filtered-graded transfer*, volume 1795 of *Lecture Notes in Mathematics*. Springer-Verlag, Berlin, 2002.

[18] G. Regensburger and M. Rosenkranz. An algebraic foundation for factoring linear boundary problems. *Ann. Mat. Pura Appl. (4)*, 188(1):123–151, 2009.

[19] G. Regensburger, M. Rosenkranz, and J. Middeke. A skew polynomial approach to integro-differential operators. In *Proceedings of ISSAC '09*, New York, 2009. ACM. to appear.

[20] M. Rosenkranz. A new symbolic method for solving linear two-point boundary value problems on the level of operators. *J. Symbolic Comput.*, 39(2):171–199, 2005.

[21] M. Rosenkranz and G. Regensburger. Integro-differential polynomials and operators. In D. Jeffrey, editor, *Proceedings of ISSAC '08*, pages 261–268, New York, 2008. ACM.

[22] M. Rosenkranz and G. Regensburger. Solving and factoring boundary problems for linear ordinary differential equations in differential algebras. *J. Symbolic Comput.*, 43(8):515–544, 2008.

[23] M. Rosenkranz, G. Regensburger, L. Tec, and B. Buchberger. A symbolic framework for operations on linear boundary problems. In *CASC '09: Proceedings of the 11th International Workshop on Computer Algebra in Scientific Computing*, pages 269–283, Berlin, Heidelberg, 2009. Springer-Verlag.

[24] M. Rosenkranz, G. Regensburger, L. Tec, and B. Buchberger. Symbolic analysis for boundary problems: From rewriting to parametrized Gröbner bases. Technical Report 2010-05, RICAM, 2010.

[25] I. Stakgold. *Green's functions and boundary value problems*. John Wiley & Sons, New York, 1979.

[26] L. Tec, G. Regensburger, M. Rosenkranz, and B. Buchberger. An automated confluence proof for an infinite rewrite system parametrized over an integro-differential algebra. 2010. Proceedings of ICMS 2010, LNCS, to appear.

[27] M. van der Put and M. F. Singer. *Galois theory of linear differential equations*. Springer, Berlin, 2003.

[28] F. Winkler. The Church-Rosser Property in Computer Algebra and Special Theorem Proving: An Investigation of Critical-Pair/Completion Algorithms, 1984. Ph.D. dissertation.