OreTools: a computer algebra library for univariate Ore polynomial rings

S. A. Abramov[*]
Dorodnicyn Computing Centre
Russian Academy of Science
Moscow, Russia
abramov@ccas.ru

H. Q. Le[†]
Symbolic Computation Group
University of Waterloo
Waterloo, Canada
hqle@scg.math.uwaterloo.ca

Ziming Li[‡]
Symbolic Computation Group
University of Waterloo
Waterloo, Canada
z6li@scg.uwaterloo.ca

Abstract

This document presents the OreTools package which provides tools for performing basic arithmetic in Ore algebra. These tools can be used as a basis for various well-known algorithms in Ore algebra, in particular in differential and difference algebras.

1 Introduction

Pseudo–linear algebra is the study of the common properties and the common mathematical abstraction of linear functional equations such as differential and difference equations. The objects of the study of pseudo–linear algebra are skew polynomials [12] which represent single equations and pseudo-linear operators which represent systems.

[*]Partially supported by the French-Russian Lyapunov Institute under grant 98-03.

[†]Partially supported by Natural Sciences and Engineering Research Council of Canada Grant No. CRD215442-98.

[‡]Partially supported by Natural Sciences and Engineering Research Council of Canada Grant No. CRD215442-98.

Various algebraic algorithms have been generalized to arbitrary pseudo–linear equations: algorithms to uncouple systems of linear functional equations [7, 5], to construct the minimal annihilator [2], to solve the accurate integration problem [1], and to find a d'Alembertian solution of linear functional equations [3], to name a few.

The focus of this document is on the description of the `OreTools` package. It is organized in the following manner. In Section 2, we give an overview of pseudo–linear algebra (or Ore algebra). Section 5 describes the `OreTools` package where we discuss the proposed functionalities, and the implementation details of the package. The section concludes with an example which shows how to use the package to solve the accurate integration problem. In Section 6, we show that the co-existence of the two packages `Ore_algebra` and `OreTools` is necessary.

2 Univariate Ore polynomials, Ore operators, and Ore equations

The objects in the study of Ore algebra are Ore polynomials, Ore operators, and Ore equations. In this section we give an overview of these objects. See [12, 9, 6] for a detailed discussion on them.

2.1 Univariate Ore polynomials

Let k be a field of characteristic 0, and $\sigma : k \longrightarrow k$ be an automorphism of k.

Definition 2.1 *A derivation w.r.t. σ is any map $\delta : k \longrightarrow k$ satisfying*

$$\delta\,(a+b) = \delta\,a + \delta\,b \text{ and } \delta\,(ab) = \sigma(a)\,\delta\,b + \delta a\,b, \quad \text{for any } a, b \in k. \tag{1}$$

Lemma 2.1 *Let δ be a derivation of k w.r.t. σ.*

(i) *If $\sigma \neq 1_k$ then there is an element $\alpha \in k$ such that $\delta = \alpha(\sigma - 1_k) = \delta_\alpha$.*

(ii) *If $\delta \neq 0$ then there is an element $\beta \in k$ such that $\sigma = \beta\,\delta + 1_k$.*

Definition 2.2 *A univariate Ore polynomial ring over k, given by σ and δ and denoted by $k[x; \sigma, \delta]$, is the ring $(k[x], +, \cdot)$ of polynomials in x over k with the usual polynomial addition, and the multiplication given by*

$$x\,a = \sigma(a)\,x + \delta a, \quad \text{for any } a \in k. \tag{2}$$

Elements of the ring $k[x;\sigma,\delta]$ are called Ore polynomials. It can be shown (see, for instance, [12, 6]) that $k[x;\sigma,\delta]$ possesses the right and left division algorithms.

Example 2.1 The differential polynomial ring is $k[D;1_k,\delta]$ where δ is the derivation w.r.t. the identity. For $k\equiv\mathbb{C}(n)$, the shift polynomial ring is $k[E;\sigma,0]$ where σ is the automorphism of k over $\mathbb{C}$ that takes n to $n+1$. For $k\equiv\mathbb{C}(q)(t)$, the q-shift polynomial ring is $k[Q;\sigma,0]$ where σ is the automorphism of k over $\mathbb{C}(q)$ that takes t to qt. The usual polynomial ring over k is $k[x;1_k,0]\equiv k[x]$.

2.2 Ore operators and Ore equations

Definition 2.3 *Let V be a vector space over k. A map $\theta:V\to V$ is k-pseudo-linear w.r.t. σ and δ if*

$$\theta(u+v)=\theta(u)+\theta(v),\ \theta(a\,u)=\sigma(a)\,\theta u+\delta a\,u \quad \textit{for any } a\in k,\ u,v\in K. \quad (3)$$

Let K be a σ,δ-compatible extension field of k, i.e., σ and δ can be extended to an automorphism of K, and a derivation of K w.r.t. σ, respectively.

Lemma 2.2 *For any $c\in K$, the map $\theta_c:K\to K$ given by*

$$\theta_c a = c\,\sigma(a)+\delta a \quad (4)$$

is K-pseudo-linear w.r.t. σ and δ. Conversely, for any K-pseudo-linear map $\theta:K\to K$ there is an element $c\in K$ such that $\theta=\theta_c$ as given in (4).

By the pseudo linearity of θ, $\theta a=\theta(a\,1)=\sigma(a)\,\theta 1+\delta a$. Hence, the converse of Lemma 2.2 is proven by setting $\theta=\theta_c$, where $c=\theta 1$. It follows that it is sufficient to specify σ,δ, and $\theta 1$ in order to define an Ore polynomial ring $(k[x],+,\cdot)$ and the k-pseudo-linear map θ w.r.t. σ and δ.

Note that any k-pseudo-linear map $\theta:V\to V$ induces an action $*_\theta$ $k[x;\sigma,\delta]\times V\to V$ given by

$$\left(\sum_{i=0}^{n} a_i\,x^i\right) *_\theta u=\sum_{i=0}^{n} a_i\,\theta^i u,\ \ u\in V,\ a_i\in k.$$

Hence, we can consider the ring of Ore operators of the form $p(x)*_\theta$, $p(x)\in k[x;\sigma,\delta]$. This ring, denoted by $k[\theta]$, have elements which are linear over the constants of k.

Definition 2.4 *An* Ore equation *is of the form $L(y)=0$ where $L\in k[\theta]$.*

2.3 Basic Arithmetics

Let $k[x;\sigma,\delta]$ be an Ore polynomial ring, $A, B \in k[x;\sigma,\delta]\setminus\{0\}$. By applying the right division algorithm, we obtain the relation

$$A = Q_1\, B + R_1,\ \ Q_1, R_1 \in k[x;\sigma,\delta],\ \deg\ R_1 < \deg\ B.$$

R_1, Q_1 are called the *right-remainder* and the *right-quotient* of A by B, respectively. Similarly, by applying the left division algorithm, we obtain the relation

$$A = B\, Q_2 + R_2,\ \ Q_2, R_2 \in k[x;\sigma,\delta],\ \deg\ R_2 < \deg\ B.$$

R_2, Q_2 are called the *left-remainder* and the *left-quotient* of A by B, respectively.

For given $A, B \in k[x;\sigma,\delta]$, one can now find the *Greatest Common Right Divisors* (GCRD) and the *Least Common Left Multiple* (LCLM) by using the extended right algorithms.

It follows from Example 2.1 that the usual polynomial ring $k[x]$ is a special case of Ore polynomial rings. Various efficient techniques in the commutative $k[x]$ have been generalized to the non-commutative $k[x;\sigma,\delta]$. They include .... In the next section, we describe ...

3 Improvements on LCM, GCD computation

4 Adjoints and their applications

Definition 4.1 [4] *Let $k[x;\sigma,\delta]$ be a skew-polynomial ring. The adjoint of $k[x;\sigma,\delta]$ is defined by the ring $k[x;\sigma^*,\delta^*]$ where σ^*, δ^* are defined as follows:*

(A) *If $\sigma = 1$ then $\sigma^* = \sigma = 1, \delta^* = -\delta$.*

(B) *If $\sigma \neq 1$, then $\delta = \alpha(\sigma - 1), \alpha \in k$ (see Lemma 2.1). Set $\sigma^* = \sigma^{-1}$, $\delta^* = \alpha(\sigma^* - 1) = \alpha(\sigma^{-1} - 1)$.*

Let $L = a_n\, x^n + \cdots + a_1\, x + a_0 \in k[x;\sigma,\delta]$. The adjoint operator L^* is then defined by

$$L^* = x^n\, a_n + \cdots + x\, a_1 + a_0 \in k[x;\sigma^*,\delta^*].$$

Note that the product $x^i\, a_i$ must be computed in the ring $k[x;\sigma^*,\delta^*]$. It is easy to show that $(\sigma^*)^* = \sigma$, $(\delta^*)^* = \delta$. One can also verify that that the adjoint is a linear bijective map and that $(M \circ N)^* = N^* \circ M^*$.

Lemma 4.1 [4] *Let θ be a pseudo–linear map w.r.t. σ, δ. From Lemma 2.2,*

$$\theta = \theta_c = c\,\sigma + \delta.$$

Set

$$\theta^* = c\,\sigma^* + \delta^*.$$

Then θ^* is a pseudo–linear map w.r.t. σ^*, δ^*. $\square$

Define the operator ∇ as in [4]:

$$\nabla = \begin{cases} \delta & \text{if } \sigma = 1, \\ \sigma - 1 & \text{if } \sigma \neq 1. \end{cases}$$

An element $f \in k$ is an *integrating factor* for $L \in k[\nabla]$ if $f\,L = \nabla \circ M$, for some $M \in k[\nabla],\ \operatorname{ord} M = \operatorname{ord} L - 1$.

Lemma 4.2 [1] *$f \in k$ is an integrating factor for L iff $L^*(f) = 0$.* $\square$

As a consequence, a solution of the adjoint equation $L^*y = 0$ allows one to obtain an integrating factor f for the original operator L. Since $\nabla g = 0$ iff g is a constant (Proposition 1, [1]), this allows one to reduce the order of the given equation $Ly = 0$ (though we get an inhomogeneous equation instead of a homogeneous equation).

Note: In case (B), $k[x;\sigma,\delta]$ and the adjoint ring $k[x;\sigma^*,\delta^*]$ are not necessarily the same ring, and in order to work with adjoint equations, one needs to provide σ^{-1}.

5 The `OreTools` Package

5.1 Functionalities

The `OreTools` package is designed to perform the basic arithmetic in Ore algebra. The proposed functionalities can be classified into the following groups:

1. **Define an Ore algebra:** an Ore algebra is defined via the command `SetOreRing`. The *differential* and *shift* algebras are pre-defined. To define other algebras, one needs to provide procedures to compute σ, δ, θ_1, (see 2.1) and σ^{-1} (see 4).

   Example 1 To define the *shift* algebra:

   ```
   > A := OreTools[SetOreRing](n,'shift');
   ```

A := OreRing(n, shift)

To define the *q-shift* algebra:

```
> B := SetOreRing(n,'qshift', 'sigma' = proc(p,x) eval(p,x=q*x) end,

> 'sigma_inverse' = proc(p,x) eval(p,x=q^(-1)*x) end,
> 'delta' = proc(p,x) 0 end, 'theta1' = 1);
```

B := OreRing(n, qshift)

2. **Access "properties" of a given algebra:** various routines are provided to allow access to the properties of a given algebra. They are

   `GetAlgebraName` returns the name of the algebra, e.g., *shift*, *differential*;

   `GetVariable` returns the name of the independent variable that an Ore polynomial ring acts on;

   `GetSigma, GetSigmaInverse, Getdelta, GetTheta1` return $\sigma, \sigma^{-1}, \delta$, and θ_1, respectively.

   Example 2 Continue with the shift algebra A as defined in Example 1:

```
> GetAlgebraname(A);
```

shift

```
> GetVariable(A);
```

n

```
> GetSigma(A)(s(n),n);
```

s(n + 1)

```
> GetSigmaInverse(A)(s(n),n);
```

s(n - 1)

```
> Getdelta(A)(s(n),n);
```

$$0$$

```
> GetTheta1(A);
```

$$1$$

3. **Basic arithmetic:** The basic arithmetic supported includes:

   (a) Linear operations

      `Add`: addition of two Ore polynomials;

      `Minus`: subtraction of two Ore polynomials;

      `ScalarMultiply`: multiplication of an Ore polynomial by a scalar on the left;

   (b) operations for normalization

      `Content, Primitive`: computation of the content and primitive part of an Ore polynomial, resp..

      `NormalizeOrePoly`: conversion of an Ore polynomial to its monic associate.

   The above five operations do not require any information from a given Ore algebra.

   (c) Multiplication

      `Multiply`: multiplication of two polynomials in a given algebra;

   (d) Divisions

      `RightRemainder, RightQuotient, RightQuotientRemainder`: computation of the right quotient and remainder of two polynomials;

      `LeftQuotientRemainder`: computation of the left quotient and remainder of two polynomials;

      `RightPseudoRemiander,RightPseudoQuotient`: computation of the right pseudo-remainder and pseudo-quotient of two polynomials;

   (e) Euclidean algorithms

`RightEuclidean, FractionFreeRightEuclidean`: computation of the right remainder sequence (resp. subresultant sequence of the first kind);

(f) GCRD and LCLM

`ExtendedGCRD, HalfExtendedGCRD`: extended and half-extended right Euclidean algorithm;

`GCRD, LCLM`: computation of GCRD and LCLM of several Ore polynomials (cofactors can be obtained by an additional input `'cofactors'=true`

The subresultant algorithm in [10] is used in the function `FractionFreeRightEuclidean`. An improved version of the modular algorithm in [11] is used in the function `GCRD` when the coefficients of input polynomials are univariate rational functions. An improved version of the algorithm for computing LCLM in `DEtools` is used in the function `LCLM`. These techniques make a number of functions in `OreTools` more efficient. See the experimental results in Appendix B.

Example 3 Continue with the shift algebra A as defined in Example 1:

```
> Ore1 := OrePoly(-n/(n-1),-(-5*n+n^2+3)/(n-1),n-3);
```

$$\mathrm{Ore1} := \mathrm{OrePoly}\left(-\frac{n}{n-1}, -\frac{-5\,n+n^2+3}{n-1}, n-3\right)$$

```
> Ore2 := OrePoly(-n,3*n-n^2-1,(n-1)^2);
```

$$\mathrm{Ore2} := \mathrm{OrePoly}\left(-n, 3\,n-n^2-1, (n-1)^2\right)$$

```
> Ore3 := OrePoly(-n,n-n^2,n^2);
```

$$\mathrm{Ore3} := \mathrm{OrePoly}\left(-n, n-n^2, n^2\right)$$

```
> Add(Ore1,Ore2);
```

$$\mathrm{OrePoly}\left(-\frac{n^2}{n-1}, -\frac{-n-3\,n^2+2+n^3}{n-1}, -n-2+n^2\right)$$

```
> ScalarMultiply(sqrt(2),Ore1);
```

$$\mathrm{OrePoly}\left(-\frac{\sqrt{2}n}{n-1}, -\frac{\sqrt{2}\left(-5\,n+n^2+3\right)}{n-1}, \sqrt{2}\,(n-3)\right)$$

```
> Multiply(Ore1,Ore2,A);
```

$$\mathrm{OrePoly}\left(\tfrac{n^2}{n-1}, \tfrac{-7\,n^2+2\,n^3-n+3}{n-1}, \tfrac{-8\,n^3+11\,n^2+6\,n+n^4-9}{n-1}, -\tfrac{-8\,n^3+2\,n^4+n^2+7\,n-3}{n-1}, (n-3)\,(n+1)^2\right)$$

```
> LeftQuotientRemainder(Ore1,Ore2,A);
```

$$\left[\mathrm{OrePoly}\left(\frac{n-5}{(n-3)^2}\right), \mathrm{OrePoly}\left(-4\,\frac{n}{(n-1)\,(n-3)^2}, \frac{n^3-7\,n^2+15\,n-8}{(n-1)\,(n-2)^2}\right)\right]$$

```
> RightQuotientRemainder(Ore1,Ore2,A);
```

$$\left[\mathrm{OrePoly}\left(\frac{n-3}{(n-1)^2}\right), \mathrm{OrePoly}\left(-2\,\frac{n}{(n-1)^2}, 2\,\frac{n}{(n-1)^2}\right)\right]$$

```
> GCRD(Ore1,Ore2,Ore3, A);
```

$$\mathrm{OrePoly}\,(-1,1)$$

```
> GCRD(Ore1, Ore2, Ore3, 'cofactors' = true, A);
```

$$[\mathrm{OrePoly}(-1,1), [\mathrm{OrePoly}\left(\frac{n}{n-1}, n-3\right), \mathrm{OrePoly}\left(n, (n-1)^2\right), \mathrm{OrePoly}\left(n, n^2\right)]]$$

```
>LCLM(OrePoly(1,1),OrePoly(n,1), OrePoly(0,n), cofactors = true, A);
```

$$[\mathrm{OrePoly}\left(0, \frac{n^2+2n+1}{n}, \frac{3n+n^2+1}{n}, 1\right),$$

$$[\mathrm{OrePoly}\left(0, \frac{n^2+2n+1}{n}, 1\right), \mathrm{OrePoly}\left(0, \frac{n+1}{n}, 1\right), \mathrm{OrePoly}\left(\frac{n^2+2n+1}{n^2}, \frac{3n+n^2+1}{n(n+1)},\right.$$

```
> ExtendedGCRD(Ore1,Ore2,A);
```

$$\Big[\mathrm{OrePoly}\left(-2\,\tfrac{n}{(n-1)^2}, 2\,\tfrac{n}{(n-1)^2}\right), \mathrm{OrePoly}\,(1)\,, \mathrm{OrePoly}\left(-\tfrac{n-3}{(n-1)^2}\right), \mathrm{OrePoly}\left(-1/2\;(n-1)^2, -1/2\,\tfrac{(n-1)^2n^2}{n+1}\right), \mathrm{OrePoly}\left(-1/2+1/2\,n, 1/2\,\tfrac{(n-1)^2(n-2)}{n+1}\right)\Big]$$

Note that similar to the function `skew_gcdex` of the `Ore_algebra` package, for two given Ore polynomials p and q, the function `ExtendedGCRD` returns a list $[g, a, b, u, v]$ such that $u\,p + v\,q = 0$ and $a\,p + b\,q = g$. Hence, g is a GCRD of p and q, while $u\,p$ and $v\,q$ are LCLM's of p and q.

4. **Application of L and L^* to an element of k:**

   The two routines `ApplyOrePoly` and `ApplyAdjointOrePoly` yield the application of L and L^* to an element of k, respectively.

   Example 4

   Define the *difference* algebra:

```
> A := SetOreRing(n,'difference',
> 'sigma' = proc(p,x) eval(p,x=x+1) end,
> 'sigma_inverse' = proc(p,x) eval(p,x=x-1) end,
> 'delta' = proc(p,x) eval(p,x=x+1) - p end,
> 'theta1' = 0);
```

$$\mathrm{A} := \mathrm{OreRing}(\mathrm{n},\ \mathrm{difference})$$

```
> L := OrePoly(n^2/(n-1),n,1/n^2);
```

$$\mathrm{OrePoly}\left(\frac{n^2}{n-1}, n, \frac{1}{n^2}\right)$$

Apply the operator L to $s(n)$:

```
> ApplyOrePoly(L,s(n),A);
```

$$\frac{n^2}{n-1}\,s(n) + n\,(s\,(n+1) - s\,(n)) + \frac{1}{n^2}(s\,(n+2) - 2\,s\,(n+1) + s\,(n))$$

The following example is from Example 2a [1]. The operator L is the minimal annihilator for J_1.

```
> SetOreRing(x,'differential');
```

$$\mathrm{A} := \mathrm{OreRing}(\mathrm{n},\ \mathrm{differential})$$

```
> L := OrePoly(x^2-1,x,x^2);
```

$$\mathrm{OrePoly}\left(x^2-1, x, x^2\right)$$

Compute the application of the adjoint of L to a function $f(x)$:

```
> ApplyAdjointOrePoly(L,f(x),A);
```

$$x^2 f(x) + 3\,x\,\frac{d}{dx} f(x) + x^2\,\frac{d^2}{dx^2} f(x)$$

Another example (Example 3 [1]).

```
> A := SetOreRing(n,'shift');
```

$$\mathrm{A} := \mathrm{OreRing(n, shift)}$$

```
> L := OrePoly(1,-2,-2,1);
```

$$L := \mathrm{OrePoly}\,(1, -2, -2, 1)$$

```
> ApplyAdjointOrePoly(L,s(n),A);
```

$$s(n) - 2\,s(n-1) - 2\,s(n-2) + s(n-3)$$

The above result shows that $L^* = E^{-3} - 2\,E^{-2} - 2\,E^{-1} + 1$ where E denotes the shift operator w.r.t. n. Note that L and L^* are not in the same ring.

5. **Conversion between different representations:** The routine `FromOrePolyToLinearEqua` allows the conversion from an `OrePoly` structure to the corresponding linear functional equation, while `FromLinearEquationToOrePoly` allows the reversed direction (currently it only handles the *shift* and *differential* algebras (Maple does not have support for other algebras yet)).

   Example 5

   Define the *differential* algebra:

```
> A := SetOreRing(n,'differential');
```

$$\mathrm{A} := \mathrm{OreRing(x, differential)}$$

```
> L := OrePoly(1,-x,x^2+C*x);
```

$$L := \mathrm{OrePoly}\left(1, -x, x^2 + Cx\right)$$

```
> eq := FromOrePolyToLinearEquation(L,f,A);
```

$$deq := f(x) - x\frac{d}{dx}f(x) + \left(x^2 + Cx\right)\frac{d^2}{dx^2}f(x)$$

```
> FromLinearEquationToOrePoly(eq,f,A);
```

$$\mathrm{OrePoly}\left(1, -x, x^2 + Cx\right)$$

Note that one can use the function `AddConversionRule` to add rules for the conversion from a linear equation to an `OrePoly` structure. Suppose that we would like to convert a linear difference equation to an `OrePoly` structure:

Define the difference algebra

```
> A := SetOreRing(n,'difference',
> 'sigma' = proc(p,x) eval(p,x=x+1) end,
> 'sigma_inverse' = proc(p,x) eval(p,x=x-1) end,
> 'delta' = proc(p,x) eval(p,x=x+1) - p end,
> 'theta1' = 0);
```

$$\mathrm{A} := \mathrm{OreRing(n,\ difference)}$$

Consider the following difference operator:

```
> L := OrePoly(n^4,n^4-n^3,n-1);
```

$$L := \mathrm{OrePoly}(n^4, n^4 - n^3, n - 1)$$

Convert L to a linear difference equation:

```
> eq := FromOrePolyToLinearEquation(L,s,A);
```

$$n^4 s(n) + \left(n^4 - n^3\right)(s(n+1) - s(n)) + (n-1)(s(n+2) - 2\,s(n+1) + s(n))$$

Now convert *eq* to the corresponding `OrePoly` structure:

```
> FromLinearEquationToOrePoly(eq,s,A);
```

Error, (in FromLinearEquationToOrePoly) unable to handle the difference case

Now we define the rule for the difference case:

```
difference_case := proc(eq, func, A)
   local var, Del, oper, i;
   var := OreTools:-GetVariable(A);
   oper := subs(LREtools['Delta'][var]='Del', LREtools['REtodelta'](eq,func(var),{}));
   'OrePoly'(seq(coeff(oper,'Del',i),i=0..degree(oper,'Del')))
end proc:
```

Add this rule into `OreTools`, and try `FromOrePolyToLinearEquation` again:

```
> AddConversionRule('difference',difference_case);
> FromLinearEquationToOrePoly(eq,s,A);
```

$$\mathrm{OrePoly}\left(n^4, n^4 - n^3, n - 1\right)$$

Caveat: The routine `difference_case` as defined above works correctly only when the coefficients are polynomials. For a version (which is a bit longer) that also works correctly for the rational function coefficients, please check the test `convertors2.tst` in the directory `~maple/repository/scg/lib/OreTools/tst`.

5.2 Implementation Details

5.2.1 Object Representation

1. **Ore polynomials.** An Ore polynomial is represented by an OrePoly structure. It consists of the keyword OrePoly with a sequence of coefficients starting with the one of degree zero. For example, in the differential case with the differential operator D, OrePoly$(2/x, x, x+1, 1)$ represents the operator $2/x + x\,D + (x+1)\,D^2 + D^3$.

2. **Ore algebra.** Modules are used to model Ore algebra "objects", each of which satisfies the following interface:

```
> `type/OreRing` := ''module'(
>     sigma::procedure, inverse_sigma::procedure,
>     delta::{procedure,algebraic}, theta1::algebraic,
>     var::name, case::name
> )':
```

For instance,

```
> A := SetOreRing(n,'shift'):
> type(attributes(A),OreRing);
```

$$true$$

A note to internal developers: the code is structured to make the expansion of the list of pre-defined algebras quite simple. Since it is sufficient to define an algebra by providing $\sigma, \sigma^{-1}, \delta, \theta 1$, all one needs to do is to create a module with this needed information, and then use the *local* function `AddAlgebra`. For instance, if one inserts the following piece of code

```
qShiftAlgebra := module()
   description "q-shift algebra";
   export Sigma, InverseSigma, delta, Theta1;
   Theta1 := 1;
   Sigma := (p,var) → eval(p,var=q*var);
   InverseSigma := (p,var) → eval(p,var=(1/q)*var);
   delta := (p,var) → 0;
end module;
AddAlgebra('qshift',qShiftAlgebra);
```

into the module `DefineRing`, one just adds the q-shift algebra (named 'qshift') into the list of pre-defined algebra. After the code is inserted:

```
> A := SetOreRing(n,'qshift');
```

A := OreRing(n, qshift)

```
> GetSigma(A)(s(n),n);
```

s(q n)

```
> L := OrePoly(-q*(1-q*x),1);
> ApplyAdjointOrePoly(L,s(n),A);
```

$$s(n/q) - q(1 - q\,x)s(n)$$

5.2.2 Implicit Structure

An Ore polynomial ring is defined via `SetOreRing`. This function returns an implicit structure `OreRing` with two arguments: the first is the name of the independent variable, and the second one the name of the ring. This implicit structure has an attribute which is a module storing the information to define a Ore polynomial ring. For instance,

```
> A := SetOreRing(x,'differential'):
> attributes(A);
```

module() export case, var, sigma, inverse_sigma, delta, theta1; **end module**

We believe that it is an advantage to use the module-based approach to represent/model objects rather than the table-based approach which has been pretty much the *traditional* way to represent objects in Maple.

5.3 An Example

We now show an example of using the `OreTools` package as a basis for solving the accurate integration problem. The routine `IntegrateSols` is a straightforward translation of the pseudo code as given in [1]:

```
IntegrateSols := proc(L::OrePoly,x::name,case::symbol)
    local A, y, eq, sol, l, Delta, One, r, Ltilde;
    userinfo(3,'IntegrateSols',"Define the algebra");
    A := OreTools:-SetOreRing(x,case);
    Delta := GetDelta(A);
    userinfo(3,'IntegrateSols',"Set up the functional equation L*(y)=1");
    eq := OreTools:-ApplyAdjointOrePoly(L,y(x),A) = 1;
    userinfo(3,'IntegrateSols',"Find a rational solution of L*(y)=1");
    sol := RationalSolution(eq, y, x, case);
    if nops(sol) = 1 then
        userinfo(3,'IntegrateSols',"No rational solution found");
        NULL
    else
```

```
        userinfo(3,'IntegrateSols',"a rational solution is found");
        l := sol[2];
        One := 'OrePoly'(1);
        use OreTools in
          userinfo(3,'IntegrateSols',"Compute r = LeftQuotient(1-lL,Delta)");
          r := LeftQuotientRemainder(
            Add(One,ScalarMultiply(-l,L),A), Delta, A)[1];
          userinfo(3,'IntegrateSols',"Compute L = L o Delta");
          Ltilde := Add(One,ScalarMultiply(-1,Multiply(r,Delta,A)),A)
        end;
        userinfo(3,'IntegrateSols',"Computation successful");
        [Ltilde, r]
      end if;
    end proc:
```

In order to have `IntegrateSols` work in a given algebra, we need to provide routines to compute the "rational solutions" of $L^*y = 1$ and to compute Δ. For the shift and differential cases, finding the "rational solutions" are available via `LREtools[ratpolysols]` and `DEtools[ratsols]`, respectively.

```
    RationalSolution := proc(eq, y, x, case)
      local sol, inds, basis_sol, par_sol;
      if not member(case,'shift','differential') then
        error "only the shift and differential cases are supported"
      end if;
      if case = 'differential' then
        sol := DEtools['ratsols'](eq,y(x));
      else
        sol := LREtools['ratpolysols'](eq,y(x),{},'output'='basis');
        if sol = NULL then return([[]]) end if;
        # need to massage the output of LREtools[ratpolysols]
        inds := [op(indets(sol, _C[integer]))];
        basis_sol := map2(coeff,sol,inds);
        par_sol := eval(sol,map(x→x=0,inds));
        if par_sol = 0 then sol := [basis_sol]
        else sol := [basis_sol,par_sol] end if;
      end if;
      sol
```

```
end proc:

GetDelta := proc(A)
  local p, var, sig, del, Del;
  var := OreTools:-GetVariable(A);
  sig := OreTools:-GetSigma(A);
  del := OreTools:-Getdelta(A);
  # if $\sigma = 1$ then $\Delta = \delta$
  # if $\sigma \neq 1$ then $\Delta = \sigma - 1$
  if sig(p(var),var) = p(var) then
    Del := del(p(var),var)
  else
    Del := sig(p(var),var) - p(var)
  end if;
  OreTools:-FromLinearEquationToOrePoly(Del,p,A)
end proc:
```

We now present two examples. One is for the differential case, one for the shift case. `infolevel` is used for the first example to show the main steps of the algorithm.
Differential Case: an example from the help page of `DEtools[integrate_sols]`

```
> L := OrePoly(-(z-2)*(z^2-z+1)/z/(z-1)^2, -3/2/z/(z-1), 1);
```

$$L := \mathrm{OrePoly}\left(-\frac{(z-2)\,(z^2-z+1)}{z\,(z-1)^2}, -3/2\,\frac{1}{z\,(z-1)}, 1\right)$$

```
> IntegrateSols(L,z,'differential');
 IntegrateSols:   "Define the algebra"
 IntegrateSols:   "Set up the functional equation L*(y)=1"
 IntegrateSols:   "Find a rational solution of L*(y)=1"
 IntegrateSols:   "a rational solution is found"
 IntegrateSols:   "Compute r = LeftQuotient(1-lL,Delta)"
 IntegrateSols:   "Compute L~= L o Delta"
 IntegrateSols:   "Computation successful"
```

$$\left[\mathrm{OrePoly}\left(1, \frac{1}{2}\,\frac{1}{(z-1)^2}, -\frac{z}{z-1}\right), \mathrm{OrePoly}\left(-\frac{1}{2}\,\frac{1}{(z-1)^2}, \frac{z}{z-1}\right)\right]$$

Shift Case: Example 3 from [1]

```
> L := OrePoly(1,-2,-2,1);
```

$$L := \text{OrePoly}\,(1, -2, -2, 1)$$

```
> IntegrateSols(L,n,'shift');
```

$$\left[\text{OrePoly}\left(-\frac{1}{2}, 1, 1, -\frac{1}{2}\right), \text{OrePoly}\left(-\frac{3}{2}, -\frac{1}{2}, \frac{1}{2}\right)\right]$$

6 A comparison

At this point, a natural question to ask is "why do we need the `OreTools` package in addition to the existing and well-known package `Ore_algebra`?" The answer is quite simple: the skew-polynomial rings in the `OreTools` package are defined over a field, instead of over a ring as in the case of the `Ore_algebra` package. Consider the following simple example:

```
> with(Ore_algebra):
> A := shift_algebra([En,n]);
```

$$\text{A} := \text{Ore\_algebra}$$

```
> Ore1 := (3*n+1)*En-1;
```

$$\text{Ore1} := (3n + 1)En - 1$$

```
> Ore2 := (2*n-2)/(n+2)+En;
```

$$\text{Ore2} := \frac{2\,n - 2}{n + 2} + En$$

We would like to compute the right remainder of Ore1 by Ore2, and it is not possible to use directly any top-level command in `Ore_algebra` to obtain the result. Furthermore, all algebras declared in the `Ore_algebra` package are a priori with integer coefficients. Any other type of coefficients has to be explicitly declared. This makes it difficult to perform the basic arithmetics when the coefficients have parameters or algebraic numbers. See Appendix A for an example of this type.

Another factor that should be taken into consideration is *efficiency*. We believe that the `OreTools` package outperforms the `Ore_algebra` package. See appendix B for the timing comparison of our experiment. We would like to emphasize that this is by no means an attempt to disparage the quality of the `Ore_algebra` package (it is indeed a powerful package). Our point of view is that each package is designed to be suitable for some certain operations, and hence can co-exist within the Maple system.

References

[1] S.A. Abramov, M.v. Hoeij. Integration of solutions of linear functional equations. *Integral Transformations and Special Functions* **8**, No. 1-2, 1999, 3–12.

[2] S.A. Abramov, E.V. Zima. Minimal completely factorable annihilators. In: W. Küchlin (ed.), *Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation*, ACM Press, 290–297.

[3] S.A. Abramov, E.V. Zima. D'Alembertian solutions of inhomogeneous linear equations (differential, difference, and some other). In: Y.N. Lakshman (ed.), *Proceedings of the 1996 International Symposium on Symbolic and Algebraic Computation*, ACM Press, 232–240.

[4] S.A. Abramov. Ore rings and linear equations, *Unpublished.*

[5] S.A. Abramov, E.V. Zima, A universal program to uncouple linear systems. In: *Proceedings of CMCP'96 (International Conference on Computational Modeling and Computing in Physics*, Dubna, Russia, Sept. 16-21, 1996), 1997, pp. 16–26.

[6] M. Bronstein, M. Petkovšek. An introduction to pseudo–linear algebra, *Theoretical Computer Science* **157**, 1996, 3–33.

[7] M. Bronstein, B. Zürcher. Uncoupling algorithms for pseudo–linear systems, *Submitted to AAECC.*

[8] M. Giesbrecht, Y. Zhang. Factoring and decomposing Ore polynomials over $\mathbb{F}_p(t)$. To appear in the *Proceedings of the 2003 International Symposium on Symbolic and Algebraic Computation.*

[9] N. Jacobson. Pseudo-linear transformations. *Annals of Mathematics* **38**, 484–507.

[10] Z. Li. A subresultant theory for ore polynomials with applications. In: O. Gloor (ed), *Proceedings of the 1998 International Symposium on Symbolic and Algebraic Computation*, ACM Press, 132–139.

[11] Z. Li, I. Nemes. A modular algorithm for computing greatest common right divisors of Ore polynomials. In: W. Küchlin (ed.), *Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation*, ACM Press, 282–289.

[12] O. Ore. Theory of non-commutative polynomials. *Annals of Mathematics* **34**, 1933, 480–508.

[13] E. G. C. Poole. *Introduction to the theory of linear ordinary differential equations.* Dover Publications Inc., New York, 1936.

[14] J. van der Hoeven. FFT-like multiplication of linear differential operators. *Journal of Symbolic Computation* **33**, Issue 1, 2002, 123 - 127.

[15] Wedderburn J.H.M. Non-commutative domains of integrity. *J. Reine Angew. Math.* **167**, 1932, 129–141.

Appendix A

Consider the following two polynomials in shift algebra [1]:

```
> Ore1 := OrePoly(-231420*I*551^(1/2)*(1287+21*I*551^(1/2)+740*n)*

> (42*n+19-2013^(1/2))*(42*n+19+2013^(1/2)),231420*I*551^(1/2)*

> (547+21*I*551^(1/2)+740*n)*(42*n+61-2013^(1/2))*(42*n+61+2013^(1/2))):


> Ore2 := OrePoly(231420*I*551^(1/2)*(1287-21*I*551^(1/2)+740*n)*

> (42*n+19-2013^(1/2))*(42*n+19+2013^(1/2)),-231420*I*551^(1/2)*

> (547-21*I*551^(1/2)+740*n)*(42*n+61-2013^(1/2))*(42*n+61+2013^(1/2))):
```

One can perform the extended right Euclidean algorithm using `OreTools:-ExtendedGCRD`:

```
> SetOreRing(n,'shift'):
> ExtendedGCRD(Ore1,Ore2,A):
```

To perform the same operation using the `Ore_algebra` package:

```
> A := skew_algebra(shift=[E_n,n],alg_relations={i^2+1,a^2-551,b^2-2013},comm={i,a

> L1 :=231420*I*551^(1/2)*(547+21*I*551^(1/2)+740*n)*(42*n+61-2013^(1/2))*(42*

> n+61+2013^(1/2))*E_n-231420*I*551^(1/2)*(1287+21*I*551^(1/2)+740*n)*

> (42*n+19-2013^(1/2))*(42*n+19+2013^(1/2)):
> L2 :=-231420*I*551^(1/2)*(547-21*I*551^(1/2)+740*n)*(42*n+61-2013^(1/2))*(42

> *n+61+2013^(1/2))*E_n+231420*I*551^(1/2)*(1287-21*I*551^(1/2)+740*n)*

> (42*n+19-2013^(1/2))*(42*n+19+2013^(1/2)):
> L1:=subs([I=i,551^(1/2)=a,2013^(1/2)=b],L1):
```

[1]This example was originally posted by Ha Le on the mapledev mailing list. The solution obtained by using the Ore_algebra package was provided by Frédéric Chyzak.

```
> L2:=subs([I=i,551^(1/2)=a,2013^(1/2)=b],L2):
> skew_gcdex(L1,L2,E_n,A):
```

Since all algebras declared in the `Ore_algebra` package are a priori with integer coefficients, any other type of coefficients has to be explicitly declared. Hence, being able to perform the same operation (the extended right Euclidean algorithm in this example) requires a non-trivial effort and knowledge from users.

Appendix B

In this experiment, we run `OreTools:-ExtendedGCRD` and `Ore_algebra[skew_gcdex]` on a set of randomly-generated polynomials in shift algebra, and do the timing comparison. The two polynomials in each call are elements from $\mathbb{Q}[n][E_n]$ and are of degrees 7 and 8, respectively.

```
# Generate elements in Q[n]
randpol := proc(var::name)
  randpoly([var],'terms'=3,'coeffs'=RandomTools['Generate'](
    'rational'('denominator'=10),'makeproc'));
end proc:

# Generate Ore polynomials with coefficients in Q[n]
RandOrePoly := proc(var::name, deg::nonnegint)
  local terms, i;
  terms := NULL;
  for i from 0 to deg do terms := terms, randpol(var); end do;
  'OrePoly'(terms)
end proc:

for i to 10 do
   Ore := RandOrePoly(n,2);
   Ore1 := RandOrePoly(n,5); Ore1 := Multiply(Ore1,Ore,A);
   Ore2 := RandOrePoly(n,6); Ore2 := Multiply(Ore2,Ore,A);
end do;
```

Table 1 shows the time (in seconds) required for each call to `OreTools:-ExtendedGCRD` and
`Ore_algebra[skew_gcdex]` Table 2 shows the time (in seconds) required for each call to `OreTools:-GCRD`, `OreTools:-ExtendedGCRD` and `OreTools:-LCLM` with cofactors. Input polynomials are the same as those used for Table 1.
[2].

[2]All the reported timings were obtained on a 400Mhz SUN SPARC SOLARIS with 1Gb RAM.

Table 1: Time requirement for 2001 version.

	`ExtendedGCRD`	`skew_gcdex`
1	189.920	889.920
2	202.000	908.820
3	206.110	916.320
4	200.270	906.790
5	194.190	922.660
6	234.420	1405.310
7	218.070	1004.410
8	186.630	673.060
9	219.060	1056.660
10	165.020	660.830

Table 2: Time requirement for 2002 version.

	`GCRD`	`ExtendedGCRD`	`LCLM`
1	4.33	76.93	138.75
2	4.54	65.84	185.18
3	3.83	67.10	58.88
4	4.44	87.65	130.42
5	4.01	68.65	117.29
6	4.37	95.33	128.36
7	3.89	78.44	88.80
8	4.48	89.75	135.76
9	4.16	42.53	40.20
10	4.28	88.34	133.88