A generic implementation of differential characteristic set algorithms in Aldor

Christian Aistleitner
Research Institute for Symbolic Computation (RISC-Linz)
Johannes Kepler University, Linz, Austria
christian.aistleitner@gmx.at

Abstract

This report compares various implementations for differential characteristic set computations and introduces a characteristic set library for Aldor. Its design along with alternatives is discussed. The possibilities to connect the library to Maple and Mathematica are presented and a comparison of different implementations for differential polynomial rings is given.

The work described in this report has been supported by the research project SFB F013/1304 funded by the Austrian Science Foundation (FWF).

Contents

Notation

Typefaces:

SomeProgram	denotes the piece of software, called SomeProgram
SomeLanguage	denotes the programming language SomeLanguage
`SomeLib`	denotes the Aldor library SomeLib Additionally, this typeface is used to denote source code and files.
SomeType	denotes the Aldor category or domain SomeType. The first letter of names for categories and domains is an uppercase letter.
someIdentifier	denotes the Aldor function or constant someIdentifier. The first letter of names for functions and constants is a lowercase letter.

Symbols:

δ_i	the i^{th} derivation of Δ
Δ	the set $\{\delta_1, \delta_2, \ldots, \delta_n\}$ of derivations on $R\{Y\}$
m	the number of elements in Y
n	the number of elements in Δ
$\mathbb{N}_0$	$\{0, 1, 2, \ldots\}$
$R\{Y\}$	differential polynomial ring with differential indeterminates Y and the derivations Δ
Θ	the free commutative monoid generated by Δ
ΘY	the derivatives
Y	the set $\{y_1, y_2, \ldots, y_m\}$

Functions:

$\deg(p)$	the total degree of the differential polynomial p
$\deg_{\theta y}(p)$	the degree of the differential polynomial p with respect to the indeterminate θy
$\text{lead}(p)$	the greatest indeterminate of $R\{Y\}$ occurring in the differential polynomial p

Figures:

The notation of the figures in Section 3 to Section 5 is based on the UML [25] notation for class diagrams. However, the graphical distinction between an interface and its implementation is dropped, as Aldor permits interfaces (categories) to have fields that are constants. Therefore, also the distinction between a "generalization" and "realization" relation is dropped. The used notation is explained is Figure 0.1, Figure 0.2, and Figure 0.3.

Figure 8.1 uses the UML notation for sequence diagrams.

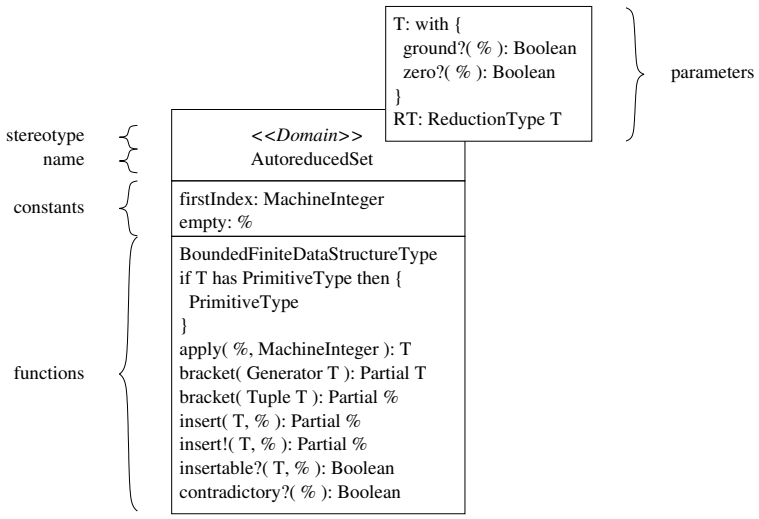

Figure 0.1: Notation for categories and domains in class diagrams

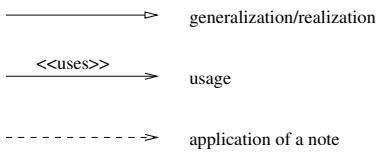

Figure 0.2: Relations in class diagrams

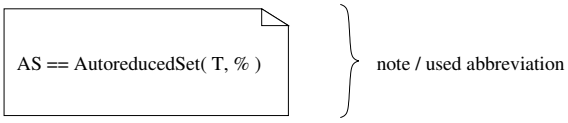

Figure 0.3: Further notation in class diagrams

1 Introduction

In this report we treat implementation aspects of characteristic set algorithms in Aldor [2]. Three objectives are identified:

- implementing characteristic set algorithms in Aldor,
- connecting these algorithms to other computer algebra systems, and
- comparing different implementations of polynomials.

During the work for the report, we implemented `CharSet` [6]. We present this generic implementation of differential characteristic set algorithms along with design decisions and a discussion of design alternatives. `CharSet` can easily be extended by further reductions, orders and implementations of differential polynomials. The performance of eleven different implementations of differential polynomials is compared to each other. Furthermore, it is shown by example how to connect Aldor code to MAPLE and MATHEMATICA. Finally, a survey about available implementations for characteristic set algorithms is given.

The report starts by motivating the use of Aldor as programming language in Section 2. This section also provides a discussion of the used libraries and environment. Additionally, an overview of the implementation is given.

Sections 3–6 presents `CharSet` and tackle the first goal, which is a characteristic set implementation in Aldor. In Section 3 the implementation of derivatives, orders and differential polynomials is discussed. Section 4 provides abstractions of the differential reduction algorithm. The necessary algorithms for characteristic set computations are presented in Section 5. In Section 6 alternatives for the presented designs of sections 3–5 are discussed.

Sections 7–10 address the second goal, which is to connect the Aldor implementation to other computer algebra systems. Section 7 shows how to export code from the Aldor environment. This exported code is connected to MAPLE (Section 8), to MATHEMATICA (Section 9), and to a command line utility (Section 10).

In Section 11, different implementations of differential polynomials are compared in the context of characteristic set calculations. Thereby, the third objective is covered.

Additionally, in Section 12 the implemented characteristic set library is compared to other existing libraries for performing characteristic set calculations.

The report closes by summarizing the made achievements and evaluating the outcome in Section 13.

2 The environment for the implementation

This section presents the environment and anatomy of the characteristic set implementation.

The first part of this section presents in broad strokes why Aldor [2] has been chosen as the language for the implementation. Afterwards, Aldor and its two core libraries `Aldor`[1] and `Algebra` are examined. Finally, an overview of the implementation is given. In this overview all used libraries are put into context.

2.1 Searching for an appropriate environment

When it comes to designing and implementing a piece of software, one major aspect is to find a language (or set of languages) to implement the tasks in. For this implementation the language Aldor [2] has been chosen along with the two libraries `Aldor` and `Algebra` that are shipped with the Aldor compiler.

The reasons for the choice of Aldor are manifold.

The language for this report's implementation should not already provide an implementation of characteristic set algorithms. Aldor does not provide algorithms for characteristic sets[2].

Additionally, an expressive language was looked for that could facilitate the generation of efficient programs. Aldor is packed with features such as parametrization, a fully typed system and treating functions and abstractions of code just like any built-in value. Although the Aldor compiler can act as interpreter, it is also used to compile Aldor code into an intermediate language[3]. The compiler can also produce executables, where no code is interpreted, which leads to a better performance.

Another aspect of this report is to investigate the ways to connect code to Maple [18] and Mathematica [21]. Aldor allows to generate code in various formats for the `GNU/Linux` platform. These formats can be used either directly (like object code for C) or can be used to generate the desired code (as object files are transformed to MathLink executables for Mathematica). More on the different formats can be found in Section 7.

The language of choice has to be able to deal with the typical data structures arising in the implementation of computer algebra algorithms. Aldor has its roots in a computer

[1] Note the different typeface in Aldor and `Aldor`. Aldor will be used to refer to the language Aldor, while `Aldor` will be used to refer to the library `Aldor` of the language Aldor. See page 6 for further notes on notation.

[2] Although, there currently exists no implementation of characteristic set algorithms in Aldor, it has been discovered during the implementation of this report that Marc Moreno Maza is implementing characteristic set algorithms in Aldor as well. However, there currently exists no publicly available distribution of his library.

[3] For Aldor, this intermediate language is FOAM (first order abstract machine). This report does not discuss FOAM. Further information about FOAM can be found for example in [52]. However, as the cited report is hard to come by, [34] may be used to get a first impression of FOAM.

algebra system, so it is naturally capable of dealing with such data structures. Besides, there already is a rich algebra library available.

Besides these good aspects of the Aldor language itself, there are hardly any tools that support Aldor developers. There is for example no tool that considerably supports developers in producing a document describing the written code.

For C++, Java and other common languages, there is for example DOXYGEN [7], which is a set of tools for generating documentation automatically from the source code. DOXYGEN imposes a special syntax on the comments of a language and is thereby able to create the documentation for a source file and whole software projects. This documentation can be stored as HTML and PDF besides other formats. The Aldor language also provides markers for comments. But in addition to this, Aldor has separate keywords for documentation of source code. These keywords are a major benefit compared to other languages which only have comment keywords, such as C++ or Java. At present state, however, there is no commonly agreed syntax of how to use the documentation keywords and there is no tool that can automatically generate PDF or HTML documentation of an Aldor source file which has been documented using these documentation keywords of Aldor.

There are currently only two documentation tools available for Aldor. On the one hand, there is ALDOC [1]. ALDOC consists of a style sheet for LaTeX and two small programs. One program is used for extracting parts of an Aldor source file and another program converts LaTeX files using the ALDOC style sheet to HTML. Although readable documentation can be generated using ALDOC, the major disadvantage is that the whole documentation has to be written by hand. There is no automatic signature extraction of functions. There is no support for generating diagrams. There is no reuse of any information the source code already contains. Everything that should be in the documentation has to be typed in explicitly and generated explicitly. This makes the use of ALDOC cumbersome, in comparison to what DOXYGEN does for a C++ or Java project. Nevertheless, ALDOC is the format in which the two libraries that are shipped with the compiler are documented.

On the other hand, there is ALDORDOC [5]. In contrast to ALDOC, ALDORDOC uses the documentation keywords provided by Aldor. ALDORDOC parses files generated by the Aldor compiler and outputs XML files. Furthermore, ALDORDOC provides a tool for transforming XML files into HTML files. For this transformation tool, a working Java environment has to be installed as well. ALDORDOC did not produce any public new version since August 2000 and does not provide the flexibility of a proper documentation tool.

Besides the lack of a useful way to generate a documentation, there is another drawback. For most widely used programming languages, there are tools supporting programmer tests. There is no such tool for Aldor.

For Java, there is for example JUNIT [14]. JUNIT provides a whole framework for writing tests for Java classes and executing them in several different ways. This framework provides ways for writing single tests and putting them together in a test case. Several

test cases are aggregated into a test suite, which can then be executed. The whole testing cycle can be done automatically and as a result, JUnit eases repeated testing.

Since such a tool has been identified to be essential, we have implemented such a framework during this report's work. This framework is called `AldorUnit` [4]. `AldorUnit` is however beyond the scope of this report and is therefore not discussed here.

Another disadvantage is that, up to now, there is no integrated developer environment for Aldor. Neither are there any plug-ins for existing integrated developer environments that ease developing in Aldor. The only available support is `aldor-mode` [37], which enables syntax highlighting and proper indentation when editing Aldor source code in Emacs [8] or XEmacs [26].

Furthermore, no CASE tool supports the Aldor language. Also UML [25] can only partly be used for modelling diagrams and aspects of Aldor code. The reason for this is mostly that Aldor is not object orientated and does not meet the UML primitives smoothly enough.

After all, it has to be admitted that most of the practices used to facilitate the software engineer's every day work are missing in Aldor.

Nevertheless, the basic needs for developing software are met by Aldor, and it has been decided that the advantages of Aldor outbalance the absence of tools for Aldor. So Aldor has been chosen to be the language to implement the characteristic set algorithms in.

A decision that is closely connected to choosing Aldor to be the language of choice is the decision for the targeted platform and operating system. Aldor has to run natively on this platform as Maple and Mathematica have to. Maple and Mathematica are necessary, because the possibilities to connect Aldor code to Maple and Mathematica shall be explored.

Among the available platforms and operating systems that are directly supported by Aldor, `GNU/Linux` [13] on `x86` has been chosen. This decision is based on the facts that `GNU/Linux` is free, is widely used, and comes with a lot of tools that ease the deployment, development, and maintenance of software projects. Furthermore, the `x86` architecture is wide-spread and commonly used. Therefore, it is likely not to hinder usage of the developed piece of software.

2.2 A closer look at the chosen environment

The language Aldor is not too well known among computer scientists. Therefore, Aldor's main features and its core libraries are presented briefly in this section.

In the first part of this section, a rough sketch of the language Aldor itself is given. This presentation is followed by a description of Aldor's first core library, which is the library `Aldor`. The second core library of Aldor is `Algebra`, whose description closes this section.

2.2.1 The Aldor language

Aldor [2] is short for "a language for describing objects and relationships".

Although this acronym contains the word "objects", Aldor is far from being object orientated – at least in the usual sense of computer science. The acronym's "objects" refer to mathematical objects. Aldor tries to ease modelling of mathematical objects and their relationships, by treating all mathematical objects equally. In contrast to other languages, such as Lisp, achieving such a goal does not result in losing structure.

Aldor's type hierarchy has three layers. First of all there are categories. These categories provide an aggregation for function signatures and further categories. This layer does not set any data representation, but is used to model a type hierarchy and collect similarities of mathematical objects. Aldor's categories can be seen as stencils for implementations of mathematical objects and are not to be mistaken for categories in a mathematical sense[4]. Necessary functions on rings or functions that the various implementations of lists have to implement are examples for categories.

The second layer is the domain layer. A domain takes the stencils provided by the categories and provides implementations of them. Such implementations come typically in the form of abstract data types. Finally, also the data representation is set in this layer. Examples are the integers or double linked lists with integer elements.

The third and last layer is the instantiation layer. Elements of domains belong to this layer. Examples are the integer eleven, a double linked list of the integers from 20 to 25, or a function that doubles every integer argument.

The important step towards the equal treatment of all mathematical objects is that domains, categories, and function's signatures can be used as types for variables. Elements from the instantiation layer, domains, and functions can be used as their values, respectively.

By this generality, functions that map their arguments to functions can be generated. The result is a lot of flexibility, as for example in functional programming. The language does also allow to have functions that map to domains or to categories. `Aldor` takes advantage of this, when defining for example Lists. In contrast to most libraries for other programming languages, `Aldor`'s List is a function. List takes the type of the elements as a parameter. As a consequence, the type of the elements in the list is always known[5].

[4] Nevertheless, Aldor's categories and mathematical categories are closely related. `Algebra` for example provides Aldor categories for basic mathematical domains, such as groups or rings. Although Aldor's category for groups and the category of groups in a mathematical sense both should be a container for groups, there are several differences. `Algebra`'s Group category does not provide morphisms between groups. Morphisms can be built using Aldor, but typically they do not reside in the category itself. Also, Aldor's domains, which correspond to the objects in the terminology of mathematical categories, have to be defined to be in a certain category. In the mathematical setting, objects are in a category due to their properties. However, there are efforts to bring mathematical category theory to Aldor, as for example [56]. Some further discussion of the different semantics of Aldor's categories and those in a mathematical sense can be found in [44].

[5] Although this looks like a restriction at first sight, it indeed is not. By using Aldor's built-in Union type as type for the List's elements, it can be precisely be stated which types are allowed to be in the List.

Furthermore, it is possible to extend a domain without having to alter the domain's source code. In *any* Aldor file, functions or categories can be added to domains. Such extensions do not have to be declared or indicated in the domains original source file. Thereby, even domains from libraries, where the source code is not available, can be extended by further functionality.

Aldor also provides support for throwing and catching exceptions.

Although being object orientated is the state of the art paradigm for computer languages, Aldor is not object orientated. Aldor is more general and object orientated contexts can be easily built within Aldor.

From now on, familiarity with Aldor is assumed. Among various sources for a broader and more detailed introduction, the Aldor tutorial by Peter Broadbery and Manuel Bronstein [33] is recommended. This tutorial does not only give an introduction to the `Aldor` library but does also provide a good and more in-depth introduction to Aldor itself. The Aldor user guide [27] provides a more formal definition of the language. Although this document gives hardly any examples, it serves as a good reference. The Aldor user guide is the only normative document that is publicly available, but it is updated neither regularly nor often. Furthermore, the current version contains several errors. Nevertheless, it is the most precise and most up-to-date source available.

Being familiar with the Aldor language itself, it is important to get an overview of the available libraries.

Discussing the available libraries is started by evaluating the libraries that are shipped with the compiler. For the used compiler, which is the Aldor compiler in version 1.0.2, these are `Aldor` and `Algebra`, both in version 1.0.2.

2.2.2 The `Aldor` library

`Aldor` is the most basic library and can be divided into the following four parts:

- the basic part,
- the supportive part
- the arithmetic part, and
- the data structure part.

In the following paragraphs, each of these parts will be treated separately.

The basic part consists of domains and categories for bootstrapping, such as Category, Type or mappings. The compiler's built-in definitions, data formats and conversion functions are also encapsulated by domains. Other basic domains provide iterators

(Generator) and basic streams for binary data (BinaryReader, BinaryWriter) and character strings (TextReader, TextWriter). Furthermore, domains for characters (Character) and bytes (Byte) belong to this part. There is also a domain Partial, which allows to add a value failed to any domain. With the help of Partial, functions are created that are allowed to fail or Boolean values can be turned into tristates.

The categories of the basic part provide function signatures for testing equality of members of a domain (PrimitiveType) and orderings (PartiallyOrderedType, TotallyOrderedType). There is also a category for copy functions (CopyableType) and categories for input and output of domain members (SerializableType, InputType, OutputType).

The supportive part of `Aldor` provides a domain for file access (File) and analyzing command line arguments (CommandLine). Additionally, there is access to timing functions (Timer) and debugging information (Trace). This part also contains a category that tries to formalize information about libraries (VersionInformationType).

The third part of the library deals with arithmetics. Besides providing built-in implementations of integers (AldorInteger, MachineInteger) and floating point numbers (SingleFloat, DoubleFloat)[6] along with proper categories (IntegerType, FloatType), domains for the integer (GMPInteger) and floating point number (GMPFloat) implementations of the GNU Multiple Precision Arithmetic Library [11] are provided. There is also an implementation of Boolean values (Boolean) along with a category for typical operations, such as or, and, and not on Boolean values (BooleanArithmeticType). IntegerType also exports BooleanArithmeticType and therefore also every implementation of integers has to provide these functions. There is also a general implementation of binary exponentiation (BinaryPowering), a domain for pointers (Pointer) and a random number generator (RandomNumberGenerator). IntegerSegment allows to model segments of integers. With this domain, it is also possible to build segments that contain only every n-th element in a segment. These segments can furthermore be ascending or descending. Therefore, this domain is especially useful to give values for `for` loops. The arithmetic part of `Aldor` also provides an implementation of complex numbers (Complex) and gives several categories (AdditiveType, ArithmeticType, LinearCombinationType, OrderedArithmeticType) for organizing further arithmetic domains.

The last part of this library focuses on data structures. Therefore, a rough partitioning of possible data structures is introduced (DataStructureType, BoundedFiniteDataStructureType, FiniteLinearStructureType, BoundedFiniteLinearStructureType, DynamicDataStructureType). This partitioning is refined by further categories reflecting the form of the data structure (ListType, TableType, ArrayType). Then the categories are populated with domains (List, CheckingList, Hashtable, Array, ...).

`Aldor` has been chosen to serve as the basic library for this report. There currently

[6] All of AldorInteger, MachineInteger, SingleFloat, and DoubleFloat are just wrappers for domains, the Aldor compiler provides. MachineInteger, SingleFloat, and DoubleFloat are mapped to the corresponding native data types of a computers processor and therefore allow most efficient operations. AldorInteger is mapped to an implementation of integers of arbitrary size. Since there currently exists no processor, which can natively deal with such integers, this type is simulated in software by the Aldor compiler.

exists no other library that can serve as a basis and would make the compiler's features available to the developer. But even if there was an alternative, `Aldor` would very likely be chosen again in order to guarantee the usability of the implementation for this report. If a non-standard base library would have been used for the implementation, it would also become necessary for users to install these non-standard libraries in order to be able to use this implementation. A cumbersome installation procedure might possibly have been the result.

2.2.3 The `Algebra` library

The second library that is shipped with the Aldor compiler is `Algebra`. This library builds on top of `Aldor` and enrichens the Aldor world with basic mathematical concepts. `Algebra` can be separated into these eight parts:

- the basic categories part,
- the commutative algebra part,
- the finite fields part
- the linear algebra part,
- the univariate polynomials and series part,
- the multivariate polynomials part,
- the parsing part, and
- the supportive part.

The first part provides categories for a vast amount of mathematical structures. Domains for these categories are presented in the following five parts. The last but one part is dealing with converting to and from expression trees. Further domains, that do not fit in the previous seven parts build the last part.

The basic categories part provides implementations of groups (Group, AbeleanGroup, rings (Ring, CommutativeRing, PolynomialRing) or fields (Field, FiniteField). Besides such well distinguished structures, categories such as FiniteCharacteristic or CharacteristicZero for rings allow to make implicit knowledge about structures explicit.

The commutative algebra part introduces fractions (Fraction, FractionBy) and also provides algebraic extensions (SimpleAlgebraicExtension). Additionally, the domains for automorphisms (Automorphism), for derivations (Derivation), and for performing Chinese remaindering (ChineseRemaindering) belong to this part.

The next part provides domains (SmallPrimeField, PrimeField2) and further categories (PrimeFieldCategory) for finite fields with a prime number of elements. There are also domains for retrieving small prime numbers efficiently (SmallPrimes, WordSizePrimes).

The linear algebra part provides domains for vectors (Vector) and matrices (DenseMatrix, LinearAlgebra). With these domains various elimination algorithms are implemented (OrdinaryGaussElimination, DivisionFreeGaussElimination, ...). Additionally, `Algebra` provides a domain for permutations on a fixed number of elements (Permutation).

Domains (DenseUnivariatePolynomial, SparseUnivariatePolynomial) and categories (UnivariatePolynomialAlgebra) for univariate polynomials are defined in the fifth part. There are also categories for further refinement (UnivariateGcdRing, FactorizationRing, FFTRing) and further algorithms (Resultant, UnivariateIntegralFactorizer) for univariate polynomials. Additionally, this part also provides a domain (DenseUnivariateTaylorSeries) and a category (UnivariateTaylorSeriesCategory) for dealing with series.

In the multivariate polynomials part, the `Algebra` library provides several implementations of multivariate polynomials (DistributedMultivariatePolynomial0, SparseMultivariatePolynomial, ...) and several categories for these implementations (PolynomialRing, PolynomialRing0, ...). Since different implementations of multivariate polynomials are important in this report's implementation, the multivariate part is discussed in more detail in Section 3.

`Algebra`'s parsing part provides an implementation of expression trees (ExpressionTree, ExpressionTreeLeaf) along with various nodes that act as operators (ExpressionTreePlus, ExpressionTreeTimes, ...). Furthermore, a parser for parsing strings in infix notation to expression trees (InfixExpressionParser) and another parser for parsing strings in Lisp's prefix notation (ListExpressionParser) are provided. Along with these two parsers, several domains (Token, SymbolTable, ...) ease the development of further parsers.

The last part provides a domain for passing commands to MAPLE[7](Maple) and a domain for read-only strings (Symbol).

The `Algebra` library is used for this report's work. The parts that will be reused are especially the implementations of multivariate polynomial rings and the parsing part.

There are further libraries available for Aldor, but none of these libraries aids in implementing the desired algorithms. For this reason, they are neither used nor covered in this discussion.

2.3 Overview of the implementation

During the work on this report, three libraries have been implemented. These libraries are `ExtIO` [9], `AldorUnit` [4], and `CharSet` [6]. `CharSet` provides the characteristic set algorithms, while `ExtIO` and `AldorUnit` support the implementation of `CharSet`.

[7]To pass commands to MAPLE is not to be mistaken for a part of the second objective of this report, which is to connect the implementation to MAPLE and MATHEMATICA. `Algebra` provides a way to call MAPLE from within Aldor, but no way to call Aldor from within MAPLE. And the latter is needed to achieve the second objective.

First, the purpose of each of these libraries is presented in short[8]. Afterwards, the separation of the functionality into three libraries is discussed. Finally, the dependencies of the libraries are discussed.

The library `ExtIO` extends several domains of `Aldor` and `Algebra` to ease input and output of these domains' values, as for example List is extended by adding a function to turn a list into an expression tree. General purpose domains such as StringTokenizer, for chopping strings into substrings, or a table of ANSI characters (AnsiCodes) also belong to `ExtIO`. Additionally, a common base category for exceptions (ExceptionType) is introduced. The most important aspect of `ExtIO` in the context of this report's work is a computer algebra system framework. This framework allows to format Aldor's expression trees in the syntax of MAPLE or MATHEMATICA. Additionally, it allows to build expression trees of strings in the syntax of MAPLE and MATHEMATICA[9]. A more detailed description of the computer algebra system framework can be found in Section 7.

`AldorUnit` provides a test framework easing programmer tests in Aldor. It supports developers in writing tests, executing tests and finding relevant information for debugging source code. This library has been implemented for testing the Aldor code of this report's work. However, it can serve as a general purpose testing library and is in many ways similar to Java's JUNIT. Further details about `AldorUnit` can be found in `AldorUnit`'s documentation at [4].

`CharSet` provides the implementation of differential polynomials, differential reductions and characteristic set algorithms. It can also be connected to MAPLE and MATHEMATICA. Furthermore, `CharSet` provides a command line utility to be able to apply the algorithms without having to use an Aldor environment, MAPLE, or MATHEMATICA.

The separation of the implemented code into the libraries `ExtIO`, `AldorUnit`, and `CharSet` decouples three major aspects of the implementation. These aspects are enhancing the core libraries and interfacing computer algebra systems (`ExtIO`), testing the implementation (`AldorUnit`), and algorithms for characteristic set computations (`CharSet`).

This decoupling eases the reuse of the developed software. Any Aldor project can use `AldorUnit` directly for performing programmer tests. Any Aldor project can use `ExtIO` directly for interfacing different computer algebra systems. Without this decoupling it would not be possible to use one part of the implementation without the others.

As can be seen from Figure 2.1, the characteristic set implementation relies on the two standard libraries `Aldor` and `Algebra` that are shipped with the Aldor compiler. Additionally, it takes advantage of `ExtIO` library for its input and output capabilities,

[8]The libraries' features that are necessary for achieving this report's goals are covered more detailed in the corresponding sections afterwards. A full treatment of each of the libraries can be found in the corresponding library's documentation on the libraries' homepages.

[9]The `Aldor` library already has some support for MAPLE's syntax. This, however, could not be used conveniently to incorporate further computer algebra systems. The implementation of `ExtIO` provides MAPLE and MATHEMATICA support and can be extended to support further computer algebra systems easily.

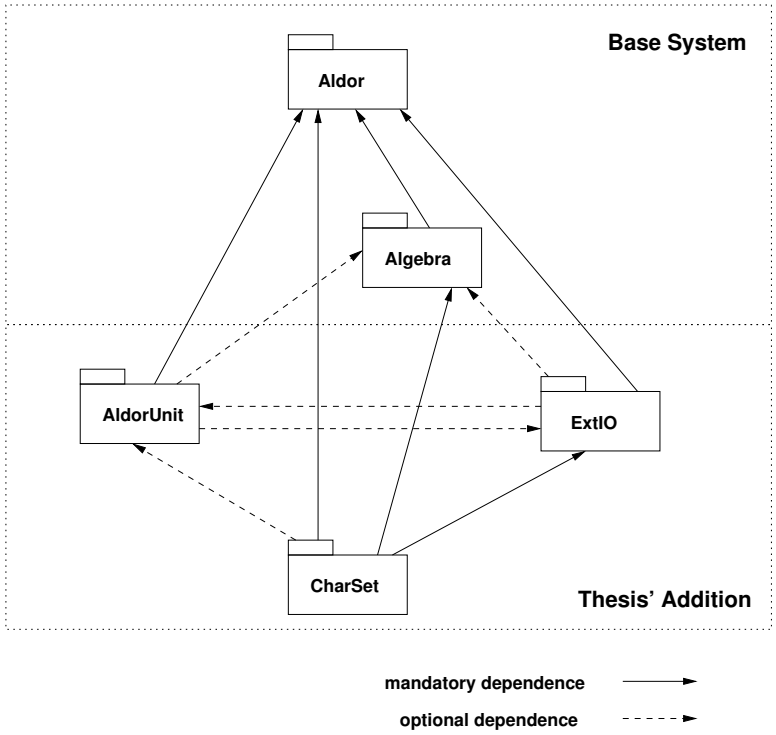

Figure 2.1: The dependencies of the used Aldor libraries

especially the computer algebra system part. Furthermore, `CharSet` uses `AldorUnit` in order to test the built libraries.

`ExtIO` depends on `Aldor` and for most parts also on `Algebra`. However, `ExtIO` can be built even without `Algebra`. Again, `AldorUnit` is used to test the `ExtIO` library.

`AldorUnit` depends on `Aldor`. For providing tests for types of `Algebra`, `AldorUnit` depends on `Algebra`. Furthermore, `ExtIO` is used by `AldorUnit` for providing more information about the performed tests. Nevertheless, `AldorUnit` is fully functional without either or both of `Algebra` and `ExtIO`.

3 Implementing differential polynomials

The central goal of this report is an implementation of algorithms for differential characteristic sets. These algorithms act on elements of differential polynomial rings. Aldor and it's core libraries do not provide differential polynomial rings. This section describes this report's implementation of differential polynomial rings. The further sections of this report use these differential polynomial rings to introduce differential reduction and finally the algorithms for computing differential characteristic sets.

Before discussing the implementation of differential polynomial rings, some notation is introduced. With the help of this notation the splitting of the implementation of differential polynomial rings into three parts is motivated. Finally these three parts, which are

- implementing differential rings,
- implementing derivatives, and
- extending polynomial rings to differential polynomial rings by adding differential structure,

are presented.

For the rest of this report, let $R\{Y\}$ be a differential polynomial ring with $Y = \{y_1, y_2, \ldots, y_m\}$ being differential indeterminates and $\Delta = \{\delta_1, \delta_2, \ldots, \delta_n\}$ the derivations of $R\{Y\}$[10]. Additionally, let Θ denote the free commutative monoid generated by Δ and let $\Theta Y = \{\theta y \mid \theta \in \Theta \wedge y \in Y\}$. The latter are called derivatives. These derivatives ΘY denote juxtaposing elements of Θ and Y and do not indicate an application of a derivation. For describing the implementation it is of advantage to distinguish between ΘY and its embeddings into $R[\Theta Y]$ and $R\{Y\}$. So $\Theta Y \not\subseteq R[\Theta Y]$ and $\Theta Y \not\subseteq R\{Y\}$. Although ΘY can be embedded into both $R[\Theta Y]$ and $R\{Y\}$, the distinction between ΘY and their embedding into the polynomial rings is essential. These embeddings are a bijection between the derivatives ΘY and the indeterminates of $R[\Theta Y]$ or $R\{Y\}$. Therefore, "derivative" is used to refer to elements of ΘY and "indeterminate" to refer to their embedded counterparts in the polynomial rings.

`Algebra` provides several implementations of polynomials (DistributedMultivariatePolynomial0, DistributedMultivariatePolynomial1, RecursiveMultivariatePolynomial0, ...). These implementations cover polynomial rings but do not cover differential polynomial rings. When omitting the differential structure of $R\{Y\}$, the resulting polynomial ring is isomorphic to the polynomial ring $R[\Theta Y]$. With a proper implementation of ΘY, `Algebra`'s implementations of multivariate polynomials can be used to implement

[10] Although the notation $R\{Y\}$ for differential polynomials only carries the parameters R and Y, $R\{Y\}$ does also depend on Δ. We would prefer a notation that also uses the parameter Δ (for example $R\{Y\}_\Delta$). However, in literature (see for example [45] or [48]) $R\{Y\}$ is constantly used without Δ. Δ is constantly omitted. To avoid introducing new notation and confusion, this report also uses $R\{Y\}$ to denote the differential polynomial rings. The parameter Δ is implicit and although necessary cannot be seen in the notation.

$R[\Theta Y]$. By adding differential structure to $R[\Theta Y]$, $R\{Y\}$ can be modelled. This report pursues this approach and tries to reuse `Algebra`'s implementation of multivariate polynomial rings. The implementation of differential polynomial rings is therefore separated into the three parts

- implementing differential rings,
- implementing the derivatives (ΘY), and
- extending polynomial rings ($R[\Theta Y]$) to differential polynomial rings ($R\{Y\}$) by adding differential structure.

3.1 Implementing differential rings

A description of how differential structure is implemented for this report can be found in this part. This structure is later on used to model differential rings. Further sections reuse this structure to model the derivatives and to add differential structure to $R[\Theta Y]$.

This part starts by motivating and presenting DifferentialType, which models the differential structure. Afterwards DifferentialRational is introduced.

`Algebra` already provides a category for differential rings with only one derivation (DifferentialRing). This category is not reused, since this restriction to only one derivation is considered to strong. It is desired that differential rings with more than one derivation can be implemented.

`CharSet` introduces the category DifferentialType, which bears functions for a differential structure. A differential ring is modelled by writing a domain that exports both, `Algebra`'s Ring and `CharSet`'s DifferentialType. Similarly, different mathematical structures are modelled by adding DifferentialType to the original structure.

Before discussing the functions of DifferentialType, a remark about the derivations is made. Derivations are mappings on rings. There is no mathematical requirement to attach a name to these functions. Functions to apply derivations to elements typically require a way to determine which derivation is to be applied. Although Aldor allows to pass functions as parameters, which allows to pass the derivations directly as parameters, DifferentialType uses numbers and Symbols[11] to refer to derivations. Section 6 discusses this and different approaches to referring to derivations. For DifferentialType, the number i is used to refer to the derivation δ_{i+1}, which is the $(i+1)^{th}$ derivation of Δ. The domains implementing DifferentialType define which Symbol is used to refer to which derivation. Throughout the discussion DifferentialType, a either a number or a Symbol can be used as designator for a derivation.

DifferentialType's most important function is differentiate, which is used to apply a derivation to an element. This function takes the element to apply the derivation to as

[11] The domain Symbol models read only character strings and is implemented in `Algebra`. Symbol will move from the `Algebra` to the `Aldor` library in future versions of `Aldor` and `Algebra`.

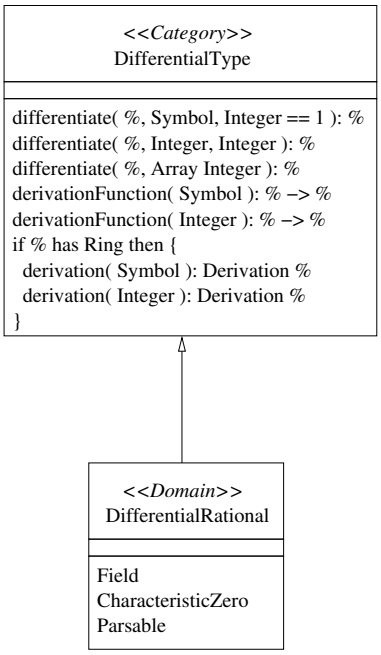

Figure 3.1: The categories for derivations

its first parameter and a designator for the derivation to apply as second parameter. A third parameter indicates how often the designated derivation is to be applied. This third parameter is optional, when a Symbol is used as designator for the derivation. When using an integer as designator, the third parameter is not optional in order to avoid name conflicts with `Algebra`'s DifferentialRing[12]. Calling `differentiate(a,4,2)` gives $\delta_5(\delta_5(a))$[13].

Furthermore, DifferentialType provides a differentiate function that takes the element to apply derivations to and an array of integers as parameters. The integers in this array indicate how often the derivation are to be applied. For example `[2,0,1]` would indicates to apply the first derivation twice and the third derivation once. The second derivation is not applied at all. As a result, calling `differentiate(a,[2,0,1])` gives $(\delta_1 \circ \delta_1 \circ \delta_3)(a)$.

As can be seen in Figure 3.1, DifferentialType also allows to obtain the mappings of the derivations by calling derivationFunction with a designator as parameter. Additionally, functions for retrieving these mappings as Derivations[14] are provided, if the domain

[12] `Algebra`'s DifferentialRing provides a function differentiate that takes two parameters. These parameters are the element to apply the derivation to and an Integer that indicates how often the derivation is to be applied. If the third parameter in DifferentialType's differentiate with an integer designator would also be optional, this would conflict with the differentiate function of DifferentialRing. While bearing different semantics, both functions would be named "differentiate" and would be called with an element of the domain followed by an integer.

[13] Since the designator i is used to refer to δ_{i+1}, `4` is used to refer to δ_5.

[14] `Algebra` provides the domain Derivation for modelling derivations. This domain allows to apply a derivation to elements of the domain on which the derivations act. Furthermore, Derivation comes with functions to convert mappings to Derivations and Derivations to mappings. Derivation also provides a module structure for the derivations.

that implements DifferentialType is a ring.

DifferentialType does not have any parameters. Neither the number of derivations in the domain nor designators of derivations occur as parameter to DifferentialType.

Section 3.2 gives a further treatment of DifferentialType and its semantics in a non-ring context.

`CharSet` provides several implementations of DifferentialType. As can be seen later, the differential polynomials and the implementation of the derivatives have DifferentialType.

Additionally, DifferentialRational is implemented in `CharSet`. This domain models the rational numbers with derivations that map any parameter to the 0 element and has DifferentialType.

3.2 Implementing the derivatives

This part describes the implementation of derivatives (ΘY) and the reuse of DifferentialType for modelling Θ's structure in ΘY. The presented implementation of ΘY is used to model $R[\Theta Y]$ with `Algebra`'s implementations of multivariate polynomial rings.

First, the requirements that arise when using the derivatives as indeterminates for polynomials in $R[\Theta Y]$ are investigated. Afterwards, available implementations in `Algebra` are discussed and the implementation itself is presented.

The efforts of `Algebra` towards implementing multivariate polynomials are to be reused when modelling $R[\Theta Y]$. However, `Algebra`'s categories and domains for polynomial rings, have different requirements on the domain for the coefficients and the domain of indeterminates, for which they model multivariate polynomial rings. The domain for the coefficients of the polynomial implementations of `Algebra` have to satisfy the type Ring[15] or even weaker conditions. In the scope of this report, the coefficients domain is always at least a ring. Therefore, the necessary condition on the coefficient domain is met for all available implementations of `Algebra`. `Algebra`'s categories for polynomial rings require the domain for the indeterminates of polynomials only to be of TotallyOrderedType[16] and ExpressionType[17]. The implementations (DistributedMultivariatePolynomial1, RecursiveMultivariatePolynomial0, ...) of these polynomial ring categories typically require the domain for the indeterminates of the polynomial rings to be of VariableType[18].

Therefore, it is necessary that ΘY is of type VariableType.

[15] `Algebra` defines the category Ring. Ring is used to model rings with 1.

[16] TotallyOrderedType is implemented in `Aldor` and used to define a total order on the elements of a domain that implements this category.

[17] A domain that implements `Algebra`'s ExpressionType provides functions to test elements for equality, converting them to expression trees, and printing them to a stream.

[18] Besides providing TotallyOrderedType and ExpressionType, domains that implement VariableType have to implement functions to convert expression trees to the domain, calculate hash values, and convert elements to and from Symbol. VariableType is defined in the `Algebra` library.

`Algebra` provides three implementations of VariableType. Two (OrderedVariableList, OrderedVariableTuple) are also of FiniteVariableType[19]. Since ΘY is in general an infinite set, these two domains cannot be reused. The other implementation of VariableType (OrderedSymbol) can handle ΘY's infinitely many elements, but does neither reflect Θ's structure within ΘY nor can it be easily adopted to do so. However, for differential reduction, which is introduced in Section 4 and constitutes an important ingredient for most characteristic set algorithms, Θ's structure within ΘY is important.

No domain of `Algebra` can be used to model ΘY adequately, only the category VariableType is reused.

`CharSet` provides the category DifferentialVariableType[20], which extends VariableType and DifferentialType by function signatures that are specific for derivatives. Domains implementing this category will be used for modelling the derivatives.

Before discussing DifferentialVariableType's own functions, the reuse of DifferentialType is discussed and motivated via ΘY and its connections to $R\{Y\}$.

ΘY is only a set and has no structure. Therefore, it is not possible to define a derivation on ΘY, as this definition would involve addition and multiplication, which are both missing in ΘY. Nevertheless, Θ has a structure. Θ is a commutative monoid. When modelling ΘY in Aldor, Θ's structure is incorporated. This additional structure is used to define a mapping on the elements of ΘY that acts like applying derivations to the indeterminates in a polynomial differential ring. With the help of this mapping, upgrading polynomial rings to differential polynomial rings is supported, as can be seen in Section 3.3.

The elements of Θ are derivations on $R\{Y\}$ and cannot act directly on ΘY[21]. However, ΘY is in one-to-one correspondence with the indeterminates of $R\{Y\}$. Therefore, ΘY can be embedded into $R\{Y\}$ by a mapping ι, which associates every element θy of ΘY with the result of applying the derivation θ to y's corresponding indeterminate in $R\{Y\}$, where $\theta \in \Theta$ and $y \in Y$. ι is a bijective map from ΘY to the indeterminates of $R[Y]$. ϕ is used for the inverse map of ι on the image of ι. Since the derivations of $R\{Y\}$ map an indeterminate to another indeterminate, the mapping $(\phi \circ \bar{\theta} \circ \iota)$, with $\bar{\theta} \in \Theta$, acts on ΘY as $\bar{\theta}$ acts on the corresponding indeterminates. Therefore, $(\phi \circ \bar{\theta} \circ \iota)$ gives a meaning to "applying a derivation of $R\{Y\}$ to elements of ΘY". This mapping is illustrated in Figure 3.2. Nevertheless, as explained in the previous paragraph, the mapping $\phi \circ \bar{\theta} \circ \iota$ is no derivation on ΘY.

This further aspect of modelling ΘY in Aldor, allows to delegate differentiation on the indeterminates in $R\{Y\}$ to ΘY when modelling $R\{Y\}$. Therefore, DifferentialVariableType has to provide functions for "applying" derivations to derivatives. These functions

[19] `Algebra`'s FiniteVariableType is a refinement of VariableType for treating only finitely many variables. In addition to VariableType, FiniteVariableType provides functions to retrieve a list of all the domain's variables and functions to convert the domain's elements to and from integers.

[20] In order to match the namings within `Algebra`, which puts emphasis on the word "variable", the category is called neither DerivativeType nor DifferentialIndeterminateType but DifferentialVariableType. DifferentialVariableType is the category for domains that model derivatives.

[21] At this point, it is important to note again that in this report $\Theta Y \not\subseteq R\{Y\}$. Nevertheless, ΘY can be embedded into $R\{Y\}$, which is done in the further discussion.

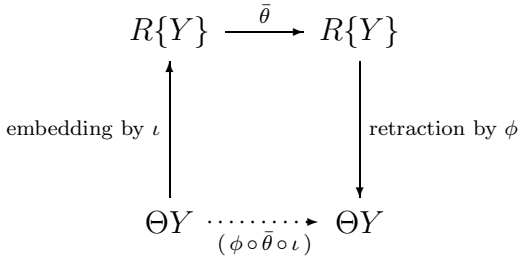

Figure 3.2: Embedding derivatives to apply derivations

have to cover the same functionality as those functions that allow to apply derivations to differential rings. For differential rings, DifferentialType is used. Since domains that implement DifferentialType do not necessarily have to be a ring, DifferentialType can be and is reused for modelling the derivatives and provides the functions for "applying" the derivations of $R\{Y\}$ to elements of ΘY.

Besides simplifying the implementation of differential polynomial rings, the use of DifferentialType to incorporate Θ's structure to ΘY manifests the general usability of DifferentialType for modelling structures that are connected with derivations. By DifferentialType, derivatives, differential rings, and all further differential algebras share common function signatures to operate with derivations.

However, DifferentialType does not provide all functions in all contexts, as can be seen in Figure 3.1. For DifferentialVariableType, DifferentialType does not provide functions to retrieve a derivation mapping as a Derivation object, since ΘY does not provide a ring's structure. Nevertheless, the associated mapping can be retrieved, although only as a plain mapping and not as Derivation, via derivationFunction.

As can be seen in Figure 3.3, DifferentialVariableType takes one parameter, which is of FiniteVariableType. This parameter corresponds to Y in ΘY. Although Y in general does not have to be finite, FiniteVariableType and not VariableType is chosen for Y's type, since there are more implementations of FiniteVariableType available in `Algebra` and in the typical applications Y is finite.

DifferentialVariableType carries three variable functions for generating elements. One function takes an integer i as argument and gives the derivative of class i and total order 0. Another one takes additionally an array of integers that represent the orders with respect to the derivations. The third function takes an element of Y and an array of integers for the orders. Besides functions for converting elements of Y to the domain that implements DifferentialVariableType, this category also provides functions for retrieving the class (class), the orders with respect to the derivations (order), and the total order (totalOrder). Additionally, a function for determining if a derivative can be reached by applying at least one derivation to another derivative (isProperDerivative?)[22] is implemented.

[22]The question mark in isProperDerivative? is part of the identifier, as Aldor allows to form identifiers containing question marks. Valid identifiers may contain arbitrarily many question marks, as long as the identifier's first character is not a question mark. Question marks are typically used, when a function maps to Boolean.

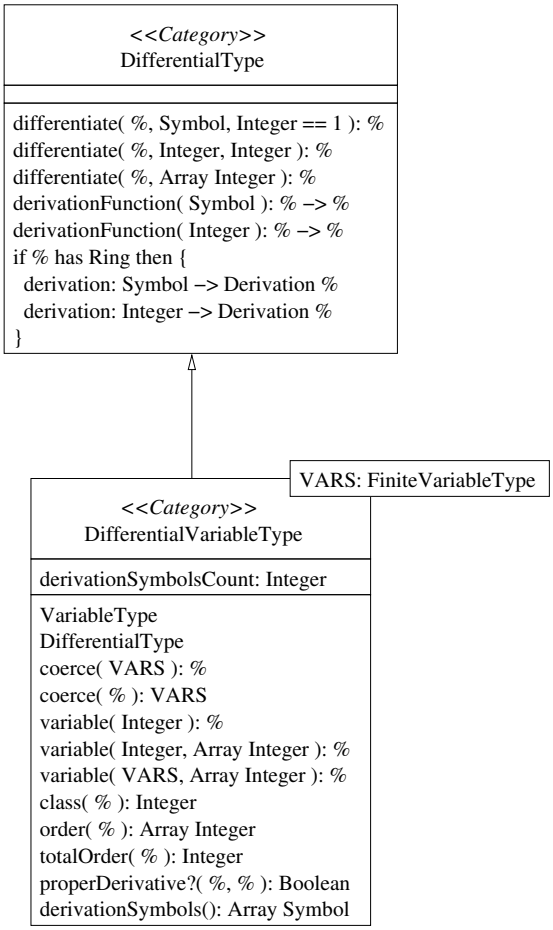

Figure 3.3: The categories for derivatives

For further use in the implementation of differential polynomial rings, DifferentialVariableType also bears the number of derivations (derivationSymbolsCount) that are defined and their Symbols (derivationSymbols).

DifferentialVariableType is also of TotallyOrderedType. On the one hand, this total order comes from satisfying VariableType, which has TotallyOrderedType. On the other, a total order on the elements of a domain implementing DifferentialVariableType is necessary for the reduction algorithm, which will be presented in section Section 4[23].

DifferentialVariable implements DifferentialVariableType and is used as implementation of ΘY. It takes three parameters: Y, the Symbols for the derivations, and an implementation of DifferentialVariableOrderToolsType. DifferentialVariableOrderToolsType models a total order on the differential variables and is described in Section 3.2.1. If the order is an elimination order[24], DifferentialVariable also exports EliminationOrderedDifferentialVariableType. This category is a refinement of DifferentialVariableType that does not provide further functions, but marks the domain's variables as having an elimination order. In Section 4, this category is used to single out domains of DifferentialVariableType for which a faster reduction algorithm can be applied.

For settings with only one derivation, `CharSet` provides OrdinaryDifferentialVariableType. This category extends DifferentialVariable by functions with simpler signatures that are only valid when there is only one derivation. OrdinaryDifferentialVariableType provides for example a function order giving the order of an element as integer. For DifferentialVariableType, the order can only be retrieved as an array of integers.

By OrdinaryDifferentialVariable `CharSet` provides an implementation of OrdinaryDifferentialVariableType. OrdinaryDifferentialVariable takes three parameters: Y, a Symbol for the derivation[25] and an order. Like DifferentialVariable, OrdinaryDifferentialVariable exports EliminationOrderedDifferentialVariableType, if its order is an elimination order.

3.2.1 Orders on differential variables

This part discusses the different domains and categories for total orders on differential variables.

First, the categories for total orders on differential variables are presented. Then, their implementations are discussed.

`CharSet` provides a separate category for total orders on differential variables, which is called DifferentialVariableOrderToolsType. DifferentialVariableOrderToolsType has

[23]For differential reduction to be finite even a total order does not suffice. The necessary, further conditions (see property (∗) on page 42) cannot be modelled in Aldor. Therefore, TotallyOrderedType is used and the other necessary properties are assumed. More on this topic can be found in Section 4 and Section 6.

[24]See Section 3.2.1 for an explanation of "elimination order".

[25]Note again that OrdinaryDifferentialVariable is used for modelling derivatives when there is only a single derivation. Therefore, it suffices to specify only a single Symbol for the derivation.

only one function. The parameters to this function are Y and an implementation of DifferentialVariableType for Y. Calling this function returns a function that allows to compare two derivatives.

`CharSet` provides a refinement of DifferentialVariableOrderToolsType by introducing DifferentialVariableEliminationOrderToolsType for marking orders as elimination orders. An elimination order is a total order $\leq$ on ΘY for which

$$\forall \theta, \bar{\theta}, \hat{\theta} \in \Theta \ \forall \bar{y}, \hat{y} \in Y \ : \ (\bar{y} \neq \hat{y} \wedge \bar{\theta}\bar{y} \leq \hat{\theta}\hat{y}) \implies (\theta\bar{\theta})\bar{y} \leq \hat{\theta}\hat{y}$$

holds. This property allows short cuts for the reduction algorithm, which are explained in Section 4. Further properties of orders on the derivatives ΘY are mentioned for example in [45]. Since these further properties cannot be exploited for making computations more efficient, they are not incorporated in this report' implementation.

As Figure 3.4 shows, there are five implementations of DifferentialVariableOrderToolsType available. Two orders implement DifferentialVariableEliminationOrderToolsType (DifferentialVariableLexicographicEliminationOrderTools, DifferentialVariableOrderlyEliminationOrderTools) and the other three do not (DifferentialVariableOrderlyOrderTools, DifferentialVariableLexicographicOrderTools, DifferentialVariableMixedOrderTools).

All five implementations extend a total order on Y[26] to a total order on ΘY that respects the order on Y. The only domain that accepts further parameters is DifferentialVariableMixedOrderTools. After discussing these parameters, the order relation of each implementation of DifferentialVariableOrderToolsType is presented.

DifferentialVariableMixedOrderTools allows to group derivatives by their classes and apply different orders on each group. This domain takes a List of pairs as parameter. Each pair denotes a grouping of differential variables and consists of a domain satisfying DifferentialVariableOrderToolsType and an Integer. The Integer indicates how many classes of derivatives are to collect in the group, while the parameter of type DifferentialVariableOrderToolsType specifies the order of this group. Those classes of derivatives that are not in any group are ordered by DifferentialVariableLexicographicEliminationOrderTools.

For the presentation of `CharSet`'s orders, assume

$$\forall i, j \in \{1, 2, \ldots, n\} : y_i < y_j \iff i < j.$$

[26] In the general case, Y does not provide an order. However, as mentioned above, Y is passed as VariableType. This type carries a total order. Therefore, representing Y in Aldor forces a total order on the elements of Y.

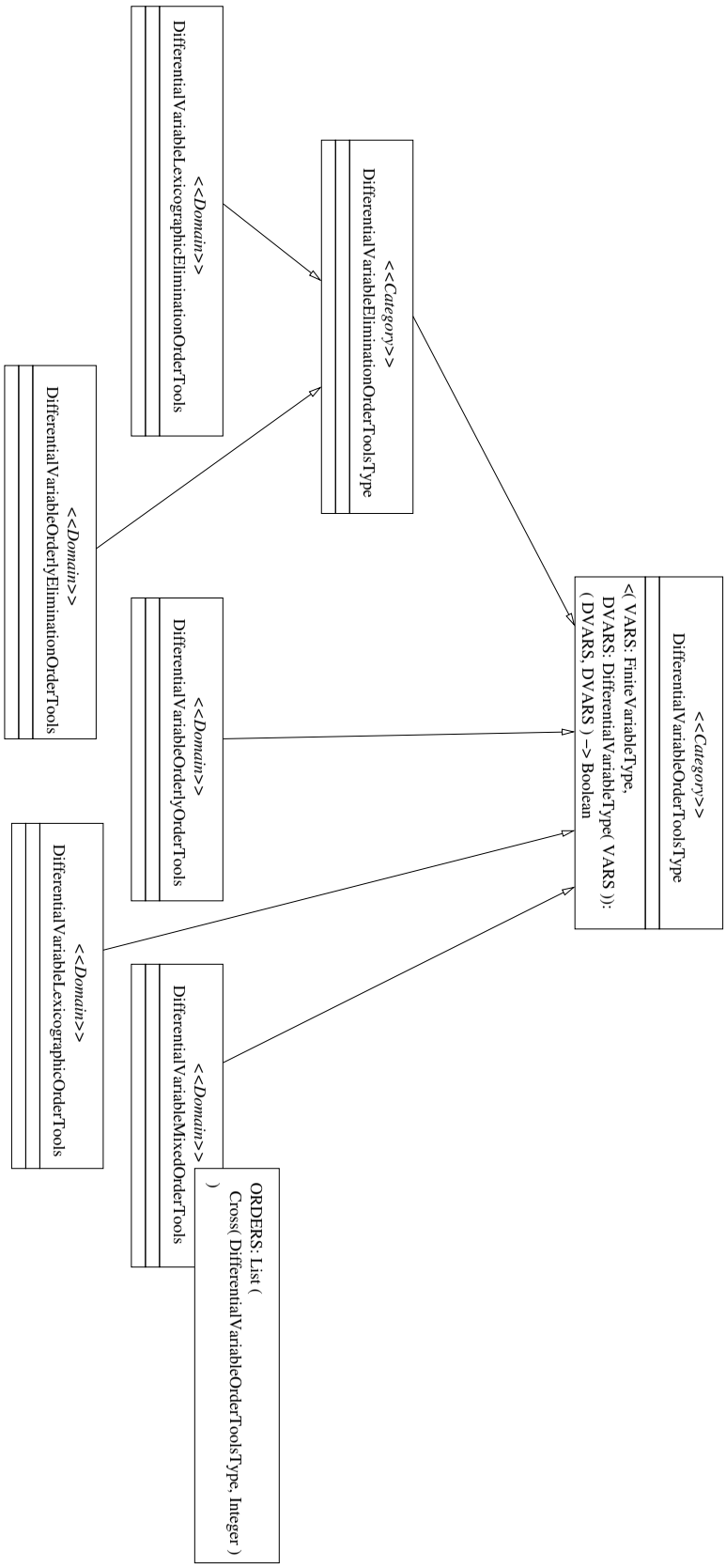

Figure 3.4: The categories and domains for orders on derivatives

Furthermore, the following notation and abbreviations are used[27] :

$$\begin{aligned}
&\forall i \in \{1,2,\ldots,n\}\ \forall \theta \in \Theta : \deg_{\delta_i}(\theta) := \max\left\{ d \in \mathbb{N}_0 \;\middle|\; \exists \bar{\theta} \in \Theta : \delta_i^d \bar{\theta} = \theta \right\} \\
&\forall \theta \in \Theta : \deg(\theta) := \textstyle\sum_{i=1}^{n} \deg_{\delta_i}(\theta) \\
&\mathit{condLex} := \exists k \in \{1,2,\ldots,n\} : \deg_{\delta_k}(\theta) < \deg_{\delta_k}(\bar{\theta}) \quad \wedge \\
&\quad \wedge \quad \forall l \in \{k+1, k+2, \ldots, n\} : \deg_{\delta_l}(\theta) = \deg_{\delta_l}(\bar{\theta}) \\
&\mathit{condOrd} := \exists k \in \{1,2,\ldots,n\} : \deg_{\delta_k}(\theta) > \deg_{\delta_k}(\bar{\theta}) \quad \wedge \\
&\quad \wedge \quad \forall l \in \{1,2,\ldots,k-1\} : \deg_{\delta_l}(\theta) = \deg_{\delta_l}(\bar{\theta})
\end{aligned}$$

- `DifferentialVariableLexicographicEliminationOrderTools`

$$\begin{aligned}
&\forall i,j \in \{1,2,\ldots,n\}\ \forall \theta, \bar{\theta} \in \Theta : \theta y_i < \bar{\theta} y_j \iff i < j \vee \\
&\quad \vee \quad (i = j \wedge \mathit{condLex})
\end{aligned}$$

  Example:

$$\begin{aligned}
&y_1 < \delta_1 y_1 < \delta_1\delta_1 y_1 < \ldots < \delta_2 y_1 < \delta_2\delta_1 y_1 < \ldots < \\
&< \delta_2\delta_2 y_1 < \ldots < y_2 < \delta_1 y_2 < \delta_1\delta_1 y_2 < \ldots
\end{aligned}$$

- `DifferentialVariableOrderlyEliminationOrderTools`

$$\begin{aligned}
&\forall i,j \in \{1,2,\ldots,n\}\ \forall \theta, \bar{\theta} \in \Theta : \theta y_i < \bar{\theta} y_j \iff i < j \vee \\
&\quad \vee \quad (i = j \wedge \deg(\theta) < \deg(\bar{\theta})) \\
&\quad \vee \quad (i = j \wedge \deg(\theta) = \deg(\bar{\theta}) \wedge \mathit{condOrd})
\end{aligned}$$

  Example:

$$\begin{aligned}
&y_1 < \delta_1 y_1 < \delta_2 y_1 < \ldots < \delta_1\delta_1 y_1 < \delta_2\delta_1 y_1 < \ldots < \\
&< \delta_2\delta_2 y_1 < y_2 < \delta_1 y_2 < \delta_2 y_2 < \ldots
\end{aligned}$$

- `DifferentialVariableOrderlyOrderTools`

$$\begin{aligned}
&\forall i,j \in \{1,2,\ldots,n\}\ \forall \theta, \bar{\theta} \in \Theta : \theta y_i < \bar{\theta} y_j \iff \deg(\theta) < \deg(\bar{\theta}) \vee \\
&\quad \vee \quad (\deg(\theta) = \deg(\bar{\theta}) \wedge \mathit{condOrd}) \\
&\quad \vee \quad (\theta = \bar{\theta} \wedge i < j)
\end{aligned}$$

  Example:

$$\begin{aligned}
&y_1 < y_2 < \ldots < \delta_1 y_1 < \delta_1 y_2 < \ldots < \delta_2 y_1 < \delta_2 y_2 < \ldots < \\
&< \delta_1\delta_1 y_1 < \delta_1\delta_1 y_2 < \ldots < \delta_2\delta_1 y_1 < \delta_2\delta_1 y_2 < \ldots
\end{aligned}$$

- `DifferentialVariableLexicographicOrderTools`

$$\begin{aligned}
&\forall i,j \in \{1,2,\ldots,n\}\ \forall \theta, \bar{\theta} \in \Theta : \theta y_i < \bar{\theta} y_j \iff \mathit{condLex} \vee \\
&\quad \vee \quad (\theta = \bar{\theta} \wedge i < j)
\end{aligned}$$

  Example:

$$\begin{aligned}
&y_1 < y_2 < \ldots < \delta_1 y_1 < \delta_1 y_2 < \ldots < \delta_1\delta_1 y_1 < \delta_1\delta_1 y_2 < \ldots < \\
&< \delta_2 y_1 < \delta_2 y_2 < \ldots < \delta_2\delta_1 y_1 < \delta_2\delta_1 y_2 < \ldots
\end{aligned}$$

[27] The lexicographic property is modelled by the abbreviation *condLex*. Although some readers may expect the index l of *condLex* to be in $\{1,2,\ldots,k-1\}$, the index l is in $\{k+1,k+2,\ldots,n\}$. This setting has no distinguished mathematical requirement, but has been chosen to meet the examples of [39].

- `DifferentialVariableMixedOrderTools( [ ( DifferentialVariableLexicographicOrderTools, s ) ] )` (for some integer `s`)[28]

$$\begin{aligned}
&\forall i,j \in \{1,2,\ldots,s\}\ \forall \theta,\bar{\theta} \in \Theta : \theta y_i < \bar{\theta} y_j \iff condLex \vee \\
&\quad \vee \quad (\theta = \bar{\theta} \wedge i < j) \\
&\forall i \in \{1,2,\ldots,s\}\ \forall j \in \{s+1,s+2,\ldots,n\}\ \forall \theta,\bar{\theta} \in \Theta : \theta y_i < \bar{\theta} y_j \\
&\forall i,j \in \{s+1,s+2,\ldots,n\}\ \forall \theta,\bar{\theta} \in \Theta : \theta y_i < \bar{\theta} y_j \iff i < j \vee \\
&\quad \vee \quad (i = j \wedge condLex)
\end{aligned}$$

  Example:

$$\begin{aligned}
&y_1 < y_2 < \ldots < y_s < \delta_1 y_1 < \delta_1 y_2 < \ldots < \delta_1 y_s < \delta_1\delta_1 y_1 < \delta_1\delta_1 y_2 < \ldots < \\
&< \delta_1\delta_1 y_s < \ldots < \delta_2 y_1 < \delta_2 y_2 < \ldots < \delta_2 y_s < \delta_2\delta_1 y_1 < \delta_2\delta_1 y_2 < \ldots < \\
&< y_{s+1} < \delta_1 y_{s+1} < \ldots < \delta_2 y_{s+2} < \ldots < y_{s+2} < \ldots
\end{aligned}$$

Figure 3.5 gives all the discussed relations of categories and domains in one picture.

3.3 Extending polynomial rings to differential polynomial rings by adding differential structure

With the help of the implementation of ΘY from Section 3.2 and `Algebra`'s domains for multivariate polynomial rings, it is possible to build $R[\Theta Y]$. This section briefly discusses the possibilities to model $R[\Theta Y]$ and presents how differential structure is added to $R[\Theta Y]$. $R[\Theta Y]$ together with the differential structure finally models $R\{Y\}$.

In the beginning of this section the polynomial implementations of `Algebra` are discussed. Afterwards, necessary efforts for being able to implement $R[\Theta Y]$ are presented. This discussion is followed by a description of the structural extension of $R[\Theta Y]$ to $R\{Y\}$ itself.

With PolynomialRing0 `Algebra` provides a category for multivariate polynomial rings. The category PolynomialRing refines PolynomialRing0 and adds functions for extracting the leading coefficient (leadingCoefficient), singling out variables (combine), and evaluation (eval). Additionally, PolynomialRing takes advantage of properties of the coefficient ring R, as for example PolynomialRing exports CommutativeRing[29] if R has CommutativeRing.

`Algebra` provides five implementations of PolynomialRing0 (RecursiveMultivariatePolynomial0, SparseIntegerMultivariatePolynomial, SparseMultivariatePolynomial, IntegerPolynomial, DistributedMultivariatePolynomial1). The first four of these implementations also satisfy PolynomialRing.

[28] To keep the mathematical conditions compact, the given example of DifferentialVariableMixedOrderTools is simple. However, DifferentialVariableMixedOrderTools allows to build more complex relations as for example shown on page 110.

[29] CommutativeRing is `Algebra`'s category for commutative rings. Besides a commuting multiplication, this category requires a function for getting the inverse of an element, which is allowed to fail, and several functions for units, as for example a test whether or not an element is a unit.

Figure 3.5: All categories and domains for differential indeterminates

By DistributedMultivariatePolynomial0 `Algebra` provides a sixth implementation of multivariate polynomial rings. However, this domain is not taken into consideration, since it does not provide PolynomialRing0 but only IndexedFreeModule[30].

Additionally, SparseIntegerMultivariatePolynomial and IntegerPolynomial are not considered further, since both implementations do not allow to use a differential ring as coefficient domain.

As a result, $R[\Theta Y]$ will be modelled by RecursiveMultivariatePolynomial0, SparseMultivariatePolynomial, and DistributedMultivariatePolynomial1.

With the presented categories and domains, it is possible to model $R[\Theta Y]$ using RecursiveMultivariatePolynomial0 and SparseMultivariatePolynomial. DistributedMultivariatePolynomial1 is a distributive (see Section 11 for further information about various different approaches to implement polynomial rings) implementation of polynomials and needs an exponent category for the derivatives. The next few paragraphs explain the consequences thereof and deal with this problem.

Besides the coefficient domain and a domain for indeterminates, DistributedMultivariatePolynomial1 also needs a third parameter, which is an implementation of ExponentCategory for the indeterminates. ExponentCategory is a category that takes a domain for indeterminates as parameter and allows to build terms of the provided indeterminates[31]. DistributedMultivariatePolynomial1 sticks coefficients to these terms and collects these monomials to model polynomials.

`Algebra` provides three implementations of ExponentCategory (MachineIntegerDegreeLexicographicalExponent, MachineIntegerDegreeReverseLexicographicalExponent, MachineIntegerLexicographicalExponent). The type of the domain for the indeterminates is FiniteVariableType for all three implementations. Since differential polynomial rings are typically polynomials in infinitely many indeterminates (namely the embedding of ΘY), none of the three implementations can be used. However, ExponentCategory does not restrict the domain of the indeterminates to be finite and can therefore be reused.

`CharSet` provides five implementations of ExponentCategory (CumulatedExponent, ListExponent, ListSortedExponent, SortedListExponent, ClassPresortedDifferentialExponent), which all can be used for building terms of elements of ΘY. These five domains for exponents allow to model $R[\Theta Y]$ in different ways that are compared in Section 11. Section 11 also discusses the five implementations in depth. A diagram of each of these implementations can be found on Figure 3.6.

[30] IndexedFreeModule is a domain of `Algebra` that puts less conditions on its parameters than PolynomialRing0. While PolynomialRing0 implements what developers typically expect from an implementation of multivariate polynomial rings, IndexedFreeModule focuses on the the separation between the coefficient and terms in the summands of a polynomial. This category is closely connected to the distributive approach for implementing polynomials, but it does not provide functions to extract further information about the terms.

[31] The documentation of ExponentCategory claims that ExponentCategory is used for modelling monomials. However, `Algebra`'s documentation uses a different notation than this report. This report uses "term" to refer to a product of indeterminates and "monomial" for a term with a coefficient.

Figure 3.6: The category and domains for differential polynomial rings

`CharSet` uses DifferentialPolynomialRingType to model $R\{Y\}$. This category takes R, Y, and ΘY as parameters and permits the common functions of multivariate polynomials by exporting PolynomialRing0. DifferentialPolynomialRingType also exports DifferentialType and RankedType[32]. If R satisfies FiniteCharacteristic[33] then also DifferentialPolynomialRingType does. The same holds for the categories CharacteristicZero[34] and Parsable[35].

Furthermore, it is possible to obtain the class of a differential polynomial (class), to retrieve the order of a differential polynomial with respect to an indeterminate (order), and to get the reciprocal of a differential polynomial, if a reciprocal exists (reciprocal). DifferentialPolynomialRingType also allows to divide a differential polynomial by an indeterminate (quotient, quotientBy) and provides a function for building partial derivatives with respect to an indeterminate (differentiate). DifferentialPolynomialRingType is illustrated in Figure 3.6

`CharSet` provides two implementations of DifferentialPolynomialRingType, which are both called DifferentiallyExtendedPolynomialRing.

These two domains share the same functionality, but have different focus.

The first DifferentiallyExtendedPolynomialRing takes R, Y, ΘY, and $R[\Theta Y]$ as parameter. The type of the parameter $R[\Theta Y]$ has to export PolynomialRing0. If this parameter has furthermore PolynomialRing, then the functions of DifferentiallyExtendedPolynomialRing are overridden by those of PolynomialRing wherever this is possible.

The second DifferentiallyExtendedPolynomialRing extends the first implementation by also exporting IndexedFreeModule. This additional export allows to access the summands of a polynomial directly, which is advantageous for several algorithms. This DifferentiallyExtendedPolynomialRing takes five arguments. These arguments are R, Y, ΘY, an ExponentCategory of ΘY, and $R[\Theta Y]$. The fifth parameter has to satisfy PolynomialRing0 and IndexedFreeModule. Due to this further requirement, this DifferentiallyExtendedPolynomialRing cannot be used with recursive implementations of $R[\Theta Y]$.

As indicated in Section 3.2 the derivations in DifferentiallyExtendedPolynomialRing are built by using the derivations on R and exploiting Θ's structure in ΘY. When applying an element $\bar{\theta}$ of Θ on an indeterminate θy of $R\{Y\}$ without exploiting Θ's structure in ΘY, the implementation of a differential polynomial ring has to store both, $\bar{\theta}$ and θy. This cannot be simplified to the indeterminate $(\bar{\theta}\theta)y$ without exploiting Θ's structure in ΘY[36]. As a result, the implementation of the differential polynomial ring has to store

[32] RankedType is `CharSet`'s category for a domain with irreflexive, transitive, asymmetric relations $<$ and $>$.

[33] The category FiniteCharacteristic is introduced by `Algebra` and marks Rings to have a finite characteristic.

[34] CharacteristicZero is `Algebra`'s category for marking Rings to have characteristic 0.

[35] The category Parsable is defined in `Algebra` and provides functions for evaluating expression trees to a domains value.

[36] From a mathematical point of view, in a differential polynomial ring

$$\forall \bar{\theta} \in \Theta \ \ \forall \theta y \in \text{indeterminates of } R\{Y\} : \bar{\theta}(\theta y) = (\bar{\theta}\theta)y,$$

additional properties (i.e.: application of $\bar{\theta}$) for every indeterminate. Therefore, the data structure holding the polynomials cannot be used to store differential polynomials as a whole, which causes further overhead. However, DifferentiallyExtendedPolynomialRing takes advantage of ΘY's additional structure by using DifferentialVariableType and therefore the application of $\bar{\theta}$ to θy can be simplified to the indeterminate $(\bar{\theta}\theta)y$ in $R\{Y\}$. As a result, the data structure holding the polynomials can be used to completely store differential polynomials. Application of a derivation to a differential polynomial is done by delegating the derivations to the coefficients and the indeterminates themselves and finally summing up the results.

with $\bar{\theta}\theta y$ being again an indeterminate of $R\{Y\}$, has to hold. Such axioms cannot be modelled, without having knowledge about the θ in θy.

4 Implementing reduction

This section deals with implementing differential reduction on differential polynomials. General categories for abstracting reductions are presented and applied to an implementation of differential reduction. Additionally, autoreduced sets are introduced.

At the beginning of this section `Aldor`'s and `Algebra`'s efforts towards reduction in general are evaluated. Afterwards the necessary functions for triangularization computations are discussed and put together in categories along with a data structure for autoreduced sets. Finally, an implementation of differential reduction is presented and its connections to the design of ΘY and $R\{Y\}$ are investigated.

Neither `Aldor` nor `Algebra` provide categories or domains that can be used for implementing differential reduction. There is only a function reductum of FreeModule that removes the leading term in a polynomial. Additionally, polynomial reduction by monic polynomials on univariate polynomials can be achieved in three steps. First, a domain that exports UnivariatePolynomialQuotient[37] has to be set up. This domain takes the monic polynomial to reduce by as parameter. Besides further functions, this new domain provides a function reduce, which performs polynomial reduction. Second, the function reduce from the new domain has to be called with the polynomial to reduce as argument[38]. The result of this function is a polynomial in the new domain. The third step is to convert this polynomial back to the domain of the original univariate polynomial. However, this polynomial reduction can only be done for univariate polynomials. There is no abstraction available that permits to bring this design to multivariate polynomials.

4.1 An abstract category for reductions

In contrast to `Algebra`'s implementation of polynomial reduction, `CharSet`'s implementation of differential reduction does not map elements from one domain to another, more special one. Therefore, no conversion of the result of a differential reduction to the original differential ring is necessary. This approach eases repeated calling of the reduction algorithm, which is typically necessary when performing characteristic set computations. However, no information is gathered about with respect to which polynomials another polynomial is reduced.

Although it is clear that in connection with differential characteristic sets, reduction is used to refer to differential reduction, other contexts have different meanings for reduction. For example when performing Gröbner bases computations [53, chapter 8], reduction is typically used to refer to polynomial reduction and not to differential reduction. In order to allow different kinds of reduction on differential polynomial rings, differential reduction is not incorporated into the categories and domains for differential

[37] `Algebra` provides the category UnivariatePolynomialQuotient for modelling univariate polynomials modulo a monic polynomial.

[38] There is no need to specify the polynomial to reduce by, as this is a parameter to the domain from which the reduce function is called.

rings. Differential reduction is implemented by a separate domain, as will be seen in the end of this section.

The basic category for all kinds of reductions is the category ReductionType of `CharSet`. These reductions can be differential reductions, polynomial reductions or completely different kinds of reductions. ReductionType takes the domain on which the reduction should act as parameter. This domain will be referred to by `T` for the rest of the description of ReductionType. For this report's implementation, `T` is typically an implementation of differential polynomials.

The two most important functions of ReductionType are reduce and reduced?. reduce takes two elements of `T` and gives the result of reducing the first one with respect to the second one. reduced? takes again two elements of `T` as arguments. This function returns true, if the first argument is reduced with respect to the second one. Otherwise, the function gives false.

Additionally, ReductionType provides reduceBy and reducedBy?. reduceBy takes an element of `T` as argument and gives a mapping that reduces elements of `T` with respect to reduceBy's argument. By calling reducedBy? an element of `T` is mapped to a function, which decides whether or not another element of `T` is reduced with respect to reducedBy?'s argument.

ReductionType is illustrated in Figure 4.1.

4.2 Incorporating autoreduced sets into the reduction category

By using ReductionType, `CharSet` implements a domain AutoreducedSet, which models autoreduced sets[39].

Figure 4.1 shows that AutoreducedSet takes two parameters. The first parameter is called `T` and denotes the domain for the elements in the autoreduced set. `T` has to provide a function to decide whether or not an element is zero (zero?) and a function for deciding whether or not an element is from the ground structure of `T` (ground?)[40]. If the domain provides functions to compare its elements to each other (PartiallyOrderedType or RankedType), the elements are stored in decreasing order to speed up further computations. A typical example of `T` is a differential polynomial ring. The second parameter of AutoreducedSet is a ReductionType, for determining with respect to which reduction the elements of the set form an autoreduced set.

[39] Autoreduced sets are sets, whose elements are pairwise reduced.

[40] `T` typically denotes a polynomial domain. There, "ground structure" refers to the constants of the polynomial ring. As a result, ground? should give true in a polynomial setting exactly for the constant elements. All polynomial implementations of `Algebra` and `CharSet` provide such a ground? function. A definition of the ground? function can be found in the documentation of `Algebra`'s domain FreeModule.

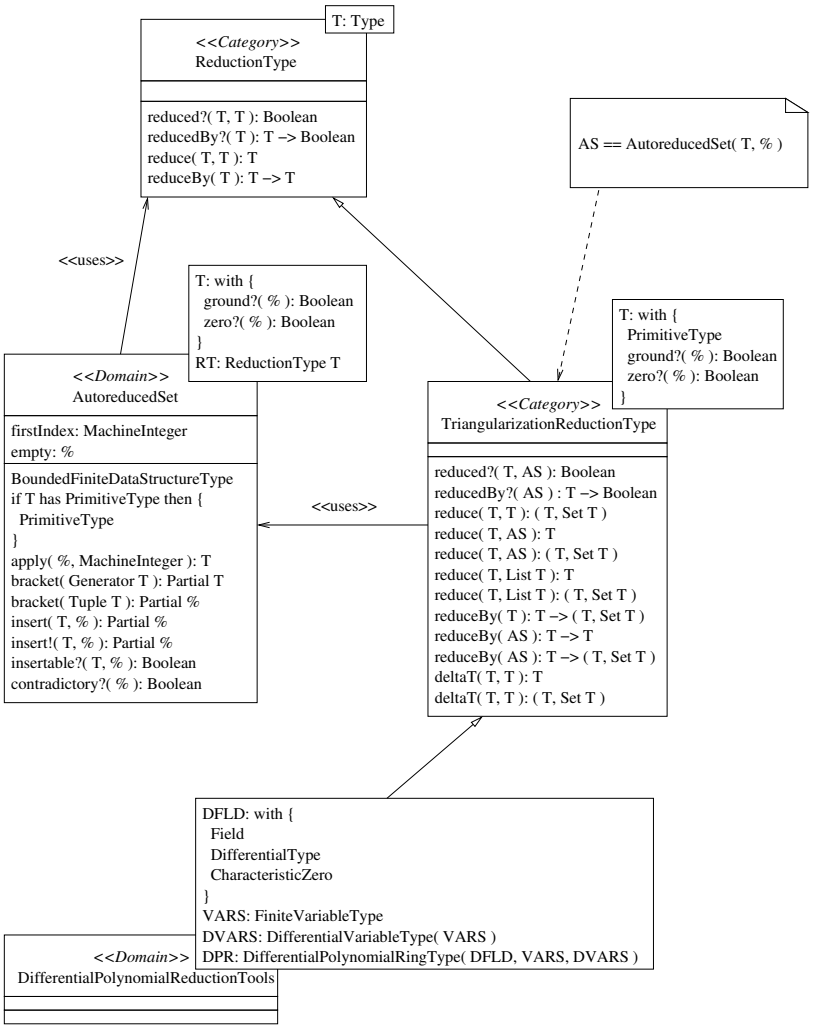

Figure 4.1: The domains and categories for reduction

AutoreducedSet exports BoundedFiniteDataStructureType[41] to provide basic functions for enumerating the elements of an AutoreducedSet or for giving the number of elements in a AutoreducedSet. Further functions and constants of AutoreducedSet are described in the next three paragraphs.

The constant firstIndex of AutoreducedSet is a number denoting the position of the first element in the autoreduced set. AutoreducedSet's function apply is used for retrieving elements of the autoreduced set and takes two parameters. The first parameter is the autoreduced set to get an element from. The second parameter is an index. Using firstIndex as index gives the first element. When `firstIndex+i` is used as index, the $(i+1)^{th}$ element is returned. firstIndex and apply are usually exported by LinearStructureType[42]. However, LinearStructureType also provides a function set! for modifying values. This function of LinearStructureType is not allowed to fail. As assigning a new value to an element of an autoreduced set might harm the property of being autoreduced, set! of LinearStructureType cannot be implemented properly in AutoreducedSet. Therefore, LinearStructureType cannot be used by itself. Only the functions for retrieving values of an autoreduced set have been reimplemented.

For obtaining an empty autoreduced set, the constant empty of AutoreducedSet can be used. This constant can also be found in the category FiniteLinearStructureType[43]. However, AutoreducedSet does not export FiniteLinearStructureType, since FiniteLinearStructureType provides functions for creating data structures that are not valid for autoreduced sets.

AutoreducedSet provides two bracket functions for generating autoreduced sets. One function takes a tuple of elements. The other function takes a sequence of elements (Generator[44]) as parameter. Both functions convert the supplied elements to an AutoreducedSet that is wrapped by Partial[45]. This wrapping by Partial is necessary, since the supplied elements may not be pairwise reduced. In such a case, bracket returns failed. Otherwise, bracket gives an autoreduced set of the supplied elements. AutoreducedSet's insert and insert![46] functions are used for inserting values to an autoreduced set. Both functions take an autoreduced set and an element, which is added to this autoreduced

[41] BoundedFiniteDataStructureType is `Aldor`'s category for data structures that can be copied (CopyableType), allow to query for their finite number of elements, and permit to enumerate them. Furthermore, properties of the elements' domain, such as having hash functions or functions to print to a stream, are inherited. If the domain for the elements provides a test for equality (PrimitiveType), this category also allows to search for elements and to decide whether or not an element is a member of the data structure.

[42] `Aldor`'s category LinearStructureType is used for linear data structures. This category provides functions for building up such data structures and setting and getting elements of them.

[43] By FiniteLinearStructureType `Aldor` provides a category that extends LinearStructure by additional functions for setting up a data structure and a constant empty, which holds an empty instance of the data structure.

[44] Generator is a domain of `Aldor`. Generator is used to provide a collection of elements in a serialized form.

[45] `Aldor` introduces the domain Partial to add a value failed to domains. With the help of Partial, functions can easily be generated that may result in failed and thereby indicate a failure.

[46] In Aldor, identifiers are allowed to contain exclamation marks. Only the first character of an identifier must not be an exclamation mark. It is common convention in Aldor that functions, which may destroy or alter their given parameters, have an exclamation mark at the end of their name.

set, as parameters. insert! inserts the new element to the autoreduced set that is passed as parameter. insert makes a copy of the autoreduced set and inserts the element to the copied autoreduced set. Thereby, the autoreduced set that is passed to the function as parameter remains unmodified. Both functions map to a Partial AutoreducedSet and fail, if inserting harms the property of being pairwise reduced. AutoreducedSet also provides the function insertable? for checking in advance, whether or not an insert will fail. Finally, AutoreducedSet's contradictory? allows to determine whether or not an autoreduced set contains elements from the ground structure of `T`.

Due to this abstract formulation of ReductionType and AutoreducedSet, arbitrary pairwise reduced objects with an arbitrary reduction can be stored by AutoreducedSet. A possible example would be non commutative polynomials with polynomial reduction.

To ease characteristic set computations, `CharSet` introduces the category TriangularizationReductionType. This category refines ReductionType and adds subsidiary functions for triangularization computations to the functions for reduction. The domain on which the reduction acts is the only parameter to TriangularizationReductionType. For the rest of the description of TriangularizationReductionType, this parameter is abbreviated by `T`. `T` is used to build AutoreducedSets and has to meet the same requirements as the domain parameter in AutoreducedSet. Again, a typical example of `T` is a differential polynomial ring.

For the rest of the description of TriangularizationReductionType, the term "autoreduced set" is used without specifying the autoreduced set's domain or reduction. The elements of these autoreduced sets are elements of `T`. The autoreduced sets' reduction is the domain that implements TriangularizationReductionType.

As can be seen in Figure 4.1, TriangularizationReductionType provides additional reduce and reduceBy functions. These functions allow to reduce by elements of an autoreduced set or a list. For every reduce and reduceBy function, TriangularizationReductionType provides a variant that does not only give the reduced element but also the elements by which has been premultiplied during the reduction. These elements are returned as Set along with the result of the reduction.

As can be seen for example in [39], characteristic set computations in differential polynomial rings involve the computation of Δ-polynomials, which correspond to S-polynomials in Gröbner basis computations. Therefore, TriangularizationReductionType provides the function deltaT. This function takes two elements of `T` as parameters and returns another element of `T`. In a general setting, deltaT gives an element of `T` that needs to be considered if both of its arguments occur in an AutoreducedSet, which should become a triangular set. In the special setting of differential reduction, this resolves to Δ-polynomials. TriangularizationReductionType provides a second deltaT function. This function takes the same parameters as the first one and returns additionally a Set of elements of `T`. The elements of this set have been use to premultiply during the computation of deltaT.

4.3 Implementing differential reduction

`CharSet` implements differential reduction in the domain DifferentialPolynomialReductionTools. DifferentialPolynomialReductionTools implements TriangularizationReductionType and takes four parameters. According to the introduced notation, these parameters correspond to R, Y, ΘY, and $R\{Y\}$. The fourth parameter is the differential ring on which the differential reduction acts. The first three parameters are only needed to properly specify the type of the differential ring $R\{Y\}$.

In order to show how the implementation of DifferentialPolynomialReductionTools influenced the design of ΘY and $R\{Y\}$, differential reduction of a differential polynomial with respect to another polynomial is discussed. For the rest of this section, let a, b be differential polynomials in $R\{Y\}$, which have at least total degree 1. Assume that a is to be reduced differentially with respect to b.

For performing differential reduction, a total order on the infinitely many indeterminates of $R\{Y\}$ is necessary. DifferentialPolynomialReductionTools uses the total order on the derivatives ΘY for differential reduction. Thereby, the greatest indeterminate in a polynomial can be determined by passing the polynomial to the function mainVariable[47] of PolynomialRing0, which is exported by any multivariate polynomial ring implementation. Since the data structure of $R[\Theta Y]$ and its functions are preserved when turning $R[\Theta Y]$ into $R\{Y\}$, calling mainVariable on a differential polynomial is just as fast as calling it on elements of $R[\Theta Y]$. Therefore, there is no performance penalty for using differential polynomials. This result again motivates the taken efforts to introduce a total order on the derivatives. Since ΘY's order has great impact on the outcome of the differential reduction, the introduction of the total order as a parameter to ΘY is emphasized. Because of this parametrization, different orders can be applied to ΘY easily. As a result, differential reduction can easily be performed with different orders.

For the rest of this section, let lead(b) denote the greatest indeterminate of $R\{Y\}$ that occurs in b.

After determining lead(b), the indeterminates of a are obtained and investigated. Therefore, the function variables of DifferentialPolynomialRingType is applied to a. This function gives a sorted list of indeterminates occurring in a in decreasing order. With the same argumentation as presented in the treatment of mainVariable, a call to the function variables of $R\{Y\}$ is just as fast as a call to the function variables of $R[\Theta Y]$. If

- any element in the list of indeterminates of a is a proper derivative of lead(b) or
- if the degree of a with respect to lead(b) is not smaller than the degree of b with respect to lead(b),

[47] mainVariable is a function in the category PolynomialRing0 of `Algebra`. This function takes a polynomial as argument and gives the greatest indeterminate that occurs in the polynomial. Here, "greatest" is used in terms of the total order defined on the indeterminates.

a suitable derivation is applied to b. After premultiplying both, a and the derivated b, they are subtracted. The result of the subtraction takes the role of a as the process is started again. Otherwise a is already reduced with respect to b. The function isProperDerivative? of DifferentialVariableType is used to determine whether or not an indeterminate is a proper derivative of lead(b). For obtaining the degree of an indeterminate in a polynomial, the function degree is used, which again is as fast as its counterpart in $R[\Theta Y]$.

As already mentioned in Section 3.2, differential reductions by autoreduced sets take advantage of further properties of ΘY. If the order on ΘY is an elimination order[48] and the reduction is started by those polynomials containing the biggest indeterminates, information about previous reduction steps is used to speed up computations. Let i be the class of the polynomial by which has been reduced in the last reduction step. Then only those elements of the autoreduced set having a class $\leq i$ have to be taken into consideration for the further reduction.

Section 3.2 indicated that a total order on the derivatives is not a sufficient condition for the termination of differential reduction. Only further restrictions on the total order of ΘY, such as

$$\begin{aligned}\forall \theta, \bar{\theta}, \tilde{\theta} \in \Theta \forall \bar{y}, \tilde{y} \in Y : \bar{\theta}\bar{y} \leq (\theta\bar{\theta})\bar{y} \\ \wedge\, \bar{\theta}\bar{y} \leq \tilde{\theta}\tilde{y} \implies (\theta\bar{\theta})\bar{y} \leq (\theta\tilde{\theta})\tilde{y} \qquad (*)\end{aligned}$$

guarantee termination of differential reduction. If the order $\leq$ has the property $(*)$, it is called "ranking on ΘY" [45]. However, such conditions cannot be modelled in Aldor. To choose an order for which differential reduction terminates is left to users of `CharSet`. The orders of this report's work are all rankings. Therefore, the orders of this report's work guarantee the termination of differential reduction.

[48] see Section 3.2.1.

5 Implementing characteristic set algorithms

Characteristic set computations are typically divided into two parts. The first part operates in a differential polynomial ring. For the second part, the differential structure is no longer relevant and therefore omitted. This second part is usually handled by well known algorithms of polynomial rings without differential structure. For example the approach of [38] uses Gröbner basis computations in polynomial rings without differential structure. These algorithms for the second part are not closely related to characteristic set computations and therefore not discussed in this report, although they are implemented in `CharSet`.

For the first part however, the differential structure is important. This section presents the algorithms that are necessary for the first of these two parts of characteristic set computation.

This section is divided into three parts. The first part defines characteristic sets, basic sets and medial sets. The necessary categories and algorithms for generic computation of basic sets are discussed in the second part. These categories are used in the third part, to provide generic implementations to compute coherent autoreduced sets.

5.1 Characteristic set algorithm terminology

The definitions of characteristic sets require a ranking of autoreduced sets. This ranking is inferred from the order of the derivatives ΘY. The required definitions for this inference are reproduced from [45] without discussing the properties of the resulting rankings. [45] gives a more detailed presentation of the ranking.

Definition 5.1 (Ranking of differential polynomials) *(compare [45]) Let $p, q \in R\{Y\}$ with q not being constant. If p is a constant, p* has lower rank than *q. If p is not a constant, p* has lower rank than *q if and only if* [49]

$$\begin{aligned} & (\mathrm{lead}(p) < \mathrm{lead}(q)) \quad \vee \\ \vee \quad & \big(\mathrm{lead}(p) = \mathrm{lead}(q) \wedge \deg_{\mathrm{lead}(p)}(p) < \deg_{\mathrm{lead}(p)}(q)\big)\,. \end{aligned}$$

$p \prec q$ is used to express "p has lower rank than q". p has the same rank as q ($p \sim q$) if and only if neither $p \prec q$, nor $q \prec p$.

Definition 5.2 (Ranking of autoreduced sets) *(compare [45]) Let P and Q be autoreduced sets with elements in $R\{Y\}$. Let r denote the number of elements in P and $p_1, p_2, \ldots, p_r$ denote the elements of P ordered by increasing rank (i.e : $p_1 \prec p_2 \prec \ldots \prec p_r$). Accordingly, let s denote the number of elements in Q and $q_1, q_2, \ldots, q_s$ denote the elements of Q ordered by increasing rank. P* has lower rank than *Q if and only if*

$$\begin{aligned} & \big(r > s \wedge \forall i \in \{1, 2, \ldots, s\} : p_i \sim q_i\big) \quad \vee \\ \vee \quad & \big(\exists k \in \{1, 2, \ldots, \min(r, s)\} : p_k \prec q_k \wedge \forall i \in \{1, 2, \ldots, k-1\} : p_i \sim q_i\big) \end{aligned}$$

[49] For all $\bar{p} \in R\{Y\}$ with $\bar{p}$ not being a constant, let $\mathrm{lead}(\bar{p})$ denote the highest ranking indeterminate of $\bar{p}$ and for all indeterminates θy of $R\{Y\}$, let $\deg_{\theta y}(\bar{p})$ denote the degree of $\bar{p}$ with respect to θy.

If P has lower rank than Q, Q has higher rank than *P.*

There are several prominent definitions of "characteristic set". All of them are in the same spirit, but bear differences. Therefore, these definitions are presented and compared to each other.

Definition 5.3 (Ritt characteristic set) *(compare [48]) Let P denote a set of differential polynomials. An autoreduced subset of P that does not rank higher than any other autoreduced subset of P is called* Ritt characteristic set of *P.*

Definition 5.4 (Kolchin characteristic set) *(compare [45]) Let P denote a differential ideal. An autoreduced subset of P that does not rank higher than any other autoreduced subset of P is called* Kolchin characteristic set of *P.*

Definition 5.5 (Wang characteristic set) *(compare [50]) Let P denote a set of differential polynomials. An autoreduced subset of the differential ideal of P that reduces every element of P to 0 is called* Wang characteristic set of *P.*

The definition of Kolchin characteristic set is a restriction of Ritt characteristic set to differential ideals. Kolchin does not define the term "characteristic set" for sets that are *not* differential ideals.

For Wang characteristic set, P is not restricted to differential ideals, but while Ritt characteristic set and Kolchin characteristic set investigate autoreduced sets in P, Wang characteristic set takes autoreduced subsets in *the differential ideal generated by P* into account.

If P is a differential ideal, the definitions of Ritt characteristic set, Kolchin characteristic set and Wang characteristic set are equivalent.

To avoid confusion, the term "characteristic set of P" is used exclusively with P being a differential ideal in this report. Therefore, no distinction between the three presented definitions is necessary.

During characteristic set computations, it is necessary to compute Ritt characteristic sets for some Ps that are not necessarily differential ideals. That is why besides the definition of Wang characteristic set, Wang also gives the definition of Ritt characteristic set in [50] – although under the name "basic set".

Definition 5.6 (basic set) *(compare [50]) Let P denote a set of differential polynomials. An autoreduced subset of P that does not rank higher than any other autoreduced subset of P is called* basic set of *P.*

This report also uses the term basic set to denote Ritt characteristic set for a set P that is not necessary a differential ideal.

Some characteristic set algorithms involve computations of basic sets several times. Therefore, the concept of basic sets is abstracted to medial sets.

Definition 5.7 (medial set) *(compare [50]) Let P be a set of differential polynomials. A* medial set of P *is an autoreduced subset of the differential ideal generated by P that does not rank higher than a basic set of P.*

In characteristic set algorithms, the computations of basic sets are replaced by medial set computations in order to provide more generality. By using different ways to compute medial sets, variations of algorithms are created. Although medial sets cannot rank higher than basic sets, they can have lower rank than basic sets, since basic sets take their elements from P, medial sets take their elements from the differential ideal generated by P.

5.2 Basic sets and medial sets

In this section algorithms to compute basic sets and medial sets are presented along with their implemented categories and domains. These efforts are reused as subalgorithms in Section 5.3.

In the beginning of this section the scope of the implementation is clarified. Afterwards, categories for triangularization algorithms and medial set algorithms are introduced. As several algorithms involve updating medial sets, a separate category for medial sets that can easily be updated is presented. Finally, two algorithms for computing basic sets are given.

The definition of basic set and medial set have been given in a differential polynomial context in Section 5.1. One important aspect of this report is to formulate the algorithms in a more abstract context than differential polynomials with differential reduction. Therefore, the implemented algorithms are not limited to differential polynomials, but can operate on arbitrary elements that provide a ranking and functions to check whether or not an element is zero or a constant. Furthermore, the implemented algorithms are not limited to differential reduction, but are implemented for arbitrary reductions that fit in the categories of Section 4 (see ReductionType and TriangularizationReductionType).

Computing a basic set involves to compute an autoreduced set of a given set of elements. This description is also valid for all other algorithms that compute medial sets. In Section 5.3 further algorithms are presented that also fit in this description. Therefore, this description is abstracted and transformed into a category.

TriangularizationAlgorithmType is `CharSet`'s category for algorithms that can compute an autoreduced set from a given collection of elements. This category takes two parameters. The first parameter denotes the domain of the elements on which the algorithm

operates. `T` is used to refer to this parameter. `T` has to provide a function to decide whether or not an element is zero (zero?) or a constant (ground?). Additionally, `T` has to provide PrimitiveType. A typical value for `T` is a domain modelling differential polynomials. The second parameter of TriangularizationAlgorithmType is a ReductionType on the elements of `T`. This parameter denotes the reduction for the autoreduced sets that are returned by the functions of TriangularizationAlgorithmType.

TriangularizationAlgorithmType provides four functions. All of them are called triangularize and compute a triangularization of a given collection of elements in `T`. The first triangularize function takes a List of elements of `T` and returns an AutoreducedSet of elements of `T` with respect to the reduction that is passed as parameter to TriangularizationAlgorithmType. The second function takes the same parameters as the first one and returns an autoreduced set as the first function. Additionally, the second function returns a Set of elements of `T` that have been used to premultiply during the computations. The third and the fourth function of TriangularizationAlgorithmType are similar to the first and second. The only difference is that the third and fourth function take Generators instead of Lists as arguments. All four functions are required to implement the same algorithm, for different input parameters and return types.

TriangularizationAlgorithmType could be used directly to model algorithms that compute a basic set. However, every basic set for a given set P of elements is also a medial set of P. Medial sets meet another condition that is not expressed by TriangularizationAlgorithmType. Medial sets of P do not rank higher than a basic set of P. To make this additional condition explicit, a further category for medial sets is introduced.

The category MedialSetAlgorithmType of `CharSet` extends TriangularizationAlgorithmType by asserting that the output of each of the four `triangularize` functions form a medial set for the given input. MedialSetAlgorithmType takes the same parameters as TriangularizationAlgorithmType and does not add new functionality. The only difference is the additional, semantic condition.

Some characteristic set algorithms involve many medial sets computation. In such algorithms, there is typically some set F of differential polynomials for which a medial set is to be computed. Then, further elements are added to F and a medial set for the updated F is to be derived. When using the functions of MedialSetAlgorithmType, every medial set computation has to restart from scratch. Reusing results from previous medial set computations allows to speed up computations. For algorithms that take advantage of previous computations, the category UpdatableMedialSetAlgorithmType is introduced.

`CharSet` provides the category UpdatableMedialSetAlgorithmType. UpdatableMedialSetAlgorithmType takes the same parameters as MedialSetAlgorithmType does and extends MedialSetAlgorithmType by the function triangularize!. triangularize! is used to update a previous computation of a medial set computation and takes four parameters. These parameters are an autoreduced set `C`, a list `F`, a set `S`, and a list `R`. The parameters `C`, `F`, `S` represent the status of the medial set computation to update. They correspond to the return values of this triangularize! function. Their values are discussed more detailed later. `R` holds new elements of `T`, by which the medial set of a previous computation is to be updated. The function acts destructive on all parameters.

The result of triangularize! is an autoreduced set C', a list F', and a set S'.

- C' holds a medial set of the union of C, F, and R.
- F' contains those elements of the union of C, F, and R that do not belong to C'.
- S' holds those elements that have been used to premultiply during the computations.

As stated before, the input parameters C, F, and S correspond to the result of a previous medial set computation. C has to be the resulting C' of a previous medial set computation. Accordingly F and S have to be resulting F' and S' of the same, previous computation. If a new computation is started from scratch, each of C, F, and S has to be empty.

UpdatableMedialSetAlgorithmType's triangularize! has to implement the same triangularization algorithm as its triangularize function.

It is not allowed to update C', F', and S' of a medial set computation in one domain via the triangularize! function of another domain. C', F', and S' have to be updated using the triangularize! function they originate from. Furthermore, it is not allowed to modify C', F' and S' in any form between updates.

The following example illustrates the usage of triangularize!.

```
--starting without a previous computation

--a medial set for the elements in R will be computed
( C', F', S' ) := triangularize!( empty, empty, empty, R );
--C' is the desired medial set
...
--R is set to a list of elements by which the previous
--medial set has to be updated
...
--None of C', F', and S' may have been modified
( C', F', S' ) := triangularize!( C', F', S', R );
--C' is the desired updated medial set
```

The basic set implementation of this report is found in the domain BasicSetTools of CharSet. This domain implements UpdatableMedialSetAlgorithmType and also takes the same parameters. The only restriction is that T, the domain for the elements to compute a basic set for, has to provide a ranking for its elements. BasicSetTools models the *BasSet* algorithm (see Algorithm 5.1) as described in [50][50]. Computing a basic set for some F with the *BasSet* algorithm starts with an empty set and successively adds the lowest elements of F that keep the set autoreduced.

[50]This algorithm is not invented by Dongming Wang, but can already be found in [48]. The algorithm is not given a name by Ritt. Wang gave a name to the algorithm, which makes it easier to refer to. Therefore, Wang's book is cited although the algorithm has not been invented by him.

Algorithm: *BasSet*

Input:
P: a set of elements in `T`
Output:
A: a basic set of P

1: $F \leftarrow P$
2: $A \leftarrow$ empty autoreduced set
3: **while** $F \neq \emptyset$ **do**
4: $B \leftarrow$ an element of F with lowest rank
5: insert B into A
6: **if** class of $B = 0$ **then**
7: $F \leftarrow \emptyset$
8: **else**
9: $F \leftarrow \{G \in F \setminus \{B\} \mid G \text{ is reduced w.r.t. } B\}$
10: **end if**
11: **end while**

Algorithm 5.1: The *BasSet* algorithm.

The *BasSet* algorithm has one major disadvantage. This disadvantage is the extraction of an element of lowest rank. For this analysis, let n denote the number of elements in P. The extraction of the element of lowest rank has worst case complexity $\mathcal{O}(n)$. Due to the **while** loop, the element of lowest rank has to be determined n times in the worst case. As a result, the overall complexity for extracting the element of lowest rank is $\mathcal{O}(n^2)$. However, F can be sorted in ascending order. Then, the element of lowest rank of F can be obtained in constant time, $\mathcal{O}(1)$. As the first element is removed from F, the remaining elements are still sorted in ascending order. Therefore, the next element of lowest rank elements can again be found in constant time. Nevertheless, the overall complexity for finding the elements of lowest rank is not $\mathcal{O}(n)$, since F needs to be sorted once, which is typically of higher complexity. `Algebra`'s List, which is used for implementing this modification, implements merge sort. When sorting F before starting the computation, $\mathcal{O}(n \log n)$ is the overall complexity for finding the elements of lowest rank[51].

The *BasSetSorted* (see Algorithm 5.2) incorporates these considerations and restructures the last step of *BasSet* to better fit the needs of Aldor's domains.

BasSetSorted has no direct checks whether or not an element is reduced with respect to another one. These checks are not omitted but hidden inside step 6.

By the domain BasicSetSortedTools, `CharSet` provides an implementation of the *BasSetSorted* algorithm. BasicSetSortedTools takes the same parameters as BasicSetTools and also implements UpdatableMedialSetType. In the general case, BasicSetSorted-

[51] Although this modification reduces the complexity for finding the elements of lowest rank from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \log n)$, it does not reduce the overall complexity of *BasSet*. Both, the original *BasSet* and the presented modification are of complexity $\mathcal{O}(n^2)$. This complexity is due to line 9 in *BasSet*.

Algorithm: *BasSetSorted*

Input:
P: a set of elements in `T`
Output:
A: a basic set of P

1: $F \leftarrow P$ in ascending order
2: $A \leftarrow$ empty autoreduced set
3: **while** $F \neq \emptyset$ **do**
4: $B \leftarrow$ first element in F
5: take B out of F without disturbing F's order
6: **if** $A \cup B$ is an autoreduced set **then**
7: insert B into A
8: **end if**
9: **end while**

Algorithm 5.2: The *BasSetSorted* algorithm.

Tools is faster than BasicSetTools. Therefore, it is suggested to use BasicSetSortedTools instead of BasicSetTools.

Figure 5.1 shows the categories and domains of this section.

5.3 Towards characteristic set computations

This section defines generic algorithms to compute coherent autoreduced sets.

After introducing a category for triangularization algorithms that allows to update a triangularization, two domains that implement this category are presented.

In this section, let $[A]$ denote the differential ideal generated by A. Furthermore, H_A is used to refer to the set of initials and separants that occur in A.

Section 5.2 presented the categories TriangularizationAlgorithmType and MedialSetAlgorithmType. Where this is possible, it is advantageous to update medial sets instead of restarting medial set computation from scratch. Therefore, UpdatableMedialSetAlgorithmType has been introduced in Section 5.2. Many triangularization algorithms that do not compute medial sets benefit from reusing previously computed results as well. Therefore, the category UpdatableTriangularizationAlgorithmType is introduced.

The category UpdatableTriangularizationAlgorithmType is defined in `CharSet` and extends TriangularizationAlgorithmType. UpdatableTriangularizationAlgorithmType takes two parameters. The first parameter is denoted by `T` and models the elements of the collections that are triangularized. `T` has to implement PrimitiveType and provide functions for deciding whether or not an element is zero (zero?) or a constant (ground?). Typical values for `T` are a domain for polynomials or differential polynomials. The second parameter of UpdatableTriangularizationAlgorithmType has to implement

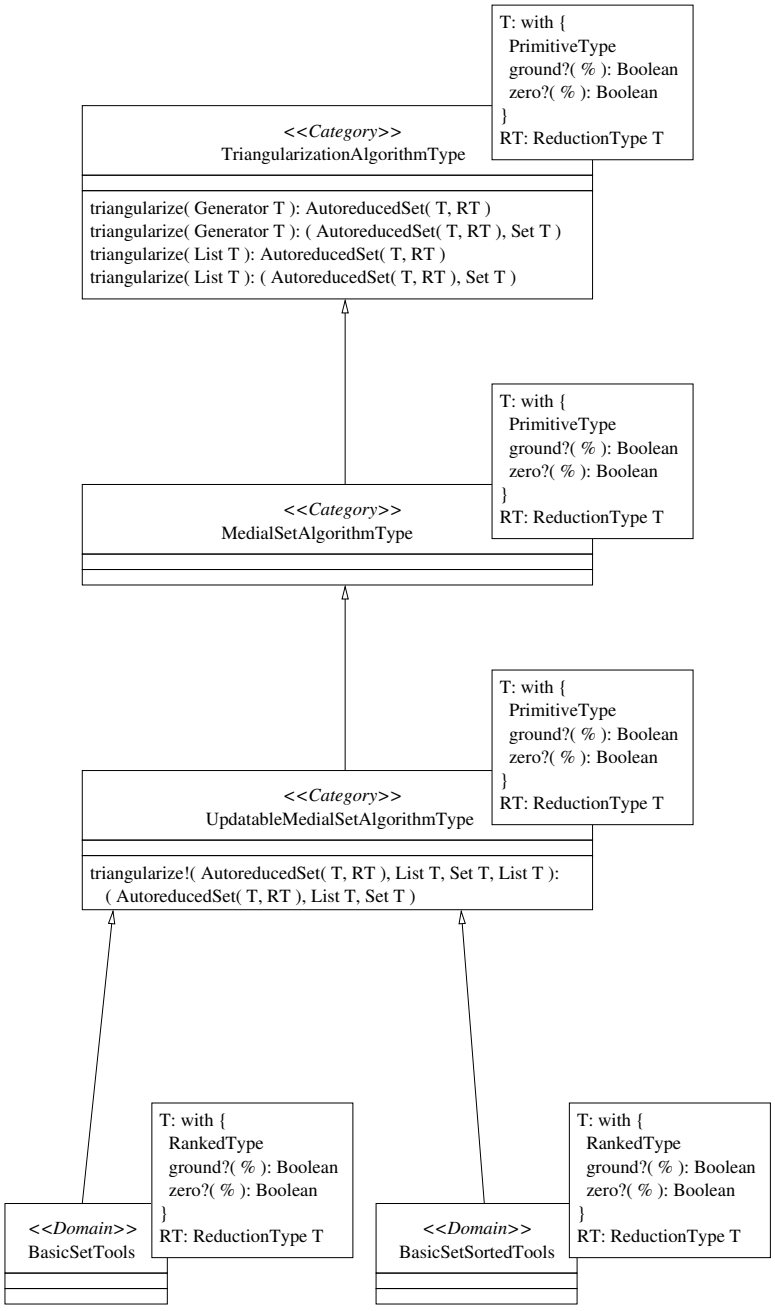

Figure 5.1: The categories and domains for medial set computations

a ReductionType for T. Algorithms have to compute a triangularization with respect to this ReductionType.

For every triangularize function of TriangularizationAlgorithmType, UpdatableTriangularizationAlgorithmType provides additionally a function that allows to update a previous computation. Therefore, UpdatableTriangularizationAlgorithmType implements four triangularize functions for updating triangularizations. These functions have the same signature as those of TriangularizationAlgorithmType but take an additional Pointer and return an additional Pointer. These Pointers reflect the internal state of the computation. When updating a triangularization, the returned Pointer of the previous computation has to be passed to the function call that is to update the triangularization. If a triangularization is started from scratch, nil has to be passed as status of the previous computation. The elements of T from the previous computations do not have to be passed to the updating function call again. Only those elements of T by which the triangularization has to be updated, have to be passed to the call of triangularize. If a domain implements UpdatableTriangularizationAlgorithmType, then the four triangularize functions from TriangularizationAlgorithmType and the four functions to update a triangularization all have to implement the same triangularization algorithm.

The following piece of code illustrates how the triangularize functions can be used for updating a triangularization.

```
--starting without a previous computation
local status: Pointer := nil;
...
--a triangularization for the elements in P will be computed
( C, status ) := triangularize( P, status );
--C is the computed triangularization
--status holds the internal status of the computation
...
--P is set to a list of elements by which the previous
--triangularization has to be updated
...
( C, status ) := triangularize( P, status );
--C is the updated triangularzation
--status holds the internal status of the updated
--computation
```

In Section 5.2 the category UpdatableMedialSetAlgorithmType has been defined for updating medial sets. UpdatableMedialSetAlgorithmType is also of TriangularizationAlgorithmType and used for updating. Nevertheless, UpdatableTriangularizationAlgorithmType is not used for providing functions to update a triangularization. This design is due to the special requirements of medial set updates. Algorithms that update medial sets typically need to separate those elements that are used in the medial set from those that are not. This separation is typically a by-product of the medial set computation. Therefore, it is incorporated into the signature for updating a medial set. For arbitrary

triangularization algorithms, such a separation is not as important. Therefore, there are two categories for updating a triangularization; UpdatableMedialSetAlgorithmType for updating medial sets and UpdatableTriangularizationAlgorithmType for arbitrary triangularizations.

`CharSet` provides two implementations of UpdatableTriangularizationAlgorithmType.

The first implementation of UpdatableTriangularizationAlgorithmType is AutoreducedSetTools, which implements *GenCharSet* (see Algorithm 5.3) of [50]. AutoreducedSetTools takes three parameters. The first two parameters correspond to `T` and the ReductionType of UpdatableTriangularizationAlgorithmType. The third parameter is a MedialSetAlgorithmType. This MedialSetAlgorithmType is used for computing the medial set in step 4 of *GenCharSet*. If the MedialSetAlgorithmType computes a basic set, then AutoreducedSetTools is equivalent to the *CharSet* algorithm of [50].

Algorithm: *GenCharSet*

Input:
P: a set of elements in `T`
Output:
A: a autoreduced subset of the differential ideal generated by P with
$\forall Q \in P : A$ reduces Q to 0.

1: $F \leftarrow P$
2: $R \leftarrow P$
3: **while** $R \neq \emptyset$ **do**
4: $A \leftarrow$ medial set of F
5: **if** A contains a constant **then**
6: $R \leftarrow \emptyset$
7: **else**
8: $R \leftarrow$ non-zero remainders of $F \backslash A$ with respect to A
9: $F \leftarrow F \cup R$
10: **end if**
11: **end while**

Algorithm 5.3: The *GenCharSet* algorithm.

Although the name *GenCharSet* in [50] presumes that the output of AutoreducedSetTools's triangularize functions are characteristic sets for the input, this is only valid in Wang's definition of a characteristic set (See page 44 for a definition of Wang characteristic set). Nevertheless, the output A of AutoreducedSetTools computes a characteristic set of the differential ideal generated by its input P if and only if[52]

- P is finite,
- those indeterminates of P that have the same class also must have the same order,

[52]The presented conditions refer to the polynomial context. However, `T` need not have a notion of "class", "order", "separant", or "initial". `T` only needs to provide ground?, zero? and PrimitiveType.

- every separant of A is a constant, and
- every initial of A is a constant.

`CharSet`'s second implementation of UpdatableTriangularizationAlgorithmType is the domain CoherentAutoreducedSetTools. CoherentAutoreducedSetTools implements a variant of the *Coherent-Autoreduced* algorithm. The original *Coherent-Autoreduced* algorithm can be found in [38]. The modification incorporates medial sets and adjust the handling of non-zero constants to the needs of `CharSet`. This modified algorithm is called *MedialSetCoherentAutoreduced* and is presented as Algorithm 5.4 in this report. *MedialSetCoherentAutoreduced* works like the *GenCharSet* algorithm, but also takes cancellation consequences (Δ-polynomials in the case of differential polynomials with differential reduction) of the reduction process into account. CoherentAutoreducedSetTools takes three parameters. The first parameter is used as domain for the elements of the collections that are triangularized. This parameter is again denoted by `T` and requires functions for checking whether or not an element is zero (zero?) or a constant (ground?). Furthermore, `T` has to provide a ranking. The second parameter is a TriangularizationReductionType on the elements of `T`. The algorithm computes a triangularization with respect to this reduction. The third parameter is of type MedialSetAlgorithmType and denotes an algorithm to compute a medial set. In a typical setting, `T` is a domain for differential polynomials, the reduction is differential reduction and the *BasSetSorted* algorithm is used to compute medial sets.

Algorithm: *MedialSetCoherentAutoreduced*

Input:
P: a set of elements in `T`
MedialSet: a procedure to compute a medial set
Output:
A: a coherent autoreduced set such that $[A] \subseteq [P] \subseteq [A] : H_A^\infty$.

1: $S \leftarrow \emptyset$
2: $A \leftarrow$ empty autoreduced set
3: $R \leftarrow P \setminus \{0\}$
4: $D \leftarrow \emptyset$
5: **while** $R \cup D \neq \emptyset \ \wedge \ S$ does not contain constants **do**
6: $\quad S \leftarrow S \cup R \cup D$
7: $\quad A \leftarrow \mathit{MedialSet}(S)$
8: $\quad R \leftarrow$ non-zero remainders with respect to A of $R \setminus A$
9: $\quad D \leftarrow$ non-zero remainders with respect to A of the Δ-polynomials of A
10: **end while**

Algorithm 5.4: The *MedialSetCoherentAutoreduced* algorithm.

In general, CoherentAutoreducedSetTools's triangularize functions compute just a coherent autoreduced A for the input P such that

$$[A] \subseteq [P] \subseteq [A] : H_A^\infty.$$

However, the output A of a triangularization by CoherentAutoreducedSetTools computes a characteristic set of the *differential ideal generated by* its input P if and only if[53]

- P is finite,
- every separant of A is a constant, and
- every initial of A is a constant.

CoherentAutoreducedSetTools is the last algorithm in characteristic set computations that deals with the differential structure of the differential polynomials. The further steps can be carried out in a polynomial setting without any notion of differential structure. These steps are splitting by initials and separants and finally Gröbner basis computations. [38] gives a detailed analysis of these further steps. Alternative algorithms that do not use Gröbner bases can be found in [39] and [32].

The algorithms of [38] are implemented in the `CharSet` library, but considered out of this report's scope, as the further steps are not specific to characteristic set computations. More information about the implementation of these algorithms can be found in the documentation of the `CharSet` library itself (see the documentation of `CharSet`'s ChiDecompositionTools).

In Figure 5.2, all categories and domains of Section 5 are put into context.

[53] Again, the presented conditions refer to the differential polynomial context. `T` need not have a notion of "separant" or "initial". `T` only needs to provide ground?, zero? and PrimitiveType.

Figure 5.2: The categories and domains for characteristic set computations

6 Design Alternatives

The previous parts of this report presented the implementation of the `CharSet` library without discussing design decisions. Such discussions can be found in the current section, which puts the chosen designs in context to alternative designs.

In the beginning of this section the implementation of differential rings is discussed in seven parts. These parts are

- implementing partial differential rings by ordinary differential rings,
- base category considerations for differential polynomial rings,
- derivations as parameters to the categories,
- use of aliases for derivations,
- modelling DifferentialType
- the order parameter to the derivative domains, and
- comparison function for derivatives.

Afterwards the two topics

- differential reduction as mapping to a quotient ring and
- decoupling differential rings and differential reduction

show alternatives for implementing differential reduction. Finally,

- data integrity and modification functions for autoreduced sets and
- structural improvements to autoreduced sets

discuss the implementation of autoreduced sets.

6.1 Implementing partial differential rings by ordinary differential rings

For the discussion of the implementation of differential rings, let

$$\begin{aligned} R_0 &:= R, \text{ and} \\ R_i &:= \left(R_{i-1}, \left\{\delta_i|_{R_{i-1}}\right\}\right). \end{aligned}$$

There are essentially two mathematically equivalent ways to equip the ring R with the differential structure of $\Delta|_R = \{\delta_1|_R, \delta_2|_R, \ldots, \delta_n|_R\}$.

- all derivations $\Delta|_R$ are added in one step—$(R, \{\delta_1|_R, \delta_2|_R, \ldots, \delta_n|_R\})$—or
- the derivations are added step by step—from R_0 to R_1 to R_2 and so on, till finally the desired R_n is obtained.

With the help of `Algebra`'s DifferentialRing[54], the second of the two alternatives can easily be modelled. This approach intrinsically requires a lot of recursion, as can be seen in Table 6.1, where R_n is written down explicitly.

$$
\begin{array}{lcll}
R_0 & := & R & = & R \\
R_1 & := & (R_0, \{\delta_1|_{R_0}\}) & = & (R, \{\delta_1|_R\}) \\
R_2 & := & (R_1, \{\delta_2|_{R_1}\}) & = & \big((R, \{\delta_1|_R\}), \{\delta_2|_{R_1}\}\big) \\
\vdots & & \vdots & & \cdots \vdots \quad \vdots \quad \vdots \ddots \\
R_n & := & \big(R_{n-1}, \{\delta_n|_{R_{n-1}}\}\big) & = & \big(\ldots\big((R, \{\delta_1|_R\}), \{\delta_2|_{R_1}\}\big)\ldots, \{\delta_n|_{R_{n-1}}\}\big)
\end{array}
$$

Table 6.1: Expansion of R_n.

For each recursive step, a new domain has to be instantiated. This instantiation is costly in terms of time and memory.

Typically, such a design also slows down the generation of and the arithmetics on the elements of the differential rings, as lots of conversions between the various recursive domains become necessary. For example the application of the derivation δ_{n-2} to an element of R_n involves the steps

- conversion from R_n to R_{n-1},
- conversion from R_{n-1} to R_{n-2},
- applying the derivation,
- conversion from R_{n-2} to R_{n-1}, and
- conversion from R_{n-1} to R_n.

Therefore, `CharSet` does not make use of DifferentialRing but implements DifferentialType and uses the first of the two alternatives.

6.2 Base category considerations for differential polynomial rings

`CharSet`'s DifferentialPolynomialRingType does not export `Algebra`'s PolynomialRing.

[54] `Algebra`'s category DifferentialRing provides functions for differential rings with exactly one derivation.

However, DifferentialPolynomialRingType exports PolynomialRing0, which is the base category of `Algebra` for multivariate rings. PolynomialRing is only an extension of PolynomialRing0 by further structure and additional functions (see Section 3.3).

If DifferentialPolynomialRingType would export PolynomialRing instead of PolynomialRing0, all domains implementing DifferentialPolynomialRingType would have to provide the extension PolynomialRing of the base category PolynomialRing0. For example extending the multivariate domain DistributedMultivariatePolynomial1[55] to DifferentialPolynomialRingType, would involve reimplementing all extensions that PolynomialRing makes to PolynomialRing0[56]. These extensions are, however, not important for characteristic set computations.

Therefore, DifferentialPolynomialRingType exports only PolynomialRing0 and not additionally PolynomialRing. Nevertheless, those parts of PolynomialRing that do not involve much extra code have been incorporated into DifferentialPolynomialRingType[57].

6.3 Derivations as parameters to the categories

In Aldor it is common practice to parametrize domains and categories. The parameters are used for decoupled, independent sub-parts and serve as hot-spots. With the help of these parameters most type-checking can be performed at compile time.

However, `CharSet` does not carry the derivations as parameters to the categories modelling differential structures (e.g.: DifferentialType).

The reason for this decision is that adding the derivations to the parameters does not bring any benefits, but only adds overhead and complexity.

Assume a new definition for differential rings like ParametricDifferentialRing.

```
ParametricDifferentialRing(
  R: Ring,
  D: List Derivation R
): Category == ...
```

Considering how to model an extension of a differential ring to a differential polynomial ring immediately shows the limitations of the parametric version, as can be seen when investigating ParametricDifferentialPolynomialRing.

[55] In Section 3.3, DistributedMultivariatePolynomial1 can be seen to be the only distributive implementation of multivariate polynomial rings that exports PolynomialRing0. There is no distributive implementation of multivariate polynomials that exports PolynomialRing or any other extension of PolynomialRing0.

[56] Although DifferentialPolynomialRing does not provide PolynomialRing, the domains for extending polynomial rings to differential rings (see Section 3.3) provide the extension PolynomialRing if the provided polynomial ring provides PolynomialRing. Such a conditional export is however only possible for DifferentiallyExtendedPolynomialRing and not for DifferentialPolynomialRing.

[57] These parts are the categories CommutativeRing, CharacteristicZero, and FiniteCharacteristic, and Parsable.

```
ParametricDifferentialPolynomialRing(
  R: Ring,
  D: List Derivation R,
  DIFFRING: ParametricDifferentialRing( R, D ),
  VARS: VariableType
): Category == with {
  ParametricDifferentialRing( %, SomeDerivations );
  ...
}
```

The `SomeDerivatons` parameter to ParametricDifferentialRing has to be a domain satisfying `List Derivation %`. However, `D` cannot be used as this parameter, as `D` denotes the derivations in the coefficient ring (`List Derivation R`). In the definition of the *category* for differential polynomial rings `D` has to be extended to a list of derivations on the differential polynomial ring. Eventhough this is possible, such an approach has a lots of drawbacks. The two most important ones are that such an approach is

- intermixing the separation of code and its interface and
- domains implementing ParametricDifferentialPolynomialRing cannot overload the value of `SomeDerivations`.

The categories of `CharSet` omit the derivation parameters. As a result, Aldor cannot check at compile time whether or not domains provide matching derivations. In exchange, the interfaces of `CharSet` are much cleaner, and do not mix code and its interfaces.

6.4 Use of aliases for derivations

`CharSet` uses numbers and Symbols as aliases for derivations. As can be seen in Section 3.1, derivations are referenced with the help of these aliases. In Aldor, functions can be directly assigned to variables and can be directly passed as parameters. Therefore, derivations could be used directly instead of using numbers and Symbols.

The use of aliases instead of the derivations is based on the need for textual representation of entities. A textual representation of a differential polynomial typically involves denoting derivations. For example the differential polynomial $2\delta_2 y_1 + 3$ in $\mathbb{Q}\{Y\}$ contains δ_2. Here, the symbol δ_2 is an alias for a derivation on $\mathbb{Q}\{Y\}$. Without the symbol δ_2 the corresponding derivation can only be written down by a description of its mapping. Such a description is cumbersome and makes it difficult to read the output. As a result, the aliases for the derivations are necessary.

Without these aliases, further complications would arise, as functions cannot be compared directly in Aldor.

6.5 Modelling DifferentialType

DifferentialType is used in two different contexts with two slightly different semantics. DifferentialType's semantics in ring contexts is described in Section 3.1, while its semantics in non-ring context is presented in Section 3.2.

From a mathematical point of view, DifferentialType has to be separated into two categories, as the mathematical interpretation of the differentiate functions differs. In ring contexts, differentiate lets a derivation mapping act on an element of the ring. In non-ring contexts the mapping is adjoined to the object of interest. For example in the context of ΘY the adjoining of a derivation resolves to a multiplication in the Θ substructure of ΘY.

From a computer science point of view, a derivation is applied to an object in both the ring and the non-ring context. In both situations the differential structure of the ring or non-ring is handled by this application of a derivation.

Splitting DifferentialType up leads to two categories that both model the differential structure of a domain. Furthermore, both categories would provide the same function signatures.

Therefore, the differential structure is modelled by DifferentialType in a ring context and also in a non-ring context. DifferentialType makes no difference whether "applying a deviation" resolves to "let a deviation act on an element" or "multiplying a derivation".

Thereby, the same functionality can be represented with the same category.

6.6 The order parameter to the derivative domains

Section 3.2 shows that DifferentialVariable and OrdinaryDifferentialVariable model the derivatives ΘY and take a parameter denoting the order of derivatives.

To decouple the domain for the derivatives ΘY and its order is more elegant. In such a design, there are two kinds of domains. The first kind of domains models ΘY without introducing an order. The second kind of domains takes an orderless implementation of ΘY and adds an order. The second kind of domains extends the first kind[58]. Such a design also reflects the mathematical setting, where ΘY primarily exists without any notion of order. Afterwards, the order relation is imposed on ΘY.

However, such a design conflicts with the provided categories of Aldor's base libraries. ΘY is used as indeterminates in a polynomial ring. Therefore, a domain for ΘY has to provide `Algebra`'s category VariableType. VariableType, however, requires a total order on its elements. Therefore, it is not possible to model ΘY without a total order and use VariableType.

[58] This design is similar to that of reductions in Section 4. Reductions also operate on a domain and this domain has no notion of reduction of its own.

As a result, DifferentialVariable and OrdinaryDifferentialVariable provide a total order on their elements. To achieve at least some decoupling of the implementation of ΘY and its order, the order takes the form of a parameter to the implementation of ΘY.

6.7 Comparison function for derivatives

The category DifferentialVariableOrderToolsType offers a function $<$ to compare the derivatives ΘY. This function's signature is

```
<: ( VARS : FiniteVariableType,
     DVARS  : DifferentialVariableType( VARS )
   ) -> ( DVARS, DVARS ) -> Boolean
```

which maps a domain for Y (`VARS`) and ΘY (`DVARS`) to a function that compares two elements of ΘY (`( DVARS, DVARS ) -> Boolean`).

As Aldor allows categories to be parametric, a straight forward design is to transform the parameters `VARS` and `DVARS` to the function $<$ to parameters of the category itself. This approach would result in a category similar to ParametricDifferentialVariableOrderToolsType.

```
ParametricDifferentialVariableOrderToolsType(
  VARS   : FiniteVariableType,
  DVARS  : DifferentialVariableType( VARS )
):Category == with {
    <:  ( DVARS, DVARS ) -> Boolean;
}
```

However, such a straight forward design cannot be instantiated properly. Assume ParametricOrder is some implementation of an order on ΘY. Furthermore, let `VARS` be an implementation for Y and `DSYMS` be an Array of Symbols denoting the derivations of ΘY.

```
ParametricOrder(
  VARS   : FiniteVariableType,
  DVARS  : DifferentialVariableType( VARS )
): with {
    ParametricDifferentialVariableOrderToolsType( VARS, DVARS );
} == add {
    <( a: DVARS, b: DVARS ): Boolean == {
       ...
    }
};
```

DifferentialVariable is instantiated by

```
DifferentialVariable(
  VARS,
  DSYMS,
  ORDER
)
```

where ORDER needs to specify a domain for an order on ΘY. Substituting ParametricOrder for ORDER leads to

```
DifferentialVariable(
  VARS,
  DSYMS,
  ParametricOrder(
    VARS,
    DVARS
  )
)
```

where DVARS needs to specify the domain for the order. However, DVARS has to refer to the domain that is currently being built up. This raises the problem of instantiating the domains recursively with the same parameters. This problem can best be seen in its representation as Aldor code:

```
DifferentialVariable(
  VARS,
  DSYMS,
  ParametricOrder(
    VARS,
    DifferentialVariable(
      VARS,
      DSYMS,
      ParametricOrder(
        VARS,
        ...
      )
    )
  )
)
```

The reason for this problematic recursion is the fact that ParametricOrder itself takes the implementation of ΘY on which it has to act as parameter. Eliminating this parameter also eliminates the recursion.

This observation motivates the design of DifferentialVariableOrderToolsType where the implementation of ΘY is not a parameter to the category itself. There, the implementation of ΘY is a parameter to the function $<$. As a result domains for ΘY and its orders can be properly instantiated.

6.8 Differential reduction as mapping to a quotient ring

From a mathematical point of view, differential reduction of an element of $R\{Y\}$ by some $B \subseteq R\{Y\}$ can be interpreted either

- as mapping from $R\{Y\}$ to $R\{Y\}$ or
- as mapping from $R\{Y\}$ to $R\{Y\}_{/[B]}$.

The first model is simpler than the second one, as its domain and codomain are the same. However, the latter model provides additional structure, by mapping to $R\{Y\}_{/[B]}$ instead of mapping to $R\{Y\}$. Therefore, the latter model converts implicit knowledge (elements of $R\{Y\}$ are separated into cosets of elements with respect to the reduction by B) to explicit knowledge (mapping to $R\{Y\}_{/[B]}$).

Aldor is a fully typed system. Fully typed systems benefit a lot from converting implicit knowledge about values into explicit knowledge. Therefore, the mapping to $R\{Y\}_{/[B]}$ is the preferred model to implement differential reduction. The `Algebra` library for example uses the second model for polynomial reduction on univariate polynomials[59].

Nevertheless, `CharSet` implements the first model. There are two reasons for this decision.

- Characteristic set algorithms typically require the result of a reduction to be in $R\{Y\}$ again. Although it is possible to promote an element of $R\{Y\}_{/[B]}$ to $R\{Y\}$ after choosing an appropriate[60] system of representatives, this promotion causes further function calls.
- Mapping to $R\{Y\}_{/[B]}$ involves creating new instances of the domain for every B. B changes a lot during characteristic set computations, therefore a lot of new instances have to be created. As these instantiations are typically time and memory consuming, the result is a performance drop off.

Although the second model of differential reduction represents the information better, the first model has been chosen, as it is better adapted to characteristic set computations and takes more care of resources.

[59] A quick overview of `Algebra`'s polynomial reduction for univariate polynomials can be found in the beginning of Section 4.

[60] A system of representatives is appropriate if all representatives are reduced with respect to $[B]$.

6.9 Decoupling differential rings and differential reduction

Section 4 shows the decoupling of `CharSet`'s differential reduction implementation and `CharSet`'s implementation of differential rings.

However, in the context of characteristic set computations on differential polynomial rings, only differential reduction is of interest. Therefore, an alternative design is to incorporate the differential reduction into the domains for the differential polynomial rings.

Such a design allows to optimize the differential reduction algorithm directly for the used differential polynomial ring implementation. However, these optimizations are also possible with `CharSet`'s decoupled design, by restrictions on the domain the reduction acts on.

Furthermore, the decoupled design of `CharSet` eases reuse of both differential polynomial ring implementations and reductions. Differential rings may be used in contexts, where no reduction is needed and reduction algorithms may be used on more than one differential polynomial ring implementation.

For example, when performing non-differential Gröbner basis computations with differential polynomials, it is important to clarify that not differential reduction, but polynomial reduction is used for reduction[61]. In the merged, alternative design, the differential polynomial ring provides a reduction algorithm by itself. However, this reduction algorithm must not be used, as it denotes differential reduction. An algorithm for polynomial reduction has to be used. In `CharSet`'s decoupled design, the differential polynomial ring does not provide a reduction algorithm by itself. Therefore, no semantic ambiguity between various reduction algorithms arises.

6.10 Data integrity and modification functions of autoreduced sets

Section 4 showed that all functions for modifying values of an AutoreducedSet map to Partial values and are therefore allowed to fail. These Partials are in conflict with the modify operations of the categories for data structures of the `Algebra` library, as already discussed.

The argument for introducing the Partial anyway is that incautious modifications of an AutoreducedSet may ruin the autoreduced property. Nevertheless, in many situations it is known in advance that the desired modification of an AutoreducedSet does not harm the autoreduced property. In these situations, `CharSet` does not allow shortcuts that evade the Partial in the result and avoid unnecessary safety checks.

Assume `CharSet`'s AutoreducedSet provides such "shortcut" functions. If it cannot be decided in advance whether or not a modification of an AutoreducedSet harms the

[61] This setting is not artificial. It typically arises when decomposing a coherent autoreduced set of a differential ideal into an intersection of ideals that are generated by characteristic sets [38].

autoreduced property, the Partial functions can be used. If it is clear in advance that a modification of an AutoreducedSet does not affect the autoreduced property, the shortcut function can be used.

Such a design allows more efficient programs at the cost of data integrity. *Any* algorithm can access the shortcut functions. Even those algorithms that harm the autoreduced property can use the shortcut functions. Aldor does not allow to restrict access to the shortcut functions to dedicated algorithms[62]. As the performance penalty for the Partial functions is only marginal and data integrity is of utter importance, AutoreducedSet does not provide shortcut functions for modifications.

6.11 Structural improvements to autoreduced sets

As discussed earlier, LibCharSet's AutoreducedSet domain cannot export categories of `Algebra`'s data structure framework (DynamicDataStructureType, LinearStructureType), as AutoreducedSet's functions to modify elements map to Partial.

Assume `CharSet`'s AutoreducedSet provides the shortcut functions to modify AutoreducedSets from the previous discussion. Then AutoreducedSet can implement all the functions of LinearStructureType and FiniteLinearStructureType. However, AutoreducedSet still cannot export these categories themselves.

The reason for this is illustrated by the function set! of LinearStructureType. set! of LinearStructureType allows to modify an element. It is not important which element is changed to which value. The set! function of AutoreducedSet with the same signature as the set! function of LinearStructureType has to restrict the possible modifications to those not harming the autoreduced property. This restriction cannot be modelled in Aldor code and has to go into the documentation of AutoreducedSet's set! function.

Although, both set! functions bear the same signature, their semantics are inherently different. Therefore, AutoreducedSet's set! does not meet the criteria for LinearStructureType's set!. As a result, AutoreducedSet cannot export LinearStructureType. The same arguments holds for DynamicDataStructureType.

As a result, the shortcut functions cannot be used for AutoreducedSet to export LinearStructureType, DynamicDataStructureType, or any of their descendants.

[62]For example C++ provides the keyword `friend`. If a function or class `A` is declared to be a `friend` of a class `B`, then `A` may use `B`'s private data.

7 Preparations in the Aldor environment

Throughout the whole Section 7, a strict separation between the Aldor environment and entities outside of the Aldor environment is made. The entities outside of the Aldor environment are not distinguished further. Therefore, the term "foreign environment" is used for the rest of this section, when referring to such entities outside of the Aldor environment. Typical foreign environments are Maple or an environment for the programming language C.

To be able to call Aldor code from foreign environments, preparations have to be made to the source code within the Aldor environment. This section presents these preparations and discusses possible alternatives.

This section is divided into three parts. The first part presents how Aldor code can be connected from foreign environments. It turns out that object files should be used for this purpose. The second part elaborates on how functions can be exported in object files and what data structures can be used to pass values to the exported functions. Character strings in C format are chosen to serve the latter purpose. The third part discusses how Aldor's expressions can be brought to and parsed from character strings in C format. Finally, this part presents the export wrapper for the triangularize algorithm of CoherentAutoreducedSetTools.

7.1 Connections to foreign environments

In this part, the possible ways to connect to Aldor code are discussed.

Two ways can be identified to connect to Aldor in a `GNU/Linux` environment. These are:

- spawning an interactive Aldor shell, and
- using Aldor code in one of the various output formats of the Aldor compiler.

After a short discussion of the first alternative, the second one is presented in detail.

Foreign environments can spawn a interactive instance of the Aldor compiler and pass appropriate commands to this instance. The foreign environment can then parse and use the output of the interactive Aldor compiler. In this approach, Aldor does not use libraries that have been compiled to machine code, but libraries in an intermediate code that needs to be interpreted. Therefore, this approach is inherently slow and not considered further.

The Aldor compiler can compile to ten different formats. These formats are

- source after inclusion (`.ai`),
- macro expanded parse tree (`.ap`),

- symbol[63] information (`.asy`),
- machine independent code (`.ao`),
- FOAM code (`.fm`),
- Lisp code (`.lsp`),
- C code (`.c`),
- C++ code (`.cpp`),
- object code (`.o`), and
- executable code.

The first five formats (source after inclusion, macro expanded parse tree, symbol information, machine independent code, FOAM code) are not considered further, since neither common tools for processing these files can be found nor are there any foreign environments available, where these formats are of use. Among the remaining five formats object code is chosen to become the desired output format. Object code contains machine code and does not need to be compiled any further. Object code can be passed to a linker directly. Therefore, it is more convenient to use object code than either of the three generated source code formats (Lisp code, C code, C++ code). Object code can be used to build libraries, shared objects, and executables. Therefore, object code gives more flexibility than compiling directly to executables. For this report, object code is used to make algorithms inside the Aldor environment accessible to foreign environments.

Object files contain relocatable machine code in the `ELF` [64]. Object files export symbols. By referring to these exported symbols, machine code within the object file can be executed. To generate object files, the parameter `-Fo` has to be passed to the Aldor compiler.

7.2 Exporting Aldor's functions

This part provides a detailed description about how Aldor functions can be exported in object files and what domains can be used for passing values to foreign environments.

First, the default exports of Aldor are discussed. Afterwards, the necessary Aldor constructs are presented to export Aldor functions in object files. This is followed by a discussion about possible data structures to pass Aldor values to foreign environments. Finally, a small code snippet shows the results of this section.

[63] In this context Symbol is not to be confused with symbol. While the first is meant to refer to `Aldor`'s domain for read only character strings, the latter is used in a general computer science sense.

[64] `ELF` is an abbreviation for "executable and linking format". At the time of writing this report `ELF` is the standard file format for executable files, relocatable code and shared objects on `GNU/Linux`. Detailed information on `ELF` can be found in [49] and [24].

Per default, Aldor exports symbols for domains, categories and their functions in object files. However, these exported symbols are not the names of the domains, categories, and functions, but their names with additional prefixes and postfixes. Furthermore, the resulting symbols are cut off after 30 characters[65]. Additionally, these kind of exports are considered not to be used by the public, since they are not documented in [27] at all. For these three reasons, these default exported symbols are not used for connecting foreign environments.

In Aldor functions can be exported as symbols within an object file, by using Aldor's `export` keyword. Such exported symbols are neither prefixed nor postfixed. Even the symbols name is not truncated to 30 characters. The `export` keyword takes two parameters. The first parameter is a function or collection of functions to export. The second parameter is a language to export to. Aldor supports three languages, which are Lisp, Fortran, and C [66]. However, functions exported to Lisp are effectively not exported in the object file. Exporting to Fortran is only suggested [27, section 20.4] if the exported symbol consists only of up to six letters, which should all be uppercase letters. Furthermore, it might be necessary to adjust some of Aldor's configuration for some platforms [27, section 20.5], when exporting to Fortran. As a result both Lisp and Fortran are neglected and C is chosen to be the language for exporting functions to foreign environments.

The current Aldor compiler does only allow to export functions to C that are defined at the top level of an Aldor source file[67]. An exported function at the top level scope, however, may call arbitrary functions, especially those that are not at top level scope. Therefore, this restriction is considered not too burdensome, since it can be overcome by building top level scope wrappers.

The algorithm, which is chosen to be exported to foreign environments in this report, typically involves polynomials. Although typical foreign environments (for example MAPLE) provide implementations of polynomials, the internal data structure of the foreign environment cannot be expected to meet all of Aldor's various implementations of polynomials. Therefore, the type of the parameters of the exported function has to be considered further. The possible options are

- to use Aldor's data representation,
- to use the data representation of the foreign environment, or

[65] To cut off symbols after 30 characters is a default for the Aldor compiler. This behaviour can be modified by passing `-C idlen=n` to the Aldor compiler, where `n` is a number. Setting `n` to 0, turns the truncation off. Turning truncation off resolves the problem of cutting off symbols, but does not resolve the problem of getting to know the prefixes and postfixes.

[66] Aldor also provides Builtin. Builtin is used to refer to the compiler's internal domains and functions. SInt is an example for such a domain. Although Builtin is treated as a language, one could export to, it makes no sense to export to Builtin, as compiler internal definitions cannot be modified. Nevertheless, the compiler compiles source files that export to Builtin without errors. However, the functions exported to Builtin cannot be found in the resulting object files and are therefore effectively not exported. Besides the Builtin language, further languages can be defined, by generating domains that have ForeignLanguage. The compiler accepts such definitions and exports. The functions exported to such new languages, however, are again not exported when exporting to such a new language.

[67] A similar a restriction can be found in [27] for exporting functions to Fortran. Nevertheless, the restriction for exporting functions to C could not be found documented anywhere.

- to define a new representation.

Using Aldor's representation involves to introduce Aldor's types in every used foreign environment. Using the data representation of the foreign environment causes to introduce the foreign environment's types to Aldor for every different foreign environment. Both alternatives are possible and need to convert data only once. However, both alternatives are likely to need updates as new versions of foreign environments appear.

Positive integers of MAPLE are a good example. Until version 9, MAPLE used a custom data representation for positive integers. From version 9 on, MAPLE uses a mixture of its custom data representation and the GNU Multiple Precision Arithmetic Library [11][68]. Such modifications are typically hidden from a MAPLE user, but have impact when passing values using MAPLE's internal data representation.

As this example shows, using the internal data structure of either Aldor or the foreign environment may become a lethal factor to the connections between Aldor and the foreign environment, if the internal data format changes. Therefore, neither the data representation of Aldor, nor the data representation of the foreign environment is used.

A new representation is used, which can easily be converted to and from Aldor's values and is expected to be implemented already in most foreign environments: character strings in C format. For example both, MAPLE and MATHEMATICA, are able to convert their expressions to and from character strings in C format. However, foreign environments use different formats for their character string encoded expressions. Therefore, algorithms have been implemented in Aldor that convert Aldor's expressions to and from strings in the format of the desired foreign environments. These algorithms are separated from the characteristic set implementation and are bundled in the `ExtIO` library. More details on this topic can be found in Section 7.3.

Character strings in C format are implemented by `Aldor`'s domain String. The examples of [27] used String directly for passing strings between C and Aldor. Nevertheless, this report does not use String itself for passing strings. This report uses Pointers to a String's character representation. The reason for this different approach are the diverse internal representations of character strings in C, the release version of `Algebra`, and the debug version of `Algebra`.

In C, character strings are realized as pointers to the first character. In the release version of `Aldor`, character strings are also implemented as pointers to the first character. The debug version of `Aldor`, however, does not only store the characters, but also the number of characters. Therefore, passing a String of the debug version to C, causes to pass a pointer to the memory region which contains first of all the number of characters followed by alignment bytes and finally the character string. The C environment, however, expects only the pointer to the first character of the character string. Therefore, the use of String does not give the desired results for the debug library and is error

[68] MAPLE's kernel from version 9 onwards uses a threshold to determine, when which implementation is used. In version 9.5 on a 32-bit architecture, MAPLE uses the custom presentation for positive integers that are less than 10^{72}. For positive integers that are greater than or equal to 10^{72}, MAPLE switches to the GNU Multiple Precision Arithmetic Library. Further details can be found in [47].

prone. Using Pointers to the Strings character data does not introduce performance losses but works with both, the release and the debug version of `Aldor`. Therefore, the use of String is neglected in favor of Pointer to increase the usability and robustness of the interface.

The following code snippet collects the ideas of this section and shows, how an Aldor function can be exported.

```
...
...
--exporting a top level function
export {
  someTopLevelFunc: ( Pointer, ..., Pointer ) -> Pointer;
} to Foreign C;


--implementing the function in top level scope
someTopLevelFunc( p1: Pointer, ..., pn: Pointer ): Pointer == {
  local string1: String := string( p1 );
  ...   ...      ...    ......     ...
  local stringn: String := string( pn );
  local result : String := empty;

  -- perform operation with the strings and fill
  -- the result

  pointer result;
}
...
...
```

7.3 Converting Aldor's expressions to character strings

The previous section motivates that character strings in C format are used for passing values to foreign environments. This section discusses the possibilities to transform values of Aldor domains to such character strings and vice versa.

In the beginning of this section `Algebra`'s framework for expression trees is presented. The framework's efforts to convert expression trees to various languages are discussed and put into context with the triangularize algorithm of CoherentAutoreducedSetTools, as this algorithm is to export. Thereby, necessary extensions of the expression tree framework are discussed. The next part describes `ExtIO`'s abstraction of computer algebra systems, which is used to convert expression trees to strings in the desired format and vice versa. Finally, the exported algorithm is presented and its options are discussed.

In order to convert the values of only some Aldor domains to character strings, it suffices to formulate functions that perform this transformation for just these domains. However,

Algebra follows a more general approach by providing an expression tree framework. Most domains and categories of Aldor and Algebra are incorporated in this framework and allow to convert their values to a simple tree data structure. In this tree data structure, the leaves are numbers and/or Symbols and the inner nodes are operations. These types of leaves and inner nodes are used to express a value of a domain and thereby convert the hidden representation and data structures of a value into a simple and common data structure. These trees are called expression trees and can be evaluated back to proper values or written to a TextWriter in either Aldor, Axiom, C, Fortran, infix, Lisp, MAPLE, or TEX notation.

Algebra implements expression trees in the domain ExpressionTree. The category ExpressionTreeOperator is used for modelling inner nodes of an ExpressionTree and the domain ExpressionTreeLeaf models the leaves of an ExpressionTree.

The algorithm that is exported in this section deals with differential polynomials and is connected to MAPLE, MATHEMATICA, and a command line utility. Algebra does not provide a derivation ExpressionTreeOperator. But differential polynomials cannot be properly represented by expression trees without such an ExpressionTreeOperator. Therefore, a derivation ExpressionTreeOperator has to be implemented. Furthermore, Algebra does not provide functions to write ExpressionTrees in MATHEMATICA's format. To implement this output mechanism is a second necessary extension of Algebra's expression tree framework.

While the additional derivation ExpressionTreeOperator can be easily implemented, it is hard to add MATHEMATICA to the output formats. For this observation it is crucial to see how outputting ExpressionTrees works. As all possible output formats work alike, MAPLE is used to illustrate the workings of the mechanism.

To output an ExpressionTree in MAPLE's syntax, the function maple of ExpressionTree is called with a TextWriter to write to and an ExpressionTree to output as parameters. This function checks whether or not the top most node in the expression tree is a leaf. If this node of the expression tree is a leaf, then the maple function of ExpressionTreeLeaf is called with this leaf. If the top node of the expression tree is an operator, then the maple function of the operator is called. This function takes the TextWriter to output to and the operator's children from the expression tree as arguments. As a result, each operator has to know how to output itself and its children for each language.

Adding MATHEMATICA's format to the output formats would involve extending ExpressionTree and all available ExpressionTreeOperators by a function to output the operator to MATHEMATICA's format. These scattered extensions are a direct consequence of Algebra's mingling of data structure (expression trees) and output mechanism of ExpressionTrees. Scattered code is both hard to maintain and hard to understand. It is considered to be more flexible to decouple the expression trees from the output functions. Therefore, Algebra's framework for expression trees is only used to store and generate the expression trees, not to output them in any language's format. The latter functionality is delegated to the computer algebra system framework of the ExtIO, which has been developed in the course of this report.

ExtIO provides the category ComputerAlgebraSystemType to encapsulate functions

for converting Algebra's ExpressionTrees to strings in a computer algebra system's format and vice versa. The first important member of ComputerAlgebraSystemType is the Symbol identifier. identifier announces the targeted computer algebra system. The following constant and functions are all understood in the scope of the described computer algebra system. encodeAsString! is used to convert an ExpressionTree to the format of the target computer algebra system's format. This function takes the expression tree to convert as parameter and gives the result as String. In the context of Section 7.2, this function is important, as the resulting String can easily be converted to a Pointer by calling String's pointer function. The constant parserDomain of ComputerAlgebraSystemType is a ParserReader[69] for the described computer algebra system. With the help of this constant, Strings in the computer algebra system's format can be parsed to ExpressionTrees. ComputerAlgebraSystemType also provides the function encodeAsString, which encodes Aldor's Strings in the computer algebra's format. Additionally, encodeAsError and encodeAsWarning are found in ComputerAlgebraSystemType to format errors and warnings using the computer algebra system's syntax.

ExtIO provides two implementations of ComputerAlgebraSystemType. The first implementation is MapleTools. MapleTools addresses MAPLE's syntax. Since Algebra already comes with support for MAPLE, these functions are reused. InfixExpressionParser[70] of Algebra serves as parserDomain and the maple function of ExpressionTree is used to convert expression trees to Strings.

The second implementation of ComputerAlgebraSystemType is MathematicaFullFormTools. MATHEMATICA is able to accept input in various formats, as can be seen in [55]. MathematicaFullFormTools treats only the FullForm format. This restriction does not hinder usability, since MATHEMATICA provides functions to convert between FullForm and any other format. The parserDomain of MathematicaFullFormTools is MathematicaFullFormParser, which is a straight forward implementation of a parser for MATHEMATICA's FullForm format. MathematicaFullFormParser is also part of ExtIO. The functions for converting ExpressoinTrees to Strings are implemented in MathematicaFullFormTools itself.

Finally, all the necessary pieces of information are put together and the triangularize algorithm of CoherentAutoreducedSet can be exported in an appropriate format to an object file. The necessary code is put together in CharSet's ExportCoherentAutoreducedSetTools. The domain exports only the function coherentAutoreducedSet. This function takes five Pointers as parameter and returns another Pointer. All Pointers refers to character strings in C format. The first input parameter refers to the identifier of an implementation of ComputerAlgebraSystemType. The next three parameters are supposed to be stored in the format of this implementation. The second parameter refers to a list of differential polynomials for which a coherent autoreduced set is to be computed.

[69] In Algebra, a Parser is bound to exactly one TextReader. Therefore, a Parser cannot be reused, after the attached TextReader ran dry of new characters. For generating new Parsers the category ParserReader is provided by Algebra. This category contains a function parser that takes a TextReader as input and returns a Parser, that acts on the passed TextWriter.

[70] The implementation of the domain InfixExpressionParser can be found in Algebra. InfixExpressionParser is a ParserReader and its generated Parsers do not only accept infix notation, but also function application in MAPLE's syntax.

The third and fourth parameter refer to the dependent and independent indeterminates of the domain for the differential polynomials. The fifth parameter allows to adjust further parameters for the domains for the differential polynomials. These further options have to be comma separated. If contradicting options are specified, the latter option has precedence. The possible options for the order of the derivatives are

- `OrderlyEliminationOrder`
  uses DifferentialVariableOrderlyEliminationOrderTools to order the derivatives,
- `LexicographicEliminationOrder`
  (default) uses DifferentialVariableLexicographicEliminationOrderTools to order the derivatives,
- `LexicographicOrder`
  uses DifferentialVariableLexicographicOrderTools to order the derivatives, and
- `OrderlyOrder`
  uses DifferentialVariableOrderlyOrderTools to order the derivatives.

The possible options for the exponent vectors are

- `presortClasses`
  is only valid for orders which export DifferentialVariableEliminationOrderTools-Type and refines the chosen exponent vector implementation by wrapping it with ClassPresortedDifferentialExponent,
- `SortedListExponent`
  uses SortedListExponent for storing exponent vectors,
- `ListExponent`
  uses ListExponent for storing exponent vectors,
- `ListSortedExponent`
  uses ListSortedExponent for storing exponent vectors, and
- `CumulatedExponent`
  uses CumulatedExponent for storing exponent vectors.

The possible options for the implementation of the polynomials are

- `SparseMultivariatePolynomial`
  models the polynomials by SparseMultivariatePolynomial and
- `DistributedMultivariatePolynomial1`
  models the polynomials by DistributedMultivariatePolynomial1.

The result Pointer of coherentAutoreducedSet is a list of polynomials, encoded as `C` string using the format of the supplied computer algebra system. These polynomials are the result of the algorithm.

Using the technique presented in Section 7.2, the function coherentAutoreducedSet is exported in the object file by the symbol `_charset_coherentAutoreducedSet`.

8 Connecting to the characteristic set library from within MAPLE

In Section 7 the triangularize algorithm of CoherentAutoreducedSetTools has been wrapped by coherentAutoreducedSet of ExportCoherentAutoreducedSetTools and this wrapper has been exported to an object file by the symbol `_charset_coherentAutoreducedSet`. This section shows how this object file and the code at this symbol can be accessed and used conveniently from within MAPLE.

This section consists of two parts. The first part discusses the possibilities of connecting Aldor's exported code to MAPLE. The second part presents further steps to adapt the imported code to the MAPLE environment and to facilitate calling the imported code.

8.1 Accessing Aldor's code from within MAPLE

In this section the exported Aldor code of Section 7 is used to examine the possibilities of MAPLE to import foreign code. The focus of this section is on the connectivity of MAPLE, not on smooth integration. Therefore, this section only brings Aldor's code to MAPLE's environment. Section 8.2 is dealing with integrating Aldor's code better into MAPLE's environment.

In the beginning of this section, the achievements of Section 7 are briefly recapitulated. Then `define_external`, MAPLE's function for importing foreign code, is introduced. This function does not accept foreign code in the object format of Section 7, but requires foreign code as dynamic shared objects. After a presentation of dynamic shared objects, a thorough discussion about migrating object files to dynamic shared objects forms the next part. Finally, the missing details of `define_external` are covered and its usage is illustrated by an example.

In Section 7 Aldor code has been exported in an object file. The exported code used C calling conventions. Therefore, the exported function appears as proper C function within the object file. Wherever a C function within an object file can be used, the object file of Section 7 can be used.

Within MAPLE[71], the function `define_external` is used to bring external functions into scope. These external functions can be C, Fortran, or Java functions. As Section 7 used C calling conventions for exporting Aldor code, focus is put on calling C code from within MAPLE. Fortran and Java are not considered further[72].

For being able to call external C code from within MAPLE via `define_external`, the

[71] For this report MAPLE has been used in version 8. The presented techniques have also been tested with MAPLE 9.5 and no problems arose. However, this section uses MAPLE to refer to MAPLE in version 8, if not noted otherwise.

[72] Details about how to use external Fortran or Java code from within MAPLE can be found in [47].

C code has to be compiled to a dynamic shared object[73]. However, the object file of Section 7 is not a dynamic shared object. Therefore, the description of the command `define_external` is interrupted in favor of a presentation of dynamic shared objects. After a further discussion how object files can be used to build dynamic shared objects, `define_external`'s presentation is resumed on page 77.

Dynamic shared objects contain executable code that can be used by other applications. These applications can be statically or dynamically linked against dynamic shared objects or load them at run-time.

In the case of static linking, the dynamic shared object acts as library, whose code is incorporated into the resulting executable. After linking, the resulting program can be executed without the dynamic shared object.

In the case of dynamic linking, the resulting executable does not incorporate the dynamic shared object's code but only the names of the symbols that are exported by a dynamic shared object. If the resulting program is executed, the operating system loads the program and invokes a dynamic loader, which loads the linked dynamic shared objects. After binding the function calls of the executable to the appropriate functions in the dynamic shared objects, the execution starts. If further executables also need already loaded dynamic shared objects, running instances are reused. This strategy leads to smaller executables, less memory overhead and better maintainability of components. Nevertheless, there is a little startup time penalty, since symbols have to be relocated when loading a dynamic shared object.

In the case of run-time usage, the application loads a dynamic shared object at run-time only if and when it is needed. However, the program has to load the dynamic shared program explicitly[74]. In the previous two cases, the operating system managed the loading of dynamic shared objects automatically. The executables did not have to bother with these tasks. However, for run-time usage, the executable does not have to be linked with the dynamic shared object and does not need to declare at build time which dynamic shared object it will probably load. Therefore, this method is commonly used in plug-in frameworks. Again, already loaded dynamic shared objects are reused.

Dynamic shared objects are created from position independent object files[75]. These

[73]The format to embed the external C code in depends on the platform on which Maple is running. In a general setting, Maple requires external code to be compiled to shared libraries in the platform's standard format for shared libraries. For `GNU/Linux`, the standard format for shared libraries are `ELF`'s dynamic shared objects. Therefore, dynamic shared objects are used in this report. On other platforms, external C code might have to be compiled to different formats. For example on `Windows` platforms, the standard format for shared libraries is `DLL` [15], which is short for dynamic link library. Therefore, on `Windows` platforms, external C code has to be compiled to `DLL`s to be usable from within Maple.

[74]Although loading a dynamic shared object has to be done explicitly, it is heavily supported by the operating system and many programming languages. There exist functions to open a dynamic shared object, look up a symbol within a dynamic shared object, and close the file. For C, these functions (`dlopen`, `dlsym`, and `dlclose`) of the `dl` library are provided by the include file `dlfcn.h`. So programmers do not have to deal themselves with relocations or sharing of dynamic shared objects.

[75]Position independent object files are object files that are prepared to be relocated in memory. Therefore, they can be executed at an arbitrary address in memory. This possibility allows to share the code of a dynamic shared object among the applications that use the dynamic shared object.

position independent object files are linked to dynamic shared objects by passing them to the GNU linker LD[76] along with the option `-shared`.

Aldor's generated object files are not position independent. Nevertheless, on the `x86` architecture, Aldor's object files can be linked directly to a dynamic shared object as described above. The resulting dynamic shared object, however, contains several parts that are not in a position independent. As as result, the dynamic linker has to copy and relocate the dynamic shared object for every process using it. These additional actions cost time and demolish the benefit of sharing memory[77]. Processes cannot reuse an instance of such a dynamic shared object. Therefore, even on `x86` architectures, it is strongly suggested to use position independent object files when building shared libraries.

Compilers for most computer languages offer command line parameters, which allow to generate position independent object files. The Aldor compiler does not provide such a parameter. Nevertheless, the Aldor compiler can be used to generate position independent object files, although this involves closer investigation of how Aldor compiles code.

When the Aldor compiler builds object files, first `.c` files are generated. These `.c` files are automatically fed to a C compiler, which is instructed to produce the desired object files. Aldor allows to pass additional parameters to the invoked C compiler. Per default the Aldor compiler uses UNICL for compiling `.c` files to object files. UNICL is part of the Aldor distribution and acts as wrapper for GCC[78]. When compiling Aldor code to object files, the generated C code is passed to UNICL, which in turn passes the C code to GCC. Since Aldor cannot be instructed to produce position independent object files directly, either UNICL or GCC are in charge.

According to its help pages, UNICL generates code for shared libraries, when passed the command line parameter `-Wshared`. Passing this option, however, does not have any effect on `GNU/Linux` on `x86`. The resulting object files are exactly the same if or if not `-Wshared` is specified. UNICL does effectively not allow to generate position independent code.

Therefore, the relevant parameters have to be passed to GCC directly. These parameters are either `-fpic` or `-fPIC`. Both options produce position independent code. `-fpic` is generally assumed to be faster, but imposes limits on the sizes of internal data structures[79]. `-fPIC` does not have these limits and can therefore be used, if the limits of `-fpic` are exceeded during compilation.

To make UNICL pass `-fPIC` to GCC, UNICL needs to be passed `-Wopts=-fPIC`. For the Aldor compiler to pass `-Wopts=-fPIC` to UNICL, `-C args=-Wopts=-fPIC` needs to be

[76] LD is the standard linker on `GNU/Linux` platforms and is part of the GNU Binutils package[12].

[77] Nevertheless, these additional tasks of copying and relocating are done by the operating system and are completely hidden from the application.

[78] GCC is the C compiler of the GNU compiler collection[10]. GCC is the standard C compiler on `GNU/Linux` systems.

[79] According to GCC's documentation, the global offset table, which is used for looking up functions in position independent code, has a size limit of 8k on `SPARC` and 32k on `m68k` and `RS/6000`. On `x86` architectures, there is no such limit.

passed to the Aldor compiler.

Generating position independent object files in the presented way and linking them to dynamic shared objects as described before, finally allows to benefit from memory reuse and sharing code.

The presented way to generate position independent object file involves passing arguments to UNICL, which are then passed to GCC. This cascading passing of parameters is considered inelegant. Nevertheless, the presented approach is the recommended approach of this report.

Alternatively, UNICL can be replaced with a different compiler that provides working parameters to generate position independent object files, as for example GCC.

Another alternative is, to adjust the Aldor configuration file `aldor.conf`. This file holds the arguments that UNICL passes to the C compiler for a variety of operating systems. Adding a trailing

```
$?shared -fPIC : ;
```

to the `cc-opts` line in the `linuxcore` section of `aldor.conf` fixes the broken `-Wshared` support of UNICL. This approach changes a configuration file of the Aldor compiler. Therefore, this approach is not suggested.

Either of these presented possibilities allows to build a dynamic shared object. With such a dynamic shared object, finally, the code is in a form that can be passed to MAPLE's `define_external` function. Therefore, further details of `define_external` can be discussed.

MAPLE's `define_external` function takes several parameters. The first parameter is the name of the external function[80]. When importing code from a dynamic shared object, this name is an exported symbol of the dynamic shared object of interest. For the exported function of Section 7, this name is `_charset_coherentAutoreducedSet`. The next parameters to `define_external` are the parameter specifications for the external function. The order of the parameter specifications have to reflect the order of the external function's parameters within the dynamic shared object. Each parameter specification is of the form name followed by double colon followed by the type of the parameter. The name in a parameter specification is arbitrary and need not reflect the name of the parameter in the external function's definition. The type in a parameter specification has to be a MAPLE type that reflects the type of the parameter in the external function[81]. If the external function returns a value, its specification is the next

[80] By "external function", the function that will be loaded and used by MAPLE is meant. As from MAPLE's point of view, the function is not an internal function, so it is an external function.

[81] In [47], a mapping between native and some advanced types of C, Fortran, Java and MAPLE's types can be found. When parameters of the external function use types that cannot be mapped to MAPLE types directly, wrappers have to be created. This creation can either be done automatically or by hand. For this report, all used types have equivalent MAPLE counterparts. So, no further wrappers are needed. Therefore, generating wrappers is not discussed in this report. How to generate wrappers can be found for example in [47].

parameter to `define_external`. The return value of an external function is specified via `RETURN` followed by a double colon followed by the return type. Again, this type has to be a MAPLE type reflecting the return type of the external function. The last parameter to `define_external`[82] is the dynamic shared object in which the external function resides. This parameter has to be specified as `LIB` followed by an equals sign followed by the filename of the dynamic shared object.

The result of a call to `define_external` yields either an error or gives a function. This function is a native MAPLE function and its parameters and return type are in accordance with the parameters of the external function. Calling the new, native MAPLE function, converts the input parameters and passes them to the external function.

Importing the exported function of Section 7, can be achieved by the following MAPLE statement, when `/some/path/dynamicsharedobject.so` is replaced with the filename of the dynamic shared object.

```
coherentAS:=define_external(
         '_charset_coherentAutoreducedSet',
         language   :: string,
         polys      :: string,
         vars       :: string,
         deps       :: string,
         domOptions :: string,
         RETURN::string,
         LIB="/some/path/dynamicsharedobject.so"
):
```

Having imported, the external function into MAPLE, it can be used for example as the following expression shows.

```
coherentAS("maple","[y1(s), diff(y2(s),s)]", "[y1, y2]", "[s]", "");
```

8.2 Further integration of Aldor's code into MAPLE

In Section 8.1, Aldor's `_charset_coherentAutoreducedSet` algorithm has been connected to MAPLE. From within MAPLE this involves importing the function by a lengthy statement, and passing parameters as character strings to the algorithm. This section improves the integration of `_charset_coherentAutoreducedSet` in MAPLE by embedding the algorithm into a MAPLE repository and allowing to pass mathematical expressions instead of character strings to the algorithm.

The improvement of the integration is organized in three stages. The first stage provides a wrapper that allows to call the algorithm with mathematical expressions instead of character strings. The second stage bundles the algorithm and its wrapper. The third

[82] `define_external` has further options. These options are however not of importance for this report and therefore not covered. These further options are discussed in detail in [47].

stage shows how the whole code along with documentation is put in a MAPLE repository, which can be loaded and used conveniently.

`_charset_coherentAutoreducedSet` expects its parameters to be character strings in C format. These strings typically represent mathematical expressions. However, for a seamless integration of `_charset_coherentAutoreducedSet` into MAPLE, it is required that mathematical expression themselves can be passed as arguments directly.

Therefore, a new function is written that takes mathematical expressions as input. This input is then automatically converted to strings and passed to the imported Aldor algorithm.

The conversion from mathematical expressions to character strings is achieved by calling MAPLE's `convert` function. This function takes two parameters, the expression to convert and the type to convert to.

```
SomeStr := convert( SomeExpr, string )
```

converts the expression `SomeExpr` to a character string in C format and stores the result in `SomeStr`.

The function `coherentAutoreducedSet` on page 80 shows, how to use `convert` in a wrapper for exported code.

The result of calling `_charset_coherentAutoreducedSet` is a character string. Again, this string represents a mathematical expression. For good integration of the imported piece of code into MAPLE, it is inevitable to automatically convert the returned character string to a mathematical expression. This conversion can also be done in the wrapper that converts the input parameters.

In this report, MAPLE's `parse` function is used converting character strings to mathematical expressions[83]. `parse` parses a string and returns the unevaluated expression. If the additional option `statement` is specified, the string is not only parsed, but also evaluated. As MAPLE's character strings are in C format, this function can be used to convert the result of `_charset_coherentAutoreducedSet` to a mathematical expression.

```
SomeExpr := parse( SomeStr, statement )
```

parses the string `SomeStr`, evaluates the resulting mathematical expression, and stores the result of the evaluation in `SomeExpr`.

Finally, with the help of `convert` and `parse` it is possible to write a wrapper for `_charset_coherentAutoreducedSet` that accepts mathematical expressions as parameters

[83]Note, that `convert` cannot be reused. The previous conversion of the input parameters turned mathematical expressions to character strings. This conversion needs to do the opposite, namely to convert character strings to mathematical expressions. MAPLE's `convert` is not capable of this. Therefore, `parse` is used.

and also returns a mathematical expression. The function `coherentAutoreducedSet` on page 80 gives an example of a wrapper function that converts both the input parameters and the result from and to appropriate formats.

The wrapper function needs to call the imported function, it is wrapping. This calling can be done in two different ways. Either the wrapper imports the wrapped function at every call or there is some variable that stores the imported function.

In the first case, the main problem is that the wrapped function has to be imported for every call of the wrapping function. This loading costs resources. The advantage of this approach is that the imported function is not stored in a publicly visible variable. Therefore, this variable cannot be altered by either the user or other functions. As a result, this method is less error-prone than the second one.

In the second case, it suffices to import the wrapped function once. However, there is the problem of storing the imported function in a variable, as this variable can possibly be changed either accidentally or on purpose.

The advantages of both approaches can be combined by using MAPLE's `module`s. A `module` is an aggregation of functions and constants. `module`s permit information hiding and exporting of functions and constants among other features[84].

Describing MAPLE's `module`s in all details is beyond the scope of this report and can be found for example in can be found in [47]. Therefore, only an example for the use of `module` is given that covers the necessary details and constructs for this report. These details are described afterwards.

```
CharSet := module()
    description "interface for Aldor's characteristic set library";
    option package,load=importAlgo;
    export coherentAutoreducedSet;
    local algoAldor, importAlgo;

    importAlgo := proc()
      algoAldor := define_external(
       '_charset_coherentAutoreducedSet',
        languageAldorParam   :: string,
        polysAldorParam      :: string,
        varsAldorParam       :: string,
        depsAldorParam       :: string,
        domOptionsAldorParam :: string,
        RETURN::string,
        LIB="/some/path/dynamicsharedobject.so"
       ):
    end proc;

    coherentAutoreducedSet := proc(
       polys   :: list,
       vars    :: list,
       deps    :: list
     )
```

[84] An in-depth description of `module` can be found in [47].

```
      local result, domOptions;
      if nargs > 3 then
        if type(args[4],string) then
          domOptions:=args[4];
        else
          ERROR(`the optional parameter for the options has to be of type string.`);
        fi
      fi;
      result:=algoAldor(
        "maple",
        convert( polys, string ),
        convert( vars , string ),
        convert( deps , string ),
        domOptions
      );
      parse( result, statement );
    end proc;
end module:
```

The first line of the above piece of code defines a `module` and stores it as `CharSet`. In the `description` line, a short description for the `module` is provided. Two options are specified for the module, namely `package` and `load`. `package` is used to generate a MAPLE package, which automatically protects exported functions from modification. `load` allows to specify a function that is evaluated, as the module is loaded. In the presented example, the function `importAlgo` is called, when the `module CharSet` is loaded. `importAlgo` imports Aldor's `_charset_coherentAutoreducedSet`. The `export` line denotes those identifiers that are visible from outside of the module. In this example, only the function `coherentAutoreducedSet` is visible. `coherentAutoreducedSet` is the wrapper of Aldor's exported function that allows to be called with mathematical expressions and returns a mathematical expression. In the `local` line, those identifiers are declared that are visible from inside the module but not from outside of the module. In this example there are two local functions, `algoAldor` and `importAlgo`. `algoAldor` holds Aldor's imported function and `importAlgo` effectively imports Aldor's function and assigns it to `algoAldor`. The rest of the piece of code defines the functions `algoAldor` and `coherentAutoreducedSet` as described earlier.

For calling the `coherentAutoreducedSet` function of `CharSet`, either

```
CharSet:-coherentAutoreducedSet( ... )
```

or

```
with(CharSet);
coherentAutoreducedSet( ... )
```

has to be called with appropriate parameters.

With the help of this `module` Aldor's `_charset_coherentAutoreducedSet` algorithm is imported only once, when the module is loaded. Furthermore, the imported algorithm is not visible to user's of the `module` and can therefore not be modified.

Although the basic problems of integrating Aldor's exported code into Maple are handled by the previous piece of code, it is still not covered how this code is best input in Maple. Maple provides a concept of external repositories. Each Maple repository stores Maple expressions and a name for these Maple expressions permanently. By their name, the Maple expressions can be retrieved later. Each Maple repository is stored in two files, can easily be brought into scope, and hides the implementation from the users. Therefore, Maple repositories are typically used for extending Maple. This report uses such a repository to store the presented Maple code for importing and integrating Aldor's `_charset_coherentAutoreducedSet` into Maple. The next paragraphs show how to build a Maple repository for the `CharSet module` and how to use it.

A Maple repository for the `CharSet module` is built in two steps. First an empty repository is created. Afterwards expressions are added permanently to the repository. Typical expressions to store in a repository are `module`s, such as `CharSet`.

For creating a new Maple repository,

```
march('create', archive, size );
```

is called, where `archive` is the either the name of the new repository or a directory. The name of the repository is relative to the current directory. If only a directory is given, then a repository with the name "maple" is created in the specified directory. `size` is the approximate numbers of functions in the repository[85]. Every Maple repository must reside in an own directory. Therefore, creation of a repository in a directory, where another repository already exists, will fail. In order to avoid unnecessary confusion, it is suggested to create repositories only in existing, empty directories.

After generating a Maple repository, expressions are saved to the repository, with the help of Maple's `savelib` function[86]. This function takes names of expressions as arguments. Maple stores these names along with their associated expressions in the first repository specified by `savelibname`. If this attempt fails, the second repository specified in `savelibname` is used, and so on. If `savelibname` is not assigned, `libname` is used instead.

After the module `CharSet` is defined, the following piece of code creates the Maple repository in the directory `lib` and adds the `CharSet module` to it.

```
newlibdir:="lib";
march('create',newlibdir,5);
savelibname:=newlibdir;
savelib('CharSet');
```

[85]Note that each member of a `module` is counted, in addition to the `module` as a whole. So if you plan to create a repository which contains four modules and each module contains five functions, then you should specify 24, not four. More information, on counting functions can be found in the documentation of Maple's `march`.

[86]Formerly, Maple repositories have been called libraries. This prior naming is the reason that some functions and variables that deal with repositories contain a `lib` in their name. Some examples are `libname`, `savelibname`, and `savelib`. However, the pursued name is repository not library.

To provide users additionally with proper documentation within MAPLE's help system, help pages have to be added to the repository. For this purpose, MAPLE provides the `makehelp` function. This function takes three parameters. The first parameter is the topic under which a help page is registered. The second parameter is the filename of the prepared help page. A help page can either be a text file or a MAPLE worksheet. These files do not have to obey any special syntax rules and are displayed when looking up the topic, which has been specified as first parameter. The third parameter to `makehelp` is the repository to store the documentation in. The following piece of code embeds the file `/some/path/charset.mws` as help page for the function `coherentAutoreducedSet` of the package `CharSet` in the MAPLE repository that can be found in the `/another/path` directory.

```
makehelp(‘CharSet/coherentAutoreducedSet‘,
         ‘/some/path/charset.mws‘,
         ‘/another/path‘
    )
```

By the created repository and the integrated wrapper for `_charset_coherentAutoreducedSet`, Aldor's code is seamlessly integrated into MAPLE and can be used conveniently. Typical invocation of the algorithm within involves three stages.

The first stage is to inform MAPLE of the repository. Therefore, the directory of the repository has to be added to the `libname` variable.

```
libname := "/path/to/repository", libname;
```

This setting of `libname` suffices to load the repository. In the second stage, the `CharSet module` is brought into scope.

```
with(CharSet);
```

Finally, the `coherentAutoreducedSet` function can be called.

```
coherentAutoreducedSet( [y1(s),y2(s)*y1(s)], [y1,y2], [s] );
```

The inner workings of these three stages are illustrated in Figure 8.1.

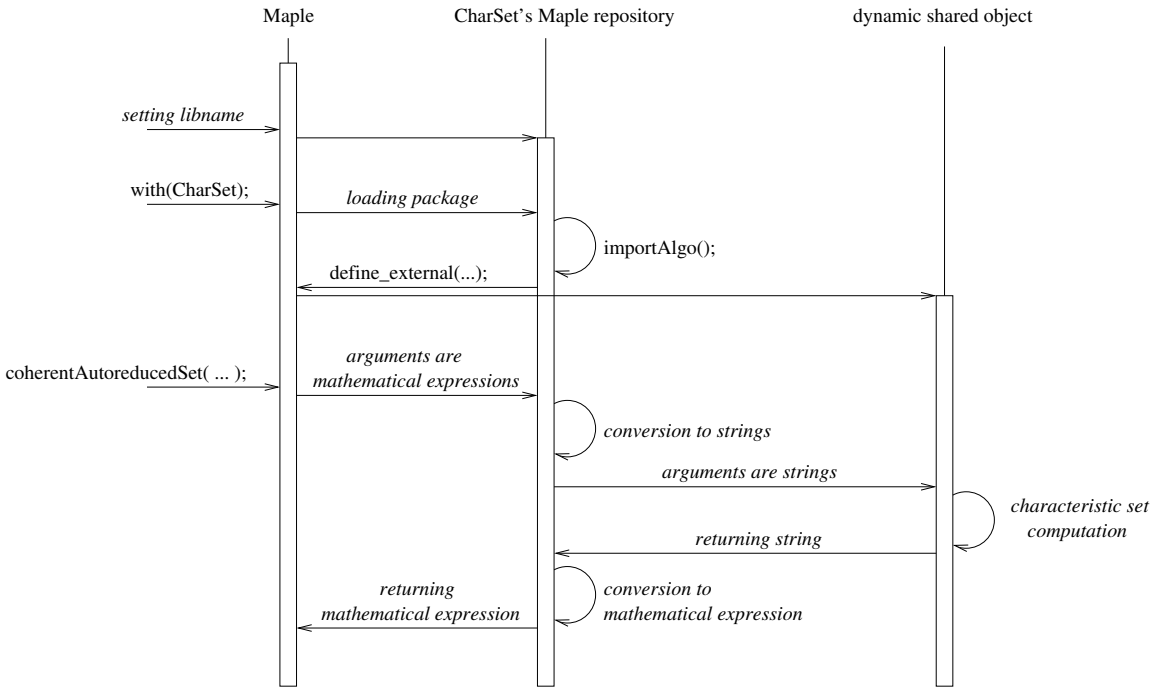

Figure 8.1: The inner workings of loading and using `_charset_coherentAutoreduced-Set` from within MAPLE

9 Connecting to the characteristic set library from within MATHEMATICA

This section illustrates how the exported code of Section 7 can be connected to MATHEMATICA.

The first part of this section analyzes MATHEMATICA and MATHLINK, the framework to connect code to MATHEMATICA. The second part discusses how MATHLINK enabled executables are generated. This second part heavily relies on template files which are described in depth in the last part of this section.

9.1 A closer look at MATHEMATICA

Before connecting `_charset_coherentAutoreducedSet` to MATHEMATICA, it is important to investigate MATHEMATICA closer and identify its sub-parts. Afterwards a sub-part to connect `_charset_coherentAutoreducedSet` to is identified and the used technology is presented.

The computer algebra system MATHEMATICA [21] provides two main components, which are

- the notebook interface and
- the kernel.

While the notebook interface is typically used to input expressions and receive output, all computations are performed by the kernel. This splitting up is not artificial, but can be seen already in the file system. The executable `math` represents MATHEMATICA's kernel, `mathematica` represents MATHEMATICA's notebook interface. These two programs are independent from each other. The notebook interface can be used without the kernel and vice versa. However, the notebook interface has to be connected to a kernel for performing computations[87]. As the computational engine of MATHEMATICA is its kernel, `_charset_coherentAutoreducedSet` has to be connected to the kernel. MATHEMATICA provides an interface to its kernel via MATHLINK. MATHLINK [19] is MATHEMATICA's standard to pass data between applications. For example, the connection between the notebook interface and a kernel is made through MATHLINK.

The rest of Section 9 discusses how `_charset_coherentAutoreducedSet` is connected to MATHEMATICA's kernel via MATHLINK.

It is important to see that MATHLINK is not a further program, but simply a name for the specification of how to represent and transmit data. MATHEMATICA's kernel does not allow to import libraries. It can only import code via MATHLINK.

[87] On evaluating the first cell within a notebook, the notebook interface typically starts a kernel locally. The user is given no feedback about this, except that evaluating the first cell takes considerably longer than subsequent evaluation of cells. This delay is due to starting a local kernel. It is not necessary to start a local kernel. The notebook interface also allows to use kernels running on other computers.

MathLink is Mathematica's proposed way to connect foreign code to the Mathematica kernel. Foreign code has to implement the MathLink functionality to be able to connect to the kernel via MathLink. For common languages like C, or Java software development kits are provided. These kits can be used to built MathLink enabled programs.

As `_charset_coherentAutoreducedSet` follows C calling conventions, the C software development kit is used for this report's work. Although this kit is shipped with Mathematica, it can be obtained independently from Mathematica at [20].

For this report's work, MathLink's C software development kit that is shipped with Mathematica 5.1 is used. This version's documentation of the contained mcc script does not match the mcc script itself. This fact also holds for the other versions of the software development kit that are available to the author. These are those of Mathematica 3.0.2, Mathematica 4.0, Mathematica 4.1, Mathematica 4.2, Mathematica 5.0, and the kit from [20] on 15^{th} October 2005. However, the version from Mathematica 5.1 is chosen.

The main parts of the C software development kit for Mathematica are

- mcc,
- mprep,
- `mathlink.h`, and
- `libML.a`.

mcc is a script automating compilation of MathLink template files to MathLink enabled executables. MathLink template files are treated separately in Section 9.3.

mprep is a preprocessor converting MathLink templates to C files.

`mathlink.h` is a C header source file. This file is needed to compile the C files generated by mprep.

`libML.a` is a library implementing the definitions of `mathlink.h`. This file is needed to link object files to executables.

For this report, mcc is investigated further, as it hides the use of the other three parts from the user. With the help of mcc, a MathLink enabled executable is built that can easily be connected to the Mathematica kernel.

9.2 MathLink's mcc compiler script and MathLink enabled executables

This section focuses on mcc and its use to compile Aldor's exported C code to MathLink enabled executables. Additionally, at the end of this section it is shown how to connect MathLink enabled executables to Mathematica.

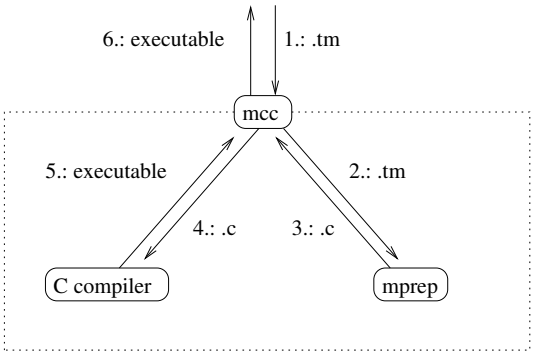

Figure 9.1: The internals of MCC

MCC is a wrapper for the C compiler and converts MATHLINK template files to MATHLINK enabled executable files. MATHLINK template files specify how and which C functions are to be made accessible through MATHLINK. These template files are discussed in detail in Section 9.3.

Internally, the MCC script operates in two stages. The first stage passes the MATHLINK template files to MPREP. MPREP is a preprocessor that converts MATHLINK templates to a single C file. This C file depends on `mathlink.h`. In the second stage of MCC, the generated C file from MPREP is compiled and linked against the library `libML.a`. The work-flow for MCC is depicted in Figure 9.1.

In the further discussion, command line options to the MCC script are presented. These command line options refer to the script itself. These options' implementation sometimes does not match their documentation. Other options are completely undocumented. However, as mentioned above, the documentation of MCC that is shipped with the used software development kit is not accurate. Therefore, the documentation is ignored and the script itself is analyzed. Furthermore, MCC relies on a C compiler. As MCC passes several options directly to the C compiler, it is important to use a compatible C compiler. It is assumed, that GCC is used as C compiler for MCC[88]. Additionally, let `MLINCDIR` denote the directory of `mathlink.h` and `MLLIBDIR` the directory of `libML.a`. Typically both directories refer to the `AddOns/MathLink/DeveloperKit/Linux/CompilerAdditions` sub-directory of MATHEMATICA.

When adapting C code for the use with MATHLINK, it usually suffices to incorporate the implemented C code into a template file. Assuming this template file is called `sometemplate.tm`, a call to

```
mcc sometemplate.tm
```

compiles it to a MATHLINK executable. This executable is called `a.out`. MCC's command line option `-o` can be used to override the output file.

```
mcc -o someexecutable sometemplate.tm
```

[88] MCC tries to use the C compiler specified by the environment variable `$CC`. If `$CC` is empty, MCC falls back to using `cc`. On most `GNU/Linux` systems, `$CC` is empty and `cc` refers to GCC. Therefore, on most `GNU/Linux` systems, MCC defaults to using GCC.

compiles `sometemplate.tm` to the executable `someexecutable`. Such a call however does not work for the `charset.tm` MathLink template file of Section 9.3. That MathLink template has to be linked with the implementation of `_charset_coherentAutoreducedSet`. This implementation can be found in the `CharSet` library. Therefore, mcc has to instruct the underlying compiler to link against the `CharSet` library[89]. The necessary option for this linkage is `-lcharset`. However, when calling mcc by

```
mcc -o charset charset.tm -lcharset
```

mcc invokes the C compiler with

```
gcc -o charset -I MLINCDIR -lcharset charset.tm.c \
  -L MLLIBDIR -lML -lm
```

while the correct call would be

```
gcc -o charset -I MLINCDIR charset.tm.c -lcharset \
  -L MLLIBDIR -lML -lm
```

It is however crucial that `charset.tm.c` comes *before* `-lcharset`. Otherwise, the compiled `charset.tm.c` is not linked against the `CharSet` library[90].

The possible workarounds are

- to call mprep and gcc manually,
- to use mcc only to compile the file and link the file manually later on, or

[89] When compiling to an executable, it does not suffice, to link against `CharSet`, as `CharSet` again has unresolved symbols. These symbols are resolved in `ExtIO`, `Algebra`, `Aldor`. These libraries again have unresolved symbols. In the end, the MathLink file has to be linked against the libraries `CharSet`, `ExtIO`, `Algebra`, `Aldor`, `FOAM`, and `m`. Specifying all these libraries for every upcoming command line renders the given commands unreadable. Therefore, only the first of these libraries is given, as the `CharSet` library suffices to illustrate the problems.

[90] This behaviour of gcc is not a bug. gcc resolves the symbols of object files from left to right. Therefore, only the object files of `libML.a` and `libm.a` are used to resolve the symbols of `charset.tm.c`'s object file when using mcc. ld, which is used by gcc for linking, can be set up to use groups of libraries. This grouping of libraries allows to pass `charset.tm.c` after `-lcharset` and still have all symbols resolved. For example
`gcc -o charset -I MLINCDIR -Xlinker --start-group charset.tm.c -lcharset \`
`-Xlinker --end-group -L MLLIBDIR -lML -lm`
groups `CharSet` and the object file of `charset.tm.c`. Although this call to gcc works and the call states `-lcharset` after `charset.tm.c`, it cannot be used to solve the mcc issues. With this solution `-lcharset` can be specified after `charset.tm.c`. However, `-Xlinker --start-group` has to occur *before* `charset.tm.c`. Therefore, an equivalent problems arise, as
`mcc -o charset -Xlinker --start-group charset.tm -lcharset -Xlinker \`
`--end-group`
resolves to
`gcc -o charset -I MLINCDIR -Xlinker --start-group -lcharset \`
`-Xlinker --end-group charset.tm.c -L MLLIBDIR -lML -lm`

- to extract and modify the call to the C compiler.

Calling MPREP and GCC manually overcomes the need for MCC. The commands

```
mprep -o charset.tm.c charset.tm
gcc -o charset -I MLINCDIR charset.tm.c -lcharset \
  -L MLLIBDIR -lML -lm
```

compile the `charset.tm` template to the C file `charset.tm.c`, which is in turn compiled to the executable `charset`. This solution is the most flexible of the three workarounds. Every command can be called with exactly those options that are required. The major disadvantage of this workaround is that it does not use MCC at all. However, MCC is the MATHLINK's proposed way to compile MATHLINK enabled executables. Therefore, future versions of MATHLINK will focus on keeping the use of MCC consistent. The use of MPREP is subject to changes. As a result, it can be expected that the presented use of MPREP is obsolete in further versions of the MATHLINK C software development kit. Furthermore, the corresponding `MLINCDIR` and `MLLIBDIR` have to be determined manually. When using MCC to build the executable, MCC locates the proper directories on its own.

The second workaround is to use MCC only to compile the file and link manually. The first line of the command sequence

```
mcc -c charset.tm
gcc -o charset charset.tm.o -lcharset -L MLLIBDIR -lML -lm
```

uses MCC to compile the template `charset.tm` to the object file `charset.tm.o`. The command line option `-c` tells MCC to use MPREP to generate the C file `charset.tm.c` and finally use the C compiler to compile the C file to the object file `charset.tm.o`. In the second line, this object file is linked to an executable by GCC. It is also possible to use LD, the GNU Linker, for linking the object file. However, using LD requires to pass further options and link against further libraries. When using GCC, these issues are hidden from the user.

The presented workaround does no longer invoke MPREP explicitly and does not require to pass the directory `MLINCDIR` to any command. However, `MLLIBDIR` is still required. Furthermore, future versions of MATHLINK may come with additional or other libraries and may require to link against some of these libraries. As a result, the call to gcc has to be adjusted. Therefore, the presented workaround is again likely to fail for future versions of MATHLINK's C software development kit.

The last workaround is to extract and modify the call to the C compiler. The commands

```
export OLDCC=$CC
if [ -z "$OLDCC" ] ; then export OLDCC=cc ; fi
export CC=echo
$OLDCC ‘mcc -g -o charset charset.tm‘ -lcharset
export CC=$OLDCC
```

can be used in any BASH-like shell. The first three lines are used to store the system's C compiler in the environment variable `$OLDCC` and set the environment variable `$CC` to the `echo` command. As MCC uses `$CC` as C compiler, MCC's calls to the C compiler (see line 4) are echoed. The command ‘`mcc -g -o charset charset.tm`‘ evaluates to

```
-o charset.mathlink -I MLINCDIR -g charset.tm.c -L MLLIBDIR -lML -lm
```

which are the arguments to the C compiler to compile to an executable. The forth line of this workaround prepends the C compilers executable and appends the `-lcharset` to link against the `CharSet` library. Thereby, the C compiler is invoked with the proper options and the appended `-lcharset`. The last line of the workaround resets the environment variable `$CC`.

This workaround passes the command line parameter `-g` to MCC. After compilation, MCC typically removes the generated C and object files. For this workaround, it is essential that MCC does *not* remove the generated C file, as this file is needed for the C compiler. Therefore, MCC is called with the parameter `-g`. This parameter tells MCC to not remove the generated C file. Additionally, MCC passes the `-g` to the C compiler. C compilers generate debugging information when passed the `-g` option.

This workaround is the most complex of the three presented alternatives. However, it does not require to specify `MLLIBDIR` or `MLINCDIR`. Therefore, it is the only workaround that works without further knowledge of the used system. Additionally, it is the only workaround that can be expected to work in future versions of MathLink, as it does not call MPREP directly and makes no assumption on which libraries to use. Therefore, the third workaround is proposed among the three presented ones.

Each of the three presented workarounds can be used to generate a MathLink enabled executable `charset` the MathLink template of Section 9.3.2. To use this executable from within Mathematica it suffices to call

```
Install["/path/to/charset"]
```

in either the notebook interface or the kernel itself, where `/path/to` denotes the directory of `charset`. `Install` starts the `charset` executable and connects the kernel to it. Afterwards the function `CoherentAutoreducedSet` of `charset` can be used. There is no need to explicitly bring the package `CharacteristicSet` into scope.

9.3 MathLink template files

Section 9.2 used MathLink template files without specifying what MathLink template files are. The current section gives a detailed discussion of these template files.

The first part of this section gives the syntax for a MathLink template file and provides a basic example for `_charset_coherentAutoreducedSet`. In the second part of this section, this basic example is refined and better integrated into Mathematica.

9.3.1 A basic MathLink template file

MathLink template files are used to describe C functions to make available through MathLink. These template files may contain C source code and MathLink commands. Lines not beginning with a colon are treated as C code. Lines beginning with a colon are treated as MathLink commands. Such lines are of the form colon, command name, colon optionally followed by arguments to the command. MathLink provides eight different commands.

- `Begin` is used to start the definition of a function that is to be made accessible through MathLink. This command takes no options.
- `Function` takes one parameter, which is the name of the C function to make accessible.
- `Pattern` is used to define the function's MathLink name. The MathLink name and the C name of a function need not be identical. `Pattern` takes one argument, which is the Mathematica calling pattern for the function.
- `Arguments` takes one argument which is the list of arguments to pass to the C function. This list may contain arbitrary Mathematica expressions, for example `If` constructs. Additionally, these expressions may contain the parameters of the `Pattern` command. The expressions of `Arguments` are evaluated for every function call of the MathLink function.
- `ArgumentTypes` denotes the corresponding types for the arguments specified by `Arguments`.
- `ReturnType`'s argument specifies the return type of the C function.
- `End` is used to end the definition of a function that is to be made accessible through MathLink.
- `Evaluate` allows to execute arbitrary Mathematica commands. However, the command `Evaluate` is discussed in more detail in Section 9.3.2, as it is not necessary for making C functions accessible via MathLink.

With the first seven commands the mapping between the C function and the MathLink function can be described completely.

- `Begin` and `End` are used for scoping.
- `Function`, `ArgumentTypes`, and `ReturnType` describe the C function.
- `Pattern` and `Arguments` describe the MathLink function.

An example template specification for `_charset_coherentAutoreducedSet` is given by the following piece of code. The argument of `Pattern` is too long to fit on the page. Therefore, it is broken up into two lines. Breaking lines with MathLink commands is not allowed in MathLink template files and renders the given code an invalid MathLink template file. Nevertheless, the line is kept broken into two lines and has to be interpreted as if these two line were actually one line. For Section 9.3.1 and Section 9.3.2, `--` at the end of a line marks a line break that must not occur in the MathLink template file, but is necessary for formatting the code in this report.

```
:Begin:
:Function:      _charset_coherentAutoreducedSet
:Pattern:       CoherentAutoreducedSetAldor[ langStr_String, polyStr_String,      --
  varStr_String, depStr_String, optStr_String ]
:Arguments:     { langStr, polyStr, varStr, depStr, optStr }
:ArgumentTypes: { String, String, String, String, String }
:ReturnType:    String
:End:
```

In addition to the MathLink definition, a MathLink template has to contain an entrance point for MathLink and an implementation of the C function that is referenced by MathLink's `Function` command. The entry point for Mathematica is set up by calling the MathLink's C function `MLMain`. This call can be set up with the following piece of code.

```
int main(int argc, char* argv[])
{
  return MLMain(argc, argv);
}
```

As mentioned before, the MathLink template also has to provide the C function which is made accessible via MathLink. For this report's work, the implementation of `_charset_coherentAutoreducedSet` is hidden within the `CharSet` library and cannot be given in the template file. However, C allows to defer the implementation to a library by the keyword `extern`. The following C code denotes that the implementation of the function `_charset_coherentAutoreducedSet` is not contained in this file, but can expected to be found in a library later on.

```
extern char * _charset_coherentAutoreducedSet( kcharp_ct langStr, kcharp_ct polyStr,
  kcharp_ct varStr, kcharp_ct depStr, kcharp_ct optStr);
```

In Aldor (see Section 7.2), `_charset_coherentAutoreducedSet` has been exported as function taking five pointers and returning a pointer. Although each of the pointers is expected to point to a string, this cannot be seen from `_charset_coherentAutoreducedSet`'s definition in Aldor. This knowledge is implicit. However, the given piece of code denotes that `_charset_coherentAutoreducedSet` takes five arguments of `kcharp_ct` and returns a `char *`. Both, `char *` and `kcharp_ct` denote pointers to

strings in C and are discussed later. Although C allows to model arbitrary pointers[91] which correspond to Aldor's Pointer, the given piece of code uses pointers to strings. These types are consequences of the MathLink template on page 92. There, `String`s are used as types for `ArgumentTypes` and `ReturnType`. In this template, the implicit knowledge that the pointers actually point to strings is made explicit. However, it is not possible to have arbitrary pointers in `ArgumentTypes` or `ReturnType`, as Mathematica does not provide the concept of arbitrary pointers. Therefore, it is crucial for MathLink to know what kind of pointers a function is dealing with.

The MathLink template file uses the types `char *` and `kcharp_ct` for pointers to strings. In C, `char` is used as type for a character and `*` denotes a pointer to a type. Therefore, `char *` is actually a pointer to a character. However, in C strings are typically represented by a pointer to the first character of a string. All subsequent characters and a trailing zero byte are stored in the consecutive bytes. Therefore, there is no syntactical difference between a pointer to a character and a pointer to a string in C. `kcharp_ct` is an alias of MathLink to C's `const char *`[92]. `const char *` is a pointer to a string, whose value cannot be changed[93].

The complete MathLink template file is given by the following piece of code.

```
:Begin:
:Function:      _charset_coherentAutoreducedSet
:Pattern:       CoherentAutoreducedSetAldor[ langStr_String, polyStr_String,      --
  varStr_String, depStr_String, optStr_String ]
:Arguments:     { langStr, polyStr, varStr, depStr, optStr }
:ArgumentTypes: { String, String, String, String, String }
:ReturnType:    String
:End:

extern char * _charset_coherentAutoreducedSet( kcharp_ct langStr, kcharp_ct polyStr,
  kcharp_ct varStr, kcharp_ct depStr, kcharp_ct optStr);

int main(int argc, char* argv[])
{
  return MLMain(argc, argv);
}
```

[91] In C, pointers to arbitrary entities are modelled by `void *`.

[92] This statement holds for operating systems like `GNU/Linux` on `x86`. However, the exact expansion of `kcharp_ct` can be found in `mathlink.h` and is `MLCONST char FAR *`. The definition of `MLCONST` is far too complicated to be reproduced in this report and is again found in `mathlink.h`. However, in typical applications on up-to-date operating systems and C compilers, it is save to assume `MLCONST` to evaluate to `const`. `FAR` evaluates to `far` on operating systems that use segmented pointers to memory, like `MS-DOS`. There, `far` has to be used for pointers to memory in other segments than the current one. On other operating systems like `GNU/Linux`, `FAR` is blank.

[93] `const char *` strings cannot be modified directly. However it is possible to discard the `const` qualifier (for example by casting to `char *`) and modify the string afterwards. Although possible, it is considered bad style to discard qualifiers. C compilers typically issue warnings, when qualifiers are removed from a type.

9.3.2 Advanced topics of MathLink templates

The template file of Section 9.3.1 allows `_charset_coherentAutoreducedSet` to be called directly from Mathematica. When calling `_charset_coherentAutoreducedSet` directly, the language parameter has to be specified for every call. Furthermore, it is necessary to convert the parameters to strings and the result to a Mathematica expression for every call. Although Mathematica allows to attach short help texts to functions, the MathLink template of Section 9.3.1 does not use this feature.

This section overcomes these issues by MathLink's `Evaluate` command. After discussing each of these issues separately, a complete MathLink file is given that incorporates the presented changes.

`Evaluate` allows to execute arbitrary Mathematica commands upon connection to the Mathematica kernel. The following piece of code defines a wrapper for the function `CoherentAutoreducedSetAldor`. This wrapper is called `CoherentAutoreducedSet` and takes care of the conversion between strings and Mathematica expressions. Additionally, the used language is automatically set to `mathematicafullform`.

```
:Evaluate:      CoherentAutoreducedSet[ poly_List, var_List, dep_List,           --
    opts_String:"" ] := ToExpression[ CoherentAutoreducedSetAldor[               --
    "mathematicafullform", ToString[ poly//FullForm ], ToString[ var//FullForm ], --
    ToString[ dep//FullForm ], ToString[ opts//FullForm ] ] ]
```

The function `ToExpression` converts a string to a Mathematica expression. `FullForm` formats an expression in full form syntax. However, the result of `FullForm` is not a string but again an expression. Therefore, Mathematica's `ToString` is used to convert the full form expressions to strings.

A help text is attached to a function by setting its `usage` message. The following code gives an example for `CoherentAutoreducedSet`.

```
:Evaluate:      CoherentAutoreducedSet::usage = "CoherentAutoreducedSet[         --
    { a, b, ...}, {y1,y2,...}, { s1, s2, ... }, Options ] derives a coherent     --
    autoreduced set of the polynomials a, b, ... in the dependent variables y1,  --
    y2,  ... and independent variables s1, s2, ... . Options is an optional      --
    string denoting options as described in the reference manual for the         --
    characteristic set library of Aldor."
```

Finally, the code is put into a package by the the following pattern.

```
:Evaluate:      BeginPackage["PackageName`"]

    public definitions

:Evaluate:      Begin["`Private`"]

    private definitions

:Evaluate:      End[ ]
:Evaluate:      EndPackage[ ]
```

In this pattern, `public definitions` is to be replaced with the definitions that should be publicly available and accessible after connection to a kernel. `private definitions` is to be replaced by definitions that are not intended for public use. Although this pattern is proposed in MathLink's documentation, Mathematica does allow to hide a function completely. The definitions in the `Private` part can be accessed, when completely qualifying their name. Although, the function `CoherentAutoreducedSetAldor` of the MathLink template file on page 96 is defined within a `Private` part,

```
‘CharacteristicSet‘Private‘CoherentAutoreducedSet
```

can be used to access the function.

The MathLink template on page 96 adds the improvements of this section to the basic template of Section 9.3.1.

In Section 9 only the MathLink template commands of the MathLink's C software development kit have been used. These commands are sufficient for connecting `_charset_coherentAutoreducedSet` code to Mathematica. However, the MathLink C software development kit also provides several C functions to be used in C code. There are, for example, functions to access Mathematica's internal data structures for expressions. These functions are documented in the manual of the C software development kit and allow to convert custom data types to Mathematica expressions and vice versa. Since `_charset_coherentAutoreducedSet` takes string parameters, these functions need not be used and conversion between Aldor and Mathematica expressions is left to Mathematica's `ToString`, `FullForm`, and `ToExpression`.

```
:Evaluate:      BeginPackage["CharacteristicSet`"]
:Evaluate:      CoherentAutoreducedSet::usage = "CoherentAutoreducedSet[          --
    { a, b, ...}, {y1,y2,...}, { s1, s2, ... }, Options ] derives a coherent      --
    autoreduced set of the polynomials a, b, ... in the dependent variables y1,   --
    y2,  ... and independent variables s1, s2, ... . Options is an optional       --
    string denoting options as described in the reference manual for the          --
    characteristic set library of Aldor."

:Evaluate:      Begin["`Private`"]

:Evaluate:      CoherentAutoreducedSetAldor::usage = "CoherentAutoreducedSetAldor --
    [ polys, depVars, indepVars, Options ] derives a coherent autoreduced set of  --
    the polynomials encoded in polys in the dependent variables encoded in        --
    depVars and independent variables encoded in indepVars. Options is a string   --
    denoting options as described in the reference manual for the characteristic  --
    set library of Aldor."

:Begin:
:Function:      _charset_coherentAutoreducedSet
:Pattern:       CoherentAutoreducedSetAldor[ langStr_String, polyStr_String,      --
    varStr_String, depStr_String, optStr_String ]
:Arguments:     { langStr, polyStr, varStr, depStr, optStr }
:ArgumentTypes: { String, String, String, String, String }
:ReturnType:    String
:End:

:Evaluate:      CoherentAutoreducedSet[ poly_List, var_List, dep_List,            --
    opts_String:"" ] := ToExpression[ CoherentAutoreducedSetAldor[                --
    "mathematicafullform", ToString[ poly//FullForm ], ToString[ var//FullForm ], --
    ToString[ dep//FullForm ], ToString[ opts//FullForm ] ] ]

:Evaluate:      End[ ]
:Evaluate:      EndPackage[ ]

extern char * _charset_coherentAutoreducedSet( kcharp_ct langStr, kcharp_ct polyStr,
  kcharp_ct varStr, kcharp_ct depStr, kcharp_ct optStr);

int main(int argc, char* argv[])
{
  return MLMain(argc, argv);
}
```

10 Connecting to the characteristic set library on the command line

Section 8 and Section 9 showed how the `_charset_coherentAutoreducedSet` function of Section 7 can be used from within computer algebra systems. This section presents the implementation of a command line utility that allows to use the exported function without Aldor and without other computer algebra systems.

This presentation is split into two parts. The first part presents the design decisions, while the second part focuses on the usage of the command line utility.

10.1 Designing the command line utility

The organization of this section sticks to the following enumeration of requirements and their order.

The requirements for the command line tool are

- to allow execution of the `_charset_coherentAutoreducedSet`,
- to require no further software (Aldor or computer algebra systems),
- to be simple,
- to use convenient syntax to enter and to represent data , and
- to be interactive.

The `_charset_coherentAutoreducedSet` function is exported as C function (see Section 7.2). Therefore, any environment that can import C code can call the function `_charset_coherentAutoreducedSet`. Among such environments, only those allowing to produce standalone programs are of interest.

Aldor allows to import C code and can produce standalone programs. As this report's work is already focused on Aldor, Aldor is again chosen to implement the command line utility in.

Aldor's keyword `import` is typically used to bring functions from Aldor domains into scope. However, `import` also allows to import code from other languages, as for example C or Fortran. The function `_charset_coherentAutoreducedSet` from the `CharSet` library is imported by

```
import {
  __charset__coherentAutoreducedSet: ( Pointer, Pointer, Pointer,
      Pointer, Pointer ) -> Pointer ;
} from Foreign C;
```

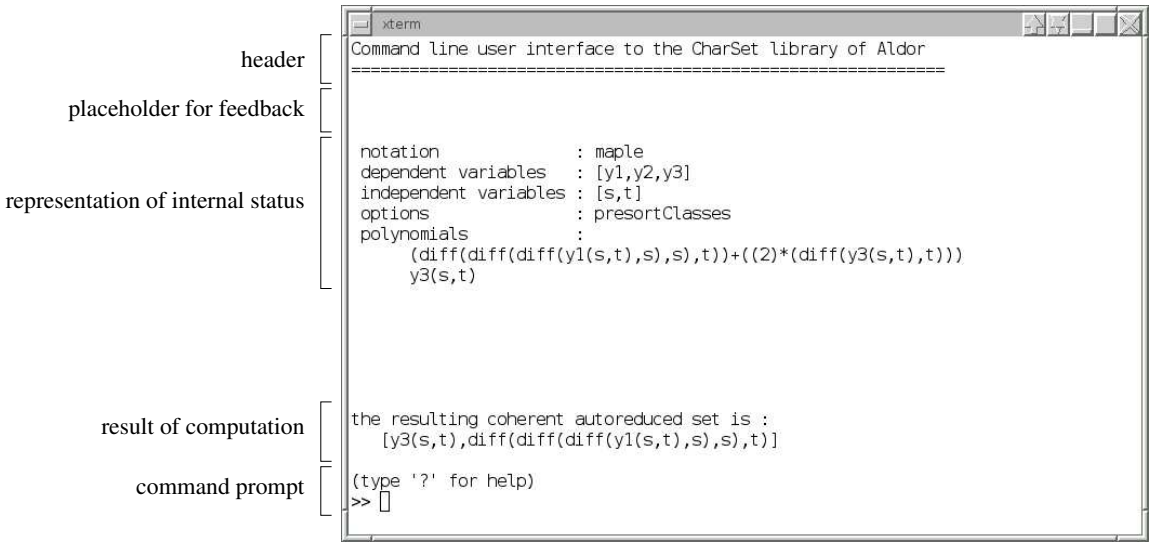

Figure 10.1: Screen layout for the command line utility

Aldor provides various output formats as discussed in Section 7.1. For the command line utility it is desired to get a standalone executable file. The Aldor compiler compiles to such an executable file when given the option `-Fx`. For the execution of the resulting executable neither Aldor nor `CharSet` are required anymore. All the necessary parts of the libraries and Aldor environment are incorporated into the executable.

The command line utility is implemented as interactive program holding five values that correspond to each of the five input parameters of `_charset_coherentAutoreducedSet`.

The first value denotes the used notation. This parameter denotes the used syntax for the second, third and fourth parameter to `_charset_coherentAutoreducedSet`. The second, third and fourth value correspond to the polynomials, dependent and independent indeterminates. The fifth value denotes further options for the computation and corresponds to the fifth parameter of `_charset_coherentAutoreducedSet`.

While the first and the last parameter are internally stored in string representation, the second to fourth parameter are stored as expression trees. The use of expression trees decouples the values from the used notation. This allows to switch between the available notations during the execution of the program. When a textual representation for a value is needed, the expression trees are formatted on the fly in the current notation.

When switching the notation for the command line utility, not only the output of values adapts to the new notation, also the input adapts. The values that are input into the command line utility always have to conform to the currently selected notation.

The program's main screen is divided into five parts, as can be seen in Figure 10.1.

- The header forms the topmost part.

- The second part is used to give feedback about change of values and errors. Any function of the command line utility can add messages to a feedback container. Upon every screen redraw, which typically happens after every command, all messages of the feedback container are printed to the screen and the feedback container is emptied.
- The third part of the main screen of the command line utility gives a representation of the five internal values (used notation, polynomials, dependent indeterminates, independent indeterminates, and options).
- If the previous command was to start the computation with the current values, the result of this computation forms the fourth part.
- The last part is a prompt for accepting commands, which are described in Section 10.2. `Aldor` provides the function << of the domain String for parsing Strings from the standard input of a program. This function, however, comes with the restriction that a string has either to be quoted or consist of only alphanumeric characters[94]. This restriction is not suitable for command prompts. Therefore, the command line utility provides its own function to read a newline separated character string from the standard input.

10.2 Using the command line tool

The command line utility accepts the following commands:

- `help` or `?`

  Entering `help` or `?` gives an overview about the possible commands and their behaviour.

- `notation`

  The command `notation` allows to change the used notation. The default setting is to use Maple's notation. As the command line utility uses `ExtIO`s computer algebra framework, every notation implemented in `ExtIO` can be set. However, at present state `ExtIO` only implements

  - Maple and
  - Mathematica's `FullForm`

  notation.

- `depvars`

  To set the dependent variables, the command `depvars` is used. This command prompts for a list of symbols in the currently set notation and uses these symbols as dependent variables.

[94] This restriction, however, cannot be found in the documentation of String. Only the source code of String has comments mentioning this restriction.

- `indepvars`

  The command `indepvars` allows to set the independent variables to a list of symbols in the currently used notation.

- `polynomials`

  The input polynomials for the computation is set by the `polynomials` command. These polynomials have to be entered in the currently set notation.

- `options`

  Further options for the computation can be set by issuing the command `options`. The available options can be found in Section 7.3.

- `calculate`

  After issuing the `calculate` command, the command line utility passes the supplied data to `CharSet`'s exported `_charset_coherentAutoreducedSet` function. After `CharSet` finishes with the computation, the result is presented on the screen.

- `quit` or `exit`

  The commands `quit` and `exit` are used to quit the command line utility.

11 Comparison of different implementation of polynomials

For many algorithms in computer algebra, there is a consensus which implementation is most efficient. For example in Gröbner basis computations, typically sparse, distributive implementations are used. In the context of characteristic sets, no comparison of the different implementations is available. Such a comparison is presented in this section. However, only the differential part of characteristic set computations (computing coherent autoreduced sets) is investigated, as the algebraic part is typically solved by Gröbner basis techniques, for which sparse, distributive implementations are favorable.

This section is split into three parts. The first part presents the various different approaches to implement polynomials. In the second part, the corresponding implementations of Aldor are discussed. Finally, the comparisons of these implementations in context of coherent autoreduced set computations are presented in the third part.

11.1 Classification of polynomial ring implementations

Implementations for polynomial rings are typically classified by their representation model and their mathematical model.

The representation model can be either dense or sparse. These two terms are explained in the context of the univariate ring $R[y_1]$. In a dense representation, the polynomial

$$r_d y_1^d + \ldots + r_1 y_1^1 + r_0 y_1^0 \qquad (**)$$

with $\forall i \in \{0, 1, \ldots, d\} : r_i \in R$ is stored by explicitly storing every r_i for $\forall i \in \{0, 1, \ldots, d\}$, regardless of the value of r_i. There is no need to store the corresponding term of a coefficient, as there is typically a one-to-one mapping between the term and the position of the coefficient within the data structure[95]. Dense representations are typically used for univariate polynomials of small degree. There, the benefit of efficient access to every coefficient permits efficient addition and multiplication of polynomials. For univariate polynomials of higher degree, where typically many coefficients are zero, a large amount of memory is wasted by storing the zero coefficients. The same holds for dense implementations of multivariate polynomial rings. For example when storing a polynomial of total degree e in the polynomial ring $R[y_1, y_2, \ldots, y_n]$, the coefficients for all terms of total degree less than e have to be stored as well. However, there are $\binom{e+n-1}{e-1}$

[95] Assume that the polynomials of $R[y_1]$ are stored by an array of coefficients that are sorted in ascending order with respect to the degrees of their corresponding terms. Then the polynomial

$$25y_1^3 + 4y_1 + 2$$

is stored as the array

$$2, 4, 0, 25.$$

The i^{th} entry in the array contains the coefficient to y_1^{i-1}. For example, the coefficient to y_1^3 is the fourth element in the array. So the coefficient to y_1^3 is 25.

of such terms with total degree less than e, as can also be seen in [28]. For characteristic set computations, the polynomial ring of interest is $R\{Y\}$, which is a polynomial ring in infinitely many indeterminates. However, almost all of these indeterminates do not occur during the computations. Therefore, dense representations are not considered further for representing $R\{Y\}$.

In a sparse representation only the non-zero r_i of $(**)$ are stored. Therefore, no memory is wasted for storing coefficients that are 0. In contrast to a dense representation, there is no mapping between the term to a coefficient and the position of the coefficient within the data structure. Therefore, sparse representations effectively have to store a representation of the monomial $r_i y_1^i$. Sparse representations are typically used for univariate polynomials of high degree and multivariate polynomials.

The mathematical model of polynomial implementations can be either distributive, recursive, or factorized. The first two models are described in the context of the ring $R[y_1, y_2, \ldots, y_n]$. In the recursive approach, the polynomial ring $R[y_1, y_2, \ldots, y_n]$ is reinterpreted as $R[y_1, y_2, \ldots, y_{n-1}][y_n]$ (i.e.: the univariate polynomial ring in y_n with coefficients in $R[y_1, y_2, \ldots, y_{n-1}]$). $R[y_1, y_2, \ldots, y_{n-1}]$ is again reinterpreted in the same way, and so on. Finally, $R[y_1, y_2, \ldots, y_n]$ can be seen to be modelled by $R[y_1][y_2]\ldots[y_n]$. For example the decomposition of

$$2y_1^2y_2y_3^3 + 7y_2 + y_1$$

in $R[y_1, y_2, y_3]$ can be seen in Table 11.1, where the coefficients are underlined.

$R[y_1, y_2, y_3]$	:	$\underline{2y_1^2y_2y_3^3}$	+	$\underline{7y_2}$	+	y_1
$R[y_1, y_2][y_3]$	:	$\underline{2y_1^2y_2}y_3^3$	+	$\underline{7y_2 \quad + \quad y_1}$		
$R[y_1][y_2]$	:	$\underline{2y_1^2}y_2$				
$R[y_1]$	:	$\underline{2y_1^2}$			$\downarrow$	
R	:	2				
$R[y_1][y_2]$	:			$\underline{7y_2}$	+	$\underline{y_1}$
$R[y_1]$	:			$\underline{7}$		$\downarrow$
R	:			7		
$R[y_1]$	:					y_1
R	:					1

Table 11.1: Decomposition of $2y_1^2y_2y_3^3 + 7y_2 + y_1$ for recursive implementations of polynomial rings.

In the second line $7y_2 + y_1$ is underlined as a whole, as it is the coefficient to y_3^0. The same holds for y_1 in the sixth line, which is the coefficient to y_2^0 and 7 in the seventh line, which is the coefficient of y_1^0.

In recursive implementations, the extraction of the largest indeterminate and the extraction of coefficients with respect to the largest indeterminate are cheap operations. Furthermore, adjoining new indeterminates to the polynomial ring does not require updating the used data structures.

Distributive implementations for polynomials make use of the module structure of the polynomial ring. A polynomial is interpreted as sum of monomials. These monomials

are dissected into coefficient and term. While storing the coefficient is a matter of the coefficient ring, there is a vast amount of different strategies for storing the term. Typically the terms are stored by "exponent vectors", which is a list or an array of the exponents to the indeterminates of the term.

In distributive implementations of polynomial rings arithmetics can be performed faster than in recursive implementations. However, the algorithms for the arithmetics are more complicated and harder to implement.

The third possibility for storing polynomials are factorized implementations. Factorized implementation provide some implementation for representing the irreducible polynomials of the polynomial ring. For this task, recursive, distributive, or a completely different representation[96] is used. In such representations multiplication and divisions can be performed extremely fast. However, addition is a very expensive operation in such a representation. In the general setting it cannot be expected that the summands and the sum have common factors. Therefore, for every addition in a factorized setting, the sum has to be factored into irreducible polynomials, which is expensive.

In characteristic set computations additions typically occur in reductions. There, for every addition only two multiplications occur. So, two multiplications have to make up for the addition and its factorization. As factorization itself typically is more costly than two multiplications, a factorized representation of polynomials cannot be expected to outperform recursive or distributive implementations. Therefore, the factorized representation of polynomials is not taken further into account.

Note that there is a difference between, the sparse representation and the distributive implementation of polynomials. For example MAXIMA [22] uses a sparse, *recursive* implementation to model polynomials. The important property of the sparse representation is to omit those monomials, whose coefficient are zero. The important property of a distributive implementation is the focus on the module structure.

Among the various systems for performing computer algebra, only MOCKMMA [23] is known to use a dense, recursive implementation of polynomials, as illustrated in [35]. Most systems use either a sparse, recursive implementation (e.g.: MACSYMA [16] and its free successor MAXIMA) or a sparse, distributive implementation (e.g.: MAGMA [17] and SINGULAR [97][36]) to model polynomials. No system implementing a dense, distributive representation or a factorized implementation could be found. There are, however, systems that collect information about expressions in dictionaries (e.g.: MATHEMATICA). If a polynomial is factored in such a system, the system stores the factorization additionally to the internal representation of the polynomial.

[96] By using an a priori mapping between the natural numbers and the irreducible polynomials of a ring, numbers can be used to refer to the differential polynomials.

[97] The authors of SINGULAR state in [30] to use a dense, distributive implementation for polynomials in SINGULAR. However, their definition of "dense" does not match the definition of dense in this report. They use "dense" to denote exponent vectors that store the exponent of every indeterminate within a monomial, regardless whether or not the exponent is zero (compare also the use of "dense" in [29]). Using the notation of this report, SINGULAR uses a *sparse*, distributive implementation for polynomials.

11.2 Polynomial ring implementations for characteristic set computations

This section describes and classifies the available implementations of differential polynomial rings in Aldor using the classification of Section 11.1.

In the first part of this section, the recursive implementations are presented. Afterwards, the distributive implementation is described along with the various implementations for storing terms. Finally all possible implementations are listed and given abbreviations, which are used in Section 11.3.

The `Algebra` library provides three implementations of multivariate polynomials that can be used to model $R\{Y\}$, as previously discussed in Section 3.3.

These implementations are.

- RecursiveMultivariatePolynomial0,
- SparseMultivariatePolynomial, and
- DistributedMultivariatePolynomial1.

None of these implementations models a dense representation. However, for modelling $R\{Y\}$, a dense representation is not feasible anyway, as $R\{Y\}$ is a polynomial ring in infinitely many variables. Even for polynomial rings in finitely many variables sparse representations outperform dense representations. Therefore, dense representations are neglected.

RecursiveMultivariatePolynomial0 is a sparse, recursive implementation taking three parameters. The first parameter is a mapping of a coefficient domain to a UnivariatePolynomialCategory[98] category of this coefficient domain. This mapping is used for obtaining a univariate polynomial domain, when recursively breaking multivariate polynomials into univariate polynomials in their largest indeterminate. The second and third parameter to RecursiveMultivariatePolynomial0 denote the coefficient domain and a domain for the indeterminates. For this comparison, the second and third parameter refer to R and the indeterminates of $R\{Y\}$.

`Algebra` provides two implementations, which satisfy UnivariatePolynomialCategory and can therefore be used in the first parameter to RecursiveMultivariatePolynomial0. These implementations are

- DenseUnivariatePolynomial and
- SparseUnivariatePolynomial.

[98] UnivariatePolynomialCategory is `Algebra`'s category for univariate polynomials.

DenseUnivariatePolynomial is `Algebra`'s implementation for univariate polynomials in dense representation. This domain takes two parameters. The first parameter denotes the coefficient domain of the polynomial ring and has to provide ArithmeticType and ExpressionType. The second parameter is an optional Symbol which denotes the indeterminate of the polynomial ring. The internal representation of DenseUnivariatePolynomial is an array of coefficients. The position of a coefficient in this array is determined by the degree of its corresponding term. The i^{th} entry in the array holds the coefficient to the term of degree $i-1$ (compare the footnote in the discussion of dense in Section 11.1). The array representing a polynomial of degree d has at least $d+1$ slots but not necessarily exactly $d+1$. Thereby, changes in the degree do not necessarily lead to reallocating memory for a new coefficient array. Nevertheless, the degree can always be determined in constant time, as it is stored explicitly in DenseUnivariatePolynomial's internal representation of a polynomial.

SparseUnivariatePolynomial is `Algebra`'s implementation for univariate polynomials in sparse representation. This domain takes the same parameters as DenseUnivariatePolynomial. Internally, SparseUnivariatePolynomials are represented by a sorted list of monomials. This list is sorted decreasing with respect to the degree of the monomials. The monomials are represented as pairs of the coefficient and the degree of the corresponding term.

RecursiveMultivariatePolynomial0 together with DenseUnivariatePolynomial or SparseUnivariatePolynomial allows to model sparse, recursive multivariate polynomials where the recursive step is carried out using dense or sparse univariate polynomials.

SparseMultivariatePolynomial is a sparse, recursive implementation of multivariate polynomials, where the recursive step is performed with the help of SparseUnivariatePolynomial. This domain take two parameters, which are the coefficient domain and the indeterminates for the multivariate polynomial ring. SparseMultivariatePolynomial is equivalent to RecursiveMultivariatePolynomial0 with the first parameter set to SparseUnivariatePolynomial. When comparing the source code of SparseMultivariatePolynomial and RecursiveMultivariatePolynomial0, it can be seen that only the header of the files and the documentation within the files differ. The only difference is, that for RecursiveMultivariatePolynomial0, the univariate polynomial domain is given by a parameter, while it is fixed with the help of a macro in SparseMultivariatePolynomial.

DistributedMultivariatePolynomial1 is a sparse, distributive implementation for multivariate polynomials. This domain takes three parameters, of which the first and second denote the coefficient domain and the indeterminates for the multivariate polynomial ring. The third parameter is an ExponentCategory of the indeterminates. This parameter is used to model the terms of monomials. The category ExponentCategory is treated in more detail after the discussion of DistributedMultivariatePolynomial1. The internal representation of DistributedMultivariatePolynomial1 is a sorted list. The list's elements are pairs of coefficient and term. As stated before, the term is represented by the given implementation of ExponentCategory. The list of pairs is sorted in decreasing order with respect to the total order of terms provided by the implementation of ExponentCategory.

`Algebra` provides the two categories ExponentCategory and GeneralExponentCategory

for modelling terms. ExponentCategory is an extension of GeneralExponentCategory. GeneralExponentCategory interprets terms as elements in an *additive* monoid with a total order. Additionally, implementations of GeneralExponentCategory have to provide a cancellation function. According to the documentation of `Algebra`, a function f is called cancellation function, if and only if it maps two terms a and b to a term c, such that $a = b + c$, if such a c exists. ExponentCategory extends GeneralExponentCategory by conversion functions between indeterminates and terms. Furthermore, ExponentCategory provides functions that interpret terms as *multiplicative* monoid, like

- degree for obtaining the degree of a term with respect to an indeterminate,
- gcd to compute the greatest common divisor of terms, and
- lcm to compute the least common multiple of terms.

Despite this mixture of interpreting the monoid additively and multiplicatively, ExponentCategory is documented to interpret terms as additive monoid with a total order.

`Algebra`'s implementations of ExponentCategory pick their elements from a finite set of indeterminates. For $R\{Y\}$, the set of indeterminates is infinite. Therefore, none of `Algebra`'s implementations of ExponentCategory can be reused. `CharSet` implements five implementations of ExponentCategory that allow to build terms with elements from an infinite set of elements. These implementations are

- ListExponent,
- ListSortedExponent,
- SortedListExponent, and
- CumulatedExponent, and
- ClassPresortedDifferentialExponent.

The first four of the presented domains take a parameter of VariableType. This parameter is referred to as `VARS` and denotes the indeterminates that may be used to built terms. For $R\{Y\}$, this parameter is an implementation for the derivatives ΘY. The first three domains export ExponentCategory and CopyableType.

ListExponent represents terms as list of `VARS`. This list is not sorted. This implementation is not meant to be performant, but is used to compare the effect of sorting the representation.

Terms are represented as decreasingly sorted list of `VARS` in ListSortedExponent. The sorted list is not represented as SortedList, which is `Aldor`'s domain for sorted lists, but via a plain List that is only sorted when necessary. However, SortedList are sorted in increasing order.

To show the difference between various orders, SortedListExponent represented as increasingly sorted list of VARS. Internally, the list is represented as SortedList.

CumulatedExponent models the term as sorted list of pairs. Each pair consists of an indeterminate (i.e.: an element of VARS) and an integer, denoting its positive degree within the term. In the list of pairs, each indeterminate occurs at most once. The pairs within the list are stored in descending order with respect to the indeterminates of the pairs. Table 11.2 illustrates the implemented strategy with the help of the term $y_1^2 y_3$, where $y_1 < y_2 < y_3$.

representation	valid?	
$[(y_3, 1), (y_1, 2)]$	yes	
$[(y_3, 1), (y_2, 0), (y_1, 2)]$	no	pair with degree 0
$[(y_3, 1), (y_1, 1), (y_1, 1)]$	no	y_1 occurs in more than one pair
$[(y_1, 2), (y_3, 1)]$	no	wrong order of pairs

Table 11.2: Examples of CumulatedExponent.

ClassPresortedDifferentialExponent is tailored for modelling terms of indeterminates of differential polynomials. This domain separates a term into terms that contain only indeterminates of one class. Each of these terms is then represented by a separate implementation of ExponentCategory. These separated terms are stored in an array. ClassPresortedDifferentialExponent takes three parameters. The first parameter, VARS, is a FinitieVariableType and corresponds to Y in $R\{Y\}$. The second parameter corresponds to the indeterminates of the differential polynomial ring $R\{Y\}$ and has to be an EliminationOrderedVariableType[99] for VARS. An implementation that satisfies ExponentCategory and CopyableType is the third parameter. This implementation is used to store the separated terms, as described earlier.

For example, the array for the separation of $(\delta_2 y_1)^2 y_1 y_3$ of the differential polynomial $R\{Y\}$ can be seen in Table 11.3[100].

position	0	1	2	3	4	$\cdots$	n
value	1	$(\delta_2 y_1)^2 y_1$	1	y_3	1	$\cdots$	1

Table 11.3: Separation of $(\delta_2 y_1)^2 y_1 y_3$ for ClassPresortedDifferentialExponent.

The terms $(\delta_2 y_1)^2 y_1$ and y_3 are modelled individually by the ExponentCategory parameter of ClassPresortedDifferentialExponent.

The presented implementations allow eleven different implementations for differential polynomials. The following list gives these implementations along with the used abbreviations for Section 11.3.

- RESP is RecursiveMultivariatePolynomial0 with SparseUnivariatePolynomial

[99] A description of EliminationOrderedVariableType can be found in Section 3.2.

[100] Note that the value 1 in the table corresponds to 0 in the additive representation of terms used by Algebra's ExpressionCategory.

- `REDE` is RecursiveMultivariatePolynomial0 with DenseUnivariatePolynomial
- `SPAR` is SparseMultivariatePolynomial
- `DILI` is DistributedMultivariatePolynomial1 with ListExponent
- `DILS` is DistributedMultivariatePolynomial1 with ListSortedExponent
- `DISL` is DistributedMultivariatePolynomial1 with SortedListExponent
- `DICU` is DistributedMultivariatePolynomial1 with CumulatedExponent
- `DCLI` is DistributedMultivariatePolynomial1 with ClassPresortedDifferentialExponent and ListExponent
- `DCLS` is DistributedMultivariatePolynomial1 with ClassPresortedDifferentialExponent and ListSortedExponent
- `DCSL` is DistributedMultivariatePolynomial1 with ClassPresortedDifferentialExponent and SortedListExponent
- `DCCU` is DistributedMultivariatePolynomial1 with ClassPresortedDifferentialExponent and CumulatedExponent

11.3 Run-time comparison of the differential polynomial ring implementations

This section gives a comparison of the performance of available implementations of differential polynomial rings, as presented in Section 11.2.

First, general remarks about the environment for the tests are made. These remarks are followed by a presentation of the test cases. Finally, the table with the results of the tests is given and the results are discussed.

For a convenient notation of the compared differential polynomial ring implementations, the abbreviations of page 107 are used throughout this section. For example, `RESP` is used for denoting RecursiveMultivariatePolynomial0 with SparseUnivariatePolynomial for the recursive step.

The tests are executed on an AMD Athlon 64 3000+ with 1 GiBi of main memory. Although this processor is a 64-bit processor, the tests are compiled to 32-bit executables, as Aldor cannot produce proper 64-bit executables at the time of executing the tests[3].

The domain Timer of `Aldor` allows to measure the time Aldor code takes to execute. On `GNU/Linux`, measures by Timer denote the sum of the user time[101] and system time

[101] `GNU/Linux` associates four different values with a process. "User time" denotes the time the processor executes instructions of the process. This value denotes how much time has been spent executing the process itself. "System time" refers to the time spent executing instructions by the system on behalf of the calling process. The third and fourth value denote the sum of user time and system time for all terminated child processes. More information about this topic can be found in [43, times].

for the current process and all terminated child processes. As the provided examples do not fork off child processes, there is no contribution of terminated child processes. Therefore, for the examples of this section Timer measures the sum of the user time and the system time of the test. Timer does not allow to single out the user time. However, the user time is the time of interest. Therefore, `ExtIO`'s domain Times is used to time the examples, as Times allows to obtain the user time of a process.

Each of the tests is run three times. For each test and each run, taking the time starts right before calling CoherentAutoreducedSet's triangularize function. At this time the polynomials have already been set up and collected in a list. Taking the time stops right after triangularize returns. Finally, the arithmetic mean of the user times of a test is considered the time a test takes.

The tests themselves are artificial tests (except `pendulum`) for benchmarking only. Most of the few examples for characteristic set computations found in literature cannot be used in this comparison. They are either too simple to obtain meaningful timings or so hard that at least some of the presented implementations cannot compute them with the given memory[102]. Especially the second limitation had impact when designing the tests. Many implementations are expected to become more efficient as the polynomials get more sophisticated. However, such polynomials cannot be used due to the excessive memory consumption of Aldor and especially its recursive implementations for polynomial rings.

The first test is denoted by `selection`. In this test a coherent autoreduced set for the polynomials

$$\left\{\delta_1\delta_2^3 y_i \;\middle|\; i \in \{1, 2, \ldots, 128\}\right\}$$

in $\mathbb{Q}\{y_1, y_2, \ldots, y_{128}\}$ with $\Delta = \{\delta_1, \delta_2\}$ has to be computed. DifferentialVariableOrderlyEliminationOrderTools is used as total order of the derivatives.

During the test `selection`, no new polynomials have to be taken into consideration. No Δ-polynomials arise during the computation. This test's focus is on the performance of selecting the elements of minimal rank.

The second test focuses on Δ-polynomials and is therefore called `delta`. The set of polynomials for `delta` is

$$\{\delta_i y_3 + \delta_{i+1} y_2 \mid i \in \{1, 3, 5, 7\}\} \cup \{\delta_{i+1} y_3 + \delta_i y_1 \mid i \in \{1, 3, 5, 7\}\}$$

in $\mathbb{Q}\{y_1, y_2, y_3\}$ with $\Delta = \{\delta_1, \delta_2, \ldots, \delta_8\}$. Again, DifferentialVariableOrderlyEliminationOrderTools is used as total order of the derivatives.

[102] Aldor is a fully typed language and has to provide a domain for every object it is dealing with. Especially for recursive implementations of polynomials, this requirement causes instantiation of many domains. These instantiations can be seen to allocate large amounts of memory again and again. On the described test machine, 100 MeBi of memory have been allocated every other second. This behaviour is typical for Aldor programs and is not limited to the use of polynomials. When running out of main memory, Aldor programs typically cause a segmentation fault. Compiled Aldor code does not give any error message or debug information. However, the memory allocations are not due to the used source code, but due to the code generated by the compiler. Therefore, failed memory allocations cannot be handled by the source code but are a matter of the compiler itself.

`delta` starts with polynomials of the same class but different order with respect to their class. Thus, the computation of Δ-polynomials yields new polynomials. Δ-polynomial computation for these new polynomials yield further polynomials and so on.

The third test, called `pendulum`, models a planar pendulum of constant mass and is described in [38, example 7.2]. The polynomials for the test are

$$\left\{m\delta_t^2 x + \lambda x, m\delta_t^2 y + \lambda y - g, x^2 + y^2 - l^2\right\}$$

in $\mathbb{Q}\{g, l, m, y, x, \lambda\}$ with $\Delta = \{\delta_t\}$. The used order is

```
DifferentialVariableMixedOrderTools( [
  ( DifferentialVariableLexicographicOrderTools, 3 ),
  ( DifferentialVariableOrderlyOrderTools, 2 ),
  ( DifferentialVariableLexicographicOrderTools, 1 )
] );
```

This order is a lexicographic order on $\{g, l, m\}$ and an orderly order on $\{y, x\}$. Furthermore, any derivative of g, l, or m is less than any derivative of y or x, which is in turn less than any derivative of λ.

As this order is not an elimination order, ClassPresortedDifferentialExponent cannot be used. Therefore, `DCLI`, `DCLS`, `DCSL`, and `DCCU` cannot be used to perform this test.

The test `tail` computes a coherent autoreduced set of the polynomials

$$\left\{(1 + \delta_j^3 y_i)\delta_{j+1}^3 y_i \;\middle|\; i \in \{1, 2, \ldots, 5\} \wedge j \in \{1, 2, 3\}\right\}$$

in $\mathbb{Q}\{y_1, y_2, \ldots, y_5\}$ with $\Delta = \{\delta_1, \delta_2, \delta_3, \delta_4\}$. The total order on the derivatives is DifferentialVariableOrderlyEliminationOrderTools.

Table 11.4 gives the run-time of the presented tests and implementations of polynomial rings in seconds of user-time.

For `selection`, the recursive implementations (`RESP`, `REDE`, `SPAR`) themselves are relatively close to each other, but are far slower than the distributive ones, as they take more than 100-times as long as `DILS`. It can also be seen that using ClassPresortedDifferentialExponent adds a performance penalty of about factor 8 in this test. This observation is not unexpected, as the polynomials only involve a single indeterminate. It is astonishing to see `DCCU` taking more than twice as long as the other implementations that use ClassPresortedDifferentialExponent.

In the test `delta`, the times for the recursive implementations for differential polynomial rings are again similar to each other. Although the difference between recursive and distributive implementation is less than in the test `selection`, recursive implementations are still 20 to 45 times slower than distributive implementations. Again, the use of ClassPresortedDifferentialExponent results in a performance penalty. This penalty is around factor 0.45. This penalty is also valid for `DCCU`.

	selection	delta	pendulum	tail
RESP	21.62	18.37	0.65	52.80
REDE	21.54	18.30	0.50	47.13
SPAR	21.58	17.40	0.65	53.06
DILI	0.18	0.49	0.26	39.30
DILS	0.13	0.42	0.10	21.73
DISL	0.17	0.61	0.43	56.01
DICU	0.16	0.53	0.11	23.50
DCLI	1.42	0.73	—	46.56
DCLS	1.37	0.58	—	22.01
DCSL	1.44	0.82	—	42.36
DCCU	3.91	0.84	—	25.06

Table 11.4: Comparison of polynomial implementations.

The gap between recursive and distributive implementations is narrow for the test `pendulum`. The fastest recursive implementation (`REDE` with 0.50 seconds) is only 0.07 seconds slower than the slowest distributive implementation (`DISL` with 0.43 seconds). As mentioned before, the missing values of `DCLI`, `DCLS`, `DCSL`, and `DCCU` are not caused by a program fault but the setting of the test `pendulum`. As this test does not use an elimination order, ClassPresortedDifferentialExponent cannot be used.

For last test (`tail`), the times between the different distributive implementations vary a lot. The times range from 21.73 seconds for `DILS` to 56.01 seconds for `DISL`. `DISL` is even slower than any recursive implementation. However, this bad performance of `DISL` cannot be explained, especially since its counterpart `DCSL`, which typically performs worse than `DISL` due to ClassPresortedDifferentialExponent, only takes 42.36 seconds to finish the test. Therefore, it is assumed, that the performance loss is due to compiler issues and not directly caused by the structure of `DISL` itself. Even, when omitting the result for `DISL`, the recursive implementations are close to some of the distributive implementation. However, the fastest distributive implementations (`DILS`, `DCLS`, `DICU`, and `DCCU`) perform the test in half the time of the recursive implementations.

Finally, the tests showed that for characteristic set computations, Aldor's recursive implementations of polynomials is less performant than the distributive ones. Furthermore, the use of ClassPresortedDifferentialExponent caused a penalty. However, the polynomials used in tests may be too simple to take advantage of ClassPresortedDifferentialExponent. For each of the four tests `DISL` performed best. The second best implementation is `DICU`. However, the used polynomials may be too small to show benefits for using `DICU`.

12 Comparison to other characteristic set implementations

In this section, other implementations for performing characteristic set computations are investigated and compared to each other.

The investigated implementations are

- `diffalg`,
- `epsilon`, and
- `WuRittSolva`.

Each of them is treated in their corresponding section. Finally, the table of features of the implementations is given and the advantages of each of the implementations is discussed.

12.1 `diffalg`

`diffalg` [31] is a MAPLE package for differential characteristic set computations.

To distinguish between the various versions of `diffalg`, each version is numbered by their year of distribution.

Although, the original version of `diffalg`, which was `diffalg(96)`, has been released in 1996 for MAPLE V.5, `diffalg` is still under active development. The most recent release of `diffalg` is `diffalg(04)` and can be obtained at the `diffalg` homepage at [31]. Additionally, `diffalg` is also a core package of MAPLE. Therefore, `diffalg` is shipped with every version of MAPLE. Although the `diffalg` package provides a function (`version`) to determine its version, this function is not available in the packages that are shipped with MAPLE. Therefore, it cannot be identified which version of `diffalg` is used in which version of MAPLE. According to the homepage of `diffalg`, MAPLE uses the `diffalg(00)` from MAPLE 6 onwards. However, two newer versions are available (`diffalg(01)` and `diffalg(04)`).

For this report, `diffalg(04)` is evaluated. In the further description, `diffalg` is used to refer to `diffalg(04)`.

The `diffalg` package performs characteristic set computations for equations and inequations of differential polynomials. Furthermore, `diffalg` is not limited to commutative polynomials, but can also deal with non-commutative polynomials. The implemented algorithms are described in [40], [39], [41], and [42]. Besides the main algorithms for characteristic set computation, `diffalg` also provides various fundamental functions for differential polynomials. Among others, these functions are applying a derivation to a

differential polynomial, reducing differential polynomials, computing initials, separants, or Δ-polynomials.

Although, MAPLE typically does not urge its users to specify the domains for the computations, the `diffalg` package requires to specify the differential polynomial ring to compute in. It is necessary to specify an identifier for each derivation, and a ranking. `diffalg` provides six different rankings, which can be combined like `CharSet`'s DifferentialVariableMixedOrderTools (see Section 3.2.1). Although generally used rankings can be modelled with `diffalg`'s rankings, the implemented set of rankings cannot be extended. There is no possibility to introduce new rankings.

In `diffalg`, differential polynomials can be entered using either

- `diff`,
- `jet`, or
- `vjet` notation.

`diff` notation uses MAPLE's `diff` operator build indeterminates. `jet` notation attaches the symbols of the derivations to differential indeterminates using MAPLE's `indexed` expressions. Similarly, `vjet` notation attaches the powers of derivations to differential indeterminates. While `diff` notation expresses the semantic of a differential polynomial better, `jet` and `vjet` notation are more compact. Let δ_1 and δ_2 denote taking the partial derivatives with respect to some x and y and let u be some object to apply the derivations to. Then some examples for the different notations are given by Table 12.1.

mathematical	`diff`	`jet`	`vjet`
u	`u(x,y)`	`u[]`	`u[0,0]`
$\delta_2 u$	`diff(u(x,y),y)`	`u[y]`	`u[0,1]`
$\delta_1^2\delta_2 u$	`diff(diff(diff(u(x,y),x),x),y)`	`u[x,x,y]`	`u[2,1]`
$\delta_1\delta_2 u + \delta_1 u$	`diff(diff(u(x,y),x),y)+` `+diff(u(x,y),x)`	`u[x,y]+u[x]`	`u[1,1]+` `+u[1,0]`

Table 12.1: Examples for the available notations of `diffalg`.

For the following discussion, let $[p]$ denote the differential ideal generated by some p. Furthermore, let $[\![p]\!]$ be the radical differential ideal generated by p, and s_p denote the separant of p. For the demonstration of `diffalg`, example 8.1 of [39] is solved. This example is looking for solutions to $p = 0$ with

$$p := (y_{tt} + y^3 y_t)^2 - (y y_t)^2 (4y_t + y^4)$$

in $\mathbb{Q}\{y\}$ with $\Delta = \{t\}$. The solutions to $p = 0$ can be found by computing characteristic decomposition of $[\![p]\!]$ and investigating each of the components. This characteristic decomposition is

$$[\![p]\!] = [q_1] : \mathrm{s}_{q_1}^{\infty} \cap [q_2] : \mathrm{s}_{q_2}^{\infty} \cap [y]$$

with

$$\begin{aligned} q_1 &:= \quad y_{tt}^2 + 2y_{tt}y_t y^3 - 4y_t^3 y^2 \\ q_2 &:= \qquad y_t(4y_t + y^4). \end{aligned}$$

The following piece of code illustrates, how this result can be achieved with `diffalg`. Note it is assumed that `diffalg(04)` is installed in the current directory. Then the first two lines load `diffalg(04)` instead of the MAPLE's built-in `diffalg`.

```
    |\^/|     Maple 9.5 (IBM INTEL LINUX)
._|\|   |/|_. Copyright (c) Maplesoft, a division of Waterloo Maple Inc. 2004
 \  MAPLE  /  All rights reserved. Maple is a trademark of
 <____ ____>  Waterloo Maple Inc.
      |       Type ? for help.
> libname := ".", libname:
> with(diffalg):
Warning, the protected name version has been redefined and unprotected
> R := differential_ring (derivations=[t], ranking=[y], notation=vjet):
> p:=( y[2]+y[0]^3*y[1] )^2 - ( y[0]*y[1] )^2 * ( 4*y[1]+y[0]^4 );
                              3      2       2     2               4
             p := (y[2] + y[0]  y[1])  - y[0]  y[1]  (4 y[1] + y[0] )

> S := Rosenfeld_Groebner( [ p ], R );
             S := [characterisable, characterisable, characterisable]

> equations(S[1]);
                         2              3              2     3
                    [y[2]  + 2 y[2] y[0]  y[1] - 4 y[0]  y[1] ]

> inequations(S[1]);
                                      3
                                [y[2] + y[0]  y[1]]

> equations(S[2]);
                                    4         2
                               [y[1] y[0]  + 4 y[1] ]

> inequations(S[2]);
                                   4
                                [y[0]  + 8 y[1]]

> equations(S[3]);
                                    [y[0]]

> inequations(S[3]);
                                      []
```

12.2 epsilon

epsilon [51] is again a MAPLE package for characteristic set computations.

epsilon has been released in 2003 in version 0.618 and has not seen any updates since then. In contrast to diffalg, epsilon is not a core package of MAPLE and needs to be installed separately.

The author of epsilon, Dongming Wang, uses a different notation than this report (see Section 5.1). Therefore, the function names of epsilon do *not* correspond to the terminology of this report. For example, the epsilon's charset function does not compute a characteristic set in the notation of this report.

epsilon package can deal with polynomial equations and inequations as well as with ordinary differential polynomial equations and inequations. However, there is no support for partial differential equations or inequations. Neither is there any support for non-commutative settings. A description of the implemented algorithms can be found in [50]. While the algorithms of diffalg are rigid, several algorithms of epsilon can be tweaked. Algorithms that use subalgorithms (e.g.: charser) typically allow to specify them explicitly. These subalgorithms have to be chosen from a given set of algorithms, which cannot be extended. Therefore, epsilon cannot be considered to be a generic implementation for characteristic set computations.

In contrast to diffalg, the polynomial ring for the computations need not be modelled explicitly. For the ordinary differential case it is, however, necessary to declare the dependencies among the variables. Furthermore, MAPLE's diff notation cannot be used. Each indeterminate of an ordinary differential polynomial ring is represented by its own symbol in epsilon. There is, however, a function (format) that allows to build these symbols. Nevertheless, differential polynomials in MAPLE's diff format have to be converted indeterminate by indeterminate to epsilon's symbols.

epsilon contains functions aiding geometric theorem proving. These functions form an important sub-part of epsilon and can be used to produce interactive visualizations. This part is however beyond the scope of this report and is therefore not considered.

The usage of epsilon is illustrated with the help of example 8.1 of [39] (see page 113). The following piece uses epsilon to compute the characteristic decomposition for this problem.

```
    |\^/|     Maple 9.5 (IBM INTEL LINUX)
._|\|   |/|_. Copyright (c) Maplesoft, a division of Waterloo Maple Inc. 2004
 \  MAPLE  /  All rights reserved. Maple is a trademark of
 <____ ____>  Waterloo Maple Inc.
      |       Type ? for help.
> libname := ".", libname:
> with(dtrisys):
                        dTriSys 1.1 (C) 2003 by Dongming Wang

dTriSys> depend([t,y]):
dTriSys> yt:=df(y,1):
dTriSys> ytt:=df(y,2):
dTriSys> p:=( ytt+y^3*yt )^2 - ( y*yt )^2 * ( 4*yt+y^4 );
                                   3    2    2   2          4
                     p := (y'' + y  y')  - y  y'  (4 y' + y )

dTriSys> dtriser([[p],[]],[t,y]);
     2          3         2   3           3                     4            4
[[y''  + 2 y'' y  y' - 4 y  y' ], {y'' + y  y'}], [[y' (4 y' + y )], {8 y' + y }],

    [[y], {}]
```

12.3 WuRittSolva

WuRittSolva [46] is a MATHEMATICA package for characteristic set computations.

WuRittSolva has been published in 2005 in version 1.0. Although its documentation claims to be a work in progress and still many features are missing, it has not seen any updates since then. WuRittSolva is not a core package of MATHEMATICA and requires manual installation. Although WuRittSolva can only be obtained as an installer for Windows, the extracted MATHEMATICA files work properly under GNU/Linux.

WuRittSolva uses the notation of Dongming Wang for characteristic sets (see Section 5.1), which is different to the notation for this report. Therefore, the names of WuRittSolva's functions do *not* correspond to the algorithms and concepts of this report's work. Additionally, the documentation of WuRittSolva's function is misleading, as for example the function CharacteristicSet only computes a quasi-Wang characteristic set [103], although the documentation claims to compute a Wang characteristic set. According to Hua-Shan Liu, the author of WuRittSolva, these issues will be corrected in future versions.

At the time of writing this theses, WuRittSolva has only seen a quarter of a man-year of development and is therefore still in an early stage of development. As a result, the package cannot compete with the features of diffalg or epsilon. In the current version, WuRittSolva covers only commutative, algebraic polynomials. The implemented algorithms allow to compute basic sets and autoreduced sets and can do some basic geometric theorem proving. Additionally, several basic algorithms for polynomials are provided, like computing the class of a polynomial.

A unique feature of WuRittSolva is the ability to monitor the process of computation. As seen in the example at the end of this description, WuRittSolva can give information about intermediate results. Although this feature is not useful for performing larger computations, it can be used to demonstrate how characteristic set computations work for small systems.

The following code uses WuRittSolva to solve example 2.2.3 in [50]. This example computes a Wang characteristic set of $x_1x_4^2 + x_4^2 - x_1x_2x_4 - x_2x_4 + x_1x_2 + 3x_2$, $x_1x_4 + x_3 - x_1x_2$, and $x_3x_4 - 2x_2^2 - x_1x_2 - 1$.

The following MATHEMATICA code assumes that WuRittSolva is installed into the Autoload directory and therefore automatically loaded when starting MATHEMATICA.

[103] Let P denote a set of polynomials. According to [50], a subset C of the differential ideal of P with

$$\forall p_1, p_2 \in C : p_1 = p_2 \vee \text{class of } p_1 \neq \text{class of } p_2$$

that reduces every element of P to 0 is called *quasi-Wang characteristic set of* P. While a Wang characteristic set is an autoreduced set, a quasi-Wang characteristic set need not be autoreduced. For algebraic polynomial rings and differential polynomial rings with exactly one derivation, every Wang characteristic set is a quasi-Wang characteristic set. In the differential polynomial rings with two or more derivatives this implication need not hold.

```
Mathematica 5.1 for Linux
Copyright 1988-2004 Wolfram Research, Inc.
 -- Motif graphics initialized --

In[1]:= F1:=x1*x4^2+x4^2-x1*x2*x4-x2*x4+x1*x2+3*x2

In[2]:= F2:=x1*x4+x3-x1*x2

In[3]:= F3:=x3*x4-2*x2^2-x1*x2-1

In[4]:= {c1,c2,c3}=CharacteristicSet[{F1,F2,F3},{x1,x2,x3,x4},TracePrintOn->True]
{CS_STEP:1, {x3 + x1 (-x2 + x4)}}
                     2             2                2
{CS_STEP:2, {-x1 - x1  x2 - 2 x1 x2  + x1 x2 x3 - x3 , x3 + x1 (-x2 + x4)}}
                                           2          2
{CS_STEP:3, {-(x1 (1 + x1 - 2 x1 x2 + 2 x2  + 2 x1 x2 )),

             2             2                2
>    -x1 - x1  x2 - 2 x1 x2  + x1 x2 x3 - x3 , x3 + x1 (-x2 + x4)}}
                                           2          2
{A New Component:1, 1 + x1 - 2 x1 x2 + 2 x2  + 2 x1 x2 }
{Total 1 Branch(s) of New Component(s) Discovered}

                  2       2             2       2   2
Out[4]= {-x1 - x1  + 2 x1  x2 - 2 x1 x2  - 2 x1  x2 ,

             2             2                2
>    -x1 - x1  x2 - 2 x1 x2  + x1 x2 x3 - x3 , -(x1 x2) + x3 + x1 x4}
```

As discussed before, `CharacteristicSet` computes a quasi-Wang characteristic set instead of a Wang characteristic set. Therefore, the result has to be transformed into an autoreduced set. In this case, it suffices to reduce `c2` with respect to `c1` to obtained an autoreduced set.

```
In[5]:= {c1, Expand[c2*(-x1-1)-(-1)*c1], c3}

                  2       2             2       2   2
Out[5]= {-x1 - x1  + 2 x1  x2 - 2 x1 x2  - 2 x1  x2 ,

         2        3                   2           2        2
>    3 x1  x2 + x1  x2 - x1 x2 x3 - x1  x2 x3 + x3  + x1 x3 ,

>    -(x1 x2) + x3 + x1 x4}
```

12.4 Comparison chart

Table 12.2 compares the features of the `CharSet` library to those of `diffalg`, `epsilon`, and `WuRittSolva`.

`CharSet`'s main advantage is its generic approach. `CharSet` allows to be extended by new rankings, new notation, and new reductions. Furthermore, it does not require any proprietary computer algebra system, like MAPLE or MATHEMATICA. Nevertheless, its

	CharSet	diffalg	epsilon	WuRittSolva
partial differential equations	yes	yes	no	no
partial differential inequations	no	yes	no	no
ordinary differential equations	yes	yes	yes	no
ordinary differential inequations	no	yes	yes	no
algebraic equations	yes	yes	yes	yes
algebraic inequations	no	yes	yes	no
non-commutative polynomials	no	yes	no	no
fixed ranking	no	no[a]	yes	yes
fixed notation for derivatives	yes[b]	no	yes	n/a[c]
fixed reduction	no	yes	yes	yes

[a] Although the ranking itself is not fixed, only a given set of rankings can be used. This set of possible rankings cannot be extended by the user.

[b] The `CharSet` library itself only accepts Aldor entities as input. With the help of the `ExtIO` library, as illustrated in Section 7.3, also MAPLE's and MATHEMATICA's `FullForm` notation can be used.

[c] `WuRittSolva` does not provide support for differential polynomials at all. Therefore, neither "yes" nor "no" is appropriate for this cell.

Table 12.2: Feature comparison of packages for characteristic set computations.

downside comes from the used language, Aldor. Aldor programs are greedy in terms of memory. Therefore, using `CharSet` also requires a huge amount of memory.

`diffalg` is the most versatile implementation, as it is the only system that can handle commutative and even non-commutative settings. Its implementation is efficient and very robust.

`epsilon` is recommended, when characteristic set computations are used for geometric theorem proving. Its capabilities in creating interactive visualizations are outstanding.

For becoming acquainted with characteristic set computations, `WuRittSolva` can be used. It is very informative to see which elements are introduced in which steps of the various algorithms.

13 Conclusion

This report presented `CharSet` [6], which is a generic implementation for differential characteristic sets in Aldor. `CharSet` provides a wide set of features and is the only known implementation that allows extensions in terms of reductions, orders and implementations for polynomial rings. A wide range of alternative designs has been presented.

The possibilities to connect Aldor code to computer algebra systems have been presented. The used technologies have been explained and difficulties of the various systems have been discussed.

Different implementations of differential polynomial rings have been explained and compared.

Possible future work might be to extend the characteristic set implementation. The current version of `CharSet` does not treat inequations and non-commutative settings. Furthermore, the reduction sub-part can be abstracted further to permit more general settings, like those of [57]. Additionally, differential polynomial rings that mix the recursive and distributive approach may be implemented and evaluated.

Index

List of Algorithms

List of Figures

List of Tables

References

[1] Aldoc. *Internet*. URL `ftp://ftp-sop.inria.fr/cafe/software/aldoc` (last seen on 2005-12-06).

[2] Aldor. *Internet*. URL `http://www.aldor.org/` (last seen on 2005-12-06).

[3] x86-64 version of aldor. *Internet*. URL `http://www.aldor.org/pipermail/aldor-l/2005-June/000072.html` (last seen on 2005-12-06).

[4] AldorUnit. *Internet*. URL `http://www.risc.uni-linz.ac.at/software/aldor/project.php?id=AldorUnit` (last seen on 2005-12-06).

[5] Aldordoc. *Internet*. URL `http://www.aldor.org/projects/aldordoc/index.html` (last seen on 2005-12-06).

[6] CharSet. *Internet*. URL `http://www.risc.uni-linz.ac.at/software/aldor/project.php?id=CharSet` (last seen on 2005-12-06).

[7] Doxygen. *Internet*. URL `http://www.doxygen.org/` (last seen on 2005-12-06).

[8] Emacs. *Internet*. URL `http://www.gnu.org/software/emacs/emacs.html` (last seen on 2005-12-06).

[9] ExtIO. *Internet*. URL `http://www.risc.uni-linz.ac.at/software/aldor/project.php?id=ExtIO` (last seen on 2005-12-06).

[10] GNU Compiler Collection. *Internet*. URL `http://gcc.gnu.org/` (last seen on 2005-12-06).

[11] The GNU Multiple Precision Bignum Library. *Internet*. URL `http://www.swox.com/gmp/` (last seen on 2005-12-06).

[12] GNU Binutils. *Internet*. URL `http://www.gnu.org/software/binutils/` (last seen on 2005-12-06).

[13] GNU/Linux. *Internet*. URL `http://www.gnu.org/` (last seen on 2005-12-06).

[14] JUnit. *Internet*. URL `http://sourceforge.net/projects/junit/` (last seen on 2005-12-06).

[15] About dynamic-link libraries. *Internet*. URL `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dllproc/base/about_dynamic_link_libraries.asp` (last seen on 2005-12-06).

[16] Macsyma. *Internet*. URL `http://www.symbolics.com/Macsyma-1.htm` (last seen on 2005-12-06).

[17] Magma. *Internet*. URL `http://magma.maths.usyd.edu.au/magma/` (last seen on 2005-12-06).

[18] Maple. *Internet*. URL `http://www.maplesoft.com/products/maple/` (last seen on 2005-12-06).

[19] MathLink. *Internet*. URL `http://www.wolfram.com/solutions/mathlink/mathlink.html` (last seen on 2005-12-06).

[20] MathLink Software Developer Kit. *Internet*. URL `http://www.wolfram.com/solutions/mathlink/devkits.html` (last seen on 2005-12-06).

[21] Mathematica. *Internet*. URL `http://www.wolfram.com/products/mathematica/index.html` (last seen on 2005-12-06).

[22] Maxima. *Internet*. URL `http://maxima.sourceforge.net/` (last seen on 2005-12-06).

[23] MockMMA. *Internet*. URL `http://www.cs.berkeley.edu/~fateman/mma1.6.tar.gz` (last seen on 2005-12-06).

[24] System V application binary interface. *Internet*. URL `http://www.caldera.com/developers/gabi/latest/contents.html` (last seen on 2005-12-06).

[25] Unified modeling language. *Internet*. URL `http://www.uml.org/` (last seen on 2005-12-06).

[26] XEmacs. *Internet*. URL `http://www.xemacs.org/` (last seen on 2005-12-06).

[27] Aldor.org. Aldor User Guide. *Internet*. URL `http://www.aldor.org/docs/aldorug.pdf` (last seen on 2005-12-06).

[28] J. Apel and U. Klaus. Representing polynomials in computer algebra systems. In *Proc. New Computer Technologies in Control Systems*, 1994. URL `http://lips.informatik.uni-leipzig.de:80/pub/showDoc.Fulltext/dokument.pdf?lang=de&doc=1994-9&format=pdf&compression=&rank=0` (last seen on 2005-12-06).

[29] O. Bachmann and H. Schönemann. Monomial representations for gröner bases computations. In *ISSAC '98: Proceedings of the 1998 international symposium on Symbolic and algebraic computation*, pages 309–316, New York, NY, USA, 1998. ACM Press. ISBN 1-58113-002-3.

[30] O. Bachmann, H. Schönemann, and S. Gray. Mpp: A framework for distributed polynomial computations. In *ISSAC '96: Proceedings of the 1996 international symposium on Symbolic and algebraic computation*, pages 103–112, New York, NY, USA, 1996. ACM Press. ISBN 0-89791-796-0.

[31] F. Boulier and E. Hubert. diffalg. *Internet*. URL `http://www-sop.inria.fr/cafe/Evelyne.Hubert/diffalg/` (last seen on 2005-12-06).

[32] F. Boulier and F. Lemaire. Computing canonical representatives of regular differential ideals. In *ISSAC '00: Proceedings of the 2000 international symposium on Symbolic and algebraic computation*, pages 38–47, New York, NY, USA, 2000. ACM Press. ISBN 1-58113-218-2.

[33] P. Broadbery and M. Bronstein. A First Course on Aldor with libaldor. *Internet*. URL `http://www.aldor.org/docs/tutorial.pdf` (last seen on 2005-12-06).

[34] L. Dragan. Introduction to FOAM (the intermediate language for Aldor). *Internet*. URL `http://www.orcca.on.ca/~ldragan/aldor/fp/fp.html` (last seen on 2005-12-06).

[35] R. Fateman. Comparing the speed of programs for sparse polynomial multiplication. *ACM SIGSAM Bulletin*, 37(1), 2003. URL `http://www.cs.berkeley.edu/~fateman/papers/fastmult.pdf` (last seen on 2005-12-06).

[36] G.-M. Greuel, G. Pfister, and H. Schönemann. SINGULAR 3.0. A Computer Algebra System for Polynomial Computations, Centre for Computer Algebra, University of Kaiserslautern, 2005. URL `http://www.singular.uni-kl.de` (last seen on 2005-12-06).

[37] R. Hemmecke. aldor-mode for emacs. *Internet*. URL `http://www.hemmecke.de/aldor/` (last seen on 2005-12-06).

[38] E. Hubert. Factorization free decomposition algorithms in differential algebra. *Journal of Symbolic Computation*, 29(4–5):641–662, 2000.

[39] E. Hubert. Notes on triangular sets and triangulation-decomposition algorithms II: Differential systems. In Winkler and Langer [54]. ISBN 3-540-40554-2.

[40] E. Hubert. Notes on triangular sets and triangulation-decomposition algorithms I: Polynomial systems. In Winkler and Langer [54]. ISBN 3-540-40554-2.

[41] E. Hubert. Improvements to a triangulation-decomposition algorithm for ordinary differential systems in higher degree cases. In *ISSAC '04: Proceedings of the 2004 international symposium on Symbolic and algebraic computation*, pages 192–198, New York, NY, USA, 2004. ACM Press. ISBN 1-58113-827-X.

[42] E. Hubert. Differential algebra for derivations with nontrivial commutation rules. *Journal of Pure and Applied Algebra*, 200:163–190, 2005.

[43] Institute of Electrical and Electronics Engineers, Inc and The Open Group. IEEE Standard 1003.1-2004 Standard for Information Technology. *Internet*. URL `http://www.unix.org/single_unix_specification/` (last seen on 2005-12-06).

[44] A. D. Kennedy. Semantics of Categories in Aldor. *Internet*. URL `http://www.ph.ed.ac.uk/~bj/paraldor/WWW/docs/discussion/define.pdf` (last seen on 2005-12-06).

[45] E. R. Kolchin. *Differential Algebra and Algebraic Groups*, volume 54 of *Pure and Applied Mathematics*. Academic Press, 1973.

[46] H.-S. Liu. WuRittSolva: Implementation of Wu-Ritt Characteristic Set Method. *Internet*. URL `http://library.wolfram.com/infocenter/MathSource/5716/` (last seen on 2005-12-06).

[47] M. B. Monagan, K. O. Geddes, K. M. Heal, G. Labahn, S. M. Vorkoetter, J. McCarron, and P. DeMarco. *Maple 9 Advanced Programming Guide*. Maplesoft, 2003.

[48] J. F. Ritt. *Differential Algebra*, volume 33 of *American Mathematical Society Colloquium Publications*. American Mathematical Society, 1950.

[49] TIS Committee. Tool interface standacd (TIS) executable and linking format (ELF) specification. *Internet*. URL `http://x86.ddj.com/ftp/manuals/tools/elf.pdf` (last seen on 2005-12-06).

[50] D. Wang. *Elimination Methods*. Springer, 2001. ISBN 3-211-83241-6.

[51] D. Wang. Epsilon. *Internet*. URL `http://www-calfor.lip6.fr/~wang/epsilon/` (last seen on 2005-12-06).

[52] S. Watt, P. Broadbery, P. Iglio, S. Morrison, and J. Steinbach. Foam: A First Order Abstract Machine, V 0.35. Technical Report IBM Research Report RC 19528, IBM, 1994.

[53] F. Winkler. *Polynomial algorithms in computer algebra*. Springer, 1996. ISBN 3-211-82759-5.

[54] F. Winkler and U. Langer, editors. *Symbolic and Numerical Scientific Computation, Second International Conference, SNSC 2001, Hagenberg, Austria, September 10-11, 2001, Revised Papers*, volume 2630 of *Lecture Notes in Computer Science*, 2003. Springer. ISBN 3-540-40554-2.

[55] S. Wolfram. *The Mathematica Book*. Wolfram Media, 5th edition, 2003. ISBN 1-57955-022-3. URL `http://documents.wolfram.com/mathematica/book/` (last seen on 2005-12-06).

[56] S. Youssef. Prospects for Category Theory in Aldor. *Internet*. URL `http://atlas.bu.edu/~youssef/papers/math/aldor/aldor.pdf` (last seen on 2005-12-06).

[57] M. Zhou and F. Winkler. Gröbner bases in difference-differential modules and their applications. Technical Report 05-06, RISC Report Series, University of Linz, Austria, Octorber 2005. URL `http://apache.risc.uni-linz.ac.at/internals/ActivityDB/publications/download/risc_2734/dd.pdf` (last seen on 2005-12-06).