JOHANNES KEPLER
UNIVERSITÄT LINZ

Eingereicht von
**DI Alexander Maletzky**

Angefertigt am
**Research Institute for Symbolic Computation**

Erstbeurteiler
**Prof. Dr. Dr.h.c.mult. Bruno Buchberger**

Zweitbeurteiler
**Prof. Dr. Martin Kreuzer**

Mai 2016

# Computer-Assisted Exploration of Gröbner Bases Theory in Theorema

Dissertation

zur Erlangung des akademischen Grades

Doktor der Technischen Wissenschaften

im Doktoratsstudium der

Technischen Wissenschaften

**JOHANNES KEPLER UNIVERSITÄT LINZ**
Altenbergerstraße 69
4040 Linz, Österreich
www.jku.at
DVR 0093696

# Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Dissertation selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.
Die vorliegende Dissertation ist mit dem elektronisch übermittelten Textdokument identisch.

Linz, im Mai 2016

Alexander Maletzky

# Kurzfassung

Die vorliegende Dissertation präsentiert die formale, computerunterstützte Exploration der Theorie der Gröbner Basen im mathematischen Assistenzsystem Theorema 2.0. Die Hauptmotivation für diese Arbeit ist unsere Überzeugung, dass eine vollständig formale, verifizierte und sogar exekutierbare Darstellung der Theorie deren zukünftige Erweiterung deutlich vereinfachen kann.

Den Kern der Formalisierung stellen sogenannte Reduktionsringe dar, die vor über 30 Jahren von Buchberger eingeführt wurden. Reduktionsringe verallgemeinern das ursprüngliche “Setting” von Gröbner Basen in Polynomringen über Körpern zu einer viel größeren Klasse algebraischer Strukturen, nämlich im Wesentlichen zu unitären kommutativen Ringen mit ein paar zusätzlichen Funktionen und Relationen, die eine Handvoll nicht-trivialer Axiome erfüllen. Wir haben alle zentralen Aspekte dieser Theorie in Theorema dargestellt, d. h. insbesondere alle wichtigen Definitionen und Sätze, und haben sämtliche Resultate mithilfe der automatischen und interaktiven Beweisfunktionen von Theorema bewiesen. Außerdem haben wir den Buchberger-Algorithmus zur Berechnung von Gröbner Basen in einer komplett generischen und direkt ausführbaren Weise implementiert und die totale Korrektheit dieser Implementierung bewiesen. Das Ergbnis von alledem ist nun als Sammlung von 16 Theorema-Theorien, bestehend aus insgesamt ca. 2240 Formeln, verfügbar, welche als Basis für zukünftige Theorieexplorationen in Theorema dienen kann.

Obwohl die mathematischen Theorien, die wir formalisiert haben, keineswegs neu sind, konnten wir dennoch auch direkt zu ihnen beitragen, und zwar durch die drastische Verinfachung eines Beweises, die Verallgemeinerung von verschiedenen Definitionen und Resultaten, und, am allerwichtigsten, durch das Korrigieren eines kleinen Fehlers in der Reduktionsringtheorie. Das alles unterstützt definitiv die Behauptung, Mathematik würde davon profitieren, formal in einem Computersystem behandelt zu werden, und ist deshalb eine zusätzliche Motivation für unsere Arbeit im Speziellen und für computerunterstützte Theorieexploration im Allgemeinen.

Schließlich berichtet die Dissertation auch über vier neue Werkzeuge in Theorema, die wir im Rahmen unserer Arbeit entwickelt haben: einen Rewriting-Mechanismus für Regeln erster- und höherer Ordnung, eine Sammlung von Inferenzregeln für allgemeine Prädikatenlogik, eine Beweisstrategie für interaktives Beweisen, und eine Sammlung von Funktionen zur Analyse der logischen Struktur von Theorema-Theorien. Jedes dieser Werkzeuge hat sich bereits als äußerst hilfreich in der Formalisierung der Gröbner Basen Theorie herausgestellt und wird deshalb sicherlich auch in zukünftigen Theorieexplorationen in Theorema Anwendung finden.

# Abstract

This thesis presents the formal, computer-supported exploration of the theory of Gröbner bases in the mathematical assistant system Theorema 2.0. The main motivation for this work is our conviction that a fully formal, verified and even executable representation of the theory has the potential to facilitate its further expansion in the future.

The core component of the computer-formalization are so-called reduction rings, introduced by Buchberger more than 30 years ago. Reduction rings generalize the original setting of Gröbner bases in polynomial rings over fields to a much wider class of algebraic structures, namely essentially commutative rings with multiplicative identity and a couple of further functions and relations, satisfying a handful of non-trivial axioms. We represented the central aspects of this theory in Theorema, including all the main definitions and theorems, and proved the results correct using the automated- and interactive proving facilities of Theorema. Moreover, we also implemented Buchberger's algorithm for actually computing Gröbner bases in a completely generic and directly executable manner and proved it totally correct. The result of all this is now available as a collection of 16 Theorema theories, consisting in total of roughly 2240 formulas, that may serve as the basis for future theory explorations in Theorema; this, in particular, holds for eight theories exclusively dealing with elementary mathematical concepts, such as sets, numbers, tuples, etc., that are themselves absolutely independent of Gröbner bases, reduction rings and Buchberger's algorithm.

Although the mathematical theories we considered for formalization are by no means novel, we nevertheless managed to contribute to them as well, by (drastically) simplifying one proof, generalizing various definitions and results, and, most importantly, correcting a subtle error in the theory of reduction rings. This definitely gives evidence to the claim that mathematics profits from being treated formally in software systems and hence constitutes another motivation for our work in particular and for computer-assisted theory exploration in general.

Finally, the thesis also reports on four tools in Theorema we developed in the frame of our studies: a powerful first- and higher-order rewriting mechanism, a set of inference rules for general predicate logic, a versatile proof strategy for interactive proving, and a package of functions for analyzing the logical structure of one or more Theorema theories. Each of these four tools already proved extremely useful during our formal treatment of Gröbner bases theory and will certainly aid future theory explorations in Theorema as well.

# Acknowledgments

First and foremost I thank my doctoral adviser Bruno Buchberger for giving me the opportunity to do my studies in his Theorema group at RISC, and of course for many interesting and motivating discussions about Gröbner bases, formal mathematics and Theorema.

I also thank my colleagues at RISC for the creating the friendly and inspiring atmosphere I enjoyed. Special thanks go to Wolfgang Windsteiger, the head developer of Theorema, for his patient assistance in all questions related to the system, and to Temur Kutsia for proofreading the section about higher-order rewriting.

I thank Tobias Nipkow and his group at the Technical University Munich for helping me getting acquainted with the Isabelle proof assistant and showing me a different viewpoint on formal mathematics during my stay there. I also thank Martin Kreuzer for reviewing this thesis and serving as the second examiner.

And, finally, I am grateful to my family for their constant support over the years that allowed me to pursue my studies in the first place.

This thesis is based upon research funded by the Austrian Science Fund (FWF): grant no. W1214-N15, project DK1

# Contents

# Chapter 1

# Introduction

The work presented in this thesis belongs to the intersection of mathematics and computer science: it is concerned with the formal, computer-supported exploration of a mathematical theory in a mathematical assistant system. The ultimate goal of computer-supported theory exploration in general is representing a substantial part of the whole corpus of mathematics in a *fully formal* and *machine-checked* form and collecting it in extensive structured *knowledge archives* that are *accessible* and *comprehensible* by humans and software systems alike. This is motivated by the convictions of numerous scholars working in this field that a formal, trusted and ideally even executable representation of the constituents of mathematical theories (definitions, theorems, algorithms) bears the prospect of leading to interesting new insights and thus significantly aiding the further expansion of the respective theories. We do not only genuinely share these convictions but even managed to give further evidence to them by means of concrete improvements of existing theories obtained in the frame of our studies, as will be seen.

The mathematical theory we considered for formalization[1] is the theory of *Gröbner bases*, originally invented by Buchberger in [Buc65]. More precisely, because of the enormous size of the theory with hundreds of specializations, generalizations and applications, we could only concentrate on small fragment of it: the analysis of the complexity of Buchberger's algorithm in the bivariate case, and a generic, verified, executable implementation of Buchberger's algorithm in so-called *reduction rings*.

The formal treatment of said theory was carried out in the Theorema mathematical assistant system [BJK$^+$16], originally conceived by Buchberger in the mid-nineties and now developed by his Theorema research group at the Research

[1]Throughout this thesis the meaning of the terms "formalization", "formal treatment" and "theory exploration" is representing mathematical content (definitions, theorems, algorithms, etc.) in software systems.

Institute for Symbolic Computation. In fact, the concrete version of the system used in our studies is the recently (summer 2014) released Theorema 2.0, which, albeit following the same paradigms and design principles as its predecessor version Theorema 1, considerably differs from Theorema 1 in a couple of respects. This is the reason why we included a separate chapter on Theorema 2.0 in this thesis. Anyway, the resulting formalization is now available online from http://www.risc.jku.at/people/amaletzk/Formalizations.html.

Please note that this introductory chapter is deliberately kept rather short, since the subsequent chapters each start with quite thorough introductions to the topics discussed there, including reviews of specific related work.

## 1.1 State-of-the-Art and Related Work

In this section we briefly summarize the current state-of-the-art in formal, computerized mathematics and automated reasoning *in general*. Work related specifically to reduction rings can be found in Section 3.1, and to the formalization of Gröbner bases theory in mathematical assistant systems in Section 4.1.1. Still, we explicitly want to point out already here that the theory of reduction rings our formalization is based upon has never been considered in any other mathematical assistant system before—at least we are not aware of any.

Among the most widely used mathematical assistant systems are, in alphabetical order, ACL2 [KMM00], Coq [BC04], the members of the HOL family (e. g. HOL Light [Har96]), Isabelle (in particular Isabelle/HOL) [NPW02, Wen16], Mizar [BBG$^+$15] and NuPRL [C$^+$85]; see also [Wie06] for a qualitative comparison of these and other systems. Most of the systems listed here not only support isolated theorem proving, i. e. where individual conjectures are entered into the system one after the other and then proved or disproved (automatically or interactively), but also the systematic development and structured, formal representation of whole mathematical theories in digital knowledge bases. Examples of such knowledge bases are the Mizar Mathematical Library[2] currently containing roughly 1250 articles, Isabelle's Archive of Formal Proofs[3] and The Coq Users' Contributions[4], each covering a wide range of theories from basically all areas of mathematics. Furthermore, HOL Light and Isabelle were heavily involved in the recently finished formal proof of the Kepler conjecture [H$^+$15], and Coq in the formal proof of the Feit-Thompson theorem [G$^+$13].

Although a digital knowledge base in the spirit of the aforementioned examples does not exist in Theorema (yet), there are nevertheless some remark-

[2]http://mmlquery.mizar.org/
[3]http://afp.sourceforge.net/
[4] http://www.lix.polytechnique.fr/coq/pylons/contribs/index

able achievements made using Theorema. For instance, Buchberger and Craciun [Buc04b, Cra08] managed to automatically synthesize Buchberger's algorithm from its specification using the novel "lazy thinking" approach, and Rosenkranz [Ros05] developed a novel algebraic method for solving linear boundary value problems. More recently, Windsteiger [LCK$^+$13] formalized the theory of so-called *Vickrey auctions* from theoretical economics in Theorema 2.0, marking the first real application of the new version of the system.

## 1.2 Summary of Contributions

Our work can be divided into five parts, contributing to three different levels of computer-assisted theory exploration.

First of all, we formalized and formally verified the complexity analysis of Buchberger's algorithm in the bivariate case in Theorema, following his original elaboration in [BW79, Buc83c]. As soon as this was finished, we turned to the formalization of the theory of reduction rings and the implementation and verification of Buchberger's algorithm in this far more general setting, according to [Buc83a, Sti88]. Not very surprisingly, this necessitated the representation of a range of elementary mathematical theories independent of reduction rings, e. g. set theory, some algebraic structures, numbers, tuples, etc., in Theorema as well. Therefore, we also took the effort of formalizing these theories, resulting in stand-alone components that may now serve as solid foundations for other theory explorations in Theorema in the future. Just to make this point very clear we want to highlight that every single result in each of these three major formalizations was of course proved formally with Theorema.

The three contributions mentioned above are all on the *formalization* level. However, it is important to note that we could contribute to the *theory* level itself as well, in the sense that during our work we managed to drastically simplify a proof and generalize some definitions and results in connection with the complexity analysis, and even to find and correct subtle errors in the literature on reduction rings. These improvements are explained thoroughly in the subsequent chapters.

Finally, in addition to the theory- and formalization levels, we also worked on the *system* level, i. e. on Theorema itself, by developing a couple of useful tools of general interest, i. e. not specifically related to particular theories: a mechanism for first- and higher-order rewriting, a collection of general predicate logic inference rules combined in a separate Theorema prover, a proof strategy for interactive proving, and a package for analyzing the logical structure of Theorema theories. Interestingly, none of these four tools was on our agenda at the beginning of our studies but all of them turned out to be highly desirable only during our work.

Summarizing, our contributions are the following:

1. formalizing the complexity analysis of Buchberger's algorithm in the bivariate case,
2. formalizing a considerable amount of the theory of reduction rings,
3. formalizing hundreds of results related to basic mathematical concepts such as sets, numbers, etc.,
4. slightly improving the formalized theories by simplifications, generalizations and even minor corrections, and
5. implementing four useful tools for enhancing the Theorema system.

To the best of our knowledge we are not aware of any other research in connection with the first or second item in the above list. So far, Gröbner bases theory has only been considered for formal treatment in its most basic form (without any advanced applications), solely in polynomial rings over fields; see Section 4.1.1 for more details. Although the tools mentioned in the fifth item above are integral parts of most other mathematical assistant systems, an environment for interactive proving has already been available in Theorema 1, and a prover for general predicate logic existed in Theorema 2.0 already, we still want to emphasize that the mechanism for higher-order rewriting and the package for analyzing theories are completely novel in the frame of the Theorema project.

## 1.3 Organization of the Thesis

The remainder of this thesis is organized as follows. In Chapter 2 we briefly summarize the most important features and main design principles of the Theorema system, in particular of Theorema 2.0. In addition, we explain the syntax and semantics of constructs and notions part of the Theorema language that appear frequently in the thesis. In Chapter 3 we thoroughly review the theory of reduction rings and Gröbner bases by stating essentially all definitions and results. Moreover, we explicitly point out our own contributions to the theory and discuss why they were necessary. Afterward, in Chapter 4, we present the main part of our work: the formalization of various elementary mathematical theories as well as reduction ring theory, including a generic, verified and directly executable implementation of Buchberger's algorithm, in Theorema. To that end, we describe the coarse structure and size of the formalization and have a closer look at its individual components. In Chapter 5 we give account of another formalization, but this time not related to reduction rings, but to the complexity analysis of Buchberger's algorithm in the bivariate case (in polynomial rings over fields). Then, in Chapter 6 we turn our attention away from working *with* Theorema to working *on* Theorema by presenting the four new tools we developed in the frame of our studies and which already proved to be extremely helpful. In Chapter 7, finally, we conclude the thesis by giving a short summary of our results and findings and listing directions for potential future work.

Appendix A contains a sample Theorema-proof of a simple theorem in our reduction-ring formalization for illustrating how such proofs typically look like.

# Chapter 2

# Overview of Theorema

Theorema[1] [BCJ$^+$06, Win14, BJK$^+$16] is a mathematical assistant system, supporting the user in many aspects of his everyday mathematical work. The Theorema project was initiated in 1995 by Bruno Buchberger, who has served as its leader ever since.

From its outset, Theorema was designed as a system for *natural-style* mathematics. This means that users of the system should not have to get acquainted to a completely new language or syntax, a completely new way of proving, or even a completely new style of doing mathematics first. Instead, the syntax of Theorema closely resembles the one found in usual mathematical text-books, with all sorts of Unicode characters, two-dimensional syntax (subscripts, underscripts, matrix-arrangements) etc. that are the basic ingredients of the rich and expressive language of mathematics. Moreover, proofs generated by Theorema cannot only be inspected as abstract, hardly readable data structures where lots of fantasy might be needed to guess the meaning of certain constructs, but instead they are displayed in nicely-formatted, structured *proof documents*, where informal explanatory English (or whatever language) text is interspersed with formal content; see Appendix A for a sample proof.

Another design goal in the development of Theorema was, and still is, the integration of *all* facets of mathematical theory exploration in one coherent software system: inventing new problems, introducing new concepts, designing and implementing algorithms, performing computations, making conjectures, proving theorems, and even disseminating results. In short, Theorema advocates the systematic development and formalization of theories instead of only isolated theorem proving. Nevertheless, automated (and also interactive) mechanical proving constitutes one of the core components of the system and is briefly looked at separately in Section 2.3.

[1] http://www.risc.jku.at/research/theorema/software/

In 2010, for various reasons described in [BJK$^+$16], the Theorema system was re-implemented from scratch, leading to version Theorema 2.0. The main design goals did not change, of course, and as its predecessor version it is still based on *Mathematica* [Wol]. The most evident difference between Theorema 1 and Theorema 2.0, and actually one of the very reasons for developing Theorema 2.0 at all, is perhaps the improved user interface: unlike before, user interactions (inserting new formulas, performing computations, initiating proofs, etc.) almost exclusively happen through a graphical user interface rather than a command-oriented mode, making the system easier to handle and thus more attractive for end-users. Since all our work was carried out entirely in the new version of the system, the term "Theorema" in the sequel always refers to Theorema 2.0 unless explicitly stated otherwise.

One point needs to be addressed explicitly: although Theorema is based on *Mathematica*, i. e. programmed in the *Mathematica* programming language and distributed as a *Mathematica* package, no appeal to *Mathematica*'s huge algorithm library and multitude of simplification rules is made by default. Instead, content input into Theorema is somehow "shielded" from unwanted evaluation that usually happens in *Mathematica*, giving the user full control over how the input shall be processed further. For example, the user can explicitly specify the set of simplification rules to be used in computations and proofs.

Before we present some further details of the syntax and semantics of Theorema below, some words on notation and typesetting in this thesis are in place: in the rest of this chapter and throughout this thesis, Theorema constants that are no number literals are printed in `typewriter` font, whereas variables are printed in $italics$. Theorema formulas are usually displayed in separate boxes with light-gray background, as in[2]

$$\underset{m\in\mathbb{N}}{\forall}\ \underset{n\in\mathbb{N}}{\exists}\ n \geq m \wedge \mathtt{isPrime}[n-1] \wedge \mathtt{isPrime}[n+1] \qquad (\textit{prime twins})$$

In Chapter 4, sans serif font refers to Theorema theories (usually, such a theory is also suffixed by .nb, indicating that it actually is a Theorema, or, more precisely, *Mathematica* notebook).

## 2.1 Syntax and Semantics

In this section, we review the most important syntactical details of the Theorema language; a thorough description of all aspects of the syntax of Theorema goes

[2]Formula (*prime twins*) has nothing to do with our work, of course, but was merely chosen for demonstration purposes.

beyond the scope of this thesis, though.

First and foremost, the language of Theorema is essentially a variant of *untyped higher-order predicate logic with sequence variables*, where function application is denoted by square brackets ($\mathtt{foo}[x]$ instead of $\mathtt{foo}(x)$). Sequence variables [KB04] are variables that may not only be instantiated by single terms, but by sequences of terms of arbitrary finite length (possibly even the empty sequence); in Theorema, sequence variables are always suffixed by three dots (e. g. $x...$).

Note that only the *language* of Theorema is that of untyped higher-order logic. The *semantics* of an expression is solely determined by the rewrite rules (e. g. simplification rules for computation, or inference rules for logical reasoning) currently available in a theory exploration. In principle, users of Theorema have a lot of freedom to compose these rules according to their taste and needs, for working, say, in Zermelo-Fraenkel set theory, (typed or untyped) higher-order logic, constructive type theory, or whatever. It is clear, though, that Theorema provides a standard logical environment by default, which fits the language of Theorema and is more or less untyped predicate logic—first-order, for the most part (where, e. g., the application of a universally quantified variable $f$ to some arguments is interpreted just as syntactic application). In our work, we stuck to these standards, only adding a bit of higher-order logic, namely higher-order rewriting (where the application of universally quantified $f$ to arguments is *not* just interpreted syntactically; see Section 6.1), to the inferencing facilities.

In the rest of this section we discuss the various syntax elements of Theorema that occur in this thesis.

**Propositional logic.** The syntax of propositional logic in Theorema adheres to traditional conventions. Only note that conjunctions and disjunctions may be written "vertically" to save space, as

$$\bigwedge\begin{cases}\varphi_1\\ \varphi_2\\ \vdots\\ \varphi_n\end{cases}\qquad\qquad \bigvee\begin{cases}\varphi_1\\ \varphi_2\\ \vdots\\ \varphi_n\end{cases}$$

**Binders.** The syntax of variable binders, such as logical quantifiers, sums and products, and set- and tuple abstractions, resembles that of ordinary mathematics. Variables bound by such constructs, however, are always written *under* the respective binder, e. g.

$$\underset{x,y}{\forall}\ \varphi \qquad \underset{P[z]}{\exists}\ \varphi \qquad \sum_{i=1,\dots,n} t \qquad \{t \underset{a\in A}{\mid} \varphi\}$$

These four examples also illustrate the four types of *variable ranges* available in Theorema: the range of the universal quantifier is unrestricted, meaning that $x$ and $y$ can be completely arbitrary; $z$ ranges over objects satisfying the unary predicate $P$; $i$ ranges over integers between $1$ and $n$ (both inclusive); $a$ ranges over elements of the set $A$. It is possible to bind more than one variable at once, as in the first example above. For instance, the range $a, b, c \in A$ binds variables $a$, $b$ and $c$, all of which are restricted to elements of $A$. In connection with predicate-ranges, as in the second example, one must be careful to observe that the given predicates are always considered *unary*, i. e.

$$\underset{P[x,y,z]}{\forall} \varphi \quad \equiv \quad \underset{P[x],P[y],P[z]}{\forall} \varphi$$

which is particularly convenient in connection with *domain decision predicates* restricting the "type" of variables, see Section 2.2 and also Remark 1.

Of course, binders and ranges can be combined arbitrarily. Moreover, it is possible to impose further constraints on the instances of bound variables by adding conditions under the respective variable ranges, e. g.

$$\underset{\substack{a,b,c\in\mathbb{Z} \\ a^3+b^3=c^3}}{\forall} \varphi \qquad\qquad \sum_{\substack{i=2,\ldots,1000 \\ \mathtt{isPrime}[i]}} t$$

**Tuples.** Tuples are finite ordered lists of *arbitrary* elements, i. e. one and the same tuple may contain elements of completely different kinds (numbers, sets, other tuples, etc.); they are characterized by the unary predicate `isTuple`. The following notation related to tuples is used in this thesis:

- $|T|$ denotes the length of tuple $T$.
- $T_i$ denotes the $i$-th element of tuple $T$; if $i$ is not a natural number between $1$ and $|T|$, $T_i$ is undefined.
- $\langle x, y, \ldots \rangle$ denotes the tuple whose first element is $x$, second element is $y$, etc.
- Similar to set abstractions, there are also tuple abstractions $\langle t \underset{i=a,\ldots,b}{|} \varphi \rangle$ whose meaning should be obvious; note, however, that only *step ranges* are allowed here, because tuples are ordered and only step ranges entail a natural ordering on their elements.
- $T \curvearrowleft a$ denotes the tuple $T$ with $a$ appended at the end.
- $a \curvearrowright T$ denotes the tuple $T$ with $a$ prepended at the beginning.

- $S \bowtie T$ denotes the concatenation of tuples $S$ and $T$.
- $a \,\mathrm{E}\, T$ asserts that $a$ is contained in tuple $T$, i. e. that $a = T_i$ for some $i$.

**Number intervals.** In Theorema, the various sets of numbers are as usual denoted by $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, etc. By definition, $\mathbb{N}$ does *not* include 0; if 0 shall be included, $\mathbb{N}_0$ is used instead.

In addition, Theorema also provides a simple means for constructing number intervals: to that end, the respective set-symbol is sub-scripted by some kind of step range of the form $a, \ldots, b$, where $a$ is the lower boundary of the interval and $b$ is the upper one (both included); for instance, $\mathbb{Z}_{-4,\ldots,10}$ denotes the set of all integers between $-4$ and $10$. If intervals shall not be bounded from below and/or above, $-\infty$ (resp. $\infty$) may be used as the respective interval endpoint. In that sense, $\mathbb{Z}$ is a mere shorthand for $\mathbb{Z}_{-\infty,\ldots,\infty}$, and $\mathbb{N}$ for $\mathbb{Z}_{1,\ldots,\infty}$.

## 2.2 Functors and Domains in Theorema

Starting from its very beginnings, Theorema advocates the use of so-called *functors* (in the Theorema sense, see below) as the method of choice for building hierarchies of domains in a generic and nonetheless concise and structured way [Buc96, Buc03, Win99]. Because we employed functors and domains in our formalization of reduction ring theory (see Chapter 4), and because of their importance in general, we present the main ideas behind functors and domains and how they are actually used in Theorema in this section.

Roughly, a Theorema-functor is a function that maps domains (and/or other mathematical objects) to domains. A domain in Theorema, in turn, is essentially a function that *interprets* constant- and function symbols by mapping them to concrete constants and functions.[3] For instance, the most prominent examples of domains in our formalization of reduction ring theory are the various reduction rings (fields, integers, integer quotient rings, etc.) that interpret the reduction-ring-related functions $\succ$, `multN`, `mult`, `lcrd` etc., but also the purely ring-theoretic constants and functions $0$, $1$, $+$ and $*$; see Chapters 3 and 4 for more details. Of course, every domain $\mathcal{D}$ also has a "carrier", i. e. a class of objects that are meant to be contained in $\mathcal{D}$ (in some sense). Carriers, however, are not specified by sets, but merely by the domain-interpretations of the symbol $\in$ as unary *domain membership predicates.*

A functor, finally, maps existing domains and/or other arguments to a new domain, i. e. in some sense it "constructs" a new domain from existing ones (pos-

[3]Domains in Theorema must not be confused with domains in the algebraic sense, where they are assumed to be rings without zero divisors.

sibly none). The new domain usually interprets a symbol $\sigma$ in terms of the interpretation of $\sigma$ in the argument domains it depends upon.

To illustrate the abstract description of domains and functors in a concrete example, consider the product `Lex` of two ordered domains $\mathcal{A}$ and $\mathcal{B}$, ordered lexicographically; this domain is actually also included in our reduction-ring-formalization. In Theorema 2.0, `Lex` would be defined by the following functor:

$$
\begin{array}{l}
\mathtt{Lex}[\mathcal{A},\mathcal{B}] \;:=\; \underset{N}{\Delta} \\
\underset{x1,y1,x2,y2,z}{\forall} \\
\qquad \underset{N}{\in}[z] \;:\Leftrightarrow\; \bigwedge \begin{cases} \mathtt{isTuple}[z] \\ \quad |z| = 2 \\ \quad \underset{\mathcal{A}}{\in}[z_1] \\ \quad \underset{\mathcal{B}}{\in}[z_2] \end{cases} \qquad\qquad (\in \mathcal{L}) \\
\qquad \langle x1, y1\rangle \underset{N}{\succ} \langle x2, y2\rangle \;:\Leftrightarrow\; \bigvee \begin{cases} x1 \underset{\mathcal{A}}{\succ} x2 \\ x1 = x2 \wedge y1 \underset{\mathcal{B}}{\succ} y2 \end{cases} \qquad\qquad (\succ \mathcal{L})
\end{array}
$$

Let us shed some light one the syntax and semantics of the Theorema functor:

- The first two lines are a so-called *global declarations*. They bind some variables (in this case, once $N$ and once $x1, x2, y1, y2, z$) that are bound in all subsequent formulas in the scope of the global declarations. $N$ is bound by a special functor-construct, to be read as "$\mathtt{Lex}[\mathcal{A},\mathcal{B}]$ is some domain $N$ such that …", whereas the other variables are universally quantified.

  In addition to these two types of global declarations, there are also global implications of the form $A \Rightarrow$ that simply constrain all formulas in their scope by condition $A$, and global abbreviations of the form $\underset{x=t}{\mathtt{let}}$ that allow to abbreviate the term $t$ by variable $x$ in all formulas in their scope. Global declarations are not restricted to definitions of functors, but are a general means for formula-input in Theorema.

- The last two lines are Theorema formulas (in the scope of the preceding global declarations). The first one defines membership in the new domain, the second one the order relation.

- The interpretation of a symbol $\sigma$ in some domain $\mathcal{D}$ is always denoted by $\underset{\mathcal{D}}{\sigma}$. Internally, such an expression is represented by $\mathtt{DomainOperation}[\mathcal{D},\sigma]$, e. g. $x1 \succ x2$ is represented by the curried *Mathematica* expression $\mathtt{DomainOperation}[,\succ\ ][x1, x2]$.

Summarizing, `Lex`$[\mathcal{A}, \mathcal{B}]$ contains precisely those objects that are tuples of length 2, whose first element is contained in domain $\mathcal{A}$ and whose second element is contained in $\mathcal{B}$. The order relation on `Lex`$[\mathcal{A}, \mathcal{B}]$ is the lexicographic combination of the order relations on $\mathcal{A}$ and $\mathcal{B}$.

But how do we know that $\mathcal{A}$ and $\mathcal{B}$ interpret $\succ$? How do we know that they interpret $\in$? The answer is simple: we do not know anything about $\mathcal{A}$ and $\mathcal{B}$. Thus, it may well be that they leave some symbols uninterpreted, which does not cause any trouble at all; it is perfectly fine in Theorema if compound expressions cannot be simplified to atoms, not only in connection with functors. After all, we only need to pose constraints on the argument domains whenever we want to *prove* something about `Lex`. For instance, assuming that both $\mathcal{A}$ and $\mathcal{B}$ interpret $\succ$, and in both cases the resulting relation is a partial ordering, we may prove that the same is true also for `Lex`$[\mathcal{A}, \mathcal{B}]$. Moreover, the variables $x1, x2, y1, y2$ in the `Lex` functor are not restricted to elements of $\mathcal{A}$ and $\mathcal{B}$, respectively, which is due to exactly the same paradigm of allowing compound expressions not to simplify to atoms: if $x1$ or $x2$ is not contained in $\mathcal{A}$ (but even if they are!), the relation $x1 \underset{\mathcal{A}}{\succ} x2$ might not simplify to either `True` or `False`.

*Remark* 1. Abusing notation, the assertion that several objects $x_1, \ldots, x_n$ belong to a domain $\mathcal{D}$ is denoted simply by $\underset{\mathcal{D}}{\in}[x_1, \ldots, x_n]$ throughout this thesis, i. e. $\underset{\mathcal{D}}{\in}[x_1, \ldots, x_n]$ abbreviates $\underset{\mathcal{D}}{\in}[x_1] \wedge \ldots \wedge \underset{\mathcal{D}}{\in}[x_n]$.

## 2.3 Proving in Theorema

Proving in Theorema proceeds by repeatedly reducing *proof situations*, starting from an initial one, to (ideally) simpler ones until either all proof situations are trivial (success) or no more reductions can be carried out (failure). The reduction of proof situations happens by applying *inference rules*, and the overall proof search is guided by *proof strategies*. We briefly describe these three main components below.

**Proof situations.** A proof situation consists of three parts: the goal, the knowledge base, and some additional meta information (available rewrite rules, constants generated during the proof search, etc.). In this thesis, in particular in Section 4.5, proof situations are denoted by sequents $K \vdash \Gamma$ where $K$ is the knowledge base, consisting of a sequence of assumptions, and $\Gamma$ is the goal; note that the goal is always only a single formula.

**Inference rules.** An inference rule transforms a proof situation into an ordered list of new proof situations, ideally "simpler" ones, with the meaning that the logical validity of the original proof situation is implied by the validity of *all* of the new ones; thus, if the list of new situations is empty, the original one is proved. Inference rules are combined in so-called *provers*.

Inference rules have to be implemented on the meta level of Theorema, as *Mathematica* transformation rules, and no appeal to actual Theorema formulas justifying the logical correctness of the rules needs to be made, but the semantics of the logic, so to say, is solely defined by the inference rules themselves. For simple rules from the predicate logic calculus this approach might be feasible, but the philosophy of Theorema is to not only have such simple rules, but to develop *special* inference rules for particular theories (like a decision procedure for Presburger arithmetic, a method for proving geometric theorems by computing Gröbner bases, and many more; see [Buc04a]). True, the current approach makes the integration of advanced rules into the system quite straight-forward, but its drawback is the resulting gap between the formula- and the inference level: inference rules implemented as mere *Mathematica* programs might not adequately reflect the actual formulation of the corresponding object-level theory, bearing the danger of making the logic inconsistent.[4] Although several solutions to overcome this problem exist, and one of them has already been investigated in the context of Theorema 1 in [GB07], the concrete implementation of such a solution in Theorema is still future work.

Of course, the issues discussed above also affected the development of special provers in the frame of our formalizations of reduction ring theory and the complexity analysis of Buchberger's algorithm, as described in Chapters 4 and 5. Therefore, we must emphasize that all the special inference rules that are part of our provers are at least informally justified by object-level formulas, as can be seen when they are explained in detail in Sections 4.5 and 5.3.

*Remark* 2. In Theorema there is no distinction between *inference rules* (defining the semantics of objects), *tactics* (reducing proof situations by applying inference rules) and *tacticals* (combining tactics), as in other proof assistants; everything is subsumed by the kind of inference rules described above. In the future, however, a sharp distinction between these three concepts might become inevitable, especially in connection with the aforementioned inconsistency-issues of special rules: special rules rather being special *tactics* that merely *apply* actual inference rules (maybe even given as object-level formulas) is one of the possible solutions of the problem.

**Proof strategies.** A proof strategy, finally, guides the overall proof search by applying inference rules. It determines *where* to continue in the proof if there

[4]In most cases inference rules are more or less the "direct" translations of object-level formulas, but *in principle* they could be arbitrary.

are several alternatives, *how* to proceed if more than one rule is applicable to a proof situation (apply all, apply only one, etc.), and *when* to abort a proof attempt because of constraints on the total search time and proof depth.

Unlike (special) inference rules, proof strategies are typically rather generic and feasible for exploring any sort of mathematical theory. Actually, in our whole formal treatment of reduction ring theory we used only two different strategies: the default *automatic* one and, most importantly, a novel *interactive* one where it is actually the user who guides the proof search by deciding, at each stage of the proof, how to proceed. The interactive strategy is not part of the official distribution of Theorema 2.0 yet (but is planned to be included in the near future) and explained thoroughly in Section 6.3.

# Chapter 3

# Reduction Rings and Gröbner Bases

In this chapter we present the theory of reduction rings that has been formalized in the frame of this thesis. First, we start with an overview of the theory mainly consisting of the historical background and the motivation for considering reduction rings. Then, we introduce the main notions and concepts of the theory, as well as the axioms that characterize reduction rings. Finally, we present the main results of the theory.

Please note that small parts of this chapter are also contained in [Mal15b].

## 3.1 Overview

Reduction rings were first introduced by Buchberger in 1983 [Buc83a, Buc84] and later generalized by Stifter in the late 1980s [Sti85, Sti88, Sti91, Sti93]. In a nutshell, they are structures where an algorithmic approach to Gröbner bases is possible, and hence generalize the – what we will refer to as *original setting* throughout this chapter – polynomial rings over fields Gröbner bases were originally invented for [Buc65, Buc70]. Due to the numerous applications of Gröbner bases not only in computer algebra, but also many other scientific and engineering disciplines, it is apparent that a generalization to a wider class of algebraic structures than just (commutative) polynomial rings over fields is highly desirable. And indeed, apart from the reduction ring approach that we consider in this thesis there are many other approaches as well, listed in Section 3.1.1. One of the most significant differences between these approaches and reduction rings is that the former only consider polynomial rings, or at least require some kind of grading, whereas the crucial idea behind reduction rings is precisely *not* to require any polynomial structure or grading in the first place, as will be explained in the subsequent sections. Instead, polynomial rings and their coefficient domains can be treated by a *uniform* methodology, or, stated differently, polynomial rings over reduction rings *inherit* the property of being a reduction ring from their coefficient domain (this property is "preserved"): if $\mathcal{R}$ is a reduction ring, then Gröbner bases can be computed in $\mathcal{R}$, and moreover $\mathcal{R}[x_1, \ldots, x_n]$ (and also other structures, like $\mathcal{R}^k$) can be turned into a reduction ring as well.

The fundamental idea behind reduction rings was the generalization of the notion of "S-polynomial" or, equivalently, "critical pair", to rings without any grading: in such rings it is not possible to decompose the elements into a "leading part" and into a "rest", which is crucial in the original setting. Hence, the central concept in reduction rings is that of a *non-trivial common reducible* that will be introduced in Definition 3.2.16. Intuitively, a non-trivial common reducible of two ring elements can be reduced modulo both elements (*reduction* in reduction rings directly generalizes reduction in the original setting, see Definition 3.2.2) but only in a *non-trivial* way, in some sense. Quite some abstraction is necessary for extracting the characteristic properties of S-polynomials and critical pairs in the original setting that encompass precisely this kind of non-triviality.

Anyway, all notions defined in reduction rings (in particular also non-trivial common reducibles) depend on a couple of additional parameters, like a partial Noetherian order relation. For a fixed commutative ring with identity $\mathcal{R}$, different choices of these parameters may be feasible for turning $\mathcal{R}$ into a reduction ring, i. e. making $\mathcal{R}$ together with the parameters satisfy certain axioms (see Section 3.3). As noted in [Buc83a], these axioms should both be strong and weak at the same time: strong axioms facilitate the proofs of theorems in reduction rings, like Theorem 3.5.3, whereas weak axioms allow for a wider class of algebraic structures to be turned into reduction rings. Moreover, when proving that the property of being a reduction ring is preserved by certain "functors" (e. g. when moving from $\mathcal{R}$ to $\mathcal{R}[X]$), strong axioms are desirable in the premise, whereas weak axioms are desirable in the conclusion. According to [Buc83a] finding a good balance in the axioms took quite some attempts.

Although the intuition behind most of the notions and axioms is rather clear in the case of integral domains, things become much more technical and complicated if the rings in question are allowed to contain also zero divisors; this, of course, also has an effect on the resulting theorems and proofs. Nevertheless, in our formalization we decided that the reward of being able to treat also non-integral domains was worth the price of increased technicalities, so in the rest of this chapter we never assume that the ring we are talking about is free of zero divisors. Actually, these technicalities were one of the main motivations for the formal treatment of the theory in Theorema: delegating some of the technically demanding but conceptually simple tasks to the system for being taken care of either in a fully automatic, or at least interactive, manner clearly is a great advantage. Future work on the theory, which we believe is possible and even has the potential to lead to interesting new insights (see Chapter 7), may thus also benefit from our formalization.

Reduction rings generalize the original setting in the sense that Gröbner bases can be defined and computed (for a given *finite* ideal basis), and hence can be used to solve various problems related to ideals, like the membership- and congruence problem and the ideal equality problem. In addition, the well-known *elimination property* of Gröbner bases in the original setting (see, e. g., [KR00] page 195), as well as the possibility to compute Gröbner bases of syzygy-modules (see, e. g., [KR00], page 148), translate more or less one-to-one to the reduction ring setting. All this is described in detail in Section 3.5.

Although reduction ring theory was invented more than 30 years ago and we did not make any major contributions to it in the frame of our thesis, it is nevertheless presented in almost full detail[1] in the subsequent sections, which, in our opinion, is justified by the following three reasons:

1. A substantial part of this thesis is about the formalization of reduction ring theory. Hence, in order to make it as self-contained as possible, the theoretical background of reduction rings is included as well.

2. No uniform presentation of the theory in the literature exists so far. After their introduction in [Buc83a], reduction rings were slightly generalized and extended first in [Sti88] and then in [Sti91], but these latter two articles merely focus on the novel aspects and do not really draw a comprehensive picture of all of the theory.

3. We made *some* contributions to the theory itself that have not been documented yet, and also slightly deviate from the existing literature in that we distinguish between reduction rings and *algorithmic* reduction rings.

The contributions to reduction rings that are mentioned above will explicitly be pointed at as soon as they become visible in the presentation of the theory in the sections below. Nevertheless, we already hint at them now, such that the reader familiar with reduction rings knows what to expect:

- The definition of *irrelativity*, as introduced in [Sti88], turned out to be erroneous. We fixed it and complemented irrelativity by the new notion of *correlativity*.

- We introduced an equivalence relation in reduction rings that allows us to make fields algorithmic reduction rings as well.

[1]Most proofs are omitted, though.

- We distinguish between reduction rings and *algorithmic* reduction rings, the latter being a subclass of the former. In addition to the purely algebraic axioms of plain reduction rings, algorithmic reduction rings also need to satisfy axioms that ensure the (relative) computability of certain functions (most notably, Buchberger's algorithm for computing Gröbner bases).

Below, proofs of lemmas and theorems about irrelativity, correlativity and the equivalence relation are spelled out in detail, since they cannot be found anywhere in the literature. For all other proofs, however, we only refer to the articles and reports they are contained in, possibly pointing out some minor differences that may arise due to our revised/new definitions. It must be noted, however, that *every single result* in the theory was proved formally in Theorema and is now part of our formalization, where its proof can be inspected in a nicely-formatted, natural-style proof document.

### 3.1.1 Other Generalizations

There are many other approaches to generalizing Gröbner bases from polynomial rings over fields to wider classes of algebraic structures, both commutative and non-commutative. To the best of our knowledge, however, they all differ from the reduction ring approach in that they only consider polynomial rings (or at least rings some other form of grading, see e. g. [Rob85]), whereas, as mentioned above, the key idea behind reduction rings is precisely *not* to require any polynomial structure in the first place.

The following is a (non-exhaustive) list of commutative generalizations of Gröbner bases to rings of the from $A[X]$:

- The case $A = \mathbb{Z}$ is considered, for instance, by Lauer [Lau76] and Ayoub [Ayo83].
- The case of $A$ being a Euclidean domain is considered by Kandri-Rody and Kapur [KRK88].
- The case of $A$ being a principal ideal domain is considered by Pan [Pan88].
- Pauer [Pau07] and Francis and Dukkipati [FD14] treat the case where $A$ is a Noetherian ring, e. g. a polynomial ring.
- The case where $A$ is a boolean ring is dealt with by Sato *et al.* [SIS$^+$11].
- Spear [Spe77], Trinks [Tri78], Zacharias [Zac78] and Schaller [Sch79] assume that $A$ satisfies certain computability conditions, like decidability of ideal membership and solvability of systems of linear equations by means of finite sets of generators.

Besides commutative polynomial rings, various authors generalized Gröbner bases to non-commutative algebras (as before, the list is non-exhaustive):

- Bergman [Ber78] generalizes the setting to associative $A$-algebras, where $A$ is only assumed to be a commutative ring with identity.
- Rings of differential operators (e. g. Weyl algebras) are studied by Galligo [Gal85] and Insa and Pauer [IP98].
- Mora [Mor86, Mor94], Kandri-Rody and Weispfenning [KRW90], Weispfenning [Wei92] and Levandovskyy [Lev05] consider non-commutative polynomial rings over fields.
- Reinert [Rei06] generalized the setting a little further by dealing with function rings over fields.
- Non-commutative polynomial rings over rings, finally, are studied by Mialebama Bouesso and Sow [MBS15].

Although reduction ring theory subsumes most of the commutative generalizations of Gröbner bases, it cannot handle non-commutative rings so far.

### 3.1.2 Notation

Throughout the whole chapter, and also Chapter 4, we use the following notation:

- If $B \subseteq \mathcal{R}$ and $\mathcal{R}$ is a ring, then $\mathrm{ideal}(B)$ denotes the ideal generated by $B$ over $\mathcal{R}$. If $B$ is finite, $B = \{b_1, \ldots, b_n\}$, then we also denote the ideal generated by $B$ by $\mathrm{ideal}(b_1, \ldots, b_n)$, omitting the set braces.
- $\equiv_B$ denotes the ideal congruence relation modulo the ideal generated by $B$, i. e. $x \equiv_B y \;:\Leftrightarrow\; x - y \in \mathrm{ideal}(B)$.
- For $n \in \mathbb{N}$, $\mathbb{Z}_n$ abbreviates the quotient ring $\mathbb{Z}/n\,\mathbb{Z}$.

## 3.2 Auxiliary Notions

First and foremost, reduction rings need to be *commutative rings with identity*. Hence, let for the remainder of this chapter $(\mathcal{R}, +, \cdot, 0, 1)$ be such a commutative ring with identity; neither does it need to possess any polynomial structure or grading, nor does it have to be an integral domain.

The following typed variables will be used throughout this chapter:

- $a, b, c, d, m, n$ (possibly sub-scripted) for elements of $\mathcal{R}$, and
- $C, G$ for subsets of $\mathcal{R}$.

In order for $(\mathcal{R}, +, \cdot, 0, 1)$ to be turned into a reduction ring, it also has to be endowed with

- a *partial Noetherian (i. e. well-founded)* order relation $\preceq \subseteq \mathcal{R} \times \mathcal{R}$,
- for every $c$ an arbitrary *finite* set $I_c$, and
- for every $c$ and $i \in I_c$ a set $M_c^i \subseteq \mathcal{R}$.

$\preceq$, $I$ and $M$ are the basic ingredients of reduction rings. All other auxiliary notions that are introduced in this section are defined solely in terms of these three objects (and the ring constants $+$, $\cdot$, $0$ and $1$, of course). In particular, the set $M_c$ is defined as $M_c := \bigcup_{i \in I_c} M_c^i$ and is called the *set of multipliers* of $c$, i. e. it consists of those elements $c$ may be multiplied with when reducing another element (see Definition 3.2.2). The splitting of $M_c$ into the $M_c^i$ is merely for technical reasons that will become clearer in Section 3.2.3.

**Comparison to the original setting.** In the original setting, the order relation $\preceq$ is just the polynomial ordering induced by an arbitrary term ordering. The $M_c$ are identical for all polynomials $c$ and just the set of all proper monomials. ■

### 3.2.1 Reduction

We now turn to one of the most important concepts in reduction rings, namely that of reduction:

**Definition 3.2.1** (Reduction using Multipliers)**.** An element $a$ is said to be *reducible* to some element $b$ modulo $c$ using $m$, written as $a \rightarrow_{m,c} b$, iff $b \prec a$, $m \in M_c$ and $b = a - m\,c$.
$a$ is reducible modulo $c$ using $m$, written as $a \rightarrow_{m,c}$, iff there exists $b$ with $a \rightarrow_{m,c} b$.

**Definition 3.2.2** (Reduction Relation)**.** An element $a$ is said to be reducible to some element $b$ modulo the set $C$, written as $a \rightarrow_C b$, iff $a \rightarrow_{m,c} b$ for some $c \in C$, $m \in M_c$. The binary relation $\rightarrow_C$ is called the *reduction relation induced by $C$*; if $C$ is clear from the context, the "induced by $C$" will be omitted.

*Remark* 3. Since $\preceq$ is Noetherian, i. e. well-founded, and apparently $\rightarrow_C \subseteq \succ$ for every $C$, every chain of reductions $a_1 \rightarrow_C a_2 \rightarrow_C \ldots$ is necessarily finite. This important property of reduction is the prerequisite for an algorithmic treatment of the theory, as it ensures termination of various procedures.

**Comparison to the original setting.** Reduction in reduction rings is defined completely analogously to reduction in the original setting (note again that there the $M_c$ are simply the set of monomials). ■

Now that we have introduced the basic reduction relation, there are still a couple of other notions related to reduction we must introduce.

**Definition 3.2.3** (Irreducibility and Normal Forms). An element $a$ is said to be reducible modulo $C$, written as $a \rightarrow_C$, iff there exists some $b$ with $a \rightarrow_C b$; otherwise, it is said to be *irreducible*. An element $d$ is said to be a *normal form* of $a$ modulo $C$ iff $a \rightarrow_C^* d$ and $d$ is irreducible modulo $C$.

**Definition 3.2.4** (Closures of Reduction, Common Successor). As usual, $\leftrightarrow_C$, $\rightarrow_C^*$ and $\leftrightarrow_C^*$ denote the symmetric-, the reflexive-transitive-, and the reflexive-symmetric-transitive closure of $\rightarrow_C$, respectively. $a \downarrow_C^* b$ holds iff $a$ and $b$ have a *common successor*, i. e. there exists some $s$ with $a \rightarrow_C^* s$ and $b \rightarrow_C^* s$.

**Definition 3.2.5** (Connectibility Below). $a$ and $b$ are said to be *connectible below* $d$ modulo $C$, written as $a \leftrightarrow_C^{\prec d} b$, iff there exists a sequence of ring elements $e_0, e_1, \ldots, e_k$ with $a = e_0 \leftrightarrow_C e_1 \leftrightarrow_C \ldots \leftrightarrow_C e_k = b$, and $e_i \prec d$ for all $0 \leq i \leq k$.

### 3.2.2 Gröbner Bases

Having introduced the reduction relation we can immediately turn to the crucial notion in our theory, namely that of *Gröbner bases*:

**Definition 3.2.6** (Gröbner Basis). A set $G$ is called a *Gröbner basis* iff the reduction relation it induces has the Church-Rosser property, i. e. whenever $a \leftrightarrow_G^* b$ then also $a \downarrow_G^* b$. $G$ is called a Gröbner basis of $C$ iff it is a Gröbner basis and $\mathrm{ideal}(G) = \mathrm{ideal}(C)$.

It is a well-known fact that the Church-Rosser property is always equivalent to confluence, and that confluence of $\rightarrow_G$ is equivalent to local confluence since $\rightarrow_G$ satisfies the finite chain condition (see Remark 3). However, a seemingly even weaker alternative characterization of the Church-Rosser property is made use of when proving Theorem 3.5.3: the so-called "generalized Newman lemma".

**Comparison to the original setting.** Apart from the Church-Rosser property of $\rightarrow_G$, there are also several other characterizations of Gröbner bases that could all be taken as their definition in the original setting:

- the leading term of every polynomial in $\mathrm{ideal}(G)$ is the multiple of the leading term of an element of $G$,

- every ideal element can be reduced to $0$ modulo $G$, or
- every S-polynomial of elements of $G$ can be reduced to $0$ modulo $G$.

Clearly, the first alternative is meaningless in reduction rings, since in general there is no such concept as "leading term". An analogue to the third possibility is precisely the Main Theorem of reduction ring theory (Theorem 3.5.3). The second characterization with reducibility of ideal elements to $0$ is more tricky; see Section 3.6.3 for details. ■

### 3.2.3 Correlativity and Irrelativity

In this subsection we focus on the notions of *irrelativity* and *correlativity*, where the latter cannot be found in the literature but was introduced in the course of the formalization of the theory in Theorema. The following paragraphs consist of an in-depth discussion of the differences between irrelativity in our sense and in the sense of [Sti91], and the rationale behind correlativity. Readers who are not interested in all this may immediately proceed to Definition 3.2.12.

The notion of irrelativity cannot be found in the original paper by Buchberger [Buc83a], but was introduced by Stifter in [Sti85] in order to properly deal with zero-divisors; this, in fact, is also the reason for splitting $M_c$ into the $M_c^i$ (see also [Sti88]). In [Sti85, Sti88], however, the sets $I_c$ are restricted to $\{+, -\}$, meaning that there the $M_c$ are split into only two sets $M_c^+$ and $M_c^-$. Later, in [Sti91], the $I_c$ are generalized to arbitrary finite sets for being able to turn the $k$-fold direct product of $\mathcal{R}$, i. e. $\mathcal{R}^k$, into a reduction ring if $\mathcal{R}$ is one, too (see Section 3.4.5). This caused a re-definition of irrelativity that we found to be erroneous, and in order to give evidence to this claim, we first present the original definition according to [Sti91] (page 405), which we call "irrelative$_S$" to distinguish it from our revised definition of irrelativity:

**Definition 3.2.7** (Irrelative$_S$)**.** The two pairs $(m_1, c_1)$ and $(m_2, c_2)$ are said to be *irrelative*$_S$ iff either $c_1 \neq c_2$ or there exists some $i \in I_{c_1}$ such that $m_1 \in M_{c_1}^i$ and $m_2 \in M_{c_1} \backslash M_{c_1}^i$.

Even if one does not go into the subtle details of the theory, to the proofs of the main results where irrelativity is needed, one can see that irrelative$_S$ does not state what it should: although it clearly should be symmetric (this immediately becomes obvious when looking at Definitions 3.2.15 and 3.2.16, for instance), in general it is *not*. For instance, if $I_c = \{1, 2\}$, $M_c^1 \subset M_c^2$, $m_1 \in M_c^1$ and $m_2 \in M_c^2 \backslash M_c^1$, then $(m_1, c)$ and $(m_2, c)$ are irrelative$_S$, but $(m_2, c)$ and $(m_1, c)$ are not. This counterexample works because the $M_c^i$ do not need to form a partition of $M_c$: they may overlap or even be empty.

In principle, there are at least two ways to adapt the definition of irrelativity for making it symmetric. However, for the sake of simplicity we restrict the sets $I_c$ to $\{1, 2\}$ for now, because even this simpler setting suffices to demonstrate why neither of the two alternatives, called irrelative$_1$ and irrelative$_2$ below, is suitable. Later, when we present the real definitions of irrelativity and correlativity we used in our formalization, in Definitions 3.2.12 and 3.2.13, the $I_c$ may of course be arbitrary again. Moreover, since the case $c_1 \neq c_2$ is not interesting for our purpose anyway, we fix $c$ and define irrelative$_1$ and irrelative$_2$ only for that particular $c$.

**Definition 3.2.8** (Irrelative$_1$)**.** Two elements $m_1$ and $m_2$ are said to be *irrelative*$_1$, written $\mathrm{irr}_1(m_1, m_2)$, iff $m_1 \in M_c^1 \wedge m_2 \in M_c^2$ or $m_1 \in M_c^2 \wedge m_2 \in M_c^1$.

**Definition 3.2.9** (Irrelative$_2$)**.** Two elements $m_1$ and $m_2$ are said to be *irrelative*$_2$, written $\mathrm{irr}_2(m_1, m_2)$, iff $m_1 \notin M_c^1 \wedge m_2 \notin M_c^2$ or $m_1 \notin M_c^2 \wedge m_2 \notin M_c^1$.

*Remark* 4. Irrelative$_1$ corresponds to irrelativity according to [Sti85, Sti88] (where $I_c$ is $\{+, -\}$).

Obviously, both irrelative$_1$ and irrelative$_2$ are symmetric, but unfortunately neither of them can be taken as *the* definition of irrelativity, because irrelativity has to satisfy the two crucial properties (I1) and (I2), for all $m_1, m_2, m_3, m_4, m \in M_c$:

$$\mathrm{irr}(m_1, m_2) \;\Rightarrow\; (\neg\mathrm{irr}(m_1, m) \vee \neg\mathrm{irr}(m_2, m)) \tag{I1}$$

$$(\neg\mathrm{irr}(m_1, m_2) \wedge \mathrm{irr}(m_3, m_4)) \;\Rightarrow\; \bigvee \left\{ \begin{array}{l} \mathrm{irr}(m_1, m_3) \wedge \mathrm{irr}(m_2, m_3) \\ \mathrm{irr}(m_1, m_4) \wedge \mathrm{irr}(m_2, m_4) \end{array} \right. \tag{I2}$$

Property (I1) is needed in the proof of the Main Theorem of reduction ring theory (Theorem 3.5.3), and (I2) in the proof that polynomial rings over reduction rings can be made reduction rings themselves. We will prove now that neither irrelative$_1$ nor irrelative$_2$ satisfies both (I1) and (I2) at the same time (with $\mathrm{irr}$ replaced by $\mathrm{irr}_1$ or $\mathrm{irr}_2$, respectively).

**Lemma 3.2.10.** *Irrelative$_1$ does not satisfy (I1).*

*Proof.* We have to construct $M_c^1$ and $M_c^2$ and find $m_1, m_2, m$ such that the left-hand-side of (I1) holds but the right-hand-side does not. For this, let $m \in M_c^1 \cap M_c^2$. No matter how we choose $m_1$ and $m_2$, the right-hand-side definitely fails because we always have $\mathrm{irr}_1(m_1, m)$ and $\mathrm{irr}_1(m_2, m)$. Thus, we only need to make sure that $\mathrm{irr}_1(m_1, m_2)$ holds, which can always be achieved since $M_c^1 \cap M_c^2 \neq \emptyset$. □

**Lemma 3.2.11.** *Irrelative$_2$ does not satisfy (I2).*

*Proof.* We have to construct $M_c^1$ and $M_c^2$ and find $m_1, m_2, m_3, m_4$ such that the left-hand-side of (I2) holds but the right-hand-side does not. For this, let $m_1, m_2 \in M_c^1 \cap M_c^2$. No matter how we choose $m_3$ and $m_4$, the right-hand-side definitely fails because we always have $\neg\mathrm{irr}_2(m_1, m)$ and $\neg\mathrm{irr}_2(m_2, m)$ for *any* $m$. This also ensures that the first conjunct on the left-hand-side holds, and so we only need to make sure that $\mathrm{irr}_2(m_3, m_4)$ holds. This is possible if $M_c^1 \not\subseteq M_c^2$ and $M_c^2 \not\subseteq M_c^1$. □

The solution we employed to overcome the issues with irrelativity and (I1) and (I2) is not committing ourselves to one of the two alternatives, but somehow to use *both*. Our definition of irrelativity, to be found below in Definition 3.2.12, basically is the one of irrelative$_1$, and instead of irrelative$_2$ we introduce the new concept of *correlativity* in Definition 3.2.13. However, this alone does not suffice to fix our problems with (I1) and (I2), because still irrelativity and correlativity do not satisfy both of them. Instead we also had to tweak one of the reduction ring axioms listed in Section 3.3.1, namely (R6), by replacing the condition “not irrelative” by “correlative”. The effect of this adjustment was that the two crucial properties (I1) and (I2) could be replaced by only one new property. Before it is presented, though, we finally introduce our definitions of irrelativity and correlativity; note that from now on the sets $I_c$ may be arbitrary again:

**Definition 3.2.12** (Irrelative)**.** The two pairs $(m_1, c_1)$ and $(m_2, c_2)$ are said to be *irrelative* w. r. t. $i, j \in I_{c_1}$, written as $\mathrm{irr}_{i,j}((m_1, c_1), (m_2, c_2))$, iff

- $c_1 \neq c_2$ or
- $m_1 \in M_{c_1}^i$, $m_2 \in M_{c_1}^j$ and $i \neq j$.

**Definition 3.2.13** (Correlative)**.** The two elements $m_1$ and $m_2$ are said to be *correlative* w. r. t. $c$, written as $\mathrm{cor}(m_1, m_2; c)$, iff there exists some $i \in I_c$ with $m_1 \in M_c^i$ and $m_2 \in M_c^i$.

In contrast to all the previous variants of irrelativity, and also to correlativity, our variant explicitly depends on the two indices $i$ and $j$ and does not quantify them existentially. Correlativity, on the other hand, is not defined for pairs of ring elements, but only for two individual ring elements w. r. t. a third one.

The crucial property of correlativity and irrelativity in our setting, which replaces (I1), is stated in the following

**Lemma 3.2.14.** *For* $m_1, m_2 \in M_c$*:*
*If* $\neg\mathrm{cor}(m_1, m_2; c)$ *then there exist* $i, j \in I_c$ *such that* $\mathrm{irr}_{i,j}((m_1, c), (m_2, c))$*.*

*Proof.* Follows immediately from Definitions 3.2.12 and 3.2.13. □

Property (I2) does not have an analogue in our setting, because of the slightly modified definition of non-trivial common reducibles (Definition 3.2.16) where the indices $i, j$ are made explicit (just as in our definition of irrelativity). See also Section 3.4.4.

Please also note that correlativity is *not* just the negation of irrelativity. For instance, if $I_c = \{1, 2\}$ and $M_c^1 = M_c^2$, then both $\mathrm{irr}_{1,2}((m_1, c), (m_2, c))$ and $\mathrm{cor}(m_1, m_2; c)$ hold for all $m_1, m_2 \in M_c$.

### 3.2.4 Common Reducibles

After the notions of irrelativity and correlativity have been introduced, we turn again to the reduction relation, or, more precisely, to *common reducibles*.

**Definition 3.2.15** (Common Reducible)**.** An element $a$ is called a *common reducible* of $c_1$ and $c_2$ w. r. t. $i, j \in I_{c_1}$, written as $\mathrm{cr}_{i,j}(a, c_1, c_2)$, iff there exist $m_1, m_2$ such that $a \rightarrow_{m_1,c_1}$, $a \rightarrow_{m_2,c_2}$, and moreover $\mathrm{irr}_{i,j}((m_1, c_1), (m_2, c_2))$ holds.

**Definition 3.2.16** (Non-Trivial Common Reducible)**.** An element $a$ is called a *non-trivial common reducible* (ntcr) of $c_1$ and $c_2$ w. r. t. $i, j \in I_{c_1}$, written as $c_1 \triangle_{i,j}^a c_2$, iff $\mathrm{cr}_{i,j}(a, c_1, c_2)$ and there do *not* exist $m_1, m_2$ such that

1. $\mathrm{irr}_{i,j}((m_1, c_1), (m_2, c_2))$,
2. $a \rightarrow_{m_1,c_1}$,
3. $a \rightarrow_{m_2,c_2}$, and
4. $a - m_1\, c_1 \rightarrow_{m_2,c_2}$ or $a - m_2\, c_2 \rightarrow_{m_1,c_1}$.

**Definition 3.2.17** (Minimal Non-Trivial Common Reducible)**.** An element $a$ is called a *minimal non-trivial common reducible* (mntcr) of $c_1$ and $c_2$ w. r. t. $i, j \in I_{c_1}$, written as $c_1 \nabla_{i,j}^a c_2$, iff $c_1 \triangle_{i,j}^a c_2$ and $a$ is minimal (w. r. t. $\preceq$) with this property.

*Remark* 5. Since $\preceq$ in general is only a partial ordering, there may well exist several minimal non-trivial common reducibles for given $c_1, c_2, i, j$.

**Comparison to the original setting.** In the original setting, the mntcrs of two non-zero polynomials $c_1$ and $c_2$ are precisely those monomials of the form $d \cdot \mathrm{lcm}(\mathrm{lp}(c_1), \mathrm{lp}(c_2))$ for non-zero coefficients $d$ (lp denotes the leading power-product). The indices $i, j$ do not matter, since $|I_{c_1}| = 1$ in this case. ■

In addition to (minimal) non-trivial common reducibles of two elements we also need (minimal) non-trivial common reducibles of only one element. This seemingly strange concept was introduced in [Sti85] to be able to deal with possible zero-divisors. However, the "unary" versions are only needed for proving that $\mathcal{R}[X]$ is a reduction ring if $\mathcal{R}$ is, but they are not needed at all in $\mathcal{R}$ itself (e. g. for proving Theorem 3.5.3).

**Definition 3.2.18** (Unary Non-Trivial Common Reducible)**.** An element $a$ is called a *unary non-trivial common reducible* (untcr) of $c$, written as $c\triangle^a$, iff $a \rightarrow_{\{c\}}$ and there do *not* exist $m_1, m_2$ such that

1. $a \rightarrow_{m_1,c}$,
2. $a \rightarrow_{m_2,c}$, and
3. $a - m_1\, c \rightarrow_{m_2,c}$.

**Definition 3.2.19** (Unary Minimal Non-Trivial Common Reducible)**.** An element $a$ is called a *unary minimal non-trivial common reducible* (umntcr) of $c$, written as $c\nabla^a$, iff $c\triangle^a$ and $a$ is minimal (w. r. t. $\preceq$) with this property.

As pointed out in [Sti85, Sti88] there is a crucial difference between $c\triangle^a_{i,j}c$ and $c\triangle^a$ (and hence also between $c\nabla^a_{i,j}c$ and $c\nabla^a$): on the one hand, if $c\triangle^a_{i,j}c$ for some $i, j \in I_c$, then not necessarily $c\triangle^a$, because there might exist $m_1, m_2 \in M_c$ such that $\neg\mathrm{irr}_{i,j}((m_1, c), (m_2, c))$ and $a \rightarrow_{m_1,c}$, $a \rightarrow_{m_2,c}$ and $a - m_1\, c \rightarrow_{m_2,c}$. On the other hand, $c\triangle^a$ does not imply $c\triangle^a_{i,j}c$ for some $i, j \in I_c$ either: $a$ might not even be a common reducible of $c$ and $c$ w. r. t. $i, j$ (in the sense of Definition 3.2.15), for instance if $|I_c| = 1$.

The last two definitions related to common reducibles are that of *critical pair multipliers* and *critical pairs*:

**Definition 3.2.20** (Critical Pair Multipliers)**.** The two elements $m_1$ and $m_2$ are called *critical pair multipliers* of $c_1$ and $c_2$ w. r. t. $a$ and $i, j \in I_{c_1}$, written as $(m_1, c_1)\Diamond^a_{i,j}(m_2, c_2)$, iff

1. $c_1\nabla^a_{i,j}c_2$,
2. $\mathrm{irr}_{i,j}((m_1, c_1), (m_2, c_2))$,
3. $a \rightarrow_{m_1,c_1}$ and
4. $a \rightarrow_{m_2,c_2}$.

**Definition 3.2.21** (Critical Pair)**.** The two elements $b_1$ and $b_2$ constitute a *critical pair* of $c_1$ and $c_2$ w. r. t. $a$ and $i, j \in I_{c_1}$ iff there exist $m_1, m_2$ such that $(m_1, c_1)\Diamond^a_{i,j}(m_2, c_2)$ and $a \rightarrow_{m_k,c_k} b_k$ for $k = 1, 2$.

*Remark* 6. It is important to note that two elements $c_1$ and $c_2$ may have several critical pairs (possibly infinitely many) *even for fixed* $a$ *and* $i, j \in I_{c_1}$.

**Comparison to the original setting.** In the original setting two polynomials have exactly one critical pair for each mntcr. The differences of the critical pairs are precisely the S-polynomials (differing only by constant factors). ■

### 3.2.5 Equivalence Relation

The last auxiliary notion needed before we can specify what it means for $\mathcal{R}$ to be a reduction ring is the binary relation $\sim$. This relation cannot be found in the existing literature but rather was introduced in the course of our formalization, in order to be able to turn fields into algorithmic reduction rings.

**Definition 3.2.22.** Fix $m \in \mathcal{R}$. Two elements $x, y \in \mathcal{R}$ are called *equivalent modulo* $m$, written as $x \sim_m y$, iff

1. $y = m\,x$,
2. $a \prec b \;\Leftrightarrow\; m\,a \prec m\,b$ for all $a, b \in \mathcal{R}$,
3. $n \in M_c^i \;\Leftrightarrow\; m\,n \in M_c^i$ for all $c \in \mathcal{R}\backslash\{0\}$ and $i \in I_c$,
4. $c_1 \nabla_{i,j}^{x} c_2 \;\Leftrightarrow\; c_1 \nabla_{i,j}^{y} c_2$ for all $c_1, c_2 \in \mathcal{R}$ and $i, j \in I_{c_1}$, and
5. $c \nabla^{x} \;\Leftrightarrow\; c \nabla^{y}$ for all $c \in \mathcal{R}$

**Definition 3.2.23** (Relation $\sim$)**.** Two elements $x, y \in \mathcal{R}$ are called *equivalent*, written as $x \sim y$, iff there exists a *unit* (i. e. an invertible element) $m$ with $x \sim_m y$.

As shown in Lemma 3.2.24, $\sim$ is an equivalence relation on $\mathcal{R}$. Apparently, it is quite strong, which is reflected in the fact that in the reduction ring $\mathbb{Z}$ it coincides with equality. In fields, however, all non-zero elements are equivalent to each other, as can be seen in Section 3.4.1. Maybe it is possible to relax the definition of $\sim$ a bit and still keep the validity of the lemmas stated below; after all, fewer equivalence classes in a reduction ring have the potential to increase the efficiency of Buchberger's algorithm for computing Gröbner bases, because fewer mntcrs might have to be considered (see Section 3.5.1).

We now state and prove some important facts about $\sim$ and $\sim_m$.

**Lemma 3.2.24.** *$\sim$ is an equivalence relation on $\mathcal{R}$.*

*Proof.* Reflexivity is obvious, since we trivially have $a \sim_1 a$ for all $a \in \mathcal{R}$ and 1 is a unit. For symmetry and transitivity note that all five requirements in Definition 3.2.22 are either equalities or equivalences and hence symmetric and transitive, that the inverse of a unit is itself a unit, and that the product of two units is again a unit. Hence, if $a \sim_m b$ and $m\,n = 1$, then $b \sim_n a$, and if $a \sim_{m_1} b$ and $b \sim_{m_2} c$, then $a \sim_{m_2\,m_1} c$. □

**Lemma 3.2.25.** *Assume* $a \sim_m b$ *and* $x \rightarrow_{n,c} y$. *Then also* $m\,x \rightarrow_{m\,n,c} m\,y$.

*Proof.* From $a \sim_m b$, by Definition 3.2.22, we know $y \prec x \Leftrightarrow m\,y \prec m\,x$ as well as $n \in M_c \Leftrightarrow m\,n \in M_c$ by the definition of $M_c$ as the union of all the $M_c^i$. From $x \rightarrow_{n,c} y$, by Definition 3.2.1, we know $y = x - n\,c$, $y \prec x$ and $n \in M_c$, and we have to show $m\,y = m\,x - (m\,n)\,c$, $m\,y \prec m\,x$ and $m\,n \in M_c$. All three sub-goals follow immediately from what we know and the fact that $\mathcal{R}$ is a ring, so we are done. □

**Lemma 3.2.26.** *Assume* $a \sim_m b$ *and* $\mathrm{irr}_{i,j}((m_1, c_1), (m_2, c_2))$ *for some* $i, j \in I_{c_1}$. *Then also* $\mathrm{irr}_{i,j}((m\,m_1, c_1), (m\,m_2, c_2))$.

*Proof.* If $c_1 \neq c_2$ the the goal follows trivially from Definition 3.2.12; hence, assume $c_1 = c_2 =: c$. From $\mathrm{irr}_{i,j}((m_1, c), (m_2, c))$ we know $i \neq j$, $m_1 \in M_c^i$ and $m_2 \in M_c^j$, and we have to show $i \neq j$, $m\,m_1 \in M_c^i$ and $m\,m_2 \in M_c^j$. The first sub-goal is already known, and the latter two follow immediately from the third requirement in Definition 3.2.22. □

**Lemma 3.2.27.** *Assume* $a \sim_m b$ *and* $(m_1, c_1) \Diamond_{i,j}^a (m_2, c_2)$ *for some* $i, j \in I_{c_1}$. *Then also* $(m\,m_1, c_1) \Diamond_{i,j}^b (m\,m_2, c_2)$.

*Proof.* The proof proceeds similarly as the ones before, using Definitions 3.2.22 and 3.2.20 and Lemmas 3.2.25 and 3.2.26. □

**Lemma 3.2.28.** *Assume* $a \sim_m b$ *and* $x \leftrightarrow_C^{\prec a} y$ *for some* $x, y \in \mathcal{R}$. *Then also* $m\,x \leftrightarrow_C^{\prec b} m\,y$.

*Proof.* From $x \leftrightarrow_C^{\prec a} y$ we obtain $e_0, \dots, e_k$ such that $x = e_0 \leftrightarrow_C \dots \leftrightarrow_C e_k = y$ and $e_i \prec a$ ($0 \leq i \leq k$). From Lemma 3.2.25 we can infer $m\,e_i \leftrightarrow_C m\,e_{i+1}$ ($0 \leq i < k$), and from Definition 3.2.22 we know $m\,e_i \prec m\,a = b$ ($0 \leq i \leq k$). Thus, our goal is witnessed by the sequence $m\,e_0, \dots, m\,e_k$. □

## 3.3 Reduction Ring Axioms

Now that all auxiliary notions have been introduced, we can present the axioms describing reduction rings and algorithmic reduction rings in detail. The distinction between (plain) reduction rings and algorithmic reduction rings was first introduced in our Theorema-formalization; in the literature, the term *reduction ring* is used for what we call algorithmic reduction ring. Although the driving intention behind reduction ring theory from its outset was the *algorithmic* treatment of ideal-theoretic problems, we nevertheless found a clear separation between the algorithmic and the non-algorithmic aspects of the theory reasonable.

## 3.3.1 Plain Reduction Rings

The following is the list of the nine axioms characterizing (plain) reduction rings in terms of the usual ring operations and the auxiliary notions defined in Section 3.2. It has to be pointed out that no single axiom requires the solvability of certain "higher-order" problems in $\mathcal{R}$, like deciding ideal membership. The names of the axioms are precisely the same as in [Sti85, Sti88, Sti91], and apart from some minor differences, to be discussed below, the axioms themselves are the same as well.

**(R0)** If $c \neq 0$, then $1 \in M_c$.

**(R1)** If $c \neq 0$ and $m \in M_c$, then $-m \in M_c$.

**(R2)** If $c \neq 0$ and $m \in M_c$, then $m\,c \neq 0$.

**(R3)** If $b, c \neq 0$, there exist $m_1, m_2, \ldots, m_k \in M_c$ such that $b = \sum_{i=1,\ldots,k} m_i$.

**(R4)** If $a \neq 0$, then $0 \prec a$.

**(R5)** If $a \rightarrow_{m,c} b$, then there are $m_1, m_2, \ldots, m_x$ and $n_1, n_2, \ldots, n_y$ such that

1. $a+d \rightarrow_{m_1,c} a+d-m_1\,c \rightarrow_{m_2,c} \ldots \rightarrow_{m_x,c} a+d-(m_1+\ldots+m_x)\,c = b+d-(n_1+\ldots+n_y)\,c \leftarrow_{n_y,c} \ldots \leftarrow_{n_2,c} b+d-n_1\,c \leftarrow_{n_1,c} b+d$ and
2. $m_1 + \ldots + m_x = m + n_1 + \ldots + n_y$.

**(R6)** If $a \rightarrow_{m_1,c} b_1$, $a \rightarrow_{m_2,c} b_2$ and $\mathrm{cor}(m_1, m_2; c)$, then there are $n_1, n_2, \ldots, n_k$ such that

1. $b_1 \leftrightarrow_{n_1,c} a-m_1\,c-n_1\,c \leftrightarrow_{n_2,c} \ldots \leftrightarrow_{n_k,c} a-(m_1+n_1+\ldots+n_k)\,c = b_2$,
2. $m_1 + n_1 + \ldots + n_k = m_2$, and
3. $a - (m_1 + n_1 + \ldots + n_i)\,c \prec a$ for all $1 \leq i \leq k$.

**(R7)** If $c_1 \triangle^a_{i,j} c_2$ then there are $k, l \in I_{c_1}$, $\overline{a} \preceq a$ and $\overline{m}$ such that

1. $c_1 \nabla^{\overline{a}}_{i,j} c_2$,
2. for every $e \neq 0$ and $m \in M_e$, also $\overline{m}\,m \in M_e$,
3. if $\overline{a} + e \prec \overline{a}$ then $a + \overline{m}\,e \prec a$, for all $e$,
4. if $b \rightarrow_e d$ then $\overline{m}\,b \leftrightarrow_e \overline{m}\,d$, for all $b, d, e$, and
5. in case $c_1 = c_2 =: c$: if $\overline{a} \rightarrow_{m_1,c}$ and $\overline{a} \rightarrow_{m_2,c}$ then $\overline{m}\,m_1 \in M^i_c$ and $\overline{m}\,m_2 \in M^j_c$, or vice versa, for all $m_1 \in M^k_c$ and $m_2 \in M^l_c$.

**(R8)** If $c \triangle^a$ then there are $\overline{a} \preceq a$ and $\overline{m}$ such that

1. $c\nabla^{\overline{a}}$,
2. for every $e \neq 0$ and $m \in M_e$, also $\overline{m}\, m \in M_e$,
3. if $\overline{a} + e \prec \overline{a}$ then $a + \overline{m}\, e \prec a$, for all $e$, and
4. if $b \rightarrow_e d$ then $\overline{m}\, b \leftrightarrow_e \overline{m}\, d$, for all $b, d, e$.

Some remarks on the axioms are in place:

- Axiom (R3) is only needed to prove that $\equiv_B$ coincides with $\leftrightarrow_B^*$ (Theorem 3.5.7). It is neither needed for proving the Main Theorem, nor for establishing the correctness of Buchberger's algorithm.
- In (R5) and (R6) the sum-constraints on the multipliers are only present because of possible zero divisors (multipliers, i. e. elements of $M_c$, may be zero divisors!). If $\mathcal{R}$ is an integral domain, though, (R5) could simply be phrased as
$$a \rightarrow_C b \Rightarrow a + d \downarrow_C^* b + d$$
and (R6) as
$$a \rightarrow_{m_1,c} b_1 \wedge a \rightarrow_{m_2,c} b_2 \wedge \operatorname{cor}(m_1, m_2; c) \Rightarrow b_1 \leftrightarrow_{\{c\}}^{\prec a} b_2.$$
- In [Sti85, Sti88, Sti91], the condition in (R6) is "$(m_1, c)$ and $(m_2, c)$ not irrelative" instead of "$m_1$ and $m_2$ correlative w. r. t. $c$". This is due to the problem related to the original definition of irrelativity, as discussed in detail in Section 3.2.3.
- In [Sti85, Sti88, Sti91], the fifth item in (R7) only requires $(\overline{m}\, m_1, c)$ and $(\overline{m}\, m_2, c)$ to be irrelative if $(m_1, c)$ and $(m_2, c)$ are. Here, we explicitly specify the subsets of $M_c$ the two multipliers $\overline{m}\, m_1$ and $\overline{m}\, m_2$ have to be contained in. However, this only needs to hold if $m_1 \in M_c^k$ and $m_2 \in M_c^l$, not for all $m_1, m_2 \in M_c$.
- Axiom (R8) is only needed to prove that polynomial rings over reduction rings can be made reduction rings themselves, but as (R3) it is not needed at all in the proof of the Main Theorem.

Finally, we can state the definition of (plain) reduction rings:

**Definition 3.3.1** (Reduction Ring). The structure $(\mathcal{R}, +, \cdot, 0, 1, \preceq, I, M)$ is a *reduction ring* iff $(\mathcal{R}, +, \cdot, 0, 1)$ is a commutative ring with identity, $\preceq$ is a partial Noetherian order relation, the $I_c$ (for $c \in \mathcal{R}$) are finite index sets, the $M_c^i$ (for $c \in \mathcal{R}$, $i \in I_c$) are sets of ring elements, and the axioms (R0)–(R8) are satisfied.

### 3.3.2 Algorithmic Reduction Rings

In order to be an algorithmic reduction ring, the reduction ring $(\mathcal{R}, +, \cdot, 0, 1, \preceq, I, M)$ must additionally be endowed with the four functions $lcrd$, $ulcrd$, $rdm$ and $cpm$. The behavior of these functions is specified by the first of the following four axioms, whereas the last axiom only depends on the usual constants and operations from plain reduction rings.

**(R9)** For every $c_1, c_2 \neq 0$ and $i, j \in I_{c_1}$, $lcrd(c_1, c_2, i, j) =: L$ is a ***finite*** set only containing ring elements $a$ with $c_1 \nabla^a_{i,j} c_2$. Moreover, for *every* $\overline{a}$ with $c_1 \nabla^{\overline{a}}_{i,j} c_2$ there exists $a \in L$ with $\overline{a} \sim a$.

**(R10)** For every $c \neq 0$, $ulcrd(c) =: L$ is a ***finite*** set only containing ring elements $a$ with $c\nabla^a$. Moreover, for *every* $\overline{a}$ with $c\nabla^{\overline{a}}$ there exists $a \in L$ with $\overline{a} \sim a$.

**(R11)** For all $a$ and $c$, if $a \rightarrow_{\{c\}}$, then $a \rightarrow_{rdm(a,c),c}$. Otherwise, $rdm(a, c) = 0$.

**(R12)** For all $c_1, c_2 \neq 0$, $i, j \in I_{c_1}$ and $a \in lcrd(c_1, c_2, i, j)$, $cpm(c_1, c_2, a, i, j)$ is a pair $(m_1, m_2)$ with $(m_1, c_1) \Diamond^a_{i,j} (m_2, c_2)$.

**(R13)** There does not exist an infinite sequence of sets $C_1, C_2 \ldots$ with $\mathrm{Red}(C_1) \subsetneq \mathrm{Red}(C_2) \subsetneq \ldots$, where $\mathrm{Red}(C) := \{a \in \mathcal{R} | a \rightarrow_C\}$.

Again, some remarks on the axioms are in place:

- Axioms (R9) and (R10) imply that there can only be finitely many equivalence classes of mntcrs of $c_1$ and $c_2$ w. r. t. $i, j$, and umntcrs of $c$, respectively; this ensures termination of Buchberger's algorithm. In [Sti85, Sti88, Sti91], where the two axioms are subsumed under the single axiom (RT2), the axioms require the numbers of mntcrs and umntcrs themselves to be finite, which is too strong.

- According to (R11), $rdm(a, c)$ has to be a suitable multiplier for reducing $a$ modulo $c$, if this is possible. This axiom corresponds to (RE2) in the literature.

- Axiom (R12) and function $cpm$ are new. In the literature, suitable critical-pair multipliers are simply obtained using $rdm$, but since we have to make sure that the multipliers are irrelative, we need a separate function.

- The last axiom, (R13), is (only) needed to ensure termination of Buchberger's algorithm. In [Sti85, Sti88, Sti91], it is referred to as (RT1).

The definition of algorithmic reduction rings is obvious now:

**Definition 3.3.2** (Algorithmic Reduction Ring)**.** The structure $(\mathcal{R}, +\cdot, 0, 1, \preceq, I, M, lcrd, ulcrd, rdm, \mathrm{cpm})$ is an *algorithmic reduction ring* iff $(\mathcal{R}, +\cdot, 0, 1, \preceq, I, M)$ is a reduction ring and axioms (R9)–(R13) are satisfied as well.

It is important to note that the term “algorithmic” in “algorithmic reduction ring” does *not* mean that the various operations ($lcrd$, $ulcrd$, $\dots$, but also $+$, $\cdot$, $\preceq$, $\dots$) are effectively computable. It only means that in algorithmic reduction rings, Buchberger’s algorithm for computing Gröbner bases can be defined and behaves according to its specification (see Section 3.5.1), but it is an algorithm only relative to the effective computability of the underlying operations it is defined in terms of (which are precisely the aforementioned operations in reduction rings). This computability is nowhere required, but as can be seen in Section 3.4, all known algorithmic reduction rings do indeed have computable $\preceq$, $I$, $M$, $lcrd$, $ulcrd$, $rdm$ and $cpm$, if the basic ring operations are computable.

## 3.4 Known Algorithmic Reduction Rings

In this section we list some well-known commutative rings with identity and describe how the various additional functions and relations ($\preceq$, $I$, $M$, $lcrd$, $\dots$) can be defined for turning them into algorithmic reduction rings. The first four rings in the list below are part of our formalization, the others are not (yet).

### 3.4.1 Fields

Fields are the simplest examples of reduction rings, but as has been pointed out a couple of times already, it took some effort to make them algorithmic reduction rings as well. Following [Buc83a, Buc84], any field $K$ (of any characteristic, not necessarily algebraically closed, finite or infinite, etc.) can be turned into a reduction ring by setting

$$a \preceq b \;:\Leftrightarrow\; a = 0$$

$$I_c := \{1\}$$

$$M_c^1 := K \backslash \{0\}$$

With these definitions, the proof that $(K, +, \cdot, 0, 1, \preceq, I, M)$ is a (plain) reduction ring is almost immediate; note, for instance, that $a \to_c b$ iff $a, c \neq 0$ and $b = 0$, and that actually *every* set $C \subseteq K$ is already a Gröbner basis. This might raise the question why one should consider fields as reduction rings at all, and why we introduced the equivalence relation $\sim$ for making fields algorithmic reduction rings. The answer is that even though the treatment of fields themselves is completely trivial, fields may form the basis of a hierarchy of functors[2] that are known to preserve the property of being a (plain or algorithmic) reduction ring; this, in particular, includes the functor that maps domains $\mathcal{D}$ to polynomial rings over $\mathcal{D}$, see Section 3.4.4. Clearly, the polynomial ring $K[x_1, \dots, x_n]$ cannot be dealt with so easily any more, but if the field $K$ was known to be an algorithmic reduction ring, it could be treated by exactly the same methodology as any other polynomial ring over an algorithmic reduction ring, and even any other *non*-polynomial algorithmic reduction ring due their uniform construction.

The problem with the original versions of Axioms (R9) and (R10) without the equivalence relation $\sim$, which can be found in [Buc83a] as Axiom (T2), is that they require the numbers of mntcrs and umntcrs to be finite. In a field with $\preceq$, $I$ and $M$ defined as above, however, one can easily see that for all $c_1, c_2 \neq 0$ and $c_1 \neq c_2$, *every* $a \neq 0$ satisfies $c_1 \nabla^a_{1,1} c_2$. Hence, in infinite fields the original versions of the two axioms are violated.[3] Our revised version of the axioms hold trivially, though, thanks to the following

**Lemma 3.4.1.** *In* $(K, +, \cdot, 0, 1, \preceq, I, M)$ *there are only two equivalence classes modulo* $\sim$*:* $\{0\}$ *and* $K \setminus \{0\}$.

*Proof.* Trivial. □

For completing the informal "proof" that $K$ can be turned into an algorithmic reduction ring we now present feasible definitions of the remaining functions:

$$lcrd(c_1, c_2, i, j) := \begin{cases} \emptyset & \Leftarrow \quad c_1 = 0 \vee c_2 = 0 \vee c_1 = c_2 \\ \{1\} & \Leftarrow \quad \text{otherwise} \end{cases}$$

$$ulcrd(c) := \begin{cases} \emptyset & \Leftarrow \quad c = 0 \\ \{1\} & \Leftarrow \quad \text{otherwise} \end{cases}$$

$$rdm(a, c) := \begin{cases} 0 & \Leftarrow \quad c = 0 \\ a/c & \Leftarrow \quad \text{otherwise} \end{cases}$$

$$cpm(c_1, c_2, i, j, a) := (rdm(a, c_1), rdm(a, c_2))$$

It is apparent that $(K, +, \cdot, 0, 1, \preceq, I, M, lcrd, ulcrd, rdm, cpm)$ satisfies (R9)–(R13) and thus is an algorithmic reduction ring.

*Remark* 7. Before introducing $\sim$ we tried to solve the problem with infinitely many mntcrs differently, by changing the ordering $\preceq$ in $K$ to $0 \prec 1 \preceq x$ for $x \neq 0$. This would indeed solve the problem, but on the other hand violate (R7) and (R8).

[2]In the Theorema sense; see Section 2.2.

[3]Interestingly, this fact is explicitly pointed out also in [Buc83a], but no solution is provided there.

### 3.4.2 Integers

The construction of the reduction ring of integers $\mathbb{Z}$ is similar as for fields, apart from $\preceq$:

$$a \preceq b \;:\Leftrightarrow\; |a| < |b| \vee (|a| = |b| \wedge a \leq b)$$
$$I_c := \{1\}$$
$$M_c^1 := \mathbb{Z}\backslash\{0\}$$

where $<$ is the standard ordering on $\mathbb{Z}$.

The proof that $(\mathbb{Z}, +, \cdot, 0, 1, \preceq, I, M)$ really constitutes a reduction ring can be modeled after the one in [Buc83a]. The new version of (R6) does not play a role anyway, because the $I_c$ are all singletons, meaning that all multipliers $m_1, m_2 \in M_c$ are correlative and none are irrelative.

Knowing that $(\mathbb{Z}, +, \cdot, 0, 1, \preceq, I, M)$ is a reduction ring makes finding suitable definitions for the remaining functions needed in algorithmic reduction rings rather easy. As before, we follow [Buc83a]; for this, we first introduce the auxiliary function lr as

$$\mathrm{lr}(c) := \begin{cases} \mathrm{quo}(|c|, 2) & \Leftarrow \quad \mathrm{even}(c) \\ -\mathrm{quo}(|c|+1, 2) & \Leftarrow \quad \text{otherwise} \end{cases}$$

where $\mathrm{quo}(a, b)$ denotes the quotient of the division of $a$ by $b$. If $c \neq 0$, $\mathrm{lr}(c)$ is the least element (w. r. t. $\preceq$) that can be reduced modulo $c$. Now the remaining functions can be defined as

$$lcrd(c_1, c_2, i, j) := \begin{cases} \emptyset & \Leftarrow \quad c_1 = 0 \vee c_2 = 0 \vee c_1 = c_2 \\ \{\max_{\preceq}(\mathrm{lr}(c_1), \mathrm{lr}(c_2))\} & \Leftarrow \quad \text{otherwise} \end{cases}$$

$$ulcrd(c) := \begin{cases} \emptyset & \Leftarrow \quad c = 0 \\ \{\mathrm{lr}(c)\} & \Leftarrow \quad \text{otherwise} \end{cases}$$

$$rdm(a, c) := \begin{cases} 0 & \Leftarrow \quad c = 0 \\ \mathrm{quo}(a + \mathrm{quo}(c, 2), c) & \Leftarrow \quad c > 0 \wedge a \geq 0 \\ -\mathrm{quo}(-a + \mathrm{quo}(c-1, 2), c) & \Leftarrow \quad c > 0 \wedge a < 0 \\ -rdm(a, -c) & \Leftarrow \quad \text{otherwise} \end{cases}$$

$$cpm(c_1, c_2, i, j, a) := (rdm(a, c_1), rdm(a, c_2))$$

The sets returned by $lcrd(c_1, c_2, i, j)$ and $ulcrd(c)$ not only contain one representative of each equivalence class of mntcrs and umntcrs, respectively, but they actually contain *all* such elements: $\preceq$, in $\mathbb{Z}$, is a total order relation, hence mntcrs and umntcrs are unique. Thus, the equivalence relation $\sim$ is of no relevance at all in this case.

The definition of $rdm(a,c)$ is not the simplest one: in [Buc83a] it is just defined as the sign of $a\,c$. Our definition, however, is more "efficient" in the sense that $rdm(a,c)$ always returns the *greatest* reduction multiplier, i. e. $a - rdm(a,c)\,c$ cannot be further reduced modulo $\{c\}$.

## 3.4.3 Quotient Rings of Integers

The last "basic" reduction rings that are also part of our formalization are the rings $\mathbb{Z}_k$ for arbitrary $k \in \mathbb{N}$. In the sequel, to simplify matters, we assume that $\mathbb{Z}_k$ is represented by $\{0,\dots,k-1\}$. Following [Sti85], we set

$$a \preceq b \;:\Leftrightarrow\; a \leq b$$

$$I_c := \{1,2\}$$

$$M_c^1 := \{m \in \mathbb{Z}_n | 1 \leq m < \mathrm{quo}(k, \mathrm{gcd}(k,c))\}$$

$$M_c^2 := -M_c^1$$

As can be seen easily, $\mathrm{quo}(k,\mathrm{gcd}(k,c))$ is the least element that, when multiplied with $c$, gives $0 \pmod k$. Therefore, $M_c^1$ (and hence also $M_c^2$) only contains non-zero-divisors of $c$, as required by Axiom (R2). The proof that $(\mathbb{Z}_n, +, \cdot, 0, 1, \preceq, I, M)$ constitutes a reduction ring is essentially the same as in [Sti85, Sti88]. The fact that "not irrelative" has been replaced by "correlative" in Axiom (R6) does not have any effect on the proof, since in the present setting these two notions can easily be shown to be equivalent (note that we always either have $M_c^1 = M_c^2$ or $M_c^1 \cap M_c^2 = \emptyset$).

For proving that $\mathbb{Z}_n$ can also be turned into an algorithmic reduction ring we first define the auxiliary function $\mathrm{lr}(c)$, as in $\mathbb{Z}$, to give the least element that is reducible modulo $c$ provided that $c \neq 0$. A simple argument (spelled out in detail in [Sti88]) shows that this element is precisely the greatest common divisor of $k$ and $c$, justifying the definition

$$\mathrm{lr}(c) := \mathrm{gcd}(k,c)$$

Knowing $\mathrm{lr}(c)$ it is obvious how $lcrd(c_1,c_2,i,j)$ and $ulcrd(c)$ have to be defined:

$$lcrd(c_1,c_2,i,j) := \begin{cases} \emptyset & \Leftarrow \quad c_1 = 0 \vee c_2 = 0 \\ \emptyset & \Leftarrow \quad c_1 = c_2 \wedge i = j \\ \{\max_\preceq(\mathrm{lr}(c_1), \mathrm{lr}(c_2))\} & \Leftarrow \quad \text{otherwise} \end{cases}$$

$$ulcrd(c) := \begin{cases} \emptyset & \Leftarrow \quad c = 0 \\ \operatorname{lr}(c) & \Leftarrow \quad \text{otherwise} \end{cases}$$

In the definition of $lcrd(c_1, c_2, i, j)$ note that now there is a mntcr even if $c_1 = c_2$, in contrast to the reduction rings discussed above. For the same reason as in $\mathbb{Z}$, also here the sets returned by $lcrd(c_1, c_2, i, j)$ and $ulcrd(c)$ contain *all* mntcrs and umntcrs, respectively. Thus, $\sim$ is irrelevant also in $\mathbb{Z}_n$.

$rdm(a, c)$ can be defined in various ways. However, in our formalization we simply defined it to be some $m \in M_c^1$ such that $a - m\,c \prec a$, if such an $m$ exists; otherwise it is 0. This definition is justified by the fact that $a \rightarrow_{m,c}$, for $m \in M_c^1$, iff there exists some $n \in M_c^2$ with $a \rightarrow_{n,c}$, meaning that it suffices to consider only multipliers from $M_c^1$. The actual computation of $rdm(a, c)$ in the Theorema-formalization is performed by iterating through the (finite!) set $M_c^1$ until a suitable multiplier is found, although there might be more efficient ways.

Finally, $cpm(c_1, c_2, i, j, a)$ is defined as

$$cpm(c_1, c_2, i, j, a) := \\ \begin{cases} (rdm(a, c_1), rdm(a, c_1) - \operatorname{quo}(k, \gcd(k, c_1))) & \Leftarrow \quad c_1 = c_2 \\ (rdm(a, c_1), rdm(a, c_2)) & \Leftarrow \quad \text{otherwise} \end{cases}$$

If $c_1 \neq c_2$ we do not have to care about the subsets of $M_{c_1}$ the multipliers returned by $cpm(c_1, c_2, i, j, a)$ are contained in, so we simply return $rdm(a, c_1)$ and $rdm(a, c_2)$. Otherwise, if $c_1 = c_2$, we exploit the fact that by our definition of $rdm$ we have $m_1 := rdm(a, c_1) \in M_c^1$, meaning that $m_2 := m_1 - \operatorname{quo}(k, \gcd(k, c_1)) \in M_c^2$ and $a \rightarrow_{m_2,c_2}$, as can be seen by an easy argument.

### 3.4.4 Polynomial Rings

In this subsection assume that $(\mathcal{R}, +, \cdot, 0, 1, \preceq, I, M)$ is a (plain) reduction ring, and fix a finite set $X := \{x_1, \ldots, x_k\}$ of indeterminates. Let $[X]$, the multiplicative commutative monoid generated by $X$, be ordered by the *term order* $\leq$, and let in the sequel $p, q$ always denote polynomials in $\mathcal{R}[X]$ and $s, t$ elements of $[X]$ ("power products").

Define $\mathrm{C}(p, t)$ as the coefficient of $p$ at $t$, $\mathrm{H}(p, t)$ as the "higher part" of $p$ w. r. t. $t$ (i. e. $p$ where the coefficients of all $s \leq t$ are set to 0), $\operatorname{lp}(p)$ as the "leading power product" of $p$ for $p \neq 0$ (i. e. the greatest power product appearing in $p$ with non-zero coefficient), and $\operatorname{lc}(p) := \mathrm{C}(p, \operatorname{lp}(p))$ (the "leading coefficient" of $p$).

Following [Buc83a, Buc84, Sti85], the ring $\mathcal{R}[X]$ can be turned into a reduction ring by setting

$$p \trianglelefteq q \ :\Leftrightarrow\ p = q \vee \left(\exists_{t \in [X]} \mathrm{H}(p, t) = \mathrm{H}(q, t) \wedge \mathrm{C}(p, t) \prec \mathrm{C}(q, t)\right)$$

$$IP_p := I_{\operatorname{lc}(p)}$$

$$MP_p^i := \{m \cdot t \mid m \in M^i_{\mathrm{lc}(p)}, t \in [X]\}$$

where the ordering on $\mathcal{R}[X]$ is denoted by $\trianglelefteq$ rather than $\preceq$ in order to distinguish it from the ordering on $\mathcal{R}$, and $I$ and $M$ are denoted by $IP$ and $MP$, respectively, for the same reason.

The proof that $(\mathcal{R}[X], +, \cdot, 0, 1, \trianglelefteq, IP, MP)$ is a reduction ring is essentially contained in [Sti85]. The revised variant of Axiom (R6) does not cause any problems at all, and the crucial property of irrelativity (I2) that is used in the proof in [Sti88] and which does not have an analogue for our new definition of irrelativity (Definition 3.2.12) fortunately turned out not be be needed, because in our definition of ntcrs (Definition 3.2.16) we make the indices $i, j$ explicit.

More precisely: property (CR1) in whose original proof (I2) is made use of has to be rephrased as

$p_1 \vartriangle^q_{i,j} p_2$ iff

1. $p_1 = p_2 \Rightarrow i \neq j$,
2. there do *not* exist $s \neq t \in [X]$, $m_1, m_2 \in \mathcal{R}$ with
   (a) $\mathrm{lp}(p_1) \mid s$ ("$\mathrm{lp}(p_1)$ divides $s$"),
   (b) $\mathrm{lp}(p_2) \mid t$,
   (c) $\mathrm{C}(q, s) \rightarrow_{m_1, \mathrm{lc}(p_1)}$,
   (d) $\mathrm{C}(q, t) \rightarrow_{m_2, \mathrm{lc}(p_2)}$, and (CR1)
   (e) $p_1 = p_2 \Rightarrow (m_1 \in M^i_{\mathrm{lc}(p_1)} \wedge m_2 \in M^j_{\mathrm{lc}(p_1)})$, but
3. there *does* exist $t \in [X]$ with
   (a) $\mathrm{lp}(p_1) \mid t$,
   (b) $\mathrm{lp}(p_2) \mid t$,
   (c) $(p_1 = p_2 \vee \mathrm{lc}(p_1) \neq \mathrm{lc}(p_2)) \Rightarrow \ \mathrm{lc}(p_1) \vartriangle^{\mathrm{C}(q,t)}_{i,j} \mathrm{lc}(p_2)$, and
   (d) $(p_1 \neq p_2 \wedge \mathrm{lc}(p_1) = \mathrm{lc}(p_2)) \Rightarrow \ \mathrm{lc}(p_1) \vartriangle^{\mathrm{C}(q,t)}$.

which obviates the necessity of something like (I2).

However, not only the property of being a plain reduction ring is preserved in polynomial rings, but also of being an *algorithmic* one. Hence, assume in the sequel that $(\mathcal{R}, +, \cdot, 0, 1, \preceq, I, M, lcrd, ulcrd, rdm, \mathrm{cpm})$ is an algorithmic reduction ring. The four additional functions in $\mathcal{R}[X]$ (which are suffixed by "P" in order to distinguish them from the ones in $\mathcal{R}$) are defined as

$$lcrdP(p_1, p_2, i, j) :=$$
$$\begin{cases} \emptyset & \Leftarrow \quad p_1 = 0 \vee p_2 = 0 \\ lcrd(\mathrm{lc}(p_1), \mathrm{lc}(p_2), i, j) \cdot \mathrm{lcm}(\mathrm{lp}(p_1), \mathrm{lp}(p_2)) & \Leftarrow \quad p_1 = p_2 \vee \mathrm{lc}(p_1) \neq \mathrm{lc}(p_2) \\ ulcrd(\mathrm{lc}(p_1)) \cdot \mathrm{lcm}(\mathrm{lp}(p_1), \mathrm{lp}(p_2)) & \Leftarrow \quad \text{otherwise} \end{cases}$$

$$ulcrdP(p) := \begin{cases} \emptyset & \Leftarrow \quad p = 0 \\ ulcrd(\mathrm{lc}(p)) \cdot \mathrm{lp}(p) & \Leftarrow \quad \text{otherwise} \end{cases}$$

where the products in $lcrdP(p_1, p_2, i, j)$ and $ulcrdP(p)$ are element-wise.

$rdmP(q, p)$ can be defined analogously to the original setting, by iterating through the monomials of $q$ until one reaches some $c \cdot t$ with $c \rightarrow_{\{\mathrm{lc}(p)\}}$ and $\mathrm{lp}(p) \mid t$. In that case, $rdmP(q, p) = rdm(c, \mathrm{lc}(p)) \cdot (t/\mathrm{lp}(p))$.

$cpmP(p_1, p_2, i, j, q)$, finally, is defined as

$$cpmP(p_1, p_2, i, j, q) :=$$
$$\begin{cases} (m_1 \cdot (\mathrm{lp}(q)/\mathrm{lp}(p_1)), m_2 \cdot (\mathrm{lp}(q)/\mathrm{lp}(p_2))) & \Leftarrow \quad p_1 = p_2 \vee \mathrm{lc}(p_1) \neq \mathrm{lc}(p_2) \\ (r \cdot (\mathrm{lp}(q)/\mathrm{lp}(p_1)), r \cdot (\mathrm{lp}(q)/\mathrm{lp}(p_2))) & \Leftarrow \quad \text{otherwise} \end{cases}$$

for $(m_1, m_2) = cpm(\mathrm{lc}(p_1), \mathrm{lc}(p_2), i, j, \mathrm{lc}(q))$ and $r = rdm(\mathrm{lc}(q), \mathrm{lc}(p_1))$.

The validity of Axioms (R11) and (R12) can be seen rather easily, at least when consulting [Buc83a, Sti85]. The proof that (R13) holds is a bit more involved but can be directly carried over from [Buc83a] just as it is. The validity of axioms (R9) and (R10), finally, follows readily from the corresponding axioms in $\mathcal{R}$ and the following lemma:

**Lemma 3.4.2** ($\sim$ in Polynomial Rings)**.** *For all $a, b \in \mathcal{R}$ with $a \sim b$ and $t \in [X]$ we also have $a \cdot t \sim b \cdot t$ (where the latter equivalence is in $\mathcal{R}[X]$, of course).*

*Proof.* From our assumption we know that there is some unit $m \in \mathcal{R}$ with $a \sim_m b$. Let $\overline{a} := a \cdot t$, $\overline{b} := b \cdot t$ and $\overline{m} := m \cdot 1$; we claim that $\overline{a} \sim_{\overline{m}} \overline{b}$. First of all, $\overline{m}$ apparently is a unit in $\mathcal{R}[X]$, because $m$ is one in $\mathcal{R}$, meaning that we only have to show

$$\overline{b} = \overline{m}\,\overline{a} \tag{G.1}$$

$$p \lhd q \Leftrightarrow \overline{m}\,p \lhd \overline{m}\,q \tag{G.2}$$

$$n \in MP_p^i \Leftrightarrow \overline{m}\,n \in MP_p^i \qquad (p \neq 0) \tag{G.3}$$

$$p_1 \nabla_{i,j}^{\overline{a}} p_2 \Leftrightarrow p_1 \nabla_{i,j}^{\overline{b}} p_2 \tag{G.4}$$

$$p \nabla^{\overline{a}} \Leftrightarrow p \nabla^{\overline{b}} \tag{G.5}$$

(G.1): From $a \sim_m b$ we know $b = m\,a$, hence $b \cdot t = (m \cdot 1)\,(a \cdot t)$.

(G.2): We only prove the direction from left to right, as the converse direction is completely analogous. From $p \lhd q$ we obtain $s \in [X]$ such that $\mathrm{C}(p, s) \prec \mathrm{C}(q, s)$ and $\mathrm{H}(p, s) = \mathrm{H}(q, s)$. From $a \sim_m b$ we can thus infer $\mathrm{C}(\overline{m}\,p, s) = m\,\mathrm{C}(p, s) \prec m\,\mathrm{C}(q, s) = \mathrm{C}(\overline{m}\,q, s)$, and since also $\mathrm{H}(\overline{m}\,p, s) = \overline{m}\,\mathrm{H}(p, s) = \overline{m}\,\mathrm{H}(q, s) = \mathrm{H}(\overline{m}\ , q, s)$ , we conclude $\overline{m}\,p \lhd \overline{m}, q$.

(G.3): As before we only prove "⇒". From $n \in MP^i_p$ we obtain $n_0 \in M^i_{\mathrm{lc}(p)}$ and $s \in [X]$ with $n = n_0 \cdot s$. From this, together with $a \sim_m b$, we can infer $m\,n_0 \in M^i_{\mathrm{lc}(p)}$, and thus also $\overline{m}\,n = (m\,n_0) \cdot s \in MP^i_p$.

(G.4): Again we only prove "⇒". By a well-known fact proved in [Sti85] (called "(CR2)" there) we know that $p_1 \nabla^{\overline{a}}_{i,j} p_2$ is equivalent to

1. $t = \mathrm{lcm}(\mathrm{lp}(p_1), \mathrm{lp}(p_2))$ and
2. $\mathrm{lc}(p_1) \nabla^a_{i,j} \mathrm{lc}(p_2)$ if $p_1 = p_2 \vee \mathrm{lc}(p_1) \neq \mathrm{lc}(p_2)$, and $\mathrm{lc}(p_1) \nabla^a$ otherwise.

Hence, we distinguish two cases based on whether or not $p_1 = p_2 \vee \mathrm{lc}(p_1) \neq \mathrm{lc}(p_2)$; however, since the two cases are quite similar, we assume that the formula holds and therefore also $\mathrm{lc}(p_1) \nabla^a_{i,j} \mathrm{lc}(p_2)$. Thus, together with $a \sim_m b$, we can infer $\mathrm{lc}(p_1) \nabla^b_{i,j} \mathrm{lc}(p_2)$, and by the same argument as before conclude $p_1 \nabla^{\overline{b}}_{i,j} p_2$.

(G.5): Analogous. □

Summarizing, $(\mathcal{R}[X], +, \cdot, 0, 1, \unlhd, IP, MP, lcrdP, ulcrdP, rdmP, cpmP)$ is an algorithmic reduction ring. Actually, this does not come "by chance" but was one of the fundamental design goals when reduction rings were first introduced in [Buc83a].

**Comparison to the original setting.** The case of reduction rings of the form $K[X]$ ($K$ a field) coincides exactly with the original setting in terms of the order relation on polynomials, the reduction relation and mntcrs. In the original setting, two monomials differing only by a constant factor can be regarded as "equivalent" (e. g. when forming mntcrs and S-polynomials), and in the reduction ring setting is equivalence is made explicit by $\sim$ and Lemma 3.4.2. The difference of a critical pair in reduction rings corresponds to the S-polynomial (again, up to a constant factor).

Furthermore, the well-known *elimination property* of Gröbner bases in the original setting carries over to polynomial reduction rings as well, regardless of whether the coefficient domain is a field or not; see Section 3.5.3. ■

### 3.4.5 Other Reduction Rings

The reduction rings discussed so far are all part of our formalization in Theorema; in this subsection we list other important (algorithmic) reduction rings. Our presentation mainly follows [Sti91], where the interested reader may find much more details and even further examples of reduction rings.

*Remark* 8. Our modifications and improvements of the original reduction ring theory, namely co- and irrelativity and the equivalence relation, have not been shown to be feasible for the reduction rings listed in this subsection yet. However, a rough overview of the rings in question gives confidence that our revised/new notions do not cause any problems there; at least we are not aware of any.

The following rings can be turned into algorithmic reduction rings:

- The cyclic foldings $C_{p,k}$ for prime $p$ and $k \in \mathbb{N}$, defined as $C_{p,k} = \mathbb{Z}_p^{k-1}$ with component-wise addition and $a\,b := \left(\sum_{j=0}^{k-1} a_j\, b_{(i-j) \bmod k}\right)_{i=0}^{k-1}$.
- The Gaussian integers $\mathbb{Z} + \mathrm{i}\,\mathbb{Z}$.
- The $k$-fold direct product of the algorithmic reduction ring $\mathcal{R}$ (i. e. $\mathcal{R}^k$ with component-wise addition and multiplication).

It is important to note that the fact that $\mathcal{R}^k$ is a reduction ring not only enables the computation of Gröbner bases of *ideals* in $\mathcal{R}^k$, but also of *modules* in $\mathcal{R}$. This, essentially, is achieved by restricting the sets of multipliers in $\mathcal{R}^k$ to contain only elements of the form $(m, \ldots, m)^T$ where all components are identical (and where multiplication by such a multiplier thus resembles multiplication by a single ring element); all this is exploited when proving that it is possible to compute Gröbner bases of syzygies in $\mathcal{R}$, see Section 3.5.4. In fact, treating modules in $\mathcal{R}$ is easier than treating ideals in $\mathcal{R}^k$ because of the smaller sets of multipliers. More information on modules can be found in [Sti93].

## 3.5 Main Results

In this section we present the main results in reduction ring theory, as well as some important differences to the original setting of polynomial rings over fields.

Throughout the whole section let $(\mathcal{R}, +, \cdot, 0, 1, \preceq, I, M)$ be a (not necessarily algorithmic) reduction ring.

### 3.5.1 Buchberger's Criterion and Buchberger's Algorithm

The first result we consider is the most important one. It corresponds to Buchberger's criterion on S-polynomials in the original setting, i. e. it yields an algorithmic criterion for deciding whether a given set is a Gröbner basis or not.

**Definition 3.5.1** (Connectible Critical Pairs). The two ring elements $c_1$ and $c_2$ are said to have *connectible critical pairs* modulo $C \subseteq \mathcal{R}$, written as $\mathrm{cpc}(c_1, c_2, C)$, iff for all $i, j \in I_{c_1}$ and all $a$ with $c_1 \nabla^a_{i,j} c_2$ there exist $\overline{a}, b_1, b_2$ such that

1. $a \sim \overline{a}$,
2. $b_1$ and $b_2$ constitute a critical pair of $c_1$ and $c_2$ w. r. t. $\overline{a}$ and $i, j$, and
3. $b_1 \leftrightarrow_C^{\prec a} b_2$.

The Main Theorem is based on the following

**Lemma 3.5.2.** *Fix $c_1, c_2 \in C$. If $\mathrm{cpc}(c_1, c_2, C)$, $\mathrm{cpc}(c_1, c_1, C)$ and $\mathrm{cpc}(c_2, c_2, C)$ then, for all $d, d_1, d_2$ with $d \rightarrow_{\{c_1\}} d_1$ and $d \rightarrow_{\{c_2\}} d_2$, we have $d_1 \leftrightarrow_C^{\prec d} d_2$.*

*Proof.* The proof is more or less the same as in [Sti85]. The crucial property of irrelativity needed is not (I1) any more, but what is stated in Lemma 3.2.14.

The only aspect of the lemma that is not reflected in the proof in [Sti85] is the fact that one representative of each equivalence class (modulo $\sim$) of mntcrs suffices. More precisely: if $\overline{b}_1, \overline{b}_2$ is a critical pair of $c_1$ and $c_2$ w. r. t. $\overline{a}$, $i, j$ and $C$, $a \sim \overline{a}$ and $\overline{b}_1 \leftrightarrow_C^{\prec \overline{a}} \overline{b}_2$, then there also exists a critical pair $b_1, b_2$ of $c_1$ and $c_2$ w. r. t. $a, i, j$ and $C$ with $b_1 \leftrightarrow_C^{\prec a} b_2$. Namely, if $a \sim_m \overline{a}$, then $m\, b_1$ and $m\, b_2$ is such a critical pair; this is an immediate consequence of Lemmas 3.2.27 and 3.2.28. □

**Theorem 3.5.3** (Main Theorem; Buchberger's Criterion). *Let $G \subseteq \mathcal{R}$. Then $G$ is a Gröbner basis iff for all $g_1, g_2 \in G \backslash \{0\}$ we have $\mathrm{cpc}(g_1, g_2, G)$.*

*Proof.* The theorem follows readily from Lemma 3.5.2 and a "generalized Newman lemma", invented and proved by Buchberger in [WB83]. □

The criterion of Theorem 3.5.3 is not effective yet, even for finite sets $G$, because of the in general undecidable condition $b_1 \leftrightarrow_G^{\prec \overline{a}} b_2$. However, we can also prove the following

**Corollary 3.5.4.** *Let $G \subseteq \mathcal{R}$. Then $G$ is a Gröbner basis iff for all $g_1, g_2 \in G \backslash \{0\}$, all $i, j \in I_{g_1}$ and all $a$ with $g_1 \nabla^a_{i,j} g_2$ there exist $\overline{a}, b_1, b_2$ such that*

1. *$a \sim \overline{a}$,*
2. *$g_1 \nabla^{\overline{a}}_{i,j} g_2$,*

*3. $b_1$ and $b_2$ constitute a critical pair of $g_1$ and $g_2$ w. r. t. $\overline{a}$ and $i, j$, and*

*4. $b_1$ and $b_2$ have identical normal forms modulo $G$.*

*Proof.* The direction from left to right is obvious: set $\overline{a} := a$ and choose *any* critical pair $b_1$ and $b_2$. Since $G$ is Gröbner basis and hence $\rightarrow_G$ is Church-Rosser, every element has a unique normal form modulo $G$. This uniqueness of normal forms guarantees that the normal forms of $b_1$ and $b_2$ are identical, as they both are also normal forms of $a$.

The converse direction follows from Theorem 3.5.3, because if the normal forms (modulo $G$) of two elements $b_1$ and $b_2$ are identical, we in particular know $b_1 \downarrow_G^* b_2$ and hence $b_1 \leftrightarrow_G^{\prec \overline{a}} b_2$ (because $b_1, b_2 \prec \overline{a}$!) □

One might argue that the criterion from the corollary is still not effective, because of the requirement on the existence of *some* $\overline{a}, b_1, b_2$. However, analogous to the proof of Corollary 3.5.4 it is easy to see that given $g_1, g_2, i, j$ and $a$ it suffices to consider *any* $\overline{a}$, $b_1$ and $b_2$ with the required properties ($a \sim \overline{a}$, $g_1 \nabla_{i,j}^{\overline{a}} g_2$, $b_1$ and $b_2$ critical pair) and reduce $b_1$ and $b_2$ to *some* normal forms: if the normal forms thus obtained are not identical, $G$ is definitely no Gröbner basis (even if $b_1$ and $b_2$ have some other identical normal forms!), and if they are, we found suitable witnesses (if not all normal forms are identical, though, other critical pairs will violate the criterion of the corollary).

**Comparison to the original setting.** Theorem 3.5.3 differs from Buchberger's criterion in the original setting in quite a few respects. First and foremost, the elements $g_1, g_2 \in G$ do not need to be distinct, but even identical pairs have to be considered. Second, even for fixed $i, j$ a pair $g_1, g_2$ might have several mntcrs, all of which have to be dealt with (or, at least, one representative of each equivalence class modulo $\sim$). Third, instead of reducing the difference of the critical pair (corresponds to the S-polynomial) to some normal form and checking whether this normal form is 0, one really has to reduce the constituents of the critical pair separately and check whether they have identical normal forms: as a counterexample take, as in [Sti85], the set $G := \{5x, 2y\}$ in the reduction ring $\mathbb{Z}[x, y]$ (according to Sections 3.4.2 and 3.4.4). All "S-polynomials" originating from $G$ can be reduced to 0, but both 0 and $-xy$ are normal forms of $3xy$ modulo $G$. ■

It is obvious that Theorem 3.5.3 and Corollary 3.5.4, respectively, not only allow to *check* whether a given (finite) set is a Gröbner basis or not, but also to actually *compute* Gröbner bases (at least if $\mathcal{R}$ constitutes an algorithmic reduction ring with functions $lcrd$, $ulcrd$, $rdm$ and $cpm$). The resulting *Buchberger algorithm* (Algorithm 1) is the direct translation of the algorithm from the original setting to reduction rings, with some adjustments due to the aforementioned differences between Theorem 3.5.3 and the corresponding theorem in the original setting.

**Algorithm 1** Buchberger's algorithm in the algorithmic reduction ring $\mathcal{R}$

**Input:** $D = \{d_1, \ldots, d_k\} \subseteq \mathcal{R}$
**Output:** $G \subseteq \mathcal{R}$ s. t. $G$ is a Gröbner basis of $D$

```
 1: function GB($D$)
 2:     $P \leftarrow \{(d_i, d_j) | 1 \leq i \leq j \leq k\}$
 3:     $G \leftarrow D$
 4:     while $P \neq \emptyset$ do
 5:         $(c_1, c_2) \leftarrow$ some element from $P$
 6:         $P \leftarrow P \backslash \{(c_1, c_2)\}$
 7:         for all $i, j \in I_{c_1}$ do
 8:             $L \leftarrow lcrd(c_1, c_2, i, j)$
 9:             for all $a \in L$ do
10:                 $(m_1, m_2) \leftarrow cpm(c_1, c_2, i, j, a)$
11:                 $h_1 \leftarrow \textsc{trd}(a - m_1\, c_1, G)$
12:                 $h_2 \leftarrow \textsc{trd}(a - m_2\, c_2, G)$
13:                 $h \leftarrow h_1 - h_2$
14:                 if $h \neq 0$ then
15:                     $P \leftarrow P \cup \{(h, h)\} \cup \{(g, h) | g \in G\}$
16:                     $G \leftarrow G \cup \{h\}$
17:                 end if
18:             end for
19:         end for
20:     end while
21:     return $G$
22: end function
```

Function GB depends on the auxiliary function TRD: this function is defined in terms of $rdm$ and totally reduces its first argument to some normal form modulo its second argument.

As noted already in Section 3.3.2 functions $lcrd$, $ulcrd$ (which is not needed in Algorithm 1, but only in the transition from $\mathcal{R}$ to $\mathcal{R}[X]$), $rdm$ and $cpm$ do not have to be effectively computable. Algorithm 1 is an algorithm only relative to the computability of these functions and to the computability of the basic ring operations in $\mathcal{R}$.

Completing the basis $G$ by $h_1 - h_2$ in Line 16 ensures that $h_1$ and $h_2$ have a common successor (and hence are connectible below $a$) modulo the new basis, because from $h_1 - h_2 \rightarrow_{\{h_1 - h_2\}} 0$ and Axiom (R5) we can infer $h_1 \downarrow^*_{\{h_1 - h_2\}} h_2$.

Algorithm 1 is not yet the best possible one. As in the original setting, we can employ the so-called *chain criterion* to get rid of useless reductions; see [Buc79] for a thorough description of the chain criterion in the original setting. Indeed, the chain criterion there is almost exactly the same in reduction rings:

**Definition 3.5.5** (Chain Criterion). Let $C \subseteq \mathcal{R}$, $c_1, c_2 \in C$, $a \in \mathcal{R}$ and $P \subseteq C \times C$. Then the *chain criterion* holds for $c_1$ and $c_2$ w. r. t. $a$, $C$ and $P$, written as $\mathrm{ccrit}(c_1, c_2, a, C, P)$, iff there exists $c \in C$ such that

1. $c_1$, $c_2$ and $c$ are pairwise distinct,
2. $(c_k, c_k) \notin P$ ($k = 1, 2$),
3. $(c, c) \notin P$,
4. $(c_k, c) \notin P$ and $(c, c_k) \notin P$ ($k = 1, 2$), and
5. $a \rightarrow_{\{c\}}$.

**Lemma 3.5.6.** *After Line 9 of Algorithm 1: if* $\mathrm{ccrit}(c_1, c_2, a, G, P)$ *holds, then all critical pairs pair of* $c_1$ *and* $c_2$ *w. r. t.* $a$ *and* $i, j$ *can be connected below* $a$ *modulo* $G$.

*Proof.* The proof is mainly based on the fact that if $a \rightarrow_{\{c\}} b$ for some $c \in G$, $b \in \mathcal{R}$ and $\mathrm{cpc}(c_1, c_1, G)$, $\mathrm{cpc}(c_2, c_2, G)$, $\mathrm{cpc}(c, c, G)$, $\mathrm{cpc}(c_1, c, G)$ and $\mathrm{cpc}(c_2, c, G)$, then the critical pairs of $c_1$ and $c_2$ w. r. t. $a$ and $i, j$ can be connected below $a$ modulo $G$. This follows readily from Lemma 3.5.2: let $b_1$ and $b_2$ be any critical pair of $c_1$ and $c_2$ w. r. t. $a$ and $i, j$. In particular we know $a \rightarrow_{\{c_1\}} b_1$ and $a \rightarrow_{\{c_2\}} b_2$, so from our assumptions and Lemma 3.5.2 we can infer both $b_1 \leftrightarrow_G^{\prec a} b$ and $b \leftrightarrow_G^{\prec a} b_2$. Combining these two results yields the desired $b_1 \leftrightarrow_G^{\prec a} b_2$.

Note that $\mathrm{cpc}(c_1, c_1, G)$ holds because from assumption $\mathrm{ccrit}(c_1, c_2, a, G, P)$ we can infer that the pairs $(c_1, c_1)$ etc. are not included in $P$ any more, meaning that they have already been processed in the algorithm. □

Lemma 3.5.6 suggests that in Algorithm 1, after choosing $a$ it is better to first check whether the chain criterion holds for the respective arguments: if it holds we can be sure that the criterion of Theorem 3.5.3 is satisfied for $c_1$, $c_2$ and $a$, meaning that no (expensive) reductions have to be performed.

### 3.5.2 Ideal Congruence and the Reduction Relation

Knowing how to compute Gröbner bases in reduction rings is nice, but for effectively solving ideal-theoretic problems we need a connection between the reduction relation and ideal congruence. Thanks to Axiom (R3), there is one:

**Theorem 3.5.7.** *For every set $C$, $\leftrightarrow_C^*$ coincides with $\equiv_C$.*

*Proof.* The proof of this theorem can be found in [Buc83a, Sti85]. $\square$

Now it is clear how various ideal-theoretic problems in reduction rings, like deciding ideal membership, ideal congruence and ideal equality, can effectively be solved if the ideals in question are given by *finite* bases. All this proceeds exactly as in the original setting of polynomials over fields, so we spare the (obvious) details here.

### 3.5.3 Elimination Property

One of the most important properties of Gröbner bases in the original setting is the so-called *elimination property*: if $X = \{x_1, \ldots, x_k\}$ is a set of indeterminates, $Y \subseteq X$, $\leq_Y$ is an *elimination order* for $Y$ on $X$ (i. e. $s <_Y t$ whenever $s \in [X \backslash Y]$ and $t \in [X] \backslash [X \backslash Y]$; see, e. g., [KR00], page 196), and $G$ is a Gröbner basis w. r. t. $\leq_Y$, then $G \cap K[X \backslash Y]$ is a Gröbner basis of $\mathrm{ideal}(G) \cap K[X \backslash Y]$ in $K[X \backslash Y]$.

Fortunately, the elimination property not only holds in polynomial rings over fields, but in polynomial rings over arbitrary reduction rings $\mathcal{R}$ (which are themselves reduction rings, according to Section 3.4.4). This important fact is summarized in the following

**Theorem 3.5.8** (Elimination Property)**.** *Let $X = \{x_1, \ldots, x_k\}$ a set of indeterminates, $Y \subseteq X$, $\leq_Y$ an elimination term order for $Y$, $G$ a Gröbner basis in the reduction ring $\mathcal{R}[X]$, and set $Z := X \backslash Y$. Then $G_Z := G \cap \mathcal{R}[Z]$ is a Gröbner basis of $\mathrm{ideal}(G) \cap \mathcal{R}[Z]$ in $\mathcal{R}[Z]$.*

*Proof.* The proof is basically the same as in the original setting: if $G$ is a Gröbner basis, every element of $\mathrm{ideal}(G)$ can be reduced to $0$ modulo $G$. This, in particular, also holds for the elements of $\mathrm{ideal}(G) \cap \mathcal{R}[Z]$, so let $a$ be such an element. $a$ can only be reduced modulo some $g \in G$ with $\mathrm{lp}(g) \in [Z]$, which further implies $g \in G_Z$ because $\leq_Y$ is an elimination order. Hence, the result of reducing $a$ once is still contained in $\mathcal{R}[Z]$, so applying this argument inductively we find that in the whole reduction process of $a$ to $0$, only reductors from $G_Z$ can be used, meaning that every element of $\mathrm{ideal}(G) \cap \mathcal{R}[Z]$ can be reduced to $0$ modulo $G_Z$. This proves $\mathrm{ideal}(G_Z) = \mathrm{ideal}(G) \cap \mathcal{R}[Z]$, but it might *not* prove that $G_Z$ is a Gröbner basis (see Section 3.6.3).

Nevertheless, $G_Z$ is a Gröbner basis thanks to Corollary 3.5.4: all mntcrs originating from $G_Z$ can easily be shown to be contained in $\mathcal{R}[Z]$, which implies that also all critical pairs are contained in $\mathcal{R}[Z]$. Now, since the critical pairs have identical normal forms modulo $G$, they must have identical normal forms modulo $G_Z$ as well, because in the reductions only reductors from $G_Z$ can be used. $\square$

*Remark* 9. Theorem 3.5.8 is not included in our formalization yet.

### 3.5.4 Gröbner Bases of Syzygies

The computation of a basis of the module of syzygies of a finite set of ring elements $\{b_1, \ldots, b_k\}$ is another important application of Gröbner bases theory in the original setting, see for instance [KR00], page 148. Fortunately, also this application can be transferred to reduction rings thanks to the fact that $\mathcal{R}^k$ with multipliers of the form $(m, \ldots, m)^T$ is an algorithmic reduction ring if $\mathcal{R}$ is, see Section 3.4.5.[4]

**Theorem 3.5.9** (Gröbner Bases of Syzygies). *Let $\mathcal{R}$ be a reduction ring, $B = \{b_1, \ldots, b_k\} \subseteq \mathcal{R}$ and set*

$$\overline{B} := (b_1, \ldots, b_k)$$

$$D := \{ \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ b_1 \end{pmatrix}, \ldots, \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ b_k \end{pmatrix} \} \subseteq \mathcal{R}^{k+1}$$

*Let $H$ be a Gröbner basis of $D$ in $\mathcal{R}^{k+1}$ (where the multipliers are of the form $(m, \ldots, m)^T$) according to [Sti93]. Write $H$ as*

$$H := \{ \begin{pmatrix} c_{1,1} \\ \vdots \\ c_{k,1} \\ g_1 \end{pmatrix}, \ldots, \begin{pmatrix} c_{1,\alpha} \\ \vdots \\ c_{k,\alpha} \\ g_\alpha \end{pmatrix}, \begin{pmatrix} s_{1,1} \\ \vdots \\ s_{k,1} \\ 0 \end{pmatrix}, \ldots, \begin{pmatrix} s_{1,\beta} \\ \vdots \\ s_{k,\beta} \\ 0 \end{pmatrix} \}$$

*where the $g_i$ are all non-zero. Then*

1. *$G := \{g_1, \ldots, g_\alpha\}$ is a Gröbner basis of $B$ in $\mathcal{R}$,*
2. *$c_i := (c_{1,i}, \ldots, c_{k,i})^T$ is the cofactor tuple of $g_i$ w. r. t. $\overline{B}$, i. e. $g_i = \overline{B}\, c_i$, and*
3. *$S := \{(s_{1,i}, \ldots, s_{k,i})^T \mid 1 \leq i \leq \beta\}$ is a Gröbner basis of the module of syzygies of $B$ (or, more precisely, $\overline{B}$) in $\mathcal{R}^k$.*

[4]Actually, it is not quite a reduction ring because it obviously violates (R3); this, however, does not matter in this case.

*Proof.* The proof is almost identical to the one in the original setting. The crucial property of reduction in $\mathcal{R}^{k+1}$ according to [Sti93] is

$$\begin{pmatrix} x_1 \\ \vdots \\ x_{k+1} \end{pmatrix} \rightarrow_{(m,\ldots,m)^T,(z_1,\ldots,z_i,0,\ldots,0)^T} \begin{pmatrix} y_1 \\ \vdots \\ y_{k+1} \end{pmatrix} \Leftrightarrow$$

$$\bigwedge \begin{cases} m \in M_{z_i} \\ y_j = x_j - m\, z_j \quad (1 \leq j \leq k+1) \\ y_i \prec x_i \end{cases} \qquad (1)$$

where $z_i \neq 0$ and $\prec$ and $M$ on the right-hand-side of the equivalence are in $\mathcal{R}$, of course.

1. We first prove that every $a \in \text{ideal}(B)$ can be reduced to $0$ modulo $G$. For this, assume $a \in \text{ideal}(B)$, i. e. $a = q_1\, b_1 + \ldots + q_k\, b_k$ for some $q_i \in \mathcal{R}$. Then apparently $(0, \ldots, 0, a)^T \equiv_D (-q_1, \ldots, -q_k, 0)^T$, and since $H$ is a Gröbner basis and because of (1), we know $(0, \ldots, 0, a) \rightarrow_H^* (*, \ldots, *, 0)$ and thus $a \rightarrow_G^* 0$. This proves $\text{ideal}(G) = \text{ideal}(B)$, but as in Theorem 3.5.8 it might not prove that $G$ is a Gröbner basis. However, since the criterion of Corollary 3.5.4 is satisfied for $H$, it is also satisfied for $G$, and $G$ indeed is a Gröbner basis (note that all mntcrs of $(c_{1,i}, \ldots, c_{k,i}, g_i)^T$ and $(c_{1,j}, \ldots, c_{k,j}, g_j)^T$ are of the form $(0, \ldots, 0, *)^T$, and so are the constituents of all critical pairs).

2. First observe that $(0, \ldots, 0, a) \in \text{module}(D)$ implies $a = 0$. Therefore, since $(0, \ldots, 0, g_i - c_{1,i}\, b_1 - \ldots - c_{k,i}\, b_k)^T \in \text{module}(D)$ we can conclude $g_i = c_{1,i}\, b_1 + \ldots + c_{k,i}\, b_k = \overline{B}\, c_i$.

3. By a similar argument as in 1. we can prove that $S$ is a Gröbner basis: the criterion of Corollary 3.5.4 holds for $H$, so it must hold for $S$ as well because of (1) and the shapes of the mntcrs. Furthermore, given any syzygy $(\sigma_1, \ldots, \sigma_k)^T$ of $\overline{B}$, we can easily infer $(\sigma_1, \ldots, \sigma_k, 0)^T \in \text{module}(D)$ as well and hence $(\sigma_1, \ldots, \sigma_k, 0)$ can be reduced to $0$ modulo $H$, meaning that the syzygy itself can be reduced to $0$ modulo $S$ (again because of (1)). □

*Remark* 10. Theorem 3.5.9 is not included in our formalization yet.

## 3.6 Differences to the Original Setting

We end this chapter by pointing out some striking differences between the original setting of polynomial rings over fields and the much more general reduction ring setting. Actually, the last difference listed below is merely a conjecture and not a known fact.

### 3.6.1 Singletons

The first difference is related to singleton sets. In the original setting, a singleton is automatically a Gröbner basis, but in reduction rings this is not the case any more.[5] The main reason for this seemingly strange phenomenon lies in the presence of *zero divisors*: reduction rings are not restricted to integral domains. For example, in the reduction ring $\mathbb{Z}_4[x]$ the singleton $\{2x+1\}$ is no Gröbner basis. This is because $2 = 2\,(2x+1) \in \mathrm{ideal}(2x+1)$, but $2$ cannot be reduced modulo $2x+1$ because the leading term of $2x+1$ does not divide any term in $2$ (see Section 3.4.4). Hence, $\{2x+1\}$ cannot be a Gröbner basis.

### 3.6.2 Reducibility and Irreducibility

In contrast to the original setting, the following seemingly obvious statements do *not* hold in reduction rings in general. In each case the reduction ring $\mathbb{Z}$ according to Section 3.4.2 serves as a counterexample.

- $a \to_C \;\Rightarrow\; -a \to_C$
  Counterexample: $C = \{2a\}$ and $a > 0$ (w. r. t. the standard ordering on $\mathbb{Z}$, not the reduction ring ordering),
- $a + b \to_C \;\Rightarrow\; (a \to_C \vee b \to_C)$
  Counterexample: $C = \{2a + 2b\}$ and $a, b > 0$,
- $a - b \to_C \;\Rightarrow\; (a \to_C \vee b \to_C)$
  Counterexample: $C = \{2a - 2b\}$ and $a > 0 > b$,
- $a \to_C \wedge \neg b \to_C \;\Rightarrow\; a + b \to_C$
  Counterexample: $C = \{2a\}$, $a > 1$ and $b = -1$.

### 3.6.3 0-Reducibility of Ideal Elements

The last difference concerns a necessary condition for being a Gröbner basis, which in the original setting is also sufficient: the reducibility of all ideal elements to $0$. In other words, property (Red 0) *might* not hold in reduction rings in general (but the converse still holds, of course).

$$\left(\forall_{a \in \mathrm{ideal}(C)}\; a \to_C^* 0\right) \;\Rightarrow\; C \text{ is a Gröbner basis} \qquad \text{(Red 0)}$$

[5] This situation parallels the one in non-commutative polynomial rings, where singletons are not necessarily Gröbner bases either.

The emphasis in the last sentence is on the word *might*: neither did we find a concrete example of a reduction ring and a set $C$ that violate (Red 0), nor did we manage to prove it.

A sufficient condition for (Red 0) to hold is

$$(a - b \to_C^* 0 \wedge a \neq b) \;\Rightarrow\; (a \to_C \vee b \to_C) \tag{Red 0'}$$

which trivially holds in fields and is preserved in the transition from $\mathcal{R}$ to $\mathcal{R}[X]$.

*Remark* 11. One might argue that if the left-hand-side of (Red 0) held but its right-hand-side failed for a $C$, then $C$ could still be completed to a Gröbner basis by Algorithm 1; an element $h$ thus added to $C$ is of course contained in $\mathrm{ideal}(C)$, so $h \to_C^* 0$ by assumption. In the original setting, $h$ would hence be redundant, and so $C$ would already be a Gröbner basis, contradicting our assumption. However, in reduction rings elements that can be reduced to $0$ might not be redundant in Gröbner bases—where as before the emphasis is on *might*.

# Chapter 4

# Formalization of Reduction Ring Theory

The focus of this chapter is on the formalization of reduction ring theory, as presented in the previous chapter, in Theorema, which constitutes the main achievement in our study. First, we give a general overview of the formalization and some design decisions we made. Then, we describe the individual components (i. e. sub-theories) of our formalization in more detail, before having a close look at the implementation, specification and verification of Buchberger's algorithm in Theorema. Afterward, the *special prover* developed for efficiently proving results in reduction ring theory is presented, followed by a comparison of the Theorema-formalization to our formalization of Gröbner bases theory in the Isabelle proof assistant.

Some parts of this chapter have already been published in [Mal15a, Mal15b, Mal15c, Mal16b].

## 4.1 Overview of the Formalization

The main objective of our PhD study was the formalization of the theory of reduction rings and Gröbner bases in the Theorema system. "Formalization", in that sense, has to be understood as the representation of all of the theory, i. e. axioms, definitions, theorems and algorithms, in terms of higher-order predicate logic formulas, together with the computer-supported and -checked formal verification of all results.

A substantial part of our formalization comprises elementary, general mathematical theories, such as sets, algebraic structures, integers and tuples, that are themselves independent of reduction ring theory and merely serve as its logical backbone. In this respect, our formalization can also be regarded a major contribution to a structured knowledge base of the foundations of mathematics in Theorema 2.0 that can be reused in future theory explorations. Such a knowledge base did not exist in Theorema 2.0 before, which justifies, in our opinion, presenting it just alongside the formal treatment of the “real” reduction ring theory in this chapter, in Section 4.2.

*Remark* 12. In this chapter the word “theory” will not only be used to refer to all of reduction ring theory, as presented in Chapter 3, but also to individual components (i. e. Theorema notebooks) of our formalization.

### 4.1.1 Related Work

To the best or our knowledge, reduction rings have never been the subject of computer-formalization in *any* mathematical assistant system before; Gröbner bases theory in the original setting (or at least parts thereof), though, has already been formalized in various systems, listed in chronological order:

- Théry and Persson [Thé01, Per01] formalized the basics of multivariate polynomials and Gröbner bases in Coq [BC04], and based on this formalization implemented and formally verified Buchberger’s algorithm (also in Coq), even incorporating Buchberger’s criteria to avoid useless reductions. From the provenly correct algorithm in Coq, an OCaml program was then extracted automatically.

- Medina-Bulo, Palomo-Lozano, Alonso-Jiménez and Ruiz-Reina [MPAR04, MPR10] implemented Buchberger’s algorithm in ACL2 [KMM00] and also proved it correct there. Just as in our case, the provenly correct algorithm can directly be executed without any prior compilation or translation into another system.

- Buchberger and Craciun [Buc04b, Cra08] managed to automatically synthesize Buchberger’s algorithm from its specification by the so-called *lazy thinking* method. The system they chose for their work was Theorema 1, the predecessor of our system.

- Schwarzweller [Sch06] formalized the theory of Gröbner bases in the Mizar system [BBG$^+$15], based on a range of auxiliary Mizar-theories on multivariate polynomials, term orders, polynomial reduction, etc. Algorithms are not executable in Mizar directly.

- Jorge, Guilas and Freire [JGF09] implemented Buchberger’s algorithm in OCaml and proved it correct using a formal representation of the underlying theory in Coq. In contrast to [Thé01] the algorithm is not implemented in Coq and then translated to OCaml, but rather *directly* implemented in OCaml, leading to an increased efficiency.

Furthermore, the purely algorithmic aspect of a variant of reduction ring theory has been considered in Theorema 1 in [Buc03], where Buchberger's algorithm was implemented in a generic way that served as a model for our implementation (see Section 4.4). However, said elaboration exclusively focuses on *computations* and the investigation of domains, functors and categories in Theorema; no single theorem, e. g. about the correctness or termination of the algorithm, is proved nor even stated there formally.

### 4.1.2 Design Decisions

It is obvious that a mathematical theory cannot be directly translated from a text-book representation into a fully formal one, without any minor (or even major) adjustments of the syntax, notions or even theorems. In our formalization, however, there are not that many differences compared to what is presented in Chapter 3:

- The ordering in reduction rings is not $\preceq$, but the strict, converse variant $\succ$ thereof. The reason for this lies in the fact that the reduction relation also has its smaller argument to the right, i. e. $\rightarrow_C \subseteq \succ$. Statements like Noetherianity of a relation can hence be formulated generally and applied to the ordering and to the reduction relation alike, without the having to first form the converse of one of them (using a higher-order function `converse`, for instance).

  In fact, the preference of $\succ$ over $\preceq$ in reduction rings had an effect on the standard ordering on numbers, too. In the various number-theories (Numbers.nb, NatInt.nb and NatIntExtended.nb) the basic relation all others are defined in terms of is $\geq$ and not $\leq$, as usually in mathematics. In LogicSets.nb, however, the basic relations are $\in$ and $\subseteq$ rather than $\ni$ and $\supseteq$.

- The sets $I_c$ are fixed to sets of the form $\{1, \ldots, \texttt{multN}[c]\}$, where $\texttt{multN}[c]$ is a function that maps $c$ to the number of sets of multipliers of $c$. The reason for this is simply because sets of natural numbers can be handled easier in computations. For instance, in Corollary 3.5.4 and hence also in Buchberger's algorithm it is apparent that for given $c_1$ and $c_2$ it suffices to consider either $i, j \in I_c$ or $j, i \in I_c$, but not both. If $I_c$ is a set of (natural) numbers, this observation can easily be implemented to increase the efficiency of the algorithm.

- The conditions $m \in M_c^i$ are replaced by ternary predicates $\texttt{mult}[c, m, i]$, meaning that the sets $M_c^i$ (and also $M_c$) are not explicitly included in the formalization. This, of course, is only a minor difference and does not have any deeper reason either.
- Functions $lcrd$ and $ulcrd$ in algorithmic reduction rings are both called `lcrd` in the formalization, making use of the different numbers of arguments (four vs. one) to distinguish them from each other.
- Buchberger's algorithm is implemented as a tail-recursive functional program rather than the imperative one shown in Algorithm 1. Although writing imperative programs is possible in Theorema in principle, functional programs typically facilitate correctness proofs. Buchberger's algorithm in our functional implementation, as well as its correctness proof, are the topic of Section 4.4.

One major design decision concerns the systematic construction of hierarchies of mathematical domains, e. g. $\mathcal{D} \to \mathcal{D}[X] \to \mathcal{D}[X]^2 \to \dots$. A natural candidate for achieving this goal is the use of *functors* and *domains*, as described in Section 2.2, and this is indeed the approach we pursue in our formalization.

There is, however, one subtle detail about domains in Theorema that we encountered during our work. Usually, the carrier of an algebraic structure, be it a group, a ring, a topology, or whatever, has to be a *set* and not a *proper class*. In some situations this requirement might be superfluous, but in other situations it is of utmost importance. Consider, for instance, a domain $\mathcal{D}$ ordered by $\succ$ (the underscript is omitted here for better readability). Assume further that $\succ$ is Noetherian on $\mathcal{D}$, meaning that every non-empty set of elements of $\mathcal{D}$ has a $\prec$-minimal element. Thus, one might want to employ Noetherian induction for proving universally quantified formulas ranging over all elements of $\mathcal{D}$ … but proving a corresponding induction rule is only possible if we know that the carrier of $\mathcal{D}$ itself is a *set* rather than a proper class.

The situation sketched above was not invented for making our point, but unfortunately really arose in our formalization and caused us to re-prove quite some theorems that had already been proved (but under a wrong assumption). The solution we employed to overcome the problem was to introduce a unary predicate, `isDomain`, defined in LogicSets.nb as

$$\underset{\mathcal{D}}{\forall} \quad \texttt{isDomain}[\mathcal{D}] :\Leftrightarrow \underset{\texttt{isSet}[A]}{\exists} \ \underset{\underset{\mathcal{D}}{\in}[x]}{\forall} x \in A \qquad (\textit{isDomain})$$

that specifies whether its argument is a "real" domain in the sense of our formalization, where we are adamant that the carriers of such structures are sets.

Besides the issue with the carriers of domains being sets, we also came across another minor drawback when working with domains and sets in parallel. If one wishes to introduce a notion or prove a theorem that is relevant both for sets and domains, one always has to do it twice: once for sets and once for domains. To continue our example with Noetherianity, one might want to express that a binary relation is Noetherian on a set (with the usual set membership predicate $x \in A$), but one might also want to express that it is Noetherian on a domain (with the domain membership predicate $\underset{\mathcal{D}}{\in}[x]$). As before, such a situation really occurred in our formalization, where we instead introduced the predicate isWellFounded to express that a relation is well-founded on a set, and isNoetherian for asserting the Noetherianity of a relation on a domain. Note that, according to widely-accepted mathematical standards, a binary relation $\succ$ is said to be Noetherian iff its converse is well-founded; in the formalization we adhere to this convention, except that isWellFounded[ $\prec, A$] is defined for *sets* $A$, whereas isNoetherian[ $\succ, \mathcal{D}$] is defined for *domains* $\mathcal{D}$.

### 4.1.3 Structure and Size of the Formalization

Figure 4.1 shows the dependencies of the individual sub-theories on each other. Each node represents a sub-theory, contained in a separate Theorema notebook, and a directed edge from theory $T_1$ to theory $T_2$ means that $T_2$ logically depends on $T_1$ in the sense that formulas (i. e. axioms or theorems) contained in $T_1$ are used in the proof of a theorem in $T_2$. The color of the frame of a node indicates whether the corresponding theory belongs to the knowledge base of elementary theories (blue; Section 4.2) or is directly related to reduction rings (red; Section 4.3). Note also that transitive edges are omitted for better readability, e. g. theory Numbers.nb not only depends *indirectly* on theory LogicSets.nb (via AlgebraicStructures.nb), but also *directly*; this fact is not reflected in Figure 4.1.

Figure 4.2 displays the sizes of the individual sub-theories in terms of the numbers of unproved axioms (including definitions) and proved theorems. Table 4.1 contains the accumulated formula numbers, again both unproved and proved, in the entire formalization. It reveals that the two main parts of the formalization (elementary theories and reduction ring theories) contain roughly the same number of axioms, but that a bit more theorems are proved in the elementary theories.

Table 4.2, finally, contains the average- and maximum sizes of the proofs in each theory, in terms of the numbers of inference steps. As can be seen, proofs in the elementary theories tend to be much shorter than proofs in the reduction ring theories. This, actually, is no surprise: many theorems in the former category are merely simple lemmas stating rather obvious and immediate properties of the

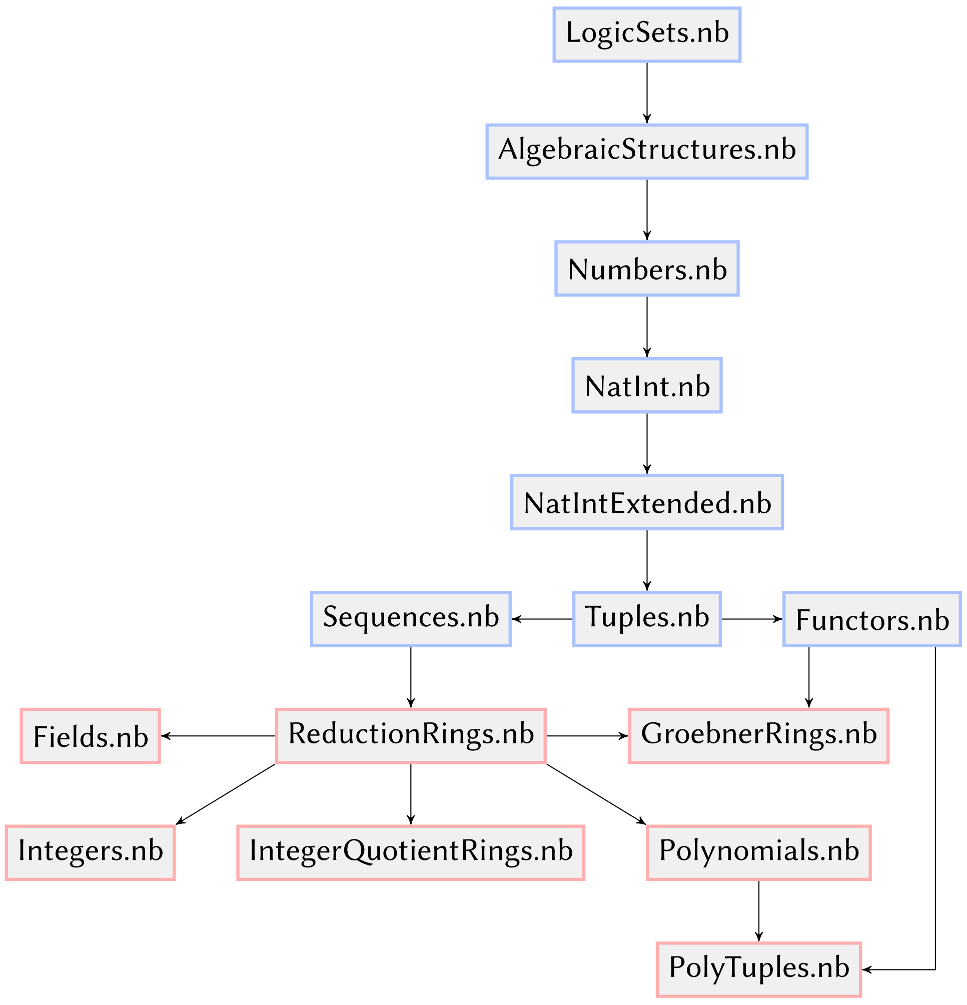


Figure 4.1: The theory dependency graph. Blue-framed nodes correspond to elementary theories, red-framed nodes to theories directly related to reduction rings.

notions involved, whereas theorems in the latter category usually deal with the rather complicated concepts related to reduction rings, as presented in Section 3.

At the moment, the formalization with all Theorema notebooks and proofs is not yet publicly available (e. g. in an online repository), because the mechanism for turning Theorema theories into so-called *Theorema Knowledge Archives* that can easily be shared amongst the users of the system is still in the development stages. As soon as it is completed, we will immediately put our formalization into a public repository that will be linked on the official Theorema web page.[1] The interested reader may nevertheless obtain the full formalization (or part thereof)

[1] http://www.risc.jku.at/research/theorema/software/

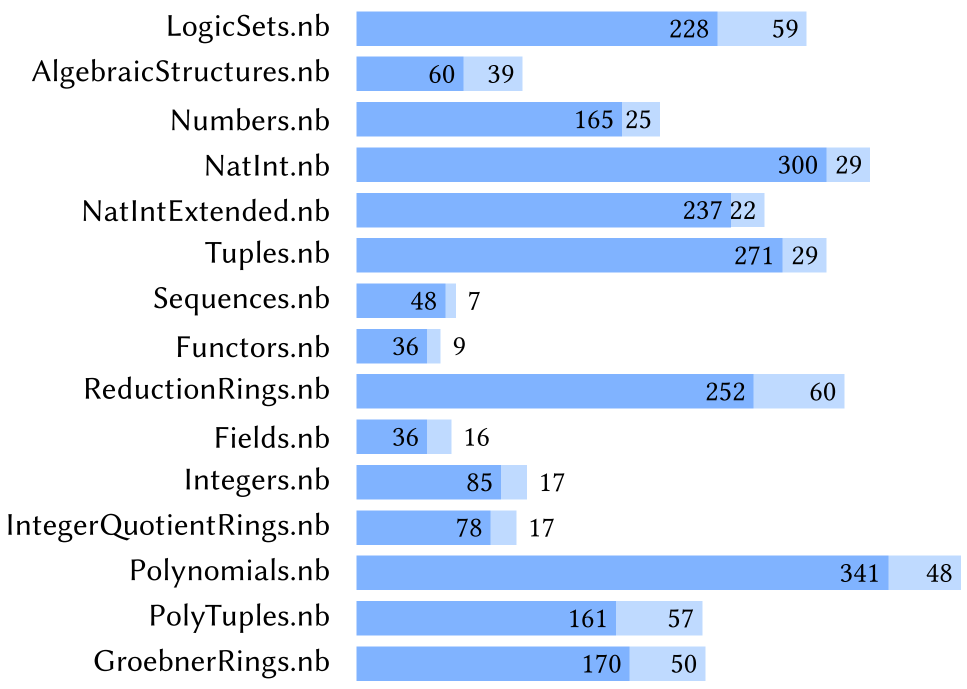


Figure 4.2: The sizes of the individual theories. The larger number in each row corresponds to the number of proved theorems, the smaller one to the number of definitions and axioms.

| | Axioms | Theorems | **Total** |
|---|---|---|---|
| Elementary theories | 219 | 1344 | 1563 |
| Reduction ring theories | 265 | 1123 | 1388 |
| **Total** | 484 | 2467 | **2951** |

Table 4.1: The accumulated formula numbers in the formalization.

in its current form by contacting the author.

## 4.2 Elementary Theories

In this section we have a closer look at the elementary theories we formalized in the course of the formal treatment of reduction ring theory. Please note that these elementary theories so far only include mathematical content that was explicitly needed for reduction rings; although this is quite comprehensive and covers many different concepts and results, it is still fairly incomplete.

| **Theory** | **Average** | **Maximum** |
|---|---|---|
| LogicSets.nb | 22.6 | 178 |
| AlgebraicStructures.nb | 32.6 | 91 |
| Numbers.nb | 18.4 | 135 |
| NatInt.nb | 17.2 | 137 |
| NatIntExtended.nb | 25.7 | 203 |
| Tuples.nb | 19.3 | 93 |
| Sequences.nb | 21.1 | 82 |
| Functors.nb | 23.5 | 106 |
| ReductionRings.nb | 38.3 | 183 |
| Fields.nb | 42.3 | 147 |
| Integers.nb | 52.9 | 214 |
| IntegerQuotientRings.nb | 52.4 | 203 |
| Polynomials.nb | 44.0 | 322 |
| PolyTuples.nb | 36.3 | 151 |
| GroebnerRings.nb | 36.1 | 149 |

Table 4.2: The average and maximum sizes of the proofs in the individual theories. "Size", in this sense, refers to the number of inference steps.

## 4.2.1 LogicSets.nb

The name of this theory is actually slightly misleading: it consists of roughly 99% set theory and only 1% logic, namely a tiny little bit of $\lambda$-calculus ($\beta$-reduction, function composition). All other aspects of the logic underlying our formalization, including propositional and (higher-order) predicate logic ($\alpha$-equivalence, higher-order rewriting, etc.) is encoded on the meta level of Theorema, in *Mathematica*, and hence not part of our formalization at all.

The formalization of set theory, however, really starts from the very axioms of Zermelo-Fraenkel set theory (without the axiom of choice; the logic of Theorema contains a choice-binder $\epsilon$, though, that is equivalent to it). The axioms are almost exactly those that can be found in the literature on set theory, only that we distinguish between—and allow quantification over—sets and non-sets ("urelements" and proper classes), whereas in traditional set theory quantification is restricted to sets. The predicate `isSet` is used to determine whether an object is a set or not.

As an example consider the axiom of infinity, which reads as

$$\underset{\texttt{isSet}[N]}{\exists} \bigwedge \begin{cases} \emptyset \in N \\ \underset{n \in N}{\forall} \texttt{succ}[n] \in N \end{cases} \qquad \text{(\textit{Axiom of Infinity})}$$

where succ is the successor function on sets, defined as $\texttt{succ[}A\texttt{]} := A \cup \{A\}$.

After the axioms, the usual basic set-theoretic functions and relations are introduced: general and binary union, general and binary intersection (where the intersection over the empty set is defined to be the empty set), the subset relation, set abstractions, power sets, the aforementioned succ function, functions and relations (surjectivity, injectivity, quotient sets), and finally Kuratowski ordered pairs and Cartesian products. For each of these notions some simple properties are proved, e. g. that succ is injective, that Kuratowski ordered pairs possess all characteristic properties of ordered pairs, that binary union and -intersection satisfy some lattice properties, etc. We omit the details here, since all this follows more or less exactly every standard introduction to set theory in a text-book on the subject.

We do want to emphasize, however, that *least fixpoints* are formalized in LogicSets.nb as well. The least fixpoint of a function $f$ on a set $A$ is defined as the intersection of all sets $B \subseteq A$ with $f\texttt{[}B\texttt{]} \subseteq B$; if $f$ has certain properties (boundedness, monotonicity), the result really is a fixpoint of $f$, and this fact is also proved in the theory, of course. The reason why we needed least fixpoints in our formalization is twofold: once for defining *finite sets* (also in LogicSets.nb), and once for constructing the set of *natural numbers* as the smallest set that contains $\emptyset$ and the successor of every element (see Numbers.nb). Finite sets, in turn, are needed for instance in Polynomials.nb, because by definition the support of a polynomial must be finite.

LogicSets.nb is concluded by the definitions of isDomain, as discussed in Section 4.1.2, and the DomainSets-functor that simply returns the domain of all sets over its argument domain.

## 4.2.2 AlgebraicStructures.nb

As its name suggests, AlgebraicStructures.nb contains the definitions and some basic properties of the algebraic structures used in reduction ring theory, namely cancellative commutative monoids (this is what power-products in polynomial rings are), groups, rings, commutative rings with identity, and fields. These structures are defined as unary predicates on domains, e. g. commutative rings with identity are defined as

$$\underset{\mathcal{D}}{\forall}\ \mathtt{isCommRing1}[\mathcal{D}] :\Leftrightarrow \bigwedge \begin{cases} \mathtt{isRing}[\mathcal{D}] \\ \mathtt{isNeutral}[\underset{\mathcal{D}}{*}, \underset{\mathcal{D}}{1}, \mathcal{D}] \\ \mathtt{isComm}[\underset{\mathcal{D}}{*}, \mathcal{D}] \end{cases} \qquad (\textit{isCommRing1})$$

The auxiliary notions, like `isNeutral` and `isComm`, are also defined in this theory in the obvious way, for instance

$$\underset{\mathcal{D},\circ}{\forall}\ \mathtt{isComm}[\circ, \mathcal{D}] :\Leftrightarrow \underset{\underset{\mathcal{D}}{\in}[x,y]}{\forall}\ x \circ y = y \circ x \qquad (\textit{isComm})$$

It is important to note that *all* notions in AlgebraicStructures.nb are defined for *domains*, not for sets; in particular, the definition of `isComm` in (*isComm*) refers to $\underset{\mathcal{D}}{\in}[x, y]$ rather than $x, y \in \mathcal{D}$.

Apart from group- and ring-like structures and related concepts, the theory provides the definitions of various properties of binary relations, too. These include (partial/total/reflexive/strict) order relations, equivalence relations and Noetherian relations. Recall that a binary relation $\succ$ is Noetherian over a domain $\mathcal{D}$ iff every non-empty set $A$ of elements of $\mathcal{D}$ (i. e. element of `DomainSets`$[\mathcal{D}]$) has a minimal element w. r. t. the converse relation $\prec$; in short, $\prec$ is well-founded on the carrier of $\mathcal{D}$.

Finally, AlgebraicStructures.nb also introduces the concept of *quotient domains*. To that end, a functor called `QuoDom` is defined that maps a domain $\mathcal{D}$ and a binary relation $\sim$ to the quotient domain $\mathcal{Q}$ over $\mathcal{D}$ w. r. t. $\sim$, i. e. the elements of $\mathcal{Q}$ are precisely the equivalence classes of elements of $\mathcal{D}$. In addition, the four higher-order functions `Fun1`, `Fun2`, `Rel1` and `Rel2` are defined in $\mathcal{Q}$, mapping unary and binary functions in $\mathcal{D}$, and unary and binary relations on $\mathcal{D}$, respectively, to the corresponding functions and relations in $\mathcal{Q}$. For instance, `Fun2` is defined as

$$\underset{f}{\forall}\ \underset{\mathcal{Q}}{\mathtt{Fun2}}[f] := \underset{X,Y}{\lambda} \bigcup_{x \in X} \{f[x, y] \mid_{y \in Y} \} \qquad (\textit{Fun2})$$

It is important to note that `Fun2`$[f]$ is always well-defined, regardless of whether $\sim$ is a congruence relation for $f$ or not (because it does not even appear in the definition). If $\sim$ *is* a congruence relation for $f$, though, then it is possible to prove the crucial identity

$$\underset{f,\ \underset{\mathcal{D}}{\in}[x,y]}{\forall} \mathtt{Fun2}_{\mathcal{Q}}[f][\mathtt{EC}_{\mathcal{Q}}[x], \mathtt{EC}_{\mathcal{Q}}[y]] = \mathtt{EC}_{\mathcal{Q}}[f[x,y]] \qquad \textit{(Fun2 EC QuoDom)}$$

that is usually taken as the definition of binary functions in quotient domains. $\mathtt{EC}_{\mathcal{Q}}[x]$ denotes the equivalence class of $x$ modulo $\sim$.

In our formalization, quotient domains are needed in Numbers.nb for constructing the domain of integers as the quotient of pairs of natural numbers modulo some equivalence relation, see below.

### 4.2.3 Numbers.nb

This theory introduces the sets of natural numbers and integers, together with the usual arithmetical functions and relations, in a systematic way. Indeed, the principle employed here can easily be extended to construct the rationals, reals, and even complex numbers in pretty much the same spirit on top of the integers as well. The only reason why this has not happened yet is that we do not need them in our formalization.

First, the set of *set-theoretic* natural numbers, called `natSet`, is defined by a least fixpoint construction as the smallest inductive set containing $\emptyset$ and $\mathtt{succ}[n]$ for every $n$ in the set.[2] Then, the usual operations on natural numbers, namely addition, pseudo-subtraction, multiplication and the order relation are introduced as functions of the functor `Nat` by interpreting the respective operator symbols, and all their characteristic properties are proved; this, in particular, also includes the usual induction rule as a higher-order formula. Hence, `natSet` and `Nat` are not defined by means of axioms, but they are constructed in a purely set-theoretic manner. However, they only serve as the basis for the further construction of integers, as described in the subsequent paragraphs, and are never used outside this theory.

Next, the `PreInt` functor is introduced. This functor returns the domain of Kuratowski ordered pairs $(m, n)$ of elements of `Nat`, with the intended meaning that $(m, n)$ represents the integer $m - n$. `PreInt` also defines the various arithmetical operations on such pairs, as well as the equivalence relation $\sim$ that expresses that two pairs actually represent the same integer, namely $(m, n) \sim (k, l) :\Leftrightarrow m + l = k + n$.

On top of that, the set-theoretic integers are defined as the quotient domain `Int` of `PreInt` modulo $\sim$; the `QuoDom` functor from AlgebraicStructures.nb is made use of for that. Having constructed the set-theoretic integers, the set of "real" integers $\mathbb{Z}$ is finally obtained by wrapping each set-theoretic integer

[2] "Set-theoretic" here means that everything, in particular the elements of `natSet`, is a set.

into the free “datatype” constructor `INT` in theory NatInt.nb, only to ensure that integers are no sets. This has the advantage that functions can be overloaded for sets and integers without ambiguity or inconsistency; a similar approach is pursued when introducing tuples in Tuples.nb. Finally, the set of “real” natural numbers, $\mathbb{N}_0$, is simply obtained from the set $\mathbb{Z}$ by separation, i. e. as $\mathbb{N}_0 := \{n \in \mathbb{Z} \mid n \geq 0\} \subset \mathbb{Z}$.

Summarizing, any integer in our formalization is internally represented by something like $\mathtt{INT}[\underset{\mathtt{QuoDom[PreInt,\sim]}}{\mathtt{EC}}[(m,n)]]$, although in NatInt.nb we introduce the usual literal abbreviations $0$, $1$, $-1$ and $2$ for the most frequently used integers. After all, internal representations become immaterial anyway once enough properties of the objects in question have been proved.

## 4.2.4 NatInt.nb

The contents of this theory can be described easily: NatInt.nb states and proves hundreds of lemmas, some absolutely trivial, others a bit more involved, about addition, subtraction, multiplication and the order relation on the set of integers. This includes the proof that $(\mathbb{Z}, +, \cdot)$ constitutes an integral domain, that $\geq$ is a discrete linear order relation, how it is related to addition and multiplication, and many more. Also, a couple of results about interval arithmetic are provided, for instance

$$\underset{a \in \mathbb{N}_0}{\forall}\ \underset{i}{\forall}\quad i \in \mathbb{N}_{1,\ldots,a-1} \Rightarrow (a-i) \in \mathbb{N}_{1,\ldots,a-1} \qquad (\mathbb{N}\ \textit{intervals 18})$$

Most of the results in this theory are needed in connection with tuples, since tuples are of course indexed by natural numbers. Others are needed in the theories Integers.nb and IntegerQuotientRings.nb, in the proofs that $\mathbb{Z}$ and $\mathbb{Z}_k$ can be turned into algorithmic reduction rings.

## 4.2.5 NatIntExtended.nb

NatIntExtended.nb, as its name suggests, extends the theory of natural numbers and integers by further number-theoretic concepts, such as sign and absolute value, quotient and remainder, the greatest common divisor, and finite sums in arbitrary domains.

First, however, the notion of *primitive recursion* on $\mathbb{N}_0$ is introduced by means of the binary higher-order function `natRec0`, which is first defined by a least fixpoint construction and then shown to satisfy the two characteristic identities

$$\underset{b,c}{\forall}$$
$$\texttt{natRec0}[b,c][0] = b \qquad (\textit{natRec0 base})$$
$$\underset{n\in\mathbb{N}_0}{\forall}\ \texttt{natRec0}[b,c][n+1] = c[n, \texttt{natRec0}[b,n]] \qquad (\textit{natRec0 step})$$

that allow for expressing primitive recursive functions on $\mathbb{N}_0$ by a single term. Although primitive recursive functions, and even arbitrary recursive functions, may simply be defined in Theorema by a collection of recurrence equations (each in an individual formula) without caring about termination of the functions and consistency of the definitions, there are also situations where natRec0 (and related functions for primitive recursion on tuples, for instance) is absolutely necessary. In proofs, quantified variables can only be instantiated by single terms, so when the instance happens to be a primitive recursive function, natRec0 is needed.

Afterward, sign and absolute value are introduced, and some simple lemmas are proved (the triangle inequality, for instance). Both notions are restricted to $\mathbb{Z}$, as this is the only number domain included in our formalization; in the future, though, the definitions might be extended to encompass also rationals, reals, and complex numbers.

Next, division with quotient and remainder and divisibility are introduced by means of the two functions quo and mod and the binary relation $\mid$. It must be noted that both quo$[a,b]$ and mod$[a,b]$ are defined implicitly to be the integers $q$ and $r$ satisfying $a = q\,b + r$ and $0 \leq \texttt{sign}[b]\,r < |b|$ (the existence of such integers had to be proved first, of course); no appeal to any algorithmic treatment of division with remainder is made. Apart from some simple lemmas about quo and mod, e. g. what they are for certain specific values of $a$ and $b$, various congruence properties of mod are proved as well, which are needed in IntegerQuotientRings.nb.

Having division and divisibility in $\mathbb{Z}$, now the greatest common divisor and least common multiple of two integers are introduced. The former is defined implicitly, requiring an existence proof first, whereas the latter is defined in terms of the former as $\texttt{lcm}[a,b] := \texttt{quo}[|a\,b|, \texttt{gcd}[a,b]]$. The main results about gcd proved in this theory are Bézout's identity, that multiplication distributes over gcd, and that gcd$[a,b]$ divides mod$[c\,b,a]$ provided that $a \neq 0$. The only theories using gcd are Integers.nb and IntegerQuotientRings.nb.

Finally, the theory is concluded by the definition of finite sums in arbitrary domains $\mathcal{D}$. Finite sums play an important role in reduction ring theory, see for instance Axiom (R5) in Section 3.3. NatIntExtended.nb defines them inductively in terms of $\underset{\mathcal{D}}{0}$ and $\underset{\mathcal{D}}{+}$ using natRec0, and proves some useful identities, like

$$\underset{\mathcal{D},f}{\forall}\ \underset{a,b\in\mathbb{N}_0}{\forall}$$
$$\mathtt{isMonoid}[\mathcal{D}] \wedge \underset{i=1,\ldots,a+b}{\forall} \underset{\mathcal{D}}{\in}[f[i]] \Rightarrow$$
$$\sum_{i=1,\ldots,a+b}^{\mathcal{D}} f[i] = \sum_{i=1,\ldots,a}^{\mathcal{D}} f[i] \underset{\mathcal{D}}{+} \sum_{i=1,\ldots,b}^{\mathcal{D}} f[a+i] \qquad (\Sigma\ \textit{splitting})$$

### 4.2.6 Tuples.nb

As its name suggests, this theory introduces tuples (i. e. lists of arbitrary finite length) and some well-known operations on them, and proves lots of helpful lemmas. Although tuples could in principle be defined by analogy with natural numbers by purely set-theoretic means as the smallest inductive set containing the empty tuple `Nil` and `Cons`$[a, t]$ for every $t$ in the set, this approach unfortunately turned out to be impossible in our case. The reason is quite simple: we do not restrict the elements of tuples to belong to certain underlying sets, i. e. the $a$ in `Cons`$[a, t]$ above could be pretty much anything, and this prevents us from setting up a fixpoint construction (fixpoint constructions are *always* w. r. t. an underlying set). Hence, we had to follow an axiomatic approach where we simply state the characteristic properties of `Nil` and `Cons` as axioms; in particular, we require them to be free (i. e. injective and distinct) "data-type" constructors not returning sets. The reason for the latter is the same as for wrapping `INT` around set-theoretic integers in NatInt.nb: it shall be possible to define functions once for set-arguments and once for tuple-arguments without introducing any inconsistencies. In fact, in our formalization we define the reduction relation in reduction rings (see Definition 3.2.2) both for sets and for tuples, because even though sets are more convenient to deal with in connection with the theoretical aspects of the theory (Main Theorem etc.), tuples perform better when it comes to algorithms and computations.

The operations on tuples that are included in Tuples.nb are the following, all of them being defined in terms of the two "constructors" `Nil` and `Cons`: length, concatenating tuples, appending an element after the last element of a tuple, prepending an element before the first element of a tuple, dropping the first or last element of a tuple, and reverting the order of elements. Moreover, analogous to set abstractions we formally introduced *tuple abstractions* as well; they are defined inductively making use of `natRec0`. Of course, the theory does not only consist of the definitions of all these notions, but also of hundreds of lemmas, most of them being more or less obvious for the experienced mathematician.

Besides the usual first-order properties of the operations on tuples, we also proved some very useful higher-order lemmas. Apart from induction on tuples, this includes lemmas of the following kind:

$$\underset{\texttt{isTuple}[T],\ a,\ P}{\forall}\ \underset{S=T\frown a}{\texttt{let}}\ \left(\underset{i=1,\ldots,|S|-1}{\forall} P[S_i, S_{i+1}]\right) \Leftrightarrow \bigwedge\begin{cases}\underset{i=1,\ldots,|T|-1}{\forall} P[T_i, T_{i+1}]\\ T \neq \langle\rangle \Rightarrow P[T_{|T|}, a]\end{cases} \qquad (\textit{append P binary})$$

The theory ends with the definition of the `DomainTuples`-functor that, analogous to `DomainSets`, simply returns the domain of all tuples over its argument domain.

## 4.2.7 Sequences.nb

In contrast to integers and tuples, sequences are not defined as a separate entity at all, neither axiomatically nor in a set-theoretic manner. Instead, basically *every* object $s$ may be regarded a sequence, simply by viewing it as a function whose domain is $\mathbb{N}$ (0 excluded!) and forming expressions of the form $s$`[1]`, $s$`[2]` etc. where it does not matter whether $s$ is "defined" for an $n \in \mathbb{N}$ or not—if not, the corresponding sequence element is literally $s$`[`$n$`]`.

However, in order to cut down this huge degree of freedom a bit, the very first notion introduced in Sequences.nb is that of the `DomainSequences`-functor. For a given argument $\mathcal{D}$, this functor returns the domain containing precisely those sequences whose elements are in $\mathcal{D}$, analogous to `DomainSets` and `DomainTuples`. Most of the main results about sequences contained in this theory are actually stated only for such `DomainSequences`-elements.

Many of said results prove the existence or non-existence of certain sequences or sub-sequences, illustrated by the following two examples:

$$\underset{\mathcal{D},\ P}{\forall}\ \underset{\underset{\texttt{DomainSequences}[\mathcal{D}]}{\in}[f]}{\forall}\left(\underset{a\in\mathbb{N}}{\forall}\ \underset{\substack{i\in\mathbb{N}\\ i\geq a}}{\exists}\ \underset{\substack{j\in\mathbb{N}\\ j>i}}{\forall} P[f[i], f[j]]\right) \Rightarrow \underset{\underset{\texttt{DomainSequences}[\mathcal{D}]}{\in}[h]}{\exists} h \lhd_{|} f \wedge \underset{\substack{i,j\in\mathbb{N}\\ j>i}}{\forall} P[h[i], h[j]] \qquad (\textit{subSeq DomainSequences 5})$$

$$\left(\neg \underset{\underset{\texttt{DomainSequences}[\mathcal{D}]}{\in}[f]}{\exists}\ \underset{i\in\mathbb{N}}{\forall} P[f[i], f[i+1]]\right) \Rightarrow$$

$$\underset{\substack{\in\\ \texttt{DomainSequences}[\mathcal{D}]}}{\forall}\texttt{[}f\texttt{]}\ \underset{i\in\mathbb{N}}{\exists}\ \underset{\substack{j\in\mathbb{N}\\ j>i}}{\forall}\ \neg P\texttt{[}f\texttt{[}i\texttt{]},f\texttt{[}j\texttt{]]} \qquad \text{(\textit{non-existence seq P binary})}$$

where $h \lhd\!\shortmid f$ means that $h$ is a sub-sequence of $f$.

In addition, we also proved an alternative description of Noetherianity of a relation $\succ$ in terms of the non-existence of infinite strictly ascending chains. This, as well as basically all other results proved in Sequences.nb, is only needed to deal with Axiom (R13) of reduction ring theory, which is concerned with the non-existence of certain sequences of subsets (i. e. `DomainSets`) of the rings in question. To that end, we also introduced the notion of *accumulating set sequence* of a given sequence $f$, called `setSeq[`$f$`]`, whose $n$-th element is the set consisting of the first $n$ elements of $f$. When proving (R13) in concrete rings it turned out that often it is better to first prove the non-existence of such accumulating set sequences that would violate the axiom, and from this conclude the non-existence of arbitrary violating sequences.

### 4.2.8 Functors.nb

The last elementary theory, Functors.nb, only contains a couple of generic functors for constructing products of given domains (whose elements are then tuples). In particular, the functor `LexOrder` takes two input domains $\mathcal{D}_1$ and $\mathcal{D}_2$ together with two symbols $\rhd$ and $\gg$ and returns a new domain $\mathcal{L}$ whose carrier is the direct product of the carriers of $\mathcal{D}_1$ and $\mathcal{D}_2$ and which interprets the symbol $\succ$ as the lexicographic combination of $\underset{\mathcal{D}_1}{\rhd}$ and $\underset{\mathcal{D}_2}{\gg}$; the sample functor in Section 2.2 is nearly the same. Moreover, the theory also contains the functor `LexOrder3` that combines three domains lexicographically, although this would not really be necessary (the same effect can be achieved by iterating `LexOrder`). The crucial results proved for these two functors are conservation theorems stating that partial and/or Noetherian order relations in $\mathcal{D}_1$ and $\mathcal{D}_2$ make $\underset{\mathcal{L}}{\succ}$ partial and/or Noetherian as well.

Lexicographic orders are needed in GroebnerRings.nb for proving the termination of Buchberger's algorithm. This proceeds by finding a Noetherian order relation on the arguments of the function that can be shown to decrease in every recursive call. In our case of Buchberger's algorithm, the desired relation is the lexicographic combination of three basic relations, as can be seen in Section 4.4.3. Furthermore, general direct products (not necessarily ordered) are needed in PolyTuples.nb for proving Dickson's lemma in power-product domains.

## 4.3 Reduction Ring Theories

The seven Theorema theories listed in this section are all directly related to reduction rings.

### 4.3.1 ReductionRings.nb

This theory is the core of our formalization. As its title suggests, it consists of the definitions of plain and algorithmic reduction rings and the various notions related to them (reduction, ideal congruence, etc.) together with the proofs of the central theorems of all of reduction ring theory, in particular the Main Theorem (Theorem 3.5.3) and the fact that ideal congruence coincides with the symmetric-reflexive-transitive closure of the reduction relation. Please note that ReductionRings.nb mainly considers the purely algebraic, *non-algorithmic* aspects of reduction rings, in the sense that no single algorithm is implemented here. Instead, the algorithmic aspects can be found in GroebnerRings.nb.

Because of its significance, we present this theory in more detail. The variable $\mathcal{D}$ in each formula presented in this subsection is universally quantified, although this is not written out explicitly.

**Notions related to reduction rings.** ReductionRings.nb begins with the introduction of the various notions related to reduction rings, or, more precisely, those notions that can be defined in terms of the ring operations, the order relation $\succ$, the sets $I_c = \{1, \ldots, \texttt{multN}[c]\}$ and the sets of multipliers $M_c^i$ (characterized by the predicate $\texttt{mult}[c, m, i]$). Here it is important to note that all these operations $\succ$, multN, mult, as well as the new notions defined in terms of them, depend on the universally quantified underlying domain $\mathcal{D}$ and hence are domain operations of $\mathcal{D}$, meaning that they actually carry the under-script $\mathcal{D}$.

As has already been indicated, the reduction relation (and also some other notions) is defined both for sets and for tuples of elements of $\mathcal{D}$, i. e. as

$$\underset{a,\,b}{\forall}$$

$$\underset{\underset{\texttt{DomainSets}[\mathcal{D}]}{\in}[C]}{\forall} \quad a \underset{\mathcal{D}}{\rightarrow_C} b \;:\Leftrightarrow\; \underset{c \in C}{\exists} \; \underset{\underset{\mathcal{D}}{\in}[m]}{\exists} \; a \underset{\mathcal{D}}{\rightarrow_{m,c}} b \qquad (\textit{reduction modulo set})$$

$$\underset{\underset{\texttt{DomainTuples}[\mathcal{D}]}{\in}[T]}{\forall} \quad a \underset{\mathcal{D}}{\rightarrow_T} b \;:\Leftrightarrow\; \underset{i=1,\ldots,|T|}{\exists} \; \underset{\underset{\mathcal{D}}{\in}[m]}{\exists} \; a \underset{\mathcal{D}}{\rightarrow_{m,T_i}} b \qquad (\textit{reduction modulo tuple})$$

Stating and proving purely algebraic results is definitely easier when working with sets, as there is no need to mess about with indices and the order of elements.

Also, using sets is certainly more intuitive and closer to text-book mathematics. However, as soon as algorithms and computations come into play, tuples turn out to be more suitable; in particular, they allow to explicitly specify the order of their elements, which in some situations is highly desirable, for instance when taking issues of efficiency into account. The obvious but nonetheless crucial relation between reduction modulo sets and reduction modulo tuples is of course the following:

$$\underset{a,\,b}{\forall}\ \underset{\mathtt{[}T\mathtt{]}\ \in\ \mathtt{DomainTuples[}\mathcal{D}\mathtt{]}}{\forall}\quad a \underset{\mathcal{D}}{\rightarrow_T} b \Leftrightarrow a \underset{\mathcal{D}}{\rightarrow_{\mathtt{TupleToSet[}T\mathtt{]}}} b \qquad (\rightarrow \textit{TupleToSet})$$

where `TupleToSet` simply returns the set of all elements of its argument. Note that the various closures of the reduction relation are defined for arbitrary reductor $r$, obviating duplicate work for sets *and* tuples.

Right after introducing all the auxiliary notions, many simple facts are proved about them. About half of these facts does not put any constraints on the underlying domain $\mathcal{D}$ at all, as for example ($\rightarrow$ *TupleToSet*), whereas the other half requires $\mathcal{D}$ to be a commutative ring with multiplicative identity. In any case, no appeal to reduction rings or individual reduction ring axioms is made—this would not even make sense, because they have not been introduced yet.

**Reduction rings and algorithmic reduction rings.** Next, the classes of reduction rings and algorithmic reduction rings are introduced. For this, first the individual axioms are defined as unary predicates on domains, e. g. in the case of (R4) as

$$\mathtt{R4[}\mathcal{D}\mathtt{]}\ :\Leftrightarrow\ \underset{\underset{\mathcal{D}}{\in}\mathtt{[}a\mathtt{]}}{\forall}\ a \neq \underset{\mathcal{D}}{0} \Rightarrow a \underset{\mathcal{D}}{\succ} \underset{\mathcal{D}}{0} \qquad (\textit{R4})$$

Reduction rings and algorithmic reduction rings are then merely the conjunction of all axioms, e. g.

$$\mathtt{isReductionRing[}\mathcal{D}\mathtt{]}\ :\Leftrightarrow\ \bigwedge \begin{cases} \mathtt{isCommRing1[}\mathcal{D}\mathtt{]} \\ \mathtt{isNoetherian[}\underset{\mathcal{D}}{\succ}\mathtt{,}\mathcal{D}\mathtt{]} \\ \mathtt{R0[}\mathcal{D}\mathtt{]} \\ \vdots \end{cases} \qquad (\textit{reduction ring})$$

Separating the definitions of the axioms from the definitions of (algorithmic) reduction rings has the advantage that the axioms can easily be dealt with individually, both when proving them for concrete rings and when having them as assumptions of lemmas. Indeed, right after introducing reduction rings, further lemmas about the reduction ring notions introduced before are stated and proved, but now assuming that $\mathcal{D}$ is not only a commutative ring with identity, but also that it satisfies *some* reduction ring axioms (namely precisely those that are really needed in the lemmas). The advantage of listing the required axioms explicitly over making the sweeping assumption of $\mathcal{D}$ being a reduction ring is quite obvious: when proving that a concrete ring is a reduction ring, after having proved some of the axioms already, those general lemmas that depend only on the proved axioms may be used to prove the remaining ones. And this technique, in fact, was employed extensively when showing that polynomial rings over reduction rings are reduction rings themselves.

There is another lesson we have learned from definitions as the one of reduction rings, where the right-hand-side is a big conjunction (or, analogously, a disjunction), like $P :\Leftrightarrow C_1 \wedge \ldots \wedge C_n$. In such cases it is good practice to immediately prove, for each $C_i$, a *destruction rule* of the form $P \Rightarrow C_i$ that allows to infer $C_i$ from $P$ in proofs. Otherwise, without such destruction rules, the very definition of $P$ has to be unfolded, which is all but robust. Our own experience with the formalization of reduction ring theory shows that quite often one of the conjuncts $C_i$ has to be slightly changed, even if the definition of $P$ has already been used in proofs to infer some of the other conjuncts. In such situations, all proofs have to be re-done if $C_i$ is changed—even though in fact $C_i$ does not affect the proofs at all. With destruction rules the unfolding of definitions can be avoided in the vast majority of cases, making the exploration of theories more robust.

**Main results.** Finally, the main results of reduction ring theory are stated and proved: the Main Theorem (Theorem 3.5.3) and the fact that ideal congruence coincides with the symmetric-reflexive-transitive closure of the reduction relation (Theorem 3.5.7). To that end, first the notions of Gröbner basis, Church-Rosser property and "connectible critical pair" (Definition 3.5.1) are introduced; the latter is called `cpConnectible` in the formalization.

Before turning to the Main Theorem, a couple of simple properties of `cpConnectible` are proved, some of them even for arbitrary $\mathcal{D}$. Furthermore, since the Main Theorem depends on the generalized Newman lemma [WB83], this lemma is stated and proved, too. Although it holds in a very general setting, for arbitrary relations $\rightarrow$ and $\succ$ where $\rightarrow \subseteq \succ$ and $\succ$ is Noetherian, we only prove it for the special case of the reduction- and order relation in reduction rings.

The Main Theorem, stating that $G$ is a Gröbner basis iff for all $g_1, g_2 \in G\backslash\{0\}$ we have $\mathrm{cpc}(g_1, g_2, G)$, is proved by splitting it into several lemmas: one lemma stating that pairs originating from *trivial* common reducibles $c$ of $g_1$ and $g_2$ can always be connected below $c$, one lemma stating that $\mathrm{cpc}(g_1, g_2, G)$ implies that the pairs originating from *non-trivial* common reducibles $a$ of $g_1$ and $g_2$ can be connected below $a$ if $g_1 \neq g_2$, one lemma stating the same for $g_1 = g_2$, and a couple of lemmas that combine these results. The overall structure resembles that of the paper-and-pencil proof given in [Buc83a, Sti85], so we omit the details here. The whole elaboration eventually culminates in

$$\mathtt{isReductionRing}[\mathcal{D}] \Rightarrow$$

$$\underset{\underset{\mathtt{DomainSets}[\mathcal{D}]}{\in}[G]}{\forall} \underset{\mathcal{D}}{\mathtt{cpConnectible}}[G] \Rightarrow \underset{\mathcal{D}}{\mathtt{isGroebnerBasis}}[G] \quad (\textit{Main Theorem})$$

where $\underset{\mathcal{D}}{\mathtt{cpConnectible}}[G]$ is a shorthand for $\underset{a,b\in G}{\forall} \underset{\mathcal{D}}{\mathtt{cpConnectible}}[a, b, G]$. Although the theorem is apparently formulated for sets, it is rephrased for tuples in a corollary, for being able to prove the correctness of Buchberger's algorithm as described in Section 4.4.5.[3]

Finally, the theory concludes with the proof that $\equiv_B$ and $\leftrightarrow_B^*$ are identical, for every set *or tuple* $B$; the formula whose proof is presented in Appendix A is one of the lemmas needed for establishing said identity. To that end, also the notion of ideal is introduced (again for both sets and tuples), and some general ideal-theoretic results fully independent of reduction ring theory are proved, e. g. that ideals are closed under addition and subtraction.

## 4.3.2 Fields.nb

This theory contains a functor, `ReductionField`, that takes an input domain $K$ and extends it by those operations that are required to turn $K$ into an algorithmic reduction ring, according to Section 3.4.1. Following common practice in Theorema $K$ is not required to possess any particular algebraic structure, but if it is a field, then the extension really *is* an algorithmic reduction ring—otherwise nothing can be said about it. The proof of this claim constitutes the core part of Fields.nb, and actually is quite straight-forward, as can be seen from comparatively small number of proved formulas in this theory in Fig. 4.2.

In addition to the purely theoretical contents, the theory also includes a couple of sample computations of reduction multipliers, least common reducibles

[3] `isGroebnerBasis` and `cpConnectible` are defined for sets and tuples alike.

and critical-pair multipliers in `ReductionField[`$\mathbb{Q}$`]` for the Theorema-built-in field of rational numbers $\mathbb{Q}$.

### 4.3.3 Integers.nb

This theory contains a functor, `ReductionIntegers`, that does not take any arguments but simply constructs a new domain whose carrier is $\mathbb{Z}$ and that provides the additional functions and relations for turning $\mathbb{Z}$ into an algorithmic reduction ring, as described in Section 3.4.2. The proof of this claim, split across several lemmas, is part of the theory as well, and it is definitely more involved than the corresponding proof in the case of fields. Part of its complexity stems from the fact that often many cases based on the signs of some numbers have to be distinguished. However, the overall structure of the proof closely resembles the one in [Buc83a], so we omit the details here.

As in Fields.nb, a handful of sample computations of reduction multipliers etc. in `ReductionIntegers[]` are included in the theory, too, building upon the computational capabilities of the Theorema-built-in domain of integers $\mathbb{Z}$.

### 4.3.4 IntegerQuotientRings.nb

This theory contains a functor, `ReductionIQR`, that takes a positive integer $k$ and constructs a new domain whose carrier is the set $\{0, \ldots, k-1\}$ and that provides the additional functions and relations for turning $\mathbb{Z}_k$, represented by $\{0, \ldots, k-1\}$, into an algorithmic reduction ring, following Section 3.4.3. As in Fields.nb and Integers.nb, the proof of this claim, split across several lemmas, is of course included in the theory as well. Again, it proceeds in very much the same way is the original proof in [Sti85]. It is interesting to observe that although turning $\mathbb{Z}_k$ into a reduction ring is conceptually more delicate than $\mathbb{Z}$ because of the presence of zero divisors, fewer auxiliary results are needed in IntegerQuotientRings.nb than in Integers.nb. This is due to the fact that the reduction ring ordering is much simpler in $\mathbb{Z}_k$ than in $\mathbb{Z}$.

The theory is concluded by some sample computations in `ReductionIQR[13]` and `ReductionIQR[24]`.

### 4.3.5 Polynomials.nb

This theory defines the class of *abstract* reduction polynomial domains over a coefficient domain $\mathcal{R}$ and a power-product domain $\mathcal{T}$. “Abstract”, in this sense, means that no concrete representation of polynomials, e. g. as tuples of monomials, is presumed, but polynomials are solely characterized by a coefficient function `C` mapping power-products to coefficients.

First, the class of *abstract* commutative power-product domains, specified by the unary predicate `isCommPPDomain`, is introduced. A domain $\mathcal{T}$ belongs to this class iff it is a multiplicative cancellative commutative monoid, the interpretation of $\succ$ in $\mathcal{T}$ is a total Noetherian ordering with $\underset{\mathcal{T}}{1}$ as its least element and which is monotonic w. r. t. multiplication, $\mathcal{T}$ defines the notions of divisibility ($\mid$) and division (/) in the usual way, and any two elements $s, t$ have a least common multiple (making $\mathcal{T}$ a join-semilattice). It is important to note that no appeal to "indeterminates" or "dimension" is made. Only when proving that (R13) is preserved in polynomial domains, $\mathcal{T}$ is further assumed to satisfy the "Dickson property", meaning that for every infinite sequence $S$ of power-products there exist $i < j$ with $S_i \underset{\mathcal{T}}{\mid} S_j$.

Afterward, the class of abstract reduction polynomial domains, specified by the unary predicate `isReductionPolynomialDomain`, is introduced. A domain $\mathcal{P}$ belongs to this class iff it interprets `C`, $\succ$, `multN`, `mult` and the usual ring operations, and the resulting constants and operations satisfy certain properties, as listed below. Although these properties naturally depend on the underlying coefficient- and power-product domain, `isReductionPolynomialDomain` is a *unary* predicate depending only on $\mathcal{P}$: throughout the whole theory, the coefficient domain is fixed to $\overline{\mathcal{R}}$ and the power-product domain is fixed to $\overline{\mathcal{T}}$, where $\overline{\mathcal{R}}$ and $\overline{\mathcal{T}}$ are "arbitrary but fixed" constants. Moreover, all theorems are stated and proved for the arbitrary but fixed reduction polynomial domain $\overline{\mathcal{P}}$ rather than for all such domains.[4] This approach, which is somehow in the spirit of the *locale system* of the Isabelle proof assistant [Bal10], simplifies matters (no need to instantiate universally quantified variables in proofs) and eases notation.

The properties that must be satisfied by $\underset{\mathcal{P}}{\mathtt{C}}$ etc. in order for $\mathcal{P}$ to be a reduction polynomial domain are as follows:

- The carrier of $\mathcal{P}$, specified by $\underset{\mathcal{P}}{\in}$, must be such that it contains $\underset{\mathcal{P}}{0}$ and $\underset{\mathcal{P}}{1}$, that it is closed under addition and subtraction in $\mathcal{P}$, and that every $p$ in it has finite support, i. e. $\underset{\mathcal{P}}{\mathtt{C}}[p,t]$ is different from $\underset{\mathcal{P}}{0}$ only for finitely many $t$ with $\underset{\mathcal{T}}{\in}[t]$. Furthermore, for every coefficient $c$ and power-product $t$ it must contain a *monomial* $m$ with $\underset{\mathcal{P}}{\mathtt{C}}[m,t] = c$ and $\underset{\mathcal{P}}{\mathtt{C}}[m,s] = \underset{\mathcal{R}}{0}$ for all $s \neq t$. In the theory, such monomials play an important role and are denoted by $c \underset{\mathcal{P}}{\cdot} t$.
- For fixed $t$, the coefficient function $\underset{\mathcal{P}}{\mathtt{C}}$ must be a group homomorphism w. r. t. addition, e. g. $\underset{\mathcal{P}}{\mathtt{C}}[p \underset{\mathcal{P}}{+} q, t] = \underset{\mathcal{P}}{\mathtt{C}}[p,t] \underset{\mathcal{R}}{+} \underset{\mathcal{P}}{\mathtt{C}}[p,t]$. Moreover, the coeffi-

[4]In the formalization, the constants are just called $\mathcal{R}$, $\mathcal{T}$ and $\mathcal{P}$.

cients of $\underset{\mathcal{P}}{0}$ and $\underset{\mathcal{P}}{1}$ must be as expected.

- Multiplication in $\mathcal{P}$ must be defined in the usual way, satisfying the recursion $p \underset{\mathcal{P}}{*} q = \underset{\mathcal{P}}{\mathtt{lm}}[p] \underset{\mathcal{P}}{\otimes} q \underset{\mathcal{P}}{+} \underset{\mathcal{P}}{\mathtt{R}}[p] \underset{\mathcal{P}}{*} q$ if $p$ is not zero, where $\underset{\mathcal{P}}{\otimes}$ is multiplication with a monomial and $\underset{\mathcal{P}}{\mathtt{lm}}[p]$ and $\underset{\mathcal{P}}{\mathtt{R}}[p]$ refer to the *leading monomial* and *remainder* (i. e. $p \underset{\mathcal{P}}{-} \underset{\mathcal{P}}{\mathtt{lm}}[p]$) of $p$ (w. r. t. the ordering on $\overline{\mathcal{T}}$), respectively.

- $\mathcal{P}$ must be *extensional* in the sense that two of its elements are equal if they have the same coefficients.

- The order relation $\underset{\mathcal{P}}{\succ}$ and the multiplier-related notions must be defined as explained in Section 3.4.4.

All auxiliary functions, such as $\underset{\mathcal{P}}{\cdot}$, $\underset{\mathcal{P}}{\otimes}$, $\underset{\mathcal{P}}{\mathtt{lm}}$, $\underset{\mathcal{P}}{\mathtt{R}}$ etc. are defined solely in terms of the coefficient function, zero, addition and subtraction (the latter three both in $\mathcal{P}$ and in $\overline{\mathcal{R}}$).

After having introduced the class of reduction polynomial domains, the class of *algorithmic* reduction polynomial domains, specified by the unary predicate `isAlgoReductionPolynomialDomain`, is introduced. A domain $\mathcal{P}$ belongs to this class iff it belongs to the class of reduction polynomial domains and moreover interprets `lcrd` (both the four-argument- and the one-argument version), `rdm` and `cpm` according to Section 3.4.4.

In total, three main theorems are proved in Polynomials.nb: assuming that $\overline{\mathcal{P}}$ is a reduction polynomial domain, we show that it constitutes a commutative ring with identity if $\overline{\mathcal{R}}$ does and that it constitutes a (plain) reduction ring if $\overline{\mathcal{R}}$ does. Furthermore, we prove that if $\overline{\mathcal{P}}$ is even an algorithmic polynomial domain and $\overline{\mathcal{R}}$ an algorithmic reduction ring, then $\overline{\mathcal{P}}$ is an algorithmic reduction ring as well. In either case, $\overline{\mathcal{T}}$ is assumed to be a commutative power-product domain. Although we prove these theorems for the arbitrary but fixed constants $\overline{\mathcal{R}}$, $\overline{\mathcal{T}}$ and $\overline{\mathcal{P}}$, in the end we generalize them to *all* domains using a meta-argument, eventually leading to

$$\underset{\mathcal{F},\,\mathcal{R},\,\mathcal{T}}{\forall} \; \bigwedge \left\{ \begin{array}{c} \mathtt{isAlgoReductionPolynomialDomain}[\mathcal{F}[\overline{\mathcal{R}},\overline{\mathcal{T}}]] \\ \mathtt{isAlgoReductionRing}[\mathcal{R}] \\ \mathtt{isCommPPDomain}[\mathcal{T}] \end{array} \right. \Rightarrow$$

$$\mathtt{isAlgoReductionRing}[\mathcal{F}[\mathcal{R},\mathcal{T}]] \qquad \text{(\textit{Conservation Theorem Polynomials})}$$

where $\mathcal{F}$ is supposed to be a functor mapping a coefficient domain and a power-product domain to a polynomial domain. It is important to see that in the first conjunct on the left-hand-side of the implication $\mathcal{F}$ is not applied to the universally quantified $\mathcal{R}$ and $\mathcal{T}$ but to the arbitrary but fixed $\overline{\mathcal{R}}$ and $\overline{\mathcal{T}}$, because `isAlgoReductionPolynomialDomain` is defined in terms of these constants.

### 4.3.6 PolyTuples.nb

Theory PolyTuples.nb builds upon Polynomials.nb but deals with the concrete representation of polynomials as ordered tuples of pairs of coefficients and power-products.

First, however, an intermediate layer of abstraction is inserted between the class of very abstract domains of commutative power-products from Polynomials.nb and any concrete representation of such domains (e. g. where power-products are represented by their exponent vectors). This intermediate layer is made up by yet another class of domains, specified by the unary predicate `isExpPPDomain`. A domain $\mathcal{S}$ belongs to this class iff its interpretation of `dim` is a natural number (including 0; this is the *dimension*, i. e. the number of indeterminates of $\mathcal{S}$), its interpretation of `Exp` is a function from $\{1, \ldots, \underset{\mathcal{S}}{\mathtt{dim}}\}$ to $\mathbb{N}_0$ (the *exponent function* of $\mathcal{S}$), and also interprets $*$, $1$, `lcm`, $\mid$ and $/$. Analogous to `isReductionPolynomialDomain`, these constants and operations need to satisfy certain compatibility conditions, e. g. $\underset{\mathcal{S}}{\mathtt{Exp}}[t \underset{\mathcal{S}}{*} s, i] = \underset{\mathcal{S}}{\mathtt{Exp}}[t, i] + \underset{\mathcal{S}}{\mathtt{Exp}}[s, i]$ for all $i \in \{1, \ldots, \underset{\mathcal{S}}{\mathtt{dim}}\}$.

Then, a functor called `OrderedPP` is introduced. This functor maps a domain $\mathcal{S}$ and a binary relation $\rhd$ to a new domain $\mathcal{T}$ extending $\mathcal{S}$ by interpreting $\succ$ by $\rhd$, i. e. all constants and operations in $\mathcal{T}$ are the same as in $\mathcal{S}$ and $\underset{\mathcal{T}}{\succ} := \rhd$. The crucial relation between all these notions is summarized in

$$\underset{\mathcal{S},\, \rhd}{\forall}$$

$$\mathtt{isExpPPDomain}[\mathcal{S}] \wedge \mathtt{isTermOrder}[\rhd, \mathcal{S}] \Rightarrow$$

$$\mathtt{isCommPPDomain}[\mathtt{OrderedPP}[\mathcal{S}, \rhd]] \qquad (\textit{OrderedPP isCommPPDomain})$$

where `isTermOrder[` $\rhd, \mathcal{S}$`]` simply expresses that $\rhd$ is a *term order* on $\mathcal{S}$ in the usual sense. This means that for any given domain $\mathcal{S}$ that is known to satisfy `isExpPPDomain` and any term order $\rhd$, we can be sure that applying `OrderedPP` to $\mathcal{S}$ and $\rhd$ yields a suitable power-product domain for (reduction) polynomial domains. Please note that (*OrderedPP isCommPPDomain*) contains

Dickson's lemma about the non-existence of certain sequences of tuples of natural numbers as a sub-result.

The reason for the separation of the term order from `isExpPPDomain` is that we might want to endow one and the same `isExpPPDomain` $\mathcal{S}$ with several different term orders, without having to prove again and again that the result really satisfies `isCommPPDomain`.

Now, there are two layers of abstract power-product domains: on the one hand there is `isCommPPDomain`, and on the other hand there is `isExpPPDomain` together with `OrderedPP`. What is still missing are concrete power-product domains and concrete term orders, and this is exactly what comes next in the theory. First, three concrete term orders are introduced: lexicographic (`Lex`), degree-lexicographic (`DegLex`) and degree-reverse-lexicographic (`DegRevLex`). All of them are parametrized over an underlying domain $\mathcal{S}$ and defined in terms of `dim` and `Exp` in $\mathcal{S}$, and all of them are immediately shown to really constitute term orders, e. g.

$$\underset{\texttt{isExpPPDomain}[\mathcal{S}]}{\forall} \quad \texttt{isTermOrder}[\texttt{Lex}[\mathcal{S}],\mathcal{S}] \qquad (\textit{Lex isTermOrder})$$

Next, concrete power-product domains are introduced by means of functor `PPTuples` that maps $n \in \mathbb{N}_0$ to the domain of power-products represented by tuples of natural numbers of length $n$ (i. e. exponent vectors). These domains are immediately shown to satisfy `isExpPPDomain`. Of course, other representations of power-products are possible, but not included in our formalization yet.

Afterward, functor `PolyTuples` is introduced. This functor takes two domains $\mathcal{R}$ and $\mathcal{T}$ and constructs the domain $\mathcal{P}$ over polynomials over coefficient domain $\mathcal{R}$ and power-product domain $\mathcal{T}$, represented as ordered (w. r. t. the ordering on $\mathcal{T}$) tuples of pairs of non-zero coefficients and power-products, as indicated at the beginning of this subsection. $\mathcal{P}$ defines the various constants, functions and relations required by `isAlgoReductionPolynomialDomain`, e. g. the coefficient function `C` in $\mathcal{P}$ is defined recursively as

$$\underset{c,\, s,\, t,\, r\ldots}{\forall}$$

$$\underset{\mathcal{P}}{\texttt{C}}[\langle\rangle, s] := \underset{\mathcal{R}}{0} \qquad (\textit{PolyTuples C base})$$

$$\underset{\mathcal{P}}{\texttt{C}}[\langle\langle c,t\rangle, r\ldots\rangle, s] := \begin{cases} c & \Leftarrow \quad t = s \\ \underset{\mathcal{P}}{\texttt{C}}[\langle r\ldots\rangle, s] & \Leftarrow \texttt{otherwise} \end{cases} \qquad (\textit{PolyTuples C rec})$$

The fact that the tuples representing polynomials are ordered is crucial, because of the extensionality requirement in `isReductionPolynomialDomain`: if the coefficient function agrees for two polynomials $p$ and $q$, then $p$ must be equal to $q$. This, however, would be violated in the case of *unordered* tuples: given two different unordered tuples $p$ and $q$, the coefficient function $\underset{\mathcal{P}}{\mathtt{C}}$, as defined above, might still yield the same result for all power-products $t$.

The remaining part of theory PolyTuples.nb is all about proving that `PolyTuples[`$\overline{\mathcal{R}}, \overline{\mathcal{T}}$`]`, for the two arbitrary but fixed constants $\overline{\mathcal{R}}$ and $\overline{\mathcal{T}}$, is an algorithmic reduction polynomial domain according to Polynomials.nb. As soon as the correctness of this theorem has been established, we can conclude that given any algorithmic reduction ring $\mathcal{R}$ and any commutative power-product domain $\mathcal{T}$, the domain `PolyTuples[`$\mathcal{R}, \mathcal{T}$`]` is an algorithmic reduction ring as well. This, for instance, holds in particular for $\mathbb{Z}_{24}[x, y]$ ordered lexicographically:

```
PolyTuples[
   ReductionIQR[24],
   OrderedPP[PPTuples[2],Lex[PPTuples[2]]]
]
```

For the sake of completeness we point out that the theory provides functionality for formatted in- and output of tuple-polynomials, too. In computations, polynomials may thus be written in the usual way as, say, $x + 2y$, instead of the far more cumbersome and confusing $\langle\langle 1, \langle 1, 0\rangle\rangle, \langle 2, \langle 0, 1\rangle\rangle\rangle$.

### 4.3.7 GroebnerRings.nb

The last theory in our formalization is GroebnerRings.nb. It introduces the functor `GroebnerRing`, that extends its argument domain $\mathcal{D}$ by functions for totally reducing an element modulo a given tuple of elements (`trd`), for checking whether the chain criterion holds in a certain situation (`chainCrit`; see Definition 3.5.5), and, most importantly, for computing Gröbner bases from given input-tuples (`GB`). All of these functions are defined in terms of the interpretations of `lcrd`, `rdm`, `cpm`, $+$ etc. in $\mathcal{D}$ in such a way that if $\mathcal{D}$ happens to be an algorithmic reduction ring, then $\underset{\mathtt{GroebnerRing}[\mathcal{D}]}{\mathtt{GB}}$ really computes Gröbner bases. The proof of this claim constitutes the main part of the theory and is not quite trivial, despite already knowing the Main Theorem (Theorem 3.5.3); it is discussed in detail in the next section, where the implementation of `GB` as a tail-recursive program is looked at thoroughly. Once more we point out that $\underset{\mathtt{GroebnerRing}[\mathcal{D}]}{\mathtt{GB}}$ is effectively computable only if the underlying operations in $\mathcal{D}$ are effectively computable, which is *not* required in algorithmic reduction rings.

Please also note that `GroebnerRing[`$\mathcal{D}$`]` itself is still an algorithmic reduction ring if $\mathcal{D}$ is; this (easy) fact is needed for applying the main results about plain and algorithmic reduction rings to `GroebnerRing[`$\mathcal{D}$`]`.

Finally, GroebnerRings.nb contains a lot of sample computations of Gröbner bases in various domains, including $\mathbb{Q}$, $\mathbb{Z}$, $\mathbb{Z}_{24}$, and bi- and trivariate polynomial rings thereof (ordered degree-reverse-lexicographically). As expected, our implementation cannot compete with the built-in *Mathematica* implementation of Buchberger's algorithm in terms of efficiency by any means.

## 4.4 Buchberger's Algorithm in Theorema

In Section 3.5.1, Buchberger's critical-pair/completion algorithm for computing Gröbner bases in reduction ring is presented in an imperative style. In our formalization, however, we implement the algorithm by means of a tail-recursive function in theory GroebnerRings.nb, called GB. In this section, we present GB and its specification in detail and explain how its correctness is established within the formalization.

*Remark* 13. Throughout this section all domain-underscripts, as in $\underset{\texttt{GroebnerRing}[\mathcal{D}]}{\texttt{GB}}$, are omitted for the sake of better readability. It is should be clear, though, that all constants, functions and programs appearing below are actually defined in `GroebnerRing[`$\mathcal{D}$`]` and thus implicitly depend on the underlying domain $\mathcal{D}$. In the correctness proof of function GB, $\mathcal{D}$ is assumed to be an algorithmic reduction ring.

Everything we present in this section is contained in theory GroebnerRings.nb.

### 4.4.1 Implementation

Buchberger's algorithm is implemented by function GB, which is defined as

$$\underset{C,P,i,j,i0,j0,a,mi,mj,p\ldots,r\ldots}{\forall}$$

$$\texttt{GB}[C] := \texttt{GBAux}[C, \texttt{allPairs}[|C|], 1, 1, \langle\rangle] \qquad (GB)$$

$$\texttt{GBAux}[C, \langle\rangle, i, j, \langle\rangle] := C \qquad (GBAux\ 1)$$

$$\texttt{GBAux}[C, \langle\langle i, j\rangle, p\ldots\rangle, i0, j0, \langle\rangle] := \begin{cases} \texttt{GBAux}[C, \langle p\ldots\rangle, i, j, \langle\rangle] & \Leftarrow C_i = 0 \vee C_j = 0 \\ \texttt{GBAux}[C, \langle p\ldots\rangle, i, j, \texttt{lmTuple}[C_i, C_j]] & \Leftarrow \texttt{otherwise} \end{cases} \qquad (GBAux\ 2)$$

$$\texttt{GBAux}[C, P, i, j, \langle\langle a, \langle mi, mj\rangle\rangle, r\ldots\rangle] :=$$

$$\underset{h=\texttt{trd}[a-mi\,C_i,C]-\texttt{trd}[a-mj\,C_j,C]}{\texttt{let}} \begin{cases} \texttt{GBAux}[C, P, i, j, \langle r\ldots\rangle] & \Leftarrow \texttt{cc}[i, j, a, C, P] \\ \texttt{GBAux}[C, P, i, j, \langle r\ldots\rangle] & \Leftarrow \quad h = 0 \\ \texttt{GBAux}[C \curvearrowleft h, \texttt{up}[P, |C|], i, j, \langle r\ldots\rangle] & \Leftarrow \quad \texttt{otherwise} \end{cases} \qquad (\textit{GBAux 3})$$

Before we can explain these formulas in detail, some preliminary remarks on the typesetting of (*GBAux 3*) are in place:

- `cc` abbreviates `chainCrit`, for space reasons. In the sequel, we refer to it again as `chainCrit`.
- up is called `update` in the formalization and in the sequel; it is abbreviated for space reasons.
- $h$ is actually computed *after* the chain criterion is checked, since the very purpose of the chain criterion is to avoid computing $h$ if not necessary. In (*GBAux 3*), the order of computation is reversed due to the lack of space.

As can be seen in (*GB*), `GB` merely “calls” `GBAux` with suitable initial arguments, and it is in fact `GBAux` that is defined recursively and “does all the computation”. In the sequel, hence, we restrict the presentation and explanation to `GBAux`.

The five arguments of `GBAux` have the following meaning:

- The first argument $C$ is the basis constructed so far, i. e. it serves as the accumulator of the tail-recursive algorithm. As such, it is a tuple of elements of the underlying domain that is initialized by the original input-tuple in (*GB*) and returned as the final result in (*GBAux 1*). The only place where it is modified is in the third case of (*GBAux 3*), where a new element $h$ is added to it.
- The second argument is a tuple of pairs of indices of the accumulator $C$. It contains precisely those indices corresponding to pairs of elements of $C$ that still have to be processed; hence, it is initialized by *all* possible pairs of element-indices in (*GB*), using `allPairs`, and updated whenever a new element is added to $C$ in (*GBAux 3*) using `update`.
- The third and fourth arguments $i$ and $j$ are the indices of the pair of elements of $C$ whose mntcrs and critical pairs are currently under consideration.

- The last argument is a tuple consisting of elements of the form $\langle a, \langle mi, mj\rangle\rangle$, where $a$ is a mntcr of $C_i$ and $C_j$ and $mi, mj$ are corresponding critical-pair multipliers. It contains precisely those mntcrs of $C_i$ and $C_j$ that have not been considered yet. Once initialized by function `lmTuple` in (*GBAux 2*), it is simply traversed from beginning to end without being enlarged at any point.

Before we show the specifications of `GB` and `GBAux`, we give a brief overview of the auxiliary functions used in their definitions; most of them are quite self-explanatory anyway, though:

**`allPairs`.** `allPairs[`$n$`]` returns the tuple of all pairs of the form $\langle i, j\rangle$ with $1 \leq i \leq j \leq n$. For making `GBAux` more efficient, in particular for increasing the number of situations where `chainCrit` holds, all pairs of the form $\langle i, i\rangle$ are put at the beginning of `allPairs[`$n$`]`, because pairs of identical basis elements cannot be discarded anyway by the chain criterion.

`allPairs` is no domain function of `GroebnerRing[`$\mathcal{D}$`]`, as it is only concerned with tuples of pairs of natural numbers.

**`update`.** `update[`$P, n$`]` returns the tuple $P$ with all pairs of the form $\langle i, n+1\rangle$ appended, for $1 \leq i \leq n+1$. The pair $\langle n+1, n+1\rangle$ is appended first.

`update` is no domain function of `GroebnerRing[`$\mathcal{D}$`]` either, as it is only concerned with tuples of pairs of natural numbers (just as `allPairs`).

**`lmTuple`.** `lmTuple[`$x, y$`]` returns a tuple of pairs of the form $\langle a, \langle mx, my\rangle\rangle$, where all the $a$ are mntcrs of $x$ and $y$ w. r. t. any indices (see Definition 3.2.17), and $mx$, $my$ are corresponding critical pair multipliers (see Definition 3.2.20). `lmTuple` is defined using functions `lcrd` and `cpm` of the underlying domain $\mathcal{D}$, ensuring that its output really contains *all* mntcrs (or, at least, one from each equivalence class) if $\mathcal{D}$ is an algorithmic reduction ring.

**`chainCrit`.** `chainCrit[`$i, j, a, C, P$`]` indicates whether the chain criterion holds for $C_i$ and $C_j$ w. r. t. $a$, $C$ and $P$, according to Definition 3.5.5 (only that now $C$ and $P$ are tuples rather than sets).

**`trd`.** `trd[`$x, C$`]` repeatedly reduces $x$ modulo the tuple $C$ as long as possible, employing function `rdm` of the underlying domain $\mathcal{D}$. If $\mathcal{D}$ is an algorithmic reduction ring, the result is known to be a normal form of $x$ modulo $C$.

### 4.4.2 Specification

For the rest of this section assume that $\mathcal{D}$ is an algorithmic reduction ring.

Function GB has to meet the following requirements, for every tuple $C$ of elements of $\mathcal{D}$ (i. e. $\underset{\texttt{DomainTuples}[\mathcal{D}]}{\in}[C]$):

1. GB$[C]$ must again be a tuple of elements of $\mathcal{D}$; in particular, the function must terminate.

2. The ideal (over $\mathcal{D}$) generated by GB$[C]$ must be the same as the ideal generated by $C$.

3. GB$[C]$ must be a Gröbner basis.

Note that termination of GB, or, more precisely, GBAux, is all but obvious: the second argument $P$ of GBAux eventually must become empty in order for the function to terminate, but $P$ is enlarged in the third case of (*GBAux 3*); Axiom (R13) is needed to ensure termination, see below. If the function terminates, however, it is easy to see that the result is again a tuple of elements of $\mathcal{D}$—even that it is of the form $C \bowtie D$ for some tuple $D$. Moreover, every element in GB$[C]$ can apparently be expressed as a $\mathcal{D}$-linear combination of the elements of $\mathcal{D}$, implying that the ideal is preserved. The proof that GB$[C]$ is a Gröbner basis makes use of Theorem 3.5.3, of course, but still is far from trivial.

### 4.4.3 Termination

Proving the termination of function GB amounts to proving the termination of function GBAux. GBAux is defined recursively, and a common practice for establishing the termination of such functions proceeds by finding a Noetherian *termination order* $\gg$ on the lits of arguments that decreases in each recursive call. In our case of GBAux, $\gg$ is the lexicographic combination of the three orders $\gg_1$, $\gg_2$ and $\gg_3$, defined as

$$\underset{\underset{\texttt{DomainTuples}[\mathcal{D}]}{\in}[A,B]}{\forall} \quad A \gg_1 B \;:\Leftrightarrow\; \texttt{Red}[A] \subsetneq \texttt{Red}[B] \tag{$\gg_1$}$$

$$\underset{\underset{\texttt{DomainTuples}[\texttt{ProdDomain}[\mathbb{Z},\mathbb{Z}]]}{\in}[S,T]}{\forall} \quad S \gg_2 T \;:\Leftrightarrow\; |S| > |T| \tag{$\gg_2$}$$

$$\underset{\underset{\texttt{DomainTuples}[\texttt{ProdDomain}[\mathcal{D},\texttt{DomainTuples}[\mathcal{D}]]]}{\in}[S,T]}{\forall} \quad S \gg_3 T \;:\Leftrightarrow\; |S| > |T| \tag{$\gg_3$}$$

where `Red[`$A$`]`, similar as in Axiom (R13), denotes the set of all elements that are reducible modulo the tuple $A$, and where `ProdDomain[`$\mathcal{D}_1, \mathcal{D}_2$`]`, as its name suggests, is simply the Cartesian product of the two domains $\mathcal{D}_1$ and $\mathcal{D}_2$. Although both $\gg_2$ and $\gg_3$ simply compare the lengths of their arguments, they cannot be replaced by a single order on tuples in general: the class of all tuples does not constitute a set, but we need a *set* for proving $\gg_2$ and $\gg_3$ Noetherian; `DomainTuples` and `ProdDomain` are known to map domains with set-carrier (i. e. satisfying `isDomain` from LogicSets.nb) to domains with set-carrier. Thus, both $\gg_2$ and $\gg_3$ are obviously partial Noetherian order relations, and $\gg_1$ being Noetherian is precisely what (R13) states (note the order of $A$ and $B$ on the two sides of the equivalence in Formula ($\gg_1$)!)

$\gg$, finally, is defined for argument-lists of `GBAux` according to

$$\underset{C1,C2,P1,P2,i1,i2,j1,j2,L1,L2}{\forall} \langle C1, P1, i1, j1, L1\rangle \gg \langle C2, P2, i2, j2, L2\rangle \;:\Leftrightarrow\; \langle C1, P1, L1\rangle \underset{\texttt{LexOrder3}[\gg_1,\gg_2,\gg_3]}{\succ} \langle C2, P2, L2\rangle \qquad (\gg)$$

A general result on lexicographic orders ensures that $\gg$, restricted to the Cartesian product of the respective domains of $\gg_1$, $\gg_2$ and $\gg_2$, is again a partial, and most importantly Noetherian, order relation. The crucial property of $\gg$ is that it decreases in each of the five recursive calls of `GBAux`, as can be verified rather easily. Note here that adding $h$ to $C$ in the third case of (*GBAux 3*) surely enlarges the set of reducible elements, hence decreasing $\gg_1$, although $h$ itself might already be reducible modulo $C$: $h = h_1 - h_2$ for two distinct irreducible elements $h_1$ and $h_2$, so Axiom (R5) implies $h_1 \downarrow^*_{\{h\}} h_2$, meaning that $h_1$ or $h_2$ is reducible modulo $C \curvearrowleft h$.

Apparently, the third and fourth arguments of `GBAux` do not have any influence on the termination of the function, which is not surprising at all.

When proving properties of recursively defined functions whose termination has already been established, it is convenient to have tailor-made induction rules w. r. t. the respective termination order at one's disposal. In our case, we proved an induction rule that allows us to reduce goals of the form $\underset{C,P,i,j,L}{\forall} \varphi[C, P, i, j, L]$, for an arbitrary formula (i. e. higher-order predicate) $\varphi$ and where the quantified variables range over the respective domains (`DomainTuples[`$\mathcal{D}$`]` etc.), to five sub-goals, each corresponding to a recursive call of `GBAux`. In each sub-goal, the *induction hypothesis* assumes that $\varphi$ holds for the arguments of the respective recursive call. As an example of such a sub-goal consider the one corresponding to the second case of (*GBAux 2*):

$$\underset{\substack{C,p\ldots,i,j,i0,j0\\ \in\\ \texttt{DomainTuples[}\mathcal{D}\texttt{]}}}{\forall} [C] \wedge \underset{\texttt{DomainTuples[ProdDomain[}\mathbb{Z},\mathbb{Z}\texttt{]]}}{\in} [\langle p\ldots\rangle] \wedge i \geq j \wedge i0 \geq j0 \Rightarrow$$
$$\varphi[C, \langle p\ldots\rangle, i, j, \texttt{lmTuple}[C_i, C_j]] \wedge C_i \neq 0 \wedge C_j \neq 0 \Rightarrow$$
$$\varphi[C, \langle\langle i,j\rangle, p\ldots\rangle, i0, j0, \langle\rangle] \qquad (\textit{GBAuxRec2})$$

In (*GBAuxRec2*) $\varphi$ must be proved for the arguments of GBAux on the left-hand-side of (*GBAux 2*), assuming $C_i \neq 0$ and $C_j \neq 0$ as the case conditions and $\varphi$ of the arguments of the recursive call of GBAux as induction hypothesis.

It is important to note that the induction rule never mentions function GBAux explicitly.

### 4.4.4 Preservation of the Ideal

The proof that the ideal generated by the accumulator of GBAux is preserved throughout the computation is absolutely straight-forward. The induction rule described above is used to prove that the ideal is preserved in every recursive call, which is completely trivial in all but the third case of (*GBAux 3*) where $h$ is added to $C$. However, $h$ is clearly a $\mathcal{D}$-linear combination of the elements already contained in $C$, meaning that it is contained in the ideal generated by $C$. A general result on ideals states that an ideal does not change when adding an ideal element to its set of generators, so the proof is finished. The final result thus is

$$\underset{C,P,i,j,L}{\forall} \texttt{ideal[GBAux[}C,P,i,j,L\texttt{]]} = \texttt{ideal[}C\texttt{]} \qquad (\textit{ideal GBAux})$$

where, as usual, $C$, $P$, $i$, $j$ and $L$ actually only range over the respective argument-"types" of GBAux.

### 4.4.5 Completion to a Gröbner Basis

The proof that the result of GB is a Gröbner basis is far more involved, because the analogous claim for GBAux simply does not hold unconditionally. Instead, the following two inductive assertions are shown to be always satisfied (by induction, of course):

$$\underset{C,P,i,j,L}{\forall}\ \underset{G=\text{GBAux}[C,P,i,j,L]}{\texttt{let}}$$

$$\texttt{isTODOTuple}[P,C,i,j] \wedge \texttt{isCPTODOTuple}[L,P,C,i,j] \Rightarrow$$

$$\underset{\substack{k,l=1,\ldots,|G| \\ \langle k,l\rangle \in P}}{\forall} \texttt{cpConnectible}[G,G_k,G_l] \qquad (\textit{GBAuxCPConnectible2 GBAux})$$

$$\texttt{isTODOTuple}[P,C,i,j] \wedge \texttt{isCPTODOTuple}[L,P,C,i,j] \Rightarrow$$

$$\underset{\substack{k,l=1,\ldots,|G| \\ l\geq k \wedge l>|C|}}{\forall} \texttt{cpConnectible}[G,G_k,G_l] \qquad (\textit{GBAuxCPConnectible3 GBAux})$$

The first inductive assertion states that under certain conditions (see below), the critical pairs of elements $G_k, G_l$, where the pair $\langle k,l\rangle$ occurs in $P$, can be connected below the corresponding mntcrs, whereas the second inductive assertion states that the critical pairs of elements $G_k, G_l$, where $l > |C|$ and hence $G_l$ is not contained in the original tuple $C$, can be connected. Together they ensure that *all* critical pairs of *all* elements of GB[$C$] can be connected below the corresponding mntcrs, because GB initializes $P$ by *all* index-pairs and the two requirements `isTODOTuple` and `isCPTODOTuple` are trivially met by the initial arguments $C$, `allPairs[`$|C|$`]`, 1, 1 and $\langle\rangle$ of GBAux. Thus, GB[$C$] really is a Gröbner basis, thanks to Theorem 3.5.3.

The exact definitions of `isTODOTuple` and `isCPTODOTuple` are a bit lengthy, so we only provide an informal description here. `isTODOTuple[`$P,C,i,j$`]` holds iff `cpConnectible[`$C,C_k,C_l$`]` is satisfied by all $C_k, C_l$, where neither of the two pairs $\langle k,l\rangle$ and $\langle l,k\rangle$ occurs in $P$ or equals $\langle i,j\rangle$. Intuitively speaking, `isTODOTuple[`$P,C,i,j$`]` asserts that all pairs not occurring in $P$, except maybe $\langle i,j\rangle$, "have already been processed" by GBAux in the sense that they satisfy the criterion of Theorem 3.5.3.

On the other hand, `isCPTODOTuple[`$L,P,C,i,j$`]` holds iff either $\langle i,j\rangle$ is still contained in $P$ or the critical pairs of $C_i$ and $C_j$ w. r. t. those mntcrs $a$ *not* occurring in $L$ can be connected below $a$. Intuitively, `isCPTODOTuple[`$L,P,C,i,j$`]` asserts that the mntcrs of $C_i$ and $C_j$ not occurring in $L$ "have already been processed" by GBAux in the sense that the critical pairs they give rise to can be connected below them.

*Remark* 14. The first inductive assertion alone is not sufficient, even though one might think so: for GB it only implies that the critical pairs originating from the *original* basis $C$ can be connected modulo the *resulting* basis GB[$C$], but it does not say anything about the critical pairs originating from *new* basis elements.

### 4.4.6 Sample Computations

We conclude this section by presenting some sample computations contained in GroebnerRings.nb. All of them were carried out directly within Theorema, using function GB. $\mathcal{Z}$ denotes the algorithmic reduction ring $\mathbb{Z}$, $\mathcal{Z}24$ denotes $\mathbb{Z}_{24}$, and $\mathcal{Z}24xy$ denotes $\mathbb{Z}_{24}[x, y]$ with degree-reverse-lexicographic term order.

In $\mathbb{Z}$, the computation of Gröbner bases amounts to the computation of greatest common divisors:

in $\underset{\mathcal{Z}}{\mathrm{GB}}[\langle 2091, 2337, 2829\rangle]$

out (0.7s) $\langle 2091, 2337, 2829, 1845, 1599, 1353, 1107, 861, 615, 369, 123\rangle$

In $\mathbb{Z}_{24}$, new basis elements might be greater (w. r. t. the reduction ring ordering) than old ones:

in $\underset{\mathcal{Z}24}{\mathrm{GB}}[\langle 8, 6\rangle]$

out (0.3s) $\langle 8, 6, 22\rangle$

In $\mathbb{Z}_{24}[x, y]$, singletons might not be Gröbner bases:

in $\underset{\mathcal{Z}24xy}{\mathrm{GB}}[\langle 16xy + 2\rangle]$

out (0.6s) $\langle 16xy + 2, 18, 22xy + 2\rangle$

in $\underset{\mathcal{Z}24xy}{\mathrm{GB}}[\langle x + 4y + 2, x^2y + 4x + 3\rangle]$

out (8.2s) $\langle x + 4y + 2, x^2y + 4x + 3, 16y^3 + 16y^2 + 12y + 19, 12y + 15,$
$18y + 15, 18, 21, y^3 + y^2 + 1\rangle$

## 4.5 ReductionRingProver

In order to formally prove the results of reduction ring theory in Theorema, we designed and implemented a *special prover* consisting of a collection of special inference rules (see Section 2.3), called ReductionRingProver. Actually, the name

"ReductionRingProver" is a bit misleading: the prover does *not* incorporate any knowledge about reduction rings or related concepts, but merely "lifts" definitions and (rewrite-) properties of basic notions (integers, order relations, sets, tuples, monoids, rings) to the inference level. In that sense, it is more closely related to the elementary theories listed in Section 4.2 than to the reduction ring theories listed in Section 4.3.

One important point has to be made explicit: as explained above, the special inference rules of the ReductionRingProver implicitly depend on properties of various notions. When proving these properties, which are included in our formalization of the elementary theories, it is clear that the special inference rules must not be used, for otherwise one would end up with a circular argument where the validity of a statement depends on the correctness of an inference rule, which in turn depends on the validity of that statement. In the formal development of our theories, we of course took all this into account.

In total, the ReductionRingProver consists of eleven special inference rules, partitioned into the five categories discussed below. Recall that inference rules have to be implemented in the underlying *Mathematica* programming language, so we occasionally might use *Mathematica* terminology below. Since special rules alone do not suffice, the general-purpose predicate logic rules from the RewriteInteractiveProver, discussed in Section 6.2, are integrated in the ReductionRingProver as well. A sample proof some the special rules of the prover are used in can be found in Appendix A.

### 4.5.1 Intervals of Integers

One of the eleven special inference rules, called `membershipIntegerInterval`, is concerned with goals making assertions about the membership of certain objects in intervals of integers. When applied to a proof situation $K \vdash \Gamma$, it behaves as follows:

- If $\Gamma$ is a conjunction, the conjuncts are split into the two sets $C_1$ and $C_2$, where all elements of $C_1$ are (possibly universally quantified) formulas of the form $t \in \mathbb{Z}_{a,\ldots,b}$, and $C_2$ contains all remaining conjuncts. If neither of the two sets is empty, the proof branches, where in the first branch the new proof goal is $\bigwedge C_1$ and in the second branch the goal is $\bigwedge C_2$ and $C_1$ is added to the list of assumptions. If $C_2$ is empty, the conjunction $\bigwedge C_1$ is processed according to the method described below, and if $C_1$ is empty the rule is not applicable at all and returns a value indicating failure.

- If $\Gamma$ is a (possibly universally quantified) formula of the form $t \in \mathbb{Z}_{a,\ldots,b}$, it is processed according to the method described below.

- In all other cases, the rule is not applicable.

We may assume now that $A$ is a list whose elements are propositions of the form $t \in \mathbb{Z}_{a,\ldots,b}$, where the three terms $t$, $a$ and $b$ possibly contain free variables originating from universal quantifiers (that have been stripped away already) and where no two elements are $\alpha$-equivalent to each other; the interval endpoints $a$ and $b$ may be $\pm\infty$, and $\mathbb{Z}_{-\infty,\ldots,\infty}$ is abbreviated simply by $\mathbb{Z}$. Now, the auxiliary function `simplifyIntegerConditions` is called on $A$. This function tries to reduce the elements of its argument or even eliminate them entirely, making use of the current knowledge $K$. To that end, the following basic inferences are repeatedly applied as long as possible for reducing or eliminating elements $F \equiv t \in \mathbb{Z}_{a,\ldots,b}$ of the list:

- If all of $t$, $a$ and $b$ are integer literals or $\pm\infty$, membership is immediately decided by computation and hence $F$ is either removed or replaced by `False`.
- If $F$ follows readily from known facts in $K$,[5] it is removed.
- If $t \equiv b - (x - 1)$ and $a \equiv 1$, $F$ is replaced by $\{x \in \mathbb{Z}_{1,\ldots,b}, b \in \mathbb{Z}\}$.
- If $t \equiv a \equiv b$, $F$ is replaced by $t \in \mathbb{Z}$.
- If $t \equiv a$ and $a$ is different from $-\infty$, $F$ is replaced by $a \in \mathbb{Z}_{-\infty,\ldots,b}$.
- If $t \equiv b$ and $b$ is different from $\infty$, $F$ is replaced by $b \in \mathbb{Z}_{a,\ldots,\infty}$.
- If $t \equiv |T|$, $b \equiv \infty$ and $a$ is either a non-positive literal or $-\infty$, and $T$ is known to be a tuple (i. e. $\mathtt{isTuple}[T]$ or $\underset{\mathtt{DomainTuples}[\mathcal{D}]}{\in}[T]$ appears in $K$), $F$ is dropped because the length of a tuple is always a natural number.
- If $t$ is known to be contained in a sub-interval of $\mathbb{Z}_{a,\ldots,b}$, $F$ is dropped. Deciding membership in sub-intervals is accomplished by recursively calling `simplifyIntegerConditions`: if, say, $t \in \mathbb{Z}_{x,\ldots,y}$ is known and $A_0 := \{a \in \mathbb{Z}_{-\infty,\ldots,x}, b \in \mathbb{Z}_{y,\ldots,\infty}\}$ can be reduced to the empty list by `simplifyIntegerConditions`, then $t$ is known to be a member of $\mathbb{Z}_{a,\ldots,b}$ as well.
- If $t \equiv x + y$ or $t \equiv xy$, $a$ is either $-\infty$ or a non-negative literal, and $b \equiv \infty$, then $F$ is replaced by $\{x \in \mathbb{Z}_{a,\ldots,\infty}, y \in \mathbb{Z}_{a,\ldots,\infty}\}$. This is justified by the fact that both addition and multiplication are closed in $\mathbb{Z}$ as well as in $\mathbb{Z}_{a,\ldots,\infty}$ with $a$ non-negative.

[5]"$\varphi$ follows readily from $K$" always means that $\varphi$ can be reduced to `True` by successive backward rewriting w. r. t. known (universally quantified) implications in $K$.

- If $t \equiv x - y$, $a \equiv -\infty$ and $b \equiv \infty$, $F$ is replaced by $\{x \in \mathbb{Z}, y \in \mathbb{Z}\}$.
- If $t \equiv -x$, $a \equiv -\infty$ and $b \equiv \infty$, $F$ is replaced by $x \in \mathbb{Z}$.

At the end, if the resulting list $B$ is empty the goal is proved and the branch of the proof is finished. Otherwise, $\bigwedge B$ becomes the new goal.

## 4.5.2 Order Relations

Three inference rules deal with order relations: `orderingGoal`, `orderingKB` and `orderingEqualGoal`. Each of them can handle the following three types of order relations appearing in our formalization, defined in AlgebraicStructures.nb:

- `isPartIrreflOrder[` $\prec, \mathcal{D}$`]` means that $\prec$ is an irreflexive order relation on the domain $\mathcal{D}$.
- `isPartReflOrder[` $\preceq, \mathcal{D}$`]` means that $\preceq$ is a partial (reflexive) order relation on $\mathcal{D}$.
- `isTotalIrreflOrder[` $\prec, \mathcal{D}$`]` means that $\prec$ is a total irreflexive order relation on $\mathcal{D}$.

Furthermore, any binary function $\circ$ might be an *order embedding* w. r. t. $\prec$ on $\mathcal{D}$, i. e. $y \prec z \Leftrightarrow x \circ y \prec x \circ z$ for all $x, y, z$ belonging to $\mathcal{D}$; this is characterized by `isOrderEmbedding[`$\circ, \prec, \mathcal{D}$`]`.

**`orderingGoal`.** This rule combines the following seven basic inferences:

$$\frac{K \vdash \underset{\mathcal{D}}{\in}[x]}{K \vdash \neg x \prec x}\ (1) \qquad \frac{K \vdash \underset{\mathcal{D}}{\in}[x]}{K \vdash x \preceq x}\ (2)$$

where in (1) $\mathcal{D}$ is such that $K$ contains `isPartIrreflOrder[` $\prec, \mathcal{D}$`]` or `isTotalIrreflOrder[` $\prec, \mathcal{D}$`]`, and in (2) it is such that $K$ contains `isPartReflOrder[` $\preceq, \mathcal{D}$`]`;

$$\frac{K \vdash \underset{\mathcal{D}}{\in}[x, y, z_1, \ldots, z_n]}{K, y \prec z_1, z_1 \prec z_2, \ldots, z_n \prec x \vdash \neg x \prec y}\ (3)$$

where $\mathcal{D}$ is such that $K$ contains `isPartIrreflOrder[` $\prec, \mathcal{D}$`]` or `isTotalIrreflOrder[` $\prec, \mathcal{D}$`]` (note that $n$ may well be 0);

$$\frac{K \vdash \underset{\mathcal{D}}{\in}[x, y, z_1, \ldots, z_n] \qquad K \vdash x \neq y}{K, y \preceq z_1, z_1 \preceq z_2, \ldots, z_n \preceq x \vdash \neg x \preceq y}\ (4)$$

where $\mathcal{D}$ is such that $K$ contains `isPartReflOrder[` $\preceq, \mathcal{D}$`]` (as before $n$ may be 0);

$$\frac{K \vdash \underset{\mathcal{D}}{\in}[x, y, z_1, \ldots, z_n]}{K, x \prec z_1, z_1 \prec z_2, \ldots, z_n \prec y \vdash \; x \prec y} \; (5)$$

where $\mathcal{D}$ is such that $K$ contains any of the three order-predicates for $\prec$ and $\mathcal{D}$;

$$\frac{K \vdash \underset{\mathcal{D}}{\in}[x, y, z] \qquad K, \underset{\mathcal{D}}{\in}[x, y, z] \vdash \; y \prec z}{K \vdash \; x \circ y \prec x \circ z} \; (6)$$

where $\mathcal{D}$ is such that $K$ contains `isOrderEmbedding[`$\circ, \prec, \mathcal{D}$`]`;

$$\frac{K \vdash \underset{\mathcal{D}}{\in}[x, y, z] \qquad K, \underset{\mathcal{D}}{\in}[x, y, z] \vdash \; \neg\, y \prec z}{K \vdash \; \neg\, x \circ y \prec x \circ z} \; (7)$$

where as before $\mathcal{D}$ is such that $K$ contains `isOrderEmbedding[`$\circ, \prec, \mathcal{D}$`]`.

**`orderingKB`.** This rule combines the following two basic inferences:

$$\frac{K \vdash \underset{\mathcal{D}}{\in}[x]}{K, \neg\, x \preceq x \vdash \Gamma} \; (1) \qquad\qquad \frac{K \vdash \underset{\mathcal{D}}{\in}[x]}{K, x \prec x \vdash \Gamma} \; (2)$$

where in (1) $\mathcal{D}$ is such that $K$ contains `isPartReflOrder[` $\preceq, \mathcal{D}$`]`,
and in (2) it is such that $K$ contains `isPartIrreflOrder[` $\preceq, \mathcal{D}$`]` or `isTotalIrreflOrder[` $\preceq, \mathcal{D}$`]`.

**`orderingEqualGoal`.** This rule combines the following two basic inferences:

$$\frac{K \vdash \underset{\mathcal{D}}{\in}[x, y] \qquad K, \underset{\mathcal{D}}{\in}[x, y] \vdash \; x \preceq y \wedge y \preceq x}{K \vdash x = y} \; (1)$$

where $\mathcal{D}$ is such that $K$ contains `isPartReflOrder[` $\preceq, \mathcal{D}$`]`, as well as $\underset{\mathcal{D}}{\in}[x]$ or $\underset{\mathcal{D}}{\in}[y]$;

$$\frac{K \vdash \underset{\mathcal{D}}{\in}[x, y] \qquad K, \underset{\mathcal{D}}{\in}[x, y] \vdash \; \neg\, x \prec y \wedge \neg\, y \prec x}{K \vdash x = y} \; (2)$$

where $\mathcal{D}$ is such that $K$ contains `isTotalIrreflOrder[` $\prec, \mathcal{D}$`]`, as well as $\underset{\mathcal{D}}{\in}[x]$ or $\underset{\mathcal{D}}{\in}[y]$.

### 4.5.3 Sets and Tuples

The following two rules handle goals that assert membership of certain objects in certain domains. These domains typically are either `DomainTuples[`$\mathcal{D}$`]` or `DomainSets[`$\mathcal{D}$`]`.

**`membershipDomainTuples`.** This rule is a combination of the following 14 basic inferences:

$$\frac{}{\underset{\mathtt{DomainTuples[}\mathcal{D}\mathtt{]}}{\in}\mathtt{[}T\mathtt{]} \vdash \mathtt{isTuple[}T\mathtt{]}} \tag{1}$$

$$\frac{K \vdash \underset{\mathtt{DomainTuples[}\mathcal{D}\mathtt{]}}{\in}\mathtt{[}T\mathtt{]} \qquad K, \underset{\mathtt{DomainTuples[}\mathcal{D}\mathtt{]}}{\in}\mathtt{[}T\mathtt{]} \vdash i \in \mathbb{Z}_{1,\ldots,|T|}}{K \vdash \underset{\mathcal{D}}{\in}\mathtt{[}T_i\mathtt{]}} \tag{2}$$

$$\frac{K \vdash \underset{\mathtt{DomainTuples[}\mathcal{D}\mathtt{]}}{\in}\mathtt{[}T\mathtt{]}}{K, x \mathrel{\mathtt{E}} T \vdash \underset{\mathcal{D}}{\in}\mathtt{[}x\mathtt{]}} \tag{3}$$

$$\frac{K \vdash \underset{\mathtt{DomainTuples[}\mathcal{D}\mathtt{]}}{\in}\mathtt{[}A, B\mathtt{]}}{K \vdash \underset{\mathtt{DomainTuples[}\mathcal{D}\mathtt{]}}{\in}\mathtt{[}A \bowtie B\mathtt{]}} \tag{4}$$

$$\frac{K \vdash \underset{\mathcal{D}}{\in}\mathtt{[}x\mathtt{]} \qquad K, \underset{\mathcal{D}}{\in}\mathtt{[}x\mathtt{]} \vdash \underset{\mathtt{DomainTuples[}\mathcal{D}\mathtt{]}}{\in}\mathtt{[}T\mathtt{]}}{K \vdash \underset{\mathtt{DomainTuples[}\mathcal{D}\mathtt{]}}{\in}\mathtt{[}x \curvearrowright T\mathtt{]}} \tag{5}$$

$$\frac{K \vdash \underset{\mathcal{D}}{\in}\mathtt{[}x\mathtt{]} \qquad K, \underset{\mathcal{D}}{\in}\mathtt{[}x\mathtt{]} \vdash \underset{\mathtt{DomainTuples[}\mathcal{D}\mathtt{]}}{\in}\mathtt{[}T\mathtt{]}}{K \vdash \underset{\mathtt{DomainTuples[}\mathcal{D}\mathtt{]}}{\in}\mathtt{[}T \curvearrowleft x\mathtt{]}} \tag{6}$$

$$\frac{K \vdash \underset{\mathcal{D}}{\in}\mathtt{[}x_1, \ldots, x_n\mathtt{]}}{K \vdash \underset{\mathtt{DomainTuples[}\mathcal{D}\mathtt{]}}{\in}\mathtt{[}\langle x_1, \ldots, x_n\rangle\mathtt{]}} \tag{7}$$

$$\frac{K \vdash \underset{\mathtt{DomainTuples[}\mathcal{D}\mathtt{]}}{\in}\mathtt{[}T\mathtt{]}}{K \vdash \underset{\mathtt{DomainTuples[}\mathcal{D}\mathtt{]}}{\in}\mathtt{[Reverse[}T\mathtt{]]}} \tag{8}$$

$$\frac{K \vdash \underset{\mathtt{DomainTuples[}\mathcal{D}\mathtt{]}}{\in}\mathtt{[}T\mathtt{]} \qquad K \vdash T \neq \langle\rangle}{K \vdash \underset{\mathtt{DomainTuples[}\mathcal{D}\mathtt{]}}{\in}\mathtt{[Rest[}T\mathtt{]]}} \tag{9}$$

$$\frac{K \vdash \underset{\mathtt{DomainTuples[}\mathcal{D}\mathtt{]}}{\in}\mathtt{[}T\mathtt{]} \qquad K \vdash T \neq \langle\rangle}{K \vdash \underset{\mathtt{DomainTuples[}\mathcal{D}\mathtt{]}}{\in}\mathtt{[Most[}T\mathtt{]]}} \tag{10}$$

$$\frac{K \vdash \underset{\texttt{DomainTuples}[\mathcal{D}]}{\in}\texttt{[}\langle f\texttt{[}i\texttt{]} \underset{i=a,\ldots,b}{|} P\texttt{[}i\texttt{]}\rangle\texttt{]}}{K \vdash \underset{\texttt{DomainTuples}[\mathcal{D}]}{\in}\texttt{[}\langle f\texttt{[}b-(i-a)\texttt{]} \underset{i=a,\ldots,b}{|} P\texttt{[}b-(i-a)\texttt{]}\rangle\texttt{]}} \tag{11}$$

$$\frac{K \vdash \underset{\texttt{DomainTuples}[\mathcal{D}]}{\in}\texttt{[}S\texttt{]}}{K \vdash \underset{\texttt{DomainTuples}[\mathcal{D}]}{\in}\texttt{[}\langle S_i \underset{i=1,\ldots,|S|}{|} P\texttt{[}i\texttt{]}\rangle\texttt{]}} \tag{12}$$

$$\frac{K \vdash a \in \mathbb{Z} \wedge a \in \mathbb{Z} \qquad K, a \in \mathbb{Z}, b \in \mathbb{Z} \vdash \underset{\substack{i=a,\ldots,b \\ P[i]}}{\forall} \underset{\texttt{DomainTuples}[\mathcal{D}]}{\in}\texttt{[}f\texttt{[}i\texttt{]]}}{K \vdash \underset{\texttt{DomainTuples}[\mathcal{D}]}{\in}\texttt{[}\langle f\texttt{[}i\texttt{]} \underset{i=a,\ldots,b}{|} P\texttt{[}i\texttt{]}\rangle\texttt{]}} \tag{13}$$

$$\frac{K \vdash \texttt{isTuple[}T\texttt{]} \qquad K, \texttt{isTuple[}T\texttt{]} \vdash \underset{i=1,\ldots,|T|}{\forall} \underset{\mathcal{D}}{\in}\texttt{[}T_i\texttt{]}}{K \vdash \underset{\texttt{DomainTuples}[\mathcal{D}]}{\in}\texttt{[}T\texttt{]}} \tag{14}$$

The last basic inference is actually simply the definition of membership in `DomainTuples`$[\mathcal{D}]$. The true power of `membershipDomainTuples` stems from the fact that it automatically reduces goals of the form $\underset{\texttt{DomainTuples}[\mathcal{D}]}{\in}\texttt{[}T\texttt{]}$ *repeatedly*, and moreover always immediately checks whether the current goal (of whatever form) follows directly from the current knowledge base by backward rewriting.

**`membershipDomainSets`.** This rule is analogous to the previous one, only for (domain-)sets and set operations rather than tuples and tuple operations.

## 4.5.4 Cancellative Commutative Monoids

The ReductionRingProver contains two special rules dedicated to cancellative commutative monoids $\mathcal{M}$ (w. r. t. the operation $\circ$ and neutral element $n$), denoted by $\texttt{isCCMonoid[}\mathcal{M}, \circ, n\texttt{]}$ in the formalization. One of these two rules deals with domain-membership goals, the other one with proving equalities. For the rest of this subsection assume that the formula $\texttt{isCCMonoid[}\mathcal{M}, \circ, n\texttt{]}$ is included in the knowledge base of the current proof situation, for some $\mathcal{M}$, $\circ$ and $n$.

**membershipCCMonoid.** This rule is applicable if the current goal is of the form $\underset{\mathcal{M}}{\in}[t]$ (possibly universally quantified), a conjunction thereof, or even a conjunction involving also conjuncts of different form. In the latter case, the goal is split into two sub-goals, where the first one only consists of the aforementioned domain-membership conjuncts and the second one of the rest; this is totally analogous to `membershipIntegerInterval`. Hence, assume now that we are given a list $A$ consisting entirely of propositions of the form $\underset{\mathcal{M}}{\in}[t]$, where the terms $t$ may depend on free variables originating from universal quantifiers.

The main idea behind `membershipCCMonoids` is simple enough: exploiting closure of $\circ$ in $\mathcal{M}$, i. e. $\underset{M}{\in}[x, y]$ allows to infer $\underset{\mathcal{M}}{\in}[x \circ y]$. As in the corresponding rule for integer intervals, the elements of the list $A$ are repeatedly reduced or even eliminated, according to the following two basic inferences:

$$\frac{}{K \vdash \underset{\mathcal{M}}{\in}[n]} \qquad\qquad \frac{K \vdash \underset{\mathcal{M}}{\in}[x] \qquad K, \underset{\mathcal{M}}{\in}[x] \vdash \underset{\mathcal{M}}{\in}[y]}{K \vdash \underset{\mathcal{M}}{\in}[x \circ y]}$$

As always, new formulas are immediately checked whether they follow readily from the current knowledge base by backward rewriting w. r. t. (quantified) implications. Furthermore, they are also checked for $\alpha$-equivalence with formulas that have already been processed.

**CCMonoidEqual.** This rule is applicable if the current goal is a chain of equalities of the form $a_1 = a_2 = \ldots = a_n$, where the outermost function- or constant symbol of at least one of the $a_i$ is $\circ$ or $n$. In that case, the chain is decomposed into the individual equalities $a_1 = a_2$, …, $a_{n-1} = a_n$, each of them being treated separately by the procedure described below.

Assume now we are given an equality $a = b$, where the outermost function or constant symbol of $a$ or $b$ is $\circ$ or $n$.

1. $a$ is traversed in breadth-first manner, as long as the head of the current sub-expression is $\circ$.

2. Let $x$ be one of the sub-expressions thus visited, and assume that $x$ is distinct from $n$. Now, $b$ is searched for a sub-expression $x'$ that is $\alpha$-equivalent to $x$, occurring at a position where all outer function symbols are $\circ$. If such a $x'$ is found, both $x$ in $a$ and $x'$ in $b$ are replaced by $n$, and all sub-expressions appearing in an argument list of any of the outer function symbols of $x$ and $x'$, respectively, are collected in a separate list $C$; this, in particular, includes $x$ and $x'$ themselves. For instance, if $a \equiv (x \circ y) \circ (u \circ v)$ and $b \equiv (u \circ (v \circ y)) \circ x'$ then $C$ would consist of $x$, $y$, $u \circ v$, $x'$ and $u \circ (v \circ y)$.

3. Finally, occurrences of $n$ in $a$ and $b$ are eliminated by exploiting that $n$ is the neutral element of $\circ$. However, if a sub-expression $x \circ n$ is simplified to $x$, $x$ is added to the list $C$.

These three steps yields a new (ideally trivial) equality $a' = b'$ together with the list $C$, with the meaning that $a = b$ follows from $a' = b'$ given that all elements of $C$ belong to $\mathcal{M}$. Hence, the original equality is reduced to $a' = b'$ (unless this equality is already trivial), but in addition the new sub-goal asserting membership of all of $C$'s elements in $\mathcal{M}$ has to be proved.

*Remark* 15. Repeatedly adding terms to list $C$ might appear a bit cumbersome at first sight, especially if one is aware that typically many expressions thus added are in fact sub-expressions of other elements already contained in the list. So, why not just immediately prove domain-membership of *all* atomic sub-terms of $a$ and $b$? The best way of answering this question is by means of an example: Assume the equality one wishes to prove is $(x \circ y) \circ (u \circ v) = ((x \circ y) \circ u) \circ v$. The equality can easily be proved, provided that $x \circ y$, $u$ and $v$ belong to $\mathcal{M}$—so, $x \circ y$ does not have to be decomposed into $x$ and $y$ at all! It might well be that the membership of $x$ or $y$ in $\mathcal{M}$ cannot be established, meaning that a strategy that simply tries to prove domain-membership of all atoms would fail in such a situation.

### 4.5.5 Commutative Rings with Identity

Similar as for cancellative commutative monoids there are three special rules for working in the theory of commutative rings with identity: one for proving domain-membership, one for proving equality of terms, and additionally one for simplifying known equalities. The overall ideas behind the ring-rules are precisely the same as for the monoid-rules, so we do not rephrase them here (e. g. try to avoid proving domain-membership of terms that actually do not have to belong to the ring, when proving equality of two terms; see Remark 15 above). Instead, we point out the main differences here, in particular what makes matters much more complicated when dealing with commutative rings rather than cancellative commutative monoids.

Essentially, there are three main differences between the ring-rules and the monoid-rules:

- in rings there are additive inverses and subtraction,
- in rings, addition (and subtraction) and multiplication satisfy a distributivity law, and
- in rings we also have to deal with *finite sums*, as they play an important role in reduction ring theory.

Let in the rest of this subsection $\mathcal{R}$ be a domain such that `isCommRing1[`$\mathcal{R}$`]` is contained in the current knowledge base. Although $\mathcal{R}$ is a domain and the ring-constants and -operations $0$, $+$, $*$ etc. are thus under-scripted by $\mathcal{R}$, we omit these underscripts here for the sake of readability.

**`membershipCommRing1`.** This rule, analogous to `membershipCCMonoids`, is applicable to (universally quantified) goals of the form $\underset{\mathcal{R}}{\in}[t]$, conjunctions thereof, and even conjunctions involving conjuncts of different form; the latter are treated precisely as in the corresponding rule for monoids. Hence, assume once more that we are given a list $A$ of domain-membership propositions of the aforementioned form. The elements of $A$ are repeatedly reduced exploiting closure properties of $+$, $-$ (both unary and binary) and $*$ in $\mathcal{R}$. With finite sums, though, matters are slightly more complicated; they are handled according to

$$\frac{K \vdash a \in \mathbb{Z} \wedge b \in \mathbb{Z} \qquad K, a \in \mathbb{Z}, b \in \mathbb{Z}, \bar{i} \in \mathbb{Z}_{a,\ldots,b} \vdash \underset{\mathcal{R}}{\in}[f[\bar{i}]]}{K \vdash \underset{\mathcal{R}}{\in}[\sum_{i=a,\ldots,b} f[i]]}$$

where $\bar{i}$ is an arbitrary but fixed constant. As can be seen, due to the presence of finite sums ring-membership goals may be reduced to sub-goals of a different kind, namely integer-interval conditions. Simple inferences try to eliminate such sub-goals instantly (for instance, duplicate formulas are removed as soon as possible), but ultimately many of them will be left as separate sub-goals to be dealt with by their dedicated `membershipIntegerInterval` rule. Moreover, `membershipCommRing1` also incorporates one of the basic inferences of `membershipDomainTuples` for directly reducing goals of the form $\underset{\mathcal{R}}{\in}[T_i]$ to the conjunction of $\underset{\texttt{DomainTuples}[\mathcal{R}]}{\in}[T]$ and $i \in \mathbb{Z}_{1,\ldots,|T|}$.

**`CommRing1Equal`.** The main idea behind proving equalities $a = b$ in commutative rings with identity is absolutely is the same is in cancellative commutative monoids, where $+$ and $0$ take the roles of $\circ$ and $n$. The procedure is as follows:

1. First, common[6] sub-expressions of $a$ and $b$ are replaced by $0$, following precisely the same method as in `CCMonoidEqual`; $x - y$ is replaced by $x+(-y)$ and finite sums are considered as atomic. Negative sub-terms, i. e. terms with an odd number of outer unary "$-$" are moved to the respective other side of the equality to get rid of the negations. Of course, all terms that have to belong to $\mathcal{R}$ in order for a transformation to be applicable are again collected in a list $C$.

[6]Taking into account associativity and commutativity of $*$.

2. If the resulting equality $a' = b'$ is not yet trivial, the two sides are fully expanded using distributivity of $*$ for rewriting $x(y+z)$ to $xy + xz$ and $x \sum_{i=a,\ldots,b} f\texttt{[}i\texttt{]}$ to $\sum_{i=a,\ldots,b} x\, f\texttt{[}i\texttt{]}$, and associativity/commutativity of $+$ for rewriting $\sum_{i=a,\ldots,b} (f\texttt{[}i\texttt{]} + g\texttt{[}i\texttt{]})$ to $\sum_{i=a,\ldots,b} f\texttt{[}i\texttt{]} + \sum_{i=a,\ldots,b} g\texttt{[}i\texttt{]}$. Summands are expanded recursively, and once more all terms whose membership in $\mathcal{R}$ is required for a transformation to be applicable are added to $C$; now, this may also include terms of the form $f\texttt{[}\bar{i}\texttt{]}$ for arbitrary but fixed constants $\bar{i}$, originating from finite sums.

3. If the resulting equality $a'' = b''$ is different from the original one, the procedure resumes with Step 1. Otherwise, if no further simplifications could be carried out, $a'' = b''$ and the list $C$ are returned, with the meaning that $a = b$ follows from $a'' = b''$ if all terms in $C$ belong to $\mathcal{R}$.

**`CommRing1EqualKB`.** This rule does literally the same for known equalities as what `CommRing1Equal` does for equalities to be proved.

## 4.6 Formalization in Isabelle/HOL

We conclude the chapter on the formalization of Gröbner bases and reduction rings by reporting on a different formalization of the same theory in another proof assistant, namely Isabelle/HOL. Actually, it is not really the *same* theory, since in Isabelle we only considered Gröbner bases in the original setting of polynomial rings over fields, without making any appeal to reduction rings or other generalizations. Note that the theory of Gröbner bases has not been formalized in Isabelle before, although a proof procedure for handling generic equalities in commutative rings, which heavily relies on Gröbner bases techniques, is included in the system already [CW07]. The resulting formalization is now included in the Archive of Formal Proofs[7], see [IM16]; it was created with the help of Fabian Immler.

Before we give an overview of how the formalization is structured and what exactly it consists of, some general remarks on Isabelle and Isabelle/HOL are in place. Isabelle [Pau90, Pau94, Wen16] is a generic theorem prover in the tradition of the LCF system [Mil79, GMW79], implemented in ML. The term *generic* refers to the fact that Isabelle does not fix any particular object logic its users have to get along with, but that in principle *arbitrary* object logics may form the basis of formalizations in Isabelle. Examples of pre-defined environments distributed together with the system are first-order logic (Isabelle/FOL), Zermelo-Fraenkel set

[7]The Archive of Formal Proofs is an online collection of Isabelle theories, see http://afp.sf.net.

theory (Isabelle/ZF), constructive type theory (Isabelle/CTT) and, finally, higher-order logic (Isabelle/HOL, see also [NPW02]). Isabelle/HOL is the most widely used variant of Isabelle, equipped with the most extensive theory library and collection of reasoning tools. It is based on simply-typed (classical) higher-order logic with ML-style polymorphism.

In the rest of this section, familiarity with the syntax and semantics of Isabelle/HOL is assumed.

## 4.6.1 Overview of the Formalization

**Power-products.** Our formalization starts by introducing the Isabelle type-class `powerprod` of cancellative, commutative, multiplicative monoids additionally possessing a couple of further properties power-products are usually expected to have; in particular, they must be such that every infinite sequence $s$ contains two elements $s_i$ and $s_j$ with $i < j$ and $s_i \mid s_j$ (where the divisibility relation $\mid$ is pre-defined in every monoid in terms of the monoid operation).

Afterward, the parametric type 'a pp is defined as the type of all functions from type `'a` to type `nat`, the type of natural numbers. Terms $t$ of type 'a pp represent power-products in indeterminates in `'a` in the obvious way, by mapping each indeterminate to its exponent in $t$; no restrictions regarding the finiteness and/or orderedness of the type variable `'a` are imposed at this stage, although later on most of the key results can only be proved assuming that `'a` *is* finite and linearly ordered. One of these results states that 'a pp indeed constitutes a power-product type in the sense that it belongs to class `powerprod`; Dickson's lemma ensures that the requirement on sequences mentioned above is met. Although a variant of Dickson's lemma has been part of the Archive of Formal Proofs, and hence at our disposal, already [Ste12], we nevertheless proved it again in order for our formalization to be self-contained.

Finally, before turning the focus to polynomials, the notion of *term orders* as binary relations on types belonging to class `powerprod` is introduced. The only concrete term orders included in the formalization are the purely lexicographic and the degree-lexicographic ones, called `lex` and `dlex`, respectively; needless to say that we also proved that they are indeed a term orders.

Summarizing, the whole treatment of power-products, first in a very abstract setting, later in more concrete terms as functions from indeterminates to natural numbers, closely resembles the treatment of power-product domains in our Theorema-formalization, in particular in theories Polynomials.nb and PolyTuples.nb; this, of course, is not surprising at all.

**Multivariate polynomials.** The type of multivariate polynomials over power-product type `'a` and coefficient type `'b`, called ('a, 'b) `mpoly`, is defined as the type of all (coefficient-)functions from `'a` to `'b` having only finite support (meaning that `'b` must at least belong to class `zero`, for otherwise the support could not even be defined properly).

In the sequel, the various "standard" operations on polynomials, like addition, subtraction and multiplication, are introduced. These operations are first defined for functions from `'a` to `'b` and then "lifted" to type ('a, 'b) `mpoly`, using the `lift_definition` command from the Lifting package [HK13]. Afterward, dozens of more or less obvious properties of the operations thus introduced are stated and proved, including the fact that ('a, 'b) `mpoly` constitutes a commutative ring with identity (i. e. belongs to class `comm_ring_1`) if `'a` belongs to class `powerprod` and `'b` to `comm_ring_1`.

Similar to our Theorema-formalization, general multiplication of polynomials is defined in terms of an auxiliary function `monom_mult`, which takes a coefficient $c$ from `'b`, a power-product $t$ from `'a`, and a polynomial $p$ from ('a, 'b) `mpoly` as input and returns the result of multiplying $p$ by $c$ and $t$ as output (thus resembling multiplication by a monomial). Interestingly, general multiplication is in fact only needed to prove that ('a, 'b) `mpoly` forms a ring and for establishing the connection between Gröbner bases- and ideal theory (which we did), but not at all for defining the reduction relation, proving the Main Theorem, and implementing Buchberger's algorithm: there, only multiplication by monomials is needed. In reduction ring theory, in contrast, multiplication of polynomials is crucial, since first and foremost polynomial domains must constitute reduction *rings*.

At the end, fixing an arbitrary term order $\preceq$ on `'a`, the notions of leading power-product (called `lp`), leading coefficient (called `lc`) and tail (called `tail`) of a polynomial w. r. t. $\preceq$ are introduced, and so is the order relation $\preceq_p$ induced by $\preceq$ on polynomials. Summarizing, the development of multivariate polynomials in our Isabelle-formalization proceeds analogous to their development in Theorema, in theory Polynomials.nb.

Please note that the Archive of Formal Proofs already contains an entry about multivariate polynomials [ST10]. There, polynomials are represented in two ways: either as tree-expressions built from constructors like `PVar`, `PSum` etc., or directly as association lists representing coefficient functions. We, however, wanted to work with the more abstract representation of polynomials as coefficient functions (and no particular representation thereof) when stating and proving all the theoretical results, and this is the reason why we did not fall back to [ST10] but really started from scratch again. Still, we eventually introduced the concrete representations of power-products and multivariate polynomials as association lists as well for executing the various algorithms we implemented, see Section 4.6.2.

**Gröbner bases.** We present this part of the formalization more thoroughly. Clearly, the most basic concept needed for Gröbner bases theory is the reduction relation. In Isabelle it is called `red`, taking as input a set of polynomials $F$ and two polynomials $p$ and $q$, and returning `True` iff $p$ can be reduced to $q$ modulo $F$. Hence, because of currying, `red` $F$ is a binary relation of type `('a, 'b) mpoly ⇒ ('a, 'b) mpoly ⇒ bool`.[8]

After proving a couple of properties of the reduction relation, e. g. Noetherianity, Gröbner bases are defined to be sets $G$ such that `red` $G$ is Church-Rosser, just like in Theorema. The notions Church-Rosser, confluence and local confluence, as well as some further properties of arbitrary binary relations, are defined in a separate theory, which itself is completely independent of polynomials and Gröbner bases.

In Isabelle-syntax, the definition of Gröbner bases simply reads as

```
definition is_Groebner_basis::"('a,b') mpoly set ⇒ bool"
where "is_Groebner_basis F ≡ is_ChurchRosser (red F)"
```

The key result about Gröbner bases is of course Buchberger's criterion for deciding whether a given (finite) set is a Gröbner basis or not, by checking whether all S-polynomials can be reduced to 0. In Isabelle, the precise formulation of the statement is

```
theorem Buchberger_criterion:
  fixes F::"('a::powerprod, 'b::field) mpoly set"
  assumes "⋀p q. p∈F ⟹ q∈F ⟹ (red F)** (spoly p q) 0"
  shows "is_Groebner_basis F"
```

where $r^{**}$ denotes the reflexive-transitive closure of a binary relation $r$ and function `spoly` returns the S-polynomial of its arguments.

The proof of this theorem, albeit based on a multitude of auxiliary lemmas, is still quite lengthy: in its current (quite verbose) version, it is made up of almost 100 lines of Isar code, although an experienced user knowing some shortcuts could definitely shorten it drastically. Anyway, it is modeled after the proof given in [Buc98] and proceeds by showing that `red F` is "locally connective", i. e. whenever a polynomial $p$ can be reduced to both $q_1$ and $q_2$, then $q_1$ and $q_2$ can be connected below $p$ (in the sense of Definition 3.2.5). The following is a small fragment of the proof, in order to illustrate how proofs in Isabelle, in the Isar language, typically look like:

```
proof -
 have "is_loc_connective (red F) (op ≺p)"
     unfolding is_loc_connective_def
```

[8]Keep in mind that some arbitrary term order on `'a` has been fixed already.

```
  proof (intro allI, intro impI)
    fix p q1 q2
    assume "red F p q1 ∧ red F p q2"
    hence "red F p q1" and "red F p q2" by auto
    from red_setE[OF ‹red F p q1›] obtain f1 and t1
        where "f1∈F" and r1: "red_single p q1 f1 t1" .
    from red_setE[OF ‹red F p q2›] obtain f2 and t2
        where "f2∈F" and r2: "red_single p q2 f2 t2" .
    ⋮
    show "cbelow (op ≺p) p (symcl (red F)) q1 q2"
    proof (cases "t1 ∗ lp f1 = t2 ∗ lp f2")
      case False
      ⋮
    next
      case True
      ⋮
    qed
  qed
  thus ?thesis using loc_connectivity_equiv_CR
      unfolding is_Groebner_basis_def by simp
qed
```

Finally, the formalization also contains the implementation of Buchberger's algorithm as a tail-recursive function, very similar to the one GroebnerRings.nb (in particular, the function is defined for lists rather than sets of polynomials). The only major difference between the two implementations in Isabelle and Theorema, besides the fact that the latter is in the much more general setting of reduction rings, is that in Isabelle no criteria for avoiding useless reductions are taken into account at the moment. The main function, called gb, is defined explicitly in terms of the auxiliary tail-recursive function gbaux; gbaux, in turn, is defined using the Function package [Kra09]. Since the termination of Buchberger's algorithm, and hence of gbaux, is non-trivial, the Function package is not able to prove it automatically, meaning that a hand-crafted proof must be provided by the user. In Isabelle-syntax, the definitions of gb and gbaux read as

```
function gbaux::"('a::powerprod, 'b::field) mpoly list ⇒
     (('a, 'b) mpoly × ('a, 'b) mpoly) list ⇒ ('a, 'b) mpoly list"
where
   "gbaux B [] = B"|
   "gbaux B ((p,q) # R) =
      (let h = trd B (spoly p q) in
        (if h = 0 then
          gbaux B R
        else
          gbaux (h # B) (update R B h)
        )
      )"
by pat_completeness auto
termination ⟨proof⟩
```

```
definition gb::"('a::powerprod, 'b::field) mpoly list ⇒
      ('a, 'b) mpoly list" where "gb B ≡ gbaux B (pairs B)"
```

Needless to say that the correctness of function gb is proved as well:

```
theorem gb_isGB:
   shows "is_Groebner_basis (set (gb B))"
⟨proof⟩

theorem gb_ideal:
   shows "ideal (set (gb B)) = ideal (set B)"
⟨proof⟩

theorem in_ideal_gb:
   shows "p ∈ ideal (set B) ⟷ trd (gb B) p = 0"
⟨proof⟩
```

### 4.6.2 Making Things Executable

After having implemented Buchberger's algorithm and proved it correct, we of course also wanted to make it executable on actual input. First, however, we had to come up with concrete representations of power-products and multivariate polynomials. To that end, we took the approach of so-called *data refinement* [HKKN13] for turning the abstract representations of power-products and multivariate polynomials as plain functions into something more concrete. Obviously, the data structure of choice for representing (finite) functions are association lists, i. e. lists of explicit input-output pairs. Association lists are feasible for exactly representing both power-products and polynomials, since the former must be restricted to finitely many indeterminates anyway and the latter have finite support by definition, meaning that it suffices to only store the finitely many non-zero values in the representing list. In fact, the situation in Isabelle parallels the one in Theorema, where power-products and polynomials are represented as (something like) association tuples as well.

More concretely, in order to represent power-products of type 'a pp as association lists we introduced the new *code data-type* `PP`. `PP` is essentially a function mapping an association list, i. e. a list of type `('a × nat) list`, to a power-product of type 'a pp. Having `PP` it is now possible to provide so-called *code equations* specifying how the various operations on the abstract type 'a pp, like multiplication, division, etc., can effectively (and efficiently) be computed when terms of "type" `PP` are given as input. For instance, in the case of multiplication the corresponding code equation is

```
lemma compute_times_pp[code]:
"(PP xs) * (PP ys) =
  PP (map_ran (λkv. lookup xs k + lookup ys k) (merge xs ys))"
⟨proof⟩
```

where `lookup` is a simple lookup function for association lists and `map_ran` and `merge` are two functions defined in the default Isabelle/HOL theory about association lists. As can be seen, the above code equation cannot simply be stated as an axiom, but is a lemma that has to be proved. This guarantees that generated code always behaves according to the specifications and definitions of the corresponding abstract functions.

Since 'a pp is parametrized over a type `'a` of indeterminates, we provide such a type, too: the algebraic data-type of three indeterminates, called `var`, with the three constructors `X`, `Y` and `Z` representing the three indeterminates. Unfortunately, it seems to be necessary to introduce a separate indeterminate-type for each number of indeterminates, since it is not possible to parametrize types in Isabelle/HOL over, say, natural numbers (indicating the number of indeterminates).

**Example 1.** The power-product $x^2z$ is represented by `PP` as

```
PP [(X, 2), (Z, 1)]
```

when viewed as a power-product in the three indeterminates of type `var`, since the exponent of all indeterminates not appearing in the list (like `Y` above) is automatically 0. The order of the elements in the list is immaterial. ■

Multivariate polynomials are represented as association lists in pretty much the same fashion as power-products; the code data-type is now called `MP`. Actually, there is really nothing special about `MP` compared to `PP`. Only note that in contrast to the tuples representing polynomials in the Theorema-formalization, in theory PolyTuples.nb, the association lists representing polynomials in Isabelle/HOL do *not* need to be ordered w. r. t. the chosen term order.

**Example 2.** The following two `MP`-terms both represent the polynomial $x^2z^2-y$, when viewed as a polynomial of power-product type `var pp` and coefficient type `int`:

```
MP [(PP [(X, 2), (Z, 2)], 1), (PP [(Y, 1)], -1)]
MP [(PP [(Y, 1)], -1), (PP [(X, 2), (Z, 2)], 1)]
```

Of course, the coefficients of all power-products not appearing in the list are meant to be $0$. ■

Finally, executing functions on concrete input is easily possible directly within Isabelle, making use of the `value` command. For instance, computing a (not necessarily reduced) Gröbner basis of the two trivariate polynomials $x^2z^2 - y$ and $y^2z - 1$ w. r. t. the purely lexicographic term order with function gb amounts to typing

```
value [code]
   "gb_lex [
       MP [(PP [(X, 2), (Z, 2)], 1), (PP [(Y, 1)], -1)],
       MP [(PP [(Y, 2), (Z, 1)], 1), (PP [], -1)]
   ]"
```

After a couple of seconds, Isabelle outputs the Gröbner basis

```
[MP [(PP [(X, 2)], -1), (PP [(Y, 5)], 1)],
MP [(PP [(X, 2), (Z, 1)], 1), (PP [(Y, 3)], -1)],
MP [(PP [(X, 2), (Z, 2)], 1), (PP [(Y, 1)], -1)],
MP [(PP [(Y, 2), (Z, 1)], 1), (PP [], -1)]]
```

representing the set $\{x^2z^2 - y, y^2z - 1, -x^2 + y^5, x^2z - y^3\}$.

The reason for renaming gb to gb_lex is related to the fact that gb is defined inside a locale that fixes an arbitrary term order, meaning that the function itself implicitly depends on that term order as well. Generating code for a function defined inside a locale requires a bit more effort and is explained in detail on pages 37–38 in [HB16].

### 4.6.3 Comparison of the Formalizations

Ignoring the fact that the formalization of Gröbner bases theory in Isabelle is only concerned with polynomial rings over fields, whereas the one in Theorema is based on reduction rings, the overall structures of the two formalizations are quite similar. In particular, power-products and multivariate polynomials are first dealt with on a very abstract level, and only represented as concrete association lists (or tuples) when it comes to actual computations. Furthermore, the various algorithms are implemented in pretty much the same way in Isabelle and Theorema; for instance, Buchberger's algorithm in both cases is implemented as a tail-recursive function. The resulting functions can be executed directly withing the respective systems, although one must note that Isabelle/HOL supports the automatic generation of executable SML-, Haskell-, OCaml- and Scala programs [HB16], too.

Although the two formalizations are similar on the *object level*, they differ considerably on the *meta level*, especially in the additional effort of developing new tools and reasoning techniques for efficiently exploring the theory: in Theorema we had to develop the ReductionRingProver and the various tools presented in Chapter 6, whereas Isabelle/HOL by default provides a range of useful packages serving exactly the purpose of automating frequently occurring tasks in typical theory explorations, like the Lifting- and Transfer packages [HK13], the Datatype package [BBD$^+$16] and the Function package [Kra09]. Moreover, Isabelle's powerful simplification mechanism in conjunction with an extensive library of simplification rules saved us from inventing special tactics or proof procedures for handling, say, equalities in commutative rings with multiplicative identity (as in the ReductionRingProver): calling the simplifier with the default rule set, and maybe adding a dedicated set of AC-rules, was sufficient for proving all monoid-, group- and ring-equalities we encountered.

Theorema might still have some weaknesses compared to Isabelle in terms of the degree of automation of theory exploration, but in our opinion it is Theorema that gives rise to the more nicely-formatted, readable and "natural" documents, due to its close connection to *Mathematica*'s typesetting and formatting facilities. Isabelle-theories, on the other hand, are just plain (Unicode, i. e. with many special mathematical symbols) text; it is possible, though, to automatically translate the text to LaTeX code and compile it, yielding again nicely-readable PDF documents. Anyway, not only the *style* of presenting theories, and in particular proofs, differs significantly, but also the *language* of presentation: in Theorema, a big portion of the content of proof documents is informal English (or whatever language) text, explaining in detail what is going on in the proof and making it easy to follow the proof even for inexperienced users of the system. In Isabelle, in contrast, one first needs to get acquainted to the Isar proof language, both for constructing and for reading/understanding proofs.

# Chapter 5

# Complexity Analysis of Buchberger's Algorithm

Besides the formal treatment of the theory of reduction rings discussed in the previous chapter, we also formalized the complexity analysis of Buchberger's algorithm in the original setting. Hence, from now on we leave reduction rings again and restrict the domain of discourse to polynomial rings over fields.

Most of this chapter is also contained in [Mal14], and parts of it have been published in [MB14, BJK$^{+}$16].

## 5.1 Underlying Theory

The complexity of Buchberger's algorithm in the *bivariate case* was investigated by Buchberger around 1980 in [Buc79, BW79, Buc83b, Buc83c]. Hence, it has to be pointed out that, similar to Chapter 3, this section does not present any significant new results obtained in the frame of our studies, but rather summarizes known results our formalization of the complexity analysis in Theorema is based upon. Still, it also must be mentioned that indeed some minor improvements, namely generalizations and simplifications, compared to the original elaboration could be achieved; they will be pointed at explicitly in the upcoming subsections.

Since we are back in the original setting, we fix now an arbitrary field $K$, a finite set of indeterminates $X = \{x_1, \ldots, x_k\}$ and a term order $\preceq$ on $X$. Although the majority of results presented below is valid only in the bivariate case, i. e. for $k = 2$, and only for graded term orders, we nevertheless state most definitions and some intermediate lemmas for arbitrary $k$ and arbitrary term orders.

The following (standard) notation is used throughout this chapter:

- $\mathrm{lp}(p)$ denotes the leading power-product (w. r. t. $\preceq$) of $p \in K[X]\backslash\{0\}$.

- $\operatorname{lcm}(s,t)$ denotes the least common multiple of $s,t \in [X]$ (where $[X]$ is again the set of power-products in indeterminates $X$).
- $\deg(s)$ denotes the degree of $s \in [X]$.
- $s \mid t$ asserts that $s$ divides $t$, for $s,t \in [X]$.

*Convention* 16. Here and henceforth we assume that every non-zero polynomial is *monic*, i. e. has leading coefficient 1. Otherwise, a non-monic polynomial can of course always be made monic by dividing through its leading coefficient, without affecting on the computation of Gröbner bases by Buchberger's algorithm *at all.*

### 5.1.1 The Algorithm

The algorithm whose complexity we are interested in is Algorithm 2. The similarities between this algorithm and Algorithm 1 in the reduction ring setting are apparent, and the differences are sketched in Section 3.5.1. sPoly and trd are auxiliary functions for computing the S-polynomial of two input polynomials and for totally reducing a polynomial to normal form, respectively.

**Algorithm 2** Buchberger's algorithm in the original setting

**Input:** $F = \{f_1, \ldots, f_n\} \subseteq K[X]$
**Output:** $G \subseteq K[X]$ s. t. $G$ is a Gröbner basis of $F$

```
function polyGB($F$)
    $P \leftarrow \{(f_i, f_j) | 1 \leq i < j \leq n\}$
    $G \leftarrow F$
    while $P \neq \emptyset$ do
        $(p, q) \leftarrow$ some element from $P$
        $P \leftarrow P \setminus \{(p, q)\}$
        if $\neg \mathrm{ccritP}(p, q, G)$ then
            $h \leftarrow$ sPoly$(p, q)$
            $h \leftarrow$ trd$(h, G)$
            if $h \neq 0$ then
                $P \leftarrow P \cup \{(g, h) | g \in G\}$
                $G \leftarrow G \cup \{h\}$
            end if
        end if
    end while
    return $G$
end function
```

The use of the *chain criterion* in Line 7 of Algorithm 2 is crucial for analyzing its complexity; without it, the results obtained simply would not hold. However, the chain criterion must be slightly rephrased compared to Definition 3.5.5 in order to fit into our current setting; in this chapter, it is hence suffixed by "P" (for "Polynomials", because $\mathrm{ccritP}$ only makes sense for polynomials) for distinguishing the two variants.

**Definition 5.1.1** (Chain Criterion in Polynomial Rings)**.** Let $G \subseteq K[X]$, let $p, q \in K[X] \backslash \{0\}$ and set $r := \mathrm{lcm}(\mathrm{lp}(p), \mathrm{lp}(q))$. Then $\mathrm{ccritP}(p, q, G)$ holds iff there exists $g \in G$ such that

1. $\mathrm{lp}(g) \mid r$,
2. $\deg(\mathrm{lcm}(\mathrm{lp}(p), \mathrm{lp}(g))) < \deg(r)$ and
3. $\deg(\mathrm{lcm}(\mathrm{lp}(q), \mathrm{lp}(g))) < \deg(r)$.

Please observe that the chain criterion used here is only one variant among many. For instance, alternatively one could compare the three least common multiples $\mathrm{lcm}(\mathrm{lp}(p), \mathrm{lp}(g))$, $\mathrm{lcm}(\mathrm{lp}(q), \mathrm{lp}(g))$ and $r$ not w. r. t. their degrees, but directly w. r. t. the term order $\preceq$. The reason for defining $\mathrm{ccritP}$ as above is that we will mainly restrict $\preceq$ to *graded* orders anyway. Moreover, the differences between the different variants are comparatively small.

## 5.1.2 Complexity of the Algorithm

It is well-known that the asymptotic complexity of Algorithm 2 is double exponential in the number of indeterminates $k$ [MM82], and, for a fixed number of indeterminates, polynomial in the maximum degree of the input set $F$ [Giu84, MM84].

In order to obtain sharp bounds on the complexity of the algorithm in the case of two indeterminates, $k = 2$, in terms of the number of elementary operations that must be executed for given input $F$, it turns out to be sufficient to only know bounds on the degrees of the polynomials in the resulting Gröbner basis $G$, as shown in [Buc79]:

**Theorem 5.1.2.** *For any non-empty finite $F \subset K[x, y]$ let*

$$D_F := \max\{\deg(\mathrm{lp}(g)) \mid g \in \textit{POLYGB}(F)\}$$

*i. e. the maximum degree of all leading terms in the Gröbner basis computed by Algorithm 2. Furthermore, let $C_F := \frac{(D_F+2)(D_F+1)}{2}$. Then at most*

$$\binom{|F| + C_F}{2} \cdot \left( C_F \left( |F| + C_F \right) + \binom{C_F}{2} \right)$$

*additions, multiplications and comparisons of polynomials are needed to compute* $\textsc{poly}GB(F)$.

Because of Theorem 5.1.2 the remaining part of this section is all about obtaining tight bounds for $D_F$ in terms of the degrees of the polynomials in $F$. Moreover, this is precisely what our formalization in Theorema is exclusively about; no connection to the actual algorithm is made there.

In order to derive bounds for $D_F$, at the very beginning the case of arbitrary term orders is reduced to the case of *graded* orders, i. e. orders where the first criterion to determine which of two power-products is greater is their degree. Knowing a bound for such orders one can easily derive a bound that holds for any term order, if the ideal in question is 0-dimensional. For more details on this we refer to [Buc83c]. Summarizing, from now on we assume that $\preceq$ is a graded term order, which, furthermore, is the only case that is dealt with in our formalization.

The final result about $D_F$ is summarized in the following

**Theorem 5.1.3.** *For all non-empty finite* $F \subset K[x, y]$ *and all graded term orders* $\preceq$*: the Gröbner basis* $G$ *computed by Algorithm 2 on input* $F$ *w. r. t.* $\preceq$ *satisfies*

$$\operatorname{maxdeg}(G) \leq 2 \operatorname{maxdeg}(F) \tag{5.1}$$

*where* $\operatorname{maxdeg}(G)$ *and* $\operatorname{maxdeg}(F)$ *denote the maximum degrees of the leading terms of the polynomials in* $G$ *and* $F$*, respectively.*

## 5.1.3 Proof Outline

We now describe the general strategy for proving Theorem 5.1.3 according to Buchberger's original papers. In our Theorema-formalization, we follow more or less the same strategy (up to some minor deviations).

**1. Exponent vectors.** First of all, the problem of bounding the degrees of polynomials is reduced from a commutative-algebra- to a *combinatorial* problem, by mapping each non-zero polynomial to the exponent vector of its leading term (w. r. t. the graded ordering $\preceq$). This is justified by the fact that in Algorithm 2 it is only the leading terms of polynomials that influence the behavior of the algorithm and the contents of resulting Gröbner basis (e. g. when forming S-polynomials). Exponent vectors are pairs of natural numbers, meaning that from now on we work solely in the space $\mathbb{N}^2$, and no appeal to polynomials is made any longer. This, in fact, is now precisely where the formalization in Theorema starts: there, everything is concerned with exponent vectors (and sets thereof) rather than polynomials. Since functions and predicates like $\operatorname{lcm}$, $\deg$, divisibility and $\mathrm{ccritP}$, originally defined for power-products and polynomials, in fact only depend on their arguments' exponent vectors, the same functions and predicates, by abuse of notation, will also be used for exponent vectors. For instance, if $p$ and $q$ are two exponent vectors, then $p \mid q$ holds iff $p_i \leq q_i$ for $i = 1, 2$, where $p_i$ and $q_i$ refer to the $i$-th component of $p$ and $q$, respectively.[1]

**2. Loop invariant.** For each non-empty finite $G \subseteq \mathbb{N}^2$ (corresponding to the current basis in Algorithm 2) the quantity $M_G + W_G$ is shown to be some kind of "loop invariant" of the main loop in the algorithm, in the sense that it does not *increase* but might only *decrease*. $M_G$ and $W_G$ are defined as

$$M_G := \max\{\deg(\mathrm{lcm}(a, b)) \mid a, b \in G \wedge \neg\mathrm{ccritP}(a, b, G)\} \tag{5.2}$$
$$W_G := \min\{e_1 \mid e \in G\} + \min\{e_2 \mid e \in G\} \tag{5.3}$$

and the goal of this second step is to show

$$M_{G'} + W_{G'} \leq M_G + W_G \tag{5.4}$$

where $G'$ is obtained from $G$ by adding a new exponent vector $h$, corresponding to line 12 of Algorithm 2 where a new polynomial is added to the current basis. Of course, $h$ is not completely arbitrary but has some specific properties, like the very important $\deg(h) \leq M_G$ since $\preceq$ is graded and $h$ corresponds to a polynomial obtained by reducing the S-polynomial of two polynomials $p$ and $q$ violating the chain criterion, meaning that by definition of $M_G$ we know $\deg(\textsc{sPoly}(p, q)) \leq M_G$.

**3. Maximum degree.** For each non-empty finite $F \subseteq \mathbb{N}^2$, $\mathrm{maxdeg}(F)$ is shown to be bounded from above by $M_F$, i. e.

$$\mathrm{maxdeg}(F) \leq M_F \tag{5.5}$$

$\mathrm{maxdeg}(F)$ is defined for sets of exponent vectors analogous to sets of polynomials according to Theorem 5.1.3.

**4. Degree bound.** The quantity $M_F + W_F$ that was shown not to increase in the course of the algorithm in step 2 is now shown to be bounded from above by $2\,\mathrm{maxdeg}(F)$, for all non-empty finite $F \subseteq \mathbb{N}^2$, i. e.

$$M_F + W_F \leq 2\,\mathrm{maxdeg}(F) \tag{5.6}$$

[1]Exponent vectors are tuples, hence the subscript-notation.

As soon as all this is established, the whole elaboration is finished, as we can now conclude

$$\operatorname{maxdeg}(G) \leq_{(5.5)} M_G \leq M_G + W_G \leq_{(*)} M_F + W_F \leq_{(5.6)} 2\operatorname{maxdeg}(F)$$

where $F$ is the set of exponent vectors corresponding to the input of Algorithm 2 and $G$ to its output. $(*)$ is justified by an inductive argument making use of formula (5.4).

### 5.1.4 Improvements in the Formalization

As indicated already above, we were able to achieve some improvements in the formalization in Theorema compared to the original elaboration of the theory by Buchberger in the referenced literature. Before having a closer look at the formalization in Section 5.2, we list and discuss the improvements below.

**Ground domain.** By definition, exponents of power-products are non-negative integers. Hence, it is only natural to carry out steps 2–4 of the general proof strategy in the space $\mathbb{N}^2$, just as described in the previous subsection; this is exactly how it is done in [BW79, Buc83c].

However, since absolutely no appeal to polynomials is made in these steps and really everything happens in the "space of exponent vectors" $\mathbb{N}^2$, there is nothing that hinders us from trying to generalize the ground domain from $\mathbb{N}$ to wider classes of mathematical structures (even though this might not make sense when going back to polynomials in the end). And indeed, a detailed analysis of the proofs of the various formulas revealed that only quite a few properties of $\mathbb{N}$ are actually needed, therefore allowing us to replace $\mathbb{N}$ by the much wider class of so-called *totally-ordered commutative monoids*, defined as follows:

**Definition 5.1.4.** $(\mathcal{M}, +, \leq)$ is a totally-ordered commutative monoid iff

1. $(\mathcal{M}, +)$ is a cancellative commutative monoid,
2. $(\mathcal{M}, \leq)$ is a total ordering, and
3. $+$ is an order embedding (or monotonic) w. r. t. $\leq$ on $\mathcal{M}$, in the sense

$$x \leq y \;\Rightarrow\; x + z \leq y + z$$

   for all $x, y, z \in \mathcal{M}$.

As can easily be seen, apart from $\mathbb{N}$ there are many other well-known mathematical structures that are totally-ordered commutative monoids, including $\mathbb{Z}$,

$\mathbb{Q}$, $\mathbb{R}$, and $\mathbb{C}$ with any order relation that corresponds to a term order (e. g. lexicographic). So, replacing $\mathbb{N}$ by totally-ordered commutative monoids really is a massive generalization.

There is one important thing to note, though: apparently, totally-ordered commutative monoids are not required to have a least element, or even to be well-ordered by $\leq$. This might appear strange at first sight, since domains that are not well-ordered make it impossible to draw any conclusions about the complexity of Buchberger's algorithm, even when knowing bounds on the degrees of the polynomials in the final Gröbner basis. But keep in mind that steps 2–4 of Section 5.1.3 are only concerned with finding exactly such degree bounds, but *not* with the actual complexity of the algorithm. The degree bound $2\,\mathrm{maxdeg}(F)$ is valid for all non-empty finite $F \subset \mathcal{M}^2$ (if $\mathcal{M}$ is a totally-ordered commutative monoid and a graded order on $\mathcal{M}^2$ is used), but deriving actual complexity results, as it is done in Theorem 5.1.2, is indeed only possible if much stronger constraints on $\mathcal{M}$ are imposed.

We fix now a totally-ordered commutative monoid $(\mathcal{M}, +, \leq)$ that serves as the ground domain in the remainder of this chapter. Also recall that $k$ denotes the number of indeterminates, i. e. the dimension of the *exponent space* $\mathcal{M}^k$.

**Number of indeterminates.** Although the main result, Theorem 5.1.3, is only proved for the case of two indeterminates $k = 2$, or, in exponent vector parlance, in the space $\mathcal{M}^2$, many intermediate results are stated and proved for arbitrary $k$. This will certainly prove a huge benefit if the complexity of Buchberger's algorithm in a higher number of indeterminates (e. g. in three indeterminates, following [Win84]), or even arbitrarily many indeterminates, is investigated by similar means in the future.

**Cover of exponent space.** The next improvement is a bit technical: in the proof of Formula (5.4) various cases are distinguished depending on the relative position of the new vector $h$ in $\mathcal{M}^2$ w. r. t. the current set $G$. To this end, the exponent space $\mathcal{M}^2$ (where the dimension $k$ really must be 2 now) is partitioned into several sets: in the original elaboration in [BW79] these sets are "above $G$", "below $G$", "interior of $G$", and "exterior of $G$".[2]

In our formalization in Theorema, we consider a different splitting of the exponent space: "above $G$" (same as before), "rectangular region of $G$", and "far exterior of $G$" (again divided into two symmetric sets). This splitting only *covers* the exponent space but does not *partition* it, since the set "above $G$" and the rectangular region of $G$ are not disjoint in general.

[2]Strictly speaking, the exterior is again divided into two sets that can be dealt with by symmetric arguments, though.

Before we can give the exact definitions of the three cover-sets, we need some auxiliary notions; therefore, let here and henceforth $G$ be a non-empty finite set of exponent vectors:

- $\mathrm{ccritP}$ as defined above, with the slight modification that it is now defined for exponent vectors rather than polynomials.
- Function $\cdot^- : \{1,2\} \to \{1,2\}$, defined as $1^- := 2, 2^- := 1$.
- Exponent vector $L(G)$ denotes the least common multiple of *all* exponent vectors in $G$.
- Exponent vector $K(G)$ denotes the greatest common divisor of *all* exponent vectors in $G$ (such that $W_A = \deg(K(G))$).

With these auxiliary notions, the three sets (or, more precisely, predicates) covering the exponent space $\mathcal{M}^2$ are defined according to

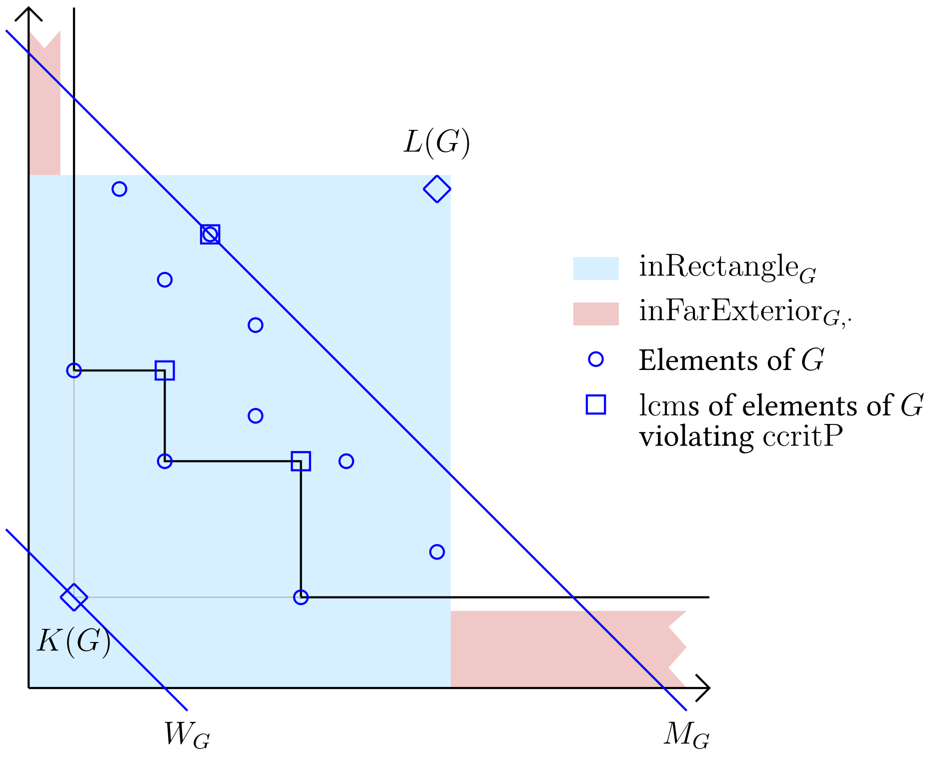
**Definition 5.1.5** (Cover of Exponent Space)**.** Let $G$ be a non-empty finite set of exponent vectors in $\mathcal{M}^2$. Then

$$\begin{aligned}
\mathrm{isAbove}_G(x) \quad &:\Leftrightarrow \quad \underset{g \in G}{\exists}\, g \mid x \\
\mathrm{inRectangle}_G(x) \quad &:\Leftrightarrow \quad x \mid L(G) \\
\mathrm{inFarExterior}_{G,i}(x) \quad &:\Leftrightarrow \quad x_i < K(G)_i \wedge x_{i^-} > L(G)_{i^-}
\end{aligned}$$

As can easily be verified, every $x \in \mathcal{M}^2$ satisfies at least one of $\mathrm{isAbove}_G$, $\mathrm{inRectangle}_G$ or $\mathrm{inFarExterior}_{G,i}$ for $i = 1$ or $i = 2$, justifying our claim that the predicates really cover the whole exponent space. Figure 5.1 illustrates the newly introduced notions in $\mathbb{N}^2$ ($\mathrm{isAbove}_G$ is not explicitly shown, but it is just the whole region “above” the staircase).

The reason for this deviation from the literature is the following: the new rectangular region of $G$ comprises the whole set “below $G$”, the interior of $G$ and parts of the exterior of $G$ from the original partition. Hence, what have previously been three cases is now only one single case that, furthermore, can be dealt with by an elegant argument that in fact proves the much stronger claim $M_{G'} \leq M_G$ for $h$ lying in the rectangular region of $G$; see below.

**Simplification of proof of (5.4).** The proof of $M_{G'} + W_{G'} \leq M_G + W_G$, where $G' = G \cup \{h\}$ for the $h$ computed in Line 9 of Algorithm 2 (or, more, precisely, its corresponding exponent vector), proceeds by distinguishing two cases based on whether $h$ lies in the rectangular region of $G$ or in the far exterior of $G$; the third possibility, $\mathrm{isAbove}_G(h)$, cannot happen for otherwise $h$ would be further



$L(G)$
$K(G)$
$W_G$
$M_G$
inRectangle$_G$
inFarExterior$_{G,\cdot}$
Elements of $G$
lcms of elements of $G$ violating ccritP

Figure 5.1: Cover of the exponent space $\mathbb{N}^2$ w. r. t. some set $G$.

reducible modulo $G$. Moreover, from the construction of $h$ we can immediately infer $\deg(h) \leq M_G$, too.

Let us consider the case $\text{inRectangle}_G(h)$ now: in that case it turns out that not only $M_{G'} + W_{G'} \leq M_G + W_G$, but even $M_{G'} \leq M_G$; $W_{G'} \leq W_G$ is obvious anyway. This claim is nowhere stated explicitly in the literature, hence we state and prove it here.

**Theorem 5.1.6.** *Let $G \subseteq \mathcal{M}^2$ be non-empty and finite, and let $h \in \mathcal{M}^2$ be such that $\text{inRectangle}_G(h)$ and $\deg(h) \leq M_G$ hold. Then $M_{G'} \leq M_G$, where $G' = G \cup \{h\}$.*

*Proof.* Let $G \subseteq \mathcal{M}^2$ be an arbitrary but fixed non-empty finite set of exponent vectors, and let $h \in \mathcal{M}^2$ be an arbitrary but fixed exponent vector satisfying

$$\deg(h) \leq M_G \tag{A.1}$$

$$\text{inRectangle}_G(h) \tag{A.2}$$

We have to show $M_{G'} \leq M_G$, which is accomplished by showing

$$\deg(\text{lcm}(a, h)) \leq M_G$$

for every element $a \in G$ such that $\operatorname{ccritP}(a, h, G)$ does not hold (follows readily from the definition of $M$ in (5.2)). Hence, we choose some a. b. f. $a \in G$, assume

$$\neg \operatorname{ccritP}(a, h, G) \tag{A.3}$$

and show

$$\deg(\operatorname{lcm}(a, h)) \leq M_G \tag{G.1}$$

Now we distinguish four cases, based on the relative positions of $a$ and $h$ in $\mathcal{M}^2$.

**Case I:** $h_1 \leq a_1$ and $h_2 \leq a_2$.
In this case we obviously have $\operatorname{lcm}(a, h) = a$ and hence also $\deg(\operatorname{lcm}(a, h)) = \deg(a) \leq \operatorname{maxdeg}(G)$. Together with (5.5) we get the desired result.

**Case II:** $a_1 \leq h_1$ and $a_2 \leq h_2$.
In this case we obviously have $\operatorname{lcm}(a, h) = h$ and hence also $\deg(\operatorname{lcm}(a, h)) = \deg(h)$. Together with assumption (A.1) we get the desired result.

**Case III:** $a_1 < h_1$ and $h_2 < a_2$.
In order to prove (G.1) it suffices to find an element $b$ with

$$\deg(\operatorname{lcm}(a, h)) \leq \deg(\operatorname{lcm}(a, b)) \tag{G.2}$$

$$\neg \operatorname{ccritP}(a, b, G) \tag{G.3}$$

Let $C := \{\underset{c \in G}{c \mid} h_1 \leq c_1\}$. $C$ is finite, and because of assumption (A.2) it is also non-empty, meaning that it contains an element $b$ such that $b_1$ is minimal among all $c_1$ for $c \in C$ (c. f. Figure 5.2). Of course, in general such a $b$ might not be unique, but this does not matter.

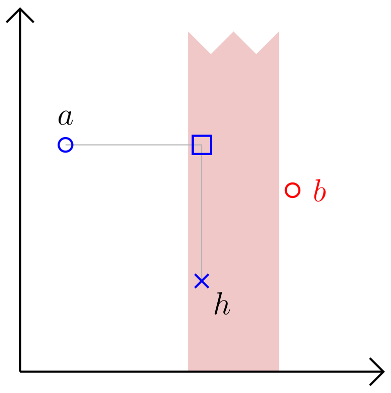


Figure 5.2: The relative positions of $a$, $b$ and $h$. No element of $G$ lies in the shaded region.

We claim that $b$ witnesses (G.2) and (G.3). (G.2) is trivially witnessed by $b$, since

$$\begin{aligned}\deg(\operatorname{lcm}(a,h)) &\underset{\text{case assumption}}{=} h_1 + a_2 \leq \\ &\underset{h_1 \leq b_1}{\leq} b_1 + a_2 \leq \\ &\leq \deg(\operatorname{lcm}(a,b))\end{aligned}$$

For proving (G.3) we again distinguish two cases.

**Case III.A:** $a_2 \leq b_2$.
In this case we have $a \mid b$, and therefore $\operatorname{ccritP}(a,b,G)$ certainly does not hold (as always if one vector divides the other, as can easily be verified).

**Case III.B:** $b_2 < a_2$.
According to the definition of ccritP in Definition 5.1.1, we have to prove that there does not exist $g \in G$ with

$$g \mid \langle b_1, a_2 \rangle \tag{G.4}$$

$$\deg(\operatorname{lcm}(a,g)) < b_1 + a_2 \tag{G.5}$$

$$\deg(\operatorname{lcm}(b,g)) < b_1 + a_2 \tag{G.6}$$

We assume the opposite, i. e. we assume that there exists some $g$ with all these properties. In fact, as one can easily prove, (G.4), (G.5) and (G.6) can only be satisfied if $g$ fulfills

$$g_1 < b_1 \tag{A.4}$$

$$g_2 < a_2 \tag{A.5}$$

(A.4) together with the minimality of $b_1$ now implies

$$g_1 < h_1 \tag{A.6}$$

Figure 5.3 illustrates the possible positions of $g$.

However, the existence of $g$ satisfying both (A.5) and (A.6) contradicts (A.3), as can be seen easily.

**Case IV:** $h_1 < a_1$ and $a_2 < h_2$. Analogous to Case III. □

**Simplification of proof of (5.6).** Originally, [Buc83c] proves Formula (5.6) by first reducing the case of arbitrary sets $F$ to the case of so-called *contours* and then proving the formula only for contours. The latter part is easy, but the reduction of arbitrary sets to contours is very cumbersome and involves many tedious case distinctions, making up in total eleven pages of the paper. However, a close investigation revealed that all this cumbersome reduction to contours is

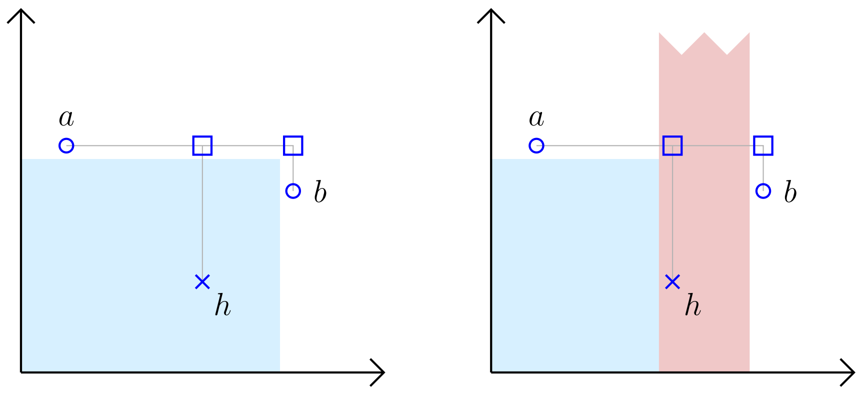


Figure 5.3: The blue-shaded region is where $g$ might lie, before (left) and after (right) taking into account the minimality of $b_1$.

*not needed at all*, since the proof of the second part given for the case of contours works more or less in exactly the same way for *any* set of exponent vectors. Since it is really short, we spell out the proof in full detail:

*Proof of (5.6).* Choose non-empty finite $F \subseteq \mathcal{M}^2$ arbitrary but fixed, where $\mathcal{M}$ is a totally-ordered commutative monoid. Recall the definitions of $L(F)$ and $K(F)$ as the least common multiple and greatest common divisor of all elements in $F$, respectively. Hence,

$$\begin{aligned} L(F)_i &:= \max\{e_i \mid e \in F\} \\ K(F)_i &:= \min\{e_i \mid e \in F\} \end{aligned}$$

for $i = 1, 2$. We apparently have $W_F = \deg(K(F))$; moreover, since $M_F$ is the maximum degree of the least common multiples of some exponent vectors in $F$, it can certainly be bounded from above by $\deg(L(F))$. Thus:

$$\begin{aligned} M_F + W_F = M_F + \deg(K(F)) \leq \deg(L(F)) + \deg(K(F)) = \\ = (L(F)_1 + K(F)_2) + (L(F)_2 + K(F)_1) \end{aligned}$$

We show that both summands on the right-hand-side of the last equality can be bounded from above by $\operatorname{maxdeg}(F)$, which finishes the proof. W. l. o. g. we only consider the first summand $L(F)_1 + K(F)_2$; the other one can be treated analogously. Since $L(F)_1$ is the maximum first component of all vectors in $F$, there must be some vector $e \in F$ with $e_1 = L(F)_1$. By definition of $K(F)$, the second component of $e$, $e_2$, must be at least as big as $K(F)_2$, allowing us to conclude

$$L(F)_1 + K(F)_2 = e_1 + K(F)_2 \leq e_1 + e_2 = \deg(e) \leq \operatorname{maxdeg}(F)$$

□

## 5.2 Formalization in Theorema

We now discuss the formalization of the complexity analysis of Buchberger's algorithm in Theorema. Of course, this formalization parallels the one of reduction ring theory in many respects, but there are still some differences that must be pointed at explicitly; note, in particular, that the complexity analysis was dealt with *before* reduction ring theory. In fact, the most apparent difference between the two formalizations concerns their overall structure: whereas reduction ring theory is split across 15 sub-theories in total, the complexity analysis is altogether contained in a single Theorema theory, i. e. Theorema notebook, called Complexity.nb. The reason for this is quite simple: because of the relatively small size of the formalization (see below), there was no need to split it into sub-theories.

The Theorema-formalization of the complexity analysis consists in total of 292 formulas, 230 of which have been proved semi-automatically in Theorema using the special prover described in Section 5.3. The remaining 62 formulas are axioms and definitions that do not require any kind of proofs (including well-definedness proofs) at all. Table 5.1 lists the numbers of axioms/definitions and theorems in more detail. The average size of proofs in terms of the number of inference steps is 40.3, and the largest proof in the formalization requires 285 steps.

| | Axioms | Theorems | Total |
|---|---:|---:|---:|
| Basic notions | 37 | 70 | 107 |
| Specific notions | 25 | 160 | 185 |
| **Total** | **62** | **230** | **292** |

Table 5.1: Size of the formalization in terms of the numbers of formulas.

When comparing the sizes of the two formalizations of reduction ring theory and the complexity analysis one must be careful, though: the proofs of the former were carried out mostly *fully interactively*, whereas the proofs of the latter were carried out *(semi-) automatically*, with much less user interaction. This, however, does not mean that more sophisticated and powerful automatic proving facilities were at our disposal for the complexity analysis, but only that we had to divide complicated theorems into lots of simple lemmas that could be proved automatically; moreover, proofs of course tend to be longer when created automatically, because an experienced human operator guiding the proof search usually has some intuition about how to construct the proof in the best, i. e. shortest way. See also Section 7.1 for a discussion of automatic- versus interactive proof development.

Before we finally have a closer look at the individual parts of the formalization, we list some design decisions and deviations from what is presented in the previous section:

- Similar as in theory Polynomials.nb, the whole formalization is only concerned with an arbitrary but fixed totally-ordered commutative monoid $\overline{\mathcal{D}}$ as the underlying domain of discourse.[3] Furthermore, the dimension is fixed to the arbitrary but fixed constant $\overline{\mathrm{k}}$ as well; results holding only in the two-dimensional case are proved with $\overline{\mathrm{k}} = 2$ among the assumptions.

- *Tuples* are used instead of sets throughout the whole formalization; sets do not appear at all. Note that all sets mentioned in the previous section have to be finite anyway (e. g. sets of exponent vectors), justifying their replacement by (finite!) tuples.

- The *negated* version of the chain criterion, as given in Definition 5.1.1, is used. This does not have any deeper reason, except that in most of the lemmas and theorems one has to assume that a pair of basis elements cannot be ruled out by the chain criterion, i. e. that it does *not* hold, and hence negating it in the first place slightly simplifies such assumptions.

Moreover, it is important to note that in contrast to the formal treatment of reduction ring theory, $\mathbb{Z}$ and intervals thereof are not interpreted as sets but rather as *domains* in the present formalization, meaning that $\underset{\mathbb{Z}}{\in}[x]$ replaces $x \in \mathbb{Z}$, $x \underset{\mathbb{Z}}{+} y$ replaces $x + y$ and $x \underset{\mathbb{Z}}{\leq} y$ replaces $x \leq y$.

We now describe the coarse structure of our formalization in Theorema.

## 5.2.1 Basic Notions

Complexity.nb begins with the definitions of various basic notions the complexity analysis builds upon. These include tuples, $+$ and $\leq$ on $\mathbb{Z}$ (needed because tuples are indexed by natural numbers), and of course the class of totally-ordered commutative monoids (called `isTOM`), according to Definition 5.1.4. Note that in total only twelve basic properties of tuples and integers are required, e. g. what it means for two tuples to be equal, how the append-function is defined, that $\mathbb{Z}$ itself constitutes a totally-ordered commutative monoid, and that $\leq$ is a discrete order relation on $\mathbb{Z}$. These properties are not proved, though, but just stated as axioms.

[3]In the formalization this domain is actually just called $D$, but following our convention of denoting domains by calligraphic characters and adding over-bars to a. b. f. constants we write $\overline{\mathcal{D}}$ here.

In addition, the crucial notions of *minimum* (`min`) and *maximum* (`max`) of a tuple of elements in an ordered domain are introduced. They are defined as domain functions of an underlying domain $\mathcal{D}$ in terms of the `argmin` and `argmax` binders, respectively; for given $\mathcal{D}$, $\underset{\mathcal{D}}{\leq}$ is taken as the order relation $\underset{\mathcal{D}}{\mathtt{min}}$ and $\underset{\mathcal{D}}{\mathtt{max}}$ depend upon. Please note that the semantics of `argmin` and `argmax` is *not* defined by means of formulas, but solely on the inference level by two rules of the special prover described in Section 5.3.5. `argmin` and `argmax` are variable binders, necessitating the availability of higher-order rewriting if they are to be defined as mere object-level formulas; higher-order rewriting as described in Section 6.1, however, has not been implemented in Theorema at that time yet.

Afterward, a couple of more or less obvious lemmas about the previously introduced notions are proved. Most of them are concerned with properties of the order relation in totally-ordered commutative monoids (e. g. monotonicity properties) and properties of `min` and `max`, also in totally-ordered commutative monoids. Although the domain of exponents is fixed to $\overline{\mathcal{D}}$, all these results are stated and proved for *all* totally-ordered commutative monoids, because in addition to $\overline{\mathcal{D}}$, $\mathbb{Z}$ forms such a structure as well and the results can be thus be used for $\overline{\mathcal{D}}$ and $\mathbb{Z}$ alike.

Two of the crucial properties of `min` (and `max`) that are proved in Complexity.nb are the following:

$$\underset{\mathtt{isTOM}[\mathcal{M}]}{\forall}\ \underset{\underset{\mathcal{M}}{\in}[x]}{\forall}\ \underset{\underset{\mathtt{DomainTuples}[\mathcal{M}]}{\in}[A]}{\forall}$$

$$1 \underset{\mathbb{Z}}{\leq} |A| \Rightarrow$$

$$\underset{\mathcal{M}}{\mathtt{min}}[A \curvearrowleft x] = \underset{\mathcal{M}}{\mathtt{min}}[\langle \underset{\mathcal{M}}{\mathtt{min}}[A], x\rangle] \qquad (min\curvearrowleft)$$

$$\underset{\mathcal{M}}{\mathtt{min}}[A] \underset{\mathcal{M}}{+} x = \underset{\mathcal{M}}{\mathtt{min}}[\langle A_i \underset{\mathcal{M}}{+} x \ \underset{i=1,\ldots,|A|}{|}\ \rangle] \qquad (min+)$$

*Remark* 17. Clearly, most of the basic notions and proved properties thereof are contained in the elementary theories compiled in the frame of our formalization of reduction ring theory. The only reason why Complexity.nb starts from scratch and does not make use of these elementary theories is that the complexity analysis was formalized *before* reduction ring theory, at a time when they have not been available yet. In fact, the two formalizations are *completely independent* of each other.

### 5.2.2 Specific Notions

After having introduced general-purpose notions the complexity analysis builds upon, now *specific* notions directly related to it are introduced, all of them being defined in terms of the arbitrary but fixed constants $\overline{\mathcal{D}}$ and $\overline{\mathrm{k}}$. For instance, the unary predicate `isExponentVector` expressing that its argument is an exponent vector, i. e. an element of $\overline{\mathcal{D}}^{\overline{\mathrm{k}}}$, is simply defined as

$$\underset{x}{\forall}\quad \texttt{isExponentVector}[x] :\Leftrightarrow \bigwedge\begin{cases} \underset{\texttt{DomainTuples}[\overline{\mathcal{D}}]}{\in}[x] \\ |x| = \overline{\mathrm{k}} \end{cases} \qquad (\textit{isExponentVector})$$

Apart from exponent vectors and tuples thereof (characterized by the unary predicate `isExpVectorTuple`), also divisibility, degrees and least common multiples of exponent vectors are defined in a straight-forward manner. Then, the chain criterion (or, more precisely, its negation) is introduced for two exponent vectors w. r. t. a tuple thereof; it is called `chainCrit` in the formalization but referred to as `chainCritP` here in order not to confuse it with the analogous predicate in theory GroebnerRings.nb of reduction ring theory.

Finally, `maxdeg`, `M`, `W`, `L` and the three predicates `isAbove`, `inRectangle` and `inFarExterior` are introduced. Their definitions are exactly as in Section 5.1, except that the tuple of exponent vectors these notions depend upon is not denoted by a subscript, but just passed as an additional argument: $\texttt{M}[A]$ instead of $\texttt{M}_A$ and $\texttt{inFarExterior}[x, A, i]$ instead of $\texttt{inFarExterior}_{A,i}[x]$. For instance, the definition of `M` in Complexity.nb is

$$\underset{A,\,x}{\forall}$$

$$\texttt{M}[A] := \underset{\overline{\mathcal{D}}}{\texttt{max}}[\langle \texttt{M}[A_i, A] \underset{i=1,\ldots,|A|}{|} \rangle] \qquad (M)$$

$$\texttt{M}[x, A] := \underset{\overline{\mathcal{D}}}{\texttt{max}}[\langle \texttt{deg}[\texttt{lcm}[x, A_i]] \underset{i=1,\ldots,|A|}{|} \texttt{chainCritP}[x, A_i, A]\rangle] \qquad (M\,2)$$

where once again one must recall that $\texttt{chainCritP}[x, y, A]$ holds iff the reduction of the S-polynomial corresponding to the pair $x$ and $y$ *cannot* be avoided. As can be seen, `M` also takes the degrees of the least common multiples of pairs of identical elements, i. e. $\texttt{deg}[\texttt{lcm}[A_i, A_i]]$, into account (because `chainCritP` always trivially holds for such pairs). This might appear a bit strange at first sight, because Algorithm 2 never considers pairs of identical elements, as their S-polynomial is known to be 0 anyway. However, the definition of M as given above is justified by the fact that identical pairs do not matter at all—the resulting value is always the same regardless of whether identical pairs are considered or not. The proof of this claim is part of our formalization.

### 5.2.3 Main Results

At the end, Complexity.nb states and proves all the main results needed in the derivation and proof of the degree bound presented in Theorem 5.1.3, closely following the overall proof outline according to Section 5.1.3 and incorporating the improvements listed in Section 5.1.4. Some results are proved in arbitrary dimension, without any conditions imposed on $\overline{\mathrm{k}}$ (apart from being a natural number, of course), others are constrained by $\overline{\mathrm{k}} = 2$.

We now list some interesting auxiliary lemmas that are stated and proved in our formalization but have not been mentioned yet. In each of the formulas below, the universally quantified variable $A$ ranges over all non-empty exponent vectors. This fact is not made explicit in the formulas for the sake of brevity.

**General bound of $\mathtt{M}[A \curvearrowleft x]$.** The following formula is essential for our formal proof of Theorem 5.1.6. It is valid in all dimensions:

$$\underset{\mathtt{isExponentVector}[x]}{\forall} \quad \mathtt{M}[A \curvearrowleft x] \underset{\overline{\mathcal{D}}}{\leq} \underset{\overline{\mathcal{D}}}{\mathrm{max}}[\langle \mathtt{M}[A], \mathtt{M}[x, A], \mathrm{deg}[x] \rangle] \qquad (M\curvearrowleft)$$

**Bounding $\mathtt{M}[A \curvearrowleft x] + x_k$ if $x$ is in far exterior.** The following formula is needed for proving (5.4). Apparently, it only holds in the two-dimensional case:

$$\overline{\mathrm{k}} = 2 \Rightarrow \underset{i=1,\ldots,2}{\forall} \; \underset{\mathtt{ixExpVector}[x]}{\forall} \; \mathrm{deg}[x] \underset{\overline{\mathcal{D}}}{\leq} \mathtt{M}[A] \wedge \mathtt{inFarExterior}[x, A, i] \Rightarrow \mathtt{M}[A \curvearrowleft x] \underset{\overline{\mathcal{D}}}{+} x_i \underset{\overline{\mathcal{D}}}{\leq} \mathtt{M}[A] \underset{\overline{\mathcal{D}}}{+} \mathtt{K}[A]_i \qquad (M\curvearrowleft \textit{ far exterior})$$

**Final result.** The final result of our formalization is

$$\overline{\mathrm{k}} = 2 \;\Rightarrow$$

$$\underset{\mathtt{isExpVectorTuple}[F,G]}{\forall}$$

$$1 \underset{\mathbb{Z}}{\leq} |F| \wedge 1 \underset{\mathbb{Z}}{\leq} |G| \wedge \underset{\overline{\mathcal{D}}}{0} \underset{\overline{\mathcal{D}}}{\leq} \mathtt{W}[G] \wedge \mathtt{M}[G] \underset{\overline{\mathcal{D}}}{+} \mathtt{W}[G] \underset{\overline{\mathcal{D}}}{\leq} \mathtt{M}[F] \underset{\overline{\mathcal{D}}}{+} \mathtt{W}[F] \;\Rightarrow$$

$$\mathtt{maxdeg}[G] \underset{\overline{\mathcal{D}}}{\leq} \mathtt{maxdeg}[F] \underset{\overline{\mathcal{D}}}{+} \mathtt{maxdeg}[F] \qquad (\textit{main theorem})$$

Apparently, Formula (*main theorem*) is not exactly what Theorem 5.1.3 states; in particular, no appeal to Algorithm 2 is made. It should be clear, however, that the output $G$ of Algorithm 2 on non-empty input $F$ satisfies all the requirements of (*main theorem*), at least if $\overline{\mathcal{D}}$ is restricted to $\mathbb{N}_0$: it is non-empty, $\mathtt{W}[G]$ is an element of $\overline{\mathcal{D}} = \mathbb{N}_0$ and hence non-negative, and $\mathtt{M}[G] \underset{\overline{\mathcal{D}}}{+} \mathtt{W}[G]$ being bounded from above by $\mathtt{M}[F] \underset{\overline{\mathcal{D}}}{+} \mathtt{W}[F]$ can be inferred by an inductive argument: whenever the current basis $G$ is updated in Line 12 of Algorithm 2 by adding a new element, the quantity $\mathtt{M}[G] \underset{\overline{\mathcal{D}}}{+} \mathtt{W}[G]$ does not increase thanks to the inductive assertion (5.4).

## 5.3 ComplexityProver

Just as in the formal development of reduction ring theory, we also designed and implemented a special Theorema prover, called ComplexityProver, for the verification of the complexity analysis described in the previous sections. Clearly, the main intentions and ideas behind the ComplexityProver are absolutely the same as for the ReductionRingProver: knowledge about basic mathematical concepts is “lifted” to the inference level in order to effectively and efficiently reason about them; no appeal to notions directly related to the complexity analysis, like the chain criterion, is made.

Nonetheless, there are some differences between the two special provers, too. For one thing, the ComplexityProver does not employ any of the general predicate logic rules from the RewriteInteractiveProver but instead implements such rules itself, the reason for this being the same as the reason for not making use of the elementary theories: the ComplexityProver was implemented long before the RewriteInteractiveProver. However, since the rules themselves are almost identical anyway, and in fact quite some ideas originally developed for the ComplexityProver were later transferred to the RewriteInteractiveProver and are thus discussed in Section 6.2, we only sketch the *special* rules in this section.

A second aspect the two provers differ in is the number of special rules: in total, the ComplexityProver consists of 29 special rules, compared to only eleven of the ReductionRingProver. The reason for this is twofold: first, using an automatic rather than an interactive proving strategy necessitated inference rules applicable only in quite specific proof situations, in order not to needlessly blow up the search space, and second, each identity involving binders (like logical quantifiers and tuple abstractions) had to be phrased as a separate special inference rule, because the higher-order rewriting mechanism (see Section 6.1) that would have enabled stating it as an object-level formula and then applying it by general-purpose rewriting techniques has not been available at that time yet. In contrast, the formal verification of reduction ring theory heavily relies on higher-order rewriting, keeping the number of special rules of the ReductionRingProver comparatively small.

We now briefly sketch the five categories of special rules of the ComplexityProver.

### 5.3.1 Integers

Two rules take care of unfolding the definition of integer intervals, one for formulas in the knowledge base of a proof situation and one for its goal. In more concrete terms, propositions of the form $\underset{\mathbb{Z}_{a,\ldots,b}}{\in}[x]$, where $a \neq -\infty$ or $b \neq \infty$, are simply rewritten into the conjunction $\underset{\mathbb{Z}}{\in}[x] \wedge a \underset{\mathbb{Z}}{\leq} x \wedge x \underset{\mathbb{Z}}{\leq} b$. Although the same effect could also easily be achieved by stating the definition of integer intervals as an ordinary (first-order) formula and applying it as an equational rewrite-rule by the standard first-order rewriting-mechanism of Theorema, we decided to include the two special rules for the following reason: usually, unfolding definitions in a proof should somehow be the last resort, when nothing else can be done. In our setting, though, we quickly realized that integer intervals deviate from this rule of thumb, in that for an efficient automatic proof generation the definition of integer intervals shall ideally be unfolded instantly, hence requiring separate rules independent of the default ones for unfolding definitions.

### 5.3.2 Tuples

This category consists of eleven rules incorporating knowledge about tuples and operations on tuples. As for the two integer rules, some of the tuple rules could in principle be simulated by object-level formulas in conjunction with first-order rewriting as well; others, however, deal with tuple abstractions ("`TupleOf`") and hence would require higher-order rewriting when stated as mere formulas. Examples of the latter are listed below. To that end, let $T$ abbreviate $\langle t(i) \mathop{|}\limits_{i=a,\ldots,b} \varphi(i)\rangle$, where $t(i)$ and $\varphi(i)$ indicate that the term $t$ and the formula $\varphi$ both may depend on the bound variable $i$.

$$\frac{K \vdash \texttt{isTuple}[T] \qquad K \vdash \mathop{\exists}\limits_{i=a,\ldots,b} \varphi(i)}{K \vdash 1 \mathop{\leq}\limits_{\mathbb{Z}} |T|} \text{ (nonemptyTupleOf)}$$

$$\frac{K \vdash \mathop{\in}\limits_{\mathbb{Z}}[a] \qquad K \vdash \mathop{\in}\limits_{\mathbb{Z}}[b]}{K \vdash \texttt{isTuple}[T]} \text{ (isTupleTupleOf)}$$

$$\frac{K \vdash \texttt{isTuple}[T] \qquad K \vdash \mathop{\exists}\limits_{i=a,\ldots,b} \varphi(i) \wedge \psi(t(i))}{K \vdash \mathop{\exists}\limits_{j=1,\ldots,|T|} \psi(T_j)} \text{ (existsTupleOfGoal)}$$

$$\frac{K \vdash \texttt{isTuple}[A] \quad K, \texttt{isTuple}[A], \mathop{\forall}\limits_{i=1,\ldots,|A|} \psi(A_i), \psi(x) \vdash \Gamma}{K, \mathop{\forall}\limits_{j=1,\ldots,|A \curvearrowleft x|} \psi((A \curvearrowleft x)_j) \vdash \Gamma} \text{ (forallKBAppend)}$$

*Remark* 18. Note that below the line of the inference in (existsTupleOfGoal) and (forallKBAppend) it is of utmost importance that the bound variable $j$ must *only* occur as the subscript of $T$ and $A \curvearrowleft x$, respectively, in the formula $\psi$.

### 5.3.3 Addition

Two special inference rules take care of the various algebraic properties (associativity, cancellativity, etc.) of the monoid operation $+$ in totally-ordered commutative monoids, one for formulas in the knowledge base of a proof situation and one for its goal. More precisely: if $\texttt{isTOM}[\mathcal{M}]$ is known for some domain $\mathcal{M}$ and a proposition of the form $x_1 \mathop{+}\limits_{\mathcal{M}} x_2 \mathop{+}\limits_{\mathcal{M}} \ldots \mathop{+}\limits_{\mathcal{M}} x_m \sim y_1 \mathop{+}\limits_{\mathcal{M}} y_2 \mathop{+}\limits_{\mathcal{M}} \ldots \mathop{+}\limits_{\mathcal{M}} y_n$ appears in the current proof situation, where $\sim$ is one of $=$, $\mathop{\leq}\limits_{\mathcal{M}}$ or $\mathop{<}\limits_{\mathcal{M}}$ and the two sides of the relation might be grouped arbitrarily, all common terms appearing on either of the two sides are canceled. Of course, first the membership of all the $x_i$ and $y_i$ in the domain $\mathcal{M}$ has to be checked.

### 5.3.4 Order Relations

This category consists of seven rules handling the order relations $\mathop{\leq}\limits_{\mathcal{M}}$ and $\mathop{<}\limits_{\mathcal{M}}$ in totally-ordered commutative monoids $\mathcal{M}$, exploiting reflexivity, totality and transitivity, as well as monotonicity of $\mathop{+}\limits_{\mathcal{M}}$. All this happens in the same spirit as in the ordering rules of the ReductionRingProver, so we omit the details here; only note that in order for the rules to be applicable it suffices if `isTOM[`$\mathcal{M}$`]` is known instead of `isTotalIrreflOrder[` $<, \mathcal{M}$`]` etc.

## 5.3.5 Minimum and Maximum

The seven remaining rules are concerned with the minimum and maximum of tuples of elements in some totally-ordered commutative monoid $\mathcal{M}$. Two of them, however, in fact give semantics to the `argmin` and `argmax` quantifiers: for instance, every occurrence of $\underset{\substack{x=a,\ldots,b \\ \varphi(x)}}{\mathtt{argmin}_{\mathcal{M}}} t(x)$ in a proof situation is replaced by a fresh arbitrary but fixed constant $\overline{x}$ satisfying $\underset{y=a,\ldots,b}{\forall} \left( \varphi(y) \Rightarrow t(\overline{x}) \underset{\mathcal{M}}{\leq} t(y) \right)$, at least provided that the range of the `argmin`-binder is non-empty and finite, and all the $t(x)$ are elements of $\mathcal{M}$.

The other rules are all derived from simple equalities and equivalences involving $\underset{\mathcal{M}}{\mathtt{min}}$, $\underset{\mathcal{M}}{\mathtt{max}}$ and the order relations $\underset{\mathcal{M}}{\leq}$ and $\underset{\mathcal{M}}{<}$, for instance

$$\frac{K \vdash \underset{\mathtt{DomainTuples}[\mathcal{M}]}{\in} [S] \qquad K \vdash \underset{i=1,\ldots,|S|}{\exists} S_i \prec x}{K \vdash \underset{\mathcal{M}}{\mathtt{min}}[S] \prec x} \text{ (minGoalLeft)}$$

and

$$\frac{K \vdash \underset{\mathtt{DomainTuples}[\mathcal{M}]}{\in} [S] \qquad K, \underset{i=1,\ldots,|S|}{\forall} S_i \prec x \vdash \Gamma}{K, \underset{\mathcal{M}}{\mathtt{max}}[S] \prec x \vdash \Gamma} \text{ (maxKBLeft)}$$

where $\prec$ is one of $\underset{\mathcal{M}}{\leq}$ or $\underset{\mathcal{M}}{<}$ and $K$ contains `isTOM[`$\mathcal{M}$`]`.

# Chapter 6

# Contributions to Theorema

This chapter contains the descriptions of four Theorema tools developed in the frame of our formalization of reduction ring theory. Still, all of them are *completely independent* of both the formalization and the ReductionRingProver, and hence bear the prospect of aiding explorations of all kinds of mathematical theories in Theorema in the future. In fact, the four tools are essentially independent of each other as well, although the higher-order rewriting mechanism, the RewriteInteractiveProver and the interactive proof strategy were designed in such a way that they fit together perfectly.

Each of the four tools is implemented in the *Mathematica* programming language and distributed as a *Mathematica* package, meaning that it can easily be loaded into a Theorema session when needed. Since the tools were, of course, heavily made use of in the formal treatment of reduction ring theory and proved extremely helpful in many situations, they are intended to be integrated into the official version of Theorema in the future—at present, they are still stand-alone packages.

Parts of this chapter are also contained in [Mal16a].

## 6.1 Higher-Order Rewriting

The driving engine behind computations in *Mathematica* and Theorema alike is *rewriting*: expressions are transformed (ideally to "simpler" expressions) by repeatedly replacing sub-expressions according to a set of rewrite rules. For instance, Theorema itself is basically implemented as a large collection of *Mathematica* rewrite rules.

Within Theorema, rewriting plays an essential role not only in computations, but also in proofs [BDJ$^+$00]. To that end, formulas appearing among the assumptions of a proof situation are translated into one or more rewrite rules, rep-

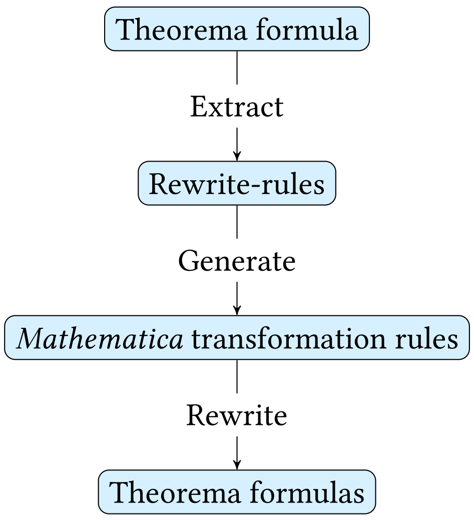


Figure 6.1: Overview of rewriting in Theorema.

resented as plain *Mathematica* transformation rules (`RuleDelayed`, `:>`), which may then be employed for equational-, forward- and backward rewriting of other formulas, see Figure 6.1 for an overview of this process and [Win13] for details. This approach, pursued also in other proof assistants, enables efficient and "natural-style" proving and as such constitutes an integral component of the Theorema system.

Although the whole mechanism of translating assumptions into rewrite rules and then applying these rules already works quite well, it is constrained by one major limitation: *Mathematica* only supports *syntactic* pattern matching and hence *first-order* rewriting; variables are not automatically instantiated by functions, e. g. $\lambda$-abstractions, or, in *Mathematica* parlance, by `Function[...]`-expressions during the matching phase, just in order to match the given expression modulo $\alpha\beta\eta$-equivalence. To illustrate this, consider the following illuminating example of an equivalence in set theory:

$$\underset{P,\,A,\,a}{\forall}\quad a \in \{\underset{x\in A}{x \mid} P[x]\} \Leftrightarrow (a \in A \wedge P[a]) \qquad \text{(\textit{membership abstraction term II})}$$

where $P$ is apparently a higher-order variable, since it is applied to arguments. Formula (*membership abstraction term II*), if used as a rewrite rule from left to right, can be applied to, say, $\{\underset{x\in\mathbb{Z}}{x \mid} x^2 < 90\}$ if higher-order rewriting is employed (with $P$ instantiated by $\underset{y}{\lambda}\, y^2 < 90$), but not in connection with first-order rewriting and hence purely syntactic matching.

The limitations of first-order rewriting, unfortunately, turned out to be quite inconvenient for computer-assisted theory exploration, as lots of formulas one typically encounters in mathematics are actually higher-order: the most well-known examples are perhaps induction rules (natural induction, well-founded induction, etc.), but also the two higher-order formulas subsuming the axiom schemas of replacement and separation in Zermelo-Fraenkel set theory are worth being mentioned. Since all of these examples, together with many other higher-order formulas, feature a prominent role in our formalization of reduction ring theory, we decided to enhance the default first-order rewriting mechanism of Theorema and also implement a higher-order rewriting mechanism for efficiently handling higher-order formulas in proofs. This mechanism, basically consisting of a *compiler* for translating (first- or higher-order) rewrite-rules into *Mathematica* transformation rules (corresponding to the second step in Figure 6.1), is described in detail in the remainder of this section.

Before, however, we address some questions of theoretical nature arising in higher-order logic, in particular in higher-order matching (as this is the most difficult and thus interesting aspect of our rewriting mechanism). Already back in 1982, Goldfarb [Gol82] proved that higher-order unification, and even only second-order unification, is undecidable in general. Higher-order matching in a simply-typed environment is known to be decidable for orders two [Hue76], three [Dow93] and four [Pad96], and under certain conditions on the types involved was shown to be decidable a couple of years ago for arbitrary order [Sti09]—in the most general setting, however, no definite decidability results are known. The complexity of second-, third- and fourth-order matching is discussed in [Wie99]; in any case, it is at least NP-hard in general.

All this indicates that we had to make concessions when implementing the rewrite-rule compiler, i. e. especially the part responsible for matching. As can be seen below, the resulting algorithm is *highly incomplete* in the precise sense that the *Mathematica* transformation rules generated do not fully reflect the higher-order nature of the rewrite-rules they originate from, meaning that in practice they might not be applicable to expressions the corresponding rewrite-rules would in fact be applicable to in theory. Furthermore, one must note that the current implementation of our mechanism is still quite preliminary: although it has already been extensively tested, no formal or informal proofs of its correctness (or completeness) have been carried out yet. A detailed analysis of the theoretical foundations of, and ideas behind, our higher-order rewriting mechanism in the future would certainly put the implementation on more solid grounds.

However, one might also argue that establishing the correctness of the implementation is strictly speaking not necessary: whenever an automatically generated *Mathematica* transformation rule $r$ is used to rewrite a concrete expression $e$ into $e'$, it is easily possible to check whether the rewrite is justified by the formula $\varphi$ the rule $r$ originates from: simply instantiate $\varphi$ by the substitution that matches the left-hand-side of $r$ with $e$ and then check whether $e$ is indeed identical to the part of $\varphi$ corresponding to the left-hand-side of $r$ and $e'$ to the part corresponding to its right-hand-side. In that sense, applying transformation rules only yields *candidate* rewrites that still have to pass (automatic) "security checks". Such checks can easily be implemented, are very cheap from the efficiency point of view, and are in fact work in progress.

*Remark* 19. Although Theorema does not support higher-order rewriting by default, this does *not* mean that higher-order formulas cannot be dealt with at all, but it only means that purely syntactic pattern matching, without any kind of $\alpha\beta\eta$-equivalence, is employed, significantly restricting the set of rewritable expressions.

### 6.1.1 Notation and Terminology

In the sequel we assume that we are given a rewrite rule originating from a Theorema formula; we are not interested in how the rule was extracted from the formula and whether it corresponds to an equality, and equivalence or an implication.

Throughout this section, rewrite rules are denoted as $r : L \mapsto R$, where $r$ serves as an (optional) identifier, $L$ is the left-hand-side of the rule, and $R$ is its right-hand-side. Both $L$ and $R$ may contain free variables, originally bound by universal quantifiers in the corresponding formula that have already been stripped away; of course, as always in this context, a general constraint imposed on rewrite rules is that every free variable occurring in $R$ must also occur in $L$.

*Remark* 20. In practice, the applicability of a rewrite rule $r$ may be constrained by an additional condition $\varphi$. In this section, however, we ignore this fact entirely, as such side-conditions do not cause any specific difficulties when turning higher-order rules into *Mathematica* transformation rules at all. How to efficiently handle side-conditions when applying rewrite rules, and whether the resulting rewrite-systems are terminating and/or confluent, are interesting questions in their own right, but not discussed here.

An important question concerns the distinction between first- and higher-order variables in a rule $r$: recall that Theorema is untyped, meaning that expressions, and in particular variables, do not have any type information attached to them that would allow characterizing higher-order variables as variables of some function type. Instead, in the present framework we simply regard a variable $F$ higher-order iff at least one of its occurrences in the given rule, no matter whether in $L$ or $R$, is the head of a sub-expression, i. e. an expression of the form $F\,[\,x_1, \ldots, x_n\,]$; all other variables are regarded first-order. In the context of rewriting, however, it is important to note that for bound variables it is absolutely irrelevant whether they are first- or higher-order: the matching algorithm treats bound variables always in exactly the same way, as they can only be renamed to other bound variables anyway, for $\alpha$-equivalence. Furthermore, it must be noted that in an untyped language $\beta$-reduction is not strictly normalizing; how this problem is dealt with in our setting is explained a bit later, in Section 6.1.4.

In the sequel we adhere to the following naming conventions of variables in rewrite rules: free higher-order variables have upper-case names and are printed in italics, free first-order variables, as well as all bound variables, have lower-case names and are printed in italics, too, and constants are printed in typewriter-font, as usual.

**Example 3.** In the rule $a \in \{x \underset{x \in b}{\mid} P\,[\,x\,]\} \mapsto a \in b \wedge P\,[\,a\,]$, $a$ and $b$ are free first-order variables, $P$ is a free higher-order variable, and $x$ is a bound (first-order) variable. ■

We assume familiarity with the main concepts of $\lambda$-calculus, in particular with $\alpha\eta$-equivalence, denoted by $\simeq$, and $\beta$-reduction, denoted by $\rightarrow_\beta$. Two expressions $e_1$ and $e_2$ are said to be $\alpha\beta\eta$-equivalent to each other iff there exist expressions $e_1'$ and $e_2'$ such that $e_1 \rightarrow_\beta^* e_1'$, $e_2 \rightarrow_\beta^* e_2'$ and $e_1' \simeq e_2'$.

The rewrite-rule compiler, described in the next subsection, partitions the free higher-order variables of a rewrite-rule $r$ into two sets. Variables belonging to the first set, although being higher-order, are nevertheless treated just like ordinary first-order variables: they have to be matched syntactically (or, more precisely, only modulo $\alpha$- but not $\beta$-equivalence) by the concrete expressions to be rewritten; these variables are called *rigid.* On the other hand, free higher-order variables belonging to the second set are really treated as higher-order variables when applying a rule $r$, i. e. they are automatically tried to be instantiated by suitable $\lambda$-terms that finally give rise to a match modulo $\alpha\beta$-equivalence; these variables are referred to as *flexible.* Whether a variable is rigid or flexible in $r$ is no intrinsic property of $r$ but solely determined and taken into account by the compiler, for reasons explained below.

### 6.1.2 The Rewrite-Rule Compiler: Overview

The *rewrite-rule compiler* constitutes the core component of the higher-order rewriting mechanism. It takes a (not necessarily higher-order) rewrite-rule $r$

as input and translates it into an ordinary *Mathematica* transformation rule, i. e. `RuleDelayed`-expression, that can then be applied to expressions by the standard *Mathematica* rule application functions, e. g. `Replace`, `ReplaceList`, etc., relying on the standard *Mathematica* pattern-matching and instantiation facilities. Since *Mathematica* by default only supports first-order, syntactic pattern-matching, all the functionality needed for higher-order matching, like $\alpha\beta\eta$-equivalence, explicitly has to be “attached” to the generated transformation rules, thus simulating higher-order- by first-order rewriting.

**Example 4.** Consider the higher-order rewrite-rule

$$\left(\sum_{i=1,\ldots,n+1} F[i]\right) - \left(\sum_{i=1,\ldots,n} F[i]\right) \mapsto F[n+1]$$

Ignoring syntactical details of the internal representation of Theorema expressions and subtleties related to capture-avoiding substitutions, and not taking into account any code optimizations performed by default either, the *Mathematica* transformation rule returned by the compiler reads as something like

```
Sum[{i1_, 1, n_+1}, F1_] - Sum[{i2_, 1, n_}, F2_] :>
  substFree[ F1, {i1 -> n + 1}] /;
     alphaEquiv[ substFree[ F1, {i1 -> i2}], F2]
```

As can be seen, $F$ is actually never instantiated by a $\lambda$-term, but rather the instance of the right-hand-side of the rule is constructed by directly substituting for the bound variable $i$ in the instance of the first sum-term. ■

One of the main design goals in the development of the whole rewriting mechanism was to pre-compute as much as possible already when generating the *Mathematica* transformation rules at compile-time, in order to increase the efficiency of applying them at run-time. This lead to quite a lot of performance optimizations, e. g. in connection with the names of bound variables and substitutions, that are presented in Section 6.1.5. The price of a slightly more involved, and hence less efficient, compiler is definitely worth the reward of being able to apply the resulting rules efficiently later on.

## 6.1.3 Pre-Processing

Consider a rewrite-rule $r : L \mapsto R$. In a very first pre-processing step, the compiler somehow “normalizes” the rule by introducing a unique name for each bound variable and by ensuring that every occurrence of any free higher-order variable in $r$ is applied to the same number of arguments. If this is not the case, it can always be achieved by first determining the *maximum* number of arguments for each free higher-order variable and then adding further variables bound by new $\lambda$-abstractions at the end of all argument-lists with fewer elements. Here it is important to note that *curried* expressions whose head is a free variable are un-curried first.

**Example 5.** Pre-processing the (nonsense) rule $F[1,2] + F[a] \mapsto F$ yields the $\eta$-equivalent rule $F[1,2] + \underset{x}{\lambda} F[a,x] \mapsto \underset{x,y}{\lambda} F[x,y]$. The same result is obtained when pre-processing $F[1][2] + F[a] \mapsto F$, because the curried term $F[1][2]$ is un-curried to $F[1,2]$. ■

The pre-processing described above has two important exceptions:

- No new $\lambda$-abstractions are added for free higher-order variables never applied to arguments in $L$ (but only in $R$). Such variables are furthermore rendered rigid instantly (recall the definition of a rigid higher-order variable as a variable that is treated just like a first-order variable by the compiler). Actually, treating such variables like first-order variables does not do any harm: $F$, say, matches *any* expression, whereas $\underset{x}{\lambda} F[x]$ only matches $\lambda$-expressions, limiting the applicability of the corresponding rule, because usually expressions are not $\eta$-expanded before they are rewritten.

- No new $\lambda$-abstractions are added for free higher-order variables applied to *free* sequence variables somewhere in $r$ either, because the "arity" of such variables cannot be determined at compile-time; they are rendered rigid, too.

**Example 6.** Pre-processing the rule

$$\langle A[1], B, C[a], b... \rangle \mapsto \{A[a, C[0, b...]], B[a]\}$$

yields

$$\langle \underset{x}{\lambda} A[1,x], B, C[a], b... \rangle \mapsto \{A[a, C[0, b...], B[a]]\}$$

where $A$ is still flexible, but $B$ (never applied to arguments on the left-hand-side) and $C$ (applied to the free sequence variable $b...$) are rigid. ■

### 6.1.4 Generating *Mathematica* Transformation Rules

We now have a closer look at the act of transforming a given, pre-processed rewrite rule $r : L \mapsto R$ into a *Mathematica* transformation rule $m$ : p :> b, where p stands for "pattern" and b for "body", satisfying the *correctness property*

$$e' \in \texttt{ReplaceList}[e, m] \;\Rightarrow\; e \twoheadrightarrow_r e' \tag{6.1}$$

for all well-formed *Mathematica* expressions $e$ and $e'$ (where $e \twoheadrightarrow_r e'$ means that $e$ can be rewritten to $e'$ by $r$ modulo $\alpha\beta\eta$-equivalence). The other direction of (6.1), though desirable in principle, is out of reach because of the decidability-issues related to higher-order matching mentioned above.

The pattern `p` in $m$ is an ordinary *Mathematica* pattern expression, built from constants and syntactic variables. The higher-order aspect of matching, including testing $\alpha$-equivalence of expressions and performing substitutions of free variables, is actually entirely incorporated in the body `b` by means of the `Condition` construct (infix notation `/;`) from *Mathematica*, as can be seen also in Example 4.

The same example in fact reveals one of the crucial design principles of our rewrite-rule compiler: it does not implement only one single, general higher-order matching algorithm that is called whenever a rule, like $m$, shall be applied, but rather it equips each rule with its specific, tailor-made, highly optimized matching algorithm. This specific matching algorithm is constructed automatically by the compiler while generating a transformation rule, say, $m$ from $r$, and the aforementioned optimizations are achieved by thoroughly analyzing the structure of $r$ and then hard-coding knowledge thus obtained in the algorithm, eliminating the necessity of computing auxiliary results again and again and trying possible alternatives during the matching phase that are already known to lead to failure. In short: many results typically computed by a general higher-order matching algorithm are instead pre-computed at compile-time *if they are really needed* in the concrete case; otherwise the specific algorithm is modified in such a way that it does not compute them at all. In Example 4, for instance, it suffices to check whether the instances of two syntactically matched variables `F1` and `F2` are $\alpha$-equivalent to each other, up to a substitution of bound variables—and this fact is detected fully automatically by the compiler.

It is well-known that higher-order matching is in general not *unitary*, i. e. more than one (most general) matchers might exist for two given expressions (one of them being closed, of course). The rewrite-rule compiler, however, can basically only deal with rules giving rise to a (essentially) unitary matching problem, for a couple of subtle reasons related to implementation technicalities (for instance, a backtracking strategy would have to be employed otherwise). Although this is a severe limitation in theory, the experience we gained from the formal treatment of reduction ring theory revealed that our rewriting mechanism is still sufficiently powerful in practice, as the vast majority of higher-order formulas contained in the formalization yield rewrite-rules with unitary matching problems associated to them. The claim that considering unitary matching suffices is also supported by the fact that automatic simplification modulo higher-order rules in the widely-used Isabelle proof assistant is only possible if the left-hand-sides of these rules are so-called *higher-order patterns* (see [Nip93] and [Wen16], pp. 205–206), which is even more restrictive than our approach.

However, not only must the matching problem associated to rewrite-rule $r$ be unitary, but the compiler must even be able to *detect* this fact such that it can be exploited when constructing the transformation rule $m$. To that end, *bound variables* (first- or higher-order) occurring in the formal arguments of free, flexible higher-order variables in $L$ are used to uniquely determine the instances of these higher-order variables when matched against a concrete expression. How this proceeds in practice is best illustrated in a couple of examples.

**Example 7.** Consider the rewrite-rule

$$\sum_{i=0,\dots,n-1} F[n-i] \mapsto \sum_{i=1,\dots,n} F[i]$$

The matching problem associated to the rule is evidently unitary because of the occurrence of the bound variable $i$ in the argument of $F$ on the left-hand-side:

- $\sum_{i=0,\dots,4-1} (4-i)^2$ matches the left-hand-side of the rule, instantiating $F$ by $\lambda_x\, x^2$.
- $\sum_{i=0,\dots,4-1} ((4-i)^2+i)$ does not match, as the second occurrence of $i$ in the sum-term is not in a sub-expression of the form $4-i$.
- $\sum_{i=0,\dots,4-1} ((4-2)^2+1)$ matches again, instantiating $F$ by the constant function $\lambda_x\, ((4-2)^2+1)$.

■

Clearly, the bound variables used to determine the instances of free, flexible higher-order variables must be bound *outside* their argument-lists: the $x$ in $F[\{x^2 \mid_{x\in a}\}] \mapsto F[\emptyset]$ cannot be used to determine the instance of $F$. Moreover, these bound variables must not only occur in the formal arguments of the respective higher-order variables, but there also *outside* the argument-list of any other higher-order variable:

**Example 8.** The matching-problem associated to the left-hand-side of

$$\sum_{i=1,\dots,n} F[G[i]] \mapsto F[0]$$

is not unitary, since the bound variable $i$, although appearing in the formal arguments of $F$, is somehow “consumed” by $G$ already. The simple expression

$\sum\limits_{i=1,\ldots,10} i^2$ can be matched in (at least) two ways, namely if one of $F$ and $G$ is instantiated by the identity function and the other by $\underset{x}{\lambda}\, x^2$.

A minor modification of the left-hand-side of the rule renders the matching problem unitary, though:

$$\sum_{i=1,\ldots,n} (F[G[i]] + F[i]) \mapsto F[0]$$

Here, the sub-expression $F[i]$ can be used to uniquely determine the instance of $F$, and once knowing this, $F[G[i]]$ uniquely determines the instance of $G$ *unless $F$ is instantiated by a constant function.* ■

The second rewrite-rule in the previous example exhibits an interesting phenomenon: although in principle the instances of both $F$ and $G$ can be uniquely determined, in the special case where $F$ has to be instantiated by a constant function determining the instance of $G$ is not possible any more; in fact, $G$ may well be instantiated by *anything*. Since all this clearly depends on the concrete expression the left-hand-side of the rule is matched against and hence cannot be foreseen by the compiler, the following general strategy is pursued in such situations:

1. When generating the *Mathematica* transformation rule, simply assume that $F$ is *not* instantiated by a constant function, meaning that the instance of $G$ can be determined.

2. If, however, $F$ *does* happen to be instantiated by a constant function when matching a concrete expression $e$ and hence the instance of $G$ cannot be determined, distinguish two cases:

   (a) If the instance of $G$ must be known for constructing the instance of the right-hand-side of the rule, return failure, i. e. the rule cannot be applied.

   (b) If the instance of $G$ does not need to be known for constructing the instance of the right-hand-side, simply proceed; after all, it is definitely possible to instantiate $G$ *somehow* in order to match $e$. Note that this case may even occur in situations where $G$ appears on the right-hand-side of the rewrite-rule.

**Example 9.** Consider the following three rewrite-rules, all having identical left-hand-sides:

$$\begin{aligned}\sum_{i=1,\ldots,n}(F[G[i]]+F[i]) &\mapsto G[0]\\ \sum_{i=1,\ldots,n}(F[G[i]]+F[i]) &\mapsto F[0]\\ \sum_{i=1,\ldots,n}(F[G[i]]+F[i]) &\mapsto F[G[0]]\end{aligned}$$

If $F$ is instantiated by a constant function when trying to apply the first rule, the application fails: the precise instance of $G$ would have to be known for constructing $G[0]$. In the second and third rule, however, $F$ being instantiated by a constant function does not do any harm: in the second rule, $G$ does not appear on the right-hand-side at all, and in the third rule, the instance of $G$ must only be known for constructing $F[G[0]]$ if $F$ is non-constant. ■

Summarizing: the applicability of the transformation rule $m$ resulting from a rewrite-rule $r : L \mapsto R$ to a concrete expression $e$ not only depends on $L$, but also on $R$.

In the examples above, all higher-order variables are unary; unary variables can be dealt with quite easily. General $n$-ary higher-order variables complicate matters considerably, since in such a situation the potential *unifiability* of their formal arguments must be taken into account.

**Example 10.** The matching problem associated to the (nonsense) rewrite-rule

$$\underset{x}{\forall} P[x^2, x] \mapsto P[1,2]$$

is not unitary, because the expression $\underset{x}{\forall} x^2 < 1$ is matched by instantiating $P$ either by $\underset{x,y}{\lambda}\, x < 1$ or by $\underset{x,y}{\lambda}\, y^2 < 1$. The reason for this lies in the unifiability of the second formal argument of $P$ on the left-hand-side, $x$, with a sub-expression of its first argument, $x^2$. ■

**Example 11.** The matching problem associated to

$$\underset{x}{\forall}(P[x^2, a] \wedge P[x^2, x]) \mapsto P[a, 2]$$

is unitary, because given a concrete expression $e$,

1. $x^2$ may be used to uniquely determine the list $p_1$ of positions of the first formal argument of $P$ in the sub-expression $e_1$ of $e$ being matched against $P[x^2, a]$.

2. Let $e_2$ be the sub-expression of $e$ matched against $P[x^2, x]$. Since the positions of the first formal argument of $P$ in both $e_1$ and $e_2$ must be identical, and these positions are known already, one must now
   (a) ensure that the sub-expressions of $e_2$ at these positions are all $x^2$, and
   (b) determine the list $p_2$ of positions of the second formal argument of $P$ in $e_2$, making use of $x$; occurrences of $x$ in $e_2$ at a sub-position of an element in $p_1$ are known to actually belong to the first argument of $P$ and hence must be discarded.
3. Finally, knowing $p_2$, the sub-expressions of $e_1$ at these positions must be checked to be $\alpha$-equivalent to each other; if they are, they constitute the instance of $a$ (unless the list $p_2$ is empty, which is possible in principle, leading to a similar situation as in Example 8).

The compiler detects and exploits these non-trivial dependencies and relations between the two occurrences of $P$ on the left-hand-side of the rule fully automatically and generates efficient code that exactly follows the three steps sketched above. ■

We already pointed out that higher-order unification is an extremely hard problem in general, meaning that the unification algorithm we use to find out whether two formal arguments are unifiable or not *cannot* be complete. The algorithm, hence, was designed to "stay on the safe side", i. e. to rather return false-positive results (two non-unifiable expressions are claimed to be unifiable) than false-negative ones (two unifiable expressions are claimed to be non-unifiable): in the worst case, a free higher-order variable is wrongly made rigid (see below), but at least the resulting transformation rule is correct in the sense that it satisfies (6.1).

The various examples presented above demonstrate that automatically detecting as many unitary matching problems as possible and generating efficient *Mathematica* code for matching the left-hand-side and instantiating the right-hand-side of a given rewrite-rule can be a difficult, time-consuming task where lots of subtle details have to be taken into account, e. g. that even problems detected as unitary may actually give rise to infinitely many instantiations of free variables (Examples 8 and 11). Still, the compiler shall also be able to deal with rules whose associated matching problems are evidently non-unitary: in such a situation, those free, flexible higher-order variables causing the problems are simply made rigid. Rigid variables are treated just like first-order variables, thus eliminating any problems with non-unitarity.

*Remark* 21. In some cases it might not be clear which of the free higher-order variables cause problems, as in the first rule in Example 8. Then, the compiler randomly chooses *any* of the possible candidates for making them rigid; users of the compiler should not rely on a particular choice.

Due to the all-important role of bound variables on the left-hand-sides of rewrite-rules, there is a subtle issue related to the extraction of rewrite-rules from Theorema formulas even *before* these rules are then translated in *Mathematica* transformation rules by the compiler. Typically, rewrite-rules are extracted by stripping away universal quantifiers and then interpreting the remaining equalities/equivalences/implications as (conditional) rewrite rules; however, stripping away universal quantifiers entails the danger of stripping away "too many" of them. Consider, for instance, the ordinary induction principle on ℕ:

$$\underset{P}{\forall}$$

$$P[0] \wedge \underset{n\in\mathbb{N}}{\forall}(P[n] \Rightarrow P[n+1]) \;\Rightarrow\; \underset{n\in\mathbb{N}}{\forall} P[n] \qquad (\textit{induction})$$

Induction rules are usually used for backward rewriting, i. e. the resulting rewrite-rule would read as

$$P[n] \mapsto P[0] \wedge \underset{n\in\mathbb{N}}{\forall}(P[n] \Rightarrow P[n+1]) \tag{6.2}$$

where the $n$ on the left-hand-side is free—and this is precisely the problem, as $P$ has to be rendered rigid now (no bound variable appears in its arguments). Hence, it would be better to keep the universal quantifier binding the $n$ on the left-hand-side, leading to

$$\underset{n\in\mathbb{N}}{\forall} P[n] \mapsto P[0] \wedge \underset{n\in\mathbb{N}}{\forall}(P[n] \Rightarrow P[n+1]) \tag{6.3}$$

where $P$ can apparently be kept flexible. For induction rules it is quite obvious that the latter alternative is more desirable, but in general it is hard to decide how many quantifiers shall be stripped away. The strategy pursued in our mechanism is as follows:

1. Strip away as many universal quantifiers as possible, such that *every* free higher-order variable can be kept flexible (except those that have to be made rigid anyway in the pre-processing step).

2. If this is not possible because always at least one additional variable must be made rigid, strip away all universal quantifiers.

This strategy is certainly not optimal yet, but at least it allows to accommodate the class of induction rules in an elegant way.

It is important to note that the matching algorithms generated by the compiler actually do not explicitly compute the instances of free, flexible higher-order variables, but instead only compute the positions of the sub-expressions corresponding to their formal arguments in the matched expression (see also Example 11). This approach entails the advantage that, when constructing the instance of the right-hand-side of a rule, higher-order variables are not instantiated by $\lambda$-terms leading to redexes that still have to be $\beta$-reduced afterward, but the new arguments of higher-order variables on the right-hand-side can directly be substituted at the respective, already-known positions without the need for any explicit $\beta$-reductions any more; this makes rewriting a lot faster. Furthermore, avoiding $\beta$-reductions apparently also circumvents possible problems related to the non-termination of $\beta$-reduction in untyped environments, as in the well-known term $(\underset{x}{\lambda} x[x])[\underset{x}{\lambda} x[x]]$. [1]

**Example 12.** Consider the rewrite-rule (6.3). The resulting *Mathematica* transformation rule generated automatically by the compiler reads as (up to some syntactical details)

```
Forall[{n_, ℕ}, P_] :>
  Module[{pos, n0, P0, P1, P2}
     pos = allOccurrences[P, n];
     n0 = freshVar[n];
     P0 = subst[P, pos, 0];
     P1 = subst[P, pos, n0];
     P2 = subst[P, pos, n0 + 1];
     And[P0, Forall[{n0, ℕ}, Implies[P1, P2]]]
  ]
```

First of all, the positions of the sub-expressions of the instance of `P` corresponding to the formal argument $n$ of $P$ in the rewrite-rule are computed and stored in the local program variable `pos`; this, in fact, is part of the higher-order matching algorithm, which in the present case specializes to a trivial one that, apart from filling `pos`, does not need to do anything else. Then, a fresh name for the bound variable in the instantiated right-hand-side is computed by `freshVar` and stored in the local program variable `n0`, before `subst` finally performs the actual substitutions of the new arguments in the instance of `P`, namely 0, `n0` and `n0+1`, making use of the known positions stored in `pos`. It basically calls *Mathematica*'s `ReplacePart` function, but additionally takes care of avoiding variable-capture.

[1] Substituting new arguments directly somehow corresponds to a single $\beta$-reduction.

Note that in Example 4 matters are actually very similar; there, the presentation of the resulting *Mathematica* transformation rule is simplified for the sake of simplicity, as it is meant to be only an introductory example. ■

Before we conclude this subsection and turn to some optimizations of the compiler, we want to point out three important details:

- Occurrences of free variables appearing in the scope of a binder on the left-hand-side are checked to be free of bound variables not appearing among their arguments. Any instance of $a$ in Example 11 must be free of the bound variable $x$, of course, which is checked during the matching process.
- Bound variables may only be instantiated by bound variables of the same “kind”: individual variables may only be instantiated by individual variables, and sequence variables may only be instantiated by sequence variables.
- Substitutions performed during the matching process and when constructing the instance of the right-hand-side of a rule ensure that no free variable gets bound by an enclosing binder, i. e. substitutions avoid variable capture by renaming bound variables if necessary. This is the very reason why in Example 12 the new function `subst` is used instead of the built-in `ReplacePart`.

## 6.1.5 Optimizations

The rewrite-rule compiler incorporates two optimizations for making the application of the resulting *Mathematica* transformation rules more efficient.

The first of these optimizations concerns the names of bound variables when instantiating the right-hand-side of a rule. In general, names of bound variables are chosen in such a way that

1. as few substitutions as possible have to be carried out at run-time (i. e. when the transformation rule is applied), and
2. as few fresh names have to be generated as possible at run-time.

Both tasks may be time-consuming, so detecting situations where they can be avoided and generating code that takes this information into account is certainly a benefit.

**Example 13.** Consider the rewrite-rule

$$a \in \{F[x] \underset{x \in b}{|} P[x]\} \mapsto \underset{x \in b}{\exists}(P[x] \wedge a = F[x])$$

The resulting *Mathematica* transformation rule is something like

```
Element[a_, SetOf[{x_, b_}, P_, F_]] :>
  Exists[{x, b}, And[P, a = F]]
```

As can be seen, the name of the bound variable $x$ in the expression to be rewritten, stored in the program variable x, is literally taken over when the instance of the right-hand-side of the rule is generated. This avoids generating a fresh name using `freshName` and performing any substitutions using `subst`, as the instances of $P[x]$ and $F[x]$, stored in P and F, respectively, can be literally taken over as well. ■

Of course, taking names of bound variables in the expression to be rewritten and re-using them in the new expression to be constructed must be done with care: in some situations, fresh names *must* be generated in order to avoid variable capture. In any case, the compiler generates code that re-uses as many names as possible; sometimes, however, the ultimate decision whether a name may be re-used or has to be postponed to run-time, when the concrete expression to be rewritten is known.

*Remark* 22. The preceding discussion implies that the name of a bound variable in the *instance* of the right-hand-side of a rewrite-rule might not agree with its name in the rule itself; it may even be *completely* different (e. g. not just the original name suffixed by a number to avoid name clashes).

The second optimization is achieved by automatically detecting *clones* in the code generated by the compiler and eliminating them still at compile-time. In particular, if the time-critical functions `alphaEquivalent`, `freshName`, `subst`, etc., are called several times with identical arguments, these calls are replaced by a single one storing the result in an auxiliary program variable. In connection with `alphaEquivalent`, the compiler even takes into account that the respective arguments do not even have to be identical: if the $\alpha$-equivalence of two expressions $e_1$ and $e_2$ is already known, and $e_1'$ and $e_2'$ are two sub-expressions of $e_1$ and $e_2$, respectively, both originating from the same positions, and furthermore all free variables in $e_1'$ and $e_2'$ are also free in $e_1$ and $e_2$, then $e_1'$ and $e_2'$ are apparently $\alpha$-equivalent as well; no further call to `alphaEquivalent` is necessary. Clearly, clone detection could further be improved by considering the use of *anti-unification*, as described for instance in [BKLV15].

### 6.1.6 Issues

In this subsection we address two issues arising in connection with our higher-order rewriting mechanism, in particular with the matching algorithms.

The first issue is related to free sequence variables appearing in the formal arguments of free, flexible higher-order variables on the left-hand-side of rewrite rules. Recall that sequence variables may never be *the* arguments of a flexible higher-order variable $F$, because $F$ would be immediately made rigid then, but it is well possible that they are *sub-expressions* of formal arguments. The problem with such sequence variables in the current implementation is as follows: each of the specific higher-order matching algorithms basically proceeds by solving smaller sub-problems one after the other and collecting the respective matchers; in the end, however, the individual matchers must satisfy some compatibility conditions, as the sub-problems are typically not independent of each other, and if these constraints are not met, the whole matching process fails. Now, if one individual sub-problem gives rise to more than one matcher (e. g. if it involves sequence variables), still only *one* is propagated further—but if that particular one does not meet the compatibility conditions in the end, matching fails, even though another of the possible matchers would have met the conditions.

A solution to overcome said problem is near at hand: employ a backtracking strategy that automatically tries all of the (finitely many) possible matchers of a sub-problem in order, until either one succeeds or all fail. Unfortunately, our compiler does not fall back to such a strategy yet, except for the very outermost level of rewrite-rules, i. e. outside the argument-lists of any free, flexible higher-order variables. There, the default backtracking strategy employed by *Mathematica* for syntactic matching with sequence variables is relied on.

**Example 14.** Consider the matching problems associated to the following (nonsense) rewrite-rules:

$$\begin{aligned}\underset{x}{\forall}(a... \land P\texttt{[}x\texttt{]} \land b...) &\mapsto \texttt{True}\\ (\underset{x}{\forall} P\texttt{[}x\texttt{]}) \land P\texttt{[}a... + b...\texttt{]} \land P\texttt{[}b... + a...\texttt{]} &\mapsto \texttt{True}\\ (\underset{x}{\forall} P\texttt{[}x\texttt{]}) \land P\texttt{[}a... + c\texttt{]} \land P\texttt{[}c + a...\texttt{]} &\mapsto \texttt{True}\end{aligned}$$

In the first rule, everything is fine: the two sequence variables $a...$ and $b...$ appear at the outermost level, outside the argument-list of $P$, meaning that *Mathematica*'s default backtracking strategy will find suitable instances for $a...$ and $b...$ if they exist. In the second rule, the situation is different: there, the two sequence variables *only* appear in the arguments of $P$, meaning that existing matchers might not be found. In the third rule, the situation is again different: although there the sequence variable $a...$ only appears in the arguments of $P$, too, the sub-problems of matching $P\texttt{[}a...{+}c\texttt{]}$ and $P\texttt{[}c{+}a...\texttt{]}$ against concrete terms are both *unitary* (knowing the instance of $P$, of course), meaning that no backtracking is necessary. ■

The second issue is related to the names of bound variables in the expressions to be rewritten. It is best illustrated in an example:

**Example 15.** Consider the rewrite-rule $\underset{x}{\forall}\underset{y}{\forall} P\texttt{[}x + y\texttt{]} \mapsto \texttt{True}$ and the concrete expression $\underset{z}{\forall}\underset{z}{\forall} z + z = 0$. This expression does *not* match the left-hand-side of the rule, because the variable bound by the outermost universal quantifier does not appear in the quantified expression. The matching algorithm generated by our compiler might not detect this fact, though, and claim $\underset{x}{\lambda}\, x = 0$ to be a suitable instance of $P$. ■

The approach we pursue in order to overcome the problem is fairly straightforward: before an expression is rewritten, the names of its bound variables are simply made unique in a pre-processing step, by replacing single names $x$ by pairs $(n, x)$ (where $n$ is a unique number), and reverted back to single (not necessarily unique) names in a post-processing step. Fortunately, pre- and post-processing expressions in this way has to be done only *once*, even if they are rewritten several times by several rules in one stroke: applying a *Mathematica* transformation rule generated by our compiler preserves the uniqueness-property of the names of bound variables.

### 6.1.7 More Features

Finally, we list two additional features of our higher-order rewriting mechanism that have not been mentioned so far.

The first of them concerns again sequence variables, but this time sequence variables that are themselves free, flexible higher-order variables, i. e. that appear in sub-expressions of the form $F...[x]$ etc. The rewrite-rule compiler can handle such constructs as well, interpreting, say, $F...[x]$, as a sequence of the form $F_1[x], F_2[x], \ldots$ of arbitrary length, where the $F_i$ are ordinary individual higher-order variables that are furthermore completely independent of each other. In other words: $F...[x]$ may be instantiated by an arbitrary sequence of expressions, each possibly depending on $x$. This situation parallels the ones in [BF93] and in [Kut07], where *vectors of functions* and *sequence functions*, respectively, abbreviate arbitrary sequences of functions, too.

The second feature is specific to the language of Theorema, in particular to variable-ranges of binders. Recall from Section 2.1 that there are four different kinds of variable-ranges: simple ranges, predicate ranges, set ranges and step ranges. If one wishes to introduce a rewrite-rule with a binder on its left-hand-side that shall be applicable to expressions with *any* of the four variable-ranges under the respective binder, one can do so: whenever the variable-range of a binder $\mathfrak{B}$ on the left-hand-side of a rule $r : L \mapsto R$ is a *simple* range and an additional condition is imposed on the variables bound by $\mathfrak{B}$, where the condition is a conjunction containing a free higher-order variable $P$ applied to (some of) the variables bound by $\mathfrak{B}$, and moreover $P$ only appears in such a form in $L$ and the arguments of $P$ are always distinct bound variables, then this somehow "encodes" a random range for those variables bound by $\mathfrak{B}$ that appear as arguments of $P$. The rationale behind this approach is quite simple: a range (other than a simple range) is nothing else than a condition on the bound variables; so, if a variable bound in a simple range in $r$ is constrained by other, arbitrary conditions, this justifies instantiating it with variables that are themselves bound in *any* of the four kinds of ranges.

**Example 16.** Consider the rewrite-rule

$$\underset{\substack{x\\ P[x]}}{\exists} (A[x] \vee B[x]) \mapsto \left(\underset{\substack{x\\ P[x]}}{\exists} A[x]\right) \vee \left(\underset{\substack{x\\ P[x]}}{\exists} B[x]\right)$$

Because of the additional condition $P[x]$ under the quantifier on the left-hand-side, the transformation rule generated by the compiler is applicable to *all* existentially quantified disjunctions, where the range of the bound variable may be completely arbitrary; this includes, for instance,

$$\underset{a\in\mathbb{R}}{\exists} a^2 < 2 \vee \sqrt{a} > 3 \qquad \text{and} \qquad \underset{\substack{i=2,\dots,23\\ i!>2^i}}{\exists} \texttt{isPrime}[i] \vee 3 \mid i$$

■

**Example 17.** The (nonsense) rewrite-rule

$$\underset{\substack{x\\ P[x]}}{\forall} F[x] \geq \texttt{Log}[x] \mapsto P[F[\pi]]$$

rewrites $\underset{\substack{x\in\mathbb{R}\\ x<3}}{\forall} \sqrt{x} \geq \texttt{Log}[x]$ into $\sqrt{\pi} \in \mathbb{R} \wedge \sqrt{\pi} < 3$. As can be seen, the higher-order variable justifying random ranges under the universal quantifier, $P$, comprises both the condition corresponding to the variable-range ($x \in \mathbb{R}$) and the additional condition on the bound variable ($x < 3$). ■

## 6.2 RewriteInteractiveProver

The RewriteInteractiveProver is a Theorema prover we developed in the course of the formal treatment of reduction ring theory. It consists of a collection of inference rules for general predicate logic with equality; no appeal to any kind of specific notions, such as sets, tuples, numbers etc. is made (in contrast to the ReductionRingProver and the ComplexityProver), meaning that it can easily be used in the future for explorations of other mathematical theories in Theorema, regardless of what they are about. This, in fact, is the reason why we present it here, in the chapter on contributions to Theorema, instead of Chapter 4.

As the name “RewriteInteractiveProver” suggests, many inference rules put the focus on rewriting (modulo first- and higher-order rewrite-rules extracted from formulas in the set of assumptions) and user interactions; this is what distinguishes it from the default predicate-logic prover of Theorema, called BasicTheoremaLanguage, which only supports first-order rewriting and provides only very limited possibilities for user interaction. The RewriteInteractiveProver was heavily used in the formal verification of reduction ring theory alongside the ReductionRingProver; it was not used, however, in the formal verification of the complexity analysis, simply because it has been developed only afterward.

In total the prover consists of 40 inference rules, implemented in almost 3000 lines of *Mathematica* code (not counting the code for automatically turning abstract proof objects into readable proof documents). In the following subsections, we describe some of the more interesting inference rules in more detail.

### 6.2.1 Predicate-Logic Rules

The first class of rules consists of 23 natural-deduction rules for predicate logic, including introduction- and elimination rules for conjunctions, disjunctions and equivalences, the usual introduction rules for implications and universal quantifiers, the usual elimination rule for existential quantifiers, and also a rule for classical contradiction. In addition, four rules are responsible for detecting *terminal* proof situations, i. e. proof situations that apparently hold true (e. g. if the set of assumptions is evidently inconsistent, or it contains the proof goal). Still, there are two inference rules whose operational semantics deserves closer attention.

**Modus ponens and modus tollens.** Modus ponens and modus tollens are two simple inferences from the natural-deduction proof calculus. In the RewriteInteractiveProver they are subsumed into the single rule `modusPonensTollens` that, applied to a proof situation $K, \varphi \Rightarrow \psi \vdash \Gamma$, behaves as follows:

1. First, it tries to prove the premise $\varphi$ of the implication solely by repeated backward rewriting modulo formulas in $K$, up to a certain (small) bound on the search depth. If it succeeds, the proof situation is transformed into $K, \varphi, \psi \vdash \Gamma$.

2. If $\varphi$ cannot be proved by backward rewriting as a whole, but it is a conjunction and some of the individual conjuncts *can* be proved, the proof situation is transformed into a new proof situation where these conjunctions do not appear in the premise of the implication any longer.

3. If the second step cannot be carried out either, `modusPonensTollens` tries to prove $\neg\psi$, again by backward rewriting. If it succeeds, the proof situation is transformed into $K, \neg\varphi, \neg\psi \vdash \Gamma$.

If the knowledge base of the current proof situation contains more than one implications, `modusPonensTollens` iterates through all of them (or, more precisely, the *activated* ones; see Section 6.3) until it finds one it can simplify.

*Remark* 23. `modusPonensTollens` is a generalization of both modus ponens and modus tollens, in the sense that the premise of an implication or the negation of its conclusion, respectively, does not need to appear literally among the assumptions, as long as it "follows readily" from them by repeated backward rewriting. Note here that *any* assumption $\varphi$ gives rise to the trivial backward-rule $\varphi \mapsto$ `True`.

**`Piecewise` expressions.** In Theorema as well as *Mathematica*, `Piecewise` expressions are terms or formulas of the form

$$\begin{cases} e_1 & \Leftarrow & \varphi_1 \\ e_2 & \Leftarrow & \varphi_2 \\ & \vdots & \\ e_n & \Leftarrow & \varphi_n \end{cases}$$

that are equal/equivalent to $e_i$ iff $\neg\varphi_1 \wedge \neg\varphi_2 \wedge \ldots \wedge \neg\varphi_{i-1} \wedge \varphi_i$ holds true, for $1 \leq i \leq n$. Rule `expandPiecewise` of the RewriteInteractiveProver can handle such expressions whenever they appear in a proof situation and are closed (i. e. do not depend on free variables). It proceeds in the following way:

1. First, the given `Piecewise` expression is tried to be simplified by proving and disproving the conditions $\varphi_i$ in order, again by backward rewriting. Every case $e_i \Leftarrow \varphi_i$ whose condition $\varphi_i$ can be disproved is immediately eliminated, and if $\varphi_j$ is the first condition that can be proved, all cases coming after $e_j \Leftarrow \varphi_j$ are eliminated as well and $\varphi_j$ is replaced by `True`.

2. If the resulting expression contains only one single case $e_j \Leftarrow$ `True`, it is simply replaced by $e_j$. If it is empty, i. e. all cases were eliminated in the previous step, the expression is not defined and nothing else can be done (usually, the last condition $\varphi_n$ is just the Boolean constant `True`, though).

3. Otherwise, if $m$ cases remain in the simplified `Piecewise` expression, `expandPiecewise` distinguishes $m$ cases in the proof search, in each case replacing the `Piecewise` expression by the respective $e_i$ and assuming the validity of the corresponding condition $\varphi_i$, as well as the negations of all previous conditions.

Similar to `modusPonensTollens`, `expandPiecewise` simply treats the *first* `Piecewise` expression it encounters in the current proof situation (in an activated formula; see Section 6.3).

### 6.2.2 Rules for Rewriting

Six inference rules are responsible for rewriting proof situations, either in a backward-manner or modulo equations and equivalences. Rewrite-rules are generated as soon as a formula enters the knowledge base of assumptions, be it at the very beginning of a proof in the initial proof situation, or at some point during the proof search; of course, rewrite-rules extracted from a formula $\varphi$ are only available to the rewriting mechanism in branches of the proof where $\varphi$ is known. Needless to say that the higher-order rewriting mechanism discussed in the previous section is employed for extracting rewrite-rules from formulas and translating them into *Mathematica* transformation rules.[2]

Backward rewriting proceeds by applying each available backward-rule to the current goal *once*; if several rules are applicable, an alternative branch in the proof search is initiated for each of the possible rewrites. In contrast to the BasicTheoremaLanguage prover, no *forward rewriting* is employed by default for inferring new facts from known ones.

Equational rewrite-rules are partitioned into four categories:

1. rules stemming from universally quantified equalities or equivalences, meaning that they involve free variables,

2. rules stemming from ground equalities or equivalences, meaning that they do not involve free variables,

3. rules stemming from explicit definitions, and

4. rules stemming from implicit definitions; an implicit definition is a definition whose right-hand-side is in the scope of one of the two choice-binders $\epsilon$ ("such a") or `the`.

There is a separate inference rule for dealing with each for these four categories, allowing one to apply, say, only rewrite-rules stemming from explicit definitions for unfolding definitions, without having to try the whole arsenal of all available rewrite-rules, which would be highly inefficient. In any case, note that the RewriteInteractiveProver turns equalities, equivalences and definitions into

[2]Keep in mind that the higher-order rewriting mechanism can deal with ordinary first-order rules, too.

rewrite-rules *from left to right* only, in contrast to the BasicTheoremaLanguage prover.

The *control* guiding the application of rewrite-rules distinguishes two cases based on whether the overall proof development happens automatically or interactively. In the first case, each (activated) formula is traversed in a breadth-first manner, and all rewrite-rules from the current rule-category are tried one after the other to the sub-expressions thus visited. As soon as the first rule is applicable, the respective sub-expression is rewritten once, and then the control immediately turns to the next formula; hence, at most *one* sub-expression of every formula is rewritten *once*. In the second case, when developing the proof interactively, the procedure is similar, only that *all* sub-expression of the activated formulas are rewritten *once* by *all* applicable rules, each of these rewrites leading to a separate alternative in the proof search (where this and only this rewrite is performed); hence, the eventual decision which of the possible alternatives to pursue further is left to the human user (see Section 6.3.1).

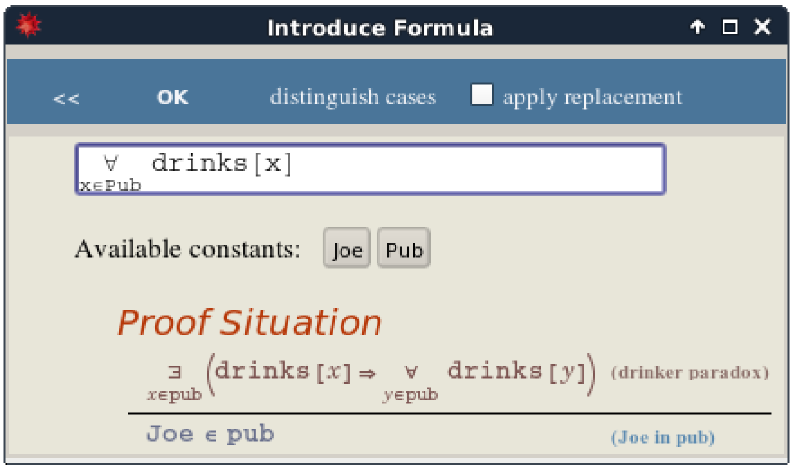
The applicability of an equational rewrite-rule $r$ to an expression $e$ not only depends on whether $e$ matches the left-hand-side of $r$, but may additionally be constrained by conditions (stemming from implications in the formula $r$ originates from). Whenever such conditions pop up, they are immediately proved by repeated backward rewriting, just as in the `modusPonensTollens` and the `expandPiecewise` inference rules. All remaining conditions that cannot be proved in this way are left as additional sub-goals; if they cannot be proved at all, maybe because they simply are not satisfied, the proof gets stuck. Therefore, the inference rule for rewriting proof situations *interactively*, discussed in the next subsection, might be advantageous whenever conditional rewrite-rules are included in the category of rules currently considered.

*Remark* 24. The inference rules for rewriting do not care at all whether the underlying set of equational rewrite-rules is confluent, terminating, etc., but simply apply the given rules *once*. This approach circumvents possible issues related to non-terminating rule sets—at least in *one single* application of the respective inference rules.

### 6.2.3 Interactive Rules

The RewriteInteractiveProver contains eleven inference rules that require some sort of user interaction when they are about to be applied to a proof situation. The four most interesting of them are listed below.

Note that at the moment *all* user interaction during the proof search, be it in connection with interactive inference rules or in connection with the interactive proof strategy described in Section 6.3, happens exclusively through *dialog*



$\forall_{x \in Pub}$ drinks[x]

Proof Situation

$\exists_{x \in pub} (\text{drinks}[x] \Rightarrow \forall_{y \in pub} \text{drinks}[y])$ (drinker paradox)

Joe ∈ pub (Joe in pub)

Figure 6.2: The dialog window for introducing a formula. The formula above the black line in the proof situation is the current goal, the one below the line is the current knowledge.

*windows* displaying the current proof situation and providing input fields, checkboxes, buttons, etc. for the user to communicate with the system.

**`instantiateInteractive`.** A typical—and very simple—example of an interactive inference rule is `instantiateInteractive`, for interactively instantiating logical quantifiers (existential quantifiers in the goal, universal quantifiers in the knowledge base), where the user is asked to provide suitable instances for the bound variables. In fact, a similar rule is part of the default Basic-TheoremaLanguage prover as well, differing from `instantiateInteractive` only in that the latter allows to instantiate several quantifiers with identical variable-ranges at once, which turned out to be quite useful in many situations.

**`introduceFormula`.** This rule provides the human user with the possibility of initiating *sub-proofs* at any point in the main proof. In more concrete terms, it enables stating an arbitrary formula in the current proof context (i. e. possibly depending on arbitrary-but-fixed constants) and proving it from the current knowledge base of assumptions in a sub-proof; afterward, the proved formula is added to the assumptions in the "main branch" of the proof, of course. Alternatively, it is also possible not to prove the given formula, but rather to distinguish two cases based on whether the formula holds or not.

Figure 6.2 shows the dialog window of `introduceFormula` as appearing in the proof of the so-called *drinker paradox* [Smu78], where two cases based on whether "everybody in the pub drinks" are distinguished.

**`rewritePS`.** The name of this interactive rule stands for "rewrite the proof situation", which is exactly what it does: similar to the rules for rewriting discussed in the previous subsection, it applies available equational rewrite-rules to formulas in the current proof situation. However, it allows the user to fine-tune the rewriting process in the sense that she may explicitly specify

- the individual rewrite-rules to be used, or, more precisely, the formulas they originate from (not only the categories they belong to!),
- the formulas to be rewritten,
- how conditional rewrite-rules are to be treated, i. e. whether rules constrained by conditions whose validity cannot be established instantly by backward rewriting shall still be applied, and
- one of four available controls guiding the application of the rewrite-rules.

The four controls being available at the moment include the two that have already been mentioned in the previous subsection, together with two additional controls, resembling *Mathematica*'s `ReplaceAll` and `ReplaceRepeated` functions, respectively: in the first case, every activated formula is traversed in a breadth-first manner, and for each sub-expression thus visited the specified rewrite-rules are tried one after the other until an applicable rule is found, in which case the rule is applied to the respective sub-expression *once*; the resulting new expressions are not rewritten further, but it may well be that *several* sub-expressions of a single formula are rewritten in this way. In the second case the process is basically the same, only that, as it name suggests, the activated formulas are rewritten *repeatedly* until no more rewrites are possible; this, of course, bears the danger of non-termination.

**`addKnowledge`.** In Theorema, the knowledge from the background theory to be used in a proof must in principle be specified when the proof attempt is initiated and is then fixed throughout the whole proof search. In practice, however, this approach turned out not to be feasible, because very often the knowledge thus specified happens to be insufficient for finishing the proof (either because one simply forgot to include some formulas, or because further lemmas are needed in the theory before one can proceed). Inference rule `addKnowledge` finds a remedy for such scenarios: it enables the user to add arbitrary knowledge from the background theory in the midst of the proof search, by selecting the respective formulas in a graphical "knowledge browser" that resembles the default one from the Theorema commander. Whenever new knowledge is added, the abstract proof object representing the proof constructed so far, including all pending proof situations, is updated in such a way that afterward it looks as if the newly added knowledge had been there the whole time. Moreover, the resulting proof document neither distinguishes between “new” and “old” formulas when summarizing the knowledge used in the proof, nor mentions the application of `addKnowledge` at all.

*Remark* 25. Although the interactive inference rules from the RewriteInteractiveProver can in principle be used together with any underlying proof strategy, it must nevertheless be noted that they were designed to work particularly well together with the *interactive* proof strategy discussed in the next section.

## 6.3 Interactive Proof Strategy

Traditionally, the Theorema system has always put the emphasis on *automatic* rather than *interactive* proof development, meaning that users are usually supposed to only set up the proof attempt (specify the proof goal, select the knowledge base, adjust some prover settings) and then wait for the system to either find a proof automatically or fail. After finishing the formalization of the complexity analysis of Buchberger’s algorithm presented in Section 5, though, we realized that automatic proving quickly becomes unfeasible if theories grow big and formulas become lengthy and technical, so we decided to implement an *interactive proof strategy* for efficiently handling the formalization of reduction ring theory presented in Section 4.

The result is a dialog-oriented user interface, giving the user full control over the development of proofs (e. g. which inference rule to apply in a certain situation, how to apply it, etc.; see below). A text-based interface, where the user simply writes down the individual inference steps one after the other and has them checked and carried out by the underlying system, as in basically all other well-known proof assistants, is planned to be added to Theorema in the mid-term future.

Actually, there has already been an environment for interactive proof generation in the old version of Theorema our interactive proof strategy has many features in common with (e. g. both are dialog-oriented), see [PK05]. Still, we did not just migrate the existing environment to Theorema 2.0 and extended and modified it a bit to accommodate our needs, but really started completely from scratch again; this seemed to be the more reasonable approach, as the internal architecture of Theorema 2.0 differs considerably from the one of Theorema 1. Also note that the dynamic, graphical presentation of *proof trees*, which constitutes a central component of the interactive environment discussed in [PK05], is now an integral part of Theorema 2.0 [Win12] and hence has absolutely nothing to do specifically with our interactive proof strategy.

### 6.3.1 How it Works in Practice

The interactive proof strategy, as its name suggests, is simply a Theorema proof strategy that can be selected just as any other (automatic) proof strategy when initiating a new proof; no further actions are required. As soon as the proving starts, and whenever a new proof situation arises whilst proving, the interactive proof strategy is called (note that internally it is just a *Mathematica* function). Then, it either

- applies a high-priority inference rule from the selected prover fully automatically, or, if this is not possible,
- opens a dialog-window displaying all relevant information for the user to decide how to proceed.

By default, every Theorema inference rule has a *priority* attached to it: the priority is an integer between 1 and 100, where the lower the number, the higher the priority of the respective rule; typically, proving strategies take these priorities into account when applying the rules, in the sense that they try to apply rules with higher priority first. Although every rule gets a default priority, the user can easily adjust them according to his needs; the same is true for the threshold determining whether a rule is regarded a high-priority rule or not by the interactive strategy (by default this threshold is 20). Examples of inference rules typically having a very high priority are rules for detecting terminal proof situations, as well as the rule that splits conjunctions in the knowledge base into their conjuncts; these rules definitely "do not do any harm", in the sense that they will never render a provable proof situation unprovable—this being the reason why such inference are performed instantly.

Anyway, the interactive strategy tries all high-priority rules in order and applies the first possible one to the current proof situation, in which case new proof situations arise and the whole process starts again. Otherwise, if none of these high-priority rules is applicable, a dialog-window pops up, displaying the current proof situation (i. e. the current proof goal and list of assumptions, as ordinary Theorema formulas in easily readable, two-dimensional syntax) and asking the human user to decide how to continue (by clicking on buttons or menu items). In particular, the user may choose one of three main possibilities, namely

- specify the inference rule to apply to the current proof situation by choosing from the list of all available inference rules from the selected prover (except the high-priority rules that have already been tried unsuccessfully), or

- select a different pending proof situation where the proof search shall continue, e. g. if the current one is only one of several alternatives, which furthermore does not look very promising, or

- abort the proof attempt.

There are a couple of things to note in connection with the first of these three possibilities, choosing an inference rule. First of all, no pre-selection of applicable rules is made, meaning that the chosen rule might actually not be applicable to the current proof situation. In such a case, no progress in the proof development happens, of course, and the user may simply do something else instead. However, it could well be the other way round, too: the requested inference rule might be applicable in more than one way (inference rules for rewriting the proof situation are perhaps the best example for that). In such a case, the user may choose one of the possible alternatives in yet another dialog window. Finally, before applying an inference rule, it is possible to activate and deactivate formulas in the current proof situation (by marking the respective check-boxes in the main dialog window; see Figure 6.3) for specifying the "target" formulas of the rule. For instance, the two rules `modusPonensTollens` and `expandPiecewise` discussed in Section 6.2.1 usually operate on the first *activated* implication or `Piecewise`-expression, respectively, they encounter when scanning the proof situation, thus giving the user the opportunity to explicitly specify which formulas they shall be applied to in advance. Note, however, that an inference rule may also ignore whether a formula is activated or not; this solely depends on its concrete implementation and cannot be influenced by the interactive proof strategy.

Besides working directly on the development of the proof, the user may take a range of further actions as well, including

- adjusting the prover settings, e. g. the priorities of the inference rules or the proof strategy (in particular, the interactive proof strategy may be replaced by an automatic one if the current proof situation seems to be simple enough),

- inspecting the proof document of the proof created so far,

- inspecting the internal representations of the current proof situation (goal, knowledge base, additional data), the whole proof object, and the list of available rewrite-rules (e. g. for debugging),

- saving the proof object in an external file for creating a "secure point" the proof attempt may be resumed at later on; note that our interactive proof strategy lacks the possibility to undo erroneous steps.

Figure 6.3 illustrates the application of an inference rule to a proof situation, using the interactive strategy. The first of the three windows depicts the main dialog window of the interactive strategy, titled “Proof Commander”. The topmost bar (white text on blue background) constitutes the menu bar. Below, the name of the inference rule about to be applied when hitting “Enter”, as selected by the user, is displayed (black text on light-blue background); in the situation displayed, a quantified formula is going to be instantiated interactively. The current proof situation is presented in the remaining part of the window, divided into the goal (above the black line) and the list of assumptions (below the black line). As can be seen, the proof goal is the only activated formula, meaning that the interactive instantiation will instantiate the existential quantifier in the goal rather than the universal quantifier in the assumptions.

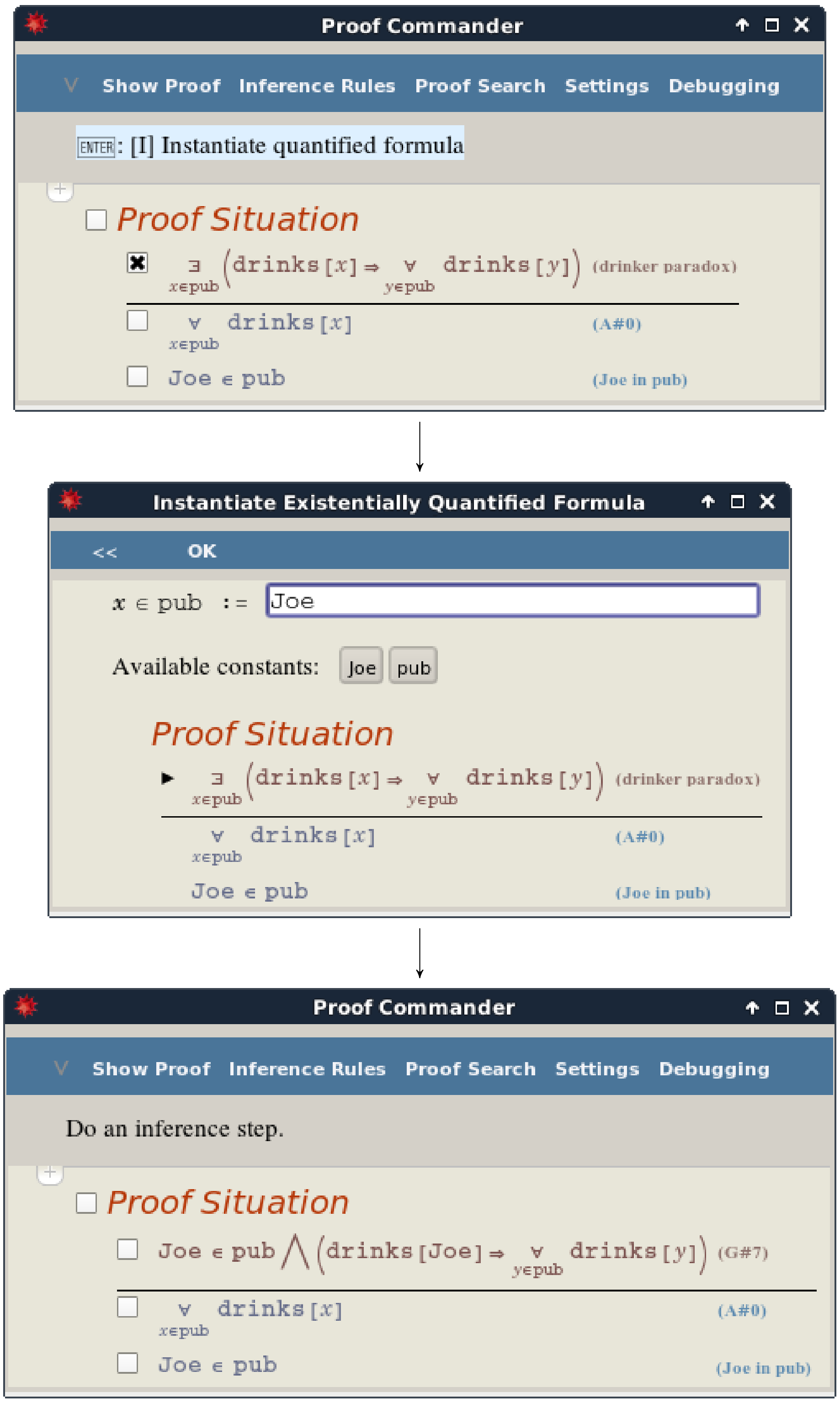
The second window in Figure 6.3 actually has nothing to do with the interactive proof strategy but stems from the requested (interactive!) inference rule, `instantiateInteractive`, allowing the user to input a witness term for the existential quantifier. The third window, finally, displays the new proof situation after instantiating the quantifier, and waits again for instructions from the user.

### 6.3.2 Implementation Details

As already mentioned above, the interactive proof strategy is integrated into the proving architecture of Theorema 2.0 as a separate proof strategy and implemented a such, i. e. it is essentially a function mapping a proof situation to a list of new proof situations; a little bit of a hack is needed to realize the more advanced features of the interactive strategy, like selecting different pending proof situations where the proof search shall continue.

Something similar is also true for the communication between the strategy and the inference rules, e. g. about which formulas are activated and which are not: at the moment, all this happens through global variables whose values are set accordingly. Passing the information directly to the inference rules would definitely be more elegant and robust, but is currently not possible because of the very architecture of Theorema 2.0.

Another weakness in the implementation of the interactive proof strategy is related to its dialog windows: whenever the user confirms an interactive action (e. g. by clicking on a button), the respective window is closed and a new window with updated content pops up. The entire interface could certainly be made much more attractive if one single window, whose content is updated dynamically, was fixed once and for all while proving. Even more, all the graphical interface for interactive proving could perhaps also be integrated into the main Theorema-Commander window—this remains future work.



ENTER: [I] Instantiate quantified formula

Proof Situation

$\exists_{x \in \text{pub}} \left(\text{drinks}[x] \Rightarrow \forall_{y \in \text{pub}} \text{drinks}[y]\right)$ (drinker paradox)

$\forall_{x \in \text{pub}} \text{drinks}[x]$ (A#0)

$\text{Joe} \in \text{pub}$ (Joe in pub)

$x \in \text{pub} :=$ Joe

Available constants: Joe pub

Proof Situation

$\exists_{x \in \text{pub}} \left(\text{drinks}[x] \Rightarrow \forall_{y \in \text{pub}} \text{drinks}[y]\right)$ (drinker paradox)

$\forall_{x \in \text{pub}} \text{drinks}[x]$ (A#0)

$\text{Joe} \in \text{pub}$ (Joe in pub)

Do an inference step.

Proof Situation

$\text{Joe} \in \text{pub} \bigwedge \left(\text{drinks}[\text{Joe}] \Rightarrow \forall_{y \in \text{pub}} \text{drinks}[y]\right)$ (G#7)

$\forall_{x \in \text{pub}} \text{drinks}[x]$ (A#0)

$\text{Joe} \in \text{pub}$ (Joe in pub)

Figure 6.3: Performing an (interactive) inference using the interactive proof strategy.

## 6.4 TheoryTools

One of the lessons we learned from the formalization of reduction ring theory is that keeping track of the (logical) structure of libraries of extensive Theorema theories, though highly desirable, is almost impossible without the support of automatic tools that serve precisely this purpose. Hence, we developed a *Mathematica*-package, called TheoryTools, consisting of methods for automatically retrieving and analyzing the logical structure of Theorema theories, in particular the dependencies of their formulas on each other.

Although the implementation of most functions contained in the package is fairly simple and straightforward, TheoryTools nevertheless proved extremely useful in practice: quite often, existing Theorema theories have to be restructured (e. g. changing the order of formulas, relabeling formulas, modifying formulas that have already been used as assumptions in proofs, etc.), making it of utmost importance to know how these changes affect other theories, and in particular proofs that have already been carried out. In Theorema 2.0 proofs are stored in external proof files; hence, if a formula occurring in a proof is modified afterward (e. g. by relabeling it), the respective file should be updated as well—this, however, does *not* happen automatically, but must be taken care of by the user herself.

The TheoryTools package is divided into three main components: the TheoryAnalyzer for analyzing the logical dependencies of formulas, the (still prototypical) ConsistencyChecker for checking whether sets of formulas are consistent, and the TheoryManipulator for automatically manipulating sets of proof files (e. g. changing formula-labels).

### 6.4.1 TheoryAnalyzer

The main purpose of the TheoryAnalyzer, as mentioned above, lies in analyzing the logical structure of Theorema theories, i. e. the logical dependencies of their formulas on each other. It does so by first reading all proof files in a given list of directories and retrieving, from each file, the *proof goal* and the used *assumptions* as Theorema formulas; these formulas have additional context information, such as unique identifiers, the names and locations of the notebooks they are contained in, as well as their labels attached to them, making it easy to identify formulas occurring in several different files. From this information, the TheoryAnalyzer then constructs a directed graph whose nodes correspond to the formulas thus found, and where a two nodes $A$ and $B$ are connected by a directed edge from $A$ to $B$ iff the formula $\varphi_A$ corresponding to $A$ is an assumption in a proof of the formula $\varphi_B$ corresponding to $B$, i. e. one of the proof files read has $\varphi_B$ as its goal and $\varphi_A$ in its list of assumptions—in this case, $\varphi_B$ *depends* *on* $\varphi_A$.[3] Because of the obvious one-to-one correspondence between formulas in the theory and nodes in the graph, the terms "formula" and "node" are used interchangeably below.

Once the dependency graph has been created, the TheoryAnalyzer offers a range of well-known graph-theoretic functions for analyzing it, namely

- collecting all direct/indirect/basic assumptions of a given formula,
- collecting all direct/indirect/terminal consequences of a given formula,
- finding the relation between two given formulas (i. e. whether one of them depends on the other), and
- detecting cycles; a cycle in the graph corresponds to a circular argument in the theory, meaning that the graph should ideally be *acyclic*.

A formula $\varphi$ is a *basic assumption* of a formula $\psi$ iff $\psi$ depends on $\varphi$, either directly or indirectly, and $\varphi$ is not proved itself. $\varphi$ is a *terminal consequence* of $\psi$ iff $\varphi$ depends on $\psi$, again either directly or indirectly, and $\varphi$ is never used as an assumption in a proof.

Apart from the functions operating on the dependency graph, the TheoryAnalyzer also provides functionality for automatically retrieving the sizes of individual theories by means of the numbers of unproved axioms and proved theorems, as well as the sizes of proofs of individual formulas by means of the numbers of inference steps. Moreover, the information thus gathered can then be visualized automatically in nicely-formatted theory dependency graphs (similar to Figure 4.1) and statistics diagrams (similar to Figure 4.2).

*Remark* 26. In the current implementation of the TheoryAnalyzer, dependency graphs are not updated when the corresponding theories change (e. g. new formulas are inserted). Instead, the graphs must be constructed from scratch again, but only on an explicit request by the user.

## 6.4.2 ConsistencyChecker

The very nature of Theorema is not to force its users to adhere to any kind of conventions when formulating their mathematical theories (although they are advised to do so, of course). This, in particular, is true for conventions ensuring the *consistency* of the respective theories, at least relative to the presumed consistency of the foundations of mathematics (like Zermelo-Fraenkel set theory). Many other mathematical assistant systems adopt a far more rigorous point

[3] $\varphi_B$ might have several proofs with different assumptions. The TheoryAnalyzer cannot handle such situations properly, though.

of view; for instance, in the well-known Isabelle system [Wen16] new notions (types, functions, predicates) may only be introduced by quite specific means, e. g. primitive- or well-founded recursion, minimizing the danger of contradictions.

It is clear, though, that the consistency of formalizations is an important issue in Theorema, too, and therefore we developed a tool for checking the consistency of a set of Theorema formulas, called ConsistencyChecker. Of course, checking whether a set of formulas is consistent of contradictory is a highly non-trivial task, and even undecidable in our setting of general untyped higher-order predicate logic. So, what the ConsistencyChecker actually does is merely analyzing/inspecting the given formulas and partitioning them into basically two subsets: a subset of those formulas that are *possibly problematic* (but not necessarily) and an as-large-as-possible subset that is *probably consistent* (but not with $100\%$ certainty either), assisting the user in discovering potential contradictions. Hence, the ConsistencyChecker in its current incarnation *never* returns definite answers such as “The given set is surely consistent” but *always* leaves the final decision to the user.

Let us now shed some light on how the detection of possibly problematic formulas proceeds; to that end, let $F$ be an arbitrary set of closed (i. e. without free variables) Theorema formulas. First and foremost one must note that at present, the ConsistencyChecker can only deal with *definitions*; all other formulas in $F$ are immediately, without any closer look, marked as possibly problematic. The reason behind this seemingly strange behavior is that the ConsistencyChecker was designed to work together with the TheoryAnalyzer, operating on the sets of all basic assumptions of important theorems, which are typically made up mainly from definitions.

So, assume that $F$ entirely consists of (universally quantified, conditional) definitions. The ConsistencyChecker now iterates through all pairs of elements of $F$, and for each pair $(\gamma, \delta)$ checks whether the left-hand-sides of the definitions $\gamma$ and $\delta$ *overlap*, in which case it instantly marks both of them as possibly problematic. In our sense, two expressions, potentially containing free variables stemming from universal quantifiers, overlap iff one of them can be unified with some sub-expression of the other. For instance, $x\,(2 + y)$ and $z + \pi$ overlap if $y$ and $z$ are free variables, the unifying substitution being $\{y \leftarrow \pi, z \leftarrow 2\}$.

The attentive reader certainly notices that the procedure carried out by the ConsistencyChecker for detecting contradictory definitions is neither correct nor complete, meaning that contradictory definitions might be marked as unproblematic and consistent definitions as problematic:

**Example 18.** The single definition $\mathrm{c} := \mathrm{c} + 1$ is marked as unproblematic by the ConsistencyChecker if the constant $\mathrm{c}$ does not occur in the left-hand-side of any other formula in $F$. Still, the definition *is* problematic in some sense, as unfolding it would cause an infinite loop. ■

The previous example illustrates that no special attention is payed to (mutual) recursive definitions and their “termination”. More than that, the right-hand-sides of definitions are completely ignored at all, even if taking them into account would render otherwise problematic formulas unproblematic:

**Example 19.** The ConsistencyChecker marks the two definitions

$$\mathtt{f[0]} := 0$$

and

$$\underset{x}{\forall}\ \mathtt{f[}x\mathtt{]} := x$$

as problematic, because their left-hand-sides apparently overlap. However, the only unifier also unifies their right-hand-sides, meaning that the definitions are in fact consistent. ■

Conditions constraining definitions are not taken into account either:

**Example 20.** The ConsistencyChecker marks the two definitions

$$\underset{x}{\forall}\ \mathtt{P[}x\mathtt{]} \Rightarrow \mathtt{f[}x\mathtt{]} := 0$$

and

$$\underset{x}{\forall}\ \neg\mathtt{P[}x\mathtt{]} \Rightarrow \mathtt{f[}x\mathtt{]} := 1$$

as problematic, because their left-hand-sides apparently overlap. However, no instance of $x$ satisfies both P and $\neg$P at the same time, meaning that as before the definitions are actually consistent. ■

Although matters are already complicated enough for *first-order* definitions, *higher-order* definitions make things even worse:[4]

**Example 21.** It is well-known that the innocent-looking higher-order definition

$$\underset{P}{\forall}\ \mathtt{R[}P\mathtt{]} :\Leftrightarrow \neg P\mathtt{[}P\mathtt{]}$$

is contradictory; this is essentially Russel’s paradox phrased for predicates rather than sets. ■

In the absence of types, detecting whether a *single* higher-order definition is consistent or contradictory can be arbitrarily complex. The approach pursued

[4]Recall that a universally quantified variable is considered higher-order iff it is applied to arguments somewhere in the respective formula.

by the ConsistencyChecker to circumvent this problem is simple: it tests higher-order definitions for overlapping left-hand-sides (with other first- and higher-order formulas) as well, just as any other definitions, but even if no overlaps are found, higher-order definitions are still marked as possibly problematic—only because they are higher-order.[5]

Since the core component for detecting overlapping definitions is unification, and (higher-order) unification, maybe even with sequence variables, is known to be undecidable in general (see the remarks in Section 6.1), the design of the unification algorithm of the ConsistencyChecker is similar to that of the unification algorithm in the higher-order rewriting mechanism. Informally, the algorithm tries to answer as accurately as possible, but in case of doubt simply returns "unifiable"; more formally, it prefers false-positive answers over false-negative ones, to be on the safe side: claiming that two overlapping definitions are non-overlapping is clearly worse than claiming that two non-overlapping definitions are overlapping.

Definitions with universally quantified sequence variables appearing on their left-hand-sides have to marked as possibly problematic, too, since pattern-matching with sequence variables is not unitary in general and may thus cause problems, as can be seen in the following

**Example 22.** The first-order definition

$$\underset{x,y\ldots,z\ldots}{\forall} \mathtt{g}[y\ldots, x, z\ldots] := x$$

is contradictory, as it allows to infer both $\mathtt{g}[1,2] = 1$ and $\mathtt{g}[1,2] = 2$. ■

Finally, the ConsistencyChecker also pays special attention to *implicit definitions*, i. e. definitions whose right-hand-side is in the scope of one of the two choice-binders "such a" (denoted by $\epsilon$ in Theorema) or "the": if a definition of this kind is not explicitly constrained by existence (and uniqueness) conditions on the defining term, it is marked as possibly problematic.

**Example 23.** The implicit definition

$$\underset{x}{\forall} (\underset{y}{\exists!}\, x = y^2) \Rightarrow \mathtt{sqrt1}[x] := \underset{y}{\mathtt{the}}\, x = y^2$$

is fine, as the existence and uniqueness of a term $t$ satisfying $s = t^2$ is explicitly required in order for the definition to be applicable to some term $\mathtt{sqrt1}[s]$; only note that the "unique existence"-quantifier $\exists!$ is not part of the Theorema language and only used here to abbreviate an otherwise lengthy formula.

[5]Testing higher-order definitions for overlaps although they are marked as problematic anyway might also detect problematic first-order definitions.

In contrast, the ConsistencyChecker marks the definition

$$\underset{x\in\mathbb{R}}{\forall}\ x \geq 0 \Rightarrow \texttt{sqrt2}[x] := \underset{y\in\mathbb{R}}{\epsilon}\ x = y^2 \wedge y \geq 0$$

as possibly problematic, even though the existence (and uniqueness) of a suitable term might be provable in the underlying background theory. ■

Summarizing, a definition is marked as possibly problematic by the ConsistencyChecker iff

- its left-hand-side overlaps with the left-hand-side of another definition,
- it is higher-order,
- sequence variables appear in its left-hand-side, or
- it is an implicit definition where existence (and uniqueness) of the defining term are not explicitly required.

The preceding discussion shows that the current implementation of our ConsistencyChecker is still prototypical and incomplete, with room for improvement in a variety of respects. Just as an example, external first-order reasoners, like CVC4 [BT$^+$], Vampire [VRH$^+$] or Z3 [dMB$^+$], could be employed to detect further contradictions or, conversely, formulas that have erroneously been marked as problematic.

In any case, the consistency of formalized mathematics is a serious issue that might have to be addressed more carefully in Theorema in the future.

### 6.4.3 TheoryManipulator

The third and last component of the TheoryTools-package is the so-called TheoryManipulator. It provides functions for automatically manipulating proof files, like adjusting the labels or sources of formulas (the source of a formula is a string composed from the name and location of the notebook it is contained in). Our own experience shows that these tasks occur quite frequently when formalizing large mathematical theories, hence having a tool for automating them at one's disposal facilitates matters considerably. After all, one definitely does not want to update hundreds of files manually.

*Remark* 27. Updating proof files when relabeling a formula in a notebook is actually not *necessary* (at least from Theorema's point of view), but highly *desirable* for maintaining a coherent formalization.

# Chapter 7

# Conclusion

In this thesis we presented the formalization of a variety of aspects of the theory of Gröbner bases in Theorema 2.0, including a completely generic, formally verified implementation of Buchberger's algorithm in reduction rings and the analysis of the algorithm in the bivariate case in the original setting of polynomial rings over fields. Although neither of these two theories is itself novel, we still managed to make some contributions (simplifications, generalizations, corrections) to both of them in the frame of our formal treatment.

A substantial portion of reduction ring theory is now available in a fully formal and verified form as a collection of Theorema theories, including all the main definitions, results and algorithms. It is clear, though, that because of the sheer size of Gröbner bases theory in the original setting, which is a special case of reduction ring theory, there are still hundreds of interesting and important results that are not part of the formalization yet; see Section 7.2 for a list of some of these. Still, the formalization of reduction ring theory also necessitated the formal representation of many elementary mathematical concepts, such as sets, numbers and tuples, resulting in several *independent* Theorema theories that may easily be reused in future theory explorations in Theorema.

The apparent value of all our formalization, of the elementary theories, reduction ring theory and the complexity analysis alike, is that it may serve as the foundation for further (formal and even informal) investigations in the field of Gröbner bases and related subjects. We are convinced that, in general, computerized and certified mathematics bears the prospect of aiding the extension of existing and the development of completely new theories in many ways.

Nevertheless, our work may also be regarded a major case study of mathematical theory exploration in Theorema. In that sense, we gained a lot of valuable experience in how to reasonably structure and build up formal theories and how to approach extensive formalizations in the first place; our concrete findings are summarized in Section 7.1. Moreover, our extensive use of Theorema was the main incentive for finally implementing a couple of strongly needed missing features for efficiently working with the system, listed in Chapter 6. Some of these features were known to be missing for a long time already or have even been part of the old version of Theorema (like an environment for interactive proving), but others, like most of those implemented in the TheoryTools package, seem to not have attracted the attention of the Theorema project before at all. However, there are also equally important features that still await being looked at more closely; some of them are listed in Section 7.2.

## 7.1 Findings

Below we summarize our most important findings from the formal treatment of reduction ring theory and of the complexity analysis in Theorema. Each of them might well be relevant for future theory explorations, too.

The first finding concerns functors and domains, and their relation to sets. As explained in Section 4.1.2, the carrier of a domain does not necessarily have to be a set (according to the axioms of Zermelo-Fraenkel set theory), necessitating the explicit use of a separate predicate, `isDomain`, to characterize domains whose carriers *do* constitute sets. Furthermore, we also gave an account on the duplicate effort that is needed when working with domains and sets in parallel, e. g. because notions have to be introduced once for domains and once for sets. All this is truly not very elegant and should be addressed more thoroughly in the future (of course, one could in principle restrict oneself to work with domains *only* and completely forget about sets, but this does not seem to be a feasible solution either, as sets are ubiquitous in mathematics).

The second finding is related to interactive proof development, as opposed to automatic proving. Clearly, the ultimate goal of systems like Theorema is to automate as much aspects, not only of proving but also of theory exploration as a whole, but especially the formalization of the complexity analysis revealed that proving complicated and technical formulas, at present, cannot be done (semi-)automatically in a satisfactory fashion in Theorema—even following Theorema's paradigm of equipping the reasoning mechanism with tailor-made, theory-specific inference rules.

The third and last major finding concerns the consistency of Theorema theories. As mentioned at the beginning of Section 6.4.2, Theorema adopts a fairly liberal point of view toward the way how mathematical content is allowed to enter the system. This, apparently, entails the danger of introducing inconsistencies, maybe even inconsistencies that are extremely hard to detect (ignoring issues related to the consistency of the well-established foundations of mathematics, of course). We believe, though, that a mathematical assistant system, like

Theorema, should not only support its users in proving, computing, solving, etc., but also ensure that what the users do really “makes sense”.[1] A first step in this direction could be the integration of a *type system* enforcing the well-typedness of formulas already when they are entered into the system (and not only when they appear in proofs).

A problem related to the consistency of mathematical theories is the equally important problem of establishing the correctness of a given set of inference rules (e. g. relative to a fixed set of basic inferences). Preliminary research in this area, in the frame of Theorema 1, has already been conducted in [GB07], but no concrete implementation has come into existence up to now.

## 7.2 Future Work

Future work related to the contents of this thesis can be divided into three parts: work on the theoretical aspects of reduction rings, work on their formalization (and that of Gröbner bases in general), and work on the Theorema system itself.

### 7.2.1 Reduction Rings

Regarding the further development of reduction ring theory, two research directions could be of particular interest. On the one hand, one could try to generalize the commutative setting to *non-commutative* reduction rings and thus non-commutative Gröbner bases (of one- and two-sided ideals). Apart from adjusting the axioms and perhaps adding new ones, the crucial modification of the existing formulation would probably affect the reduction relation: instead of multiplying the reductor only by *one* multiplier from the left, allow for a second multiplier that is multiplied from the right. In fact, some of the phenomena Gröbner bases theory in non-commutative polynomial rings over fields exhibits, like the presence of more than one critical pair for a given pair of polynomials, are already present in commutative reduction ring theory anyway and should hence not cause any major difficulties. Other issues, like the non-existence of finite Gröbner bases for certain finitely generated ideals, might have to be addressed in more detail in connection with non-commutative reduction rings, though.

On the other hand, an interesting question raised recently by Buchberger in a personal communication is the following: is it possible to generalize the all-important elimination property of Gröbner bases in *polynomial* rings (see Section 3.5.3) to *arbitrary* reduction rings without any polynomial structure, and

[1]Theorema could still offer opportunities for “experimental” content, as long as it is explicitly marked as such.

would this make sense? A quick inspection of the elimination property in polynomial rings reveals that the crucial property of the reduction relation in such structures probably is

$$p \rightarrow_q \Rightarrow \; \mathrm{lp}(q) \preceq \mathrm{lp}(p) \tag{7.1}$$

allowing one to conclude that a polynomial $p$ may only be reduced by polynomials $q$ not involving indeterminates that are greater than those appearing in $p$, if a lexicographic term order is used. Generalizing (7.1) to arbitrary reduction rings, where the notion of "leading power-product" does not exist, might be challenging.

### 7.2.2 Formalization

The existing formalization of reduction rings presented in Chapter 4 can be extended in many ways. First and foremost, there are still some rings that are known to be reduction rings, like the Gaussian integers, cyclic foldings and the $k$-fold direct product of a reduction ring with itself; each of these structures could be integrated into the formalization. In particular, direct products are essential for computing Gröbner bases of syzygies, as discussed in Section 3.5.4, which themselves play an important role in Gröbner bases theory in the original setting.

Furthermore, the original setting of polynomial rings over fields being a special case of reduction rings opens a multitude of opportunities for further extending the formalization, concentrating now exclusively on the original setting. For instance, applications of Gröbner bases in various areas, e. g. solving systems of algebraic equations, geometric theorem proving, any many more, could first be formalized as an object-level theory and then employed as certified reasoning techniques on the meta level. In addition, the research on the relation between Gröbner bases and *generalized Sylvester matrices* conducted by Manuela Wiesinger-Widi in her PhD thesis [WW15] has the potential of becoming a promising research direction that may profit from a formal treatment in a mathematical assistant system.

### 7.2.3 Theorema

In connection with Theorema there are at least two possibilities for further enhancements that have not been mentioned at all so far in this thesis, the first of these being a direct consequence of interactive proving: if proofs are developed interactively, there is a dire need for a mechanism that *automatically* "reruns" existing proofs. As it turned out, quite frequently proved theorems have to be slightly modified to properly incorporate changes in the background theory, meaning that it should be possible to take their existing (interactively generated) proofs and pass them to a tool that somehow "checks" them again, automatically adjusting inference steps that are not correct any more because of the aforementioned modifications. Of course, the tool envisioned will only be able to cope with situations where the old- and the new versions of the theorems are sufficiently close to each other (e. g. equal up to renaming of constants, adding further conjuncts to conjunctions in the knowledge base, or removing conjuncts from the proof goal)—but this should suffice for the vast majority of practical applications.

The second enhancement aims at the automation not only of proving, but of basically all tasks frequently encountered in theory explorations. Probably the best example for illustrating this idea is the construction of a *quotient algebra* $\mathcal{A}_{/\sim}$ (represented, e. g., as a Theorema functor) from a given algebra $\mathcal{A}$ and an equivalence relation $\sim$. Once sufficiently many facts about $\mathcal{A}$ and $\sim$ are known, i. e. proved by the user, constructing $\mathcal{A}_{/\sim}$ and transferring definitions and theorems from $\mathcal{A}$ to $\mathcal{A}_{/\sim}$ becomes trivial and may thus be delegated to a tool that takes care of it fully automatically. Besides constructing quotient algebras, the envisioned tool could also be employed for defining inductive sets, algebraic data types, (primitive) recursive functions, etc., and for automatically proving simple facts about these notions following a well-defined pattern. Please note that tools serving precisely this purpose already exist in other mathematical assistant systems, e. g. Isabelle/HOL [BBD+16, HK13, Wen16].

# Appendix A

# A Sample Proof in Theorema

Below, we present a sample Theorema-proof of a formula appearing in our formalization of the theory of reduction rings. Although the validity of the formula is quite apparent and the proof thus comparatively short, it nevertheless exhibits a couple of interesting aspects of both the natural-style presentation of proofs in Theorema in general, as well as proving in reduction ring theory specifically. Note that the proof presented here, as most other proofs, was developed fully interactively with our interactive proof strategy discussed in Section 6.3.

*Remark* 28. The main contents of the screen-shots shown below have not been modified in any way and precisely illustrate how proof documents in Theorema look like.

We start with some general remarks on the presentation of proofs in Theorema; see also [BJK$^+$16]:

- Proof documents are displayed in separate notebooks (see Figures A.2 to A.6), whereas proof trees are visualized in the Theorema Commander window (see Figure A.1).
- Proof documents are automatically created from abstract proof objects stored in separate files, meaning that the informal explanatory text can easily be adapted even after the proof was constructed; this, in particular, means that a different language may be chosen (at least if corresponding language data is installed in Theorema).
- At the beginning of any proof document, the proof goal and the used knowledge are explicitly summarized (as can be seen in Figure A.2).
- If the cursor is moved over the label of a formula, the whole formula is displayed in a tooltip (see the tooltip in Figure A.3). Furthermore, formula-labels are in fact hyperlinks, linking to the first occurrence of the respective formula in the proof document.

- Every node in the proof tree corresponds to an inference step in the proof. Clicking on a node selects precisely those cells in the proof document that explain the corresponding inference.

- The cell brackets at the right of proof documents group together common parts of proofs, e. g. sub-goals, and hence facilitate navigation and allow to hide uninteresting parts.

The formula whose proof is presented below is

$$\underset{\mathtt{isReductionRing}[\mathcal{R}]}{\forall}\ \underset{[B]\underset{\mathtt{DomainSets}[\mathcal{R}]}{\in}}{\forall}\ \underset{[a,b]\underset{\mathcal{R}}{\in}}{\forall}\ a \underset{\mathcal{R}}{\rightarrow_B} b \Rightarrow a \underset{\mathcal{R}}{\equiv_B} b \qquad (\rightarrow \subseteq \equiv)$$

in Theory ReductionRings.nb. Informally, it states that for any reduction ring $\mathcal{R}$, all sets $B$ of ring elements and all ring elements $a$ and $b$, if $a$ can be reduced to $b$ modulo $B$ in one step, then $a$ and $b$ are congruent modulo the ideal generated by $B$. The knowledge used in the proof includes the definitions of ideal congruence and the reduction relation, and some simple facts about reduction modulo sets and finite sums; see Figure A.2 for the complete list.

The proof basically proceeds by unfolding the definition if ideal congruence, instantiating the resulting existentially quantified goal by the (obvious) terms, and then doing some rewriting modulo known equalities, in particular the two equalities involving finite sums; the rest of the proof is completely trivial. Nevertheless, some specific remarks are in place:

- As can be seen, the two trivial identities $1 - 1 = 0$ and $\sum_{\substack{\mathcal{D} \\ i=1,\ldots,0}} f[i] = \underset{\mathcal{D}}{0}$ are explicitly included among the assumptions for rewriting. This would not be necessary: Theorema provides a lot of built-in computation rules for the most common operations in arithmetic, set theory, etc. that can be employed in proofs. These built-in rules are not made use of here only for demonstrating that Theorema does not force its users to rely on any specific built-ins.

- In contrast, the computation of the lengths of tuples happened automatically in the proof: in Figure A.5, after instantiating the existential quantifier, $|\langle c\rangle|$ is instantly computed to be $1$ (the upper endpoint in the sum), without any explicit inference (e. g. for rewriting) being applied. However, the original, unsimplified formulas can still be inspected in tooltips, see Figure A.5.

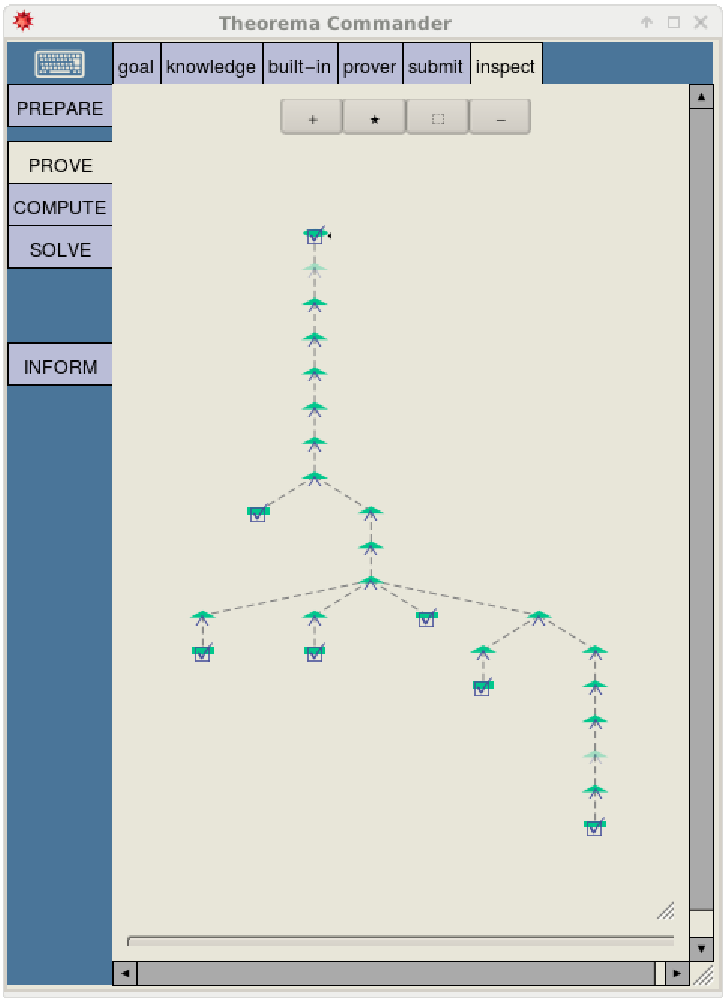
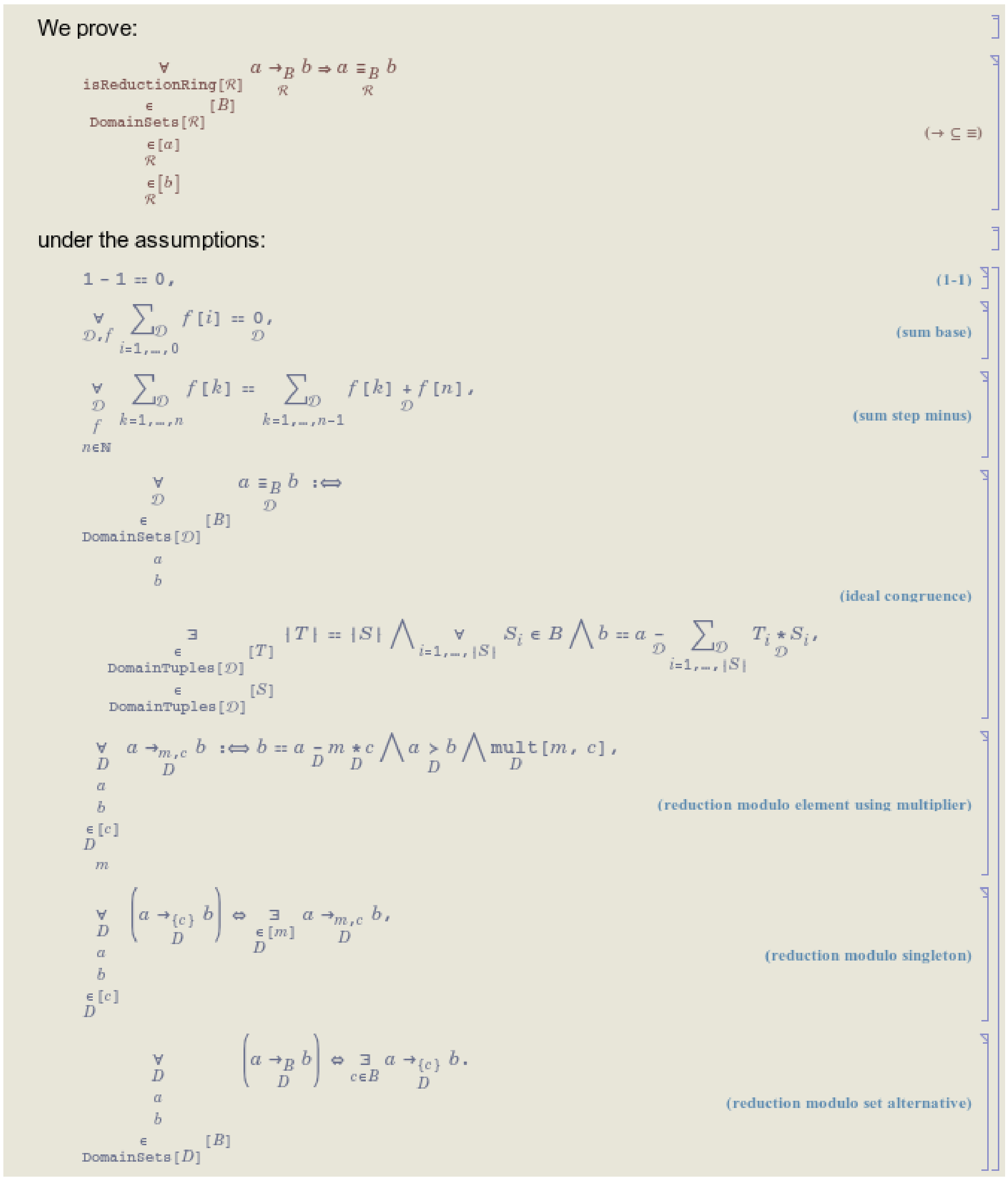


Figure A.1: The visualization of the proof tree corresponding to the proof of Formula (→ ⊆ ≡) in the Theorema Commander.

- Assumptions (sum base) and (sum step minus) are higher-order formulas, necessitating the use of the higher-order rewriting mechanism presented in Section 6.1; rewriting modulo these equalities takes place at the bottom of Figure A.5 and in the middle of Figure A.6.

- The very last step in the proof, depicted in Figure A.6, is the application of the special inference rule `equalityCommRing1` of the ReductionRing-Prover for simplifying equalities in commutative rings with identity, as explained in Section 4.5.5. Note that $\mathcal{R}$ is known to be a reduction ring and hence is a commutative ring with identity by definition.



We prove:

$$\underset{\substack{\text{isReductionRing}[\mathcal{R}] \\ \underset{\text{DomainSets}[\mathcal{R}]}{\in}[B] \\ \underset{\mathcal{R}}{\in}[a] \\ \underset{\mathcal{R}}{\in}[b]}}{\forall} a \underset{\mathcal{R}}{\to_B} b \Rightarrow a \underset{\mathcal{R}}{\equiv_B} b \qquad (\to \subseteq \equiv)$$

under the assumptions:

$$1 - 1 == 0, \qquad \text{(1-1)}$$

$$\underset{\mathcal{D},f}{\forall} \sum_{i=1,\ldots,0}^{\mathcal{D}} f[i] \underset{\mathcal{D}}{==} 0, \qquad \text{(sum base)}$$

$$\underset{\substack{\mathcal{D} \\ f \\ n\in\mathbb{N}}}{\forall} \sum_{k=1,\ldots,n}^{\mathcal{D}} f[k] == \sum_{k=1,\ldots,n-1}^{\mathcal{D}} f[k] \underset{\mathcal{D}}{+} f[n], \qquad \text{(sum step minus)}$$

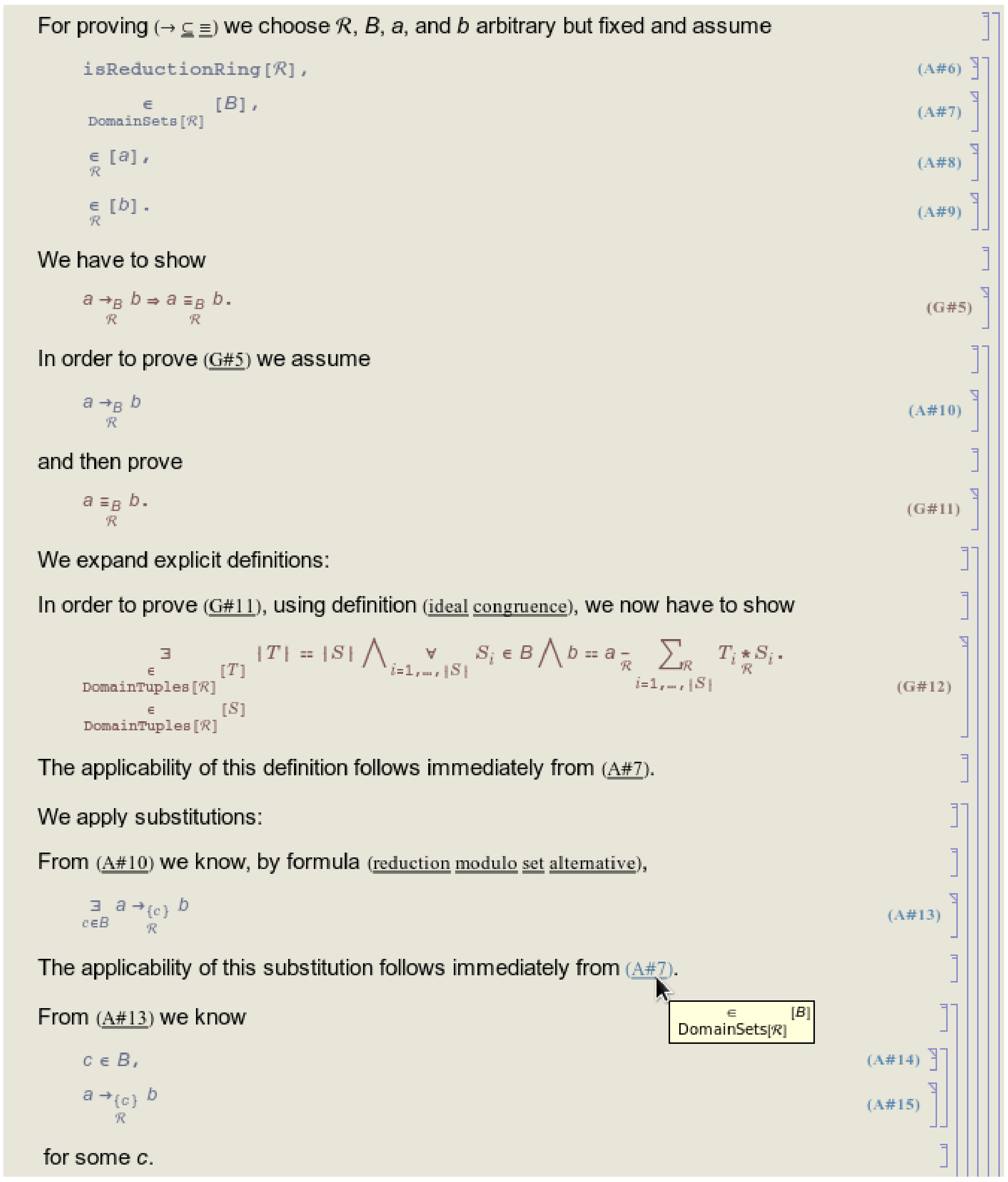
$$\underset{\substack{\mathcal{D} \\ \underset{\text{DomainSets}[\mathcal{D}]}{\in}[B] \\ a \\ b}}{\forall} a \underset{\mathcal{D}}{\equiv_B} b :\Longleftrightarrow \underset{\substack{\underset{\text{DomainTuples}[\mathcal{D}]}{\in}[T] \\ \underset{\text{DomainTuples}[\mathcal{D}]}{\in}[S]}}{\exists} |T| == |S| \bigwedge \underset{i=1,\ldots,|S|}{\forall} S_i \in B \bigwedge b == a \underset{\mathcal{D}}{-} \sum_{i=1,\ldots,|S|}^{\mathcal{D}} T_i \underset{\mathcal{D}}{*} S_i, \qquad \text{(ideal congruence)}$$

$$\underset{\substack{D \\ a \\ b \\ \underset{D}{\in}[c] \\ m}}{\forall} a \underset{D}{\to_{m,c}} b :\Longleftrightarrow b == a \underset{D}{-} m \underset{D}{*} c \bigwedge a \underset{D}{>} b \bigwedge \underset{D}{\text{mult}}[m, c], \qquad \text{(reduction modulo element using multiplier)}$$

$$\underset{\substack{D \\ a \\ b \\ \underset{D}{\in}[c]}}{\forall} \left(a \underset{D}{\to_{\{c\}}} b\right) \Leftrightarrow \underset{\underset{D}{\in}[m]}{\exists} a \underset{D}{\to_{m,c}} b, \qquad \text{(reduction modulo singleton)}$$

$$\underset{\substack{D \\ a \\ b \\ \underset{\text{DomainSets}[D]}{\in}[B]}}{\forall} \left(a \underset{D}{\to_B} b\right) \Leftrightarrow \underset{c\in B}{\exists} a \underset{D}{\to_{\{c\}}} b. \qquad \text{(reduction modulo set alternative)}$$

Figure A.2: The first part of the proof, where the proof goal and the used knowledge are summarized.



For proving (→ ⊆ ≡) we choose $\mathcal{R}$, $B$, $a$, and $b$ arbitrary but fixed and assume

$$\mathtt{isReductionRing}[\mathcal{R}], \qquad \text{(A\#6)}$$

$$\underset{\mathtt{DomainSets}[\mathcal{R}]}{\in}[B], \qquad \text{(A\#7)}$$

$$\underset{\mathcal{R}}{\in}[a], \qquad \text{(A\#8)}$$

$$\underset{\mathcal{R}}{\in}[b]. \qquad \text{(A\#9)}$$

We have to show

$$a \underset{\mathcal{R}}{\rightarrow_B} b \Rightarrow a \underset{\mathcal{R}}{\equiv_B} b. \qquad \text{(G\#5)}$$

In order to prove (G#5) we assume

$$a \underset{\mathcal{R}}{\rightarrow_B} b \qquad \text{(A\#10)}$$

and then prove

$$a \underset{\mathcal{R}}{\equiv_B} b. \qquad \text{(G\#11)}$$

We expand explicit definitions:

In order to prove (G#11), using definition (ideal congruence), we now have to show

$$\underset{\substack{\underset{\mathtt{DomainTuples}[\mathcal{R}]}{\in}[T] \\ \underset{\mathtt{DomainTuples}[\mathcal{R}]}{\in}[S]}}{\exists} |T| = |S| \bigwedge \underset{i=1,\ldots,|S|}{\forall} S_i \in B \bigwedge b = a \underset{\mathcal{R}}{-} \underset{i=1,\ldots,|S|}{\sum_{\mathcal{R}}} T_i \underset{\mathcal{R}}{*} S_i. \qquad \text{(G\#12)}$$

The applicability of this definition follows immediately from (A#7).

We apply substitutions:

From (A#10) we know, by formula (reduction modulo set alternative),

$$\underset{c \in B}{\exists} a \underset{\mathcal{R}}{\rightarrow_{\{c\}}} b \qquad \text{(A\#13)}$$

The applicability of this substitution follows immediately from (A#7).

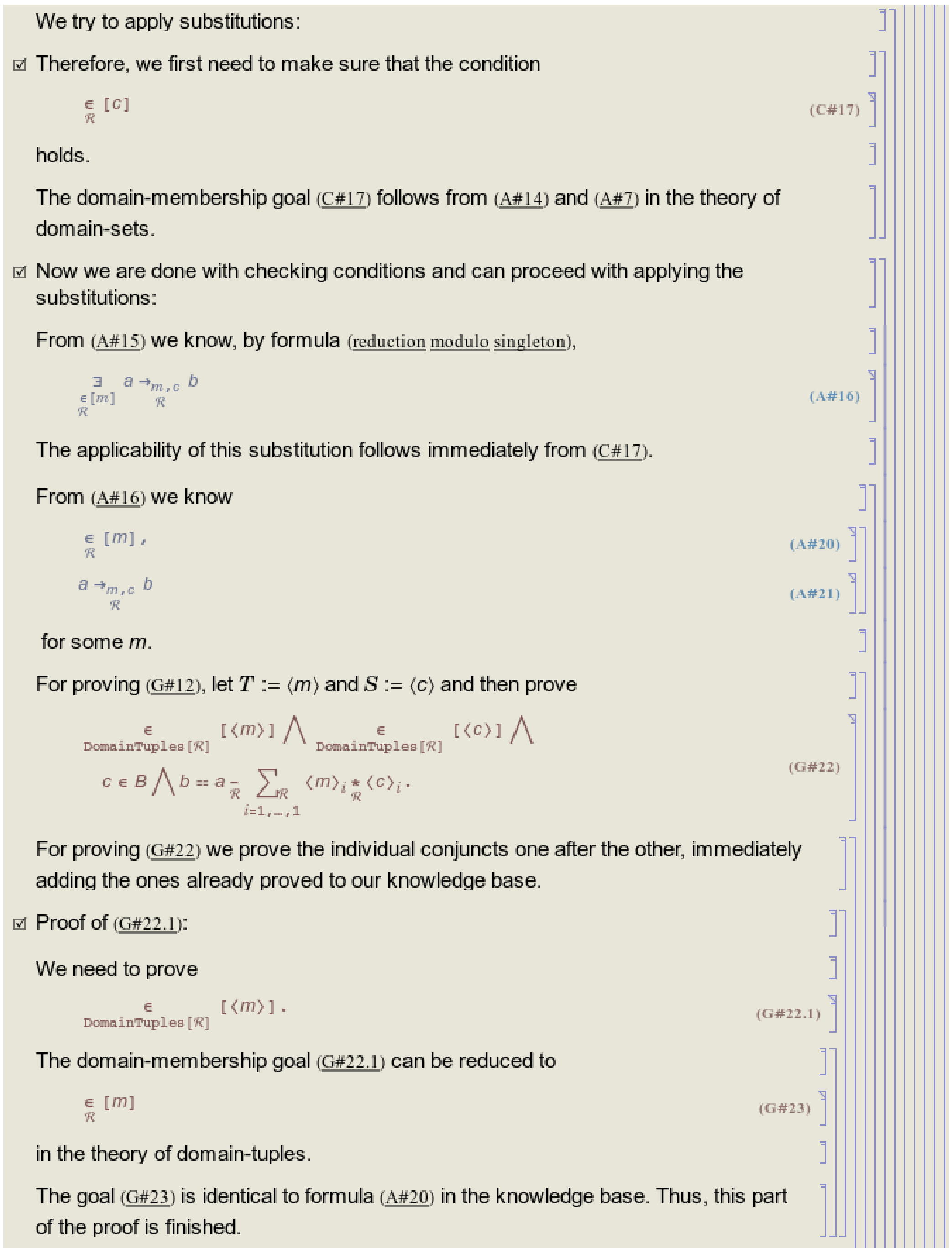
From (A#13) we know

$$c \in B, \qquad \text{(A\#14)}$$

$$a \underset{\mathcal{R}}{\rightarrow_{\{c\}}} b \qquad \text{(A\#15)}$$

for some $c$.

Figure A.3: The second part of the proof, where some simple inferences are performed and definitions are unfolded.



We try to apply substitutions:

☑ Therefore, we first need to make sure that the condition

$$\underset{\mathcal{R}}{\in}[c] \qquad \text{(C\#17)}$$

holds.

The domain-membership goal (C#17) follows from (A#14) and (A#7) in the theory of domain-sets.

☑ Now we are done with checking conditions and can proceed with applying the substitutions:

From (A#15) we know, by formula (reduction modulo singleton),

$$\underset{\underset{\mathcal{R}}{\in}[m]}{\exists}\; a \underset{\mathcal{R}}{\to_{m,c}} b \qquad \text{(A\#16)}$$

The applicability of this substitution follows immediately from (C#17).

From (A#16) we know

$$\underset{\mathcal{R}}{\in}[m], \qquad \text{(A\#20)}$$

$$a \underset{\mathcal{R}}{\to_{m,c}} b \qquad \text{(A\#21)}$$

for some $m$.

For proving (G#12), let $T := \langle m\rangle$ and $S := \langle c\rangle$ and then prove

$$\underset{\text{DomainTuples}[\mathcal{R}]}{\in}[\langle m\rangle] \bigwedge \underset{\text{DomainTuples}[\mathcal{R}]}{\in}[\langle c\rangle] \bigwedge c \in B \bigwedge b == a \underset{\mathcal{R}}{-} \underset{i=1,\ldots,1}{\textstyle\sum_{\mathcal{R}}} \langle m\rangle_i \underset{\mathcal{R}}{*} \langle c\rangle_i. \qquad \text{(G\#22)}$$

For proving (G#22) we prove the individual conjuncts one after the other, immediately adding the ones already proved to our knowledge base.

☑ Proof of (G#22.1):

We need to prove

$$\underset{\text{DomainTuples}[\mathcal{R}]}{\in}[\langle m\rangle]. \qquad \text{(G\#22.1)}$$

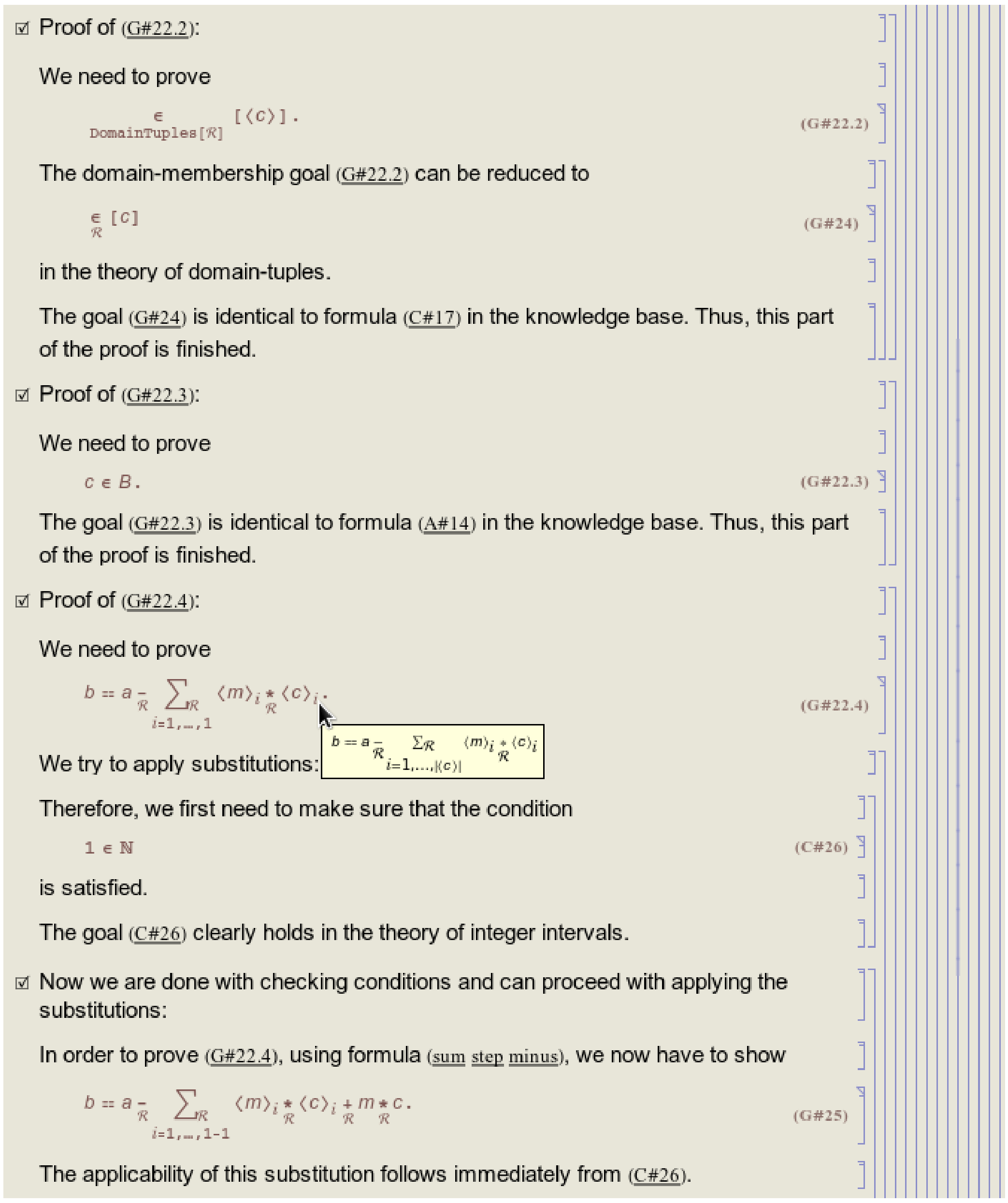
The domain-membership goal (G#22.1) can be reduced to

$$\underset{\mathcal{R}}{\in}[m] \qquad \text{(G\#23)}$$

in the theory of domain-tuples.

The goal (G#23) is identical to formula (A#20) in the knowledge base. Thus, this part of the proof is finished.

Figure A.4: The third part of the proof, where the existentially quantified goal is instantiated by concrete terms.



☑ Proof of (G#22.2):

We need to prove

$$\underset{\text{DomainTuples}[\mathcal{R}]}{\in}[\langle c\rangle]. \qquad \text{(G\#22.2)}$$

The domain-membership goal (G#22.2) can be reduced to

$$\underset{\mathcal{R}}{\in}[c] \qquad \text{(G\#24)}$$

in the theory of domain-tuples.

The goal (G#24) is identical to formula (C#17) in the knowledge base. Thus, this part of the proof is finished.

☑ Proof of (G#22.3):

We need to prove

$$c \in B. \qquad \text{(G\#22.3)}$$

The goal (G#22.3) is identical to formula (A#14) in the knowledge base. Thus, this part of the proof is finished.

☑ Proof of (G#22.4):

We need to prove

$$b == a \underset{\mathcal{R}}{-} \sum_{\mathcal{R}\, i=1,\ldots,1} \langle m\rangle_i \underset{\mathcal{R}}{*} \langle c\rangle_i. \qquad \text{(G\#22.4)}$$

$$b == a \underset{\mathcal{R}}{-} \sum_{\mathcal{R}\, i=1,\ldots,|\langle c\rangle|} \langle m\rangle_i \underset{\mathcal{R}}{*} \langle c\rangle_i$$

We try to apply substitutions:

Therefore, we first need to make sure that the condition

$$1 \in \mathbb{N} \qquad \text{(C\#26)}$$

is satisfied.

The goal (C#26) clearly holds in the theory of integer intervals.

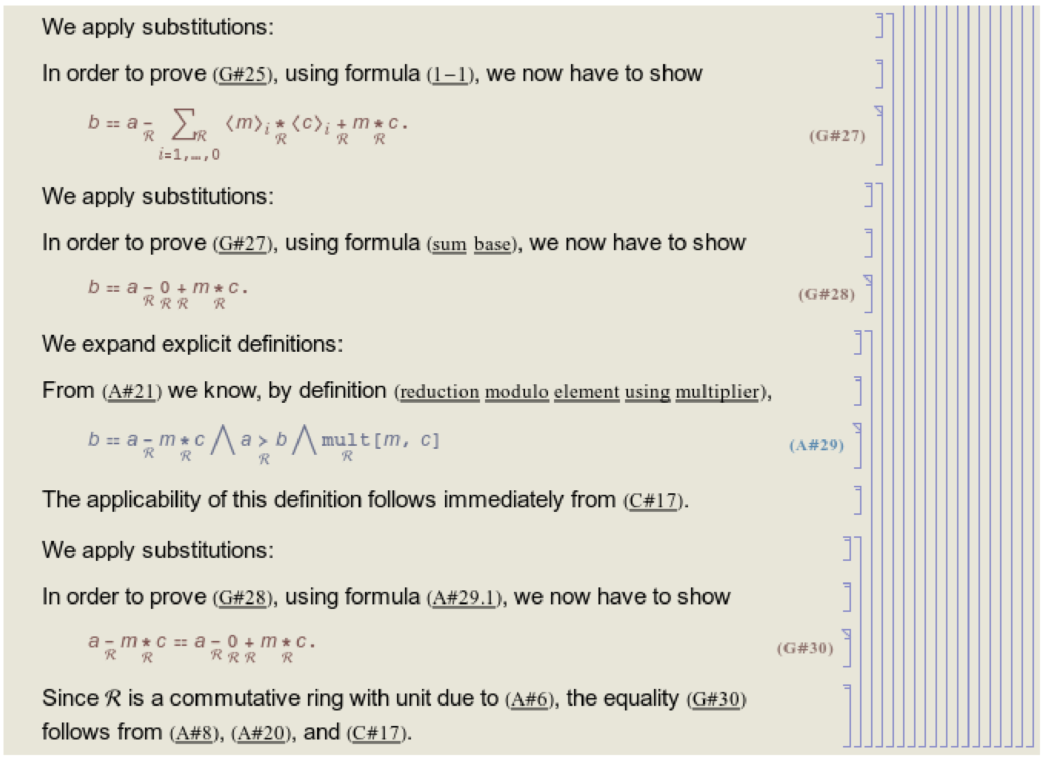
☑ Now we are done with checking conditions and can proceed with applying the substitutions:

In order to prove (G#22.4), using formula (sum step minus), we now have to show

$$b == a \underset{\mathcal{R}}{-} \sum_{\mathcal{R}\, i=1,\ldots,1-1} \langle m\rangle_i \underset{\mathcal{R}}{*} \langle c\rangle_i \underset{\mathcal{R}}{+} m \underset{\mathcal{R}}{*} c. \qquad \text{(G\#25)}$$

The applicability of this substitution follows immediately from (C#26).

Figure A.5: The fourth part of the proof, where three of the four resulting sub-goals are shown.



We apply substitutions:

In order to prove (G#25), using formula (1 − 1), we now have to show

$$b == a \underset{\mathcal{R}}{-} \sum_{i=1,\ldots,0}^{\mathcal{R}} \langle m\rangle_i \underset{\mathcal{R}}{*} \langle c\rangle_i \underset{\mathcal{R}}{+} m \underset{\mathcal{R}}{*} c. \qquad \text{(G\#27)}$$

We apply substitutions:

In order to prove (G#27), using formula (sum base), we now have to show

$$b == a \underset{\mathcal{R}}{-} 0_{\mathcal{R}} \underset{\mathcal{R}}{+} m \underset{\mathcal{R}}{*} c. \qquad \text{(G\#28)}$$

We expand explicit definitions:

From (A#21) we know, by definition (reduction modulo element using multiplier),

$$b == a \underset{\mathcal{R}}{-} m \underset{\mathcal{R}}{*} c \bigwedge a \underset{\mathcal{R}}{>} b \bigwedge \underset{\mathcal{R}}{\mathtt{mult}}[m, c] \qquad \text{(A\#29)}$$

The applicability of this definition follows immediately from (C#17).

We apply substitutions:

In order to prove (G#28), using formula (A#29.1), we now have to show

$$a \underset{\mathcal{R}}{-} m \underset{\mathcal{R}}{*} c == a \underset{\mathcal{R}}{-} 0_{\mathcal{R}} \underset{\mathcal{R}}{+} m \underset{\mathcal{R}}{*} c. \qquad \text{(G\#30)}$$

Since $\mathcal{R}$ is a commutative ring with unit due to (A#6), the equality (G#30) follows from (A#8), (A#20), and (C#17).

Figure A.6: The last part of the proof, where the remaining sub-goal is shown by rewriting and simplifying an equality in $\mathcal{R}$.

# Bibliography

[Ayo83] Christine W. Ayoub. On Constructing Bases for Ideals in Polynomial Rings over the Integers. *Journal of Number Theory*, 17:204–225, 1983.

[Bal10] Clemens Ballarin. Tutorial to Locales and Locale Interpretation. In Laureano Lambán, Ana Romero, and Julio Rubio, editors, *Contribuciones Científicas en Honor de Mirian Andrés Gómez*, pages 123–140. Servicio de Publicaciones de la Universidad de La Rioja, 2010. Part of the Isabelle documentation.

[BBD$^+$16] Julian Biendarra, Jasmin C. Blanchette, Martin Desharnais, Lorenz Panny, Andrei Popescu, and Dmitriy Traytel. *Defining (Co)datatypes and Primitively (Co)recursive Functions in Isabelle/HOL*, 2016. Part of the Isabelle documentation, https://isabelle.in.tum.de/dist/Isabelle2016/doc/datatypes.pdf.

[BBG$^+$15] Grzegorz Bancerek, Czeslaw Byliński, Adam Grabowski, Artur Korniłowicz, Roman Matuszewski, Adam Naumowicz, Karol Pąk, and Josef Urban. Mizar: State-of-the-art and Beyond. In Manfred Kerber, Jacques Carette, Cezary Kaliszyk, Florian Rabe, and Volker Sorge, editors, *Intelligent Computer Mathematics (Proceedings of CICM 2015, Washington, US, July 13–17)*, volume 9150 of *Lecture Notes in Artificial Intelligence*, pages 261–279. Springer, 2015.

[BC04] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development – Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004.

[BCJ$^+$06] Bruno Buchberger, Adrian Craciun, Tudor Jebelean, Laura Kovacs, Temur Kutsia, Koji Nakagawa, Florina Piroi, Nikolaj Popov, Judit Robu, Markus Rosenkranz, and Wolfgang Windsteiger. Theorema: Towards Computer-Aided Mathematical Theory Exploration. *Journal of Applied Logic*, 4(4):470–504, 2006.

[BDJ$^+$00] Bruno Buchberger, Claudio Dupré, Tudor Jebelean, Franz Kriftner, Koji Nakagawa, Daniela Văsaru, and Wolfgang Windsteiger. The Theorema Project: A Progress Report. In Manfred Kerber and Michael Kohlhase, editors, *Symbolic Computation and Automated Reasoning (Proceedings of CALCULEMUS 2000, Symposium on the Integration of Symbolic Computation and Mechanized Reasoning, St. Andrews, UK, August 6–7)*, pages 98–113. A.K. Peters, 2000.

[Ber78] George M. Bergman. The Diamond Lemma for Ring Theory. *Advances in Mathematics*, 29(2):178–218, 1978.

[BF93] Woodrow W. Bledsoe and Guohui Feng. SET-VAR. *Journal of Automated Reasoning*, 11(3):293–314, 1993.

[BJK$^+$16] Bruno Buchberger, Tudor Jebelean, Temur Kutsia, Alexander Maletzky, and Wolfgang Windsteiger. Theorema 2.0: Computer-Assisted Natural-Style Mathematics. *Journal of Formalized Reasoning*, 9(1):149–185, 2016.

[BKLV15] Alexander Baumgartner, Temur Kutsia, Jordi Levy, and Mateu Villaret. Nominal Anti-Unification. In Maribel Fernandez, editor, *26th International Conference on Rewriting Techniques and Applications (Proceedings of RTA'15, Warsaw, Poland, June 29–July 3)*, volume 36 of *Leibniz International Proceedings in Informatics*, pages 57–73, 2015.

[BT$^+$] Clark Barrett, Cesare Tinelli, et al. CVC4. http://cvc4.cs.nyu.edu/web/.

[Buc65] Bruno Buchberger. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal (An Algorithm for Finding the Basis Elements in the Residue Class Ring Modulo a Zero Dimensional Polynomial Ideal)*. PhD thesis, Mathematical Institute, University of Innsbruck, Austria, 1965. English translation in *Journal of Symbolic Computation* 41(3–4):475–511, Special Issue on Logic, Mathematics, and Computer Science: Interactions.

[Buc70] Bruno Buchberger. Ein algorithmisches Kriterium für die Lösbarkeit eines algebraischen Gleichungssystems (An Algorithmic Criterion for the Solvability of an Algebraic System of Equations). *Aequationes Mathematicae*, pages 374–383, 1970. (English translation in *Gröbner Bases and Applications (Proceedings of the International Conference "33 Years of Gröbner Bases", 1998)*, London Mathematical Society Lecture Note Series 251, Cambridge Univerity Press, 1998, pages 535–545).

[Buc79] Bruno Buchberger. A Criterion for Detecting Unnecessary Reductions in the Construction of Gröbner Bases. In Edward W. Ng, editor, *Symbolic and Algebraic Computation (Proceedings of EUROSAM'79, Marseille, June 26-28)*, volume 72 of *Lecture Notes in Computer Science*, pages 3–21. Springer, 1979.

[Buc83a] Bruno Buchberger. A Critical-Pair/Completion Algorithm in Reduction Rings. RISC Report Series 83-21, Research Institute for Symbolic Computation (RISC), Johannes Kepler University Linz, 1983.

[Buc83b] Bruno Buchberger. A Note on the Complexity of Constructing Gröbner-Bases. In J. A. van Hulzen, editor, *Computer Algebra (Proceedings of EUROCAL'83, European Computer Algebra Conference, London, UK, March 28–30)*, volume 162 of *Lecture Notes in Computer Science*, pages 137–145. Springer, 1983.

[Buc83c] Bruno Buchberger. Miscellaneous Results on Gröbner-Bases for Polynomial Ideals II. Technical Report 83-1, Department of Computer And Information Sciences, University of Delaware, 1983.

[Buc84] Bruno Buchberger. A Critical-Pair/Completion Algorithm for Finitely Generated Ideals in Rings. In Egon Börger, Gisbert Hasenjaeger, and Dieter Rödding, editors, *Logic and Machines: Decision Problems and Complexity (Proceedings of "Rekursive Kombinatorik", Münster, Germany, May 23–28)*, volume 171 of *Lecture Notes in Computer Science*, pages 137–161. Springer, 1984.

[Buc96] Bruno Buchberger. Mathematica as a Rewrite Language. In Tetsuo Ida, Atsushi Ohori, and Masato Takeichi, editors, *Functional and Logic Programming (Proceedings of the 2nd Fuji International Workshop on Functional and Logic Programming, Shonan Village Center, Japan, November 1–4)*, pages 1–13. World Scientific, 1996.

[Buc98] Bruno Buchberger. Introduction to Gröbner Bases. In Bruno Buchberger and Franz Winkler, editors, *Gröbner Bases and Applications*, number 251 in London Mathematical Society Lectures Notes Series, pages 3 – 31. Cambridge University Press, 1998.

[Buc03] Bruno Buchberger. Gröbner Rings in Theorema: A Case Study in Functors and Categories. Technical Report 2003-49, Research Institute for Symbolic Computation, Johannes Kepler University Linz, November 2003.

[Buc04a] Bruno Buchberger. Proving by First and Intermediate Principles, 2004. Invited talk at Workshop on Types for "Mathematics / Libraries of Formal Mathematics", University of Nijmegen, The Netherlands. http://www.risc.jku.at/publications/download/risc_2344/2004-11-01-A.pdf.

[Buc04b] Bruno Buchberger. Towards the Automated Synthesis of a Gröbner Bases Algorithm. *RACSAM - Revista de la Real Academia de Ciencias (Review of the Spanish Royal Academy of Science), Serie A: Mathematicas*, 98(1):65–75, 2004.

[BW79] Bruno Buchberger and Franz Winkler. Miscellaneous Results on the Construction of Gröbner-Bases for Polynomial Ideals. Technical Report 137, Institut für Mathematik, Johannes Kepler University Linz, 1979.

[C$^+$85] Robert L. Constable et al. *Implementing Mathematics with The Nuprl Proof Development System.* Prentice Hall, 1985. Available online at http://www.nuprl.org/book/.

[Cra08] Adrian Craciun. *Lazy Thinking Algorithm Synthesis in Gröbner Bases Theory.* PhD thesis, Research Institute for Symbolic Computation, Johannes Kepler University Linz, Austria, 2008.

[CW07] Amine Chaieb and Makarius Wenzel. Context aware Calculation and Deduction: Ring Equalities via Gröbner Bases in Isabelle. In Manuel Kauers, Manfred Kerber, Robert Miner, and Wolfgang Windsteiger, editors, *Towards Mechanized Mathematical Assistants (Proceedings of Calculemus'2007, Hagenberg, Austria, June 27–30)*, volume 4573 of *Lecture Notes in Computer Science*, pages 27–39. Springer, 2007.

[dMB$^+$] Leonardo de Moura, Nikolaj Bjørner, et al. Z3. https://github.com/Z3Prover/z3.

[Dow93] Gilles Dowek. Third-order Matching is Decidable. *Annals of Pure and Applied Logic*, 69(2–3):135–155, 1993.

[FD14] Maria Francis and Ambedkar Dukkipati. Reduced Gröbner bases and Macaulay–Buchberger Basis Theorem over Noetherian rings. *Journal of Symbolic Computation*, 65:1–14, November 2014.

[G$^+$13] Georges Gonthier et al. A Machine-Checked Proof of the Odd Order Theorem. In Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie, editors, *Interactive Theorem Proving (Proceedings of*

*ITP'2013, Rennes, France, July 22–26)*, volume 7998 of *Lecture Notes in Computer Science*, pages 163–179. Springer, 2013.

[Gal85] André Galligo. Some Algorithmic Questions on Ideals of Differential Operators. In Bob F. Caviness, editor, *EUROCAL'85 (Proceedings of the European Conference on Computer Algebra, Linz, Austria, April 1–3)*, volume 204 of *Lecture Notes in Computer Science*, pages 413–421. Springer, 1985.

[GB07] Martin Giese and Bruno Buchberger. Towards Practical Reflection for Formal Mathematics. RISC Report Series 07-05, Research Institute for Symbolic Computation, Johannes Kepler University Linz, Schloss Hagenberg, 4232 Hagenberg, Austria, 2007.

[Giu84] Marc Giusti. Some Effectivity Problems in Polynomial Ideal Theory. In John Fitch, editor, *EUROSAM'84 (International Symposium on Symbolic and Algebraic Computation, Cambridge, UK, July 9–11)*, volume 174 of *Lecture Notes in Computer Science*, pages 159–171. Springer, 1984.

[GMW79] Michael J. Gordon, Arthur J. Milner, and Christopher P. Wadsworth. *Edinburgh LCF: A Mechanised Logic of Computation*, volume 78 of *Lecture Notes in Computer Science*. Springer, 1979.

[Gol82] Warren D. Goldfarb. The Undecidability of the Second-Order Unification Problem. *Theoretical Computer Science*, 13(2):225–230, 1982.

[H$^+$15] Thomas C. Hales et al. A Formal Proof of the Kepler Conjecture, 2015. arXiv:1501.02155 [math.MG].

[Har96] John Harrison. HOL Light: A Tutorial Introduction. In Mandayam Srivas and Albert Camilleri, editors, *Proceedings of the 1st International Conference on Formal Methods in Computer-Aided Design (FMCAD'96)*, volume 1166 of *Lecture Notes in Computer Science*, pages 265–269. Springer-Verlag, 1996.

[HB16] Florian Haftmann and Lukas Bulwahn. *Code Generation from Isabelle/HOL Theories*, 2016. Part of the Isabelle documentation, https://isabelle.in.tum.de/dist/Isabelle2016/doc/codegen.pdf.

[HK13] Brian Huffman and Ondřej Kunčar. Lifting and Transfer: A Modular Design for Quotients in Isabelle/HOL. In Georges Gonthier and Michael Norrish, editors, *Certified Programs and Proofs (Proceedings* *of CPP'2013, Melbourne, Australia, December 11–13)*, volume 8307 of *Lecture Notes in Computer Science*, pages 131–146. Springer, 2013.

[HKKN13] Florian Haftmann, Alexander Krauss, Ondřej Kunčar, and Tobias Nipkow. Data Refinement in Isabelle/HOL. In Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie, editors, *Interactive Theorem Proving (Proceedings of ITP'2013, Rennes, France, July 22–26)*, volume 7998 of *Lecture Notes in Computer Science*, pages 100–115. Springer, 2013.

[Hue76] Gérard Huet. Résolution d'équations dans les langages d'ordre $1, 2, \ldots, \omega$, 1976. Thèse d'état, Université Paris 7, Paris, France.

[IM16] Fabian Immler and Alexander Maletzky. Gröbner Bases Theory. *Archive of Formal Proofs*, 2016. http://afp.sf.net/entries/Groebner_Bases.shtml, Formal proof development.

[IP98] Mariano Insa and Franz Pauer. Gröbner Bases in Rings of Differential Operators. In Bruno Buchberger and Franz Winkler, editors, *Gröbner Bases and Applications*, volume 251 of *London Mathematical Society Lecture Note Series*, pages 367–380. Cambridge University Press, 1998.

[JGF09] J. Santiago Jorge, Victor M. Guilas, and Jose L. Freire. Certifying properties of an efficient functional program for computing Gröbner bases. *Journal of Symbolic Computation*, 44(5):571–582, 2009.

[KB04] Temur Kutsia and Bruno Buchberger. Predicate Logic with Sequence Variables and Sequence Function Symbols. In Andrea Asperti, Grzegorz Bancerek, and Andrzej Trybulec, editors, *Mathematical Knowledge Management (Proceedings of the 3rd International Conference, Białowieża, Poland, September 19–21)*, volume 3119 of *Lecture Notes in Computer Science*, pages 205–219. Springer, 2004.

[KMM00] Matt Kaufmann, Panagiotis Manolios, and J. Strother Moore. *Computer-Aided Reasoning: An Approach.* Kluwer Academic Publishers, 2000.

[KR00] Martin Kreuzer and Lorenzo Robbiano. *Computational Commutative Algebra 1.* Springer, 2000.

[Kra09] Alexander Krauss. *Automating Recursive Definitions and Termination Proofs in Higher-Order Logic.* PhD thesis, Institut für Informatik, Technische Universität München, Germany, 2009.

[KRK88] Äbdelilah Kandri-Rody and Deepak Kapur. Computing a Gröbner Basis of a Polynomial Ideal over a Euclidean Domain. *Journal of Symbolic Computation*, 6:37–57, 1988.

[KRW90] Äbdelilah Kandri-Rody and Volker Weispfenning. Non-commutative Gröbner Bases in Algebras of Solvable Type. *Journal of Symbolic Computation*, 9:1–26, 1990.

[Kut07] Temur Kutsia. Solving Equations with Sequence Variables and Sequence Functions. *Journal of Symbolic Computation*, 42(3):352–388, 2007.

[Lau76] Markus Lauer. Canonical representatives for residue classes of a polynomial ideal. Master's thesis, University of Kaiserslautern, 1976.

[LCK$^+$13] Christoph Lange, Marco B. Caminati, Manfred Kerber, Till Mossakowski, Colin Rowat, Makarius Wenzel, and Wolfgang Windsteiger. A Qualitative Comparison of the Suitability of Four Theorem Provers for Basic Auction Theory. In Jacques Carette, David Aspinall, Christoph Lange, Petr Sojka, and Wolfgang Windsteiger, editors, *Intelligent Computer Mathematics (Proceedings of CICM 2013, Bath, UK, July 8–12)*, volume 7961 of *Lecture Notes in Artificial Intelligence*, pages 200–215. Springer, 2013.

[Lev05] Viktor Levandovskyy. *Non-commutative Computer Algebra for Polynomial Algebras: Gröbner bases, Applications and Implementation.* PhD thesis, Universität Kaiserslautern, June 2005.

[Mal14] Alexander Maletzky. Complexity Analysis of the Bivariate Buchberger Algorithm in Theorema. Technical Report 2014-10, Doctoral Program "Computational Mathematics", Johannes Kepler University Linz, October 2014.

[Mal15a] Alexander Maletzky. Automated Reasoning in Reduction Rings Using the Theorema System. In Vladimir P. Gerdt, Wolfram Koepf, Werner M. Seiler, and Evgenii V. Vorozhtsov, editors, *Computer Algebra in Scientific Computing (17th International Workshop, Aachen, Germany, September 18–24)*, volume 9301 of *Lecture Notes in Computer Science*, pages 307–321. Springer, 2015.

[Mal15b] Alexander Maletzky. Exploring Reduction Ring Theory in Theorema. Technical Report 15-11, Research Institute for Symbolic Computation (RISC), Johannes Kepler University Linz, 2015. Also available as DK-report 2015-06.

[Mal15c] Alexander Maletzky. Verifying Buchberger's Algorithm in Reduction Rings. In Tudor Jebelean and Dongming Wang, editors, *Proceedings of PAS'2015 (Program Verification, Automated Debugging, and Symbolic Computation, Beijing, China, October 21–23)*, pages 16–23, 2015. arXiv:1604.08736 [cs.SC].

[Mal16a] Alexander Maletzky. Interactive Proving, Higher-Order Rewriting, and Theory Analysis in Theorema 2.0, 2016. Accepted at ICMS'2016 (5th International Congress on Mathematical Software), Berlin, Germany, July 11–14. http://www.risc.jku.at/publications/download/risc_5282/Extended_Abstract.pdf.

[Mal16b] Alexander Maletzky. Mathematical Theory Exploration in Theorema: Reduction Rings, 2016. Accepted at CICM'2016 (9th Conference on Intelligent Computer Mathematics), Białystok, Poland, July 25–29. arXiv:1602.04339 [cs.SC].

[MB14] Alexander Maletzky and Bruno Buchberger. Complexity Analysis of the Bivariate Buchberger Algorithm in Theorema. In Hoon Hong and Chee Yap, editors, *Mathematical Software – ICMS 2014 (Proceedings of the International Congress on Mathematical Software, Seoul, Korea, August 5–9)*, volume 8592 of *Lecture Notes in Computer Science*, pages 41–48. Springer, 2014.

[MPAR04] Inmaculada Medina-Bulo, Francisco Palomo-Lozano, José A. Alonso-Jiménez, and Jose-Luis Ruiz-Reina. Verified Computer Algebra in ACL2 (Gröbner Bases Computation). In Bruno Buchberger and John A. Campbell, editors, *Artificial Intelligence and Symbolic Computation (Proceedings of AISC'2004, Linz, Austria, September 22–24)*, volume 3249 of *Lecture Notes in Artificial Intelligence*, pages 171–184. Springer, 2004.

[MPR10] Inmaculada Medina-Bulo, Francisco Palomo-Lozano, and Jose-Luis Ruiz-Reina. A verified Common Lisp implementation of Buchberger's algorithm in ACL2. *Journal of Symbolic Computation*, 45(1):96–123, 2010.

[MBS15] André Saint Eudes Mialebama Bouesso and Djiby Sow. Noncommutative Gröbner Bases over Rings. *Communications in Algebra*, 43(2):541–557, 2015.

[Mil79] Robin Milner. LCF: A Way of Doing Proofs with a Machine. In Jiří Bečvář, editor, *Mathematical Foundations of Computer Science 1979*

*(Proceedings of the 8th Symposium, Olomouc, Czechoslovakia, September 3–7)*, volume 74 of *Lecture Notes in Computer Science*, pages 146–159. Springer, 1979.

[MM82] Ernst W. Mayr and Albert R. Meyer. The Complexity of the Word Problems for Commutative Semigroups and Polynomial Ideals. *Advances in Mathematics*, 46(3):305–329, 1982.

[MM84] H. Michael Möller and Ferdinando Mora. Upper and Lower Bounds for the Degree of Gröbner Bases. In John Fitch, editor, *EUROSAM'84 (International Symposium on Symbolic and Algebraic Computation, Cambridge, UK, July 9–11)*, volume 174 of *Lecture Notes in Computer Science*, pages 172–183. Springer, 1984.

[Mor86] Ferdinando Mora. Gröbner bases for Non-commutative Polynomial Rings. In Jaques Calmet, editor, *Algebraic Algorithms and Error-Correcting Codes (3rd International Conference, France, July 15–19, 1985)*, volume 229 of *Lecture Notes in Computer Science*, pages 353–362. Springer, 1986.

[Mor94] Teo Mora. An Introduction to Commutative and Non-Commutative Gröbner Bases. *Theoretical Computer Science*, 134(1):131–173, 1994.

[Nip93] Tobias Nipkow. Functional Unification of Higher-Order Patterns. In Moshe Vardi, editor, *Eighth Annual Symposium on Logic in Computer Science*, pages 64–74. IEEE Computer Society Press, 1993.

[NPW02] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL—A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science.* Springer, 2002.

[Pad96] Vincent Padovani. *Filtrage d'ordre supérieure.* PhD thesis, Université Paris 7, Paris, France, 1996.

[Pan88] Luquan Pan. On the D-bases of Polynomial Ideals over Principal Ideal Domains. *Journal of Symbolic Computation*, 7(1):55–69, 1988.

[Pau90] Lawrence C. Paulson. Isabelle: The next 700 Theorem Provers. In Piergiorgio Odifreddi, editor, *Logic and Computer Science*, pages 361–386. Academic Press, 1990.

[Pau94] Lawrence C. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *Lecture Notes in Computer Science.* Springer, 1994.

[Pau07] Franz Pauer. Gröbner bases with coefficients in rings. *Journal of Symbolic Computation*, 42(11–12):1003–1011, 2007.

[Per01] Henrik Persson. An Integrated Development of Buchberger's Algorithm in Coq. Technical Report 4271, INRIA Sophia Antipolis, September 2001.

[PK05] Florina Piroi and Temur Kutsia. The Theorema Environment for Interactive Proof Development. In Geoff Sutcliffe and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning (Proceedings of LPAR'05, Montego Bay, Jamaica, December 2–6)*, volume 3835 of *Lecture Notes in Artificial Intelligence*, pages 261–275. Springer, 2005.

[Rei06] Birgit Reinert. Gröbner Bases in Function Rings – A Guide for Introducing Reduction Relations to Algebraic Structures. *Journal of Symbolic Computation*, 41(11):1264–1294, 2006.

[Rob85] Lorenzo Robbiano. On the Theory of Graded Structures. *Journal of Symbolic Computation*, 2:138–170, 1985.

[Ros05] Markus Rosenkranz. A New Symbolic Method for Solving Linear Two-Point Boundary Value Problems on the Level of Operators. *Journal of Symbolic Computation*, 39(2):171–199, 2005.

[Sch79] Stuart C. Schaller. *Algorithmic Aspects of Polynomial Residue Class Rings*. PhD thesis, The University of Wisconsin – Madison, 1979.

[Sch06] Christoph Schwarzweller. Gröbner Bases – Theory Refinement in the Mizar System. In Michael Kohlhase, editor, *Mathematical Knowledge Management (4th International Conference, Bremen, Germany, July 15–17)*, volume 3863 of *Lecture Notes in Artificial Intelligence*, pages 299–314. Springer, 2006.

[SIS$^+$11] Yosuke Sato, Shaturo Inoue, Akira Suzuki, Katsusuke Nabeshima, and Ko Sakai. Boolean Gröbner bases. *Journal of Symbolic Computation*, 46(5):622–632, 2011.

[Smu78] Raymond Smullyan. *What is the Name of this Book? The Riddle of Dracula and Other Logical Puzzles*, chapter 14, pages 209–211. Prentice Hall, 1978.

[Spe77] D. A. Spear. A Constructive Approach to Commutative Ring Theory. In R. J. Fateman, editor, *Proceedings of the MACSYMA User's Conference, Berkeley*, pages 369–376. MIT, 1977.

[ST10] Christian Sternagel and René Thiemann. Executable Multivariate Polynomials. *Archive of Formal Proofs*, 2010. http://afp.sf.net/entries/Polynomials.shtml, Formal proof development.

[Ste12] Christian Sternagel. Well-Quasi-Orders. *Archive of Formal Proofs*, 2012. http://afp.sf.net/entries/Well_Quasi_Orders.shtml, Formal proof development.

[Sti85] Sabine Stifter. Computation of Gröbner Bases over the Integers and in General Reduction Rings. Master's thesis, Institut für Mathematik, Johannes Kepler University Linz, Austria, 1985.

[Sti88] Sabine Stifter. A Generalization of Reduction Rings. *Journal of Symbolic Computation*, 4(3):351–364, 1988.

[Sti91] Sabine Stifter. The Reduction Ring Property is Hereditary. *Journal of Algebra*, 140(89–18):399–414, 1991.

[Sti93] Sabine Stifter. Gröbner Bases of Modules over Reduction Rings. *Journal of Algebra*, 159(1):54–63, 1993.

[Sti09] Colin Stirling. Decidability of Higher-Order Matching. *Logical Methods in Computer Science*, 5(3):1–52, 2009.

[Thé01] Laurent Théry. A Machine-Checked Implementation of Buchberger's Algorithm. *Journal of Automated Reasoning*, 26(2):107–137, 2001.

[Tri78] Wolfgang Trinks. Über B. Buchbergers Verfahren, Systeme algebraischer Gleichungen zu lösen (On B. Buchberger's Method of Solving Systems of Algebraic Equations). *Journal of Number Theory*, 10:475–488, 1978.

[VRH$^+$] Andrei Voronkov, Alexandre Riazanov, Krystof Hoder, Laura Kovács, et al. VAMPIRE. http://www.vprover.org/index.cgi.

[WB83] Franz Winkler and Bruno Buchberger. A Criterion for Eliminating Unnecessary Reductions in the Knuth-Bendix Algorithm. In J. Demetrovics, G. Katona, and A. Salomaa, editors, *Proceedings of Algebra and Logic in Computer Science, Győr, Hungary*, volume 42 of *Colloquia Mathematica Societatis Janos Bolyai*, pages 849–869. North Holland, 1983.

[Wei92] Volker Weispfenning. Finite Gröbner bases in Non-Noetherian Skew Polynomial Rings. In P. Wang, editor, *ISSAC'92 (Proceedings of the*

*International Symposium on Symbolic and Algebraic Computation, Nagoya, Japan*, pages 329–334. ACM Press, 1992.

[Wen16] Makarius Wenzel. *The Isabelle/Isar Reference Manual*, 2016. Part of the Isabelle documentation, https://isabelle.in.tum.de/dist/Isabelle2016/doc/isar-ref.pdf.

[Wie99] Tomasz Wierzbicki. Complexity of the Higher Order Matching. In Harald Ganzinger, editor, *Automated Deduction—CADE-16 (Proceedings of the 16th International Conference, Trento, Italy, July 7–10)*, volume 1632 of *Lecture Notes in Computer Science*, pages 82–96. Springer, 1999.

[Wie06] Freek Wiedijk. *The Seventeen Provers of the World*, volume 3600 of *Lecture Notes in Artificial Intelligence.* Springer-Verlag, 2006.

[Win84] Franz Winkler. On the Complexity of the Gröbner-Bases Algorithm over $K[x, y, z]$. In John Fitch, editor, *EUROSAM'84 (International Symposium on Symbolic and Algebraic Computation, Cambridge, UK, July 9–11)*, volume 174 of *Lecture Notes in Computer Science*, pages 184–194. Springer, 1984.

[Win99] Wolfgang Windsteiger. Building Up Hierarchical Mathematical Domains Using Functors in THEOREMA. In Alessandro Armando and Tudor Jebelean, editors, *Proceedings of Calculemus'99, Trento, Italy*, volume 23 of *Electronic Notes in Theoretical Computer Science*, pages 401–419. Elsevier, 1999.

[Win12] Wolfgang Windsteiger. Theorema 2.0: A Graphical User Interface for a Mathematical Assistant System. In Cezary Kaliszyk and Christoph Lueth, editors, *Proceedings of the Workshop on User Interfaces for Theorem Provers (UITP'2012, Bremen, Germany, July 11)*, volume 118 of *Electronic Proceedings in Theoretical Computer Science*, pages 72–82. Open Publishing Association, 2012.

[Win13] Wolfgang Windsteiger. Theorema 2.0: An Open-Source Mathematical Assistant System for Automated and Interactive Reasoning, 2013. Invited talk at PAS'2013 (Second International Seminar on Program Verification, Automated Debugging and Symbolic Computation, Beijing, China, October 23–25). http://www.risc.jku.at/publications/download/risc_4840/talk.pdf.

[Win14] Wolfgang Windsteiger. Theorema 2.0: A System for Mathematical Theory Exploration. In Hoon Hong and Chee Yap, editors, *Mathematical Software – ICMS 2014 (Proceedings of the International Congress on Mathematical Software, Seoul, Korea, August 5–9)*, volume 8592 of *Lecture Notes in Computer Science*, pages 49–52. Springer, 2014.

[Wol] Wolfram Research, Inc. *Mathematica.* http://www.wolfram.com/mathematica/.

[WW15] Manuela Wiesinger-Widi. *Gröbner Bases and Generalized Sylvester Matrices.* PhD thesis, Research Institute for Symbolic Computation, Johannes Kepler University Linz, Austria, 2015. http://epub.jku.at/obvulihs/content/titleinfo/776913.

[Zac78] Gail Zacharias. Generalized Gröbner Bases in Commutative Polynomial Rings, 1978. Bachelor's thesis.

# Curriculum Vitae

Alexander Maletzky

May 2016

## Personal Data

| | |
|---|---|
| First Name | Alexander |
| Last Name | Maletzky |
| Date of Birth | December 1, 1988 |
| Place of Birth | Ried im Innkreis |
| Nationality | Austria |

## Contact Data

| | |
|---|---|
| Address | Research Institute for Symbolic Computation<br>Johannes Kepler University Linz<br>Altenberger Straße 69<br>4040 Linz, Austria |
| Telephone | +43 732 2468 9936 |
| e-Mail | alexander.maletzky@risc.jku.at |
| Web | https://www.risc.jku.at/home/amaletzk |

## Education

| | |
|---|---|
| 2013–present | PhD studies in the FWF Doctoral Program "Computational Mathematics", JKU Linz |
| 2011–2013 | Master degree studies in Computer Mathematics, JKU Linz |
| 2008–2011 | Bachelor degree studies in Technical Mathematics, JKU Linz |
| 2007 | High School Diploma (Matura), BRG Ried im Innkreis |

## Research Stays

| | |
|---|---|
| Oct. 2015 – Jan. 2016 | Chair for Logic and Verification (Prof. Tobias Nipkow), Technical University Munich, Germany |

## Teaching Experience

- Exercises for "Analysis for Computer Scientists", JKU Linz, October 2014 – January 2015
- "Programming in Mathematica" (intensive course) for students of the International Master's Program Informatics, JKU Linz, February 2014
- "Programming in Mathematica" for students of the International Master's Program Informatics, JKU Linz, October 2013 – January 2014

## List of Publications

1. Alexander Maletzky. Mathematical Theory Exploration in Theorema: Reduction Rings. Accepted at CICM'2016 (9th Conference on Intelligent Computer Mathematics), Białystok, Poland, July 25–29, 2016. arXiv:1602.04339 [cs.SC].

2. Alexander Maletzky. Interactive Proving, Higher-Order Rewriting, and Theory Analysis in Theorema 2.0. Accepted at ICMS'2016 (5th International Congress on Mathematical Software), Berlin, Germany, July 11–14, 2016. http://www.risc.jku.at/publications/download/risc_5282/Extended_Abstract.pdf.

3. Bruno Buchberger, Tudor Jebelean, Temur Kutsia, Alexander Maletzky and Wolfgang Windsteiger. Theorema 2.0: Computer-Assisted Natural-Style Mathematics. *Journal of Formalized Reasoning*, **9**(1):149–185, 2016. ISSN 1972-5787, doi: 10.6092/issn.1972-5787/4568.

4. Alexander Maletzky. Verifying Buchberger's Algorithm in Reduction Rings. In Tudor Jebelean and Dongming Wang, editors, *Proceedings of PAS'2015 (4th International Seminar on Program Verification, Automated Debugging and Symbolic Computation, Beijing, China, October 21–23, 2015)*, pp. 16–23. arXiv:1604.08736 [cs.SC].

5. Markus Rosenkranz, Jane Liu, Alexander Maletzky and Bruno Buchberger. Two-Point Boundary Problems with One Mild Singularity and an Application to Graded Kirchhoff Plates. In Vladimir P. Gerdt, Wolfram Koepf,

Werner M. Seiler and Evgenii Vorozhtsov, editors, *Computer Algebra in Scientific Computing (17th International Workshop, Aachen, Germany, September 14–18)*. Volume 9301 of *Lecture Notes in Computer Science*, pp. 406–423, Springer, 2015. doi: 10.1007/978-3-319-24021-3_30.

6. Alexander Maletzky. Automated Reasoning in Reduction Rings using the Theorema System. In Vladimir P. Gerdt, Wolfram Koepf, Werner M. Seiler and Evgenii Vorozhtsov, editors, *Computer Algebra in Scientific Computing (17th International Workshop, Aachen, Germany, September 14–18)*. Volume 9301 of *Lecture Notes in Computer Science*, pp. 305–319, Springer, 2015. doi: 10.1007/978-3-319-24021-3_23.

7. Alexander Maletzky and Bruno Buchberger. Complexity Analysis of the Bivariate Buchberger Algorithm in Theorema. In Hoon Hong and Chee Yap, editors, *Mathematical Software – ICMS 2014 (The 4th International Congress on Mathematical Software, Seoul, Korea, August 5–9)*. Volume 8592 of *Lecture Notes in Computer Science*, pp. 41–48, Springer, 2014. doi: 10.1007/978-3-662-44199-2_8.

8. Bruno Buchberger and Alexander Maletzky. Gröbner Bases in Theorema. In Hoon Hong and Chee Yap, editors, *Mathematical Software – ICMS 2014 (The 4th International Congress on Mathematical Software, Seoul, Korea, August 5–9)*. Volume 8592 of *Lecture Notes in Computer Science*, pp. 374–381, Springer, 2014. doi: 10.1007/978-3-662-44199-2_58.