Vom Fachbereich Mathematik der Technischen Universität Kaiserslautern
zur Verleihung des akademischen Grades Doktor der Naturwissenschaften
(Doctor rerum naturalium, Dr. rer. nat.) genehmigte Dissertation

Algorithms in SINGULAR: Parallelization, Syzygies, and Singularities

Andreas Steenpaß

1. Gutachter: Prof. Dr. Wolfram Decker
2. Gutachter: Prof. Dr. Dorin Popescu

Vollzug der Promotion: 24. Juli 2014

D 386

Preface

The subject of this thesis is located in the field of algorithmic commutative algebra and algebraic geometry. This area offers a rich interplay between the mathematical theory on the one hand and the implementation of the derived algorithms on the other hand. By the use of computers, the theory has become accessible for experiments and, in turn, the demand for faster algorithms has triggered new theoretical developments. The aim of this thesis is to provide a substantial progress on both, the theoretical and the implementational side.

The theoretical results of this thesis include, to mention a few highlights:

- a new local-to-global approach to normalization, see Section 2.3;
- a new method to verify that a given ideal is indeed the intersection of a given set of ideals, see Proposition 3.2.7 and Corollary 3.2.8;
- an in-depth analysis of Schreyer's syzygy algorithm, see Sections 4.2 and 4.3;
- the algorithmic classification of the simple real singularities, see Section 5.4;
- a theorem on the shape of the automorphisms between the normal forms of certain unimodal real singularities, see Theorem 6.4.3;
- an explicit description of the structure of the equivalence classes for the unimodal real singularities of corank 2, see Section 6.6.

There are more than twenty algorithms discussed in this thesis, see the List of Algorithms on page xvii. All of them are implemented in the open source computer algebra system SINGULAR [39]. In fact, a multitude of SINGULAR libraries have been written or rewritten as part of or in the context of this thesis. For details, please refer to the List of Contributions on page vii. Together, these libraries contain more than 6400 lines of code. One highlight of the implementational part of this thesis is SINGULAR's new parallel framework which allows SINGULAR users and authors of libraries to take advantage of the computational power of modern multicore processors via a technique called parallelization.

Most chapters of this thesis have emerged from joint research projects with various colleagues. This is also reflected in the variety of the discussed

topics. Several chapters are, or will be, published separately in a slightly different form, see the List of Contributions for details. Some of the research projects have led to new open problems and will thus be continued, based on the results presented here.

This thesis consists of three parts: The first part is devoted to parallelization. New algorithms for the computation of syzygies are presented in the second part. Finally, the third part deals with the algorithmic classification of real singularities.

Part I comprises the first three chapters. SINGULAR's parallel framework, which is discussed in Chapter 1, is a sophisticated tool for parallelization and has been developed as part of this thesis. It is specifically designed for applications in mathematical research and serves as the technical basis for the implementation of the parallel algorithms in the two subsequent chapters. After a general introduction to parallelization in Section 1.1, the features of this framework are described in Section 1.2. The parallel framework has proven to be stable and of great benefit for applications in research. However, we plan to improve it even further. An outlook on future plans is given in Subsection 1.2.8.

Parallel algorithms for the normalization of a reduced affine algebra A over a perfect field are presented in Chapter 2. Starting from the algorithm of Greuel et al. [17], we propose two approaches which can also be combined. For the local-to-global approach presented in Section 2.3, we stratify the singular locus $\mathrm{Sing}(A)$ of A, compute the normalization locally at each stratum and finally reconstruct the normalization of A from the local results. Modular methods, which have proven to be a powerful tool for the computation of standard bases (cf. [2, 22]), are applied to both the global and the local-to-global normalization algorithm in Section 2.4. The timings given in Section 2.5 show that, in most examples, the new algorithms outperform the algorithm of Greuel et al. by far, even if they are not run in parallel.

In Chapter 3, we present a parallel version of the algorithm of Gianni, Trager, and Zacharias [14] for primary decomposition. For the parallelization of this algorithm, we combine four different approaches. First, whenever we encounter a zero-dimensional ideal, we use the modular algorithm of Idrees, Pfister, and Steidel [22] to compute the associated primes. The corresponding primary ideals are then extracted from these via saturation. Second, we also use modular methods for the computations of standard bases which occur at the intermediate steps. Third, we use an innovative fast method to verify that the result is indeed a primary decomposition of the input ideal, see Proposition 3.2.7 and Corollary 3.2.8. This allows us to skip the verification step at each of the intermediate modular computations. Finally, we parallelize the trivially parallelizable parts such as the extraction of the primary components mentioned above. These four approaches are described in detail in Section 3.2. Timings are given in Section 3.3. Especially for large examples, the parallel algorithm is faster then the sequential one and scales

well with the number of processor cores. However, we also found examples where we could not achieve a considerable speedup. We therefore intend to continue this project with an investigation of these examples.

Part II of this thesis consists of Chapter 4 only. Based on an in-depth analysis of Schreyer's algorithm [34, 33, 11] which is stated in Section 4.2, we propose new algorithms for the computation of syzygies. Here, the main ideas are that we may leave out so-called "lower order terms" which do not contribute to the result of the algorithm, that we do not need to order the terms of certain module elements which occur at intermediate steps, and that some partial results can be cached and reused. These ideas are explained in detail in Section 4.3. An extensive example is given in Section 4.4. We expect that the new algorithms are considerably faster, especially for large examples. However, this research project is still ongoing because the implementation of the new algorithms is not yet completely finished, so we do not present any timings here.

The last two chapters make up Part III which treats the algorithmic classification of singularities over the real numbers, based on the classification theorems of Arnold et al. [4]. Chapter 5 covers the Splitting Lemma and the simple singularities. Section 5.2 contains some prerequisites for the classification. In particular, we prove two results regarding the factorization of homogeneous polynomials over $\mathbb{R}$ and $\mathbb{Q}$ (see Proposition 5.2.8 and Lemma 5.2.9) which are important for the algorithmic aspect. Hereafter, we present a real version of the Splitting Lemma in Section 5.3. By applying the Splitting Lemma, the classification of a given real singularity can be reduced to the classification of the residual part which, for the simple singularities, is treated in Section 5.4. In addition to the algorithms, we also provide insights into how real and complex singularities are related geometrically such as in Remark 5.4.1.

For the unimodal real singularities, the situation is more complex because the normal forms given by Arnold et al. [4] are not always uniquely determined. In Chapter 6, we explicitly describe the structure of the equivalence classes of the unimodal real singularities of corank 2. The equivalences are given by automorphisms whose shape is restricted by a theorem proven in Section 6.4. Based on this theorem, we explain in detail how the structure of the equivalence classes can be computed using SINGULAR in Section 6.5. The results are presented in concise form in Section 6.6. Finally, we point out some interesting aspects of these results in Section 6.7. The probably most surprising outcome is that the real singularity type J_{10}^- is actually redundant.

Financial Support

This thesis was partly supported by the German National Academic Foundation (Studienstiftung des deutschen Volkes) via a PhD scholarship.

Acknowledgements

There is a variety of persons to whom I am indebted for supporting the work on this thesis. In particular, I would like to thank, in alphabetical order:

Petra Bäsell
Mohamed Barakat
Reimer Behrends
Wolfram Decker
Christian Eder
Claus Fieker
Gert-Martin Greuel
Christoph Lossen
Magdaleen Marais
Gerhard Pfister
Hans Schönemann
Frank-Olaf Schreyer
Christa Schulte

List of Contributions

Chapter 1 has been written exclusively by the author of this thesis.

Chapter 2 is the result of a joint project with J. Böhm, W. Decker, S. Laplagne, G. Pfister, and S. Steidel. A slightly different version of this chapter has been published in the Journal of Symbolic Computation, cf. [7].

Chapter 3 is the result of a joint project with G. Pfister. We intend to publish this chapter separately.

Chapter 4 is the result of a joint project with B. Eröcal, O. Motsak, and F.-O. Schreyer. We intend to publish this chapter separately.

Chapter 5 is the result of a joint project with M. Marais. It has been accepted for publication in the Journal of Symbolic Computation, cf. [30].

Chapter 6 is the result of a joint project with M. Marais. It has been submitted to the Journal of Symbolic Computation, cf. [31].

The following SINGULAR libraries have been written or rewritten in the context of this thesis:

`assprimeszerodim.lib` **[46]** for computing the associated prime ideals of a zero-dimensional ideal, with N. Idrees, G. Pfister, and S. Steidel;

`locnormal.lib` **[42]** for computing the normalization of affine domains using local methods, with J. Böhm, W. Decker, S. Laplagne, G. Pfister, and S. Steidel;

`modnormal.lib` **[43]** for computing the normalization of affine domains using modular methods, with J. Böhm, W. Decker, S. Laplagne, G. Pfister, and S. Steidel;

`modquotient.lib` **[51]** for computing quotients and saturations of ideals using modular methods;

`modstd.lib` **[45]** for computing Gröbner bases using modular methods, with A. Hashemi, G. Pfister, H. Schönemann, and S. Steidel;

`modular.lib` **[52]** providing abstraction layer for modular techniques;

`parallel.lib` **[53]** providing an abstraction layer for parallel skeletons;

`primdec_parallel.lib` **[50]** for parallel primary decomposition of ideals, with code from `primdec.lib` [44];

`realclassify.lib` **[48]** for the classification of singularities over the real numbers, with M. Marais;

`resources.lib` **[54]** for managing computational resources;

`tasks.lib` **[55]** providing a parallel framework based on tasks.

Together, these libraries contain more than 6400 lines of code. In addition to this, more than 1000 lines of code have been contributed to other SINGULAR libraries and to the SINGULAR kernel by the author in the context of this thesis.

Contents

List of Figures

List of Tables

List of Algorithms

Part I

Parallelization

Chapter 1

SINGULAR's Parallel Framework

1.1 General Introduction to Parallelization

In 1965, computer scientist Gordon E. Moore noticed that the development of microelectronic devices had been exponential over the past few years, and he predicted that the rate of increase would remain "nearly constant for at least 10 years" [32]. This forecast has proven to be quite precise until today and it is expected to remain valid in the near future.

For a long period, this development was partly achieved by increasing the clock rate of microprocessors. The IBM PC introduced in 1981 was shipped with the Intel 8088 microprocessor which ran at 4.77 MHz. Around the year 2000, desktop computers reached the milestone of 1 GHz. When higher clock rates became harder to attain in the middle of the 2000s due to technical reasons, mainly evolution of heat, the performance improvement of microprocessors continued by other means. One of these means are multicore processors which were introduced to the mass consumer market at about this time. As of 2014, most desktop computers and notebooks are shipped with multicore processors.

In order to take advantage of the enormous potential of modern processors with several cores, the software must split the calculation into parts which can be run in parallel on the different cores and whose results are then joined together. This technique is called parallelization. In some cases, such as for video encoding software, there are obvious ways how this can be done. A video can be split into parts which can be encoded (almost) independently of each other.

For mathematical software, parallelization is often more challenging, but it has also proven to be particularly fruitful for research. Some recent results in computational mathematics such as the computation of a standard basis of the ideal Cyclic 9 in characteristic 0 (cf. [22]) would not have been possible without this technique. But the importance of the technical development in this area for mathematical research goes beyond setting new benchmark

records. First of all, the question how a mathematical algorithm can be parallelized is a research problem on its own. Which parts of the computation are independent of each other? Which other methods yield the same result, but are more suitable for parallelization? These questions have also triggered new developments in adjacent areas of mathematical research such as the in-depth analysis of modular methods. Two examples for this are highlighted in Chapters 2 and 3. In some cases, the search for parallel algorithms led to new approaches which are sometimes faster than the sequential approach, even if they are not computed in parallel, see Section 2.5 for examples.

Parallelization allows us to tackle research problems which have been previously unsolvable. Areas such as cryptography and phylogenetics provide examples for computational challenges which are far out of reach without parallel software. The technical development over the past few years makes these areas of theoretical research accessible for experiments, provided that the employed software can benefit from the power of multicore processors.

On the other hand, there are a few obstacles for the usage of parallelization which should not be silently ignored. Parallel programming is generally difficult to handle for end-users which have often very little or no knowledge in this field. In order to prove the correctness of a parallel program, that is, a program where different parts are run in parallel, one has to consider every single branch and every possible interaction between those parts because the program flow often depends on race conditions and is thus unpredictible. This makes parallel programs in general by far more complex than sequential ones. If errors occur, they are typically very difficult to track down. They might only occur at one in a million times due to race conditions. Probably due to these difficulties, there is no well-established standard framework for parallel programming which would provide simple ready-to-use high-level directives. Thus parallelizing a sequential program generally requires deep modifications. However, the software may also benefit from this process. In writing parallel code, the programmer is obliged to modularize the software such that every part plays a distinct role and interacts with the other parts in a well-defined way.

Parallelization also has its limits. First, parallel programs always involve a certain amount of overhead in comparison to sequential ones. This overhead mainly consists in the communication between the concurrent parts and in the scheduling of the computational tasks. Therefore the communication, however it is done in practice, should be both kept to a minimum and be very efficient. The scheduling should be simple, but still efficient enough to yield a near-optimal allocation of the available resources.

Second, there are even theoretical limits for parallelization. The following theorem, known as Amdahl's law, describes the influence of non-parallelizable portions of a program on the maximum possible speedup:

Theorem 1.1.1 ([1]). *If r_s and r_p denote the sequential and parallel portion*

of a program, respectively, with $r_s + r_p = 1$, then the speedup $S(N)$ which can be achieved for a fixed problem size under ideal circumstances by using N processor cores is

$$S(N) = \frac{1}{r_s + \frac{r_p}{N}}\,.$$

Therefore the maximum possible speedup that can be achieved by using an arbitrary number of processor cores is

$$S = \lim_{n\to\infty} S(N) = \frac{1}{r_s}\,.$$

Although the proof of this theorem is straightforward, it has surprising consequences: If 90 % of a program can be fully parallelized and scale arbitraryly well with the number of cores, but only 10 % remain sequential, then the maximum possible speedup is 10. Even with eight processor cores, the speedup is restricted to little more than 4.7, so we lose about 40 % compared to a linear speedup of 8.

On the other hand, if we replace the condition that the problem size is fixed by the assumption that the run time is constant, and if we additionally assume that "the amount of work that can be done in parallel varies linearly with the number of processors" [20], we get Gustafson's law:

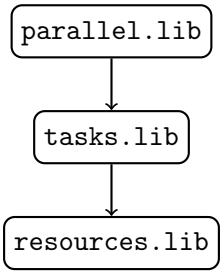
Theorem 1.1.2. *If s and p represent the serial and parallel portion of time, respectively, spent on a parallel system with N cores, with $s + p = 1$, then a single processor core would require time $s + pN$ to perform the same task. Therefore, the "scaled speedup" $S'(N)$ is*

$$S'(N) = \frac{s + pN}{s + p} = pN + s\,.$$

Since the serial time s can be neglected, we get $S'(N) \approx pN$, that is, the scaled speedup grows almost linearly for increasing N where the linear factor is p. If we assume $s = 0.1$ and $N = 8$ as above, then Gustafson's law predicts a scaled speedup of 7.3. For the parallel framework and its applications discussed in this thesis, we might expect speedups which lie somewhere in the range between the predictions of Amdahl's and Gustafson's law.

On the technical level, we have to distinguish between parallelization where the parallel parts are run in separate processes and parallelization where they are run in different threads within the same process. On most operating systems, the main difference between threads and processes is that multiple threads may share the same memory while each process has its own memory address space by default. Thus data does not need to be transferred between threads within the same process. Because of this, thread-based parallelization is dominated by synchronisation overhead whereas parallelization based on processes is dominated by copying overhead (cf. [26, 23]). While the overhead can be neglegted for coarse-grained parallelization where the

Figure 1.1: Dependencies in SINGULAR's parallel framework



whole computation is split up into long-running tasks, it is especially important for fine-grained parallelization where each parallel task takes only a few milliseconds.

1.2 Implementation in SINGULAR

The main goal of the framework described in this chapter is to provide parallel functionality for users of the software SINGULAR.

SINGULAR [39] is a computer algebra system for polynomial equations whose main areas of application are commutative and non-commutative algebra, algebraic geometry, and singularity theory. The software is open-source under the GNU General Public Licence (version 2 or version 3). SINGULAR admits an interpreter which has its own programming language with a C-like syntax. The mathematical functionality is provided at two different levels: Core algorithms such as the standard basis algorithm or polynomial factorization are implemented in the SINGULAR kernel which is mainly written in C and C++. The kernel can be extended by libraries written in SINGULAR's own programming language. The functionality provided by the numerous libraries includes normalization (see Chapter 2), primary decomposition (see Chapter 3), and the classification of singularities (see Chapter 5).

SINGULAR's parallel framework has been developed as part of this thesis. It is specifically designed for applications in research and serves as the technical basis for the implementation of the parallel algorithms discussed in Chapters 2 and 3. The framework consists of the three SINGULAR libraries `resources.lib` [54], `tasks.lib` [55], and `parallel.lib` [53]. Figure 1.1 shows how these libraries depend on each other.

We highlight the most important features of SINGULAR's parallel framework in the following subsections. Please note that the current implementation of these features may change in the future due to changes in other parts of SINGULAR and is therefore not explained here. In particular, the parallel parts are currently run in separate processes whereas in the future

it will also be possible to run them in threads within the same process. For technical details of the framework, please refer to the extensive comments in the source code. Details on the user interface can be found in the SINGULAR manual.

1.2.1 Task-Oriented Design

The design of the parallel framework is strictly task-oriented, that is, in order to parallelize a computation, it is divided into user-defined chunks called tasks which are independent of each other and can be performed simultaneously. This is opposed to data parallelism where a single command is executed on different data sets.

To reflect the task-oriented design in the SINGULAR interpreter, a new data type `task` has been introduced. An object of type `task` can be created by a command given as a string and a list of arguments as in the following example:

```
> task t = "intersect", list(I, J);
```

Here, an object `t` of type `task` is defined as the task to compute the intersection of `I` and `J`. Tasks may differ completely in both their running time and the kind of computation. In particular, the granularity of the tasks is left to the user.

1.2.2 User Interface

The user interface of the parallel framework provides easy-to-use single-line commands for SINGULAR users and authors of libraries who have often no or very little knowledge about parallelization. Using this framework, they are able to parallelize their code without having to consider issues such as data formats, scheduling, and inter-process communication.

The library `tasks.lib` is an interface for the basic handling of parallel tasks as in the following example:

```
> LIB "tasks.lib";
> ring R = 0, (x,y), lp;
> ideal I1 = x2-3y2, 2xy;
> ideal I2 = 3x2+y3, 3xy2;
> task t1 = "std", list(I1);
> task t2 = "std", list(I2);
> startTasks(t1, t2);
> waitAllTasks(t1, t2);
> getResult(t1);
_[1]=y3
_[2]=xy
_[3]=x2-3y2
```

```
> getResult(t2);
_[1]=y5
_[2]=xy2
_[3]=3x2+y3
```

Here, standard bases of the two ideals $\langle x^2 - 3y^2, 2xy\rangle$ and $\langle 3x^2 + y^3, 3xy^2\rangle$ in $\mathbb{Q}[x, y]$ are computed in parallel. The library `tasks.lib` provides, among others, the following commands:

- `copyTask`, `compareTask`, and `printTask`;
- `startTasks`, `stopTask`, `waitTasks`, `waitAllTasks`, and `pollTask`;
- `getCommand`, `getArguments`, `getResult`, and `getState`.

For an exhaustive list of commands and for detailed information on their usage, please refer to the manual.

Based on `tasks.lib`, the SINGULAR library `parallel.lib` implements several parallel skeletons. A parallel skeleton is a programming pattern for performing several tasks in parallel. For example, it is a common programming pattern to start some tasks in parallel and to wait for all of them to finish as in the example above. Using the command `parallelWaitAll` from `parallel.lib`, the example can be rewritten as follows:

```
> LIB "parallel.lib";
> ring R = 0, (x,y), lp;
> ideal I1 = x2-3y2, 2xy;
> ideal I2 = 3x2+y3, 3xy2;
> parallelWaitAll("std", list( list(I1), list(I2) ));
[1]:
   _[1]=y3
   _[2]=xy
   _[3]=x2-3y2
[2]:
   _[1]=y5
   _[2]=xy2
   _[3]=3x2+y3
```

The following commands are implemented in the library `parallel.lib` so far:

`parallelWaitN`	perform several tasks in parallel and wait for a certain number of them to finish
`parallelWaitFirst`	perform several tasks in parallel and wait for the first of them to finish
`parallelWaitAll`	perform several tasks in parallel and wait for all of them to finish

`parallelTestAND`	run several tests in parallel and determine if they all succeed
`parallelTestOR`	run several tests in parallel and determine if any of them succeeds

This list might be extended in the future. Again, please refer to the manual for detailed instructions on how to use these commands.

1.2.3 Transparency and Responsiveness

The parallel framework is transparent in the sense that performing tasks in parallel yields the same results as executing them sequentially. In the parallel examples above, we get the same standard bases as with the following sequential code:

```
> ring R = 0, (x,y), lp;
> ideal I1 = x2-3y2, 2xy;
> ideal I2 = 3x2+y3, 3xy2;
> std(I1);
_[1]=y3
_[2]=xy
_[3]=x2-3y2
> std(I2);
_[1]=y5
_[2]=xy2
_[3]=3x2+y3
```

Transparency is especially important in view of the fact that SINGULAR is not state-less, that is, the same command may yield different results depending on circumstances such as global options, defined variables, or the current basering. For example, the result of the SINGULAR command `std()` depends on the option `redSB`: If this option is set, `std()` returns a reduced standard basis whereas the result is not necessarily reduced if `redSB` is not set.

For a task-based parallel framework, this has be taken into account. When a task is started in the SINGULAR interpreter, it waits in the background until enough computational resources become avaiblable. As soon as this is the case, the parallel framework starts to execute the task. In the meanwhile, the state of the SINGULAR process may have changed in a way which influences the result of the task. In the parallel framework, however, every task yields the same result as if it had been executed right at the point where it was started in the SINGULAR interpreter. In the parallel examples above, this point is given by the commands `startTasks` and `parallelWaitAll`, respectively.

SINGULAR's parallel framework is optimized for responsiveness in the sense that each command finishes (almost) immediately if it does not require to wait for a task to complete. Responsiveness is related to transparency because of implementational issues. For example, when some tasks are started via the command `startTasks`, an easy possibility to achieve transparency would be to block the execution of further interpreter commands until all tasks are completed. However, such an implementation of `startTasks` would not be responsive.

1.2.4 Recursive Usage

SINGULAR's parallel framework allows the user to define parallel tasks recursively within other tasks. Algorithms can thus be parallelized even on different levels. The scheduling of the tasks in the resulting tree structure is handled by the framework and the user does not need to worry about it. As an example for parallelization on different levels, the two standard bases from the examples above can be computed using modular methods (cf. Chapters 2 and 3) as follows:

```
> ring R = 0, (x,y), lp;
> ideal I1 = x2-3y2, 2xy;
> ideal I2 = 3x2+y3, 3xy2;
> parallelWaitAll("modStd", list( list(I1), list(I2) ));
[1]:
   _[1]=y3
   _[2]=xy
   _[3]=x2-3y2
[2]:
   _[1]=y5
   _[2]=xy2
   _[3]=x2+1/3y3
```

In this SINGULAR session, the command `modStd` from the library `modstd.lib` [45] is applied to both ideals `I1` and `I2` in parallel, and both standard bases computations are, in turn, further parallelized using commands from the libraries `tasks.lib` and `parallel.lib`.

The recursive usage of the parallel framework is especially important for parallelization within SINGULAR libraries. Thanks to this feature, a library can be parallelized regardless of whether or not other libraries depending on that library use parallel functionality as well.

1.2.5 Resource Management

The parallel trees arising from recursive usage of the parallel framework as explained in the previous subsection may grow very fast. However, the

computational resources are limited. Too many tasks running at the same time may overload the compute server and, in some cases, a SINGULAR user may want to restrict the computational resources taken up by SINGULAR explicitly. For this purpose, the library `resources.lib` allows the user to set the maximal number of simultaneously running tasks as in the following example:

```
> LIB "resources.lib";
> setcores(4);
4
> ;   // [parallel computations on four processor cores]
> addcores(2);
6
> getcores();
6
> ;   // [parallel computations on six processor cores]
```

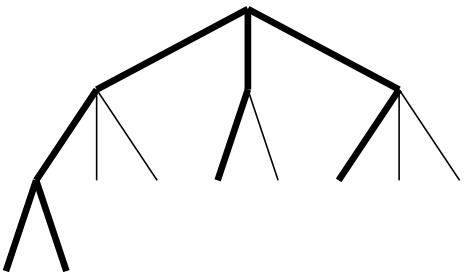
SINGULAR's parallel framework respects these settings, that is, it does not execute more tasks at the same time than permitted by these commands. If the tasks are defined recursively, then this restriction applies to the whole parallel tree. The default value for the maximal number of simultaneously running tasks is the number of processor cores of the compute server which SINGULAR is running on. Again, please refer to the manual for a detailed description of the commands provided by `resources.lib`. The functionality of this library might be extended in the future to support, for example, distributed parallel computing on several compute servers.

The parallel framework automatically decides in which order the tasks are executed if there are not enough resources available to run all of them at the same time. This scheduling tends to be optimal in the sense that the concurrently running tasks are equally distributed to the main branches of the parallel tree. For example, when the maximal number of simultaneously running tasks is set to four, and the parallel tree looks like the graph in Figure 1.2, then the parallel framework tends to execute the tasks along the bold branches first.

1.2.6 Data Transfer

The handling of data within the parallel framework is optimized for keeping the amount of transferred or copied data as little as possible. For this purpose, the internal data structures of the SINGULAR type `task` rely on a pointer-like concept. That is, the data associated to objects of type `task` such as the command, the arguments to that command, or the result of a task, are stored in an internal list and only transferred or copied if necessary. This is important because objects in SINGULAR may become very large. In the current implementation of the framework, tasks are executed in separate

Figure 1.2: Scheduling in a parallel tree



processes which are generated by forking (or, in other words, copying) the parent process. Thus the input data for tasks does not need to be transferred or copied.

The technical basis for data transfer within the parallel framework will change in the future when tasks can be executed in threads within the same process. This is expected to speed up the data transfer even further.

1.2.7 Performance and Stability

Parallel programs always involve a certain amount of overhead in comparison to their sequential counterparts. In SINGULAR's parallel framework, this overhead is kept to a minimum in order to maximize the performance. This is achieved by optimizing the data transfer as explained in the previous subsection and by keeping the scheduling as simple as possible. The performance of the framework is especially crucial for fine-grained parallelization, that is, for applications where the running time of the tasks is relatively short. An efficient framework allows us to achieve considerable speedups even in situations where the running time of each tasks is in the range of a few milliseconds.

To measure the performance of the current implementation of SINGULAR's parallel framework, we use parallel trees where each task does nothing else than to start a certain number of subtasks and to wrap the result of these subtasks in a list. Thus the result of the task at the root node of the tree is a nested list which reflects the tree's structure. Table 1.1 shows timings for several trees. The number of nodes and leaves in each tree is calculated from the number of children per node and the number of levels. These examples were computed on a compute server with an eight-core Intel® Core™ i7 CPU (4 physical cores plus Hyper-threading) and 16 GB RAM running under Linux 3.14.

The timings show that the parallel framework tends to perform better on both, trees with more nodes and trees with more children per node. Only

Table 1.1: Performance of SINGULAR's parallel framework

Parallel tree				Time in ms	ms/node
Children per node	Levels	Nodes	Leaves		
1	5	6	1	40	6.67
1	10	11	1	88	8.00
2	2	7	4	17	2.43
2	5	63	32	62	0.98
2	8	511	256	425	0.83
2	10	2047	1024	1762	0.86
2	12	8191	4096	7526	0.92
3	3	40	27	35	0.88
3	5	364	243	258	0.71
3	7	3280	2187	2089	0.64
5	2	31	25	27	0.87
5	4	781	625	450	0.58
5	6	19531	15625	10040	0.51
10	1	11	10	30	2.73
10	2	111	100	64	0.58
10	3	1111	1000	561	0.50
10	4	11111	10000	5256	0.47

for trees with only one or two children per node, increasing the number of levels may have a negative impact on the average running time per node. Note that there is no parallelization going on in the corner case with only one child per node.

As a rule of thumb, using SINGULAR's parallel framework pays off for applications where the average running time per task is about 10 ms or more, provided that the compute server has several processor cores.

In addition, the framework has proven to be very stable. Experiments have shown that it works reliably even for parallel trees with twenty levels and two children per node, that is, with more than one million leaves and two million nodes.

1.2.8 Outlook

In its current implementation, SINGULAR's parallel framework is already a very sophisticated tool for parallelization which includes all the features described in the previous subsections. It is thus well adapted to the needs of SINGULAR users as well as authors of SINGULAR libraries and provides a great benefit for applications in mathematical research, cf. Chapters 2 and 3.

However, there are plans to improve the framework even further in the future. Its design is in principle compatible with other software systems for mathematical research, notably GAP [40]. Thus both systems can be connected for joint parallel computations.

Currently, parallel tasks run in separate processes. In the future, it will be possible to run tasks in separate threads within the same process. This yields a much better performance and is thus an important condition for fine-grained parallelization as explained above. It also allows us to define more explicit scheduling rules without performance impact.

Finally, the management of the computational resources will be extended to distributed computing, that is, parallel computations running on several compute servers.

Chapter 2

Parallel Algorithms for Normalization

Given a reduced affine algebra A over a perfect field K, we present parallel algorithms to compute the normalization $\overline{A}$ of A. Our starting point is the algorithm of Greuel et al. [17], which is an improvement of de Jong's algorithm, see [24, 12]. First, we propose to stratify the singular locus $\mathrm{Sing}(A)$ in a way which is compatible with normalization, apply a local version of the normalization algorithm at each stratum, and find $\overline{A}$ by putting the local results together. Second, in the case where $K = \mathbb{Q}$ is the field of rationals, we propose modular versions of the global and local-to-global algorithms. We have implemented our algorithms in the computer algebra system SINGULAR and compare their performance with that of the algorithm of Greuel et al. [17]. In the case where $K = \mathbb{Q}$, we also discuss the use of modular computations of Gröbner bases, radicals, and primary decompositions. We point out that in most examples, the new algorithms outperform the algorithm of Greuel et al. [17] by far, even if we do not run them in parallel.

2.1 Introduction

Normalization is an important concept in commutative algebra, with applications in algebraic geometry and singularity theory. We are interested in computing the normalization $\overline{A}$ of a reduced affine K-algebra A, where K is a perfect field. For this, a number of algorithms have been proposed, but not all of them are of practical interest (see the historical account in [17]). A milestone is de Jong's algorithm, see [24, 12], which is based on the normality criterion of Grauert and Remmert [15], and which has been implemented in SINGULAR (see [39]), MACAULAY2 (see [41]), and MAGMA (see [9]). The algorithm of Greuel, Laplagne, and Seelisch [17] (*GLS normalization algorithm* for short), which is also based on the Grauert and Remmert criterion, is an improvement of de Jong's algorithm. It is implemented in SINGULAR.

The algorithm proposed in [29] and [37] is designed for the characteristic p case. It is implemented in SINGULAR and MACAULAY2 and works well for small p.

Our objective in this chapter is to present parallel versions of the GLS normalization algorithm in that we reduce the general problem to computational problems which are easier and do not depend on each other. It turns out that in most cases, the new algorithms outperform the GLS algorithm by far, even if we do not run them in parallel.

We start in Section 2.2 by reviewing the basic ideas of the GLS algorithm. In particular, we recall the normality criterion of Grauert and Remmert. In Section 2.3, we present a local version of the normality criterion which applies to a stratification of the singular locus $\mathrm{Sing}(A)$ of A. This allows us to find $\overline{A}$ by a local-to-global approach. Section 2.4 contains a discussion of modular methods for the GLS algorithm and its local-to-global version. Timings are presented in Section 2.5.

2.2 The GLS Normalization Algorithm

Referring to [17] and [19] for details and proofs, we sketch the GLS normalization algorithm. We begin with some general remarks. For these, A may be any reduced Noetherian ring.

Definition 2.2.1. Let A be a reduced Noetherian ring. The *normalization* of A, written $\overline{A}$, is the integral closure of A in its total ring of fractions $\mathrm{Q}(A)$. We call A *normal* if $A = \overline{A}$.

We write

$$\mathrm{Spec}(A) = \{P \subseteq A \mid P \text{ prime ideal}\}$$

for the *spectrum* of A and $V(J) = \{P \in \mathrm{Spec}(A) \mid P \supseteq J\}$ for the *vanishing locus* of an ideal J of A. If $P \in \mathrm{Spec}(A)$, then A_P denotes the localization of A at P. More generally, if S is a multiplicatively closed subset of A and M is an A-module, then $S^{-1}M$ denotes the localization of M at S.

Taking into account that normality is a local property, we call

$$N(A) = \{P \in \mathrm{Spec}(A) \mid A_P \text{ is not normal}\}$$

the *non-normal locus* of A. Furthermore, we write

$$\mathrm{Sing}(A) = \{P \in \mathrm{Spec}(A) \mid A_P \text{ is not regular}\}$$

for the *singular locus* of A. Then $N(A) \subseteq \mathrm{Sing}(A)$.

Remark 2.2.2. A Noetherian local ring of dimension one is normal if and only if it is regular. See [25, Theorem 4.4.9].

Definition 2.2.3. Let A be a reduced Noetherian ring. The *conductor* of A in $\overline{A}$ is the ideal

$$\mathcal{C}_A = \mathrm{Ann}_A(\overline{A}/A) = \{a \in A \mid a\overline{A} \subseteq A\}.$$

Lemma 2.2.4. *Let A be a reduced Noetherian ring. Then $N(A) \subseteq V(\mathcal{C}_A)$. Furthermore, $\overline{A}$ is module-finite over A if and only if $\mathcal{C}_A$ contains a non-zerodivisor of A. In this case, $N(A) = V(\mathcal{C}_A)$.*

To state the aforementioned Grauert and Remmert criterion, we need:

Lemma 2.2.5. *Let A be a reduced Noetherian ring, and let $J \subseteq A$ be an ideal containing a non-zerodivisor g of A. Then the following hold:*

1. *If $\varphi \in \mathrm{Hom}_A(J, J)$, then the fraction $\varphi(g)/g \in \overline{A}$ is independent of the choice of g, and φ is multiplication by $\varphi(g)/g$.*
2. *There are natural inclusions of rings*

$$A \subseteq \mathrm{Hom}_A(J, J) \cong \frac{1}{g}\,(gJ :_A J) \subseteq \overline{A} \subseteq \mathrm{Q}(A),\ a \mapsto \varphi_a,\ \varphi \mapsto \frac{\varphi(g)}{g},$$

where $\varphi_a : J \to J$ denotes the multiplication by $a \in A$.

Proposition 2.2.6 ([15])**.** *Let A be a reduced Noetherian ring, and let $J \subseteq A$ be an ideal satisfying the following conditions:*

1. *J contains a non-zerodivisor g of A,*
2. *J is a radical ideal,*
3. *$V(\mathcal{C}_A) \subseteq V(J)$.*

Then A is normal iff $A \cong \mathrm{Hom}_A(J, J)$ via the map which sends a to φ_a.

Definition 2.2.7. A pair (J, g) as in Proposition 2.2.6 is called a *test pair* for A, and J is called a *test ideal* for A.

By Lemma 2.2.4, test pairs exist iff $\overline{A}$ is module-finite over A. Given such a pair (J, g), the idea of finding $\overline{A}$ is to successively enlarge A until the normality criterion allows us to stop (since A is Noetherian, this will eventually happen in the module-finite case). Starting from $A_0 = A$, we get a chain of extensions of reduced Noetherian rings

$$A = A_0 \subseteq \ldots \subseteq A_{i-1} \subseteq A_i \subseteq \ldots \subseteq A_m = \overline{A}.$$

Here, $A_{i+1} = \mathrm{Hom}_{A_i}(J_i, J_i) \cong \frac{1}{g}(gJ_i :_{A_i} J_i)$, where J_i is the radical of the extended ideal JA_i, for $i \geq 1$. Note that (J_i, g) is indeed a test pair for A_i:

Remark 2.2.8 ([17, Prop. 3.2]). Let A be a reduced Noetherian ring such that $\overline{A}$ is module-finite over A, and let $A \subseteq A' \subseteq \overline{A}$ be an intermediate ring. Clearly, every non-zerodivisor $g \in A$ of A is a non-zerodivisor of $\mathrm{Q}(A)$. In particular, it is a non-zerodivisor of A'. Furthermore, if $\mathcal{C}_{A'}$ is the conductor of A' in $\overline{A'} = \overline{A}$, then $\mathcal{C}_{A'} \supseteq \mathcal{C}_A$. It follows that every prime ideal $Q \in N(A') = V(\mathcal{C}_{A'})$ contracts to a prime ideal $P = Q \cap A \in N(A) = V(\mathcal{C}_A)$. Hence, if (J, g) is a test pair for A, then $P \supseteq J$, which implies that $Q \supseteq \sqrt{JA'} =: J'$. We conclude that (J', g) is a test pair for A'.

Explicit computations rely on explicit representations of the A_i as A-algebras. These will be obtained as an application of Lemma 2.2.9 below. To formulate the lemma, we use the following notation. Let $J \subseteq A$ be an ideal containing a non-zerodivisor g of A, and let A-module generators $u_0 = g, u_1, \ldots, u_s$ for $gJ :_A J$ be given. Choose variables $T_1, \ldots, T_s$, and consider the epimorphism

$$\Phi : A[T_1, \ldots, T_s] \to \frac{1}{g}\,(gJ :_A J),\ T_i \mapsto \frac{u_i}{g}.$$

The kernel of Φ describes the A-algebra relations on the u_i/g. We single out two types of relations:

- Each A-module syzygy

  $$\alpha_0 u_0 + \alpha_1 u_1 + \ldots + \alpha_s u_s = 0, \quad \alpha_i \in A,$$

  gives an element $\alpha_0 + \alpha_1 T_1 + \ldots + \alpha_s T_s \in \ker \Phi$, which we call *linear relation*.

- Developing each product $\frac{u_i}{g}\frac{u_j}{g}$, $1 \leq i \leq j \leq s$, as a sum $\frac{u_i}{g}\frac{u_j}{g} = \sum_k \beta_{ijk}\frac{u_k}{g}$, we get elements $T_iT_j - \sum_k \beta_{ijk}T_k$ in $\ker \Phi$, which we call *quadratic relations*.

It is easy to see that these linear and quadratic relations already generate $\ker \Phi$. We thus have:

Lemma 2.2.9. *Let A be a reduced Noetherian ring, and let $J \subseteq A$ be an ideal containing a non-zerodivisor g of A. Then, given A-module generators $u_0 = g, u_1, \ldots, u_s$ for $gJ :_A J$, we have an isomorphism of A-algebras*

$$A[T_1, \ldots, T_s]/R \cong \frac{1}{g}\,(gJ :_A J),\ T_i \mapsto \frac{u_i}{g},$$

where R is the ideal generated by the linear and quadratic relations described above.

The following result from [17] will allow us to find the normalization in a way such that all calculations except the computation of the radicals $\sqrt{J_i}$ can be carried through in the original ring A:

Theorem 2.2.10. *Let A be a reduced Noetherian ring, let $J \subseteq A$ be an ideal containing a non-zerodivisor g of A, let $A \subseteq A' \subseteq \mathrm{Q}(A)$ be an intermediate ring such that A' is module-finite over A, and let $J' = \sqrt{JA'}$. Let U and H be ideals of A and $d \in A$ such that $A' = \frac{1}{d}U$ and $J' = \frac{1}{d}H$, respectively. Then*

$$(gJ' :_{A'} J') = \frac{1}{d}\,(dgH :_A H) \subseteq \mathrm{Q}(A).$$

Remark 2.2.11. In the case where $A = K[X_1 \ldots, X_n]/I$ is a reduced affine algebra over a field K, let $P_1, \ldots, P_r$ be the associated primes of the radical ideal I. Then

$$\overline{A} \cong \overline{K[X_1, \ldots, X_n]/P_1} \times \cdots \times \overline{K[X_1, \ldots, X_n]/P_r},$$

and $\overline{A}$ is module-finite over A by Emmy Noether's finiteness theorem (see [38]). Thus, using techniques for primary decomposition as in [17, Remark 4.6], the computation of normalization can be reduced to the case where A is an affine domain (that is, I is a prime ideal). When writing our algorithms in pseudocode, we will always start from a domain A. Talking about a non-zerodivisor then just means to talk about a non-zero element.

Remark 2.2.12. If A is an affine domain over a perfect field K, we can apply the Jacobian criterion (see [13]): If M is the Jacobian ideal[1] of A, then M is non-zero and contained in the conductor $\mathcal{C}_A$ (see [17, Lemma 4.1]). Hence, we may choose $\sqrt{M}$ together with any non-zero element g of $\sqrt{M}$ as an initial test pair. Implementing all this, the GLS normalization algorithm will find an ideal $U \subseteq A$ and a denominator $d \in \mathcal{C}_A$ such that

$$\overline{A} = \frac{1}{d}\,U \subseteq \mathrm{Q}(A).$$

Since M is contained in $\mathcal{C}_A$, any non-zero element of M is valid as a denominator: If $0 \neq c \in M$, then $c \cdot \frac{1}{d}U =: U'$ is an ideal of A, so that $\frac{1}{d}U = \frac{1}{c}U'$.

For the purpose of comparison with the local approach of the next section, we illustrate the GLS algorithm by an example:

Example 2.2.13. For

$$A = K[x, y] = K[X, Y]/\langle X^4 + Y^2(Y-1)^3\rangle,$$

the radical of the Jacobian ideal is

$$J := \langle x, y(y-1)\rangle_A,$$

and we can take $g := x \in J$ as a non-zerodivisor of A. In its first step, starting with the initial test pair (J, x), the normalization algorithm produces

[1] The *Jacobian ideal* of A is generated by the images of the $c \times c$ minors of the Jacobian matrix $\left(\frac{\partial f_i}{\partial x_j}\right)$, where c is the codimension, and $f_1, \ldots, f_r$ are polynomial generators for I.

the following data:

$$U^{(1)} := (xJ \ :_A \ J) = \langle x, y(y-1)^2 \rangle_A \text{ and}$$
$$A_1 := A[t_1] := A[T_1]/I_1 \cong \frac{1}{x}U^{(1)},$$

with relations and isomorphism given by

$$I_1 = \langle -T_1 x + y(y-1)^2,\, T_1 y(y-1) + x^3,\, T_1^2 + x^2(y-1) \rangle_{A[T_1]}$$

and

$$t_1 \mapsto \frac{y(y-1)^2}{x},$$

respectively. In the next step we find

$$J_1 := \sqrt{\langle x, y(y-1) \rangle_{A_1}} = \langle x, y(y-1), t_1 \rangle_{A_1}$$
$$= \frac{1}{x}\langle x^2, xy(y-1), y(y-1)^2 \rangle_A =: \frac{1}{x}H_1.$$

Using the test pair (J_1, x) and applying Theorem 2.2.10 and Lemma 2.2.9, we get

$$\frac{1}{x}(xJ_1 \ :_{A_1} \ J_1) = \frac{1}{x^2}(x^2 H_1 \ :_A \ H_1)$$
$$= \frac{1}{x^2}\langle x^2, xy(y-1), y(y-1)^2 \rangle_A =: \frac{1}{x^2}U^{(2)}$$

and

$$A_2 := A[t_2, t_3] := A[T_2, T_3]/I_2 \cong \frac{1}{x^2}U^{(2)},$$

with relations and isomorphism given by

$$I_2 = \langle T_2 x - T_3(y-1), -T_3 x + y(y-1), T_2 y(y-1) + x^2,$$
$$T_2 y^2 (y-1)^2 + T_3 x^3, T_2^2 + (y-1), T_2 T_3 + x, T_3^2 - T_2 y \rangle$$

and

$$t_2 \mapsto \frac{y(y-1)^2}{x^2},\ t_3 \mapsto \frac{y(y-1)}{x},$$

respectively. In the final step, we find that A_2 is normal, so that $\overline{A} = A_2$.

2.3 Normalization via Localization

In this section, we discuss a local-to-global approach for computing normalization. Our starting point is the following result:

Proposition 2.3.1. *Let A be a reduced Noetherian ring. Suppose that the singular locus* $\mathrm{Sing}(A) = \{P_1, \ldots, P_s\}$ *of A is finite. For $i = 1, \ldots, s$, let $S_i = A \setminus P_i$, and let an intermediate ring $A \subset A^{(i)} \subset \overline{A}$ be given such that $S_i^{-1}A^{(i)} = \overline{S_i^{-1}A}$. Then*

$$\sum_{i=1}^{s} A^{(i)} = \overline{A}.$$

Proof. We will show a more general result in Proposition 2.3.2 below. □

That $\mathrm{Sing}(A)$ is finite is, for example, true if A is the coordinate ring of a curve. Whenever $\mathrm{Sing}(A) = \{P_1, \ldots, P_s\}$ is finite, the proposition allows us to find $\overline{A}$ by normalizing locally at each P_i using Proposition 2.3.3 below, and putting the local results together. In the case where $\mathrm{Sing}(A)$ is not finite, working just with the (finitely many) minimal primes in $\mathrm{Sing}(A)$ will not give the correct result. However, it is still possible to obtain $\overline{A}$ as a finite sum of local contributions: The idea is to stratify $\mathrm{Sing}(A)$ in a way which is compatible with normalization. For this, if $P \in \mathrm{Sing}(A)$, set

$$L_P = \bigcap_{P \supseteq \widetilde{P} \in \mathrm{Sing}(A)} \widetilde{P}.$$

We stratify $\mathrm{Sing}(A)$ according to the values of the function $P \mapsto L_P$. That is, if

$$\mathcal{L} = \{L_P \mid P \in \mathrm{Sing}(A)\}$$

denotes the set of all possible values, then the strata are the sets

$$V_L = \{P \in \mathrm{Sing}(A) \mid L_P = L\},\ L \in \mathcal{L}.$$

We write $\mathrm{Strata}(A) = \{V_L \mid L \in \mathcal{L}\}$ for the set of all strata. If $P_1, ..., P_r$ denote the minimal primes in $\mathrm{Sing}(A)$, we have

$$\mathcal{L} \subseteq \left\{\textstyle\bigcap_{i\in\Gamma} P_i \mid \Gamma \subseteq \{1, ..., r\}\right\}.$$

Hence, the set of strata is finite. By construction, the singular locus is the disjoint union of all strata. For $V \in \mathrm{Strata}(A)$, write L_V for the constant value of $P \mapsto L_P$ on V.

We can now state and prove a result which is more general than Proposition 2.3.1:

Proposition 2.3.2. *Let A be a reduced Noetherian ring with stratification of the singular locus* $\mathrm{Strata}(A) = \{V_1, ..., V_s\}$. *For $i = 1, \ldots, s$, let an intermediate ring $A \subseteq A^{(i)} \subseteq \overline{A}$ be given such that $S^{-1}A^{(i)} = \overline{S^{-1}A}$ for each $S = A \setminus P$, $P \in V_i$. Then*

$$\sum_{i=1}^{s} A^{(i)} = \overline{A}.$$

Proof. By construction, $B := \sum_{i=1}^{s} A^{(i)} \subseteq \overline{A}$. We wish to show equality. It suffices to show that if $P \in \mathrm{Spec}(A)$ is a prime ideal and $S = A \setminus P$, then $S^{-1}B = S^{-1}\overline{A}$. If $P \in \mathrm{Sing}(A)$, then $P \in V_i$ for some i. Hence, $S^{-1}A^{(i)} = \overline{S^{-1}A}$, and the local equality is obtained from the chain of inclusions

$$S^{-1}A^{(i)} \subseteq S^{-1}B \subseteq S^{-1}\overline{A} = \overline{S^{-1}A}.$$

If $P \notin \mathrm{Sing}(A)$, then $S^{-1}A$ is normal, and the local equality follows likewise from the chain of inclusions

$$S^{-1}A \subseteq S^{-1}B \subseteq S^{-1}\overline{A} = \overline{S^{-1}A}.$$

□

For a given stratum $V = V_i$, the modification of the Grauert and Remmert criterion below will allow us to find a ring $A^{(i)}$ as above along the lines of the previous section:

Proposition 2.3.3. *Let A be a reduced Noetherian ring such that $\overline{A}$ is module-finite over A, and let $A \subseteq A' \subseteq \overline{A}$ be an intermediate ring. Let $V \in \mathrm{Strata}(A)$, and let $J' = \sqrt{L_V A'}$. Suppose that L_V contains a non-zerodivisor g of A. If*

$$A' \cong \mathrm{Hom}_{A'}(J', J')$$

via the map which sends a' to $\varphi_{a'}$, then the localization $S^{-1}A'$ with $S = A \setminus P$ is normal for each $P \in V$.

Proof. The assumption and [13, Proposition 2.10] give

$$S^{-1}A' \cong S^{-1}(\mathrm{Hom}_{A'}(J', J')) \cong \mathrm{Hom}_{S^{-1}A'}(S^{-1}J', S^{-1}J').$$

Hence, the result will follow from the Grauert and Remmert criterion (Proposition 2.2.6) applied to $S^{-1}A'$ once we show that the localized ideal $S^{-1}J'$ satisfies the three conditions of the criterion. First, since forming radicals commutes with localization, $S^{-1}J'$ is a radical ideal. Second, the image of g in $S^{-1}A'$ is a non-zerodivisor of $S^{-1}A'$ contained in $S^{-1}J'$. Third, we show that $V(\mathcal{C}_{S^{-1}A'}) = N(S^{-1}A') \subseteq V(S^{-1}J')$. For this, we first note that

$$V(\mathcal{C}_{S^{-1}A}) = N(S^{-1}A) = \{S^{-1}\widetilde{P} \mid \widetilde{P} \in N(A),\ \widetilde{P} \subseteq P\}.$$

Indeed, prime ideals of $S^{-1}A$ correspond to prime ideals of A contained in P. Let now $Q \in N(S^{-1}A')$. Then, as shown in Remark 2.2.8, Q contracts to some $S^{-1}\widetilde{P} \subseteq S^{-1}A$ with $\widetilde{P} \in N(A)$, $\widetilde{P} \subseteq P$. This implies that

$$Q \supseteq \sqrt{(S^{-1}\widetilde{P})(S^{-1}A')} = \sqrt{S^{-1}(\widetilde{P}A')} = S^{-1}(\sqrt{\widetilde{P}A'}) \supseteq S^{-1}J',$$

as desired. □

In the situation of Proposition 2.3.3, let a non-zerodivisor $g \in L_V$ of A be known. Then, using (L_V, g) instead of a test pair as in Definition 2.2.7, and proceeding as in the previous section, we get a chain of rings

$$A \subseteq A_1 \subseteq \cdots \subseteq A_m \subseteq \overline{A}$$

such that $S^{-1}(A_m)$ is normal and, hence, equal to $S^{-1}\overline{A} = \overline{S^{-1}A}$ for all $S = A \setminus P$, $P \in V$.

Summing up, we are led to Algorithms 2.1 and 2.2 below.

Algorithm 2.1 Normalizing the localizations

Input: An affine domain $A = K[X_1, \ldots, X_n]/I$ over a perfect field K, a stratum $V \in \mathrm{Strata}(A)$, and $0 \neq g \in L_V$

Output: An ideal $U \subseteq A$ and $d \in A$ with $\frac{1}{d}U \subseteq \overline{A}$ and $S^{-1}(\frac{1}{d}U) = \overline{S^{-1}A}$ for all $S = A \setminus P$, $P \in V$

1: **return** the result of the GLS normalization algorithm applied to (L_V, g)

Algorithm 2.2 Normalization via localization

Input: An affine domain $A = K[X_1, \ldots, X_n]/I$ over a perfect field K

Output: An ideal $U \subseteq A$ and $d \in A$ such that $\overline{A} = \frac{1}{d}U \subseteq \mathrm{Q}(A)$

1: $J := \sqrt{M}$, where M is the Jacobian ideal of A
2: choose $0 \neq g \in J$
3: compute the strata of the singular locus $\mathrm{Strata}(A) = \{V_1, ..., V_s\}$
4: **for all** i **do**
5: apply Algorithm 2.1 to (V_i, g) to find an ideal $U_i \subseteq A$ and a power $d_i = g^{m_i}$ with $A \subseteq \frac{1}{d_i}U_i \subseteq \overline{A}$ and $S^{-1}(\frac{1}{d_i}U_i) = \overline{S^{-1}A}$ for all $S = A \setminus P$, $P \in V_i$
6: $m := \max\{m_1, \ldots, m_s\}$, $d := g^m$, $U := \sum_i g^{m-m_i}U_i$
7: **return** (U, d)

Remark 2.3.4. In Algorithm 2.2, it may be more efficient to choose possibly different non-zero elements $g_i \in L_{V_i}$. In Step 5, the algorithm computes, then, pairs (U_i', d_i) with ideals $U_i' \subseteq A$ and powers $d_i = g_i^{m_i}$. As explained in [17, Remark 4.3], starting from the (U_i', d_i), we may always find a denominator $d \in M$ and ideals $U_i \subseteq A$ such that $\frac{1}{d}U_i = \frac{1}{d_i}U_i'$ for all i. Then, the desired result is $(\sum_i U_i, d)$.

Remark 2.3.5. In Algorithm 2.2, it is sufficient to consider the minimal strata, that is, the strata V such that L_V is minimal with respect to inclusion. Denote, as above, the minimal primes of the singular locus of A by $P_1, ..., P_r$. We can obtain the minimal L_V as all possible intersections $\bigcap_{i\in\Gamma} P_i$, with subsets $\Gamma \subseteq \{1, ..., r\}$ which are maximal with the property that $\sum_{i\in\Gamma} P_i \neq \langle 1 \rangle$.

Example 2.3.6. We come back to the coordinate ring A of the curve C with defining polynomial $f(X,Y) = X^4 + Y^2(Y-1)^3$ from Example 2.2.13 to discuss normalization via localization. The curve C has a double point of type A_3 at $(0,0)$ and a triple point of type E_6 at $(0,1)$. We illustrate Algorithm 2.2, using for both singular points the non-zerodivisor $g = x$: For the A_3-singularity, consider

$$P_1 = \langle x, y \rangle_A \text{ and } S_1 = A \setminus P_1.$$

The local normalization algorithm yields $\overline{S_1^{-1}A} = S_1^{-1}(\frac{1}{d_1}U_1)$, where

$$d_1 = x^2 \text{ and } U_1 = \left\langle x^2,\ y(y-1)^3 \right\rangle_A.$$

For the E_6-singularity, considering

$$P_2 = \langle x, y-1 \rangle_A \text{ and } S_2 = A \setminus P_2,$$

we get $\overline{S_2^{-1}A} = S_2^{-1}(\frac{1}{d_2}U_2)$, where

$$d_2 = x^2 \text{ and } U_2 = \left\langle x^2,\ xy^2(y-1),\ y^2(y-1)^2 \right\rangle_A.$$

Combining the local contributions, we get

$$\frac{1}{d}U = \frac{1}{d_1}U_1 + \frac{1}{d_2}U_2,$$

with $d = x^2$ and

$$U = \left\langle x^2,\ xy^2(y-1),\ y(y-1)^3,\ y^2(y-1)^2 \right\rangle_A.$$

A moment's thought shows that U coincides with the ideal $U^{(2)}$ found in Example 2.2.13.

The local-to-global approach is usually much faster than the global algorithm even when not run in parallel. The reason is that the minimal primes of the singular locus are much simpler than the singular locus itself. Therefore, in the local-to-global case, the intermediate rings are much easier to handle. Most notably, the representations of the intermediate rings as affine rings involve considerably less variables than in the global case. In the following example, we exemplify this difference.

Example 2.3.7. Consider the *projective* plane curve defined by the polynomial

$$f_{1,4} = (X^5 + Y^5 + Z^5)^2 - 4(X^5Y^5 + X^5Z^5 + Y^5Z^5) \in \mathbb{Q}[X,Y,Z],$$

which will be reconsidered in Section 2.5 with respect to timings. After the coordinate transformation $Z \mapsto 3X - 2Y + Z$, all singularities of the projective curve lie in the affine chart $Z \neq 0$. Write

$$f = f_{1,4}(X, Y, 3X - 2Y + 1) \in \mathbb{Q}[X,Y] =: W$$

for the defining polynomial of the affine curve, and let $A = \mathbb{Q}[x, y] = W/\langle f \rangle$.

The curve has 15 singular points: the radical of the Jacobian ideal M decomposes as

$$\begin{aligned}\sqrt{M} = &\langle y, 121x^4 + 142x^3 + 64x^2 + 13x + 1\rangle \cap \langle y, 2x + 1\rangle \\ &\cap \langle 211y^4 - 131y^3 + 51y^2 - 11y + 1, 3x - 2y + 1\rangle \\ &\cap \langle 11y^4 - 23y^3 + 19y^2 - 7y + 1, x\rangle \cap \langle y + 1, x + 1\rangle \cap \langle 3y - 1, x\rangle.\end{aligned}$$

We compare the global approach to the local strategy at the singularity corresponding to the test ideal $J = \langle y, 2x + 1\rangle$.

In the local setting, we use the non-zerodivisor $g = y$ and compute the ideal quotient

$$\begin{aligned}U_1 &= gJ : J \\ &= \langle y, 29282x^9 + 83369x^8 + 105668x^7 + 78296x^6 + 37382x^5 + 11926x^4 \\ &\qquad + 2542x^3 + 349x^2 + 28x + 1\rangle.\end{aligned}$$

We observe that in addition to y, the ideal U_1 requires only one more generator. Hence, the representation of $A_1 \cong \frac{1}{g}(gJ : J)$ as an affine ring $A_1 = A[T_1]/I_1$ requires only one additional variable T_1. The ideal I_1 is generated by 10 linear relations and one quadratic relation. Next, we compute the radical of the image of J in A_1. Technically, this means to compute the radical $\sqrt{J + I_1}$ in the polynomial ring $W[T_1]$ (here, by abuse of notation, we denote the preimage of J in the polynomial ring also by J). The ideal I_1 is quite complicated. Since J is generated by linear polynomials, however, the reduced Gröbner basis of $J + I_1$ is very simple. It is easily computed as

$$J + I_1 = \langle Y, 2X + 1, 16384T_1^2 - T_1 + 625\rangle.$$

As a consequence, the computation of the radical

$$J_1 = \sqrt{J + I_1} = \langle Y, 2X + 1, 128T_1 - 25\rangle$$

is cheap as well. In the next step, A_2 can be represented as an affine algebra over A with, again, only one new variable. Hence, verifying that A_2 is already a local contribution to $\overline{A}$ at the singularity corresponding to J is also cheap.

In contrast, the global approach uses the test ideal $J = \sqrt{M}$, which is generated by one polynomial of degree 3 and three polynomials of degree 6. As a non-zerodivisor, we consider the lowest degree generator $g = 3x^2y - 2xy^2 + xy$. As in the local case, the first ideal quotient $gJ : J$ is easily obtained. However, in addition to g, it requires three more generators. Hence, as an affine algebra over A, it is represented as $A_1 = A[T_1, T_2, T_3]/I_1$, where the ideal of relations $I_1 \subseteq A[T_1, T_2, T_3]$ is generated by 6 linear and 6 quadratic relations. No significant reduction occurs in $J + I_1$ since J does

not contain any linear polynomial. The complexity of Buchberger's algorithm grows doubly-exponentially in the number of variables. Compared to the local case, this increase in complexity makes the computation of $\sqrt{J + I_1}$ considerably more expensive. In fact, SINGULAR does not compute the corresponding Gröbner basis within 2000 seconds.

2.4 Modular Methods

Algorithm 2.2 from Section 2.3 is parallel in nature since the computations of the local normalizations do not depend on each other. In this section, we describe a modular way of parallelizing both the GLS normalization algorithm from Section 2.2 and the local-to-global algorithm from Section 2.3 in the case where $K = \mathbb{Q}$ is the field of rationals. One possible approach is to just replace *all* involved Gröbner basis respectively radical computations by their modular variants as introduced in [2] and [22]. These variants are either probabilistic or require rather expensive tests to verify the results at the end. In order to reduce the number of verification tests, we provide a direct modularization for the normalization algorithms. The approach we propose requires, in principle, only one verification at the end. In the local-to-global setup, however, it is reasonable to additionally handle the Gröbner basis computation for the Jacobian ideal, the subsequent primary decomposition, and the recombination of the local results by modular techniques. We exemplarily describe the modularization of the GLS normalization algorithm as outlined in Section 2.2. Each of the local normalizations in Algorithm 2.2 from Section 2.3 can be modularized similarly.

Fix a global monomial ordering $>$ on the semigroup of monomials in the set of variables $X = \{X_1, \ldots, X_n\}$. Consider the polynomial rings $W = \mathbb{Q}[X]$ and, given an integer $N \geq 2$, $W_N = (\mathbb{Z}/N\mathbb{Z})[X]$. If $T \subseteq W$ or $T \subseteq W_N$ is a set of polynomials, then denote by $\mathrm{LM}(T) := \{\mathrm{LM}(f) \mid f \in T\}$ its set of leading monomials. If $\frac{a}{b} \in \mathbb{Q}$ with $\gcd(a, b) = 1$ and $\gcd(b, N) = 1$, set $\left(\frac{a}{b}\right)_N := (a + N\mathbb{Z})(b + N\mathbb{Z})^{-1} \in \mathbb{Z}/N\mathbb{Z}$. If $f \in W$ is a polynomial such that N is coprime to any denominator of a coefficient of f, then $f_N \in W_N$ is the polynomial obtained by reducing each coefficient modulo N as just described. If $H = \{h_1, \ldots, h_t\}$ is a Gröbner basis in W such that N is coprime to any denominator in any h_i, then write $H_N = \{(h_1)_N, \ldots, (h_t)_N\}$. Given an ideal $I \subseteq W$, set $I_N = \langle f_N \mid f \in I \cap \mathbb{Z}[X] \rangle \subseteq W_N$ and $(W/I)_N = W_N/I_N$.

Remark 2.4.1. For practical purposes, the ideal $I \subseteq W$ is given by a set of generators $f_1, \ldots, f_r$. Then for all but finitely many primes p, the ideal I_p can be realized via the equality

$$I_p = \langle (f_1)_p, ..., (f_r)_p \rangle \subseteq W_p.$$

When performing the modular algorithm described below, we reject a prime p if the ideal on the right hand side is not well-defined. Otherwise, we

work with this ideal instead of I_p. The finitely many primes where the two ideals differ will not influence the result if we apply error tolerant rational reconstruction (see Remark 2.4.2).

From a practical point of view, we work with ideals of the polynomial ring W containing I, but think of them as ideals of the quotient ring $A = W/I$. Therefore, we simplify our notation as follows: If $I \subseteq J \subseteq W$ are ideals, then we denote the ideal induced by J in A also by J. Vice versa, if $J \subseteq A$ is an ideal, then its preimage in W is also denoted by J. Similarly for W_N.

From now on, $I = \langle f_1, \ldots, f_r \rangle \subseteq W$ will be a prime ideal. We wish to compute the normalization of the affine domain $A = W/I$ using modular methods. For this, we fix a polynomial $d \in W$ which represents a non-zero element in the Jacobian ideal M of A. This element of M will also be denoted by d. It will serve as a "universal denominator" for all normalizations in positive characteristic as well as for the final normalization in characteristic 0 (see Remark 2.2.12 for the choice of denominators). In characteristic 0, we write $U(0)$ for the ideal of A which satisfies $\frac{1}{d}U(0) = \overline{A}$, and $G(0)$ for the reduced Gröbner basis[2] of $U(0)$. Furthermore, we write $V(0) \subseteq A[T_1, \ldots, T_s]$ for the ideal[3] of relations on the elements of $\frac{1}{d}G(0)$ which represents $\overline{A}$ as an A-algebra as in Lemma 2.2.9. We denote the reduced Gröbner basis of $V(0)$ by $R(0)$. In the same way, if p is a prime number which does not divide any denominator in the reduced Gröbner basis of I and such that A_p is a domain and d_p is non-zero and contained in the conductor[4] of A_p, we use $U(p)$, $G(p)$, $V(p)$, and $R(p)$ in characteristic p.

Note that $G(0)_p$ is not necessarily equal to $G(p)$. However, as we will show in Lemma 2.4.5 below, equality holds for all but finitely many primes p. Relying on this fact, the basic idea of the modular normalization algorithm can be described as follows. First, compute the Jacobian ideal M of A and choose a polynomial $d \in W$ representing a non-zero element $d \in M$. Second, choose a set $\mathcal{P}$ of prime numbers at random, and compute, for each $p \in \mathcal{P}$, reduced Gröbner bases $G(p) \subseteq W_p$ such that $\frac{1}{d_p}\langle G(p) \rangle \subseteq \mathrm{Q}(A_p)$ is the normalization of A_p. Third, lift the modular Gröbner bases to a set of polynomials $G \subseteq W$ and define $U := \langle G \rangle$. We then expect that $U = U(0)$ and $G = G(0)$.

The lifting process has two steps. First, assuming that all $\mathrm{LM}(G(p))$, $p \in \mathcal{P}$, are equal, we can lift the Gröbner bases in the set $\mathcal{GP} := \{G(p) \mid p \in \mathcal{P}\}$ to a set of polynomials $G(N) \subseteq W_N$, with $N := \prod_{p \in \mathcal{P}} p$. For this, apply the Chinese remainder algorithm to the coefficients of the corresponding polynomials occurring in the $G(p)$, $p \in \mathcal{P}$. Second, compute a set of

[2]Recall that reduced Gröbner bases are uniquely determined. For practical purposes, however, we do not need to reduce the Gröbner bases involved since the lifting process described below only requires that the result is uniquely determined by the algorithm.

[3]With respect to ideals of $W[T_1, \ldots, T_s]$ and $A[T_1, \ldots, T_s]$, we use the same setup and notation as for ideals of W and A.

[4]From a practical point of view, we check whether d_p is in the Jacobian ideal of A_p.

polynomials $G \subseteq W$ by lifting the modular coefficients occurring in $G(N)$ to rational coefficients as described in [8]:

Remark 2.4.2. Rational reconstruction via the Chinese remainder theorem and Gaussian reduction is error-tolerant in the following sense: Let N_1 and N_2 be integers with $\gcd(N_1, N_2) = 1$, and let $\frac{a}{b} \in \mathbb{Q}$ with $\gcd(a, b) = \gcd(N_1, b) = 1$. Set $r_1 := \left(\frac{a}{b}\right)_{N_1} \in \mathbb{Z}/N_1\mathbb{Z}$, let $r_2 \in \mathbb{Z}/N_2\mathbb{Z}$ be arbitrary, and denote by r the image of (r_1, r_2) under the isomorphism

$$\mathbb{Z}/N_1\mathbb{Z} \times \mathbb{Z}/N_2\mathbb{Z} \to \mathbb{Z}/(N_1N_2)\mathbb{Z}.$$

Lifting r to a rational number via Gaussian reduction will generate, starting from $(a_0, b_0) = (N_1N_2, 0)$ and $(a_1, b_1) = (r, 1)$, a sequence of rational numbers (a_i, b_i) obtained by setting

$$(a_{i-2}, b_{i-2}) = q_i(a_{i-1}, b_{i-1}) + (a_i, b_i),$$

where q_i is chosen such that (a_i, b_i) has minimal Euclidean length. Computing this sequence until the Euclidean length does not decrease strictly any more, we obtain a tuple (a_i, b_i) with $\frac{a_i}{b_i} = \frac{a}{b}$, provided that $N_2 \ll N_1$. For details, see [8].

Just as for $\mathcal{GP}$, we proceed for the set of reduced Gröbner bases $\mathcal{RP} := \{R(p) \mid p \in \mathcal{P}\}$ giving the modular algebra relations.

As for other modular algorithms based on Chinese remaindering, we need suitably adapted notions of a lucky prime and a sufficiently large set of lucky primes:

Definition 2.4.3. Using the notation introduced above, we define:

1. A prime number p is called *lucky* for A if $U(0)_p = U(p)$, $V(0)_p = V(p)$, and the following hold:

   (a) A_p is a domain.

   (b) d_p is a non-zero element in the conductor of A_p.

   (c) $\mathrm{LM}(G(0)) = \mathrm{LM}(G(p))$.

   (d) $\mathrm{LM}(R(0)) = \mathrm{LM}(R(p))$.

   Otherwise p is called *unlucky* for A.

2. A finite set $\mathcal{P}$ of lucky primes for A is called *sufficiently large* for A if

$$\prod_{p \in \mathcal{P}} p \geq \max\left\{ 2 \cdot |c|^2 \;\middle|\; \begin{array}{l} c \text{ a denominator or numerator of a} \\ \text{coefficient occurring in } G(0) \text{ or } R(0) \end{array} \right\}$$

Remark 2.4.4. A modular algorithm for the basic task of computing Gröbner bases is presented in [2] and [22]. In contrast to our situation here, where we wish to find the ideal $U(0)$ by computing its reduced Gröbner basis $G(0)$, Arnold's algorithm starts from an ideal which is already given. If p is a prime number, $J \subset W$ is an ideal, $H(0)$ is the reduced Gröbner basis of J, and $H(p)$ the reduced Gröbner basis of J_p, then p is lucky for J in the sense of Arnold if $\mathrm{LM}(H(0)) = \mathrm{LM}(H(p))$. It is shown in [2, Thm. 5.12 and 6.2] that if p is lucky for J in this sense, then $H(0)_p$ is well-defined and equal to $H(p)$. By [2, Cor. 5.4 and Thm. 5.13], all but finitely many primes are Arnold-lucky for J. Moreover, if $\mathcal{P}$ is a set of primes satisfying Arnold's condition $\mathrm{LM}(H(0)) = \mathrm{LM}(H(p))$ for all $p \in \mathcal{P}$, and such that $\mathcal{P}$ is sufficiently large with respect to the coefficients occurring in $H(0)$, then the $H(p)$, $p \in \mathcal{P}$, lift to $H(0)$.

In our situation, if p is a prime number, we find $U(p)$ on our way, but $U(0)_p$ is only known to us after $U(0)$ has been computed. Therefore, the condition $U(0)_p = U(p)$ in our definition of lucky can only be checked a posteriori. Similarly for $V(0)_p = V(p)$. However, when performing our modular algorithm, by part 1 of Lemma 2.4.5 below and Remark 2.4.2, there are only finitely many primes not satisfying these conditions and these primes will not influence the result of the algorithm.

Lemma 2.4.5. *With notation as above, we have:*

1. *All but a finite number of primes are lucky for A.*
2. *If $\mathcal{P}$ is a sufficiently large set of lucky primes for A, then the reduced Gröbner bases $G(p)$, $p \in \mathcal{P}$, lift to the reduced Gröbner basis $G(0)$. In the same way, the $R(p)$, $p \in \mathcal{P}$, lift to $R(0)$.*

Proof. With respect to part 1, it is clear that conditions (1a) and (1b) in our definition of lucky are true for all but finitely many primes. Moreover, $\frac{1}{d_p}U(0)_p$ is integral over A_p for all but finitely many p. Since testing normality via the Grauert and Remmert criterion amounts to a Gröbner basis computation, and since reducing a Gröbner basis modulo a sufficiently general prime p gives a Gröbner basis of the reduced ideal, we conclude that $U(0)_p = U(p)$ for all but finitely many primes. Furthermore, if $U(0)_p = U(p)$, condition (1c) from our definition of lucky is equivalent to asking that p is lucky for $U(0)$ in the sense of Arnold, so that also this condition holds for all but finitely many primes. For $V(0)_p = V(p)$ and condition (1d), we may argue similarly since finding the ideal of algebra relations amounts to another Gröbner basis computation.

For part 2, let $\mathcal{P}$ be a sufficiently large set of lucky primes for A. Then, as pointed out above, $G(0)_p$ is well-defined and equal to $G(p)$ for all $p \in \mathcal{P}$. Furthermore, since $\mathcal{P}$ is sufficiently large, the $G(0)_p$, $p \in \mathcal{P}$, lift to $G(0)$. In the same way, we may argue for the relations. □

From a theoretical point of view, the idea of the algorithm is now as follows: Consider a sufficiently large set $\mathcal{P}$ of lucky primes for A, compute the reduced Gröbner bases $G(p)$, $p \in \mathcal{P}$, and lift the results to the reduced Gröbner basis $G(0)$ as described above.

From a practical point of view, we face the problem that the conditions (1c), (1d), and (2) from Definition 2.4.3 cannot be tested a priori. To remedy the situation, we proceed in a randomized way. First, we fix an integer $t \geq 1$ and choose a set of t primes $\mathcal{P}$ at random. Second, we delete all primes p from $\mathcal{P}$ which do not satisfy conditions (1a) and (1b). Third, we compute $\mathcal{GP} = \{G(p) \mid p \in \mathcal{P}\}$ and $\mathcal{RP} = \{R(p) \mid p \in \mathcal{P}\}$, and use the following test to modify $\mathcal{P}$ so that all primes in $\mathcal{P}$ satisfy (1c) and (1d) with high probability:

deleteUnluckyPrimesNormal: *Define an equivalence relation on $\mathcal{P}$ by setting $p \sim q :\Longleftrightarrow \big(\mathrm{LM}(G(p)) = \mathrm{LM}(G(q))$ and $\mathrm{LM}(R(p)) = \mathrm{LM}(R(q))\big)$. Then replace $\mathcal{P}$ by an equivalence class of largest*[5] *cardinality, and change $\mathcal{GP}$ and $\mathcal{RP}$ accordingly.*

Only now, we lift the Gröbner bases in $\mathcal{GP}$ and $\mathcal{RP}$ to sets of polynomials G and R, respectively. Since we do not know whether all primes in the chosen equivalence class are indeed lucky and whether the class is sufficiently large, a final verification step is needed: We have to check whether $\frac{1}{d}\langle G\rangle$ is integral over A and normal. Since this can be expensive, especially if the result is false, we test the result at first in positive characteristic:

pTestNormal: *Randomly choose a prime number $p \notin \mathcal{P}$ such that A_p is a domain, d_p is a non-zero element in the conductor of A_p, and p does not divide the numerator and denominator of any coefficient occurring in a polynomial in G, R, or $\{f_1, \ldots, f_r\}$. Return true if $\frac{1}{d_p}\langle G_p\rangle$ is the normalization of A_p and satisfies the relations R_p, and false otherwise.*

If pTestNormal returns false, then $\mathcal{P}$ is not sufficiently large for A or not all primes in $\mathcal{P}$ are lucky (or the extra prime chosen in pTestNormal is unlucky). In this case, we enlarge the set $\mathcal{P}$ by t primes not used so far and repeat the whole process. On the other hand, if pTestNormal returns true, then most likely $G = G(0)$ and, thus, $\frac{1}{d}\langle G\rangle = \overline{A}$. It makes, then, sense to verify the result over the rationals by applying the following lemma. If the verification fails, we enlarge $\mathcal{P}$ and repeat the process.

Lemma 2.4.6. *With notation as above, the ring $\frac{1}{d}\langle G\rangle \subseteq \mathrm{Q}(A)$ is the normalization of A if and only if the following two conditions hold:*

1. *The ring $\frac{1}{d}\langle G\rangle$ is integral over A. This holds if G and R are Gröbner bases, and the elements of $\frac{1}{d}G$ satisfy the relations R.*

[5] If applicable, take Remark 2.4.7 below into account.

2. *The ring $\frac{1}{d}\langle G\rangle$ is normal. Equivalently, $\frac{1}{d}\langle G\rangle$ satisfies the conditions of the Grauert and Remmert criterion.*

Proof. If $\frac{1}{d}\langle G\rangle$ is integral over A, then $\frac{1}{d}\langle G\rangle \subseteq \overline{A}$. If $\frac{1}{d}\langle G\rangle$ is also normal, then equality holds. Note that if R is a Gröbner basis, then $\dim\langle R\rangle = \dim\langle R(p)\rangle$ for all $p \in \mathcal{P}$. Hence, if the elements of $\frac{1}{d}G$ satisfy the relations R, and G is a Gröbner basis, then $\frac{1}{d}\langle G\rangle$ is integral over A. □

We summarize modular normalization in Algorithm 2.3.

Algorithm 2.3 Modular normalization

Input: A prime ideal $I \subseteq \mathbb{Q}[X]$
Output: A Gröbner basis $G \subseteq \mathbb{Q}[X]$ and $d \in \mathbb{Q}[X]$ such that $\frac{1}{d}\langle G\rangle \subseteq \mathrm{Q}(A)$ is the normalization of $A = \mathbb{Q}[X]/I$

1: compute M, the Jacobian ideal of A
2: choose a polynomial $d \in \mathbb{Q}[X]$ representing a non-zero element $d \in M$
3: choose $\mathcal{P}$, a list of random primes
4: $\mathcal{GP} = \emptyset$, $\mathcal{RP} = \emptyset$
5: **loop**
6: **for** $p \in \mathcal{P}$ **do**
7: **if** A_p is not a domain or $d_p \in A_p$ is zero or d_p is not contained in the conductor of A_p **then**
8: delete p
9: **else**
10: use the GLS algorithm to compute $G(p)$, the reduced Gröbner basis such that $\frac{1}{d_p}\langle G(p)\rangle \subseteq \mathrm{Q}(A_p)$ is the normalization of A_p, and $R(p)$, the reduced Gröbner basis of the ideal of algebra relations
11: $\mathcal{GP} = \mathcal{GP} \cup \{G(p)\}$, $\mathcal{RP} = \mathcal{RP} \cup \{R(p)\}$
12: $(\mathcal{GP}, \mathcal{RP}, \mathcal{P}) =$ deleteUnluckyPrimesNormal$(\mathcal{GP}, \mathcal{RP}, \mathcal{P})$
13: lift $(\mathcal{GP}, \mathcal{RP}, \mathcal{P})$ to $G \subseteq \mathbb{Q}[X]$ and $R \subseteq W[T_1, \ldots, T_s]$ via Chinese remaindering and the Farey rational map
14: **if** the lift succeeds and pTestNormal$(I, d, G, R, \mathcal{P})$ **then**
15: **if** $\frac{1}{d}\langle G\rangle \subseteq \mathrm{Q}(A)$ is integral over A **then**
16: **if** $\frac{1}{d}\langle G\rangle \subseteq \mathrm{Q}(A)$ is normal **then**
17: **return** (G, d)
18: enlarge $\mathcal{P}$

Remark 2.4.7. If the loop in Algorithm 2.3 requires more than one round, we have to apply deleteUnluckyPrimesNormal in Step 12 with some care. Otherwise, it may happen that always classes containing only unlucky primes are selected. To avoid this problem, when determining the cardinality of the classes considered in a certain round of the loop, we count all prime numbers

in the class selected in the previous round as just one element. Then $\mathcal{P}$ will eventually contain lucky primes and termination of the algorithm is ensured by Lemma 2.4.5 and Remark 2.4.2.

Remark 2.4.8. In Algorithm 2.3, the normalizations $\frac{1}{d_p}\langle G(p)\rangle$ can be computed in parallel. Furthermore, we can parallelize the final verifications of integrality and normality.

Remark 2.4.9. Algorithm 2.3 is also applicable without the final tests (that is, without the verification that $\frac{1}{d}\langle G\rangle \subseteq \mathrm{Q}(A)$ is integral over A and normal). In this case, the algorithm is probabilistic, that is, the output $\frac{1}{d}\langle G\rangle \subseteq \mathrm{Q}(A)$ is the normalization of A only with high probability. This usually accelerates the algorithm considerably.

Remark 2.4.10. The computation of the algebra structure R of the normalization via lifting of the relations $R(p)$ may require a large number of primes. Hence, if the number of cores available is limited, a better choice is to obtain just G by the modular approach and then compute the relations $R(0)$ over the rationals. For this approach, the initial ideals of the relations need not be tested in DELETEUNLUCKYPRIMESNORMAL and PTESTNORMAL.

2.5 Timings

We compare the GLS normalization algorithm[6] (denoted in the tables below by `normal`) with Algorithm 2.2 from Section 2.3 (`locNormal`) and Algorithm 2.3 from Section 2.4 (`modNormal`)[7]. For all modular computations, we use the simplified algorithm as specified in Remark 2.4.10. Note that at this writing, modularized versions of `locNormal` have not yet been implemented.

In many cases, it turns out that the final verification is a time-consuming step of `modNormal`. To quantify the improvement of computation times by omitting the verification, we give timings for the resulting, now probabilistic, version of Algorithm 2.3 (denoted by `modNormal`* in the tables). In all examples computed so far, the result of the probabilistic algorithm is indeed correct.

All timings are in seconds on an AMD Opteron 6174 machine with 48 cores, 2.2 GHz, and 128 GB of RAM, running a Linux operating system. Computations which did not finish within 2000 seconds are marked by a dash. The maximum number of cores used is written in square brackets. For the single core version of `modNormal`, we indicate the number of primes used by the algorithm in brackets.

The projective plane curves defined by the equations

$$f_{1,k} = \left(X^{k+1} + Y^{k+1} + Z^{k+1}\right)^2 - 4\left(X^{k+1}Y^{k+1} + Y^{k+1}Z^{k+1} + Z^{k+1}X^{k+1}\right)$$

[6] We use the implementation available in the SINGULAR library `normal.lib`.

[7] To implement our algorithms, we have created the SINGULAR libraries `modnormal.lib` and `locnormal.lib`.

were constructed in [21]. They have $3(k+1)$ singularities of type A_k, provided that k is even. If k is odd, the curves are reducible, in which case the normalization algorithms still work in the same way as in the irreducible case as long as they do not detect a zerodivisor. After the coordinate transformation $Z \mapsto 3X - 2Y + Z$, all singularities of the projective curves lie in the affine chart $Z \neq 0$. We apply the algorithm to the affine curves. The timings for $k = 2, ..., 5$ are shown in Table 2.1.

Table 2.1: Timings for plane curves with many A_k singularities

	$f_{1,2}$	$f_{1,3}$	$f_{1,4}$	$f_{1,5}$
`normal[1]`	.34	14	–	–
`locNormal[1]`	.57	2.0	2.1	38
`locNormal[20]`	.42	1.3	1.4	11
`modNormal[1]`	4.4 (3)	73 (4)	250 (5)	–
`modNormal[10]`	4.1	68	240	–
`modNormal*[1]`	.57 (3)	7.4 (4)	11 (5)	–
`modNormal*[10]`	.31	2.1	2.5	–

Both the local and the probabilistic modular approach have a better performance than the GLS algorithm, and they improve further in their parallel versions. The modular algorithm with final verification is slower, but can still handle much bigger examples than GLS.

Timings for the affine plane curves defined by

$$\begin{aligned}
f_{2,k} &= ((X-1)^k - Y^3)((X+1)^k - Y^3)(X^k - Y^3)((X-2)^k - Y^3) \\
&\quad \cdot ((X+2)^k - Y^3) + Y^{15}, \\
f_3 &= X^{10} + Y^{10} + (X - 2Y + 1)^{10} \\
&\quad + 2(X^5(X - 2Y + 1)^5 - X^5Y^5 + Y^5(X - 2Y + 1)^5), \\
f_4 &= (Y^5 + 2X^8)(Y^3 + 7(X-1)^4)((Y+5)^7 + 2X^{12}) + Y^{11}, \\
f_5 &= 9127158539954X^{10} + 3212722859346X^8Y^2 + 228715574724X^6Y^4 \\
&\quad - 34263110700X^4Y^6 - 5431439286X^2Y^8 - 201803238Y^{10} \\
&\quad - 134266087241X^8 - 15052058268X^6Y^2 + 12024807786X^4Y^4 \\
&\quad + 506101284X^2Y^6 - 202172841Y^8 + 761328152X^6 \\
&\quad - 128361096X^4Y^2 + 47970216X^2Y^4 - 6697080Y^6 - 2042158X^4 \\
&\quad + 660492X^2Y^2 - 84366Y^4 + 2494X^2 - 474Y^2 - 1
\end{aligned}$$

are presented in Table 2.2.

Table 2.2: Timings for plane curves with various types of singularities

	$f_{2,7}$	$f_{2,8}$	$f_{2,9}$	f_3	f_4	f_5
`normal[1]`	7.7	12	383	–	474	1620
`locNormal[1]`	4.4	13	118	1.9	19	1.2
`locNormal[20]`	1.4	3.3	31	1.4	18	.93
`modNormal[1]`	38 (3)	69 (3)	146 (3)	142 (3)	–	50 (8)
`modNormal[10]`	38	69	146	84	–	43
`modNormal*[1]`	.70 (3)	1.2 (3)	1.2 (3)	88 (3)	9.8 (3)	7.0 (8)
`modNormal*[10]`	.47	.70	.74	30	4.7	.98

Table 2.3: Timings for the normalization of surfaces in $\mathbb{A}^3$

	$f_{6,11}$	$f_{6,12}$	$f_{6,13}$	$f_{7,2}$	$f_{7,3}$	f_8
`normal[1]`	2.6	11	6.4	–	–	–
`locNormal[1]`	.25	.26	.29	80	113	70
`locNormal[20]`	.21	.22	.24	80	113	70
`modNormal*[1]`	2.2 (2)	.60 (2)	.78 (2)	12 (5)	17 (5)	2.3 (2)
`modNormal*[10]`	1.5	.52	.67	3.5	4.7	1.7

In Table 2.3, we consider surfaces in $\mathbb{A}^3$ cut out by

$$\begin{aligned} f_{6,k} &= XY(X-Y)(X+Y)(Y-1)Z + (X^k - Y^2)(X^{10} - (Y-1)^2)\,, \\ f_{7,k} &= Z^2 - (Y^2 - 1234X^3)^k(15791X^2 - Y^3)(1231Y^2 - X^2(X+158)) \\ &\quad (1357Y^5 - 3X^{11})\,, \\ f_8 &= Z^5 - ((13X - 17Y)(5X^2 - 7Y^3)(3X^3 - 2Y^2) \\ &\quad (19Y^2 - 23X^2(X+29)))^2\,. \end{aligned}$$

We omit the verification step in the modular algorithm, as this is too time-consuming.

Timings for the curves in $\mathbb{A}^3$ defined by the ideals

$$I_{9,k} = \left\langle Z^3 - (19Y^2 - 23X^2(X+29))^2, X^3 - (11Y^2 - 13Z^2(Z+1))^k \right\rangle$$

and the surface in $\mathbb{A}^4$ defined by

$$\begin{aligned} I_{10} = \big\langle & Z^2 - (Y^3 - 123456W^2)(15791X^2 - Y^3)^2, \\ & WZ - (1231Y^2 - X(111X + 158)) \big\rangle \end{aligned}$$

Table 2.4: Timings for curves in $\mathbb{A}^3$ and a surface in $\mathbb{A}^4$

	$I_{9,1}$	$I_{9,2}$	I_{10}
`normal[1]`	3.2	–	150
`locNormal[1]`	4.2	36	83
`locNormal[20]`	4.1	35	82
`modNormal[1]`	–	–	28 (4)
`modNormal[10]`	–	–	14
`modNormal*[1]`	8.9 (5)	–	8.4 (4)
`modNormal*[10]`	2.1	–	2.5

are given in Table 2.4.

To summarize, both the local and the probabilistic modular approach provide a significant improvement over the GLS algorithm in computation times and size of the examples covered. The probabilistic method is very stable in the sense that it produces the correct result in all examples computed so far. As usual, the verification step in the modular setup is the most time-consuming task, and a refinement of this step will be the focus of further research. The modular technique parallelizes completely, the local approach parallelizes best if the complexity distributes evenly over the minimal strata of the singular locus. In general, the localization technique, even when not run in parallel, is a major improvement to the GLS algorithm. Note, that the local contribution can also be obtained by other means. See, for example, [6] for a fast method in the case of curves, using Hensel lifting and Puiseux series.

Chapter 3

Parallel Primary Decomposition

We present a parallel version of the algorithm of Gianni, Trager, and Zacharias [14] for primary decomposition which relies on four key ingredients: First, we use the parallel algorithm of Idrees, Pfister, and Steidel [22] to compute the associated primes whenever we encounter a zero-dimensional ideal. Its primary components are then extracted from the associated primes via saturation. Second, we use modular methods for computing standard bases and ideal quotients at the intermediate steps, such as the reduction to the zero-dimensional case. Third, we use fast tests to check the final result. In particular, we use a new method to verify that the input ideal is indeed the intersection of the computed primary ideals. Finally, we parallelize the trivially parallelizable parts of the algorithm. For example, the saturations mentioned above can be computed in parallel for each primary component.

In Section 3.1, we recall the original algorithm of Gianni, Trager, and Zacharias [14] and the basic theory for primary decomposition. The proposed parallel algorithm and its four key ingredients are described in detail in Section 3.2. Timings are given in Section 3.3.

Using the proposed parallel algorithm, we are able to compute primary decompositions for examples which have been up to now unsolvable. However, we have also found examples where the new algorithm is not faster than the original one, or where it does not scale well with the number of processor cores. These examples are especially challenging with respect to parallelization and need further investigation.

3.1 The Algorithm of Gianni, Trager, and Zacharias

In this section, we briefly outline the algorithm of Gianni, Trager, and Zacharias [14] to compute a primary decomposition of a polynomial ideal. This

algorithm serves as a basis for the parallel algorithm for primary decomposition which we introduce in Section 3.2. We closely follow the presentation in [19], omitting the proofs and examples. Let us start with some basic definitions and theorems.

Definition 3.1.1 ([19, Definition 4.1.1])**.** Let R be a ring.

1. A proper ideal $Q \subset R$ is a *primary ideal* if, for any two elements $a, b \in R$ with $ab \in Q$, $a \notin Q$ implies $b \in \sqrt{Q}$. In this case, the radical $P := \sqrt{Q}$ of Q is a prime ideal, and Q is also called P-primary.

2. Let $I \subset R$ be a proper ideal. A *primary decomposition* of I is an expression of the form $I = Q_1 \cap \ldots \cap Q_r$ with primary ideals $Q_i \subset R$.

3. A primary decomposition $I = Q_1 \cap \ldots \cap Q_r$ is called *irredundant* if $\bigcap_{j \neq i} Q_j \not\subset Q_i$ for all i and $\sqrt{Q_i} \neq \sqrt{Q_j}$ for all $i \neq j$.

4. For an ideal $I \subset R$, the set of *associated primes* of I, denoted by $\mathrm{Ass}(I)$, is defined as

$$\mathrm{Ass}(I) := \{P \subset R \mid P \text{ prime}, P = I : \langle a \rangle \text{ for some } a \in R\}.$$

5. Let $I \subset R$ be an ideal and let $P \in \mathrm{Ass}(I)$ be an associated prime ideal of I. If there exists an ideal $P' \in \mathrm{Ass}(I)$ with $P' \subset P$, then P is called an *embedded prime ideal* of I, otherwise it is called an *isolated prime ideal* of I.

6. An ideal I in R is called *equidimensional* if $\dim(P_1) = \dim(P_2)$ for all $P_1, P_2 \in \mathrm{Ass}(I)$.

Theorem 3.1.2 ([19, Theorem 4.1.4])**.** *Let R be a Noetherian ring, and let I be a proper ideal in R. Then I admits an irredundant primary decomposition $I = Q_1 \cap \ldots \cap Q_r$ with primary ideals $Q_i \subset R$.*

Theorem 3.1.3 (1st uniqueness theorem, [19, Theorem 4.1.5])**.** *Let R be a ring, and let $I \subset R$ be an ideal which admits an irredundant primary decomposition $I = Q_1 \cap \ldots \cap Q_r$. Then*

$$\mathrm{Ass}(I) = \left\{\sqrt{Q_1}, \ldots, \sqrt{Q_r}\right\}.$$

In particular, the set $\left\{\sqrt{Q_1}, \ldots, \sqrt{Q_r}\right\}$ is uniquely determined by I and does not depend on the particular primary decomposition.

Definition 3.1.4. Let R be a ring, and let $I \subset R$ be an ideal. A subset $\Sigma \subset \mathrm{Ass}(I)$ is called an *isolated set of associated primes* of I if, for all $P \in \Sigma$ and for all $P' \in \mathrm{Ass}(I)$, $P' \subset P$ implies $P' \in \Sigma$.

Theorem 3.1.5 (2nd uniqueness theorem, [5, Theorem 4.10]). *Let R be a ring, and let $I \subset R$ be an ideal which admits an irredundant primary decomposition $I = Q_1 \cap \ldots \cap Q_r$. Denote the associated primes of I by $P_i := \sqrt{Q_i}$, $i = 1, \ldots, r$, and let $\{P_{i_1}, \ldots, P_{i_s}\} \subset \mathrm{Ass}(I)$ be an isolated set of associated primes of I.*

Then $Q_{i_1} \cap \ldots \cap Q_{i_s}$ is independent of the primary decomposition. In particular, the primary ideals Q_i corresponding to isolated prime ideals P_i are uniquely determined by I.

For the rest of this section, we consider primary decompositions of ideals in the polynomial ring $K[x_1, \ldots, x_n]$ over some field K. Note that every proper ideal in this ring admits an irredundant primary decomposition by Theorem 3.1.2.

To start with the algorithmic aspect of primary decomposition, let us recall the algorithm for zero-dimensional ideals. In this case, all associated primes are maximal ideals. We first need the following notion.

Definition 3.1.6 ([19, Definition 4.1.1]).

1. A maximal ideal $\mathfrak{m} \subset K[x_1, \ldots, x_n]$ is called in *general position* w.r.t. the lexicographical ordering with $x_1 > \ldots > x_n$ if there exist polynomials $g_1, \ldots, g_n \in K[x_n]$ with
$$\mathfrak{m} = \langle x_1 + g_1(x_n), \ldots, x_{n-1} + g_{n-1}(x_n), g_n(x_n) \rangle .$$
2. A zero-dimensional ideal $I \subset K[x_1, \ldots, x_n]$ is called in *general position* w.r.t. the lexicographical ordering with $x_1 > \ldots > x_n$ if all its associated primes $P_1, \ldots, P_k$ are in general position and if $P_i \cap K[x_n] \neq P_j \cap K[x_n]$ for $i \neq j$.

In characteristic 0, a zero-dimensional ideal can be brought into general position by a generic coordinate change according to the following statement.

Proposition 3.1.7 ([19, Proposition 4.2.2]). *Let K be a field of characteristic 0, and let $I \subset K[x]$, $x = (x_1, \ldots, x_n)$, be a zero-dimensional ideal. Then there exists a non-empty, Zariski-open subset $U \subset K^{n-1}$ such that for all $\underline{a} = (a_1, \ldots, a_{n-1}) \in U$, the coordinate change $\varphi_{\underline{a}} : K[x] \to K[x]$ defined by*
$$\varphi_{\underline{a}}(x_i) := x_i \quad \text{for } i < n \text{ and}$$
$$\varphi_{\underline{a}}(x_n) := x_n + \sum_{i=1}^{n-1} a_i x_i$$
has the property that $\varphi_{\underline{a}}(I)$ is in general position w.r.t. to the lexicographical ordering defined by $x_1 > \ldots > x_n$.

Once the zero-dimensional input ideal is in general position, the next result gives a possibility to compute a primary decomposition thereof.

Proposition 3.1.8 ([19, Proposition 4.2.3]). *Let $I \subset K[x_1, \ldots, x_n]$ be a zero-dimensional ideal. Let $g \in K[x_n]$ be a generator of the principal ideal $\langle g \rangle = I \cap K[x_n]$, and let $g = g_1^{\nu_1} \ldots g_s^{\nu_s}$ be a factorization of g into irreducible factors $g_i \in K[x_n]$ with $g_i \neq g_j$ for $i \neq j$. Then*

$$I = \bigcap_{i=1}^{s} \langle I, g_i^{\nu_i} \rangle .$$

If I is in general position w.r.t. to the lexicographical ordering defined by $x_1 > \ldots > x_n$, then the ideals $\langle I, g_i^{\nu_i} \rangle$, $i = 1, \ldots, s$, are primary ideals, and $I = \bigcap_{i=1}^{s} \langle I, g_i^{\nu_i} \rangle$ is an irredundant primary decomposition of I.

Based on this proposition, we get the algorithm zeroDecomp below (Algorithm 3.1).

Algorithm 3.1 zeroDecomp ([19, Algorithm 4.2.7])

Input: a zero-dimensional ideal $I := \langle f_1, \ldots, f_k \rangle \subset K[x]$, $x = (x_1, \ldots, x_n)$
Output: a set of pairs (Q_i, P_i) of ideals in $K[x]$, $i = 1, \ldots, r$, such that

- $I = Q_1 \cap \ldots \cap Q_r$ is a primary decomposition of I, and
- $P_i = \sqrt{Q_i}$, $i = 1, \ldots, r$

```
result := ∅
choose a random a ∈ K^{n-1}, and apply the coordinate change I' := φ_a(I)
compute a standard basis G of I' w.r.t. the lexicographical ordering with
    x_1 > ... > x_n, and let g ∈ G be the element with the smallest leading
    monomial
factorize g = g_1^{ν_1} · ... · g_s^{ν_s} ∈ K[x_n]
for i = 1, ..., s do
    Q'_i := ⟨I', g_i^{ν_i}⟩
    Q_i := ⟨I, φ_a^{-1}(g_i)^{ν_i}⟩
    if Q'_i is primary and in general position then
        P_i := φ_a^{-1}(√Q'_i)
        result := result ∪ {(Q_i, P_i)}
    else
        result := result ∪ zeroDecomp(Q_i)
return result
```

Remark 3.1.9. In positive characteristic, Algorithm 3.1 may run into an infinite loop if it does not succeed in putting the input ideal in general position. However, the result is still correct in case the algorithm terminates.

The primary decomposition of an arbitrary ideal in $K[x_1, \ldots, x_n]$ can be reduced to the zero-dimensional case using maximal independent sets.

Definition 3.1.10 ([19, Definition 3.5.3])**.** Let $I \subset K[x_1, \ldots, x_n]$ be an ideal. Then a subset

$$u \subset x = \{x_1, \ldots, x_n\}$$

is called an *independent set* (w.r.t. I) if $I \cap K[u] = 0$. An independent set $u \subset x$ (w.r.t. I) is called *maximal* if $\dim(K[x]/I) = \#u$.

Proposition 3.1.11 ([19, Proposition 4.3.1])**.** *Let $I \subset K[x_1, \ldots, x_n]$ be an ideal, and let $u \subset x = \{x_1, \ldots, x_n\}$ be a maximal independent set of variables w.r.t. I. Then the following statements hold:*

1. *$IK(u)[x \setminus u] \subset K(u)[x \setminus u]$ is a zero-dimensional ideal.*

2. *Let $S = \{g_1, \ldots, g_s\} \subset K(u)[x \setminus u]$ be a standard basis of $IK(u)[x \setminus u]$, and let $q := \mathrm{LCM}(\mathrm{LC}(g_1), \ldots, \mathrm{LC}(g_s)) \in K[u]$. Then*

$$IK(u)[x \setminus u] \cap K[x] = I : \langle q^\infty \rangle \,,$$

   and this ideal is equidimensional of dimension $\dim(I)$.

3. *Let $IK(u)[x \setminus u] = Q_1 \cap \ldots \cap Q_r$ be an irredundant primary decomposition. Then*

$$IK(u)[x \setminus u] \cap K[x] = (Q_1 \cap K[x]) \cap \ldots \cap (Q_r \cap K[x])$$

   is also an irredundant primary decomposition.

With notation as above, let $h := q^m \in K[u]$, where $m \in \mathbb{N}$ is an integer such that $I : \langle q^m \rangle = I : \langle q^{m+1} \rangle$. Then this proposition gives an algorithm to compute a primary decomposition of the ideal $I : \langle h \rangle$. On the other hand, we have $I = (I : \langle h \rangle) \cap \langle I, h \rangle$. Thus a primary decomposition of I can be computed by recursively applying the same procedure to $\langle I, h \rangle$. This process terminates because at each step either $\dim(\langle I, h \rangle) < \dim(I)$ or the number of maximal independent sets w.r.t. $\langle I, h \rangle$ is smaller than the number of maximal independent sets w.r.t. I.

The necessary data for reducing the general case to the zero-dimensional case is computed in REDUCTIONTOZERO (Algorithm 3.2). Proceeding dimension by dimension as described above, the algorithm DECOMP (Algorithm 3.3) computes a primary decomposition of an arbitrary ideal.

Remark 3.1.12 ([19, p. 276]). The intersections $Q' \cap K[x]$ in line 3 of Algorithm 3.3 can be computed by saturation: Let Q' be given by a standard basis $\{g'_1, \ldots, g'_s\} \subset K[x]$, and let $g' := \prod_{i=1}^{s} \mathrm{LC}(g'_i) \in K[u]$, then

$$Q' \cap K[x] = \langle g'_1, \ldots, g'_s \rangle : \langle g'^\infty \rangle \subset K[x] \,.$$

Algorithm 3.2 REDUCTIONTOZERO ([19, Algorithm 4.3.2])

Input: an ideal $I := \langle f_1, \ldots, f_k \rangle \subset K[x]$, $x = (x_1, \ldots, x_n)$
Output: a list (u, G, h), where

- $u \subset x$ is a maximal independent set w.r.t. I,
- $G = \{g_1, \ldots, g_s\} \subset I$ is a standard basis of $IK(u)[x \setminus u]$,
- $h \in K[u]$ such that $IK(u)[x \setminus u] \cap K[x] = I : \langle h \rangle = I : \langle h^\infty \rangle$

1: compute a maximal independent set $u \subset x$ w.r.t. I
2: compute a standard basis $G = \{g_1, \ldots, g_s\}$ of I w.r.t. the lexicographical ordering with $x_i > x_j$ for all $x_i \in x \setminus u$ and $x_j \in u$
3: set $q := \prod_{i=1}^{s} \mathrm{LC}(g_i) \in K[u]$, where the g_i are considered as polynomials in $x \setminus u$ with coefficients in $K(u)$
4: compute $m \in \mathbb{N}$ such that $\langle g_1, \ldots, g_s \rangle : \langle q^m \rangle = \langle g_1, \ldots, g_s \rangle : \langle q^{m+1} \rangle$
5: **return** (u, G, q^m)

3.2 A Parallel Algorithm for Primary Decomposition

Modular methods have been successfully employed for computing standard bases of certain types of polynomial ideals over rings of characteristic 0, cf. [2] and [22]. The basic idea is to do the computation modulo several primes, and to lift the results back to characteristic 0 using the Chinese remainder theorem and the Farey rational map (cf. [16]). This approach has two main benefits. First, it allows us to handle examples where huge coefficients occur at the intermediate steps or in the result. This phenomenon is called coefficient swell. Second, it gives a possibility to parallelize the computation. However, a drawback of this method is that the final result has to be tested for correctness, and in some cases, this final verification step is computationally expensive.

Recently, modular methods have also been applied to the computation of the associated primes of a zero-dimensional ideal (cf. [22]). For this, the idea described above has to be slightly modified. Using modular methods for this purpose in a naive way, by computing the associated prime ideals of a given zero-dimensional ideal modulo several prime numbers, would yield two inevitable problems: First, it is not clear how to relate the associated prime ideals computed modulo some prime number p_1 to those computed modulo another prime number p_2 in order to lift the results back to characteristic 0. And second, even worse, the modular approach in [22] is build upon an algorithm which relies on polynomial factorization. But this is not compatible with modular methods in the following sense: Let $f \in \mathbb{Q}[x_1, \ldots, x_n]$ be a polynomial, and for two different prime numbers p_1, p_2, let f_1 and f_2 be the images of f mapped to the corresponding polynomial rings of characteristic

Algorithm 3.3 DECOMP ([19, Algorithm 4.3.4])

Input: an ideal $I := \langle f_1, \ldots, f_k \rangle \subset K[x]$, $x = (x_1, \ldots, x_n)$
Output: a set of pairs (Q_i, P_i) of ideals in $K[x]$, $i = 1, \ldots, r$, such that

- $I = Q_1 \cap \ldots \cap Q_r$ is a primary decomposition of I, and
- $P_i = \sqrt{Q_i}$, $i = 1, \ldots, r$

1: $(u, G, h) :=$ REDUCTIONTOZERO(I)
2: change ring to $K(u)[x \setminus u]$ and compute
qprimary := ZERODECOMP($\langle G \rangle_{K(u)[x \setminus u]}$)
3: change ring to $K[x]$ and compute
primary := $\{(Q' \cap K[x], P' \cap K[x]) \mid (Q', P') \in \text{qprimary}\}$
4: primary := primary $\cup$ DECOMP($\langle I, h \rangle$)
5: **return** primary

p_1 and p_2, respectively. Then f_1 may have a different number of irreducible factors than f_2.

The algorithm proposed in [22] avoids these problems by dividing the computation into two parts. The result of the first part is a certain uniquely determined polynomial which can be obtained using modular methods. In the second part, this polynomial is factorized in characteristic 0, and the associated primes of the input ideal are derived from its factors.

Based on the modular algorithms for standard bases and associated prime ideals, we propose a parallel algorithm for the primary decomposition of polynomial ideals in general dimension. However, applying modular methods to the algorithm of Gianni, Trager, and Zacharias (cf. Section 3.1, GTZ algorithm for short) in a naive way would lead to the same problems as described above for the computation of the associated primes. Additionally, an irredundant primary decomposition of a given ideal is in general not unique. In theory, it is possible to get rid of these problems in a way similar to the solution proposed in [22] for computing the associated primes of a zero-dimensional ideal. But since the GTZ algorithm is more complex and requires, in general, several polynomial factorizations for a given input ideal, this does not seem feasible in practice.

Instead, we base the new parallel algorithm for primary decomposition on the following four key ingredients:

1. Whenever we encounter a zero-dimensional ideal, we use the modular algorithm from [22] to compute the associated primes. The primary ideals are extracted from the associated primes by saturation. This replaces the algorithm ZERODECOMP (Algorithm 3.1).

2. We use modular methods for the essential steps in the remaining parts of the GTZ algorithm wherever they are applicable. This includes the

main standard basis computations and the ideal quotients which are needed to compute the aforementioned saturations.

3. The final result is tested for correctness using fast, specific tests. Especially the test whether the input ideal is indeed the intersection of the primary components is, for many examples, much faster than actually intersecting the primary components. This allows us to skip the time-consuming tests of the intermediate modular computations mentioned above.

4. Trivially parallelizable parts of the algorithm are parallelized. An example for this are the intersections in line 3 of Algorithm 3.3 which can be computed in parallel for multiple primary components.

These aspects will be discussed in detail in the following sections.

An implementation of the new algorithm is available in the SINGULAR library `primdec_parallel.lib` [50]. It is based on the implementation of the GTZ algorithm in SINGULAR's `primdec.lib` [44] which is highly optimized and differs in many details from the algorithm described in Section 3.1.

The parallel functionality of the library `primdec_parallel.lib` is provided by SINGULAR's new parallel framework, see Chapter 1. This framework comprises the three libraries `resources.lib` [54], `tasks.lib` [55], and `parallel.lib` [53]. All modular computations in the new algorithm rely on SINGULAR's `modular.lib` [52]. This library is an abstraction layer for modular methods which, in turn, is based on SINGULAR's parallel framework. The crucial capability of this framework in view of the presented algorithm is that the computation can be easily parallelized even on different levels. That is, tasks running in parallel to other tasks can be further parallelized in a recursive manner by dividing them into subtasks. The scheduling of the resulting tree structure is handled within the parallel framework and thus completely separated from the implementation of the parallel algorithm for primary decomposition in `primdec_parallel.lib`. An example for parallelization on different levels is the extraction of the primary ideals from the associated primes mentioned above. The primary ideals can be computed in parallel, and modular methods can be used for the saturation step which the extraction of each component involves.

For the remaining part of this section, we work over the polynomial ring $\mathbb{Q}[x_1, \ldots, x_n]$.

3.2.1 The Zero-Dimensional Case

In order to parallelize the primary decomposition of zero-dimensional ideals, we first compute the associated primes via the parallel algorithm ASSPRIMES from [22]. The corresponding primary ideals are then extracted from the associated primes using a method of Shimoyama and Yokoyama [35]. Let us first consider the theoretical background for the extraction step.

Definition 3.2.1. Let $P_1, \ldots, P_r$ be ideals in $\mathbb{Q}[x_1, \ldots, x_n]$. A *system of separators* for $P_1, \ldots, P_r$ is an r-tuple $(s_1, \ldots, s_r) \in (\mathbb{Q}[x_1, \ldots, x_n])^r$ of polynomials such that

$$s_i \in \bigcap_{j \neq i} P_j \setminus P_i$$

for all $i = 1, \ldots, r$ if $r > 1$, and $s_1 \notin P_1$ for $r = 1$.

Remark 3.2.2. Let $P_1, \ldots, P_r$ be the associated primes of a zero-dimensional ideal in $\mathbb{Q}[x_1, \ldots, x_n]$. Then the P_i are maximal ideals and thus $P_j \setminus P_i \neq \varnothing$ for $j \neq i$. For $i, j = 1, \ldots, r$ with $j \neq i$, let $p_j^{(i)}$ be an element in $P_j \setminus P_i \neq \varnothing$, and define

$$s_i := \prod_{j \neq i} p_j^{(i)} \in \mathbb{Q}[x_1, \ldots, x_n]\,.$$

For $r = 1$, set $s_1 := 1$. Then $s_i \notin P_i$ and $s_i \in P_j$ for all $i, j = 1, \ldots, r$ with $j \neq i$, that is, $(s_1, \ldots, s_r)$ is a system of separators for $P_1, \ldots, P_r$. In particular, such an r-tuple exists.

Proposition 3.2.3 (cf. [35, Theorem 2.7] and [10, Lemmata 26 and 28])**.** *Let $I \subset \mathbb{Q}[x_1, \ldots, x_n]$ be a zero-dimensional ideal. Let $P_1, \ldots, P_r$ be the associated primes of I, and let $s_1, \ldots, s_r$ be a system of separators for the prime ideals $P_1, \ldots, P_r$. Then*

$$Q_i := I : \langle s_i^\infty \rangle$$

is a P_i-primary ideal for $i = 1, \ldots, r$, and

$$I = Q_1 \cap \ldots \cap Q_r$$

is an irredundant primary decomposition of I.

Using the result above for the extraction step, zeroDecompModular (Algorithm 3.4) computes an irredundant primary decomposition of a zero-dimensional ideal in $\mathbb{Q}[x_1, \ldots, x_n]$.

Remark 3.2.4.

1. In line 1 of zeroDecompModular, we suggest to use the modular algorithm assPrimes from [22] for the computation of the associated primes. An implementation of assPrimes is available in the Singular library `assprimeszerodim.lib` [46].

2. The saturations in line 9 of zeroDecompModular are independent from each other and can thus be computed in parallel. The ideal quotients required by each saturation can be computed in a modular way. An implementation of modular ideal quotients can be found in the Singular library `modquotient.lib` [51].

Algorithm 3.4 zeroDecompModular

Input: a zero-dimensional ideal $I \subset \mathbb{Q}[x]$, $x = (x_1, \ldots, x_n)$, and an algorithm assPrimes to compute the associated prime ideals of a zero-dimensional ideal

Output: a set of pairs (Q_i, P_i) of ideals in $\mathbb{Q}[x]$, $i = 1, \ldots, r$, such that

- $I = Q_1 \cap \ldots \cap Q_r$ is a primary decomposition of I, and
- $P_i = \sqrt{Q_i}$, $i = 1, \ldots, r$

1: $P_1, \ldots, P_r :=$ assPrimes(I)
2: **if** $r = 1$ **then**
3: $\quad Q_1 := I$
4: $\quad$ **return** $\{(Q_1, P_1)\}$
5: **for** $i = 1, \ldots, r$ **do**
6: $\quad$ **for** $j = 1, \ldots, r$ with $j \neq i$ **do**
7: $\quad\quad$ choose $p_j^{(i)} \in P_j \setminus P_i$
8: $\quad s_i := \prod_{j \neq i} p_j^{(i)}$
9: $\quad Q_i := I : \langle s_i^\infty \rangle$
10: **return** $\{(Q_1, P_1), \ldots, (Q_r, P_r)\}$

3. Modular methods perform particularly well at time-consuming examples where huge coefficients occur. For small examples, they are sometimes even slower than the corresponding straightforward methods due to the additional overhead. In order to minimize this drawback, our implementation tries both zeroDecompModular and zeroDecomp in parallel and takes the result from whatever method finishes first.

In decomp, we replace zeroDecomp with zeroDecompModular and thus get the algorithm decompModular (Algorithm 3.5) below.

3.2.2 Verification

At the end of the parallel algorithm for primary decomposition which we propose, the result is tested for correctness. This allows us to skip the tests for the numerous intermediate modular computations. For many examples, testing just the final result of the primary decomposition is much faster than testing all the intermediate steps.

Given an ideal I and pairs $(Q_1, P_1), \ldots, (Q_r, P_r)$ of ideals in $\mathbb{Q}[x_1, \ldots, x_n]$, we would like to test whether $I = Q_1 \cap \ldots \cap Q_r$ is indeed a primary decomposition of I and if $P_i = \sqrt{Q_i}$ for $i = 1, \ldots, r$. For this test, we proceed in three steps and verify the following statements:

1. For $i = 1, \ldots, r$, the ideal Q_i is primary and $P_i = \sqrt{Q_i}$.

Algorithm 3.5 DECOMPMODULAR

Input: an ideal $I := \langle f_1, \ldots, f_k \rangle \subset \mathbb{Q}[x]$, $x = (x_1, \ldots, x_n)$
Output: a set of pairs (Q_i, P_i) of ideals in $\mathbb{Q}[x]$, $i = 1, \ldots, r$, such that

- $I = Q_1 \cap \ldots \cap Q_r$ is a primary decomposition of I, and
- $P_i = \sqrt{Q_i}$, $i = 1, \ldots, r$

1: $(u, G, h) :=$ REDUCTIONTOZERO(I)
2: change ring to $\mathbb{Q}(u)[x \setminus u]$ and compute
qprimary := ZERODECOMPMODULAR$(\langle G \rangle_{\mathbb{Q}(u)[x \setminus u]})$
3: change ring to $\mathbb{Q}[x]$ and compute
primary := $\{(Q' \cap \mathbb{Q}[x], P' \cap \mathbb{Q}[x]) \mid (Q', P') \in \text{qprimary}\}$
4: primary := primary $\cup$ DECOMPMODULAR$(\langle I, h \rangle)$
5: **return** primary

2. $I \subset Q_i$ for $i = 1, \ldots, r$.

3. $Q_1 \cap \ldots \cap Q_r \subset I$.

The first statement can be checked by simply computing a primary decomposition of each ideal Q_i using DECOMP. The statement is true if the decomposition has only one primary component and if the associated prime ideal coincides with P_i. While the second step is straightforward, computing the intersection in the third statement can be very time-consuming. We therefore use a trick which we present in the following.

Throughout this subsection, let R be the polynomial ring $\mathbb{Q}[x_1, \ldots, x_n]$ and let $>$ be a fixed monomial ordering on R.

Definition 3.2.5. Let $F := R^s$ be the free R-module of rank s, and let $\mathrm{e}_1, \ldots, \mathrm{e}_s$ be the canonical basis of F. We define $>_c$ to be the (module) monomial ordering on F given by

$$x^\alpha \mathrm{e}_i >_c x^\beta \mathrm{e}_j :\Leftrightarrow \begin{cases} i < j, \text{ or} \\ i = j \text{ and } x^\alpha > x^\beta \end{cases}$$

for all monomials $x^\alpha \mathrm{e}_i$ and $x^\beta \mathrm{e}_j$ in F. That is, $>_c$ gives priority to the component with the smaller index.

Notation 3.2.6. If $I \subset R$ is an ideal given by generators $f_1, \ldots, f_k$, that is $I = \langle f_1, \ldots, f_k \rangle$, then we use $(-I-)$ as a shorthand notation for the $(1 \times k)$-matrix $(f_1, \ldots, f_k)$.

The following proposition combines well-known methods for computing ideal intersections, syzygies, and intersections with free submodules. It allows us to reduce the computation of $Q_1 \cap \ldots \cap Q_r$ to a single standard basis computation of a suitable R-module.

Proposition 3.2.7. *Let $Q_1, \ldots, Q_r$ be ideals in R, let $c_1, \ldots, c_l$ denote the columns of the matrix*

$$M := \left(\begin{array}{c|ccc} 1 & -Q_1- & & \\ \vdots & & \ddots & \\ 1 & & & -Q_r- \\ \hline 1 & 0 & \cdots & 0 \end{array}\right),$$

and let $J := \langle c_1, \ldots, c_l\rangle_R \subset R^{r+1} = \bigoplus_{i=1}^{r+1} R \cdot \mathrm{e}_i$ be the R-module generated by these columns. Furthermore, let G be a standard basis of J w.r.t. $>_c$, let $S := G \cap (R \cdot \mathrm{e}_{r+1})$ be the set of elements in G whose first r components are zero, and let $\pi : R^{r+1} \to R$ be the projection given by $\pi(\mathrm{e}_i) := 0$ for $i = 1, \ldots, r$ and $\pi(\mathrm{e}_{r+1}) := 1$. Then

$$\langle \pi(S)\rangle_R = Q_1 \cap \ldots \cap Q_r\,.$$

Proof. Let $M_{r\times l}$ be the upper $(r \times l)$-submatrix of M, and let $c'_1, \ldots, c'_l$ be the columns of $M_{r\times l}$. Consider the first syzygy module of these columns, $\mathrm{Syz}(c'_1, \ldots, c'_l) \subset R^l = \bigoplus_{i=1}^{l} R \cdot \mathrm{e}_i$, and let $\psi : R^l \to R$ be the projection given by $\psi(\mathrm{e}_1) := 1$ and $\psi(\mathrm{e}_i) := 0$ for $i = 2, \ldots, l$. Then

$$\psi(\mathrm{Syz}(c'_1, \ldots, c'_l)) = Q_1 \cap \ldots \cap Q_r$$

according to [19, Lemma 2.8.4].

To get a set of generators for $\mathrm{Syz}(c'_1, \ldots, c'_l)$, let G' be a standard basis of $J' := \langle c'_1 + \mathrm{e}_{r+1}, \ldots, c'_l + \mathrm{e}_{r+l}\rangle_R$ w.r.t. the monomial ordering $>_c$ on R^{r+l}. Moreover, let $S' := G' \cap \bigoplus_{i=r+1}^{r+l} R \cdot \mathrm{e}_i$ be the set of elements in G' whose first r components are zero, and let $\pi' : R^{r+l} \to R^l$ be the canonical projection. Then

$$\mathrm{Syz}(c'_1, \ldots, c'_l) = \langle \pi'(S')\rangle_R\,,$$

cf. [19, Lemma 2.5.3].

Since the ordering $>_c$ gives priority to the component with the smaller index, the elements in $(\psi \circ \pi')(S')$ are determined by the first $(r + 1)$ components of the generators of J'. Therefore, replacing $c'_1 + \mathrm{e}_{r+1}, \ldots, c'_l + \mathrm{e}_{r+l}$ by $c_1, \ldots, c_l$ and J' by J, we have

$$Q_1 \cap \ldots \cap Q_r = \psi(\langle \pi'(S')\rangle_R) = \langle (\psi \circ \pi')(S')\rangle_R = \langle \pi(S)\rangle_R$$

which proves the claim. □

Proposition 3.2.7 gives a way to compute the intersection $Q_1 \cap \ldots \cap Q_r$. If we only want to test whether $I = Q_1 \cap \ldots \cap Q_r$ for a given ideal I, this method can be speeded up enormously by adding more columns to the matrix M as in Corollary 3.2.8 below. The additional columns make the computation much faster by anticipating the result while they do not influence its correctness. Hence the following corollary is crucial for our purpose.

Corollary 3.2.8. *With notation as above, let $I = \langle f_1, \ldots, f_k \rangle$ be an ideal in R such that $I \subset Q_1 \cap \ldots \cap Q_r$. Then Proposition 3.2.7 still holds if the matrix M is replaced by*

$$M' := \left(\begin{array}{c|ccc|c} 1 & -Q_1- & & & 0 \\ \vdots & & \ddots & & \vdots \\ 1 & & & -Q_r- & 0 \\ \hline 1 & 0 & \cdots & 0 & -I- \end{array}\right).$$

Proof. Let $c_{l+1}, \ldots, c_{l+k}$ be the additional columns in M'. Then

$$\langle c_{l+1}, \ldots, c_{l+k} \rangle_R = I \cdot \mathrm{e}_{r+1} \subset (Q_1 \cap \ldots \cap Q_r) \cdot \mathrm{e}_{r+1} = \langle S \rangle_R \subset J\,.$$

This implies that J, and thus G and S, do not change if the matrix M is replaced by M'. □

Based on this result, the primary decomposition can be tested as in testPrimdec (Algorithm 3.6).

Remark 3.2.9.

1. Various steps in the algorithm testPrimdec require computing standard bases of the input ideals I, $Q_1, \ldots, Q_r$, $P_1, \ldots, P_r$. This applies to the tests in lines 5, 8, and 12, as well as to the primary decompositions in line 2. Besides, the standard basis computation in line 11 is usually faster if the ideals $Q_1, \ldots, Q_r$, and I are given as standard bases. It is thus convenient to compute standard bases of these ideals beforehand if necessary.

2. The tests in lines 5, 8, and 12 do not take much time in comparison to the primary decomposition of I if the input ideals I, $Q_1, \ldots, Q_r$, $P_1, \ldots, P_r$ are given as standard bases, and the primary decompositions of the presumable primary ideals Q_i in line 2 are usually fast, too. So the computationally hardest part of testPrimdec is computing the standard basis in line 11 in order to check that the intersection $Q_1 \cap \ldots \cap Q_r$ is contained in I. For this step, using the idea of Corollary 3.2.8 gives an enormous speedup.

3. Several parts of testPrimdec can be parallelized, in particular the for-loops in line 1 and line 7. For the primary decompositions in line 2, several methods can be tried in parallel if they are available. Finally, we can use modular methods for the standard basis computation in line 11.

Algorithm 3.6 testPrimdec

Input: an ideal $I \subset R = \mathbb{Q}[x_1, \ldots, x_n]$ and a set of pairs (Q_i, P_i) of ideals in R, $i = 1, \ldots, r$

Output: – **true** if $I = Q_1 \cap \ldots \cap Q_r$ is a primary decomposition of I and $P_i = \sqrt{Q_i}$ for $i = 1, \ldots, r$;
– **false**, otherwise

// Step 1: check that each ideal Q_i is primary and that $P_i = \sqrt{Q_i}$
1: **for** $i = 1, \ldots, r$ **do**
2: $L := \{(q_1, p_1), \ldots, (q_s, p_s)\} :=$ decomp(Q_i)
3: **if** L has more than one component ($s > 1$) **then**
4: **return false**
5: **if** $p_1 \neq P_i$ **then**
6: **return false**

// Step 2: check that $I \subset Q_1 \cap \ldots \cap Q_r$
7: **for** $i = 1, \ldots, r$ **do**
8: **if** $I \not\subset Q_i$ **then**
9: **return false**

// Step 3: check that $Q_1 \cap \ldots \cap Q_r \subset I$
10: let J be the submodule of the free module R^{r+1} (with canonical basis $\mathrm{e}_1, \ldots, \mathrm{e}_{r+1}$) generated by the columns of the matrix

$$\left(\begin{array}{c|ccc|c} 1 & -Q_1- & & & 0 \\ \vdots & & \ddots & & \vdots \\ 1 & & & -Q_r- & 0 \\ \hline 1 & 0 & \cdots & 0 & -I- \end{array}\right)$$

11: compute a standard basis G of J w.r.t. $>_c$
12: **if** $G \cap (R \cdot \mathrm{e}_{r+1}) \not\subset I \cdot \mathrm{e}_{r+1}$ **then**
13: **return false**

// all tests succeeded
14: **return true**

In the unlikely case that testPrimdec fails, we restart the parallel primary decomposition, but this time, we do not skip the intermediate test. That is, we test the result of each intermediate modular computation for correctness and, if necessary, continue the computation with more prime numbers until the test succeeds. Thus every intermediate result is correct and we finally obtain a primary decomposition for sure. To achieve this, we call decompModular with an optional argument "with/without intermediate tests" as in primdecParallel (Algorithm 3.7) below.

Algorithm 3.7 PRIMDECPARALLEL

Input: an ideal $I := \langle f_1, \ldots, f_k \rangle \subset \mathbb{Q}[x]$, $x = (x_1, \ldots, x_n)$
Output: a set of pairs (Q_i, P_i) of ideals in $\mathbb{Q}[x]$, $i = 1, \ldots, r$, such that

- $I = Q_1 \cap \ldots \cap Q_r$ is a primary decomposition of I, and
- $P_i = \sqrt{Q_i}$, $i = 1, \ldots, r$

1: $L :=$ DECOMPMODULAR(I, "without intermediate tests")
2: **if** TESTPRIMDEC(I, L) fails **then**
3: $L :=$ DECOMPMODULAR(I, "with intermediate tests")
4: **return** L

3.2.3 Modular Methods

In this subsection, we summarize where modular methods are used in our implementation of the proposed parallel algorithm:

- After reducing the general case to the zero-dimensional case if necessary, modular methods are used to compute the associated primes of the given ideal, cf. line 1 of the algorithm ZERODECOMPMODULAR.
- We use the modular computation of ideal quotients from SINGULAR's `modquotient.lib` for the required saturations. This applies to the saturation in REDUCTIONTOZERO (line 4), to the extraction step in ZERODECOMPMODULAR (line 9), and to the intersections with $\mathbb{Q}[x]$ in DECOMPMODULAR (line 3).
- Modular standard bases are used at the following places in our implementation of the proposed algorithm:
  - within the modular computations of ideal quotients and of the associated primes, see above;
  - within the computation of independent sets in line 1 of the algorithm REDUCTIONTOZERO;
  - for the standard basis computation w.r.t. the lexicographical ordering in line 2 of REDUCTIONTOZERO;
  - for various implementational tricks and shortcuts such as to keep track of the dimension of the input ideal I at recursive calls of DECOMPMODULAR, and to take advantage of generators which are linear in one variable;
  - for the standard basis computation in Step 3 of TESTPRIMDEC (line 11).

All modular computations in our implementation rely on SINGULAR's `modular.lib` and are thus automatically parallelized. As a further advantage, the source code for the modular functionality such as the lifting to characteristic 0 is separated from our code for primary decomposition.

3.2.4 Trivially Parallelizable Parts

Aside from the modular computations mentioned in the previous subsection, the following parts of the proposed algorithm are parallelized in our implementation:

- The saturations in line 9 of zeroDecompModular are computed in parallel for $i = 1, \ldots, r$.
- In the algorithm decompModular, we try both zeroDecomp and zeroDecompModular parallelly if enough computational resources are available.
- The intersections with $\mathbb{Q}[x]$ in line 3 of decompModular are computed in parallel for every component $(Q', P') \in$ qprimary.
- In testPrimdec, both Step 1 and Step 2 are done in parallel for $i = 1, \ldots, r$.
- For the primary decomposition in line 2 of testPrimdec, several methods are tried at once.

The parallel computations mentioned above, including the modular computations in the previous subsection, are based on SINGULAR' parallel framework. This allows us to take advantage of SINGULAR's parallel functionality without having to worry about the scheduling of the tasks, even if the parallelization is done on different levels as in our case and the tasks build up a non-trivial tree structure.

3.3 Timings

We compare the algorithm primdecParallel (Algorithm 3.7) with the original algorithm of Gianni, Trager, and Zacharias (GTZ, cf. [14]) and the algorithm of Shimoyama and Yokoyama (SY, cf. [35]). For the GTZ and the SY algorithm, we use the corresponding commands from the SINGULAR library `primdec.lib`. For primdecParallel, we use the implementation from the library `primdec_parallel.lib`.

In order to show the parallel speedup, we run primdecParallel on 1, 4, 16, and 48 processor cores for each example. Note that the GTZ and the SY algorithm are sequential, that is, these timings are computed on one core only.

We compute primary decompositions of the following ideals:

1. Hawes2

$$
\begin{aligned}
f_1 &:= 5d^4 + 3d^2f + 2cd + 2de + b\,,\\
f_2 &:= 5g^4 + 3fg^2 + 2cg + 2gh + b\,,\\
f_3 &:= 20d^3 + 6df + 2c + 2e\,,\\
f_4 &:= 20g^3 + 6fg + 2c + 2h\,,\\
f_5 &:= d^2 + 3e^2 + a\,,\\
I &:= \langle f_1, \ldots, f_5\rangle \subset \mathbb{Q}[a,b,c,d,e,f,g]\,.
\end{aligned}
$$

2. Huneke

$$
\begin{aligned}
f &:= x^5 - \big(y - z^2\big)\big(y - z^3\big)\big(y - z^4\big)\big(z - t^2\big)\big(z - t^3\big) \in \mathbb{Q}[t,x,y,z]\,,\\
I &:= \left\langle \frac{\partial}{\partial t}f, \frac{\partial}{\partial x}f, \frac{\partial}{\partial y}f, \frac{\partial}{\partial z}f \right\rangle \subset \mathbb{Q}[t,x,y,z]\,.
\end{aligned}
$$

3. Laplagne4

$$
\begin{aligned}
z &:= x - 2y + 1 \in \mathbb{Q}[x,y]\,,\\
f &:= x^{18} + y^{18} + z^{18} + 2\big(x^9z^9 - x^9y^9 + y^9z^9\big) \in \mathbb{Q}[x,y]\,,\\
I &:= \left\langle f, \frac{\partial}{\partial x}f, \frac{\partial}{\partial y}f \right\rangle \subset \mathbb{Q}[x,y]\,.
\end{aligned}
$$

4. Random1

$$
\begin{aligned}
f_1 &:= 411y^5 + 654y^4z + 371y^2z^3 - 854yz^4\,,\\
f_2 &:= -871xy^4 - 490x^4z + 112y^2z^3 + 940yz^3\,,\\
f_3 &:= 661y^2z^3 + 309xz^4 + 607x^2y^2\,,\\
I &:= \langle f_1, f_2, f_3\rangle \subset \mathbb{Q}[x,y,z]\,.
\end{aligned}
$$

5. Random2

$$
\begin{aligned}
f_1 &:= 37x_1x_2x_4 + 3x_1x_4^2 - 90x_2x_3 - 66x_1x_4 - 70x_5^2 + 51x_1\,,\\
f_2 &:= -76x_1^2x_2 - 72x_2x_4^2 - 62x_1^2x_5 + 59x_2x_4x_5 + 53x_1x_5^2\,,\\
f_3 &:= 36x_1^2x_2 - 84x_1^2x_3 - 37x_1x_2x_5 + 92x_3x_4x_5\,,\\
f_4 &:= 36x_2x_3^2 + 75x_1x_2x_4 - 98x_4^3 - 77x_1x_3x_5 + 46x_3x_4x_5 - 94x_4^2x_5\,,\\
f_5 &:= 59x_1^3 - 21x_2^2x_4 + 86x_3x_4^2 - 94x_2x_4x_5 + 13x_3x_4x_5 + 37x_2x_5^2 - 89x_1^2\,,\\
f_6 &:= -59x_2x_3x_4 + 50x_2x_4^2\,,\\
I &:= \langle f_1, \ldots, f_6\rangle \subset \mathbb{Q}[x_1,x_2,x_3,x_4,x_5]\,.
\end{aligned}
$$

Table 3.1: Algebraic and geometric properties of the examples

	#Ass	Dimensions		radical
		isolated	embedded	
Hawes2	3	$3, 3, 3$	–	yes
Huneke	15	$1, \ldots, 1, 0, 0$	$0, 0$	no
Laplagne4	9	$0, \ldots, 0$	–	no
Random1	4	$1, 1, 0$	0	no
Random2	3	$1, 1$	0	no

Table 3.2: Timings comparing primdecParallel, GTZ, and SY (in seconds)

	primdecParallel				GTZ	SY
	1 core	4 cores	16 cores	48 cores		
Hawes2	20	9	9	12	148	2
Huneke	21938	8369	5210	1897	–	–
Laplagne4	194	142	42	35	173	–
Random1	351	275	244	236	–	–
Random2	26	8	9	12	249	–

Table 3.1 shows, for each of these ideals, the number of associated primes, the dimensions of the isolated and embedded components, and if the ideal is radical or not.

The timings (in seconds) for these ideals are shown in Table 3.2. They were computed on an AMD Opteron 6174 machine with 128 GB of RAM and 48 cores, each with 2.2 GHz, running a Linux operating system. A dash (–) indicates that the computation does not finish within 24 hours. For all examples, the monomial ordering is chosen to be the degree reverse lexicographical ordering (`dp` in Singular).

In some cases, the final verification step takes a considerable amount of time, and we therefore give separate timings for this step, see Table 3.3. It is worth noting that the verification succeeds for all examples presented here. Thus, a re-computation with intermediate test (cf. Algorithm 3.7) is not necessary.

The proposed parallel algorithm is faster than the original GTZ algorithm for all examples presented here. We are even able to crack examples in less than one hour for which the GTZ and the SY algorithm do not finish within a day. So far, the example named Huneke could not be solved at

Table 3.3: Timings for the verification step of primdecParallel (in seconds)

	primdecParallel			
	1 core	4 cores	16 cores	48 cores
Hawes2	4	3	4	5
Huneke	117	90	150	151
Laplagne4	7	6	6	8
Random1	167	166	167	165
Random2	3	2	2	4

all by either GTZ or SY. During the computation of this example, one of the intermediate modular standard basis computations uses more than 2000 large prime numbers. This shows that primdecParallel performs and scales particularly well for examples where very large coefficients occur at either the intermediate steps or in the result. For medium-sized examples, it still gives a speedup as the number of processor cores increases. For an algorithm as complex as primary decomposition, we cannot expect that a parallel approach scales linearly with the number of processor cores.

However, we also found examples where the parallel algorithm is slower than GTZ. Especially for small examples, the overhead inherited by the modular methods and the parallelization may even lead to longer running times as the number of cores increases. Furthermore, the final verification is a time-consuming step for some of the examples. The timings also show that this step is not yet well parallelized.

We plan to continue this research project and to tackle these issues in order to improve the proposed algorithm even further.

Part II

Syzygies

Chapter 4

New Algorithms to Compute Syzygies

Based on Schreyer's algorithm [34, 33, 11], we present two new algorithms for the computation of syzygies. The two main ideas of the first algorithm, called LiftHybrid, are the following: First, we may leave out certain terms of module elements during the computation which do not contribute to the result. These terms are called "lower order terms", see Definition 4.3.2. Second, we do not need to order the remaining terms of these module elements during the computation. This significantly reduces the number of monomial comparisons for the arithmetic operations. For the second algorithm, called LiftTree, we additionally cache some partial results and reuse them at the remaining steps.

In Section 4.1, we introduce some basic terminology. The induced ordering, Schreyer's Theorem, and the corresponding algorithm are discussed in Section 4.2. Based on an analysis of this algorithm, we present the two new algorithms in Section 4.3. A detailed example is given in Section 4.4.

We expect the new algorithms to be considerably faster, especially for large examples and for the computation of entire free resolutions. However, the implementation is not yet completely finished, so we do not present any timings here.

4.1 Introduction

Throughout this chapter, let K be a field, and let $R := K[x_1, \ldots, x_n]$ be the polynomial ring in n variables over K. We denote the monoid of monomials in $x_1, \ldots, x_n$ by $\mathrm{Mon}(x_1, \ldots, x_n)$.

We briefly recall some terminology for dealing with R-module syzygies and their computation.

Definition 4.1.1. Let $F := R^r$ be the free R-module of rank r, and let $\mathrm{e}_1, \ldots, \mathrm{e}_r$ be the canonical basis of F.

1. A *monomial* in F is the product of an element in $\mathrm{Mon}(x_1, \ldots, x_n)$ with a basis element e_i. The set of monomials in F is denoted by $\mathrm{Mon}(F)$.

2. Accordingly, a *term* in F is the product of a monomial in F with a scalar in K.

3. A monomial $m_1 \mathrm{e}_i$ *divides* a monomial $m_2 \mathrm{e}_j$ if $i = j$ and m_1 divides m_2; in this case, the *quotient* $m_2 \mathrm{e}_j / m_1 \mathrm{e}_i$ is defined as $m_2/m_1 \in \mathrm{Mon}(x_1, \ldots, x_n)$. We also say that $m_1 \in \mathrm{Mon}(x_1, \ldots, x_n)$ divides $m_2 \mathrm{e}_j$ if m_1 divides m_2, and in this case $m_2 \mathrm{e}_j / m_1$ is defined to be $(m_2/m_1) \mathrm{e}_j \in \mathrm{Mon}(F)$.

4. The *least common multiple* of two monomials $m_1 \mathrm{e}_i, m_2 \mathrm{e}_j \in F$ is

$$\mathrm{lcm}(m_1 \mathrm{e}_i, m_2 \mathrm{e}_j) := \begin{cases} \mathrm{lcm}(m_1, m_2) \mathrm{e}_i, & \text{if } i = j, \\ 0, & \text{otherwise.} \end{cases}$$

5. A *monomial ordering* on F is a total ordering $\succ$ on $\mathrm{Mon}(F)$ such that if $m_1 \mathrm{e}_i$ and $m_2 \mathrm{e}_j$ are monomials in F, and m is a monomial in R, then

$$m_1 \mathrm{e}_i \succ m_2 \mathrm{e}_j \implies (m \cdot m_1) \mathrm{e}_i \succ (m \cdot m_2) \mathrm{e}_j \ .$$

   In this chapter, we require in addition that

$$m_1 \mathrm{e}_i \succ m_2 \mathrm{e}_i \iff m_1 \mathrm{e}_j \succ m_2 \mathrm{e}_j \text{ for all } i, j \, .$$

6. Let $\succ$ be a monomial ordering on F, let $f \in F \setminus \{0\}$ be an element of F, and let $f = cm \mathrm{e}_i + f^*$ be the unique decomposition of f with $c \in K \setminus \{0\}$, $m \mathrm{e}_i \in \mathrm{Mon}(F)$, and $m \mathrm{e}_i > m^* \mathrm{e}_j$ for any non-zero term $c^* m^* \mathrm{e}_j$ of f^*. We define the *leading monomial*, the *leading coefficient*, the *leading term*, and the *tail* of f as

$$\begin{aligned} \mathrm{LM}(f) &:= m \mathrm{e}_i \, , \\ \mathrm{LC}(f) &:= c \, , \\ \mathrm{LT}(f) &:= cm \mathrm{e}_i \, , \\ \mathrm{tail}(f) &:= f - \mathrm{LT}(f) \, , \end{aligned}$$

   respectively.

7. For any subset $S \subset F$, we call

$$\mathrm{L}(S) := \langle \mathrm{LM}(f) \mid f \in S \setminus \{0\} \rangle_R \subset F$$

   the *leading module* of S.

Remark 4.1.2. Let $\succ$ be a monomial ordering on the free R-module $F := R^r$ as defined above. Then there is a unique monomial ordering $>$ on R which is compatible with $\succ$ in the obvious way, and we say that $\succ$ is *global* if $>$ is global. In this chapter, all monomial orderings are supposed to be global.

Definition 4.1.3. Let M be an R-module, let $G := \{f_1, \ldots, f_r\} \subset M$ be a finite subset of M, and let $F := R^r$ be the free R-module of rank r as above. Consider the homomorphism

$$\begin{aligned} \psi_G : F &\to M\,, \\ \mathrm{e}_i &\mapsto f_i\,. \end{aligned}$$

A *syzygy* of $G = \{f_1, \ldots, f_r\}$ is an element of $\ker \psi_G$. We call $\ker \psi_G$ the (first) *syzygy module* of G, written

$$\mathrm{Syz}(G) := \ker \psi_G.$$

Definition 4.1.4. Let N be an R-module. A *free resolution* of N is an exact sequence

$$\mathcal{F}: \quad \ldots \longrightarrow F_{i+1} \xrightarrow{\phi_{i+1}} F_i \xrightarrow{\phi_i} F_{i-1} \longrightarrow \ldots \longrightarrow F_1 \xrightarrow{\phi_1} F_0 \longrightarrow N \longrightarrow 0$$

with free R-modules F_i, $i \in \mathbb{N}$.

Remark 4.1.5. Let the notation be as in Definitions 4.1.3 and 4.1.4. In this chapter, we only consider the case where M is a free module over the polynomial ring R and where we wish to construct a free resolution of $N = M/\langle G\rangle_R$. For this, with notation as in Definition 4.1.3, set $F_0 := M$, $F_1 := F$, and $\phi_1 := \psi_G$. Now, starting with $G_1 := G$, let G_{i+1} be a finite set of generators for $\mathrm{Syz}(G_i)$ and, inductively, define ϕ_i to be the map ψ_{G_i} for $i \in \mathbb{N} \setminus \{0\}$. We then have $\mathrm{Syz}(G_i) = \ker \phi_i$, that is, $\mathcal{F}$ is obtained by repeatingly computing the syzygies of finite subsets of free R-modules.

Definition 4.1.6. Let $F_0 := R^s$ be the free R-module of rank s, let $>$ be a monomial ordering on F_0, and let $G := \{f_1, \ldots, f_r\} \subset F_0 \setminus \{0\}$ be a set of non-zero vectors in F_0.

1. We define m_{ji} as

$$m_{ji} := \frac{\mathrm{lcm}(\mathrm{LM}(f_j), \mathrm{LM}(f_i))}{\mathrm{LT}(f_i)} \in R\,.$$

2. For $i, j \in \{1, \ldots, r\}$, we define the *S-vector* of f_i and f_j as

$$S(f_i, f_j) := m_{ji} f_i - m_{ij} f_j \in \langle G\rangle_R \subset F_0\,.$$

3. For $g \in F_0$, we call an expression

$$g = g_1 f_1 + \ldots + g_r f_r + h$$

with $g_i \in R$ and $h \in F_0$ a *standard representation* for g with remainder h (and w.r.t. G and $>$) if the following conditions are satisfied:

   (a) $\mathrm{LM}(g) \geq \mathrm{LM}(g_i f_i)$ for all $i = 1, \ldots, r$ whenever both g and $g_i f_i$ are non-zero.

   (b) If h is non-zero, then $\mathrm{LT}(h)$ is not divisible by any $\mathrm{LT}(f_i)$.

Remark 4.1.7. Standard representations can be computed by multivariate division with remainder. With notation as above, let now G be a Gröbner basis, and let g be an element of $\langle G \rangle_R$. In this case, the remainder h is zero by Buchberger's criterion for Gröbner bases. For S-vectors of elements of G, each standard representation

$$S(f_i, f_j) = m_{ji} f_i - m_{ij} f_j = g_1^{(ij)} f_1 + \ldots + g_r^{(ij)} f_r$$

yields an element $m_{ji}\,\mathrm{e}_i - m_{ij}\,\mathrm{e}_j - \big(g_1^{(ij)}\,\mathrm{e}_1 + \ldots + g_r^{(ij)}\,\mathrm{e}_r\big) \in \mathrm{Syz}(G)$.

This gives one possibility to compute syzygies which we will now discuss in detail.

4.2 Schreyer's Syzygy Algorithm

4.2.1 The Induced Ordering

Definition 4.2.1. Given a monomial ordering $>$ on $F_0 := R^s$ and a set of non-zero vectors $G := \{f_1, \ldots, f_r\} \subset F_0 \setminus \{0\}$, we define the *induced ordering* on $F_1 := R^r$ (w.r.t. $>$ and G) as the monomial ordering $\succ$ given by

$$\begin{aligned} m_1\,\mathrm{e}_i \succ m_2\,\mathrm{e}_j \;:\Leftrightarrow\; & \mathrm{LT}(m_1 f_i) > \mathrm{LT}(m_2 f_j) \\ & \text{or } (\mathrm{LT}(m_1 f_i) = \mathrm{LT}(m_2 f_j) \text{ and } i > j) \end{aligned}$$

for all monomials $m_1, m_2 \in \mathrm{Mon}(x_1 \ldots, x_n)$, and for all basis elements $\mathrm{e}_i, \mathrm{e}_j \in F_1$.

This definition implies that both $>$ and $\succ$ yield the same ordering on R if restricted to one component.

Monomial comparisons w.r.t. induced orderings are computionally expensive and should therefore be avoided in practice. This holds in particular in the case of chains $(\succ_i)_{i=1,\ldots,k}$ of orderings with $\succ_{i+1}$ induced by $\succ_i$ which appear in the computation of free resolutions.

4.2.2 Schreyer's Theorem

Theorem 4.2.2 ([11, Corollary 2.3.19]). *Let $G = \{f_1, \ldots, f_r\} \subset F_0 := R^s$ be a Gröbner basis w.r.t. a monomial ordering $>$ on F_0. For each pair (f_i, f_j) with $i, j \in \{1, \ldots, r\}$, let*

$$S(f_i, f_j) = m_{ji} f_i - m_{ij} f_j = g_1^{(ij)} f_1 + \ldots + g_r^{(ij)} f_r$$

be a standard representation of the corresponding S-vector. Then the relations

$$m_{ji}\, \mathrm{e}_i - m_{ij}\, \mathrm{e}_j - \left(g_1^{(ij)}\, \mathrm{e}_1 + \ldots + g_r^{(ij)}\, \mathrm{e}_r \right) \in F_1 := R^r$$

form a Gröbner basis of $\mathrm{Syz}(G)$ *w.r.t. the monomial ordering on F_1 induced by $>$ and G. In particular, these relations generate the syzygy module* $\mathrm{Syz}(G)$.

Based on this theorem, there is an obvious algorithm for the computation of syzygy modules: Given a Gröbner basis G as above, it suffices to compute standard representations for all S-vectors $S(f_i, f_j)$ by division with remainder.

Of course, one can do much better. Since $S(f_i, f_j) = -S(f_j, f_i)$, it is sufficient to consider those pairs (f_i, f_j) with $j < i$. It is well-known that even more pairs can be left out using the following notation (cf. [11]):

Notation 4.2.3. Let $F_0 := R^s$ be the free R-module of rank s, and let $G := \{f_1, \ldots, f_r\} \subset F_0 \setminus \{0\}$ be a set of non-zero vectors in F_0. For $i = 2, \ldots, r$, we define the monomial ideal M_i as

$$M_i := \langle \mathrm{LT}(f_1), \ldots, \mathrm{LT}(f_{i-1}) \rangle : \langle \mathrm{LT}(f_i) \rangle \subseteq R\,.$$

Remark 4.2.4. Recall that if N_1 and N_2 are submodules of an R-module M, then the module quotient $N_1 : N_2$ is defined to be the ideal

$$N_1 : N_2 := \{a \in R \mid an \in N_1 \text{ for all } n \in N_2\} \subseteq R\,.$$

In particular, in the situation of Notation 4.2.3, we have $\langle m_1\, \mathrm{e}'_i \rangle : \langle m_2\, \mathrm{e}'_j \rangle = 0$ for any two monomials $m_1, m_2 \in R$ and any two basis elements e'_i, e'_j of F_0 with $i \neq j$.

Proposition 4.2.5 ([11, Theorem 2.3.10]). *Let $G = \{f_1, \ldots, f_r\} \subset F_0$ be as in Theorem 4.2.2. For each $i = 2, \ldots, r$, and for each minimal generator x^α of the monomial ideal $M_i \subset R$, let $j = j(i, \alpha) < i$ be an index such that m_{ji} divides x^α. Then it is sufficient in Theorem 4.2.2 to consider only the corresponding pairs (f_i, f_j).*

Taking this proposition into account, we get Algorithm 4.1 below.

Algorithm 4.1 SyzSchreyer

Input: A Gröbner basis $G = \{f_1, \ldots, f_r\} \subset F_0 := R^s$ w.r.t. some monomial ordering $>$

Output: A Gröbner basis of $\mathrm{Syz}(G) \subset F_1 := R^r$ w.r.t. the monomial ordering induced by $>$ and G

1: $S := \varnothing$
2: **for** $i = 2, \ldots, r$ **do**
3: **for** each minimal generator x^α of the monomial ideal M_i **do**
4: choose an index $j < i$ such that m_{ji} divides x^α
5: $h := S(f_i, f_j) = m_{ji} f_i - m_{ij} f_j \in F_0$
6: $s := m_{ji}\, \mathrm{e}_i - m_{ij}\, \mathrm{e}_j \in F_1$
7: **while** $h \neq 0$ **do**
8: choose an index λ such that $\mathrm{LT}(f_\lambda)$ divides $\mathrm{LT}(h)$
9: $h := h - \frac{\mathrm{LT}(h)}{\mathrm{LT}(f_\lambda)} f_\lambda$
10: $s := s - \frac{\mathrm{LT}(h)}{\mathrm{LT}(f_\lambda)}\, \mathrm{e}_\lambda$
11: $S := S \cup \{s\}$
12: **return** S

4.2.3 Schreyer Frame

The leading module of the syzygy module will serve as a starting point for the algorithms which we propose in Section 4.3. Its computation is based on the following observation.

Remark 4.2.6. With notation as in Theorem 4.2.2, $g_1^{(ij)} f_1 + \ldots + g_r^{(ij)} f_r$ is a standard representation of the S-vector $S(f_i, f_j)$, and therefore we have $\mathrm{LM}(m_{ji} f_i) = \mathrm{LM}(m_{ij} f_j) > \mathrm{LM}\Big(g_k^{(ij)} f_k\Big)$ for all $k = 1, \ldots, r$ with $g_k^{(ij)} \neq 0$, cf. Definition 4.1.6(3). For $i > j$, this implies

$$m_{ji}\, \mathrm{e}_i \succ m_{ij}\, \mathrm{e}_j \succ \mathrm{LM}\Big(g_k^{(ij)}\Big)\, \mathrm{e}_k\,,$$

where $\succ$ is the monomial ordering on $F_1 = R^r$ induced by $>$ and G. Therefore the leading syzygy module of G w.r.t. $\succ$ is

$$\mathrm{L}_\succ(\mathrm{Syz}(G)) = \bigoplus_{i=2,\ldots,r} M_i\, \mathrm{e}_i\ .$$

Thus, for a given Gröbner basis G, the leading module of $\mathrm{Syz}(G)$ w.r.t. the induced ordering can be easily computed by throwing away superfluous elements, see Algorithm 4.2.

In Algorithm 4.2, only the leading terms of the Gröbner basis G contribute to the computation of the set S (via the term $m_{ji} \in R$, cf. Definition 4.1.6(2)). For a free resolution as constructed in Remark 4.1.5, we can

Algorithm 4.2 LeadSyz

Input: A Gröbner basis $G = \{f_1, \dots, f_r\} \subset F_0 := R^s$ w.r.t. some monomial ordering $>$ on F_0

Output: A minimal set of generators for the leading syzygy module $\mathrm{L}_{\succ}(\mathrm{Syz}(G))$ of G w.r.t. the monomial ordering $\succ$ on $F_1 := R^r$ induced by $>$ and G

```
1:  L := ∅
2:  for 1 ≤ j < i ≤ r do
3:      t := m_ji e_i ∈ F_1
4:      for s ∈ L do
5:          if s | t then
6:              t := 0
7:              break
8:          else if t | s then
9:              L := L \ {s}
10:     if t ≠ 0 then
11:         L := L ∪ {t}
12: return L
```

thus, starting with the leading terms of G, inductively compute sets of generators for all leading syzygy modules. The sequence of these sets of leading syzygy terms is called a *Schreyer frame* by La Scala and Stillman in [28].

It is worth noting that the algorithm to compute a minimal free resolution by La Scala and Stillman is compatible with our algorithms for the computation of syzygies in the sense that both approaches are based on the Schreyer frame and can thus be combined.

Remark 4.2.7. In the computation of a free resolution, reordering the syzygies after each step may yield smaller generators for higher syzygy modules. With notation as in Remark 4.1.5, we expect that reordering G_i w.r.t. the negative degree reverse lexicographical ordering on F_{i-1} before computing G_{i+1} is generally the best choice.

4.3 New Algorithms

Throughout this section, let $G := \{f_1, \dots, f_r\} \subset F_0 := R^s$ be a Gröbner basis w.r.t. some monomial ordering $>$ and let $\succ$ be the monomial ordering on $F_1 := R^r$ induced by $>$ and G. Furthermore, let $\mathcal{L}$ be the minimal generating set of the monomial submodule $\mathrm{L}_{\succ}(\mathrm{Syz}(G)) \subset F_1$. We simply write ψ for the map $\psi_G : F_1 \to F_0$ defined by $\psi_G(\mathrm{e}_i) := f_i$ as in Definition 4.1.3.

By Remark 4.2.6, there is a one-to-one correspondence between the minimal generators of the monomial ideals M_i and the elements of $\mathcal{L}$. Instead

of processing S-pairs, we can therefore directly start with the minimal generating set of leading syzygy terms. This is equivalent to applying the chain criterion for syzygies to the set of all S-pairs, cf. [19, Lemma 2.5.10].

The algorithmic idea is that each leading syzygy term $s \in \mathcal{L}$ gives rise to a pair of indices (i, j) with $s = m_{ji}\,\mathrm{e}_i$, which, through a standard representation of the corresponding S-vector $S(f_i, f_j)$, gives rise to a syzygy $\bar{s}$ of G with $\mathrm{LT}_\succ(\bar{s}) = s$. Note that both the pair of indices and the standard representation obtained thereof are in general not unique.

This motivates the following definition.

Definition 4.3.1. Let $s \in \mathrm{L}_\succ(\mathrm{Syz}(G)) \subset F_1$ be a leading syzygy term. We call $\bar{s} \in F_1$ a *lifting* of s w.r.t. G and $\succ$ if the following conditions hold:

1. $\mathrm{LT}_\succ(\bar{s}) = s$, and
2. $\bar{s} \in \mathrm{Syz}(G)$.

If we know how to compute such a lifting, then we can use Algorithm 4.3 to obtain a generating set S of the syzygy module. Since $\mathrm{L}_\succ(S)$ is equal to $\mathrm{L}_\succ(\mathrm{Syz}(G))$, this set is even a Gröbner basis of $\mathrm{Syz}(G)$ w.r.t. $\succ$. From the computational point of view, Algorithm 4.1 can be regarded as the special case of Algorithm 4.3 where the liftings are computed by the usual reduction. This can be reformulated as in Algorithm 4.4.

Algorithm 4.3 SyzLift

Input: A Gröbner basis $G \subset F_0$ w.r.t. $>$ and an algorithm Lift to compute, for a leading syzygy term $s \in \mathrm{L}_\succ(\mathrm{Syz}(G))$, a lifting w.r.t. G and $\succ$
Output: A Gröbner basis of $\mathrm{Syz}(G) \subset F_1$ w.r.t. $\succ$

1: $\mathcal{L} := \mathrm{LeadSyz}(G)$
2: $S := \varnothing$
3: **for** $s \in \mathcal{L}$ **do**
4: $\quad \bar{s} := \mathrm{Lift}(s)$
5: $\quad S := S \cup \{\bar{s}\}$
6: **return** S

Let us now discuss algorithms for lifting leading syzygy terms in detail. LiftReduce (Algorithm 4.4) computes a lifting of a given leading syzygy term $s \in \mathrm{L}_\succ(\mathrm{Syz}(G))$ via multivariate division of the polynomial $g := \psi(s) \in \langle G \rangle \subset F_0$ by the elements of G. This is computationally the same as the division of h w.r.t. G in the while-loop of SyzSchreyer (Algorithm 4.1). At each step, the leading term of g is reduced, and this process finally reaches $g = 0$ since G is a Gröbner basis.

Let $g_1, \ldots, g_k \in F_0$ be the sequence of values which g takes when the algorithm LiftReduce is applied to a leading syzygy term $s \in \mathrm{L}_\succ(\mathrm{Syz}(G))$.

Algorithm 4.4 LiftReduce

Input: A Gröbner basis $G = \{f_1, \ldots, f_r\} \subset F_0$ w.r.t. $>$ and a leading syzygy term $s \in \mathrm{L}_{\succ}(\mathrm{Syz}(G)) \subset F_1$
Output: A lifting $\bar{s} \in \mathrm{Syz}(G) \subset F_1$ of s w.r.t. G and $\succ$

1: $g := \psi(s)$
2: $\bar{s} := s$
3: **while** $g \neq 0$ **do**
4: $\quad t := \mathrm{LT}(g)$
5: $\quad$ choose a term $m\,\mathrm{e}_i \in F_1$ with $m\,\mathrm{LT}(f_i) = t$ and $s \succ m\,\mathrm{e}_i$
6: $\quad g := g - mf_i$
7: $\quad \bar{s} := \bar{s} - m\,\mathrm{e}_i$
8: **return** $\bar{s}$

Since we have $g_k = 0$, every single term occurring in this sequence is eventually cancelled at one of the reduction steps in line 6, but only the processing of the leading terms $\mathrm{LT}(g_1), \ldots, \mathrm{LT}(g_k)$ contributes to the syzygy $\bar{s} \in \mathrm{Syz}(G)$. In particular, those terms which are not divisible by one of the leading monomials $\mathrm{LM}(f_i)$, $i = 1, \ldots, r$, do not contribute to $\bar{s}$ and can therefore be left out. We use the following terminology to refer to these terms.

Definition 4.3.2. Let $S \subset F_0$ be a set of vectors and let $t \in F_0$ be a term. Then t is called a *lower order term* w.r.t. S if

$$\mathrm{LM}(f) \nmid t \quad \text{for all } f \in S \setminus \{0\}\,.$$

For an element $g \in F_0$, we define $\mathrm{LOT}(g|S)$ to be the sum of those terms occuring in g which are of lower order w.r.t. S.

Furthermore, instead of reducing the leading term of g at a given step, we may choose any term of g which is not of lower order. Taking the above observations into account, we get the algorithm LiftHybrid (Algorithm 4.5). Note that the lower order terms which are left out at the intermediate steps sum up to zero.

We can even go further and consider the set T of terms in g rather than the polynomial g itself. In other words, we do not need to sort the terms in g and we do not need to carry out the cancellations of terms which may occur in line 6 of LiftHybrid. Then each term in T can be reduced independently as in LiftTree (Algorithm 4.6). This yields a tree structure by the recursive calls of LiftSubtree (Algorithm 4.7) for each term in T.

The algorithm applied at the root node of this tree, LiftTree, slightly differs from the algorithm applied at the other nodes, LiftSubtree. In LiftTree, the leading term of $\psi(s)$ is included in T, whereas at the other nodes, this term has been cancelled by the reduction in the previous step and is therefore left out in LiftSubtree. Because of this difference, we

Algorithm 4.5 LiftHybrid

Input: A Gröbner basis $G = \{f_1, \ldots, f_r\} \subset F_0$ w.r.t. $>$ and a leading syzygy term $s \in \mathrm{L}_\succ(\mathrm{Syz}(G)) \subset F_1$
Output: A lifting $\bar{s} \in \mathrm{Syz}(G) \subset F_1$ of s w.r.t. G and $\succ$

1: $g := \psi(s) - \mathrm{LOT}(\psi(s)|G)$
2: $\bar{s} := s$
3: **while** $g \neq 0$ **do**
4: choose a term t of g
5: choose a term $m\,\mathrm{e}_i \in F_1$ with $m\,\mathrm{LT}(f_i) = t$ and $s \succ m\,\mathrm{e}_i$
6: $g := g - (mf_i - \mathrm{LOT}(mf_i|G))$
7: $\bar{s} := \bar{s} - m\,\mathrm{e}_i$
8: **return** $\bar{s}$

need the following definition to give a proper description of the output of LiftSubtree.

Definition 4.3.3. Let $s \in F_1$ be a term. We call $\hat{s} \in F_1$ a *subtree lifting* of s w.r.t. G and $\succ$ if the following conditions hold:

1. $\mathrm{LT}_\succ(\hat{s}) = s$, and
2. all terms in $\mathrm{tail}(\psi(\hat{s})) \in F_0$ are lower order terms w.r.t. G and $\succ$.

Algorithm 4.6 LiftTree

Input: A Gröbner basis $G = \{f_1, \ldots, f_r\} \subset F_0$ w.r.t. $>$ and a leading syzygy term $s \in \mathrm{L}_\succ(\mathrm{Syz}(G)) \subset F_1$
Output: A lifting $\bar{s} \in \mathrm{Syz}(G) \subset F_1$ of s w.r.t. G and $\succ$

1: $g := \psi(s)$
2: $T :=$ set of terms in $(g - \mathrm{LOT}(g|G))$
3: $\bar{s} := s$
4: **for** all $t \in T$ **do**
5: choose a term $m\,\mathrm{e}_i \in F_1$ with $m\,\mathrm{LT}(f_i) = t$ and $s \succ m\,\mathrm{e}_i$
6: $\bar{s} := \bar{s} - \mathrm{LiftSubtree}(m\,\mathrm{e}_i)$
7: **return** $\bar{s}$

LiftTree terminates when $T = \varnothing$ is reached in every branch of the tree. One can easily check that this algorithm returns indeed a lifting of the input by comparing it to LiftHybrid.

Remark 4.3.4. Let $s \in \mathrm{L}_\succ(\mathrm{Syz}(G)) \subset F_1$ be a leading syzygy term. If $\bar{s}$ is a lifting of s w.r.t. G and $\succ$, then $\bar{s}$ is a subtree lifting of s, but the converse statement is not true in general.

Algorithm 4.7 LiftSubtree

Input: A Gröbner basis $G = \{f_1, \ldots, f_r\} \subset F_0$ w.r.t. $>$ and a term $s \in F_1$
Output: A subtree lifting $\hat{s} \in F_1$ of s w.r.t. G and $\succ$

1: $g := \psi(s) - \mathrm{LT}(\psi(s))$
2: $T :=$ set of terms in $(g - \mathrm{LOT}(g|G))$
3: $\hat{s} := s$
4: **for** all $t \in T$ **do**
5: choose a term $m\,\mathrm{e}_i \in F_1$ with $m\,\mathrm{LT}(f_i) = t$
6: $\hat{s} := \hat{s} -$ LiftSubtree$(m\,\mathrm{e}_i)$
7: **return** $\hat{s}$

For any term $s' \in F_1$, a proper lifting of s' (w.r.t. G and $\succ$) exists if and only if s' is leading syzygy term, that is, an element of $\mathrm{L}_\succ(\mathrm{Syz}(G))$. Hence we cannot expect to find proper liftings of the terms $m\,\mathrm{e}_i \in F_1$ to which LiftSubtree is applied in Algorithms 4.6 and 4.7.

Remark 4.3.5. The condition $s \succ m\,\mathrm{e}_i$ in line 5 of Algorithm 4.6 is always satisfied at the analogous step in Algorithm 4.7 and thus does not need to be checked there.

Remark 4.3.6. Let $m_1\,\mathrm{e}_{i_1}, \ldots, m_k\,\mathrm{e}_{i_k} \in F_1$ be the sequence of terms chosen in line 5 of LiftReduce when this algorithm is applied to a leading syzygy term $s \in \mathrm{L}_\succ(\mathrm{Syz}(G))$. Then we have

$$s \succ m_1\,\mathrm{e}_{i_1} \succ \ldots \succ m_k\,\mathrm{e}_{i_k}\ .$$

However, the terms $m\,\mathrm{e}_i \in F_1$ chosen in LiftHybrid, LiftTree, and LiftSubtree satisfy $s \succ m\,\mathrm{e}_i$, but they are not necessarily ordered.

LiftTree has two main advantages in comparison to LiftHybrid. First, no reductions as in line 6 of LiftHybrid occur. Second, the results of LiftSubtree can be cached and reused. We will see an example for this in the next section.

4.4 Example

In this section, we give an example in order to illustrate the differences between the three approaches which we presented in the previous section. The example has been chosen in such a way that it shows the benefits, but also possible drawbacks of the new methods. However, note that a considerable speed-up can only be expected for large examples.

Throughout this section, let $F_0 := R := \mathbb{Q}[x, y, z]$ be endowed with the lexicographical ordering, denoted by $>$. We compute the first syzygy module

of $G := (f_1, f_2, f_3) \subset R$ with

$$\begin{aligned} f_1 &:= xy + x + y^2 + 2y - 1\,, \\ f_2 &:= xz - x - y - z - 2\,, \\ f_3 &:= yz + 1\,. \end{aligned}$$

Note that G is a Gröbner basis w.r.t. $>$. Let $\succ$ be the Schreyer ordering on $F_1 := R^3$ induced by $>$ and G. We can use Algorithm 4.2 to check that a minimal generating set of the leading syzygy module of G w.r.t. $\succ$ is given by

$$\mathcal{L} = \{y \cdot \mathrm{e}_2,\, x \cdot \mathrm{e}_3\} \subset F_1\,.$$

Our goal is to extend these leading syzygy terms to generators of the syzygy module. Let us first consider the usual LiftReduce approach (Algorithm 4.4). Flow charts of LiftReduce applied to the two leading syzygy terms above are shown in Figure 4.1.

They start with the input term on the syzygy level and its image g under $\psi : F_1 \to F_0$, $\mathrm{e}_i \mapsto f_i$, on the level of F_0. At each step, the leading term of g is reduced w.r.t. G while $\bar{s}_1$ and $\bar{s}_2$ keep track of these reductions. Both charts have the shape of a chain because every step depends on the previous one. We could choose a different reduction at the first step of the diagram on the right hand side, but there is no other choice at the other steps. The process ends when $g = 0$ is reached, and we finally get the syzygies

$$\begin{aligned} \bar{s}_1 &= (-z+1) \cdot \mathrm{e}_1 + (y+1) \cdot \mathrm{e}_2 + (y+3) \cdot \mathrm{e}_3 \in \mathrm{Syz}(G) \quad \text{and} \\ \bar{s}_2 &= (-z) \cdot \mathrm{e}_1 + \mathrm{e}_2 + (x+y+2) \cdot \mathrm{e}_3 \in \mathrm{Syz}(G) \end{aligned}$$

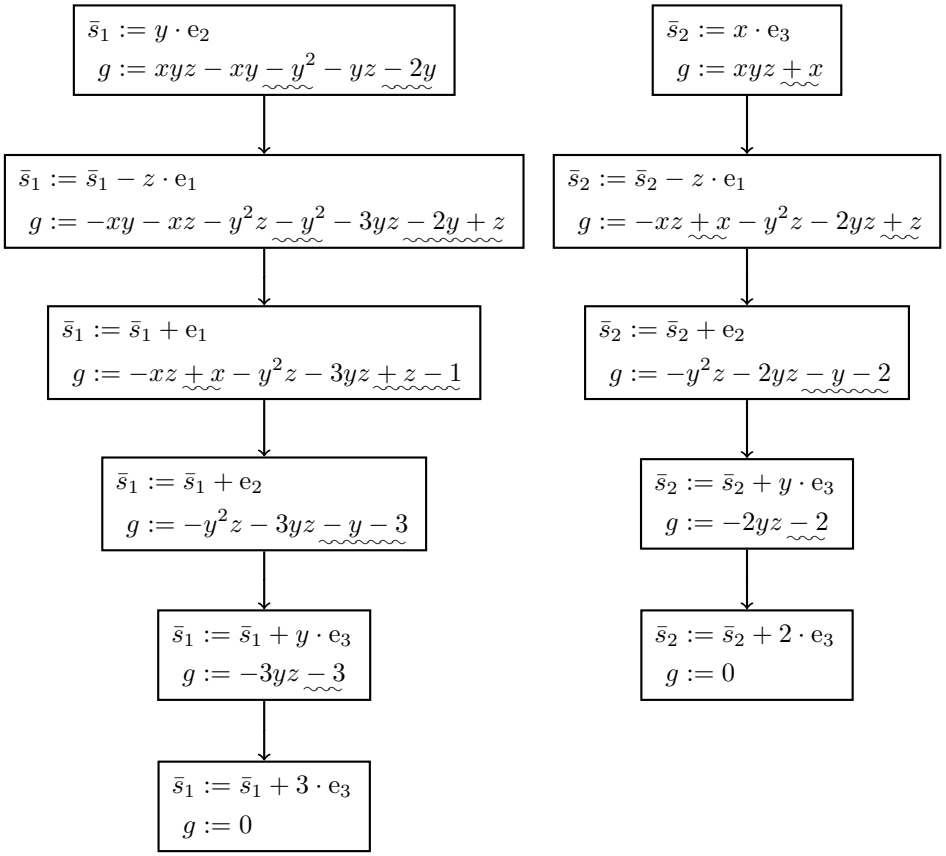
as liftings of the leading syzygy terms $y \cdot \mathrm{e}_2$ and $x \cdot \mathrm{e}_3$, respectively.

The main innovation of the algorithm LiftHybrid (Algorithm 4.5) is to leave out the lower order terms in g. This in turn allows us to choose, at each step, any of the remaining terms for reduction, in contrast to LiftReduce. Hence the terms in g do not have to be ordered at all. If we always choose, however, to reduce the leading term as in LiftReduce, then the flow charts of LiftHybrid applied to $y \cdot \mathrm{e}_2$ and $x \cdot \mathrm{e}_3$, respectively, can be obtained from those for LiftReduce by leaving out the underlined lower order terms, see Figure 4.1.

In LiftTree (Algorithm 4.6), the polynomial g is replaced by a set of terms denoted by T and each term is treated independently. The corresponding flow charts in Figure 4.2 and Figure 4.3 thus have a tree structure where each node represents one of the recursive calls of LiftTree and LiftSubtree. In Figure 4.3, the result of LiftTree$(y \cdot \mathrm{e}_2)$ can be read off as the sum of all the terms $\bar{s}_1$. Similarly, LiftTree$(x \cdot \mathrm{e}_3)$ yields $(-z) \cdot \mathrm{e}_1 + \mathrm{e}_2 + (x+y+2) \cdot \mathrm{e}_3$.

Again, the underlined lower order terms are left out. The process ends when $T = \varnothing$ is reached in every branch. It is worth noting that although each

Figure 4.1: LiftReduce/LiftHybrid applied to $y \cdot \mathrm{e}_2$ and $x \cdot \mathrm{e}_3$



step resembles a reduction step, no reductions as in the first two approaches occur. The main advantage of the LiftTree approach is that intermediate results can be cached and reused. In Figure 4.3, the term xyz occurs as an element of T, but the whole subtree which corresponds to this element has already been computed when LiftTree was applied to $y \cdot \mathrm{e}_2$ in Figure 4.2 and we can therefore just plug in the cached result.

A possible drawback of this method can be observed in Figure 4.2: Two steps are necessary to compute the term $(3 \cdot \mathrm{e}_3)$ in the result whereas LiftReduce and LiftHybrid need only one step for this. On the other hand, we could also cache the result of LiftSubtree(e_3) and reuse it for the computation of LiftSubtree$(2\,\mathrm{e}_3)$, of course.

Figure 4.2: LiftTree applied to $y \cdot \mathrm{e}_2$

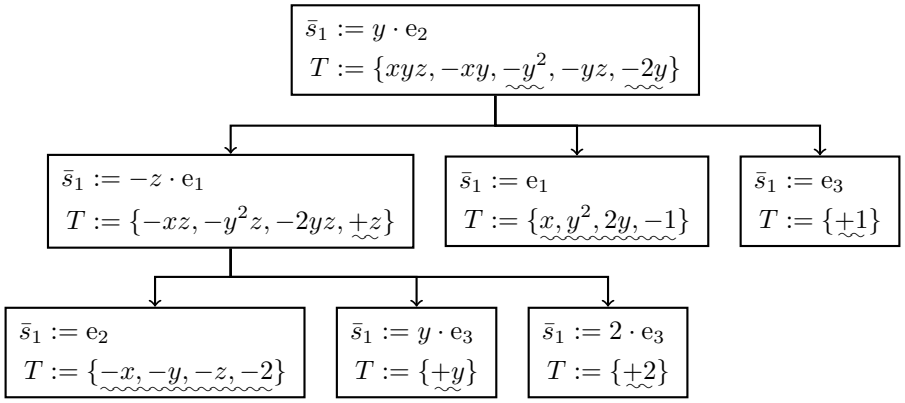
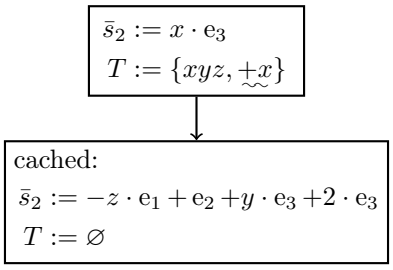


Figure 4.3: LiftTree applied to $x \cdot \mathrm{e}_3$



Part III

Real Singularities

Chapter 5

Algorithmic Classification of the Simple Real Singularities

We present algorithms to classify isolated hypersurface singularities over the real numbers according to the classification by V.I. Arnold [4]. This chapter covers the splitting lemma and the simple singularities. We plan to continue this research project and to provide algorithms for the classification of the unimodal real singularities up to corank 2 as well. Chapter 6 is the first important step in this direction. All algorithms are implemented in the SINGULAR library `realclassify.lib` [48].

5.1 Introduction

Arnold et al. [4] present classification theorems for singularities over the complex numbers up to modality 2 and for singularities over the real numbers up to modality 1, including complete sets of normal forms. For the complex case, they also give an algorithm how the type of a given singularity can be computed, called the "determinator of singularities" (cf. [4, Chapter 16]), but this question is left open for the real case. The goal of this chapter is to fill this gap for the simple singularities. For this purpose, we present both, algorithms and an implementation thereof, for the classification of the simple hypersurface singularities over the real numbers w.r.t. right equivalence.

We consider real functions with a critical point at the origin and critical value 0, i.e. functions in $\mathfrak{m}^2$, where $\mathfrak{m}$ denotes the ideal of function germs vanishing at the origin. Two function germs $f, g \in \mathfrak{m}^2 \subset \mathbb{R}[[x_1, \ldots, x_n]]$ are considered as right equivalent, denoted by $f \overset{r}{\sim} g$, if there exists an $\mathbb{R}$-algebra automorphism ϕ of $\mathbb{R}[[x_1, \ldots, x_n]]$ such that $\phi(f) = g$.

We have implemented all the algorithms presented here in the computer algebra system SINGULAR [39]. The implementation is freely available as a SINGULAR library called `realclassify.lib` [48] which relies on SINGULAR's `classify.lib` [47] to determine, for a given polynomial, the type in Arnold's

classification over the complex numbers. The methods used in `classify.lib` will not be discussed in this chapter. For more information in this regard, [27] can be studied.

In Section 5.2, we introduce basic notions and methods which are frequently used for the algorithmic classification in the subsequent sections. We first give an overview of the different notions of equivalence in singularity theory and how they are related in Subsection 5.2.1. Thereafter we recall some basic results on the Milnor number and the determinacy in Subsections 5.2.2 and 5.2.3, and we also recall how these invariants can be computed. As a further prerequisite, we show that the homogeneous parts of lowest degree of two right equivalent functions factorize in the same way over $\mathbb{R}$ (Section 5.2.4, Proposition 5.2.8). We also show that in some cases, this factorization can even be carried out over $\mathbb{Q}$ which is important for the algorithmic aspect (Lemma 5.2.9).

Using the Splitting Lemma (Theorem 5.3.2), any function germ f over the real numbers with an isolated singularity at the origin can be written, after choosing a suitable coordinate system, as the sum of two functions of which the variables are disjoint. One of the functions, called the nondegenerate part of f, is a nondegenerate quadratic form and the other function, called the residual part of f, is an element of $\mathfrak{m}^3$. The number of variables in the residual part is equal to the corank of f, denoted by $\operatorname{corank}(f)$. In this chapter, we only consider germs with corank 0, 1 and 2. A version of the Splitting Lemma for singularities over $\mathbb{R}$ and a corresponding algorithm are discussed in Section 5.3.

In [4], the real singularities of modality 0 and 1 are classified up to stable equivalence into main types which split up into more subtypes depending on the sign of certain terms. Two functions are stably equivalent if they are right equivalent after the direct addition of nondegenerate quadratic forms. Hence after applying the Splitting Lemma, we only need to consider the residual part in order to compute the correct subtype. It can be easily seen that the subtypes are complex equivalent to a complex singularity type of the same name as its corresponding real main singularity type (see Table 5.1). In fact there is a bijection between the complex types of modality 0 and 1 and the real main types. Thus, if we can determine the complex type of a function germ, we only need to determine the correct subtype of the corresponding real main type. The classification of the residual part is given in Section 5.4, together with explicit algorithms for each singularity type.

5.2 Prerequisites

5.2.1 Equivalence

There are different notions for the equivalence of two power series in singularity theory:

Definition 5.2.1. Let $\mathbb{K}$ be either $\mathbb{R}$ or $\mathbb{C}$ and let $f, g \in \mathbb{K}[[x_1, \ldots, x_n]]$ be two power series.

1. f and g are called *right equivalent*, denoted by $f \overset{r}{\sim} g$, if there exists a $\mathbb{K}$-algebra autmorphism ϕ of $\mathbb{K}[[x_1, \ldots, x_n]]$ such that
$$\phi(f) = g\,.$$
2. f and g are called *contact equivalent*, denoted by $f \overset{c}{\sim} g$, if there exist a $\mathbb{K}$-algebra autmorphism ϕ of $\mathbb{K}[[x_1, \ldots, x_n]]$ and a unit $u \in \mathbb{K}[[x_1, \ldots, x_n]]^*$ such that
$$\phi(f) = u \cdot g\,.$$
3. f and g are called *stably equivalent*, denoted by $f \overset{s}{\sim} g$, if there exist indices $k, l \in \{1, \ldots, n\}$ such that $f \in \mathbb{K}[[x_1, \ldots, x_k]]$, $g \in \mathbb{K}[[x_1, \ldots, x_l]]$, and the two power series become right equivalent after the addition of nondegenerate quadratic forms in the additional variables, i.e.
$$\begin{aligned} & f(x_1, \ldots, x_k) \pm x_{k+1}^2 \pm \ldots \pm x_n^2 \\ \overset{r}{\sim}\ & g(x_1, \ldots, x_l) \pm x_{l+1}^2 \pm \ldots \pm x_n^2\,. \end{aligned}$$

Remark 5.2.2. Note that right equivalence implies both contact and stable equivalence, but the converse statements are not true in general. For instance, $x_1^2 + x_2^2$ and $-x_1^2 - x_2^2$ are contact, but not right equivalent over $\mathbb{R}$.

This chapter and the SINGULAR library `realclassify.lib` both deal with the classification of the simple singularities w.r.t. *right* equivalence over $\mathbb{K} = \mathbb{R}$. We first use the Splitting Lemma and Algorithm 5.2 from Section 5.3 to get rid of the nondegenerate part. We can then apply the classification by Arnold et al. [4] w.r.t. stable equivalence to the residual part in Section 5.4.

From the point of view of real algebraic geometry, a classification w.r.t. contact rather than right equivalence might be more interesting because it better reflects the local real geometry of a singularity. In the example from the remark above, $x_1^2 + x_2^2$ and $-x_1^2 - x_2^2$ both define a solitary point in the plane, as opposed to the two intersecting lines defined by $-x_1^2+x_2^2$ and $x_1^2-x_2^2$. But note that a classification w.r.t. right equivalence is only finer than one based on contact equivalence. Hence the shape of the local real geometry of a singularity can always be read off from its right equivalence class, given by its stable equivalence class together with the inertia index introduced in Theorem 5.3.2; the SINGULAR library `realclassify.lib` indeed also serves this purpose. For the simple singularities, it is moreover easy to see which of the right equivalence classes are contact equivalent.

5.2.2 The Milnor Number

We briefly recall the following well-known definition:

Definition 5.2.3. For $f \in \mathbb{R}[[x_1, \ldots, x_n]]$ and $p \in \mathbb{A}^n_{\mathbb{R}}$, the *Milnor number* of f at p is defined as

$$\mu(f,p) := \dim_{\mathbb{R}}\left(\mathbb{R}[[x_1 - p_1, \ldots, x_n - p_n]] \Big/ \left\langle \frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n} \right\rangle\right) \in \mathbb{N} \cup \{\infty\}\,.$$

If p is the origin, we simply write $\mu(f)$ instead of $\mu(f,p)$.

The Milnor number is known to be finite at isolated singularities (cf. [18, Chapter I, Lemma 2.3]) and to be invariant under right equivalence (cf. Lemma 2.10 ibid.). It is thus an important tool for the classification of isolated singularities. We refer to [18] for more properties of this invariant.

There is a well-known algorithm for the computation of the Milnor number which is implemented in SINGULAR, see [19, pp. 526–528].

5.2.3 The Determinacy

In general, the singularities we deal with in this chapter are defined by power series, but algorithmically, we want to work with polynomials. It is thus important for our algorithmic approach that any power series defining an isolated singularity is right equivalent to a polynomial which can be obtained from it by leaving out terms of sufficiently high order.

Definition 5.2.4. Let $f \in \mathbb{R}[[x_1, \ldots, x_n]]$ be a power series.

1. Let $f = \sum_{j=0}^{\infty} f_j$ be the decomposition of f into homogeneous parts f_j of degree j. For $k \in \mathbb{N}$, we define the *k-jet* of f as

$$\mathrm{jet}(f,k) := \sum_{i=0}^{k} f_i\,.$$

   In other words, the k-jet of f can be obtained from f by leaving out all terms of order higher than k.

2. f is called *k-determined* if

$$f \overset{r}{\sim} \mathrm{jet}(f,k) + g \quad \text{for all } g \in \mathfrak{m}^{k+1}\,.$$

The determinacy is, just as the Milnor number, both invariant under right equivalence and finite for isolated singularities. We cite the following statement (cf. [18, Chapter I, Supplement to Theorem 2.23]) due to its importance for the algorithmic approach and refer to [18] for further results regarding the determinacy:

Proposition 5.2.5. *Let $f \in \mathfrak{m} \subset \mathbb{R}[[x_1, \ldots, x_n]]$. If*

$$\mathfrak{m}^{k+1} \subset \mathfrak{m}^2 \left\langle \frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n} \right\rangle_{\mathbb{R}[[x_1,\ldots,x_n]]}$$

holds, then f is k-determined.

As a consequence of this, any power series f which has an isolated singularity at the origin is $(\mu(f) + 1)$-determined (cf. [18, Chapter I, Corollary 2.24]). But we can often compute a much better upper bound for the determinacy by using the above statement as in Algorithm 5.1.

Algorithm 5.1 DETERMINACY

Input: $f \in \mathbb{Q}[x_1, \ldots, x_n]$ with an isolated singularity at the origin
Output: an upper bound for the determinacy of f

1: $k :=$ MILNOR$(f) + 1$
2: $J := \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}\right) \subset \mathbb{Q}[x_1, \ldots, x_n]$
3: compute a standard basis G of $(\mathfrak{m}^2\, J)$ w.r.t. a local monomial ordering $<$
4: **for** $(l = 1, \ldots, k-1)$ **do**
5: **if** $(\mathrm{NF}_<(\mathfrak{m}^{l+1}, G) = 0)$ **then**
6: $k := l$
7: **break**
8: **return** k

Remark 5.2.6. In Algorithm 5.1, the for-loop computes the minimal $k \in \mathbb{N}$ such that the condition in Proposition 5.2.5 holds. This number is equal to the degree of the so-called highest corner of $\langle G \rangle = (\mathfrak{m}^2\, J)$ (cf. [19, Corollary A.9.7]) and can thus also be computed by combinatorial means with the SINGULAR command `highcorner()` which is often much faster.

It is worth to note that both the Milnor number and the determinacy of an arbitrary power series $f \in \mathbb{R}[[x_1, \ldots, x_n]]$ do not change if we regard f as an element of $\mathbb{C}[[x_1, \ldots, x_n]]$. The same holds for the output of the corresponding algorithms presented here.

5.2.4 Results Regarding the Factorization of Homogeneous Polynomials over $\mathbb{R}$ and $\mathbb{Q}$

Definition 5.2.7. Let ϕ be an $\mathbb{R}$-algebra automorphism of $\mathbb{R}[[x_1, \ldots, x_n]]$. For $j \geq 0$ we define the *j-jet* of ϕ, denoted by ϕ_j, to be the automorphism given by

$$\phi_j(x_i) := \mathrm{jet}(\phi(x_i),\, j+1) \quad \text{for all } i = 1, \ldots, n\,.$$

The next result is in many cases a starting point for the algorithmic classification of the residual part, see Section 5.4. Given f and g with $f \overset{r}{\sim} g$, it can be used to determine ϕ_0 for some automorphism ϕ such that $\phi(f) = g$.

Proposition 5.2.8. *Let $f, g \in \mathbb{R}[[x_1, \ldots, x_n]]$ be two power series with $f \overset{r}{\sim} g$ and $k := \operatorname{ord}(f) > 1$. Let ϕ be an $\mathbb{R}$-algebra automorphism of $\mathbb{R}[[x_1, \ldots, x_n]]$ such that $\phi(f) = g$.*

If $\operatorname{jet}(f, k)$ *factorizes as*

$$\operatorname{jet}(f, k) = f_1^{s_1} \cdots f_t^{s_t}$$

in $\mathbb{R}[x_1, \ldots, x_n]$, *then* $\operatorname{jet}(g, k)$ *factorizes as*

$$\operatorname{jet}(g, k) = \phi_0(f_1)^{s_1} \cdots \phi_0(f_t)^{s_t}\,.$$

Proof. By assumption we have that $f = f_1^{s_1} \cdots f_t^{s_t} + f'$, where $f_1^{s_1} \cdots f_t^{s_t}$ is homogeneous of degree k and the order of f' is greater than k. We denote the higher order parts of ϕ by $\phi^* := \phi - \phi_0$. Since ϕ is a homomorphism, it follows that

$$\begin{aligned}\phi(f) &= \phi(f_1^{s_1} \cdots f_t^{s_t}) + \phi(f') \\ &= \phi_0(f_1^{s_1} \cdots f_t^{s_t}) + \phi^*(f_1^{s_1} \cdots f_t^{s_t}) + \phi(f')\end{aligned}$$

where $\phi_0(f_1^{s_1} \cdots f_t^{s_t})$ is homogeneous of degree k and both $\phi^*(f_1^{s_1} \cdots f_t^{s_t})$ and $\phi(f')$ are of order higher than k. Hence

$$\operatorname{jet}(g, k) = \operatorname{jet}(\phi(f), k) = \phi_0(f_1^{s_1} \cdots f_t^{s_t}) = \phi_0(f_1)^{s_1} \cdots \phi_0(f_t)^{s_t}\,.$$

□

Since we do not want to work with rounding errors nor field extensions in the implementation of the proposed algorithms, the above result would not be of much help for this purpose without the following result.

Lemma 5.2.9. *If $f \in \mathbb{Q}[x, y]$ is homogeneous and factorizes as*

$$\textit{(i) } g_1^d \textit{ or (ii) } g_1 g_2^d\,,$$

where $g_1, g_2 \in \mathbb{R}[x, y]$ are polynomials of degree 1 and $d > 1$, then f factorizes as

$$\textit{(i) } a g_1'^d \textit{ or (ii) } a g_1' g_2'^d\,,$$

respectively, where $g_1', g_2' \in \mathbb{Q}[x, y]$ are polynomials of degree 1 and $a \in \mathbb{Q}$.

Proof. (i) Let $f = (a_1 x + a_2 y)^d$, $a_1, a_2 \in \mathbb{R}$. Without loss of generality, suppose $a_1 \neq 0$. Then $f = a_1^d (x + \frac{a_2}{a_1} y)^d$. Since the coefficient of x^d in $f \in \mathbb{Q}[x, y]$ is a_1^d, we have $a_1^d \in \mathbb{Q}$ and therefore $(x + \frac{a_2}{a_1} y)^d \in \mathbb{Q}[x, y]$ which, by dehomogenization, leads to $(x + \frac{a_2}{a_1})^d \in \mathbb{Q}[x]$. Since $\mathbb{Q}$ is a perfect field it follows that $\frac{a_2}{a_1} \in \mathbb{Q}$. Thus $f = a g_1'^d$, where $a := a_1^d \in \mathbb{Q}$ and $g_1' = x + \frac{a_2}{a_1} y \in \mathbb{Q}[x, y]$.

(ii) Let $f = (a_1 x + a_2 y)(a_3 x + a_4 y)^d$, $a_1, \ldots, a_4 \in \mathbb{R}$. Suppose $a_1, a_3 \neq 0$. For the cases $a_1, a_4 \neq 0$, $a_2, a_3 \neq 0$ and $a_2, a_4 \neq 0$ the proofs are similar. We

have $a_1 a_3^d \in \mathbb{Q}$ analogously to part (i). Hence $(x + \frac{a_2}{a_1}y)(x + \frac{a_4}{a_3}y)^d \in \mathbb{Q}[x, y]$ which in turn implies $(x + \frac{a_2}{a_1})(x + \frac{a_4}{a_3})^d \in \mathbb{Q}[x]$. Since $\mathbb{Q}$ is a perfect field it follows that the roots of this polynomial are rational. Therefore $f = a g_1' g_2'^d$ with $a := a_1 a_3^d \in \mathbb{Q}$, $g_1' := (x + \frac{a_2}{a_1}y) \in \mathbb{Q}[x, y]$, and $g_2' := (x + \frac{a_4}{a_3}y) \in \mathbb{Q}[x, y]$. □

5.3 The Splitting Lemma

Definition 5.3.1. For $f \in \mathbb{R}[[x_1, \ldots, x_n]]$, we define the corank of f, denoted by corank(f), as the corank of the Hessian matrix $H(f)$ at $\mathbf{0}$, i.e.

$$\operatorname{corank}(f) := \operatorname{corank}(H(f)(\mathbf{0}))\,.$$

The following well-known theorem, called the Splitting Lemma, allows us to reduce the classification to germs of full corank or, algorithmically, to a polynomial contained in $\mathfrak{m}^3 \cap \mathbb{R}[x_1, \ldots, x_c]$ for a given input polynomial of corank c. We present a version for singularities over the real numbers, taking into account the signs of the squares.

Theorem 5.3.2. *If $f \in \mathfrak{m}^2 \subset \mathbb{R}[[x_1, \ldots, x_n]]$ has an isolated singularity and if its corank is c, then*

$$f \overset{r}{\sim} g - \sum_{i=c+1}^{c+\lambda} x_i^2 + \sum_{i=c+\lambda+1}^{n} x_i^2$$

with $g \in \mathfrak{m}^3 \cap \mathbb{R}[[x_1, \ldots, x_c]]$. g is called the residual part of f and λ is called the inertia index of f. Both λ and the right equivalence class of g are uniquely determined by f.

The following proof is based upon the proofs of the Theorems 2.46 and 2.47 in Chapter I of [18].

Proof. The corank of the Hessian matrix of f at 0 is c, so by the theory of quadratic forms over $\mathbb{R}$ there is a transformation matrix T such that

$$T^t \cdot \tfrac{1}{2} H(f)(\mathbf{0}) \cdot T = \operatorname{diag}(0, \ldots, 0, -1, \ldots, -1, 1, \ldots, 1)\,.$$

Therefore the linear coordinate change $(x_1, \ldots, x_n) \mapsto (x_1, \ldots, x_n) \cdot T^t$ transforms the 2-jet of f into $\left(-\sum_{i=c+1}^{c+\lambda} x_i^2 + \sum_{i=c+\lambda+1}^{n} x_i^2\right)$ where λ is the inertia index of f. Applied to f, this transformation leads to

$$\begin{aligned} f^{(3)}(x_1, \ldots, x_n) &:= f((x_1, \ldots, x_n) \cdot T^t) \\ &= g_3 - \sum_{i=c+1}^{c+\lambda} x_i^2 + \sum_{i=c+\lambda+1}^{n} x_i^2 + \sum_{i=c+1}^{n} x_i \cdot h_i^{(3)} \end{aligned}$$

with $g_3 \in \mathfrak{m}^3 \cap \mathbb{R}[[x_1, \ldots, x_c]]$ and $h_i^{(3)} \in \mathfrak{m}^2$. The coordinate change $\phi^{(3)}$ defined by

$$\phi^{(3)}(x_i) := \begin{cases} x_i, & i = 1, \ldots, c, \\ x_i + \frac{1}{2} h_i^{(3)}, & i = c+1, \ldots, c+\lambda, \\ x_i - \frac{1}{2} h_i^{(3)}, & i = c+\lambda+1, \ldots, n. \end{cases}$$

yields

$$\begin{aligned} f^{(4)}(x_1, \ldots, x_n) &:= f^{(3)}(\phi^{(3)}(x_1, \ldots, x_n)) \\ &= g_3 + g_4 - \sum_{i=c+1}^{c+\lambda} x_i^2 + \sum_{i=c+\lambda+1}^{n} x_i^2 + \sum_{i=c+1}^{n} x_i \cdot h_i^{(4)} . \end{aligned}$$

with $g_4 \in \mathfrak{m}^4 \cap \mathbb{R}[[x_1, \ldots, x_c]]$ and $h_i^{(4)} \in \mathfrak{m}^3$. Continuing in the same manner, the last sum will be of arbitrarily high order. It can be eventually left out because f is finitely determined as an isolated singularity. □

Since this proof is constructive, we can immediately derive Algorithm 5.2 from it.

5.4 The Real Classification of the Residual Part w.r.t. Stable Equivalence

Arnold et al. [4] present independent classifications of the simple singularities over the complex and over the real numbers, using stable equivalence. We refer to the equivalence classes of the complex classification as *complex types*. In the classification over the real numbers, the simple singularities are divided into *main types* which split up into one or more *subtypes*. These subtypes differ from each other only in the sign of certain terms.

It is known that the modality does not decrease under complexification [4, pp. 273-274]. So by applying the algorithms for the complex classification to the real normal forms, it is easy to see that in modality 0, there is a one-to-one correspondence between the complex types and the real main types. The real classification can thus be seen as a refinement of the complex one. The same holds true also in modality 1, but in both cases, this is not clear a priori and can only be deduced from the independently derived complex and real classifications. In fact, it is not known whether the modality is preserved under complexification in general [4, pp. 273-274].

Both the real and complex normal forms of the simple singularities are listed in Table 5.1. From here onwards we will work with stable equivalence, cf. Definition 5.2.1(3). For all degenerate forms it is thus only necessary, after applying the Splitting Lemma, to consider their residual parts, i.e. germs in $\mathfrak{m}^3$. Note that the right equivalence class of a real singularity is

Algorithm 5.2 Algorithm for the Splitting Lemma

Input: $f \in \mathfrak{m}^2 \subset \mathbb{Q}[x_1, \ldots, x_n]$ and $k \in \mathbb{N}$ such that f is k-determined

Output: the corank c of f, the inertia index λ of f, and a polynomial $g \in \mathfrak{m}^3 \cap \mathbb{Q}[x_1, \ldots, x_c]$ such that

$$f \overset{r}{\sim} g - \sum_{i=c+1}^{c+\lambda} x_i^2 + \sum_{i=c+\lambda+1}^{n} x_i^2$$

1: compute a transformation matrix $T \in \mathbb{R}^{n \times n}$ such that

$$T^t \cdot \tfrac{1}{2} H(f)(\mathbf{0}) \cdot T = \operatorname{diag}(0, \ldots, 0, -1, \ldots, -1, 1, \ldots, 1) =: N$$

2: $c :=$ number of zeroes on the diagonal of N

3: $\lambda :=$ number of entries equal to -1 on the diagonal of N

4: $f^{(3)}(x_1, \ldots, x_n) := f((x_1, \ldots, x_n) \cdot T^t)$

5: **for** $(l = 3, \ldots, k)$ **do**

6: write $f^{(l)}$ as

$$f^{(l)} = \sum_{j=3}^{l} g_j - \sum_{i=c+1}^{c+\lambda} x_i^2 + \sum_{i=c+\lambda+1}^{n} x_i^2 + \sum_{i=c+1}^{n} x_i \cdot h_i^{(l)}$$

with $g_j \in \mathfrak{m}^j \cap \mathbb{Q}[x_1, \ldots, x_c]$ and $h_i^{(l)} \in \mathfrak{m}^{l-1}$

7: $f^{(l+1)} := \phi^{(l)}(f^{(l)})$ where $\phi^{(l)}$ is defined by

$$\phi^{(l)}(x_i) := \begin{cases} x_i, & i = 1, \ldots, c, \\ x_i + \frac{1}{2} h_i^{(l)}, & i = c+1, \ldots, c+\lambda, \\ x_i - \frac{1}{2} h_i^{(l)}, & i = c+\lambda+1, \ldots, n. \end{cases}$$

8: $g := \sum_{j=3}^{k} g_j$

9: **return** c, λ, g

given by its stable equivalence class together with its inertia index which can be computed using Algorithm 5.2.

Using the SINGULAR library `classify.lib` [47] for the complex classification and the one-to-one correspondence between the real main singularity types and the complex types, the algorithmic classification of a real germ boils down to determining to which of the corresponding subtypes the germ is equivalent. For the singularity types E_7 and E_8, there is nothing left to do because each of these types has only one real subtype. The rest of the cases is considered one by one in the following subsections.

Throughout the rest of this chapter, we write f for the given input polynomial, g for its residual part which can be obtained by applying the Splitting Lemma, and c for the corank of f. We also assume that f, and thus g, is a polynomial over $\mathbb{Q}$. With these notations, g is a polynomial in c variables.

Table 5.1: Real normal forms of singularities of modality 0

	Complex normal form	Normal forms of real subtypes	Equivalences	Values of k
A_k	x^{k+1}	$+x^{k+1}$ (A_k^+)	$A_k^+ \overset{r}{\sim} A_k^-$ for even k	$k \geq 1$
		$-x^{k+1}$ (A_k^-)		
D_k	$x^2y + y^{k-1}$	$x^2y + y^{k-1}$ (D_k^+)	-	$k \geq 4$
		$x^2y - y^{k-1}$ (D_k^-)		
E_6	$x^3 + y^4$	$x^3 + y^4$ (E_6^+)	-	-
		$x^3 - y^4$ (E_6^-)		
E_7	$x^3 + xy^3$	$x^3 + xy^3$	-	-
E_8	$x^3 + y^5$	$x^3 + y^5$	-	-

5.4.1 A_1

If $c = 0$, then f is of complex type A_1. The residual part in this case is $g = 0$, even though Table 5.1 assigns the normal form x^2 to this type for formal reasons. As a consequence, all the real singularities of main type A_1 are stably equivalent and their right equivalence class is completely determined by their inertia index λ.

5.4.2 $A_k, k > 1$

If $c = 1$, then the singularity is of complex type A_k for some $k > 1$. Over the real numbers, this type splits up into the subtypes A_k^+ and A_k^- if k is odd. Furthermore g is a univariate polynomial in this case, say $g \in \mathbb{Q}[x]$. The value of k is given by the order of g minus 1 because $\pm x^{k+1}$ and g are right equivalent and thus have the same order.

Note that if k is even, then $A_k^+ \overset{r}{\sim} A_k^-$ and we have only one real subtype which we denote by A_k. Let k be odd. Then the sign of the singularity type is determined by the sign of the coefficient of x^{k+1}. This follows since Proposition 5.2.8 implies $\mathrm{jet}(g, k+1) = \pm(\phi_0(x))^{k+1} = \pm(\alpha x)^{k+1}$, where $\phi(\pm x^{k+1}) = g$, $\alpha \in \mathbb{R}$, and the sign depends on the singularity type. Since $k+1$ is even and $\alpha \in \mathbb{R}$, ϕ does not change the sign of the coefficient of x^{k+1}. We use Algorithm 5.3, after applying the Splitting Lemma in case $c = 0$ or $c = 1$.

For the rest of the chapter, we turn our attention to singularities of corank 2. In these cases $0 \neq g \in \mathfrak{m}^3$ is a polynomial in two variables, say $g \in \mathbb{Q}[x, y]$. Using the SINGULAR library `classify.lib`, we determine the complex singularity type and thus the real main singularity type of g, or

Algorithm 5.3 Algorithm for the case A_k

Input: $f \in \mathbb{Q}[x_1, \ldots, x_n]$ of complex singularity type A_k, the output polynomial g after applying Algorithm 5.2, and the corank c of f
Output: the real singularity type of f, i.e. A_k, A_k^+ or A_k^-, $k \in \mathbb{N}$

```
1: if c = 0 then
2:     type := A_1
3: if c = 1 then
4:     k := ord(g) − 1
5:     if k is even then
6:         type := A_k
7:     else
8:         s := coefficient of x^{k+1} in g
9:         if s > 0 then
10:            type := A_k^+
11:        else
12:            type := A_k^−
13: return type
```

equivalently f. The purpose of the remaining algorithms in this chapter is to determine the correct real subtype of g, or equivalently f. We now consider each complex type, or equivalently every real main type, separately.

5.4.3 D_4

The normal form of the complex singularity type D_4 is x^2y+y^3, which splits up into x^2y+y^3 (D_4^+) and x^2y-y^3 (D_4^-) in the real case. The two cases can be distinguished by factorization; the details are carried out in Algorithm 5.4. Since the determinacy of D_4 is 3, it suffices to look at the 3-jet. The number of factors of the 3-jet over $\mathbb{R}$ is an invariant of the real subtype which is 1 in the case D_4^+ and 3 for D_4^-.

However, using the SINGULAR command `factorize` in order to determine the number of factors is problematic because the factorization over $\mathbb{R}$ differs from those over $\mathbb{Q}$ and $\mathbb{C}$ in some cases. As an alternative, we dehomogenize the 3-jet and count the number of real roots of the resulting univariate polynomial which is exactly the same as the number of factors of the 3-jet over $\mathbb{R}$.

If we want to dehomogenize the 3-jet via $x \mapsto x$, $y \mapsto 1$ without reducing its degree, we first have to make sure that the coefficient of x^3 is non-zero. It is easy to check that this is achieved by lines 2 to 13 of Algorithm 5.4. For the implementation in SINGULAR, we used the library `rootsur.lib` [56] to count the number of real roots of a univariate polynomial.

Remark 5.4.1. Geometrically, the dehomogenization in Algorithm 5.4 corre-

Algorithm 5.4 Algorithm for the case D_4

Input: $g \in \mathfrak{m}^3 \subset \mathbb{Q}[x,y]$ of complex singularity type D_4
Output: the real singularity type of g, i.e. D_4^+ or D_4^-

```
h := jet(g, 3)
s_1 := coefficient of x^3 in h
s_2 := coefficient of y^3 in h
if (s_1 = 0) then
    if (s_2 ≠ 0) then
        swap the variables x and y in h
    else
        t_1 := coefficient of x^2 y in h
        t_2 := coefficient of x y^2 in h
        if (t_1 + t_2 ≠ 0) then
            apply x ↦ x, y ↦ x + y to h
        else
            apply x ↦ x, y ↦ 2x + y to h
apply x ↦ x, y ↦ 1 to h
n := number of real roots of h
if (n < 3) then
    return D_4^+
else
    return D_4^-
```

sponds to blowing the 3-jet up at the origin plus choosing a chart. Since the 3-jet is homogeneous, blowing-up always yields three lines in the complex case. In the real case, however, we get either one or three lines depending on their position w.r.t. the real subspace in the complex picture. All the lines lie in the chosen chart because the coefficient of x^3 is non-zero.

5.4.4 $D_k, k > 4$

For the cases D_k with $k > 4$, the complex normal form is $x^2y + y^{k-1}$. It splits up into $x^2y + y^{k-1}$ (D_k^+) and $x^2y - y^{k-1}$ (D_k^-) for each k over the reals. We use the following two results from [36, p. 35] to distinguish between the two cases:

Lemma 5.4.2. *A singularity of type D_k^+ or D_k^- is $(k-1)$-determined.*

Lemma 5.4.3. *Let $j \geq 4$. Then there exists a polynomial $R \in \mathfrak{m}^{j+1} \subset \mathbb{R}[[x,y]]$ such that*

$$x^2y + a_0x^j + a_1x^{j-1}y + \ldots + a_jy^j \overset{r}{\sim} x^2y + a_jy^j + R, \quad a_0, \ldots, a_j \in \mathbb{R},$$

using the $\mathbb{R}$*-algebra automorphism*

$$x \mapsto x + p_1, \text{ where } p_1 = -\frac{1}{2}(a_1 x^{j-2} + \ldots + a_{j-1} y^{j-2}),$$
$$y \mapsto y + p_2, \text{ where } p_2 = -a_0 x^{j-2}.$$

By Lemma 5.4.2, the determinacy of a singularity of main type D_k is $k-1$. Therefore we only need to consider the $(k-1)$-jet of g in this case. By Proposition 5.2.8, the 3-jet of g factorizes as $\mathrm{jet}(g,3) = g_1^2 g_2$ over $\mathbb{R}$, where g_1 and g_2 are homogeneous polynomials of degree 1. Note that Lemma 5.2.9 ensures that this factorization can be carried out even over $\mathbb{Q}$. We can thus transform g into a polynomial of the form

$$x^2 y + \text{terms of degree higher than 3}$$

by applying the automorphism defined by $g_1 \mapsto x$, $g_2 \mapsto y$ to g.

We now systematically consider the terms of each degree $3 < j < k$. By applying the transformations in Lemma 5.4.3, for each j, the only term of total degree j which possibly remains is $a_j y^j$. This term vanishes for $j < k-1$ and it does not vanish for $j = k-1$, otherwise g is not of complex type D_k. Thus, after applying these transformations, we can write g as $g = x^2 y + \alpha y^{k-1}$ with $\alpha \neq 0$. Clearly if $\alpha > 0$ then $x^2 y + \alpha y^{k-1} \overset{r}{\sim} x^2 y + y^{k-1}$ and if $\alpha < 0$ then $x^2 y + \alpha y^{k-1} \overset{r}{\sim} x^2 y - y^{k-1}$.

Algorithm 5.5 Algorithm for the case D_k, $k > 4$

Input: $g \in \mathfrak{m}^3 \subset \mathbb{Q}[x,y]$ of complex singularity type D_k, $k \in \mathbb{N}$, $k > 4$
Output: the real singularity type of g, i.e. D_k^+ or D_k^-

```
k := μ(g)
h := jet(g, k − 1)
factorize jet(h, 3) as h_1^2 h_2, where h_1 and h_2 are linear
apply h_1 ↦ x, h_2 ↦ y to h
for (j = 4, ..., k − 1) do
    if (jet(h, j) − x^2 y ≠ 0) then
        write jet(h, j) − x^2 y as a_0 x^j + a_1 x^{j−1} y + ··· + a_j y^j,   a_0, ... a_j ∈ Q
        apply x ↦ x − (1/2)(a_1 x^{j−2} + ··· + a_{j−1} y^{j−2}), y ↦ y − a_0 x^{j−2} to h
        h := jet(h, k − 1)
write h as h = x^2 y + α y^{k−1}, 0 ≠ α ∈ Q
if (α > 0) then
    return D_k^+
else
    return D_k^−
```

5.4.5 E_6

In this case, whose complex normal form is $x^3 + y^4$, we have that either $g \overset{r}{\sim} x^3 + y^4$ (E_6^+) or $g \overset{r}{\sim} x^3 - y^4$ (E_6^-). Therefore there exists an $\mathbb{R}$-algebra automorphism ϕ of $\mathbb{R}[[x, y]]$ such that $\phi(g) = (\phi(x))^3 + (\phi(y))^4$ or such that $\phi(g) = (\phi(x))^3 - (\phi(y))^4$. Since the coefficients of x^3 and y^3 in g cannot both be zero, we can ensure that the coefficient of x^3 is non-zero by swapping the variables if necessary. Now, by Proposition 5.2.8 and Lemma 5.2.9, the 3-jet of g factorizes as $c(g_1)^3$ with $c \in \mathbb{Q}$ and $g_1 = b_0x + b_1y \in \mathbb{Q}[x, y]$, $b_0 \neq 0$. By applying $x \mapsto \frac{x - b_1y}{b_0}$, $y \mapsto y$ to g, we can thus assume without loss of generality that ϕ_0 is of the form $\phi_0(x) = c'x$, $\phi_0(y) = d_0x + d_1y$ with $c', d_0, d_1 \in \mathbb{R}$. Since ϕ is an automorphism, we have that $d_1 \neq 0$. Hence

$$(\phi(y))^4 = d_1^4y^4 + (\text{terms of degree 4 and higher, not of the form } \alpha y^4,\ \alpha \in \mathbb{R})\,.$$

If we can show that $(\phi(x))^3$ does not contain a term of the form αy^4, $\alpha \in \mathbb{R}$, then we can determine whether g is of type E_6^- or E_6^+ by considering the sign of the coefficient of the monomial y^4. A simple calculation yields

$$\begin{aligned}\operatorname{jet}((\phi(x))^3, 4) - \operatorname{jet}((\phi(x))^3, 3) &= 3(\phi_0(x)^2)(\phi_1(x) - \phi_0(x)) \\ &= 3(c'x)^2(\phi_1(x) - \phi_0(x))\,,\end{aligned}$$

which means that $(\phi(x))^3$ does not have any term of the form αy^4, $\alpha \in \mathbb{R}$.

Algorithm 5.6 Algorithm for the case E_6

Input: $g \in \mathfrak{m}^3 \subset \mathbb{Q}[x, y]$ of complex singularity type E_6
Output: the real singularity type of g, i.e. E_6^+ or E_6^-

1: $h := \operatorname{jet}(g, 3)$
2: $s :=$ coefficient of x^3 in h
3: **if** $(s = 0)$ **then**
4: swap the variables x and y
5: factorize h into linear factors over $\mathbb{Q}[x, y]$, with a factor $g_1 = b_0x + b_1y$
6: apply $x \mapsto \frac{x - b_1y}{b_0}$, $y \mapsto y$ to g
7: $d :=$ coefficient of y^4 in g
8: **if** $(d > 0)$ **then**
9: **return** E_6^+
10: **else**
11: **return** E_6^-

Chapter 6

The Structure of the Equivalence Classes of the Unimodal Real Singularities up to Corank 2

In the classification of real singularities by Arnold et al. [4], normal forms, as representatives of equivalence classes under right equivalence, are not always uniquely determined. We describe the complete structure of the equivalence classes of the unimodal real singularities of corank 2. In other words, we explicitly answer the question which normal forms of different type are equivalent, and how a normal form can be transformed within the same equivalence class by changing the value of the parameter. This provides new theoretical insights into these singularities and has important consequences for their algorithmic classification.

6.1 Introduction

This chapter is the first important step towards the algorithmic classification of the unimodal real singularities up to corank 2. It can thus be seen as a continuation of Chapter 5 which contains the algorithmic classification of the simple real singularities. All the algorithms presented there have been implemented in the computer algebra system SINGULAR [39] as a library called `realclassify.lib` [48].

Our work is based on the classifications of complex and real singularities of small modality up to stable equivalence by Arnold et al. [4]. Two power series $f, g \in \mathbb{K}[[x_1, \ldots, x_n]]$ with a critical point at the origin and critical value 0 are complex (if $\mathbb{K} = \mathbb{C}$) or real (if $\mathbb{K} = \mathbb{R}$) equivalent, denoted by $f \overset{\mathbb{K}}{\sim} g$, if there exists a $\mathbb{K}$-algebra automorphism ϕ of $\mathbb{K}[[x_1, \ldots, x_n]]$ such that

$\phi(f) = g$. They are stable (complex or real) equivalent if they become (complex or real) equivalent after the direct addition of non-degenerate quadratic terms.

In this chapter, we focus on the unimodal singularities of corank 2. Their complex and real normal forms can be found in Table 6.1. Just as for the simple singularities (cf. Chapter 6), it turns out that the complex singularity types split up into one or several real subtypes and that the normal forms of the real subtypes belonging to the same complex type differ from each other only in the signs of some terms. We therefore sometimes refer to the complex singularity types as main types. The hyperbolic type $\widetilde{Y}_r$ is an exception because it is complex equivalent to $Y_{r,r}$ and only occurs as a type on its own in the real classification.

The normal forms in Table 6.1 cover the equivalence classes of the unimodal singularities of corank 2, but some of them are equivalent to others. Such equivalences occur both between different real subtypes and between normal forms with different values of the parameter a. However, there are no equivalences between different main types. To give an example, $x^4 - 4x^2y^2 + y^4$, the normal form of X_9^{++} with $a = -4$, is equivalent to $-x^4 + 10x^2y^2 - y^4$, the normal form of X_9^{--} with $a = 10$, via the coordinate transformation $x \mapsto c(x + y)$, $y \mapsto c(x - y)$ with $c = \left(\sqrt[4]{2}\right)^{-1}$. Examples like this one have consequences for the algorithmic classification of real singularities. The question if the singularity in the example is of real type X_9^{++} or of real type X_9^{--} is not well-posed and the value of the parameter is not uniquely determined. Note that this problem does not occur for the simple singularities: By definition, their normal forms do not admit parameters, and there are no equivalences between different real subtypes except for the main types A_k where k is even, cf. Chapter 6.

The goal of this chapter is to determine the complete structure of the equivalence classes for the unimodal real singularities of corank 2. Based on these results, we plan to present algorithms to determine the equivalence class of a given unimodal real singularity of corank 2 in a separate article. If T_1 and T_2 are subtypes of the same singularity main type T and if $g_1(a)$ and $g_2(a)$ are the normal forms of T_1 and T_2, respectively, where a denotes the value of the parameter, then we are interested in the set of all pairs (u, v) such that $g_1(u)$ is equivalent to $g_2(v)$. This question can be asked in three different ways: If we consider complex values of u and v and complex coordinate transformations, we denote the corresponding set by $P_1(T_1, T_2)$, for real values of u and v, but still complex transformations by $P_2(T_1, T_2)$, and finally by $P_3(T_1, T_2)$ if we consider only real values of u and v and real transformations, cf. Definition 6.2.6.

The formal definitions of these sets and other basic notations are introduced in Section 6.2, along with different ways how $P_1(T_1, T_2)$, $P_2(T_1, T_2)$, and $P_3(T_1, T_2)$ can be conveniently written down in concrete cases. The following sections are devoted to the computation of these sets for any two

Table 6.1: Normal forms of singularities of modality 1 and corank 2

		Complex normal form	Normal forms of real subtypes	Restrictions
Parabolic	X_9	$x^4+ax^2y^2+y^4$	$+x^4+ax^2y^2+y^4$ (X_9^{++})	$a^2\neq+4$
			$-x^4+ax^2y^2-y^4$ (X_9^{--})	
			$+x^4+ax^2y^2-y^4$ (X_9^{+-})	$a^2\neq-4$
			$-x^4+ax^2y^2+y^4$ (X_9^{-+})	
	J_{10}	$x^3+ax^2y^2+xy^4$	$x^3+ax^2y^2+xy^4$ (J_{10}^{+})	$a^2\neq+4$
			$x^3+ax^2y^2-xy^4$ (J_{10}^{-})	$a^2\neq-4$
Hyperbolic	J_{10+k}	$x^3+x^2y^2+ay^{6+k}$	$x^3+x^2y^2+ay^{6+k}$ (J_{10+k}^{+})	$a\neq0$,
			$x^3-x^2y^2+ay^{6+k}$ (J_{10+k}^{-})	$k>0$
	X_{9+k}	$x^4+x^2y^2+ay^{4+k}$	$+x^4+x^2y^2+ay^{4+k}$ (X_{9+k}^{++})	$a\neq0$,
			$-x^4-x^2y^2+ay^{4+k}$ (X_{9+k}^{--})	$k>0$
			$+x^4-x^2y^2+ay^{4+k}$ (X_{9+k}^{+-})	
			$-x^4+x^2y^2+ay^{4+k}$ (X_{9+k}^{-+})	
	$Y_{r,s}$	$x^2y^2+x^r+ay^s$	$+x^2y^2+x^r+ay^s$ $(Y_{r,s}^{++})$	$a\neq0$,
			$-x^2y^2-x^r+ay^s$ $(Y_{r,s}^{--})$	$r,s>4$
			$+x^2y^2-x^r+ay^s$ $(Y_{r,s}^{+-})$	
			$-x^2y^2+x^r+ay^s$ $(Y_{r,s}^{-+})$	
	$\widetilde{Y}_r$	$(x^2+y^2)^2+ax^r$	$+(x^2+y^2)^2+ax^r$ $(\widetilde{Y}_r^{+})$	$a\neq0$,
			$-(x^2+y^2)^2+ax^r$ $(\widetilde{Y}_r^{-})$	$r>4$
Exceptional	E_{12}	$x^3+y^7+axy^5$	$x^3+y^7+axy^5$	-
	E_{13}	$x^3+xy^5+ay^8$	$x^3+xy^5+ay^8$	-
	E_{14}	$x^3+y^8+axy^6$	$x^3+y^8+axy^6$ (E_{14}^{+})	-
			$x^3-y^8+axy^6$ (E_{14}^{-})	
	Z_{11}	$x^3y+y^5+axy^4$	$x^3y+y^5+axy^4$	-
	Z_{12}	$x^3y+xy^4+ax^2y^3$	$x^3y+xy^4+ax^2y^3$	-
	Z_{13}	$x^3y+y^6+axy^5$	$x^3y+y^6+axy^5$ (Z_{13}^{+})	-
			$x^3y-y^6+axy^5$ (Z_{13}^{-})	
	W_{12}	$x^4+y^5+ax^2y^3$	$+x^4+y^5+ax^2y^3$ (W_{12}^{+})	-
			$-x^4+y^5+ax^2y^3$ (W_{12}^{-})	
	W_{13}	$x^4+xy^4+ay^6$	$+x^4+xy^4+ay^6$ (W_{13}^{+})	-
			$-x^4+xy^4+ay^6$ (W_{13}^{-})	

real subtypes T_1 and T_2 listed in Table 6.1. We first recall the definitions of (piecewise) weighted jets and filtrations in Section 6.3. They play a major role in the proof of Theorem 6.4.3, the main result of Section 6.4. This theorem allows us to restrict ourselves to a small subset of coordinate transformations, which we call a sufficient set, if we want to determine $P_1(T_1, T_2)$. It is thus the theoretic basis for Section 6.5 where we explain how $P_1(T_1, T_2)$, $P_2(T_1, T_2)$, and $P_3(T_1, T_2)$ can be computed using SINGULAR. We also give an example with explicit SINGULAR commands. These methods do not apply for the singularity type $\widetilde{Y}_r$ which is treated separately in Section 6.5.4. Section 6.6 contains the results of these computations in a concise form. Finally, we point out some remarkable aspects of the results in this chapter as well as their consequences for the algorithmic classificiation of the unimodal real singularities of corank 2 in Section 6.7. The maybe most surprising outcome is that the real subtype J_{10}^- is actually redundant whereas J_{10}^+ is not.

6.2 The Sets of Parameter Transformations P_1, P_2, and P_3

Let us start with some basic definitions. Throughout the rest of this chapter, let $\mathbb{K}$ be, in each case, either $\mathbb{R}$ or $\mathbb{C}$.

Definition 6.2.1. Two power series $f, g \in \mathbb{K}[[x_1, \ldots, x_n]]$ are called $\mathbb{K}$-equivalent, denoted by $f \overset{\mathbb{K}}{\sim} g$, if there exists a $\mathbb{K}$-algebra automorphism ϕ of $\mathbb{K}[[x_1, \ldots, x_n]]$ such that $\phi(f) = g$.

Note that $\overset{\mathbb{K}}{\sim}$ is an equivalence relation on $\mathbb{K}[[x_1, \ldots, x_n]]$. Arnold et al. [4] give the following formal definition for normal forms w.r.t. this relation:

Definition 6.2.2. Let $K \subset \mathbb{K}[[x_1, \ldots, x_n]]$ be a union of equivalence classes w.r.t. the relation $\overset{\mathbb{K}}{\sim}$. A *normal form* for K is given by a smooth map

$$\Phi : B \longrightarrow \mathbb{K}[x_1, \ldots, x_n] \subset \mathbb{K}[[x_1, \ldots, x_n]]$$

of a finite-dimensional $\mathbb{K}$-linear *space of parameters* B into the space of polynomials for which the following three conditions hold:

1. $\Phi(B)$ intersects all the equivalence classes of K;
2. the inverse image in B of each equivalence class is finite;
3. the inverse image of the whole complement to K is contained in some proper hypersurface in B.

Remark 6.2.3. Note that the term *normal form* is subtly ambiguous. According to the above definition, a normal form is a smooth map where the

inverse image of each equivalence class may contain more than one element, whereas the common meaning of this term rather refers to the polynomials which are the images under this map. We could be more precise and avoid this ambiguity by introducing a new term for either of the two meanings. However, we stay with the common usage of the term *normal form* in order to prevent confusion.

Definition 6.2.4. Let $S \subset \mathrm{Aut}_{\mathbb{K}}(\mathbb{K}[[x_1, \ldots, x_n]])$ be a set of $\mathbb{K}$-algebra automorphisms of $\mathbb{K}[[x_1, \ldots, x_n]]$ and let $f, g \in \mathbb{K}[[x_1, \ldots, x_n]]$ be two power series.

1. We denote the set of all automorphisms in S which take f to g by $\mathrm{T}^S_{\mathbb{K}}(f, g)$, i.e.
$$\mathrm{T}^S_{\mathbb{K}}(f, g) := \{\phi \in S \mid \phi(f) = g\}\,.$$

2. If $S = \mathrm{Aut}_{\mathbb{K}}(\mathbb{K}[[x_1, \ldots, x_n]])$, we simply write $\mathrm{T}_{\mathbb{K}}(f, g)$ for $\mathrm{T}^S_{\mathbb{K}}(f, g)$, i.e.
$$\mathrm{T}_{\mathbb{K}}(f, g) := \{\phi \in \mathrm{Aut}_{\mathbb{K}}(\mathbb{K}[[x_1, \ldots, x_n]]) \mid \phi(f) = g\}\,.$$

The above definition is the key ingredient for the definition of P_1, P_2, and P_3. We also need the following notation.

Remark 6.2.5. As usual, we denote the field of quotients $\mathrm{Quot}(\mathbb{K}[a])$ by $\mathbb{K}(a)$. Let $f \in \mathbb{K}(a)[[x_1, \ldots, x_n]]$ be a power series over this quotient field. Then f can be written as $f = \sum_{\nu \in \mathbb{N}^n} c_\nu \boldsymbol{x}^\nu$ with coeffients $c_\nu = \frac{p_\nu}{q_\nu} \in \mathbb{K}(a)$ where $p_\nu, q_\nu \in \mathbb{K}[a]$ are polynomials of minimal degree with this property and $q_\nu \neq 0$ for all $\nu \in \mathbb{N}^n$.

If we consider the polynomials p_ν, q_ν as polynomial functions $p_\nu, q_\nu : \mathbb{K} \to \mathbb{K}$, then we may also consider the coefficients c_ν as functions $c_\nu : \mathbb{K} \setminus V(q_\nu) \to \mathbb{K}$ where $V(q_\nu)$ is the set of points where q_ν vanishes. Via this correspondence, we finally get power series $f(u) := \sum_{\nu \in \mathbb{N}^n} c_\nu(u) \boldsymbol{x}^\nu \in \mathbb{K}[[x_1, \ldots, x_n]]$ for each value $u \in \mathbb{K} \setminus \bigcup_{\nu \in \mathbb{N}^n} V(q_\nu)$.

We use the notation $f(u)$ throughout this chapter. Likewise, we add the value of the parameter which occurs in the normal form as given in Table 6.1 in parentheses to the name of the singularity (sub-)type if we want to refer specifically to the corresponding equivalence class. For instance, we denote by $E_{14}(3)$ the (complex or real) right-equivalence class of $x^3 + y^8 + 3xy^6$.

For any specific singularity type T, we denote by $\mathrm{NF}(T)$ its normal form as shown in Table 6.1, i.e. we write $\mathrm{NF}(E_{14}(a)) = \mathrm{NF}\big(E^+_{14}(a)\big)$ for the polynomial $x^3 + y^8 + axy^6$ and $\mathrm{NF}\big(E^-_{14}(5)\big)$ for $x^3 - y^8 + 5xy^6$.

We can now state the main definition of this section.

Definition 6.2.6. We define the following sets of parameter transformations:

1. Given power series $f, g \in \mathbb{C}(a)[[x_1, \ldots, x_n]]$, we define the first set of parameter transformations of f and g as

$$P_1(f,g) := \big\{(u,v) \in \mathbb{C}^2 \mid f(u) \text{ and } g(v) \text{ are well-defined and } \mathrm{T}_{\mathbb{C}}(f(u), g(v)) \neq \varnothing\big\}.$$

2. Given power series $f, g \in \mathbb{R}(a)[[x_1, \ldots, x_n]]$, we define the second set of parameter transformations of f and g as

$$P_2(f,g) := \big\{(u,v) \in \mathbb{R}^2 \mid f(u) \text{ and } g(v) \text{ are well-defined and } \mathrm{T}_{\mathbb{C}}(f(u), g(v)) \neq \varnothing\big\}.$$

3. Given power series $f, g \in \mathbb{R}(a)[[x_1, \ldots, x_n]]$, we define the third set of parameter transformations of f and g as

$$P_3(f,g) := \big\{(u,v) \in \mathbb{R}^2 \mid f(u) \text{ and } g(v) \text{ are well-defined and } \mathrm{T}_{\mathbb{R}}(f(u), g(v)) \neq \varnothing\big\}.$$

Remark 6.2.7.

1. Note that we have $P_3(f,g) \subseteq P_2(f,g) \subseteq P_1(f,g)$ for any two power series $f, g \in \mathbb{R}(a)[[x_1, \ldots, x_n]]$.

2. For any two unimodal singularity (sub-)types T_1, T_2 and $i \in \{1,2,3\}$, we simply write $P_i(T_1, T_2)$ instead of $P_i(\mathrm{NF}(T_1(a)), \mathrm{NF}(T_2(a)))$, e.g. we write $P_1\big(E_{14}^+, E_{14}^+\big)$ for $P_1\big(\mathrm{NF}\big(E_{14}^+(a)\big), \mathrm{NF}\big(E_{14}^+(a)\big)\big)$.

For the parabolic singularity types X_9 and J_{10}, the sets P_1, P_2, and P_3 can be described in terms of the following definition.

Definition 6.2.8. For $\Omega \subset \mathbb{C}$, let $(f_i : \Omega \to \mathbb{C})_{i \in I}$ be a family of complex-valued functions on Ω. We define the joint graph of $(f_i)_{i \in I}$ over Ω as

$$\Gamma_\Omega((f_i)_{i \in I}) := \{(a, f_i(a)) \in \Omega \times \mathbb{C} \mid a \in \Omega,\ i \in I\}.$$

It turns out that for the hyperbolic and exceptional unimodal singularities, P_1, P_2 and P_3 are just unions of sets of the form $(a, ra)_{a \in \mathbb{K}}$ for some $r \in \mathbb{K}$. For those cases we use the following notations.

Definition 6.2.9. For any polynomial $p(X) \in \mathbb{C}[X]$, we define the sets $C_0(p(X))$ and $R_0(p(X))$ as

$$C_0(p(X)) := \{(a, ra) \in \mathbb{C}^2 \mid a, r \in \mathbb{C},\ p(r) = 0\},$$
$$R_0(p(X)) := \{(a, ra) \in \mathbb{R}^2 \mid a, r \in \mathbb{R},\ p(r) = 0\}.$$

Additionally, we define $C(p(X))$ and $R(p(X))$ as

$$C(p(X)) := C_0(p(X)) \setminus \{(0,0)\},$$
$$R(p(X)) := R_0(p(X)) \setminus \{(0,0)\}.$$

Remark 6.2.10. We occasionally use the notation $R\big(X^l - s\big)$ with $l \in \mathbb{N}\setminus\{0\}$ and $s \in \{-1, +1\}$, e.g. in Tables 6.5 and 6.6. Of course, this could be written in a more explicit way for many values of l and s; for instance, we could write $\varnothing$ instead of $R\big(X^4 + 1\big)$. But distinguishing between different cases would spoil the symmetries of those tables and we therefore stick to the shorthand notation.

6.3 Weighted Jets and Filtrations of Power Series and Transformations

We briefly introduce the concepts of (piecewise) weighted jets and filtrations. For background regarding the definitions in this section, we refer to [3]. We assume that the reader is familiar with the notions of weighted degrees, quasihomogeneous polynomials, and Newton polygons.

Remark 6.3.1. Let w be a weight on the variables $(x_1, \ldots, x_n)$. Throughout this chapter, we always assume that the weighted degree of x_i, denoted by $w\text{-deg}(x_i)$, is a natural number for each $i = 1, \ldots, n$.

Definition 6.3.2. Let $w_0 := (w_1, \ldots, w_s) \in (\mathbb{N}^n)^s$ be a finite family of weights on the variables $(x_1, \ldots, x_n)$. For any term $t \in \mathbb{K}[x_1, \ldots, x_n]$, we define the piecewise weight of t w.r.t. w_0 as

$$w_0\text{-deg}(t) := \min_{i=1,\ldots,s} w_i\text{-deg}(t)\,.$$

A polynomial $f \in \mathbb{K}[x_1, \ldots, x_n]$ is called piecewise quasihomogeneous of degree d w.r.t. w_0 if $w_0\text{-deg}(t) = d$ for any term t of f.

Definition 6.3.3. Let w be a (possibly piecewise) weight on the variables $(x_1, \ldots, x_n)$.

1. Let $f = \sum_{i=0}^{\infty} f_i$ be the decomposition of $f \in \mathbb{K}[[x_1, \ldots, x_n]]$ into weighted homogeneous parts f_i of w-degree i. We denote the weighted j-jet of f w.r.t. w by
$$w\text{-jet}(f, j) := \sum_{i=0}^{j} f_i\,.$$
2. A power series in $\mathbb{K}[[x_1, \ldots, x_n]]$ has filtration $d \in \mathbb{N}$ if all its terms are of weighted degree d or higher. The power series of filtration d form a vector space $E_d^w \subset \mathbb{K}[[x_1, \ldots, x_n]]$.

Remark 6.3.4. Note that $d < d'$ implies $E_{d'}^w \subseteq E_d^w$. Since the filtration of the product $E_{d'}^w \cdot E_d^w$ is $d' + d$, it follows that E_d^w is an ideal in the ring of power series. We denote the ideal consisting of power series of filtration strictly greater than d by $E_{>d}^w$. If the weight of each variable is 1, we simply write E_d and $E_{>d}$, respectively.

There are also similar concepts for coordinate transformations:

Definition 6.3.5. Let ϕ be a $\mathbb{K}$-algebra automorphism of $\mathbb{K}[[x_1,\dots,x_n]]$ and let w be a (piecewise) weight on the variables.

1. For $j>0$ we define the weighted j-jet of ϕ w.r.t. w, denoted by ϕ_j^w, to be the map given by

$$\phi_j^w(x_i) := w\text{-jet}(\phi(x_i), w\text{-deg}(x_i)+j) \quad \forall i=1,\dots,n\,.$$

   If the weight of each variable is 1, i.e. $w=(1,\dots,1)$, we simply write ϕ_j for ϕ_j^w.

2. ϕ has filtration d if, for all $\lambda\in\mathbb{N}$,

$$(\phi-\mathrm{id})E_\lambda^w \subset E_{\lambda+d}^w\,.$$

Remark 6.3.6. Let ϕ be a $\mathbb{K}$-algebra automorphism of $\mathbb{K}[[x_1,\dots,x_n]]$.

1. Note that $\phi_0(x_i)=\mathrm{jet}(\phi(x_i),1)$ for all $i=1,\dots,n$. Furthermore note that ϕ_0^w may have filtration less than or equal to 0 for any weight w.

2. Let $w_0=(w_1,\dots,w_s)\in(\mathbb{N}^n)^s$ be a piecewise weight on $(x_1,\dots,x_n)$, let $f_0\in\mathbb{K}[x_1,\dots,x_n]$ be piecewise quasihomogeneous of degree d_0 w.r.t. w_0 and $f_1\in\mathbb{K}[x_1,\dots,x_n]$ quasihomogeneous of degree d_1 w.r.t. w_1. For any $\delta\geq 0$, we always have $(\phi-\phi_\delta^{w_1})(f_1)\in E^{w_1}_{>d_1+\delta}$, but the analogon for w_0 does not hold in general: To give a counterexample, let us consider the case $n=s=2$, $w_0=((1,4),(4,1))$, $f_0=x_1x_2$, and let ϕ be given by $\phi(x_1):=x_1+x_2^2$, $\phi(x_2):=x_2+x_2^2$. Then f_0 is of degree $d_0=5$, but $(\phi-\phi_0^{w_0})(f_0)=x_2^4$ is of degree 4 w.r.t. w_0 and thus not an element of $E_5^{w_0}=E_{d_0+0}^{w_0}$.

6.4 Sufficient Sets of Transformations

The results in this section considerably narrow down the transformations we need to consider between specific unimodal normal forms of the same main type in order to check if they are equivalent or not. In fact these results are in many cases the main step for determining the structure of the equivalence classes of the unimodal singularities up to corank 2.

Definition 6.4.1. Let f and g be elements in $\mathbb{C}(a)[[x_1,\dots,x_n]]$ and let S be a subset of $\mathrm{Aut}_{\mathbb{C}}(\mathbb{C}[[x_1,\dots,x_n]])$. We call S a sufficient set of coordinate transformations for the pair (f,g) if

$$\forall u,v\in\mathbb{C}: \quad \big(\mathrm{T}_{\mathbb{C}}(f(u),g(v))\neq\varnothing \;\Leftrightarrow\; \mathrm{T}^S_{\mathbb{C}}(f(u),g(v))\neq\varnothing\big)\,.$$

The sufficient sets which we consider here can be described using the following notation.

Definition 6.4.2. Let M_x and M_y be sets of monomials in $\mathbb{C}[[x,y]]$ and let $\mathbb{C}M_x$ and $\mathbb{C}M_y$ be the $\mathbb{C}$-vector spaces spanned by these sets, i.e. $\mathbb{C}M_x := \bigoplus_{m\in M_x} \mathbb{C}\,m$ and analogously for $\mathbb{C}M_y$. We define the set of coordinate transformations spanned by M_x and M_y as

$$\mathrm{S}(M_x, M_y) := \{\phi \in \mathrm{Aut}_{\mathbb{C}}(\mathbb{C}[[x,y]]) \mid \phi(x) \in \mathbb{C}M_x,\ \phi(y) \in \mathbb{C}M_y\}\,.$$

Theorem 6.4.3. *Let T be one of the main singularity types listed in Table 6.2, let S be the corresponding set of automorphisms, and let T_1 and T_2 be subtypes of T. Then S is a sufficient set of coordinate transformations for $\big(\mathrm{NF}(T_1(a)), \mathrm{NF}(T_2(a)\big)$.*

Table 6.2: Sufficient sets for unimodal singularities of corank 2

	T		S
P.	X_9		$\mathrm{S}(\{x,y\},\{x,y\})$
	J_{10}		$\mathrm{S}(\{x,y^2\},\{y\})$
Hyperbolic	J_{10+k}		$\mathrm{S}(\{x\},\{y\})$
	X_{9+k}		$\mathrm{S}(\{x\},\{y\})$
	$Y_{r,s}$	$r \neq s$	$\mathrm{S}(\{x\},\{y\})$
		$r = s$	$\mathrm{S}(\{x\},\{y\}) \cup \mathrm{S}(\{y\},\{x\})$
Except.	E_{12}, E_{13}, E_{14}		$\mathrm{S}(\{x\},\{y\})$
	Z_{11}, Z_{12}, Z_{13}		$\mathrm{S}(\{x\},\{y\})$
	W_{12}, W_{13}		$\mathrm{S}(\{x\},\{y\})$

Proof of Theorem 6.4.3. We give different proofs for the parabolic, the hyperbolic, and the exceptional cases as indicated in Table 6.2.

In each case, let T_1 and T_2 be subtypes of the same main type T, and for $u \in \mathbb{C}$ let $\phi \in \mathrm{Aut}_{\mathbb{C}}(\mathbb{C}[[x,y]])$ be a coordinate transformation which takes $f := \mathrm{NF}(T_1(u))$ to $\mathrm{NF}(T_2(v))$ for some $v \in \mathbb{C}$.

Parabolic cases: The normal forms of both X_9 and J_{10} are quasihomogeneous with weights $w := (1,1)$ and $w := (2,1)$, respectively. Let us first consider the case $T = X_9$. We have

$$\phi(f) = \phi_0^w(f) + (\phi - \phi_0^w)(f) = \phi_0^w(f) + R$$

with $R \in E^w_{>4}$. This implies $\phi(f) = \phi_0^w(f)$ because $\phi(f) = \mathrm{NF}(T_2(v))$ is homogeneous of degree 4 w.r.t. the weight w. So any possible value of v

which can be reached via some $\phi \in \mathrm{Aut}_{\mathbb{C}}(\mathbb{C}[[x,y]])$ can also be obtained by $\phi_0^w \in \mathrm{S}(\{x,y\},\{x,y\})$, i.e., $\mathrm{S}(\{x,y\},\{x,y\})$ is a sufficient set of coordinate transformations for the pair $(\mathrm{NF}(T_1(a)), \mathrm{NF}(T_2(a)))$.

Let us now consider the case $T = J_{10}$. Again we have $\phi(f) = \phi_0^w(f)$, but in this case ϕ_0^w is of the form $\phi_0^w(x) = \alpha x + \beta y + \gamma y^2$, $\phi_0^w(y) = \delta y$ with $\alpha, \beta, \gamma, \delta \in \mathbb{C}$. Comparing the coefficients of $\phi(f) = \mathrm{NF}(T_2(v))$ and $\phi_0^w(f) = \beta^3 y^3 +$(other terms) yields $\beta = 0$ and therefore $\phi_0^w \in \mathrm{S}(\{x, y^2\}, \{y\})$ as expected.

Hyperbolic cases: We present a proof for the main type $T = J_{10+k}$, the proofs for X_{9+k} and $Y_{r,s}$ are similar. For $Y_{r,s}$ with $r = s$, we have to take the special shape of S into account, cf. Table 6.2.

It does not matter for the arguments below whether we assume $T_1 = J^+_{10+k}$ or $T_1 = J^-_{10+k}$, the same holds for T_2. We write $\pm$ whenever the sign can be either plus or minus in order to prove all cases at once.

The Newton polygon of $f = \mathrm{NF}(T_1(u)) = x^3 \pm x^2y^2 + uy^{6+k}$ has two faces defined by $f_1 := x^3 \pm x^2y^2$ and $f_2 := \pm x^2y^2 + uy^{6+k}$. Let w_0 be the piecewise weight given by the two weights $w_1 := (12+2k, 6+k)$ and $w_2 := (12+3k, 6)$. Then f is piecewise quasihomogeneous of degree $d := 36 + 6k$ w.r.t. w_0.

We now proceed in three steps: In the first two, we show $\phi_0^{w_1} \in \mathrm{S}(\{x\},\{y\})$ and $\phi_0^{w_2} \in \mathrm{S}(\{x\},\{y\})$. Finally we conclude that $\phi(f)$ is equal to $\phi_0^{w_0}(f)$ and that $\phi_0^{w_0}$ is an element of $\mathrm{S}(\{x\},\{y\})$ which proves the claim.

The transformation $\phi_0^{w_1}$ is generically of the form

$$\begin{aligned} \phi_0^{w_1}(x) &= \alpha x + \beta_1 y + \beta_2 y^2\,, \\ \phi_0^{w_1}(y) &= \gamma y \end{aligned}$$

with coefficients $\alpha, \beta_1, \beta_2, \gamma \in \mathbb{C}$. With these notations we have

$$\begin{aligned} \phi(f) &= \phi_0^{w_1}(f) + (\phi - \phi_0^{w_1})(f) \\ &= \beta_1^3 y^3 + (3\alpha\beta_2^2 \pm 2\alpha\beta_2\gamma^2)xy^4 + (\beta_2^3 \pm \beta_2^2\gamma^2)y^6 + (\text{other terms}) \end{aligned}$$

on the one hand and

$$\phi(f) = \mathrm{NF}(T_1(v)) = x^3 \pm x^2y^2 + vy^{6+k}$$

on the other hand. This implies (in this order) $\beta_1 = 0$, $\alpha \neq 0$, $\beta_2 = 0$ and hence $\phi_0^{w_1} \in \mathrm{S}(\{x\},\{y\})$.

The second step is a proof by contradiction. Let m be the largest integer which is not greater than $\frac{k}{2} + 2$. Then similar as above, the automorphism $\phi_0^{w_2}$ is of the form

$$\begin{aligned} \phi_0^{w_2}(x) &= \alpha x + \beta_1 y + \beta_2 y^2 + \ldots + \beta_m y^m\,, \\ \phi_0^{w_2}(y) &= \gamma y \end{aligned}$$

with $\alpha, \beta_1, \ldots, \beta_m, \gamma \in \mathbb{C}$ and $\alpha, \gamma \neq 0$. We have already shown $\beta_1 = \beta_2 = 0$. Assume $\beta_s \neq 0$ for some $s \in \{3, \ldots, m\}$ and let s be minimal with this property. Then the coefficient of xy^{s+2} in $\phi(f) = \phi_0^{w_2}(f) + (\phi - \phi_0^{w_2})(f)$ is $\pm 2\alpha\beta_s\gamma^2$ which implies $\beta_s = 0$ in contradiction to the assumption. Hence $\beta_3 = \ldots = \beta_m = 0$ and $\phi_0^{w_2} \in \mathrm{S}(\{x\}, \{y\})$.

For the last step, we consider the following equations:

$$\begin{aligned}
\phi(f) &= \phi_0^{w_1}(f) + \underbrace{(\phi - \phi_0^{w_1})(f)}_{=: R_1 \in E_{>d}^{w_1}} \\
\phi(f) &= \phi_0^{w_2}(f) + \underbrace{(\phi - \phi_0^{w_2})(f)}_{=: R_2 \in E_{>d}^{w_2}} \\
\phi(f) &= \phi_0^{w_0}(f) + \underbrace{(\phi - \phi_0^{w_0})(f)}_{=: R_0}
\end{aligned}$$

Note that it is not a priori clear that R_0 lies in $E_{>d}^{w_0}$ if we only consider these equations, cf. Remark 6.3.6(2). Nevertheless, this can be shown if we take into account the results of the two previous steps: By definition of the piecewise weight w_0, any term in $\phi_0^{w_0}(x)$ also appears in $\phi_0^{w_1}(x)$ or $\phi_0^{w_2}(x)$ (or both), analogously for $\phi_0^{w_0}(y)$. Therefore we have $\phi_0^{w_0}(x) = \alpha x$ and $\phi_0^{w_0}(y) = \gamma y$, hence $\phi_0^{w_0} = \phi_0^{w_1} = \phi_0^{w_2}$ and $\phi_0^{w_0} \in \mathrm{S}(\{x\}, \{y\})$. This implies

$$R_0 = R_1 = R_2 \in E_{>d}^{w_1} \cap E_{>d}^{w_2} = E_{>d}^{w_0}\,.$$

Since $\phi(f) = \mathrm{NF}(T_2(v))$ is piecewise quasihomogeneous of degree d w.r.t. w_0, we finally get $R_0 = 0$ and $\phi(f) = \phi_0^{w_0}(f)$. This proves the claim.

Exceptional cases: The normal forms of all the exceptional cases in Table 6.2 are semi-quasihomogeneous polynomials, i.e., in these cases $f = \mathrm{NF}(T_1(u))$ is of the form $f = f_0 + f_1$ where f_0 is quasihomogeneous of degree $d \in \mathbb{N}$ w.r.t. some weight $w = (w_x, w_y)$, f_1 has weighted degree $d + \delta > d$, and the Milnor number $\mu(f_0)$ of f_0 is finite (for the definition of the Milnor number, see Definition 5.2.3). In all the cases, f_1 consists of the term which contains the parameter and we have

$$\begin{aligned}
\phi(f) &= \phi_\delta^w(f_0) + \phi_0^w(f_1) + (\phi - \phi_\delta^w)(f_0) + (\phi - \phi_0^w)(f_1) \\
&= w\text{-jet}(\phi_\delta^w(f_0), d + \delta) + \phi_0^w(f_1) + R
\end{aligned}$$

with $R \in E_{>d+\delta}^w$. As above, $\phi(f) = \mathrm{NF}(T_2(v))$ implies $R = 0$. If we show

$$\phi_\delta^w \in \mathrm{S}(\{x\}, \{y\})\,, \tag{$*$}$$

then it follows that ϕ_δ^w is equal to ϕ_0^w and therefore

$$\begin{aligned}
\phi(f) &= w\text{-jet}(\phi_0^w(f_0), d + \delta) + \phi_0^w(f_1) \\
&= \phi_0^w(f_0) + \phi_0^w(f_1) \\
&= \phi_\delta^w(f)\,.
\end{aligned}$$

This, together with $(*)$, proves the claim.

The statement $(*)$ can be shown seperately for each of the eight cases by some easy computations. We carry out the proof for W_{13}, the other cases follow similarly. The normal forms of the subtypes of W_{13} are $\pm x^4 + xy^4 + ay^6$, so in this case we have $w = (4, 3)$, $d = 16$, and $\delta = 2$. The $\pm$-sign does not matter for the computations which follow, but we carry it along in order to prove all subcases at once. The transformation ϕ_δ^w is generically of the form

$$\begin{aligned}\phi_\delta^w(x) &= \alpha x + \beta y + \gamma y^2\,,\\ \phi_\delta^w(y) &= \varepsilon x + \zeta y\end{aligned}$$

with $\alpha, \beta, \gamma, \varepsilon, \zeta \in \mathbb{C}$ because any other term would raise the weighted degree by more than δ. With these notations, we now successively compare the coefficients of $\phi_\delta^w(f)$ and $\phi(f) = \mathrm{NF}(T_2(v)) = \pm x^4 + xy^4 + vy^6$. The coefficient of y^4 in $\phi_\delta^w(f)$ is $\pm\beta^4$, therefore we have $\beta = 0$. The remaining coefficients of xy^4, x^2y^3, and x^3y^2 are now $\alpha\zeta^4$, $4\alpha\varepsilon\zeta^3$, and $\pm 4\alpha^3\gamma + 6\alpha\varepsilon^2\zeta^2$, respectively, which shows that (in this order) $\alpha\zeta \neq 0$, $\varepsilon = 0$, and $\gamma = 0$. Hence ϕ_δ^w is in fact of the form $\phi_\delta^w(x) = \alpha x$, $\phi_\delta^w(y) = \zeta y$ which proves $(*)$ for $T_1, T_2 \in \{W_{13}^+, W_{13}^-\}$.

□

6.5 On the Computation of the Results

Based on the previous section, the results presented in Section 6.6 can be computed using SINGULAR for all those singularity types which are covered by Theorem 6.4.3. The main tools for these computations are elimination, Gröbner covers, and primary decomposition. For each pair of singularity subtypes T_1, T_2, the computation follows the same structure: One can first compute the set $P_1(T_1, T_2)$ using elimination and factorization. The set $P_2(T_1, T_2)$ can then be derived from this as the intersection of $P_1(T_1, T_2)$ with $\mathbb{R} \times \mathbb{R}$. In order to determine $P_3(T_1, T_2)$, one finally has to check for each point or branch in $P_2(T_1, T_2)$ whether or not there is a real transformation which changes the parameter in such a way. Gröbner covers and primary decomposition are convenient tools to simplify the often complicated ideals which occur in this last step.

Although our approach is almost algorithmic, we do not present it as an algorithm here because each case requires slightly different means depending on the intermediate results. Especially the computation of $P_3(T_1, T_2)$ is rather straighforward in some cases whereas it requires careful considerations in other cases.

However, writing down every detail of the computations for each case is beyond the scope of this section. Instead, we present the general framework

and give explicit SINGULAR commands for $T_1 = T_2 = X_9^{++}$ which is one of the more complicated cases (cf. Theorem 6.6.1).

The singularity type $\widetilde{Y}_r$ does not appear in Table 6.2 and thus needs special care. The structure of the equivalence classes of this type can be computed on the basis of the data for the type $Y_{r,s}$, cf. Section 6.5.4.

6.5.1 How to Compute $P_1(T_1, T_2)$

Let $S = \mathrm{S}(M_x, M_y) \subset \mathrm{Aut}_{\mathbb{C}}(\mathbb{C}[[x,y]])$ be the sufficient set of $\mathbb{C}[[x,y]]$-automorphisms for $(\mathrm{NF}(T_1(a)), \mathrm{NF}(T_2(a)))$ given in Theorem 6.4.3. Let $t_1, \ldots, t_r$ be coefficients for the monomials in M_x and M_y and let ϕ be a generic element of S with these coefficients, i.e. let ϕ be of the form $\phi(x) = t_1 \cdot x + (\text{other terms})$ (or of the form $\phi(x) = t_1 \cdot y + (\text{other terms})$ in case T_1 and T_2 are subtypes of $Y_{r,s}$ with $r = s$).

We denote the parameter occurring in $\mathrm{NF}(T_1)$ by a and the one in $\mathrm{NF}(T_2)$ by b. By comparing the coefficients in $\phi(\mathrm{NF}(T_1(a)))$ and $\mathrm{NF}(T_2(b))$, we get a set of equations in $a, b, t_1, \ldots, t_r$ which is equivalent to $\phi(\mathrm{NF}(T_1(a))) = \mathrm{NF}(T_2(b))$. Let $I \subset \mathbb{C}[a, b, t_1, \ldots, t_r]$ be the ideal generated by these equations. Then the vanishing set $V(I)$ describes completely which transformations take $\mathrm{NF}(T_1(a))$ to $\mathrm{NF}(T_2(b))$ for which values of a and b.

We can now eliminate the variables $t_1, \ldots, t_r$ from I and thus obtain an ideal $I' \subset \mathbb{C}[a,b]$ which is in all cases generated by one polynomial g. This elimination geometrically corresponds to the projection $\mathbb{A}^{2+r}_{\mathbb{C}} \supset V(I) \mapsto V(I') \subset \mathbb{A}^2_{\mathbb{C}}$. After factorizing $g \in \mathbb{C}[a,b]$ into irreducible factors $g_1, \ldots, g_s$, we compute the roots in b of each factor (over $\mathbb{C}(a)$ or suitable extensions thereof if necessary). We thus get roots of the form $b - f(a)$ where $f(a)$ can be considered as a function in a. These functions explicitly determine the possible values of b for each given a and their joint graph is exactly $P_1(T_1, T_2)$.

Example 6.5.1. We compute $P_1\big(X_9^{++}, X_9^{++}\big)$ with Singular. For convenience we work over $\mathbb{Q}(a, b, t_1, t_2, t_3, t_4)[x, y]$:

```
> ring R = (0,a,b,t1,t2,t3,t4), (x,y), dp;
> poly f = x^4+a*x^2*y^2+y^4;
```

According to Theorem 6.4.3,

$$S = \big\{\phi \in \mathrm{Aut}_{\mathbb{C}}(\mathbb{C}[[x,y]]) \mid \phi(x) = t_1 x + t_2 y,\ \phi(y) = t_3 x + t_4 y,\\ t_1, \ldots, t_4 \in \mathbb{C}\big\}$$

is a sufficient set of automorphisms for X_9:

```
> map phi = R, t1*x+t2*y, t3*x+t4*y;
> matrix C = coef(phi(f), xy);
> print(C);
```

```
x^4,   x^3*y, x^2*y^2,x*y^3, y^4,
C[2,1],C[2,2],C[2,3], C[2,4],C[2,5]
> C[2,1];
(a*t1^2*t3^2+t1^4+t3^4)
```

Now the second row of the matrix `C` contains the coefficients of `phi`$\big(X_9^{++}(a)\big)$, `C[2, 1]` for instance is the one belonging to x^4. Using the corresponding coefficients of $X_9^{++}(b) = x^4 + b \cdot x^2y^2 + y^4$, we can define the ideal I as above:

```
> matrix D[1][5] = 1, 0, b, 0, 1;
> ideal I = C[2,1..5]-D[1,1..5];
```

As the next step, we map this ideal to $\mathbb{Q}(a)[b, t_1, t_2, t_3, t_4]$ and eliminate the variables t_i:

```
> ring S = (0,a), (b,t1,t2,t3,t4), dp;
> ideal I = imap(R, I);
> ideal g = eliminate(I, t1*t2*t3*t4);
> g;
g[1]=(a^4-8*a^2+16)*b^6+(-a^6-720*a^2-1152)*b^4
+(8*a^6+720*a^4+20736)*b^2+(-16*a^6+1152*a^4-20736*a^2)
```

Factorizing the single generator of this ideal finally yields the functions $f_1^{1,1}, \ldots, f_6^{1,1}$ defined in Theorem 6.6.1. Note that $a^2 \neq 4$.

```
> factorize(g[1]);
[1]:
   _[1]=1
   _[2]=b+(-a)
   _[3]=b+(a)
   _[4]=(a-2)*b+(-2*a-12)
   _[5]=(a+2)*b+(-2*a+12)
   _[6]=(a+2)*b+(2*a-12)
   _[7]=(a-2)*b+(2*a+12)
[2]:
   1,1,1,1,1,1,1
```

6.5.2 How to Compute $P_2(T_1, T_2)$

Given $P_1(T_1, T_2)$, it is easy to compute $P_2(T_1, T_2)$ even "by hand" because we have

$$P_2(T_1, T_2) = P_1(T_1, T_2) \cap (\mathbb{R} \times \mathbb{R}) .$$

Example 6.5.2. Continuing the example above, the values $f_1^{1,1}(a), \ldots, f_6^{1,1}(a)$ are clearly real for $a \in \mathbb{R}$, cf. Theorem 6.6.1. The set $P_2\big(X_9^{++}, X_9^{++}\big)$ is thus the joint graph of these functions over $\mathbb{R} \setminus \{-2, 2\}$.

To give another example, for $T_1 = T_2 = X_9^{+-}$ the set $P_1(T_1, T_2)$ is the joint graph of $f_1^{i,i}, \ldots, f_6^{i,i}$ over $\mathbb{C} \setminus \{-2i, 2i\}$. The values of $f_1^{i,i}(a)$ and $f_2^{i,i}(a)$ are clearly real for $a \in \mathbb{R}$, but those of $f_3^{i,i}(a), \ldots, f_6^{i,i}(a)$ are not except at some exceptional points which are already covered by $f_1^{i,i}$ and $f_2^{i,i}$. So in this case we have

$$P_2\big(X_9^{+-}, X_9^{+-}\big) = \Gamma_{\mathbb{R}}\Big(f_1^{i,i}, f_2^{i,i}\Big) = \Gamma_{\mathbb{R}}\Big(f_1^{1,1}, f_2^{1,1}\Big)\,.$$

6.5.3 How to Compute $P_3(T_1, T_2)$

Since $P_3(T_1, T_2) \subset P_2(T_1, T_2)$ by definition, we can determine $P_3(T_1, T_2)$ by checking for each pair $(a, b) \in P_2(T_1, T_2)$ whether or not there is a *real* coordinate transformation $\phi \in \mathrm{Aut}_{\mathbb{R}}(\mathbb{R}[[x, y]])$ which takes $\mathrm{NF}(T_1(a))$ to $\mathrm{NF}(T_2(b))$. This can be reduced to a finite problem as follows: Let g_j, $j \in \{1, \ldots, s\}$, be the irreducible factors of the polynomial g as in Section 6.5.1. Then in all the cases, $P_2(T_1, T_2)$ is a finite union of "branches" of the form $V(g_j)$ and some exceptional points. We can check whether a branch $V(g_j)$ or an exceptional point (q_a, q_b) in $P_2(T_1, T_2)$ belongs $P_3(T_1, T_2)$ by simply adding appropriate relations to the ideal I and looking at the real solutions of the resulting ideal. In other words, we define $J := I + \langle g_j \rangle$ or $J := I + \langle a - q_a, b - q_b \rangle$, respectively, and investigate $V_{\mathbb{R}}(J)$. Note that we have $I \subset \mathbb{R}[a, b, t_1, \ldots, t_r]$ and $g_j \in \mathbb{R}[a, b]$ and thus $J \subset \mathbb{R}[a, b, t_1, \ldots, t_r]$ in all the cases.

$P_3(T_1, T_2)$ is the image of $V_{\mathbb{R}}(J) \subset \mathbb{A}_{\mathbb{R}}^{2+r}$ under the projection $\mathbb{A}_{\mathbb{R}}^{2+r} \to \mathbb{A}_{\mathbb{R}}^2$, i.e. we have $(p_a, p_b) \in P_3(T_1, T_2)$ if and only if there is a coordinate transformation with real coefficients $(p_{t_1}, \ldots, p_{t_r})$ such that $(p_a, p_b, p_{t_1}, \ldots, p_{t_r})$ is an element of $V_{\mathbb{R}}(J) \subset \mathbb{A}_{\mathbb{R}}^{2+r}$.

It turns out that the ideal J is quite complicated in some cases and that it can be difficult to determine $V_{\mathbb{R}}(J)$ by just computing a Gröbner basis of J. One way out is then to consider J as a parametric ideal $J \subset \mathbb{R}(a)[b, t_1, \ldots, t_r]$ and to compute a Gröbner cover thereof by using the SINGULAR library `grobcov.lib` [49]. A Gröbner cover completely describes the possible shapes of Gröbner bases of J for different values of a. It contains a generic Gröbner basis of J, i.e. one which is a Gröbner basis except for finitely many exceptional values of a, and additionally Gröbner bases of J for each of these exceptional values. The ideals in a Gröbner cover of J typically have a much easier structure than J itself. We can thus treat them one by one and determine their real solutions. We will often find generators such as $(t_j)^4 + 1$, indicating that the vanishing set over $\mathbb{R}$ of this ideal is empty.

If any of the ideals in the Gröbner cover of J are still to complicated and if their vanishing set over $\mathbb{R}$ cannot be easily read off, another trick is to compute a primary decomposition of these ideals with the SINGULAR library `primdec.lib` [44]. Typically, it is then easy to see that some of the

primary components have no solutions over $\mathbb{R}$ whereas the real solutions of the remaining components can be easily determined.

Example 6.5.3. We have already seen in Example 6.5.2 that $P_2\big(X_9^{++}, X_9^{++}\big)$ is the joint graph of $f_1^{1,1}, \ldots, f_6^{1,1}$ over $\mathbb{R}\setminus\{-2,2\}$. We now have to check for each of these functions whether their graph is also contained in the set $P_3\big(X_9^{++}, X_9^{++}\big)$.

This is clearly the case for $f_1^{1,1} = \mathrm{id}$. To check this for $f_3^{1,1}$, we continue the SINGULAR session from Example 6.5.1, add the corresponding relation to the ideal I and compute a Gröbner cover of the resulting ideal J:

```
> ideal J = I, (a-2)*b+(-2*a-12);
> LIB "grobcov.lib";
> grobcov(J);
```

The output of the last command is too long to be printed here. We will find that the Gröbner basis of J for generic a contains the generators $(t_2)^2+(t_4)^2$ and $(t_3)^2+(t_4)^2$ which imply $t_2 = t_3 = t_4 = 0$ for any real solution of this ideal. But this is a contradiction to `phi` $\in \mathrm{Aut}_{\mathbb{R}}(\mathbb{R}[[x,y]])$. The exceptional cases for the parameter a are $a+2=0$, $a-2=0$, $a^2+12=0$, $a+6=0$, $a-6=0$, and $a=0$. The first two cases are excluded by the definition of the singularity type X_9^{++}, $a^2+12=0$ would imply $a \notin \mathbb{R}$, for $a+6=0$ and $a=0$ the corresponding Gröbner bases of J contain generators similar to those mentioned above, and finally $a-6=0$ implies $b-6=0$ such that this case is already covered by the graph of $f_1^{1,1}$.

To give one more example, let us consider $f_5^{1,1}$:

```
> J = I, (a+2)*b+(2*a-12);
> grobcov(J);
```

The crucial generator of the Gröbner basis of J for generic a is now the polynomial $(a+2)(t_4)^4-1$ which has a real root if and only if $a > -2$. Considering the other generators, it is easy to see that given $t_4 \in \mathbb{R}$, $t_1 = t_2 = t_3 = -t_4$ is a real solution. The exceptional values of a in this case are the same as above and again, we do not have to consider $a+2=0$, $a-2=0$, and $a^2+12=0$. The relation $a+6=0$ implies $b+6=0$ which is already covered by $f_1^{1,1}$. Finally, $t_1 = t_2 = t_3 = \frac{1}{\sqrt[4]{2}}$, $t_4 = -\frac{1}{\sqrt[4]{2}}$ and $t_1 = t_2 = t_3 = \frac{1}{\sqrt[4]{8}}$, $t_4 = -\frac{1}{\sqrt[4]{8}}$ are real solutions for the cases $a=0$ and $a-6=0$, respectively. To sum up, the graph of $f_5^{1,1}$ over $\mathbb{R}^{>-2}$ belongs to $P_3\big(X_9^{++}, X_9^{++}\big)$, but not the part over $\mathbb{R}^{<-2}$.

Continuing in this manner, one can show that $f_2^{1,1}$, $f_4^{1,1}$ and $f_6^{1,1}$ do not contribute any additional points, so we get

$$P_3\big(X_9^{++}, X_9^{++}\big) = \Gamma_{\mathbb{R}'}\Big(f_1^{1,1}\Big) \cup \Gamma_{\mathbb{R}^{>-2}}\Big(f_5^{1,1}\Big)$$

where $\mathbb{R}' := \mathbb{R}\setminus\{-2,2\}$.

Remark 6.5.4. With the above notations, the irreducible factors g_j, $j = 1, \ldots, s$, of the polynomial g are luckily of degree 1 in b in almost all cases. If one of those factors, say g_1, has degree in b greater than 1, and if additionally the corresponding ideal $J = I + \langle g_1 \rangle$ has both real and complex solutions, then an extra calculation is needed: Let $f_1(a), \ldots, f_k(a)$ be the roots of g_1 in b as above, i.e. $g_1 = (b - f_1(a)) \ldots (b - f_k(a))$ (over $\mathbb{C}(a)$ or over a suitable extension thereof if necessary). Then we have to check which of these roots $f_1(a), \ldots, f_k(a)$ belong to the real solutions of J and which of them can only reached via complex transformations.

This is especially crucial for the singularities of type J_{10} in order to distinguish between $f_3^{\sigma,\rho}$, $f_4^{\sigma,\rho}$, $f_5^{\sigma,\rho}$, and $f_6^{\sigma,\rho}$, cf. Theorem 6.6.2.

Remark 6.5.5. The hyperbolic singularity types listed in Table 6.1 are actually infinite series of types. One might argue that the computations described in Sections 6.5.1 to 6.5.3 must be carried out for each single $k > 0$ (for J_{10+k} and X_{9+k}) and for each pair $r, s > 4$ (for $Y_{r,s}$) in order to check the results presented in Theorems 6.6.4 to 6.6.6. This is, of course, impossible in practice. But it turns out that the results are periodic in k and r, s, respectively. Hence it suffices to carry these computations out for sufficiently many values of k and r, s. If we closely examine the intermediate steps, then we can easily check that the results are indeed periodic.

6.5.4 The Special Type $\widetilde{Y}_r$

Theorem 6.4.3 does not give any sufficient set for subtypes of $\widetilde{Y}_r$ and indeed it turns out that there is no degree-bounded sufficient set for this case, cf. Remark 6.5.9.

But since $\widetilde{Y}_r$ is $\mathbb{C}$-equivalent to $Y_{r,r}$, we can use the structure of the equivalence classes of $Y_{r,r}$ (cf. Theorem 6.6.6) to determine $P_1(T_1, T_2)$, $P_2(T_1, T_2)$, and $P_3(T_1, T_2)$ for $T_1, T_2 \in \{\widetilde{Y}_r^+, \widetilde{Y}_r^-\}$. To do so, let us first define the principal part of a power series.

Definition 6.5.6. Let $f \in \mathbb{K}[[x_1, \ldots, x_n]]$ be a power series, let Γ_f be its Newton polygon, and let f_0 be the sum of those terms of f which lie on Γ_f. Then we call f_0 the principal part of f.

The following result is due to [3, Corollary 9.9].

Lemma 6.5.7. *Let $f \in \mathbb{C}[[x, y]]$ be a power series whose principal part is of the form $f_0 = x^a + \lambda x^2 y^2 + y^b$, where $0 \neq \lambda \in \mathbb{C}$, $a \geq 4$, and $b \geq 5$. Then f and its principal part f_0 are $\mathbb{C}$-equivalent, i.e. $f \overset{\mathbb{C}}{\sim} f_0$.*

Based upon this lemma, we can now specify an explicit equivalence between the normal forms of $\widetilde{Y}_r$ and $Y_{r,r}$.

Lemma 6.5.8. *For any $r > 4$ and any $a \in \mathbb{C} \setminus \{0\}$, we have*

$$\left(a, \left(\tfrac{1}{4}\right)^r a^2\right) \in P_1\left(\widetilde{Y}_r^+, Y_{r,r}^{++}\right) \cap P_1\left(\widetilde{Y}_r^-, Y_{r,r}^{-+}\right).$$

Figure 6.1: Equivalences between $\mathrm{NF}\big(\widetilde{Y}_r^+\big)$ and $\mathrm{NF}\big(Y_{r,r}^{++}\big)$ $\big(c := \big(\frac{1}{4}\big)^r\big)$

$$\begin{array}{ccc} \mathrm{NF}\Big(\widetilde{Y}_r^+(a)\Big) & \longleftrightarrow & \mathrm{NF}\Big(\widetilde{Y}_r^+\big(\pm\sqrt{\zeta}a\big)\Big) \\ \updownarrow & \circlearrowleft & \updownarrow \\ \mathrm{NF}\big(Y_{r,r}^{++}\big(ca^2\big)\big) & \longleftrightarrow & \mathrm{NF}\big(Y_{r,r}^{++}\big(\zeta ca^2\big)\big) \end{array}$$

Proof. Let $\phi \in \mathrm{Aut}_{\mathbb{C}}(\mathbb{C}[[x,y]])$ be the coordinate transformation defined by $\phi(x) := \frac{1}{2}(x+y)$ and $\phi(y) := \frac{1}{2}i(x-y)$. Then the principal parts of $\phi\big(\mathrm{NF}\big(\widetilde{Y}_r^+(a)\big)\big)$ and $\phi\big(\mathrm{NF}\big(\widetilde{Y}_r^-(a)\big)\big)$ are of the form $\big(\frac{1}{2}\big)^r a \cdot x^r + \lambda x^2y^2 + \big(\frac{1}{2}\big)^r a \cdot y^r$ with $\lambda = 1$ and $\lambda = -1$, respectively, so the result follows from Lemma 6.5.7. □

Section 6.5.1 tells us how to compute $P_1(T_1,T_2)$ for $T_1,T_2 \in \{Y_{r,r}^{++}, Y_{r,r}^{-+}\}$, cf. Theorem 6.6.6. We can use this data and the above lemma to compute $P_1(T_1,T_2)$ for $T_1,T_2 \in \big\{\widetilde{Y}_r^+, \widetilde{Y}_r^-\big\}$. Let us consider the case $P_1\big(\widetilde{Y}_r^+, \widetilde{Y}_r^+\big)$, the other cases follow similarly. According to Lemma 6.5.8, $\mathrm{NF}\big(\widetilde{Y}_r^+(a)\big)$ is $\mathbb{C}$-equivalent to $\mathrm{NF}\big(Y_{r,r}^{++}\big(ca^2\big)\big)$ with $c := \big(\frac{1}{4}\big)^r$ for any $r > 4$ and any $a \in \mathbb{C}\setminus\{0\}$. This in turn is $\mathbb{C}$-equivalent to $\mathrm{NF}\big(Y_{r,r}^{++}\big(\zeta ca^2\big)\big)$ for any ζ satisfying $\zeta^l - 1 = 0$ where $l = \gcd(2, r+1)$, cf. Theorem 6.6.6. Applying Lemma 6.5.8 again leads to $\mathrm{NF}\big(Y_{r,r}^{++}\big(\zeta ca^2\big)\big) \overset{\mathbb{C}}{\sim} \mathrm{NF}\big(\widetilde{Y}_r^+\big(\pm\sqrt{\zeta}a\big)\big)$, and we thus get the diagram shown in Figure 6.1.

This proves $\mathrm{NF}\big(\widetilde{Y}_r^+(a)\big) \overset{\mathbb{C}}{\sim} \mathrm{NF}\big(\widetilde{Y}_r^+\big(\pm\sqrt{\zeta}a\big)\big)$ for ζ as above, and since the diagram is commutative, there are no equivalences for other values of the parameters than these. Hence

$$P_1\Big(\widetilde{Y}_r^+, \widetilde{Y}_r^+\Big) = C\Big(X^{2l} - 1\Big)$$

with l as above. The set $P_2\big(\widetilde{Y}_r^+, \widetilde{Y}_r^+\big)$ can now be determined as in Section 6.5.2. In fact it is easy to see that

$$P_2\Big(\widetilde{Y}_r^+, \widetilde{Y}_r^+\Big) = R\big(X^2 - 1\big)\,.$$

We clearly have $(a,a) \in P_3\big(\widetilde{Y}_r^+, \widetilde{Y}_r^+\big)$ for $a \in \mathbb{R}\setminus\{0\}$, and also $(a,-a) \in P_3\big(\widetilde{Y}_r^+, \widetilde{Y}_r^+\big)$ if r is odd. For the case where r is even, let us consider $\mathrm{NF}\big(\widetilde{Y}_r^+(a)\big)$ as a function in x and y over $\mathbb{R}^2$ and let the parameter a be positive. In this case the function $\mathrm{NF}\big(\widetilde{Y}_r^+(a)\big) = \big(x^2+y^2\big)^2 + ax^r$ takes only non-negative values whereas $\mathrm{NF}\big(\widetilde{Y}_r^+(-a)\big) = \big(x^2+y^2\big)^2 - ax^r$ attains

also negative values. Hence there is no real coordinate transformation which takes $\mathrm{NF}\big(\widetilde{Y}_r^+(a)\big)$ to $\mathrm{NF}\big(\widetilde{Y}_r^+(-a)\big)$. The argument is similar for $a < 0$. To sum up, we have

$$P_3\Big(\widetilde{Y}_r^+, \widetilde{Y}_r^+\Big) = \begin{cases} R(X^2-1), & \text{if } r \text{ is odd,} \\ R(X-1), & \text{if } r \text{ is even.} \end{cases}$$

Remark 6.5.9. Let $r \geq 8$ be a multiple of 4 and let $\phi_r \in \mathrm{Aut}_{\mathbb{C}}(\mathbb{C}[[x,y]])$ be a coordinate transformation which takes $f := \mathrm{NF}\big(\widetilde{Y}_r^+(a)\big)$ to $\mathrm{NF}\big(\widetilde{Y}_r^+(-a)\big)$. Assume that the degree of both $\phi_r(x)$ and $\phi_r(y)$ is less than $\frac{r}{4}$ and let $f = f_0 + f_1$ be decomposed into its principal part $f_0 := \big(x^2+y^2\big)^2$ and $f_1 = ax^r$. Then we have

$$\phi(f) = \phi(f_0) + \phi(f_1) = \big(x^2+y^2\big)^2 - ax^r$$

where the degree of $\phi(f_0)$ is less than r. Therefore $\phi(f_0) = \phi_0(f_0) = \big(x^2+y^2\big)^2$ and $\phi(f_1) = \phi_0(f_1) = -ax^r$. If ϕ_0 is given by $\phi_0(x) = \alpha x + \beta y$, $\phi_0(y) = \gamma x + \delta y$ with $\alpha, \beta, \gamma, \delta \in \mathbb{C}$, the second of these two equations implies $\beta = 0$ and $\alpha^r = -1$, but the first one in turn implies $\gamma = 0$ and $\alpha^4 = 1$ which is a contradiction.

So the degree of either $\phi_r(x)$ or $\phi_r(y)$ must at least $\frac{r}{4}$. This shows that a degree-bounded sufficient set of coordinate transformations for the pair $\big(\mathrm{NF}\big(\widetilde{Y}_r^+(a)\big), \mathrm{NF}\big(\widetilde{Y}_r^+(a)\big)\big)$ and for arbitrarily high r does not exist.

6.6 Results

In this section we present the sets P_1, P_2, P_3 in table form for every unimodal real singularity type up to corank 2.

Theorem 6.6.1. *The structure of the equivalence classes of the X_9 singularities is as shown in Table 6.3 where for $j = 1, \dots, 6$ and $\rho, \sigma \in \{1, i\}$, the function $f_j^{\rho,\sigma}$ is defined as follows:*

$$f_1^{\rho,\sigma}(a) := +\rho\sigma \cdot a\,, \quad f_3^{\rho,\sigma}(a) := \frac{+2\sigma a + 12\rho\sigma}{a - 2\rho}\,, \quad f_5^{\rho,\sigma}(a) := \frac{-2\sigma a + 12\rho\sigma}{a + 2\rho}\,,$$
$$f_2^{\rho,\sigma}(a) := -\rho\sigma \cdot a\,, \quad f_4^{\rho,\sigma}(a) := \frac{+2\sigma a - 12\rho\sigma}{a + 2\rho}\,, \quad f_6^{\rho,\sigma}(a) := \frac{-2\sigma a - 12\rho\sigma}{a - 2\rho}\,.$$

Furthermore, we use the following notations:

$$\mathbb{C}' := \mathbb{C} \setminus \{-2, 2\}\,, \qquad \mathbb{R}' := \mathbb{R} \setminus \{-2, 2\}\,, \qquad \mathbb{C}'' := \mathbb{C} \setminus \{-2i, 2i\}\,.$$

Table 6.3: P_1, P_2 and P_3 for the X_9 singularities

T_1	T_2	$P_1(T_1, T_2)$	$P_2(T_1, T_2)$	$P_3(T_1, T_2)$
X_9^{++}	X_9^{++}	$\Gamma_{\mathbb{C}'}\big(f_1^{1,1}, \dots, f_6^{1,1}\big)$	$\Gamma_{\mathbb{R}'}\big(f_1^{1,1}, \dots, f_6^{1,1}\big)$	$\Gamma_{\mathbb{R}'}\big(f_1^{1,1}\big)$ $\cup\, \Gamma_{\mathbb{R}'>-2}\big(f_5^{1,1}\big)$
X_9^{--}	X_9^{--}			$\Gamma_{\mathbb{R}'}\big(f_1^{1,1}\big)$ $\cup\, \Gamma_{\mathbb{R}'<+2}\big(f_3^{1,1}\big)$
X_9^{++}	X_9^{--}			$\Gamma_{\mathbb{R}<-2}\big(f_4^{1,1}\big)$
X_9^{--}	X_9^{++}			$\Gamma_{\mathbb{R}>+2}\big(f_6^{1,1}\big)$
X_9^{+-}	X_9^{+-}	$\Gamma_{\mathbb{C}''}\big(f_1^{i,i}, \dots, f_6^{i,i}\big)$	$\Gamma_{\mathbb{R}}\big(f_1^{1,1}, f_2^{1,1}\big)$	$\Gamma_{\mathbb{R}}\big(f_1^{1,1}\big)$
X_9^{-+}	X_9^{-+}			
X_9^{+-}	X_9^{-+}			
X_9^{-+}	X_9^{+-}			
X_9^{++}	X_9^{+-}	$\Gamma_{\mathbb{C}'}\big(f_1^{1,i}, \dots, f_6^{1,i}\big)$	$\{(0,0)\}$ $\cup\, \{(0,-6), (0,6)\}$	$\varnothing$
X_9^{++}	X_9^{-+}			
X_9^{--}	X_9^{+-}			
X_9^{--}	X_9^{-+}			
X_9^{+-}	X_9^{++}	$\Gamma_{\mathbb{C}''}\big(f_1^{i,1}, \dots, f_6^{i,1}\big)$	$\{(0,0)\}$ $\cup\, \{(0,-6), (0,6)\}$	$\varnothing$
X_9^{-+}	X_9^{++}			
X_9^{+-}	X_9^{--}			
X_9^{-+}	X_9^{--}			

Theorem 6.6.2. *The structure of the equivalence classes of the J_{10} singularities is as shown in Table 6.4 where for $j = 1, \dots, 6$ and $\rho, \sigma \in \{-1, +1\}$, the function $f_j^{\rho,\sigma}$ is defined as follows:*

$$f_1^{\rho,\sigma}(a) := +\sqrt{\rho\sigma} \cdot a\,,$$
$$f_2^{\rho,\sigma}(a) := -\sqrt{\rho\sigma} \cdot a\,,$$
$$f_3^{\rho,\sigma}(a) := +\sqrt{\frac{-\rho\sigma(a^2-\rho\cdot 4)(a^2-\rho\cdot 9)+a(a^2-\rho\cdot 3)\sqrt{a^2-\rho\cdot 4}}{2(a^2-\rho\cdot 4)}}\,,$$
$$f_4^{\rho,\sigma}(a) := -\sqrt{\frac{-\rho\sigma(a^2-\rho\cdot 4)(a^2-\rho\cdot 9)+a(a^2-\rho\cdot 3)\sqrt{a^2-\rho\cdot 4}}{2(a^2-\rho\cdot 4)}}\,,$$
$$f_5^{\rho,\sigma}(a) := +\sqrt{\frac{-\rho\sigma(a^2-\rho\cdot 4)(a^2-\rho\cdot 9)-a(a^2-\rho\cdot 3)\sqrt{a^2-\rho\cdot 4}}{2(a^2-\rho\cdot 4)}}\,,$$
$$f_6^{\rho,\sigma}(a) := -\sqrt{\frac{-\rho\sigma(a^2-\rho\cdot 4)(a^2-\rho\cdot 9)-a(a^2-\rho\cdot 3)\sqrt{a^2-\rho\cdot 4}}{2(a^2-\rho\cdot 4)}}\,.$$

In each case, ρ and σ are given by

$$\rho := \begin{cases} +1, & \text{if } T_1 = J_{10}^+\,, \\ -1, & \text{if } T_1 = J_{10}^-\,, \end{cases} \qquad \sigma := \begin{cases} +1, & \text{if } T_2 = J_{10}^+\,, \\ -1, & \text{if } T_2 = J_{10}^-\,. \end{cases}$$

Furthermore, we use the following notations:

$$\xi := \tfrac{3}{\sqrt{2}}\,, \qquad I_1 :=]-\infty, -\xi[\subset \mathbb{R}\,,$$
$$\mathbb{C}' := \mathbb{C} \setminus \{-2, 2\}\,, \qquad I_2 :=]-\xi, -2[\subset \mathbb{R}\,,$$
$$\mathbb{R}' := \mathbb{R} \setminus \{-2, 2\}\,, \qquad I_3 :=]+2, +\xi[\subset \mathbb{R}\,,$$
$$\mathbb{C}'' := \mathbb{C} \setminus \{-2i, 2i\}\,, \qquad I_4 :=]+\xi, +\infty[\subset \mathbb{R}\,.$$

Remark 6.6.3. In Theorem 6.6.2, the definitions of $f_1^{\rho,\sigma}, \dots, f_6^{\rho,\sigma}$ involve square roots of possibly complex values. These square roots are defined as follows: For any complex number $z = re^{i\phi} \in \mathbb{C}$ with $r, \phi \in \mathbb{R}$, $r > 0$, and $0 \le \phi < 2\pi$, we set

$$\sqrt{z} := \sqrt{r} e^{i\frac{\phi}{2}}\,.$$

In particular, $\mathrm{Im}(\sqrt{z}) > 0$ for all $z \in \mathbb{C} \setminus \mathbb{R}^{>0}$ and $\sqrt{z} \ge 0$ for all $z \in \mathbb{R}^{>0}$.

Theorem 6.6.4. *The structure of the equivalence classes of the J_{10+k} singularities is as shown in Table 6.5 where in each case, l and s are given by*

$$l := \frac{6}{\gcd(6,k)}, \text{ and}$$
$$s := \begin{cases} +1, & \text{if } k \equiv 2 \pmod 4, \\ -1, & \text{else.} \end{cases}$$

Table 6.4: P_1, P_2 and P_3 for the J_{10} singularities

T_1	T_2	$P_1(T_1,T_2)$	$P_2(T_1,T_2)$	$P_3(T_1,T_2)$
J_{10}^+	J_{10}^+	$\Gamma_{\mathbb{C}'}(f_1^{\rho,\sigma},\ldots,f_6^{\rho,\sigma})$	$\Gamma_{\mathbb{R}'}(f_1^{\rho,\sigma},f_2^{\rho,\sigma})$ $\cup\,\Gamma_{\mathbb{R}^{>+2}}(f_3^{\rho,\sigma},f_4^{\rho,\sigma})$ $\cup\,\Gamma_{\mathbb{R}^{<-2}}(f_5^{\rho,\sigma},f_6^{\rho,\sigma})$ $\cup\{(0,-\xi),(0,+\xi)\}$ $\cup\{(-\xi,0),(+\xi,0)\}$	$\Gamma_{\mathbb{R}'}(f_1^{\rho,\sigma})$ $\cup\,\Gamma_{\mathbb{R}^{>+2}}(f_4^{\rho,\sigma})$ $\cup\,\Gamma_{\mathbb{R}^{<-2}}(f_5^{\rho,\sigma})$
J_{10}^-	J_{10}^-	$\Gamma_{\mathbb{C}''}(f_1^{\rho,\sigma},\ldots,f_6^{\rho,\sigma})$	$\Gamma_{\mathbb{R}}(f_1^{\rho,\sigma},f_2^{\rho,\sigma})$	$\Gamma_{\mathbb{R}}(f_1^{\rho,\sigma})$
J_{10}^+	J_{10}^-	$\Gamma_{\mathbb{C}'}(f_1^{\rho,\sigma},\ldots,f_6^{\rho,\sigma})$	$\{(0,0)\}$ $\cup\,\Gamma_{\mathbb{R}^{>+2}}(f_3^{\rho,\sigma},f_4^{\rho,\sigma})$ $\cup\,\Gamma_{\mathbb{R}^{<-2}}(f_5^{\rho,\sigma},f_6^{\rho,\sigma})$	$\Gamma_{I_4}(f_3^{\rho,\sigma})$ $\cup\,\Gamma_{I_3}(f_4^{\rho,\sigma})$ $\cup\,\Gamma_{I_2}(f_5^{\rho,\sigma})$ $\cup\,\Gamma_{I_1}(f_6^{\rho,\sigma})$ $\cup\{(-\xi,0)\}$ $\cup\{(+\xi,0)\}$
J_{10}^-	J_{10}^+	$\Gamma_{\mathbb{C}''}(f_1^{\rho,\sigma},\ldots,f_6^{\rho,\sigma})$	$\{(0,0)\}$ $\cup\,\Gamma_{\mathbb{R}}(f_3^{\rho,\sigma},\ldots,f_6^{\rho,\sigma})$	$\Gamma_{\mathbb{R}}(f_3^{\rho,\sigma},f_6^{\rho,\sigma})$

Table 6.5: P_1, P_2 and P_3 for the J_{10+k} singularities

T_1	T_2	$P_1(T_1,T_2)$	$P_2(T_1,T_2)$	$P_3(T_1,T_2)$
J_{10+k}^+	J_{10+k}^+	$C(X^l-1)$	$R(X^l-1)$	$R(X^l-1)$
J_{10+k}^-	J_{10+k}^-			
J_{10+k}^+	J_{10+k}^-	$C(X^l-s)$	$R(X^l-s)$	$\varnothing$
J_{10+k}^-	J_{10+k}^+			

Theorem 6.6.5. *The structure of the equivalence classes of the X_{9+k} singularities is as shown in Table 6.6 where in each case, l and s are given by*

$$l := \frac{4}{\gcd(4,k)}, \text{ and}$$
$$s := \begin{cases} +1, & \text{if } k \equiv 4 \pmod 8, \\ -1, & \text{else.} \end{cases}$$

Table 6.6: P_1, P_2 and P_3 for the X_{9+k} singularities

T_1	T_2	$P_1(T_1,T_2)$	$P_2(T_1,T_2)$	$P_3(T_1,T_2)$
X_{9+k}^{++}	X_{9+k}^{++}	$C(X^l-1)$	$R(X^l-1)$	$R(X^{k+1}-1)$
X_{9+k}^{+-}	X_{9+k}^{+-}			
X_{9+k}^{-+}	X_{9+k}^{-+}			
X_{9+k}^{--}	X_{9+k}^{--}			
X_{9+k}^{++}	X_{9+k}^{+-}	$C(X^l-1)$	$R(X^l-1)$	$\varnothing$
X_{9+k}^{+-}	X_{9+k}^{++}			
X_{9+k}^{-+}	X_{9+k}^{--}			
X_{9+k}^{--}	X_{9+k}^{-+}			
X_{9+k}^{++}	X_{9+k}^{-+}	$C(X^l-s)$	$R(X^l-s)$	$\varnothing$
X_{9+k}^{+-}	X_{9+k}^{--}			
X_{9+k}^{-+}	X_{9+k}^{++}			
X_{9+k}^{--}	X_{9+k}^{+-}			
X_{9+k}^{++}	X_{9+k}^{--}	$C(X^l-s)$	$R(X^l-s)$	$\varnothing$
X_{9+k}^{+-}	X_{9+k}^{-+}			
X_{9+k}^{-+}	X_{9+k}^{+-}			
X_{9+k}^{--}	X_{9+k}^{++}			

Theorem 6.6.6. *The structure of the equivalence classes of the $Y_{r,s}$ singularities is as shown in Table 6.7 where in each case, l, s_1 and s_2 are given by*

$$l := \frac{r}{\gcd(r,s)} \cdot \gcd(2, r+1, s+1)\,,$$

$$s_1 := \begin{cases} +1, & \text{if } r \equiv 0 \pmod 4 \text{ or } s \equiv 0 \pmod 4, \\ -1, & \text{else,} \end{cases}$$

$$s_2 := \begin{cases} +1, & \text{if } r \not\equiv 0 \pmod 2 \text{ or } \frac{s}{\gcd(r,s)} \equiv 0 \pmod 2, \\ -1, & \text{else.} \end{cases}$$

In the special case where $r = s$, additional equivalences occur. They are listed in Table 6.8.

Remark 6.6.7. Note that there are also equivalences between subtypes of $Y_{r,s}$ and subtypes of $Y_{s,r}$ which can be obtained by just swapping the variables x and y. For $r = s$ these are exactly the additional equivalences listed in Table 6.8. But equivalences of this kind also occur for $r \neq s$, e.g. we have $R(X-1) \subset P_1(Y_{5,7}^{++}, Y_{7,5}^{++})$, but we do not consider those cases in Theorem 6.6.6.

Table 6.7: P_1, P_2 and P_3 for the $Y_{r,s}$ singularities

T_1	T_2	$P_1(T_1,T_2)$	$P_2(T_1,T_2)$	$P_3(T_1,T_2)$
$Y_{r,s}^{++}$	$Y_{r,s}^{++}$	$C(X^l-1)$	$R(X^l-1)$	$R(X^{s+1}-1)$
$Y_{r,s}^{-+}$	$Y_{r,s}^{-+}$			
$Y_{r,s}^{+-}$	$Y_{r,s}^{+-}$			
$Y_{r,s}^{--}$	$Y_{r,s}^{--}$			
$Y_{r,s}^{++}$	$Y_{r,s}^{-+}$	$C(X^l-s_1)$	$R(X^l-s_1)$	$\varnothing$
$Y_{r,s}^{-+}$	$Y_{r,s}^{++}$			
$Y_{r,s}^{+-}$	$Y_{r,s}^{--}$			
$Y_{r,s}^{--}$	$Y_{r,s}^{+-}$			
$Y_{r,s}^{++}$	$Y_{r,s}^{+-}$	$C(X^l-s_2)$	$R(X^l-s_2)$	$R(X^l-s_2)$, if $r \not\equiv 0 \pmod 2$, and $\varnothing$, if $r \equiv 0 \pmod 2$
$Y_{r,s}^{-+}$	$Y_{r,s}^{--}$			
$Y_{r,s}^{+-}$	$Y_{r,s}^{++}$			
$Y_{r,s}^{--}$	$Y_{r,s}^{-+}$			
$Y_{r,s}^{++}$	$Y_{r,s}^{--}$	$C(X^l-s_1s_2)$	$R(X^l-s_1s_2)$	$\varnothing$
$Y_{r,s}^{-+}$	$Y_{r,s}^{+-}$			
$Y_{r,s}^{+-}$	$Y_{r,s}^{-+}$			
$Y_{r,s}^{--}$	$Y_{r,s}^{++}$			

Table 6.8: Additional equivalences for the $Y_{r,s}$ singularities in the special case $r = s$

T_1	T_2	Additional elements of $P_3(T_1,T_2)$
$Y_{r,s}^{++}$	$Y_{r,s}^{+-}$	$\begin{cases} R(X+1), & \text{if } r \equiv 0 \pmod 2 \text{ and } a<0 \\ \varnothing, & else \end{cases}$
$Y_{r,s}^{-+}$	$Y_{r,s}^{--}$	
$Y_{r,s}^{+-}$	$Y_{r,s}^{++}$	$\begin{cases} R(X+1), & \text{if } r \equiv 0 \pmod 2 \text{ and } a>0 \\ \varnothing, & else \end{cases}$
$Y_{r,s}^{--}$	$Y_{r,s}^{-+}$	

Theorem 6.6.8. *The structure of the equivalence classes of the $\widetilde{Y}_r$ singularities is as shown in Table 6.9 where in each case, l and s are given by*

$$l := 2 \cdot \gcd(2, r+1), \text{ and}$$
$$s := \begin{cases} +1, & \text{if } r \equiv 0 \pmod 4, \\ -1, & \text{else.} \end{cases}$$

Table 6.9: P_1, P_2 and P_3 for the $\widetilde{Y}_r$ singularities

T_1	T_2	$P_1(T_1,T_2)$	$P_2(T_1,T_2)$	$P_3(T_1,T_2)$
$\widetilde{Y}_r^+$	$\widetilde{Y}_r^+$	$C(X^l-1)$	$R(X^2-1)$	$\begin{cases} R(X^2-1), & \text{if } r \equiv 1 \pmod 2 \\ R(X-1), & \text{if } r \equiv 0 \pmod 2 \end{cases}$
$\widetilde{Y}_r^-$	$\widetilde{Y}_r^-$			
$\widetilde{Y}_r^+$	$\widetilde{Y}_r^-$	$C(X^l-s)$	$R(X^2-s)$	$\varnothing$
$\widetilde{Y}_r^-$	$\widetilde{Y}_r^+$			

Theorem 6.6.9. *The structure of the equivalence classes of the exceptional unimodal singularities is as shown in Table 6.10.*

Table 6.10: P_1, P_2 and P_3 for the exceptional unimodal singularities

T_1	T_2	$P_1(T_1, T_2)$	$P_2(T_1, T_2)$	$P_3(T_1, T_2)$
E_{12}	E_{12}	$C_0(X^{21}-1)$	$R_0(X-1)$	$R_0(X-1)$
E_{13}	E_{13}	$C_0(X^{15}-1)$	$R_0(X-1)$	$R_0(X-1)$
E_{14}^+	E_{14}^+	$C_0(X^{12}-1)$	$R_0(X^2-1)$	$R_0(X-1)$
E_{14}^-	E_{14}^-	$C_0(X^{12}-1)$	$R_0(X^2-1)$	$R_0(X-1)$
E_{14}^+	E_{14}^-	$C_0(X^{12}+1)$	$\varnothing$	$\varnothing$
E_{14}^-	E_{14}^+	$C_0(X^{12}+1)$	$\varnothing$	$\varnothing$
Z_{11}	Z_{11}	$C_0(X^{15}-1)$	$R_0(X-1)$	$R_0(X-1)$
Z_{12}	Z_{12}	$C_0(X^{11}-1)$	$R_0(X-1)$	$R_0(X-1)$
Z_{13}^+	Z_{13}^+	$C_0(X^9-1)$	$R_0(X-1)$	$R_0(X-1)$
Z_{13}^-	Z_{13}^-	$C_0(X^9-1)$	$R_0(X-1)$	$R_0(X-1)$
Z_{13}^+	Z_{13}^-	$C_0(X^9+1)$	$R_0(X+1)$	$\varnothing$
Z_{13}^-	Z_{13}^+	$C_0(X^9+1)$	$R_0(X+1)$	$\varnothing$
W_{12}^+	W_{12}^+	$C_0(X^{10}-1)$	$R_0(X^2-1)$	$R_0(X-1)$
W_{12}^-	W_{12}^-	$C_0(X^{10}-1)$	$R_0(X^2-1)$	$R_0(X-1)$
W_{12}^+	W_{12}^-	$C_0(X^{10}+1)$	$\varnothing$	$\varnothing$
W_{12}^-	W_{12}^+	$C_0(X^{10}+1)$	$\varnothing$	$\varnothing$
W_{13}^+	W_{13}^+	$C_0(X^8-1)$	$R_0(X^2-1)$	$R_0(X-1)$
W_{13}^-	W_{13}^-	$C_0(X^8-1)$	$R_0(X^2-1)$	$R_0(X-1)$
W_{13}^+	W_{13}^-	$C_0(X^8+1)$	$\varnothing$	$\varnothing$
W_{13}^-	W_{13}^+	$C_0(X^8+1)$	$\varnothing$	$\varnothing$

6.7 Interpretation of the Results

Looking more closely at Theorem 6.4.3 and Section 6.6, it turns out that the structure of the equivalence classes is quite simple for some singularity types whereas it is very involved for others. To describe this in more detail, let us first consider the *sufficient sets* given in Theorem 6.4.3:

- It suffices to work with scalings of the form $\phi(x) = \alpha x$, $\phi(y) = \beta y$ as coordinate transformations to figure out the structure of the equivalence classes of the hyperbolic and exceptional unimodal singularities. (To be precise, for $Y_{r,s}$ with $r = s$ we also have to take into account transformations of the form $\phi(x) = \beta y$, $\phi(y) = \alpha x$ where the variables are swapped, cf. Table 6.2.)

- For the two parabolic types X_9 and J_{10}, scalings are not sufficient. Instead, we have to consider more complicated transformations which involve more terms.

- One can see from the proof of Theorem 6.4.3 that these differences reflect the different shapes of the normal forms: The normal forms of the hyperbolic singularity types listed in Table 6.2 are weighted quasihomogeneous, those of the exceptional singularity types are semi-quasihomogeneous. Both shapes turn out to be very restrictive w.r.t. possible coordinate transformations. In contrast to this, the normal forms of the parabolic types are quasihomogeneous and thus allow for more freedom in this regard.

- The singularity type $\widetilde{Y}_r$ is an exception. It is complex equivalent to $Y_{r,r}$, but it appears as a separate singularity type over $\mathbb{R}$. There is no degree-bounded sufficient set for the normal form of this type (cf. Remark 6.5.9), so the computational methods described in Sections 6.5.1 and 6.5.3 do not work. Instead, we have to use other methods, cf. Section 6.5.4.

As a consequence of the differences w.r.t. sufficient sets described above, there are two *general forms of equivalences* as presented in Section 6.6:

- For the hyperbolic and the exceptional singularities, the equivalences between different subtypes can be described by constant factors. More precisely, if T_1 and T_2 are subtypes of the same hyperbolic or exceptional main singularity type, then there exists a finite set of constants $r_1, \ldots, r_m \in \mathbb{C}$ such that the equivalences between the normal forms of T_1 and T_2 are exactly those of the form $\mathrm{NF}(T_1(a)) \sim \mathrm{NF}(T_2(r_i a))$ with $a \in \mathbb{C}$ or $a \in \mathbb{R}$ as appropriate. Therefore we use the notations $C(p(X))$, $R(p(X))$ and $C_0(p(X))$, $R_0(p(X))$ with $p(X) \in \mathbb{C}[X]$ (see Definition 6.2.9) for the hyperbolic and the exceptional cases, respectively, cf. Theorems 6.6.4 to 6.6.9.

- The equivalences which occur among subtypes of the two parabolic singularity types X_9 and J_{10} are much more involved and cannot be written down in terms of constant factors. We describe them as joint graphs of certain functions, cf. Theorems 6.6.1 and 6.6.2.

The results presented in Section 6.6 have *consequences for the algorithmic classification of the unimodal singularities of corank* 2 *over* $\mathbb{R}$. They are indeed intended to be the first step in this direction. Once again, we can distinguish between different cases. Note that the following remarks apply to the classification over $\mathbb{R}$ and therefore only deal with real coordinate transformations and real values of the involved parameters:

- In the exceptional cases, there are no equivalences between different subtypes of the same main type and the value of the parameter is uniquely determined, cf. Theorem 6.6.9.

- For the singularity types J_{10+k}, X_{9+k}, and $\widetilde{Y}_r$, there are no equivalences between different subtypes of the same main type, but the parameter can in some cases change its sign within the same subtype, cf. Theorems 6.6.4, 6.6.5, and 6.6.8.

- For $Y_{r,s}$, there are equivalences even between different subtypes. Therefore the question which real subtype a given singularity of main type $Y_{r,s}$ belongs to is not always well-posed, e.g., a singularity can be both of type $Y_{5,7}^{++}$ and of type $Y_{5,7}^{+-}$. However, the first of the two signs is always uniquely determined. The parameter can change its sign, but its absolute value is uniquely determined, cf. Theorem 6.6.6.

- The structures of the equivalence classes of the two parabolic cases X_9 and J_{10} are the most complicated among all the cases discussed here. There are equivalences between different subtypes and the parameter may change in non-trivial ways. For X_9, the possible values which a given parameter can be transformed to can be expressed as rational functions of this parameter (cf. Theorem 6.6.1), whereas the corresponding functions for J_{10} involve radical expressions (cf. Theorem 6.6.2). Note that there are, however, no equivalences between the subtypes X_9^{++}, X_9^{--} on the one hand and X_9^{+-}, X_9^{-+} on the other hand, i.e. the product of the two signs which occur in the subtypes of X_9 is uniquely determined.

- It is a remarkable result that for any value of $a \in \mathbb{R}$, the normal form of $J_{10}^{-}(a)$ is $\mathbb{R}$-equivalent to the normal form of $J_{10}^{+}(a')$ for some $a' \in \mathbb{R}$ while the converse is not true, cf. Theorem 6.6.2. In other words, the real subtype J_{10}^{-} is redundant whereas J_{10}^{+} is not.

To sum up, the normal forms which are listed in the classifications of the unimodal singularitities over $\mathbb{C}$ and over $\mathbb{R}$ by Arnold et al. [4] cover the whole

space of unimodal singularities, but some of them are equivalent. There are equivalences between normal forms for different values of the parameter and also between different subtypes, to an extend that the subtype J_{10}^{-} is even redundant.

Bibliography

[1] G. Amdahl: Validity of the single processor approach to achieving large-scale computing capabilities. In: *AFIPS Conference Proceedings* 30 (1967), pp. 483–485.

[2] E.A. Arnold: Modular algorithms for computing Gröbner bases. In: *J. Symb. Comp.* 35 (2003), pp. 403–419.

[3] V.I. Arnold: Normal forms of functions in neighbourhoods of degenerate critical points. In: *Russ. Math. Surv.* 29.2 (1974), pp. 10–50.

[4] V.I. Arnold, S.M. Gusein-Zade, and A.N. Varchenko: *Singularities of Differential Maps*. Vol. I. Boston: Birkhäuser, 1985.

[5] M.F. Atiyah and I.G. Macdonald: *Introduction to Commutative Algebra*. Reading, MA: Addison-Wesley, 1969.

[6] J. Böhm, W. Decker, S. Laplagne, and F. Seelisch: Computing integral bases via localization and Hensel lifting. In preparation.

[7] J. Böhm, W. Decker, S. Laplagne, G. Pfister, A. Steenpaß, and S. Steidel: Parallel Algorithms for Normalization. In: *J. Symb. Comp.* 51 (2013), pp. 99–114.

[8] J. Böhm, W. Decker, C. Fieker, and G. Pfister: The use of bad primes in rational reconstruction. To appear. 2013. URL: `http://arxiv.org/abs/1207.1651`.

[9] W. Bosma, J. Cannon, and C. Playoust: The Magma algebra system. I. The user language. In: *J. Symb. Comp.* 24 (1997), pp. 235–265.

[10] W. Decker, G.-M. Greuel, and G. Pfister: Primary Decomposition: Algorithms and Comparisons. In: *Algorithmic Algebra and Number Theory*. Ed. by G.-M. Greuel, B.H. Matzat, and G. Hiss. Heidelberg: Springer, 1998, pp. 187–220.

[11] W. Decker and F.-O. Schreyer: Varieties, Gröbner Bases, and Algebraic Curves. Book in preparation.

[12] W. Decker, G.-M. Greuel, G. Pfister, and T. de Jong: The normalization: a new algorithm, implementation and comparisons. In: *Computational methods for representations of groups and algebras*. Essen: Birkhäuser, 1999.

[13] D. Eisenbud: *Commutative Algebra with a View Toward Algebraic Geometry*. Springer, 1995.

[14] P. Gianni, B. Trager, and G. Zacharias: Gröbner Bases and Primary Decomposition of Polynomial Ideals. In: *J. Symb. Comp.* 6 (1988), pp. 149–167.

[15] H. Grauert and R. Remmert: *Analytische Stellenalgebren*. Springer, 1971.

[16] R.T. Gregory and P. Kornerup: Mapping Integers and Hensel Codes onto Farey Fractions. In: *BIT Numerical Mathematics* 23.1 (1983), pp. 9–20.

[17] G.-M. Greuel, S. Laplagne, and Seelisch. F.: Normalization of rings. In: *J. Symb. Comp.* 45 (2010), pp. 887–901.

[18] G.-M. Greuel, C. Lossen, and E. Shustin: *Introduction to Singularities and Deformations*. Berlin: Springer, 2007.

[19] G.-M. Greuel and G. Pfister: *A* SINGULAR *Introduction to Commutative Algebra*. Second edition. Berlin: Springer, 2008.

[20] J.L. Gustafson: Reevaluating Amdahl's Law. In: *Communications of the ACM* 31.5 (1988), pp. 532–533.

[21] A. Hirano: Construction of plane curves with cusps. In: *Saitama Mathematical Journal* 10 (1992), pp. 21–24.

[22] N. Idrees, G. Pfister, and S. Steidel: Parallelization of Modular Algorithms. In: *J. Symb. Comp.* 46 (2011), pp. 672–684.

[23] D. Jiang and J.P. Singh: Scaling Application Performance on a Cache-coherent Multiprocessor. In: *Proceedings of the 26th Annual International Symposium on Computer Architecture*. IEEE Computer Society, 1999, pp. 305–316.

[24] T. de Jong: An algorithm for computing the integral closure. In: *J. Symb. Comp.* 26 (1998), pp. 273–277.

[25] T. de Jong and G. Pfister: *Local Analytic Geometry*. Vieweg, 2000.

[26] A.R. Karlin, K. Li, M.S. Manasse, and S. Owicki: Empirical Studies of Competitve Spinning for a Shared-memory Multiprocessor. In: *Proceedings of the Thirteenth ACM Symposium on Operating Systems Principles*. ACM Press, 1991, pp. 41–55.

[27] K. Krüger: Klassifikation von Hyperflächensingularitäten. MA thesis. University of Kaiserslautern, 1997. URL: `ftp://www.mathematik.uni-kl.de/pub/Math/Singular/doc/Papers/diplom_krueger.ps.gz`.

[28] R. La Scala and M.E. Stillman: Strategies for Computing Minimal Free Resolutions. In: *J. Symb. Comp.* 26 (1998), pp. 409–431.

[29] D.A. Leonard and R. Pellikaan: Integral closures and weight functions over finite fields. In: *Finite Fields and their Applications* 9 (2003), pp. 479–504.

[30] M.S. Marais and A. Steenpaß: The Classification of Real Singularities Using SINGULAR. Part I: Splitting Lemma and Simple Singularities. Accepted for publication in: *J. Symb. Comp.* 2013.

[31] M.S. Marais and A. Steenpaß: The Classification of Real Singularities Using SINGULAR. Part II: The Structure of the Equivalence Classes of the Unimodal Singularities. Submitted. 2014.

[32] G.E. Moore: Cramming more components onto integrated circuits. In: *Electronics* 38.8 (1965).

[33] F.-O. Schreyer: A Standard Basis Approach to Syzygies of Canonical Curves. In: *J. Reine Angew. Math.* 421 (1991), pp. 83–123.

[34] F.-O. Schreyer: Die Berechnung von Syzygien mit dem verallgemeinerten Weierstrasschen Divisionssatz. Universität Hamburg. Diplomarbeit. 1980.

[35] T. Shimoyama and K. Yokoyama: Localization and Primary Decomposition of Polynomial Ideals. In: *J. Symb. Comp.* 22 (1996), pp. 247–277.

[36] D. Siersma: Classification and Deformation of Singularities. PhD thesis. Univ. of Amsterdam, 1974. URL: `http://www.staff.science.uu.nl/~siers101/ArticleDownloads/DissertationSiersma.pdf`.

[37] A. Singh and I. Swanson: An algorithm for computing the integral closure. 2009. URL: `http://arxiv.org/abs/0901.0871`.

[38] I. Swanson and C. Huneke: *Integral closure of ideals, rings, and modules*. Cambridge University Press, 2006.

Software Systems

[39] W. Decker, G.-M. Greuel, G. Pfister, and H. Schönemann: SINGULAR 3-1-6 – A computer algebra system for polynomial computations. 2012. URL: `http://www.singular.uni-kl.de`.

[40] *GAP – Groups, Algorithms, and Programming, Version 4.7.5*. The GAP Group. 2014. URL: `http://www.gap-system.org`.

[41] D.R. Grayson and M.E. Stillman: MACAULAY2 – A software system for research in algebraic geometry. 2013. URL: `http://www.math.uiuc.edu/Macaulay2/`.

Singular Libraries

[42] J. Böhm, W. Decker, S. Laplagne, G. Pfister, A. Steenpaß, and S. Steidel: `locnormal.lib`. A Singular 3-1-6 library for the normalization of affine domains using local methods. 2013.

[43] J. Böhm, W. Decker, S. Laplagne, G. Pfister, A. Steenpaß, and S. Steidel: `modnormal.lib`. A Singular 3-1-6 library for the normalization of affine domains using modular methods. 2013.

[44] W. Decker, S. Laplagne, G. Pfister, and H. Schönemann: `primdec.lib`. A Singular 3-1-6 library for primary decomposition and radicals of ideals. 2012.

[45] A. Hashemi, G. Pfister, H. Schönemann, A. Steenpaß, and S. Steidel: `modstd.lib`. A Singular 3-1-6 library for computing Gröbner bases of ideals using modular methods. 2014.

[46] N. Idrees, G. Pfister, A. Steenpaß, and S. Steidel: `assprimeszerodim.lib`. A Singular 3-1-6 library for computing the associated primes of a zero-dimensional ideal. 2014.

[47] K. Krüger: `classify.lib`. A Singular 3-1-6 library for classifying isolated hypersurface singularities w.r.t. right equivalence. 2012.

[48] M.S. Marais and A. Steenpaß: `realclassify.lib`. A Singular 3-1-6 library for classifying isolated hypersurface singularities over the reals w.r.t. right equivalence. 2012.

[49] A. Montes and H. Schönemann: `grobcov.lib`. A Singular 3-1-6 library for Gröbner covers of parametric ideals. 2012.

[50] G. Pfister and A. Steenpaß: `primdec_parallel.lib`. A Singular 3-1-6 library for parallel primary decomposition. 2014.

[51] A. Steenpaß: `modquotient.lib`. A Singular 3-1-6 library for computing ideal quotients and saturations using modular methods. 2014.

[52] A. Steenpaß: `modular.lib`. A Singular 3-1-6 library for modular computations. 2014.

[53] A. Steenpaß: `parallel.lib`. A Singular 3-1-6 library providing an abstraction layer for parallel skeletons. 2014.

[54] A. Steenpaß: `resources.lib`. A Singular 3-1-6 library for managing computational resources. 2014.

[55] A. Steenpaß: `tasks.lib`. A Singular 3-1-6 library providing a parallel framework based on tasks. 2014.

[56] E. Tobis: `rootsur.lib`. A Singular 3-1-6 library for counting the number of real roots of a univariate polynomial. 2012.

Wissenschaftlicher Werdegang

06/2004	Abitur am Gymnasium Hammonense
10/2004 – 09/2009	Studium der Mathematik und Philosophie, HU Berlin
07/2009	Diplom in Mathematik, HU Berlin
seit 01/2010	Doktorand bei Prof. Dr. Wolfram Decker, TU Kaiserslautern

Curriculum Vitae

06/2004	Abitur at the Gymnasium Hammonense
10/2004 – 09/2009	Studies of mathematics und philosophy, HU Berlin
07/2009	Diplom in mathematics, HU Berlin
since 01/2010	PhD studies with Prof. Dr. Wolfram Decker, TU Kaiserslautern