Universidad de Buenos Aires

Facultad de Ciencias Exactas y Naturales

Departamento de Computación

Complexity as Quality Attribute in Software Design

Tesis de Licenciatura en Ciencias de la Computación

Alumno : Andrés Rojas Paredes [a] L.U.: 574/04*

Director : Joos Heintz [b]

[a] andresrz1@gmail.com [b] joos@dc.uba.ar

June 29, 2011

*Research partially supported by the grant "Beca de Ayuda Económica Dr. Manuel Sadosky"

Resumen extendido

La arquitectura de software nació como concepto a fines de los años setenta y durante los años ochenta con el objetivo de refinar el arte del diseño, mediante especificaciones de grandes paquetes de programas. Desde el principio estuvo en el centro de la atención una interacción con un tejido social, frecuentemente llamado "contrato" o "proyecto". Para tal fin se desarrolló un aparato conceptual con el fin de captar de manera uniforme una gran variedad de situaciones muy diferentes entre sí. Hubo también algunos intentos de modelización matemática, siendo la más destacable y avanzada la lógica de Hoare del asserted programming. Aunque la lógica de Hoare es demasiado restrictiva para su aplicación práctica, tuvo una gran repercusión en la enseñanza de la informática porque formaliza dos aspectos fundamentales de la ingeniería de software:

(1) Un programa no es simplemente un algoritmo modelizado mediante un cierto tipo de máquina, sino contiene también una semántica definida en términos de especificaciones y una demostración de su correctitud.

(2) La demostración de la correctitud de un programa no tiene la forma de una demostración matemática cualquiera, sino esta sometida a una cierta estandardización.

El aspecto (2) representa una piedra fundamental para la visión arquitectónica de la programación por dos razones. Una es la indecidibilidad de la equivalencia de programas en lenguajes mínimamente recursivos y la otra es el requerimiento práctico que no debería ser tarea del programador demostrar la conjetura de Riemann antes de meterse en la implementación de un algoritmo de teoría de números.

El ejemplo de la lógica de Hoare nos sirve en esta exposición como testigo para la afirmación de que no todos los aspectos de la ingeniería de software tienen carácter subjetivo o social. Existen también aspectos objetivizables e independientes del individuo.

La meta de esta tesis es elaborar algunos de estos aspectos objetivizables y estudiarlos mediante un método científico. Esto requiere una modelización matemática y la justificación informática de la misma. Tal modelización permite la aplicación de herramientas matemáticas para sacar conclusiones matemáticas que a su vez deben ser retraducibles al contexto original informático. La problemática de este procedimiento es la siguiente: la matemática es la ciencia de las trivialidades y tautologías y por lo tanto sus herramientas son inespecíficas. No es de esperar que una modelización exacta de un contexto informático permita sacar grandes conclusiones matemáticas que a su vez conducen a sorprendentes y desconocidas inferencias informáticas. Por otro lado una modelización grosera matemática permitiría solamente conclusiones muy generales e inespecíficas para la informática. Por esta razón el trabajo propuesto se concentra en el "tuning" adecuado de la modelización y no en el aparato matemático (que, sea dicho aparte, es de ninguna manera trivial) para sacar conclusiones informáticas.

La ingeniería de software de los años setenta y ochenta tuvo que luchar con estas dificultades. La generalidad del planteo no permitió formular y demostrar conclusiones rigurosas. Por esta razón este trabajo limita la aplicación de conceptos de la ingeniería de

software al campo más reducido de la computación científica y recurre a una restricción adicional sobre la arquitectura fijando un nivel de abstracción predeterminado.

Dentro de la computación científica el trabajo se concentra en problemas y algoritmos de eliminación e interpolación polinomiales y el nivel de abstracción predeterminado es el modelo algebraico de complejidad en cuerpos como $\mathbb{R}$ y $\mathbb{C}$ con las operaciones aritméticas (adición, substracción, multiplicación y división) implementadas a costo unidad. La idea es que los programas considerados admitan polimorfismo y puedan ser ejecutados tanto en entorno numérico con precisión finita como en un entorno simbólico con precisión infinita.

Este nivel de abstracción impone sus reglas de juego. Divisiones en algoritmos del modelo algebraico conducen a branchings, lo que no es apropiado para una interpretación numérica. Sin embargo, ciertas divisiones de tipo $0/0$ pueden ser remplazadas por limites que se adaptan mucho mejor a la interpretación numérica (un ejemplo es la regla de L´Hôpital). Esto conduce a un modelo donde se restringen las divisiones a los casos donde ellas representan límites. Para las tareas algorítmicas consideradas en este trabajo esta restricción alcanza y representa adecuadamente el concepto sintáctico de branching–freeness. Sin embargo, detrás de esto se esconde un concepto más profundo: los problemas que se consideran en este trabajo admiten degeneraciones a problemas límites. Por lo tanto es natural pedir que los algoritmos que resuelven estos problemas capten estas degeneraciones. En teoría de interpolación esto se llama informalmente "coalescencia". La noción de coalescencia permite una modelización nítida, transparente y geométrica que refleja las divisiones admitidas en el modelo bajo consideración. Esta modelización se encuentra realizada por la noción precisa y matemática de la "robustez geométrica" de una aplicación racional. La robustez geométrica representa un atributo de calidad dicotómico (quality attribute) de un software, y tiene su análogo discreto en el aprendizaje de patrones (ver Sección 5.3.1). Uno de los objetivos principales del trabajo es un análisis en profundidad de propiedades de algoritmos como la "coalescencia" o el "branching–freeness" bajo el aspecto de atributo de calidad.

Las herramientas matemáticas utilizadas permiten ahora conclusiones sorprendentes. Otro atributo de calidad, esta vez cuantitativo, de un algoritmo o programa es su complejidad contada en cantidad de operaciones aritméticas. Las herramientas matemáticas mencionadas permiten ahora establecer un trade–off entre el atributo de calidad dicotómico de la robustez geométrica y el atributo de calidad cuantitativo de la complejidad: algoritmos generales ("universales") y geométricamente robustos, que resuelven ciertos problemas naturales de eliminación e interpolación, tienen necesariamente una complejidad que es exponencial en el tamaño de la representación sintáctica de la entrada, y esto con mínimos requerimientos en la arquitectura y sin haber detallado las estructuras o tipos de datos que iban a intervenir.

La técnica para demostrar tal resultado representa en si misma una innovación en el campo de la geometría algebraica. Partes del método motivan en este campo preguntas totalmente nuevas como la cuestión de la desingularización de una variedad uniracional dada como imagen de un morfismo (y no por ecuaciones) o cuantos blow ups requiere la transformación de una aplicación racional dada en una polinomial? La sorpresa es que la cuestión de la desingularización y el análisis geométrico de las aplicaciones racionales está íntimamente relacionada con la arquitectura de software en cálculo científico. En ambos campos surgen las mismas preguntas y se aportan las mismas respuestas. La Sección 6.5.5 del trabajo será dedicada a la exposición de esta sorprendente relación entre dos campos

que a primera vista están completamente alejados.

Necesariamente el trabajo debe contener una parte introductoria donde se explican las herramientas matemáticas. En esta parte las demostraciones estarán remplazadas por referencias a publicaciones recientes. Sin embargo, una parte (más pequeña) de las herramientas es nueva y aparecerá con demostraciones en un apéndice (ver Appendix A).

La parte central y más innovativa del trabajo es la Sección 6 donde se introduce un modelo matemático para el análisis general de tareas computacionales del cálculo científico en el contexto de los algoritmos seminuméricos (que usan como estructura de datos básica los circuitos aritméticos). La atención será puesta sobre el caso particular de tareas de eliminación en geometría algebraica efectiva. El aporte principal de esta sección consiste en la fundamentación, motivación y justificación de este modelo matemático mediante criterios que provienen de la ingeniería de software. Se demuestra que las características matemáticas esenciales son consecuencias directas de los requerimientos funcionales o no–funcionales de software que se encuentran altamente aceptados por la comunidad de ingeniería de software y que se pueden aplicar alternativamente. El primero de estos requerimientos es funcional, el segundo es no–funcional. La conclusión matemática de esta modelización en la forma dada en esta tesis, es nueva y representa un resultado informático sorprendente: en la Sección 6.5.1 se exhibe una familia infinita de problemas básicos de eliminación geométrica tal que cualquier algoritmo de nuestro modelo que calcula los polinomios de eliminación asociados a partir de la entrada dada requiere un tiempo exponencial. Sin embargo, no se responde la pregunta si los polinomios de eliminación, con el algoritmo que sea, fueran difíciles de evaluar. Pretendiendo responder a esta última pregunta, todos los intentos anteriores de atacar el problema resuelto en esta tesis, fracasaron. Una variante decisional de este problema es la conjetura $P_{\mathbb{C}} \neq NP_{\mathbb{C}}$ en el modelo de las máquinas de Turing continuas de Blum, Shub y Smale [BSS89], el modelo BSS. Como nuestro resultado no se refiere a un contexto decisional, sino al cálculo de un objeto matemático, un polinomio de eliminación, no demostramos esta conjetura, ni siquiera en un modelo de recursos restringidos de máquinas de Turing BSS.

Finalmente recalcamos que el resultado principal de la Sección 6 puede ser interpretado como un trade–off entre un atributo de calidad cualitativo (el requerimiento no–funcional de la robustez en combinación con otro requerimiento, llamado isoparametría) y uno cuantitativo (una clase de complejidad algorítmica). En este sentido se trata de una certificación matemática de un trade–off entre dos atributos de calidad. No conocemos otro caso en la literatura.

Agradecimientos

Umberto Eco recomienda en su libro [Eco06] no agradecer al director de la tesis. No haciendo caso a esta regla deseo expresar mi más profundo agradecimiento a mi director Joos Heintz por su gran dedicación para concluir este trabajo y por su guía y acompañamiento en el camino de esta investigación. Sin él este trabajo no hubiera sido posible. Agradezco de la misma forma a mi mamá Felicidad Susana Paredes por su ayuda, sacrificio y continuo aliento para terminar esta tesis.

Deseo agradecer también a la señorita Hvara Azul Ocar por sus ideas, consejos y ayuda en la redacción de más de una sección de este trabajo. Gracias a Diego de Estrada por sus consejos de latex que me ahorraron horas y horas de compilación. Y gracias a mis compañeros que durante la carrera estuvieron ahi en el "horno" junto conmigo.

Andrés Rojas Paredes

Complexity as Quality Attribute in Software Design

Abstract

We introduce a software architecture based computation model for Scientific Computing. Its relevance becomes illustrated by the precise formulation and solution of a more than thirty years open complexity problem in Effective Algebraic Geometry (elimination theory).

Contents

List of Tables

List of Figures

List of Definitions

1 Introduction

In this thesis we hike[1] through three scientific fields, namely Algebraic Geometry, Computational Complexity Theory and Software Engineering[2]. More specifically, we combine Complexity Theory and Software Engineering in the aim to analyse trade–offs between computational complexity and certain other quality attributes of algorithms in Scientific Computing, in particular in computational Algebraic Geometry. The main tool to achieve this goal belongs to pure mathematics. Figure 1 illustrates the topography of the scientific region where our hike takes place.

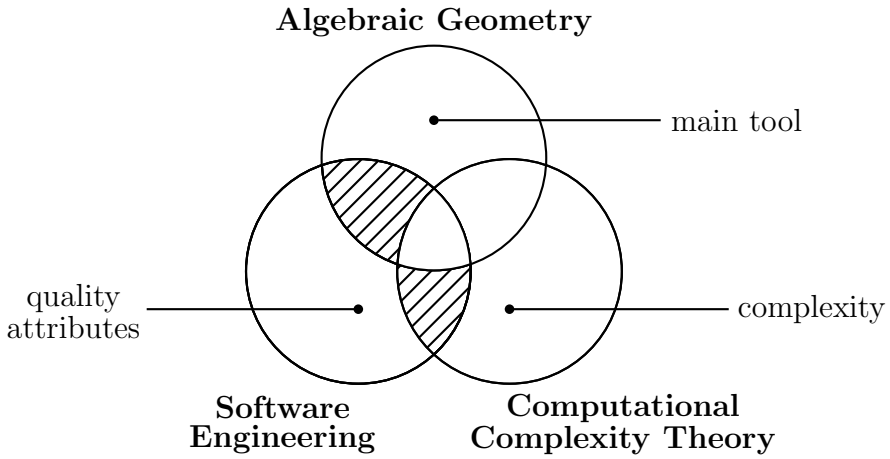


Figure 1: Three scientific fields.

The shadowy part of Figure 1 represents the area of Software Engineering where our hike ends. We are now going to explain our itinerary. We start with our subject matter in the next Section 1.1, continue with our main goal in Section 1.2 and describe finally in Section 1.3 the organization of this thesis.

1.1 The subject matter

Already known results and notions from the field of Algebraic Geometry do not suffice to state and prove the main conclusion of this thesis in Section 6.5.1. Therefore, we have to undertake some new and genuine work, which is of its own interest in the context of rational maps and singularities. The field of Algebraic Geometry does not only provide our main tool for the intended trade–off analysis between quality attributes, but also our

[1] To go for a long walk in the land, which is left in its natural condition.

[2] Algebraic Geometry is a branch of Mathematics concerned with the study of algebraic varieties defined by polynomial equations systems that deals with the resources necessary to solve computational problems, see e.g. [Har92]. Computational Complexity Theory is a research area of Computer Science, see e.g. [RW04]. Software Engineering is the field of Computer Science that deals with the building of software systems, see e.g. [GJM03].

main application domain. All computational problems we consider in this thesis have a background, which may be expressed as special instances of the general task of geometric elimination consisting of the effective elimination of a fixed number of quantifier blocks in prenex formulas of the first order theory of algebraically closed fields of characteristic zero. In such formulas quantifiers may be eliminated by means of known algorithms whose sequential time complexity in the bit model depends exponentially on the number of occurring variables, polynomially on the number and degree, and polylogarithmically on the bit length of the coefficients of the polynomials involved.

In this context polynomials and integers are considered as objects belonging to abstract data types, which become implemented by their coefficients and by their bit representation, respectively. This distinction between abstract data types and their implementations is important because the mentioned upper bound for the complexity of geometric elimination, which is based on the coefficient and bit implementation of polynomials and integers, turns out to be asymptotically optimal. Therefore, improvements and refinements of this bound can only be expected if we take into account non–standard implementations of the abstract data types polynomial and integer, e.g. by means of arithmetic circuits.

At this point we become aware about the interaction between implementation and complexity in geometric elimination. Since this interaction depends on the implementation of a previously fixed abstract data type, we arrive finally to an architectural point of view which involves not just the components of a geometric elimination algorithm, but also the structure of these components and the relationships between them. Thus, we arrive to a Software Engineering perspective.

In this thesis we shall limit the application of concepts of Software Engineering to the area of elimination theory in elementary Algebraic Geometry and to Interpolation Theory. As far as software architecture is concerned, we restrict our attention to a predetermined level of abstraction. The level of abstraction we choose is the algebraic complexity model over $\mathbb{C}$ with the arithmetic operations and the equality check implemented at unit costs.

This architectural view involves not only the implementation of the abstract data type polynomial (or rational function) in the algebraic complexity model over $\mathbb{C}$, but also the fulfillment of certain quality attributes, also called non–functional requirements. These quality attributes appear by means of a limit behaviour of the computational problems considered in this thesis. Thus it is natural to require that the algorithms that solve these problems should be able to capture such a limit behaviour. We refer to this situation using an informal concept of Interpolation Theory, namely *coalescence*[3].

We shall formalize the concept of coalescence by means of a mathematical model, introducing the notion of geometrical robustness of a rational map[4]. In other words, geometrical robustness will formalize the dichotomic quality attribute of coalescence and this will allow us to draw interesting conclusions.

Another quality attribute, this time a quantitative one, is the computational complexity of an algorithm, measured by the number of arithmetic operations executed in run time. Our mathematical tools will allow to certify a trade–off between the dichotomic

[3]An introduction to the concept of coalescence in the context of Interpolation Theory is given in Section 4.2.3. An interpretation of coalescence as quality attribute and its similarity with another quality attribute, namely performance, can be found in Section 5.

[4]The definition of geometrical robustness can be found in Section 2.3 and the rôle of this concept in the formulation of trade–offs is explained in Section 4.6.

quality attribute of geometrical robustness and the quantitative quality attribute of computational complexity:
geometrically robust algorithms that solve certain natural problems of geometric elimination and interpolation, have necessarily a computational complexity that is exponential in the size of the syntactic representation of the input.

1.2 Motivation

Our main goal is to establish a mathematical computation model that allows a kind of trade–off considerations that are not possible in the actual state of the art in Software Engineering. The model we are going to introduce admits certain formal mathematical conclusions which can be reinterpreted in terms of Computer Science (see Figure 2).

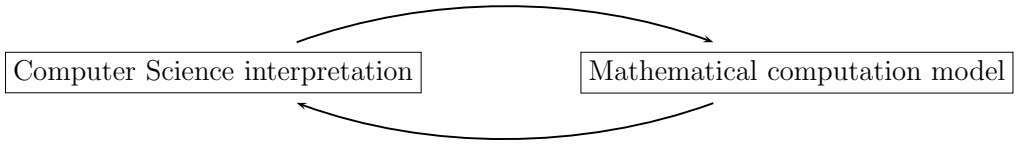


Figure 2: Interpretation and reinterpretation

Our mathematical model has two aspects. The first aspect concerns the complexity issues of elimination in elementary Algebraic Geometry with special emphasis on lower bounds. The second aspect is related to the first and concerns trade–offs between quality attributes in scientific computing.

There is a practical aim behind our theoretical research. Consider the process in software design where a software architecture is developed in order to solve a certain computational problem which is supposed to be given by a formal specification. Assume also that one of the non–functional requirements of the software design project consists of a restriction on the run time computational complexity of the program which is going to be developed. The software engineer may wish to know at an early stage of the design process whether the decisions already taken by him will not violate at the end the non–functional requirements which he has to satisfy. Our practical aim is to provide the software engineer with an efficient tool which allows him to answer the question whether his software design process is entering at some moment in conflict with the given complexity requirement. If this is the case, the software engineer will be able to change at this early stage his design and may look for an alternative software architecture.

An example of this way of thinking is the implementation of the mathematical concept of a finite subset of a possibly infinite ordered ambient set, e.g. the natural numbers $\mathbb{N}$. Suppose that our task is to implement the abstract concept of finite sets and that we have chosen an implementation by means of arrays. Figure 3 illustrates this simple software architecture.

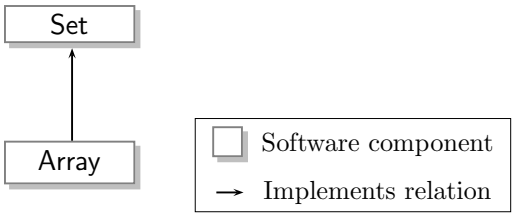


Figure 3: Simple software architecture

Suppose now that one has to satisfy the requirement that membership to a finite set S of cardinality n can be decided using only $O(\log n)$ comparisons with the elements of any given array implementing S. If the set S is implemented by an unordered array, we will be unable to satisfy our complexity requirement. So we are forced to think in alternative implementations of the abstract concept of a finite set, e.g. by ordered arrays, special trees or any other data type which is well suited for our task.

This example represents a case where it may be impossible to satisfy a given complexity requirement by means of a previously fixed software architecture. Such impossibility results are usually difficult to infer when the number of components of the system under consideration is large or when the predicate to decide or the function to compute becomes more sophisticated like in polynomial equation solving.

This leads us to the idea to fix in advance only a small selection of architectural features, e.g. the abstraction levels or part of the language of our system (not the algorithms and routines themselves). Such a computation model should allow conclusions about the satisfiability of certain functional and non–functional requirements.

1.3 Organization of the thesis

This thesis is composed by seven sections and two appendices, including this introductory section. In Section 2 we recall definitions and notations from Algebraic Geometry and develop the main tool for our research on quality attribute trade–offs. In Section 3 we recall definitions and concepts from Software Engineering, in particular from software architecture and the theory of quality attributes.

The exposition of our own work starts with Section 4 where we combine the concepts introduced in Sections 2 and 3. In this section we discuss Hermite–Lagrange interpolation and its computational issues. Following [GHMS11] we carry out a strict distinction between the concepts of interpolation problem and interpolation algorithm. This distinction leads us to the formulation of an interpolation model in terms of software architecture which includes quality attribute restrictions. We present the mentioned informal concept of coalescence as a quality attribute of interpolation algorithms.

In Section 4.5 we illustrate our interpolation model by a couple of well known examples from univariate Hermite–Lagrange interpolation. In Section 4.6 we continue explaining how our computation model can be used in order to certify quality attribute trade–offs. In Section 4.7 we present some open questions about our interpolation model, e.g. some questions concerned with the rôle of memory during the execution of an interpolation algorithm. In Section 4.8 we establish a link between the concepts of polymorphism and software architecture of interpolation algorithms.

We finish Section 4 with a dictionary. The mathematical and complexity theoretical aspects of this thesis were introduced in [GHMS11], where the authors used their own terminology. Here we present the results of [GHMS11] from a different point of view, the difference consisting mainly of the language which we adopt in this thesis. We standardize the terminology of [GHMS11] by means of concepts taken from object oriented programming in the spirit of B. Meyer [Mey00]. This translation of concepts is carried out by means of a dictionary which we present and justify in Section 4.9.

In Section 5 we discuss another, quality attribute, called *branching parsimoniousness*, which keeps some similarity with coalescence. These quality attributes become then analysed from the point of view of scenarios (see [BCK03]). We exhibit properties and examples of quality attributes in Software Engineering which may be compared to coalescence and branching parsimoniousness.

In Section 6 we present a mathematical computation model which captures all known algorithms in effective Algebraic Geometry and almost all in symbolic or seminumeric scientific computing. The model becomes justified by arguments coming from Software Engineering and in Section 6.5.1 we exhibit an infinite family of simple elimination problems whose computational solution in our model requires algorithms of exponential time complexity. The model is based on the representation of polynomials by arithmetic circuits.

In Section 6.5.5 our mathematical tools are discussed from the point of view of (qualitative) Algebraic Geometry, especially desingularization techniques (blow ups). Section 6.5.6 is devoted to the comparison of our point of view with other computation models.

Section 7 is devoted to conclusions, future work and open questions.

We finish this thesis with Appendix A where we present an original result which is of its own interest for the discussion of rational functions and maps in Algebraic Geometry, and Appendix B.1 contains the correctness proofs of the interpolation algorithms discussed in Section 4.5.

2 Concepts and tools from Algebraic Geometry

In this section we use freely standard notions and notations from Commutative Algebra and Algebraic Geometry. These can be found for example in [Lan93], [ZS60], [Kun85] and [Sha94]. In Sections 2.2 and 2.3 we introduce the notions and definitions which constitute our fundamental tool for the modeling of elimination problems and algorithms. All these notions and their definitions are taken from [GHMS11].

2.1 Basic notations and definitions

2.1.1 Basic notations

For any $n \in \mathbb{N}$, we denote by $\mathbb{A}^n := \mathbb{A}^n(\mathbb{C})$ the n–dimensional affine space $\mathbb{C}^n$ equipped with its respective Zariski and Euclidean topologies over $\mathbb{C}$. In algebraic geometry, the Euclidean topology of $\mathbb{A}^n$ is also called the *strong topology*. We shall use this terminology only exceptionally.

Let $X_1, \ldots, X_n$ be indeterminates over $\mathbb{C}$ and let $X := (X_1, \ldots, X_n)$. We denote by $\mathbb{C}[X]$ the ring of polynomials in the variables X with complex coefficients.

We denote by Π_D the $\mathbb{C}$–vector space of polynomials of degree at most D of the polynomial ring $\mathbb{C}[X]$.

Let V be a closed affine subvariety of $\mathbb{A}^n$, that is, the set of common zeros in $\mathbb{A}^n$ of a finite set of polynomials belonging to $\mathbb{C}[X]$. As usual, we write $\dim V$ for the dimension of the variety V. Let $C_1, \ldots, C_s$ be the irreducible components of V. For $1 \leq j \leq s$ we define the degree of C_j as the number of points which arise when we intersect C_j with $\dim C_j$ many generic affine hyperplanes of $\mathbb{A}^n$. Observe that this number is a well determined positive integer which we denote by $\deg C_j$. The *(geometric) degree* $\deg V$ of V is defined by $\deg V := \sum_{1 \leq j \leq s} \deg C_j$. This notion of degree satisfies the so called Bezout Inequality. Namely, for another closed affine subvariety W of $\mathbb{A}^n$ we have

$$\deg V \cap W \leq \deg V \cdot \deg W.$$

For details we refer to [Hei83], where the notion of geometric degree was introduced and the Bezout Inequality was proved for the first time (other references are [Ful84] and [Vog84]).

For $f_1, \ldots, f_s, g \in \mathbb{C}[X]$ we shall use the notation $\{f_1 = 0, \ldots, f_s = 0\}$ in order to denote the closed affine subvariety V of $\mathbb{A}^n$ defined by $f_1, \ldots, f_s$ and the notation $\{f_1 = 0, \ldots, f_s = 0, g \neq 0\}$ in order to denote the Zariski open subset V_g of V defined by the intersection of V with the complement of $\{g = 0\}$. Observe that V_g is a locally closed affine subvariety of $\mathbb{A}^n$ whose coordinate ring is the localization $\mathbb{C}[V]_g$ of $\mathbb{C}[V]$.

We denote by $I(V) := \{f \in \mathbb{C}[X] : f(x) = 0 \text{ for any } x \in V\}$ the ideal of definition of V in $\mathbb{C}[X]$ and by $\mathbb{C}[V] := \{\varphi : V \to \mathbb{C} \text{ ; there exists } f \in \mathbb{C}[X] \text{ with } \varphi(x) = f(x) \text{ for any } x \in V\}$ its coordinate ring. Observe that $\mathbb{C}[V]$ is isomorphic to the quotient $\mathbb{C}$–algebra $\mathbb{C}[V] := \mathbb{C}[X]/I(V)$. If V is irreducible, then $\mathbb{C}[V]$ is zero–divisor free and we denote by $\mathbb{C}(V)$ the field formed by the rational functions of V with maximal domain which is called the rational function field of V. Observe that $\mathbb{C}(V)$ is isomorphic to the fraction field of the integral domain $\mathbb{C}[V]$.

In the general situation where V is an arbitrary closed affine subvariety of $\mathbb{A}^n$, the notion of a rational function of V has also a precise meaning. The only point to underline

is that the domain, say U, of a rational function of V has to be a maximal Zariski open and dense subset of V to which the given rational function can be extended. In particular, U has a nonempty intersection with any of the irreducible components of V.

We denote by $\mathbb{C}(V)$ the $\mathbb{C}$–algebra formed by the rational functions of V. In algebraic terms, $\mathbb{C}(V)$ is the total quotient ring of $\mathbb{C}[V]$ and is isomorphic to the direct product of the rational function fields of the irreducible components of V.

Table 1 summarizes these notations.

Notation	**Meaning**
$\mathbb{A}^n$	the n–dimensional affine space $\mathbb{C}^n$
$\mathbb{C}[X]$	the ring of polynomials in the variable X with complex coefficients
Π_D	the $\mathbb{C}$–vector space of polynomials of degree at most D of $\mathbb{C}[x]$
$\dim V$	the dimension of the variety V
$\deg V$	the (geometric) degree of the variety V
$\{f_1 = 0, \ldots, f_s = 0\}$	the closed affine subvariety V of $\mathbb{A}^n$ defined by $f_1, \ldots, f_s \in \mathbb{C}[X]$
$I(V)$	the ideal of definition of V in $\mathbb{C}[X]$
$\mathbb{C}[V]$	the coordinate ring of V in $\mathbb{C}[X]$
$\mathbb{C}(V)$	the $\mathbb{C}$–algebra formed by the rational functions of V

Table 1: Basic notations

2.1.2 Basic definitions

Let be given a partial map $\phi : V \dashrightarrow W$, where V and W are closed subvarieties of some affine spaces $\mathbb{A}^n$ and $\mathbb{A}^m$, and let $\phi_1, \ldots, \phi_m$ be the components of ϕ. With these notations we have the following definitions which can be found in [GHMS11]:

Definition 1 (Polynomial map) *The map ϕ is called a morphism of affine varieties or just polynomial map if the complex valued functions $\phi_1, \ldots, \phi_m$ belong to $\mathbb{C}[V]$. Thus, in particular, ϕ is a total map.*

Definition 2 (Rational map) *We call ϕ a rational map of V to W, if the domain U of ϕ is a Zariski open and dense subset of V and $\phi_1, \ldots, \phi_m$ are the restrictions of suitable rational functions of V to U.*

Observe that our definition of a rational map differs from the usual one in Algebraic Geometry, since we do not require that the domain U of ϕ is maximal. Hence, in the case $m := 1$, our concepts of rational function and rational map do not coincide.

2.1.3 Constructible sets and constructible maps

Let $\mathcal{M}$ be a subset of some affine space $\mathbb{A}^n$ and, for a given nonnegative integer m, let $\phi : \mathcal{M} \dashrightarrow \mathbb{A}^m$ be a partial map.

Definition 3 (Constructible set) *We call the set* $\mathcal{M}$ constructible *if* $\mathcal{M}$ *is definable by a Boolean combination of polynomial equations.*

A basic fact we shall use in the sequel is that if $\mathcal{M}$ is constructible, then its Zariski closure is equal to its Euclidean closure (see, e.g. [Mum88], Chapter I, §10, Corollary 1). In the same vein we have the following definition.

Definition 4 (Constructible map) *We call the partial map* ϕ constructible *if the graph of* ϕ *is constructible as a subset of the affine space* $\mathbb{A}^n \times \mathbb{A}^m$.

We say that ϕ is *polynomial* if ϕ is the restriction of a morphism of affine varieties $\mathbb{A}^n \to \mathbb{A}^m$ to a constructible subset $\mathcal{M}$ of $\mathbb{A}^n$ and hence a total map from $\mathcal{M}$ to $\mathbb{A}^m$. Furthermore we call ϕ a *rational* map of $\mathcal{M}$ if the domain U of ϕ is contained in $\mathcal{M}$ and ϕ is the restriction to $\mathcal{M}$ of a rational map of the Zariski closure $\overline{\mathcal{M}}$ of $\mathcal{M}$. In this case U is a Zariski open and dense subset of $\mathcal{M}$.

Since the elementary, i.e. first order theory of algebraically closed fields with constants in $\mathbb{C}$ admits quantifier elimination, constructibility means just elementary definability. In particular, ϕ constructible implies that the domain and the image of ϕ are constructible subsets of $\mathbb{A}^n$ and $\mathbb{A}^m$, respectively.

A useful fact concerning constructible maps we are going to use in the sequel is the following result (see, e.g. [Mar02], Proposition 3.2.14).

Lemma 1 *Let* $\mathcal{M}$ *be a constructible subset of* $\mathbb{A}^n$ *and let* $\phi : \mathcal{M} \dashrightarrow \mathbb{A}^m$ *be a partial map. Then* ϕ *is constructible if and only if there exists a partition of its domain in finitely many constructible subsets, say* $\mathcal{M}_1, \ldots, \mathcal{M}_s$, *such that for any* $1 \leq k \leq s$ *the restriction of* ϕ *to* $\mathcal{M}_k$ *is a rational map of* $\mathcal{M}_k$ *which is defined at any point of* $\mathcal{M}_k$.

In particular, if $\phi : \mathcal{M} \to \mathbb{A}^m$ *is a total constructible map, then there exists a Zariski open and dense subset* U *of* $\mathcal{M}$ *such that the restriction* $\phi|_U$ *of* ϕ *to* U *is a rational map.*

2.2 Weakly continuous, strongly continuous, topologically robust and hereditary maps

We are now going to introduce the notions of a weakly continuous, a strongly continuous, a topologically robust, a geometrically robust and a hereditary map of the constructible set $\mathcal{M}$. These five notions will constitute our fundamental tool for the modeling of elimination problems and algorithms.

Definition 5 (Conditions and notions) *Let* $\mathcal{M}$ *be a constructible subset of* $\mathbb{A}^n$ *and let* $\phi : \mathcal{M} \to \mathbb{A}^m$ *be a (total) constructible map. We consider the following four conditions:*

(i) there exists a Zariski open and dense subset U *of* $\mathcal{M}$ *such that the restriction* $\phi|_U$ *of* ϕ *to* U *is a rational map of* $\mathcal{M}$ *and the graph of* ϕ *is contained in the Zariski closure of the graph of* $\phi|_U$ *in* $\mathcal{M} \times \mathbb{A}^m$*;*

(ii) ϕ *is continuous with respect to the Euclidean, i.e. strong, topologies of* $\mathcal{M}$ *and* $\mathbb{A}^m$*;*

(iii) for any sequence $(x_k)_{k\in\mathbb{N}}$ *of points of* $\mathcal{M}$ *which converges in the Euclidean topology to a point of* $\mathcal{M}$*, the sequence* $(\phi(x_k))_{k\in\mathbb{N}}$ *is bounded;*

(iv) for any constructible subset $\mathcal{N}$ *of* $\mathcal{M}$ *the restriction* $\phi|_{\mathcal{N}} : \mathcal{N} \to \mathbb{A}^m$ *is an extension of a rational map of* $\mathcal{N}$ *and the graph of* $\phi|_{\mathcal{N}}$ *is contained in the Zariski closure of this rational map in* $\mathcal{N} \times \mathbb{A}^m$*.*

We call the map ϕ

- **weakly continuous** *if* ϕ *satisfies condition* (i)*,*
- **strongly continuous** *if* ϕ *satisfies condition* (ii)*,*
- **topologically robust** *if* ϕ *satisfies conditions* (i) *and* (iii)*,*
- **hereditary** *if* ϕ *satisfies condition* (iv)*.*

In all these cases we shall refer to $\mathcal{M}$ as the domain of definition of ϕ or we shall say that ϕ is defined on $\mathcal{M}$.

Remark 2 *Let* $\phi : \mathcal{M} \to \mathbb{A}^m$ *be a weakly continuous total constructible map. Then* ϕ *is topologically robust if and only if there exists a Zariski open and dense subset* U *of* $\mathcal{M}$ *such that the restriction* $\phi|_U$ *of* ϕ *to* U *is a rational map of* $\mathcal{M}$ *and, for any sequence* $(x_k)_{k\in\mathbb{N}}$ *of points of* U *which converges in the Euclidean topology to a point of* $\mathcal{M}$*, the sequence* $(\phi(x_k))_{k\in\mathbb{N}}$ *is bounded.*

Let us now describe the existing interdependence of the notions of a weakly continuous, a strongly continuous, a topologically robust and a hereditary map.

Lemma 3 *Let* $\phi : \mathcal{M} \to \mathbb{A}^m$ *be a strongly continuous constructible map. Then* ϕ *is weakly continuous, topologically robust and hereditary.*

On the other hand, a weakly continuous or a topologically robust map is not necessarily strongly continuous (see [GHMS11], Example 5).

The concept of hereditarity sounds rather abstract and axiomatic. We shall need it in Section 6 for a mathematically correct and complete formulation of our computation model. In Section 2.3 we shall establish an algebraic condition, namely geometric robustness, which implies hereditarity.

2.3 The concept of robustness for constructible maps

In this section we introduce an algebraic–geometric tool we shall use in Section 4 and 6 for the mathematical modeling of the informal concept of coalescence. The exact definition of coalescence depends on the context. For example in Interpolation Theory coalescence refers to certain types of "convergence" of problems and algorithms (see [BC97], [dBR92], [Olv06] and Section 4.2.3 for details).

The main issue of this section will be the notion of a *geometrically robust* map which captures simultaneously the concepts of topological robustness and hereditarity introduced

in Section 2.2. We start with an algebraic characterization of the notion of a topologically robust map (Theorem 5 and Corrollary 6 below). Then we introduce the notion of a geometrically robust map and show that such maps are always hereditary (see Theorem 9 and Proposition 8 below).

2.3.1 An algebraic characterization of the notion of topological robustness

In this subsection we shall exhibit a series of algebraic–geometric results of [GHMS11] which we shall use later in Sections 4 and 6. The proofs can be found in [GHMS11] and will be omitted here.

For the moment let us fix a constructible subset $\mathcal{M}$ of the affine space $\mathbb{A}^n$ and a (total) constructible map $\phi : \mathcal{M} \to \mathbb{A}^m$ with components $\phi_1, \ldots, \phi_m$. Suppose the map ϕ is weakly continuous in the sense of Definition 5 in Section 2.2, namely,

> *there exists a Zariski open and dense subset U of $\mathcal{M}$ such that the restriction $\phi|_U$ is a rational map of $\mathcal{M}$ and the graph of ϕ is contained in the Zariski closure Γ of the graph of $\phi|_U$ in $\mathcal{M} \times \mathbb{A}^m$.*

Observe that Γ is a constructible subset of $\mathbb{A}^n \times \mathbb{A}^m$ that contains the graph of ϕ. Furthermore, let π:

$$\begin{array}{ccc} \mathbb{A}^n \times \mathbb{A}^m & \to & \mathbb{A}^n \\ \subset & & \subset \\ \Gamma & \stackrel{\pi}{\to} & \mathcal{M} \end{array}$$

be the first projection of Γ onto $\mathcal{M}$ which for $(x, y) \in \Gamma$ is defined by $\pi(x, y) := x$. Observe that π is a polynomial map.

We recall from Definition 5 of Section 2.2 that the constructible map $\phi : \mathcal{M} \to \mathbb{A}^m$ is topologically robust if and only if it is weakly continuous and satisfies the following condition:

(∗) *for any sequence $(x_k)_{k\in\mathbb{N}}$ of $\mathcal{M}$ which converges in the Euclidean topology to a point of $\mathcal{M}$, the sequence $(\phi(x_k))_{k\in\mathbb{N}}$ is bounded.*

This condition is equivalent to the robustness of the surjective polynomial map $\pi : \Gamma \to \mathcal{M}$ in the sense of [CGH$^+$03], Definition 3. More precisely, we have the following result.

Remark 4 *With notations and assumptions as above, the weakly continuous constructible map ϕ satisfies condition (∗) if and only if for any sequence $(x_k, y_k)_{k\in\mathbb{N}}$ of points of Γ such that $(x_k)_{k\in\mathbb{N}}$ converges to a point $x_0 \in \mathcal{M}$, there exists an accumulation point y_0 of the sequence of $(y_k)_{k\in\mathbb{N}}$ with $(x_0, y_0) \in \Gamma$.*

We consider now the Zariski closure $\overline{\mathcal{M}}$ of the constructible subset $\mathcal{M}$ of $\mathbb{A}^n$. Observe that $\overline{\mathcal{M}}$ is a closed affine subvariety of $\mathbb{A}^n$ and that we may interpret $\mathbb{C}(\overline{\mathcal{M}})$ as a $\mathbb{C}[\overline{\mathcal{M}}]$–module (or $\mathbb{C}[\overline{\mathcal{M}}]$–algebra).

Fix now an arbitrary point x of $\overline{\mathcal{M}}$.

By $\mathfrak{M}_x$ we denote the maximal ideal of coordinate functions of $\mathbb{C}[\overline{\mathcal{M}}]$ which vanish at the point x.

By $\mathbb{C}[\overline{\mathcal{M}}]_{\mathfrak{M}_x}$ we denote the local $\mathbb{C}$–algebra of the variety $\overline{\mathcal{M}}$ at the point x, i.e. the localization of $\mathbb{C}[\overline{\mathcal{M}}]$ at the maximal ideal $\mathfrak{M}_x$.

By $\mathbb{C}(\overline{\mathcal{M}})_{\mathfrak{M}_x}$ we denote the localization of the $\mathbb{C}[\overline{\mathcal{M}}]$–module $\mathbb{C}(\overline{\mathcal{M}})$ at $\mathfrak{M}_x$.

We suppose now that the constructible map $\phi : \mathcal{M} \to \mathbb{A}^m$ is topologically robust. Then we may interpret $\phi_1, \ldots, \phi_m$ as rational functions of the affine variety $\overline{\mathcal{M}}$ and therefore as elements of the total fraction ring $\mathbb{C}(\overline{\mathcal{M}})$ of $\mathbb{C}[\overline{\mathcal{M}}]$.

Thus $\mathbb{C}[\overline{\mathcal{M}}][\phi_1, \ldots, \phi_m]$ and $\mathbb{C}[\overline{\mathcal{M}}]_{\mathfrak{M}_x}[\phi_1, \ldots, \phi_m]$ are $\mathbb{C}$–subalgebras of $\mathbb{C}(\overline{\mathcal{M}})$ and $\mathbb{C}(\overline{\mathcal{M}})_{\mathfrak{M}_x}$ which contain $\mathbb{C}[\overline{\mathcal{M}}]$ and $\mathbb{C}[\overline{\mathcal{M}}]_{\mathfrak{M}_x}$, respectively.

With these notations we are able to formulate the following statement which establishes the bridge to an algebraic understanding of the notion of topological robustness.

Theorem 5 *([GHMS11], Theorem 9) Let notations and assumptions be as before. Assume that the constructible map $\phi : \mathcal{M} \to \mathbb{A}^m$ is topologically robust and let x be an arbitrary point of $\mathcal{M}$. Then $\mathbb{C}[\overline{\mathcal{M}}]_{\mathfrak{M}_x}[\phi_1, \ldots, \phi_m]$ is a finite $\mathbb{C}[\overline{\mathcal{M}}]_{\mathfrak{M}_x}$–module.*

Theorem 5 is an immediate consequence of Remark 4 and [CGH$^+$03], Lemma 3, which in its turn is based on a non-elementary and deep result from Algebraic Geometry, namely Zariski's Main Theorem (see e.g. [Ive73], §IV.2). In the context of Interpolation and Elimination Theory, Theorem 5 and Theorem 9 below will be omnipresent in Sections 4 and 6. They contribute to establish a well–founded link between Computer Science and Algebraic Geometry.

Let us observe here that the proof of Theorem 5 requires sophisticated tools from Algebraic Geometry.

Let $\phi : \mathcal{M} \to \mathbb{A}^m$ be a topologically robust constructible map and let u be an arbitrary point of $\mathcal{M}$. From Theorem 5 one deduces easily that for all sequences $(u_k)_{k\in\mathbb{N}}$ of points $u_k \in \mathcal{M}$ which converge to u the sequences $(\phi(u_k))_{k\in\mathbb{N}}$ have only finitely many distinct accumulation points. In what follows, Theorem 5 will be only used in Section 6 in order to comment the notion of coalescence introduced here.

From Theorem 5 we deduce the following results.

Corollary 6 *Let notations and assumptions be as before and suppose in particular that the constructible map $\phi : \mathcal{M} \to \mathbb{A}^m$ is weakly continuous. Then ϕ is topologically robust if and only if for any point x of $\mathcal{M}$ the $\mathbb{C}$–algebra $\mathbb{C}[\overline{\mathcal{M}}]_{\mathfrak{M}_x}[\phi_1, \ldots, \phi_m]$ is a finite $\mathbb{C}[\overline{\mathcal{M}}]_{\mathfrak{M}_x}$–module.*

Corollary 7 *Let $\phi : \mathcal{M} \to \mathbb{A}^m$ be topologically robust and suppose that the affine variety $\overline{\mathcal{M}}$ is normal at any point of $\mathcal{M}$. Then $\phi : \mathcal{M} \to \mathbb{A}^m$ is a rational map of $\overline{\mathcal{M}}$ whose domain contains $\mathcal{M}$ and is therefore strongly continuous.*

2.3.2 The notion of geometrical robustness

The main mathematical tool of Sections 4 and 6 of this thesis is the notion of geometrical robustness we are going to introduce now. We shall use the same notations as in Section 2.3.1.

Definition 6 *Let $\mathcal{M}$ be a constructible subset of a suitable affine space and let $\phi : \mathcal{M} \to \mathbb{A}^m$ be a (total) constructible map with components $\phi_1, \ldots, \phi_m$. Based on Lemma 1 we may interpret $\phi_1, \ldots, \phi_m$ as rational maps of $\overline{\mathcal{M}}$. We call ϕ geometrically robust if for any point $x \in \mathcal{M}$ the following two conditions are satisfied:*

(i) $\mathbb{C}[\overline{\mathcal{M}}]_{\mathfrak{M}_x}[\phi_1,\ldots,\phi_m]$ *is a finite* $\mathbb{C}[\overline{\mathcal{M}}]_{\mathfrak{M}_x}$*–module.*

(ii) $\mathbb{C}[\overline{\mathcal{M}}]_{\mathfrak{M}_x}[\phi_1,\ldots,\phi_m]$ *is a local* $\mathbb{C}[\overline{\mathcal{M}}]_{\mathfrak{M}_x}$*–algebra whose maximal ideal is generated by* $\mathfrak{M}_x$ *and* $\phi_1-\phi_1(x),\ldots,\phi_m-\phi_m(x)$.

Observe that the notion of a geometrically robust map makes also sense when $\mathbb{C}$ is replaced by an arbitrary algebraically closed field (of any characteristic). In this sense we have the following fundamental result.

Proposition 8 *Geometrically robust constructible maps are weakly continuous, topologically robust and hereditary. Moreover the composition of two geometrically robust constructible maps is geometrically robust.*

We are not going to prove Proposition 8 here. Weak continuity and hereditarity of geometrically robust constructible maps with *irreducible* domains of definition was shown in [GHMS11], Proposition 16, Theorem 17 and Corollary 18. From this one deduces immediately the same result also for the case of arbitrary domains of definition. The alluded proofs work over arbitrary algebraically closed fields.

Topological robustness follows from the previous Corollary 6 and closedness under composition is a consequence of the transitivity law for integral dependence.

In this thesis we shall restrict our attention to the algebraically closed field $\mathbb{C}$. In this particular case we have the following characterization of geometrically robust constructible maps.

Theorem 9 *Let assumptions and notations be as before. Then the constructible map* $\varphi:\mathcal{M}\to\mathbb{A}^m$ *is geometrically robust if and only if* ϕ *is strongly continuous.*

Proof. Suppose that the constructible map ϕ is geometrically robust. Since ϕ satisfies condition (ii) for any point of $\mathcal{M}$, we conclude that ϕ is weakly continuous. Let be given an arbitrary point $x\in\mathcal{M}$ and a sequence $(x_k)_{k\in\mathbb{N}}$, $x_k\in\mathcal{M}$, which converges to x in the strong topology of $\mathcal{M}$. It suffice to show that the sequence $(\phi(x_k))_{k\in\mathbb{N}}$ converges to $\phi(x)$.

Since ϕ is weakly continuous, we deduce from condition (i) of Definition 6 and Corollary 6 that the sequence $(\phi(x_n))_k\in\mathbb{N}$ contains at least one accumulation point, say $a=(a_1,\ldots,a_m)$, which belongs to $\mathbb{A}^m$. Let $Y_1,\ldots,Y_m$ be new indeterminates, $Y:=(Y_1,\ldots,Y_m)$ and let $\mathfrak{a}$ be the ideal of all polynomials $A\in\mathbb{C}[\mathcal{M}]_{\mathfrak{M}_x}[Y]$ that vanish at the point $(x,a)\in\mathbb{A}^n\times\mathbb{A}^m$. Without loss of generality we may assume that the sequence $(\phi(x_k))_{k\in\mathbb{N}}$ converges to a. Let $\widetilde{\mathfrak{a}}:=\{A(\phi);A\in\mathfrak{a}\}$ be the image of the ideal $\mathfrak{a}$ under the surjective $\mathbb{C}[\overline{\mathcal{M}}]_{\mathfrak{M}_x}$–algebra homomorphism $\mathbb{C}[\overline{\mathcal{M}}]_{\mathfrak{M}_x}[Y]\to\mathbb{C}[\overline{\mathcal{M}}]_{\mathfrak{M}_x}[\phi_1,\ldots,\phi_m]$ which maps $Y_1,\ldots,Y_m$ onto $\phi_1,\ldots,\phi_m$. Observe that $\widetilde{\mathfrak{a}}$ is an ideal of $\mathbb{C}[\overline{\mathcal{M}}]_{\mathfrak{M}_x}[\phi_1,\ldots,\phi_m]$.

We are now going to show the following statement.

Claim 10 *The ideal* $\widetilde{\mathfrak{a}}$ *is proper.*

Proof of the claim. Suppose that the ideal $\widetilde{\mathfrak{a}}$ is not proper. Then there exists a polynomial $A=\sum_{j_1,\ldots,j_m}a_{j_1\ldots j_m}Y_1^{j_1}\ldots Y_m^{j_m}$ of $\mathfrak{a}$, with $a_{j_1\ldots j_m}\in\mathbb{C}[\overline{\mathcal{M}}]_{\mathfrak{M}_x}$, which satisfies the condition $\sum_{j_1,\ldots,j_m}a_{j_1\ldots j_m}\phi_1^{j_1}\ldots\phi_m^{j_m}=A(\phi)=1$. Since for any m–tuple of indices $j_1,\ldots,j_m$ the rational function $a_{j_1\ldots j_m}$ of $\overline{\mathcal{M}}$ is defined at x and the sequence $(x_k)_{k\in\mathbb{N}}$

converges to x, we may assume without loss of generality that $a_{j_1\dots j_m}$ is defined at x_k for any $k \in \mathbb{N}$ and that $(a_{j_1\dots j_m}(x_k))_{k\in\mathbb{N}}$ converges to $a_{j_1\dots j_m}(x)$. We may therefore write $A^{(x')} := \sum a_{j_1\dots j_m}(x')Y_1^{j_1}\dots Y_m^{j_m} \in \mathbb{C}[Y]$ for $x' := x$ or $x' := x_k$, $k \in \mathbb{N}$. From $A \in \mathfrak{a}$ we deduce $A^{(x)}(a) = 0$. By assumption $(\phi(x_k))_{k\in\mathbb{N}}$ converges to a. Hence the sequence of complex numbers $(A^{(x_k)}(\phi(x_k)))_{k\in\mathbb{N}}$ converges to $A^{(x)}(a) = 0$. On the other hand $A(\phi) = 1$ and the weak continuity of ϕ imply $A^{(x_k)}(\phi(x_k)) = 1$ for any $k \in \mathbb{N}$. This contradiction proves our claim.

■

From condition (ii) of Definition 6 we deduce that the $\mathbb{C}[\overline{\mathcal{M}}]_{\mathfrak{M}_x}$–algebra $\mathbb{C}[\overline{\mathcal{M}}]_{\mathfrak{M}_x}[\phi_1,\dots,\phi_m]$ contains a single maximal ideal, say $\mathfrak{M}$, and that $\mathfrak{M}$ is generated by $\mathfrak{M}_x$ and $\phi_1 - \phi_1(x),\dots,\phi_m - \phi_m(x)$.

Since by Claim 10 the ideal $\widetilde{\mathfrak{a}}$ is proper, $\widetilde{\mathfrak{a}}$ must be contained in $\mathfrak{M}$. Observe that the polynomials $Y_1 - a_1,\dots,Y_m - a_m$ belong to $\mathfrak{a}$. Hence $\phi_1 - a_1,\dots,\phi_m - a_m$ belong to $\widetilde{\mathfrak{a}}$ and therefore also to $\mathfrak{M}$. Since $\mathfrak{M}$ is proper, this is only possible if $a_1 = \phi_1(x),\dots,a_m = \phi_m(x)$ holds.

Thus the sequence $(\phi(x_k))_{k\in\mathbb{N}}$ converges to $\phi(x)$.

Suppose now that the constructible map ϕ is strongly continuous. From Lemma 3 we deduce that ϕ is topologically robust. Corollary 6 implies now that ϕ satisfies condition (i) of Definition 6 at any point of $\mathcal{M}$.

Let x be an arbitrary point of $\mathcal{M}$. We have to show that ϕ satisfies at x condition (ii) of Definition 6.

Since the graph of ϕ is constructible, its strong and Zariski closures in $\mathcal{M}\times\mathbb{A}^m$ coincide. Moreover, since ϕ is by assumption strongly continuous, its graph is closed with respect to the strong topology of $\mathcal{M}\times\mathbb{A}^m$ and therefore also with respect to the Zariski topology. Let $\mathfrak{a}$ be an arbitrary maximal ideal of the $\mathbb{C}[\overline{\mathcal{M}}]_{\mathfrak{M}_x}$–algebra $\mathbb{C}[\overline{\mathcal{M}}]_{\mathfrak{M}_x}[\phi_1,\dots,\phi_m]$. Then there exists a point $a = (a_1,\dots,a_m)$ of $\mathbb{A}^m$ such that $\mathfrak{a}$ is generated by $\mathfrak{M}_x$ and $\phi_1 - a_1,\dots,\phi_m - a_m$. Thus $(x,a) \in \mathcal{M}\times\mathbb{A}^m$ belongs to the Zariski closure of the graph of ϕ in $\mathcal{M}\times\mathbb{A}^m$ and therefore to the graph of ϕ itself. This implies $a = \phi(x)$. With other words, $\mathfrak{a}$ is generated by $\mathfrak{M}_x$ and $\phi_1 - \phi_1(x),\dots,\phi_m - \phi_m(x)$. There is exactly one ideal of $\mathbb{C}[\overline{\mathcal{M}}]_{\mathfrak{M}_x}[\varphi_1,\dots,\varphi_m]$ which satisfies this condition. Therefore the $\mathbb{C}[\overline{\mathcal{M}}]_{\mathfrak{M}_x}$–algebra $\mathbb{C}[\overline{\mathcal{M}}]_{\mathfrak{M}_x}[\varphi_1,\dots,\varphi_m]$ is local and condition (ii) and Definition 6 is satisfied at the point $x \in \mathcal{M}$.

■

Theorem 9 is new. It gives a topological motivation for the rather abstract, algebraic notion of geometrical robustness. The reader not acquainted with commutative algebra may just identify the concept of geometrical robustness with that of strong continuity of constructible maps.

Observe that Proposition 8 follows immediately from Theorem 9 in the case of the algebraically closed field $\mathbb{C}$.

Definition 7 (Finitely determined map) *Let $\mathcal{M}$ be a constructible subset of a suitable affine space and $\phi : \mathcal{M} \to \mathbb{A}^m$ and $\psi : \mathcal{M} \to \mathbb{A}^r$ two geometrically robust constructible maps. Let $x \in \mathcal{M}$. We say that ψ is finitely determined by ϕ at x if for any sequence $(x_k)_{k\in\mathbb{N}}$ with $x_k \in \mathcal{M}$ such that $(\phi(x_k))_{k\in\mathbb{N}}$ converges to $\phi(x)$ the sequence $(\psi(x_k))_{k\in\mathbb{N}}$ is bounded. Otherwise we call ψ indetermined by ϕ at x. If ψ is finitely determined by ϕ at any point $x \in \mathcal{M}$ then we call ψ finitely determined by ϕ. Otherwise we call ψ idetermined by ϕ.*

We are now going to show that indeterminacy of ψ by ϕ at a given point $x \in \mathcal{M}$ has a very strong meaning.

Lemma 11 *Let assumptions and notations be as in Definition 7 and suppose that the constructible set $\{(\phi(x), \psi(x)); x \in \mathcal{M}\}$ is locally closed in $\mathbb{A}^m \times \mathbb{A}^r$. Let x be a point of $\mathcal{M}$ and suppose that ψ is indetermined by ϕ at x. Then there exists a sequence $(z_k)_{k\in\mathbb{N}}$ with $z_k \in \mathcal{M}$ and $\phi(z_k) = \phi(x)$ such that $(\psi(z_k))_{k\in\mathbb{N}}$ is unbounded. In particular, if $r = 1$, the set of complex numbers $\{\psi(z); z \in \mathcal{M}, \phi(z) = \phi(x)\}$ is cofinite. If ψ is indetermined by ϕ then there exists a point $x \in \mathcal{M}$ having this property.*

Proof. Let $\phi = (\phi_1, \ldots, \phi_m)$ and $\psi = (\psi_1, \ldots, \psi_r)$ and interpret $\phi_1, \ldots, \phi_m$ and $\psi_1, \ldots, \psi_r$ as rational functions of $\overline{\mathcal{M}}$. Let be given $x \in \mathcal{M}$ and suppose that ψ is indetermined by ϕ at x. Then there exists a sequence $(x_k)_{k\in\mathbb{N}}$ with $x_k \in \mathcal{M}$ such that $(\phi(x_k))_{k\in\mathbb{N}}$ converges to $\phi(x)$ and such that $(\psi(x_k))_{k\in\mathbb{N}}$ is unbounded. Without loss of generality we may suppose that the sequence of complex numbers $(\psi_1(x_k))_{k\in\mathbb{N}}$ is unbounded. Let $T_1, \ldots, T_m$ and $Y_1, \ldots, Y_r$ be new indeterminates, let $T := (T_1, \ldots, T_m)$ and $Y := (Y_1, \ldots, Y_r)$ and consider the ideal $\mathfrak{a}$ of all polynomials $A \in \mathbb{C}[T, Y]$ such that $A(\phi, \psi) = 0$ holds in $\mathbb{C}(\overline{\mathcal{M}})$.

Since $\phi_1, \ldots, \phi_m$ and ψ are continuous with respect to the strong topologies of $\mathcal{M}$ and $\mathbb{C}$, we have $A(\phi(z), \psi(z)) = 0$ for any $A \in \mathfrak{a}$ and $z \in \mathcal{M}$.

Claim 12 *Any $A \in \mathfrak{a} \cap \mathbb{C}[T, Y_1]$ satisfies the condition $A(\phi(x), Y_1) = 0$.*

Proof of Claim. Suppose at the contrary that there exists a polynomial $A \in \mathfrak{a} \cap \mathbb{C}[T, Y_1]$ with $A(\phi(x), Y_1) \neq 0$. Then we may suppose without loss of generality that $A(\phi(x_k), Y_1) \neq 0$ holds for any $k \in \mathbb{N}$. Since $(\phi(x_k))_{k\in\mathbb{N}}$ converges to $\phi(x)$ there exists a constant $c > 0$ such that for $k \in \mathbb{N}$ the absolute value of all zeros of $A(\phi(x_k), Y_1)$ is bounded by c. From $A(\phi(x_k), \psi_1(x_k)) = 0$ we deduce now the estimate $|\psi_1(x_k)| \leq c$ for any $k \in \mathbb{N}$. This implies that the sequence $(\psi_1(x_k))_{k\in\mathbb{N}}$ is bounded, contradicting our assumptions on ψ_1 and $(x_k)_{k\in\mathbb{N}}$. ■

From the claim we deduce that the ideal $\mathfrak{a}_x := \{A(\phi(x), Y); A \in \mathfrak{a}\}$ is proper and defines a closed subvariety V_x of A^r which has positive dimension. Since by assumption the constructible set $\{(\phi(z), \psi(z)); z \in \mathcal{M}\}$ is locally closed in $\mathbb{A}^m \times \mathbb{A}^r$ there exists a Zariski open subset O_x of $\mathbb{A}^r$ with $\{y \in \mathbb{A}^r; \exists z \in \mathcal{M}, \phi(z) = \phi(x), y = \psi(z)\} = V_x \cap O_x$. Since $V_x \cap O_x$ is a locally closed subvariety of $\mathbb{A}^r$ of positive dimension, we deduce from [CGH$^+$03], Lemma 1, that $V_x \cap O_x$ is unbounded. Therefore there exists a sequence $(z_k)_{k\in\mathbb{N}}$ with $z_k \in \mathcal{M}$ and $\phi(z_k) = \phi(x)$ such that $(\psi(z_k))_{k\in\mathbb{N}}$ is unbounded. In case $r := 1$ this means that the constructible set $\{\psi(z); z \in \mathcal{M}, \phi(z) = \phi(x)\}$ is cofinite. ■

The origin of the concept of a geometrically robust map can be found, implicitly, in [GH01] as an analysis of the Example 21 in Section 6.5.5 of this thesis. This analysis refers to a Hermite–Lagrange interpolation problem in the sense of Section 4 and is therefore well motivated from the point of view of Computer Science.

Our conclusion is that the mathematical notion of a geometrically robust map is highly meaningful for Computer Science applications as those treated in Sections 4 and 6 of this thesis.

3 Concepts from Software Engineering

In this section we collect some basic facts from Software Engineering which allow us to establish a computationally meaningful mathematical model for fundamental aspects of interpolation and effective geometric elimination theory. We use standard notions and notations, which can be found in [BCK03], [Mey00], [GJM91] and [CBB+02].

3.1 Basic definitions and notations

According to Ghezzi [GJM03], Software Engineering is the field of Computer Science that deals with the building of large software systems by teams of programmers. This process of building software involves various aspects such as management and technical activities. Management is necessary in order to give order and to guide the project to success. On the other hand, the technical activities provide the main techniques for software construction. Technical activities can be organized in specification, software design and implementation. In this section we focus our attention on software design.

According to Pressman [Pre01], design provides a representation of the software that will be built. This representation consist of data structures, relations between data structures, interface design and procedural description of software components. This elements become concentrated in the main outcome of the software design activity which is the software architecture.

3.1.1 Software architecture

According to Pressman [Pre01] a software architecture consists of a representation of the software to be implemented. Thus a software architecture is a description of the components of the given system, the structure of these components and the relationships between them. Similarly, [CBB+02] describes software architecture as a complex entity which cannot be described in a simple one–dimensional fashion. This complex entity requires descriptions based on particular perspectives called viewtypes, as for example the decomposition of the system by units of implementation (module viewtype), its decomposition by units of runtime execution (C&C viewtype) or the mapping from software elements to environmental structures (allocation viewtype).

In this thesis we describe software architectures which come close to the module viewtype perspective. We are now going to introduce the elements and relationships which constitute our software architectures.

Abstract Data Types and Classes

Software architecture has its origins in the notion of abstract data type. According to [GS94], in the historical development of abstraction techniques, abstract data types raise from the work of programmers of the late 1960's. They saw in data structures the key to an easier development of software. Abstract data types was a technique to understand a module and its particular purpose. For example, the services provided by a module are the functions declared in an abstract data type specification. This last characteristic was adopted by object oriented programming where, according to Meyer [Mey00], abstract data types become used as a tool to break down the given system into modules.

Definition 8 (Abstract Data Type) *According to Meyer [Mey00] an abstract data type is a set of elements defined by a list of operations applicable to these elements. An abstract data type is formalized by an abstract data type specification which describes, by available services (functions) and formal properties (axioms), a set of elements.*

Abstract data types constitute the starting point to the construction of the modules in object oriented approaches. These modules become realized by classes which become then modular units.

Definition 9 (Class) *According to [Mey00] a class is a software element describing an abstract data type and its partial or total implementation. A class is described by a list of features (attributes and routines). The features constitute the basis of the interaction of the class with the rest of the software.*

Here arise two important questions about a given software architecture. What are the components? How are they composed? In object oriented programming components may be classes. In this context, Meyer [Mey00] argues that there exist only two types of relations between classes: client and inheritance relations.

We wish to clarify that a class may be considered as an abstract data type equipped with a concrete implementation. This introduces another relation between the components of an architecture, namely the "Implements" relation.

For example, consider the problem of evaluating a polynomial. Suppose that our implementation consists of a class "Polynomial" whose evaluation returns a class "Integer". Here "Polynomial" is a client of the class "Integer", i.e. the class "Polynomial" uses the routines available on the class "Integer", namely arithmetic operations. In this scenario we suppose that the class "Integer" inherits relations and operations from a class "Number" (observe that we meet here an example of inheritance relation). On the other hand, the "Implements" relation appears in the context of the class "Polynomial". Consider now "Polynomial" as an abstract data type. In this meaning "Polynomial" requires an implementation of its (abstract) objects, which may be realized by the coefficients of these objects, which on their turn may be implemented directly by bits. Figure 4 illustrates this kind of relations.

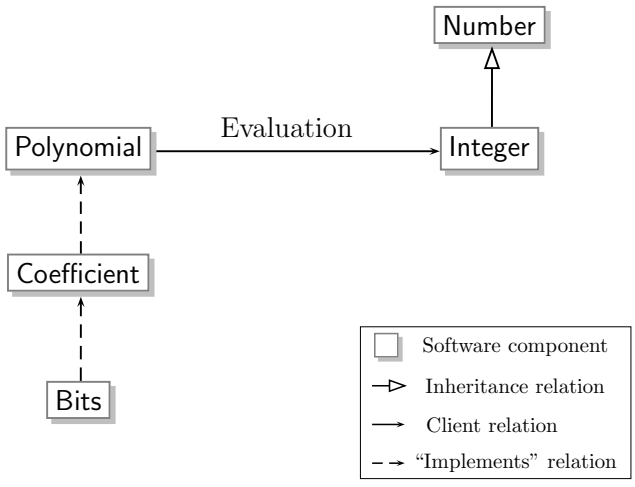


Figure 4: Relations between the components of a software architecture

In this thesis we focus our attention on the "Implements" relation between the different levels of abstraction. In Section 4.3 we shall describe a software architecture for the problem of Hermite–Lagrange interpolation where, for the sake of simplicity, we identify two levels of abstraction.

Let us now introduce other important notions from software architecture.

3.1.2 Functional and (non–functional) requirements

According to [BD09] a functional requirement is a specification of a mathematical function that the given system has to support. In other words, following [MAS+03], a functional requirement can be expressed in mathematical terms: as an input–output behaviour of a given (mathematical) function. An example for a functional requirement are the axioms satisfied by the functions of an abstract data type (these axioms form part of the specification).

A linguistic analogy is described in [Dou06] where a functional requirement is compared with a verb. In its turn, the meaning of the verb may become modified by an adverb. This adverb plays the rôle of a non–functional requirement of the given system.

Quality attributes

According to [Cer09] "non–functional requirement" represents an old term for "quality attribute". We consider now the notion of quality attribute under different names in the literature. For example, according to [Mey88] a software product is affected by *quality factors* which may be present or absent in the software. Another terminology is used in [Par72] where flexibility and comprehensibility are called *benefits*. Still another terminology appears in [Med00] where *non–functional properties* represent additional requirements that cannot be derived from the specification. Non–functional properties are required to select appropriate configurations of the architecture structure under consideration.

On the other hand, [BCK98] explains that the term non–functional requirement is incorrect. However, in the present contribution the terms "non–functional requirement" and "quality attribute" have the same meaning. According to [BKLW95] a critical system must satisfy functional requirements and quality attributes. The quality attributes are properties of the service delivered by the system. In this sense, [BCK03] explains that quality attributes affect negatively or positively other quality attributes.

Taking into account this review, we fix the meaning of the term "quality attribute" in this thesis by the following way:

Definition 10 (Quality attribute) *A quality attribute is a property or requirement on the software system which concerns its behaviour beyond its functionality given by the specification and the correctness of the underlying programs. A quality attribute may constitute a property of the system that affects negatively or positively other quality attributes.*

In [BCK03] the author introduce a methodology for the analysis of quality attributes. This methodology consists in so called "quality attribute scenarios" which we shall discuss in Section 5.1.

Quality attribute trade–offs

The term trade–off refers to a situation where two opposing qualities become balanced. For example, there is a trade–off between doing a task accurately and doing it quickly. Another trade–off exists between security and performance.

This thesis is devoted to the study of the trade–off between computational complexity as quality attribute of algorithms of scientific computing and certain other quality attributes which become introduced in Section 4 and 6. According to [CBK+98], software architecture is the discipline of quality attributes. Thus we shall deal with a central aspect of software architecture theory.

Arithmetic circuits

Algorithms in computational algebraic geometry are usually described by software architectures where a polynomial is implemented by means of the vector of all its (or of all its nonzero) coefficients, i.e. using the standard dense (or sparse) complexity encoding.

Taking into account that a generic n–variate polynomial of degree $d \geq 2$ has $\binom{d+n}{n} = O(d^n)$ nonzero coefficients, we see that such an implementation of multivariate polynomials requires an exponential size. Moreover, their manipulation usually requires a data structure of exponential number of arithmetic operations with respect to the parameters d and n. In order to avoid this exponential behaviour, we are going to use alternative software architectures where polynomials are implemented by means of arithmetic circuits (or straight–line programs in terminology of [GHMS11]).

Definition 11 (Arithmetic circuit) *Let $X_1, \ldots, X_n$ be indeterminates over $\mathbb{C}$. An (ordinary) arithmetic circuit over $\mathbb{C}$ called β, with inputs $X_1, \ldots, X_n$, consists of a labelled directed acyclic graph (labelled DAG) satisfying the following conditions:*
each node of indegree zero is labelled by a complex number (called scalar of β) or an input variable $X_1, \ldots, X_n$, Following the case, we shall refer to them as scalar and input nodes of β. All other nodes of β have indegree zero and unlimited outdegree. They are called internal nodes of β and they are labelled by one of the arithmetic operations, namely addition, subtraction, multiplication or division. At least one of the nodes become labelled as output. Without loss of generality we shall suppose that all nodes of outdegree zero are output nodes.

We may consider β as a syntactical object which we wish to equip with a certain semantics. In principle there exists a canonical evaluation procedure of β which assigns to each node a rational function of $\mathbb{A}^n$. We call such a rational function an *intermediate* result of β. The intermediate results of β which become associated with output nodes are called *final results*.

The evaluation procedure may fail if we divide at some moment an intermediate result by another one which is identically zero on $\mathbb{A}^n$. If this is the case we call the circuit *inconsistent*, otherwise *consistent*. From [CGH+03], Corollary 2 (see also [HS], Theorem 4.4 and [GH01], Lemma 3) we deduce that consistency of arithmetic circuits can be deduced using a quantity of arithmetic operations in $\mathbb{C}$ which is polynomial in the size (i.e. number of internal nodes) of the given circuit. The rest exists in a uniform probabilistic and in a non–uniform deterministic variant which can be bolled down to the bit model. However,

whether this test can be performed efficiently in the *uniform* complexity model, is a major open question in Computer Science.

We call the circuit β *(totally) division–free* if it contains only divisions by scalars. Observe that the intermediate results of division–free arithmetic circuits are all polynomials over $\mathbb{C}$ in the input variables and that they are always consistent. From now on we shall suppose that all our arithmetic circuits are consistent.

In this thesis we shall mainly be concerned with sequential complexity time, measured in terms of the size of β. We shall limit our attention to the *non–scalar size* (or sequential execution time) of the circuit β. This means that we count, at unit costs, only *essential* multiplications and divisions (involving input variables in both arguments in the case of multiplication and in the second argument in the case of a division), whereas $\mathbb{C}$–linear operations are free.

Frequently we shall use (consistent) arithmetic circuits as a data structure to represent polynomials and rational functions. This converts them into *objects* of a *class* where two fundamental *services* are *available*, which we are going to explain soon, namely an efficient identity test and the evaluation function.

The arithmetic circuit β represents by its output nodes a finite number of rational functions of $\mathbb{C}(X_1, \ldots, X_n)$ or polynomials of $\mathbb{C}[X_1, \ldots, X_n]$ (if β is division–free). This defines an *abstraction function* from the class of arithmetic circuits into the *abstract data type* of rational functions and polynomials over $\mathbb{C}$ (see Section 4.9). In this thesis we shall be largely concerned with this abstraction function.

Finally we are going to explain the two services available on the class of arithmetic circuits.

Let be given a final result $F \in \mathbb{C}(X_1, \ldots, X_n)$ of the arithmetic circuit β. Then F can be evaluated efficiently by β in almost any point of $\mathbb{A}^n$, and, if β is division–free, in any point of $\mathbb{A}^n$. Thus the value of F in suitable points of $\mathbb{A}^n$ is the first service available on input β for the class of arithmetic circuits.

Given two circuits β y β' and two final results F and F' of them, we may ask whether $F = F'$ holds. This question may be answered efficiently in terms of the size of β and β' by [CGH$^+$03], Corollary 2. Thus the answer to the question $F \stackrel{?}{=} F'$ is the second service available on input (β, β') for the class of arithmetic circuits.

4 Univariate Hermite–Lagrange interpolation

At this point all the notions we need from Algebraic Geometry and Software Engineering have been introduced. These notions allow us to study the mathematical model for elimination and interpolation algorithms of [GHMS11], [CGH$^+$03] and Section 6 below in terms of Software Engineering. We use standard notations of Numerical Analysis and Interpolation Theory, which can be found in, e.g. [BC97], [BF02], [Gau97], [Kre07] and [SB93].

4.1 Interpolation: an area of Numerical Analysis

Numerical Analysis deals with the approximative treatment of finite data, as vectors or arrays (e.g. matrices) of real numbers. Methodologically it is based on Classical Analysis, Diophantine Approximation, and, in some extent, on Algebraic Complexity Theory and marginally also on non–archimedean theories like p–adic analysis.

In applied mathematics, generally infinite objects, as functions and maps, become treated by Approximation Theory which deals with the following situation:
there are given a set of "complicated" functions, a set of "simple" approximants and a metric which establishes a link between both sets. Given a "complicated" function f, the theoretic part of Approximation Theory discusses the existence and uniqueness of a best approximant. The next question deals with more practical aspects such as which data of the "complicated" function f are required to determine and compute a best (or simply good) approximant? If f is a function which maps reals onto reals, the approximants may be special functions, e.g. rational or trigonometric functions. This gives rise to different types of approximation.

In this thesis we are only concerned with approximants which are polynomials, i.e. with polynomial approximation. According to [Ral70] the field of polynomial approximation uses different techniques:

- Approximation by interpolation
- Least-squares method
- Error min-max method

Maybe the most important link between Approximation Theory and Numerical Analysis becomes represented by Interpolation Theory. Interpolation constitutes the starting point to many methods in other areas of Numerical Analysis, e.g. in numerical differentiation and solution of differential equations. Interpolation has also direct applications, following [Con65] interpolation is normally used to:

- integrate and differentiate complicated functions replacing them by simple interpolants,
- evaluate a function given by values at some fixed points at an extra point.

In the present work we do not enter into the relationship between approximation and interpolation. Thus, we do not discuss the error associated with the approximations which is a common subject in Numerical Analysis textbooks. We discuss interpolation just as an

example for a software architecture which visualizes an interaction between certain quality attributes.

Interpolation can also be interpreted as a special case of quantifier elimination in elementary Algebraic Geometry. Namely, we may formulate interpolation problems by means of particular quantified first order formulas over suitable fields (e.g. the real numbers). This relationship becomes explained in more detail at the end of Section 4.3 where we introduce a mathematical model for interpolation problems and algorithms.

4.2 Lagrange interpolation problems and algorithms

Let be given a function $f : \mathbb{C} \to \mathbb{C}$. In the terminology of [BF02] the Lagrange interpolation problem consists of *finding* for any finite set S of distinct nodes of $\mathbb{C}$ the unique polynomial of minimal degree which agrees with f on S. Such a polynomial is called an interpolant.

If we consider this formulation as part of an informal specification of an interpolation algorithm, we see that the word *finding* hides important aspects concerning data types and routines. For example, the interpolant may be represented as a Lagrange Interpolating Polynomial, in monomial form or by Newton's Divided–Difference Formula.

In the following sections we shall introduce a mathematical model which allows us to reveal these hidden aspects by means of the notions of interpolation problem and interpolation algorithm.

4.2.1 Lagrange interpolation problems

Let n be a fixed natural number. Informally, an (n–variate) Lagrange interpolation problem is determined by the following items.

Definition 12 (Interpolation datum) *An interpolation datum d is a K–tuple $((x_1, y_1), \ldots, (x_K, y_K))$ of nodes $x_i \in \mathbb{A}^n$ and values $y_i \in \mathbb{A}^1$ with $x_i \neq x_j$, $1 \leq i < j \leq K$.*

We suppose now that there is given a set $\mathcal{O}^*$ of n–variate polynomials over $\mathbb{C}$, called *interpolants*.

Definition 13 (Interpolant) *For each interpolation datum $d := ((x_1, y_1), \ldots, (x_K, y_K))$ as in Definition 12, there exists exactly one interpolant $p \in \mathcal{O}^*$ which satisfies the interpolation condition $p(x_i) = y_i$ for any $1 \leq i \leq K$.*

For example, the K–tuple $d := ((0, 1), (1, 6), (-1, 2))$ expresses that for each node $x_1 := 0$, $x_2 := 1$, $x_3 := -1$ the value of the interpolant is $y_1 := 1$, $y_2 := 6$, $y_3 := 2$ respectively. This interpolation condition is satisfied by the polynomial $p(X) := 3X^2 + 2X + 1$.

At this stage we are trying to find a mathematical model for the informal concept of an interpolation problem and we are not concerned yet with the implementation of a solution of such a problem. In terms of Software Engineering we are only dealing with an abstract function Φ which associates to each interpolation datum d its interpolant p. In this spirit, the notions we have introduced, namely interpolation datum and interpolant may be specified in the following way:

- The set of all tuples of interpolation data is specified by an abstract data type $\mathcal{O}$.
- The set of all interpolants is specified by an abstract data type $\mathcal{O}^*$.
- All the possible algorithms that relate an interpolation datum d with its corresponding interpolant p are specified by the abstract function Φ. We may visualize this situation as follows:

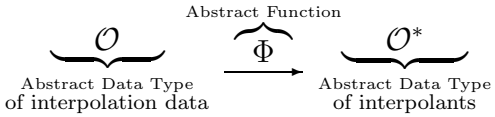


Figure 5: Lagrange interpolation problem

4.2.2 Mathematical modeling of the notion of Lagrange interpolation problem

Let n, D, K be discrete parameters belonging to $\mathbb{N}$. Let $X := (X_1, \ldots, X_n)$, where $X_1, \ldots, X_n$ are indeterminates over $\mathbb{C}$, and denote by Π (or, more precisely, by $\Pi^{(n)}$) the polynomial ring $\mathbb{C}[X] = \mathbb{C}[X_1, \ldots, X_n]$ and by Π_D the $\mathbb{C}$–vector space of polynomials of Π of degree at most D.

The abstract data types $\mathcal{O}$, $\mathcal{O}^*$ and the abstract function Φ become realized by the following mathematical structures:

- The abstract data type $\mathcal{O}$ of interpolation data is a constructible subset of the affine ambient space $\mathbb{A}^{(n+1)\times K}$ consisting of suitable K–tuples $((x_1, y_1), \ldots, (x_K, y_K))$ of interpolation data, with $x_i \in \mathbb{A}^n$, $y_i \in \mathbb{A}^1$ and $1 \leq i \leq K$.
- The abstract data type $\mathcal{O}^*$ of interpolants is a constructible subset of the finite dimensional vector space Π_D.
- The abstract function Φ is a surjective constructible map $\Phi : \mathcal{O} \to \mathcal{O}^*$ which associates to each interpolation datum $d \in \mathcal{O}$ an interpolant $\Phi(d) \in \mathcal{O}^*$.

We may now ask whether it is meaningful from the point of view of Computer Science to realize an abstract data type by mathematical structures like a constructible subset of the affine ambient space $\mathbb{A}^{(n+1)\times K}$ or a finite dimensional vector space Π_D. This question is justified since an abstract data type is a notion of Software Engineering and apparently unrelated with algebraic–geometric structures.

By a closer view to the essence of Computer Science we see that this discipline is effectively based on a careful manipulation of mathematical objects which appear there in a highly structured way. There is a clear division of types of mathematical objects which become organized by levels of abstraction in hierarchies. The key concept is the use of mathematical structures of low hierarchical level to represent mathematical structures of high hierarchical level. Low level structures are used to implement high level structures, which on their turn play the role of abstract data type specifications of the low level structures.

4.2.3 Additional requirements: Coalescence

The abstract data types $\mathcal{O}$, $\mathcal{O}^*$ and the abstract function Φ specify an interpolation problem by means of a clear functionality:
for a given input, namely an interpolation datum, an output, namely an interpolant, is returned.

In addition to this functionality we may consider the following property of the constructible abstract function Φ, which is very natural in the context of interpolation (in another context maybe not, see e.g. machine learning in Section 5). Classical Interpolation Theory refers to this property as *coalescence*.

Definition 14 (Coalescence) *Let assumptions and notations be as before. We call the abstract function Φ coalescent if the following condition is satisfied:*
for any sequence $(d_k)_{k\in\mathbb{N}}$ of interpolation data converging to an interpolation datum d belonging to $\mathcal{O}$, the sequence $(\Phi(d_k))_{k\in\mathbb{N}}$ converges to $\Phi(d)$.

Since by assumption the abstract function Φ is constructible, Theorem 9 of Section 2.3.2 implies that Φ is coalescent if and only if Φ is geometrically robust. Thus, in the given context, the notions of coalescence and geometrical robustness coincide.

In the sequel we shall require that the abstract function Φ is coalescent. In mathematical terms coalescence may be paraphrased as strong continuity of Φ. The map Φ and its property of coalescence establish a certain interdependence between the interpolation data from $\mathcal{O}$ and the interpolants from $\mathcal{O}^*$. We require also that the essential features of this interdependence become preserved when we restrict the elements of $\mathcal{O}$ to an arbitrary constructible subset. This points to hereditarity which is guaranteed by Proposition 8 of Section 2.3.2 if Φ is coalescent. Since $\mathcal{O}$ and $\mathcal{O}^*$ have affine ambient spaces, the essential feature of their interdependence is of topological or geometrical nature.

Needless to say that in classic Lagrange interpolation theory the abstract function Φ is always strongly continuous (and hence geometrically robust).

4.2.4 Lagrange interpolation algorithms

For the modeling of the concept of a Lagrange interpolation algorithm, we have to deal with the implementations of abstract data types, namely classes. Another point is the relationship between abstract data type and class, which ensures that the class correctly implements the corresponding abstract data type. For this purpose [Hoa72] introduces the notion of an abstraction function. Finally, we have to model the concept of a routine which computes the interpolant. This point requires special attention because of our quality attribute concerns.

In this sense, the components which constitute an interpolation algorithm may be listed as follows:

- A class $\mathcal{D}$ which implements the abstract data type $\mathcal{O}$ of interpolation data and a class $\mathcal{D}^*$ which implements the abstract data type $\mathcal{O}^*$ of interpolants.
- The connection between abstract data types and classes is realized by abstraction functions which we denote by $\omega^* : \mathcal{D}^* \to \mathcal{O}^*$ and $id : \mathcal{D} \to \mathcal{O}$. For the sake of

simplicity we suppose $\mathcal{D} = \mathcal{O}$ and that $id : \mathcal{D} \to \mathcal{O}$ is the identity function since we do not consider alternative implementations for interpolation data. In other words, the abstract data type $\mathcal{O}$ and its implementing class $\mathcal{D}$ are notions which reflect distinct aspects of the same mathematical object, namely the interpolation data. With respect to the interpolants, we wish to admit as the class $\mathcal{D}^*$ more general implementations like, for example, the domain of parameter instances of a suitable generic arithmetic circuit (see Section 6.3 for a definition). In order to illustrate this view we shall exhibit some examples at the end of this section.

- The algorithm in the narrow sense is a routine $\Psi : \mathcal{D} \to \mathcal{D}^*$. This routine computes for each concrete input object d of the class $\mathcal{D}$ a concrete output object, say $\Psi(d)$ of the class $\mathcal{D}^*$. This relation becomes visualized by the following figure.

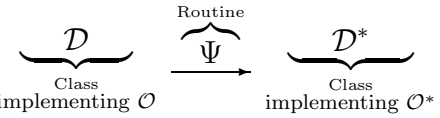


Figure 6: Algorithm in the narrow sense

4.2.5 Mathematical modeling of the notion of Lagrange interpolation algorithm

The classes $\mathcal{D}$, $\mathcal{D}^*$, the routine Ψ and the abstraction function ω^* become realized by the following mathematical structures:

- The classes $\mathcal{D}$ and $\mathcal{D}^*$ are constructible subsets of the affine ambient spaces $\mathbb{A}^N$ and $\mathbb{A}^M$ respectively, where N and M are given natural numbers.
- The abstraction function ω^* is a polynomial map, i.e. a morphism of affine spaces $\omega^* : \mathbb{A}^M \to \Pi_D$. For the sake of notational simplicity we shall also write $\omega^* : \mathcal{D}^* \to \mathcal{O}^*$ for the restriction of ω^* to the domain $\mathcal{D}^*$ and the image $\mathcal{O}^* := \omega^*(\mathcal{D}^*)$.
- We model the routine Ψ using a total map $\Psi : \mathcal{D} \to \mathcal{D}^*$ which satisfies the condition $\omega^*(\Psi(d)) = \Phi(d)$ for any $d \in \mathcal{D}$.

We require the routine Ψ to satisfy certain additional conditions we are going to explain now.

According to the requirement made before on the Lagrange interpolation problem, we wish that Ψ is in some sense *computable* and that Ψ remains computable if we restrict its domain to an arbitrary constructible subset of $\mathcal{D}$. Thus, we require that Ψ is hereditary.

The coalescence of the abstract function Φ was one of the requirements of our formulation of an interpolation problem. We may ensure coalescence of Φ by the requirement $\mathcal{D}$ is irreducible and that Ψ is geometrically robust. Then Ψ is automatically constructible, hereditary and topologically robust. Since ω^* is a polynomial map (and hence geometrically robust), the abstract function $\Phi := \omega^* \circ \Psi$ becomes geometrically robust and therefore

topologically robust and hereditary, as required. In particular Φ is coalescent (see Section 2).
We discuss now in the case of univariate Lagrange interpolation two examples of the class $\mathcal{D}^*$ and the abstraction function ω^*.

Coefficients
In this example, $\mathcal{D}^*$ and ω^* constitute the representation of the interpolants by their coefficients in the context of classical Lagrange interpolation. To be concrete, consider the polynomial $p := 3X^2 + 2X + 1$ which we assume to belong to $\mathcal{O}^*$. The representation of p is the tuple $r := (3, 2, 1),\ \ r \in \mathcal{D}^*$ with $p := \omega^*(r)$ as in Figure 7.

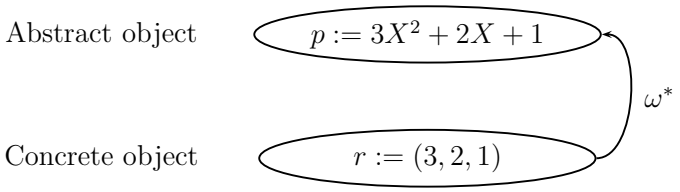


Figure 7: Representation of p by the coefficients r

Arithmetic circuits
For the notion of arithmetic circuit see Definition 11. According to [GHMS11], Section 3.1.2 we may fix a generic division–free arithmetic circuit β such that for any polynomial $p \in \mathcal{O}^*$ there exists a parameter instance $r' \in \mathbb{A}^M$ such that $\beta(r')$ becomes an arithmetic circuit, which evaluates the polynomial p. To be concrete, let β be the generic arithmetic circuit described by the following sequence of intermediate results:

$$(X,\ \ AX,\ \ AX + B,\ \ (AX + B) * X,\ \ ((AX + B) * X) + C) \tag{1}$$

where A, B, C are new indeterminates. Then, instantiating (A, B, C) in (1) to $r' := (3, 2, 1)$, we obtain an ordinary arithmetic circuit with intermediate results $(X,\ \ 3X,\ \ 3X + 2,\ \ (3X + 2) * X,\ \ ((3X + 2) * X) + 1)$ where the final result $(((3X + 2) * X) + 1)$ computes again the polynomial $p := 3X^2 + 2X + 1$. Selecting this final result of (1) we obtain an abstraction function $\omega^* : \mathbb{A}^3 \to \Pi_2^{(1)}$ with $\omega^*(r') = 3X^2 + 2X + 1$, as shown in Figure 8.

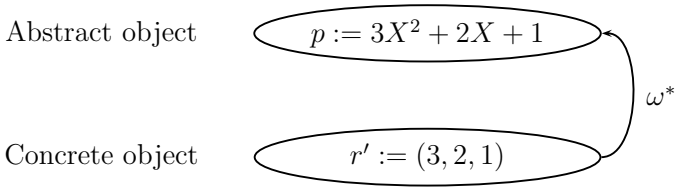


Figure 8: Representation of the final result p by the vector of scalars r' of an arithmetic circuit

4.3 A general interpolation model

We are now ready to describe a general model for Hermite–Lagrange interpolation by multivariate polynomials, which includes the quality attributes complexity and coalescence.

Let n, D, M and N be fixed natural numbers. Replacing in the previous discussion of Lagrange interpolation the quantity $(n+1)K$ or just K by the parameter N, we arrive to the following formulation.

Definition 15 (Interpolation problem) *A given Hermite–Lagrange interpolation problem is determined by:*

- *A suitable constructible subset $\mathcal{O}$ of the affine space $\mathbb{A}^N$, acting as abstract data type of interpolation data.*
- *A suitable surjective, topologically robust and hereditary map $\Phi : \mathcal{O} \to \mathcal{O}^*$, acting as abstract function which maps elements of $\mathcal{O}$ to elements of $\mathcal{O}^*$, where $\mathcal{O}^*$ is a suitable constructible subset of Π_D.*

Definition 16 (Interpolation algorithm) *Furthermore we say that a Hermite–Lagrange interpolation algorithm solving the given interpolation problem is determined by:*

- *A constructible subset $\mathcal{D}^*$ of the affine space $\mathbb{A}^M$, acting as a class which implements the abstract data type $\mathcal{O}^*$ of interpolants.*
- *A polynomial map $\omega^* : \mathcal{D}^* \to \mathcal{O}^*$ acting as abstraction function which maps concrete objects of the class $\mathcal{D}^*$, with abstract objects of the abstract data type $\mathcal{O}^*$.*
- *For $\mathcal{D} := \mathcal{O}$ hereditary map $\Psi : \mathcal{D} \to \mathcal{D}^*$, such that the diagram*

$$\begin{array}{ccc} \mathcal{O} & \xrightarrow[\Phi]{} & \mathcal{O}^* \\ \uparrow{\scriptstyle id} & & \uparrow{\scriptstyle \omega^*} \\ \mathcal{D} & \xrightarrow[\Psi]{} & \mathcal{D}^* \end{array} \tag{2}$$

commutes. The map Ψ represents the interpolation algorithm in the narrow sense.

4.4 Relationship between interpolation and geometric elimination

Here a word has to be said about the relationship between the notions of Lagrange interpolation problem and algorithm on one side, and quantifier elimination in the first order theory of algebraically closed fields of characteristic zero on the other.

Let notations and assumptions be as in Section 4.3. Then for any concrete object $d \in \mathcal{D}$ the following first order formula is valid:

$$(\exists d^* \in \mathcal{D}^*)\ \Phi(d) = \omega^*(d^*). \tag{3}$$

Eliminating now the existential quantifier block in (3) by any standard quantifier elimination procedure, we see that for any pair of points $(d, d^*) \in \mathcal{D} \times \mathcal{D}^*$ satisfying the condition $\Phi(d) = \omega^*(d^*)$ the entries of d^* may be described as algebraic functions of the entries of d (and, possibly, some extra parameters).

We are looking for solutions of the elimination problem expressed by (3) which satisfy certain quality attributes. We require that the entries of d^* depend rationally on the entries of d and moreover we require that this dependency is consistent with the hereditarity of Φ. This leads us to realize the solution of our elimination problem by a hereditary map $\Psi : \mathcal{D} \to \mathcal{D}^*$ which satisfies for any $d \in \mathcal{D}$ the condition

$$\Phi(d) = \omega^*(\Psi(d)) \tag{4}$$

which means that the diagram (2) commutes. A similar consideration may be applied to the requirement that Φ is coalescent.

When we look for a map $\Psi : \mathcal{D} \to \mathcal{D}^*$ satisfying the mentioned non–functional requirements, we solve a problem which has a background in quantifier elimination. Here, quantifier elimination is meant in the strict sense of first order logic, although the search for an interpolation algorithm cannot be simply reformulated as a task to eliminate quantifier blocks in a first order formula of the elementary theory of algebraically closed fields of characteristic zero.

In Algebraic Geometry it is normally accepted to call such problems elimination tasks. In this sense the meaning of elimination theory in Algebraic Geometry slightly differs from the strict use of the quantifier elimination in logics.

4.5 Examples of Univariate Hermite–Lagrange interpolation

In this section we illustrate the general interpolation model introduced in the previous section. We consider examples which come from standard univariate Lagrange and Hermite interpolation at fixed nodes.

4.5.1 Univariate Lagrange interpolation at fixed nodes

Let K be a given natural number. Fix an arbitrary point $\alpha := (\alpha_1, \ldots, \alpha_K) \in \mathbb{A}^K$ with $\alpha_i \neq \alpha_j$ for $1 \leq i < j \leq K$. The *univariate Lagrange interpolation problem at fixed nodes* $\alpha_1, \ldots, \alpha_K$ consists in finding, for any $y := (y_1, \ldots, y_K) \in \mathbb{A}^K$, the unique polynomial $f_{\alpha,y} \in \Pi^{(1)}_{K-1}$ satisfying the condition

$$f_{\alpha,y}(\alpha_j) = y_j \quad \text{for } 1 \leq j \leq K. \tag{5}$$

We are now going to formulate this problem by means of the notions interpolation problem and algorithm previously introduced.

Let $\mathcal{O}_\alpha$ be the constructible subset $\mathcal{O}_\alpha := \{\alpha_1\} \times \mathbb{A}^1 \times \cdots \times \{\alpha_K\} \times \mathbb{A}^1$ of $\mathbb{A}^{2K}$ acting as abstract data type of interpolation data. Let $\mathcal{O}^*_\alpha$ be the constructible set $\Pi^{(1)}_{K-1}$ acting as abstract data type of interpolants. Then the *univariate Lagrange interpolation problem at fixed nodes* $\alpha_1, \ldots, \alpha_K$ is represented by the map $\Phi_\alpha : \mathcal{O}_\alpha \to \mathcal{O}^*_\alpha$ which associates to each interpolation datum $d := (\alpha_1, y_1, \ldots, \alpha_K, y_K)$ of $\mathcal{O}_\alpha$ the unique polynomial $f_d := f_{\alpha,y}$ of $\mathcal{O}^*_\alpha$ determined by condition (5). Since Φ_α is a polynomial map, we conclude that $\mathcal{O}_\alpha$ and Φ_α determine a Lagrange interpolation problem in the sense of Definition 15.

Let $\mathcal{D}^*_\alpha := \mathbb{A}^K$ be the class which implements the elements of $\mathcal{O}^*_\alpha := \Pi^{(1)}_{K-1}$ by their dense representation. This determines an abstraction function ω^* which maps $\mathcal{D}^*_\alpha$ into

$\mathcal{O}_\alpha^*$. Let $\mathcal{D}_\alpha$ be the same constructible set as $\mathcal{O}_\alpha$ and let $\mathcal{D}_\alpha$ act as a class. Then we know that for every interpolation datum $d \in \mathcal{D}_\alpha$, the dense representation of the interpolant $f_d \in \mathcal{O}_\alpha^*$ is given by a routine $\Psi_\alpha : \mathcal{D}_\alpha \to \mathcal{D}_\alpha^*$. Since Ψ_α, ω^* and $\mathcal{D}_\alpha^*$ determine a Lagrange interpolation algorithm in the sense of Definition 16, the diagram

$$\begin{array}{ccc} \mathcal{O}_\alpha & \xrightarrow[\Phi_\alpha]{} & \mathcal{O}_\alpha^* \\ \uparrow{\scriptstyle id} & & \uparrow{\scriptstyle \omega^*} \\ \mathcal{D}_\alpha & \xrightarrow[\Psi_\alpha]{} & \mathcal{D}_\alpha^* \end{array} \tag{6}$$

commutes.

Routine Ψ_α is an algorithm that calculates from $d \in \mathcal{D}_\alpha$ the dense representation of the interpolant $f_d := \Phi_\alpha(d)$. We shall make this now more precise. Let us fix as before an arbitrary point $(\alpha_1, \ldots, \alpha_K) \in \mathbb{A}^K$, $\alpha_i \neq \alpha_j$ for $1 \leq i \neq j \leq K$ and an interpolation datum $d := ((\alpha_1, y_1), \ldots, (\alpha_K, y_k))$ of $\mathcal{D}_\alpha := \mathcal{O}_\alpha$. Consider the interpolant $f_d := \Phi_\alpha(d)$. Then there exists a unique point $a := (a_0, \ldots, a_{K-1}) \in \mathbb{A}^K$ such that $f_d = a_0 + a_1 X + \cdots + a_{K-1} X^{K-1}$ holds. We call the expression $a_0 + a_1 X + \cdots + a_{K-1} X^{K-1}$ the Monomial Form of the interpolant f_d.

The Monomial Form of an interpolating polynomial

For a given interpolation datum $d \in \mathcal{O}_\alpha$ with $d = ((\alpha_1, y_1), \ldots, (\alpha_K, y_K))$, the Monomial Form will not be the only expression we shall use to describe the interpolant f_d. Other expressions will be the Lagrange and the Newton Form of the interpolant. These different forms are due to the fact that the ring Π_{K-1} is a vector space which may be described by different canonical bases. Each one of these bases determines a particular form for the representation of the interpolant. For example, the Monomial Form of the interpolants is determined by the $\mathbb{C}$–vector space basis

$$B_1 := \left\{1, X, X^2, \ldots, X^{K-1}\right\}$$

of $\Pi_{K-1}^{(1)}$.

In this case, we have the routine $\Psi_\alpha : \mathcal{D}_\alpha \to \mathcal{D}_\alpha^*$ which maps $d \in \mathcal{D}_\alpha$ into $\mathcal{D}_\alpha^*$ becomes

$$\Psi_\alpha(d) \;:=\; V_\alpha^{-1} y^t$$

where $V_\alpha := (\alpha_i^{j-1})_{1 \leq i,j \leq K} \in \mathbb{A}^{K \times K}$ is the Vandermonde matrix associated to α, and $y := (y_1, \ldots, y_K)$ is the vector of values determined by d.

The Monomial Form determines an abstraction function ω^* which, for $a := (a_0, \ldots, a_{K-1}) \in \mathcal{D}_\alpha^*$, looks as follows:

$$\omega^*(a) \;:=\; \sum_{i=0}^{K-1} a_i X^i.$$

The Lagrange Form of an interpolating polynomial

For $1 \leq i \leq K$ let $L_i := \displaystyle\prod_{\substack{j=1 \\ j \neq i}}^{K} \frac{X - \alpha_j}{\alpha_i - \alpha_j}$ and observe that $L_1, \ldots, L_K$ form a $\mathbb{C}$–vector space

basis of $\Pi^{(1)}_{K-1}$. Let $d \in \mathcal{O}_\alpha$ with $d = ((\alpha_1, y_1), \ldots, (\alpha_K, y_K))$. The Lagrange Form of the interpolant f_d is determined by the $\mathbb{C}$–vector space basis

$$B_2 := \{L_1 \ldots, L_K\}$$

of $\Pi^{(1)}_{K-1}$.

In this case, we have $\mathcal{D}^*_\alpha := \mathbb{A}^K$ and $f_d = y_1 L_1 + \cdots + y_K L_K$. The expression $y_1 L_1 + \cdots + y_K L_K$ is called the Lagrange Form of the interpolant f_d. The routine $\Psi_\alpha : \mathcal{D}_\alpha \to \mathcal{D}^*_\alpha$ becomes the simple map

$$\Psi_\alpha(d) \ := (y_1, \ldots, y_K).$$

The Lagrange Form determines an abstraction function ω^* which for $y := (y_1, \ldots, y_K) \in \mathcal{D}^*_\alpha$ looks as follows:

$$\omega^*(y) \ := \ \sum_{i=1}^{K} y_i L_i.$$

The Newton Form of the interpolating polynomial
Observe that

$$B_3 := \left\{ 1, (X - \alpha_1), (X - \alpha_1)(X - \alpha_2), \ldots, \prod_{j=1}^{K-1} (X - \alpha_j) \right\}$$

forms a $\mathbb{C}$–vector space basis of $\Pi^{(1)}_{K-1}$.

There exists a unique point $a := (a_0, \ldots, a_{K-1}) \in \mathbb{A}^K$ such that $f_d = a_0 + a_1(X - \alpha_1) + \cdots + a_K \prod_{j=1}^{K-1} (X - \alpha_j)$ holds. We call the expression $a_0 + a_1(X - \alpha_1) + \cdots + a_K \prod_{j=1}^{K-1} (X - \alpha_j)$ the Newton Form of f_d. Let $\mathcal{D}^*_\alpha := \mathbb{A}^K$. The routine $\Psi_\alpha : \mathcal{D}_\alpha \to \mathcal{D}^*_\alpha$ becomes the map

$$\Psi_\alpha(d) \ := (f[\alpha_1], \ldots, f[\alpha_1, \ldots, \alpha_K])$$

where for $1 \leq i \leq K$ and $0 \leq k \leq K - i$ the *divided difference* $f[\alpha_i, \ldots, \alpha_{i+k}]$ is recursively defined by

$$\begin{aligned} f[\alpha_i] &= y_i \\ f[\alpha_i, \ldots, \alpha_{i+k}] &= \frac{f[\alpha_{i+1}, \ldots, \alpha_{i+k}] - f[\alpha_i, \ldots, \alpha_{i+k-1}]}{\alpha_{i+k} - \alpha_i}. \end{aligned} \tag{7}$$

The Newton Form determines an abstraction function ω^* which for $a := (a_0, \ldots, a_{K-1}) \in \mathcal{D}^*_\alpha$ looks as follows:

$$\omega^*(a) \ := \ \sum_{i=0}^{K-1} a_i \prod_{j=1}^{i} (X - \alpha_j)$$

Therefore each basis of the $\mathbb{C}$–vector space bases B_1, B_2, B_3 of $\Pi^{(1)}_{K-1}$ determine their own interpolation algorithm given by Ψ_α and ω^*. The correctness proofs of these algorithms may be found in Section B.1.

4.6 Trade-off results: complexity and coalescence

In this section we discuss the trade–off between the quality attributes computational complexity and coalescence. This trade–off constitutes an interpretation in terms of Software Engineering of the lower bound complexity results of [GHMS11]. We shall focus our attention on the formulation of the concept of coalescence by means of the notion of geometrical robustness introduced in Section 2.3.

In Section 4.2.3 we discussed the informal concept of coalescence in Interpolation Theory and argued that it was adequately modeled by the notion of geometric robustness. In the sequel, we consider the general interpolation model introduced in Section 4.3 and we interpret geometrical robustness as a (dichotomic) quality attribute.

Polynomials without restrictions

Let assumptions and notations be as in Section 4.3 and let $\mathcal{O}$, $\mathcal{O}^*$, $\Phi : \mathcal{O} \to \mathcal{O}^*$ with $\mathcal{O} \subset \mathbb{A}^N$ determine an interpolation problem according to Definition 15. Furthermore, let $\mathcal{D} := \mathcal{O}$, $\mathcal{D}^*$ with $\mathcal{D}^* \subset \mathbb{A}^M$ and $\omega^* : \mathcal{D}^* \to \mathcal{O}^*$, $\Psi : \mathcal{D} \to \mathcal{D}^*$ determine an interpolation algorithm according to Definition 16.

The given interpolation problem and algorithm become linked by the following commutative diagram:

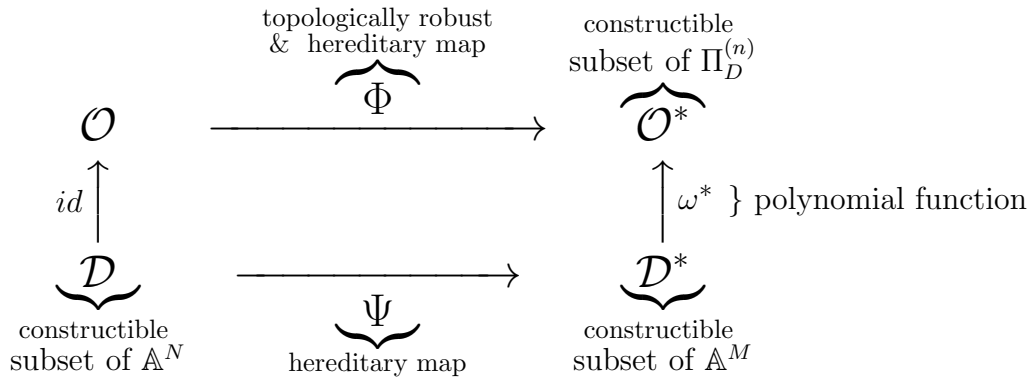


Figure 9: General interpolation model

According to [GHMS11], Section 5 the following estimate is consequence of the software architecture described by Figure 9:

Suppose that the given interpolation algorithm satisfies the conditions listed in Figure 9 an that $\mathcal{D} = \mathcal{O}$ is Zariski dense in $\mathbb{A}^N$, then

$$M \geq K = \frac{N}{(n+1)}. \tag{8}$$

This estimate means that in the case of the multivariate interpolation problem above with $\mathcal{O}$ Zariski dense in $\mathbb{A}^N$ the complexity of the interpolation algorithm determined by Ψ and ω^* is at least $N/(n+1)$. We shall refer to $\Phi : \mathcal{O} \to \mathcal{O}^*$ with $\mathcal{O}$ Zariski dense in $\mathbb{A}^N$ as a *generic* interpolation problem. Such problems represent in the literature the standard case of Lagrange interpolation.

We exhibit now an example of a very special family $\mathcal{O}^*$ of univariate polynomials which cannot be more efficiently implemented than by classical Lagrange interpolation.

Polynomials that are easy to evaluate

Let K and M be natural numbers with $K \geq 2$ and let $N := 2K$, $D := K-1$. We consider the following set of univariate interpolants, namely

$$\mathcal{O}^* := \{F(X,t) : t \in \mathbb{C}\} \;\; where \;\; F(X,T) := (T^{D+1}-1)\sum_{i=0}^{D} T^i X^i$$

Observe that each polynomial $f \in \mathcal{O}^*$ may be evaluated using $O(logK)$ arithmetic operations, i.e. f is easy to evaluate. We consider now the following set of of interpolation data, namely

$$\mathcal{O} := \left\{((x_1 y_1)\ldots(x_K y_K)) \in \mathbb{A}^N : \exists t \in \mathbb{C} \;/\; F(x_i,t) = y_i \;\; \forall i, \;\; x_i \neq x_j \;\; \forall i,j\right\}$$

Observe that $\mathcal{O}$ is a constructible subset of $\mathbb{A}^N$ and that the map $\Phi : \mathcal{O} \to \mathcal{O}^*$, which assigns to each interpolation datum $d \in \mathcal{O}$ the polynomial f_d of Section 4.5.1, is a polynomial map. According to [GHMS11] a polynomial map is geometrically robust and therefore topologically robust and hereditary. Thus, ($\mathcal{O}$ and Φ) determine an interpolation problem according to Definition 15.

Let be given an arbitrary interpolation algorithm, determined by $\mathcal{D} := \mathcal{O}$, $\mathcal{D}^*$ with $\mathcal{D}^* \subset \mathbb{A}^M$ constructible, $\omega^* : \mathcal{D}^* \to \mathcal{O}^*$ and $\Psi : \mathcal{D} \to \mathcal{D}^*$, according to Definition 16. We require now that the map Ψ is *geometrically robust* (and not just hereditary as in the general model depicted in Figure 9).

We obtain now the following commutative diagram:

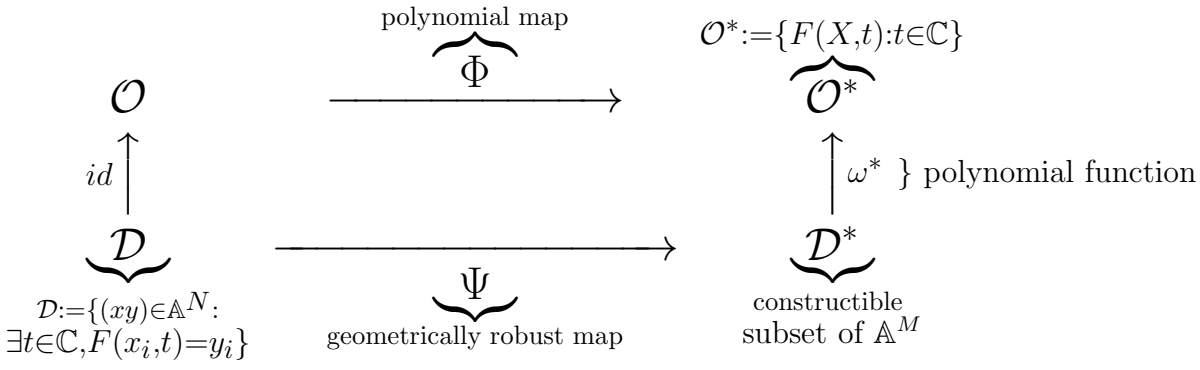


Figure 10: Interpolation model with polynomials easy to evaluate

According to [GHMS11], Section 5 the following estimate is a consequence of the architecture described by Figure 10:

$$M \geq K = \frac{N}{2} \tag{9}$$

This result means that if an interpolation algorithm satisfies the conditions required by the architecture of Figure 10, then the complexity of the algorithm represented by Ψ and ω^* is at least $K = \frac{N}{2}$.

A somewhat different lower bound result is the main result of [GHMS11], where the set of interpolants is the set of n–variate polynomials which can be evaluated by a division–free arithmetic circuit of non–scalar size at most L. In this case the lower bound becomes exponential in L (see Section 6.5.1 and Example 21 for details).

4.7 Discussion of the mathematical interpolation model

In this section we discuss some computational aspects concerning the interpolation model introduced in Section 4.3. We discuss four topics, namely the memory consumed by a computation, libraries, software architecture and the mathematical structure of our interpolation model.

Memory

Our algorithmic model is unstructured and leaves unspecified how ongoing computations are performed and how intermediate results are handled. It even admits maps Ψ which are difficult to identify with algorithms in the standard sense. Therefore our model is unsuited for the treatment of memory issues. It is not clear when such an algorithm saves the intermediate results in order to avoid unnecessary computing.

Libraries

The libraries used in real world computation raise the following question. Let $l_1, \ldots, l_k$ be routines whose computational complexity is optimal. Then, the composition

$$l_1 \circ l_2 \circ \cdots \circ l_k$$

is not necessarily optimal. In this sense there may exist thinkable libraries with optimal routines, but the composition of these routines are not necessarily complexity–optimal. We ask when the complexity of a composition of routines is the sum of the complexities of its components or more generally, how the complexity of a single component contributes to the complexity of the final algorithm. In case that this component is an interpolation algorithm belonging to a library, an answer to this question would be of high relevance in software design.

Software Architecture

According to [Cor01] a data structure implements a (finite) set which may be modified by insert and delete operations. Such a data structure has a dynamic aspect in the sense that the data structure may be expanded.

On the other hand, data structures in the sense of [Cor01] are described in this thesis by means of classes. Thus, our classes should also include a dynamic aspect. However, we model a class as a constructible subset of an affine space $\mathbb{A}^N$ where N is a fixed natural number. How does this instantiation of the discrete parameter N affect the dynamic aspect of a class? Doubtless, an answer to this question may increase the relevance of our model for research in Software Engineering. Instead of dealing with vectors of previously

fixed length, we should be able to deal with dynamic vectors. In Section 6 we are going to describe a model for arithmetic circuit based scientific computing which overcomes this difficulty.

Interpolation model

The mathematical structure of our interpolation model raises the following questions.

(*i*) *Side effects:* Let β_1 and β_2 be two circuits implementing a polynomial and let $\mathcal{A}$ be a circuit transformation such that $\mathcal{A}(\beta_1) = \beta_2$ holds. If the circuit β_1 is modified because one of its nodes is redundant, we want to do the same operation in β_2 taking care of side effects. How can we guarantee that side effects of the modification of β_1 produce the same side effects in a possible modification of β_2? This question will be discussed in Section 6 and will lead us to the notion of "well behavedness under reductions" for $\mathcal{A}$.

(*ii*) *Dependencies:* Consider the following commutative diagram representing as before a software architecture, with $\mathcal{O}$ and $\mathcal{O}^*$ being abstract data types, $\mathcal{D}$ and $\mathcal{D}^*$ being classes and ω and ω^* being abstraction functions:

$$\begin{array}{ccc} \mathcal{O} & \xrightarrow[\Phi]{} & \mathcal{O}^* \\ \uparrow{\scriptstyle \omega} & & \uparrow{\scriptstyle \omega^*} \\ \mathcal{D} & \xrightarrow[\Psi]{} & \mathcal{D}^* \end{array} \tag{10}$$

Following Diagram (10) we may interpret $\mathcal{D}$ as a representation of $\mathcal{O}$ and $\mathcal{D}^*$ as a representation of $\mathcal{O}^*$. We may also interpret $\mathcal{D}^*$ as generated from $\mathcal{D}$ by means of Ψ. However, we cannot conclude without further assumptions that $\mathcal{D}^*$ depends directly on $\mathcal{O}$.

Suppose now that Diagram (10) represents the interpolation model introduced in Section 4.3. In this case $\mathcal{D}^*$ depends clearly on $\mathcal{O}$ since we have $\mathcal{D} = \mathcal{O}$ and ω is the identity function.

Consider now another example, interpreting Diagram (10) as a model for the following computational problem. Let D be a natural number and consider the task of computing the leading coefficient of a given univariate polynomial of degree at most D. This problem may be expressed as a tuple $(\mathcal{O}, \Phi)$ where $\mathcal{O}$ is the $\mathbb{C}$–vector space of univariate polynomials of degree at most D and Φ maps any polynomial of $\mathcal{O}$ to its leading coefficient in $\mathcal{O}^* := \mathbb{C}$. The standard algorithm which solves this problem may be represented by $\mathcal{D}$, Ψ and ω, where $\mathcal{D}$ is the affine space $\mathbb{A}^{D+1}$, the abstraction function $\omega : \mathcal{D} \to \mathcal{O}$ is given by the dense representation of the elements of $\mathcal{O}$, $\mathcal{D}^* := \mathcal{O}^*$, $\omega^* : \mathcal{D}^* \to \mathcal{O}^*$ is the identity function and Ψ maps each polynomial of $\mathcal{O}$ to its leading coefficient.

Let $d \in \mathcal{D}$ be a coefficient instance. How do we select the leading coefficient from d? Does the leading coefficient occupy the first, second or another position in d? This information is intrinsic to the given polynomial and does not depend a priori on its representation. Therefore we conclude that $\mathcal{D}^*$ depends directly on $\mathcal{O}$.

Since $\mathcal{O}$ is an abstract data type and $\mathcal{D}$ is the class which implements $\mathcal{O}$, the meaning of the elements in $\mathcal{D}$ is given through the abstraction function. Thus, any routine which processes the elements in $\mathcal{D}$ has to take into account the abstract data type $\mathcal{O}$. We shall come back to this question in Section 4.9.

4.8 Polymorphism in the general interpolation model

The notion of polymorphism appears originally in functional programming in the form of polymorphic functions and polymorphic variables. The notion of polymorphism was later adopted by object oriented programming where several new forms of polymorphism such as generics, overloading and overriding appeared as a consequence of inheritance.

In this section we give an overview over different forms of polymorphism. The aim is to analyse the presence of polymorphism in the interpolation model presented in Section 4.3.

4.8.1 The origin of polymorphism

According to [Bud02] the notion of polymorphism appears first in the functional programming world. The paradigm of functional programming has its roots in lambda calculus[5], where computational problems become described by functions between sets. In [Pie02] lambda calculus is studied as the basis of functional programming. The notion of type[6] is crucial in this context. This notion gave rise to the first form of polymorphism, namely the polymorphic function (also called parametric polymorphism in [Pie02] and polymorphic variable in [Bud02]). We speak about a polymorphic function when a given function may be applied to more than one type. Usually functions are associated with expressions of a fixed type. For example, $Integer$ is a type and the identity function of integers has type $Integer \to Integer$. Polymorphic functions use type variables instead of type expressions in their signature. A type variable is a kind of variable used as a mechanism for abstracting types. Moreover, a type variable allows the polymorphic function to be applied to an argument of more than one type. For example, the identity function, when applied to elements of any domain (type) has the type $T \to T$, where T is a type variable which can be replaced by any type expression. Another example given in [ASU86] is the pointer operator &. This operator of the programming language C has type $T \to pointer\ to\ T$.

Polymorphic functions are realized by adding to the set of type expressions also type variables. This view is reflected in the polymorphic lambda calculus.

4.8.2 Polymorphism as a property of the implementation

According to [ASU86] polymorphic functions facilitate the implementation of algorithms that manipulate data structures, regardless of the types of the elements which constitute the given data structure. The example proposed in [ASU86] is the function that determines the length of a list without knowing the types of the elements of the list. This function has type $list[T] \to Integer$, where T is a type variable. This example reveals that polymorphism is a feature of the implementation, i.e. a property of the classes and the concrete objects used in the implementation. Therefore we may ask whether polymorphism appears as a feature of implementations of Hermite–Lagrange interpolation problems and algorithms (recall that the interpolation model introduced in Section 4.3 consist of two parts namely the interpolation problem and the algorithm that solves the given problem).

There exist always various algorithms that compute the required interpolants. The main difference between these algorithms is the form how the interpolant is returned,

[5]Lambda calculus is a formal system introduced by Alonzo Church in the 1930s.

[6]According to [Dat09] a type is a named set of values and it is characterized by a set of operations [HKB93]. This definition is similar to the definition of Abstract Data Type.

e.g. by its coefficient vector or by an arithmetic circuit. Each such representation of the interpolants determines a different algorithm which is formalized in the interpolation model by a hereditary and topologically robust map $\Psi : \mathcal{D} \to \mathcal{D}^*$ and a polynomial map ω^* from $\mathcal{D}^*$ onto the interpolants. A polymorphic aspect appears when we think all possible maps $\Psi : \mathcal{D} \to \mathcal{D}^*$ as particular instances of one routine. Each such instance is characterized by the class $\mathcal{D}^*$. In terms of [ASU86], if the corresponding type is written as $\Psi : \mathcal{D} \to \mathcal{D}^*$, the expression $\mathcal{D}^*$ acts as a type variable.

4.8.3 Polymorphism in object oriented programming

In object oriented programming, classes which depend on type variables are called Generics (or Templates in C++). Functions which act on Generics are polymorphic. Another kind of polymorphism emerges from an important feature of object oriented programming: inheritance.

In this section we introduce the notion of inheritance and discuss the forms of polymorphism produced by inheritance. Finally we discuss the relation between inheritance and the notion of representation in case of our interpolation model.

Inheritance

According to [Mey88] inheritance is a relation between classes which has two meanings: from the module perspective, inheritance is a key reusability technique. From the type perspective, inheritance represents the "is–a" relation, which on its turn constitutes a set inclusion used to describe a hierarchy of types.

For example, Figure 4.8.3 illustrates a hierarchy of types where a group is a monoid but a monoid is not a group. The class "Monoid" is an implementation of the algebraic structure monoid and the class "Group" is an implementation of the algebraic structure group. The arrow between the classes "Monoid" and "Group" says that the class "Group" inherits for example from the class "Monoid" the multiplication and its associativity. The class "Group" is also called a child class of the parent class "Monoid".

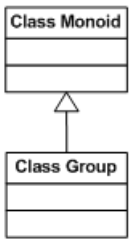


Figure 11: Example of inheritance

Inheritance and overriding polymorphism

The hierarchy defined by inheritance allows a form of polymorphism called overriding. We speak about overriding when we have to deal with two or more different versions of the

same routine. Typically this situation is produced by inheritance. Figure 12 illustrates an example of overriding in the context of a graphic library.

Consider a class "Polygon" which implements all polygons by means of a list of vertices. One of the routines in this class is *perimeter* which returns the perimeter of the polygon under consideration. This routine is implemented as a cycle that sums successive distances between adjacent vertices. The class "Rectangle" is a subclass of the class "Polygon". Therefore, the class "Rectangle" inherits all the routines implemented in the class "Polygon". However, the routine *perimeter* may be redefined in the class "Rectangle" as twice the sum of the two side lengths. Thus, we have two implementations of the same routine, a first implementation in the class "Polygon" and a more efficient one in the class "Rectangle".

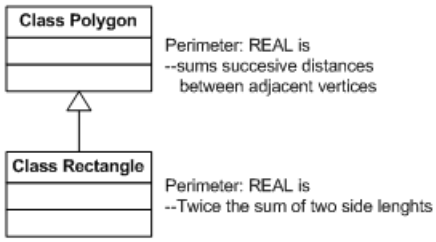


Figure 12: Example of overriding polymorphism

Another type of polymorphism in object oriented programming is overloading. This form is similar to overriding, it appears when the routine of a class has several implementations belonging to the same class. All the implementations have the same routine name. The different routines become distinguished by the signature, i.e. the types and number of parameters used.

Inheritance and the notion of representation in the interpolation model

In this section we discuss the relation between the notion of representation in our model and the notion of inheritance. Figure 13 illustrates this question by means of the highlighted arrows. The first arrow indicates that $\mathcal{D}^*$ is the representation of $\mathcal{O}^*$ and the second arrow indicates that the class "Group" inherits from the class "Monoid".

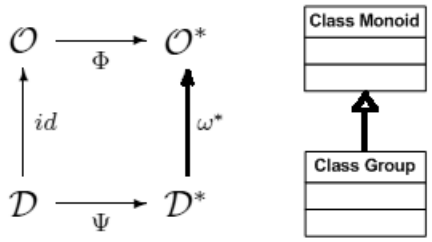


Figure 13: representation vs. inheritance

We are going to explain that an inheritance relation between $\mathcal{D}^*$ and $\mathcal{O}$ is not possible in our model and therefore, the notions of representation and inheritance are different. Hence, the notion of polymorphism as a consequence of inheritance is not compatible with the notion of representation.

Suppose that there is a relation of inheritance between $\mathcal{D}^*$ and $\mathcal{O}^*$. If $\mathcal{D}^*$ inherits from $\mathcal{O}^*$, all the operations available in $\mathcal{O}^*$ are also available in $\mathcal{D}^*$. However, $\mathcal{O}^*$ is a subset of a polynomial ring and $\mathcal{D}^*$ is constructible subset of an affine space. A polynomial ring is as algebraic structure richer than an affine space. Thus $\mathcal{D}^*$ does not share operations and properties (axioms) with $\mathcal{O}^*$.

On the other hand, if $\mathcal{O}^*$ inherits from $\mathcal{D}^*$, all the operations available in $\mathcal{D}^*$ are available in $\mathcal{O}^*$. This last situation is not possible since $\mathcal{O}^*$ is part of the specification of the problem of interpolation and $\mathcal{D}^*$ is part of the implementation. Specification and implementation are separated levels of abstraction and inheritance is a static feature of the implementation.

4.9 A terminology dictionary

The mathematical and complexity theoretic aspects of this thesis were treated in [CGH$^+$03] and [GHMS11] where the authors introduced their own particular terminology. In this section we shall compare this terminology with the standard terminology in Software Engineering, following the presentation of [Mey00]. The following Table 2 gives an overview.

Id	**Software Engineering terminology**	**Terminology in [CGH$^+$03] and [GHMS11]**
1	Abstract Data Type	Object Class
2	Abstract Object	Mathematical Object or Member of an Object Class
3	Class	Data Structure
4	Concrete Object or just Object	Code or Member of a Data Structure
5	Abstraction Function	Encoding
6	Axioms of an Abstract Data Type	Constructible Constraints of an Object Class
7	Implementation Invariant	Constructible Constraints of a Data Structure
8	Function of an Abstract Data Type	Identity and Value Question
9	Routine	Algorithm

Table 2: A terminology dictionary.

Here we remark that the correspondences 6 and 8 are not perfect, since the terminology in [CGH$^+$03] and [GHMS11] refers to special cases of axioms and functions.

Since the terminology of [CGH$^+$03] and [GHMS11] was introduced in a specific context based on particular ad–hoc definitions and the terminology in Software Engineering is considerably more general, we have to justify our dictionary. This is done by specializing the terminology of Software Engineering to the given specific context. In this way we may compare both terminologies. We shall refer to them as [CGH$^+$03][GHMS11] terms and [Mey00] terms.

Our comparison follows a scheme of argumentation which looks as follows:

[Mey00] terms	t_1
[CGH$^+$03][GHMS11] terms	t_2
Definition of t_1:	Definition of t_1 is given.
Definition of t_2:	Definition of t_2 is given.
Argument:	Verification that t_2 may be replaced by t_1 in the given context.

Table 3: Scheme of argumentation

In Section 4.3 we have introduced a general data model for Hermite–Lagrange interpolation. In this case our comparison is based on the following set up.

Let $(\mathcal{O},\ \Phi : \mathcal{O} \to \mathcal{O}^*)$ be an interpolation problem according to Definition 15. Furthermore, let $(\mathcal{D}^*,\ \Psi : \mathcal{D} \to \mathcal{D}^*,\ \omega^* : \mathcal{D}^* \to \mathcal{O}^*)$ be an interpolation algorithm according to Definition 16 solving the given interpolation problem. Thus, our software architecture for interpolation is described by the following commutative diagram:

$$\begin{array}{ccc} \mathcal{O} & \xrightarrow[\Phi]{} & \mathcal{O}^* \\ \uparrow{\scriptstyle id} & & \uparrow{\scriptstyle \omega^*} \\ \mathcal{D} & \xrightarrow[\Psi]{} & \mathcal{D}^* \end{array} \tag{11}$$

We shall refer to this general software architecture in our subsequent comparisons.

Comparison 1

[Mey00] terms :	Abstract Data Type
[CGH$^+$03][GHMS11] terms :	Object Class

Definition (Abstract Data Type) *Following Definition 8, an Abstract Data Type is a set of elements defined by a list of operations applicable to these elements. An Abstract Data Type is formalized by an Abstract Data Type Specification which describes by available services (functions) and formal properties (axioms) a set of elements.*

Definition 17 (Object Class) *According to [CGH$^+$03] and [GHMS11] an Object Class consists of a constructible subset of a suitable affine space whose elements represents mathematical objects of any finitary nature, e.g. polynomials over a given field. Following the context, these mathematical objects become considered as equipped with mathematical functions whose values satisfy certain axioms. In the case of interpolation, $\mathcal{O}$ and $\mathcal{O}^*$ of Diagram (11) are Object Classes. The Object Class $\mathcal{O}$ is typically a constructible set of interpolation data (given by nodes and values) and the elements of $\mathcal{O}^*$ are the interpolants, e.g. polynomials in $X := (X_0, \ldots, X_n)$ over $\mathbb{C}$ contained in a finite–dimensional $\mathbb{C}$–linear subspace of $\mathbb{C}[X]$. Here we require that $\mathcal{O}^*$ forms a constructible subset of this linear subspace.*

Argument The definition of Abstract Data Type has two parts:

1. An Abstract Data Type is a set of elements.
2. Through its specification an Abstract Data Type is equipped with available services which satisfy formal conditions, called axioms.

For the sake of explicitness suppose that our Object Class is a constructible subset of a finite dimensional $\mathbb{C}$–vector space of the polynomial ring $\mathbb{C}[X]$. Obviously this is a set of elements which satisfy the first condition of the definition of the notion Abstract Data Type. On the other hand, the polynomial ring $\mathbb{C}[X]$ as algebraic structure includes certain arithmetic operations (services) with the elements of $\mathbb{C}[X]$. Those arithmetic operations satisfy certain axioms, e.g. the commutative and associative law. In this sense the notion Object Class satisfies also the second condition of the definition of the notion Abstract Data Type. Thus, Abstract Data Type captures the meaning of Object Class in the given context. This allows us to replace the term Object Class by the term Abstract Data Type.

Comparison 2

[Mey00] terms :	Abstract Object
[CGH$^+$03][GHMS11] terms :	Mathematical Object or Member of an Object Class

Definition 18 (Abstract Object) *According to Meyer [Mey00] an Abstract Object is an instance of an Abstract Data Type.*

Definition 19 (Mathematical Object) *According to [CGH$^+$03] and [GHMS11] a Mathematical Object is a member of an Object Class.*

Argument According to [Mey00] an Abstract Object is an instance of an Abstract Data Type. Since the notion of Abstract Data Type may be replaced by the notion of Object Class in the given context, we conclude that a Mathematical Object is a member an Abstract Data Type. Thus, the notion of Abstract Object in [Mey00] captures the meaning of Mathematical Object in [CGH$^+$03] and [GHMS11]. Therefore, we may replace the term Mathematical Object by the term Abstract Object.

Comparison 3

[Mey00] terms :	Class
[CGH$^+$03][GHMS11] terms :	Data Structure

Definition (Class) *Following Definition 9, a Class is a software element describing an Abstract Data Type and its partial or total implementation. A Class is described by a list of features (attributes and Routines). The features constitute the basis of the interaction of the Class with the rest of the software.*

Definition 20 (Data Structure) *A Data Structure is a representation of an Object Class.*

For the sake of explicitness let us consider the Data Structure $\mathcal{D}^*$ and the Object Class $\mathcal{O}^*$ in diagram (11). Suppose that $\mathcal{O}^*$ is a constructible subset of polynomials contained in a finite dimensional $\mathbb{C}$–vector subspace of $\mathbb{C}[X]$. We consider $\mathcal{O}^*$ as an Abstract Data Type.

Argument The elements belonging to $\mathcal{D}^*$ represent the interpolants belonging to the Abstract Data Type $\mathcal{O}^*$. Since the elements of $\mathcal{O}^*$ are polynomials of bounded degree and number of variables, we may think them represented by their coefficients or by arithmetic circuits or whatever else. This is just the meaning of an implementation. Since $\mathcal{D}^*$ represents such an implementation of $\mathcal{O}^*$, we may conclude that $\mathcal{D}^*$ is a Class in the sense of [Mey00] because the Data Structure $\mathcal{D}^*$ implements the Abstract Data Type $\mathcal{O}^*$. Thus, the notion of a Class in [Mey00] captures the meaning of Data Structure in [CGH$^+$03] and [GHMS11]. This allows us to replace the term Data Structure by the term Class.

Comparison 4

[Mey00] terms :	Concrete Object or just Object
[CGH$^+$03][GHMS11] terms :	Code or Member of a Data Structure

Definition 21 (Object) *According to [Mey00] an Object is an instance of a Class. An Object is a run time element, whereas a Class is an element of the program text. Figure 14 illustrates that an Object has a state which is determined by certain fields that acquire specific values at run time. An Object may also contain operations, which are the runtime version of the routines in the Class.*

Definition 22 (Code) *Let $\mathcal{D}^*$ be a Data Structure which encodes the Object Class $\mathcal{O}^*$. According to [CGH$^+$03] an element D of $\mathcal{D}^*$ is called a code of an element O of $\mathcal{O}^*$.*

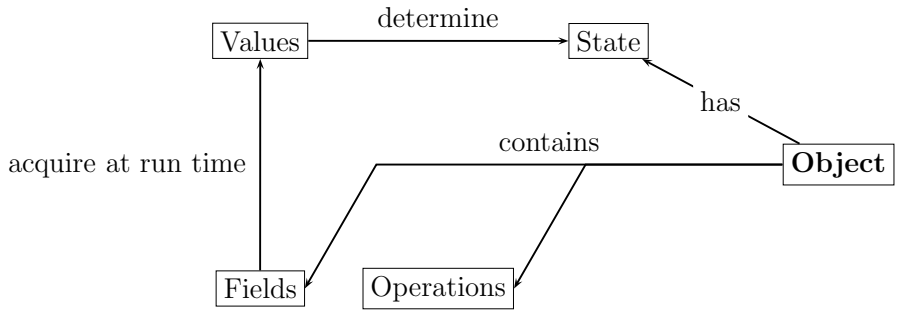


Figure 14: Parts of a Concrete Object

Argument We have seen that the Data Structure $\mathcal{D}^*$ is a Class in terms of [Mey00]. Since a Code is an instance of a Data Structure, the codes of $\mathcal{D}^*$ are instances of a Class. Thus, the notion of a Concrete Object in [Mey00] captures the meaning of Code. This allows us to replace the term Code by the term Concrete Object.

Comparison 5

[Mey00] terms :	Abstraction Function
[CGH$^+$03][GHMS11] terms :	Encoding

Definition 23 (Abstraction Function) *According to [Mey00] a Class describes a possible representation of an Abstract Data Type; the correspondence between Class and Abstract Data Type is called Abstraction Function. When we apply a given Abstraction Function to any instance of the Class, or Concrete Object, we obtain an instance of the Abstract Data Type, or Abstract Object.*

Definition 24 (Encoding) *Let $\mathcal{D}^*$ be a Data Structure and $\mathcal{O}^*$ an Object Class. In [CGH$^+$03] and [GHMS11] an Encoding of $\mathcal{O}^*$ by $\mathcal{D}^*$ is a constructible (elementary definable) map $\omega^* : \mathcal{D}^* \to \mathcal{O}^*$ which is continuous with respect to the Zariski topologies of $\mathcal{D}^*$ and $\mathcal{O}^*$. Frequently ω^* is supposed to be a polynomial map. In the latter case ω^* is called holomorphic.*

Argument We have seen that:

1. The term Data Structure may be replaced by the term Class.
2. The term Object Class may be replaced by the term Abstract Data Type.

The Encoding ω^* maps the Data Structure $\mathcal{D}^*$ to the Object Class $\mathcal{O}^*$. In other words, the Encoding ω^* maps a Class to an Abstract Data Type, i.e. for any Concrete Object D of $\mathcal{D}^*$ the image $\omega^*(D)$ is an Abstract Object that represents D. Therefore ω^* constitutes a correspondence between the Class $\mathcal{D}^*$ and the Abstract Data Type $\mathcal{O}^*$. Hence ω^* is by definition an Abstraction Function. Thus, the notion of Abstraction Function captures the meaning of term Encoding. We may therefore replace the term Encoding by the term Abstraction Function.

Comparison 6

[Mey00] terms :	Axioms of Abstract Data Type
[CGH$^+$03][GHMS11] terms :	Constructible Constraints of Object Class

Definition 25 (Axioms of Abstract Data Type) *According to [Mey00] an Abstract Data Type defines a set of abstract objects implicitly, through the applicable functions. Moreover, the functions themselves are also defined implicitly: instead of explicit definitions by reference to a concrete representation, an Abstract Data Type Specification describes the functions properties by means of axioms. These axioms are predicates (in the sense of logic) which express that a given condition is satisfied by any possible instance of the Abstract Data Type.*

Eventually the elements of an Abstract Data Type form a subset of an ambient set. In this case the Abstract Data Type may contain a service which answers the membership question in order to distinguish elements of the Abstract Data Type from the elements of the ambient set. Such a membership question corresponds to an abstract invariant in the sense of [LG01]. We interpret this abstract invariant as an special axiom in the sense of [Mey00].

Definition 26 (Constructible Constraints of Object Class)
In [CGH$^+$03] and [GHMS11] an Object Class is given as a constructible subset of a suitable affine space. Thus, an Object Class is defined by a restriction consisting of a Boolean combination of polynomial equations. In other words, the Constructible Constraints of an Object Class are Boolean combinations of polynomial equations that define the elements of the Object Class.

Argument Following Definition 25 the Axioms of an Abstract Data Type define (implicitly) the set of abstract objects of the Abstract Data Type. On the other hand, the Constructible Constraints of Object Class define the elements of the Object Class. Since the notion of an Abstract Data Type may be replaced by the notion of an Object Class in the given context, the Constructible Constraints play the role of Axioms in the sense of [Mey00]. This allows us to replace the term Constructible Constraints of Object Class by the term Axioms of Abstract Data Type

Comparison 7

[Mey00] terms :	Class Invariants
[CGH$^+$03][GHMS11] terms :	Constructible Constraints of Data Structure

Definition 27 (Implementation Invariant) *According to [Mey00] an Implementation Invariant is part of a set of assertions called Class Invariant which express general consistency constraints that apply to every instance of the Class as a whole. The Implementation Invariant expresses the correctness of the representation vis-a-vis the corresponding Abstract Data Type. In mathematical terms, the Implementation Invariant is the characteristic function of the domain of the Abstraction Function, that is to say, the property that determines when that function is applicable. The Implementation Invariant determines when a candidate Concrete Object is indeed the implementation of one (and then only one) Abstract Object.*

Definition 28 (Constructible Constraints of Data Structure) *A Data Structure is a constructible subset of a suitable affine space. Thus, a Data Structure is defined by a Boolean combination of polynomial equations called Constructible Constraints.*

Argument An Implementation Invariant is a list of restrictions which defines the instances of a Class in the sense of [Mey00]. In a similar way, a Data Structure is given by Constructible Constraints. Since the notion of Class captures the notion of Data Structure, we deduce that Constructible Constraints define the instances of a Class. Thus, the notion of Implementation Invariant captures the meaning of Constructible Constraints of a Data Structure. This allows us to replace the term Constructible Constraints of a Data Structure by the term Implementation Invariant

Comparison 8

[Mey00] terms :	Function of an Abstract Data Type
[CGH$^+$03][GHMS11] terms :	Identity and Value Question

Definition 29 (Function of an Abstract Data Type) *According to [Mey00] an Abstract Data Type specification describes and defines a set of elements together with services available and applicable to each element. A Function of an Abstract Data Type is a service consisting of a mathematical function.*

Definition 30 (Identity and Value Question) *The paper [CGH$^+$03] explains two particular instances of services of an Abstract Data Type $\mathcal{O}^*$ which consists of n–variate polynomials: the identity and the value question. The first one is the identity relation between elements of $\mathcal{O}^*$ and the second one returns for a given polynomial $p \in \mathcal{O}^*$ and a point $x \in \mathbb{A}^n$ the value $p(x)$.*

Argument The Identity Question is realized by a Boolean valued function defined on $\mathcal{O}^* \times \mathcal{O}^*$ and the Value Question by a complex valued function on $\mathcal{O}^* \times \mathbb{A}^n$ in the sense of [CGH$^+$03] and [GHMS11]. These mathematical functions are applicable to each element of $\mathcal{O}^*$. Thus, Identity Question and Value Question are services in the sense of [Mey00]. This allows us to consider Identity and Value Question as Functions of an Abstract Data Type.

There is another element in the terminology of [GHMS11] which may be interpreted as a Function of an Abstract Data Type. This element is the map Φ of Diagram (11). This map determines for each interpolation data in $\mathcal{O}$ its unique interpolant belonging to $\mathcal{O}^*$. We may interpret $\mathcal{O}$ as an Abstract Data Type which exports the service Φ. In this sense, the notion Function of an Abstract Data Type captures also the meaning of the map Φ.

Comparison 9

[Mey00] terms :	Routine
[CGH$^+$03][GHMS11] terms :	Algorithm

Definition 31 (Routine) *A Routine consists in some sense of a computation (algorithm) applicable to all instances of a given Class. According to [Mey00] routines may be classified as Procedures and Functions. A Routine is called a Method in Smalltalk; applying a routine to an Object is called sending a message to the Object.*

Here a word should be said about the classification of Routine into Functions and Procedures. Let Query, Command and Creator be classifications of the notion of Abstract Function. Let Function and Procedure be classifications of the notion of Routine. A Function is the implementation of a Query and correspond to a Routine which returns a result. A Procedure is the implementation of a Command and corresponds to a Routine which do not return an explicit result. Figure 15 illustrates the correspondence between the notions Routine and Abstract Function.

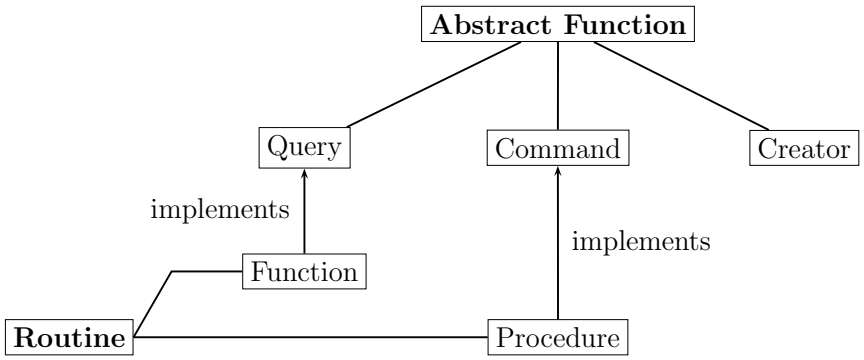


Figure 15: Correspondence between Routine and Abstract Function

Definition 32 (Algorithm) *In [CGH$^+$03] an Algorithm solves a problem. An Algorithm computes in a "uniform" and deterministic manner for each input Code an output Code. In [GHMS11] a Hermite–Lagrange interpolation Algorithm is determined by a constructible subset $\mathcal{D}^*$ of the affine space $\mathbb{A}^M$ acting as an output Data Structure, a polynomial Encoding w^* and a hereditary map Ψ namely the Algorithm in the narrow sense.*

Argument The definitions of Algorithm and Routine are similar in the sense that both belong to the context of implementation. The notion of Routine refers to an operation applicable to any instance of a given Class. In the same way, an Algorithm is in the sense of [CGH$^+$03] a computation applicable to any instance of a given Data Structure. Thus, the notion of Routine captures the meaning of Algorithm. This allows us to replace the term Algorithm of [CGH$^+$03] and [GHMS11] by the term Routine.

5 Coalescence and branching parsimoniousness compared with other quality attributes

At this point the intuitive concepts of coalescence and branching parsimoniousness (or branching–freeness) were introduced as quality attributes which affect negatively the complexity of interpolation and elimination algorithms. In this section we consider the methodology of quality attribute scenarios described in [BCK03] and we obtain examples of other quality attributes with similar properties as coalescence.

We shall explain that the notion of branching parsimoniousness is closely related to quality attributes such as modifiability. We shall also explain that the notion of coalescence is closely related to a special kind of external quality attributes, namely quality attributes that restrict the set of possible outputs. In order to achieve these conclusions, we describe in Section 5.1 coalescence and branching parsimoniousness by means of quality attribute scenarios. In Section 5.2 we compare, in terms of scenarios, coalescence and branching parsimoniousness with some classical quality attributes. Finally, in Section 5.3 we exhibit a couple of quality attributes with similar scenarios to coalescence and branching parsimoniousness.

5.1 Quality attribute scenarios for coalescence and branching parsimoniousness

In this section the method of quality attribute scenarios of [BCK03] is used to define coalescence and branching parsimoniousness.

Quality attribute scenarios

A quality attribute scenario is a method to formulate a quality attribute as an specific requirement. A quality attribute scenario divides a quality attribute in six parts:

- Source of stimulus : A stimulus has a generator called source. Usually the source of stimulus is the user, e.g. a user requiring the software of a data base.
- Stimulus : It arrives to the system, e.g. as a query to the data base.
- Environment : Conditions of the system present when the stimulus arrives, e.g. data base system overloaded of queries or data base system at normal conditions.
- Artifact : Part of the system stimulated, e.g. the algorithm that retrieve information from the data base.
- Response : Activity undertaken in order to give response to the stimulus, e.g, the data base processes the query.
- Response measure : It serves to test the requirement, e.g. the speed of data base processing.

It is usual to distinguish between general scenarios and specific ones. A general scenario formulates each part in general terms, whereas an specific scenario formulates each part in terms of specific situations. In this section coalescence and branching parsimoniousness are described by means of general scenarios.

5.1.1 Quality attribute scenario for coalescence

Let Φ be the abstract function introduced in Section 4.2.1. According to Definition 14 coalescence refers to properties of interpolation problems and algorithms, like for example: if a sequence $(d_i)_{i\in\mathbb{N}}$ of interpolation data converges to an interpolation datum d, then the sequence $(\Phi(d_i))_{i\in\mathbb{N}}$ should be bounded. We describe now this situation in terms of the following quality attribute scenario:

Source	Stimulus	Artifact	Environment	Response	Measure
User	Converging sequence of input data	Output, i.e. the service of the system	Interpolation program at run time	Data are processed	The converge rate of the output sequence

Figure 16: Quality attribute scenario for coalescence

The source is the user because the user chooses a sequence $(d_i)_{i\in\mathbb{N}}$ of interpolation data and runs the algorithm on each interpolation datum of the sequence.

The stimulus is represented by the sequence $(d_i)_{i\in\mathbb{N}}$ of interpolation data which converge to an interpolation datum d. This convergence is a "pattern to arrive" in terms of [BCK03]. The pattern to arrive is a characteristic of the stimulus in scenarios for the quality attribute performance. For example, when the system is supposed to achieve a certain level of performance, it may be required that the system works properly under stimuli arriving at high frequency. In this case the pattern to arrive is high frequency. In the case of coalescence the pattern to arrive is convergence to a interpolation datum.

The artifact or the part of the system stimulated is the interpolation algorithm and indirectly the service of the program which is represented by the output. In the context of Hermite-Lagrange interpolation the interpolant is the service of the program.

The environment is described by the algorithm of interpolation at run time.

The response to the stimulus is data processing by computing the interpolant.

In case that the given sequence of interpolants converges, the measure may be the convergence rate.

5.1.2 Quality attribute scenario for branching parsimoniousness

According to concepts of Section 6 branching parsimoniousness is a property of algorithms.

Definition 33 (Branching parnimoniousness) *Branching parnimoniousness is a property which expresses that branchings, i.e. points where the algorithm bifurcates by means of conditionals or other control structures, are avoided when a branching–free alternative for the computational problem under consideration exists.*

This description allows the elaboration of the following quality attribute scenario:

The source formulates the requirement of branching parsimoniousness.

The stimulus is represented by the requirement of branching parsimoniousness. This requirement is viewed as an stimulus because it determines the form and the construction of the algorithm.

The artifact in this case is the algorithm because it is the part of the system affected by the requirement.

The environment may be development or maintenance because requirements like branching parsimoniousness are usually solved in these stages.

The response to the stimulus is the fact that the algorithm was constructed with a minimal number of branchings.

The measure in this case is the number of branchings. If this number is zero, the stimulus is satisfied.

Source	Stimulus	Artifact	Environment	Response	Measure
User or developer	Requirement that the algorithm should be branching-free	Algorithm	Development or Maintenance	Algorithm constructed with minimal number of branchings	Number of branchings

Figure 17: Quality attribute scenario for branching parsimoniousness

5.2 Scenario comparison with classical quality attributes

In this section the quality attribute scenarios of coalescence and branching parsimoniousness become compared with the quality attributes scenarios of performance, security, availability, testability and usability which can be found in [BCK03]. Figure 18 gives an overview of these quality attribute scenarios[7].

[7]For the sake of simplicity Source and Environment features of quality attribute scenarios were not included since they are irrelevant for our discussion.

QA	Stimulus	Artifact	Response	Measure
Availability	System failure	Resource that is required to be highly available	Record the failure. Notify or disable sources	Time to repair. Availability time
Modifiability	Changes to be made	What is to be changed (code)	Modification is made with no side effects	Number of elements affected. Time to perform modification
Performance	Event arrivals. Arrival pattern (periodic, sporadic)	Services of the system	Process the arriving events	Time to process
Security	Attack or an attempt to break security	Services of the system and data within	Authorize or denying access	Probability of detect attack
Testeability	Unit test	Component of the system	System can be controlled to perform the test	Percentage of statements executed in test
Usability	User input	The system	Anticipate the user's needs	Task time or number of problems solved

Figure 18: Overview of quality attributes scenarios

5.2.1 Comparison with coalescence

Figure 19 below visualizes that the scenario of coalescence is, between the quality attribute of Figure 18, most similar to the scenario of performance. The similarity between coalescence and performance is not total since there are no coincidences in the column of measure. This aspect reveals that measure is an important feature in the definition of quality attributes related to coalescence.

QA	Stimulus	Artifact	Response	Measure
Performance	Interpolation data are events arriving	Output data, i.e., interpolants are the service offered	The response to the stimulus is to interpolate the data	-
Security	Interpolation data could be wrong	The part affected is the service, i.e., the output interpolants	-	-
Testeability	Interpolation data could be test data	-	The response is to process the data, i.e., perform the test	-
Availability	-	The output interpolants are the resources required and affected by faults	-	-
Usability	Interpolation data is user input	-	-	-
Modifiability	-	-	-	-

Figure 19: Coincidences between coalescence and quality attributes

On the other hand, Figure 20 below highlights the differences between coalescence and performance. In the case of coalescence the measure is the convergence rate of the sequence of interpolants (which is supposed to be convergent), whereas in the case of performance the measure is the time to process events. With respect to the other scenarios, the differences with security are response and measure, whereas the differences with testability are artifact and measure. These facts suggest again that the key feature of coalescence is the measure.

The measure of a quality attribute scenario for coalescence consists of the convergence rate of the output data. This convergence rate restricts the set of possible outputs. Thus, quality attributes similar to coalescence are those that restrict the set of possible outputs.

QA	Stimulus	Artifact	Response	Measure
Performance	-	-	-	Convergence rate is not comparable with time of process
Security	-	-	Data interpolation is not comparable with authorization, denegation or reporting an attack	Convergence rate is not related to a measure of the resistance to attacks
Testeability	-	The output interpolant is not comparable with the whole system	-	Convergence rate of the output data is not comparable to easy to test
Availability	Interpolation data are not a failure. However, they could produce one	-	Data interpolation is not comparable with processing a failure	Convergence rate is not related to time to repair
Usability	-	The output interpolant is not comparable with the whole system	Data interpolation is not anticipating the user's need	Convergence is not a measure for the task performed by the user
Modifiability	Interpolation data are not a requirement of change	The output interpolant is not a code	Interpolate data is not comparable with modifying the code	Convergence rate does not allow to measure the cost of change

Figure 20: Differences between coalescence and quality attributes

5.2.2 Comparison with branching parsimoniousness

Figure 21 below visualizes that the scenario of branching parsimoniousness is, between the quality attributes of Figure 18, most similar to the scenario of modifiability. The lack of comparable properties in column "Response" of Figure 21 reveals the relevance of this feature for quality attributes related to branching parsimoniousness.

On the other hand, Figure 22 below highlights the difference between branching parsimoniousness and modifiability. In the case of branching parsimoniousness the (positive) response consists of a branching–free design of the algorithm, whereas in the case of modifiability the response consists of a modification of the algorithm. With respect to the other

scenarios, the main difference is in the response column. This fact suggest that response is the key feature of branching parsimoniousness.

The response in a quality attribute scenario for branching parsimoniousness captures a restriction on the structure of a program, in this case branching–freeness. Thus, quality attributes which are comparable to branching parsimoniousness have to restrict the structure of the program.

QA	Stimulus	Artifact	Response	Measure
Modifiability	Requirement to avoid branchings is a need of change	The object affected is the code	-	Number of branchings in the code is comparable to number of elements affected
Availability	-	The code is a resource required to be available	-	-
Testeability	-	The code is a component of the system capable to be tested	-	-
Performance	-	The code provides the service. Thus, affect the code is affect the service	-	-
Security	-	Idem	-	-
Usability	-	-	-	-

Figure 21: Coincidences between branching parsimoniousness and quality attributes

QA	Stimulus	Artifact	Response	Measure
Modifiability	-	-	Construction of a branching-free algorithm is not a modification of a preexisting system	-
Availability	A system failure is not a requirement of branching-freeness	-	Processing a failure is an activity different from developing an algorithm	Time to repair a failure is a measure of an activity, but not a measure of a property of the code
Testeability	A test unit is not a requirement of branching-freeness	-	Performing a test is not an algorithm design	Number of branching is not comparable with easiness for testing
Performance	Arrivals of events do not represent a requirement of branching-freeness	-	Processing arriving events is not an algorithm design	Number of branchings does not measure processing time
Security	The requirement of branching-freeness is not a system attack	-	The algorithm design is not comparable with authorization or denying of attacks	Resistance to attacks is not measured by number of branchings
Usability	User input is not a requirement of branching-freeness	The code is not the whole system	The algorithm design is not anticipate user's need	Number of branchings does not measure the task realized by the user

Figure 22: Differences between branching parsimoniousness and quality attributes

5.3 Examples of suitable quality attributes

According to Ghezzi [GJM91] and Meyer [Mey88] quality attributes can be classified as: external and internal ones. An alternative classification in [BCK98] distinguishes between attributes which are observable via execution and those which are not. The following examples consider both classifications.

5.3.1 Quality attributes that restrict the set of possible outputs

External quality attributes are visible to the user of the system or program. The output of a program is always visible to the user. Thus, quality attributes that restrict the set of possible outputs are external.

There are important examples of external quality attributes such as efficiency and usability. However, it is difficult to find examples which can be compared with coalescence. The main difficulty for the exhibition of suitable examples is the continuous aspect of the coalescence restriction. Another difficult point is the similarity between restricting the output and formulating a functional requirement.

Despite of these difficulties, we are able to exhibit an example in the context of machine learning. A classifier is an algorithm that is used to classify data. For example, such a classifier could categorize clients of a bank by means of selected variables. The classifier

identifies clients that will leave the bank and clients that will continue stay in the bank. The input of the classifier is a set of variables representing a client and the output is the type of the client, see Figure 23.

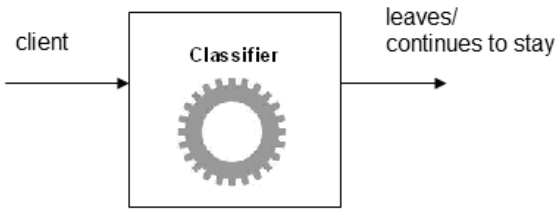


Figure 23: Input and output of a classifier

The classifier is based on a model which in Figure 23 becomes represented by the gear. This model includes in its construction a training of the classifier. This training represents some common feature with coalescense.

Figure 24 illustrates the process of training a classifier. The training consists of feeding the classifier with selected client examples. Each example helps the classifier to construct a model that captures the characteristics of the clients that will leave the bank.

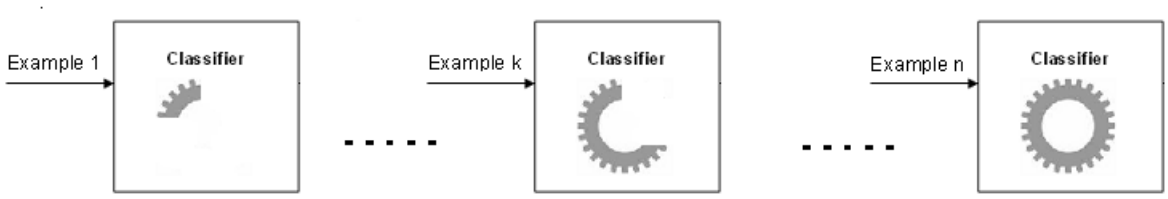


Figure 24: Training of a classifier

It is supposed that if the examples converge to the characterization of a client that leaves the bank, the training process leads to a model which recognize clients that leaves the bank. This may be compared with the convergence of a output data sequence in interpolation, i.e. with coalescence. If the "convergence" does not take place during the training, an inappropriate model is obtained.

Thus the "convergence" of the examples is a requirement which restricts the output of the classifier. The problem is how to realize this restriction. In the context of machine learning this could be a problem difficult to solve.

5.3.2 Quality attributes that restrict the structure of the program

The quality attributes related to the structure of the software are called internal quality attributes. For example modifiability is a quality attribute related to the cost of changing

the system and is therefore internal. Observe that internal quality attributes are visible to the developer of the system, but not to the user. The developer may use certain internal quality attributes in order to satisfy other quality attributes which are difficult to realize directly. This is the case of the quality attribute reliability.

Nevertheless, internal quality attributes do not affect directly the structure of a system. The satisfaction of internal quality attributes requires techniques which are called tactics. For example, a system is easily modified when it is structured, modularized and well documented.

According to [BCK03] a tactic is a design decision that influences the control of a quality attribute response[8]. In Section 5.2 it was explained that the key feature of branching parsimoniousness is its response. Since tactics act on the feature response, branching parsimoniousness becomes achieved by tactics.

The relation between quality attributes and tactics is explained with more detail by the following example.

Figure 25 below illustrates that testability requires modularity. Therefore modularity is a tactic or a design decision which allows to achieve testability. Modularity is viewed as a tactic because modularity is a technique which the program designer uses in order to satisfy certain previously fixed requirements. Relevant is the distinction between the 'what' and the 'how'. For example, testability is the 'what' and modularity is the 'how'. Another example of a tactic that appears in Figure 25 is redundancy. According to [BW96] the tactic of redundancy promotes directly the quality attribute reliability. Notice that redundancy is based on a decomposition of the system into modules or components. Thus, redundancy includes modularity.

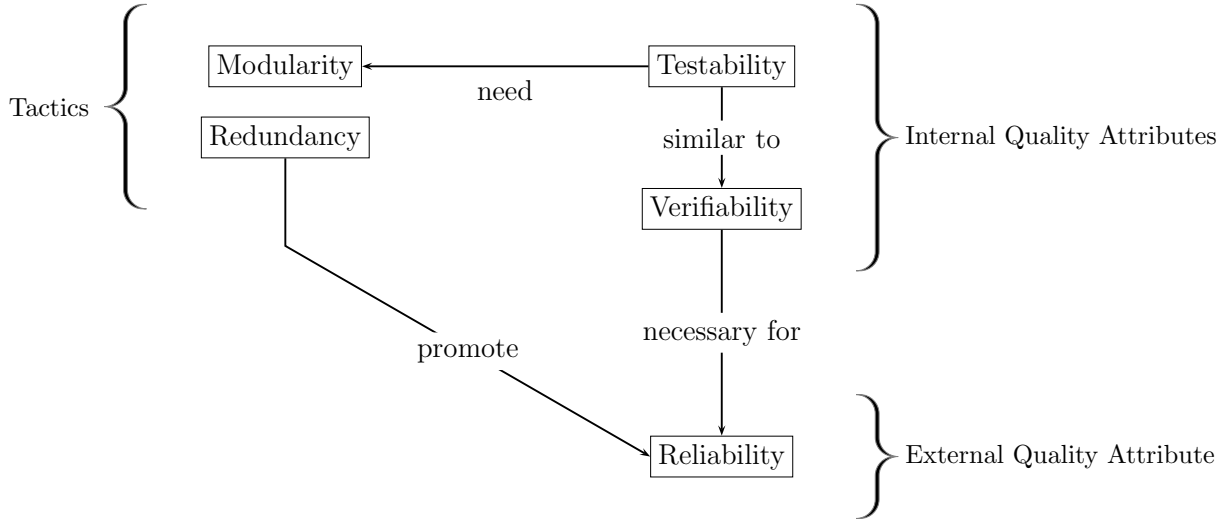


Figure 25: Relation between (internal/external) quality attributes and tactics

[8]According to [PA09] the notion of design decision is quite general. It could refer to a decision about data format or operations on data; the hardware devices or other components with which the software must interoperate; protocols of messages between components, or the choice of algorithms.

Figure 25 illustrates two interesting points. The first point is that quality attributes depend on tactics or design decisions such as modularity or redundancy. The second point is that these tactics can be viewed as restrictions on the structure of the system. Restrictions on the structure, as e.g. redundancy and loose of coupling[9], are typically linked to modularity.

5.3.3 Remarks about suitable Quality Attributes

Our previous considerations suggest that we should focus our attention to tactics for the realization of quality attributes linked to branching parsimoniousness and that the comparison of branching parsimoniousness with other, standard, quality attributes is only clumsy and less relevant. Tactics are the tool for achieving quality attributes including quality attributes which restrict the structure of the system. Thus, exploring tactics is the best way to find elements of software engineering relevant for branching parsimoniousness. Among the known tactics, modularity is the most important one because it constitutes a fundamental design concept.

On the other hand, coalescence is very close to external quality attributes such as performance. This proximity suggests the exploration of the relation between coalescence and other external quality attributes.

[9]Loose of coupling is a characteristic of a modularized system where the modules have low coupling, i.e. low dependencies between modules. According to [Mey88] the quality attribute reusability requires loose of coupling.

6 A software architecture based computation model for arithmetic circuits

6.1 Aims and paradigmatic examples

The aim of this section is to introduce and motivate a practically feasible, software architecture based model of branching–free (or at least branching parsimonious) computation using the circuit representation of rational functions as fundamental data type. In this computation model, a procedure or routine will accept a circuit as input and produce another circuit as output. Since the basic ingredients of our computations with circuits are supposed to be branching–free and circuits themselves may be interpreted as computations, the circuits used as data types in our model should be branching–free too. This leads us, after some motivating considerations, to introduce and discuss the concept of a *parameterized arithmetic circuit*. However, branchings are sometimes unavoidable. Nevertheless, frequently they may be replaced by limit processes. In order to capture this situation, we shall also introduce and discuss the notion of a *robust* parameterized arithmetic circuit.

An important issue of this section is the concept of well behavedness, under certain modifications of the input circuits, of procedures and routines, which solve formally specified computational problems. This concept will finally allow us to establish our software architecture based model of computation with robust parameterized arithmetic circuits.

It is well known that there exist geometric elimination problems which are closely related to NP–hardness, although the arithmetic and the bit computation models are substantially different (see e.g. [BSS89], [SS95], [BCSS98] and [HM93]). In the bit model, branchings may be mimicked by boolean operations and appear more related to algorithms than to the computational problem itself.

It is well known that an efficient general purpose algorithm may become inefficient when it becomes applied to a subproblem consisting of non–generic input instances. This may even occur in unexpected situations as e.g. in the case of the knapsack based Merkle–Hellman cryptosystem which in 1984 was broken by a specific polynomial time algorithm ([Sha84]).

In the continuous world of arithmetic computation models over fields like $\mathbb{R}$ and $\mathbb{C}$, things look quite different. The reader might be aware that, in the formulation of a Hermite–Lagrange interpolation problem in Section 4.2, we limited our attention to topologically robust and hereditary constructible maps Φ. The reason was to allow that the subsequent interpolation algorithms, which solve these problems, could be performed in a numerically meaningful and, in particular, branching–free way on any given input. Without this limitation we implicitly would have admitted algorithmic situations where branchings are unavoidable. Especially, if the constructible map Φ is geometrically robust, we may conclude that Φ is also topologically robust and hereditary and that the Hermite–Lagrange interpolation problem determined by Φ can be solved by a geometrically robust algorithm.

With the conceptual couple of topological robustness and hereditarity, and the concept of geometrical robustness we were able to introduce into the discussion a certain architectural requirement that guarantees "*well behavedness*" of interpolation problems

(and algorithms) *under restrictions* of the inputs to special sets of instances (this includes also the case of *specializations* to single inputs). Problems and algorithms which are well behaved under restrictions have the following property:

> if we restrict the inputs we obtain a *subproblem* and a *subalgorithm*, respectively.

A more home–made way to tackle this kind of situations consists of extracting from the given interpolation problem "meaningful" classes of subproblems and submit them to individual solutions.

In the bit–world nobody questions the existence of such a hierarchy of "meaningful" problems. This becomes visualized by the following example. Although the general graph coloring problem is NP–complete, nobody considers as senseless the search for polynomial time coloring algorithms for special graph types. This illustrates also the implicit ubiquity of conceptual opposition of branchings and branching–freeness even for computational problems formulated in the bit model.

Until now we modeled in this thesis concrete objects by means of points belonging to affine spaces of *fixed* dimension and characterized routines by means of (mostly topological robust and hereditary) constructible maps of a *fixed* number of arguments. This entailed the non–uniform nature of our computation model. Our routines operated between complex vectors of fixed length and not, as usual in programming, on the corresponding dynamic objects. However, in real world Software Engineering, abstract data types and classes implementing them have often an interleaved recursive structure which must be taken into account in the program design. Let us illustrate this alternative view by means of two examples: integer multiplication in the bit model and matrix multiplication in the arithmetic model.

In the first case we start from the (non–unique) representation of non–negative integers by (not too short) dynamic bit vectors. This representation may be implemented in different ways, e.g. by stacks or nested (linear) arrays. Figure 26 illustrates this software architecture.

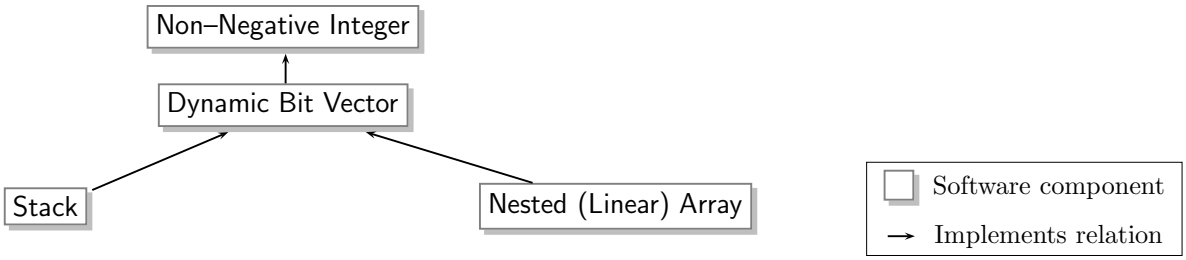


Figure 26: Software architecture for non-negative integers

Let us first consider the stack implementation of integers of bit length n. In this case we may realize integer multiplication by a recursive algorithm using $O(n^2)$ bit operations. It is unlikely that the asymptotic order of this complexity can be improved for the stack implementation of integers.

Things change drastically when integers become implemented by nested arrays. The Karatsuba–Ofman and the Schönhage–Strassen integer multiplication algorithms ([vzGG03]) are designed for the nested array implementation of integers and the complexity of these procedures, counted in bit operations, becomes now of order $O(n^{\log 3}) = O(n^{1.585})$ and $O(n \log n \log\log n)$, respectively.

In a similar way matrices over fields may be implemented by stacks of stacks or by nested (bidimensional) arrays. In the first case the multiplication of two $(n \times n)$–matrices may be realized by a (for this implementation probably optimal) recursive algorithm using $O(n^3)$ field operations and in the second case by Strassens's recursive matrix multiplication algorithm which uses only $O(n^{\log 7}) = O(n^{2.804})$ arithmetical operations.

In the cases of the array based fast integer and matrix multiplication the following occurs. Let us fix two positive integers n and m subject to the condition $n \leq 2^m$ (for example $m := \lceil \log n \rceil$). Let be given two n–bit vectors α and β representing integers a and b and let α' and β' be the 2^m–bit vectors obtained by "filling up" the boolean vectors α and β at the right hand end by zero bits. Observe that the 2^m–bit vectors α' and β' still represent the integers a and b.

Similarly let be given two $(n \times n)$–matrices A and B representing two $\mathbb{C}$–linear endomorphisms φ and ψ of the $\mathbb{C}$–linear space $\mathbb{A}^n$. Let A' and B' be the $(2^m \times 2^m)$–matrices obtained by "filling up" the matrices A and B at the top and at the right hand end by suitable zero rows and columns. Then φ and ψ may be canonically extended to $\mathbb{C}$–linear endomorphisms φ' and ψ' of the $\mathbb{C}$–linear space $\mathbb{A}^{2^m}$.

Now we represent α', β' and A', B' by nested arrays and apply to α', β' the Karatsuba–Ofman or the Schönhage–Strassen integer multiplication and to A', B' the Strassen matrix multiplication algorithm. The outcome is a 2^{m+1}–bit vector γ' which represents the integer ab and the $(2^m \times 2^m)$–matrix $A'B'$ which represents the composition $\varphi' \circ \psi'$ of the $\mathbb{C}$–linear maps φ' and ψ'. A $(2n-1)$–bit vector γ representing the product ab and the entries of the matrix AB representing the composition $\varphi \circ \psi$ can now easily be read–off from the $(2^m - 1)$–bit vector γ' and the $2^m \times 2^m$–matrix $A'B'$ without realizing any additional boolean or arithmetic operations. The Karatsuba–Ofman and the Schönhage–Strassen integer multiplication algorithms compute γ' using $O(2^{m \log 3})$ and $O(2^m m \log m)$ bit operations, respectively, and the Strassen matrix multiplication algorithm computes $A'B'$ using $O(2^{m \log 7})$ arithmetical operations. The corresponding computations may be arranged, following the context, as boolean or arithmetic circuits.

If we consider now the entries of α and β as boolean variables and the entries of A and B as indeterminates over $\mathbb{C}$, we see that the computation of γ' and $A'B'$ (and hence of γ and AB) involves only $O(n^{\log 3})$, $O(n \log n \log\log n)$ and $O(n^{\log 7})$ genuine boolean and arithmetical operations, independently of the current value of m. The rest of the boolean or arithmetic operations become applied to pairs of arguments from which at least one has the value zero. These operations may therefore be canceled out, avoiding side effects. This cancellation process is purely combinatorial and does not require any computational effort. Of course, each cancellation involves a rearrangement of the circuit representing the computation. In the sequel we shall refer to this kind of operations, namely cancellations and circuit rearrangements, as circuit *reductions*.

If we consider now the problems of the efficient multiplication of n bit integers and $(n \times n)$–matrices as multiplication of 2^m–bit integers and $(2^m \times 2^m)$–matrices, we see that the recursive, array based Karatsuba–Ofman and Schönhage–Strassen integer multiplication

and the Strassen matrix multiplication algorithms are *well behaved under restrictions*. This means that these algorithms admit an efficient adaptation to reduced input instances, by means of a simple combinatorial cancellation and rearrangement, namely a reduction process.

A separate class of algorithms in scientific complexity consists of the *approximative* ones. The concept of approximative algorithms is motivated by numerical analysis applications. Approximative algorithms constitute an important tool for fast matrix multiplication (see e.g. [Sch81] or [BCS97], Chapter 15) and for polynomial equation solving over $\mathbb{C}$ as well as over $\mathbb{R}$ (see e.g. [Can88], [GH93], [HKP$^+$00]). In all these cases the approximative algorithm under consideration becomes finally transformed into a symbolic, infinite precision routine. This is done by a division by an infinitesimal which disguises in fact a blow up (see Section 6.5.5 below).

Instead of introducing infinitesimals we shall adopt in this thesis another point of view which comes closer to that of "numerical algebraic geometry" (see e.g. [Ste04], [SW05], [EGL97] and [CYGM09]). Under the name of "coalescence" (term borrowed from interpolation theory) we shall algebraically mimic a kind of convergence of algorithms. A basic requirement for that will be well behavedness under restrictions (see Section 6.4 below).

6.2 Parameterized arithmetic circuits and their semantics

The routines of our computation model, which will be introduced in Section 6.4, operate with circuits representing parameter dependent rational functions. They will behave well under restrictions. In this spirit, the objects of our abstract data types will be parameter dependent multivariate rational functions over $\mathbb{C}$, the concrete objects of our classes will be parameterized arithmetic circuits and our abstraction function will be the circuit representation of rational functions. In what follows, $\mathbb{C}$ may always be replaced, mutatis mutandis, by an arbitrary algebraically closed field (of any characteristic).

Let us fix natural numbers n and r, indeterminates $X_1, \ldots, X_n$ and a non–empty constructible subset $\mathcal{M}$ of $\mathbb{A}^r$. By $\pi_1, \ldots, \pi_r$ we denote the restrictions to $\mathcal{M}$ of the canonical projections $\mathbb{A}^r \to \mathbb{A}^1$.

A *(by $\mathcal{M}$) parameterized arithmetic circuit* β (with *basic parameters* $\pi_1, \ldots, \pi_r$ and *inputs* $X_1, \ldots, X_n$) is a labelled directed acyclic graph (labelled DAG) satisfying the following conditions:
each node of indegree zero is labelled by a scalar from $\mathbb{C}$, a basic parameter $\pi_1, \ldots, \pi_r$ or a input variable $X_1, \ldots, X_n$. Following the case, we shall refer to the *scalar, basic parameter* and (standard) *input* nodes of β. All others nodes of β have indegree two and are called *internal*. They are labelled by arithmetic operations (addition, subtraction, multiplication, division). A *parameter* node of β depends only on scalar and basic parameter nodes, but not on any input node of β. An addition or multiplication node whose two ingoing edges depend on an input is called *essential*. The same terminology is applied to division nodes whose second argument depends on an input. Moreover, at least one circuit node becomes labelled as output. Without loss of generality we may suppose that all nodes of outdegree zero are outputs of β. Figure 27 illustrates this description.

We consider β as a syntactical object which we wish to equip with a certain semantics. In principle there exists a canonical evaluation procedure of β assigning to each node a

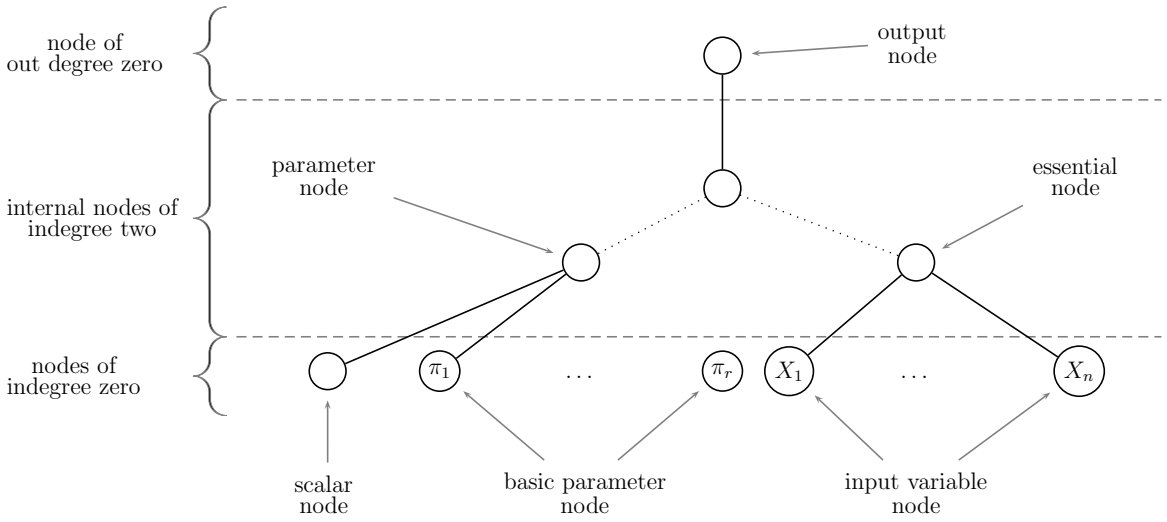


Figure 27: Parts of parameterized arithmetic circuit β

rational function of $\mathcal{M} \times \mathbb{A}^n$ which, in case of a parameter node, may also be interpreted as a rational function of $\mathcal{M}$. We call such a rational function an *intermediate result* of β.

The evaluation procedure may fail if we divide at some moment an intermediate result by another one which vanishes on a Zariski dense subset of a whole irreducible component of $\mathcal{M} \times \mathbb{A}^n$. If this occurs, we call the labelled DAG β *inconsistent*, otherwise *consistent*. From [CGH$^+$03], Corollary 2 (compare also [HS], Theorem 4.4 and [GH01], Lemma 3) one deduces easily that testing whether an intermediate result of β vanishes on a Zariski dense subset of a whole irreducible component of $\mathcal{M} \times \mathbb{A}^n$ can efficiently be reduced to the same task for circuit represented rational functions of $\mathcal{M}$ (the procedure is of non–uniform deterministic or alternatively of uniform–probabilistic nature).

Mutatis mutandis the same is true for identity checking between intermediate results of β. If $\mathcal{M}$ is irreducible, both tasks boil down to an identity–to–zero test on $\mathcal{M}$. In case that $\mathcal{M}$ is not Zariski dense in $\mathbb{A}^r$, this issue presents a major open problem in modern Theoretical Computer Science (see [Sax09] for details).

From now on we shall always assume that β is a consistent parameterized arithmetic circuit. The intermediate results associated with output nodes will be called *final results* of β.

We call an intermediate result associated with a parameter node a *parameter* of β and interpret it generally as a rational function of $\mathcal{M}$. A parameter associated with a node which has an outgoing edge into a node which depends on one of the inputs of β is called *essential*. In the sequel we shall refer to the constructible set $\mathcal{M}$ as the *parameter domain* of β.

We consider β as a syntactic object which represents the final results of β, i.e. the rational functions of $\mathcal{M} \times \mathbb{A}^n$ assigned to its output nodes. In this way becomes introduced an abstraction function which associates with β these rational functions. This abstraction function assigns therefore to β a rational map $\mathcal{M} \times \mathbb{A}^n \dashrightarrow \mathbb{A}^q$, where q is the number of

output nodes of β. On its turn, this rational map may also be understood as a (by $\mathcal{M}$) parameterized family of rational maps $\mathbb{A}^n \dashrightarrow \mathbb{A}^q$.

Now we suppose that the parameterized arithmetic circuit β has been equipped with an additional structure, linked to the semantics of β. We assume that for each node ρ of β there is given a *total* constructible map $\mathcal{M} \times \mathbb{A}^n \to \mathbb{A}^1$ which extends the intermediate result associated with ρ. Therefore, if β has K nodes, we obtain a total constructible map $\Omega : \mathcal{M} \times \mathbb{A}^n \to \mathbb{A}^K$ which extends the rational map $\mathcal{M} \times \mathbb{A}^n \dashrightarrow \mathbb{A}^K$ given by the labels at the indegree zero nodes and the intermediate results of β.

Definition 34 (Robust circuit) *Let notations and assumptions be as before. The pair (β, Ω) is called a robust parameterized arithmetic circuit if the constructible map Ω is geometrically robust.*

We shall make the following two observations to this definition.

We state our first observation. Suppose that (β, Ω) is robust. Then the constructible map $\Omega : \mathcal{M} \times \mathbb{A}^n \to \mathbb{A}^K$ is geometrically and hence also topologically robust and hereditary. Moreover, there exists at most one geometrically robust constructible map $\Omega : \mathcal{M} \times \mathbb{A}^n \to \mathbb{A}^K$ which extends the rational map $\mathcal{M} \times \mathbb{A}^n \dashrightarrow \mathbb{A}^K$ introduced before. Therefore we shall apply the term "robust" simply to the circuit β.

Let us now state our second observation. We may consider the parameterized circuit β as a program which solves the problem to evaluate, for any sufficiently generic parameter instance $u \in \mathcal{M}$, the rational map $\mathbb{A}^n \dashrightarrow \mathbb{A}^q$ which we obtain by specializing to the point u the first argument of the rational map $\mathcal{M} \times \mathbb{A}^n \dashrightarrow \mathbb{A}^q$ defined by the final results of β. In this sense, the "problem" solved by β is given by the final results of β.

Being robust becomes now an architectural requirement for the circuit β and for its output. Robustness implies well behavedness under restrictions as described in Section 6.1. Let us formulate this more precisely.

Let $\mathcal{N}$ be a constructible subset of $\mathcal{M}$ and suppose that (β, Ω) is robust. Then the restriction $\Omega|_{\mathcal{N} \times \mathbb{A}^n}$ of the constructible map Ω to $\mathcal{N} \times \mathbb{A}^n$ is still a geometrically robust constructible map.

This implies that (β, Ω) induces a by $\mathcal{N}$ parameterized arithmetical circuit $\beta_{\mathcal{N}}$ such that $(\beta_{\mathcal{N}}, \Omega|_{\mathcal{N} \times \mathbb{A}^n})$ becomes robust. We call $(\beta_{\mathcal{N}}, \Omega|_{\mathcal{N} \times \mathbb{A}^n})$, or simply $\beta_{\mathcal{N}}$, the *restriction* of (β, Ω) or β to $\mathcal{N}$.

We say that the parameterized arithmetic circuit β is *totally division–free* if any division node of β corresponds to a division by a non–zero complex scalar.

We call β *essentially division–free* if only parameter nodes are labelled by divisions. Thus the property of β being totally division–free implies that β is essentially division–free, but not vice versa. Moreover, if β is totally division-free, the rational map given by the intermediate results of β is polynomial and therefore a geometrically robust constructible map. Thus, any by $\mathcal{M}$ parameterized, totally division–free circuit is in a natural way robust.

In the sequel we shall need the following elementary fact whose easy proof we omit here.

Lemma 13 *Let notations and assumptions be as before and suppose that the parameterized arithmetic circuit β is robust. Then all intermediate results of β are polynomials in*

$X_1, \ldots, X_n$ over the $\mathbb{C}$–algebra of geometrically robust constructible functions defined on $\mathcal{M}$.

The statement of this lemma should not lead to confusions with the notion of an essentially division–free parameterized circuit. We say just that the intermediate results of β are polynomials in $X_1, \ldots, X_n$ and do not restrict the type of arithmetic operations contained in β.

Whether divisions of polynomials by their factors may always be substituted efficiently by additions and multiplications is an important issue in Theoretical Computer Science (compare [Str73]).

To our parameterized arithmetic circuit β we may associate different complexity measures and models. In this thesis we shall mainly be concerned with *sequential computing time*, measured by the *size* of β. Here we refer with "size" to the number of internal nodes of β which count for the given complexity measure. Our basic complexity measure is the *non–scalar* one (also called *Ostrowski measure*) over the ground field $\mathbb{C}$. This means that we count, at unit costs, only essential multiplications and divisions (involving basic parameters or input variables in both arguments in the case of a multiplication and in the second argument in the case of a division), whereas $\mathbb{C}$–linear operations are free (see [BCS97] for details).

We describe now how, based on its semantics, the given parameterized arithmetic circuit β may be rewritten into a new circuit which computes the same final results as β.

The resulting two rewriting procedures, called reduction and broadcasting, will neither be unique and nor generally confluent. For his easiness, the reader may suppose that there is given an (efficient) algorithm which allows identity checking between intermediate results of β. However, we shall not make explicit reference to this assumption. We are now going to introduce the first rewriting procedure.

Suppose that the parameterized arithmetic circuit β computes at two different nodes, say ρ and ρ', the same intermediate result. Assume first that ρ neither depends on ρ', nor ρ' on ρ. Then we may erase ρ' and its two ingoing edges (if ρ' is an internal node) and draw an outgoing edge from ρ to any other node of β which is reached by an outgoing edge of ρ'. If ρ' is an output node, we label ρ also as output node. Observe that in this manner a possible indexing of the output nodes of β may become changed but not the final results of β themselves.

Suppose now that ρ' depends on ρ. Since the DAG β is acyclic, ρ does not depend on ρ'. We may now proceed in the same way as before, erasing the node ρ'.

Let β' be the parameterized arithmetic circuit obtained, as described before, by erasing the node ρ'. Then we call β' a *reduction* of β and call the way we obtained β' from β a *reduction step*. A *reduction procedure* is a sequence of successive reduction steps.

One sees now easily that a reduction procedure applied to β produces a new parameterized arithmetic circuit β^* (also called a *reduction* of β) with the same basic parameter and input nodes, which computes the same final results as β (although their possible indexing may be changed). Moreover, if β is a robust parameterized circuit, then β^* is robust too. Observe also that in the case of robust parameterized circuits our reduction commutes with restriction.

Now we introduce the second rewriting procedure.

Let assumptions and notations be as before and let be given a set P of nodes of β and a robust parameterized arithmetic circuit γ with parameter domain $\mathcal{M}$ and $\#P$ input nodes, namely for each $\rho \in P$ one which becomes labelled by a new input variable Y_ρ. We obtain a new *robust* parameterized arithmetic circuit, denoted by $\beta *_P \gamma$, when we join γ with β, replacing for each $\rho \in P$ the input node of γ, which is labelled by the variable Y_ρ, by the node ρ of β. The output nodes of β constitute also the output nodes of $\beta *_P \gamma$. Thus β and $\beta *_P \gamma$ compute the same final results. We call the circuit $\beta *_P \gamma$ and all its reductions *broadcastings* of β. Thus broadcasting a robust parameterized arithmetic circuit means rewriting it using only valid polynomial identities.

Notice that our treatment of the Strassen matrix multiplication algorithm in Section 6.1 contains a reduction procedure which involves additionally the erasing of certain output nodes. If we consider arithmetic circuits as computer programs, then reduction represents a kind of program transformation.

6.2.1 A specification language for circuits

Computer programs (or "programmable algorithms") written in high level languages are not the same thing as just "algorithms" in Complexity Theory. Whereas in the uniform view algorithms become implemented by suitable machine models and in the non–uniform view by devices like circuits, specifications and correctness proofs are not treated by the general theory, but only, if necessary, outside of it in an ad–hoc manner. The meaning of "algorithm" in Complexity Theory is therefore of syntactic nature.

On the other hand, computer programs, as well as their subroutines (modules) include specifications and correctness proofs, typically written in languages organized in a hierarchy of different abstraction levels. In this sense *programmable algorithms* become equipped with semantics. This is probably the main difference between Complexity Theory and Software Engineering.

In this thesis we are only interested in algorithms which in some sense are programmable. The routines of our computation model will operate on parameterized arithmetic circuits (see Section 6.4). Therefore we are now going to fix a (many–sorted) first order specification language $\mathcal{L}$ for these circuits.

The language $\mathcal{L}$ will include the following non–logical symbols:

- $0, 1, +, -, \times$, and a constant for each complex number,

- as variables

$$n_1, \ldots, n_s \ldots$$
$$\alpha^{(1)}, \ldots, \alpha^{(t)} \ldots$$
$$\beta_1, \ldots, \beta_k \ldots$$
$$\rho_1, \ldots, \rho_l \ldots$$
$$\mathcal{M}_1, \ldots, \mathcal{M}_k \ldots$$
$$U^{(1)}, \ldots, U^{(m)} \ldots$$
$$X^{(1)}, \ldots, X^{(h)} \ldots$$
$$Y^{(1)}, \ldots, Y^{(q)} \ldots$$

to denote non–negative integers and vectors of them, robust parameterized arithmetic circuits, their nodes, their parameter domains, their parameter instances, their input variable vectors and instances of input variable vectors in suitable affine spaces,

- suitable binary predicate symbols to express relations like "ρ is a node of the circuit β", "multiplication is the label of the node ρ of the circuit β", "$\mathcal{M}$ is the parameter domain of the circuit β", "U is a parameter instance of the circuit β", "r is a non–negative integer and the vector length of U is r", "X is the input variable vector of the circuit β" and "n is a non–negative integer and the vector length of X is n",

- a ternary predicate symbol to express "ρ_1 and ρ_2 are nodes of the circuit β and there is an edge of β from ρ_1 to ρ_2",

- binary function symbols to express "U is a parameter instance, k is a natural number and U_k is the k–th entry of U" and "X is an input variable vector, n is a natural number and X_n is the n–th entry of X" and "Y is a variable vector instance, n is a natural number and Y_n is the n–th entry of Y",

- a unary function and a binary predicate symbol to express "the set of output nodes of the circuit β" and "ρ is an output node of the circuit β"

- a quaternary function symbol $G_\rho(\beta; U; X)$ to express "ρ is a node of the circuit β, U is a parameter instance and X is the input variable vector of β and $G_\rho(\beta; U; X)$ is the intermediate result of β at the node ρ and the parameter instance U",

- a predicate symbol for equality for any of the sorts just introduced.

For the treatment of non–negative integers we add the Presburger arithmetic to our first order specification language $\mathcal{L}$.

At our convenience we may add new function and predicate symbols and variable sorts to $\mathcal{L}$. Typical examples are for β a circuit, U a parameter instance, X the input variable vector and $\rho, \rho_1, \ldots, \rho_m$ nodes of β:
"degree of $G_\rho(\beta; U; X)$" and "the vector lengths of X and Y are equal (say n) and Y is a point of the closed subvariety of $\mathbb{A}^n$ defined by the polynomials $G_{\rho_1}(\beta; U; X)$, $\ldots$, $G_{\rho_m}(\beta; U; X)$".

In the same spirit, we may increase the expressive power of $\mathcal{L}$ in order to be able to express for a robust parameterized circuit β with irreducible parameter domain, U a parameter instance, X the input variable vector, ρ a node of β and α a vector of non–negative integers of the same length as X (say n), "the coefficient of the monomial X^α occurring in the polynomial $G_\rho(\beta; U; X)$" (recall Lemma 13). Here we denote for $X := (X_1, \ldots, X_n)$ and $\alpha := (\alpha_1, \ldots, \alpha_n)$ by X^α the monomial $X^\alpha := X_1^{\alpha_1}, \ldots, X_n^{\alpha_n}$.

The semantics of the specification language $\mathcal{L}$ is determined by the universe of all robust parameterized arithmetic circuits, where we interpret all variables, function symbols and predicates as explained before. We call this universe the *standard model* of $\mathcal{L}$. The set of all closed formulas of $\mathcal{L}$ which are true in this model form the *elementary theory* of $\mathcal{L}$.

6.3 Generic computations

In the sequel we shall use ordinary arithmetic circuits over $\mathbb{C}$ as *generic computations* [BCS97] (also called *computation schemes* in [Hei89]). The indegree zero nodes of these arithmetic circuits are labelled by scalars and parameter and input variables.

The aim is to represent different parameterized arithmetic circuits of similar size and appearance by different specializations (i.e. instantiations) of the parameter variables in one and the same generic computation. For a suitable specialization of the parameter variables, the original parameterized arithmetic circuit may then be recovered by an appropriate reduction process applied to the specialized generic computation.

This alternative view of parameterized arithmetic circuits will be fundamental for the design of routines of the branching–free computation model we are going to describe in Section 6.4.1. The routines of our computation model will operate on robust parameterized arithmetic circuits and their basic ingredients will be subroutines which calculate parameter instances of suitable, by the model previously fixed, generic computations. These generic computations will be organized in finitely many families which will only depend on a constant number of discrete parameters. These discrete families constitute the basic building block of our model for branching–free computation.

We shall now exemplify these abstract considerations in the concrete situation of the given parameterized arithmetic circuit β. Mutatis mutandis we shall follow the exposition of [KP96], Section 2. Let $l, L_0, \ldots, L_{l+1}$ with $L_0 \geq r+n+1$ and $L_{l+1} \geq q$ be given natural numbers. Without loss of generality we may suppose that the non–scalar depth of β is positive and at most l, and that β has an oblivious leveled structure of $l+2$ levels of width at most $L_0, \ldots, L_{l+1}$. Let $U_1, \ldots, U_r$ be new indeterminates (they will play the role of a set of "special" parameter variables which will only be instantiated by $\pi_1, \ldots, \pi_r$).

We shall need the following indexed families of "scalar" parameter variables (which will only be instantiated by complex numbers).

- for $n + r < j \leq L_0$ the indeterminate V_j
- for $1 \leq i \leq l$, $1 \leq j \leq L_i$, $0 \leq h \leq i$, $1 \leq k \leq L_h$, the indeterminates $A_{i,j}^{(h,k)}$, $B_{i,j}^{(h,k)}$ and $S_{i,j}$, $T_{i,j}$
- for $1 \leq j \leq L_{l+1}$, $1 \leq k \leq L_l$ the indeterminate C_j^k.

We consider now the following function Q which assigns to every pair (i,j), $1 \leq i \leq l$, $1 \leq j \leq L_i$ and $(l+1, j)$, $1 \leq j \leq L_{l+1}$ the rational expressions defined below:

$$Q_{0,1} := U_1, \ldots, Q_{0,r} := U_r,$$

$$Q_{0,r+1} := X_1, \ldots, Q_{0,r+n} := X_n,$$

$$Q_{0,r+n+1} := V_{r+n+1}, \ldots, Q_{0,L_0} := V_{L_0}$$

For $1 \leq i \leq l$ and $1 \leq j \leq L_i$ the value $Q_{i,j}$ of the function Q is recursively defined by

$$Q_{i,j} := S_{i,j}\Big(\sum_{\substack{0 \leq h < i \\ 1 \leq k \leq L_h}} A_{i,j}^{(h,k)} Q_{h,k} \;.\; \sum_{\substack{0 \leq k' < i \\ 1 \leq k' \leq L_{h'}}} B_{i,j}^{(h',k')} Q_{h',k'}\Big) \;+$$

$$T_{i,j}(\sum_{\substack{0\leq h<i\\ 1\leq k\leq L_h}} A_{i,j}^{(h,k)} Q_{h,k} \;/\; \sum_{\substack{0\leq h'<i\\ 1\leq k'\leq L_{h'}}} B_{i,j}^{(h',k')} Q_{h',k'})$$

Finally, for $(l+1,j)$, $1 \leq j \leq L_{l+1}$ we define $Q_{(l+1,j)} := \sum_{1\leq k\leq L_l} C_j^k Q_{l,k}$

We interpret the function Q as a (consistent) ordinary arithmetic circuit, say Γ, over $\mathbb{Z}$ (and hence over $\mathbb{C}$) whose indegree zero nodes are labelled by the "standard" input variables $X_1, \ldots, X_n$, the special parameter variables $U_1, \ldots, U_r$ and the scalar parameter variables just introduced.

We consider first the result of instantiating the scalar parameter variables contained in Γ by complex numbers. We call such an instantiation a *specialization* of Γ. It is determined by a point in a suitable affine space. Not all possible specializations are *consistent*, giving rise to an assignment of a rational function of $\mathbb{C}(U_1, \ldots, U_r, X_1, \ldots, X_n)$ to each node of Γ as intermediate result.

We call the specializations which produce a failing assignment *inconsistent*. If in the context of a given specialization of the scalar parameter variables of Γ we instantiate for each index pair (i,j), $1 \leq i \leq l$, $1 \leq j \leq L_i$ the variables $S_{i,j}$ and $T_{i,j}$ by two different values from $\{0,1\}$, the labelled directed acyclic graph Γ becomes an ordinary arithmetic circuit over $\mathbb{C}$ of non–scalar depth at most l and non–scalar size at most $L_1 + \cdots + L_l$ with the inputs $U_1, \ldots, U_r, X_1, \ldots, X_n$.

We may now find a suitable specialization of the circuit Γ into a new circuit Γ' over $\mathbb{C}$ such that the following condition is satisfied:

> the (by $\mathcal{M}$) parameterized circuit obtained from Γ' by replacing the special parameter variables $U_1, \ldots, U_r$ by $\pi_1, \ldots, \pi_r$, is consistent and can be reduced to the circuit β.

We may now consider the circuit Γ as a generic computation which allows to recover β by means of a suitable specialization of its scalar and special parameter variables into complex numbers and basic parameters $\pi_1, \ldots, \pi_r$ and by means of circuit reductions. Moreover, any by $\mathcal{M}$ parameterized, consistent arithmetic circuit of non–scalar depth at most l, with inputs $X_1, \ldots, X_n$ and q outputs, which has an oblivious level structure with $l+2$ levels of width at most $L_0, \ldots, L_{l+1}$, may be recovered from Γ by suitable specializations and reductions (see [BCS97], Chapter 9 for more details on generic computations).

6.4 A model for branching–free computation. Informal discussion

We are now going to introduce a model of branching–free computation with parameterized arithmetic circuits. We shall first require that the routines of this computation model should be well behaved under restrictions of the inputs. We discuss this issue now informally.

Suppose for the moment that our computation model is already established. Then its routines transform a given parameterized arithmetic (input) circuit into another parameterized (output) circuit such that both circuits have the same parameter domain. Applied to a given parameterized input circuit, a routine of our computation model generates by

means of its intermediate steps a DAG of parameterized arithmetic circuits, one contained in the other, which have all the same parameter domain.

Let $\mathcal{A}$ be a routine of our computation model and consider the previously introduced parameterized circuit β. Let $\mathcal{N}$ be a constructible subset of $\mathcal{M}$ and suppose that β is an admissible input for the routine $\mathcal{A}$. Then $\mathcal{A}$ produces on input β a parameterized arithmetic output circuit with parameter domain $\mathcal{M}$ which we denote by $\mathcal{A}(\beta)$. In order to formulate for the routine $\mathcal{A}$ the requirement of well behavedness under restriction of the inputs, we must be able to restrict β and $\mathcal{A}(\beta)$ to $\mathcal{N}$. Thus β and $\mathcal{A}(\beta)$ should be *robust*, $\beta_{\mathcal{N}}$ should be an admissible input for $\mathcal{A}$ and $\mathcal{A}(\beta_{\mathcal{N}})$ should be robust too.

Our first architectural requirement on the routine $\mathcal{A}$ may now be formulated as follows:

> *The parameterized arithmetic circuit $\mathcal{A}(\beta_{\mathcal{N}})$ can be recovered from $\mathcal{A}(\beta)$ by restriction to $\mathcal{N}$ and circuit reduction.*

We call this requirement *well behavedness under restrictions*.

The routine $\mathcal{A}$ performs with the parameterized arithmetic circuit β a transformation whose crucial feature is that only nodes which depend on the inputs $X_1, \ldots, X_n$ of β become modified, whereas parameter nodes remain substantially conserved. This needs an explanation.

Suppose that β has t essential parameter nodes. Then the essential parameters (intermediate results) of β associated with these nodes define a geometrically robust constructible map $\theta : \mathcal{M} \to \mathbb{A}^t$. The image $\mathcal{T}$ of θ is a constructible subset of $\mathbb{A}^t$. We require now that, as far as $\mathcal{A}$ performs arithmetic operations with parameters of β, $\mathcal{A}$ does it only with essential ones, and that all essential parameters of $\mathcal{A}(\beta)$ are obtained in this way. Further we require that there exists a geometrically robust constructible map ν defined on $\mathcal{T}$ (e.g. a polynomial map) such that the results of these arithmetic operations occur as entries of the composition map $\nu \circ \theta$. For this we make use of the following elementary fact.

Lemma 14 *Let notations and assumptions be as before. Then $\nu \circ \theta$ is a geometrically robust, constructible map defined on $\mathcal{M}$.*

Our basic construction method of routines will be recursion. A routine of our computation model which can be obtained in this way is called *recursive*.

Suppose now that $\mathcal{A}$ is a recursive routine of our computation model. Then $\mathcal{A}$ should be organized in such a way that for each internal node ρ of β, which depends on at least one input, there exists a set of nodes of $\mathcal{A}(\beta)$, also denoted by ρ, with the following property:

> the elements of the set ρ of nodes of $\mathcal{A}(\beta)$ represent the outcome of the action of $\mathcal{A}$ at the node ρ of β.

We fix now a node ρ of β which depends on at least one input. Let G_ρ be the intermediate result associated with the node ρ of β and let F_ρ be a vector whose entries are composed by the intermediate results of $\mathcal{A}(\beta)$ at the nodes contained in the set ρ of nodes of $\mathcal{A}(\beta)$. Thus F_ρ is a vector of rational functions in a suitable tuple of (standard) variables, say X'. Figure 28 illustrates this description.

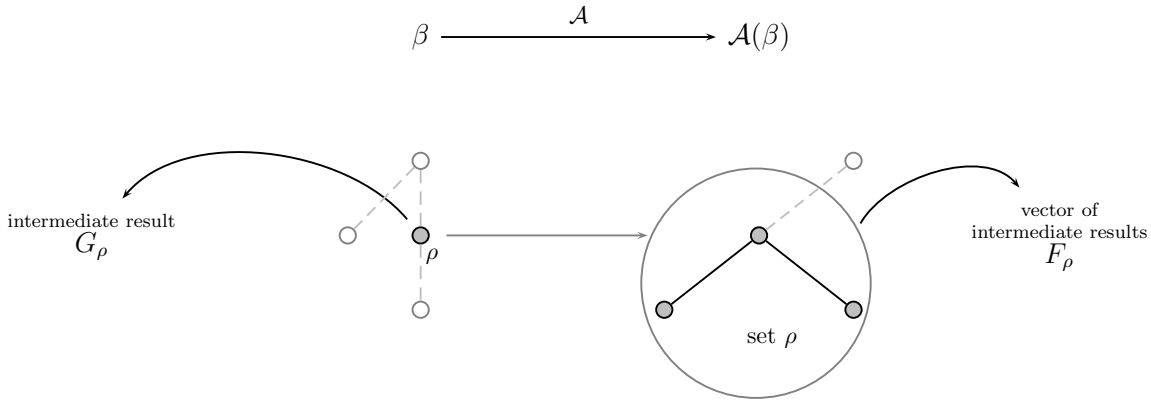


Figure 28: Intermediate results G_ρ and F_ρ

Recall that by assumption β and $\mathcal{A}(\beta)$ are robust parameterized arithmetic circuits with parameter domain $\mathcal{M}$. Therefore we deduce from Lemma 13 that G_ρ and the entries of F_ρ are in fact polynomials in $X_1, \ldots, X_n$ and X', respectively, and that this coefficients are geometrically robust functions defined on $\mathcal{M}$.

As part of our second and main requirement of our computation model we demand now that $\mathcal{A}$ satisfies at the node ρ of β the following isoparametricity condition:

(i) for any two parameter instances u_1 and u_2 of $\mathcal{M}$ the assumption

$$G_\rho(u_1, X_1, \ldots, X_n) = G_\rho(u_2, X_1, \ldots, X_n)$$

implies

$$F_\rho(u_1, X') = F_\rho(u_2, X').$$

Let θ_ρ be the coefficient vector of G_ρ and observe that θ_ρ is a geometrically robust constructible map defined on $\mathcal{M}$, whose image, say $\mathcal{T}_\rho$, is an irreducible constructible subset of a suitable affine space.

Since the first order theory of the algebraically closed field $\mathbb{C}$ admits quantifier elimination, one concludes easily that condition (i) is satisfied if and only if there exists a constructible map σ_ρ defined on $\mathcal{T}_\rho$ such that the composition map $\sigma_\rho \circ \theta_\rho$ (which is also constructible) represents the coefficient vector of (all entries of) F_ρ.

In the sequel we shall need that the dependence σ_ρ of the coefficient vector of F_ρ on the coefficient vector of G_ρ is in some stronger sense uniform (and not just constructible). Therefore we include the following condition in our requirement:

(ii) the constructible map σ_ρ is geometrically robust.

The map σ_ρ is uniquely determined by condition (i). Moreover, the map σ_ρ depends on the (combinatorial) labelled DAG structure of β below the node ρ, but not in direct form on the basic parameters $\pi_1, \ldots, \pi_r$. This is the essence of the isoparametric nature of conditions (i) and (ii). We shall therefore require that our recursive routine is *isoparametric* in this sense, i.e. that $\mathcal{A}$ satisfies conditions (i) and (ii) at any internal node ρ of β which depends at least on one input.

Observe that the geometrically robust constructible map σ_ρ (which depends on β as well as on ρ) is not an artifact, but emerges naturally from the recursive construction of a circuit semantic within the paradigm of object oriented programming. To explain this, let notations and assumptions be as before and suppose that $\mathcal{A}$ is a isoparametric recursive routine of our model and that we apply $\mathcal{A}$ to the robust parameterized arithmetic circuit β. Let ρ again be a node of β which depends at least on one input. Let u be a parameter instance of $\mathcal{M}$ and denote by $\beta^{(u)}$, $G_\rho^{(u)}$, $\mathcal{A}(\beta)^{(u)}$ and $F_\rho^{(u)}$ the instantiations of β, G_ρ, $\mathcal{A}(\beta)$ and F_ρ at u. Then the intermediate results of $\mathcal{A}(\beta)^{(u)}$ contained in $F_\rho^{(u)}$ depend only on the intermediate result $G_\rho^{(u)}$ of $\beta^{(u)}$ and not on the parameter instance u itself. In this spirit we may consider the sets $\Gamma_\rho := \{G_\rho^{(u)} \ ; \ u \in \mathcal{M}\}$ and $\Phi_\rho := \{F_\rho^{(u)} \ ; \ u \in \mathcal{M}\}$ as abstract data types and β and $\mathcal{A}(\beta)$ as syntactic descriptions of two abstraction functions which associate to any concrete object $u \in \mathcal{M}$ the abstract objects $G_\rho^{(u)}$ and $F_\rho^{(u)}$, respectively. The identity map $id_\mathcal{M} : \mathcal{M} \to \mathcal{M}$ induces now an *abstract function* [Mey00] from Γ_ρ to Φ_ρ, namely $\sigma_\rho : \Gamma_\rho \to \Phi_\rho$. In this terminology, $id_\mathcal{M}$ is just an implementation of σ_ρ. If we now consider that each recursive step of the routine $\mathcal{A}$ on input β has to be realized by another routine of the object oriented programming paradigm, we arrive to a situation where a geometrically robust constructible map $\sigma_\rho : \Gamma_\rho \to \Phi_\rho$ necessarily arises.

In terms of Hoare logic we may interpret the map $\sigma_\rho : \Gamma_\rho \to \Phi_\rho$ also as a (part of a) specification of the recursive routine $\mathcal{A}$. The map σ_ρ may be thought as an operational specification which determines F_ρ in function of G_ρ. A weaker specification would be a descriptive one which relates G_ρ and F_ρ without determining F_ρ from G_ρ completely. Hence the existence of a geometrically robust constructible map $\sigma_\rho : \Gamma_\rho \to \Phi_\rho$ is closely related to expressiveness of the recursive routine $\mathcal{A}$ in the sense of Hoare logic. We shall turn back to this subject later.

We meet a similar situation in primitive recursion over $\mathbb{N}$. The value of a univariate primitive recursive function at a natural number k depends only on the unary or bit representation of k, i.e. on the number k itself, but not on the way how k was previously generated.

In order to motivate the requirement that the recursive routine $\mathcal{A}$ should be isoparametric, we shall consider the following condition for recursive routines which we call *well behavedness under reductions*.

We only outline here this condition and leave the details until Section 6.4.1.

Suppose now that we apply a reduction procedure to the robust parameterized input circuit β producing thus another robust, by $\mathcal{M}$ parameterized circuit β^* which computes the same final results as β. Then the reduced circuit β^* should also be an admissible input for the routine $\mathcal{A}$. We call the recursive routine $\mathcal{A}$ *well behaved under reductions* if on input β it is possible to extend the given reduction procedure to the output circuit $\mathcal{A}(\beta)$ in such a way, that the extended reduction procedure, applied to $\mathcal{A}(\beta)$, reproduces the circuit $\mathcal{A}(\beta^*)$.

Obviously well behavedness under reductions limits the structure of $\mathcal{A}(\beta)$. Later in Section 6.4.1 we shall see that, cum grano salis, any recursive routine, which is well behaved under restrictions and reductions, is necessarily isoparametric. Since well behavedness under restrictions and reductions are very natural quality attributes for routines which transform robust parameterized arithmetic circuits, the weaker requirement, namely that recursive routines should be isoparametric, turns out to be well motivated.

In Section 6.4.1 we shall formally introduce a simplified version of our branching–free computation model. We postpone for then the precise definition of the notion of well behavedness under reductions.

In the sequel we shall always tacitly assume the following properties of our computation model.

(i) All parameterized arithmetic circuits are robust and have irreducible parameter domain.

(ii) If $\mathcal{A}$ is a routine of our computation model admitting the robust parameterized arithmetic circuit β as input, and if β^* is a reduction or a restriction to a constructible subset of the parameter domain of β, then β^* is also an admissible input for $\mathcal{A}$.

There exists a second reason to restrict the recursive routines of our computation model to isoparametric ones. Isoparametric recursive routines have considerable advantages for program specification and verification by means of Hoare Logics [Apt81]. We shall come back to this issue in Section 6.4.1.

6.4.1 The simplified, branching–free computation model

In this section we shall distinguish sharply between the notions of input variable and parameter and the corresponding categories of circuit nodes.

Input variables called "standard", will occur in parameterized arithmetic circuits and generic computations. The input variables of generic computations will appear subdivided in sorts, namely as "parameter", "argument" and "standard" input variables.

In order to further the transparency of our description, we shall refrain from exposing a possible, more general computation model which avoids the mentioned sharp distinction between input variables and parameters. The number of variable sorts may increase indefinitely in this generalized computation model. However, our simplified model will be comprehensive enough that a limited extension of it is able to capture the essence of all known circuit based elimination algorithms in effective algebraic geometry and, mutatis mutandis, also of all other (linear algebra and truncated rewriting) elimination procedures (see Sections 6.4.2, 6.5, [Mor03], [Mor05], and the references cited therein, and for truncated rewriting methods especially [DFGS91]). The only algorithm from symbolic arithmetic circuit manipulation which will escape from our model is the Baur–Strassen gradient computation [BCS97], Chapter 7.2.

Our simplified branching–free computation model will assume different *shapes*, each shape being determined by a finite number of a priori given *discrete* (i.e. by tuples of natural numbers indexed) families of generic computations. The labels of the inputs of the ordinary arithmetic circuits which represent these generic computations will become subdivided into *parameter*, *argument* and *standard* input variables. We shall use the letters like $U, U', U'', \ldots$ and W, W', W'' to denote vectors of parameters, $Y, Y', Y'', \ldots$ and Z, Z', Z'' to denote vectors of argument and $X, X', X'', \ldots$ to denote vectors of standard input variables (see Section 6.3).

We shall not write down explicitly the indexations of our generic computations by tuples of natural numbers. Generic computations will simply be distinguished by subscripts and superscripts, if necessary.

Ordinary arithmetic circuits of the form

$$\begin{array}{lll} R_{X_1}(W_1; X^{(1)}), & R_{X_2}(W_2; X^{(2)}), & \dots \\ R'_{X_1}(W_{1'}; X^{(1')}), & R'_{X_2}(W_{2'}; X^{(2')}), & \dots \\ \dots & \dots & \dots \end{array}$$

represent a first type of a discrete family of generic computations (for each variable $X_1, X_2, \dots, X_n$ we suppose to have at least one generic computation). Other types of families of generic computations are of the form

$$\begin{array}{llll} R_+(W; U, Y; X), & R_+(W'; U', Y'; X'), & R_+(W''; U'', Y''; X'') & \dots \\ R_{\cdot/}(W; U, Y; X), & R_{\cdot/}(W'; U', Y'; X'), & R_{\cdot/}(W''; U'', Y''; X'') & \dots \\ R_{add}(W; Y, Z; X), & R_{add}(W'; Y', Z'; X'), & R_{add}(W''; Y'', Z''; X'') & \dots \\ R_{mult}(W; Y, Z; X), & R_{mult}(W'; Y', Z'; X'), & R_{mult}(W''; Y'', Z''; X'') & \dots \end{array}$$

and

$$R_{div}(W; Y, Z; X), \quad R_{div}(W'; Y', Z'; X'), \quad R_{div}(W''; Y'', Z''; X'') \quad \dots.$$

Here the subscripts refer to addition of, and multiplication or division by a parameter (or scalar) and to essential addition, multiplication and division. A final type of families of generic computations is of the form

$$R(W; Y; X), \;\; R'(W'; Y'; X'), \;\; R''(W''; Y''; X''), \dots$$

We recall from the beginning of Section 6.4 that the objects handled by the routines of any shape of our computation model will always be robust parameterized arithmetic circuits with irreducible parameter domains. The inputs of these circuits will only consist of standard variables.

From now on we have in mind a previously fixed shape when we refer to our computation model. We start with a given finite set of discrete families of generic computations which constitute a shape as described before.

Of course, the *identity routine* which transforms each given circuit into itself, belongs to our model. Let β be a robust parameterized arithmetic circuit with parameter domain $\mathcal{M}$. The *constant* routine determined by β assigns to each robust parameterized arithmetic circuit, which has also parameter domain $\mathcal{M}$, the circuit β.

Let γ_1 and γ_2 be two robust parameterized arithmetic circuits with parameter domain $\mathcal{M}$ and suppose that there is given a one–to–one correspondence λ which identifies the output nodes of γ_1 with the input nodes of γ_2 (thus they must have the same number). Using this identification we may now join the circuit γ_1 with the circuit γ_2 in order to obtain a new parameterized arithmetic circuit $\gamma_2 *_\lambda \gamma_1$ with parameter domain $\mathcal{M}$. The circuit $\gamma_2 *_\lambda \gamma_1$ has the same, input nodes as γ_1 and the same output nodes as γ_2 and one deduces easily from Lemma 13 that $\gamma_2 *_\lambda \gamma_1$ is robust. The semantics is clear: if consistent, the circuit $\gamma_2 *_\lambda \gamma_1$ represents a composition of the rational functions defined by γ_2 and γ_1. The circuit $\gamma_2 *_\lambda \gamma_1$ is called the *join* of γ_1 with γ_2.

Suppose that there are given two routines $\mathcal{A}$ and $\mathcal{B}$ of our computation model which applied to an input circuit β with parameter domain $\mathcal{M}$ produce as outputs two circuits $\mathcal{A}(\beta)$ and $\mathcal{B}(\beta)$ (also with parameter domain $\mathcal{M}$). Suppose that there is given a one–to–one correspondence λ which identifies the output nodes of $\mathcal{A}(\beta)$ with the input nodes of $\mathcal{B}(\beta)$. Then we may join $\mathcal{A}(\beta)$ and $\mathcal{B}(\beta)$ to the circuit $\mathcal{B}(\beta) *_\lambda \mathcal{A}(\beta)$.

The corresponding routine belongs also to our computation model and is called a join of $\mathcal{A}$ with $\mathcal{B}$. Since each circuit may be interpreted as the output of a constant routine, we may interpret the join of two circuits as the join of two constant routines.

Observe that the final results of a given parameterized arithmetic circuit may constitute a vector of parameters. The join of such a circuit with another circuit, say β, is called an *evaluation* of β. Hence, mutatis mutandis, the notion of join of two routines includes also the case of evaluations of circuits. All these operations with routines lead to new routines which on a given (robust) input circuit produce new robust circuits.

Our list of simple operations with routines of our computation model may be extended, but we shall not do this here because this is not relevant for the following considerations.

A fundamental issue is how we recursively transform a given input circuit into another one with the same parameter domain. During such a transformation we make an iterative use of previously fixed generic computations. On their turn these determine the corresponding *recursive routine* of our computation model.

We consider again our input circuit β. We suppose that we have already chosen for each node ρ, which depends at least on one of the input variables $X_1, \ldots, X_n$, a generic computation

$$R_{X_i}^{(\rho)}(W_\rho; X^{(\rho)}),$$

$$R_{+}^{(\rho)}(W_\rho; U_\rho, Y_\rho; X^{(\rho)}),$$

$$R_{./}^{(\rho)}(W_\rho; U_\rho, Y_\rho; X^{(\rho)}),$$

$$R_{add}^{(\rho)}(W_\rho; Y_\rho, Z_\rho; X^{(\rho)}),$$

$$R_{mult}^{(\rho)}(W_\rho; Y_\rho, Z_\rho; X^{(\rho)}),$$

$$R_{div}^{(\rho)}(W_\rho; Y_\rho, Z_\rho; X^{(\rho)}),$$

and that this choice was made according to the label of ρ, namely $X_i, 1 \leq i \leq n$, or addition of, or multiplication or division by an essential parameter, or essential addition, multiplication or division. Here we suppose that U_ρ is a single variable, whereas W_ρ, Y_ρ, Z_ρ and $X^{(\rho)}$ may be arbitrary vectors of variables.

Furthermore we suppose that we have already precomputed for each node ρ of β, which depends at least on one input, a vector w_ρ of geometrically robust constructible functions defined on $\mathcal{M}$. If ρ is an input node we assume that w_ρ is a vector of complex numbers. Moreover, we assume that the length of w_ρ equals the length of the variable vector W_ρ. We call the entries of w_ρ the *parameters at the node* ρ of the routine $\mathcal{A}$ applied to the input circuit β.

We are now going to develop the routine $\mathcal{A}$ step by step. The routine $\mathcal{A}$ takes over all computations of β which involve only parameter nodes, without modifying them. Then $\mathcal{A}$ replaces each node ρ of β which is labelled by an input variable $X_i, 1 \leq i \leq n$, by the ordinary arithmetic circuit $R_{X_i}^{(\rho)}(w_\rho; X^{(\rho)})$ over $\mathbb{C}$ which is obtained by substituting in the generic computation $R_{X_i}^{(\rho)}(W_\rho; X^{(\rho)})$ for the vector of parameter variables W_ρ the vector of complex numbers w_ρ.

Consider now an arbitrary internal node ρ of β which depends at least on one input. The node ρ has two ingoing edges which come from two other nodes of β, say ρ_1 and ρ_2.

Suppose that the routine $\mathcal{A}$, on input β, has already computed two results, namely F_{ρ_1} and F_{ρ_2}, corresponding to the nodes ρ_1 and ρ_2. Suppose inductively that these results are vectors of polynomials depending on those standard input variables that occur in the vectors of the form $X^{(\rho')}$, where ρ' is any predecessor node of ρ. Furthermore, we assume that the coefficients of these polynomials constitute the entries of a geometrically robust, constructible map defined on $\mathcal{M}$. Finally we suppose that the lengths of the vectors F_{ρ_1} and Y_ρ (or U_ρ) and F_{ρ_2} and Z_ρ coincide.

The parameter vector w_ρ of the routine $\mathcal{A}$ forms a geometrically robust, constructible map defined on $\mathcal{M}$, whose image we denote by $\mathcal{K}_\rho$. Observe that $\mathcal{K}_\rho$ is a constructible subset of an affine space of the same dimension as the length of the vectors w_ρ and W_ρ. Denote by κ_ρ the vector of the restrictions to $\mathcal{K}_\rho$ of the canonical projections of this affine space. We consider $\mathcal{K}_\rho$ as a new parameter domain with basic parameters κ_ρ. For the sake of simplicity we suppose that the node ρ is labelled by a multiplication. Thus the corresponding generic computation has the form

$$R^{(\rho)}_{\cdot/}(W_\rho; U_\rho, Y_\rho; X^{(\rho)}) \tag{12}$$

or

$$R^{(\rho)}_{mult}(W_\rho; Y_\rho, Z_\rho; X^{(\rho)}). \tag{13}$$

Let $R^{(\rho)}_{\cdot/}(\kappa_\rho, U_\rho, Y_\rho, X^{(\rho)})$ and $R^{(\rho)}_{mult}(\kappa_\rho, Y_\rho, Z_\rho, X^{(\rho)})$ be the by $\mathcal{K}_\rho$ parameterized arithmetic circuits obtained by substituting in the generic computations (12) and (13) for the vector of parameter variables W_ρ the basic parameters κ_ρ. We shall now make at the node ρ the following requirement on the routine $\mathcal{A}$ applied to the input circuit β:

(A) The by $\mathcal{K}_\rho$ parameterized arithmetic circuit which corresponds to the current case, namely

$$R^{(\rho)}_{\cdot/}(\kappa_\rho; U_\rho, Y_\rho; X^{(\rho)})$$

or

$$R^{(\rho)}_{mult}(\kappa_\rho; Y_\rho, Z_\rho; X^{(\rho)}),$$

should be consistent and robust.

Observe that the requirement (A) is automatically satisfied if all the generic computations of our shape are realized by totally division–free ordinary arithmetic circuits.

Assume now that the routine $\mathcal{A}$ applied to the circuit β satisfies the requirement (A) at the node ρ of β.

Let us first suppose that the node ρ is labelled by a multiplication involving an essential parameter. Recall that in this case we assumed earlier that the length of the vector F_{ρ_1} is one, that F_{ρ_1} is an essential parameter of β and that the vectors F_{ρ_2} and Y_ρ have the same length. Joining now to the generic computation $R^{(\rho)}_{\cdot/}(W_\rho; U_\rho, Y_\rho; X^{(\rho)})$ at W_ρ, U_ρ and Y_ρ the previous computations of w_ρ, F_{ρ_1} and F_{ρ_2}, we obtain a parameterized arithmetic circuit with parameter domain $\mathcal{M}$, whose final results are the entries of a polynomial vector which we denote by F_ρ.

Suppose now that the node ρ is labelled by an essential multiplication. Recall again that in this second case we assumed earlier the vectors F_{ρ_1} and Y_ρ and F_{ρ_2} and Z_ρ have the same length. Joining to the generic computation

$$R^{(\rho)}_{mult}(W_\rho; Y_\rho, Z_\rho; X^{(\rho)})$$

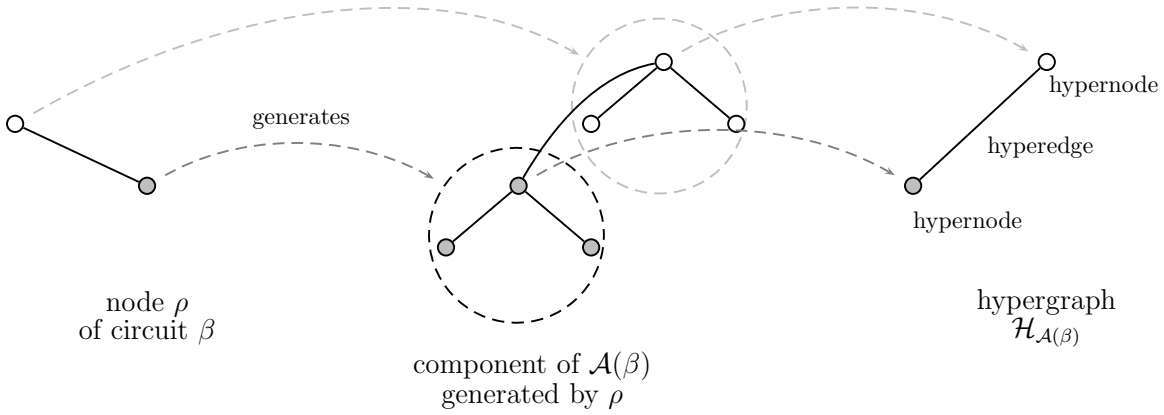
at W_ρ, Y_ρ and Z_ρ the previous computations of w_ρ, F_{ρ_1} and F_{ρ_2} we obtain also a parameterized arithmetic circuit with parameter domain $\mathcal{M}$, whose final results are the entries of a vector which we denote again by F_ρ.

One deduces easily from our assumptions on w_ρ, F_{ρ_1} and F_{ρ_2} and from the requirement (A) in combination with Lemma 13 and 14, that in both cases the resulting parameterized arithmetic circuit is consistent and robust. The other possible labellings of the node ρ by arithmetic operations are treated similarly. In particular, in case that ρ is an input node labelled by the variable $X_i, 1 \leq i \leq n$, the requirement (A) says that the ordinary arithmetic circuit $R_{X_i}^{(\rho)}(w_\rho; X^{(\rho)})$ is consistent and that all its intermediate results are polynomials in $X^{(\rho)}$ over $\mathbb{C}$ (although $R_{X_i}^{(\rho)}(w_\rho; X^{(\rho)})$ may contain divisions).

In view of our comments in the first part of Section 6.4, we call the recursive routine $\mathcal{A}$ (on input β) *well behaved under restrictions* if the requirement (A) is satisfied at any node ρ of β which depends at least on one input. If the routine $\mathcal{A}$ is well behaved under restrictions, then $\mathcal{A}$ transforms step by step the input circuit β into another robust arithmetic circuit, namely $\mathcal{A}(\beta)$, with parameter domain $\mathcal{M}$.

As a consequence of the recursive structure of $\mathcal{A}(\beta)$, each node ρ of β generates a subcircuit of $\mathcal{A}(\beta)$ which we call the *component of* $\mathcal{A}(\beta)$ *generated by* ρ. The output nodes of each component of $\mathcal{A}(\beta)$ form the hypernodes of a hypergraph $\mathcal{H}_{\mathcal{A}(\beta)}$ whose hyperedges are given by the connections of the nodes of $\mathcal{A}(\beta)$ contained in distinct hypernodes of $\mathcal{H}_{\mathcal{A}(\beta)}$. The hypergraph $\mathcal{H}_{\mathcal{A}(\beta)}$ may be shrinked to the DAG structure of β and therefore we denote the hypernodes of $\mathcal{H}_{\mathcal{A}(\beta)}$ in the same way as the nodes of β. Notice that well behavedness under restrictions is in fact a property which concerns the hypergraph $\mathcal{H}_{\mathcal{A}(\beta)}$. Figure 29 illustrates this description.

Figure 29: Generation of the hypergraph $\mathcal{H}_{\mathcal{A}(\beta)}$



We call $\mathcal{A}$ a (recursive) *parameter routine* if $\mathcal{A}$ does not introduce new standard variables. In the previous recursive construction of the routine $\mathcal{A}$, the parameters at the nodes of β, used for the realization of the circuit $\mathcal{A}(\beta)$, are supposed to be generated by recursive parameter routines.

We are now going to consider another requirement of our recursive routine $\mathcal{A}$, which will lead us to the notion of isoparametricity of $\mathcal{A}$.

Let us turn back to the previous situation at the node ρ of the input circuit β. Notations and assumptions will be the same as before. From Lemma 13 we deduce that the intermediate result of β associated with the node ρ, say G_ρ, is a polynomial in $X_1, \ldots, X_n$ whose coefficients form the entries of a geometrically robust, constructible map defined on $\mathcal{M}$, say θ_ρ. Let $\mathcal{T}_\rho$ be the image of this map and observe that $\mathcal{T}_\rho$ is a constructible subset of a suitable affine space. The intermediate results of the circuit $\mathcal{A}(\beta)$ at the elements of the hypernode ρ of $\mathcal{H}_{\mathcal{A}(\beta)}$ constitute a polynomial vector which we denote by F_ρ.

We shall now make another requirement at the node ρ on the routine $\mathcal{A}$ applied to the input circuit β:

(B) There exists a geometrically robust, constructible map σ_ρ defined on $\mathcal{T}_\rho$ such that $\sigma_\rho \circ \theta_\rho$ constitutes the coefficient vector of F_ρ.

In view of the comments made at the beginning of Section 6.4 we call the recursive routine $\mathcal{A}$ *isoparametric* (on input β) if requirement (B) is satisfied at any node ρ of β which depends at least on one input.

Let assumptions and notations be as before and consider again the node ρ of the circuit β. Assume that the recursive routine $\mathcal{A}$ is well behaved under restrictions and that $\mathcal{A}$, applied to the circuit β, fulfills the requirement (B) at ρ. Then the topological robustness (which is a consequence of the geometrical robustness) of σ_ρ implies that the following condition is satisfied:

(B') Let $(u_k)_{k\in\mathbb{N}}$ be a (not necessarily convergent) sequence of parameter instances $u_k \in \mathcal{M}$ and let $u \in \mathcal{M}$ such that $(\theta_\rho(u_k))_{k\in\mathbb{N}}$ converges to $\theta_\rho(u)$. Denote by τ_ρ the coefficient vector of F_ρ. Then τ_ρ is a geometrically robust constructible map defined on $\mathcal{M}$ and the sequence $(\tau_\rho(u_k))_{k\in\mathbb{N}}$ is bounded.

Suppose now that the recursive routine $\mathcal{A}$ is well behaved under restrictions and satisfies instead of (B) only condition (B') at the node ρ of β. Let $u \in \mathcal{M}$ be an arbitrary parameter instance. Then τ_ρ takes on the set $\{u' \in \mathcal{M}; \theta_\rho(u') = \theta_\rho(u)\}$ only finitely many values whose number is a priori bounded. Moreover, for $\mathfrak{M}_u$ being the vanishing ideal of the $\mathbb{C}$–algebra $\mathbb{C}[\theta_\rho]$ at $\theta_\rho(u)$, the entries of τ_ρ are integral over the local $\mathbb{C}$–algebra $\mathbb{C}[\theta_\rho]_{\mathfrak{M}_u}$ (the argument for that relies on Zariski's Main Theorem and is exhibited in [CGH$^+$03], Sections 3.2 and 5.1). This algebraic characterization implies that for given $u \in \mathcal{M}$ all the sequences $(\tau_\rho(u_k))_{k\in\mathbb{N}}$ of condition (B') have only finitely many distinct accumulation points. This shows that requirement (B) and condition (B') are closely related.

On the other hand, suppose that condition (B') is not satisfied at the node ρ of β. Then following Definition 7 in Section 2.3.2 the geometrically robust constructible map τ_ρ is indetermined by θ_ρ. Since our model is aimed to be branching–free at this stage we suppose that membership to the constructible set $\mathcal{N}_\rho := \{(\theta_\rho(u), \tau_\rho(u)); u \in \mathcal{M}\}$ can be decided by a branching–free algorithm. Moreover we suppose that $\mathcal{M}$ is locally closed in its ambient space $\mathbb{A}^r$. Then the set $\mathcal{N}_\rho$ is locally closed in its ambient space. Therefore Lemma 11 in Section 2.3.2 may be applied to conclude that there exists a parameter instance $u \in \mathcal{M}$ and a sequence $(u_k)_{k\in\mathbb{N}}$ with $u_k \in \mathcal{M}$ and $\theta_\rho(u_k) = \theta_\rho(u)$ such that the sequence $(\tau_\rho(u_k))_{k\in\mathbb{N}}$ is unbounded. In particular the image of $\{u \in \mathcal{M}; \theta_\rho(u') = \theta_\rho(u)\}$ under one of the entries of τ_ρ is a cofinite set of complex numbers. This shows that τ_ρ is highly indetermined by θ_ρ at u.

We shall see later that this situation cannot occur if the recursive routine $\mathcal{A}$ is expressive in the sense of Hoare Logics adapted to our computation model.

Adopting the terminology of [GHMS11] (see also Section 4.2.3 of this thesis) we call $\mathcal{A}$ *coalescent* (on input β), if $\mathcal{A}$ satisfies condition (B') for any node ρ of β. Thus isoparametricity implies coalescence for $\mathcal{A}$, but not vice versa. Nevertheless the notions of isoparametricity and coalescence become quite close for recursive routines which are well behaved under restrictions.

Suppose again that the recursive routine $\mathcal{A}$ is well behaved under restrictions. We call $\mathcal{A}$ well behaved under reductions (on input β) if $\mathcal{A}(\beta)$ satisfies the following requirement:

> *Let ρ and ρ' be distinct nodes of β which compute the same intermediate results. Then the intermediate results at the hypernodes ρ and ρ' of $\mathcal{H}_{\mathcal{A}(\beta)}$ are identical. Mutatis mutandis the same is true for the computation of the parameters of $\mathcal{A}$ at any node of β.*

Assume that the routine $\mathcal{A}$ is recursive and well behaved under restrictions. One verifies then easily that, taking into account the hypergraph structure $\mathcal{H}_{\mathcal{A}(\beta)}$ of $\mathcal{A}(\beta)$, any reduction procedure on β may canonically be extended to a reduction procedure of $\mathcal{A}(\beta)$.

In the first part of Section 6.4 we claimed that, cum grano salis, the requirement of well behavedness under reductions implies the requirement of isoparametricity for recursive routines. We are going now to prove this.

Let notations and assumptions be as before and let us analyse what happens to the recursive routine $\mathcal{A}$ at the node ρ of β. For this purpose we shall use the following broadcasting argument.

Recall that G_ρ and the entries of F_ρ are the intermediate results of β and $\mathcal{A}(\beta)$ associated with ρ, where ρ is interpreted as a node of the input circuit β in the first case and as a hypernode of $\mathcal{H}_{\mathcal{A}(\beta)}$ in the second one. Moreover recall that G_ρ is a polynomial in $X_1, \ldots, X_n$, that the geometrically robust, constructible map θ_ρ, defined on $\mathcal{M}$, represents the coefficient vector of G_ρ and that the irreducible constructible set $\mathcal{T}_\rho$ is the image of θ_ρ. We consider now the parameterized arithmetic circuit γ_ρ which realizes the following trivial evaluation of the polynomial G_ρ:

- compute simultaneously from $\pi_1, \ldots, \pi_r$ all entries of θ_ρ and from $X_1, \ldots, X_n$ all monomials occurring in G_ρ
- compute G_ρ as a linear combination of the monomials of G_ρ using as coefficients the entries of θ_ρ.

Since θ_ρ is a robust, constructible map defined on $\mathcal{M}$, it is easy to see that the computation of the entries of θ_ρ can be realized by a robust subcircuit of γ_ρ . Thus γ_ρ becomes a robust parameterized arithmetic circuit with parameter domain $\mathcal{M}$. Observe that γ_ρ has a single output node, say ρ', which computes the polynomial G_ρ.

Now we paste, as disjointly as possible, the circuit γ_ρ to the circuit β obtaining thus a new robust, parameterized arithmetic circuit β_ρ with parameter domain $\mathcal{M}$. Observe that β_ρ contains β and γ_ρ as subcircuits and that ρ and ρ' are distinct nodes of β_ρ which compute the same intermediate result, namely G_ρ. The entries of θ_ρ are essential parameters of γ_ρ and hence also of β_ρ. We suppose now that β_ρ is, like β, an admissible

input for the recursive routine $\mathcal{A}$. Let $F_{\rho'}$ be a vector whose entries are the intermediate results at the nodes of $\mathcal{A}(\beta_\rho)$ contained in the hypernode ρ' of $\mathcal{H}_{\mathcal{A}(\beta_\rho)}$. Analyzing now how $\mathcal{A}$ operates on the structure of the subcircuit γ_ρ of β_ρ, we see immediately that there exists a geometrically robust constructible map σ_ρ defined on $\mathcal{T}_\rho$ such that the composition map $\sigma_\rho \circ \theta_\rho$ constitutes the coefficient vector of F_ρ. Since by assumption the recursive routine $\mathcal{A}$ is well behaved under reductions and the intermediate results of β_ρ at the nodes ρ and ρ' consist of the same polynomial G_ρ, we conclude that the intermediate results at the hypernodes ρ and ρ' of $\mathcal{H}_{\mathcal{A}(\beta_\rho)}$ are also the same. Therefore we may assume without loss of generality $F_\rho = F_{\rho'}$. This implies that the geometrically robust, constructible map $\sigma_\rho \circ \theta_\rho$ constitutes the coefficient vector of F_ρ.

This proves that the recursive routine $\mathcal{A}$ satisfies, on input β and at the node ρ, the requirement (B). Since β was an arbitrary admissible input circuit for the recursive routine $\mathcal{A}$ and ρ was an arbitrary node of β which depends on at least one input, we may conclude that $\mathcal{A}$ is isoparametric. The only assumption we made to draw this conclusion was that the extended circuit β_ρ is an admissible input for the routine $\mathcal{A}$. This conclusion is however not very restrictive because β and β_ρ compute the same final results.

At the beginning of Section 6.4 we mentioned that isoparametric routines are advantageous for program specification and verification. We are now going to explain this.

Let notations and assumptions be as before and let in particular $\mathcal{A}$ be a recursive routine of our computation model which behaves well under restrictions. Assume that β is an admissible input for $\mathcal{A}$ and consider the specification language $\mathcal{L}$ introduced in Section 6.2.1. Suppose that the routine $\mathcal{A}$ is given by an asserted program Π formulated in the elementary Hoare Logics of $\mathcal{L}$ ([Apt81]). The standard model of the elementary theory of $\mathcal{L}$ provides us with the states which define the semantics of Π. The asserted program Π represents the routine $\mathcal{A}$ as a loop. Then, by its instructions, the program Π transforms node by node the labelled DAG structure of β into the labelled DAG structure of $\mathcal{A}(\beta)$.

At each step of the loop a purely syntactic action, namely a graph manipulation, takes place. This action consists of the join of two or more labelled directed acyclic graphs. Simultaneously, in order to guarantee the correctness of the program Π, a loop invariant, formulated in our specification language $\mathcal{L}$, has to be satisfied.

This involves the semantics of $\mathcal{L}$ consisting of the universe of all robust parameterized arithmetic circuits. A loop invariant as above is given by a formula $\bigwedge(\beta_1, \beta_2, \mathcal{M}_1, \mathcal{M}_2, \rho_1, \rho_2)$ of $\mathcal{L}$ containing the free variables β_1, β_2 for circuits, $\mathcal{M}_1$, $\mathcal{M}_2$ for their parameter domains and ρ_1, ρ_2 for their (hyper)nodes which become instantiated by β, $\mathcal{A}(\beta)$, $\mathcal{M}$, the node ρ of β and the hypernode ρ of $\mathcal{A}(\beta)$. The variables $U^{(1)}, \ldots, U^{(m)}, \ldots$ and the standard input variable vectors $X^{(1)}, \ldots, X^{(h)}, \ldots$ occur only bounded in $\bigwedge(\beta_1, \beta_2, \mathcal{M}_1, \mathcal{M}_2, \rho_1, \rho_2)$ and the variables $\rho_1, \ldots, \rho_l, \ldots$ occur all bounded except two, namely ρ_1 and ρ_2.

For $\pi := (\pi_1, \ldots, \pi_r)$ and given variables X, X' and ρ expressing a parameter instantiation, the input variable vectors of β and $\mathcal{A}(\beta)$ and a node of β, we denote by $G_\rho(\beta; \pi; X)$ and $F_\rho(\mathcal{A}(\beta); \pi; X')$ the function symbols (or vectors of them) which express the intermediate results of β or $\mathcal{A}(\beta)$ corresponding to ρ.

We require now that any formula of $\mathcal{L}$ built up by $G_{\rho_1}, \ldots, G_{\rho_l}$ and $F_{\rho'_1}, \ldots, F_{\rho'_{l'}}$, and containing only β, $\mathcal{M}$ and ρ_1 as free variables is equivalent to a formula built up only by $G_{\rho_1}, \ldots, G_{\rho_l}$ and $G_{\rho'_1}, \ldots, G_{\rho'_{l'}}$. This implies that in $\mathcal{L}$ the intermediate result F_ρ of

$\mathcal{A}(\beta)$ is definable in terms of the intermediate result G_ρ of β. Applied to the node ρ of the concrete circuit β with parameter domain $\mathcal{M}$, this means that for θ_ρ and τ_ρ being the coefficient vectors of $G_\rho(\beta, \pi, X)$ and $F_\rho(\mathcal{A}(\beta), \pi, X')$ and $\mathcal{T}_\rho$ being the image of θ_ρ, there exists a constructible map σ_ρ with domain of definition $\mathcal{T}_\rho$ such that $\tau_\rho = \sigma_\rho \circ \theta_\rho$ holds. In particular, for $u', u'' \in \mathcal{M}$ the assumption $\theta_\rho(u') = \theta_\rho(u'')$ implies $\tau_\rho(u') = \tau_\rho(u'')$. Hence for any $u \in \mathcal{M}$ and any sequence $(u_k)_{k\in\mathbb{N}}$ with $u_k \in \mathcal{M}$ and $\theta_\rho(u_k) = \theta_\rho(u)$ the sequence $(\tau_\rho(u_k))_{k\in\mathbb{N}}$ is bounded. If $\mathcal{M}$ is locally closed in $\mathbb{A}^r$ we deduce now from our previous argumentation that $\mathcal{A}$ satisfies condition (B') at the node ρ of β.

Let $\beta_1, \beta_2, \mathcal{M}_1, \mathcal{M}_2$ and ρ_1, ρ_2 be variables for robust parameterized arithmetic circuits, their parameter domains and their (hyper)nodes. We assume that for any formula $\Phi(\beta_1, \mathcal{M}_1)$ of $\mathcal{L}$ in the free variables β_1 and $\mathcal{M}_1$ there exist two formulas

$$\Psi(\beta_1, \beta_2, \mathcal{M}_1, \mathcal{M}_2, \rho_1, \rho_2) \text{ and } \Omega(\beta_1, \beta_2, \mathcal{M}_1, \mathcal{M}_2, \rho_1, \rho_2)$$

in the free variables $\beta_1, \beta_2, \mathcal{M}_1, \mathcal{M}_2, \rho_1, \rho_2$ such that for any concrete, for $\mathcal{A}$ admissible circuit β with parameter domain $\mathcal{M}$ and basic parameter vector π and for any node ρ of β the following two conditions are satisfied:

(*i*) the validity of $\Phi(\beta, \mathcal{M})$ entails that $\Psi(\beta, \mathcal{A}(\beta), \mathcal{M}, \mathcal{M}, \rho, \rho')$ implies all existing, in $\mathcal{L}$ expressible relations between $G_\rho(\beta, \pi, X)$ and $F_\rho(\mathcal{A}(\beta), \pi, X')$ (here ρ' is the hypernode ρ of $\mathcal{A}(\beta)$ and X and X' are vectors of the input variables of β and $\mathcal{A}(\beta)$).

(*ii*) $\Omega(\beta, \mathcal{A}(\beta), \mathcal{M}, \mathcal{M}, \rho, \rho')$ determines the polynomial $F_\rho(\mathcal{A}(\beta), \pi, X')$ in terms of $G_\rho(\beta, \pi, X)$.

If $\mathcal{L}$ and $\mathcal{A}$ satisfy this assumption we say in the spirit of Hoare Logics that $\mathcal{L}$ is *expressive* for the routine $\mathcal{A}$.

From our previous considerations we conclude that expressivity of $\mathcal{L}$ for $\mathcal{A}$ implies coalescence of $\mathcal{A}$ for input circuits β whose parameter domains are locally closed in their ambient spaces. Moreover admissibility for $\mathcal{A}$ is given by a precondition which can be expressed by a formula in $\mathcal{L}$ which contains only two free variables, one for the input circuit and one for its parameter domain.

Suppose now that our specification language $\mathcal{L}$ is expressive for the asserted program Π, i.e. for the recursive routine $\mathcal{A}$. Then Π can be derived by means of the inference rules of Hoare Logics from the elementary theory of the specification language $\mathcal{L}$ (see [Apt81], Section 2.8 for details). Moreover, any in $\mathcal{L}$ formulated postcondition on the output circuits entails a weakest precondition on the input circuits of $\mathcal{A}$.

Let $\mathcal{A}$ and $\mathcal{B}$ be recursive routines as before and suppose that they are well behaved under restrictions and isoparametric or well behaved under reductions. Assume that $\mathcal{A}(\beta)$ is an admissible input for $\mathcal{B}$. We define the composed routine $\mathcal{B} \circ \mathcal{A}$ in such a way that $\mathcal{B}(\mathcal{A}(\beta))$ becomes the output of $\mathcal{B}\circ\mathcal{A}$ applied to the input β. Since the routines $\mathcal{A}$ and $\mathcal{B}$ are well behaved under restrictions, we see easily that $(\mathcal{B} \circ \mathcal{A})(\beta) := \mathcal{B}(\mathcal{A}(\beta))$ is a consistent, robust parameterized arithmetic circuit with parameter domain $\mathcal{M}$. From Lemma 13 and 14 we deduce that $\mathcal{B} \circ \mathcal{A}$ is a isoparametric recursive routine if $\mathcal{A}$ and $\mathcal{B}$ are isoparametric. In case that $\mathcal{A}$ and $\mathcal{B}$ are well behaved under reductions, one verifies immediately that $\mathcal{B} \circ \mathcal{A}$ is also well behaved under reductions. Therefore, under these assumptions, we shall consider $\mathcal{B} \circ \mathcal{A}$ also as a routine of our computation model.

Unfortunately the composition of two arbitrary coalescent recursive routines need not to be coalescent. Therefore we shall focus in the sequel our attention on isoparametric recursive routines as basic building blocks of the branching–free computation model we are going to introduce.

The identity and any constant routine are trivially well behaved under restrictions and reductions and in particular isoparametric.

Let $\mathcal{A}$ and $\mathcal{B}$ be two routines of our computation model and suppose for the sake of simplicity that they are recursive. Assume that the robust parameterized arithmetic circuit β is an admissible input for $\mathcal{A}$ and $\mathcal{B}$ and that there is given a one–to–one correspondence λ which identifies the output nodes of $\mathcal{A}(\beta)$ with the input nodes of $\mathcal{B}(\beta)$. Often, for a given input circuit β, the correspondence λ is clear by the context. If we limit ourselves to input circuits β where this occurs, we obtain from $\mathcal{A}$ and $\mathcal{B}$ a new routine, called their *join*, which transforms the input circuit β into the output circuit $\mathcal{B}(\beta) *_\lambda \mathcal{A}(\beta)$. Analyzing now $\mathcal{B}(\beta) *_\lambda \mathcal{A}(\beta)$, we see that the join of $\mathcal{A}$ with $\mathcal{B}$ is well behaved under restrictions in the most obvious sense. Since by assumption the routines $\mathcal{A}$ and $\mathcal{B}$ are recursive, the circuits $\mathcal{A}(\beta)$ and $\mathcal{B}(\beta)$ inherit from β a superstructure given by the hypergraphs $\mathcal{H}_{\mathcal{A}(\beta)}$ and $\mathcal{H}_{\mathcal{B}(\beta)}$. Analyzing again this situation, we see that any reduction procedure on β can be extended in a canonical way to the circuit $\mathcal{B}(\beta) *_\lambda \mathcal{A}(\beta)$. This means that the join of $\mathcal{A}$ with $\mathcal{B}$ is also well behaved under reductions. More caution is at order with the notions of isoparametricity and coalescence. In a simple minded and strict sense, the join of two isoparametric or coalescent recursive routines $\mathcal{A}$ and $\mathcal{B}$ is not necessarily isoparametric or coalescent. However the conditions (B) or (B') are still satisfied between the output nodes of β and $\mathcal{B}(\beta) *_\lambda \mathcal{A}(\beta)$. A routine with one of these two properties is called *output isoparametric* or *output coalescent*, respectively.

The *union* of the routines $\mathcal{A}$ and $\mathcal{B}$ assigns to the input circuit β the juxtaposition of $\mathcal{A}(\beta)$ and $\mathcal{B}(\beta)$. Thus, on input β, the final results of the union of $\mathcal{A}$ and $\mathcal{B}$ are the final results of $\mathcal{A}(\beta)$ and $\mathcal{B}(\beta)$ (taken separately in case of ambiguity). The union of $\mathcal{A}$ and $\mathcal{B}$ is an output isoparametric routine which behaves well under restrictions and reductions.

Observe also that for a recursive routine $\mathcal{A}$ which behaves well under restrictions and reductions the following holds: let β be a robust parameterized arithmetic circuit that broadcasts to a circuit β^* and assume that β and β^* are admissible circuits for $\mathcal{A}$. Then $\mathcal{A}(\beta)$ broadcasts to $\mathcal{A}(\beta^*)$.

From these considerations we conclude that routines, constructed as before by iterated applications of the operations recursion, composition, join and union, are still, in a suitable sense, well behaved under restrictions and output isoparametric. If only recursive routines become involved that behave well under reductions, we may also allow broadcastings at the interface of two such operations.

This remains true when we introduce, as we shall do now, in our computational model the following additional type of routine construction.

Let β be the robust, parameterized circuit considered before, and let $R(W;Y;X)$ be a generic computation belonging to our shape list. Let w be a vector of complex numbers and suppose that w and W have the same vector length. Moreover suppose that the final results of β form a vector of the same length as Y.

We denote by $R(w;Y;X)$ the ordinary arithmetic circuit over $\mathbb{C}$ obtained by substituting in the generic computation $R(W;Y;X)$ the vector of parameter variables W by w. We shall now make the following requirement:

(C) The ordinary arithmetic circuit $R(w; Y; X)$ should be consistent and robust.

Observe that requirement (C) is obsolete when $R(W; Y; X)$ is a totally division–free ordinary arithmetic circuit.

Suppose now that requirement (C) is satisfied. Then the new routine, say $\mathcal{B}$, applied to the circuit β, consists of joining to the generic computation $R(W; Y; X)$ and of replacing W by the vector w.

From Lemma 13 and 14 we deduce that the resulting parameterized arithmetic circuit $\mathcal{B}(\beta)$ has parameter domain $\mathcal{M}$ and is consistent and robust. One sees immediately that the routine $\mathcal{B}$ is well behaved under restrictions and reductions and is output isoparametric.

From now on we shall always suppose that all our recursive routines are isoparametric and well behaved under restrictions and that requirement (C) is satisfied when we apply this last type of routine construction.

An *elementary routine* of our simplified *branching–free computation model* is finally obtained by the iterated application of all these construction patterns, in particular the last one, recursion, composition and join. As far as only recursion becomes involved that is well behaved under reductions, we allow also broadcastings at the interface of two constructions. Of course, the identity and any constant routine belong also to our model. The set of all these routines is therefore closed under these constructions and operations.

We call an elementary routine *essentially division–free* if it admits as input only essentially division–free, robust parameterized arithmetic circuits and all generic computations used to compose it are essentially division–free. The outputs of essentially division–free elementary routines are always essentially division–free circuits. The set of all essentially division–free elementary routines is also closed under the mentioned constructions and operations.

We have seen that elementary routines are, in a suitable sense, well behaved under restrictions. In the following statement we formulate explicitly the property of an elementary routine to be output isoparametric. This will be fundamental in our subsequent complexity considerations.

Proposition 15 *Let $\mathcal{A}$ be an elementary routine of our branching–free computation model. Then $\mathcal{A}$ is output isoparametric. More explicitly, let β be a robust, parameterized arithmetic circuit with parameter domain $\mathcal{M}$. Suppose that β is an admissible input for $\mathcal{A}$. Let θ be a geometrically robust, constructible map defined on $\mathcal{M}$ such that θ represents the coefficient vector of the final results of β and let $\mathcal{T}$ be the image of θ. Then $\mathcal{T}$ is a constructible subset of a suitable affine space and there exists a geometrically robust, constructible map σ defined on $\mathcal{T}$ such that the composition map $\sigma \circ \theta$ represents the coefficient vector of the final results of $\mathcal{A}(\beta)$.*

A complete proof of this proposition is just tedious and will be omitted here. In case that $\mathcal{A}$ is a recursive routine, Proposition 15 expresses nothing but the requirement (B) applied to the output nodes of β.

Let us observe that Proposition 15 implies the following result.

Corollary 16 *Let assumptions and notations be as in Proposition 15. Then the routine $\mathcal{A}$ is output coalescent and satisfies the following condition:*

(∗) *Let u be an arbitrary parameter instance of $\mathcal{M}$ and let $\mathfrak{M}_u$ be the vanishing ideal of the $\mathbb{C}$–algebra $\mathbb{C}[\theta]$ at the point $\theta(u)$. Then the entries of the coefficient vector of the final results of $\mathcal{A}(\beta)$ are integral over the local $\mathbb{C}$–algebra $\mathbb{C}[\theta]_{\mathfrak{M}_u}$.*

The output coalescence of $\mathcal{A}$ and condition (∗) are straight–forward consequences of the output isoparametricity of $\mathcal{A}$. We remark here that condition (∗) follows already directly from the output coalescence of $\mathcal{A}$. This highlights again the close connection between isoparametricity and coalescence. The argument requires Zariski's Main Theorem. For details we refer to [CGH$^+$03], Sections 3.2 and 5.1.

Before finishing this Section 6.4.1, let us make the following two observations.

In a practical project design of an elementary routine, requirements like (A), (B) and (C) become properties of subroutines which have to be verified. It is clear that an equational theory is sufficient for this purpose. In this aim one has to work with specifications which can be expressed by a low level (concrete) object language. This language has to deal with constructible sets (and their elements), robust constructible maps defined on them and even with rational functions depending on parameters and input variables and with circuits of them. For example, the constructible sets $\mathcal{M}$ and $\mathcal{T}$ of Proposition 15 are classes of objects, namely points of suitable affine spaces, which have to be described by class invariants taking the form of boolean combinations of polynomial equations. Of course, many such class invariants are possible for given $\mathcal{M}$ and $\mathcal{T}$. They act as pre– and postconditions of a given correctness proof of the routine under consideration. Notice also that requirement (B) is particularly well suited for proofs based on induction on recursive elementary routines.

We mentioned already that our simplified branching–free computation model may be generalized.

In our simplified model the parameter vectors of an elementary recursive routine $\mathcal{A}$ applied to the input circuit β are always geometrically robust constructible maps defined on the parameter domain $\mathcal{M}$ of β. The entries of these parameter vectors may now be replaced by intermediate results of robust, parameterized arithmetic circuits with parameter domain $\mathcal{M}$. These intermediate results have to depend on inputs consisting of new parameter variables which are distinct from the standard input variables which become introduced when the routine $\mathcal{A}$ is applied to the input β. This implies that the input variables of the generalized model have to be subdivided into an infinite, hierarchical system of categories. The arguments needed to justify the description of this more general branching–free computation model are very similar to those used for the simplified one. It is not clear what could be the contribution of such a generalization and therefore we refrain from going into details.

6.4.2 The extended computation model

We are now going to extend our simplified branching–free computation model of elementary routines by a new model consisting of *algorithms* and *procedures* which may contain some limited branchings. Our description of this model will be rather informal. An algorithm will be a dynamic DAG of elementary routines which will be interpreted as pipes. At the end point of the pipes, decisions may be taken which involve only identity tests between robust constructible functions defined on the parameter domain under consideration. The output of such an identity test is a boolean vector which determines the

next elementary routine (i.e. pipe) to be applied to the output circuit produced by the preceding elementary routine (pipe). This gives now rise to a *extended computation model* which contains branchings. These branchings depend on a limited type of decisions at the level of the underlying abstract data type, namely the mentioned identity tests. We need to include this type of branchings in our extended computation model in order to capture the whole spectrum of known elimination procedures in effective algebraic geometry. Because of this limitation of branchings, we shall call the algorithms of our model *branching parsimonious* (compare [GH01] and [CGH$^+$03]). A branching parsimonious algorithm $\mathcal{A}$ which accepts a robust parameterized arithmetic circuit β with parameter domain $\mathcal{M}$ as input produces a new robust circuit $\mathcal{A}(\beta)$ with parameter domain $\mathcal{M}$ such that $\mathcal{A}(\beta)$ *does not contain any branchings*. In this sense $\mathcal{A}$ acts *uniformly* on β.

Recall that our two main constructions of elementary routines depend on a previous selection of generic computations from our shape list. This selection may be handled by calculations with the indexations of its members. We shall think that these calculations become realized by deterministic Turing machines. At the beginning, for a given robust parametric input circuit β with parameter domain $\mathcal{M}$, a tuple of fixed (i.e. of β independent) length of natural numbers is determined. This tuple constitutes an initial configuration of a Turing machine computation which determines the generic computations of our shape list that intervene in the elementary routine under construction. The entries of this tuple of natural numbers are called *invariants* of the circuit β. These invariants, whose values may also be boolean (i.e. realized by the natural numbers 0 or 1), depend mainly on algebraic or geometric properties of the final results of β. However, they may also depend on structural properties of the labelled DAG β.

For example, the invariants of β may express that β has r parameters, n inputs and outputs, (over $\mathbb{C}$) non–scalar size and depth at most L and l, that β is totally division–free and that the final results of β have degree at most $d \leq 2^l$ and form for all parameter instances a reduced regular sequence in $\mathbb{C}[X]$, where $X := (X_1, \ldots, X_n)$ and $X_1, \ldots, X_n$ are the inputs of β.

Some of these invariants (e.g. the syntactical ones like number of parameters, inputs and outputs and non–scalar size and depth) may simply be read–off from the labelled DAG structure of β. Others, like the truth value of the statement that the final results of β form a reduced regular sequence, have to be precomputed by an elimination algorithm from a previously given software library in effective commutative algebra or algebraic geometry or its value has to be fixed in advance (generally to the boolean value one) as a precondition for the elementary routine which becomes applied to β.

In the same vein we may equip any elementary routine $\mathcal{A}$ with a Turing computable function which from the values of the invariants of a given input circuit β decides whether β is admissible for $\mathcal{A}$, and, if this is the case, determines the generic computations of our shape list which intervene in the application of $\mathcal{A}$ to β.

We shall now go a step further letting depend the structure of $\mathcal{A}$ itself on the invariants of β. In the simplest case this means that we admit that the vector of invariants of β, denoted by $\mathrm{inv}(\beta)$, determines the internal structure of an elementary routine, say $\mathcal{A}_{\mathrm{inv}(\beta)}$, which admits β as input. Observe that the internal structure of the elementary routines of our computation model may be characterized by tuples of fixed length of natural numbers. We consider this characterization as an *indexation* of the elementary routines of our computation model. We may now use this indexation in order to combine

dynamically elementary routines by composition and join. Let us limit the attention to the case of composition. In this case the output circuit of one elementary routine is the input for the next routine. The elementary routines which compose this display become implemented as pipes which start with the final results of the input circuits of the routine representing the pipe and end with the final results of the output circuits of the routine. Given such a pipe and an input circuit γ for the elementary routine $\mathcal{B}$ representing the pipe, we may apply suitable identity tests to the final results of $\mathcal{B}(\gamma)$ in order to determine a boolean vector which we use to compute the index of the next elementary routine (seen as a new pipe) which will be applied to $\mathcal{B}(\gamma)$ as input.

A *low level program* of our extended computation model is now a text, namely the transition table of a deterministic Turing machine which computes a function ψ realizing the following tasks.

Let β be as before a robust parameterized arithmetic circuit. Then ψ returns first on input $\mathrm{inv}(\beta)$ a boolean value, zero or one, where one is interpreted as the informal statement "β is an admissible input". If this is the case, then ψ returns the index vector $\mathrm{inv}(\beta)$ of an elementary routine, say $\mathcal{A}_{\mathrm{inv}(\beta)}$, which admits β as input. Then ψ determines on $\mathrm{inv}(\beta)$ the identity tests which have to be realized with the final results of $\mathcal{A}_{\mathrm{inv}(\beta)}(\beta)$. The outcome of these identity tests determine an index value $\psi_1(\beta)$ corresponding to a new elementary routine $\mathcal{A}_{\psi_1(\beta)}$ which admits $\mathcal{A}_{\mathrm{inv}(\beta)}(\beta)$ as input. In this way we continue and obtain as end result an elementary routine $\mathcal{A}^{(\beta)}$, which applied to β, produces a final output circuit $\mathcal{A}^{(\beta)}(\beta)$. The function ψ represents all these index computations.

The *algorithm* represented by ψ is the partial map between robust parametric arithmetic circuits that assigns to each admissible input β the circuit $\mathcal{A}^{(\beta)}(\beta)$ as output. Observe that elementary routines are particular algorithms. If the pipes of an algorithm are all represented by essentially division–free elementary routines, we call the algorithm itself *essentially division–free*.

One sees easily that the "Kronecker algorithm" [GLS01] (compare also [GHM$^+$98], [GHH$^+$97] and [GHMP97]) may be programmed in our extended computation model. Nevertheless, the Kronecker algorithm requires more than a single elementary routine for its design. In order to understand this, recall that the Kronecker algorithm accepts as input an ordinary division–free arithmetic circuit which represents by its output nodes a reduced regular sequence of polynomials $G_1, \ldots, G_n$ belonging to $\mathbb{C}[X_1, \ldots, X_n]$. In their turn, the polynomials $G_1, \ldots, G_n$ determine a *degree pattern*, say $\Delta := (\delta_1, \ldots, \delta_n)$, with $\delta_i := \deg\{G_1 = 0, \ldots, G_i = 0\}$ for $1 \leq i \leq n$.

After putting the variables $X_1, \ldots, X_n$ in generic position with respect to $G_1, \ldots, G_n$, the algorithm performs n recursive steps to eliminate them, one after the other. Finally the Kronecker algorithm produces an ordinary arithmetic circuit which computes the coefficients of $n+1$ univariate polynomials $P, V_1, \ldots, V_n$ over $\mathbb{C}$. These polynomials constitute a "geometric solution" (see [GLS01]) of the equation system $G_1 = 0, \ldots, G_n = 0$ because they represent the zero dimensional algebraic variety $V := \{G_1 = 0, \ldots, G_n = 0\}$ in the following "parameterized" form:

$$V := \{(V_1(t), \ldots, V_n(t)); t \in \mathbb{C}, P(t) = 0\}\,.$$

Let β be any robust, parameterized arithmetic circuit with the same number of inputs and outputs, say $X_1, \ldots, X_n$ and $G_1(U, X_1, \ldots, X_n), \ldots, G_n(U, X_1, \ldots, X_n)$, respectively. Suppose that the parameter domain of β, say $\mathcal{M}$, is irreducible and that $\mathrm{inv}(\beta)$ expresses that

for each parameter instance $u \in \mathcal{M}$ the polynomials $G_1(u, X_1, \ldots, X_n), \ldots, G_n(u, X_1, \ldots, X_n)$ form a reduced regular sequence in $\mathbb{C}[X_1, \ldots, X_n]$ with fixed (i.e. from $u \in \mathcal{M}$ independent) degree pattern. Suppose furthermore that the degrees of the individual polynomials $G_1(u, X_1, \ldots, X_n)$, $\ldots$, $G_n(u, X_1, \ldots, X_n)$ are also fixed. Then, on input β, the Kronecker algorithm runs iteratively n elementary routines of our computation model until the desired output becomes produced.

Another non–trivial example for an algorithm of our extended computation model, which involves only limited branchings, is the Gauss elimination procedure of [Edm67] (or [Bar68]) applied to matrices whose entries are polynomials represented by ordinary arithmetic circuits in combination with a identity–to–zero test for such polynomials. The variables of these polynomials are considered as basic parameters and any admissible input circuit has to satisfy a certain precondition formulated as the non–vanishing of suitable minors of the given polynomial matrix. Details and applications of this type of Gauss elimination for polynomial matrices can be found in [Hei83].

We say that a given algorithm $\mathcal{A}$ of our extended model *computes* (only) *parameters* if $\mathcal{A}$ satisfies the following condition:

for any admissible input β the final results of $\mathcal{A}(\beta)$ are all parameters.

Suppose that $\mathcal{A}$ is such an algorithm and β is the robust parametric arithmetic circuit with parameter domain $\mathcal{M}$ which we have considered before. Observe that $\mathcal{A}(\beta)$ contains the input variables $X_1, \ldots, X_n$ and that possibly new variables, which we call *auxiliary*, become introduced during the execution of the algorithm $\mathcal{A}$ on input β. Since the algorithm $\mathcal{A}$ computes only parameters, the input and auxiliary variables become finally eliminated by the application of recursive parameter routines and evaluations. We may therefore *collect garbage* reducing $\mathcal{A}(\beta)$ to a *final output circuit* $\mathcal{A}_{final}(\beta)$ which computes only parameters.

If we consider the algorithm $\mathcal{A}$ as a partial map which assigns to each admissible input circuit β its final output circuit $\mathcal{A}_{final}(\beta)$, we call $\mathcal{A}$ a *procedure*.

In this case, if ψ is a low level program defining $\mathcal{A}$, we call ψ a *low level procedure program*.

A particular feature of our extended computation model is the following:
there exist two increasing real valued functions $C_1 \geq 0$ and $C_2 \geq 0$ depending on dynamic integer vectors, such that with the previous notations and L_β, $L_{\mathcal{A}(\beta)}$ denoting the non–scalar sizes of the circuits β and $\mathcal{A}(\beta)$ the condition

$$L_{\mathcal{A}(\beta)} \leq C_1(\mathrm{inv}(\beta))L_\beta + C_2(\mathrm{inv}(\beta))$$

is satisfied.

This means that our extended computation model represents the *first level of a complexity hierarchy* for Scientific Computing which we are going to develop in future work.

In the sequel we shall need a particular variant of the notion of a procedure which enables us to capture the following situation.

Suppose we have to find a computational solution for a formally specified general algorithmic problem and that the formulation of the problem depends on certain parameter variables, say $U_1, \ldots, U_r$, input variables, say $X_1, \ldots, X_n$ and output variables, say

$Y_1, \ldots, Y_s$. Let such a problem formulation be given and suppose that its input is implemented by the robust parameterized arithmetic circuit β considered before, interpreting the parameter variables $U_1, \ldots, U_r$ as the basic parameters $\pi_1, \ldots, \pi_n$.

Then an algorithm $\mathcal{A}$ of our extended computation model which *solves* the given algorithmic problem should satisfy the architectural requirement we are going to describe now.

The algorithm $\mathcal{A}$ should be the composition of two subalgorithms $\mathcal{A}^{(1)}$ and $\mathcal{A}^{(2)}$ of our computation model which satisfy on input β the following conditions:

(i) The subalgorithm $\mathcal{A}^{(1)}$ computes only parameters, β is admissible for $\mathcal{A}^{(1)}$ and none of the indeterminates $Y_1, \ldots, Y_s$ is introduced in $\mathcal{A}^{(1)}(\beta)$ as auxiliary variable.

(ii) The circuit $\mathcal{A}^{(1)}_{final}(\beta)$ is an admissible input for the subalgorithm $\mathcal{A}^{(2)}$, the indeterminates $Y_1, \ldots, Y_s$ occur as auxiliary variables in $\mathcal{A}^{(2)}(\mathcal{A}^{(1)}_{final}(\beta))$ and the final results of $\mathcal{A}^{(2)}(\mathcal{A}^{(1)}_{final}(\beta))$ depend only on $\pi_1, \ldots, \pi_r$ and $Y_1, \ldots, Y_s$ (all other auxiliary variables become eliminated during the execution of the subalgorithm $\mathcal{A}^{(2)}$ on the input circuit $\mathcal{A}^{(1)}_{final}(\beta)$).

To the circuit $\mathcal{A}^{(2)}(\mathcal{A}^{(1)}_{final}(\beta))$ we may, as in the case when we compute only parameters, apply garbage collection. In this manner $\mathcal{A}^{(2)}(\mathcal{A}^{(1)}_{final}(\beta))$ becomes reduced to a final output circuit $\mathcal{A}_{final}(\beta)$ with parameter domain $\mathcal{M}$ which contains only the inputs $Y_1, \ldots, Y_s$.

Observe that the subalgorithm $\mathcal{A}^{(1)}$ is by Proposition 15 an output isoparametric procedure of our extended computation model (the same is also true for the subalgorithm $\mathcal{A}^{(2)}$, but this will not be relevant in the sequel).

We consider the algorithm $\mathcal{A}$, as well as the subalgorithms $\mathcal{A}^{(1)}$ and $\mathcal{A}^{(2)}$, as *procedures* of our extended computation model. In case that the *subprocedures* $\mathcal{A}^{(1)}$ and $\mathcal{A}^{(2)}$ are essentially division–free, we call also the procedure $\mathcal{A}$ *essentially division–free*. This will be of importance in Section 6.5.

The architectural requirement given by conditions (i) and (ii) may be interpreted as follows:

> the subprocedure $\mathcal{A}^{(1)}$ is a pipeline which transmits only parameters to the subprocedure $\mathcal{A}^{(2)}$. In particular, no (true) rational function is transmitted from $\mathcal{A}^{(1)}$ to $\mathcal{A}^{(2)}$.

Nevertheless, let us observe that on input β the procedure $\mathcal{A}$ establishes by means of the underlying low level program ψ an additional link between β and the subprocedure $\mathcal{A}^{(2)}$ applied to the input $\mathcal{A}^{(1)}(\beta)$. The elementary routines which constitute $\mathcal{A}^{(2)}$ on input $\mathcal{A}^{(1)}(\beta)$ become determined by index computations which realizes ψ on $\mathrm{inv}(\beta)$ and which depend on certain output isoparametric identity tests. In this sense the subprocedure $\mathcal{A}^{(1)}$ transmits not only parameters to the subprocedure but also a limited amount of digital information which stems from the input circuit β.

The decomposition of the procedure $\mathcal{A}$ into two subprocedures $\mathcal{A}^{(1)}$ and $\mathcal{A}^{(2)}$ satisfying conditions (i) and (ii) represents an architectural restriction which is justified when it makes sense to require that on input β the number of essential additions and multiplications contained in $\mathcal{A}_{\mathrm{final}}(\beta)$ is bounded by a function which depends only on $\mathrm{inv}(\beta)$. In

Section 6.5.1 we shall make a substantial use of this restriction and give such a justification in the particular case of elimination algorithms.

Here we shall only point out the following consequence of this restriction. Let assumptions and notations be as before, let G, ν and F be vectors composed by the final results of β, $\mathcal{A}^{(1)}(\beta)$ and $\mathcal{A}_{final}(\beta)$, respectively, and let θ and φ be the coefficient vectors of G and F. Then the images of θ and ν are constructible subsets $\mathcal{T}$ and $\mathcal{T}'$ of suitable affine spaces and there exist geometrically robust constructible maps σ and σ' defined on $\mathcal{T}$ and $\mathcal{T}'$ with $\nu = \sigma \circ \theta$ and $\varphi = \sigma' \circ \nu = \sigma' \circ \sigma$.

Suppose that there is given a (not necessarily convergent) sequence $(u_k)_{k\in\mathbb{N}}$ of parameter instances $u_k \in \mathcal{M}$ and that there exists a (possibly unknown) parameter instance $u \in \mathcal{M}$ such that the sequence $(\theta(u_k))_{k\in\mathbb{N}}$ converges to $\theta(u)$. In the terminology of [Ald84], [Lic90] and [BCS97] the sequence of ordinary arithmetic circuits $(\beta^{(u_k)})_{k\in\mathbb{N}}$ represents an *approximative algorithm* for the instantiation $G^{(u)}$ of G at u. Since the constructible maps σ and σ' are geometrically robust, we may conclude by Theorem 9 in Section 2.3.2 that they are continuous with respect to the Euclidean topologies of their respective ambient spaces. Under this assumption the sequences $(\nu(u_k))_{k\in\mathbb{N}}$ and $(\varphi(u_k))_{k\in\mathbb{N}}$ converge to $\nu(u)$ and $\varphi(u)$ and the sequence of ordinary arithmetic circuits $((\mathcal{A}_{final}(\beta))^{(u_k)})_{k\in\mathbb{N}}$ represents an approximative algorithm for $F^{(u)}$. This approximative algorithm has the following particular form:

> first the limit $\nu(u)$ of the sequence $(\nu(u_k))_{k\in\mathbb{N}}$ becomes "precomputed". Then the procedure $\mathcal{A}^{(2)}$ is applied to the input $\nu(u)$ and one obtains the ordinary arithmetic circuit $(\mathcal{A}_{final}(\beta))^{(u)}$ by garbage collection. The final results of $(\mathcal{A}_{final}(\beta))^{(u)}$ constitute the vector $F^{(u)}$.

Based on [HK04] and [GHKa] we shall develop in future work a high level specification language for algorithms and procedures of our computation model. The idea is to use a generalized variant of the extended constraint data base model introduced in [HK04] in order to specify algorithmic problems in symbolic scientific computing, especially in effective algebraic geometry (e.g. effective elimination problems; see Section 6.5). In this sense the procedure $\mathcal{A}$, which solves the algorithmic problem considered before, will turn out to be *query computation* composed by two subprocedures namely $\mathcal{A}^{(1)}$ and $\mathcal{A}^{(2)}$ having the following property:

> the procedures $\mathcal{A}^{(1)}$ and $\mathcal{A}^{(2)}$ compute each a subquery of the query which specifies the given algorithmic problem. All these queries are called *geometric* because the procedures $\mathcal{A}^{(1)}$, $\mathcal{A}^{(2)}$ and $\mathcal{A}$ are output isoparametric (see [GHKa]).

6.5 Applications of the extended computation model to complexity issues of effective elimination theory

In this section we shall always work with procedures of our extended, branching parsimonious computation model. We shall study two types of examples of elimination problems in effective algebraic geometry which certify, to a different extent, that *branching parsimonious* elimination procedures (see [GH01] and [CGH$^+$03] for this notion) based on our computation paradigm *cannot run in polynomial time.*

6.5.1 Flat families of zero–dimensional elimination problems

We start this section by introducing, in terms of an abstract data type, the notion of a flat family of zero–dimensional elimination problems (see also [GH01] and [CGH$^+$03]). Then we fix the classes of (concrete) objects, namely robust parameterized arithmetic circuits with suitable parameter domains, which represent ("implement") these problems by means of a suitable abstraction function.

Throughout this section we suppose that there are given indeterminates $U_1, \ldots, U_r$, $X_1, \ldots, X_n$ and Y over $\mathbb{C}$.

As concrete objects we shall consider robust parameterized arithmetic input and output circuits with parameter domain $\mathbb{A}^r$. The indeterminates $U_1, \ldots, U_r$ will play the role of the basic parameters. The input nodes of the input circuits will be labelled by $X_1, \ldots, X_n$, whereas the output circuits will have a single input node, labelled by Y. The output circuits will implement the "general solution" of the given flat family of zero–dimensional elimination problems.

Let us now define the meaning of the term "flat family of zero–dimensional elimination problems" (in the basic parameters $U_1, \ldots, U_r$ and the inputs $X_1, \ldots, X_n$). Let $U := (U_1, \ldots, U_r)$ and $X := (X_1, \ldots, X_n)$ and let $G_1, \ldots, G_n$ and H be polynomials belonging to the $\mathbb{C}$–algebra $\mathbb{C}[U, X] := \mathbb{C}[U_1, \ldots, U_r, X_1, \ldots, X_n]$. Suppose that the polynomials $G_1, \ldots, G_n$ form a regular sequence in $\mathbb{C}[U, X]$, thus defining an equidimensional subvariety $V := \{G_1 = 0, \ldots, G_n = 0\}$ of the $(n+r)$–dimensional affine space $\mathbb{A}^r \times \mathbb{A}^n$. The algebraic variety V has dimension r. Let δ be the (geometric) degree of V (observe that this degree does not take into account multiplicities or components at infinity). Suppose, furthermore, that the morphism of affine varieties $\pi : V \to \mathbb{A}^r$, induced by the canonical projection of $\mathbb{A}^r \times \mathbb{A}^n$ onto $\mathbb{A}^r$, is finite and generically unramified (this implies that π is flat and that the ideal generated by $G_1, \ldots, G_n$ in $\mathbb{C}[U, X]$ is radical). Let $\tilde{\pi} : V \to \mathbb{A}^{r+1}$ be the morphism defined by $\tilde{\pi}(v) := (\pi(v), H(v))$ for any point v of the variety V. The image of $\tilde{\pi}$ is a hypersurface of $\mathbb{A}^{r+1}$ whose minimal equation is a polynomial of $\mathbb{C}[U, Y] := \mathbb{C}[U_1, \ldots, U_r, Y]$ which we denote by F. Let us write $\deg F$ for the total degree of the polynomial F and $\deg_Y F$ for its partial degree in the variable Y. Observe that F is monic in Y and that $\deg F \leq \delta \deg H$ holds. Furthermore, for a Zariski dense set of points u of $\mathbb{A}^r$, we have that $\deg_Y F$ is the cardinality of the image of the restriction of H to the finite set $\pi^{-1}(u)$. The polynomial $F(U, H)$ vanishes on the variety V.

Let us consider an arbitrary point $u := (u_1, \ldots, u_r)$ of $\mathbb{A}^r$. For given polynomials $A \in \mathbb{C}[U, X]$ and $B \in \mathbb{C}[U, Y]$ we denote by $A^{(u)}$ and $B^{(u)}$ the polynomials $A(u_1, \ldots, u_r, X_1, \ldots, X_n)$ and $B(u_1, \ldots, u_r, Y)$ which belong to $\mathbb{C}[X] := \mathbb{C}[X_1, \ldots, X_n]$ and $\mathbb{C}[Y]$ respectively. Similarly we denote for an arbitrary polynomial $C \in \mathbb{C}[U]$ by $C^{(u)}$ the value $C(u_1, \ldots, u_r)$ which belongs to the field $\mathbb{C}$. The polynomials $G_1^{(u)}, \ldots, G_n^{(u)}$ define the zero–dimensional subvariety

$$V^{(u)} := \left\{ G_1^{(u)} = 0, \ldots, G_n^{(u)} = 0 \right\} \cong \pi^{-1}(u)$$

of the affine space $\mathbb{A}^n$. The degree (i.e. the cardinality) of $V^{(u)}$ is bounded by δ. Denote by $\tilde{\pi}^{(u)} : V^{(u)} \to \mathbb{A}^1$ the morphism induced by the polynomial $H^{(u)}$ on the variety $V^{(u)}$. Observe that the polynomial $F^{(u)}$ vanishes on the (finite) image of the morphism $\tilde{\pi}^{(u)}$. Observe also that the polynomial $F^{(u)}$ is not necessarily the minimal equation of the image of $\tilde{\pi}^{(u)}$.

We call the equation system $G_1 = 0, \ldots, G_n = 0$ and the polynomial H a *flat family of zero–dimensional elimination problems depending on the basic parameters* $U_1, \ldots, U_r$ *and the inputs* $X_1, \ldots, X_n$ and we call F the associated *elimination polynomial*, in Figure 30 below a suitable abstract function will carry out the corresponding assignment. A point $u \in \mathbb{A}^r$ is considered as a *parameter instance* which determines a *particular problem instance*, consisting of the equations $G_1^{(u)} = 0, \ldots, G_n^{(u)} = 0$ and the polynomial $H^{(u)}$. A power of the polynomial $F^{(u)}$ is called a *solution* of this particular problem instance.

The equation system $G_1 = 0, \ldots, G_n = 0$ together with the polynomial H is also called the *general instance* of the given flat family of elimination problems and any power of the elimination polynomial F is also called a *general solution* of this flat family.

We suppose now that the general instance of the given flat family of elimination problems is implemented by an essentially division–free, robust parameterized arithmetic circuit β with parameter domain $\mathbb{A}^r$ and inputs $X_1, \ldots, X_n$, whose final results are the polynomials $G_1, \ldots, G_n$ and H. The task is to find another essentially division–free, robust parameterized arithmetic circuit γ with parameter domain $\mathbb{A}^r$ having a single output node, labelled by Y, which computes for a suitable integer $q \in \mathbb{N}$ the power F^q of the polynomial F. We suppose furthermore that this goal becomes achieved by the application of an essentially division–free procedure $\mathcal{A}$ of our extended computation model to the input circuit β. Thus we have $\gamma = \mathcal{A}(\beta)$ and γ may be interpreted as an essentially division–free circuit over $\mathbb{C}[U]$ with a single input Y (observe that the parameters computed by the robust circuits β, $\mathcal{A}(\beta)$ and $\mathcal{A}_{final}(\beta)$ belong to the $\mathbb{C}$–algebra $\mathbb{C}[U]$). Using the geometric properties of flat families of zero–dimensional problems we deduce from [GHM$^+$98], [GHH$^+$97],[GHMP97], [GLS01] or alternatively from [CGH89], [DFGS91] that such essentially division–free procedures always exist and that they compute even the elimination polynomial F (the reader may notice that one needs for this argument the full power of our computation model which includes divisions by parameters).

We say that the essentially division–free procedure $\mathcal{A}$ *solves algorithmically* the general instance of the given flat family of zero–dimensional elimination problems. Figure 30 illustrates our software architecture for elimination problems.

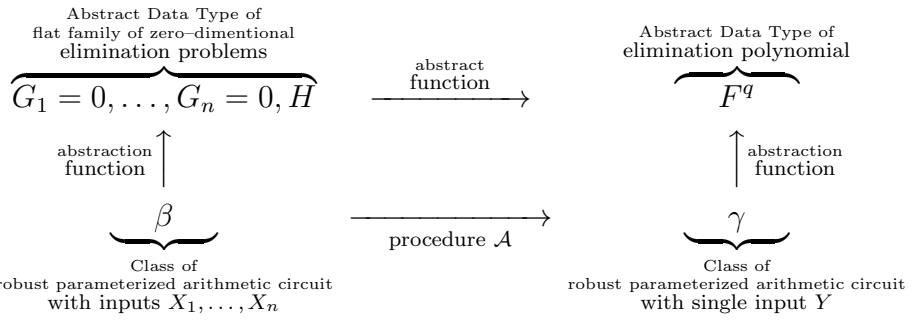


Figure 30: Software architecture for elimination problems.

From now on we suppose that there is given a procedure $\mathcal{A}$ of our extended computation model, decomposed in two essentially division–free subprocedures $\mathcal{A}^{(1)}$ and $\mathcal{A}^{(2)}$ as in Section 6.4.2, such that $\mathcal{A}$ solves algorithmically the general instance of any given flat

family of zero–dimensional elimination problems. Our circuit β is therefore an admissible input for $\mathcal{A}$ and hence for $\mathcal{A}^{(1)}$. The final results of $\mathcal{A}^{(1)}(\beta)$ constitute a geometrically robust constructible map ν defined on $\mathbb{A}^r$ which is an admissible input for the procedure $\mathcal{A}^{(2)}$ and $\gamma := \mathcal{A}_{final}(\beta)$ is an essentially division–free parameterized arithmetic circuit with parameter domain $\mathbb{A}^r$ and input Y.

Let $\mathcal{S}$ be the image of the geometrically robust constructible map ν. Then $\mathcal{S}$ is an irreducible constructible subset of a suitable affine space. Analyzing now the internal structure of the essentially division–free, robust parameterized arithmetic circuit $\mathcal{A}^{(2)}(\nu)$, one sees easily that there exists a geometrically robust constructible map ψ defined on $\mathcal{S}$ such that the entries of the geometrically robust composition map $\nu^* := \psi \circ \nu$ constitute the essential parameters of the circuit γ. Let m be the number of components of the map ν^*. Since ν and ν^* are composed by geometrically robust constructible functions defined on $\mathbb{A}^r$, we may interpret ν and ν^* as vectors of polynomials of $\mathbb{C}[U]$.

The circuit γ is essentially division–free. Hence there exists a vector ω of m–variate polynomials over $\mathbb{C}$ such that the polynomials of $\mathbb{C}[U]$, which constitute the entries of $\omega(\nu^*)$, become the coefficients of the elimination polynomial F with respect to the main indeterminate Y (see [KP96], Section 2.1). Observe that we may write $\omega(\nu^*) = \omega \circ \nu^*$ interpreting the entries of ν^* as polynomials of $\mathbb{C}[U]$.

We are now going to see what happens at a particular parameter instance $u \in \mathbb{A}^r$. Since β, $\mathcal{A}^{(1)}(\beta)$, $\mathcal{A}(\beta)$ and $\gamma = \mathcal{A}_{final}(\beta)$ are essentially division–free, robust parameterized arithmetic circuits with parameter domain $\mathbb{A}^r$, we may specialize the vector U of their basic parameters to the parameter instance $u \in \mathbb{A}^r$, obtaining thus ordinary division–free arithmetic circuits over $\mathbb{C}$ with the same inputs. We denote them by the superscript u, namely by $\beta^{(u)}$, $(\mathcal{A}^{(1)}(\beta))^{(u)}$, $(\mathcal{A}(\beta))^{(u)}$ and $\gamma^{(u)}$. One sees immediately that $G_1^{(u)}, \ldots, G_n^{(u)}$ and $H^{(u)}$ are the final results of $\beta^{(u)}$, that the entries of $\nu(u)$ are the final results of $(\mathcal{A}^{(1)}(\beta))^{(u)}$ and that $(F^{(u)})^q$ is the final result of $\mathcal{A}(\beta)^{(u)}$ and $\gamma^{(u)}$. Observe that the division–free circuit $\gamma^{(u)}$ uses only the entries of $\nu^*(u)$ and fixed rational numbers as scalars.

In the same spirit as before, we say that the procedure $\mathcal{A}$ solves algorithmically the particular instance, which is determined by u, of the given flat family of zero–dimensional elimination problems.

Let us here clarify how all this is linked to the rest of the terminology used in [CGH$^+$03]. In this terminology the polynomial map given by ω defines a "holomorphic encoding" of the set of solutions of all particular problem instances and $\nu^*(u)$ is a "code" of the particular solution $(F^{(u)})^q$. In the same context the robust constructible map ν^* is called an "elimination procedure" which is "robust" since the procedure $\mathcal{A}^{(1)}$ is output isoparametric and since ν^* is geometrically robust (see [CGH$^+$03], Definition 5 and Lemma 13 and Proposition 15 and Corollary 16 of this thesis and the comments at the end of Section 6.4.1).

In this sense we speak about *families* of zero–dimensional elimination problems and their instances and not simply about a single (particular or general) zero–dimensional elimination problem.

Let us now turn back to the discussion of the given essentially division–free procedure $\mathcal{A}$ which solves algorithmically the general instance of any flat family of zero–dimensional elimination problems.

We are now going to show the main result of this whole section, namely that the given

procedure $\mathcal{A}$ *cannot run in polynomial time.*

Theorem 17 *Let notations and assumptions be as before. For any natural number n there exists an essentially division–free, robust parameterized arithmetic circuit β_n with basic parameters T, $U_1, \ldots, U_n$ and inputs $X_1, \ldots, X_n$ which for $U := (U_1, \ldots, U_n)$ and $X := (X_1, \ldots, X_n)$ computes polynomials $G_1^{(n)}, \ldots, G_n^{(n)} \in \mathbb{C}[X]$ and $H^{(n)} \in \mathbb{C}[T, U, X]$ such that the following conditions are satisfied:*

(i) The equation system $G_1^{(n)} = 0, \ldots, G_n^{(n)} = 0$ and the polynomial $H^{(n)}$ constitute a flat family of zero–dimensional elimination problems, depending on the parameters T, $U_1, \ldots, U_n$ and the inputs $X_1, \ldots, X_n$, with associated elimination polynomial $F^{(n)} \in \mathbb{C}[T, U, Y]$.

(ii) β_n is an ordinary division–free arithmetic circuit of size $O(n)$ over $\mathbb{C}$ with inputs T, $U_1, \ldots, U_n$, $X_1, \ldots, X_n$.

(iii) $\gamma_n := \mathcal{A}_{final}(\beta_n)$ is an essentially division–free robust parameterized arithmetic circuit with basic parameters $T, U_1, \ldots, U_n$ and input Y such that γ_n computes for a suitable integer $q_n \in \mathbb{N}$ the polynomial $(F^{(n)})^{q_n}$. The circuit γ_n performs at least $\Omega(2^{\frac{n}{2}})$ essential multiplications and at least $\Omega(2^n)$ multiplications with parameters. Therefore γ_n has, as ordinary arithmetic circuit over $\mathbb{C}$ with inputs $T, U_1, \ldots, U_n, X_1, \ldots, X_n$, non–scalar size at least $\Omega(2^n)$.

Proof. During our argumentation we shall tacitly adapt to the new context the notations introduced before. We shall follow the main technical ideas behind the papers [GH01], [CGH$^+$03] and [GHMS11]. We fix now the natural number n and consider the polynomials

$$G_1 := G_1^{(n)} := X_1^2 - X_1, \ldots, G_n := G_n^{(n)} := X_1^2 - X_n$$

and

$$H := H^{(n)} := \sum_{1 \leq i \leq n} 2^{i-1} X_i + T \prod_{1 \leq i \leq n} (1 + (U_i - 1) X_i)$$

which belong to $\mathbb{C}[X]$ and to $\mathbb{C}[T, U, X]$, respectively.

Observe that $G_1, \ldots, G_n$ and H may be evaluated by a division–free ordinary arithmetic circuit $\beta := \beta_n$ over $\mathbb{C}$ which has non–scalar size $O(n)$ and inputs T, $U_1, \ldots, U_n$, $X_1, \ldots, X_n$. As parameterized arithmetic circuit β is therefore robust. Hence β satisfies condition (ii) of the theorem.

One sees easily that $G_1 = 0, \ldots, G_n = 0$ and H constitute a flat family of zero–dimensional elimination problems depending on the parameters T, $U_1, \ldots, U_n$ and the inputs $X_1, \ldots, X_n$.

Let us write H as a polynomial in the main indeterminates $X_1, \ldots, X_n$ with coefficients $\theta_{\kappa_1, \ldots, \kappa_n} \in \mathbb{C}[T, U]$, $\kappa_1, \ldots, \kappa_n \in \{0, 1\}$, namely

$$H = \sum_{\kappa_1, \ldots, \kappa_n \in \{0,1\}} \theta_{\kappa_1, \ldots, \kappa_n} X_1^{\kappa_1}, \ldots, X_n^{\kappa_n}.$$

Observe that for $\kappa_1, \ldots, \kappa_n \in \{0, 1\}$ the polynomial $\theta_{\kappa_1, \ldots, \kappa_n}(0, U) \in \mathbb{C}[U]$ is of degree at most zero, i.e. a constant complex number, independent of $U_1, \ldots, U_n$.

Let $\theta := (\theta_{\kappa_1,\ldots,\kappa_n})_{\kappa_1,\ldots,\kappa_n\in\{0,1\}}$ and observe that the vector $\theta(0,U)$ is a fixed point of the affine space $\mathbb{A}^{2^n}$. We denote by $\mathfrak{M}$ the vanishing ideal of the $\mathbb{C}$–algebra $\mathbb{C}[\theta]$ at this point.

Consider now the polynomial

$$F := F^{(n)} := \prod_{0\leq j\leq 2^n-1} (Y - (j + T \prod_{1\leq i\leq n} U_i^{[j]_i}))$$

of $\mathbb{C}[T,U,Y]$, where $[j]_i$ denotes the i–th digit of the binary representation of the integer j, $0 \leq j \leq 2^{n-1}-1$, $1 \leq i \leq n$. Let $q := q_n$.

One sees easily that F is the elimination polynomial associated with the given flat family of zero–dimensional elimination problems $G_1 = 0, \ldots, G_n = 0$ and H.

Let us write F^q as a polynomial in the main indeterminate Y with coefficients $\varphi_\kappa \in \mathbb{C}[T,U]$, $1 \leq \kappa \leq 2^n q$, namely

$$F^q = Y^{2^n q} + \varphi_1 Y^{2^n q-1} + \cdots + \varphi_{2^n q}.$$

Observe that for $1 \leq \kappa \leq 2^n q$ the polynomial $\varphi_\kappa(0,U) \in \mathbb{C}[U]$ is of degree at most zero. Let $\lambda_\kappa := \varphi_\kappa(0,U)$, $\lambda := (\lambda_\kappa)_{1\leq\kappa\leq 2^n q}$ and $\varphi := (\varphi_\kappa)_{1\leq\kappa\leq 2^n q}$. Observe that λ is also a fixed point of the affine space $\mathbb{A}^{2^n q}$.

Recall that β is an admissible input for the procedure $\mathcal{A}$ and hence for $\mathcal{A}^{(1)}$, that the final results of $\mathcal{A}^{(1)}(\beta)$ constitute the entries of the robust constructible map ν defined on $\mathbb{A}^{n+1}$, that ν is an admissible input for the procedure $\mathcal{A}^{(2)}$ and that $\gamma = \mathcal{A}_{final}(\beta)$ is an essentially division–free, parameterized arithmetic circuit with parameter domain $\mathbb{A}^{n+1}$ and input Y.

Furthermore recall that there exists a geometrically robust constructible map ψ defined on the image $\mathcal{S}$ of ν such that the entries of $\nu^* = \psi \circ \nu$ constitute the essential parameters of the circuit γ, that the entries of ν and ν^* may be interpreted as polynomials of $\mathbb{C}[T,U]$ and that for m being the number of components of the map ν^*, there exists a vector ω of m–variate polynomials over $\mathbb{C}$ such that the polynomials of $\mathbb{C}[T,U]$ which constitute the entries of $\omega(\nu^*) = \omega \circ \nu^*$ become the coefficients of the polynomial F^q with respect to the main indeterminate Y. Let $\mathcal{T}$ be the image of the coefficient vector θ of H, and interpret θ as a geometrically robust constructible map defined on $\mathbb{A}^{n+1}$. Observe that $\mathcal{T}$ is a constructible subset of $\mathbb{A}^{2^n}$. Since H is the unique, basic parameter dependent final result of the circuit β, we deduce from Proposition 15 that there exists a geometrically robust constructible map σ defined on $\mathcal{T}$ satisfying the condition $\nu = \sigma \circ \theta$. This implies $\nu^* = \psi \circ \sigma \circ \theta$ and the entries of ν^* are polynomials of $\mathbb{C}[T,U]$ which are integral over the local $\mathbb{C}$–subalgebra $\mathbb{C}[\theta]_{\mathfrak{M}}$ of $\mathbb{C}(T,U)$.

Let $\mu \in \mathbb{C}[T,U]$ be such an entry. Then there exists an integer s and polynomials $a_0, a_1, \ldots, a_s \in \mathbb{C}[\theta]$ with $a_0 \notin \mathfrak{M}$ such that the algebraic dependence relation

$$a_0\mu^s + a_1\mu^{s-1} + \cdots + a_s = 0 \tag{14}$$

is satisfied in $\mathbb{C}[T,U]$. From (14) we deduce the algebraic dependence relation

$$a_0(0,U)\mu(0,U)^s + a_1(0,U)\mu(0,U)^{s-1} + \cdots + a_s(0,U) = 0 \tag{15}$$

in $\mathbb{C}[U]$.

Since the polynomials $a_0, a_1, \ldots, a_s$ belong to $\mathbb{C}[\theta]$ and $\theta(0,U)$ is a fixed point of $\mathbb{A}^{2^n}$ we conclude that $\alpha_0 := a_0(0,U), \alpha_1 := a_1(0,U), \ldots, \alpha_s := a_s(0,U)$ are complex numbers. Moreover, $a_0 \notin \mathfrak{M}$ implies $\alpha_0 \neq 0$.

Thus (15) may be rewritten into the algebraic dependence relation

$$\alpha_0 \mu(0,U)^s + \alpha_1 \mu(0,U)^{s-1} + \cdots + \alpha_s = 0 \tag{16}$$

in $\mathbb{C}[U]$ with $\alpha_0 \neq 0$.

This implies that the polynomial $\mu(0,U)$ of $\mathbb{C}[U]$ is of degree at most zero.

Therefore $w := \nu^*(0,U)$ is a fixed point of the affine space $\mathbb{A}^m$. Since γ computes the polynomial F^q and F^q has the form $F^q = Y^{2^n q} + \varphi_1 Y^{2^n q - 1} + \cdots + \varphi_{2^n q}$ with $\varphi_\kappa \in \mathbb{C}[T,U]$, $1 \leq \kappa \leq 2^n q$, we see that $\varphi = (\varphi_\kappa)_{1 \leq \kappa \leq 2^n q}$ may be decomposed as follows:

$$\varphi = \omega(\nu^*) = \omega \circ \nu^*.$$

Recall that $\lambda = (\lambda_\kappa)_{1 \leq \kappa \leq 2^n q}$ with $\lambda_\kappa := \varphi_\kappa(0,U)$, $1 \leq \kappa \leq 2^n q$, is a fixed point of the affine space $\mathbb{A}^{2^n}$.

For $1 \leq \kappa \leq 2^n q$ we may write the polynomial $\varphi_\kappa \in \mathbb{C}[T,U]$ as follows:

$$\varphi_\kappa = \lambda_\kappa + \Delta_\kappa T + \text{ terms of higher degree in } T \tag{17}$$

with $\Delta_\kappa \in \mathbb{C}[U]$. From [CGH$^+$03], Lemma 6 we deduce that the elimination polynomial F has the form $F = Y^{2^n} + B_1 Y^{2^n - 1} + \cdots + B_{2^n}$ where for $1 \leq l \leq 2^n$ B_l is an element of $\mathbb{C}[T,U]$ with

$$B_l = (-1)^l \sum_{l \leq j_1 < \cdots < j_l < 2^n} j_1 \ldots j_l + T L_l + \text{ terms of higher degree in } T$$

and where $L_1, \ldots, L_{2^n} \in \mathbb{C}[U]$ are $\mathbb{C}$–linearly independent.

Choose now different complex numbers $\eta_0, \ldots, \eta_{2^n q}$ from $\mathbb{C} - \{j \in \mathbb{Z}; 0 \leq j < 2^n\}$ and observe that for $0 \leq \kappa' \leq 2^n q$ the identities

$$\frac{\partial F^q}{\partial T}(0,U,\eta_{\kappa'}) = q F^{q-1}(0,U,\eta_{\kappa'}) \frac{\partial F}{\partial T}(0,U,\eta_{\kappa'}) = q \prod_{0 \leq j < 2^n} (\eta_{\kappa'} - j) \sum_{1 \leq l \leq 2^n} L_l \eta_{\kappa'}^l$$

and

$$\frac{\partial F^q}{\partial T}(0,U,\eta_{\kappa'}) = \sum_{1 \leq \kappa \leq 2^n q} \Delta_\kappa \eta_{\kappa'}^\kappa$$

hold.

From the non–singularity of the Vandermonde matrix $(\eta_{\kappa'}^\kappa)_{0 \leq \kappa, \kappa' \leq 2^n q}$ we deduce now that 2^n many of the polynomials $\Delta_1, \ldots, \Delta_{2^n q}$ of $\mathbb{C}[U]$ are $\mathbb{C}$–linearly independent.

Consider now an arbitrary point $u \in \mathbb{A}^n$ and let $\epsilon_u : \mathbb{A}^1 \to \mathbb{A}^m$ and $\delta_u : \mathbb{A}^1 \to \mathbb{A}^{2^n q}$ be the polynomial maps defined for $t \in \mathbb{A}^1$ by $\epsilon_u(t) := \nu^*(t,u)$ and $\delta_u(t) := \varphi(t,u)$. Then we have $\epsilon_u(0) = \nu^*(0,u) = w$ with $w \in \mathbb{A}^m$ and $\delta_u(0) = \varphi(0,u) = \lambda$ with $\lambda \in \mathbb{A}^{2^n q}$, independently of u. Moreover, from $\varphi = \omega \circ \nu^*$ we deduce $\delta_u = \omega \circ \epsilon_u$.

Thus (17) implies

$$(\Delta_1(u), \ldots, \Delta_{2^n q}(u)) = \frac{\partial \varphi}{\partial t}(0,u) = \delta_u'(0) = (D\omega)_w(\epsilon_u'(0)), \tag{18}$$

where $(D\omega)_w$ denotes the (first) derivative of the m–variate polynomial map ω at the point $w \in \mathbb{A}^m$ and $\delta'_u(0)$ and $\epsilon'_u(0)$ are the derivatives of the parameterized curves δ_u and ϵ_u at the point $0 \in \mathbb{A}^1$. We rewrite now (18) in matricial form, replacing $(D\omega)_w$ by the corresponding transposed Jacobi matrix $M \in \mathbb{A}^{m\times 2^n q}$ and $\delta'_u(0)$ and $\epsilon'_u(0)$ by the corresponding points of $\mathbb{A}^{2^n q}$ and $\mathbb{A}^m$, respectively.

Then (18) takes the form

$$(\Delta_1(u), \ldots, \Delta_{2^n q}(u)) = \epsilon'_u(0)M, \tag{19}$$

where the complex $(m \times 2^n q)$–matrix M is independent of u.

Since 2^n many of the polynomials $\Delta_1, \ldots, \Delta_{2^n} \in \mathbb{C}[U]$ are $\mathbb{C}$–linearly independent, we may chose points $u_1, \ldots, u_{2^n} \in \mathbb{A}^n$ such that the complex $(2^n \times 2^n q)$–matrix

$$N := (\Delta_\kappa(u_l))_{\substack{1 \le l \le 2^n \\ 1 \le \kappa \le 2^n q}}$$

has rank 2^n.

Let K be the complex $(2^n \times m)$–matrix whose rows are $\epsilon'_{u_1}(0), \ldots, \epsilon'_{u_{2^n}}(0)$.

Then (19) implies the matrix identity

$$N = K \cdot M.$$

Since N has rank 2^n, the rank of the complex $(m \times 2^n)$–matrix M is at least 2^n. This implies

$$m \ge 2^n. \tag{20}$$

Therefore the circuit γ contains $m \ge 2^n$ essential parameters.

Let L be the number of essential multiplications executed by the parameterized arithmetic circuit γ and let L' be the total number of multiplications of γ, excepting those by scalars from $\mathbb{C}$. Then, after a well known standard rearrangement [PS73] of γ, we may suppose without loss of generality, that there exists a constant $c > 0$ (independent of the input circuit γ and the procedure $\mathcal{A}$) such that $L \ge cm^{\frac{1}{2}}$ and $L' \ge cm$ holds.

From the estimation (20) we deduce now that the circuit γ performs at least $\Omega(2^{\frac{n}{2}})$ essential multiplications and at least $\Omega(2^n)$ multiplications, including also multiplications with parameters. This finishes the proof of the theorem. ■

Observation Let assumptions and notations be as before. In the proof of Theorem 17 we made a substantial use of the output isoparametricity of the procedure $\mathcal{A}^{(1)}$ when we applied Proposition 15 in order to guarantee the existence of a geometrically robust constructible map σ defined on $\mathcal{T}$ which satisfies the condition $\nu = \sigma \circ \theta$. The conclusion was that the entries of $\nu^* = \psi \circ \nu$ are polynomials of $\mathbb{C}[T, U]$ which are integral over $\mathbb{C}[\theta]_{\mathfrak{M}}$. This implied finally that $\nu^*(0, U)$ is a fixed point of the affine space $\mathbb{A}^m$. Taking into account the results of [CGH$^+$03], Sections 3.2 and 5.1 it suffices to require that the procedure $\mathcal{A}^{(1)}$ is *output coalescent* in order to arrive at the same conclusion. This means that Theorem 17 remains valid if we require only that the procedure $\mathcal{A}^{(1)}$ is output coalescent.

In the proof of Theorem 17 we have exhibited an infinite sequence of flat families of zero–dimensional elimination problems represented by robust parameterized arithmetic

circuits of small size, such that any implementation of their associated elimination polynomials, obtained by a procedure of our extended computation model which solves the given elimination task for any instance, requires circuits of exponential size.

The statement of Theorem 17 may also be interpreted in terms of a mathematically certified trade–off of quality attributes. Suppose for the moment that we would have built our model for branching parsimonious computation in the same way as in Section 6.4, however omitting the requirement (B) for recursive routines. Recall that this requirement implies the output isoparametricity of any algorithm of our extended computation model and recall from Section 6.4.1 that well behavedness under reduction is a quality attribute which implies output isoparametricity and therefore also the conclusion of Theorem 17.

A complexity class like "exponential time in worst case" represents also a quality attribute. Thus we see that the quality attribute "well behavedness under reduction" implies the quality attribute "exponential time in worst case" for any essentially division–free procedure of our extended computation model which solves algorithmically the general instance of any given flat family of zero–dimensional problems.

The proof of Theorem 17 depends substantially on the decomposition of the elimination procedure $\mathcal{A}$ into two subprocedures $\mathcal{A}^{(1)}$ and $\mathcal{A}^{(2)}$ satisfying conditions (i) and (ii) of Section 6.4.2. We are now going to justify this architectural restriction on the procedure $\mathcal{A}$ for the particular case of elimination algorithms.

As at the beginning of this section, let $U := (U_1, \ldots, U_r)$, $X := (X_1, \ldots, X_n)$, $G_1, \ldots, G_n$, $H \in \mathbb{C}[U, X]$ and $F \in \mathbb{C}[U, Y]$ such that $G_1 = 0, \ldots, G_n = 0$ and H constitute a flat family of zero–dimensional elimination problems and F its associated elimination polynomial. Suppose that $G_1, \ldots, G_n$ and H are implemented by an essentially division–free, robust parameterized arithmetic circuit β with parameter domain $\mathbb{A}^r$ and inputs $X_1, \ldots, X_n$.

All *known* algorithms which solve the general instance of any flat family of zero–dimensional elimination problems may be interpreted as belonging to our restricted set of procedures. They compute directly the elimination polynomial F (and not an arbitrary power of it). Thus let $\mathcal{A}$ be such a known algorithm and let $\mathcal{A}^{(1)}$ and $\mathcal{A}^{(2)}$ be the subalgorithms which compose $\mathcal{A}$ in the same way as before. Then $\mathcal{A}^{(1)}$ computes the coefficients of F, where F is considered as a univariate polynomial over $\mathbb{C}[U]$ in the indeterminate Y. The subalgorithm $\mathcal{A}^{(2)}$ may be interpreted as the Horner scheme which evaluates F from its precomputed coefficients and Y.

Observe that F, and hence $\deg_Y F$, depends only on the polynomials $G_1, \ldots, G_n$ and H, but not on the particular circuit β. Therefore $\deg_Y F$ is determined by $\mathrm{inv}(\beta)$.

For any parameter instance $u \in \mathbb{A}^r$ we may think $(\mathcal{A}^{(1)}(\beta))^{(u)}$ as a constraint database (in the sense of [HK04] and [GHKa]) which allows to evaluate the univariate polynomial $F^{(u)} \in \mathbb{C}[Y]$ as often as we want for arbitrary inputs $y \in \mathbb{A}^1$, using each time a number of arithmetic operations in $\mathbb{C}$, namely $\deg_Y F$, which does not depend on the non–scalar size of β.

Let now $\mathcal{A}$ be an *arbitrary*, essentially division–free algorithm of our extended computation model which solves the general instance of any flat family of zero–dimensional elimination problems and let β be an input circuit for $\mathcal{A}$ which represents a particular family of such problems. Let F be the associated elimination polynomial.

Then the complexity of the algorithm $\mathcal{A}$ becomes only competitive with known elimination algorithms if we require that the number of *essential* additions and multiplications

of $\mathcal{A}_{\text{final}}(\beta)$ must be bounded by $2\deg_Y F$. This leads us to the requirement that $\mathcal{A}$ must be decomposable in two subalgorithms $\mathcal{A}^{(1)}$ and $\mathcal{A}^{(2)}$ as above.

Therefore any elimination algorithm of our extended computation model which is claimed to improve upon known algorithms for *all* admissible input circuits β, must have this architectural structure. In particular, such an algorithm cannot call the input circuit β when the output variable Y became already involved. This justifies the architectural restriction we made in the statement and proof of Theorem 17.

Recall from Section 6.4.2 that there exist increasing real valued functions $C_1 \geq 0$ and $C_2 \geq 0$ depending on dynamic integer vectors, such that for L_β and $L_{\mathcal{A}(\beta)}$ being the non–scalar sizes of the circuits β and $\mathcal{A}(\beta)$ the inequality

$$L_{\mathcal{A}(\beta)} \leq C_1(\text{inv}(\beta))L_\beta + C_2(\text{inv}(\beta))$$

holds. From Theorem 17 and its proof we deduce now the lower *worst case* bound

$$\max\{C_1(\text{inv}(\beta)), C_2(\text{inv}(\beta))\} = \Omega(\frac{\delta}{n}),$$

where n is the number of inputs of β and δ is the geometric degree of the subvariety of $\mathbb{A}^r \times \mathbb{A}^n$ defined by $G_1, \ldots, G_n$ (observe that 2^n is the geometric degree of $\{X_1^2 - X_1 = 0, \ldots, X_n^2 - X_n = 0\}$). This means that the complexity of the Kronecker algorithm [GLS01] is asymptotically optimal in our extended computation model. The particular case of a flat family of zero–dimensional elimination problems in just one input variable was treated in [HMPW98], Sections 2.1 and 2.2 with reference to ordinary arithmetic circuits over $\mathbb{C}$ and to the usual non–uniform complexity model of Algebraic Complexity Theory. Reformulated in our terminology the outcome was the worst case estimate $C_1(\text{inv}(\beta)) = \Omega(\delta)$ for the case $n := 1$.

Theorem 17 and its proof depends essentially on the assumption that the elimination procedure $\mathcal{A}$ produces on the input circuit $\beta_n, n \in \mathbb{N}$ a *branching–free* computation $\mathcal{A}(\beta_n)$. This contrasts with the fact that it is possible to construct for any $n \in \mathbb{N}$ a computation tree that decides in $n^{O(1)}$ steps whether for a given real input $(t, u_1, \ldots, u_n, y) \in [0,1]^{n+2}$ the polynomial equation system $G_1^{(n)}(X) = 0, \ldots, G_n^{(n)}(X) = 0$, $H^{(n)}(t, u_1, \ldots, u_n, X) = y$ has a solution. However, this computation tree uses a number of branchings which is exponential in n and has to be stored in advance (see [GHKb]).

6.5.2 The elimination of a block of existential quantifiers

Let notations be the same as in the proof of Theorem 17 in Section 6.5.1. Let $n \in \mathbb{N}$, $S_1, \ldots, S_n$ new indeterminates, $S := (S_1, \ldots, S_n)$, $\hat{G}_1^{(n)} := X_1^2 - X_1 - S_1, \ldots, \hat{G}_n^{(n)} := X_n^2 - X_n - S_n$ and again $H^{(n)} := \sum_{1 \leq i \leq n} 2^{i-1} X_i + T \prod_{1 \leq i \leq n} (1 + (U_i - 1)X_i)$.

Observe that the polynomials $\hat{G}_1^{(n)}, \ldots, \hat{G}_n^{(n)}$ form a reduced regular sequence in $\mathbb{C}[S, T, U, X]$ and that they define a subvariety $\hat{V}_n$ of the affine space $\mathbb{A}^n \times \mathbb{A}^1 \times \mathbb{A}^n \times \mathbb{A}^n$ which is isomorphic to $\mathbb{A}^n \times \mathbb{A}^1 \times \mathbb{A}^n$ and hence irreducible and of dimension $2n + 1$.

Moreover the morphism $\hat{\pi}_n : \hat{V}_n \to \mathbb{A}^n \times \mathbb{A}^1 \times \mathbb{A}^n \times \mathbb{A}^1$ which associates to any $(s, t, u, x) \in \hat{V}_n$ the point $(s, t, u, H^{(n)}(t, u, x)) \in \mathbb{A}^n \times \mathbb{A}^1 \times \mathbb{A}^n \times \mathbb{A}^1$ is finite and its image $\hat{\pi}_n(\hat{V}_n)$ is a hypersurface of $\mathbb{A}^n \times \mathbb{A}^1 \times \mathbb{A}^n \times \mathbb{A}^1$ with irreducible minimal equation $\hat{F}^{(n)} \in \mathbb{C}[S, T, U, Y]$.

Therefore $\hat{G}_1^{(n)} = 0, \ldots, \hat{G}_n^{(n)} = 0$ and $H^{(n)}$ represent a flat family of zero–dimensional elimination problems whose associated elimination polynomial is just $\hat{F}^{(n)}$.

Observe $\deg \hat{F}^{(n)} = \deg_Y \hat{F}^{(n)} = 2^n$ and that for $0 \in \mathbb{A}^n$ the identity $\hat{F}^{(n)}(0, T, U, Y) = F^{(n)}(T, U, Y)$ holds, where $F^{(n)}$ is the elimination polynomial associated with the flat family of zero dimensional elimination problems given by $X_1^2 - X_1 = 0, \ldots, X_n^2 - X_n = 0$ and $H^{(n)}$. Since $\hat{F}^{(n)}$ is irreducible, any equation of $\mathbb{C}[S, T, U, Y]$ which defines $\hat{\pi}_n(\hat{V}_n)$ in $\mathbb{A}^n \times \mathbb{A}^1 \times \mathbb{A}^n \times \mathbb{A}^1$ is without loss of generality a power of $\hat{F}^{(n)}$.

We consider $S_1, \ldots, S_n, T, U_1, \ldots, U_n$ as basic parameters, $X_1, \ldots, X_n$ as input and Y as output variables.

Let $\mathcal{A}'$ be an essentially division–free procedure of our extended computation model satisfying the following condition:

> $\mathcal{A}'$ accepts as input any robust parameterized arithmetic circuit β which represents the general instance of a flat family of zero–dimensional elimination problems with associated elimination polynomial F and $\mathcal{A}'_{\text{final}}(\beta)$ has a single input Y and a single final result which defines in $\mathbb{A}^n \times \mathbb{A}^1 \times \mathbb{A}^n \times \mathbb{A}^1$ the hypersurface given by F.

With this notions and notations we have the following result.

Proposition 18 *There exist an ordinary division–free arithmetic circuit $\hat{\beta}_n$ of size $O(n)$ over $\mathbb{C}$ with inputs $S_1, \ldots, S_n$, T, $U_1, \ldots, U_n$, $X_1, \ldots, X_n$ and final results $\hat{G}_1^{(n)}, \ldots, \hat{G}_n^{(n)}, H^{(n)}$. The essentially division–free, robust parameterized arithmetic circuit $\hat{\gamma}_n := \mathcal{A}'_{final}(\hat{\beta}_n)$ depends on the basic parameters $S_1, \ldots, S_n$, T, $U_1, \ldots, U_n$ and the input Y and its single final result is a power of $\hat{F}^{(n)}$. The circuit $\hat{\gamma}_n$ performs at least $\Omega(2^{\frac{n}{2}})$ essential multiplications and at least $\Omega(2^n)$ multiplications with parameters. As ordinary arithmetic circuit over $\mathbb{C}$ with inputs $S_1, \ldots, S_n$, T, $U_1, \ldots, U_n$ and Y, the circuit $\hat{\gamma}_n$ has non–scalar size at least $\Omega(2^n)$.*

Proof. The existence of an ordinary division–free arithmetic circuit as in the statement of Proposition 18 is evident. The rest follows immediately from the proof of Theorem 17 in Section 6.5.1 by restricting the parameter domain $\mathbb{A}^n \times \mathbb{A}^1 \times \mathbb{A}^n$ of $\hat{\beta}_n$ to $\mathbb{A}^1 \times \mathbb{A}^n$, i.e. by specializing S to $0 \in \mathbb{A}^n$. ■

Suppose now that there is given another essentially division–free procedure $\mathcal{A}''$ of our extended computation model satisfying the following condition:

> $\mathcal{A}''$ accepts as input any robust arithmetic circuit β which represents the general instance of a flat family of zero–dimensional elimination problems with associated elimination polynomial F and there exists a boolean circuit b in as many variables as the number of final results of $\mathcal{A}''_{\text{final}}(\beta)$ such that the algebraic variety defined by F coincides with the constructible set which can be described by plugging into b the final results of $\mathcal{A}''_{\text{final}}(\beta)$ as polynomial equations.

Observe that this represents the most general architecture we can employ to adapt in the spirit of Section 6.4.2 our extended computation model for *functions* to *parametric decision problems*.

Let $s \in \mathbb{N}$ and $A_1, \ldots, A_s$ new indeterminates with $A := (A_1, \ldots, A_s)$. We suppose that there is given an essentially division–free procedure $\mathcal{B}$ of our extended computation model which accepts as input any essentially division–free, robust parameterized arithmetic circuit γ which depends on the basic parameters $A_1, \ldots, A_s$ and the input variable Y such that $\mathcal{B}_{\text{final}}(\gamma)$ depends on the same basic parameters and the same input variable and has a single final result, namely the greatest common divisor in $\mathbb{C}[A, Y]$ of the final results of γ.

In this sense we call the procedure $\mathcal{B}$ a *GCD algorithm*.

Let $C_1 \geq 0$, $C_2 \geq 0$ and $D_1 \geq 0$, $D_2 \geq 0$ be four increasing real valued functions which depend on dynamic integer vectors and which satisfy, with the notations of Section 6.4.2, the estimates

$$L_{\mathcal{A}(\beta)} \leq C_1(\text{inv}(\beta))L_\beta + C_2(\text{inv}(\beta))$$

and

$$L_{\mathcal{B}(\gamma)} \leq D_1(\text{inv}(\gamma))L_\gamma + D_2(\text{inv}(\gamma)).$$

We consider again the ordinary division–free arithmetic circuit $\hat{\beta}_n$ of Proposition 15 which represents the polynomials $\hat{G}_1^{(n)}, \ldots, \hat{G}_n^{(n)}$ and $H^{(n)}$.

With these notions and notations we may now formulate the following statement.

Theorem 19 *Let assumptions and notations be as before. Then we have*

$$max\{C_i(inv(\hat{\beta}_n)), D_i(inv(\mathcal{A}''_{final}(\hat{\beta}_n))); i = 1, 2\} = \Omega(\frac{2^{\frac{n}{2}}}{n})$$

Proof. If we plug into the boolean circuit b the final results of $\mathcal{A}''_{\text{final}}(\hat{\beta}_n)$ as polynomial equations, we obtain by assumption a description of the hypersurface $\hat{\pi}(\hat{V}_n)$ of the affine space $\mathbb{A}^n \times \mathbb{A}^1 \times \mathbb{A}^n \times \mathbb{A}^1$. This implies that between the final results of $\mathcal{A}''_{\text{final}}(\hat{\beta}_n)$ there exists a selection, say the polynomials $P_1, \ldots, P_m$ and $R_1, \ldots, R_t$ of $\mathbb{C}[S, T, U, Y]$ such that the formula

$$P_1 = 0 \wedge \cdots \wedge P_m = 0 \wedge R_1 \neq 0 \wedge \cdots \wedge R_t \neq 0$$

defines a nonempty Zariski open (and dense) subset of the irreducible surface $\hat{\pi}(\hat{V}_n)$ of $\mathbb{A}^n \times \mathbb{A}^1 \times \mathbb{A}^n \times \mathbb{A}^1$.

Let $R := R_1 \ldots R_m$ and observe that the greatest common divisor of $P_1, \ldots, P_m$ has the form $(\hat{F}^{(n)})^q \cdot Q$, where q belongs to $\mathbb{N}$ and Q is a greatest common divisor of $P_1, \ldots, P_m, R$. Therefore we may compute $(F^{(n)})^q$ in the following way: erasing suitable nodes from the circuit $\mathcal{A}''_{\text{final}}(\hat{\beta}_n)$ and adding $t-1$ multiplication nodes we obtain two robust parameterized arithmetic circuits $\gamma_1^{(n)}$ and $\gamma_2^{(n)}$ with basic parameters $S_1, \ldots, S_n, T, U_1, \ldots, U_n$ and input Y whose final results are $P_1, \ldots, P_m$ and $P_1, \ldots, P_m, R$ respectively.

The final results of $\mathcal{B}_{\text{final}}(\gamma_1^{(n)})$ and $\mathcal{B}_{\text{final}}(\gamma_2^{(n)})$ are $(\hat{F}^{(n)})^q \cdot Q$ and Q. Connecting $\mathcal{B}_{\text{final}}(\gamma_1^{(n)})$ and $\mathcal{B}_{\text{final}}(\gamma_2^{(n)})$ by a division node and labelling this node as output we obtain finally a robust parameterized arithmetic circuit with basic parameters $S_1, \ldots, S_n$, T, $U_1, \ldots, U_n$ and input Y whose single final result is $(\hat{F}^n)^q$.

Joining the circuits $\mathcal{A}''(\hat{\beta}_n)$, $\mathcal{B}_{\text{final}}(\gamma_1^{(n)})$, $\mathcal{B}_{\text{final}}(\gamma_2^{(n)})$ and the final division node we obtain an ordinary arithmetic circuit of non–scalar size at most

$$1 + 2L_{\mathcal{B}(\mathcal{A}''_{\text{final}}(\hat{\beta}_n))} \leq$$

$$1 + 2(D_1(\mathrm{inv}(\mathcal{A}''_{\mathrm{final}}(\hat{\beta}_n)))L_{\mathcal{A}''_{\mathrm{final}}(\hat{\beta}_n)}) + D_2(\mathrm{inv}(\mathcal{A}''_{\mathrm{final}}(\hat{\beta}_n))) \leq$$

$$1 + 2C_1(\mathrm{inv}(\hat{\beta}_n))D_1(\mathrm{inv}(\mathcal{A}''_{\mathrm{final}}(\hat{\beta}_n)))L_{\hat{\beta}_n} +$$

$$2C_2(\mathrm{inv}(\beta_n))D_1(\mathrm{inv}(\mathcal{A}''_{\mathrm{final}}(\hat{\beta}_n))) + D_2(\mathrm{inv}(\mathcal{A}_{\mathrm{final}}(\hat{\beta}_n))).$$

On the other hand we deduce from Proposition 15

$$L_{\hat{\beta}_n} = O(n) \text{ and } 1 + 2L_{\mathcal{B}(\mathcal{A}''_{\mathrm{final}}(\hat{\beta}_n))} = \Omega(2^n).$$

This implies the estimate of Theorem 19. ■

In a simple minded understanding, Theorem 19 says that in our extended computation model either the elimination of a single existential quantifier block in a prenex first order formula of the elementary language of $\mathbb{C}$ or the computation of a greatest common divisor of a finite set of circuit represented polynomials requires *exponential time*. Complexity results in this spirit became already achieved in [GH01] and [CGH$^+$03] (compare also Proposition 18 and Observation in Section 6.5.1).

6.5.3 Arithmetization techniques for boolean circuits

Let $m \in \mathbb{N}$ and let $0, 1$ and $Z_1, \ldots, Z_m$ be given constants and variables. Let $Z := (Z_1, \ldots, Z_m)$. Following the context we shall interpret $0, 1$ as boolean values or the corresponding complex numbers and $Z_1, \ldots, Z_m$ as boolean variables or indeterminates over $\mathbb{C}$. With $\wedge, \vee, {}^-$ we denote boolean "and", "or" and "not". A boolean circuit b with inputs $Z_1, \ldots, Z_m$ is a DAG whose indegree zero nodes are labelled by $0, 1$ and $Z_1, \ldots, Z_m$ and whose inner nodes have indegree two or one. In the first case an inner node is labelled by $\wedge$ or $\vee$ and in the second by ${}^-$. Some inner nodes of b become labelled as outputs. We associate with b a semantics as follows:

- indegree zero nodes which are labelled by $0, 1$ become interpreted by the corresponding constant functions $\{0,1\}^m \to \{0,1\}$,
- indegree zero nodes which are labelled by $Z_1, \ldots, Z_m$ become interpreted by the corresponding projection function $\{0,1\}^m \to \{0,1\}$,
- let ρ be an inner node of b of indegree two whose parent nodes ρ_1 and ρ_2 became already interpreted by boolean functions $g_{\rho_1}, g_{\rho_2} : \{0,1\}^m \to \{0,1\}$. If ρ is labelled by $\wedge$, we interpret ρ by the boolean function $g_\rho := g_{\rho_1} \wedge g_{\rho_2}$ and if ρ is labelled by $\vee$, we interpret ρ by the boolean function $g_\rho := g_{\rho_1} \vee g_{\rho_2}$,
- let ρ be an inner node of b of indegree one whose parent node ρ' became already interpreted by a boolean function $g_{\rho'} : \{0,1\}^m \to \{0,1\}$. Then we interpret ρ by the boolean function $g_\rho := \overline{g_{\rho'}}$.

For a node ρ of b we call g_ρ the *intermediate result* of b at ρ. If ρ is an output node, we call g_ρ a *final result* of b. An arithmetization β of the boolean circuit b consists of the same DAG as b with a different labelling as follows.

Let U, V be new indeterminates over $\mathbb{C}$.

The constants $0, 1$ become interpreted by the correspondent complex numbers and $Z_1, \ldots, Z_m$ as indeterminates over $\mathbb{C}$. Let ρ be an inner node of β. If ρ has indegree two, then ρ becomes labelled by a polynomial $R_\rho \in \mathbb{Z}[U, V]$ and if ρ has indegree one by a polynomial $R_\rho \in \mathbb{Z}[U]$. The output nodes of β and b are the same.

Representing for each inner node ρ of β the polynomial G_ρ by a division–free ordinary arithmetic circuit over $\mathbb{Z}$ in the inputs U, V or U, we obtain an ordinary division–free arithmetic circuit over $\mathbb{Z}$ in the inputs $Z_1, \ldots, Z_m$.

Although this transformation of the DAG of β depends on the choice we made to represent for each node ρ of β the label R_ρ, we shall denote the resulting arithmetic circuit also by β.

Just as we did in Section 6.4.1 we may associate with β a semantics which determines for each node ρ of β a polynomial $G_\rho \in \mathbb{Z}[Z]$. We say that β is an arithmetization of the boolean circuit b if the following condition is satisfied:

> for any node ρ of b and any argument $z \in \{0,1\}^m$ the boolean value $g_\rho(z)$ coincides with the arithmetic value $G_\rho(z)$ (in a somewhat imprecise notation: $g_\rho(z) = G_\rho(z)$).

An example of an arithmetization procedure is given by the map which associates to each node ρ of b a polynomial $[g_\rho]$ of $\mathbb{Z}[Z]$ satisfying the following conditions:

- $[0] = 0$, $[1] = 1$, $[Z_1] = Z_1, \ldots, [Z_m] = Z_m$
- for ρ an inner node of indegree two of b with parents ρ_1 and ρ_2:

$$[g_\rho] = [g_{\rho_1}] \cdot [g_{\rho_2}] \text{ if the label of } \rho \text{ is } \wedge$$

and

$$[g_\rho] = [g_{\rho_1}] + [g_{\rho_2}] - [g_{\rho_1}].[g_{\rho_2}] \text{ if the label of } \rho \text{ is } \vee$$

- for ρ an inner node of indegree one of b with parent ρ':

$$[g_\rho] = 1 - [g_{\rho'}].$$

The resulting arithmetic circuit is called the *standard arithmetization* of b (see e.g. [Sha92] and [BF91]).

Let $n, r \in \mathbb{N}$ and $U_1, \ldots, U_r, X_1, \ldots, X_n$ be new variables. For $m := n + r$ we replace now Z by U and X, where $U := (U_1, \ldots, U_r)$ and $X := (X_1, \ldots, X_n)$. We shall interpret $U_1, \ldots, U_r$ as parameters and $X_1, \ldots, X_n$ as input variables.

Let b be a boolean circuit with the inputs $U_1, \ldots, U_r, X_1, \ldots, X_n$ and just one final result $h : \{0,1\}^r \times \{0,1\}^n \to \{0,1\}$.

We wish to describe the set of instances $u \in \{0,1\}^r$ where $h(u, X_1, \ldots, X_n)$ is a satisfiable boolean function.

For this purpose let us choose an arithmetization β of b. We interpret β as an ordinary arithmetic circuit over $\mathbb{Z}$ with the parameters $U_1, \ldots, U_r$ and the inputs $X_1, \ldots, X_n$. The single final result of β is a polynomial $H \in \mathbb{Z}[U, X]$ which satisfies for any $u \in \{0,1\}^r$, $x \in \{0,1\}^n$ the following condition:

$$h(u, x) = H(u, x).$$

With out loss of generality we may suppose that the polynomials $X_1^2 - X_1, \ldots, X_n^2 - X_n$ are intermediate results of β. We relabel now β such that these polynomials and H become the final results of β and observe that $X_1^2 - X_1 = 0, \ldots, X_n^2 - X_n = 0$ and H represents a flat family of zero–dimensional elimination problems.

Let Y be a new indeterminate and let $F \in \mathbb{Z}[U, Y]$ the associated elimination polynomial. One verifies easily the identity

$$F(U, Y) = \prod_{x \in \{0,1\}^n} (Y - H(U, x)).$$

Let $\mathcal{A}$ be an essentially division–free procedure of our extended computation model which solves algorithmically the general instance of any flat family of zero–dimensional elimination problems. Then β is an admissible input for $\mathcal{A}$ and there exists an integer $q \in \mathbb{N}$ such that F^q is the final result of $\mathcal{A}_{\text{final}}(\beta)$.

We consider now the task to count for any $u \in \{0,1\}^r$ the number k of instances $x \in \{0,1\}^n$ with $h(u, x) = 1$.

The polynomial F^q encodes two possible solutions of this task.

The first solution is the following: let l be the order of the univariate polynomial $F^q(u, Y)$ at zero. Then q divides l and we have $k = 2^n - \frac{l}{q}$.

The second and more interesting solution is the following: write $F^q = Y^{2^n q} + \varphi_1 Y^{2^n q - 1} + \cdots + \varphi_{2^n q}$ with $\varphi_1, \ldots, \varphi_{2^n q} \in \mathbb{Z}[U]$. Then $\varphi_1(u)$ is an integer which is divisible by q and we have $k = -\frac{\varphi(u)}{q}$.

Observe also $\deg \varphi_1 \leq \deg_U H$.

These considerations show the relevance of an *efficient* evaluation of the polynomial F^q (e.g. by the circuit $\mathcal{A}_{\text{final}}(\beta)$).

We ask therefore whether $\mathcal{A}_{\text{final}}(\beta)$ can be polynomial in the size of the boolean circuit b. The following example illustrates that the answer may become negative.

In the sequel we are going to exhibit for $r := 2n + 1$ a boolean circuit b of size $O(n)$ which evaluates a function $h : \{0,1\}^r \times \{0,1\}^n$ such that the standard arithmetization β of b represents a flat family of zero–dimensional elimination problems with associated elimination polynomial F and such that any essentially division–free procedure $\mathcal{A}$ of our extended computation model that accepts the input β and computes by means of $\mathcal{A}_{\text{final}}(\beta)$ a power of F, requires time $\Omega(2^n)$ for this task. This means that it is unlikely that algorithms designed following the paradigm of object oriented programming are able to evaluate the polynomial φ_1 efficiently.

On the other hand, since the degree of φ_1 is bounded by $\deg_U H$ and therefore "small", there exists a polynomial time interactive protocol which checks for any $u \in \{0,1\}^r$ and any $c \in \mathbb{Z}$ the equation $\varphi_1(u) = c$. Thus this problem belongs to the complexity class IP (see [LFKN92] for details).

We are now going to exhibit an example of a boolean circuit which highlights the unfeasibility of our computation task.

For this purpose let $r := 2n+1$ and $S_1, \ldots, S_n, T, U_1, \ldots, U_n$ parameters and $X_1, \ldots, X_n$ input variables and let $S := (S_1, \ldots, S_n)$ and $U := (U_1, \ldots, U_n)$.

We consider the boolean function $h : \{0,1\}^{2n+1} \times \{0,1\}^n \to \{0,1\}$ defined by the boolean formula

$$\phi := \bigwedge_{1 \leq i \leq n} (\overline{X_i} \vee (S_i \wedge X_i)) \vee (T \wedge \bigwedge_{1 \leq i \leq n} (\overline{X_i} \vee (U_i \wedge X_i))).$$

From the structure of the formula ϕ we infer that h can be evaluated by a boolean circuit b of size $O(n)$ in the inputs $S_1, \ldots, S_n, T, U_1, \ldots, U_n$.

Let β be the standard arithmetization of the boolean circuit b and let H be the final result of β. Observe that the total, and hence the non–scalar size of β is $O(n)$. Then we have

$$H = \prod_{1 \leq i \leq n} (1 + (S_i - 1)X_i) + (1 - \prod_{1 \leq i \leq n} (1 + (S_i - 1)X_i))T \prod_{1 \leq i \leq n} (1 + (U_i - 1)X_i).$$

Observe that the equations $X_1^2 - X_1 = 0, \ldots, X_n^2 - X_n = 0$ and the polynomial H represent a flat family of zero–dimensional elimination problems. Let F be the associated elimination polynomial. Then F can be written as

$$F = Y^{2^n} + B_1 Y^{2^n - 1} + \cdots + B_{2^n}$$

with

$$B_k = (-1)^k \sum_{0 \leq j_1 < \cdots < j_k < 2^n} \prod_{1 \leq h \leq k} (-(\prod_{1 \leq i \leq n} S_i^{[j_h]_i} + (1 - \prod_{1 \leq i \leq n} S_i^{[j_h]_i})T \prod_{1 \leq i \leq n} U_i^{[j_h]_i}))$$

for $1 \leq k \leq 2^n$.

Let

$$L_k := \sum_{0 \leq j_1 < \cdots < j_k < 2^n} \sum_{1 \leq h \leq k} \prod_{1 \leq i \leq n} S_i^{[j_1]_i} \ldots (1 - \prod_{1 \leq i \leq n} S_i^{[j_h]_i}) \cdots \prod_{1 \leq i \leq n} S_i^{[j_k]_i} \prod_{1 \leq i \leq n} U_i^{[j_h]_i},$$

where $1 \leq k \leq 2^n$.

Then we have

$$B_k = (-1)^k \sum_{0 \leq j_1 < \cdots < j_k < 2^n} \prod_{1 \leq i \leq n} S_i^{[j_1]_i} \cdots \prod_{1 \leq i \leq n} S_i^{[j_k]_i} + (-1)^k L_k . T + \text{ terms of higher degree in } T$$

Observe now that for $1 \leq k \leq 2^n$ the monomial $\prod_{1 \leq i \leq n} S_i^{[1]_i} \cdots \prod_{1 \leq i \leq n} S_i^{[k]_i} \prod_{1 \leq i \leq n} U_i^{[k]_i}$ occurs only in L_k with a non–zero coefficient.

This implies that $L_1, \ldots, L_{2^n}$ are $\mathbb{C}$–linearly independent in $\mathbb{C}[S, T, U]$.

With this preparation we are now able to repeat textually the arguments in the proof of Theorem 17 of Section 6.5.1 in order to show the following statement.

Theorem 20 *Let $\mathcal{A}$ be an essentially division free procedure of our extended computation model which accepts the arithmetic circuit β as input. Suppose that $\mathcal{A}_{final}(\beta)$ has a unique final result and that it is a power of the elimination polynomial F. Then the non–scalar size of $\mathcal{A}_{final}(\beta)$ is at least $\Omega(2^n)$.*

6.5.4 The multivariate resultant

Let X_0 be a new indeterminate.

Turning back to the proof of Theorem 17, one verifies easily that the polynomial F is the (multivariate) resultant of the, in the variables $X_0, \ldots, X_n$ homogeneous, polynomial equation system

$$\begin{aligned} X_1^2 - X_0 X_1 &= 0, \\ &\vdots \\ X_n^2 - X_0 X_n &= 0, \\ \overline{H} &= 0, \end{aligned} \tag{21}$$

with $\overline{H} := Y X_0^n - \sum_{1 \leq i \leq n} 2^{i-1} X_0^{n-1} X_i - T \prod_{1 \leq i \leq n} (X_0 + (U_i - 1) X_i)$.

Supposing that the polynomials of the equation system (21) are the final results of a given robust parameterized arithmetic circuit β with basic parameters $Y, T, U_1, \ldots, U_n$ and inputs $X_0, \ldots, X_n$ and applying to β any procedure $\mathcal{A}$ of our extended computation model such that $\mathcal{A}$ returns a circuit γ for the resultant of the system (21), we are unable to deduce from Theorem 17 any lower bound for the size of γ, except we make the architectural requirement that the procedure $\mathcal{A}$ does not apply joins to elementary subroutines that involve recursion. This drawback is due to the fact that the given resultant procedure $\mathcal{A}$ treats the parameters $Y, T, U_1, \ldots, U_n$ of the circuit β in equal conditions, whereas the procedure of Theorem 17 distinguishes between $T, U_1, \ldots, U_n$ as basic parameters and Y as output variable. However, many applications of resultant computations distinguish between the nature of the parameters occurring in the given homogeneous input system and in these cases Theorem 17 may be applied. Mutatis mutandis the same conclusion is true for all known standard procedures for resultant computation.

This point of view is also supported by experience in effective elimination theory. Classically the (implicit) distinction between basic parameters and input variables appears already in [Kro87]. The parameters become treated as elements of a ring $\mathcal{R}$ which contain the coefficients of the input polynomials. The input variables are treated symbolically. The classic elimination procedures are based on linear algebra in $\mathcal{R}$. The correctness proofs of these procedures use specializations, i.e. $\mathbb{C}$–algebra homomorphisms $\mathcal{R} \to \mathbb{C}$. This is exactly the way we did proceed before in the discussion of the particular instances of the flat families of zero–dimensional elimination problems. The notion of a robust parameterized arithmetic circuit makes this view sound on the level of implementations.

Suppose now we have to eliminate *two* blocks of quantifiers. Traditionally, the variables of the second block are considered as basic parameters of the first one. When we eliminate the variables of the first block we obtain expressions in these parameters. In order to apply our algorithmic model to these expressions we have to transform the basic parameters of the first block into new input variables. In standard elimination procedures this is achieved by a careful analysis of the computation previously performed with the parameters of the first block. Our way is alternative. We consider the variables of both blocks as input variables of the initial system which become eliminated in the first round $\mathcal{A}^{(1)}$ of a procedure $\mathcal{A}$ like the one introduced at the end of Section 6.4.2. Then the basic parameters of the first block, i.e. the variables of the second block, become reintroduced as output variables in a second round, realized by the subprocedure $\mathcal{A}^{(2)}$. This finishes the elimination of the first block of quantifiers. The elimination of the second block of

quantifiers takes then as input the output of $\mathcal{A}^{(2)}$ and proceeds as before. What remains at the end of an iterated quantifier elimination process are only parameters.

6.5.5 Divisions and blow ups

We are now going to analyse the main argument of the proof of Theorem 17 from a geometric point of view.

We recall first some notations and assumptions we made during this proof.

With respect to the indeterminates $X_1, \ldots, X_n$, we considered the vector θ of coefficients of the expression

$$H = \sum_{1 \leq i \leq n} 2^{i-1} X_i + T \prod_{1 \leq i \leq n} (1 + (U_i - 1)X_i)$$

as a polynomial map $\mathbb{A}^{n+1} \to \mathbb{A}^{2^n}$ with image $\mathcal{T}$. Recall that $\mathcal{T}$ is an irreducible constructible subset of $\mathbb{A}^{2^n}$.

Further, with respect to the indeterminate Y, we considered the vector φ of nontrivial coefficients of the monic polynomial

$$F = \prod_{1 \leq j \leq 2^n - 1} (Y - (j + T \prod_{1 \leq i \leq n} U_i^{[j]_i}))$$

also as a polynomial map $\mathbb{A}^{n+1} \to \mathbb{A}^{2^n}$.

One sees immediately that there exists a unique polynomial map $\eta : \mathcal{T} \to \mathbb{A}^{2^n}$ such that $\varphi = \eta \circ \theta$ holds. Using particular properties of θ and φ we showed implicitly in the proof of Theorem 17 that η satisfies the following condition:

> *Let m be a natural number, $\zeta : \mathcal{T} \to \mathbb{A}^m$ a geometrically robust constructible and $\pi : \mathbb{A}^m \to \mathbb{A}^{2^n}$ a polynomial map such that $\eta = \pi(\zeta) = \pi \circ \zeta$ holds. Then we have the following estimate:*
>
> $$m \geq 2^n.$$

This means that the following computational task cannot be solved efficiently:

Allowing certain restricted divisions reduce the datum θ consisting of 2^n entries to a datum ζ consisting of only $m \leq 2^n$ entries such that the vector η may be recovered from ζ without using any division, i.e. by an ordinary division–free arithmetic circuit over $\mathbb{C}$.

Here the divisions become restricted to those which lead to quotients which are still geometrically robust functions defined on $\mathcal{T}$.

In order to simplify the following discussion we shall assume without loss of generality that all our constructible maps may be robustly extended to $\overline{\mathcal{T}}$.

Let f and g be two elements of the coordinate ring $\mathbb{C}[\overline{\mathcal{T}}]$ of the affine variety $\overline{\mathcal{T}}$ and suppose that $g \neq 0$ holds and that the element $\frac{f}{g}$ of the rational function field $\mathbb{C}(\overline{\mathcal{T}})$ may be extended to a robust constructible function defined on $\overline{\mathcal{T}}$, which we denote also by $\frac{f}{g}$, since this extension is unique.

Then we have two cases: the coordinate function g divides f in $\mathbb{C}[\overline{\mathcal{T}}]$ or not. In the first case we may compute $\frac{f}{g}$, by means of an ordinary division–free arithmetic circuit over

$\mathbb{C}$, from the restrictions to $\overline{\mathcal{T}}$ of the canonical projections $\mathbb{A}^{2^n} \to \mathbb{A}^1$. Thus $\frac{f}{g}$ belongs to the coordinate ring $\mathbb{C}[\overline{\mathcal{T}}]$. In the second case this is not possible and $\mathbb{C}[\overline{\mathcal{T}}][\frac{f}{g}]$ is a proper extension of $\mathbb{C}[\overline{\mathcal{T}}]$ in $\mathbb{C}(\overline{\mathcal{T}})$. In both cases $\mathbb{C}[\overline{\mathcal{T}}][\frac{f}{g}]$ is the coordinate ring of an affine chart of the blow up of $\mathbb{C}[\overline{\mathcal{T}}]$ at the ideal generated by f and g. We refer to this situation as a *division blow up* which we call *essential* if $\frac{f}{g}$ does not belong to $\mathbb{C}[\overline{\mathcal{T}}]$.

Therefore we have shown in the proof of Theorem 17 that essential division blow ups do not help to solve efficiently the reduction task formulated before.

These comments lead us to the consideration of the following situation.

Example 21 Let L be a natural number, let $n_L := \binom{2^L+n}{n}$ and let $W_{L,n}$ be the set of coefficient vectors of all polynomials of $\mathbb{C}[X_1, \ldots, X_n]$ which can be evaluated by an ordinary division–free arithmetic circuit of non–scalar size at most L. Let $\overline{W_{L,n}}$ be the Zariski closure of $W_{L,n}$ in $\mathbb{A}^{n_L}$. The polynomials whose coefficient vectors belong to $\overline{W_{L,n}}$ have degree at most 2^L and therefore we may interpret them also as elements of $\mathbb{C}[X_1, \ldots, X_n]_{2^L} := \left\{ f \in \mathbb{C}[X_1, \ldots, X_n]; \deg f \leq 2^L \right\}$ (observe that $\mathbb{C}[X_1, \ldots, X_n]_{2^L}$ and $\mathbb{A}^{n_L}$ are isomorphic as $\mathbb{C}$–vector spaces).

Following [CGH$^+$03], Corollary 2 we may chose $K := L(L+n+1)^2 + 2$, points $\alpha_1, \ldots, \alpha_K \in \mathbb{Z}^n$ of bit length at most $4(L+1)$ such that the polynomial map $\Xi : \overline{W_{L,n}} \to \mathbb{A}^K$, defined for $f \in \overline{W_{L,n}}$ by $\Xi := (f(\alpha_1), \ldots, f(\alpha_K))$, becomes injective. Let $\mathcal{D}_{L,n} := \Xi(W_{L,n})$. It turns out that $\overline{\mathcal{D}_{L,n}}$ is an irreducible, closed affine subvariety of $\mathbb{A}^K$ and that the by Ξ induced morphism between irreducible affine varieties, denoted by $\Xi_{L,n} : \overline{W_{L,n}} \to \overline{\mathcal{D}_{L,n}}$ is finite, birrational and bijective (but definitely not an isomorphism of varieties). The inverse map of $\Xi_{L,n}$ induces therefore a geometrically robust constructible map $\Phi_{L,n} : \mathcal{D}_{L,n} \to \mathbb{C}[X_1, \ldots, X_n]_{2^L}$ which to any point $y = (y_1, \ldots, y_K)$ of $\mathcal{D}_{L,n}$ assigns the (unique) polynomial $f \in W_{L,n}$ with $f(\alpha_1) = y_1, \ldots, f(\alpha_K) = y_K$. Thus $\Phi_{L,n}$ determines a geometrically robust Hermite–Lagrange interpolation problem in the sense of Section 4.

A geometrically robust solution of this interpolation problem is now given by a geometrically robust constructible map $\Psi : \mathcal{D}_{L,n} \to \mathbb{A}^M$ (with irreducible, constructible image $\mathcal{D}^*$) and a polynomial map $\omega : \mathcal{D}^* \to \mathbb{C}[X_1, \ldots, X_n]_{2^L}$ such that $\Phi_{L,n} = \omega \circ \Psi$ holds (see Section 4.3)

From [GHMS11], Theorem 23 we deduce now $M \geq \binom{2^{\lfloor \frac{L}{2} - 1 \rfloor} - 1 + n}{n} = 2^{\Omega(Ln)}$. This means that one has to perform at least $2^{\Omega(Ln)}$ divisions for the geometrically robust evaluation of Ψ. With other words, any decomposition of the rational map $\Phi_{L,n}$ in (essential) division blow ups and a polynomial map requires at least $2^{\Omega(Ln)}$ blow ups.

Following [Har92], Theorem 7.2.1 any rational map may be decomposed into a finite sequence of successive blow ups followed by a regular morphism of algebraic varieties. Our method indicates the interest to find lower bounds for the number of blow ups (and their embedding dimensions) necessary for an effective variant of this result.

Problem adapted methods for proving lower bounds for the number of blow ups necessary to resolve singularities would also give indications which order of complexity can be expected for efficient desingularization algorithms (see [EV00]). At this moment only upper bound estimations are known [Bla09].

6.5.6 Comments on complexity models for geometric elimination

The question, whether $P \neq NP$ or $P_{\mathbb{C}} \neq NP_{\mathbb{C}}$ holds in the classical or the BSS Turing Machine setting, concerns only computational *decision* problems. These, on their turn, are closely related to the task to construct *efficiently*, for a given prenex existential formula, an equivalent, quantifier free one (compare [BSS89], [HM93], [SS95] and [BCSS98]). In the sequel we shall refer to this and to similar, geometrically motivated computational tasks as "*effective elimination*".

Theorem 17 in Section 6.5.1 does not establish a fact concerning decision problems like the $P_{\mathbb{C}} \neq NP_{\mathbb{C}}$ question. It deals with the *evaluation of a function* which assigns to suitable prenex existential formulas over $\mathbb{C}$ *canonical*, equivalent and quantifier–free formulas of the same elementary language.

Theorem 17 says that in our computation model this function cannot be evaluated efficiently. If we admit also *non–canonical* quantifier–free formulas as function values (i.e. as outputs of our algorithms), then this conclusion remains true, provided that the calculation of parameterized greatest common divisors is feasible and efficient in our model (see [CGH$^+$03], Section 5).

It is not clear what this implies for the $P_{\mathbb{C}} \neq NP_{\mathbb{C}}$ question.

Intuitively speaking, our exponential lower complexity bound for effective geometric elimination is only meaningful and true for computer programs designed in a professional way by software engineers. Hacker programs are excluded from our considerations.

This constitutes an enormous difference between our approach and that of Turing machine based complexity models, which always include the hacker aspect. Therefore the proof of a striking lower bound for effective elimination seems unfeasible in these models. Experience confirms this conclusion.

Our argumentation is based on the requirement of output parametricity which on its turn is the consequence of two other requirements, a functional and a non–functional one, that we may employ alternatively. More explicitly, we require that algorithms (and their specifications) are described by sound asserted programs or, alternatively, that they behave well under reductions (see Sections 6.4.1 and 6.4.2).

Let us observe that the complexity statement of Theorem 17 refers to the relationship input–output and not to a particular property of the output alone. In particular, Theorem 17 does not imply that certain polynomials, discussed below, like the permanent or the Pochhammer polynomials, are hard to evaluate.

Let notations and assumptions be as in Section 6.5.1. There we considered for arbitrary $n \in \mathbb{N}$ the flat family of zero dimensional elimination problems

$$G_1^{(n)} = 0, \ldots, G_n^{(n)} = 0, H^{(n)}$$

given by

$$G_1^{(n)} := X_1^2 - X_1, \ldots, G_n^{(n)} := X_n^2 - X_n$$

and

$$H^{(n)} := \sum_{1 \leq i \leq n} 2^{i-1} X_i \ + \ T \prod_{1 \leq i \leq n} (1 + (U_i - 1)X_i)$$

Let $X_{n+1}, \ldots, X_{3n-1}$ be new indeterminates and let us consider the following polynomials

$$G_{n+1}^{(n)} := X_{n+1} - 2X_2 - X_1, \ldots, G_j^{(n)} := X_j - X_{j-1} - 2^{j-n} X_{j-n+1}, \ \ n+2 \leq j \leq 2n-1,$$

$$G_{2n}^{(n)} := X_{2n} - U_1 X_1 + X_1 - 1,$$

$$G_k^{(n)} := X_k - U_{k-2n+1} X_{k-1} X_{k-2n+1} + X_{k-1} X_{k-2n+1} - X_{k-1},\ \ 2n+1 \leq k \leq 3n-1$$

and

$$L^{(n)} := X_{2n-1} + T X_{3n-1}.$$

One verifies easily that $G_1^{(n)} = 0, \ldots, G_{3n-1}^{(n)} = 0, L^{(n)}$ is another flat family of zero dimensional elimination problems and that both families have the same associated elimination polynomial, namely

$$F^{(n)} := \prod (Y - (j + T \prod_{1 \leq i \leq n} U_i^{[\rho]_i}))$$

Suppose now that there is given an essential division–free procedure $\mathcal{A}$ of our extended computation model which solves algorithmically the general instance of any given flat family of zero–dimensional elimination problems.

Let β_n and β_n^* be two essentially division–free, robust parameterized arithmetic circuits which implement the first and the second flat family of zero dimensional elimination problems we are considering.

Then β_n and β_n^* are necessarily distinct circuits. Therefore $\mathcal{A}_{final}(\beta_n)$ and $\mathcal{A}_{final}(\beta_n^*)$ represent two (not necessarily distinct) implementations of the elimination polynomial $F^{(u)}$ by essentially division–free, robust parameterized arithmetic circuits.

From Theorem 17 and its proof we are only able to deduce that the circuit $\mathcal{A}_{final}(\beta_n)$ has non–scalar size at least $\Omega(2^n)$ but nothing about the non–scalar size of $\mathcal{A}_{final}(\beta_n^*)$.

In the past, many attempts to show the non–polynomial character of the elimination of just one existential quantifier block in the arithmetic circuit based elementary language over $\mathbb{C}$, employed the reduction to the proof that a certain sequence of specific polynomials was hard to evaluate (this approach was introduced in [HM93] and became adapted to the BSS model in [SS95]).

The Pochhammer polynomials and the generic permanents discussed below form such sequences.

We are now going to analyse the relationship of Theorem 17 and its proof with sequences of polynomials which presumably are not evaluable in polynomial time.

The following flat family of zero dimensional elimination polynomials is inspired for $n \in \mathbb{N}$ in the corresponding real knapsack problem (see [BCS97], Chapter 3.4):

$$G_1^{(n)} = 0, \ldots, G_n^{(n)} = 0, K^{(n)}$$

with $G_1^{(n)} := X_1^2 - X_1, \ldots, G_n^{(n)} := X_n^2 - X_n$ and $K^{(n)} := U_1 X_1 + \cdots + U_n X_n$. The associated elimination polynomial $P^{(n)}$ is

$$P^{(n)} := \prod_{(\epsilon_1, \ldots, \epsilon_n) \in \{0,1\}^n} (Y - (\epsilon_1 U_1 + \cdots + \epsilon_n U_n)).$$

One sees easily that $G_1^{(n)}, \ldots, G_n^{(n)}$ and $K^{(n)}$ may be evaluated by a division–free ordinary arithmetic circuit β_n' of non–scalar size $O(n)$. Let us fix for the moment $n \in \mathbb{N}$.

We consider again the essentially division–free procedure $\mathcal{A}$ introduced before. Observe first that any recursive subroutine of $\mathcal{A}$ satisfies on input β_n' the requirement (B) of Section

6.4.1 at any node. Therefore the requirement (B) becomes redundant on input β'_n. In particular, the output isoparametricity of $\mathcal{A}$ on input β'_n becomes guaranteed and has not to be required.

Moreover, the intermediate results of $\mathcal{A}(\beta'_n)$ are all polynomials over $\mathbb{C}$. On the other hand, there exists for any sequence $\delta := (\delta_n)_{n\in\mathbb{N}}$ of ordinary division–free circuits δ_n, $n \in \mathbb{N}$, with inputs $U_1, \ldots, U_n, Y$ and with a single final result, namely $P^{(n)}$, such that all intermediate results of δ_n are polynomials over $\mathbb{C}$, an essentially division–free procedure $\mathcal{A}^{(\delta)}$ of our computation model satisfying the following conditions:

- $\mathcal{A}^{(\delta)}$ solves algorithmically the general instance of any given flat family of zero–dimensional elimination problems,

- for any $n \in \mathbb{N}$ the robust, parameterized arithmetic circuit $\mathcal{A}^{(\delta)}_{final}(\beta'_n)$ can be reduced to the circuit δ_n.

Theorem 17 implies now that a possible polynomial upper bound for the non–scalar complexity of the polynomials $P^{(n)}, n \in \mathbb{N}$, cannot be the consequence of a general upper bound for the size of the output circuits of elimination procedures of our computational model like the procedure $\mathcal{A}$ we are considering.

With other words, possible non–polynomial lower or polynomial upper bounds for the complexity of the sequence $(P^{(n)})_{n\in\mathbb{N}}$ require *ad hoc proofs*. They are not consequence of a general result about the complexity of effective elimination.

Let us finish this section with a word about hacking and interactive (zero–knowledge) proofs.

Hackers work in an ad hoc manner and quality attributes are irrelevant for them. We may simulate a hacker and his environment by an *interactive proof system* where the prover, identified with the hacker, communicates with the verifier. Thus, in our view, a hacker disposes over unlimited computational power, but his behaviour is deterministic. Only his communication with the verifier underlies some quantitative restrictions: communication channels are tight. Hacker and verifier become linked by a protocol of the following (zero–knowledge) type.

Inputs are natural numbers in *unary* representation. Each natural number represents a mathematical object belonging to a previously fixed abstract data type of polynomials. For example $n \in \mathbb{N}$ may represent the 2^n–th Pochhammer polynomial

$$T^{\overline{2^n}} := \prod_{0\leq j<2^n} (T - j)$$

or the n–th generic permanent

$$Perm_n := \sum_{\tau\in Sym(n)} X_{1\tau(1)}, \ldots, X_{n\tau(n)},$$

where T and $X_{11}, \ldots, X_{nn}$ are new indeterminates and $Sym(n)$ denotes the symmetric group operating on n elements.

On input $n \in \mathbb{N}$ the hacker sends to the verifier a division–free labelled DAG Γ_n (i.e. a division–free ordinary arithmetic circuit over $\mathbb{Z}$) of size $n^{O(1)}$ and claims that Γ_n evaluates the polynomial represented by n.

The task of the verifier is now to check this claim in uniform, bounded probabilistic or non–uniform polynomial time, namely in time $n^{O(1)}$.

In the case of the Pochhammer polynomial and the permanent such a protocol exists for the non–uniform complexity model. This can be formulated as follows.

Proposition 22 *The languages*

$$\mathcal{L}_{Poch} := \{(n, (\Gamma_j)_{0\leq j\leq n});\ \Gamma_j \textit{ is for } 0 \leq j \leq n \textit{ a division–free labelled DAG evaluating } T^{\overline{2^j}},\ n \in \mathbb{N}\}$$

and

$$\mathcal{L}_{Perm} := \{(n, \Gamma);\ \Gamma \textit{ is a labelled DAG evaluating } Perm_n,\ n \in \mathbb{N}\}$$

belong to the complexity class BPP and hence to P/poly (here $n \in \mathbb{N}$ is given in unary representation).

Proof. We show only that $\mathcal{L}_{Poch}$ belongs to the complexity class *P/poly*. The proof that $\mathcal{L}_{Poch}$ belongs to BPP follows the same kind of argumentation and will be omitted here. The case of the language $\mathcal{L}_{Perm}$ can be treated analogously and we shall not do it here (compare [KI04], Section 3).

Let $n \in \mathbb{N}$ and let Γ be a division–free labelled DAG with input T and a single output node. Let Γ' be the division–free labelled DAG which is given by the following construction:

- choose a labelled acyclic graph μ_n of size $n + O(1)$ with input T and with $T - 2^{2^{n-1}}$ as single final result
- take the union $\overline{\Gamma}$ of the circuits Γ and $\Gamma * \mu_n$ and connect the two output nodes of $\overline{\Gamma}$ by a multiplication node which becomes then the single output node of the resulting circuit Γ'.

From the polynomial identity $T^{\overline{2^n}} = T^{\overline{2^{n-1}}}(T) \ \cdot \ T^{\overline{2^{n-1}}}(T - 2^{2^{n-1}})$ one deduces easily that Γ' computes the polynomial $T^{\overline{2^n}}$ if and only if Γ computes the polynomial $T^{\overline{2^{n-1}}}$.

For $0 \leq j \leq n$ let Γ_j be a division–free labelled DAG with input T and a single output node.

Suppose that in the previous construction the circuit Γ is realized by the labelled directed acyclic graph Γ_{n-1}. Then one sees easily that $(n, (\Gamma_j)_{0\leq j\leq n})$ belongs to $\mathcal{L}_{Poch}$ if and only if the following conditions are satisfied:

(i) the circuit Γ_0 computes the polynomial T,

(ii) the circuits Γ' and Γ_n compute the same polynomial,

(iii) $(n-1, (\Gamma_j)_{0\leq j\leq n-1})$ belongs to $\mathcal{L}_{Poch}$.

Therefore, if condition (ii) can be checked in non–uniform polynomial time, the claimed statement follows.

For $0 \leq j \leq n$ let L_j and L be the sizes of the labelled directed acyclic graphs Γ_j and Γ' and observe that $L = 2L_{n-1} + n + O(1)$ holds.

Let P_{n-1} and P be the final results of the circuits Γ_{n-1} and Γ'. From [CGH$^+$03], Corollary 2 we deduce that there exist $m := 4(L+2)^2 + 2$ integers $\gamma_1, \ldots, \gamma_m \in \mathbb{Z}$ of bit length at most $2(L+1)$ such that the condition (ii) above is satisfied if and only

(iv) $P_{n-1}(\gamma_1) = P(\gamma_1), \ldots, P_{n-1}(\gamma_m) = P(\gamma_m)$

holds.

From [HM] we infer that condition (iv) can be checked by a nondeterministic Turing machine with advise in (non–uniform) time $O(L^3) = O((L_{n-1} + n)^3)$.

Applying this argument recursively, we conclude that membership of $(n, (\Gamma_j)_{0 \leq j \leq n})$ to $\mathcal{L}_{Poch}$ may be decided in non–uniform time $O(\sum_{0 \leq j \leq n}(L_j + j)^3)$ and therefore in polynomial time in the input size. Hence the language $\mathcal{L}_{Poch}$ belongs to the complexity class $P/poly$. The proof of the stronger result, namely $\mathcal{L}_{Poch} \in$ BPP, is similar. ■

Finally we observe that for $n \in \mathbb{N}$ the Pochhammer polynomial $T^{\overline{2^n}}$ is the associated elimination polynomial of the particular problem instance, given by $T := 0$, of the flat family of zero–dimensional elimination problems $G_1^{(n)} = 0, \ldots, G_n^{(n)} = 0, H^{(n)}$, which we considered at the beginning of this section.

From the point of view of effective elimination, the sequence of Pochhammer polynomials becomes discussed in [HM93] (see also [SS95]). From the point of view of factoring integers, Pochhammer polynomials are treated in [Lip94].

Let us mention that our approach to effective elimination, which led to Theorem 17 and preliminary forms of it, was introduced in [HMPW98] and extended in [GH01] and [CGH$^+$03].

The final outcome of our considerations in Sections 6.5.1 and 6.5.6 is the following: neither mathematicians nor software engineers, nor a combination of them will ever produce a practically satisfactory, *generalistic* software for elimination tasks in Algebraic Geometry. This is a job for *hackers* which may find for *particular* elimination problems *specific* efficient solutions.

7 Conclusions

7.1 Concluding remarks

This thesis can be seen as a contribution to the better understanding of quality attribute trade–offs in quantifier elimination in elementary Algebraic Geometry. We have analysed two properties namely coalescence and branching parsimoniousness which were introduced under the aspect of quality attributes. This properties were introduced by means of a software architecture which was reached through an strict distinction between the problem and the algorithm which solves it.

The mathematical notions and geometric tools which support our software architecture come from the previous works [CGH$^+$03] and [GHMS11]. This leads us to the formulation of a terminology dictionary which translates the terminology of these works to our terminology of software engineering represented by [Mey00]. This dictionary constitutes an interpretation in the language of software architecture of the works [CGH$^+$03] and [GHMS11].

7.2 An ontological view of our investigation

The Greek philosopher Plato could be considered a software engineer from the perspective of software architecture. For example, the process of software construction can be compared to the dialectical ascent described in the Allegory of the Cave in Plato's The Republic, Book VII.

In the Allegory of the Cave, according to [Her09], Plato describes fictitious prisoners who have lived their entire lives in the depths of a cave. The prisoners are chained so they can look only forward. Behind them there is a road over which individuals pass, carrying a variety of *objects*. Behind the road a fire is blazing, causing a projection of *shadows* of the travelers and the objects onto the wall in front of the prisoners. For the prisoners, the projected shadows constitute reality. Plato then describes what might happen if one of the prisoners were to leave the cave. Turning toward the fire would cause damage to his eyes. Otherwise the prisoner would adjust to the flames and finally see the objects. Only after a *period of adjustment* could he see things in this world and recognize that they were more real than the shadows that he had experienced until now in the cave.

The software engineering activity looks for solutions of computational problems and finds them by means of a process which reminds the dialectical ascent. We may think that the *shadows* are specifications, that the *period of adjustment* is a software design process and finally that the *objects* are implementations aimed to capture the entire computational problem.

This description brings us to the philosophical kernel of our investigation. When we ask a mathematician what he thinks about combining software engineering terminology and mathematical thought, the answer could be that there is already enough trouble with mathematics alone. However, mathematics becomes easy when the things we are doing are well motivated. The problem is on another level, since the task is to ensure that our model captures the real world which we wish to reflect by the software engineering approach.

According to [BM75] computational complexity analysis requires a computation model. It is common sense that a model which captures accurately the reality is in better condi-

tions to predict and to draw conclusions. We think that we come closer to the reality when we introduce into our computation model quality attributes and software architectures.

This argumentation is presented here under "conclusions". But in fact this was the very staring point of this thesis. Platonian philosophy has guided this study during all the time. Of course this is not the first time that Plato's viewpoint influenced developments in Software Engineering. "Ontologies" are another example for that (see e.g. [Gru93]).

However, we do not say that we have to study all Plato's works, there are specific issues to treat previously, for example, software reuse and libraries which are standard ingredients of the life of a programmer. Classes are elements of reuse and the theory presented in the present thesis captures the notion of a class. In this sense, we shall treat in future work reuse and combination with optimal libraries.

A Appendix : Geometrical complement

A.1 The geometrically robust closure of an irreducible affine variety

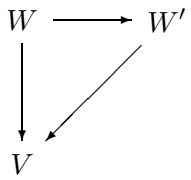
In this section we restrict our attention to affine varieties over $\mathbb{C}$. However, our main results are still valid for arbitrary algebraically closed fields.

Let V be an affine variety with coordinate ring $\mathbb{C}[V]$ and total quotient ring of rational functions $\mathbb{C}(V)$.

Then the set $A := \{\varphi : V \to \mathbb{A}^1 \ ; \ \varphi$ geometrically robust and constructible$\}$ forms a $\mathbb{C}$–subalgebra of $\mathbb{C}(V)$. Moreover A contains $\mathbb{C}[V]$ and is by Definition 6 in Section 2.3.2 a finite $\mathbb{C}[V]$–submodule of $\mathbb{C}(V)$.

Therefore A is a commutative ring which is finitely generated over $\mathbb{C}$. Hence there exists an irreducible affine variety W and a finite surjective morphism $W \to V$ such that A is isomorphic to the coordinate ring $\mathbb{C}[W]$ of W.

We call $W \to V$ a *geometrically robust closure* of V. For two geometrically robust closures $W \to V$ and $W' \to V$ there exists a unique isomorphism of affine varieties $W \to W'$ such that the diagram



commutes. In the following we fix a geometrically robust closure $\mu : W \to V$ of V.

If μ is an isomorphism we say that V is *closed* with respect to *geometrical robustness*. This means that any geometrically robust constructible function $V \to \mathbb{A}^1$ belongs already to $\mathbb{C}[V]$. Observe that normal varieties are always closed with respect to geometrical robustness.

Lemma 23 *Let notations and assumptions be as above. Then W and μ satisfy the following conditions.*

(i) μ is a finite, bijective (and hence birational) morphism of affine varieties.

(ii) μ is a homeomorphism with respect to the Zariski and strong topologies of W and V.

(iii) W is closed with respect to geometrical robustness.

Proof.

Without loss of generality we may assume that $\mathbb{C}[W]$ is contained in $\mathbb{C}(V)$. We show first statement (i).

Since $\mathbb{C}[W]$ is a finite $\mathbb{C}[V]$–module, the morphism $\mu : W \to V$ is finite and surjective. We prove that μ is also injective. Let w and w' be points of W with

$$v := \mu(w) = \mu(w').$$

Suppose $w \neq w'$. Then there exists an element φ of $\mathbb{C}[W]$, namely a geometrically robust constructible function $V \to \mathbb{A}^1$, with $\varphi(w) = 0$ and $\varphi(w') \neq 0$.

Let $\mathfrak{M}_v$ be the vanishing ideal of $\mathbb{C}[V]$ at the point v. Then $\mathbb{C}[V]_{\mathfrak{M}_v}$ is the local ring of V at v. Since φ is constructible there exist by Lemma 1 in Section 2.1.3 a Zariski open and dense subset U of V such that $\varphi|_U$ is rational. We may therefore interpret φ as a rational map on U. According to condition (ii) of Definition 6 in Section 2.3.2 the $\mathbb{C}$–algebra $\mathbb{C}[V]_{\mathfrak{M}_v}[\varphi]$ is local and its maximal ideal is generated by $\mathfrak{M}_v$ and $\varphi - \varphi(v)$. On the other hand $\mathbb{C}[V]_{\mathfrak{M}_v}[\varphi]$ is contained in $A_{\mathfrak{M}_v} = \mathbb{C}[W]_{\mathfrak{M}_v}$ and $\mathbb{C}[W]_{\mathfrak{M}_v}$ is finite module over $\mathbb{C}[V]_{\mathfrak{M}_v}$ and hence also over $\mathbb{C}[V]_{\mathfrak{M}_v}[\varphi]$. This implies that the maximal ideal of $\mathbb{C}[V]_{\mathfrak{M}_v}[\varphi]$ generates in $\mathbb{C}[W]_{\mathfrak{M}_v}$ a proper ideal which is contained in any maximal ideal of $\mathbb{C}[W]_{\mathfrak{M}_v}$ and in particular in the extensions to $\mathbb{C}[W]_{\mathfrak{M}_v}$ of the vanishing ideals of $\mathbb{C}[W]$ at the points w and w'.

This implies $\varphi(v) = \varphi(w) = \varphi(w')$, in contradiction to $\varphi(w) = 0$ and $\varphi(w') \neq 0$.

Therefore we have $w' = w$. This means that μ is injective and hence bijective.

We are now going to show statement (ii). Since $\mu : W \to V$ is a finite, bijective morphism of affine varieties, it is continuous and closed with respect to the Zariski topologies of W and V. Therefore μ is a homeomorphism with respect to the Zariski topologies of W and V.

Let us now consider the strong topologies of W and V. Let D be a closed subset of W and let v be a point of the closure of $\mu(D)$ in V.

Then there exists a sequence $(w_k)_{k\in\mathbb{N}}$ of points of D such that $(\mu(w_k))_{k\in\mathbb{N}}$ converges to v. Since the morphism μ is finite and $(\mu(w_k))_{k\in\mathbb{N}}$ is bounded, we conclude that $(w_k)_{k\in\mathbb{N}}$ is also bounded. Therefore, we may suppose without loss of generality that $(w_k)_{k\in\mathbb{N}}$ converges to a point w of W. Taking into account that all entries of the sequence $(w_k)_{k\in\mathbb{N}}$ belong to D and D is closed, we conclude $w \in D$.

As a morphism of affine varieties μ is continuous with respect to the strong topologies of W and V. This implies $v = \mu(w)$ and therefore v belongs to $\mu(D)$. Hence we conclude that μ is closed with respect to the strong topologies of W and V and consequently a homeomorphism.

Finally we show statement (iii). Let $\varphi : W \to A^1$ be an arbitrary geometrically robust constructible function of W. Interpreted as an element of $\mathbb{C}(V)$, the map φ becomes easily boiled down to a geometrically robust constructible function $V \to A^1$. This implies $\varphi \in \mathbb{C}[W]$ and therefore W is closed with respect to geometrical robustness. ■

We consider now V as a topological space equipped with its Zariski topology.

For a non-empty Zariski open subset $\mathcal{U}$ of V let

$$\Gamma_V(\mathcal{U}) := \{\varphi : \mathcal{U} \to \mathbb{A}^1 \ ; \ \varphi \text{ is geometrically robust and constructible}\}.$$

From Proposition 8 and Theorem 9 of Section 2.3.2 we deduce immediately that the map Γ_V, which associates to every non–empty Zarisky open subset $\mathcal{U}$ of V the $\mathbb{C}$–algebra $\Gamma_V(\mathcal{U})$, is a sheaf. Moreover $\Gamma_V(V)$ is isomorphic to $\mathbb{C}[W]$.

Lemma 24 *Let $f \in \mathbb{C}[V]$ and $f \neq 0$. Then $\Gamma_V(V_f)$ is canonically isomorphic to $\mathbb{C}[W]_f$*

Proof. Observe that V_f is a non-empty Zariski open subset of V and that any element of $\mathbb{C}[W]$ may be identified with a geometrically robust constructible function $V \to \mathbb{A}^1$. From Proposition 8 of Section 2.3.2 we deduce that the restriction of such a function to V_f belongs to $\Gamma_V(V_f)$. In this way we obtain a $\mathbb{C}$-algebra homomorphisms $\mathbb{C}[W] \to \Gamma_V(V_f)$ which can be extended to $\mathbb{C}[W]_f \to \Gamma_V(V_f)$. Suppose for the moment that f is not a zero–divisor of $\mathbb{C}[V]$. Under this assumption we show first that these homomorphisms are injective. It suffices to prove the injectivity of $\mathbb{C}[W] \to \Gamma_V(V_f)$. Let be given a geometrically robust constructible function $\varphi : V \to \mathbb{A}^1$ with $\varphi|_{V_f} = 0$. By Lemma 1 in Section 2.1.3 there exists a non–empty Zariski open and dense subset $\mathcal{U}$ of V such that $\varphi|_{\mathcal{U}}$ is a rational function which is regular on $\mathcal{U}$. Since by assumption f is not a zero–divisor of $\mathbb{C}[V]$, the Zariski open set V_f is also dense in V. Interpreting φ as an element of $\mathbb{C}(V)$ we deduce from $\mathcal{U} \cap V_f \neq \emptyset$ and $\varphi|_{\mathcal{U}\cap V_f} = 0$ that $\varphi = 0$ holds. We have to show that φ is identically zero as map $\varphi : W \to \mathbb{C}$. For this purpose let $w \in W$ be an arbitrary point. Since by Lemma 23 (ii) the map $\mu : W \to V$ is a homeomorphism with respect to the strong topologies of W and V, the set $\mu^{-1}(\mathcal{U})$ is dense in W. Therefore there exists a sequence $(v_k)_{k\in\mathbb{N}}$, $v_k \in \mathcal{U}$ such that $(\mu^{-1}(v_k))_{k\in\mathbb{N}}$ converges to w.

For any $k \in \mathbb{N}$ we have $\varphi(\mu^{-1}(v_k)) = \varphi(v_k) = 0$ and therefore, by the strong continuity of μ^{-1} and φ, also $\varphi(w) = 0$.

This implies that the $\mathbb{C}$-algebra homomorphism $\mathbb{C}[W] \to \Gamma_V(V_f)$ is injective.

Under the assumption that f is not a zero–divisor of $\mathbb{C}[V]$ we are now going to show that $\mathbb{C}[W]_f \to \Gamma_V(V_f)$ is surjective. Let $\varphi : V_f \to \mathbb{A}^1$ be a geometrically robust constructible function. Interpreting φ as an element of $\mathbb{C}(V)$ we deduce from Definition 6 in Section 2.3.2 that φ satisfies in $\mathbb{C}(V_f)$ an integral dependence equation over $\mathbb{C}[V]_f$. Therefore there exists a nonnegative integer r such that $f^r\varphi$ satisfies in $\mathbb{C}(V)$ also an integral dependence equation over $\mathbb{C}[V]$.

We consider now the function $\psi : V \to \mathbb{A}^1$ defined for $v \in V$ by $\psi(v) := (f^r\varphi)(v)$ in case $f(v) \neq 0$ and by $\psi(v) := 0$ in case $f(v) = 0$. We are going to show that ψ is a geometrically robust constructible function. Constructibility is clear by the definition of ψ. Let v be an arbitrary point of V. Since V_f is Zariski open and dense in V and $f^r\varphi$ as rational function of V satisfies in $\mathbb{C}(V)$ an integral dependence relation over $\mathbb{C}[V]$, one sees easily that ψ fulfills condition (i) of Definition 6 at the point v. In the same way one verifies condition (ii) of Definition 6 for v belonging to V_f. Therefore let us suppose $f(v) = 0$. With the previous notations this means $f \in \mathfrak{M}_v$.

By Lemma 1 in Section 2.1.3 we may interpret φ as a rational function of V. Observe that $\mathbb{C}[V]_{\mathfrak{M}_v}[\varphi]$ contains $\mathbb{C}[V]_{\mathfrak{M}_v}[\psi]$ and is a finite module over $\mathbb{C}[V]_{\mathfrak{M}_v}$ and therefore also over $\mathbb{C}[V]_{\mathfrak{M}_v}[\psi]$. Therefore any maximal ideal of $\mathbb{C}[V]_{\mathfrak{M}_v}[\psi]$ generates in $\mathbb{C}[V]_{\mathfrak{M}_v}[\varphi]$ a proper ideal. From $f \in \mathfrak{M}_v$ we deduce now that any maximal ideal of $\mathbb{C}[V]_{\mathfrak{M}_v}[\varphi]$ contains $f^r\varphi$. This implies that any maximal ideal of $\mathbb{C}[V]_{\mathfrak{M}_v}[\psi]$ contains ψ.

Finally we conclude that $\mathbb{C}[V]_{\mathfrak{M}_v}[\psi]$ is local and that its maximal ideal is generated by $\mathfrak{M}_v$ and ψ. Therefore condition (ii) of Definition 6 is satisfied at v, also in the case $f(v) = 0$. This implies that ψ is geometrically robust.

The constructible map ψ can be restricted to a rational function of V, which we denote also by ψ. In this sense we have the identity $\psi = f^r\varphi$ in $\mathbb{C}(V)$.

The geometrically robust constructible function ψ may be interpreted as an element

of $\mathbb{C}[W]$ and $\frac{\psi}{f^r}$ as an element of $\mathbb{C}[W]_f$. One sees now easily that $\mathbb{C}[W]_f \to \Gamma_V(V_f)$ maps $\frac{\psi}{f^r} \in \mathbb{C}[W]_f$ on $\varphi \in \Gamma_V(V_f)$. Thus $\mathbb{C}[W]_f \to \Gamma_V(V_f)$ is surjective and hence a $\mathbb{C}$-algebra isomorphism.

Suppose now that f is a zero–divisor of $\mathbb{C}[V]$ with $f \neq 0$.

Let V^* be the union of the irreducible components of V where f does not vanish identically, $W^* := \mu^{-1}(V^*)$, $\mu^* := \mu|_{W^*}$ and $f^* := f|_{V^*}$. One verifies easily that f^* is not a zero–divisor of $\mathbb{C}[V^*]$ and that $\mu^* : W^* \to V^*$ is a geometrically robust closure of V^*.

From our previous argumentation we obtain a canonical isomorphism $\mathbb{C}[W^*]_{f^*} \to \Gamma_{V^*}(V^*_{f^*})$. Taking into account $\mathbb{C}[W^*]_{f^*} \cong \mathbb{C}[W^*_{f^*}] = \mathbb{C}[W_f] \cong \mathbb{C}[W]_f$ and $\Gamma_{V^*}(V^*_{f^*}) = \Gamma_{V^*}(V_f) = \Gamma_V(V_f)$ we see that $\Gamma_V(V_f)$ and $\mathbb{C}[W]_f$ are canonically isomorphic. ■

Theorem 25 *Let notations and assumptions be as above. The affine variety W with its canonical sheaf and (V, Γ_V) are isomorphic as ringed spaces. The isomorphism is induced by $\mu : W \to V$.*

Proof.

From Lemma 23 (*ii*), we deduce that $\mu : W \to V$ is an homeomorphism with respect to their Zariski topologies of W and V.

Therefore the sets W_f, $f \in \mathbb{C}[V]$, form a basis of the topology of W. The canonical sheaf of W associates to W_f, $f \in \mathbb{C}[V]$ the $\mathbb{C}$-algebra $\mathbb{C}[W]_f$, which following Lemma 24 is isomorphic to $\Gamma_V(V_f)$, while this isomorphism is induced by μ. Thus we conclude that for any point $w \in W$ the stalks at w and $\mu(w)$ of the ringed spaces W and (V, Γ_V) are isomorphic. This implies that μ induces a sheaf isomorphism between W and (V, Γ_V). ■

Let $\mu : W \to V$ and $\mu' : W' \to V'$ be two geometrically robust closures of two irreducible affine varieties and let $\varphi : V' \to V$ be *dominating* morphism of affine varieties. One sees easily that there exists a (unique) morphism of affine varieties $\psi : W' \to W$ such that the diagram

$$\begin{array}{ccc} W' & \xrightarrow{\psi} & W \\ {\scriptstyle \mu'}\downarrow & & \downarrow{\scriptstyle \mu} \\ V' & \xrightarrow{\varphi} & V \end{array}$$

commutes.

In particular, if V' is closed with respect to geometrical robustness, there exists a (unique) morphism of affine varieties $\psi : V' \to W$ such that the diagram

$$\begin{array}{ccc} & & W \\ & {\scriptstyle \psi}\nearrow & \downarrow{\scriptstyle \mu} \\ V' & \xrightarrow{\varphi} & V \end{array}$$

commutes.

Suppose finally that there is given an irreducible closed subvariety V' of V. Let $\mu : W \to V$ and $\mu' : W' \to V'$ be geometrically robust closures of V and V'.

Then Theorem 25 and Proposition 8 in Section 2.3.2 imply that there exists a (unique) morphism of affine varieties $\psi : W' \to W$ such that the diagram

$$\begin{array}{ccc} W' & \xrightarrow{\psi} & W \\ {\scriptstyle \mu'}\downarrow & & \downarrow{\scriptstyle \mu} \\ V' & \subset & V \end{array}$$

commutes.

The inclusion of V' in V induces a surjective $\mathbb{C}$-algebra homomorphism $\mathbb{C}[V] \to \mathbb{C}[V']$. We may before ask whether $\psi : W' \to W$ induces also a surjective $\mathbb{C}$-algebra homomorphism $\mathbb{C}[W] \to \mathbb{C}[W']$.

That this is not the case can easily be seen by analyzing the following particular situation.

Example [GHMS11] Let X_1, X_2 be indeterminates over $\mathbb{C}$ and consider the plane curve $\mathcal{C}$ defined by the equation

$$X_1^3 - X_2^2 = 0$$

Observe that $\mathcal{C}$ is an irreducible curve which can be parametrized by the surjective polynomial map $\rho : \mathbb{A}^1 \to \mathcal{C}$ defined for $t \in \mathbb{A}^1$ by $\rho(t) = (t^2, t^3)$. Let ξ_1, ξ_2 be the coordinate functions of $\mathcal{C}$ induced by X_1 and X_2. Then we have $\mathbb{C}[\mathcal{C}] = \mathbb{C}[\xi_1, \xi_2]$. Consider the constructible map $\varphi : \mathcal{C} \to \mathbb{A}^1$ defined for $d := (d_1, d_2) \in \mathcal{C}$, $d \neq (0,0)$ by $\varphi(d) := \frac{d_2}{d_1}$ and for $d := (0,0)$ by $\varphi(0) := 0$.

Then the restriction of φ to $\mathcal{C} - \{(0,0)\}$ (also denoted by φ) is rational and in $\mathbb{C}(\mathcal{C})$ we have the identities $\varphi = \frac{\xi_2}{\xi_1}$ and $(\frac{\xi_2}{\xi_1})^2 - \xi_1 = 0$. From these identities one infers easily that φ is geometrically robust.

Suppose now that there exists a geometrically robust constructible function $\widetilde{\varphi} : \mathbb{A}^2 \to \mathbb{A}^1$ with $\widetilde{\varphi}|_{\mathcal{C}} = \varphi$.

Since the affine space $\mathbb{A}^2$ is normal, the map $\widetilde{\varphi}$ is polynomial. Therefore there exists a polynomial $F \in \mathbb{C}[X_1, X_2]$ with $F(d_1, d_2) = \widetilde{\varphi}(d)$ for any point $d := (d_1, d_2)$ of $\mathbb{A}^2$. Hence $\widetilde{\varphi}|_{\mathcal{C}} = \varphi$ implies $F(\xi_1, \xi_2) = \frac{\xi_2}{\xi_1}$ in $\mathbb{C}(\mathcal{C})$ or equivalently $\xi_1 F(\xi_1, \xi_2) = \xi_2$ in $\mathbb{C}[\mathcal{C}]$.

Since the curve $\mathcal{C}$ becomes parametrized by $\rho : \mathbb{A}^1 \to \mathcal{C}$ we conclude that the identity $t^2 F(t^2, t^3) = t^3$ holds for any $t \in \mathbb{A}^1$.

Let T be a new indeterminate. Then we have $T^2 F(T^2, T^3) = T^3$ and therefore $F(T^2, T^3) = T$, which is impossible. Thus there exist no geometrically robust constructible function $\widetilde{\varphi} : \mathbb{A}^2 \to \mathbb{A}^1$ with $\widetilde{\varphi}|_{\mathcal{C}} = \varphi$. Hence for a geometrically robust closure $\mu : W \to \mathcal{C}$ of $\mathcal{C}$ the canonical $\mathbb{C}$-algebra homomorphism $\mathbb{C}[X_1, X_2] \to \mathbb{C}[W]$ is not surjective.

Corollary 26 *Let notations and assumptions be as in Theorem 25 and let $\varphi, \varphi_1, \ldots, \varphi_s$ be geometrically robust constructible functions of V. Then the following statements are true.*

(i) $\varphi \neq 0$ implies that $V_\varphi := \{v \in V \ ; \ \varphi(v) \neq 0\}$ together with the restriction of the sheaf Γ_V to V_φ is an affine variety.

(ii) If the set $\{v \in V \ ; \ \varphi_1(v) = \cdots = \varphi_s(v) = 0\}$ is empty, then there exists geometrically robust constructible functions $\alpha_1, \ldots, \alpha_s$ of V which satisfy the condition $1 = \alpha_1\varphi_1 + \cdots + \alpha_s\varphi_s$

(iii) If φ vanishes at any point $v \in V$ with $\varphi_1(v) = \cdots = \varphi_s(v) = 0$, then there exists a natural number r and geometrically robust constructible functions $\alpha_1, \ldots, \alpha_s$ of V which satisfy the condition $\varphi^r = \alpha_1\varphi_1 + \cdots + \alpha_s\varphi_s$

B Appendix : Correctness proofs

B.1 Correctness of Lagrange interpolation algorithms

B.1.1 Lagrange Form

Let K be a given natural number. Following Section 4.5.1 let $(\mathcal{O},\ \Phi)$ be the *univariate Lagrange interpolation problem at fixed nodes* $\alpha_1, \ldots, \alpha_K$ with $\alpha_i \neq \alpha_j$ for $1 \leq i < j \leq K$, such that for a given interpolation datum $d := ((\alpha_1, y_1), \ldots, (\alpha_n, y_n))$ of $\mathbb{A}^{2K}$ the corresponding interpolant is the polynomial $\Phi(d)$ defined by the Lagrange Form

$$\Phi(d) := y_1 L_1(X) + \cdots + y_K L_K(X)$$

with

$$L_i(X) := \prod_{\substack{k:=1 \\ k \neq i}}^{K} \frac{X - \alpha_k}{\alpha_i - \alpha_k},\ 1 \leq i \leq K. \tag{22}$$

Observe that for $1 \leq i, k \leq K$ the value $L_i(\alpha_k)$ satisfies the following condition:

$$L_i(\alpha_k) = \begin{cases} 1 & \text{if } i = k \\ 0 & \text{otherwise.} \end{cases} \tag{23}$$

Furthermore let Ψ be the routine defined for any interpolation datum $d := ((\alpha_1, y_1), \ldots, (\alpha_K, y_K))$ with $y := (y_1, \ldots, y_K)$ by

$$\Psi(d) := y.$$

Let ω^* be the abstraction function defined for $y := (y_1, \ldots, y_K) \in \mathbb{A}^K$ by

$$\omega^*(y) := \sum_{i:=1}^{K} y_i L_i(X).$$

Proposition 27 *Let assumptions and notations be as before. Then we have for any interpolation datum* $d := ((\alpha_1, y_1), \ldots, (\alpha_K, y_K)) \in \mathbb{A}^{2K}$ *that* $\omega^*(\Psi(d)) = \Phi(d)$ *holds. In other words, the polynomial* $\omega^*(\Psi(d))$ *satisfies for any* $1 \leq i \leq K$ *the interpolation condition*

$$\omega^*(\Psi(d))(\alpha_i) = y_i. \tag{24}$$

Proof. The identity (24) is an immediate consequence of (23). ■

B.1.2 Monomial Form

Let K be a given natural number. Following Section 4.5.1 let $(\mathcal{O},\ \Phi)$ be the *univariate Lagrange interpolation problem at fixed nodes* $\alpha_1, \ldots, \alpha_K$ such that for a given interpolation datum $d := ((\alpha_1, y_1), \ldots, (\alpha_K, y_K))$ the corresponding interpolant $\Phi(d)$ is the (unique) polynomial with Monomial Form,

$$\Phi(d) := a_0 + a_1 X + \cdots + a_{K-1} X^{K-1}$$

which satisfies for any $1 \leq i \leq K$ the interpolation condition.

$$\Phi(d)(\alpha_i) = y_i.$$

Let Ψ be the routine defined for $d := ((\alpha_1, y_1), \ldots, (\alpha_K, y_K))$ with $y := (y_1, \ldots, y_K)$ by

$$\Psi(d) := V_\alpha^{-1} y^t,$$

where V_α is the Vandermonde matrix

$$V_\alpha := \begin{vmatrix} 1 & \alpha_1 & \ldots & \alpha_1^{K-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha_K & \ldots & \alpha_K^{K-1} \end{vmatrix}.$$

Finally let ω^* be the abstraction function defined for $(a_0, \ldots, a_{K-1}) \in \mathbb{A}^K$ by

$$\omega^*((a_0, \ldots, a_{K-1})) := \sum_{i:=0}^{K-1} a_i X^i.$$

Proposition 28 *Let assumptions and notations be as before. Then we have for any interpolation datum $d := ((\alpha_1, y_1), \ldots, (\alpha_K, y_K)) \in \mathbb{A}^{2K}$ that $\omega^*(\Psi(d)) = \Phi(d)$ holds. In other words, the polynomial $\omega^*(\Psi(d))$ satisfies for any $1 \leq i \leq K$ the interpolation condition*

$$\omega^*(\Psi(d))(\alpha_i) = y_i. \tag{25}$$

Proof. In order to prove that the interpolant $\omega^*(\Psi(d))$ satisfies the interpolation condition (25), we are going to show that the following inhomogeneous linear equation system in the unknowns $a_0, \ldots, a_{K-1}$ has a unique solution.

$$\begin{aligned} a_0 + a_1\alpha_1 + \cdots + a_{K-1}\alpha_0^{K-1} &= y_1 \\ &\vdots \\ a_0 + a_1\alpha_K + \cdots + a_{K-1}\alpha_K^{K-1} &= y_K \end{aligned}$$

Putting $a := (a_0, \ldots, a_{K-1})$ and $y := (y_1, \ldots, y_K)$ we may rewrite this equation system in the following matricial form:

$$V_\alpha a^t := y^t \tag{26}$$

(here a^t and y^t denote the transpositions of a and y).

We prove that for any given $y \in \mathbb{A}^K$ the system (26) has unique solution $a \in \mathbb{A}^K$ by showing that $\det(V_\alpha) \neq 0$ holds. Lemma 29 below implies:

$$\det(V_\alpha) := \textstyle\prod_{1 \leq i < j \leq K} (\alpha_j - \alpha_i).$$

Since $\alpha_i \neq \alpha_j$ holds for any $1 \leq i < j \leq K$, we conclude $\det(V_\alpha) \neq 0$.

■

Lemma 29

$$\det(V_\alpha) = \det \begin{pmatrix} 1 & \alpha_1 & \dots & \alpha_1^{K-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha_K & \dots & \alpha_K^{K-1} \end{pmatrix} = \prod_{1 \le i < j \le K} (\alpha_j - \alpha_i) \tag{27}$$

Proof. Operating in the matrix V_α with column number $K-1$ on column number K, then with column number $K-2$ on column number $K-1$, etc. we obtain finally the identity

$$\det \begin{pmatrix} 1 & 0 & \dots & 0 \\ 1 & \alpha_2 - \alpha_1 & \dots & \alpha_2^{K-1} - \alpha_1 \alpha_2^{K-2} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha_n - \alpha_1 & \dots & \alpha_K^{K-1} - \alpha_1 \alpha_K^{K-2} \end{pmatrix} := \prod_{i:=2}^{K} (\alpha_i - \alpha_1) \det \begin{pmatrix} 1 & \alpha_2 & \dots & \alpha_2^{K-2} \\ \vdots & \ddots & \vdots & \\ 1 & \alpha_K & \dots & \alpha_K^{K-2} \end{pmatrix}.$$

Lemma 29 follows from this identity iductively. ■

B.1.3 Newton Form

Let $(\mathcal{O},\ \Phi)$ be the *univariate Lagrange interpolation problem at fixed nodes* $\alpha_1, \dots, \alpha_K$ such that for a given interpolation datum $d := ((\alpha_1, y_1), \dots, (\alpha_K, y_K))$ the corresponding interpolant $\Phi(d)$ the unique polynomial in Newton Form,

$$\Phi(d) = a_0 + a_1(X - \alpha_1) + \cdots + a_K(X - \alpha_1)...(X - \alpha_K)$$

which satisfies for $1 \le i \le K$ the interpolation condition $\Phi(d)(\alpha_i) = y_i$. Let Ψ be the routine defined for $d := ((\alpha_1, y_1), \dots, (\alpha_K, y_K))$ by

$$\Psi(d) := (f[\alpha_1], \dots, f[\alpha_1, \dots, \alpha_K])$$

where for $1 \le i \le K$ and $1 \le k \le K - i$ the function f is defined by the following recursion formulas, called *divided differences*:

$$\begin{aligned} f[\alpha_i] &:= y_i \\ f[\alpha_i, \dots, \alpha_{i+k}] &:= \frac{f[\alpha_{i+1}, \dots, \alpha_{i+k}] - f[\alpha_i, \dots, \alpha_{i+k-1}]}{\alpha_{i+k} - \alpha_i}. \end{aligned} \tag{28}$$

Let ω^* be the Abstraction Function which for $(a_0, \dots, a_{K-1}) \in \mathbb{A}^K$ is defined as follows:

$$\omega^*((a_0, \dots, a_{K-1})) := \sum_{i:=0}^{K-1} a_i \prod_{j:=1}^{i-1} (X - \alpha_j)$$

Proposition 30 *Let assumptions and notations be as before. Then we have for any interpolation datum* $d := ((\alpha_1, y_1), \dots, (\alpha_K, y_K))$ *that* $\omega^*(\Psi(d)) = \Phi(d)$ *holds. In other words, the polynomial* $\omega^*(\Psi(d))$ *satisfies for any* $1 \le i \le K$ *the interpolation condition*

$$\omega^*(\Psi(d))(\alpha_i) = y_i. \tag{29}$$

Proof. Let $(a_0, \ldots, a_{K-1})$ be a point of $\mathbb{A}^K$ and consider the polynomial

$$p(X) := a_0 + a_1(X - \alpha_1) + \cdots + a_{K-1}(X - \alpha_1)...(X - \alpha_{K-1})$$

Let $y_1 := p(\alpha_1), \ldots, y_K := p(\alpha_K)$. Then we obtain the following matricial identity:

$$\underbrace{\begin{pmatrix} 1 & 0 & 0 & \ldots & 0 \\ 1 & (\alpha_2 - \alpha_1) & 0 & & \vdots \\ 1 & (\alpha_3 - \alpha_1) & (\alpha_3 - \alpha_1)(\alpha_3 - \alpha_2) & & \\ \vdots & & & \ddots & \\ 1 & (\alpha_K - \alpha_1) & (\alpha_K - \alpha_1)(\alpha_K - \alpha_2) & \ldots & \prod_{1 \le i \le K-1}(\alpha_K - \alpha_i) \end{pmatrix}}_{A:=} \underbrace{\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{K-1} \end{pmatrix}}_{a:=} := \underbrace{\begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_K \end{pmatrix}}_{y:=} \tag{30}$$

Note that A is a triangular matrix. Since $\alpha_i \neq \alpha_j$ holds for $1 \le i < j \le K$, we conclude $\det A \neq 0$. Therefore (30), interpreted as an inhomogeneus linear equation system in the unknowns $a_0, \ldots, a_{K-1}$, has a unique solution. Let be given such a solution $(a_0, \ldots, a_{K-1}) \in \mathbb{A}^K$.

Claim 31 *For any $0 \le i \le K-1$ we have $a_i = f[\alpha_1, \ldots, \alpha_{i+1}]$.*

In order to prove this claim we consider for $1 \le j \le K$ the polynomial $P_j(X)$ which is recursively defined as follows:

$$P_1(X) := f[\alpha_1]$$

$$P_j(X) := P_{j-1}(X) + f[\alpha_1, \ldots, \alpha_j](X - \alpha_1) \ldots (X - \alpha_{j-1}),\ 1 \le j \le K.$$

This proves Proposition 30 . ■

Proof of Claim 31. We proceed by induction in $0 \le i \le K-1$, assuming Lemma 32 and 33 below.

The case $i := 0$ is easy and follows from $a_0 = y_1 = f[\alpha_1]$.

Let $0 < i \le K-1$ and assume that the claim is true for $i-1$. Then we deduce from the $i+1$–th row of (30), the induction hypothesis, the definition of $P_i(X)$ and Lemma 32 and 33 the following identities:

$$\begin{aligned} a_i &= \frac{y_{i+1} - (a_0 + ... + a_{i-1}(\alpha_{i+1} - \alpha_1)...(\alpha_{i+1} - \alpha_{i-1}))}{(\alpha_{i+1} - \alpha_1)...(\alpha_{i+1} - \alpha_i)} = \\ &\frac{y_{i+1} - (f[\alpha_1] + ... + f[\alpha_1, \ldots, \alpha_i](\alpha_{i+1} - \alpha_1)...(\alpha_{i+1} - \alpha_{i-1}))}{(\alpha_{i+1} - \alpha_1)...(\alpha_{i+1} - \alpha_i)} = \\ &\frac{y_{i+1} - P_i(\alpha_{i+1})}{(\alpha_{i+1} - \alpha_1)...(\alpha_{i+1} - \alpha_i)} = f[\alpha_i, \ldots, \alpha_1, \alpha_{i+1}] = f[\alpha_1, \ldots, \alpha_{i+1}]. \end{aligned}$$

This proves the claim. ■

Lemma 32 *Let notations and assumptions be as before. Then we have for $1 \le k < j \le K$ the identity*

$$\frac{y_j - P_{j-k}(\alpha_j)}{(\alpha_j - \alpha_1)...(\alpha_j - \alpha_{j-k})} = f(\alpha_{j-k}, \ldots, \alpha_1, \alpha_j)$$

Proof. Let $1 < j \leq K$ be fixed. We proceed by induction in $1 \leq k < j$, assuming Lemma 9.

In case $k := j - 1$ we have

$$\frac{y_j - P_{j-k}(\alpha_j)}{(\alpha_j - \alpha_1)...(\alpha_j - \alpha_{j-k})} = \frac{y_j - P_1(\alpha_j)}{(\alpha_j - \alpha_1)} = \frac{f[\alpha_j] - f[\alpha_1]}{(\alpha_j - \alpha_1)} = f[\alpha_1, \alpha_j].$$

Let $1 \leq k < j-1$ and assume Lemma 32 for $k+1$. Then we deduce from the definition of P_{j-k}, the inductive hypothesis and Lemma 33 the identities:

$$\frac{y_j - P_{j-k}(\alpha_j)}{(\alpha_j - \alpha_1)...(\alpha_j - \alpha_{j-k})} =$$

$$\frac{y_j - P_{j-k-1}(\alpha_j) - f[\alpha_1, \ldots, \alpha_{j-k}](\alpha_j - \alpha_1)\ldots(\alpha_j - \alpha_{j-k-1})}{(\alpha_j - \alpha_1)\ldots(\alpha_j - \alpha_{j-k})} =$$

$$\frac{\frac{y_j - P_{j-k-1}(\alpha_j)}{(\alpha_j - \alpha_1)...(\alpha_j - \alpha_{j-k-1})} - f[\alpha_1, \ldots, \alpha_{j-k}]}{\alpha_j - \alpha_{j-k}} =$$

$$\frac{f[\alpha_{j-k-1}, \ldots, \alpha_1, \alpha_j] - f[\alpha_1, \ldots, \alpha_{j-k}]}{\alpha_j - \alpha_{j-k}} =$$

$$\frac{f[\alpha_{j-k-1}, \ldots, \alpha_1, \alpha_j] - f[\alpha_{j-k}, \ldots, \alpha_1]}{\alpha_j - \alpha_{j-k}} = f[\alpha_{j-k}, \ldots, \alpha_1, \alpha_j].$$

■

Lemma 33 *The divided difference function f is symmetric.*

Proof. With the notations introduced before it suffices to show the following identity:

$$f[\alpha_1, \ldots, \alpha_k] = \frac{y_1}{(\alpha_1 - \alpha_2)\ldots(\alpha_1 - \alpha_k)} + \cdots + \frac{y_k}{(\alpha_k - \alpha_1)\ldots(\alpha_k - \alpha_{k-1})} \qquad (31)$$

We show (31) by induction in k.

The identity (31) follows in case $k := 1$ from $f(\alpha_1) = y_1$.

Let $k > 1$. We suppose that (31) is valid for $k - 1$.

For $2 \leq i \leq k - 1$ we have the following identities:

$$\frac{1}{\alpha_k - \alpha_1}\left(\frac{y_i}{(\alpha_i - \alpha_2)\ldots(\alpha_i - \alpha_{i-1})(\alpha_i - \alpha_{i+1})\ldots(\alpha_i - \alpha_k)} - \frac{y_i}{(\alpha_i - \alpha_1)\ldots(\alpha_i - \alpha_{i-1})(\alpha_i - \alpha_{i+1})\ldots(\alpha_i - \alpha_{k-1})}\right) =$$

$$\frac{1}{\alpha_k - \alpha_1}\left(\frac{1}{\alpha_i - \alpha_k} - \frac{1}{\alpha_i - \alpha_1}\right)\frac{y_i}{(\alpha_i - \alpha_2)\ldots(\alpha_i - \alpha_{i-1})(\alpha_i - \alpha_{i+1})\ldots(\alpha_i - \alpha_{k-1})} =$$

$$\frac{y_i}{(\alpha_i - \alpha_1)\ldots(\alpha_i - \alpha_{i-1})(\alpha_i - \alpha_{i+1})\ldots(\alpha_i - \alpha_k)}.$$

Taking these identities into account we deduce from our induction hypothesis that

$$f[\alpha_1,\dots,\alpha_k] = \frac{f[\alpha_2,\dots,\alpha_k]-f[\alpha_1,\dots,\alpha_{k-1}]}{\alpha_k-\alpha_1}$$
$$\frac{1}{\alpha_k-\alpha_1}\left(\sum_{2\le i\le k-1}\frac{y_i}{(\alpha_i-\alpha_2)\dots(\alpha_i-\alpha_{i-1})(\alpha_i-\alpha_{i+1})\dots(\alpha_i-\alpha_k)} - \right.$$
$$\sum_{2\le i\le k-2}\frac{y_i}{(\alpha_i-\alpha_1)\dots(\alpha_i-\alpha_{i-1})(\alpha_i-\alpha_{i+1})\dots(\alpha_i-\alpha_{k+1})} +$$
$$\left.\frac{y_k}{(\alpha_k-\alpha_2)\dots(\alpha_k-\alpha_{k-1})}-\frac{y_1}{(\alpha_1-\alpha_2)\dots(\alpha_1-\alpha_{k-1})}\right)$$
$$=\frac{y_1}{(\alpha_1-\alpha_2)\dots(\alpha_1-\alpha_k)}+\dots+\frac{y_k}{(\alpha_k-\alpha_1)\dots(\alpha_k-\alpha_{k-1})}$$

holds. This implies identity (31). ■

References

[Ald84] A. Alder. *Grenzrang und Grenzkomplexität aus algebraischer und topologischer Sicht*. PhD thesis, Universität Zürich, Philosophische Fakultät II, 1984.

[Apt81] K. R. Apt. Ten years of Hoare's logic: A survey–part I. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 3 (4) 431–483, October 1981.

[ASU86] A. V. Aho, R. Sethi, J. D. Ullman. *Compilers: principles, techniques, and tools*. Addison-Wesley Longman Publishing Co., Boston, MA, USA, 1986.

[Bar68] E. H. Bareiss. Sylvester's identity and multistep integer-preserving Gaussian elimination. *Mathematics of Computation*, 22 (103) 565–578, 1968.

[BC97] T. Bloom, J. P. Calvi. A continuity property of multivariate Lagrange interpolation. *Math. Comp.*, 66 (220) 1561–1577, 1997.

[BCK98] L. Bass, P. Clements, R. Kazman. *Software architecture in practice*. Addison-Wesley Longman Publishing Co., Boston, MA, USA, 1998.

[BCK03] L. Bass, P. Clements, R. Kazman. *Software Architecture in Practice*. Addison–Wesley, Boston, MA, 2. edition, 2003.

[BCS97] P. Bürgisser, M. Clausen, M. A. Shokrollahi. *Algebraic Complexity Theory. Grundlehren der mathematischen Wissenschaften*, 315. Springer Verlag, 1997.

[BCSS98] L. Blum, F. Cucker, M. Shub, S. Smale. *Complexity and Real Computation*. Springer–Verlag, 1998.

[BD09] B. Bruegge, A. H. Dutoit. *Object-Oriented Software Engineering: Using UML, Patterns and Java*. Prentice–Hall, Upper Saddle River, NJ, USA, 3. edition, 2009.

[BF91] László Babai, Lance Fortnow. Arithmetization: A new method in structural complexity theory. *Computational Complexity*, 1 41–66, 1991. 10.1007/BF01200057.

[BF02] R. L. Burden, J. D. Faires. *Numerical methods*. Brooks Cole, 3. edition, 2002.

[BKLW95] M. Barbacci, M. H. Klein, T. A. Longstaff, Ch. B. Weinstock. Quality attributes. Technical report, Carnegie Mellon University, 1995.

[Bla09] R. Blanco. Complexity of Villamayor's algorithm in the non-exceptional monomial case. *International Journal of Mathematics*, 20 (6) 659–678, 2009.

[BM75] A. Borodin, I. Munro. *The computational complexity of algebraic and numeric problems*. American Elsevier, New York, 1. edition, 1975.

[BSS89] L. Blum, M. Shub, S. Smale. On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. *Bulletin of the American Mathematical Society*, 1 (21) 1–45, 1989.

[Bud02] T. Budd. *An introduction to object–oriented programming.* Addison–Wesley, 2002.

[BW96] A. Burns, A. J. Wellings. *Real-Time Systems and Programming Languages.* Addison–Wesley, USA, 2. edition, 1996.

[Can88] J. Canny. Some algebraic and geometric computations in PSPACE. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, 460–467. ACM Press, 1988.

[CBB+02] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, J. Stafford. *Documenting Software Architectures: Views and Beyond.* Addison Wesley, September 2002.

[CBK+98] S. J. Carriuere, M. Barbacci, R. Kazman, M. Klein, T. Longsta, H. Lipson. The architecture trade off analysis method. *Proceedings 4th International Conference on Engineering of Complex Computer Systems (ICECCS'98). IEEE Computer Society*, 10 68–78, 14 August, 1998.

[Cer09] S. Ceria. Clase 5: Especificacion de Atributos de Calidad y QAW, 2009.

[CGH89] L. Caniglia, A. Galligo, J. Heintz. Some new effectivity bounds in computational geometry. *Applied Algebra, Algebraic Algorithms and Error Correcting Codes. Proc. of the 6th Intern. Conference AAECC, Best Paper Award AAECC-6. Springer LNCS*, 357 131–151, 1989.

[CGH+03] D. Castro, M. Giusti, J. Heintz, G. Matera, L. M. Pardo. The hardness of polynomial equation solving. *Foundations of Computational Mathematics*, 3 (4) 347–420, 2003.

[Con65] S. D. Conte. *Elementary numerical analysis: an algorithmic approach.* McGraw–Hill, New York, 1965.

[Cor01] T. H. Cormen. *Introduction to Algorithms.* MIT Press, Massachusetts, 2001.

[CYGM09] G. Chèze, J.-C. Yakoubsohn, A. Galligo, B. Mourrain. Computing nearest gcd with certification. In *Proceedings of the 2009 Conference on Symbolic Numeric Computation*, SNC '09, 29–34. ACM Press, 2009.

[Dat09] C. Date. *SQL and Relational Theory: How to Write Accurate SQL Code.* O'Reilly Media, 2009.

[dBR92] C. de Boor, A. Ron. The least solution for the polynomial interpolation problem. *Math. Z.*, 210 (3) 347–378, 1992.

[DFGS91] A. Dickenstein, N. Fitchas, M. Giusti, C. Sessa. The membership problem of unmixed polynomial ideals is solvable in single exponential time. *Discrete Applied Mathematics*, 33 73–94, 1991.

[Dou06] B. P. Douglass. *Real-time UML workshop for embedded systems.* Newnes, Embedded technology series, Electronics and Electrical, Cambridge, MA, USA, 2006.

[Eco06] U. Eco. *Cómo se hace una tesis. Técnicas y procedimientos de estudio, investigación y escritura. Versión castellana de L. Baranda y A. Clavería Ibáñes.* 8. edition, 2006.

[Edm67] J. Edmonds. Systems of distinct representatives and linear algebra. *J. Res. Nat. Bur. Standards*, 71B 241–245, 1967.

[EGL97] I.Z. Emiris, A. Galligo, H. Lombardi. Certified approximate univariate GCD's. *Journal of Pure and Applied Algebra*, 117 229–251, 1997.

[EV00] S. Encinas, O. Villamayor. A course on constructive desingularization and equivariance. *Resolution of singularities: a research textbook in tribute to Oscar Zariski. Progress in Mathematics*, 181 147–227, 2000.

[Ful84] William Fulton. *Intersection Theory.* Number 2 in Ergebnisse der Mathematik und ihre Grenzgebiete. Springer-Verlag, Berlin, 1984. 3. Folge.

[Gau97] W. Gautschi. *Numerical analysis: an introduction.* Springer, 1997.

[GH93] M. Giusti, J. Heintz. La détermination de la dimension et des points isolés dúne variété algébrique peut se faire en temps polynomial. In *Computational Algebraic Geometry and Commutative Algebra (Cortona, 1991)*, 34 of *Symposia Mathematica*, 216–256. Istituto di Alta Matematica Francesco Severi and Cambridge University Press, Cambridge, 1993.

[GH01] M. Giusti, J. Heintz. Kronecker's smart, little black boxes. In *Foundations of Computational Mathematics, R. A. DeVore, A. Iserles, E. Süli eds.*, 284 of *London Mathematical Society Lecture Note Series*, 69–104. Cambridge University Press, Cambridge, 2001.

[GHH+97] M. Giusti, K. Hägele, J. Heintz, J. L. Montaña, J. E. Morais, L. M. Pardo. Lower bounds for diophantine approximation. *Journal of Pure and Applied Algebra*, 117 277–317, 1997.

[GHKa] M. Giusti, J. Heintz, B. Kuijpers. The evaluation of geometric queries: constraint databases and quantifier elimination. Manuscript University of Buenos Aires (2007).

[GHKb] R. Grimson, J. Heintz, B. Kuijpers. Efficient evaluation of specific queries in constraint databases. Manuscript University of Buenos Aires (2011).

[GHM+98] M. Giusti, J. Heintz, J.E. Morais, J. Morgenstern, L.M. Pardo. Straight-line programs in geometric elimination theory. *Journal of Pure and Applied Algebra*, 124 101–146, 1998.

[GHMP97] M. Giusti, J. Heintz, J. E. Morais, L. M. Pardo. Le rôle des structures de données dans les problemes d'élimination. *Comptes Rendus Acad. Sci.*, Serie 1 (325) 1223–1228, 1997.

[GHMS11] N. Giménez, J. Heintz, G. Matera, P. Solernó. Lower complexity bounds for interpolation algorithms. *Journal of Complexity*, 27 151–187, 2011.

[GJM91] C. Ghezzi, M. Jazayeri, D. Mandrioli. *Fundamentals of software engineering.* Prentice-Hall, Upper Saddle River, NJ, USA, 1991.

[GJM03] C. Ghezzi, M. Jazayeri, D. Mandrioli. *Fundamentals of software engineering.* Prentice-Hall, Upper Saddle River, NJ, USA, 2003.

[GLS01] M. Giusti, G. Lecerf, B. Salvy. A Gröbner Free Alternative for Polynomial System Solving. *Journal of Complexity*, 17 154–211, 2001.

[Gru93] Thomas R. Gruber. A translation approach to portable ontology specifications. *KNOWLEDGE ACQUISITION*, 5 199–220, 1993.

[GS94] D. Garlan, M. Shaw. An introduction to software architecture. In V. Ambriola, G. Tortora, editors, *Advances in Software Engineering and Knowledge Engineering*, 1–40. World Scientific Publishing Co., 1994.

[Har92] J. Harris. *Algebraic geometry: a first course.* Springer Verlag, 2. edition, 1992.

[Hei83] J. Heintz. Definability and fast quantifier elimination in algebraically closed fields. *Theor. Comput. Sci.*, 24 239–277, 1983.

[Hei89] J. Heintz. On the computational complexity of polynomials and bilinear mappings. A survey. *Proceedings 5th International Symposium on Applied Algebra, Algebraic Algorithms and Error Correcting Codes Springer LNCS*, 356 269–300, 1989.

[Her09] B. R. Hergenhahn. *An Introduction to the History of Psychology.* Wadsworth, 2009.

[HK04] J. Heintz, B. Kuijpers. Constraint databases, data structures and efficient query evaluation. *Constraint Databases. First International Symposium Springer LNCS*, 3074 1–24, 2004.

[HKB93] B. Hoffmann, B. Krieg-Brückner. Program development by specification and transformation, the prospectra methodology, language family, and system. London, UK, 1993. Springer-Verlag.

[HKP+00] J. Heintz, T. Krick, S. Puddu, J. Sabia, A. Waissbein. Deformation techniques for efficient polynomial equation solving. *Journal of Complexity*, 16 70–109, March 2000.

[HM] K. Hägele, J. L. Montaña. Polynomial random test for the equivalence of integers given by arithmetic circuits. Preprint 4/97, Departamento de Matemática, Estadística y Computación, Universidad de Cantabria (1997).

[HM93] J. Heintz, J. Morgenstern. On the intrinsic complexity of elimination theory. *Journal of Complexity*, 9 471–498, 1993.

[HMPW98] J. Heintz, G. Matera, L.M. Pardo, R. Wachenchauzer. The intrinsic complexity of parametric elimination methods. *Electron. J. SADIO*, 1 37–51, 1998.

[Hoa72] C. A. R. Hoare. Proof of correctness of data representations. *Acta Informatica*, 1 271–281, 1972.

[HS] J. Heintz, C.P. Schnorr. Testing polynomials which are easy to compute. *International Symposium on Logic and Algorithmic. Monogr. Enseig. Math. 30 (1982) 237–254 and 12th Annual Symposium ACM on Theory of Computing (STOC' 80) ACM Press (1980) 262–272.*

[Ive73] B. Iversen. *Generic Local Structure of the Morphisms in Commutative Algebra*. Springer-Verlag, Berlin, 1973.

[KI04] V. Kabanets, R. Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *Computational Complexity*, 1-2 (13) 1–46, 2004.

[KP96] T. Krick, L. M. Pardo. A computational method for diophantine approximation. *Algorithms in Algebraic Geometry and Applications. Proceedings of MEGA'94. Progress in Mathematics*, 143 193–254, 1996.

[Kre07] R. Kress. *Numerical analysis*. Academic Internet Publishers Incorporated, 2007.

[Kro87] L. Kronecker. Grundzüge Einer arithmetischen Theorie der algebraischen Grössen. *Festschrift zu Herrn Ernst Kummers fünfzigjährigem Doctor-Jubiläum. Cvelle J. Reine Angew. Math.*, 101 337–355, 1887.

[Kun85] E. Kunz. *Introduction to commutative algebra and algebraic geometry*. Birkhäuser, Boston, 1985.

[Lan93] S. Lang. *Algebra*. Addison-Wesley, Massachusetts, 1993.

[LFKN92] Carsten Lund, Lance Fortnow, Howard Karloff, Noam Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39 859–868, October 1992.

[LG01] B. Liskov, J. Guttag. *Program development in Java: Specification, and Object–Oriented Design*. Addison-Wesley, 3. edition, 2001.

[Lic90] T. M. Lickteig. On semialgebraic decision complexity. Habilitationsschrift, Universität Tübingen TR-90-052, Int. Comp. Sc. Inst., Berkeley, 1990.

[Lip94] R. J. Lipton. Straight–line complexity and integer factorization. *Algorithmic number theory. Springer LNCS*, 877 71–79, 1994.

[Mar02] O. Marker. *Model theory: An introduction*, 217. Springer, New York, 2002.

[MAS+03] J. McGovern, S. W. Ambler, M. E. Stevens, J. Linn, E. K. Jo, V. Sharan. *The Practical Guide to Enterprise Architecture*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2003.

[Med00] N. Medvidovic. *Software Architecture Description Languages*. University of California, 2000.

[Mey88] B. Meyer. *Object-Oriented Software Construction*. Prentice-Hall, 1. edition, 1988.

[Mey00] B. Meyer. *Object-Oriented Software Construction*. Prentice-Hall, 2. edition, 2000.

[Mor03] T. Mora. *SPES I: The Kronecker-Duval Philosophy*. Cambridge University Press, 2003.

[Mor05] T. Mora. *SPES II: Macaulay's Paradigm and Groebner Technology*. Cambridge University Press, 2005.

[Mum88] D. Mumford. *The red book of varieties and schemes*, 1358. Springer, Berlin Heidelberg, New York, 1. edition, 1988.

[Olv06] P. Olver. On multivariate interpolation. *Stud. Appl. Math.*, 116 (2) 201–240, 2006.

[PA09] S. L. Pfleeger, J. M. Atlee. *Software architecture: Theory and Practice*. Prentice Hall, Upper Suddle River, New Jersey, USA, 4. edition, 2009.

[Par72] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Commun. ACM*, 15 (12) 1053–1058, 1972.

[Pie02] B. C. Pierce. *Types and programming languages*. MIT Press, Cambridge, MA, USA, 2002.

[Pre01] R. S. Pressman. *Software Engineering: A Practitioner's Approach*. McGraw-Hill Higher Education, 2001.

[PS73] M. S. Paterson, L. J. Stockmeyer. On the number of nonscalar multiplications necessary to evaluate polynomials. *SIAM Journal on Computing*, 2 60–66, 1973.

[Ral70] A. Ralston. *Introducción al Análisis Numérico*. Limusa-Wiley, México, 1970.

[RW04] S. Rudich, A. Wigderson. *Computational complexity theory. Volumen 10 de IAS/Park City mathematics series*. AMS Bookstore, Boston, MA, USA, 2004.

[Sax09] N. Saxena. Progress on polynomial identity testing. *Bulletin of the EATCS*, 90 49–79, 2009.

[SB93] J. Stoer, R. Bulirsch. *Introduction to numerical analysis*. Springer, New York, 2. edition, 1993.

[Sch81] A. Schönhage. Partial and total matrix multiplication. *SIAM Journal of Computing*, 10 434–456, 1981.

[Sha84] A. Shamir. A polynomial time algorithm for breaking the basic Merkle–Hellman cryptosystem. *Information Theory, IEEE Transactions*, 30 699–704, 1984.

[Sha92] A. Shamir. IP=PSPACE. *J. ACM*, 39 869–877, 1992.

[Sha94] I. R. Shafarevich. *Basic algebraic geometry: Varieties in projective space.* Springer, Berlin Heidelberg, New York, 1994.

[SS95] M. Shub, S. Smale. On the intractability of Hilbert's Nullstellensatz and an algebraic version of "NP≠P?". *Duke Math. J.*, 81 47–54, 1995.

[Ste04] H. J. Stetter. *Numerical polynomial algebra.* Society for Industrial and Applied Mathematics, 2004.

[Str73] V. Strassen. Vermeidung von Divisionen. *Crelle J. Reine Angew. Math.*, 264 182–202, 1973.

[SW05] A. J. Sommese, Ch. W. Wampler. *The numerical solution of systems of polynomials arising in engineering and science.* World Scientific, 2005.

[Vog84] W. Vogel. *Results on Bézout's Theorem.* Tata Institute of Fundamental Research. Springer, 1984.

[vzGG03] J. von zur Gathen, J. Gerhard. *Modern Computer Algebra.* Cambridge University Press, 2003.

[ZS60] O. Zariski, P. Samuel. *Commutative algebra II*, 39. Springer, New York, 1960.