Software Design and Complexity in Effective Algebraic Geometry

Andrés Rojas Paredes
Departamento de Computación, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires.

Acknowledgment. The author expresses his gratitude to Joos Heintz, without him this article would not have been possible.

Abstract. We introduce a new, 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).

1 Introduction

In this paper we are going to show, by means of a mathematical computation model which mimics the notion of a programmable algorithm in Scientific Computing and a relevant example from polynomial equation solving over the complex numbers $\mathbb{C}$ (i.e. from geometric elimination theory), that *well motivated* and *natural* architectural restrictions on procedures may have devastating "side effects" on the run time complexity. This conclusion may also be expressed in terms of a trade–off between two (boolean) quality attributes: on one hand we have an architectural requirement and on the other a complexity class. The main issue of this paper is the use of Software Engineering as a metatheory which motivates the specific computation model we are going to explain in Section 2. This model (which is alternative to [2]) becomes combined in Section 3 with a novel mathematical technique, by the way related to singularity theory in algebraic geometry, in order to solve a thirty years old problem in algebraic complexity theory (see e.g. [12], [13]): we show that in circuit based effective elimination theory the elimination of a single existential quantifier block in the first order theory of algebraically closed fields of characteristic zero is *intrinsically hard* (i.e. requires *in worst case exponential time*). This contrasts with the fact that this problem can be solved in *pseudopolynomial time* (the corresponding procedure is called *Kronecker algorithm*; see [9],[6],[8],[10] for details). This is undoubtedly theoretical research. But there is also a practical aim behind that.

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 (this was the case during the implementation of the Kronecker algorithm by G. Lecerf, see [15]). 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 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.

In the rest of the paper we shall use freely notions and notations from algebraic geometry and algebraic complexity theory which, as far that they are not explained or introduced in Appendix A and B, are all standard (see for example [19] and [3]).

2 A software architecture based computation model

The aim of this section is to introduce and motivate a practically feasible, software architecture based model of (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 building blocks of our computations with circuits are supposed to be branching–free (see Section 2.2) 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 to introduce and discuss the concept of a *parameterized arithmetic circuit.* However, branchings are sometimes unavoidable, but 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.

2.1 Parameterized arithmetic circuits and their semantics

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

A *(by $\mathcal{M}$) parameterized arithmetic circuit* β (with *basic parameters* $\pi_1, \dots, \pi_r$ and *inputs* $X_1, \dots, X_n$) 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 $\pi_1, \dots, \pi_r$ or a input variable $X_1, \dots, X_n$. 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.

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 rational function of $\mathcal{M} \times \mathbb{C}^n$ 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 $\mathcal{M} \times \mathbb{C}^n$. If this occurs, we call the labelled DAG β *inconsistent*, otherwise *consistent*.

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 $\mathcal{M} \times \mathbb{C}^n$ 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 $\mathcal{M} \times \mathbb{C}^n \dashrightarrow \mathbb{C}^q$, 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 $\mathbb{C}^n \dashrightarrow \mathbb{C}^q$.

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 $\mathcal{M} \times \mathbb{C}^n \to \mathbb{C}^1$ which extends the intermediate result associated with ρ. Therefore, if β has K nodes, we obtain a total constructible map $\Omega : \mathcal{M} \times \mathbb{C}^n \to \mathbb{C}^K$ which extends the rational map $\mathcal{M} \times \mathbb{C}^n \dashrightarrow \mathbb{C}^K$ given by the labels at the indegree zero nodes and the intermediate results of β.

Definition 1 (Robust circuit). *The pair (β, Ω) is called a robust parameterized arithmetic circuit if the constructible map Ω is geometrically robust.*

For the notion of a rational, a constructible and a geometrically robust map see Appendix A. We shall make the following observation to Definition 1.

Suppose that (β, Ω) is robust. Then the constructible map $\Omega : \mathcal{M} \times \mathbb{C}^n \to \mathbb{C}^k$ is geometrically robust and hence also by Appendix A, Proposition 2 hereditary. Moreover, there exists at most one geometrically robust constructible map $\Omega : \mathcal{M} \times \mathbb{C}^n \to \mathbb{C}^K$ which extends the rational map $\mathcal{M} \times \mathbb{C}^n \dashrightarrow \mathbb{C}^K$ introduced before. Therefore we shall apply the term "robust" also to the circuit 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 2.2. Let us formulate this more precisely in the context of parameterized arithmetic circuits.

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

Lemma 1. *Suppose that the parameterized arithmetic circuit β is geometrically 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.

To our parameterized arithmetic circuit β we may associate different complexity measures and models. In this paper 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 [3] 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 rewriting procedure 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.

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.

Introducing variables 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, we may built a many sorted first order specification language $\mathcal{L}$ for parameterized arithmetic circuits. The non–logical symbols of $\mathcal{L}$ express the arithmetic operations in polynomial rings, the edge relation in DAG's, equality, node membership to a circuit or membership to a parameter domain or being an input or output of a given circuit β, etc.

The semantics of the specification language $\mathcal{L}$ is determined by the universe of all robust parameterized arithmetic circuits. The specification language $\mathcal{L}$ will allow us to apply Hoare logics to the computation model we are now going to develop in Section 2.2 and 2.3.

2.2 A 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.

Our 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* (see Appendix B for this notion). 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 $X, X', X'', \dots$ to denote vectors of standard input variables.

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 $X_1, X_2, \dots, X_n, \dots$ 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

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

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 this section that the objects handled by the routines of any shape of our computation model will always be robust parameterized arithmetic circuits. 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.

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*, say $\mathcal{A}$, of our computation model.

As input for $\mathcal{A}$ let be given a robust parameterized arithmetic circuit β. We suppose that we have already chosen for each node ρ, which depends at least on one of the input variables $X_1, \dots, X_n$, a generic computation

$$\begin{array}{lll} R^{(\rho)}_{X_i}(W_\rho; X^{(\rho)}), & R^{(\rho)}_{+}(W_\rho; U_\rho, Y_\rho; X^{(\rho)}), & R^{(\rho)}_{\cdot/}(W_\rho; U_\rho, Y_\rho; X^{(\rho)}), \\ R^{(\rho)}_{add}(W_\rho; Y_\rho, Z_\rho; X^{(\rho)}), & R^{(\rho)}_{mult}(W_\rho; Y_\rho, Z_\rho; X^{(\rho)}), & R^{(\rho)}_{div}(W_\rho; Y_\rho, Z_\rho; X^{(\rho)}), \end{array}$$

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 $R^{(\rho)}_{X_i}(w_\rho; X^{(\rho)})$ over $\mathbb{C}$ which is obtained by substituting in the generic computation $R^{(\rho)}_{X_i}(W_\rho; X^{(\rho)})$ 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 an irreducible, 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 an essential multiplication. Thus the corresponding generic computation has the form:

$$R^{(\rho)}_{mult}(W_\rho; Y_\rho, Z_\rho; X^{(\rho)}). \tag{1}$$

Let $R^{(\rho)}_{mult}(\kappa_\rho, Y_\rho, Z_\rho, X^{(\rho)})$ be the by $\mathcal{K}_\rho$ parameterized arithmetic circuits obtained by substituting in the generic computation (1) 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

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

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.

Recall again that we assumed earlier the vectors F_{ρ_1} and Y_ρ and F_{ρ_2} and Z_ρ having 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 1, that 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 $R^{(\rho)}_{X_i}(w_\rho; X^{(\rho)})$ is consistent and that all its intermediate results are polynomials in $X^{(\rho)}$ over $\mathbb{C}$ (although $R^{(\rho)}_{X_i}(w_\rho; X^{(\rho)})$ may contain divisions).

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, denoted by $\mathcal{A}(\beta)$, 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 $\mathcal{H}_{\mathcal{A}(\beta)}$.

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 1 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.

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.

The isoparametricity of recursive routines will constitute a cornerstone of the argumentation we are going to use in Section 3. Therefore we shall now comment this notion.

Since the first order theory of $\mathbb{C}$ admits quantifier elimination, one sees easily that, with the previous notations, the existence of a *constructible* map σ_ρ satisfying condition (B) is equivalent with the following 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

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

This justifies the term "isoparametric".

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, 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 $\mathit{\Gamma}_\rho := \{G_\rho^{(u)} \ ; \ u \in \mathcal{M}\}$ and $\mathit{\Phi}_\rho := \{F_\rho^{(u)} \ ; \ u \in \mathcal{M}\}$ 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* [17] from $\mathit{\Gamma}_\rho$ to $\mathit{\Phi}_\rho$, namely $\sigma_\rho : \mathit{\Gamma}_\rho \to \mathit{\Phi}_\rho$. 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 $\sigma_\rho : \mathit{\Gamma}_\rho \to \mathit{\Phi}_\rho$ necessarily arises.

Another point of view is the following:

By assumption the circuit β is an *arbitrary* admissible input for the given recursive routine $\mathcal{A}$. We say that the specification language $\mathcal{L}$ introduced in

Section 2.2 is *expressive relative to the routine* $\mathcal{A}$ if for any node ρ of β the geometrically robust constructible map σ_ρ is definable in $\mathcal{L}$. This means that there exists a vector of terms of the language $\mathcal{L}$, which depend all on the same circuit and node variables, such that this vector of terms, interpreted at the node ρ of the input circuit β, becomes the map σ_ρ.

If $\mathcal{L}$ is expressive relative to $\mathcal{A}$ and β and $\mathcal{A}(\beta)$ satisfy a precondition and a postcondition formulated in the language $\mathcal{L}$, then the correctness of $\mathcal{A}$ can be proved using Hoare Logics [1].

Suppose 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 reductions. 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 $\mathcal{A}(\beta)$.

From the point of view of software architecture, well behavedness under reductions is an absolutely natural quality attribute for recursive routines, because it authorizes simplification of "code" (represented by circuits) without any side effects. This motivates the following statement.

Lemma 2. *Let $\mathcal{A}$ be a recursive routine that behaves well under restrictions and reductions. Then $\mathcal{A}$ is isoparametric.*

One sees easily that the iterated application, i.e. *composition*, of isoparametric recursive routines which are well behaved under restriction leads again to such routines. The same happens with the *union* of such routines, which consists in the juxtaposition of their outputs (on the same input). More caution is at order when we consider the *join* of two isoparametric recursive routines which behave well under restrictions (join mimics the composition of functions). In this case the isoparametricity condition (B) becomes only satisfied at output nodes of the input circuit β. In this sense the join of two recursive isoparametric routines which are well behaved under restrictions is still *output isoparametric.*

Let $R(W;Y;X)$ be a generic computation of our shape list and w a complex vector of the same length as W such that $R(w;Y;X)$ is consistent and robust. If the vector of final results of $\mathcal{A}(\beta)$ has the same length as Y, we may join the circuits $\mathcal{A}(\beta)$ and $R(w;Y;X)$ at Y in order to construct a new robust parameterized arithmetic circuit.

Finally we may obtain any *elementary routine* of our branching–free computation model by the iterated application of all these construction patterns, in particular the last one, recursion, composition, union and join. 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 with irreducible parameter domain 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.

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 for our complexity considerations in Section 3.

Proposition 1. *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 $\mathcal{A}(\beta)$.*

2.3 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 endpoints of the pipes, decisions may be taken which involve only identity tests between robust constructible functions defined on the parameter domain under consideration. By their nature, the results of these identity tests are output isoparametric. 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 output isoparametric decisions, 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 [7] and [4]).

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 at most L, that β is totally division–free and that the final results of β have degree at most $d \leq 2^L$ and form a reduced regular sequence.

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 pre-computed 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.

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" [10] (compare also [9], [6] and [8]) may be programmed in our extended computation model.

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* $\mathcal{A}_{\text{final}}(\beta)$ 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 $\mathcal{A}_{\text{final}}(\beta)$, we call $\mathcal{A}$ a *procedure.*

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 by the language $\mathcal{L}$ 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 $\mathcal{A}^{(2)}(\mathcal{A}^{(1)}_{\text{final}}(\beta))$ we may, as in the case when we compute only parameters, apply garbage collection. In this manner $\mathcal{A}^{(2)}(\mathcal{A}^{(1)}_{\text{final}}(\beta))$ becomes reduced to a final output circuit $\mathcal{A}_{\text{final}}(\beta)$ 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 1 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 3.

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 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 on $\text{inv}(\beta)$ 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 requires a justification which cannot be given here for lack of space. In the case of elimination algorithms this restriction is necessary, since otherwise a potential new procedure could not compete with already known ones. In Section 3 we shall make a substantial use of this restriction.

3 An application of the extended computation model to complexity issues of effective elimination theory

In this section we shall consider elimination problems of the following simple type.

Let $r, n \in \mathbb{N}$, $U_1, \ldots, U_r, X_1, \ldots, X_n$ indeterminates, $U := (U_1, \ldots, U_r)$, $X := (X_1, \ldots, X_n)$ and let $G_1, \ldots, G_n$ be polynomials of $\mathbb{Q}[X]$ such that the equations $G_1 = 0, \ldots, G_n = 0$ define a non–empty, finite subset V of $\mathbb{C}^n$. Furthermore let be given an additional polynomial $H \in \mathbb{Q}[U, X]$ and observe that $F := \Pi_{x \in V}(Y - H(U, x))$ belongs to $\mathbb{Q}[U, Y]$ and describes the image of the map $\varphi : \mathbb{C}^r \times V \to \mathbb{C}^r \times \mathbb{C}^1$ which assigns to each point $(u, x) \in \mathbb{C}^r \times V$ the value $(u, H(u, x)) \in \mathbb{C}^r \times \mathbb{C}^1$.

We call F the elimination polynomial associated to $G_1 = 0, \ldots, G_n = 0$ and H. The determination of the image of φ by means of the computation of F constitutes a typical problem of effective elimination theory. We wish to make this more precise.

Let be given as input a robust parameterized arithmetic circuit β with parameter domain $\mathbb{C}^r$ (i.e. with basic parameters $U_1, \ldots, U_r$), inputs $X_1, \ldots, X_n$ and final results $G_1, \ldots, G_n$ and H.

The question is: what can be said about the size of the circuit $\mathcal{A}_{\text{final}}(\beta)$ when $\mathcal{A}$ is an *arbitrary* division–free procedure in the sense of Section 2.3 which solves the given elimination problem (this means that the output circuit $\mathcal{A}_{\text{final}}(\beta)$ has a single output node which represents the polynomial F)?

Thanks to our computation model this is a precise question to which we are now able to give a precise (but deceiving) answer.

Theorem 1. *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 equations $G_1^{(n)} = 0, \ldots, G_n^{(n)} = 0$ define a non–empty, finite subset V_n of $\mathbb{C}^n$ and constitute together with the polynomial $H^{(n)}$ an elimination problem as above, which depends on the parameters T, $U_1, \ldots, U_n$ and the inputs $X_1, \ldots, X_n$ and has an associated elimination polynomial $F^{(n)} \in \mathbb{C}[T, U, Y]$.*

(ii) *β_n is an ordinary division–free arithmetic circuit of size $O(n)$ 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 $F^{(n)}$. 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 $\Omega(2^n)$.*

There is no place here to exhibit the family of polynomials $G_1^{(n)}, \ldots, G_n^{(n)}$, $H^{(n)}$, $n \in \mathbb{N}$.

With other words, the simple elimination problem under consideration requires exponential time to represent its solution. Therefore the pseudopolynomial Kronecker and any other elimination algorithm, which is expressible in our computation model, cannot be boiled down to polynomial time.

On the other hand, Theorem 1 and its proof allow to conclude that a software engineer, using only *arithmetization techniques* [20], will never be able to find a *polynomial* algorithm to show $P = NP = coNP$.

The proof of Theorem 1 makes a substantial use of Theorem 2 in Appendix A or, in other words, of the notion of a geometrically robust constructible map. The output isoparametricity and the particular architecture of the procedure $\mathcal{A}$ play a crucial rôle in the argument.

The final outcome of our considerations 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.

A Appendix : Concepts and tools from Algebraic Geometry

A.1 Basic notations and definitions

A.1.1 Basic notations For any $n \in \mathbb{N}$, we consider the n–dimensional affine space $\mathbb{C}^n$ as equipped with its respective Zariski and Euclidean topologies over $\mathbb{C}$. In algebraic geometry, the Euclidean topology of $\mathbb{C}^n$ is also called the *strong topology.*

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.

Let V be a closed affine subvariety of $\mathbb{C}^n$, that is, the set of common zeros in $\mathbb{C}^n$ of a finite set of polynomials belonging to $\mathbb{C}[X]$.

We denote by $I(V) := \{f \in \mathbb{C}[X] : f(x) = 0$ for any $x \in V\}$ the ideal of definition of V in $\mathbb{C}[X]$ and by $\mathbb{C}[V] := \{\varphi : V \to \mathbb{C}$; there exists $f \in \mathbb{C}[X]$ with $\varphi(x) = f(x)$ 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 $\mathbb{C}^n$, 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 the $\mathbb{C}$–algebra of rational functions of V also by $\mathbb{C}(V)$. Observe that $\mathbb{C}(V)$ is the total fraction ring of $\mathbb{C}[V]$ and contains $\mathbb{C}[V]$.

A.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 $\mathbb{C}^n$ and $\mathbb{C}^m$, 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 [5]:

Definition 2 (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 3 (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.

A.1.3 Constructible sets and constructible maps Let $\mathcal{M}$ be a subset of the affine space $\mathbb{C}^n$ and, for a given nonnegative integer m, let $\phi : \mathcal{M} \dashrightarrow \mathbb{C}^m$ be a partial map.

Definition 4 (Constructible set). *We call the set $\mathcal{M}$* constructible *if $\mathcal{M}$ is definable by a Boolean combination of polynomial equations.*

A basic fact is the following: if $\mathcal{M}$ is constructible, then its Zariski closure is equal to its Euclidean closure (see, e.g. [18], Chapter I, §10, Corollary 1).

Definition 5 (Constructible map). *We call the partial map ϕ* constructible *if the graph of ϕ is constructible as a subset of the affine space $\mathbb{C}^n \times \mathbb{C}^m$.*

We say that ϕ is *polynomial* if ϕ is the restriction of a morphism of affine varieties $\mathbb{C}^n \to \mathbb{C}^m$ to a constructible subset $\mathcal{M}$ of $\mathbb{C}^n$ and hence a total map from $\mathcal{M}$ to $\mathbb{C}^m$. 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. the first order theory of algebraically closed fields with constants in $\mathbb{C}$, admits quantifier elimination, constructibility means just elementarily definability. In particular, ϕ constructible implies that the domain and the image of ϕ are constructible subsets of $\mathbb{C}^n$ and $\mathbb{C}^m$, respectively.

A useful fact concerning constructible maps we are going to use in the sequel is the following result (see, e.g. [16], Proposition 3.2.14).

Lemma 3. *Let $\mathcal{M}$ be a constructible subset of $\mathbb{C}^n$ and let $\phi : \mathcal{M} \dashrightarrow \mathbb{C}^m$ 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 $\phi : \mathcal{M} \to \mathbb{C}^m$ 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.

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.

A.2 Weakly continuous, strongly continuous and hereditary maps

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

Definition 6 (Conditions and notions). *Let $\mathcal{M}$ be a constructible subset of $\mathbb{C}^n$ and let $\phi : \mathcal{M} \to \mathbb{C}^m$ 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 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),*
- **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 $\mathcal{M}$.

A.2.1 The notion of geometrical robustness The main mathematical tool of this paper is the notion of geometrical robustness we are going to introduce now.

Let $\mathcal{M}$ be a constructible subset of the affine space $\mathbb{C}^n$.

We consider now the Zariski closure $\overline{\mathcal{M}}$ of $\mathcal{M}$ in $\mathbb{C}^n$. Since $\mathcal{M}$ is constructible, the strong and Zariski closures of $\mathcal{M}$ in $\mathbb{C}^n$ coincide. Observe that $\overline{\mathcal{M}}$ is a closed affine subvariety of $\mathbb{C}^n$ 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 $\mathfrak{M}_x$.

Let $\phi : \mathcal{M} \to \mathbb{C}^m$ be a constructible map. Then by Lemma 3 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 concept of a geometrically robust constructible map.

Definition 7. *Let $\mathcal{M}$ be a constructible subset of a suitable affine space and let $\phi : \mathcal{M} \to \mathbb{C}^m$ be a (total) constructible map with components $\phi_1, \ldots, \phi_m$. Based on Lemma 3 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) $\mathbb{C}[\overline{\mathcal{M}}]_{\mathfrak{M}_x}[\phi_1, \dots, \phi_m]$ *is a local* $\mathbb{C}[\overline{\mathcal{M}}]_{\mathfrak{M}_x}$*–algebra whose maximal ideal is generated by* $\mathfrak{M}_x$ *and* $\phi_1 - \phi_1(x), \dots, \phi_m - \phi_m(x)$.

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

In this paper we consider only the algebraically closed field $\mathbb{C}$. In this particular case we have the following characterization of geometrically robust constructible maps.

Theorem 2. *Let assumptions and notations be as before. Then the constructible map* $\phi : \mathcal{M} \to \mathbb{A}^m$ *is geometrically robust if and only if* ϕ *is strongly continuous.*

Theorem 2 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 for constructible maps.

Observe that for the algebraically closed field $\mathbb{C}$ Proposition 2 follows immediately from Theorem 2.

The proof of Theorem 2 makes a non–trivial use of deep tools from Algebraic Geometry like Zariski's Main Theorem [16]. For details we refer to [4] and [5].

The origin of the concept of a geometrically robust map can be found, implicitly, in [7] in a similar context as in Section 3 of this paper. Therefore this concept is well motivated from the point of view of Computer Science since it stems from this discipline.

B Appendix : Generic computations

For the notion of "ordinary" arithmetic circuits (or straight–line programs) we refer to [3].

In Section 2 we referred to ordinary arithmetic circuits over $\mathbb{C}$, whose indegree zero nodes are labelled by scalars and parameter and input variables, as *generic computations* [3] (also called *computation schemes* in [11]).

The aim of this concept 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 is fundamental for the design of routines of the computation model we describe in Section 2 of this paper. The routines of our computation model operate on robust parameterized arithmetic circuits and their basic ingredients are subroutines which calculate parameter instances of suitable, by the model previously fixed, generic computations. These generic computations become organized in finitely many families, called "shapes", which only depend on a constant number of discrete parameters. These discrete families constitute the basic building block of our computation model.

We shall now exemplify these abstract considerations in the concrete situation of the given parameterized arithmetic circuit β of Section 2.1. Mutatis mutandis we shall follow the exposition of [14], 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

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

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

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 specializations where this assignment fails *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 [3], Chapter 9 for more details on generic computations).

References

1. Apt, K.R.: Ten years of Hoare's logic: A survey–part I. ACM Transactions on Programming Languages and Systems (TOPLAS) 3(4), 431–483 (Oct 1981)
2. Blum, L., Cucker, F., Shub, M., Smale, S.: Complexity and Real Computation. Springer–Verlag (1998)
3. Bürgisser, P., Clausen, M., Shokrollahi, M.A.: Algebraic Complexity Theory. Grundlehren der mathematischen Wissenschaften, vol. 315. Springer Verlag (1997)
4. Castro, D., Giusti, M., Heintz, J., Matera, G., Pardo, L.M.: The hardness of polynomial equation solving. Foundations of Computational Mathematics 3(4), 347–420 (2003)
5. Giménez, N., Heintz, J., Matera, G., Solernó, P.: Lower complexity bounds for interpolation algorithms. Journal of Complexity 27, 151–187 (2011)
6. Giusti, M., Hägele, K., Heintz, J., Montaña, J.L., Morais, J.E., Pardo, L.M.: Lower bounds for diophantine approximation. Journal of Pure and Applied Algebra 117, 277–317 (1997)
7. Giusti, M., Heintz, J.: Kronecker's smart, little black boxes. In: Foundations of Computational Mathematics, R. A. DeVore, A. Iserles, E. Süli eds., London Mathematical Society Lecture Note Series, vol. 284, pp. 69–104. Cambridge University Press, Cambridge (2001)
8. Giusti, M., Heintz, J., Morais, J.E., Pardo, L.M.: Le rôle des structures de données dans les problemes d'élimination. Comptes Rendus Acad. Sci. Serie 1(325), 1223–1228 (1997)
9. Giusti, M., Heintz, J., Morais, J., Morgenstern, J., Pardo, L.: Straight-line programs in geometric elimination theory. Journal of Pure and Applied Algebra 124, 101–146 (1998)
10. Giusti, M., Lecerf, G., Salvy, B.: A Gröbner Free Alternative for Polynomial System Solving. Journal of Complexity 17, 154–211 (2001)
11. Heintz, J.: 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)
12. Heintz, J., Sieveking, M.: Absolute primality of polynomials is decidable in random polynomial time in the number of variables. Automata, languages and programming (Akko, 1981). Lecture Notes in Computer Science 115, 16–28 (1981)
13. Kaltofen, E.: Greatest common divisors of polynomials given by straight–line programs. J. Assoc. Comput. Mach. 35(1), 231–264 (1988)
14. Krick, T., Pardo, L.M.: A computational method for diophantine approximation. Algorithms in Algebraic Geometry and Applications. Proceedings of MEGA'94. Progress in Mathematics 143, 193–254 (1996)
15. Lecerf, G.: Kronecker: a Magma package for polynomial system solving. Web page. `http://lecerf.perso.math.cnrs.fr/software/kronecker/index.html`
16. Marker, O.: Model theory: An introduction, vol. 217. Springer, New York (2002)
17. Meyer, B.: Object-Oriented Software Construction. Prentice-Hall, 2. edn. (2000)
18. Mumford, D.: The red book of varieties and schemes, vol. 1358. Springer, Berlin Heidelberg, New York, 1. edn. (1988)
19. Shafarevich, I.R.: Basic algebraic geometry: Varieties in projective space. Springer, Berlin Heidelberg, New York (1994)
20. Shamir, A.: IP=PSPACE. J. ACM 39, 869–877 (1992)