Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 339 (2018) 135–146

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

# On the Computational Complexity of Information Hiding

Andrés Rojas Paredes[1,2]

*Instituto de Ciencias*
*Universidad Nacional de General Sarmiento*
*J. M. Gutiérrez 1150 (B1613GSX), Los Polvorines, Provincia de Buenos Aires, Argentina*

*Departamento de Computación*
*Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires*
*Pabellón I (C1428EGA), Ciudad Universitaria, CABA, Argentina*

**Abstract**

In this work we study the intrinsic complexity of elimination algorithms in effective algebraic geometry and we focus our attention to elimination algorithms produced within the object–oriented paradigm. To this end, we describe a new computation model called *quiz game* (introduced in [1]) which models the notions of information hiding (due to Parnas, see [8]) and non–functional requirements (e.g. robustness) among other important concepts in software engineering. This characteristic distinguish our model from classical computation models such as the Turing machine or algebraic models.
We illustrate our computation model with a non–trivial complexity lower bound for the identity function of polynomials. We show that any object–oriented (and robust) implementation of the identity function of polynomials is necessarily inefficient compared with a trivial implementation of this function. This result shows an existing synergy between Software Engineering and Algebraic Complexity Theory.

*Keywords:* Abstract data type, abstraction function, data structure, information hiding, lower complexity bound, non–functional requirement, quantifier elimination, quiz game, robustness, scientific computing.

## 1 Introduction

### *1.1 The Problem to be Solved*

The first–order theory of algebraically closed fields with constants in the complex numbers $\mathbb{C}$ (elementary theory of $\mathbb{C}$), admits quantifier elimination, i.e., for every existentially quantified formula $\Phi$, there exists a logically equivalent formula $\Psi$ which is free of quantifiers. For example, let $X_1, \ldots, X_n$ be indeterminates over $\mathbb{C}$, let $X := (X_1, \ldots, X_n)$, let $M$ be an $n \times n$ matrix, and let us denote the determinant

[1] This work is supported by the research project PIO N° 144-20140100027-CO

[2] Email:arojas@ungs.edu.ar

https://doi.org/10.1016/j.entcs.2018.06.009

of a matrix with the word 'det'; the quantified formula

$$\Phi := ((\exists X)(M * X = 0 \wedge X \neq 0))$$

describes the existence of a non–trivial solution for a given system of linear equations, and is logically equivalent to the following quantifiers free formula

$$\Psi := (\det(M) = 0).$$

A fundamental open problem in Algebraic Complexity Theory asks for the *intrinsic complexity* of eliminating a single existential quantifier block in the elementary theory of $\mathbb{C}$. We study this problem by means of non–trivial lower complexity bounds which are based on basic notions in Software Engineering.

Observe that formulas $\Phi$ and $\Psi$ show that, in the elementary theory of $\mathbb{C}$, the formulas involved in a quantifier elimination problem are typically composed by polynomial equations. Thus, quantifier elimination is in a more general context a problem of polynomial equation solving which is an important subject in scientific computing. Our complexity questions will focus on the adequacy of the software design behind polynomial equation solvers.

### *1.2 Motivation*

An approach to deal with the complexity of polynomial equation solving was to improve the algorithms and the data structures inside the algorithms. This factor affected considerably the evolution of elimination algorithms complexity, for example, first elimination algorithms (around 1980) using polynomials given by coefficients (dense and sparse representation) were not efficient, its complexity was doubly exponential, say

$$s^{O(1)}d^{n^{O(n)}} \tag{1}$$

where $d$ is the degree of the system, $s$ is the number of polynomials defining it, and $n$ is the number of variables to be eliminated (see [9] for a survey on first elimination algorithms). This complexity was later reduced to the simply exponential $s^{O(1)}d^{O(n^2)}$ by means of the Effective Nullstellensatz (see [6] and [11]). In 1992, a new data structure produced considerable progress, polynomials were implemented by means of arithmetic circuits evaluating them, this new data structure reduced the complexity to

$$s^{O(1)}d^{O(n)}. \tag{2}$$

**Definition 1.1** [arithmetic circuit] An arithmetic circuit is a directed acyclic graph which is labelled by arithmetic operations addition, subtraction or multiplication.

First elimination algorithms using arithmetic circuits were probabilistic and hybrid, i.e., outputs were in arithmetic circuit representation, whereas inputs were still in coefficient representation. The first algorithm for the resolution of polynomial equation systems over algebraically closed fields which fully implements polynomi-

als in terms of arithmetic circuits is the *Kronecker* algorithm [3] (see e.g. [3]). It was implemented in a Magma package of identical name by Grégoire Lecerf. This solver reduced the complexity (2) to the pseudo-polynomial

$$\delta^2(snd)^{O(1)} \tag{3}$$

where $\delta$ is a discrete semantic parameter which in worst case may become $d^n$ (see [3]).

The last implementation of the Kronecker algorithm is called *geomsolvex*, and solves systems of polynomial equations over $\mathbb{C}$. This package is implemented in the high level language Mathemagix, which is imperative, strongly typed with polymorphism and parametrized types. It allows the compiler to generate extremely fast code (comparable to the speed of C or C++). We want to know whether the current complexity of the *Kronecker* algorithm may be improved. This practical aim may be seen as an specific instance of a more general concern in Software Engineering which consists on the evaluation of a software design with respect to a complexity requirement.

*Kronecker* is a circuit–based elimination algorithm which implements polynomials in terms of arithmetic circuits. Thus, we may suppose that circuit–based elimination algorithms work with a class Polynomial and a class Arithmetic Circuit, such that the class diagram in Figure 1 illustrates the basic design behind this kind of algorithms.

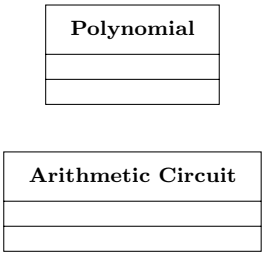


Fig. 1. Basic design behind elimination algorithms

Our practical aim is to provide a mathematical tool which allows us to answer the question whether software design in Figure 1 can be used to improve the complexity of the *Kronecker* algorithm for the computation task of quantifier elimination. If this is not the case, we may look for an alternative design, namely, a more suitable implementation of the mathematical concept of polynomials. Thus we may replace circuits by another data structure. In this scenario, the new data structure may provide a different interface such that circuit–based algorithms would become useless. To avoid such a situation, a standard strategy hides the representation and works with object-oriented elimination algorithms (see e.g. [7]). To capture such algorithms we are going to define a computation model which captures the notion of information hiding.

In the following we shall explore this example.

[3] In honour of Leopold Kronecker (1823–1891) pioneer in elimination algorithms.

# 2 Related and Prior Work

## *2.1 A Mathematical Model for a Basic Design*

We define first in a general context the basic components of a mathematical model for design in Figure 1 above. Let $(X_n)_{n\in\mathbb{N}}$ be a sequence of indeterminates over $\mathbb{C}$ and let

$$\mathcal{R} := \bigcup_{n\in\mathbb{N}} \mathbb{C}[X_1, \ldots, X_n]$$

where $\mathbb{C}[X_1, \ldots, X_n]$ denotes the polynomial ring in $X_1, \ldots, X_n$ over $\mathbb{C}$.

Our mathematical model works with the notions of constructible subset and constructible map:

**Definition 2.1** [constructible subset] Let $X_1, \ldots, X_n$ be indeterminates over $\mathbb{C}$ and let $X := (X_1, \ldots, X_n)$. Let $\mathbb{C}[X] := \mathbb{C}[X_1, \ldots, X_n]$ be the ring of polynomials in the variables $X$ with coefficients in $\mathbb{C}$. Let $\mathcal{M}$ be a subset of some affine space $\mathbb{C}^n$. We call the subset $\mathcal{M}$ constructible if $\mathcal{M}$ is definable by a Boolean combination of polynomial equations from $\mathbb{C}[X]$.

**Definition 2.2** [constructible map] A map $\Phi : \mathbb{C}^n \to \mathbb{C}^m$ is a constructible map if the graph of $\Phi$ is definable by a Boolean combination of polynomial equations from $\mathbb{C}[X]$.

In this context, class Polynomial will be a *framed abstract data type carrier of polynomials*.

**Definition 2.3** [framed abstract data type carrier] A framed abstract data type carrier of polynomials is a constructible subset of a finite dimensional $\mathbb{C}$-vector space contained in $\mathcal{R}$.

On the other hand, our model for class Arithmetic Circuit will be a *framed data structure*.

**Definition 2.4** [framed data structure] A framed data structure is a constructible subset of a suitable affine ambient space over $\mathbb{C}$.

For a framed data structure $\mathcal{N}$, the *size* of $\mathcal{N}$ is the dimension of its ambient space. This value will be a lower bound for the size of the objects belonging to the class Arithmetic Circuit.

Another element we have to take into account is that we are interested in implementations which are *robust*, namely, implementations with the ability to react appropriately to abnormal events which occur during the execution of the system. We are going to model mathematically the notion of robustness by the precise notion of a geometrically robust map:

**Definition 2.5** [geometrically robust map] Let $\mathcal{M}$ be a constructible subset of $\mathbb{C}^n$ and let $\Phi : \mathcal{M} \to \mathbb{C}^m$ be a constructible total map. Then $\Phi$ is geometrically robust if $\Phi$ is *continuous* with respect to the Euclidean topologies of $\mathcal{M}$ and $\mathbb{C}^n$ (see [4, Theorem 4]).

The notion of geometrical robustness was explicitly studied as a software quality attribute in [10]. In the following, the functions and maps we consider should be geometrically robust in order to model and capture the non-functional requirement of robustness.

A framed abstract data type carrier $\mathcal{O}$ of polynomials depends on a framed data structure $\mathcal{M}$ of parameters (see example in Section 3.2 below). The connection between a parameter in $\mathcal{M}$ and a polynomial in $\mathcal{O}$ is given by a geometrically robust constructible map $\theta$ which has the role of an abstraction function. Additionally, the polynomials in $\mathcal{O}$ can always (not necessarily efficiently) be implemented by an arithmetic circuit which we model with a suitable framed data structure $\mathcal{N}$. The coefficientwise description of the elements of $\mathcal{O}$ defines a polynomial map $\omega : \mathcal{N} \to \mathcal{O}$. On the other hand, we need to parametrize $\mathcal{N}$ by $\mathcal{M}$ using a geometrically robust constructible map $\mu : \mathcal{M} \to \mathcal{N}$ (see [4] and [1]). The situation is depicted by the following commutative diagram.

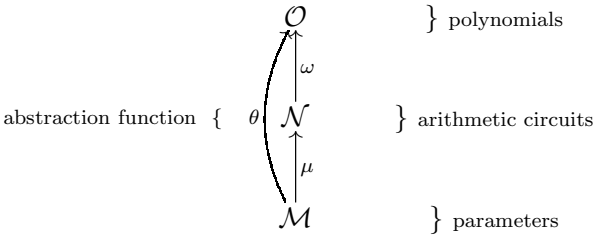


Fig. 2. Mathematical model for design in Figure 1

### *2.2 A Model for Circuit–Based Elimination*

Let be given two abstraction functions $\theta : \mathcal{M} \to \mathcal{O}$, $\theta' : \mathcal{M} \to \mathcal{O}'$ as before, associated with geometrically robust constructible maps $\mu : \mathcal{M} \to \mathcal{N}$, $\mu' : \mathcal{M} \to \mathcal{N}'$, and polynomial maps $\omega : \mathcal{N} \to \mathcal{O}$, $\omega' : \mathcal{N}' \to \mathcal{O}'$, namely $\theta = \omega \circ \mu$ and $\theta' = \omega' \circ \mu'$. Suppose that there exists a geometrically robust constructible map $\tau : \mathcal{O} \to \mathcal{O}'$ which models the computation task of quantifier elimination in terms of framed abstract data type carriers $\mathcal{O}$ and $\mathcal{O}'$. An implementation of the computation task $\tau$ is given by a map $\mu'$ such that the following diagram of geometrically robust constructible maps commutes.

We understand the framed data structure $\mathcal{N}'$ as a mathematical model for the output domain of robust and circuit–based elimination algorithms (see design in Figure 1). This model is based on [5], [2] and [4] and allows to obtain exponential lower bounds for the size of $\mathcal{N}'$. Thus, it is not likely to improve the complexity of circuit–based elimination algorithms. In the following we are going to extend this model in order to capture the complexity of information hiding.

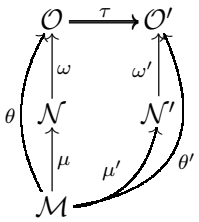


Fig. 3. Our model for circuit–based elimination

# 3 Proposed Approach

Previous conclusion suggests that we should discard design in Figure 1 and look for alternative designs if we want to improve the complexity of the *Kronecker* algorithm. Thus, we should consider the case where arithmetic circuits become replaced with another data structure. The problem of this option is that the *Kronecker* algorithm and any circuit–based elimination algorithm is highly coupled with the current design of polynomials given by arithmetic circuits. Thus, any change in the design produces a cascade of modifications along the algorithm. A strategy to avoid this situation is to hide the representation of polynomials, and to produce algorithms which only work with the interface of polynomials.

## *3.1 Quiz Games: a Model for Information Hiding*

In this section we are going to follow the terminology introduced in Section 2. We are interested in implementations of computation task $\tau$ which are independent of the representation. In order to describe and capture such implementations we are going to model the notion of information hiding given by [8] and [7]. To this end, we shall define a two–party protocol called *quiz game* which models the notion of information hiding. The model we are going to describe is the same as in [1]. However, in this work we focus on the main aspects of software engineering without renouncing on mathematical precision.

The agents in the quiz game protocol are called quizmaster and player. The *player* models the programmer and the *quizmaster* models the object–oriented paradigm where the programmer works. The player (programmer) has unlimited computational power (we do not restrict his creativity) but he is only restricted to use the interface provided by the class *Polynomial* (observers and constructors). This restriction on the tools available to the player in combination with its unlimited power to combine such tools models an object–oriented algorithm.

On the other hand, the quizmaster, who models the computation problem to solve, hides the internal representation of polynomials and only provides observers and constructors to work with polynomials. Direct access to the representation is denied and the player do not know whether the polynomials are given by coefficients, arithmetic circuits, or other data structure. Thus, the quizmaster also models the notion of information hiding.

What is specific of the object–oriented paradigm in this analysis of information hiding? the restriction to only use observers and constructors and hide the internal representation. Therefore, our model capture the elimination algorithms produced by any language which supports abstract data types.

### *3.1.1 The Protocol of the Quiz Game*

The following points describe the protocol for an instance of the game.

- In a first step, the quizmaster chooses a parameter $u$ in $\mathcal{M}$ which by means of abstraction function $\theta$ represents an input polynomial of the framed abstract data type carrier $\mathcal{O}$, namely $\theta(u)$. This polynomial $\theta(u)$ determines an output polynomial $\tau(\theta(u))$ which is the result of the elimination task $\tau$. The player is required to compute this output polynomial by means of operations applied to polynomial $\theta(u)$. The quizmaster hides the representation $u$ and only provides queries, creators and commands (observers and constructors) to operate with polynomial $\theta(u)$.
- In a second step, the player asks the quizmaster questions about the input polynomial $\theta(u)$. These questions are limited to query functions (observers). Consequently, quizmaster's answers $\widetilde{\sigma}(\theta(u))$ constitute a vector of complex values which depend only on the input polynomial $\theta(u)$ and are independent of the hidden representation $u$.
- In a third step, the player applies creators and commands (constructors) to quizmaster's answers $\widetilde{\sigma}(\theta(u))$ to compute the required output polynomial $\tau(\theta(u))$.
- Finally, the player sends to the quizmaster his computation $\mu^*(\widetilde{\sigma}(\theta(u)))$ which is a representation for the output polynomial. Then, the quizmaster tests whether player's representation $v^* := \mu^*(\widetilde{\sigma}(\theta(u)))$ represents the required output polynomial $\tau(\theta(u))$. To this end, the quizmaster checks whether $\omega^*(v^*) = \tau(\theta(u))$, in such a case the player wins the instance of the game given by $u$. Observe that this last test is carried–out by the quizmaster within efficient complexity bounds since $\theta'$ and $\theta^*$ belong to a compatible collection of abstraction functions. Thus, this step has no influence in the complexity of the whole process.

The whole situation becomes depicted by the commutative diagram in Figure 4.

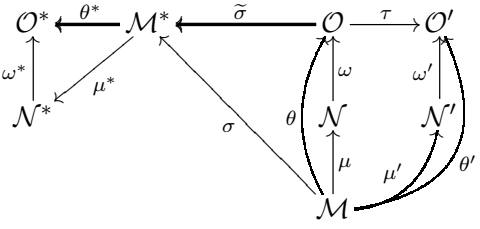


Fig. 4. Our model for object–oriented elimination

We say that the player has a *winning strategy* if he wins the game for any $u \in \mathcal{M}$.

A winning strategy is called *efficient* if the size of $\mathcal{N}^*$ is polynomial in the size of $\mathcal{N}$. Otherwise it is called *inefficient*. Observe that $\widetilde{\sigma}$ and $\theta^* = \omega^* \circ \mu^*$ define a winning strategy for the quiz game protocol if and only if $\theta' = \omega^* \circ \mu^* \circ \widetilde{\sigma} \circ \theta = \omega^* \circ \mu^* \circ \sigma = \theta^* \circ \sigma$ holds. In this case we have $\mathcal{O}' = \mathcal{O}^*$.

### *3.2 Example: an alternative identity function*

We are going to finish this section with an example of our quiz game protocol: the identity function of polynomials. We are going to show that information hiding imposes an intrinsic and non-trivial complexity lower bound to this simple function.

Let $D \in \mathbb{N}$ and

$$F_D(U, X) := (U^{D+1} - 1) \sum_{0 \leq k \leq D} U^k X^k \ \in \mathbb{C}[U, X],$$

$\mathcal{O}_D := \{F_D(u, X); u \in \mathbb{C}\}$, $\mathcal{M}_D := \mathbb{C}$ and $\theta_D : \mathcal{M}_D \to \mathcal{O}_D$, $\theta_D(u) := F_D(u, X), u \in \mathbb{C}$. The set of univariate polynomials $\mathcal{O}_D$ is a framed abstract data type. Let $\mathcal{M}_D$ and $\mathcal{N}_D := \mathcal{M}_D$ be framed data structures and $\theta_D$ an abstraction function associated with the polynomial maps $\mu_D := \mathrm{id}_{\mathcal{M}_D}$ and $\omega_D := \theta_D$. In this scenario, the computation task $\tau_D$ is the identity function of $\mathcal{O}_D$. A trivial implementation should be the identity function of the representation, namely $\mu'_D = \mathrm{id}_{\mathcal{M}_D}$. Unfortunately, this implementation accesses to the representation of polynomials. We are going to play a quiz game in order to model an implementation of $\tau_D = \mathrm{id}_{\mathcal{O}_D}$ which does not have access to the representation. Figure 5 illustrates the identity function and its implementation.

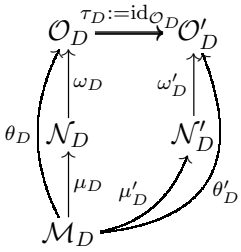


Fig. 5. Identity function of polynomials

Observe that there are ten natural (random) numbers $\xi_1^{(D)}, \ldots, \xi_{10}^{(D)}$ such that the image $\mathcal{M}_D^*$ of the polynomial map $\tilde{\sigma}_D : \mathcal{O}_D \to \mathcal{M}_D^*$ is a subset of $\mathbb{C}^{10}$, and, for $u \in \mathcal{M}_D$,

$$\tilde{\sigma}_D(F(u, X)) := (F_D(u, \xi_1^{(D)}), \ldots, F_D(u, \xi_{10}^{(D)}))$$

models mathematically a computation in terms of the interface of polynomials, more precisely, $\tilde{\sigma}_D$ models a kind of method of the class Polynomial called observer (observers and constructors of a class are a modern name for the *routines* described in [7]).

Let us consider a quiz game adapted to the situation. The player disposes over an abstraction function $\theta_D^*$ with $\mathcal{N}_D^*$ a framed data structure and $\mu_D^*$ a geometrically robust constructible and $\omega_D^*$ a polynomial map. Observe that $\theta_D^*$ interpolates the polynomials $F_D(u, X) \in \mathcal{O}_D$ from the data $\tilde{\sigma}_D(F(u, X)) \in \mathcal{M}_D^*$, $u \in \mathcal{M}_D$. Thus, $\theta_D^*$ models mathematically a constructor of the class Polynomial.

The quizmaster chooses a parameter $u \in \mathcal{M}_D$ and hides it to the player. The player asks to the player the values contained in the vector

$$\tilde{\sigma}_D(\theta_D(u)) = (F(u, \xi_1^D), \ldots, F(u, \xi_{10}^D)) \in \mathcal{M}_D^*$$

and computes from them the vector $v^* = \mu^*(\tilde{\sigma}_D(\theta_D(u)))$. Finally the quizmaster checks whether $F(u, X) = \omega_D^*(v^*)$ holds. If the player has a winning strategy we obtain the following commutative diagram

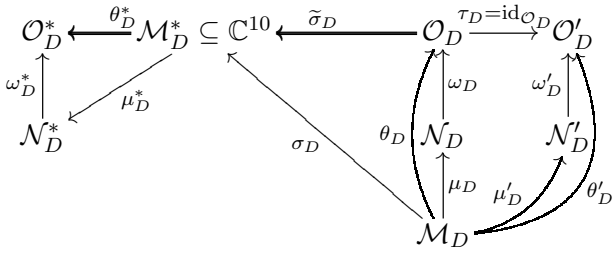


Fig. 6. An alternative identity function

Any polynomial in $\mathcal{O}_D$ can be represented by an arithmetic circuit of logarithmic size. On the other hand, observe that the player computes a polynomial whose representation $v^*$ belongs to $\mathcal{N}_D^*$. With our method we are able to prove that the size of $\mathcal{N}_D^*$ is at least $D + 1$. This exponential complexity blow up between a trivial implementation of the identity function (namely, return the representation) and an implementation which does not have access to the representation $(\tilde{\sigma}_D \circ \theta_D^*)$ is due to the fact that we imposed the conditions of geometrical robustness and information hiding.

# 4 Applications

## *4.1 Lower Complexity Bounds*

For details and proofs in this section we refer to [1].

Let $L, n \in \mathbb{N}$ with $2^{\frac{L}{4}} \geq n$ and let $\mathcal{O}_{L,n}$ be the abstract data type of all polynomials of $\mathbb{C}[X_1, \ldots, X_n]$ which can be evaluated using at most $L$ essential multiplications ($\mathbb{C}$–linear operations are free). We think the polynomials in $\mathcal{O}_{L,n}$ represented by arithmetic circuits and let us suppose that this representation is hidden behind the interface of class Polynomial (see Figure 1).

**Theorem 4.1** *Consider the task of replacing the given hidden representation of the elements of $\mathcal{O}_{L,n}$ by a known one using a quiz game with winning strategy. Then any such quiz game is inefficient requiring a representation of size at least $2^{\Omega(Ln)}$.*

In particular, Theorem 4.1 generalizes our example in Section 3.2 and says that there exist polynomials which are easy to evaluate, but difficult to interpolate. This interpolation result shows that our quiz game model allows us to obtain not only results concerning elimination but also other problems in scientific computing, for example, we have results in neural networks (see e.g. [1] Section 4.2).

We can generalize Theorem 4.1 in the following theorem.

**Theorem 4.2** *Let $n$ be a discrete parameter. There exists an existential first order formula of size $O(n^3)$ of the elementary language of $\mathbb{C}$ which possesses a canonical equivalent quantifier free formula containing a robustly parametrized univariate polynomial such that any representation of this polynomial (obtained by means of robust and object–oriented algorithms) is of size $2^n$.*

In other words, Theorem 4.2 formalizes a common intuition which states that the use of abstraction, information hiding and interfaces introduces also a certain limit to the efficiency of the underlying algorithms and programs.

### *4.2 Software Design*

In this section we highlight some practical implications of previous results. According to SEMAT [4] (the initiative to reshape software engineering to qualify as a rigorous discipline) the three main activities in software construction are specification, design and implementation. Our computation model can be used in the design activity when a software design should satisfy a certain complexity requirement. The following paragraph illustrates this application.

Let us recall that elimination algorithms are usually implemented according to design in Figure 1, namely polynomials implemented in terms of arithmetic circuits. An example of this kind of algorithms is the *Kronecker* algorithm whose complexity status was proved to be optimal (see e.g. [4]). Therefore, we may conclude that a circuit–based implementation of polynomials cannot be used in order to improve the current complexity of circuit–based elimination algorithms and we should discard design in Figure 1 and think in alternative designs. Which implementation of polynomials should we use? Figure below illustrates this question.

According to Section 3, an alternative to deal with unknown data structures is to use object–oriented programming and information hiding. Thus, we hide the representation of polynomials and we define our algorithms in terms of suitable methods of the class Polynomial (observers and constructors). Notice that we do not fix in advance the software design, but we fix the tools available for the programmer, namely, the methods interface. Our computation model based in quiz games captures these restrictions and allows us to give Theorem 4.2 which says

[4] SEMAT (Software Engineering Method and Theory) was launched in December 2009 by Ivar Jacobson, Bertrand Meyer, and Richard Soley.

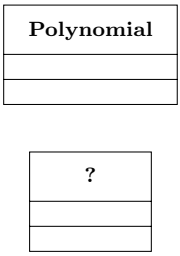


Fig. 7. Our model evaluates a design for elimination

that it is not possible to improve the complexity of elimination algorithms using the object–oriented approach (and considering robust implementations). In this sense, our computation model allows us to discard an interface. Moreover, Theorem 4.2 suggests that *there is no interface* divided into observers and constructors of polynomials which may be used to solve elimination in an efficient way. Such a conclusion suggests an intrinsic limit for the software designer of modules, class interfaces, and abstract data types of polynomials. This conclusion is very similar to a computability result but a little more specific because it refers to the ability to make an efficient software.

## 5 Open Questions and Future Work

We described a computation model which is inspired in the notion of information hiding. This computation model allows us to prove non–trivial and exponential lower complexity bounds for fundamental problems in computational mathematics, although we exhibited a simple case study, namely the identity function of polynomials. This application of our model may be summarized as follows: on one hand we have an algorithm produced by software engineering criteria (information hiding and non–functional requirements) on the other hand we have a mathematical model which captures this kind of algorithms. This mathematical model (the quiz game we introduce in Section 3) allows us the application of mathematical tools in order to obtain conclusions which must be translated back to the original computer science context. Figure 8 below illustrates this idea.

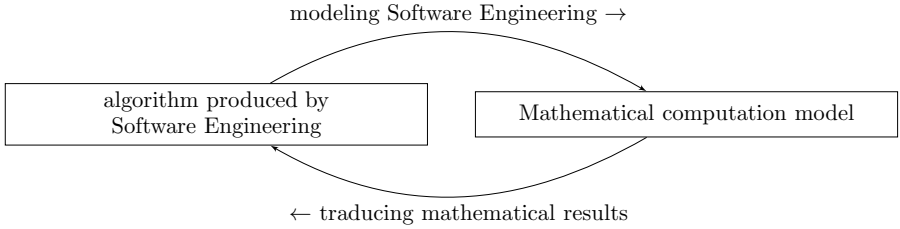


Fig. 8. Algorithms and its model.

Figure 8 shows an existing synergy between Software Engineering and Algebraic Complexity Theory. This synergy allows formal mathematical conclusions which are new in the current state of the art in Computational Complexity Theory ([1]) and

may be useful tools to understand important and still informal Software Engineering concepts, e.g. information hiding. This application may be a step towards the goal of the SEMAT kernel which wants to make Software Engineering become a rigorous discipline. However, our results are limited to examples of a very specific field of mathematics, namely algebraic geometry. An open question asks for the generalization of our model to other problems, for example, an application of our quiz game model can be found in the subject of neural networks in machine learning (see [1]).

## Acknowledgements

The author is grateful to Joos Heintz for his advice and guidance during the writing of this work, to Sebastián Uchitel and Diego Garvervetsky for helpful comments concerning software engineering, and to the referees for their valuable comments which helped to improve the manuscript.

## References

[1] Bank, B., J. Heintz, G. Matera, J. L. Montaña, L. M. Pardo and A. Rojas Paredes, *Quiz Games as a Model for Information Hiding*, Journal of Complexity **34** (2016), pp. 1 – 29.

[2] Giménez, N., J. Heintz, G. Matera and P. Solernó, *Lower Complexity Bounds for Interpolation Algorithms*, Journal of Complexity **27** (2011), pp. 151 – 187.

[3] Giusti, M., G. Lecerf and B. Salvy, *A Gröbner Free Alternative for Polynomial System Solving*, Journal of Complexity **17** (2001), pp. 154 – 211.
URL http://www.sciencedirect.com/science/article/pii/S0885064X00905715

[4] Heintz, J., B. Kuijpers and A. Rojas Paredes, *Software Engineering and Complexity in Effective Algebraic Geometry*, Journal of Complexity **29** (2013), pp. 92–138.

[5] Heintz, J. and J. Morgenstern, *On the Intrinsic Complexity of Elimination Theory*, Journal of Complexity **9** (1993), pp. 471–498.

[6] Kollar, J., *Sharp Effective Nullstellensatz*, Journal of the American Mathematical Society **1** (1988), pp. 963–975.

[7] Meyer, B., "Object–Oriented Software Construction," Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1997, 2 edition.

[8] Parnas, D. L., *On the Criteria to Be Used in Decomposing Systems into Modules*, Commun. ACM **15** (1972), pp. 1053–1058.

[9] Puddu, S. I., "Un Algoritmo Efectivo para la Eliminación de Cuantificadores," Ph.D. thesis, Universidad de Buenos Aires, Argentina (1995).

[10] Rojas Paredes, A., "Complexity as Quality Attribute in Software Design," Master's thesis, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires, Pabellón 1, Ciudad Universitaria (2011).

[11] Sombra, M., *A Sparse Effective Nullstellensatz*, Adv. Appl. Math. **22** (1999), pp. 271–295.
URL http://dx.doi.org/10.1006/aama.1998.0633