Serdica
Bulgariacae mathematicae
publicationes

Сердика
Българско математическо
списание

MATHEMATICS AND COMPUTER SCIENCE A CASE STUDY: ALGEBRAIC ALGORITHMS

A. MIOLA

The subject of this paper concerns the relations between Mathematics and Computer Science. We do like to discuss questions about the theoretical bases of the so-called *Computer Algebra,* also known as *Symbolic and Algebraic Manipulation* (see [1]). Our main goal is to show how the algebraic structure of the entire class of algorithms for computer algebra systems gives us a strong and powerful methodologies to really structure and analyze solutions to computational problems.

1. Introduction. Two main questions raise in Computer Science in term of establishing philosophical and computational principles

First: *What a machine can do?* Or, with a more precise formulation: *What computations can be performed by a machine, if you assume to ignore real limitations such as computing time and memory size?*

Second: *How easy it is for a machine to do what it can do?* Or: *What is the efficiency the machine does with what it can do?*

These two questions define two big components of computer science *Computability and computational complexity.*

Mathematics has actually given a strong contribution and powerful tools to study the fundamentals of computability. In this sense the main support to Computer Science has been in making available a deep understanding of algorithmic mechanisms. Just to substantiate this statement think to automata theory.

The obvious lake of this component of computer science concerns the efficiency considerations of a computation. And the entire component of computational complexity represents in fact a feedback from Computer Science to Mathematics. The relevance of this area comes from the capability of creating a taxonomy of algorithms at different levels of abstraction in order to design and execute an algorithm in an *effective* and *efficient* way.

We do like to present in this paper some considerations on the relation between mathematics and computer science in the particular area of computer algebra. In fact we present first the power of algebraic approaches and techniques in algorithms designing. Then we also introduce to the problem of algorithms analysis. Two ways will be underlined to study and reduce the complexity of an algorithm: the reducibility of a problem to a simpler problem and the reducibility of a problem to other problems already known.

The first part essentially gives the basic current trend in what can be done in algebraic computation saving all the power and the structure of the modern algebra.

The second and third parts concern the way how to obtain efficient algorithms. Also in order to show that the hierarchical structure, which is typical in algebra, is again alive from a computational point of view.

Altogether these considerations bring to a whole complete know-how which is a strong methodology in computer algebra.

This property of computer algebra, which is typical in mathematics, becomes also a property for the computer science itself and it can encourage successful applications of computers in other areas with poor theoretical ground works.

2. Algebraic structures and their algorithms. The area of Symbolic and Algebraic computations [2] has received a wide attention in the recent past.

Many algebraic algorithms have been defined and analyzed.

The success of this area of researches is essentially based on the very deep understanding of the relations between algebra and computer science. In fact the proper representations of algebraic data (such as polynomials, power series, etc.) and of their algebraic structures (such as rings, field, etc.) have allowed very natural and simple "algorithms driven by representation". And all that matter closely links computer science to modern mathematics.

Unfortunately the implications of these results in terms of definition of general methodologies in computer science are only now under consideration.

Let us present three different ways of looking at integers. And let us also show how these representations of integers lead to similar representations for polynomials making a unifying view of algebraic data [3;4].

One classical way to represent integers is the so-called fix radix. An integer u will be written as $u=\Sigma_{i=0}^{n-1} u_i x^i$ with $0 \leq u_i < x$ for all i, and $u_{n-1} \neq 0$, where x is also a number and it is the base of the representation, and it is fixed in the computer.

A second way of representing integers is the so called p-adic, or variable radix. An integer u will be represented, in a similar way, as $u=\Sigma_{i=0}^{n-1} u_i p^i$ with $0 \leq u_i < p$ for all i, and $u_{n-1} \neq 0$ where p is generally a prime number variable from one problem to another.

Third is the modular representation, or variable mixed radix.

Here an integer u is mapped into a set of integers u_i such that $u_i = u$ mod p_i.

The values u_i are called residues in respect to the set of prime numbers p_i, $u \longleftrightarrow (u_0, u_1, \ldots, u_{n-1})$.

Now, if we look at polynomials we can say that the domain of polynomials has exactly the same properties as integers.

In fact one can simply write polynomial in the classical form $f(x)=\Sigma_{i=0}^{d} f_i \; x^i$ with $f_i \in F$, F is a field and d is the degree of the polynomial.

In the p-adic representation we can write the polynomial as $f(x)=\Sigma_{i=0}^{k} u_i(x) p^i(x)$ which essentially is a series representation with respect to an arbitrary irreducible polynomial $p(x)$, usually chosen as $p(x)=x-b$.

Finally according to the modular representation a polynomial can be mapped into a set of polynomials $u_i(x)$

$$f(x) \longleftrightarrow (u_0(x), u_1(x), \ldots)$$

such that $u_i(x)=f(x)$ mod $p_i(x)$, where $p_i(x)$ are irreducible polynomials. In the past, even if this similarity was known, different programs for different algebraic domains have been implemented. In fact very large algebra systems such as MACSYMA [5] and SCRATCHPAD [6] don't benefit from abstract properties of general algebraic domains.

In the present time new approaches start to be followed [7;8;9]. The concept of *categories* as collections of algebraic domains (for instance the category of integral domains: integers, polynomials) is introduced.

You can build new category, such as field, using other known categories, such as group.

And also concept of *functors* as operators which *instantiate* a given category into a specific algebraic domain.

So you don't have to write different algorithms for *similar operations*, but the powerful structure that you have introduced, with its hierarchical ordering, enables you to design algorithms in a very efficient way.

3. A general approach in algebraic algorithms designing.

A typical approach in algebraic algorithms designing is based on the reducibility of a given problem to simpler problems.

This reduction is generally obtainable in an easy way for algebraic structures.

Let us consider the problem of computing multivariate polynomials greatest common divisor (GCD) [10]:

$$D(x_0, \ldots, x_n) = \mathrm{GCD}\,(F(x_0, \ldots, x_n), G(x_0, \ldots, x_n)).$$

It is well-known (but also very intuitive) that an algorithm for GCD computation can be defined recursively applying the classical Euclid's algorithm to polynomials with one less variable at the time. This algorithm comes out obviously to be exponential in the number of variables.

Let us now map the two given polynomials F and G into two univariate polynomials $\bar{F}(x)$ and $\bar{G}(x)$ by evaluating F and G such that:

$$\bar{F}(x) = F(x_0, b_1, \ldots, b_n), \bar{G}(x) = G(x_0, b_1, \ldots, b_n)$$

for given values b_i.

In other words, we do map our GCD problem from the domain $Z[x_0, \ldots, x_n]$ of multivariate polynomials into the domain $Z[x]$ of univariate polynomials, by the evaluation homomorphism.

Now these two polynomials $\bar{F}$ and $\bar{G}$ can be also mapped into other polynomials using the modular homomorphism.

Therefore we do have now the problem of computing the GCD of polynomials in a domain of polynomials with coefficients which are integers modulo some given primes $p_i : Z_{p_i}[x]$.

Once we have solved this problem in $Z_{p_i}[x]$, we do have to come back to the original domain $Z[x_0, \ldots, x_n]$. We do that by inverting the two homomorphisms, namely using first the Chinese remainder algorithm and then the interpolation algorithm.

What we have done is expressed by the following scheme:

$$\begin{tikzcd}
\boxed{\begin{array}{c}\mathrm{GCD}\,(F(x_0, \ldots, x_n)\ G(x_0, \ldots, x_n)) \\ \text{in } Z\,[x_0, \ldots, x_n]\end{array}} \arrow[d, shift left=4, "\text{evaluation}"] \\
\boxed{\mathrm{GCD}\,(\overline{F}(x), \overline{G}(x)) \text{ in } Z\,[x]} \arrow[u, shift left=4, "\text{interpolation}"] \arrow[d, shift left=4, "\text{modular}"] \\
\boxed{\mathrm{GCD}\,(\overline{F}_{p_i}(x), \ \overline{G}_{p_i}(x)) \text{ in } Z_{p_i}\,[x]} \arrow[u, shift left=4, "\text{CRA}"]
\end{tikzcd}$$

This approach to the GCD problem is an example of a general procedure which can be proposed following these steps:

— simplify the input by invertible algebraic homomorphisms

— solve the given problem in a simpler way in the image domain

— reconstruct the true solution in the original domain by inverse homomorphisms.

It is very important to recall how this fundamental approach leads to definitions of algorithms with lower complexity. Typically one can reduce the complexity from exponential to polynomial rates. For instance this is in fact the case of GCD and factorization problems [11].

This remark contributes to use these kinds of techniques in many other potential application areas. Besides it must also be underlined that uniform bounds for a class of algebraic invertible mappings have been established in the recent past [12].

4. A general approach in algebraic algorithms analysis. A current approach in algorithms analysis is based on the so-called reducibility a la Karp [13]. According to this approach one problem is shown to be reducible to another, so that any computational property of an algorithm for the second can be assumed to hold also for the first. And also any improvement of the second algorithm will be useful for the first.

As far as the algebraic algorithms are concerned this kind of algorithm analysis has been for instance used to show that the algebraic operations of multiplying, dividing, squaring and reciprocating, for objects with the same size, are equivalent problems [3].

Hence if we indicate with:

$M(n)$ the cost to multiply two objects of size n,

$D(n)$ the cost to divide an object of size $2n$ by an object of size n,

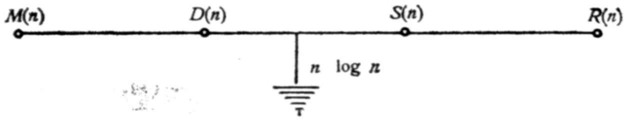
$S(n)$ the cost to square an object of size n,

$R(n)$ the cost to reciprocate an object of size n,

we do have $M(n)=D(n)=S(n)=R(n)$.

Let us recall that here an object of size n means an element of an algebraic structure (for example an integral domain), for which a valuation or size can be defined. For instance the absolute value is a size for an integer, or the degree represents a size for a polynomial.

It is well known that the cost of multiplying two objects of size n (integers, polynomials) is $n \log n$ [4, 14]. Therefore we say that the above four operations can be done in the same amount of costs. And we could express this result by assuming that all the above operations are at the same level with a distance from the ground of $n \log n$



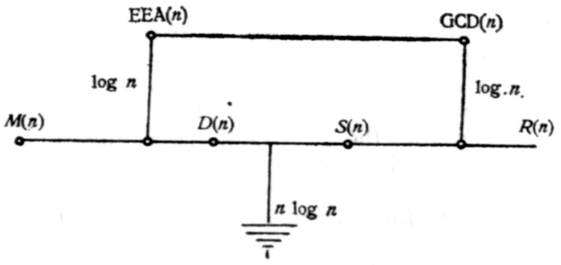
Similar approach has been followed in other algorithm analysis.

It has been shown by Schonhage [15] and Moench [14] that the GCD algorithm (of objects of size n) in an arbitrary Euclidean domain has a cost of $n \log^{a+1} n$, if in the domain a multiplication of two elements of size n can be accomplished in $n \log^a n$ time. That means that the GCD algorithm is $\log n$-reducible to a multiplication algorithm. Therefore, in our description the GCD algorithm has a level higher than the level of multiplication with a distance of $\log n$.

Moench has also shown that in an arbitrary Euclidean domain the Extendend Euclidean Algorithm (EEA) [4] for solving the equation (given a and b)

$$as+bt=\text{GCD}\ (a, b)$$

computing s and t, has a cost of $\log n\ (M(n))$ [14]. Therefore EEA (n) also has a distance $\log n$ from $M(n)$. Besides, the proposed algorithm also computes the GCD (a, b), therefore we also have that GCD algorithm is reducible to the EEA.



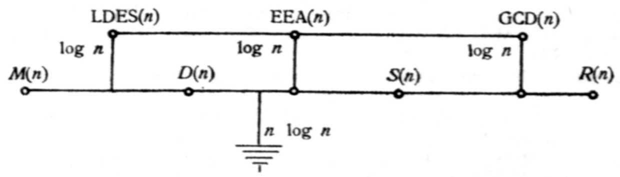
A generalization of this result, due to Yun [16; 17], shows that the socalled Linear Diophantine Equation Solver (LDES) is an algorithm equivalent to the EEA.

A linear diophantine equation is

$$au+bv=c$$

for given a, b, c. We can solve this equation for u and v, with the conditions size $(u)<$size (b), size $(v)<$size (a) iff size $(c)<$max (size (a), size (b)).

Under these conditions we implicitly also have that LDES (n) is $\log n$ reducible to $M(n)$. Thus



Yun has also studied the relations among algorithms for polynomial Square-Free decomposition (SQFR) and for rational functions integration (RFINT).

A polynomial $P(x)=\Sigma_{i=0}^{n} a_i x^i$ in a domain $D[x]$ is primitive if the GCD of all the coefficients a_i is 1 in the domain D.

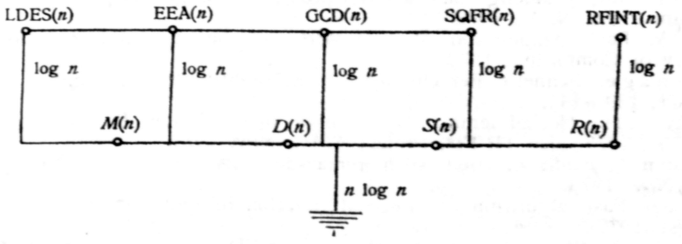
A primitive polynomial $P(x)$ in $D[x]$ is square-free if GCD $(P(x), P'(x))=1$.

Then the square-free decomposition of the given polynomial $P(x)$ is $P(x) = \Pi_{i=1}^{k} P_i^i(x)$, where each $P_i(x)$ is square-free in $D[x]$ and they are pairwise relatively prime.

It has been demonstrated that also this problem is log n-reducible to $M(n)$ [18]. Moreover this problem is equivalent to the GCD problem [19].

Similarly the problem of integrating a proper rational function $A(x)/B(x)$, (with $A(x)$, $B(x)$ in $D[x]$) is shown to be log n-reducible to $M(n)$ [18]. But the equivalence of RFINT algorithm and GCD algorithm is still an open problem.

Therefore, our diagram can be now completed as follows, where the horizontal links represent the equivalence of the linked problems.



Other similar interesting results can be found in [1]. All these results allow to define a structure of the algebraic algorithms according to their computational properties. This structure comes from a partial ordering of the algorithms which enables to study and deeply understand their computational and their algebraic properties in a unified perspective.

Beside that, the proposed diagram (which can be enlarged to include other known results) immediately underline the open problems.

REFERENCES

1. — Proceedings of I Symposium on Symbolic and Algebraic Manipulation. *Comm. ACM*, **9**, 1966, n. 8.
— Proceedings of II Symposium on Symbolic and Algebraic Manipulation. Los Angeles, 1971.
— Proceedings of EUROSAM'74. *ACM SIGSAM Bulletin*, **8**, 1974, n. 3.

— Proceedings of Symposium on Algebraic Computation SYMSAC'76 (1976).
— Proceedings of 1977 MACSYMA User's Conference. MIT. Laboratory for Computer Science, 27—29 July (1977).
— Atti del Seminario di Introduzione alla Manipolazione Algebrica. Istituto per le Applicazioni del Calcolo, Roma, 20—24 March (1978).
— Proceedings of 1979 MACSYMA User's Conference. MIT, Laboratory for Computer Science, 20—22 June (1979).
— Proceedings of EUROSAM'79. Marseille,24—26 June. *Lecture Notes in Computer Sci.* **72**.
— Proceedings of SYMSAC'81. Snowbird (Utah), August (1981).

2. A. M i o l a. Symbolic and Algebraic Manipulation Today — Systems, Algorithms, Applications. Report of Università di Roma, CSSCCA R. 80—10 (1980).
3. A. V. A h o, E. H o p c r o f t. J. D. U l l m a n. The design and analysis of computer algorithm. Addison-Wesley, 1974.
4. D. K n u t h. The art of computer programming. Vol. 2. Addison-Wesley, Reading, 1974.
5. R. B o g e n et al. MACSYMA Manual. MIT-PROJECT MAC. Cambridge, 1973.
6. J. H. G r i e s m e r et al. The SCRATCHPAD Manual. IBM Report, Yorktown Heights, N. Y., 1975.
7. G.A u s i e l l o, G. F. M a s c a r i. On the Design of Algebraic Data Structures with the Approach of Abstract Data Types. *Lecture Notes in Computer Sci.*, **72**.
8. R. D. J e n k s. MODLISP: An Introduction. *Lecture Notes in Computer Sci.*, **72**, 466—480 (also available is revised form as: "MODLISP: A Preliminary Design", IBM Research Report RC 8073, January 18, 1980).
9. R. J e n k s, B. M. T r a g e r. A language for computational algebra. SYMSAC '81, Snowbird (Utah), August (1981).
10. W. S. B r o w n (3). On Euclid's Algorithm aud the Computation of Polynomial Greatest Common Divisor. *Communications of the ACM*, **18**, 1971, 478—504.
11. P. W a n g, L. R o t h s c h i l d (22). Factoring Multivariate Polynomials Over the Integers. *Math. of Comp.*, **29**, 1975, 935—950.
12. D. Y. Y u n. Uniform bounds for a class of algebraic mappings. *SIAM J. of Comp.*, **8**, August 1979.
13. R. K a r p. Rieucibility among combinatorial problems, complexity of computer computations. Plenum Press, N. Y., 1972.
14. R. M o e n c k. Fast computation of GCD. Proc. of the 5th Annual ACM Symposium on Theory of Computing, 1973.
15. A. S c h o n h a g e. Schnelle Berechnung von Kettenbruchentwicklungen. *Acta Informatica* **1**, 1971, 139—144.
16. D. Y. Y. Y u n. The Hensel lemma in algebraic manipulation. Ph. D. Thesis, Math. Dept., M. I. T., 1973, also TR-138, Project MAC, Nov. 1974.
17. D. Y. Y. Y u n. A p-adic division with remainder algorithm. *ACM SIGSAM Bulletin*, **8**, n. 3, Nov. 1974.
18. D. Y. Y. Y u n. Fast algorithm for rational function integration. Proc. of IFIP' 77. Toronto' August (1977).
19. D. Y. Y. Y u n. On the equivalence of polynomial GCD and squarefree factorization problems. Proc. MACSYMA users' conference 1977, NASA, Washington D. C.

Istituto di Analisi dei Sistemi
ed Informatica del C. N. R.
Viale Manzoni 30 Buonarroti 12
00185 Roma, Italy

Received 4.1.1983