Journal of Number Theory 130 (2010) 232–253

Contents lists available at ScienceDirect

Journal of Number Theory

www.elsevier.com/locate/jnt

Computing partial information out of intractable: Powers of algebraic numbers as an example

Mika Hirvensalo [a,*,1], Juhani Karhumäki [a,1], Alexander Rabinovich [b]

[a] TUCS – Turku Centre for Computer Science and Department of Mathematics, University of Turku, FIN-20014 Turku, Finland
[b] Tel Aviv University, School of Computer Science, Ramat Aviv, Tel Aviv 69978, Israel

ARTICLE INFO

Article history:
Received 25 August 2007
Revised 10 August 2009
Communicated by David Goss

Keywords:
Efficient computability
Linear forms of logarithms
Baker theory

ABSTRACT

We consider an algorithmic problem of computing the first, i.e., the most significant digits of 2^n (in base 3) and of the nth Fibonacci number. While the decidability is trivial, efficient algorithms for those problems are not immediate. We show, based on Baker's inapproximability results of transcendental numbers that both of the above problems can be solved in polynomial time with respect to the length of n. We point out that our approach works also for much more general expressions of algebraic numbers.

1. Introduction

Algorithmic mathematics is a relatively new but rapidly growing research topic. Algorithmic properties of simple arithmetic questions can be very challenging. An example is the question whether the factorization of a number can be computed in polynomial time.

We consider here a related, but much simpler, algorithmic problems. In the simplest setting we want to compute the first, i.e., the most meaningful, digit of 2^n in base 3. From the classical point of view the problem is trivial: given n, compute the value 2^n and output the first digit. When doing this we are computing the value of the function which is among the simplest in classical mathematics, an exponent function.

However, if we want to answer our problem in practice, we are in trouble. The number of digits needed to print 2^n exhausts very soon the amount of paper available. Indeed, the length of 2^n

* Corresponding author.
E-mail addresses: mikhirve@utu.fi (M. Hirvensalo), karhumak@utu.fi (J. Karhumäki), rabinoa@post.tau.ac.il (A. Rabinovich).
URLs: http://www.math.utu.fi/en/personnel/?userID=244 (M. Hirvensalo), http://www.math.utu.fi/en/home/karhumak/ (J. Karhumäki), http://www.math.tau.ac.il/~rabinoa/ (A. Rabinovich).

[1] Supported by the Academy of Finland under grant 44087.

0022-314X/$ – see front matter © 2009 Elsevier Inc. All rights reserved.
doi:10.1016/j.jnt.2009.08.009

(in base 3) is exponential in terms of the length of the input, i.e., of the representation of number n.

An interesting question arises. Does the above problem allow a polynomial time solution? This can be interpreted as a question whether we can compute some partial information, that is the first digit, in polynomial time, while the whole information, the value 2^n, surely requires exponential time.

We give a detailed solution to this problem. Actually our algorithm is quite expected and straightforward to find. The problematic thing is to prove that it works correctly. We reduce this to some known, deep results on inapproximability of transcendental numbers by rationals. Consequently, our method is not specific to the function 2^n, but works for powers of all real algebraic numbers. A small modification makes it to work also for Fibonacci numbers, which extends our previous work [7]. Neither it is important whether we ask for the first digit. It is not essentially more difficult to find the first k digits of the nth power. On the other hand, our method would fail for the value of the middle digit.

Both 2^n and Fibonacci numbers can be defined as linear recurrence sequences with integer coefficients. It is worth emphasizing that computing the *least meaningful* digits of any linear recurrence $L(n)$ with integer coefficients is an easy task: any fixed number k of the least digits form an ultimately periodic sequence, and the period can be found algorithmically. It follows that there is a linear time algorithm for computing the (fixed) k least digits of $L(n)$.

From our result it follows that there is a polynomial time algorithm A_L for computing the (fixed) k most significant digits of $L(n)$, if the characteristic polynomial of $L(n)$ has a dominant real root. We also show how to design A_L when $L(n)$ is given, but notice carefully that we do not claim that A_L can be found in polynomial time from the description of $L(n)$, only that A_L itself is a polynomial time algorithm.

This paper is organized as follows: in Section 2 we give the necessary definitions and preliminaries associated with this paper. In Section 3 we present one of the basic building blocks of the main result, an algorithm for approximating the logarithm of a natural numbers. Obviously the algorithm in Section 3 is well known, but for the sake of completeness, we present it here, with a complexity analysis.

In Section 4 we concentrate on finding rational approximations of the logarithms of real algebraic numbers. It is pointed out that, given a description of an algebraic number, an exponentially precise rational approximation of the logarithm can be found in polynomial time. In the end of this section, we also give some upper bounds on the description size of a real algebraic number. These upper bounds are not needed for continuation, but we expect that some readers may be interested on them, so we included those bounds in this representation.

In Section 5 we discuss the relationships between the previous issues and linear recurrences. In Section 6 we give a simple example: 2^n in ternary basis. In that section, we also point out which are the difficulties of an obvious algorithm for finding the length of 2^n (presented in ternary). Section 7 is the most important for our main results. In that section we show, using the results of Alan Baker, that the algorithms we present work correctly.

Finally, in Section 8 we show how to apply our method for Fibonacci numbers. To conclude, in Section 9, we show how our algorithm for computing the length of a given expression generalizes to computing a fixed number of the most significant digits of that expression and discuss about some extensions.

The polynomial time algorithms presented here are practically useless due to the large constants. On the other hand, it is possible to modify them into a useful form, but the complexity analysis is much simpler in the form we present here.

2. Preliminaries and notations

By a linear recurrence $L(n)$ we mean a sequence of integers defined by giving fixed integers $L(0), L(1), \ldots, L(k-1)$, and for $n \geqslant 0$, $L(n+k)$ is defined by equation $L(n+k) + c_1 L(n+k-1) + c_2 L(n+k-2) + \cdots + c_k L(n) = 0$, where $c_1, \ldots, c_k$ are fixed integers. The *characteristic polynomial* of recurrence $L(n)$ is $x^k + c_1 x^{k-1} + c_2 x^{k-2} + \cdots + c_k$.

For any real number x, notation $\lfloor x \rfloor$ stands for the largest integer M for which inequality $M \leqslant x$ holds, and $\log_d x$ stands for the d-ary logarithm of x. We also use standard notation $\ln x$ for the natural logarithm of x.

Let $d > 1$ be an integer and M a natural number. Each natural number M admits a unique representation as

$$M = a_{\ell-1}d^{\ell-1} + a_{\ell-2}d^{\ell-2} + \cdots + a_1 d^1 + a_0, \tag{1}$$

where $a_i \in \{0, 1, \ldots, d-1\}$ and $a_{\ell-1} \neq 0$. The word $a_{\ell-1}a_{\ell-2}\ldots a_0$ is called the *d-ary representation of M*, and denoted by $M_{\mathbf{d}}$. In the case $d = 3$ we say that $M_{\mathbf{3}}$ is the *ternary* representation of M.

The *length* of d-ary representation of M is defined to be the length of word $M_{\mathbf{d}}$ and denoted by $|M_{\mathbf{d}}|$. Notice carefully that we use boldface subscripts to separate between the numbers and their representations. Especially, $|M|$ stands for the *absolute value* of M whereas $|M_{\mathbf{d}}|$ stands for the length of the word representing M.

Eq. (1) gives easily inequalities $M \geqslant d^{\ell-1}$ and

$$M \leqslant (d-1)\big(d^{\ell-1} + d^{\ell-2} + \cdots + d + 1\big) = d^\ell - 1 < d^\ell.$$

Combining these two estimates we see that $d^{\ell-1} \leqslant M < d^\ell$, or, equivalently, $\ell - 1 \leqslant \log_d M < \ell$, which is to say that $|M_{\mathbf{d}}| = \ell = \lfloor \log_d M \rfloor + 1$.

If x is a positive real number, we can write x uniquely as $x = M + y$, where M is an integer and $y \in [0, 1)$. We say that M is the *integer part* of x. Also, each real number can be represented as

$$x = a_{\ell-1}d^{\ell-1} + a_{\ell-2}d^{\ell-2} + \cdots + a_1 d + a_0 + a_{-1}d^{-1} + a_{-2}d^{-2} + \cdots, \tag{2}$$

where $a_i \in \{0, 1, \ldots, d-1\}$ and $a_{\ell-1} \neq 0$. On the other hand, the representation (2) is not necessarily unique. For instance,

$$1 = \frac{9}{10} + \frac{9}{10^2} + \frac{9}{10^3} + \cdots$$

and

$$1 = 1 + \frac{0}{10} + \frac{0}{10^2} + \frac{0}{10^3} + \cdots.$$

Yet we can get an analogous result concerning the length of the integer part: if x is given as in (2), and there exists an $i > 0$ such that $a_{-i} \neq d - 1$, then

$$a_{-1}d^{-1} + a_{-2}d^{-2} + a_{-3}d^{-3} + \cdots < (d-1)\left(\frac{1}{d} + \frac{1}{d^2} + \frac{1}{d^3} + \cdots\right) = 1,$$

and it follows that $M = a_{\ell-1}d^{\ell-1} + a_{\ell-2}d^{\ell-2} + \cdots + a_1 d + a_0$ is the integer part of x (having length ℓ). Then we can estimate x as

$$d^{\ell-1} \leqslant x < (d-1)\big(d^{\ell-1} + d^{\ell-2} + \cdots + d + 1\big) + 1 = d^\ell,$$

hence $\ell = \lfloor \log_d x \rfloor + 1$ is the length of the integer part of x. On the other hand, if $a_{-i} = d - 1$ for each $i \in \{1, 2, 3, \ldots\}$, then

$$a_{-1}d^{-1} + a_{-2}d^{-2} + a_{-3}d^{-3} + \cdots = (d-1)\left(\frac{1}{d} + \frac{1}{d^2} + \frac{1}{d^3} + \cdots\right) = 1$$

and, consequently, $x = M$ is an integer and we can apply the aforementioned formula for the integer part of x.

A polynomial $q(x) = x^n + a_{n-1}x^{n-1} + \cdots + a_1x + a_0$ with rational coefficients is the *minimal polynomial* of an algebraic number λ if (1) λ is a root of $q(x)$ and (2) if λ is a root of a (nontrivial) polynomial $p(x)$ with rational coefficients, then the degree of $p(x)$ is at least n. Any algebraic number λ (over $\mathbb{Q}$) has a unique minimal polynomial $q(x)$ belonging to $\mathbb{Q}[x]$. If $c > 0$ is the least common multiple of the nominators of the coefficients of $q(x)$, we say that $cq(x)$ is the *defining polynomial* of λ. Clearly the defining polynomial has integer coefficients and the same degree as the minimum polynomial. That degree is also called the *degree* of λ and denoted as $\deg(\lambda)$. If $p(x) = a_nx^n + a_{n-1}x^{n-1} + \cdots + a_1x + a_0$ is the defining polynomial of λ, then the *height* of λ is defined to be $H(\lambda) = \max\{|a_0|, \ldots, |a_n|\}$. The *size* of λ is defined as $S(\lambda) = |(a_n)_{\mathbf{d}}| + |(a_{n-1})_{\mathbf{d}}| + \cdots + |(a_0)_{\mathbf{d}}| + S(I)$, where d is some fixed base in which the coefficients are represented, and $S(I)$ stands for the size of the additional information which specifies uniquely one of the roots of $p(x)$. In the later sections, we specify a real root of a polynomial by giving an interval which contains only one root of the polynomial.

For instance, if $r = \frac{a}{b}$ is a nonzero rational number such that $\gcd(a, b) = 1$, the defining polynomial of r is $bx - a$, $H(r) = \max\{|a|, |b|\}$, and $S(r) = |a_{\mathbf{d}}| + |b_{\mathbf{d}}|$. In this case, an additional information to specify the root is of course unnecessary.

As defined above, the size of an algebraic number depends on the number system in which the coefficients are represented. On the other hand, since $\log_d M < |M_{\mathbf{d}}| \leqslant \log_d M + 1$ holds for any integer M, and because $\log_d M = (\ln d)^{-1} \ln M$, we see that the length of M is always $\Theta(\ln M)$. Here and hereafter, we exclude the unary representations.

3. Approximating the logarithm of a natural number

The issues treated in this section are very straightforward, but an interesting consequence can be found: recall that there is an efficient (polynomial time) algorithm for *modular exponentiation*, whereas it is widely believed that there is no polynomial-time algorithm for computing *modular logarithms* (usually referred as to *discrete logarithms*). On the other hand, *exponentiation on natural numbers* is certainly not computable in polynomial time, since the result requires exponential space. In this section we point out that the *logarithm of a natural number* can be computed efficiently in the sense that it is possible to have exponentially precise approximations in polynomial time. Therefore, it can be thought that, with respect to exponentiation and logarithm extraction, easy–difficult setup is turned upside down when moving from natural number computations to modular computations.

A polynomial time algorithm for computing exponentially precise approximations of the logarithm of a natural number is based on power series of the logarithm function. It is a well-known fact that the series

$$\ln(1 + x) = x - \frac{1}{2}x^2 + \frac{1}{3}x^3 - \frac{1}{4}x^4 + \cdots$$

converges if $|x| < 1$. By substituting $x = -y$ we see that

$$\ln \frac{1}{1 - y} = -\ln(1 - y) = y + \frac{1}{2}y^2 + \frac{1}{3}y^3 + \frac{1}{4}y^4 + \cdots,$$

and a further substitution $y = \frac{\alpha - 1}{\alpha}$ shows us that

$$\ln \alpha = \sum_{i=1}^{\infty} \frac{1}{i} \left(\frac{\alpha - 1}{\alpha} \right)^i \tag{3}$$

converges for any fixed $\alpha > 1$. Hereafter we assume that $\alpha > 1$ is a fixed integer.

The remainder of (3) can be easily estimated as

$$\left| \sum_{i=M+1}^{\infty} \frac{1}{i}\left(\frac{\alpha-1}{\alpha}\right)^i \right| < \left(\frac{\alpha-1}{\alpha}\right)^{M+1} \sum_{i=0}^{\infty}\left(\frac{\alpha-1}{\alpha}\right)^i = \alpha\left(\frac{\alpha-1}{\alpha}\right)^{M+1}.$$

Thus, by choosing

$$M > 1 + \frac{\ln\frac{\alpha}{\epsilon}}{\ln\frac{\alpha}{\alpha-1}} \tag{4}$$

the remainder is less than ϵ.

In the case $\epsilon = \frac{1}{n^T}$ we see that in order to approximate $\ln\alpha$ within precision $\frac{1}{n^T}$, it is sufficient to take

$$M = 1 + \frac{T\ln n + \ln\alpha}{\ln\frac{\alpha}{\alpha-1}} = \Theta(T\ln n)$$

first summands of the series (3). Notice that all the summands in the series are positive, which implies that an approximation obtained by taking M first terms is always smaller than the actual value of the logarithm.

Let us choose $M = \Theta(T\ln n)$. Then, by writing

$$\sum_{i=1}^{M} \frac{1}{i}\left(\frac{\alpha-1}{\alpha}\right)^i = \frac{1}{M!\alpha^M} \sum_{i=1}^{M} \frac{M!}{i}\alpha^{M-i}(\alpha-1)^i \tag{5}$$

we see that the nominator of (5) is at most $M!\alpha^M$, and its length is of order

$$\begin{aligned}\ln\left(M!\alpha^M\right) &= M\ln M + O(M) + M\ln\alpha = O(M\ln M)\\ &= O\left(T\ln n\ln(T\ln n)\right) = O(\ln n\ln\ln n),\end{aligned}$$

assuming T fixed. An estimate for the numerator can be found in a similar way: first, a trivial estimate shows that the numerator is at most $M\cdot M!\alpha^M(\alpha-1)^M$, the length of which is of order

$$\ln M + M\ln M + O(M) + M\ln\alpha + M\ln(\alpha-1) = O(M\ln M) = O(\ln n\ln\ln n)$$

similarly as in the case of the nominator.

Number $M!$, or, to be precise, its representation, can be computed in time $O(M(\ln M!)^2) = O(M^3\ln^2 M) = O(\ln^3 n(\ln\ln n)^2)$. On the other hand, number α^M can be found in time $O(M\cdot(M\ln\alpha)^2) = O(M^3) = O(\ln^3 n)$, and the product $M!\alpha^M$ in time $O(\ln^2 M!) = O((\ln n\ln\ln n)^2)$. It follows that the nominator of (5) can be computed in time $O(\ln^3 n(\ln\ln n)^2)$. A similar estimate can be found also for the numerator.

In the above estimations we used only the most well-known algorithms for the arithmetical operations (e.g. the quadratic algorithm for multiplication). Consequently, the estimates could be improved by using more efficient methods.

As a conclusion, we get the following theorem.

Theorem 1. *Let $\alpha > 1$ and T be fixed natural numbers. For every natural n it is possible to compute a rational number $q < \ln\alpha$ such that $\ln\alpha - q < \frac{1}{n^T}$ in time $O(\ln^3 n(\ln\ln n)^2)$. Moreover, the size of q is $O(\ln n\ln\ln n)$.*

Now, a rational approximation for $\log_d \alpha = \frac{\ln \alpha}{\ln d}$ can be found by finding rational approximations r and s for both $\ln \alpha$ and $\ln d$, respectively.

Assume that $\ln \alpha = r + \epsilon_1$ and $\ln d = s + \epsilon_2$, where $0 < \epsilon_1, \epsilon_2 \leqslant \epsilon$. If we further assume that s and r are so good approximations that $s \geqslant \frac{1}{2}$ (recall that $d \geqslant 2$ since we do not consider unary cases) and $r > 0$ holds, we can estimate their ratio as

$$\left| \frac{\ln \alpha}{\ln d} - \frac{r}{s} \right| = \frac{|s \ln \alpha - r \ln d|}{s \ln d} \leqslant \frac{2|\epsilon_1 s - \epsilon_2 r|}{\ln d}.$$

If $\epsilon_1 s - \epsilon_2 r \geqslant 0$, then $|\epsilon_1 s - \epsilon_2 r| = \epsilon_1 s - \epsilon_2 r \leqslant \epsilon_1 s < \epsilon \ln d$, whereas $\epsilon_1 s - \epsilon_2 r < 0$ implies $|\epsilon_1 s - \epsilon_2 r| = \epsilon_2 r - \epsilon_1 s \leqslant \epsilon_2 r < \epsilon \ln \alpha$. In both cases,

$$\left| \frac{\ln \alpha}{\ln d} - \frac{r}{s} \right| \leqslant \frac{2|\epsilon_1 s - \epsilon_2 r|}{\ln d} \leqslant \frac{2\epsilon \max\{\ln d, \ln \alpha\}}{\ln d} = K\epsilon,$$

where $K = \frac{2 \max\{\ln d, \ln \alpha\}}{\ln d}$ does not depend on s or r. Hence we get the following corollary.

Corollary 2. *Let $\alpha > 1$, $d > 1$ and T be fixed natural numbers. For every natural n it is possible to compute in time $O(\ln^3 n (\ln \ln n)^2)$ a rational number q such that $|\log_d \alpha - q| < \frac{1}{n^T}$. Moreover, the size of q is $O(\ln n \ln \ln n)$.*

4. Approximating the logarithms of algebraic numbers

Approximations of the logarithms of algebraic numbers are based on computing logarithms of their rational approximations. Because of the continuity of the logarithm and the triangle inequality, we can find good approximations of logarithms of algebraic numbers, too. Good approximations of algebraic numbers are needed when outlining an algorithm for computing the length of Fibonacci numbers, for instance.

First we have to agree on how to describe a real algebraic number. Algebraic numbers of degree 1, i.e., rational numbers are of course described by simply giving the numerator and nominator, so we can assume that the degree is at least 2.

Definition 3. A *description* of a real, irrational algebraic number λ (of degree at least 2) consists of a polynomial $P(x) \in \mathbb{Z}[x]$ having λ as a root, together with two rational numbers $\mu_0 < \nu_0$ such that $\lambda \in [\mu_0, \nu_0]$ and λ is the only zero of $P(x)$ in the interval $[\mu_0, \nu_0]$.

Remark 4. The theorem of Sturm [3,8] provides an algorithm for determining the number of roots of a given polynomial in the interval $[\mu_0, \nu_0]$. Therefore we can actually decide, whether a given polynomial and an interval are indeed an acceptable description of a real algebraic number.

It is quite straightforward to find rational approximations for λ: as a first approximation, we set $\lambda_0 = \mu_0$, and can assume, without loss of generality, that $P(\mu_0) < 0$, $P(\nu_0) > 0$. Further approximations are obtained by *binary search*: if $P(\frac{1}{2}(\mu_i + \nu_i)) > 0$, we define $\mu_{i+1} = \mu_i$, $\nu_{i+1} = \frac{1}{2}(\mu_i + \nu_i)$, and $\lambda_{i+1} = \mu_{i+1}$. On the other hand, if $P(\frac{1}{2}(\mu_i + \nu_i)) < 0$, we define $\mu_{i+1} = \frac{1}{2}(\mu_i + \nu_i)$, $\nu_{i+1} = \nu_i$, and $\lambda_{i+1} = \mu_{i+1}$. It is clear that for each $i \geqslant 0$, $\lambda \in [\mu_i, \nu_i]$, and $P(\mu_i) < 0$, $P(\nu_i) > 0$, and that λ is the only zero of $P(x)$ in the interval $[\mu_i, \nu_i]$. Denoting $d_i = \nu_i - \mu_i$ it is clear that $d_i = \frac{d_0}{2^i}$ for each $i \geqslant 0$, which implies that $|\lambda_i - \lambda| \leqslant \frac{d_0}{2^i}$.

It is worth mentioning a few words about the complexity of computing the approximations of λ. First, since each polynomial has only finitely many zeros, numbers μ_0 and ν_0 exist. Moreover, the coefficients and the degree d of the polynomial having λ as zero together with numbers μ_0 and ν_0 (no matter how they are chosen) are regarded as constants. Therefore, if q is a rational number having size L, $P(q)$ can be computed in time $O(L^2)$. Assume then that rational numbers p and q have sizes

at most L. A simple calculation shows that the average $\frac{1}{2}(p+q)$ has size at most $L+2$, which means that after i iterations, the size of the approximating λ_i has size at most $L+2i$.

Lemma 5. *Assume that a description of a real algebraic number λ and a natural number T is given. Given n as an input, there exists a polynomial time algorithm computing a rational number r such that*

$$|r-\lambda| \leqslant \frac{1}{n^T}.$$

Moreover, the size of r is polynomial in $\ln n$.

Proof. We only need to estimate how many iterations are needed to reach the required precision. As seen previously, $|\lambda_i - \lambda| \leqslant \frac{d_0}{2^i}$, and $\frac{d_0}{2^i} = \frac{1}{n^T}$, when $i = T\log_2 n + \log_2 d_0 = O(\ln n)$. □

Remark 6. Notice that the above algorithm is polynomial also in the description size of the given algebraic number, not only in $\ln n$.

Lemma 7. *Given an algebraic number $\lambda > 1$, natural number T, and input n, there exists a polynomial-time algorithm which computes a rational number s such that*

$$|s - \ln\lambda| \leqslant \frac{1}{n^T}.$$

Moreover, number s has polynomial size in $\log n$.

Proof. By the previous lemma, we can compute in polynomial time an approximation r of λ such that $|r-\lambda| \leqslant \frac{1}{2n^T}$. Without loss of generality, we can assume $1 \leqslant r < \lambda$. By the mean-value theorem, $\ln\lambda - \ln r = \frac{1}{\xi}(\lambda - r)$, where $\xi \in (r,\lambda)$. Then $|\ln\lambda - \ln r| \leqslant |\lambda - r|$, so it suffices to find a rational approximation s of $\ln r$ so precise that $|s - \ln r| \leqslant \frac{1}{2n^T}$. It follows that

$$\begin{aligned} |s-\ln\lambda| = |s - \ln r + \ln r - \ln\lambda| &\leqslant |s-\ln r| + |\ln r - \ln\lambda| \\ &\leqslant \frac{1}{2n^T} + \frac{1}{2n^T} = \frac{1}{n^T}. \end{aligned}$$

It remains to demonstrate that a rational approximation s of $\ln r$ is possible to compute in polynomial time and has the claimed size.

Let $r = \frac{\alpha}{\beta}$, where α and β are integers. The obvious strategy for computing a rational approximation of $\ln r = \ln\frac{\alpha}{\beta} = \ln\alpha - \ln\beta$ works: since r has size $S = O(\ln^k n)$ for some fixed k, it follows that α and β both have lengths proportional to $\ln^k n$. As discussed in Section 3, it is possible to find rational approximations s_α and s_β for α and β respectively (with precision $\frac{1}{4n^T}$) in time polynomial with respect to $\ln^k n$. Therefore it is also possible to compute $s = s_\alpha - s_\beta$ in polynomial time, and

$$\begin{aligned} |s-\ln r| = |s_\alpha - s_\beta - \ln\alpha + \ln\beta| &\leqslant |s_\alpha - \ln\alpha| + |s_\beta - \ln\beta| \\ &\leqslant \frac{1}{4n^T} + \frac{1}{4n^T} = \frac{1}{2n^T}. \end{aligned}$$

It is straightforward to see that the size of s is polynomial in $\ln n$. □

Remark 8. In the previous lemma we required $\lambda > 1$. Values λ with $0 < \lambda < 1$ can be treated by noticing that $\ln\frac{1}{\lambda} = -\ln\lambda$.

To conclude this section, we find some upper bounds for numbers μ_0 and ν_0. In Definition 3 we can of course choose

$$P(x) = p_d x^d + p_{d-1}x^{d-1} + \cdots + p_1 x + p_0 \tag{6}$$

to be the defining polynomial of λ. If λ_i is any root of (6) with $|\lambda_i| \geqslant 1$, then

$$\begin{aligned} |\lambda_i|^d &\leqslant p_d|\lambda_i|^d = \left|p_{d-1}x^{d-1} + \cdots + p_1 x + p_0\right| \\ &\leqslant H(\lambda)|\lambda_i|^{d-1} + \cdots + H(\lambda)|\lambda_i| + H(\lambda_i) \\ &\leqslant \deg(\lambda)H(\lambda)|\lambda_i|^{d-1}. \end{aligned}$$

It follows that $|\lambda_i| \leqslant \deg(\lambda)H(\lambda)$, and consequently numbers μ_0 and ν_0 can be chosen such that their absolute values are at most $\deg(\lambda)H(\lambda)$.

It is also possible to estimate the sizes of numbers μ_0 and ν_0. We will introduce the following lemmata for this estimation.

Lemma 9. *Let $d = \deg(\lambda)$ and $H = H(\lambda)$ be the degree and the height of λ. Then the distance between two distinct roots of a defining polynomial of λ is at least*

$$\frac{1}{(2d)^{\frac{d^2}{2}} H^{\frac{d(d+1)}{2}}}.$$

Proof. Let

$$P(x) = p_d(x - \lambda_1)\cdot\cdots\cdot(x - \lambda_d).$$

All the roots λ_i are distinct, since $P(x)$ is a scalar multiple of the minimal polynomial of λ. Also, the *discriminant*

$$d(P) = p_d^{2d-2}\prod_{i<j}(\lambda_i - \lambda_j)^2$$

is an integer [9]. If $m = |\lambda_r - \lambda_s|$ is the minimal distance between two different roots of $P(x)$ and M the corresponding maximal distance, we have

$$\begin{aligned} \left|d(P)\right| &= p_d^{2d-2}\cdot m^2 \cdot \prod_{\substack{i<j \\ (i,j)\neq(r,s)}} |\lambda_i - \lambda_j|^2 \\ &\leqslant p_d^{2d-2}\cdot m^2 \cdot \left(M^2\right)^{\frac{d(d-1)}{2}-1} \\ &\leqslant H^{2d-2}\cdot m^2 \cdot (2dH)^{d^2-d-2} \\ &< m^2(2d)^{d^2}H^{d^2+d}. \end{aligned}$$

Now that $d(P)$ is a nonzero integer, so we have that $|d(P)| \geqslant 1$, and consequently

$$m^2 > \frac{1}{(2d)^{d^2}H^{d^2+d}},$$

which implies

$$m > \frac{1}{(2d)^{\frac{d^2}{2}} H^{\frac{d(d+1)}{2}}}. \qquad \square$$

Lemma 10. *An interval $[a,b]$ of length $\epsilon = b-a > 0$ contains a rational number of size $O(\log \frac{a}{\epsilon})$.*

Proof. If $\epsilon \geqslant 1$, then $[a,b]$ contains an integer $\lfloor a \rfloor + 1$, which has size $O(\log a)$. Assume then that $\epsilon < 1$, and choose $n \geqslant \log_d \frac{1}{\epsilon}$ least possible. Then the interval $[d^n a, d^n b]$ contains an integer $\lfloor d^n a \rfloor + 1$, which has size at most $n + \log_d(a+1) = O(\log \frac{a}{\epsilon})$. Consequently, $[a,b]$ contains a rational number $\frac{\lfloor d^n a \rfloor + 1}{d^n}$, which has size $O(\log \frac{a}{\epsilon})$. $\square$

We can now estimate the sizes of numbers μ_0 and ν_0 needed in the description of an algebraic number: they can be chosen in such a way that their absolute values are at most dH, and they are found in an interval of length at least $\frac{1}{(2d)^{\frac{d^2}{2}} H^{\frac{d(d+1)}{2}}}$. Therefore, they can be chosen such that their size is

$$O\left(\log\left(dH(2d)^{\frac{d^2}{2}} H^{\frac{d(d+1)}{2}}\right)\right) = O\left(d^2 \log(Hd)\right).$$

On the other hand, the description size of the defining polynomial is

$$O(d \log H),$$

so we can conclude that a real algebraic number λ has a description size $O(d^2 \log(Hd))$.

Remark 11. The lower bound on the minimum distance between the roots can be improved [11]. In [11], also an upper bound for the distance between two distinct roots of a polynomial is given: polynomial $p(x) = x^d - 2(ax-1)^2$ has distinct roots such that their distance is less than $\frac{2}{a^{\frac{d-1}{2}}}$.

5. Approximating the logarithm of a linear combination of powers of algebraic numbers

The issues handled in this section are needed when computing the length of Fibonacci numbers. The combinations we study in this section are of form

$$G(n) = c_1 \alpha_1^n + \cdots + c_k \alpha_k^n, \tag{7}$$

where $c_1, \ldots, c_k$, and $\alpha_1, \ldots, \alpha_k$ are fixed algebraic numbers, given as in Definition 3. We also assume that $|\alpha_1| > \max\{|\alpha_2|, \ldots, |\alpha_k|\}$. Notice that all the mentioned inequalities are decidable. In fact, the above inequalities can be decided in polynomial time (in the description of the algebraic numbers), but we will not need that knowledge in the continuation. Our aim is to show how to design a polynomial time algorithm for computing the length of the real part of $G(n)$, and the most meaningful digit of $G(n)$ (in d-ary basis with $d \geqslant 3$). We do not claim that the polynomial time algorithms for the aforementioned purposes can be found in polynomial time (with respect to the description of $G(n)$) or that the sizes of those algorithms are polynomial in the description size of $G(n)$, we only claim that these polynomial time algorithms exist and can be found algorithmically. Without loss of generality, we can assume that $\alpha_1 > 0$ and $c_1 > 0$.

Remark 12. Expressions like (7) arise, for example, when studying linear recurrences. If the polynomial of the recurrence $L(n)$ has only simple roots and a unique root with maximal absolute value,

then the solution to the recurrence looks exactly like (7). For example, the n-th Fibonacci number is equal to

$$F_n = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n.$$

Remark 13. Given a linear recurrence $L(n)$, it is decidable whether its characteristic polynomial has simple roots and whether the root with maximal absolute value is real. It is also possible to recover expression (7) when $L(n)$ is given, but we do not claim that it can be done in polynomial time.

As discussed in Section 2, we can obtain the length of the integer part of $G(n)$ (in d-ary representation) by computing the quantity $\lfloor \log_d G(n) \rfloor + 1$. For that purpose, we first write $G(n)$ as

$$G(n) = c_1\alpha_1^n\left(1 + \frac{c_2}{c_1}\left(\frac{\alpha_2}{\alpha_1}\right)^n + \cdots + \frac{c_k}{c_1}\left(\frac{\alpha_k}{\alpha_1}\right)^n\right).$$

Denoting $c = \max\{|\frac{c_2}{c_1}|, \ldots, |\frac{c_k}{c_1}|\}$ and $\beta = \max\{|\frac{\alpha_2}{\alpha_1}|, \ldots, |\frac{\alpha_k}{\alpha_1}|\}$ we have the estimation

$$\left|\frac{c_2}{c_1}\left(\frac{\alpha_2}{\alpha_1}\right)^n + \cdots + \frac{c_k}{c_1}\left(\frac{\alpha_k}{\alpha_1}\right)^n\right| \leqslant c(k-1)\beta^n.$$

Now that $0 < \beta < 1$, the quantity $c(k-1)\beta^n$ is less than 1 if n is large enough. For such values of n,

$$\ln G(n) = \ln c_1 + n\ln\alpha_1 + R_1(n),$$

where

$$\left|R_1(n)\right| = \left|\ln\left(1 + \frac{c_2}{c_1}\left(\frac{\alpha_2}{\alpha_1}\right)^n + \cdots + \frac{c_k}{c_1}\left(\frac{\alpha_k}{\alpha_1}\right)^n\right)\right| \leqslant c(k-1)\beta^n.$$

Therefore,

$$\log_d G(n) = \log_d c_1 + n\log_d\alpha_1 + R(n), \tag{8}$$

where

$$\left|R(n)\right| \leqslant \frac{c(k-1)}{\ln d}\beta^n \leqslant \frac{1}{n^{C+1}} \tag{9}$$

for each positive constant C if n is large enough.

For determining how large n should be chosen to satisfy the rightmost inequality of (9), notice first that

$$\frac{c(k-1)}{\ln d}\beta^n < 2c(k-1)\beta^n,$$

and that inequality

$$2c(k-1)\beta^n \leqslant \frac{1}{n^{C+1}}$$

is implied by

$$n \ln \frac{1}{\beta} - (C+1)\ln n - \ln 2c(k-1) > 0,$$

because $\ln \frac{1}{\beta} > 0$. It is easy to see that there exists $M > 0$ such that

$$f(x) = x \ln \frac{1}{\beta} - (C+1)\ln x - \ln 2c(k-1) > 0$$

when $x > M$. In fact, since

$$f'(x) = \ln \frac{1}{\beta} - \frac{C+1}{x},$$

function $f(x)$ is increasing when

$$x \geqslant \frac{C+1}{\ln \frac{1}{\beta}}.$$

Now since we assume that all the involved algebraic numbers are given as in Definition 3, it is possible to algorithmically find an integer M' such that $f(x)$ is increasing when $x > M'$. Moreover, since we know that $f(x)$ tends to infinity as x grows, we can also find algorithmically an integer M such that $f(x) > 0$ whenever $x > M$. But this is to say that, given C, we can algorithmically find an integer M such that

$$\big|R(n)\big| \leqslant \frac{1}{n^{C+1}}$$

when $n \geqslant M$. In fact, the true vanishing rate of $|R(n)|$ is exponential, and consequently the above estimation is quite weak. However, the above estimation is good enough for our purposes.

Expression (8) is of course a good starting point when trying to find out $\lfloor \log_d G(n) \rfloor + 1$, the length of the d-ary representation of the real part of $G(n)$. However, there are some problematic things when using (8). By a simple example $G(n) = 2^n$, we point out in the next section which are the problems occurring.

Notice carefully we always regard quantities k, $c_1, \ldots, c_k$, $\alpha_1, \ldots, \alpha_k$ in (7) as *fixed constants*, and only n is the input variable to those algorithms we describe. It follows that the computational time to discover aforementioned M, for instance, is irrelevant – the time needed for that can be regarded constant.

Remark 14. A general linear recurrence $L(n)$ has solution

$$G(n) = P_1(n)\alpha_1^n + \cdots + P_k(n)\alpha_k^n, \tag{10}$$

where each P_i is a polynomial having $\deg P_i \leqslant m_i - 1$, where m_i is the multiplicity of α_i as the root of the characteristic polynomial of the recurrence [6]. Notice that the multiplicities of the roots, as well as representation (10) can be found algorithmically (probably not in polynomial time) when $L(n)$ is given. If α_1 is the unique root of maximal absolute value (with multiplicity 1), then $P_1 = c_1$ is a constant and it is possible to find an expression

$$\log_d G(n) = \log_d c_1 + n \log_d \alpha_1 + R(n)$$

as in (8). Furthermore, given $C > 0$, it is possible to find algorithmically an integer M such that

$$\left|R(n)\right| \leqslant \frac{1}{n^{C+1}},$$

whenever $n \geqslant M$. Hence all the cases where $G(n)$ has a matrix representation with nonnegative entries are covered as well, due to the Perron–Frobenius theorem, see [4].

6. An example: 2^n in ternary basis

In this section we introduce algorithms for computing $|(2^n)_{\mathbf{3}}|$, the length of ternary representation of 2^n, and the first symbol of $(2^n)_{\mathbf{3}}$, and analyze their complexities. The function considered is a special case of (7) with $k = 1$, $c_1 = 1$, and $\alpha_1 = 2$, so $G(n) = 2^n$. In the following section we prove that the algorithms represented work correctly. Here we begin with some observations.

Without loss of generality we can assume that the input n is given in binary representation, so the *size* of an input is $|n_{\mathbf{2}}|$, which, as seen previously, is $\Theta(\ln n)$. We can therefore state our problem rigorously as follows: given an input $n_{\mathbf{2}} \in \{0,1\}^*$, compute $(2^n)_{\mathbf{3}} \in \{0,1,2\}^*$, the ternary representation of 2^n. As seen in Section 2, the size of the input is $\Theta(\ln n)$, whereas the output size is $\Theta(n)$, which is exponential in the input size, and it follows that the problem is intractable by necessity.

On the other hand, computing $n_{\mathbf{2}} \mapsto |(2^n)_{\mathbf{3}}|$ is a very different problem. Now the output should be the length of the ternary representation of 2^n, or, from the algorithmic point of view, a *word* which represents the length. Again, without loss of generality, we can require the output in binary, which means that the output size would be

$$\begin{aligned}\left|\left|(2^n)_{\mathbf{3}}\right|_{\mathbf{2}}\right| &= \left\lfloor \log_2\left|(2^n)_{\mathbf{3}}\right| \right\rfloor + 1 \leqslant \log_2\left(\lfloor n\log_3 2\rfloor + 1\right) + 1 \\ &\leqslant \log_2(n\log_3 2 + n) + 1 = \log_2 n + \log_2(\log_3 2 + 1) + 1.\end{aligned}$$

Therefore, the output size is only $O(\ln n)$.

Equation

$$\left|(2^n)_{\mathbf{3}}\right| = \lfloor n\log_3 2\rfloor + 1 \tag{11}$$

gives a good starting point for computing the length, but the straightforward utilization of (11) contains at least two problematic features.

First, knowing $n_{\mathbf{2}}$ and $\log_3 2$ precisely enough allows us to compute the product $n\log_3 2$, but it must be noted that we should be able to compute $\log_3 2 \approx 0.6$ at least up to precision $\frac{1}{n}$, since for a larger imprecision the outcome could be incorrect. This is illustrated in the rightmost column of Fig. 1. In the previous sections, we have seen that this problem is easy to handle, since exponentially precise approximations of $\log_3 2$ can be computed in polynomial time.

The second, and more severe problem is, that an approximation for $\log_3 2$, does not directly offer any tools to compute $\lfloor n\log_3 2\rfloor$, no matter how precise the approximation is! To see this, let β_n, $n = 1,2,3,\ldots$, be a sequence of irrational numbers, and b_n, $n = 1,2,3,\ldots$, be a sequence of their very precise rational approximations, $|b_n - \beta_n| \ll 1$ for each n. Let us take some n, and assume, for instance, that $b_n < \beta_n$. If the interval (b_n, β_n) happens to contain an integer M, then $\lfloor b_n\rfloor = M - 1$, whereas the correct value is $\lfloor \beta_n\rfloor = M$. In other words, if we do not have *a priori* knowledge on the distance between β_n and the nearest integer M, we cannot certainly find the value $\lfloor \beta_n\rfloor$ by using only an approximation b_n of β_n. In Section 7, we use deep results of Alan Baker to solve this problem.

Remark 15. A very closely related problem has been studied in [10]. However, the approach of [10] is basically experimental and hence very different from ours.

n	$(2^n)_{\mathbf{3}}$	$\lvert(2^n)_{\mathbf{3}}\rvert$	$\lfloor 0.6n \rfloor + 1$
1	2	1	1
2	11	2	2
3	22	2	2
4	121	3	3
5	1012	4	4
6	2101	4	4
7	11202	5	5
8	100111	6	5(!)
9	200222	6	6
10	1101221	7	7
11	2210212	7	7
12	12121201	8	8
13	102020102	9	8(!)

Fig. 1. Ternary representations of 2^n for $n = 1, 2, \ldots, 13$.

When computing the most significant (leftmost) digit of $(2^n)_{\mathbf{3}}$, we note that an estimation similar to the one used in Section 2 shows that if $(2^n)_{\mathbf{3}} \in 1\{0,1,2\}^{\ell-1}$, then

$$3^{\ell-1} \leqslant 2^n < 2 \cdot 3^{\ell-1},$$

whereas $(2^n)_{\mathbf{3}} \in 2\{0,1,2\}^{\ell-1}$ implies

$$2 \cdot 3^{\ell-1} \leqslant 2^n < 3^{\ell},$$

where $\ell = \lvert(2^n)_{\mathbf{3}}\rvert$. Thus, to recover the most significant digit of $(2^n)_{\mathbf{3}}$ is to decide whether or not the inequality

$$2 \cdot 3^{\ell-1} \leqslant 2^n \quad \Longleftrightarrow \quad 3^{\ell-1} \leqslant 2^{n-1} \tag{12}$$

holds. Inequality (12) is equivalent to

$$\ell - 1 \leqslant (n-1)\log_3 2, \tag{13}$$

and (13) will be used for finding the first digit of $(2^n)_{\mathbf{3}}$.

It should be emphasized that the crucial point in these investigations is the expression $\lfloor \log_3 2^n \rfloor$. In the following sections, we will study more general expressions of form $\lfloor \log_d G(n) \rfloor$, where G is as in (7).

Now we are ready to describe the algorithms. The constant C occurring in the algorithms emerges from Baker's theorem and will be explained in the next section.

Algorithm 1: The length of the ternary representation of 2^n.
Input: A natural number n in binary representation.
Output: $\lvert(2^n)_{\mathbf{3}}\rvert$ in binary representation.

(1) Compute a rational approximation q of $\log_3 2$ so precise that

$$\lvert q - \log_3 2 \rvert \leqslant \frac{1}{n^{C+2}}.$$

(2) Compute qn.
(3) Compute $\lfloor qn \rfloor$.
(4) Output $\lfloor qn \rfloor + 1$.

In Section 3 we showed that step 1 can be done in time $O(\ln^3 n(\ln\ln n)^2)$, and, moreover, the approximating q has size $O(\ln n \ln\ln n)$. Especially, the numerator of $q = \frac{a}{b}$ has size $O(\ln n \ln\ln n)$, and hence the multiplication of q by n in step 2 can be done in time $O((\ln n \ln\ln n)^2)$, and the resulting number $qn = \frac{an}{b}$ has size $O(\ln n \ln\ln n + \ln n) = O(\ln n \ln\ln n)$. For the third step, an ordinary division of an by b is enough to reveal $\lfloor qn \rfloor$, and because of the numerator and the nominator sizes, it can be done in time $O((\ln n \ln\ln n)^2)$. As verified earlier, the outcoming number has size $O(\ln n)$, which implies that the last computation included in the fourth step can be performed in time $O(\ln n)$.

As a conclusion: Computation $n_{\mathbf{2}} \mapsto |(2^n)_{\mathbf{3}}|_{\mathbf{2}}$ can be performed in time $O(\ln^3 n(\ln\ln n)^2)$, or, to put it into other format, in time $O(|n_{\mathbf{2}}|^3 \ln^2 |n_{\mathbf{2}}|)$.

To prove the correctness, we should show that, for a suitably chosen C, equation $\lfloor qn \rfloor = \lfloor n \log_3 2 \rfloor$ holds. This is the topic of the following sections.

Algorithm 2: The most significant digit of the ternary representation of 2^n.
Input: A natural number n in binary representation.
Output: The leftmost digit of string $(2^n)_{\mathbf{3}}$.

(1) Compute $\ell = |(2^n)_{\mathbf{3}}|$ by using Algorithm 1.
(2) Compute a rational approximation q of $\log_3 2$ so precise that

$$|q - \log_3 2| \leqslant \frac{1}{(n-1)^{C+2}}.$$

(3) Compute numbers $\ell - 1$ and $(n-1)q$.
(4) Decide, whether $\ell - 1 \leqslant (n-1)q$.
(5) If $\ell - 1 \leqslant (n-1)q$, output 2, otherwise output 1.

The complexity analysis of Algorithm 2 is similar to that of Algorithm 1, and the outcoming complexity is $O(|n_{\mathbf{2}}|^3 \ln^2 |n_{\mathbf{2}}|)$.

7. Baker's theorem and its consequences

Let us recall the combinations

$$G(n) = c_1\alpha_1^n + \cdots + c_k\alpha_k^n \tag{14}$$

of Section 5. As previously, we require that $\alpha_1 > 0$, $c_1 > 0$, and that $\alpha_1 > \max\{|\alpha_2|, \ldots, |\alpha_k|\}$. In the previous section we studied a special case of (14) with $k = 1$, $c_1 = 1$, and $\alpha_1 = 2$. The problematic issue, computing proving that $\lfloor \log_d G(n) \rfloor$, can be correctly computed by using rational approximations, is treated in this section.

The information for proving the correctness is provided in the following theorem, the proof can be found in [1]. See also [5] for a special case.

Theorem 16. *(See A. Baker, 1966.) Let $\alpha_1, \ldots, \alpha_k$ be nonzero algebraic numbers with degrees at most d and heights at most A. Further, let $\beta_0, \ldots, \beta_k$ be algebraic numbers with degrees at most d and heights at most $B \geqslant 2$. Then for*

$$\Lambda = \beta_0 + \beta_1 \ln\alpha_1 + \cdots + \beta_k \ln\alpha_k$$

we have either $\Lambda = 0$ or $|\Lambda| > B^{-C}$, where C is an algorithmically computable number depending only on k, d, A, and on the principal value for the logarithms is chosen.[2]

[2] Over complex numbers, logarithm is generally many-valued. Any branch of the logarithm could be used, but the choice of the branch may affect the value of C.

Remark 17. In the case when $\beta_0 = 0$ and $\beta_1, \ldots, \beta_k \in \mathbb{Z}$, it has been shown that the theorem holds with $C = C'(\ln A)^k \ln(\ln A)^k$ and $C' = (16kd)^{200k}$ [2].

Choosing $k = 3$, $\beta_0 = 0$, $\beta_1 = -n$, $\beta_2 = -1$, $\beta_3 = M$, $\alpha_2 = c_1$, and $\alpha_3 = d$ we have

$$\Lambda = -n \ln \alpha_1 - \ln c_1 + M \ln d.$$

With these choices, $H(\beta_1) = n$, $H(\beta_2) = 1$, and $H(\beta_3) = M$, which leads us to a special case of Baker's theorem, strong enough for our purposes, stated as follows:

Theorem 18. *Let M be an integer. If $M \geqslant 1$, $n \geqslant \frac{1}{a} M$ for a given constant $a \geqslant 1$, and $M \ln d - \ln c_1 - n \ln \alpha_1 \neq 0$, then*

$$|M \ln d - \ln c_1 - n \ln \alpha_1| > \frac{1}{a^C} \frac{1}{n^C},$$

where $C > 0$ is an algorithmically computable constant.

Proof. In this case, we can choose $B = an \geqslant \max\{1, n, M\}$. □

Recall from Section 5 that for $G(n)$ as in (14) is possible to find a representation

$$\log_d G(n) = \log_d c_1 + n \log_d \alpha_1 + R(n),$$

where, for any given $C > 0$ it is possible to find algorithmically an integer M_R such that

$$\big|R(n)\big| \leqslant \frac{1}{n^{C+1}}$$

when $n \geqslant M_R$.

For the statement of the following theorem, let $G(n)$ be as before, and a, c_1, α_1, d, and C be as in Theorem 18. Let also $r \approx \log_d c_1$ and $q \approx \log_d \alpha_1$ be rational approximations such that $|r - \log_d c_1| \leqslant \frac{1}{n^{C+1}}$ and $|q - \log_d \alpha_1| \leqslant \frac{1}{n^{C+2}}$. Assume moreover that a number M_I with the following property is known: $c_1 \alpha_1^n$ is not an integer if $n \geqslant M_I$.

Theorem 19. *If n is large enough, then $M \leqslant \log_d G(n)$ if and only if $M \leqslant r + nq$.*

Proof. Assume first that $M \leqslant \log_d G(n)$ but $M > r + nq$, and that $n \geqslant \max\{M_I, M_R, 4a^C \ln d\}$.

First we have that

$$\begin{aligned}
\big|\log_d G(n) - M\big| &= \log_d G(n) - M \\
&= \log_d c_1 + n \log_d \alpha_1 + R(n) - M \\
&< \log_d c_1 + n \log_d \alpha_1 + R(n) - r - nq \\
&\leqslant |\log_d c_1 - r| + n |\log_d \alpha_1 - q| + \frac{1}{n^{C+1}} \\
&\leqslant \frac{1}{n^{C+1}} + n \frac{1}{n^{C+2}} + \frac{1}{n^{C+1}} = \frac{3}{n^{C+1}}. \qquad (15)
\end{aligned}$$

On the other hand Theorem 18 implies that

$$\begin{aligned}\left|M-\log_d G(n)\right| &= \left|M-\log_d c_1 - n\log_d \alpha_1 - R(n)\right| \\ &\geqslant |M-\log_d c_1 - n\log_d \alpha_1| - \left|R(n)\right| \\ &= \frac{1}{\ln d}|M\ln d - \ln c_1 - n\ln\alpha_1| - \left|R(n)\right| \\ &> \frac{1}{\ln d}\frac{1}{a^C}\frac{1}{n^C} - \frac{1}{n^{C+1}}. \end{aligned} \tag{16}$$

Inequalities (15) and (16) together give

$$\frac{1}{\ln d}\frac{1}{a^C}\frac{1}{n^C} - \frac{1}{n^{C+1}} < \frac{3}{n^{C+1}},$$

which is equivalent to

$$n < 4a^C \ln d.$$

By the choice of n, this is a contradiction. Therefore $M \leqslant \log_d G(n)$ implies $M \leqslant r+nq$ if n is sufficiently large. Similarly it can be shown that the inequality $M \leqslant r+nq$ implies $M \leqslant \log_d G(n)$ if n is large enough. □

Remark 20. Earlier we required that $c_1\alpha_1^n$ is not an integer when $n \geqslant M_I$. The reason for this requirement is the following: if $c_1\alpha_1^n$ is not an integer, it follows that $M\ln d - \ln c_1 - n\ln\alpha_1 \neq 0$ for each natural number M, and Theorem 18 applies. It would be enough to require that $c_1\alpha_1^M$ is not a power of d. It should also be noticed that, when studying linear recurrences with integer coefficients (assuming that the characteristic polynomial has simple dominating root), it is sometimes clear that $c_1\alpha_1^n$ ceases to be an integer when n is large enough. For instance, when thinking about Fibonacci numbers, $c_1\alpha_1^n$ + "remainder" is always an integer, whereas the remainder tends to zero but is never zero.

Theorem 21. *Let the notations be chosen as before. It n is sufficiently large, then $\lfloor \log_d G(n)\rfloor = \lfloor r+nq\rfloor$.*

Proof. Let $M = \lfloor r+nq\rfloor$. Then $M \leqslant r+nq$, and we can find a constant a such that $n \geqslant \frac{1}{a}M$, if n is large enough. It also follows that

$$M \leqslant r+nq < M+1 \tag{17}$$

and Theorem 19 implies immediately that $M \leqslant \log_d G(n)$, hence $M \leqslant \lfloor \log_d G(n)\rfloor$. On the other hand, if $M < \lfloor \log_d G(n)\rfloor$ strictly, then clearly $M+1 \leqslant \log_d G(n)$, which, by Theorem 19, implies that $M+1 \leqslant r+nq$. This contradicts (17) and therefore $\lfloor \log_d G(n)\rfloor = M = \lfloor r+nq\rfloor$. □

The correctness of the algorithms of Section 6

In Algorithm 1, we used an approximation q of $\log_3 2$ so sharp that

$$|q-\log_3 2| \leqslant \frac{1}{n^{C+2}}.$$

It remains to show that $\lfloor qn\rfloor = \lfloor n\log_3 2\rfloor$. But this is very straightforward: we can now take $a = c_1 = d = M_I = 1$, $r = 0$, and $C = 13.3$ (that this value of C can be chosen, see [12]) and apply Theorems 19 and 21 to see that $\lfloor n\log_3 2\rfloor = \lfloor qn\rfloor$. Similarly, Algorithm 2 is correct.

Remark 22. In most cases, value $\lfloor n \log_3 2 \rfloor$ can be derived from a rational approximation of $\log_3 2$ far more imprecise than that one required in Step 1 of Algorithm 1. It follows immediately that Algorithm 1 is not optimal for computing $\lfloor n \log_3 2 \rfloor$ in most cases, and a more efficient algorithm would compute rational approximations of $\log_3 2$ as long as the value $\lfloor n \log_3 2 \rfloor$ can be decided with certainty. Step 1 of Algorithm 1 gives an upper bound on how long one must compute the approximations. However, Algorithm 1 is notationally and conceptually simpler than more advanced ones. An analogous idea can be used to modify the algorithm of the next section into a practically useful one.

8. Fibonacci numbers

As another example, we mention Fibonacci numbers defined by the recurrence

$$F_n = F_{n-1} + F_{n-2} \tag{18}$$

with initial values $F_0 = F_1 = 1$. It is a well-known fact that the following closed form expression holds:

$$F_n = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n, \tag{19}$$

where the numbers

$$\frac{1 \pm \sqrt{5}}{2}$$

are the roots of the characteristic equation

$$x^2 - x - 1 = 0$$

of the recurrence (18). Now since

$$\left|\frac{1+\sqrt{5}}{2}\right| > \left|\frac{1-\sqrt{5}}{2}\right|,$$

we can apply the results of the previous section to design a polynomial time algorithm (with respect to the length of n) for computing the length of F_n, the nth Fibonacci number. We will demonstrate how to do that for ternary representation. The polynomial time algorithm presented here is of course practically useless due to a large constant associated to Baker's theorem.

Using the previous notations we have $c_1 = \frac{1}{\sqrt{5}}$, $\alpha_1 = \frac{1+\sqrt{5}}{2}$, and

$$G(n) = F_n = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n \left(1 - \left(\frac{1-\sqrt{5}}{1+\sqrt{5}}\right)^n\right).$$

It is easy to verify that $|\frac{1-\sqrt{5}}{1+\sqrt{5}}| < 2/5$ and since $\ln 3 > 1$, we have

$$\log_3 F_n = \log_3 \frac{1}{\sqrt{5}} + n \log_3 \frac{1+\sqrt{5}}{2} + R(n),$$

where $|R(n)| < (\frac{2}{5})^n$. For computing rational approximations for $c_1 = \frac{1}{\sqrt{5}}$ and $\alpha_1 = \frac{1+\sqrt{5}}{2}$ we can fix polynomials $p_1(x) = 5x^2 - 1$ and $p_2(x) = x^2 - x - 1$, respectively. The additional information required (interval) for $p_1(x)$ can be chosen as $[0, 1]$, and $[1, 2]$ for $p_2(x)$.

To find the constant a of Theorems 19 and 21, we should keep the proof of Theorem 21 in mind: we are studying expressions of form $r+nq$, where r and q are rational approximations of $c_1 = \frac{1}{\sqrt{5}}$ and $\alpha_1 = \frac{1+\sqrt{5}}{2}$, respectively. Because of the choice of the initial intervals, $r \in [0,1]$ and $q \in [1,2]$ always. Now if $M = \lfloor r+nq \rfloor$, then $M \leqslant r+nq \leqslant 1+2n$, and $n \geqslant \frac{1}{2}(M-1) \geqslant \frac{1}{3}M$, if $M \geqslant 3$. Therefore, we can choose $a = 3$ and we know that $n \geqslant \frac{1}{3}M$, whenever $M \geqslant 3$ (we need this to satisfy Theorem 18). But now $M+1 \geqslant r+nq$ implies that $M \geqslant -1+r+nq \geqslant -1+n$, so we know that $M \geqslant 3$ whenever $n \geqslant 4$.

Now $\deg(\frac{1}{\sqrt{5}}) = 2$, $\deg(\frac{1+\sqrt{5}}{2}) = 2$, $H(\frac{1}{\sqrt{5}}) = 5$, $H(\frac{1+\sqrt{5}}{2}) = 1$, and according to [2] we can choose $C = (16\cdot 3\cdot 2)^{200\cdot 3}\cdot(\ln 5)^3\cdot\ln(\ln 5)^3 \approx 1.37\times 10^{1190}$ such that either

$$M\ln 3 - \ln\frac{1}{\sqrt{5}} - n\ln\frac{1+\sqrt{5}}{2} = 0 \tag{20}$$

or

$$\left| M\ln 3 - \ln\frac{1}{\sqrt{5}} - n\ln\frac{1+\sqrt{5}}{2} \right| \geqslant \frac{1}{B^C},$$

where $B \geqslant \max\{1,n,M\}$. Whenever $n \geqslant 4$, we will have $3n \geqslant M$, and we can choose $B = 3n$ to get

$$\left| M\ln 3 - \ln\frac{1}{\sqrt{5}} - n\ln\frac{1+\sqrt{5}}{2} \right| \geqslant \frac{1}{3^C}\frac{1}{n^C} \tag{21}$$

whenever $n \geqslant 4$, if (20) does not hold. On the other hand, (20) can be written equivalently as

$$3^M = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n, \tag{22}$$

and this equation is impossible, since the left-hand side is not an integer. To see this, we recall that

$$F_n = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n,$$

and that $|\frac{1-\sqrt{5}}{2}| < \frac{2}{3}$. Therefore, for $n \geqslant 4$, the value of the right-hand side of (22) is F_n plus a number with absolute value smaller than 1, and that cannot be an integer. As a conclusion, we have that (21) holds for each $n \geqslant 4$.

The next step in designing the algorithm is to find an integer M_R such that

$$\left|R(n)\right| \leqslant \frac{1}{n^{C+1}} \tag{23}$$

whenever $n \geqslant M_R$. Knowing that $|R(n)| < (\frac{2}{5})^n$ it is straightforward to see that (23) holds whenever $n \geqslant 1.5\times 10^{1190} = M_R$.

Choose $M = \max\{1, M_R, 4\cdot 3^C\cdot 2, 4\}$ (a huge number!). Now that $\ln 3 < 2$, Theorems 19 and 21 hold whenever $n \geqslant M$.

Algorithm 3: The length of the ternary representation of F_n.
Input: A natural number n in binary representation.
Output: $|(F_n)_{\mathbf{3}}|$ in binary representation.

(1) If $n < M$, compute F_n by using the recursion, give the answer and stop.

(2) Use the method of Section 4 to compute a rational approximation q of $\log_3 \frac{1+\sqrt{5}}{2}$ so precise that

$$\left| q - \log_3 \frac{1+\sqrt{5}}{2} \right| \leqslant \frac{1}{n^{C+2}}.$$

(3) Use the method of Section 4 to compute a rational approximation r of $\log_3 \frac{1}{\sqrt{5}}$ so precise that

$$\left| r - \log_3 \frac{1}{\sqrt{5}} \right| \leqslant \frac{1}{n^{C+1}}.$$

(4) Compute $r + qn$.
(5) Compute $\lfloor r + qn \rfloor$.
(6) Output $\lfloor r + qn \rfloor + 1$.

As in the case of Algorithm 1, Theorems 19 and 21 imply the correctness of Algorithm 3 immediately.

9. Extensions and open problems

9.1. First two digits

If k is fixed, we can extend the above procedures to compute in polynomial time the k most significant digits of any expression $G(n)$ as in (14). We outline here briefly how to compute first two digits of the ternary representation of $G(n)$. The extension to $k > 2$ and any base is straightforward. It must be emphasized that the algorithm for computing the first k digits of $G(n)$ will be exponential in k, but polynomial in n.

For extracting the first two digits of a given number N given in base 3, we first denote $\ell = |N_{\mathbf{3}}|$ (if $N = G(n)$ as in (14), then ℓ is computable in polynomial time). Estimations similar to the ones in Section 2 show that, if $m_1 \in \{1, 2\}$ and $m_2 \in \{0, 1, 2\}$, then

$$\begin{aligned} & N_{\mathbf{3}} \in m_1 m_2 \{0, 1, 2\}^{\ell-2} \\ & \iff \quad (3m_1 + m_2) \cdot 3^{\ell-2} \leqslant N < (3m_1 + m_2 + 1) \cdot 3^{\ell-2}. \end{aligned}$$

Now that $m_1 \in \{1, 2\}$ and $m_2 \in \{0, 1, 2\}$, $3m_1 + m_2$ ranges from 3 to 8. This means that to find out the two most significant digits of N, we should find the largest number $m \in \{3, 4, \ldots, 8\}$ which satisfies inequality $m \cdot 3^{\ell-2} \leqslant N$ (which is equivalent to $\ell - 2 \leqslant \log_3 N - \log_3 m$). To do so, we test that inequality for each $m \in \{3, 4, \ldots, 8\}$ exhaustively, so it suffices to describe how to test an individual inequality $\ell - 2 \leqslant \log_3 N - \log_3 m$ in polynomial time.

For that purpose, we will find good approximations for $\log_3 m$ and $\log_3 N$, and then use Baker's theorem to prove that the decision $\ell - 2 \leqslant \log_3 N - \log_3 m$ made by using the approximations is correct. Recall that $\log_3 N = \log_3 G(n)$ has a representation

$$\log_3 G(n) = \log_3 c_1 + n \log_3 \alpha_1 + R(n),$$

where $R(n)$ tends to zero exponentially fast. Now we can use Baker's theorem to find a constant C such that

$$|M \ln 3 - \ln c_1 + \ln m - n \ln \alpha_1| \geqslant \frac{1}{B^C}, \tag{24}$$

where $B \geqslant \max\{M, 1, 1, n\}$. As before, one can choose a constant a such that for each M which will be considered, we have $n \geqslant \frac{1}{a}M$. This implies that B can be chosen as an, and hence

$$|M \ln 3 - \ln c_1 + \ln m - n \ln \alpha_1| \geqslant \frac{1}{a^C} \frac{1}{n^C}.$$

Then, if r, s, and q satisfy $|r - \log_3 c_1| \leqslant \frac{1}{n^{C+1}}$, $|\log_3 m - s| \leqslant \frac{1}{n^{C+1}}$, and $|q - \log_3 \alpha_1| \leqslant \frac{1}{n^{C+2}}$, we can prove, analogously to Theorem 19, that

$$M \leqslant \log_3 G(n) - \log_3 m \quad \Longleftrightarrow \quad M \leqslant r + nq - s, \tag{25}$$

if n is large enough. As previously, a lower bound for n to satisfy (25) can be found effectively, but here we omit all the details, including proofs, since they are similar to the ones of Theorem 19.

Remark 23. According to Theorem 16, the same constant C selected for $m = 8$ in (24) applies also for each choice of $m \in \{1, 2, 3, \ldots, 8\}$. Therefore, it suffices to find C only once. Notice that this value C applies to finding out the length of $G(n)$ as well.

9.2. Multiplicity greater than one

Another way to extend the previous results is to consider the case where the dominant real root of the characteristic polynomial of $L(n)$ is not simple. It turns out that a slight modification will yield a polynomial time algorithm for computing $\lfloor \log_d G(n) \rfloor$ in that case, too. Here we only outline how such an algorithm can be found, the details are left to the reader.

In this case, we have

$$G(n) = P_1(n)\alpha_1^n + \cdots + P_k(n)\alpha_k^n, \tag{26}$$

$|\alpha_1| > \max\{|\alpha_2|, \ldots, |\alpha_k|\}$, each polynomial P_i has degree at most $m_i - 1$, where m_i is the multiplicity of the root α_i. Notice that it is decidable whether $L(n)$ satisfies the aforementioned conditions and that the expression (26) can be found algorithmically but not necessarily in polynomial time.

We can write $\log_d G(n)$ as

$$\log_d G(n) = \log_d P_1(n) + n \log_d \alpha_1 + R(n),$$

where $R(n)$ tends to zero exponentially fast.

It is easy to see that if β is an algebraic number of degree e, then for each integer n, $\deg(n\beta) = e$ and $H(n\beta) \leqslant n^e H(\beta)$. It can also be shown that if $H(\beta_1)$, $H(\beta_2) \leqslant H$ and $d(\beta_1)$, $d(\beta_2) \leqslant e$, then $H(\beta_1 + \beta_2) \leqslant H^{C_e}$, where C_e is an algorithmically computable number depending only on e [1].

Assume that the coefficients of $P_1(x)$ belong to a extension of $\mathbb{Q}$ of degree e and have heights at most H. Then it can be shown that for each integer n, $\deg(P_1(n)) \leqslant e$ and $H(P_1(n)) \leqslant C_1 n^{C_2}$, where C_1 is an algorithmically computable number that depends only on m_1, H, and e, and C_2 is also an algorithmically computable number depending only on e and m_1. Notice that exponentially precise approximations of $P(n)$ can be computed in polynomial time with respect to $\log n$.

To use Baker's theorem (Theorem 16) we choose

$$\Lambda = -n \ln \alpha_1 - \ln P_1(n) + M \ln d,$$

$n \geqslant \frac{1}{a}M$, $B = an \geqslant \max\{1, n, M\}$, and have an analogous result to Theorem 18. For large n, the heights of numbers α_1, $P_1(n)$, and d is bounded above by $C_1 n^{C_2}$, so according to Remark 17 we can choose $C = C'(\ln(C_1 n^{C_2}))^3 \ln(\ln(C_1 n^{C_2}))^3$, where C' depends only on e. Simply we can find an constant C'' depending only on e and H such that $C \leqslant C'' \ln^3 n \ln(\ln n)^3$ if n is large enough.

A statement analogous to Theorem 16 now says that

$$\left|M \ln d - \ln P_1(n) - n \ln \alpha_1\right| \geqslant \frac{1}{(an)^{C'' \ln^3 n \ln(\ln n)^3}}, \tag{27}$$

if n is large enough. Even though the exponent of n in the nominator of the right-hand side of (27) is not fixed, it is anyway polynomial in $\ln n$. Analogously to previous considerations, it is possible to design a polynomial time algorithm for computing $\lfloor \log_d G(n) \rfloor$, but we leave the details to the reader.

9.3. Roots of unity times a constant

Yet another extension can be done when

$$G(n) = P_1(n)\alpha_1^n + \cdots + P_s(n)\alpha_s^n + P_{s+1}(n)\alpha_{s+1}^n + \cdots + P_k(n)\alpha_k^n,$$

where $\alpha = |\alpha_1| = \cdots = |\alpha_s| > \max\{|\alpha_{s+1}|, \ldots, |\alpha_k|\}$ and $\alpha_1/\alpha, \ldots, \alpha_s/\alpha$ are roots of unity, say $\alpha_1 = \alpha e^{\frac{2\pi i}{r_1}}, \ldots, \alpha_s = \alpha e^{\frac{2\pi i}{r_s}}$.

Here we must, however, agree on how to give a definition of a complex algebraic number in such a way that it is possible to compute exponentially precise approximations in polynomial time. There are many ways to do that, we leave it to the reader.

We can choose $r = \operatorname{lcm}(r_1, \ldots, r_s)$ and for each $n \in \mathbb{N}$ find an expression $n = qr + b$ to see that

$$G(n) = \alpha^n \left(P_1(n)\left(e^{\frac{2\pi i}{r_1}}\right)^b + \cdots + P_1(n)\left(e^{\frac{2\pi i}{r_s}}\right)^b\right) + P_{s+1}\alpha_{s+1}^n + \cdots + P_k\alpha_k^n.$$

Thus we can divide the original sequence $G(n)$ into r subcases and study them separately. Notice that this last example covers also the case where both α and $-\alpha$ ($\alpha \in \mathbb{R}$) are roots of the characteristic polynomial of the recurrence.

9.4. Open problems

We conclude with some open problems.

Problem 24. It there a polynomial time algorithm for computing $\lfloor \log_d L(n) \rfloor$ if $L(n)$ is *any* linear recurrence?

Problem 25. The "middle digit" of expression (14) seems to be difficult to compute. The problem can be posed as follows: given n, compute first $\ell = |G(n)|$ (this can be done in polynomial time), then compute the $\lfloor \ell/2 \rfloor$th most significant digit of $G(n)$. Is there a polynomial time algorithm for this task?

Acknowledgments

We wish to thank Tapani Matala-aho for providing the reference to the work of Georges Rhin [12].

References

[1] A. Baker, Transcendental Number Theory, Cambridge University Press, Cambridge, 1975.
[2] A. Baker, The theory of linear forms in logarithms, in: A. Baker, D.W. Masser (Eds.), Transcendence Theory: Advances and Applications, Academic Press, 1977, pp. 1–27.
[3] H. Dörrie, 100 Great Problems of Elementary Mathematics: Their History and Solutions, Dover, New York, 1965, pp. 112–116.
[4] F.R. Gantmacher, The Theory of Matrices, vol. II, New York, 1960.
[5] A. Gelfond, Sur les approximations des nombres transcendants par des nombres algébriques, Dokl. Acad. Sci. URSS 2 (1935) 177–182 (in Russian and French).

[6] M. Hall Jr., Combinatorial Theory, second ed., John Wiley & Sons, Inc., 1986.
[7] M. Hirvensalo, J. Karhumäki, Computing partial information out of intractable one – The first digit of 2^n at base 3 as an example, Lect. Notes Comput. Sci. 2420 (2002) 319–327.
[8] N. Jacobson, Basic Algebra, I, Freeman, 1974.
[9] A.G. Kuros, Higher Algebra, MIR Publishers, 1972.
[10] V. Lefèvre, J.-M. Muller, A. Tisserand, Toward correctly rounded transcendentals, IEEE Trans. Comput. 47 (11) (1998) 1235–1243.
[11] M. Mignotte, Some useful bounds, in: B. Buchberger, G.E. Collins, R. Loos (Eds.), Computer Algebra, Springer, 1982, pp. 259–263.
[12] G. Rhin, Approximants de Padé et mesures effectives d'irrationalité, in: Séminaire de Théorie des Nombres, Paris, 1985–1986, in: Progr. Math., vol. 71, Birkhäuser Boston, Boston, MA, 1987, pp. 155–164.