# Checking and Bounding the Solutions of Some Recurrence Relations*

Roberto Bagnara and Alessandro Zaccagnini

Department of Mathematics, University of Parma, Italy
bagnara@cs.unipr.it, alessandro.zaccagnini@unipr.it

**Abstract.** Recurrence relations play an important role in the field of complexity analysis since complexity measures can often be elegantly expressed by systems of such relations. This justifies the interest in automatic, precise, efficient and correct systems able to solve or to approximate the solution of systems of recurrence relations. Assume such a system is built. Since closed-form solutions for recurrences of even modest complexity can be so big and complex to be unmanageable, how can confidence on such a system be gained? How can we quickly validate or perhaps disprove its results? And, in those cases where the exact solution is too complex to be of practical use, how can we trade precision for efficiency by approximating them from below and from above? We also concern ourselves with a problem related to the handling of sets of solutions of recurrence relations: how can we confine by means of a lower bound and an upper bound a set of such solutions? We provide some solutions to these problems where we are careful to rely, whenever possible, on fast integer computations and/or conditions that are easy to check in a completely automatic way. The ongoing experimental evaluation of these ideas is giving very promising results, showing order-of-magnitude speedups over the more traditional methods.

## 1 Introduction

An important aspect of automatic complexity analysis [2, 6–8, 11, 13, 14, 18] is the possibility of automatically solving or approximating the solutions of systems of recurrence relations. In the literature, there are both techniques [4, 10] and software[1] for solving recurrences, but they only deal with a rather restricted range of cases and sometimes assume the interaction with a human operator. On the other hand, existing, completely automatic systems such as [6, 7] only provide quite rough approximations of the solutions. An attempt at improving the state of the

* The work of R. Bagnara has been partly supported by MURST project “Constraint Based Verification of Reactive Systems.” The work of A. Zaccagnini has been partly supported by MIUR project “Zeta and L Functions and Diophantine Problems in Number Theory.”

[1] Such as, for instance, Maxima (http://sourceforge.net/projects/maxima/) and MuPAD (http://www.mupad.de/).

art in this field is being made in the context of the PURRS project (*Parma University's Recurrence Relation Solver*, see `http://www.cs.unipr.it/purrs/`). This project aims at the creation of a software library providing all the services needed for the efficient solution and approximation of recurrence relations, and is especially targeted at fully automatic complexity analysis. Quickly finding exact solutions and/or tight approximations, in closed form, to recurrence relations is certainly an important and challenging task. In this paper, however, we concentrate on some problems concerning the manipulation of such solutions/approximations, in a way that is totally independent from the techniques with which they have been obtained.

A system for automatic complexity analysis must provide an abstraction of (possibly infinite) sets of sequences of real numbers. A sequence of real numbers expresses the "cost" of one "process" in terms of some "input measure" expressed by means of a natural number. Notice that this definition is quite general: *cost* may be in terms of elapsed time, number of statements executed, memory used, number of packets exchanged over the network, ...; a *process* may be a piece of software but also a communication protocol; the *input measure* can be any metric of the input of a program/procedure, or, say, the number of participants to some synchronization protocol. Because of well-known undecidability results, effective complexity analysis can only be based on approximations [5] and this is one of the reasons why we need to deal with *sets* of sequences. The domain of the analysis is thus given by $\wp\bigl(\mathbb{R}_\infty^{\mathbb{N}}\bigr)$: during the analysis, elements of $\wp\bigl(\mathbb{R}_\infty^{\mathbb{N}}\bigr)$ are generated by imposing a recurrence relation that the sequences must satisfy, by computing approximations of set union (this is required in order to approximate the complexity of conditionals), of addition (in order to approximate sequential composition), and so forth.

Assume that a recurrence relation solver is built and suppose, quite reasonably, that it is sophisticated and complex enough so that proving its overall correctness is beyond the state-of-the-art in program verification. Closed-form solutions for recurrences of modest complexity can require several pages to be printed out so that manually checking them is not an option. As we discovered the hard way, the naive approach of substituting the solution provided by the system into the original recurrence and simplifying the resulting expression is also impractical as it leads to serious efficiency problems. Thus, the first problem we attack in this paper is the following: how can confidence on such a solver be gained? To what extent can we validate its results? This is very important from a software engineering perspective, since it allows automatic regressions testing of the solver[2] and is also an important tool for the development of critical systems (e.g., hard real-time systems in charge of crucial tasks) where the cost of an undetected bug in the recurrence relation solver could be too high. For both

[2] There is an always increasing recognition of the importance of unit testing in software development. The *Extreme Programming* methodology (`http://www.extremeprogramming.org/`), for instance, pushes this to the point of requiring that the test is written before the unit.

these applications, it is also important that checking of the candidate solutions can be done quickly.

The problem of checking the correctness of the solution provided by a recurrence relation solver is easily reduced to the following problem: given $f \colon \mathbb{N} \to \mathbb{R}$ in some restricted, syntactically characterized class of functions, does $f$ vanish identically? That is, do we have $\forall n \in \mathbb{N} : f(n) = 0$? The automatic solution of this problem presents two main difficulties: the first is that the computer algebra system may not be able to completely simplify expressions involving one integer variable; the second is that, even when this simplification succeeds, it may require too much computation resources (on the order of several minutes of CPU time even for the verification of modestly sized recurrences). The approach proposed in this paper is much more efficient.

The exact solutions of recurrence relations can also be very big and complex. This, besides the impact on the cost of verification, poses a more fundamental problem: the solution can be so complex so as to be unmanageable and thus useless to the client application. This suggests the need to find upper and lower approximations of the exact solutions by means of suitable, "simpler" functions, and this constitutes the second contribution of this paper (the recognition of the importance of approximating the solutions of recurrence equations dates back to Cohen and Katcoff [4]).

The last problem we tackle in this paper is tightly related to the previous one as it deals with the manipulation of lower and upper bounds in order to define approximations of $\wp(\mathbb{R}_\infty^\mathbb{N})$. Formally, for some class of *boundary functions* $\mathfrak{B} \in \wp(\mathbb{R}_\infty^\mathbb{N})$, we restrict ourselves to the representation of the subset of $\wp(\mathbb{R}_\infty^\mathbb{N})$ defined by

$$\mathfrak{F} \stackrel{\mathrm{def}}{=} \Bigl\{ F \in \wp(\mathbb{R}_\infty^\mathbb{N}) \bigm| \exists l, u \in \mathfrak{B} \mathrel{.} \forall f \in F : l \leq f \leq u \Bigr\}.$$

In order to adopt this approach we need ways to compute the lower and upper bounds required for approximating the union of $F_1, F_2 \in \mathfrak{F}$, that is, for computing a (possibly) small $F \in \mathfrak{F}$ such that $F \supseteq F_1 \cup F_2$. This problem reduces to the following: given $b_1, b_2 \in \mathfrak{B}$, approximate, within the chosen class $\mathfrak{B}$ of boundary functions, $\max\{b_1, b_2\}$ from above and $\min\{b_1, b_2\}$ from below. (A definition of $\mathfrak{B}$ that suits the needs of the present paper is given in Section 5.)

Our long-term plan is, of course, to solve these three problems for a class of recurrence relations that is as wide as possible. In this paper we concentrate on the case when an exact formula for the solution of the recurrence exists, that is for linear recurrences of finite order with constant coefficients[3] and also for those recurrences with variable coefficients that possess a hypergeometric solution [12]. For these classes of recurrences (which arise with very high frequency in, e.g., the complexity analysis of high-level languages), this paper presents a unified approach to the solution of the aforementioned problems. The solutions we propose share some aspects, such as the identification of dominant terms.

[3] The classical case; see, e.g., [1] for a detailed description of both the theoretical aspects and the implementation issues.

What makes them particularly interesting is the fact that the problem of proving/disproving $\forall n \in \mathbb{N} : f(n) = 0$ and the problem of finding $g$ such that $\forall n \in \mathbb{N} : g(n) \geq \max\{f_1(n), f_2(n)\}$ or $\forall n \in \mathbb{N} : g(n) \leq \min\{f_1(n), f_2(n)\}$ are reduced to testing a *finite* (and usually very small) set of conditions. Moreover, in several cases these conditions are simple comparisons between integers (i.e., they are very fast and exact) or amount to the numerical evaluation of suitable subexpressions of $f$, $f_1$ and $f_2$ at a small number of integers. We are not aware of other research works tackling these problems.

Notice that we are mainly interested in *exact* solutions and in the possibility of extracting from them upper and lower bounds that are valid for all positive integers: this is the reason why we do not deal with asymptotics here, though there are well-known techniques for obtaining them. See for example Wilf [17, Chapter 5], for a general description, or Gourdon and Salvy [9] for a case similar to the one considered here.

The paper is structured as follows: Section 2 recalls the classes of recurrence relations relevant to the present paper; Section 3 presents our proposal to validate candidate solutions to such recurrences; Sections 4 and 5 deal with the problems of computing upper and lower bounds, respectively; Section 6 concludes by presenting our first experimental results and discussing current and future work. The proofs of all the stated results are in Section A.

## 2 Preliminaries

Consider the linear recurrence with *constant coefficients* given by

$$x_n = a_1 x_{n-1} + a_2 x_{n-2} + \cdots + a_k x_{n-k} + p(n), \tag{2.1}$$

for $n \geq k$, where $k$ is a fixed positive integer that is called *order of the recurrence relation* (2.1), and $p$ is a function defined over the set of natural numbers. Here we assume tacitly that the coefficients $a_1, \ldots, a_k$ are real numbers, that $a_k \neq 0$, and that the values $x_0, \ldots, x_{k-1}$ are given real numbers. Eq. (2.1) is associated to the homogeneous recurrence

$$\begin{cases} g_n = a_1 g_{n-1} + a_2 g_{n-2} + \cdots + a_k g_{n-k}, \\ g_0 = 1, \\ g_n = a_1 g_{n-1} + a_2 g_{n-2} + \cdots + a_n g_0, \qquad 1 \leq n < k. \end{cases} \tag{2.2}$$

This can be solved by means of the *characteristic equation*

$$\lambda^k = a_1 \lambda^{k-1} + \cdots + a_{k-1}\lambda + a_k. \tag{2.3}$$

Let this polynomial equation have roots $\lambda_1$ with multiplicity $\mu_1, \ldots, \lambda_r$ with multiplicity $\mu_r$, where the $\lambda_j$'s are distinct complex numbers, and the $\mu_j$'s are positive integers. The general solution of (2.2) is

$$g_n = \sum_{j=1}^{r} (\alpha_{j,0} + \alpha_{j,1} n + \cdots + \alpha_{j,\mu_j - 1} n^{\mu_j - 1}) \lambda_j^n, \tag{2.4}$$

and the values of the coefficients $\alpha$ must be determined to fit the initial conditions in (2.2). Now the general solution of (2.1) is given by the formula, valid for $n \geq k$,

$$x_n = \sum_{i=k}^{n} g_{n-i} p(i) + \sum_{i=0}^{k-1} g_{n-i} \left( x_i - \sum_{j=1}^{i} a_j x_{i-j} \right). \tag{2.5}$$

This is essentially equation (5) of [4]. It is worth stressing that even when it is possible to give a closed form for it, the right hand side of (2.5) can be, and in practice often is, quite complicated. This is partly due to the complex numbers arising from the solution of (2.3): these can be represented by algebraic expressions or in trigonometric form and, in the sequel, we will allow for both alternatives.

The other class of recurrences that we will deal with in this paper is the class of linear recurrences with *variable coefficients* of order 1, like

$$x_n = a(n) x_{n-1} + p(n), \tag{2.6}$$

where $a \colon \mathbb{N} \to \mathbb{R} \backslash \{0\}$ is any sequence. The simplest such example is the factorial function where $a(n) = n$, $x_0 = 1$ and $p(n) = 0$ identically. The general solution of (2.6), for $n \geq 1$, is

$$x_n = \left( x_0 + \sum_{k=1}^{n} p(k) \prod_{j=1}^{k} a(j)^{-1} \right) \prod_{k=1}^{n} a(k). \tag{2.7}$$

## 3 Validating Candidate Solutions

Assume an automatic recurrence relation solver produces the closed-form answer $X_n$ for the recurrence (2.1) or for (2.6). A natural strategy for possibly validating or rejecting $X_n$ as a solution is the following: first check whether $X_n = x_n$ for $n = 0, \ldots, k-1$, i.e., that $X_n$ satisfies the initial conditions. Secondly, define $f \colon \mathbb{N} \to \mathbb{R}$ as

$$f(n) \stackrel{\text{def}}{=} X_n - (a_1 X_{n-1} + \cdots + a_k X_{n-k}) - p(n),$$

for $n \geq k$, or

$$f(n) \stackrel{\text{def}}{=} X_n - a(n) X_{n-1} - p(n),$$

for $n \geq 1$, depending on the kind of recurrence we are dealing with, and try to verify or disprove that $f$ vanishes identically.

Of course, it is the second condition that is hard to verify in practice[4], since it is possible that the simplification routines provided by the underlying computer

[4] The verification of the initial conditions, though usually easier since it is essentially a numerical problem, is not necessarily trivial, especially in the case when the characteristic equation (2.3) has irrational roots, and it seems to be fairly hard in the case of irreducible equations of degree 3.

algebra package fail to simplify completely the expression $f$. Even in the fairly trivial case of the Fibonacci numbers, checking automatically that Binet's closed formula actually holds is not really easy:[5] in fact, it is equivalent to proving that, for all $n \in \mathbb{N}$, we have

$$\phi^n - \phi^{n-1} - \phi^{n-2} - (-\phi)^{-n} + (-\phi)^{1-n} + (-\phi)^{2-n} = 0, \tag{3.1}$$

where $\phi$ and $-\phi^{-1}$ are the two roots of the characteristic equation $\lambda^2 = \lambda + 1$. It is clear that this really is the case if we collect a factor $\phi^n$ from the first three terms, and $(-\phi)^{-n}$ from the last three, and then recall that both $\phi$ and $-\phi^{-1}$ satisfy $\lambda^2 = \lambda + 1$. Much more convoluted examples arise in practice, and we thus need an efficient procedure for deciding whether an expression which is not syntactically 0 vanishes identically as a function of the non negative integer $n$.

The objective is to find a *finite* (and possibly small) set of conditions that ensure that the expression $f$ vanishes *for all* $n \in \mathbb{N}$ (an infinite set of conditions). When possible, this task should be reduced to comparisons between integers or to the numerical evaluation of suitable subexpressions of $f$ at a small number of integers. One possible general approach can be summarized as follows:

1. Deduce from either (2.1) or (2.6) the shape that any solution $X_n$ must have, i.e., how it should look like from a purely syntactical point of view.
2. Check that $X_n$ (and therefore $f$) has the appropriate shape.
3. Break down $f$ into suitable subexpressions, and prove that each one vanishes identically, by evaluating it numerically, if necessary, at a small number of integers.

The next few subsections tackle this last problem for various classes of expressions, in increasing order of difficulty. The main idea is to find the *dominant* term in the expression $f$ because this enables us to prove or disprove that they vanish identically, and also because we will use the same procedure for finding upper and lower bounds.

### 3.1 Polynomials and Exponentials

We begin with the "easy" case of the linear combination of products of polynomials and exponentials. Let $\mathbb{C}[n]$ denote the set of polynomials in the variable $n$, with coefficients in $\mathbb{C}$; $\mathrm{lead}(p)$ denotes the leading coefficient of the polynomial $p$ and $\deg(p)$ its degree, while $\mathrm{coeff}_k(p)$ is the coefficient of the monomial $n^k$. For completeness, we also state the classical result alluded to above.

**Theorem 31** *If $q \in \mathbb{C}[n]$ then either $q$ is the zero polynomial or it vanishes for at most* $\deg(q)$ *complex numbers.*

[5] It is, admittedly, a trivial example but we give it only to make our point clearer: more realistic examples, like the full exact solution of the recurrence $x_n = x_{n-1} + x_{n-2} + x_{n-3} + n$, say, would take far too much space.

We now introduce the class where, in view of (2.4) and (2.5), the solutions lie whenever $p$, the non-homogeneous part of the recurrence (2.1), is itself in the same class. It contains linear combinations of polynomials and complex exponentials, including polynomials themselves as a special case. The solutions we are concerned with are real-valued, but the definition is more general as there is no simplification in assuming that the elements of the class are also real valued.

**Definition 32 (Exponential type functions)** *Let $\mathcal{E}$ denote the set*

$$\left\{ f\colon \mathbb{N} \to \mathbb{C} \,\middle|\, \begin{array}{l} \exists r \in \mathbb{N} \,.\, \forall j = 1, \dots, r : \exists q_j \in \mathbb{C}[n] \\ \quad .\, \exists \alpha_j \in \mathbb{C} \setminus \{0\} \,.\, f(n) = \sum_{j=1}^{r} q_j(n) \alpha_j^n \end{array} \right\}.$$

Notice that the complex numbers $\alpha_j$ need not be distinct, since this would make things harder in practice: if they are, then the representation in the definition is *unique*, apart from reordering, that is, in this case the integer $r$ and the polynomials $q_j$ are uniquely determined by $f$.

The following result generalizes Theorem 31 to the class $\mathcal{E}$. It is necessary to assume that the values of the bases of the exponentials in Definition 32 be distinct: this will make the theorem difficult to use, and shows the need for some further manipulation of the expression.

**Theorem 33** *Let $f \in \mathcal{E}$ be given as in* Definition 32 *with the additional proviso that $\alpha_j \neq \alpha_k$ if $j \neq k$. Then $f$ vanishes for all $n \in \mathbb{N}$ if, and only if, $q_j$ is the constant polynomial* 0 *for all $j = 1, \dots, r$.*

Note that the function $f(n) = 1^n + (-1)^n$ vanishes for arbitrarily large values of $n$ (but not identically for $n \in \mathbb{N}$), so that the exact analogue of the situation with polynomials (with respect to Theorem 31) does not hold. This particular $f$ is the solution of the recurrence $x(n) = x(n-2)$, with $x(0) = 2$ and $x(1) = 0$, so this is not a contrived example, but one that may actually appear in practice.

Assuming $f$ is given in such a way as to satisfy the hypothesis, the problem of deciding whether it vanishes reduces, thanks to Theorem 31, to the problem of deciding whether $\mathrm{coeff}_k(q_j) = 0$ for each $j = 1, \dots, r$ and each $k = 0, \dots, D$, where $D$ is the largest degree of the $q_j$'s. We are thus left with the problem of ensuring that the expression $f$ is of the form given in the statement of Theorem 33, and therefore that the values of $\alpha_j$ are distinct. Notice that this does not happen in (3.1). Moreover, a direct approach would require the ability to compare complex numbers given by arbitrary algebraic expressions. This problem can be avoided by rearranging $f$ in a different way: instead of collecting terms with respect to the exponential functions, as in Definition 32, collect with respect to the monomials $n^d$, so that

$$f(n) = h_0(n) + n h_1(n) + \cdots + n^D h_D(n), \tag{3.2}$$

where each $h_j$ is just a linear combination of powers of the complex numbers $\alpha_k$, and therefore has the general form

$$h(n) = \beta_1 \alpha_1^n + \cdots + \beta_r \alpha_r^n \tag{3.3}$$

for suitable coefficients $\beta_1, \ldots, \beta_r$. This manipulation can be done quite efficiently and only requires comparisons between non negative integers.

The following lemma shows that it is sufficient to evaluate each $h_j$ numerically at $r_j$ consecutive integers, where $r_j$ is the number of summands in $h_j$. If any of these values is different from 0, then $f$ does not vanish for all integers, and therefore the suggested solution is not correct.

**Lemma 34** *Let $\alpha_1, \ldots, \alpha_r$ be* distinct *non-zero complex numbers, and $\beta_1, \ldots, \beta_r \in \mathbb{C}$. If there exists an integer $n_0$ such that the expression $h$ defined by* (3.3) *vanishes for $n = n_0, \ldots, n_0 + r - 1$, then $\beta_1 = \cdots = \beta_r = 0$.*

For example, in the case of (3.1), we have $D = 0$ so that we need to evaluate the left-hand side for $n = 0, \ldots, 5$ to ensure that it actually vanishes for all $n$. This can be done automatically[6], so as to obtain a proof that Binet's formula actually holds. Of course, if we are able to rewrite the left hand side as $\phi^n(1-\phi^{-1}-\phi^{-2})+(-\phi)^{-n}(-1-\phi+\phi^2)$, then only two evaluations are needed.

If it could only be proved that $h(n)$ is very small for $n = n_0, \ldots, n_0 + r - 1$, it is clear from the proof of Lemma 34 in the appendix that all the coefficients $\beta$ are very small in their own turn.

### 3.2 Rational and Trigonometric Functions

It is possible to replace "polynomial" by "rational function" in the statement of Theorem 33: this is useful in the case when $p$, the non-homogeneous part of the recurrence (2.1), is a rational function itself.

**Theorem 35** *Let $f\colon \mathbb{N} \setminus A \to \mathbb{C}$ be defined by*

$$f(n) \stackrel{\text{def}}{=} \sum_{j=1}^{r} q_j(n)\alpha_j^n,$$

*where $r \geq 1$ is an integer, $q_j \in \mathbb{C}(n)$ for $j = 1, \ldots, r$, $\alpha_j \in \mathbb{C} \setminus \{0\}$ for $j = 1, \ldots, r$, $\alpha_j \neq \alpha_k$ if $j \neq k$, and $A$ denotes the (finite) set of all zeros of all denominators of the rational functions $q_j$. Then $f$ vanishes for all $n \in \mathbb{N} \setminus A$ if, and only if, $q_j$ is the constant rational function 0 for all $j = 1, \ldots, r$.*

There is even more in Theorem 33 than meets the eye: In fact, recall the de Moivre formulæ, valid for $\theta \in \mathbb{R}$:

$$\cos(\theta z) = \frac{e^{i\theta z} + e^{-i\theta z}}{2}, \qquad \sin(\theta z) = \frac{e^{i\theta z} - e^{-i\theta z}}{2i}. \tag{3.4}$$

They can be used to transform a function of the form

$$f(n) \stackrel{\text{def}}{=} \sum_{j=1}^{r} q_j(n)\alpha_j^n + \sum_{j=r+1}^{r+s} q_j(n)\alpha_j^n \cos(\theta_j n) + \sum_{j=r+s+1}^{r+s+t} q_j(n)\alpha_j^n \sin(\phi_j n)$$

[6] For example, as in our prototype system, using the GiNaC library (`http://www.ginac.de/`).

into a function of the shape of the statement of Definition 32, and vice versa. This can be useful when the characteristic equation (2.3) has complex roots and it is preferred to work only with real functions. Notice that even if the $\alpha$'s, $\theta$'s and $\phi$'s are distinct, the resulting values of the new $\alpha$'s need not be all different. In other words, the rewritten expression may not satisfy the hypotheses of Theorem 33. The simplest example of this phenomenon is probably the function $f(n) = \sin(\pi n)$, that vanishes for all $n \in \mathbb{Z}$ without being the zero function. Since we only need to know whether $f$ vanishes for all nonnegative integers, we could plug (3.4) into $f(n)$, and then rearrange the resulting expression according to the values of the new $\alpha_j$'s. However, these substitutions do not change the degrees of the polynomials involved and the computations can be arranged as in Section 3.1, without any *explicit* substitutions, collecting with respect to monomials, and writing $f(n)$ in the form (3.2) where $D$ is the largest degree occurring in any term in $f$, and $h_d(n)$ has the form

$$h_d(n) = \sum_{j=1}^{r} \beta_{d,j} \alpha_j^n + \sum_{j=r+1}^{r+s} \beta_{d,j} \alpha_j^n \cos(\theta_j n) + \sum_{j=r+s+1}^{r+s+t} \beta_{d,j} \alpha_j^n \sin(\phi_j n)$$

for suitable complex numbers $\beta_{d,j}$. In other words, each $h_d$ is (implicitly) a linear combination of at most $r + 2s + 2t$ complex exponentials. Lemma 34 implies that $g$ vanishes at all integers if $h_d(n)$ vanishes at $r + 2s + 2t$ consecutive integers, for $d = 0, \ldots, D$.

### 3.3 Factorials

Formula (2.7) shows that natural candidates as solutions of linear recurrences with variable coefficients like (2.6) include the factorial function, or functions allied to it. In this case the results above can be somewhat generalized, since, for $n \to +\infty$, the function $n!$ grows more rapidly than any function in the class $\mathcal{E}$: This is an immediate consequence of Stirling's formula (see [15, Section 4.42]), which we state in the slightly more general form that is needed in our applications. For any *fixed* $a \in \mathbb{N} \setminus \{0\}$ and $b \in \mathbb{Z}$, for $n \to \infty$ we have

$$\begin{aligned} \log(an+b)! = an \log n + a(\log a - 1)n + \Bigl(b + \frac{1}{2}\Bigr) \log n \\ + \Bigl(b + \frac{1}{2}\Bigr) \log a + \frac{1}{2} \log(2\pi) + \mathcal{O}\bigl(n^{-1}\bigr). \end{aligned} \tag{3.5}$$

Because of the functional equation satisfied by the factorial function (namely $(n+1)! = n!(n+1)$), in some cases it may be necessary to apply the rewrite rule below before it is possible to compare the order of magnitude of expressions containing factorials, exponentials and polynomials. Assume $a \in \mathbb{N} \setminus \{0\}$ and $b \in \mathbb{Z}$. Then

$$(an+b)! = \begin{cases} (an)!(an+1) \cdots (an+b), & \text{if } b \in \mathbb{N} \setminus \{0\}; \\ (an)!, & \text{if } b = 0; \\ \dfrac{(an)!}{(an)(an-1) \cdots (an+b+1)}, & \text{if } b \in \mathbb{Z} \setminus \mathbb{N}. \end{cases}$$

**Linear Combinations of Factorials** Consider a linear combination of expressions like $(an + b)!$, where $a \in \mathbb{N} \setminus \{0\}$, and $b \in \mathbb{Z}$, with coefficients in the form of products of a rational function and an exponential. Once the rewrite rule has been applied on each factorial function in our expression, we clear denominators and are left with an expression of the form

$$f(n) = \sum_{j=1}^{r} h_j(n)\,(a_j n)!, \tag{3.6}$$

where $h_j \in \mathcal{E}$ for $j = 1, \ldots, r$. We may assume, without any loss of generality, that $a_1 = a_2 = \cdots = a_s > a_{s+1} \geq a_{s+2} \geq \cdots \geq a_r$. Once again, only comparisons between integers are required. Stirling's Formula (3.5) implies that the dominant term in (3.6) is $g(n) \stackrel{\text{def}}{=} (h_1(n) + \cdots + h_s(n))(a_1 n)!$: therefore $f(n)$ is 0 for all $n \in \mathbb{N}$ only if $g(n)/(a_1 n)!$ vanishes for all $n \in \mathbb{N}$, and we are back to the case studied in Section 3.1, since $g(n)/(a_1 n)!$ belongs to $\mathcal{E}$. Once we have verified that $g$ vanishes identically, we can consider $f - g$, and repeat.

**Products of Factorials** More generally, we consider other combinations of the factorial function, beginning with products of several such functions: once again, formula (2.7) shows that this case is relevant for our problem.

**Definition 36 (Factorial type functions)** *Let $\mathcal{F}$ denote the set*

$$\left\{ f\colon \mathbb{N} \setminus A \to \mathbb{C} \;\middle|\; \begin{array}{l} A \text{ is a finite set,} \\ \exists r \in \mathbb{N}\,.\, \forall j = 1, \ldots, r \\ \quad : \exists a_j \in \mathbb{N} \setminus \{0\}\,.\, \exists b_j \in \mathbb{Z} \\ \qquad .\; f(n) = \prod_{j=1}^{r} (a_j n + b_j)! \end{array} \right\}.$$

In order to compare rates of growth of $f$ and $g$ in $\mathcal{F}$ and to fix notation, let $a_j \in \mathbb{N} \setminus \{0\}$ and $b_j \in \mathbb{Z}$ for $j = 1, \ldots, r$, and $\alpha_j \in \mathbb{N} \setminus \{0\}$ and $\beta_j \in \mathbb{Z}$ for $j = 1, \ldots, s$. Let

$$f(n) \stackrel{\text{def}}{=} \prod_{j=1}^{r} (a_j n + b_j)! \qquad \text{and} \qquad g(n) \stackrel{\text{def}}{=} \prod_{j=1}^{s} (\alpha_j n + \beta_j)!.$$

By (3.5) we have that

$$\log \prod_{j=1}^{r} (a_j n + b_j)! \qquad \text{is dominated by} \qquad \sum_{j=1}^{r} a_j n \log n,$$

so that we can compare rates of growth of $f$ and $g$ by means of a comparison between the integers

$$\sum_{j=1}^{r} a_j \qquad \text{and} \qquad \sum_{j=1}^{s} \alpha_j. \tag{3.7}$$

If these integers are equal, we compare the second most important terms in (3.5), that is, we compare

$$\sum_{j=1}^{r} a_j(\log a_j - 1) \qquad \text{and} \qquad \sum_{j=1}^{s} \alpha_j(\log \alpha_j - 1),$$

subject to the quantities in (3.7) being equal. Taking the latter into account and simplifying, this is equivalent to comparing the integers

$$\prod_{j=1}^{r} a_j^{a_j} \qquad \text{and} \qquad \prod_{j=1}^{s} \alpha_j^{\alpha_j}. \tag{3.8}$$

It is actually possible that two *different* functions $f$ and $g \in \mathcal{F}$ have the same values in both (3.7) and (3.8), as shown by the example

$$f(n) = (2n)!^4 \qquad \text{and} \qquad g(n) = (4n)!\, n!^4.$$

If this happens, we have to look at the third highest term in (3.5), and compare

$$\sum_{j=1}^{r} \left(b_j + \frac{1}{2}\right) \qquad \text{and} \qquad \sum_{j=1}^{s} \left(\beta_j + \frac{1}{2}\right),$$

and, after multiplication by 2, we need another integer comparison between

$$r + 2\sum_{j=1}^{r} b_j \qquad \text{and} \qquad s + 2\sum_{j=1}^{s} \beta_j. \tag{3.9}$$

The functions $f(n) = (2n)!^7(2n+1)!$ and $g(n) = (4n)!^2\, n!^8$ have a common value in (3.7)–(3.9), and therefore the ratio $f(n)/g(n)$ tends to a finite, nonzero limit as $n \to +\infty$. The other terms in (3.5) only contribute a bounded amount, and they can be used, in general, to compute the value of the limit of $f(n)/g(n)$. Indeed, the limit has the value

$$\left\{(2\pi)^{r-s} \prod_{j=1}^{r} a_j^{2b_j+1} \prod_{j=1}^{s} \alpha_j^{-2\beta_j-1}\right\}^{1/2}. \tag{3.10}$$

In this case, unless $r = s$, it is impossible to confine computations within the integers, since $\pi \notin \mathbb{Q}$.

**Products of Factorials, Exponentials and Polynomials** The situation where functions belonging to $\mathcal{F}$ are freely mixed with polynomials and exponentials is even more complicated: the simplest and most striking example of this fact is possibly given by the functions $f(n) = \pi n (2n)!^2$ and $g(n) = n!^4 \cdot 2^{4n}$, whose ratio has the limit 1 as $n \to +\infty$. In this case, Stirling's formula (3.5) is not sufficient to determine the limit of $f(n) - g(n)$ as $n \to +\infty$. The question can be settled using the more precise formula where the $\mathcal{O}\big(n^{-1}\big)$ term is replaced by $1/(12n) + \mathcal{O}\big(n^{-2}\big)$.

Let $f(n) = q(n)\alpha^n g(n)$, where $q \in \mathbb{C}[n]$, $\alpha \in \mathbb{C} \setminus \{0\}$ and $g \in \mathcal{F}$ has the shape of Definition 36. If $\operatorname{lead}(q) > 0$, then for sufficiently large $n$ we have that $\log f(n)$ is dominated by $n \log n \sum_{j=1}^{r} a_j + n\big(\log\alpha + \sum_{j=1}^{r} a_j(\log a_j - 1)\big)$. It is therefore possible to repeat, to some extent, what we did in Section 3.3 and compare the order of magnitude of two such functions, though, in general, we may have to compare quantities involving logarithms of complex numbers.

The following examples show that all situations described earlier in this section may actually arise in practice, as solutions of suitable recurrences: The functions $f(n) = n!^2$ and $g(n) = (2n)!$ are the solutions of the recurrences $x(n) = n^2 x(n-1)$ with $x(0) = 1$, and $y(n) = 2n(2n-1)y(n-1)$ with $y(1) = 1$, respectively. They have the common value 2 in (3.7), but then they have the different values 1 and 4 respectively in (3.8). The functions $f(n) = (2n)!^4$ and $g(n) = (4n)!\, n!^4$ are the solutions of the recurrences $x(n) = 16n^4 x(n-1)$ with $x(0) = 1$, and $y(n) = 4n^5(4n-1)(4n-2)(4n-3)y(n-1)$ with $y(0) = 1$ respectively. They have the common value 8 in (3.7), and the common value 256 in (3.8), but then they have the different values 4 and 5 respectively in (3.9). Note that $g(n)/f(n) \sim cn^{1/2}$ as $n \to \infty$, for a suitable constant $c > 0$. This is only one of a pattern of similar functions: take $f(n) = (kn)!^{k^k}$ and $g(n) = (k^k n)!\, n!^{k^k(k-1)}$, where $k \in \mathbb{N} \setminus \{0\}$. These functions have the common value $k^{k+1}$ in (3.7), and the common value $k^{k^{k+1}}$ in (3.8), but for $k > 1$ they have the different values $k^k$ and $k^k(k-1)+1$ respectively in (3.9). The functions $f(n) = (2n)!^7(2n+1)!$ and $g(n) = (4n)!^2\, n!^8$ are the solutions of linear, first order recurrences with variable coefficients similar to the ones in the example above. They agree throughout (3.7)–(3.9). The functions $f(n) = \pi n (2n)!^2$ and $g(n) = n!^4 2^{4n}$ are the solutions of the recurrences $x(n) = 4n^3(2n-1)^2 x(n-1)/(n-1)$ with $x(1) = \pi$, and $y(n) = 16n^4 y(n-1)$ with $y(0) = 1$ respectively. Finally, notice that the identity $(n!)! = (n!-1)! \cdot n!$ is valid for all $n \in \mathbb{N}$.

### 3.4 Higher Transcendental Functions

We recall that a function $f\colon \mathbb{C} \to \mathbb{C}$ is *entire* if it is everywhere differentiable in the complex sense. Let $A \subset \mathbb{C}$ be a (possibly infinite) set of complex numbers such that the topological closure of $A$ is empty. In other words, let $A$ intersected with any circle be a finite set. For each $a \in A$ let $m_a$ be a positive integer. A Theorem of Weierstrass (see [15, Section 8.11]) implies that there is an entire function $f$ that has a zero of multiplicity $m_a$ at the point $a \in A$, and does not vanish for $z \notin A$. This means that it is very difficult to extend the results of the previous section to a larger class of functions, including higher transcendental functions like hypergeometric functions that may arise as solutions of recurrences of the form (2.6), or Euler's Gamma: for example, the entire function $1/\Gamma(-z)$ vanishes if and only if $z \in \mathbb{N}$.

## 4 Upper Bounds and Maxima

Here we deal with the problem of determining upper bounds for the exact solution of a recurrence, or for $\max\{f_1(n), f_2(n)\}$, where both $f_1$ and $f_2$ are exact solutions of some recurrences, or some upper bounds themselves. The problem is more difficult than validation, also because we are looking for rather different answers: we are interested both in *tight* bounds (and accept the fact that they may be very complicated) and in *simple* bounds (and accept that they may not be very sharp). In the latter case, we are ready to accept that the bounds may be valid only *for all sufficiently large* natural numbers. This raises the question of knowing exactly what "sufficiently large" means.

Another feature that makes the problem more difficult than validation is the possibility that, though we deal with real-valued solutions of recurrences like (2.1), the solutions themselves may contain complex numbers arising from the (possibly complex) roots of the characteristic equation (2.3). Terms involving complex roots may be rewritten as explained in Section 3.2 (actually, going the other way around) so that complex numbers disappear, but the problem is that they oscillate in sign.

The ideas explained in Section 3 can be also used to perform these tasks, and we briefly sketch the argument.

### 4.1 Upper Bounds for Functions in $\mathcal{E}$

Let $f \in \mathcal{E}$ be written as in (3.2). For each $h_d$, defined as in (3.3), let $h_d^+(n) = \sum_j \beta_j \alpha_j^n$, where the sum is restricted to the summands in $h_d$ satisfying both $\beta_j \in \mathbb{R}^+$ and $\alpha_j \in \mathbb{R}^+$, and let $h_d^-(n) = \sum_j \beta_j \alpha_j^n$, where the sum is restricted to the summands in $h_d$ satisfying $\beta_j \in \mathbb{R}^-$ and $\alpha_j \in \mathbb{R}^+$. Observe that the function $h_d^+$ is *positive* and that $h_d^-$ is *negative*: in other words, they have a definite sign for all $n \in \mathbb{N}$. Finally, let $h_d^* = h_d - h_d^+ - h_d^-$: this part, if present, is *oscillating*. For ease of reference let

$$f^+(n) \stackrel{\text{def}}{=} \sum_{j=0}^{D} h_j^+(n) n^d, \quad f^-(n) \stackrel{\text{def}}{=} \sum_{j=0}^{D} h_j^-(n) n^d, \tag{4.1}$$

$$f^*(n) \stackrel{\text{def}}{=} \sum_{j=0}^{D} h_j^*(n) n^d. \tag{4.2}$$

An upper bound for $f$ is therefore

$$f^+(n) + f^-(n) + \bigl|f^*(n)\bigr|. \tag{4.3}$$

A simpler (but weaker) one is obtained by removing the term $f^-$ from the expression above: this is the main reason for introducing the function $f^-$. An upper bound for $\bigl|f^*(n)\bigr|$ is obtained by replacing every term of the form $\beta\alpha^n$ in any $h_d^*$ by $|\beta| \cdot |\alpha|^n$: the resulting bound for the function in (4.3) belongs to the class $\mathcal{E}$, but does not contain any oscillating term.

## 4.2 The Maximum of Two Functions in $\mathcal{E}$

If both $f_1$ and $f_2$ are real-valued elements of $\mathcal{E}$, we can determine an upper bound, $U(n)$, say, for $\max\{f_1(n),\, f_2(n)\}$ as follows. Assume that $f_1$ and $f_2$ have the shape in Definition 32 for suitable values of the integers $r_1$ and $r_2$, of the polynomials $q_{1,j}$, $q_{2,j}$, and of the complex numbers $\alpha_{1,j}$, $\alpha_{2,j}$, for $j = 1, \ldots, r_1$ and for $j = 1, \ldots, r_2$, respectively. Assume further that the $\alpha$'s are distinct, and that they are labeled so that for some $s_1 \le r_1$ we have

$$|\alpha_{1,1}| = \cdots = |\alpha_{1,s_1}| > |\alpha_{1,s_1+1}| \ge \cdots \ge |\alpha_{r_1}|,$$

and let $d_{1,j} \stackrel{\text{def}}{=} \deg(q_{1,j})$, and similarly for $f_2$. Then $f_1(n)$ is dominated by the term

$$g_1(n) \stackrel{\text{def}}{=} \sum_{\substack{j=1 \\ d_{1,j}=D_1}}^{s_1} \beta_{1,j} n^{D_1} \alpha_{1,j}^n \tag{4.4}$$

where $D_1 \stackrel{\text{def}}{=} \max\{\, d_{1,j} \mid j = 1, \ldots, s_1 \,\}$, and $\beta_{1,j} = \operatorname{lead}(q_{1,j})$. In fact

$$f_1(n) = g_1(n) + o\bigl(n^{D_1}|\alpha_{1,1}|^n\bigr) \qquad \text{as } n \to +\infty, \tag{4.5}$$

since every term in $f_1(n) - g_1(n)$ has the form $\beta n^d \alpha^n$, where $\beta \in \mathbb{C}$, and either $|\alpha| < |\alpha_{1,1}|$, or $|\alpha| = |\alpha_{1,1}|$ and $d < D_1$. Let

$$g_1^+(n) \stackrel{\text{def}}{=} \sum_{\substack{j=1 \\ d_{1,j}=D_1}}^{s_1} b_{1,j} n^{D_1} |\alpha_{1,j}|^n$$

where

$$b_{1,j} \stackrel{\text{def}}{=} \begin{cases} \beta_{1,j} & \text{if } \alpha_{1,j} \in \mathbb{R}^+, \\ |\beta_{1,j}| & \text{if } \alpha_{1,j} \notin \mathbb{R}^+. \end{cases}$$

Then $g_1(n) \le g_1^+(n)$ for all $n \in \mathbb{N}$. For a real number $x$, let

$$x^+ \stackrel{\text{def}}{=} \max\{x, 0\}, \qquad x^- \stackrel{\text{def}}{=} \min\{x, 0\}.$$

Now let $U(n) := 0$ (where ':=' denotes assignment) and repeat the following procedure.

– If $|\alpha_{1,1}| = |\alpha_{2,1}|$ and $D_1 = D_2$, let

$$\begin{cases} U(n) := U(n) + |\alpha_{1,1}|^n n^{D_1} \\ \qquad \cdot \max\bigl\{(b_{1,1} + \cdots + b_{1,s_1}), (b_{2,1} + \cdots + b_{2,s_2})\bigr\}; \\ f_1(n) := f_1(n) - g_1(n); \\ f_2(n) := f_2(n) - g_2(n). \end{cases}$$

– If $|\alpha_{1,1}| < |\alpha_{2,1}|$ or ($|\alpha_{1,1}| = |\alpha_{2,1}|$ and $D_1 < D_2$), let

$$\begin{cases} U(n) := U(n) + |\alpha_{2,1}|^n n^{D_2} (b_{2,1} + \cdots + b_{2,s_2})^+; \\ f_2(n) := f_2(n) - g_2(n). \end{cases}$$

– If $|\alpha_{2,1}| < |\alpha_{1,1}|$ or ($|\alpha_{1,1}| = |\alpha_{2,1}|$ and $D_2 < D_1$), let

$$\begin{cases} U(n) := U(n) + |\alpha_{1,1}|^n n^{D_1} (b_{1,1} + \cdots + b_{1,s_1})^+; \\ f_1(n) := f_1(n) - g_1(n). \end{cases}$$

Notice that the definition of the coefficients $b$ ensures that the bound obtained is sharper than the one given by replacing every $\beta$ with its absolute value. In fact, take $f(n) = -3^n$: the latter alternative yields the upper bound $f(n) \le 3^n$, whereas our procedure gives $f$ as an upper bound to itself. We need to take positive parts of real numbers to avoid errors in computing the upper bound of, say, $f_1(n) = 3^n$ and $f_2(n) = 3^n - 2^n$.

### 4.3 The Maximum of Two Linear Combinations of Factorials

The discussion in Section 3.3 shows that essentially the same procedure can be used in the case of linear combinations of factorials, since any $f$ of the shape in (3.6) can be broken down into terms of decreasing order of magnitude (as $n \to +\infty$), in the form of the product of a function from the set $\mathcal{E}$ and $(an)!$, for some positive integer $a$. More precisely, if $f$ has the shape in (3.6) and we assume that $h_j \in \mathcal{E}$ for $j = 1, \ldots, r$, and that $a_1 \ge a_2 \ge \cdots \ge a_s > a_{s+1} \ge \cdots \ge a_r$, we use the procedure sketched in the previous section to get an upper bound $H_1^+$ and a lower bound $H_1^-$ for the function $h_1 + \cdots + h_s \in \mathcal{E}$. Notice that in this case there will be no oscillating terms, making our analysis a little simpler. We then use the same ideas iteratively on the function $f(n) - (h_1(n) + \cdots + h_s(n))(an)!$, that has again the shape (3.6) for suitable values of the parameters, adding together the upper bounds (resp. the lower bounds) found in the successive iterations.

### 4.4 Factorial Type Functions

Here we can use the procedure outlined in Section 3.3 to compare functions belonging to $\mathcal{F}$: deciding which is the larger of $f_1$ and $f_2$ in $\mathcal{F}$ can be done via successive comparisons of the quantities in (3.7)–(3.9). If this is not enough to settle the question, it means that the ratio $f_1/f_2$ has a non-zero, finite limit as $n \to +\infty$, and therefore the constant terms in (3.5) come into play. We then compute the expression in (3.10) and if this is larger than 1, then $f_1$ is larger than $f_2$ for sufficiently large $n$; if it is smaller than 1, then $f_2$ is larger, and if it is exactly 1, then $f_1$ and $f_2$ have precisely the same behavior as $n \to \infty$, and then we need a more precise form of Stirling's Formula, as on a previous occasion in Section 3.3.

## 5 Lower Bounds and Minima

The strategy for obtaining lower bounds and minima is very similar to the one described in Section 4. Let $f \in \mathcal{E}$ be written as in (4.1)–(4.2). A lower bound for $f$ is

$$f^{+}(n) + f^{-}(n) - \bigl|f^{*}(n)\bigr|.$$

For minima, we define $g^{-}$ exactly as $g^{+}$, but with the coefficients

$$b_{1,j} \stackrel{\text{def}}{=} \begin{cases} \beta_{1,j} & \text{if } \alpha_{1,j} \in \mathbb{R}^{+}, \\ -|\beta_{1,j}| & \text{if } \alpha_{1,j} \notin \mathbb{R}^{+}. \end{cases}$$

Then $g_1(n) \geq g_1^{-}(n)$ for all $n \in \mathbb{N}$. Now let $L(n) := 0$, and repeat the following procedure.

- If $|\alpha_{1,1}| = |\alpha_{2,1}|$ and $D_1 = D_2$, let

$$\begin{cases} L(n) := L(n) + |\alpha_{1,1}|^{n} n^{D_1} \\ \qquad \cdot \min\bigl\{(b_{1,1} + \cdots + b_{1,s_1}), (b_{2,1} + \cdots + b_{2,s_2})\bigr\}; \\ f_1(n) := f_1(n) - g_1(n); \\ f_2(n) := f_2(n) - g_2(n). \end{cases}$$

- If $|\alpha_{1,1}| < |\alpha_{2,1}|$ or ($|\alpha_{1,1}| = |\alpha_{2,1}|$ and $D_1 < D_2$), let

$$\begin{cases} L(n) := L(n) + |\alpha_{2,1}|^{n} n^{D_2} (b_{2,1} + \cdots + b_{2,s_2})^{-}; \\ f_2(n) := f_2(n) - g_2(n). \end{cases}$$

- If $|\alpha_{2,1}| < |\alpha_{1,1}|$ or ($|\alpha_{1,1}| = |\alpha_{2,1}|$ and $D_2 < D_1$), let

$$\begin{cases} L(n) := L(n) + |\alpha_{1,1}|^{n} n^{D_1} (b_{1,1} + \cdots + b_{1,s_1})^{-}; \\ f_1(n) := f_1(n) - g_1(n). \end{cases}$$

The procedures for obtaining the minimum of two linear combinations of factorials, or of two functions in the class $\mathcal{F}$, are explained in Sections 4.3 and 4.4 respectively, since they are quite similar to those needed to compute the maximum.

We are now in position to specify the set $\mathfrak{B}$ of boundary functions that we consider in this paper: it is the set of linear combinations of elements of $\mathcal{F}$ with coefficients in

$$\left\{ f \colon \mathbb{N} \to \mathbb{R} \;\middle|\; \begin{array}{l} \exists r \in \mathbb{N} \,.\, \forall j = 1, \ldots, r : \exists q_j \in \mathbb{R}[n] \\ \quad .\, \exists \alpha_j \in \mathbb{R}^{+} \,.\, f(n) = \sum_{j=1}^{r} q_j(n) \alpha_j^{n} \end{array} \right\},$$

which is closed with respect to the algorithms we gave for computing lower and upper bounds.

## 6 Conclusion and Future Work

Recurrence relations play an important role in the field of complexity analysis since complexity measures of programs and procedures can often be elegantly expressed by systems of such relations. Their mechanical solution is, also for this reason, an interesting application of symbolic computation.

In this paper we have studied two problems related to the manipulation of (candidate) solutions of recurrence relations: proving or disproving that a closed-form expression actually solves a recurrence relation and computing non-trivial lower and upper bounds to finite sets of such expressions. An important feature of the techniques we propose is that they rely as much as possible on integer computations and, more generally, can be efficiently implemented.

A prototype implementation is being developed in the context of the PURRS project (`http://www.cs.unipr.it/purrs/`), and the part concerning the correctness test for the solutions is almost finished and in current use. At the time of writing, regression testing of the PURRS solver is done on a set of 1286 (generalized) recurrences collected from various sources. Of those, 612 are linear recurrences of finite order with constant coefficients, 579 of which can be solved by the current version of the system. There are 542 recurrences in this class that can be verified by the method proposed in Section 3. On 2 of them the naive method does not terminate in reasonable time. The average speedup granted by the new method on the remaining 540 recurrences is 16, whereas the maximum speedup measured for the verification of one single recurrence is about 200 (this occurs for the verification of $x_n = 2x_{n-1} + x_{n-2} + n^3$). Slowdowns are also possible, but we have only observed them for recurrences where the cost of verification is very small, on the order of milliseconds. On this timescale, our measurements of CPU time are affected by very large relative errors, to the point that a small speedup can be mistaken for a slowdown. This situation arises for 39 recurrences. There are still 37 recurrences for which we are unable to check the solution provided by PURRS for correctness (for 33 of them the verification causes system thrashing using both methods, whereas for the remaining 4 the simplifier is not powerful enough): this is, of course, material for future work.

On the subject of future work, there is often the need, given two sets of sequences $F_1, F_2 \in \mathfrak{F}$, to know whether one set majorizes the other, e.g., whether we have, for each $f_1 \in F_1$ and each $f_2 \in F_2$,

$$\forall n \in \mathbb{N} : f_1(n) \leq f_2(n). \tag{6.1}$$

This is required, for instance, in order for an optimizing program transformer to automatically decide whether a candidate transformation resulted into an actual improvement [16]. This problem reduces to the problem of *approximating*, as precisely as possible, the decision problem of the truth of $b_1 \leq b_2$ given $b_1, b_2 \in \mathfrak{B}$. (Notice the emphasis on approximation: as is customary in the field of static analysis, "yes", "no" and "don't know" are all sensible answers.)

Future work includes, in addition, the extension to more general recurrences than (2.1) and (2.6), also in those cases where it is possible to give lower and upper approximations for the solutions, but not a closed form. In this case, checking that approximations are correct involves solving the decision problem (6.1). We expect the work on such inequalities to be generally beneficial. For instance, it can be used to improve the procedure outlined in Section 4.2, which may produce an upper bound that is not the best possible (as the example with $f_1(n) = n$ and $f_2(n) = n^2$ shows).

*Acknowledgments* We wish to thank Tatiana Zolo and Enea Zaffanella for the help they gave us in the implementation of these ideas. We are also grateful to the reviewers of a previous version of this paper for their careful comments that helped us improve the paper.

## References

1. Bagnara, R., Zaccagnini, A., and Zolo, T. The automatic solution of recurrence relations. I. Linear recurrences of finite order with constant coefficients. Quaderno 334, Dipartimento di Matematica, Università di Parma, Italy, 2003. Available at `http://www.cs.unipr.it/Publications/`.
2. Blieberger, J., and Lieger, R. Worst-case space and time complexity of recursive procedures. *Real-Time Systems 11*, 2 (1996), 115–144.
3. Celis, P. Remark: Corrections and errors in John Ivie's some MACSYMA programs for solving recurrence relations. *ACM Transactions on Mathematical Software 10*, 4 (1984), 477–478. See [10].
4. Cohen, J., and Katcoff, J. Symbolic solution of finite-difference equations. *ACM Transactions on Mathematical Software 3*, 3 (1977), 261–271.
5. Cousot, P., and Cousot, R. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the Fourth Annual ACM Symposium on Principles of Programming Languages* (New York, 1977), ACM Press, pp. 238–252.
6. Debray, S., and Lin, N.-W. Cost analysis of logic programs. *ACM Transactions on Programming Languages and Systems 15*, 5 (1993), 826–875.
7. Debray, S. K., López-García, P., Hermenegildo, M. V., and Lin, N.-W. Lower bound cost estimation for logic programs. In *Logic Programming: Proceedings of the 1997 International Symposium* (Port Washington, NY, USA, 1997), J. Małuszyński, Ed., MIT Press Series in Logic Programming, The MIT Press, pp. 291–305.
8. Flajolet, P., Salvy, B., and Zimmermann, P. Automatic average-case analysis of algorithms. *Theoretical Computer Science 79*, 1 (1991), 37–109.
9. Gourdon, X., and Salvy, B. Effective asymptotics of linear recurrences with rational coefficients. *Discrete Mathematics 153*, 1-3 (1996), 145–163.
10. Ivie, J. Some MACSYMA programs for solving recurrence relations. *ACM Transactions on Mathematical Software 4*, 1 (1978), 24–33. See also [3].
11. Le Métayer, D. ACE: An automatic complexity evaluator. *ACM Transactions on Programming Languages and Systems 10*, 2 (1988), 248–266.
12. Petkovšek, M., Wilf, H. S., and Zeilberger, D. $A = B$. A. K. Peters, Natick, MA, 1996.
13. Rosendahl, M. Automatic complexity analysis. In *Proceedings of the 4th International Conference on Functional Programming Languages and Computer Architecture* (Imperial College, London, U.K., 1989), ACM Press, pp. 144–156.

14. Sands, D. Complexity analysis for a lazy higher-order language. In *Proceedings of the 3rd European Symposium on Programming* (Copenhagen, Denmark, 1990), vol. 432 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 361–376.
15. Titchmarsh, E. C. *The Theory of Functions.* Oxford University Press, London, UK, 1988.
16. Wegbreit, B. Goal-directed program transformation. In *Proceedings of the 3rd ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages* (Atlanta, Georgia, 1976), ACM Press, pp. 153–170.
17. Wilf, H. S. *Generatingfunctionology.* Academic Press, New York, 1994.
18. Zimmermann, P., and Zimmermann, W. Automatic complexity analysis of divide-and-conquer algorithms. In *Proceedings of the 6th International Symposium on Computer and Information Sciences* (Antalya, Turkey, 1991), M. Baray and B. Ozguc, Eds., Elsevier Science Publishers, pp. 395–404.

# A Proofs

**Proof of Theorem 33 on page 7.** Assume that the $\alpha$'s are labeled in such a way that for some $s \in \{1, \ldots, r\}$ we have $|\alpha_1| = \cdots = |\alpha_s| > |\alpha_{s+1}| \geq \cdots \geq |\alpha_r|$, and let $d_j \stackrel{\text{def}}{=} \deg(q_j)$. For brevity we refer to (4.4) and (4.5) with the suffix 1 removed throughout. As in (4.5), for large $n$ the function $f$ is dominated by $g(n)$, which is defined as in (4.4). If $s = 1$ then $g(n) = \mathrm{lead}(q_1)n^{d_1}\alpha_1^n$ and therefore $f$ does not vanish for large enough $n$. If $s > 1$, then let

$$g(n) \stackrel{\text{def}}{=} \sum_{\substack{j=1 \\ d_j = D}}^{s} \beta_j n^D \alpha_j^n$$

where $D \stackrel{\text{def}}{=} \max\{\, d_j \mid j = 1, \ldots, s \,\}$, and $\beta_j = \mathrm{lead}(q_j)$. Let $g(n) = n^D h(n)$. Again, we have that (4.5) holds, and this implies that the function $f$ can not vanish for *all* large $n$. Indeed, we prove in Lemma A1 below that $h(n)/\alpha_1^n$ does not have the limit 0 as $n \to +\infty$, unless all $\beta$'s are 0 (which is against their definition). In other words, there is a *positive* constant $c$ such that for some arbitrarily large values of $n$ we have $|h(n)/\alpha_1^n| > c$, and for these values of $n$ the function $f$ does not vanish. $\square$

**Proof of Lemma 34 on page 8.** By hypothesis, we have that

$$(\beta_1 \alpha_1^{n_0})\alpha_1^n + \cdots + (\beta_r \alpha_r^{n_0})\alpha_r^n$$

vanishes for $n = 0, \ldots, r-1$. Therefore, changing if necessary the value of the coefficient $\beta_j$ by the multiplicative factor $\alpha_j^{n_0}$ (which does not affect the result), we can assume that $n_0 = 0$. But the Vandermonde determinant

$$\det \begin{bmatrix} 1\ \alpha_1\ \alpha_1^2 \ldots\ \alpha_1^{r-1} \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ 1\ \alpha_r\ \alpha_r^2 \ldots\ \alpha_r^{r-1} \end{bmatrix} = \prod_{i<j} (\alpha_j - \alpha_i) \neq 0$$

since the $\alpha$'s are distinct. Therefore the system in the unknowns $\beta_1, \dots, \beta_r$

$$\begin{cases} \beta_1 + \beta_2 + \ldots + \beta_r = 0 \\ \dots\dots\dots\dots\dots\dots\dots\dots \\ \beta_1\alpha_1^{r-1} + \beta_2\alpha_2^{r-1} + \ldots + \beta_r\alpha_r^{r-1} = 0 \end{cases}$$

has only the trivial solution $\beta_1 = \cdots = \beta_r = 0$. $\square$

**Lemma A1** *Let $\omega_1, \dots, \omega_r$ be* distinct *complex numbers such that $|\omega_j| = 1$ for all $j$, and let $\alpha_1, \dots, \alpha_r$ be non-zero complex numbers. Then either $r = 1$ and $\omega_1 = 1$, or the sequence*

$$h(n) \stackrel{\text{def}}{=} \alpha_1\omega_1^n + \cdots + \alpha_r\omega_r^n$$

*does not have any limit as $n \to +\infty$.*

**Proof.** If $r = 1$ and $h(n) \to L$, then $h(n) = L + o(1)$ as $n \to \infty$, and $|h(n) - h(n+1)| = |\alpha_1\omega_1^n - \alpha_1\omega_1^{n+1}| = |\alpha_1(1-\omega_1)| = o(1)$: this implies $\omega_1 = 1$ since $\alpha_1 \neq 0$.

Now assume that $r \geq 2$, and that the statement has been proved for $r-1$. If $r > 1$ and $h(n)$ had the limit 0, then consider the sequence $g(n) = h(n)\omega_r^{-n} - \alpha_r$, which has the shape considered in the statement, with $\omega_j' = \omega_j\omega_r^{-1}$ for $j = 1, \dots, r-1$, and $r-1$ in place of $r$. Since $\omega_j' \neq 1$ for all $j$, the sequence $g(n)$ has no limit by the induction hypothesis, and therefore $h$ does not have the limit 0. If $\omega_r$ were 1, then $g(n) = h(n) - \alpha_r$ would have the limit $L - \alpha_r$, and this is again impossible by the induction hypothesis.

We may assume from now on that $h(n)$ has the limit $L \neq 0$ and that $\omega_j \neq 1$ for all $j$. We choose $\epsilon = \frac{1}{3}|L| > 0$; for $n \geq n_0(\epsilon)$ we have $h(n) = L + \delta_n$, where $|\delta_n| < \epsilon$. Set $s_{n,k} = h(n) + \cdots + h(n+k)$. On the one hand we have

$$|s_{n,k}| = \Bigl|\sum_{j=1}^{r} \alpha_j\omega_j^n \frac{1-\omega_j^{k+1}}{1-\omega_j}\Bigr| \leq \sum_{j=1}^{r} \frac{2|\alpha_j|}{|1-\omega_r|},$$

which is a fixed, finite quantity, independent of $n$ and $k$. On the other hand, for $n \geq n_0$ we have

$$|s_{n,k} - (k+1)L| \leq (k+1)\epsilon = \frac{1}{3}(k+1)|L| \quad \Longrightarrow \quad |s_{n,k}| \geq \frac{2}{3}(k+1)|L|,$$

by the triangle inequality, which is absurd for large $k$. $\square$

**Proof of Theorem 35 on page 8.** Let $q_j(z) = a_j(z)/b_j(z)$, where $a_j, b_j \in \mathbb{C}[z]$ are polynomials, and $b_j$ is not the 0 polynomial. Let $g(z) \stackrel{\text{def}}{=} f(z)b_1(z)\cdots b_r(z)$, so that $g$ has the shape of the statement of Theorem 33. By hypothesis, $g(n)$ vanishes for all large $n$, and the result follows. $\square$