Verified Solving and Asymptotics of Linear Recurrences

Manuel Eberl
Institut für Informatik
Technische Universität München
Garching bei München, Germany
eberlm@in.tum.de

Abstract

Linear recurrences with constant coefficients are an interesting class of recurrence equations that can be solved explicitly. The most famous example are certainly the Fibonacci numbers with the equation $f(n) = f(n-1) + f(n-2)$ and the quite non-obvious closed form

$$\frac{1}{\sqrt{5}}(\varphi^n - (-\varphi)^{-n})$$

where φ is the golden ratio.

This work builds on existing tools in Isabelle – such as formal power series and polynomial factorisation algorithms – to develop a theory of these recurrences and derive a fully executable solver for them that can be exported to programming languages like Haskell.

Based on this development, I also provide an efficient method to prove 'Big-O' asymptotics of a solution automatically without explicitly finding the closed-form solution first.

CCS Concepts • **Mathematics of computing → Generating functions**; *Solvers*;

Keywords linear recurrences, generating functions, asymptotics, formal power series, Fibonacci, Isabelle, theorem proving

ACM Reference Format:
Manuel Eberl. 2019. Verified Solving and Asymptotics of Linear Recurrences. In *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP '19), January 14–15, 2019, Cascais, Portugal.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3293880.3294090

CPP '19, January 14–15, 2019, Cascais, Portugal

ACM ISBN 978-1-4503-6222-1/19/01…$15.00
https://doi.org/10.1145/3293880.3294090

1 Introduction

This paper is about verifying the theory, the asymptotics, and an executable solver for linear recurrences with constant coefficients. It supports both homogeneous recurrences and inhomogeneous recurrences where the inhomogeneous part is from a certain class. From this point onward, I will use the term 'linear recurrence' and implicitly mean 'linear recurrence in one variable with constant coefficients'.

The most famous such recurrence is certainly the one defining the *Fibonacci numbers*:

$$F_0 = 0 \qquad F_1 = 1 \qquad F_n = F_{n-1} + F_{n-2}$$

They are named after the 12th-century Italian mathematician Leonardo of Pisa – who is nowadays known better by the name *Fibonacci* – although it was studied by Indian mathematicians much earlier in connection with the possible patterns in the metre of Sanskrit prosody. Fibonacci, on the other hand, presented them in the context of a puzzle about the population growth of rabbits: assuming a pair of adult rabbits produces a new pair of baby rabbits every month, rabbits take one month to mature, and rabbits never die, what is the number of adult rabbit pairs after n months, starting with a single pair of baby rabbits?

The Fibonacci numbers have a number of very interesting properties and occur in many places in mathematics. They can be written as the closed-form expression

$$F_n = \frac{\varphi^n - \psi^n}{\sqrt{5}}$$

where $\varphi = \frac{1}{2}(1 + \sqrt{5})$ is the golden ratio and $\psi = 1 - \varphi = \frac{1}{2}(1 - \sqrt{5})$.

This may seem surprising since the closed form contains irrational constants that do not cancel in an obvious way even though the Fibonacci numbers are all natural numbers.

This closed form directly implies the asymptotic estimate $f(n) \sim \varphi^n/\sqrt{5}$, which is very accurate – it turns out, in fact, that $f(n) = [\varphi^n/\sqrt{5}]$ for all positive n, where $[\cdot]$ is the 'nearest integer' function.

More generally, such recurrences arise in certain enumeration problems: the number of steps required to solve the 'Tower of Hanoi', the number of ordered partitions of integers, or enumerating all lists of a given length with some

additional restrictions (e. g. forbidden patterns). They also arise in the average-time analysis of recursive algorithms [6], in the analysis of imperative programs with loops [12, 17], and in the analysis of probabilistic algorithms like random walks[11].

It is therefore of great interest that these recurrences can be solved automatically. The key is the so-called *characteristic polynomial*, which can be read off directly from the recurrence equation. The roots of this polynomial determine the general shape of the solution, whereas the precise coefficients depend on the initial values. For example, the characteristic polynomial of the Fibonacci recurrence $F_n = F_{n-1} + F_{n-2}$ is $x^2 - x - 1$, whose roots are φ and ψ, so that the solution must have the general form $c_1\varphi^n + c_2\psi^n$ independently of the initial values. The initial values do, however, determine the values of c_1 and c_2.

Developing the theory behind this in Isabelle/HOL can be done in the most nice and abstract way using *formal power series* (FPSs); in particular, using the well-known correspondence between linear recurrences with constant coefficients and rational FPSs, i. e. power series that are of the form p/q for complex polynomials p, q with $q \neq 0$.

In the end, I will use some existing Isabelle tools to derive a fully automatic solver for linear recurrences with constant coefficients. Furthermore, I provide a more efficient tool to certify 'Big-O' asymptotic bounds for the solution of such a recurrence. To my knowledge, this is the first general treatment of linear recurrences in a proof assistant. The development is available as an entry in the *Archive of Formal Proofs* [4]. At the time of writing, a part of it is only available in the *development version* of the Archive (https://devel.isa-afp.org).

2 Mathematical Basics

2.1 Formal Power Series

A key ingredient in the textbook approach to solving linear recurrences are *formal power series* (FPSs) in the form of *ordinary generating functions*. These are purely formal objects; their purpose is nothing but to have a nice algebraic object that represents a sequence.

Formally, the commutative (semi-)ring of FPSs over a commutative (semi-)ring R with formal parameter X is written as $R[[X]]$, and its elements are written e. g. as

$$A(X) = \sum_{n=0}^{\infty} a_n X^n ,$$

where a_n is the sequence of coefficients. Let us denote a_n – the n-th coefficient of the power series A – with $[X^N]\,A$, and call $a_0 = [X^0]\,A$ the *constant coefficient*. In analogy to polynomials, the basic operations are defined as:

$$0 = \sum_{n=0}^{\infty} 0 \cdot X^n$$

$$1 = \sum_{n=0}^{\infty} (\text{if } n = 0 \text{ then } 1 \text{ else } 0) X^n$$

$$A(z) + B(z) = \sum_{n=0}^{\infty} (a_n + b_n) X^n$$

$$A(z) \cdot B(z) = \sum_{n=0}^{\infty} \left(\sum_{i=0}^{n} a_i b_{n-i} \right) X^n$$

From now one, we shall always restrict ourselves to the case where the underlying ring is a field K, and at some point, we will also introduce the additional assumption that K is algebraically closed. In practice, this field will therefore be the complex numbers, but the theory is developed as generally as possible.

In the polynomial ring $K[X]$, the only units (i. e. elements with a multiplicative inverse) are then the constant non-zero polynomials. In $K[[X]]$, on the other hand, all FPSs with non-zero constant coefficient are invertible with

$$A(z)^{-1} = \sum_{n=0}^{\infty} b_n X^n \quad \text{for } b_n = \begin{cases} \frac{1}{a_0} & \text{for } n = 0 \\ -\frac{1}{a_0} \sum_{i=1}^{n} a_i b_{n-i} & \text{otherwise} \end{cases}$$

It is clear that there exists an injective canonical homomorphism $K[X] \to K[[X]]$. One therefore implicitly identifies polynomials in $K[X]$ with the corresponding FPS in $K[[X]]$. Furthermore, if $p, q \in K[X]$ and $q(0) \neq 0$ (i. e. the constant coefficient of q is non-zero), then q is a unit in $K[[X]]$ and the quotient $p(x)/q(x) \in K[[X]]$ is well-defined. Let us call FPSs of this form *rational*, and they form a sub-ring of $K[[X]]$. Algebraically speaking, this ring of rational FPSs is the localisation of $K[X]$ w. r. t. $\{p \mid p(0) \neq 0\}$.

Note that this ring of rational FPSs is smaller than the ring of rational functions $K(X)$: The latter contains e. g. $1/X$, whereas the former does not. The rational FPSs are essentially those elements of $K(X)$ that do not have a pole at the origin.

This sub-ring is what we are interested in, since rational FPSs are precisely those whose coefficients satisfy linear recurrences. I therefore defined the Isabelle type *ratfps* that corresponds to this sub-ring and connected it to the FPS type with the injective canonical homomorphism *α ratfps → α fps* (where *α* is the type of the coefficients) on one side and to the fraction field of polynomials with an injective homomorphism *α ratfps → α poly fract* on the other side. Isabelle's Code Generator was set up to perform computations on rational FPSs by representing the quotient as a pair of coprime polynomials where the second one (the denominator) must be monic and non-zero (which is a convenient unique representation). This makes it possible to write the solving algorithms in a very abstract way, operating directly

on FPSs, and still directly generate executable code from these definitions.

A standard approach to solve recurrences is the following:

1. Use the recurrence to find a simple equation that characterises the FPS of the sequence.
2. Solve that equation to find a closed-form expression for the FPS.
3. Use algebraic transformations to break down the FPS into simpler parts.
4. Read off the solution as the coefficients of these simple parts.

In our case, more concretely, the steps are the following:

1. Write the inhomogeneous part as an FPS (in our case: as a rational FPS).
2. Use this to write the entire recurrence as a (rational) FPS.
3. Factor the denominator of the FPS into linear polynomials of the form $1 - c_i X$.
4. Perform *Partial Fraction Decomposition* to bring this rational FPS into the form.
$$p(X) + \sum_{i=1}^{k} \frac{b_i}{(1 - c_i X)^{k_i}}$$
for a polynomial $p \in K[X]$ and numbers $b_i, c_i \in K$.
5. Read off the coefficients from this sum.

2.2 The Homogeneous Part

Let us first consider a homogeneous recurrence. To obtain a more uniform presentation, let us assume that it is given in the form

$$\forall n \geq l.\ c_0 f(n) + c_1 f(n+1) + \ldots + c_m f(n+m) = 0 \qquad (1)$$

with $m + l$ base cases (otherwise, the sequence would not be uniquely defined). The number l is the number of 'excess' base cases and will usually be 0.

Let $F(X)$ be the generating function of f and define the polynomial q as

$$q(X) = c_0 X^m + c_1 X^{m-1} + \ldots + c_m\ .$$

Then, for any $n \geq m + l$, we have:

$$\begin{aligned}
&[X^n]\,(q(X)F(X)) \\
&\quad = [X^n]\left(c_0 X^m F(X) + c_1 X^{m-1} F(X) + \ldots + c_m F(X)\right) \\
&\quad = c_0 f(n-m) + c_1 f(n-m+1) + \ldots + c_m f(n) \overset{(1)}{=} 0
\end{aligned}$$

Therefore, $p(X) := q(X)F(X)$ is a polynomial of degree at most $m + l$. Its coefficients can easily be calculated as

$$[X^n]p(X) = \sum_{i=0}^{\min(m,n)} c_{m-i} f(n-i)\ .$$

In other words, we now have $F(X) = p(X)/q(X)$. Note that if $c_m \neq 0$, which is a reasonable demand to make from a well-formed recurrence equation, the constant coefficient of the denominator $q(X)$ is also non-zero.

2.3 The Inhomogeneous Part

Treating inhomogeneous recurrences can be done in a very similar way as homogeneous ones. In analogy to (1), we now consider recurrences of the form

$$\forall n \geq l.\ c_0 f(n) + c_1 f(n+1) + \ldots + c_m f(n+m) = g(n+m) \qquad (2)$$

We again let $q(X)$ be as before and obtain:

$$\begin{aligned}
&[X^n]\,(q(X)F(X)) \\
&\quad = [X^n]\left(c_0 X^m F(X) + c_1 X^{m-1} F(X) + \ldots + c_m F(X)\right) \\
&\quad = c_0 f(n-m) + c_1 f(n-m+1) + \ldots + c_m f(n) \overset{(2)}{=} g(n)
\end{aligned}$$

If we let $G(X)$ be the generating function of the inhomogeneous part g, we therefore know that $p(X) := q(X)F(X) - G(X)$ is a polynomial of degree at most $m + l - 1$, and, analogously to before, its coefficients can be calculated as

$$[X^n]p(X) = \left(\sum_{i=0}^{\min(m,n)} c_{m-i} f(n-i)\right) - g(n)$$

and we have $F(X) = (p(X) + G(X))/q(X)$. In particular, this means that if $G(X)$ is rational, $F(X)$ is as well.

The remaining question is how to go from the closed form of the sequence $g(n)$ to a rational generating function $G(X)$. However, since $G(X)$ is rational iff $g(n)$ can be written as a sum of terms of the form $cn^k a^n$ for $c, a \in \mathbb{C}$ and $k \in \mathbb{N}$, it suffices to determine what the generating function of $n^k a^n$ is. For this purpose, let $B_k(X) := \sum_{n=0}^{\infty} n^k X^n$. We obviously have:

$$\sum_{n=0}^{\infty} n^k a^n X^n = B_k(aX)$$

We therefore only have to determine $B_k(X)$. As it turns out,

$$B_k(X) = \begin{cases} 1/(1-X) & \text{for } k = 0 \\ XE_k(X)/(1-X)^{k+1} & \text{otherwise} \end{cases}$$

where $E_k(X)$ is the k-th Eulerian polynomial, defined as:

$$E_k(X) = \begin{cases} 1 & \text{for } k = 0 \\ (nX - X + 1)\,E_{k-1}(X) & \\ \quad + (X - X^2)\,E'_{k-1}(X) & \text{otherwise} \end{cases}$$

This is easily verified by induction over k. In conclusion, we have

$$\sum_{n=0}^{\infty} n^k a^n X^n = \begin{cases} 1/(1-aX) & \text{for } k = 0 \\ aXE_k(aX)/(1-aX)^{k+1} & \text{otherwise} \end{cases}$$

and can therefore write the inhomogeneous part as a rational FPS as long as it is given in polynomial-exponential form.

From now on, we will consider the following recurrence as a running example:

$$f(n) - f(n-1) - 2f(n-2) = n\,2^n \qquad f(0) = f(1) = 0$$

By the above result, the generating function of the inhomogenous part $n\,2^n$ is $G(X) = 2X/(1-2X)^2$ (since $E_1 = 1$). According to the above definitions, we compute $p(X) = -2X$ and $q(X) = -2X^2 - X + 1$, leading to the generating function

$$\sum f(n)X^n = \frac{p(X)+G(X)}{q(X)} = \frac{8X^3 - 8X^2}{8X^4 - 4X^3 - 6X^2 + 5X - 1}\,.$$

2.4 Factoring the Denominator

We now have the generating function for the recurrence in the form $p(X)/q(X)$ with $q(0) \neq 0$, and w. l. o. g. we can assume $q(X)$ to be monic. The next step is to factor $q(X)$ in order to break down the generating function into easier terms.

If the field K is algebraically closed, the $q(X)$ can always be factored into the form

$$d\,(1-c_1X)^{n_1}\ldots(1-c_kX)^{n_k} \qquad (\text{for } d, c_i \in K)$$

Thiemann *et al.* [13, 14] have implemented several methods of factoring real and complex polynomials in Isabelle/HOL, using an implementation of algebraic real numbers based on *Sturm sequences*. Their algorithm produces a factorisation in terms of linear factors of the form $X - c$; applying it to q^R (the reflected polynomial) yields a factorisation in terms of linear factors $1 - cX$, as desired.

For our running example, factoring the denominator into such linear factors yields

$$8X^4 - 4X^3 - 6X^2 + 5X - 1 = -(1+X)(1-2X)^3\,.$$

2.5 Partial Fraction Decomposition

We now have our FPS in the form

$$\frac{p}{(1-c_1X)^{n_1}\ldots(1-c_kX)^{n_k}}$$

and we want to break this up into simpler summands. This can be done with *Partial Fraction Decomposition*, which operates, more generally, on a quotient of the form

$$\frac{p}{q_1^{n_1}\ldots q_k^{n_k}}$$

where all the q_k are pairwise coprime. Partial Fraction Decomposition then brings this quotient into the form

$$r + \sum_{i=1}^{k}\sum_{j=1}^{n_i}\frac{s_{ij}}{q_i^j}$$

where $r, s_{ij} \in K[X]$ and each s_{ij} has a degree less than that of q_i. In particular, if the q_i have degree 1 (which is the case here), the s_{ij} must all be constants.

To do this, consider the fraction $p/(qr)$ for $\gcd(q,r) = 1$. Then the extended Euclidean algorithm gives us $s, t \in K[X]$ with $sq + tr = 1$ and therefore

$$\frac{p}{qr} = \frac{pt}{q} + \frac{ps}{r}$$

Iterating this process on our original quotient gives us the following decomposition:

$$\frac{p}{q_1^{n_1}\ldots q_k^{n_k}} = \sum_{i=1}^{k}\frac{r_i}{q_i^{n_i}} \tag{3}$$

Iterated polynomial division by q_i on each summand yields

$$\begin{aligned}\frac{r_i}{q_i^{n_i}} &= \frac{s_{i,n_i} + q_i s_{i,n_i-1} + \ldots + q_i^{n_i-1}s_{i,1} + q_i^{n_i}s_{i,0}}{q_i^{n_i}}\\ &= \frac{s_{i,n_i}}{q_i^{n_i}} + \frac{s_{i,n_i-1}}{q_i^{n_i-1}} + \ldots + \frac{s_{i,1}}{q_i} + s_{i,0}\end{aligned}$$

Collecting the $s_{i,0}$ into a single polynomial r then gives us the desired form.

Alternatively, one can avoid the polynomial division in the last step: Since the polynomials q_i have the form $1 - cX$ in our case, the summands in (3) have the form

$$\frac{r_{n-1}X^{n-1} + \ldots + r_0}{(1-cX)^n}$$

and we can apply a Binomial transform to the r_i to express the numerator as a polynomial in $1 - cX$. This may yield slightly better performance and could be implemented as future work, but the n_i are typically relatively low anyway.

Applied to our running example, the decomposition is:

$$-\frac{8X^3 - 8X^2}{(1+X)(1-2X)^3} = \frac{\frac{16}{27}}{1+X} + \frac{\frac{2}{3}}{(1-2X)^3} - \frac{\frac{4}{9}}{(1-2X)^2} - \frac{\frac{22}{27}}{1-2X}$$

2.6 Constructing the Solution

We now have the FPS in the form

$$r(X) + \sum_{i=1}^{k}\sum_{j=1}^{n_0}\frac{a_{ij}}{(1-c_iX)^j}\,.$$

To find the closed-form solution of the original recurrence, we must now extract the coefficients from this FPS. We can again do this for every summand individually.

For the polynomial summand r, the solution is obvious: The n-th coefficient of r as an FPS is simply the n-th coefficient of the polynomial r. If n is larger than the degree of r, we have $[X^n]\,r = 0$. The polynomial summand r can therefore be seen as an adjustment term that influences only the first few elements of the sequence. This is necessary when the recurrence is 'overspecified', i. e. there are more initial conditions given than necessary to define the sequence uniquely.

For the other summands, it suffices to consider the FPS $(1-cX)^{-j}$ for $j > 0$. Using the FPS form of the generalised

Binomial Theorem, we find that

$$\begin{aligned}(1-cX)^{-j} &= \sum_{n=0}^{\infty}\binom{-j}{n}(-cX)^n \\ &= \sum_{n=0}^{\infty} c^n\binom{j+n-1}{n}X^n \\ &= \sum_{n=0}^{\infty}\frac{c^n}{(j-1)!}((n+1)\cdot\ldots\cdot(n+j-1))X^n\end{aligned}$$

Obviously, $p_j(n) := (n+1)\cdot\ldots\cdot(n+j-1)$ is a polynomial in n. In fact, it turns out that

$$p_j(n) = \sum_{i=0}^{j-1} s_{j,i+1}n^i$$

where $s_{i,n}$ are the Stirling numbers of the first kind, which gives a more efficient formula to compute the p_j. We can conclude:

$$[X^n]\,\frac{a_{ij}}{(1-c_iX)^j} = \frac{a_{ij}}{(j-1)!}p_j(n)c_i^n$$

With this, we now have a complete procedure starting from a homogeneous or inhomogeneous recurrence and ending with a concrete and computable representation of the closed-form solution of the recurrence.

Applying this to our running example, whose generating function we decomposed into

$$\frac{\frac{16}{27}}{1+X} + \frac{\frac{2}{3}}{(1-2X)^3} - \frac{\frac{4}{9}}{(1-2X)^2} - \frac{\frac{22}{27}}{1-2X},$$

we obviously obtain the contribution $\frac{16}{27}\cdot(-1)^n$ for the first summand and $\frac{-22}{27}\cdot 2^n$ for the last one. For the other two summands, we compute

$$p_2(n) = n+1 \qquad p_3(n) = n^2+3n+2$$

and thereby:

$$[X^n]\,\frac{\frac{2}{3}}{(1-2X)^3} = \frac{\frac{2}{3}}{2!}(n^2+3n+2)2^n = \left(\frac{1}{3}n^2+n+\frac{2}{3}\right)2^n$$

$$[X^n]\,\frac{-\frac{4}{9}}{(1-2X)^2} = \frac{-\frac{4}{9}}{1!}(n+1)2^n = \left(-\frac{4}{9}n-\frac{4}{9}\right)2^n$$

Adding all the contributions together, we obtain the closed-form solution:

$$f(n) = \left(\frac{1}{3}n^2+\frac{5}{9}n-\frac{16}{27}\right)2^n + \frac{16}{27}\cdot(-1)^n$$

2.7 Asymptotics

Since the closed form of the coefficients of a rational FPS $p(X)/q(X)$ involves the complex roots of q, it can be somewhat unwieldy to work with: We have to fully factor the polynomial and do computations with (potentially complicated) algebraic numbers, which can lead to some problems, as we will see later. Fortunately, if we do not care about the precise closed form but only the asymptotics of the coefficients, there is an easier way.

Since any k-th order complex root z of the denominator $q(X)$ contributes a summand $r(n)z^{-n}$ with $\deg(r) \le k-1$ to the solution, it is clear that the asymptotically dominant summands are those where $|z|$ is minimal. In fact, it is easy to see that whenever we can find a radius $R > 0$ such that there are no roots z with $|z| < R$ and all roots with $|z| = R$ have order $\le k+1$, the solution is $O(n^kR^{-n})$. This fact has been proven in Isabelle.

Note that the numerator polynomial $p(X)$ does not appear in the asymptotics at all. In particular, this implies that the 'Big-O' bound for the solution of a linear recurrence holds irrespective of the precise initial values. However, the numerator polynomials (and thereby the initial values of a recurrence) *can* influence the asymptotics: If $p(X)$ and $q(X)$ have a non-trivial common divisor (i. e. they share at least one root), we can cancel that divisor from the fraction, which also reduces the number of roots in the denominator and may thereby lead to a smaller asymptotic upper bound.

When this is not possible – i. e. when the numerator and denominator are coprime, which can always be ensured for concrete polynomials in the numerator and denominator – then we have not only $\deg(r) \le k$, but $\deg(r) = k$ (see e. g. Theorem IV.9 by Flajolet and Sedgewick [5]), which means that the above 'Big-O' estimate is tight. If there is a single dominant root, one therefore even gets a 'Big-Θ' bound. One could show this in Isabelle, but this was not done yet, since I considered the 'Big-O' bounds to be the most relevant ones for applications, e. g. in the analysis of algorithms.

For our running example, we can automatically prove that the solution is $O(n^22^n)$: The square-free factorisation of the denominator $8X^4-4X^3-6X^2+5X-1$ already yields the full factorisation $(1+X)(2X-1)^3$. We then consider the circle $|X| = \frac{1}{2}$ and note that none of the factors have a root inside that circle and no factor with exponent $> 2+1$ has a root on the circle (since no such factor exists). This certifies the upper bound $O(n^22^n)$.

3 Formalisation in Isabelle

3.1 Formal Power Series

I build on the existing formalisation of FPSs by Chaieb [3], which I extended, among other things, with a more general notion of division: Chaieb only defined division in the case where the divisor is a unit, i. e. has a non-zero constant coefficient. This is problematic because division can also be well-defined when the divisor is not a unit, e. g. X/X or $(X^3+X)/(X^2+X)$. I therefore defined the concept of a *subdegree* in Isabelle, i. e. the index of the first non-zero coefficient. The division of two FPSs is well-defined iff the subdegree of the divisor does not exceed that of the dividend, and the new division operator I defined works in all of these cases.

Before we proceed, the concept of *normalisation* in a ring in Isabelle must be explained. In mathematics, one usually defines the greatest common divisor of the integers 4 and 6 to be 2, even though −2 would also be a perfectly adequate choice. Similarly, for the polynomials $2X$ and $3X$ in the ring $\mathbb{R}[X]$, it makes sense to define the GCD to be just X, even though any cX for $c \in \mathbb{R} \setminus \{0\}$ would also be possible. The underlying problem is that concepts like GCD and LCM do not really operate on *elements* of a ring, but on *association classes*. However, it is, of course, usually more convenient to work with ring elements, which is why one designates a single element of an association class to be the canonical representative of that class.

In Isabelle, we capture this in the class *normalization_semidom*, which assumes the existence of a *normalize* operation that, given some element $x \in R$, returns the canonical representative of the association class of x. We call an element *normalized* if it is the representative of its association class.

For us, this is useful in handling fractions: I extended Chaieb's *fract* type, which implements the fraction field of a given integral domain R, by adding the concept of a *normalised fraction*. Rational numbers, for instance, can be brought into a unique normal form by requiring the numerator and denominator to be coprime and the denominator to be positive. The same thing can be done for any integral domain R with a GCD and a concept of normalisation: One can bring any fraction a/b into normal form by dividing a and b by their GCD and then normalising the resulting denominator and adjusting the numerator accordingly. This yields a fraction a'/b' such that a' and b' are coprime and b' is normalised, and this representation can easily be shown to be unique.

I then used this to define the type *α ratfps* as the subset of *α poly fract* on which the denominator of the normalised fraction has a non-zero constant coefficient. Using Isabelle's `code_abstype` feature and the *transfer* [8] package, operations on *ratfps* can be implemented in terms of pairs of polynomials with the above-mentioned invariant and thus make the *ratfps* type fully executable. This allows us to automatically translate our abstract algorithms on FPSs directly to executable code. As a bonus, it also makes the *ratfps* and *fps* types available to Isabelle's counterexample generator QuickCheck [2], which aids us by automatically providing counterexamples when we write down an incorrect theorem statement.

3.2 Partial Fraction Decomposition

I used the following very general view of Partial Fraction Decomposition: Let R be a Euclidean domain and S an arbitrary ring with a homomorphism $\varphi : R \to S$. Let $n_1, \ldots, n_k \in \mathbb{N}_{>0}$ and $p, q_1, \ldots, q_k \in R$ such that the q_i are pairwise coprime and all the $\varphi(q_i)$ are units in S. Then, by following the process outlined in Section 2.5, we obtain $r, s_{ij} \in R$ such that

$$\frac{\varphi(p)}{\varphi(q_1)^{n_1} \ldots \varphi(q_k)^{n_k}} = \varphi(r) + \sum_{i=1}^{k} \sum_{j=1}^{n_i} \frac{\varphi(s_{ij})}{\varphi(q_i)^j}$$

and all the s_{ij} have a Euclidean norm less than the Euclidean norm of q_i.

In our case, the Euclidean domain R is the ring of polynomials $K[X]$, the ring S is the ring of rational FPSs, and φ is the canonical homomorphism *ratfps_of_poly*. Also, each of the q_i is of the form $1 - cX$, such that $\varphi(1 - cX)$ is always a unit in S.

However, thanks to this very general derivation, one could also use it for $R = \mathbb{Z}$ and $S = \mathbb{R}$ to bring $\frac{1}{90} = \frac{1}{2 \cdot 3^2 \cdot 5}$ into the form $-1 + \frac{1}{2} + \frac{1}{3^2} + \frac{2}{5}$.

3.3 Complex Root Counting

One step that was still left open in the discussion of coefficient asymptotics before was how to actually check the conditions that the polynomial has no roots inside a given circle around the origin, and all the roots on the circle itself have at most order k. To do this, we need two components:

- Wenda Li's executable root counting algorithm[9, 10] that can count the number of complex roots within certain subsets of the complex plane, e. g. circles, rectangles, and half-planes. Many variants are offered, but I use only root counting inside an open disc ($|z| < R$) and on a circle ($|z| = R$) without taking multiplicities into account.
- Square-free factorisation as formalised by Thiemann *et al.* [15, 16]. This is much less involved than a full factorisation and allows us to break up a polynomial into factors of the form $p_i(X)^{k_i}$ such that the p_i are square-free and pair-wise coprime. This means that each root of the original polynomial is present in exactly one of the p_i, and the corresponding k_i is the order of the root.

We therefore only have to run the square-free factorisation algorithm and then check that no p_i has a root with $|z| < R$, and that additionally no p_i with $k_i > k$ has any roots with $|z| = R$. None of this involves factoring the entire polynomial, and all computations can be done in the relatively pleasant field $\mathbb{Q}[i]$ – unless, of course, the coefficients or the radius R itself are already irrational themselves.

3.4 Code generation

The steps given above can be broken down into a number of Isabelle/HOL functions, which provides some modularity:

3.4.1 lhr_fps

This function takes a list of coefficients $c_0, \ldots, c_m$ and initial values $f_0, \ldots, f_{m+l-1}$ and returns the rational FPS of the corresponding recurrence as described in Section 2.2. Note that this implicitly uses the convention of Section 2.2 where the

recurrence is only required to hold for $n \geq l$; i. e. extra initial values can be used to encode exceptions to the recurrence.

3.4.2 lir_fps

This is analogous to *lhr_fps*, but additionally takes a representation of the inhomogeneous part. This representation has the form of a list of tuples (a, b, k), encoding a sum over the terms $an^k b^n$. This is done according to the process outlined in Section 2.3.

3.4.3 solve_factored_ratfps

This function takes a rational FPS $F(X) = p(X)/q(X)$ where $q \neq 0$ and $q(X)$ has already been factored into terms of the form $1 - cX$ with $c \neq 0$ and returns a representation of a closed-form expression for its coefficients. This representation consists of a complex polynomial $a_{l-1}X^{l-1} + \ldots + a_0$ and a list of pairs $(p_i(X), b_i)$ where $p_i(X)$ is a complex polynomial and b_i is a complex number with the property that

$$[X^n]F(X) = a_n + \sum p_i(n) b_i^n$$

where $a_n = 0$ for all $n \geq l$. This is done by following the process described in Section 2.5 and immediately applying the process from Section 2.6 to the results.

3.4.4 solve_ratfps

The only missing link is now to factor the denominator polynomial of the rational FPS obtained from *lhr_fps* or *lir_fps* into terms of the form $1 - cX$. This is done by simply reflecting the polynomial and calling an arbitrary factoring algorithm.

I use the formalised algorithm by Thiemann *et al.* [13, 14], which takes a complex polynomial $p(X)$ and returns some complex number d and a list of pairs (e_i, k_i) such that $p(X) = d \cdot \prod (X - e_i)^{k_i+1}$ and the e_i are distinct. Because we reflected the polynomial, we therefore get $p(X) = d \cdot \prod (1 - e_i X)^{k_i+1}$, which is exactly what we need.

Through the use of the Algebraic Number library by Thiemann *et al.*, this method works out-of-the-box for any rational FPS whose numerator and denominator have rational coefficients, and therefore the overall method works for any linear recurrence with rational coefficients.

3.5 Isabelle Theorems

For the purpose of illustration, I shall print some of the main theorems from the Isabelle formalisation. Some of the notation was adjusted very slightly to make the statements more readable without knowledge of Isabelle, but the statements as printed are still very close to the original ones in Isabelle. In particular, the explicit homomorphisms between $\mathbb{N}$ and $\mathbb{Z}$ or between $K[X]$ and $K[[X]]$ that have to be made explicit in Isabelle are still present.

The following is the statement of the Isabelle theorem about converting a linear homogeneous recurrence to a rational FPS:

lemma
  fixes f :: nat ⇒ (α :: field) **and** cs :: α list
  defines N := length $cs - 1$
  assumes $cs \neq []$
  assumes $\forall n \geq m.\ (\sum_{k \leq N}\ cs_k \cdot f\ (n + k)) = 0$
  assumes last $cs \neq 0$
  shows Abs_fps f = fps_of_poly (lhr_fps_numerator m cs f) /
                fps_of_poly (lr_fps_denominator cs)

Here, the α :: *field* stands for an arbitrary type of the type class *field*, which is a field in the algebraic sense. This type class has concrete instances like *rat*, *real*, or *complex*.

The function *Abs_fps* converts between a sequence (a function $\mathbb{N} \to \alpha$) and an FPS (α fps). The function *fps_of_poly* is the canonical homomorphism mapping a polynomial to an FPS. Note that all the functions on the right-hand side of the equation in the theorem statement have code equations in Isabelle and are therefore executable.

Furthermore, the theorem on the closed form of a rational FPS is:

lemma
  assumes is_alt_factorization_of *fctrs* q **and** $q \neq 0$
  shows Abs_fps (interp_ratfps_solution
                (solve_factored_ratfps' p *fctrs*)) =
            fps_of_poly p / fps_of_poly q

with

definition interpret_ratfps_solution (p, cs) n =
  coeff p n + ($\sum_{(q,c) \leftarrow cs}$ poly q (of_nat n) · c ^ n)

Here, *poly* evaluates a polynomial and *of_nat* is the canonical homomorphism from $\mathbb{N}$ into any other semiring. Moreover, *is_alt_factorization_of* states that *fctrs* is a factorization of q into the form $d \prod (1 - e_i X)^{n_i}$ as we have seen before, with *fctrs* being a pair consisting of the d and a list of pairs of the e_i and n_i. The solution that is being computed is a pair of a polynomial p, whose coefficients form the 'correction terms' for recurrences with additional initial values, and a list consisting of pairs of a polynomial $r_i(X)$ and a number c_i such that $r_i(n)\, c_i^n$ is a summand in the solution. The function *interpret_ratfps_solution* 'evaluates' such a solution for a given n.

Since these theorems are somewhat notationally technical, I will not print any more of them here; the correctness theorems for converting inhomogeneous recurrences to FPSs and solving recurrences look very similar to the ones printed here.

However, the following more abstract theorem about the shape of solutions of rational FPSs (and thereby linear recurrences) is probably reasonably readable:

theorem
  fixes $p\ q$ :: complex poly
  assumes coeff $q\ 0 \neq 0$
  defines $q' :=$ reflect_poly q
  obtains $r\ rs$
  where $\forall n$. fps_nth (fps_of_poly p / fps_of_poly q) $n =$
    coeff $r\ n + (\sum_{c \mid \text{poly } q'\ c = 0}$ poly $(rs\ c)$ (of_nat $n) \cdot c\ \hat{}\ n)$
  and $\forall z$. poly $q'\ z = 0 \Longrightarrow$ degree $(rs\ z) \leq$ order $z\ q' - 1$

It states that the solution of a rational FPS is a sum where each root c of the denominator contributes a summand of the form $p(n)c^n$, plus a correction term that vanishes for almost all n. Moreover, the degree of each polynomial is at most the order of the corresponding root minus 1. The variable *rs* in the above Isabelle theorem is the function that associates the polynomial to each root, and the r is a polynomial whose coefficients constitute the 'correction term'.
This also directly implies the key theorem on coefficient asymptotics:

theorem
  fixes $p\ q$ :: complex poly
  assumes square_free_factorization q (b, cs)
  assumes $q \neq 0$ **and** $R > 0$
  assumes $\forall (c, l) \in cs.\ \forall x.$ norm $x < 1/R \longrightarrow$ poly $c\ x \neq 0$
  assumes $\forall (c, l) \in cs.\ \forall x.\ l > k \wedge$ norm $x = 1/R \longrightarrow$ poly $c\ x \neq 0$
  shows fps_nth (fps_of_poly p / fps_of_poly q) $\in$
    $O(\lambda n.$ of_nat $n\ \hat{}\ k \cdot$ of_real $R\ \hat{}\ n)$

The *square_free_factorization* predicate asserts that the given polynomial $q(X)$ has a square-free factorization into some constant b and some list of factors $c_i(X)^{l_i}$ pairs, represented as a list of pairs of the form (c, l).

The Landau symbol $O(\ldots)$ is defined in the usual fashion where $f \in O(g)$ iff there exists some $C \in \mathbb{R}$ such that, for all sufficiently large x, $|f(x)| \leq C|g(x)|$.

3.6 Pretty Printing

One disadvantage of using the Algebraic Number library is that printing irrational algebraic numbers is not straightforward. Without additional setup, evaluating e. g. `sqrt 2` leads to a few lines of fairly illegible output which corresponds to the internal representation of irrational algebraic numbers in the Isabelle library as an integer polynomial with an additional upper and lower bound that uniquely identifies one root of the polynomial, which is the number that is being described.

Thankfully, Thiemann *et al.* also provide some pretty-printing functionality which converts a real-algebraic number into a human-readable string. For algebraic numbers of degree at most 2, this is exactly what one would expect: a combination of rational numbers and square roots. However, for numbers of higher degree, the output is unfortunately still of the form 'k-th root of polynomial p'; e. g. $\sqrt[3]{2}$ is rendered as `root #1 of -2 + x^3, in (1,2)`.

I add to this some more pretty-printing code in order to display rational FPSs and the solutions computed for linear recurrences in a natural, human-readable form.

It should be noted that all of the pretty-printing that Thiemann *et al.* and I do is unverified; however, in both cases, the step from the representation in Isabelle to the human-readable string is very small.

4 Evaluation

We can now evaluate the solver on some examples using Isabelle's `value` and `export_code` commands.

4.1 Fibonacci Numbers

For the Fibonacci case, we invoke `solve_lhr [-1,-1,1] [0,1]`, which – after pretty-printing the complex numbers – returns:

```
Some (0, [(sqrt(1/5), (1/2+sqrt(5/4))),
  ( -sqrt(1/5), (1/2-sqrt(5/4)))])
```

Alternatively, when using the pretty-printer for solutions of recurrences, we get:

```
(sqrt(1/5)) * (1/2+sqrt(5/4)) ^ x +
( -sqrt(1/5)) * (1/2-sqrt(5/4)) ^ x
```

We can also compute the rational FPS of the recurrence using *lhr_fps* directly:

```
-1x / (-1 + x + x^2)
```

4.2 A Higher-Degree Recurrence

Another simple example of higher degree is the sequence 0, 1, 2, 3, 0, 1, 2, 3, ... with the recurrence $f(n+4) - f(n) = 0$ and the initial values 0, 1, 2, 3. The output is

```
(-1/2) * (-1) ^ x + (-1/2+1/2i) * (1i) ^ x +
    (-1/2+-1/2i) * (-1i) ^ x + (3/2)
```

or, in a more readable form:

$$f(n) = -\frac{(-1)^n}{2} + \frac{i-1}{2} i^n - \frac{i+1}{2}(-i)^n + \frac{3}{2}$$

4.3 Running Example

Next, let us again consider our running example as an example of an inhomogenous recurrence. This can be solved by evaluating

```
solve_lir [-2, -1, 1] [0, 0] [(1, 1, 2)]
```

which returns

```
(-16/27 + 5/9x + 1/3x^2) * 2 ^ x +
    (16/27) * (-1) ^ x
```

4.4 A pathological Example

Lastly, we look the seemingly innocuous example $5f(n+4) + 4f(n+3) + 3f(n+2) + 2f(n+1) + f(n) = 0$ with arbitrary initial values. The recurrence solver does not terminate for this example and eventually runs out of memory. Factoring

the characteristic polynomial itself is no problem, but due to the way algebraic complex numbers are implemented by Thiemann *et al.*, the computations with the roots of this polynomial lead to a significant blow-up in the degrees of the integer polynomials used to represent the real and imaginary parts of complex algebraic numbers, which then need to be factored again.

However, if we only want to know the asymptotics of the solution, things are much more pleasant: The dominant roots of the characteristic polynomial have an absolute value of 1.445046…, whose reciprocal is ≈ 0.6920196, so we can easily show e. g. $f(n) \in O(0.69202^n)$ automatically.

4.5 Performance Comparison

Evaluating any of these examples directly in Isabelle with the `value` command takes about 1 minute. However, almost all of this is taken up by code generation and compilation. I therefore exported the code for the functions *solve_lhr*, *solve_lir*, and the pretty printing to Haskell using Isabelle's `code_export` command, wrote a minimal wrapper for input/output and to convert Haskell numbers into the exported datatype for complex numbers, and compiled everything with the Glasgow Haskell Compiler.

Comparing the performance of the Isabelle solver to that of systems like Mathematica and Maple does not make much sense because the computation time is completely dominated by the polynomial factorisation, so any attempt to compare the efficiency of the linear recurrence solvers essentially boils down to a mere comparison of the efficiency of the polynomial factorisation algorithms. Nevertheless, Table 1 gives a quick impression of how the verified solver and Mathematica's solver perform on the above examples and some more randomly-generated examples. The performance for certifying a reasonably tight 'Big-O' bound in Isabelle is also given.

The measurements are to be taken with a grain of salt as they were conducted on a shared machine in the computer pool at the Technical University of Munich since this was the only machine on which a Mathematica licence was available. However, repeating the measurements at different times showed that they were fairly stable; they should be good enough to at least give a qualitative comparison of the performance behaviour of the different approaches.

The table shows that Mathematica performs considerably better than the Isabelle solver on the pathological example of degree 5 mentioned before. However, for similarly pathological polynomials of even a slightly higher degree (e. g. 9), Mathematica also fails to terminate within a reasonable amount of time (around 5 minutes). The verified asymptotics certification method works much better than either solver, but its performance also degenerates very quickly as the degrees grow, and its performance also depends on how 'complicated' the fraction b in the $O(n^k b^n)$ is. The reason for this is that the current implementation of Wenda Li's root-counting method uses rational arithmetic and the numerators and denominators can grow very large, depending on the numbers in the input and the degree of the polynomial. This is a known problem with remainder-series-based approaches like Li's.

The performance comparison suggests that solving recurrences externally and somehow certifying results in Isabelle is perhaps not very useful, since the verified solver copes very well with 'simple' recurrences and when one moves to complicated recurrences, the performance of the Mathematica solver also degrades very quickly. Furthermore, since the performance degradation seems to come mainly from polynomials with 'complicated' roots and the irrational arithmetic involved in the resulting computations, it seems likely that any kind of certification in a case like this would also have to involve the same irrational arithmetic since these roots are part of the closed-form solution. It therefore seems doubtful if certification of a closed-form solution makes sense.

However, both the polynomial factorisation by Thiemann *et al.* and the root-counting procedure by Li are under active development and the code developed here will directly profit from any improvements that they make.

There is no analogue to the verified asymptotics certifier in Mathematica or Maple, although it would be fairly easy to write one and it should be very efficient, seeing as systems like Mathematica tend to have very sophisticated algorithms for root isolation. In fact, it may be useful to employ a system like Mathematica to determine the approximate or exact asymptotics of an equation using its root isolation algorithms and then certify it in Isabelle using the verified certifier. However, this has not been implemented yet.

5 Related Work

To my knowledge, this is the first work on the theory and a solver for linear recurrences in a proof assistant. The theory of these recurrences has been known for a long time and is usually taught in undergraduate courses of mathematics. The classic method of applying partial fraction decomposition to the rational FPS that was used in this work can be found in many textbooks [7]. An alternative route is to use results from Analytic Combinatorics [5] for meromorphic functions (of which rational functions are a special case) to obtain the coefficients of the FPS in terms of the complex residues at its poles.

Many different executable solvers for linear recurrences are available, e. g. PURRS [1] or those that come with computer algebra systems like Mathematica and Maple. For the homogeneous part, they also use the approach of factoring the characteristic polynomial (possibly with some preprocessing to decrease the degree of the recurrence). Unlike the Isabelle formalisation, these systems typically support

Table 1. Benchmark results of the solver (the verified one and Mathematica's) and the verified certifier. The first 4 examples are the ones from above; the remaining ones have randomly-generated coefficients ranging from −10 to 10. A time of 0 indicates that the time was below 1 ms; a time of ∞ indicates a timeout after more than 5 minutes. The given asymptotics show an approximation of the actual (irrational) basis.

Example	Degree	Asymptotics	Time (ms)		
			Solver (ver.)	Solver (Math.)	Certifier (ver.)
Fibonacci	2	$O(1.619^n)$	12	20	0
0, 1, 2, 3…	4	$O(1)$	0	12	0
$f(n) + 2f(n-1) = n \cdot 2^n$	3	$O(n \cdot 2^n)$	0	12	0
Running example	5	$O(n^2 \cdot 2^n)$	7	368	0
Pathological	5	$O(0.69202^n)$	∞	4600	0
Random	3	$O(2.331^n)$	1200	970	0
Random	9	$O(1.1552^n)$	∞	∞	260
Random	11	$O(8.876^n)$	∞	∞	5100
Random	14	$O(1.1985^n)$	∞	∞	238000

more complicated inhomogeneous parts. This typically requires finding a closed form for some symbolic sum or 'guessing' a solution. Furthermore, they also allow solving systems of recurrences. Both of this is beyond the scope of this work.

6 Conclusion

I formalised the basic theory of and an executable solver for linear recurrences with constant coefficients, both homogeneous ones and inhomogeneous ones where the inhomogeneous term is of polynomial-exponential form. An executable solver for these recurrences and a more efficient certifier for the 'Big-O' asymptotics of their solutions is also provided. This development makes use of many different components:

- Executable polynomial factorisation and algebraic numbers (Thiemann *et al.* [13, 14])
- Square-free polynomial factorisation (Thiemann *et al.* [15, 16])
- Formal power series (Chaieb [3])
- Stirling Numbers (Isabelle library)
- Counting complex roots (Li [9, 10])
- Eulerian polynomials (Eberl)
- Partial Fraction Decomposition (Eberl)
- Executable rational FPSs (Eberl)

The last three of these were motivated by this very application of solving linear recurrences and the modularity allows us to e. g. easily improve the Partial Fraction Decomposition algorithm in the future, or to directly benefit from any performance improvements made by Thiemann *et al.* without any changes to the recurrence solver.

Acknowledgments

This work was supported by Deutsche Forschungsgemeinschaft under Grants DFG RTG 1480 (PUMA) and DFG Koselleck NI 491/16-1.

Lastly, I would like to thank René Thiemann and Akihisa Yamada for their help with their Algebraic Numbers library and Wenda Li for implementing the circular case in his root-counting algorithm upon my suggestion. I also thank the anonymous reviewers for their advice.

References

[1] R. Bagnara, A. Zaccagnini, and T. Zolo. 2003. *The Automatic Solution of Recurrence Relations. I. Linear Recurrences of Finite Order with Constant Coefficients.* Quaderno 334. Dipartimento di Matematica, Università di Parma, Italy. Available at http://www.cs.unipr.it/Publications/.

[2] Lukas Bulwahn. 2012. *The New Quickcheck for Isabelle.* Springer Berlin Heidelberg, Berlin, Heidelberg, 92–108. https://doi.org/10.1007/978-3-642-35308-6_10

[3] Amine Chaieb. 2011. Formal Power Series. *Journal of Automated Reasoning* 47, 3 (01 Oct 2011), 291–318. https://doi.org/10.1007/s10817-010-9195-9

[4] Manuel Eberl. 2017. Linear Recurrences. *Archive of Formal Proofs* (Oct. 2017). http://isa-afp.org/entries/Linear_Recurrences.html, Formal proof development.

[5] Philippe Flajolet and Robert Sedgewick. 2009. *Analytic Combinatorics* (1 ed.). Cambridge University Press, New York, NY, USA.

[6] Philippe Flajolet, Paul Zimmermann, and Bruno Salvy. 1989. *Lambda-Upsilon-Omega: The 1989 cookbook.* Research Report RR-1073. INRIA. https://hal.inria.fr/inria-00075486

[7] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. 1994. *Concrete Mathematics: A Foundation for Computer Science* (2nd ed.). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[8] Ondřej Kunčar. 2016. *Types, Abstraction and Parametric Polymorphism in Higher-Order Logic.* Ph.D. Dissertation. https://www21.in.tum.de/~kuncar/documents/kuncar-phdthesis.pdf

[9] Wenda Li. 2017. Count the Number of Complex Roots. *Archive of Formal Proofs* (Oct. 2017). http://isa-afp.org/entries/Count_Complex_Roots.html, Formal proof development.

[10] Wenda Li and Lawrence C. Paulson. 2018. Evaluating Winding Numbers and Counting Complex Roots through Cauchy Indices in Isabelle/HOL. *CoRR* abs/1804.03922 (2018). http://arxiv.org/abs/1804.03922

[11] Van Chan Ngo, Quentin Carbonneaux, and Jan Hoffmann. 2018. Bounded Expectations: Resource Analysis for Probabilistic Programs.

In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2018)*. ACM, New York, NY, USA, 496–512. https://doi.org/10.1145/3192366.3192394

[12] Joël Ouaknine and James Worrell. 2015. On Linear Recurrence Sequences and Loop Termination. *ACM SIGLOG News* 2, 2 (April 2015), 4–13. https://doi.org/10.1145/2766189.2766191

[13] René Thiemann and Akihisa Yamada. 2015. Algebraic Numbers in Isabelle/HOL. *Archive of Formal Proofs* (Dec. 2015). http://isa-afp.org/entries/Algebraic_Numbers.html Formal proof development.

[14] René Thiemann and Akihisa Yamada. 2016. *Algebraic numbers in Isabelle/HOL*. Springer International Publishing, Cham, 391–408. https://doi.org/10.1007/978-3-319-43144-4_24

[15] René Thiemann and Akihisa Yamada. 2016. Formalizing Jordan Normal Forms in Isabelle/HOL. In *Proceedings of the 5th ACM SIGPLAN Conference on Certified Programs and Proofs (CPP 2016)*. ACM, New York, NY, USA, 88–99. https://doi.org/10.1145/2854065.2854073

[16] René Thiemann and Akihisa Yamada. 2016. Polynomial Factorization. *Archive of Formal Proofs* (Jan. 2016). http://isa-afp.org/entries/Polynomial_Factorization.html, Formal proof development.

[17] Bohua Zhan and Maximilian P. L. Haslbeck. 2018. Verifying Asymptotic Time Complexity of Imperative Programs in Isabelle. *CoRR* abs/1802.01336 (2018). http://arxiv.org/abs/1802.01336