Formalizing Jordan Normal Forms in Isabelle/HOL

René Thiemann   Akihisa Yamada
University of Innsbruck, Austria
{rene.thiemann, akihisa.yamada}@uibk.ac.at

Abstract

In automated complexity analysis of term rewriting, estimating the growth rate of the values in the k-th power of a matrix A – for fixed A and increasing k – is of fundamental interest. This growth rate can be exactly characterized via A's Jordan normal form (JNF). We formalize this result in our library IsaFoR and our certifier CeTA, and thereby improve the support for certifying polynomial bounds derived by (untrusted) complexity analysis tools.

To this end, we develop a new library for matrices that allows us to conveniently work with block matrices. Besides the mentioned complexity result, we formalize Gram-Schmidt's orthogonalization algorithm and the Schur decomposition in order to prove existence of JNFs. We also provide a uniqueness result for JNFs which allows us to compute Jordan blocks for individual eigenvalues. In order to determine eigenvalues automatically, we moreover formalize Yun's square-free factorization algorithm.

Categories and Subject Descriptors F.4.1 [*Mathematical Logic*]: Mechanical theorem proving; G.1.3 [*Numerical Linear Algebra*]

Keywords matrix theory, Jordan normal form, Isabelle/HOL, complexity

1. Introduction

CeTA [25] is a certifier for complexity proofs of term rewrite systems; it takes an untrusted, automatically generated proof from complexity analyzers such as AProVE, CaT, or TCT [1, 9, 29], and tries to validate it. There are three possible outcomes: (1) the proof could be validated, (2) the proof was rejected because it indeed was faulty, and (3) the proof was rejected because at some point in the analysis CeTA applied too coarse estimations or imposed too severe preconditions, so that the desired complexity bound could not be validated.

This paper aims at reducing the number of rejected proofs of the third kind, by improving the support for *matrix interpretations* [8], an important technique for complexity analysis. For instance, in *the termination competition 2015* [10], roughly 40 % of the machine readable complexity proofs contain matrix interpretations.

Given a matrix interpretation, one can provide a complexity bound by estimating the growth rate of the matrix A^k, where A is determined by the interpretation. If the values in A^k are bounded by $\mathcal{O}(k^N)$ – we write as $A^k \in \mathcal{O}(k^N)$ in the sequel – then $\mathcal{O}(k^{N+1})$ is a valid bound for the runtime of the rewrite system.

The connection between the growth rate of A^k and the complexity of the rewrite system is already covered in previous papers [2, 21]. On the other hand, for estimating the growth rate of A^k, the corresponding version of CeTA is limited: it is based on the initiating result by Moser et al. [16], where only upper-triangular matrices are allowed, and the degree of the polynomial depends only on the dimension of the matrix.

Using the *spectral radius theory*, Neurauter et al. [17] improved the result for arbitrary matrices, and the degree of polynomial bounds to be computed from the maximum *multiplicity* of *eigenvalues* of (complex) norm 1, if no eigenvalue has a larger norm.

Nevertheless, bounds derived by these results are not tight. For instance, consider the matrix

$$A = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The old version of CeTA accepts only

$$A^k \in \mathcal{O}(k^3)$$

or higher as a complexity bound. This bound would not improve by using [17], since the only eigenvalue of A is 1 and its multiplicity is four, yielding the same bound.

In this paper, we derive tight bounds via *Jordan normal forms (JNFs)* [13]. For the above matrix A, its JNF is computed as $A = PJP^{-1}$, where P is some invertible matrix and

$$J = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

A closed formula is known for k-th powers of JNFs; in this case,

$$J^k = \begin{bmatrix} 1^k & \binom{k}{1}1^{k-1} & 0 & 0 \\ 0 & 1^k & 0 & 0 \\ 0 & 0 & 1^k & 0 \\ 0 & 0 & 0 & 1^k \end{bmatrix} = \begin{bmatrix} 1 & k & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

whose elements are clearly bounded by k. Since $A^k = PJ^kP^{-1}$, we can derive the following tight bound:

$$A^k \in \mathcal{O}(k)$$

There are other approaches for tight bounds. Namely, Waldmann [27] employs weighted automata, which is later shown to be tight [15]; and Middeldorp et al. [15] also derive tight bounds by extending [17] using *minimal polynomials*.

Nevertheless, our deviation to JNFs is easily justified: We initially tried to follow the approach of [15, 17], which works via *Cayley-Hamilton theorem*, *linear recurrence equations*, and results for growth rates of linear recurrence equations. When looking for

CPP'16, January 18–19, 2016, St. Petersburg, FL, USA
ACM. 978-1-4503-4127-1/16/01...$15.00
http://dx.doi.org/10.1145/2854065.2854073

paper proofs for growth rates of recurrence equations, however, we stumbled upon proofs which converts linear recurrence equations into matrices, and then use results which are based on JNFs. Hence, the whole connection to linear recurrence equations in [17] can be seen as a detour.

Thus we choose to formalize the theory around JNFs and its connection to polynomial complexity bounds in the Isabelle proof assistant [18]. Afterwards, we integrate them in IsaFoR and CeTA, where the former contains the soundness proof of the latter. As a result, the new version of CeTA now accepts the tight $A^k \in \mathcal{O}(k)$ (and higher) for the above example, and a large variety of theorems about polynomials, matrices, JNFs, etc., constituting more than 20,000 lines of Isabelle code, become available to every Isabelle user.

The whole formalization (IsaFoR version 2.23) and details on the experiments are available at

```
http://cl-informatik.uibk.ac.at/software/ceta/
                experiments/jnf/
```

where our contribution consists of all theories in the directory `thys/Matrix`. A preliminary version of the formalization is also available in the archive of formal proofs (AFP) [26].

This paper is structured in the following way (we indicate the most important theory file for each section in the section heading).

- We motivate and describe a new library for matrices (Sect. 3).
- We formalize JNFs and provide explicit complexity bounds for matrices in this form. These bounds allow us to formalize a main result, that for complex matrices (which always possess JNFs), we have polynomial growth of A^k if and only if the spectral radius is at most one (Sect. 4).
- To formalize that every complex matrix has a JNF, we provide and prove the correctness of an algorithm to convert a matrix into JNF (taking the linearly factored characteristic polynomial as input). To this end, we formalize the *Gram-Schmidt algorithm* for creating orthogonal bases (Sect. 5), the *Schur decomposition* which converts a matrix into an upper-triangular form (Sect. 6), and an algorithm which computes a JNF of an upper-triangular matrix by means of elementary row and column transformations (Sect. 7).[1]
- In order to factor the characteristic polynomial, we formalize Yun's algorithm for square-free factorization [28] and combine it with explicit formulas for low-degree polynomials[2] and with heuristics that guess roots (Sect. 8).
- We further illustrate how all the methods have been combined in a single algorithm for validating growth rates of matrices, and discuss initial problems when executing it. When the exact growth rate of A^k is polynomial, say $\Theta(k^N)$, the algorithm accepts the sharp $\mathcal{O}(k^N)$ as an upper bound, if A has no complex eigenvalue, or if N differs from the dimension of the matrix by at most three (Sect. 9).
- We finally evaluate our experiments and discuss future work (Sect. 10).

2. Preliminaries

We assume basic knowledge of Isabelle and linear algebras.

2.1 IsaFoR and CeTA

IsaFoR is the *Isa*belle *Fo*rmalization of *R*ewriting. It contains various theorems about rewriting, including several techniques for proving confluence, complexity, and termination of rewrite systems. These techniques are combined to yield a certification algorithm for these properties within Isabelle: CeTA. It inputs a rewrite system, a property, and an untrusted proof, and it either accepts the proof, or rejects the proof with some detailed error message. The soundness of CeTA is formally proven within Isabelle; hence, if CeTA accepts a proof, then the rewrite system satisfies the property.

Although completely specified in Isabelle, usually CeTA is invoked outside Isabelle, as a compiled Haskell code generated by Isabelle's code generator. This approach has some consequences: CeTA is quite fast and it can be used without having Isabelle installed. Regarding the formalization, the most important consequence is, that no change can be made to CeTA after it has been generated. For instance, it is impossible to create a datatype on the fly depending on the input.

2.2 Linear Algebra

Whereas IsaFoR is mainly about term rewriting, this paper is mostly on linear algebra. We recapitulate some basic notions and notations.

In mathematical expressions (not in Isabelle code) we follow standard conventions, such as considering a one-dimensional vector as a scalar and a one-column matrix as a vector. For a complex matrix A, we denote its transpose by A^{T} and conjugate transpose (Hermitian transpose) by A^{H}. Hence, $\boldsymbol{v}^{\mathrm{H}}\boldsymbol{w}$ denotes the dot product of vectors $\overline{\boldsymbol{v}}$ and $\boldsymbol{w}$, where $\overline{a}$ denotes the complex conjugate of $a \in \mathbb{C}$ and extended naturally to complex vectors. Two complex vectors $\boldsymbol{v}$ and $\boldsymbol{w}$ are *orthogonal* if $\boldsymbol{v}^{\mathrm{H}}\boldsymbol{w} = \boldsymbol{0}$.

DEFINITION 1. *An* eigenvalue *of a matrix A is a scalar λ such that there exists a non-zero vector $\boldsymbol{v}$ called an* eigenvector *satisfying $A\boldsymbol{v} = \lambda\boldsymbol{v}$. The* characteristic polynomial *of a matrix A is $\det(xI - A)$, where I is the identity matrix and $\det$ the determinant.*

It is well known that the eigenvalues are precisely the roots of the characteristic polynomial.

DEFINITION 2 (Similarity). *Two $n \times n$ matrices A and B are* similar *if there exists an invertible matrix P such that $A = PBP^{-1}$.*

If $A = PBP^{-1}$, then clearly $A^k = PB^kP^{-1}$. Moreover, two similar matrices have the same characteristic polynomial and hence, the same eigenvalues.

DEFINITION 3 (Jordan Normal Form). *A square matrix A* is in JNF *if it is a block-diagonal matrix of the form*

$$A = \begin{bmatrix} J_1 & & \\ & \ddots & \\ & & J_p \end{bmatrix}$$

where each J_i, called a Jordan block, *is a square matrix of the following form:*

$$J_i = \begin{bmatrix} \lambda & 1 & & \\ & \lambda & \ddots & \\ & & \ddots & 1 \\ & & & \lambda \end{bmatrix}$$

for some λ. A square matrix A has a JNF *if there exists a matrix B in JNF which is similar to A. Blank parts in a matrix are assumed to be zero.*

From the definition it is obvious that each matrix in JNF can be represented by a list of pairs (n, λ) where every pair represents a

[1] We chose this approach over the proofs via *generalized eigenspaces*, as the former works incrementally: we started by supporting triangular matrices, and then later on added the support for arbitrary matrices.

[2] We only use the formula for degree-two polynomials, due to a limitation in Isabelle's implementation of irrational numbers.

Jordan block of size n with λ on its diagonal. Moreover, the λ's on the diagonal are exactly the eigenvalues of the matrix.

3. A New Matrix Library

3.1 Motivation

When starting our formalization, there was the design question of what matrix/vector representation the whole development should be based on.

On the one hand, there is the *HOL multivariate analysis (HMA)* representation by Harrison [11], where a vector $\boldsymbol{v} \in \mathbb{R}^n$ is represented as a function of type $\alpha \Rightarrow \mathbb{R}$ where α is a type with n elements, and similarly a matrix $A \in \mathbb{R}^{n \times m}$ is represented by a type $\alpha \Rightarrow \beta \Rightarrow \mathbb{R}$ where β is a type with m elements. A large library around this representation is available, both in the Isabelle distribution and in the AFP. Unfortunately, however, HMA turns out not to be a good choice for our development, for two reasons:

- First, due to the encoding of dimensions as types, at the moment of invoking the code generator for CeTA we would need to have created all types for every dimension that we require later on when checking generated complexity proofs. Although the current complexity tools use matrices of dimensions at most 10, it is clearly not an elegant approach to generate 10 variants of code and fail whenever a tool generates proofs with larger matrices.
- Second, since a JNF usually contains Jordan blocks of different sizes, we would not be able to express in HMA the set of the Jordan blocks. For instance, it is impossible to even specify the type of a function which takes a list of Jordan blocks and composes a JNF from it. If the input list is of type $(\alpha \Rightarrow \alpha \Rightarrow \mathbb{R})$ *list*, then all blocks must have the same dimension. Moreover the output type of the function clearly depends on the input term, which is not expressible in Isabelle/HOL as it does not feature dependent types.

On the other hand, there is another representation available in the AFP [20], which directly represents vectors as lists, and matrices as lists of lists. Here, there is no problem with code generation; however, it is a bit too low-level in our view, and there are hardly any available results except for basic matrix operations. Moreover, when performing proofs with this representation of matrices, it can easily happen that the simplification rules for lists interfere with the preferred reasoning for matrices.

3.2 New Matrix Representation (`Matrix`)

As a solution, we provide a new matrix representation which is as abstract as the HMA representation, but is flexible for dimensions. We represent a vector $(v_0, \dots, v_{n-1})$ as a pair (n, v) of the dimension n and the characteristic function v, i. e., $v\ i = v_i$. Similarly, n-row m-column matrix A is expressed as a triple (n, m, a) using the characteristic function a of A. We collect such represented vectors and matrices as types α *vec* and α *mat*, respectively, where α is the element type. More precisely, α *vec* is a subtype of $\mathbb{N} \times (\mathbb{N} \Rightarrow \alpha)$ where $(n, v) :: \alpha$ *vec* enforces the invariant that $v\ i$ is undefined[3] for $i \geq n$. Similarly, α *mat* is a subtype of $\mathbb{N} \times \mathbb{N} \times (\mathbb{N} \Rightarrow \mathbb{N} \Rightarrow \alpha)$. As a consequence, two vectors/matrices are equal iff they have the same dimension and the characteristic functions coincide inside the dimension.

The constructors for vectors and matrices are *vec* and *mat*, which take dimensions and characteristic functions and deliver vectors/matrices. Internally, we use the lifting and transfer package [12] to work with the subtypes and to define primitive operations such as the constructors or selectors for the dimensions, etc.

Based on these primitive operations, we define the basic operations, such as addition $\oplus_v / \oplus_m$, negation $\ominus_v / \ominus_m$, scalar multiplication $\odot_v / \odot_m$, dot product $\bullet$, and matrix multiplication $\otimes_m$.

Our new matrix library has several advantages.

- It is logically similar to HMA in the sense that vectors/matrices are represented by their characteristic functions. This allows to easily adapt existing proofs from HMA for our setting. Indeed, we could adapt large parts of the HMA formalization for determinants without much effort.[4]
- The reasoning can be made on an abstract level and is not scattered with low-level implementation details as when working directly with lists of lists as in [20].
- The implementation, i. e., *code equations* for matrices can be freely chosen. For instance, one might take a lists-of-lists implementation, an implementation for sparse matrices, and so on. The current formalization provides an implementation via immutable arrays, namely Isabelle's *IArray* library with the Haskell serialization provided in [5].
- The type for matrices contains only the type of elements, but not dimensions. Hence, it is suitable for dealing with block matrices which is required for JNFs. On the other hand, one has to pay for this flexibility by working with explicit carrier sets which ensure that matrix operations are applied only on sensible arguments, i. e., where the dimensions fit together.

Due to the last point, we could provide support for block matrices. The basic functionality *split_block* allows to split a matrix

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}$$

into four smaller matrices A, B, C and D, where the sizes are determined by a given row index and column index. The opposite operation, combining four matrices into one matrix, is formalized as follows, where dim_r (resp. dim_c) returns the row dimension (resp. column dimension) of a matrix.

```
four_block_mat A B C D ≡
  let nra = dim_r A; nrd = dim_r D; nca = dim_c A; ncd = dim_c D
  in mat (nra + nrd) (nca + ncd) (λ(i, j).
    if i < nra then if j < nca then A_(i, j) else B_(i, j − nca) else
    if j < nca then C_(i − nra, j) else D_(i − nra, j − nca))
```

Now we present an essential functionality for convenient reasoning on JNFs: composing a block diagonal matrix

$$\begin{bmatrix} A_1 & & \\ & \ddots & \\ & & A_n \end{bmatrix}$$

from a list of diagonal blocks $A_1, \dots, A_n$. Using *four_block_mat*, the function can be naturally defined in Isabelle as follows, here, $\mathbf{0}_m\ nr\ nc$ is the zero matrix of size $nr \times nc$.

```
diag_block_mat [] = 0_m 0 0
diag_block_mat (A # As) = (
```

[3] Whereas in principle this can be expressed using Isabelle's special constant *undefined*, we expressed it in a way that allows a list or array-based implementation of vectors and matrices without checking indices when accessing elements. For instance, to implement the function for taking the i-th element of an n-dimensional vector which might be represented as $(n, [a_1, \dots, a_n])$, we can just take the i-th element of the list instead of having to compute "if $i < n$ then a_i else *undefined*".

[4] In the case of the Gauss-Jordan elimination, we did not reuse the existing formalization [4]; instead, we developed the algorithm from scratch, to test whether the machinery works fluently, and to detect parts where proofs become to tedious.

```
let B = diag_block_mat As
in four_block_mat A ($0_m$ ($dim_r$ A) ($dim_c$ B))
  ($0_m$ ($dim_r$ B) ($dim_c$ A)) B)
```

Note that this function, as well as other block matrix operations which we omit presenting here, cannot be expressed in HMA representation.

3.3 Strassen Algorithm (Strassen_Algorithm)

As a proof-of-concept, we utilize the block matrix functions to define and prove the soundness of the *Strassen algorithm* for matrix multiplication [22]. Its advantage is an improved asymptotic runtime in comparison the direct computation: the complexity goes down from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^{log_2(7)})$ for a matrix of dimension $n \times n$.

```
strassen (A, B) = (let nr = $dim_r$ A; n = $dim_c$ A; nc = $dim_c$ B
  in if strassen_too_small A B then A $\otimes_m$ B else
  if strassen_even A B then let
    nr2 = nr div 2; n2 = n div 2; nc2 = nc div 2;
    (A1, A2, A3, A4) = split_block A nr2 n2;
    (B1, B2, B3, B4) = split_block B n2 nc2;
    M1 = strassen (A1 $\oplus_m$ A4, B1 $\oplus_m$ B4);
    M2 = strassen (A3 $\oplus_m$ A4, B1);
    M3 = strassen (A1, B2 $\ominus_m$ B4);
    M4 = strassen (A4, B3 $\ominus_m$ B1);
    M5 = strassen (A1 $\oplus_m$ A2, B4);
    M6 = strassen (A3 $\ominus_m$ A1, B1 $\oplus_m$ B2);
    M7 = strassen (A2 $\ominus_m$ A4, B3 $\oplus_m$ B4);
    C1 = M1 $\oplus_m$ M4 $\ominus_m$ M5 $\oplus_m$ M7;
    C2 = M3 $\oplus_m$ M5;
    C3 = M2 $\oplus_m$ M4 ;
    C4 = M1 $\ominus_m$ M2 $\oplus_m$ M3 $\oplus_m$ M6
    in four_block_mat C1 C2 C3 C4
  else let
    nr' = nr div 2 * 2; n' = n div 2 * 2; nc' = nc div 2 * 2;
    (A1, A2, A3, A4) = split_block A nr' n';
    (B1, B2, B3, B4) = split_block B n' nc';
    C1 = strassen (A1, B1) $\oplus_m$ A2 $\otimes_m$ B3;
    C2 = A1 $\otimes_m$ B2 $\oplus_m$ A2 $\otimes_m$ B4;
    C3 = A3 $\otimes_m$ B1 $\oplus_m$ A4 $\otimes_m$ B3 ;
    C4 = A3 $\otimes_m$ B2 $\oplus_m$ A4 $\otimes_m$ B4
    in four_block_mat C1 C2 C3 C4)
```

Note that the Strassen algorithm considers three different cases: one for small matrices, where the standard matrix multiplication is invoked; one for matrices where all dimensions are even – this part is the main Strassen construction; and the last case where at least one dimension is odd – in this case we split off single lines and invoke Strassen's algorithm recursively on the main matrix, and perform the other multiplications of single or zero lines with standard matrix multiplication. Here, one can also observe that it is beneficial to allow 0-dimensional matrices: for instance if A is a matrix of dimension 61×80, then splitting A in the last case yields matrices of dimension 60×80 (*A1*), 60×0 (*A2*), 1×80 (*A3*), and 1×0 (*A4*), and there is no need to perform a further case analysis on which of the dimensions exactly are odd and which are even.

The soundness theorem

$dim_c\ A = dim_r\ B \Longrightarrow strassen\,(A, B) = A \otimes_m B$

is proven by induction w. r. t. the algorithm whose termination is ensured by the measure which takes twice the sum of all three dimensions (plus 1, if not all dimensions are even). It contains the typical precondition that occur when working in a carrier-based setting, namely that the soundness statement is only guaranteed if the dimensions of the matrices fit together. One does not see these guards in HMA, where the type system takes care of not multiplying two matrices with incompatible dimensions. However, the type system also would at least make it quite cumbersome to formulate Strassen's algorithm in HMA, if not impossible.

We also specified a setup for the code generator to always use Strassen's algorithm for matrix multiplication. However, by default it is not enabled since it restricts matrix multiplication to rings, so for instance multiplication of two matrices over natural numbers would no longer be executable.

4. Complexity Analysis and Jordan Normal Forms (Jordan_Normal_Form)

To analyze the growth rate of A^k we first assume that A is in JNF. Then, the results immediately transfer to all matrices B which are similar to A, since similarity does not change the asymptotic growth rate: the largest entries in A^k and B^k are related by a constant factor.

4.1 Growth Rate of Jordan Normal Forms

First, we formalize the size-n Jordan block

$$jordan\_block\ n\ a = \begin{bmatrix} a & 1 & & \\ & a & \ddots & \\ & & \ddots & 1 \\ & & & a \end{bmatrix}$$

in Isabelle as follows (here we use the letter a instead of λ since λ in Isabelle is reserved for function abstraction):

DEFINITION 4. $jordan\_block\ n\ a =$
$mat\ n\ n\ (\lambda(i, j).\ \textbf{if}\ i = j\ \textbf{then}\ a\ \textbf{else if}\ Suc\ i = j\ \textbf{then}\ 1\ \textbf{else}\ 0)$

From a list of pairs (n, a), each representing a Jordan block, we formalize a JNF as follows:

DEFINITION 5. $jordan\_matrix\ n\_as =$
$diag\_block\_mat\ (map\ (\lambda(n, a).\ jordan\_block\ n\ a)\ n\_as)$

In order to estimate the growth rate of J^k for a JNF J, we first formalize the closed formula for k-th power of Jordan blocks.

LEMMA 1. $(jordan\_block\ n\ a)^k =$
$mat\ n\ n\ (\lambda(i, j).\ \textbf{if}\ i \le j\ \textbf{then}\ \binom{k}{j-i} * a^{k+i-j}\ \textbf{else}\ 0)$

From the above formula we easily derive the desired bound for k-th power of Jordan blocks:

LEMMA 2. **assumes** $i < n$ **and** $j < n$
shows $\left|(jordan\_block\ n\ a)^k{}_{(i,\,j)}\right| \le |a|^{k+i-j} * (max\ 1\ k^{n-1})$

It is straightforward to prove this statement in Isabelle using existing results on binomial coefficients and the fact that the difference $j - i$ between two indices i and j are at most $n - 1$. From the inequality and Lemma 1, on paper one easily derives the precise characterization of the growth rate for k-th power of *jordan_block n a*, as follows:

- If $|a| = 1$, then the maximum norm is in $\Theta(k^{n-1})$.
- If $|a| < 1$, then the maximum norm is bounded by a constant.
- If $|a| > 1$, then the maximum norm grows exponentially.

In the formalization we proved only the upper bounds; this was not at all immediate in the case where $|a| < 1$. An intuitive argument would be the following: if $|a| < 1$ then $|a^k|$ gets exponentially smaller, and at some point it cancels the polynomially growing coefficient. Unfortunately, we did not find such a result in the Isabelle libraries.[5] To prove this result, we first manually proved results like Bernoulli's inequality:

LEMMA 3. $-1 \le x \Longrightarrow 1 + n * x \le (1 + x)^n$

[5] it might be possible to prove the claim for the real numbers by using derivatives, but in our formalization we are working in a more general setting which also allows complex numbers, etc.

Then we stepwise developed lemmas to eventually show that, in an Archimedean field, $a^x \cdot x^d$ can be bounded by a constant if $0 < a < 1$.

LEMMA 4.
```
assumes 0 < a and a < 1
  shows 0 < c ⟹ ∃ x. a^x ≤ c
    and ∃ b. ∀ x. a^x ∗ x ≤ b
    and ∃ b. ∀ x. a^x ∗ x^d ≤ b
```

After having these results formalized, the bounds for Jordan blocks are easily obtained. It then carries over to JNFs, by using exponentiation results for block-diagonal matrices, such as

$$\begin{bmatrix} A_1 & & \\ & \ddots & \\ & & A_p \end{bmatrix}^k = \begin{bmatrix} A_1^k & & \\ & \ddots & \\ & & A_p^k \end{bmatrix}$$

This leads to the following result for polynomial growth of JNFs, where *norm_bound A x* states that the norm of each matrix entry of A is bounded by x.

THEOREM 1.
```
assumes A = jordan_matrix n_as
    and ∀ (n,a) ∈ set n_as. n > 0 ⟶ |a| ≤ 1
    and ∀ (n,a) ∈ set n_as. |a| = 1 ⟶ n ≤ N
  shows ∃ c. ∀ k. norm_bound (A^k) (c + k^(N − 1))
```

Furthermore, the criterion is precise, in the sense that if the conditions on the Jordan blocks are not satisfied for N, then the growth rate is not within $\mathcal{O}(k^{N-1})$.

The theorem also tells that a matrix in JNF is polynomially bounded if and only if the norms of the diagonal elements are at most 1, and that the degree of the polynomial bound can be determined by the maximum size of the Jordan blocks corresponding to the diagonal elements of norm 1.

4.2 Growth Rate of Arbitrary Matrices

Next we generalize the previous result for growth rate of JNFs over arbitrary square matrices.

A JNF J is a JNF of a square matrix A, if they are similar. Hence, the proposition that a matrix A has a particular JNF is defined as follows:

DEFINITION 6.
```
jordan_nf A n_as ≡ mat_similar A (jordan_matrix n_as)
```

By using complexity results for similar matrices, we obtain the following corollary of Theorem 1:

THEOREM 2.
```
assumes jordan_nf A n_as
    and ∀ (n,a) ∈ set n_as. n > 0 ⟶ |a| ≤ 1
    and ∀ (n,a) ∈ set n_as. |a| = 1 ⟶ n ≤ N
  shows ∃ c. ∀ k. norm_bound (A^k) (c + k^(N − 1))
```

The result of Neurauter et al. [17] can be obtained from Theorem 1: Since the diagonal elements of a JNF J are exactly the eigenvalues of J, and since the eigenvalues of two similar matrices A and J are identical, it implies that the elements in A^k are polynomially bounded if the *spectral radius*, i. e., the maximum norm of the eigenvalues of A, is at most 1. If we are not to actually compute a JNF, the sizes of the Jordan blocks can be only estimated, e. g., by the multiplicities of the eigenvalues, corresponding to [17].

The above textual reasoning was not immediate to formalize in Isabelle; at the time of development we could not even find a definition of eigenvalues in the Isabelle libraries (including the AFP). So, we first had to develop several well-known results on eigenvalues in our matrix library, before we could eventually prove the following theorem. Here, the multiplicities of eigenvalues are encoded by providing the characteristic polynomial in a linearly factored form. The notation *[: −a, 1 :]* is Isabelle's syntax for the polynomial $x - a$.

THEOREM 3.
```
assumes A ∈ carrier_m n n
    and char_poly A = (∏ a ← as. [: −a, 1 :])
    and ∃ n_as. jordan_nf A n_as
    and ∀ a ∈ set as. |a| ≥ 1
    and ∀ a ∈ set as. |a| = 1 ⟶ length (filter (op = a) as) ≤ N
  shows ∃ c d. ∀ k. norm_bound (A^k) (c + d ∗ k^(N − 1))
```

Whereas Theorem 2 gives precise bounds, the approximation quality of Theorem 3 can be arbitrarily rough; for instance, the k-th power of the identity matrix of dimension N is actually constant, but Theorem 3 accepts only $\mathcal{O}(k^{N-1})$ or higher. Hence, we choose to base our certifier on Theorem 2.

Finally, observe that in both theorems we have to first prove the existence of a JNF. To this end, we follow the constructive approach of a textbook [19], that eventually yields an algorithm to compute JNFs from arbitrary matrices. The algorithm is based on three components described in the following sections: the Gram-Schmidt orthogonalization (Sect. 5), the Schur decomposition (Sect. 6), and elementary row and column transformations (Sect. 7).

5. Gram-Schmidt Orthogonalization (Gram_Schmidt)

In this section, we provide a formalization of the Gram-Schmidt orthogonalization algorithm that is applicable to complex numbers.

For real-valued vectors, the algorithm is already formalized by Divasón and Aransay [6]. For our purpose, however, we cannot assume real-valued vectors; even if the original matrix is real- or even integer-valued, its eigenvalues and eigenvectors are in general complex.

Since the algorithm is also of interest of field types such as *real* and *rat*, we do not assume the concrete datatype *complex*. Instead, we introduce a new class called *conjugatable_field*:

```
class conjugatable_field = field + fixes conjugate :: α ⇒ α
```

which assumes the following basic properties of conjugation (where the application of *conjugate* is denoted by overbar):

$$\overline{a+b} = \overline{a} + \overline{b} \qquad \overline{\overline{a}} = a \qquad \overline{-a} = -\overline{a}$$
$$\overline{a \cdot b} = \overline{a} \cdot \overline{b} \qquad \overline{0} = 0 \qquad \overline{a} = \overline{b} \Longleftrightarrow a = b$$

Another key property of conjugation is that $a \cdot \overline{a}$ is a non-negative real number for every $a \in \mathbb{C}$. Using the HOL class *ordered_comm_monoid_add*, we can express the property simply as follows:

```
class conjugatable_ordered_field =
    conjugatable_field + ordered_comm_monoid_add +
  assumes a ∗ ā ≥ 0
```

The base class *ordered_comm_monoid_add* also assumes *a ≥ b ⟹ c + a ≥ c + b*, which is essential for having the following property:

LEMMA 5.
```
fixes v :: α :: conjugatable_ordered_field vec
  assumes v ∈ carrier_v n
  shows v • v̄ = 0 ⟷ v = 0_v n
```

It is trivial that *rat* and *real* are instances of *conjugatable_ordered_field* by taking the identity functions as *conjugate*. Note that the orders $<$ and $\leq$ can be *partial*. Hence also for complex numbers, the following choice of the orders trivially satisfies the required assumptions.

```
instantiation complex :: conjugatable_ordered_field
begin
  definition [simp]: conjugate ≡ cnj
  definition [simp]: x < y ≡ Im x = Im y ∧ Re x < Re y
  definition [simp]: x ≤ y ≡ Im x = Im y ∧ Re x ≤ Re y
  instance by (intro_classes, auto simp: complex.expand)
end
```

The Gram-Schmidt algorithm is presented as Algorithm 1.

```
Algorithm 1: Gram-Schmidt Orthogonalization
  Input: Linearly independent vectors $\boldsymbol{v}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_m$.
  Output: Orthogonal vectors $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_m$ such that
          $\mathsf{span}\{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_m\} = \mathsf{span}\{\boldsymbol{v}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_m\}$.
  For $i = 2, \ldots, m$, compute $\boldsymbol{v}_i = \boldsymbol{w}_i - \sum_{j=1}^{i-1} \frac{\boldsymbol{v}_j^{\mathrm{H}} \boldsymbol{w}_i}{\boldsymbol{v}_j^{\mathrm{H}} \boldsymbol{v}_j} \boldsymbol{v}_j$.
  Return $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_m$.
```

The correctness of the algorithm is stated as follows:

THEOREM 4 (Gram-Schmidt). *If input vectors $\boldsymbol{v}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_m$ are linearly independent, then Algorithm 1 returns orthogonal vectors $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_m$ that span the same space as the input vectors.*

To formalize the above claim in Isabelle/HOL, we require the notions such as orthogonality, linear dependency and spans.

Orthogonality of a list of vectors is defined as follows.

DEFINITION 7. *corthogonal vs* $\equiv$
$\forall\, i <$ *length vs.* $\forall\, j <$ *length vs.* $(vs\ !\ i) \bullet \overline{(vs\ !\ j)} = 0 \longleftrightarrow i \neq j$

For notions like linear dependency or spans, we employ the vector space library by Lee [14]. More precisely, in theory `VS_Connect` we introduce a locale *vec_space*, where the element type is assumed to be a *field*, and the dimension *n* is fixed.

```
locale vec_space = fixes f_ty :: α :: field itself and n :: nat
begin
```

Since the dimension is fixed in this locale, it is easy to show that vector operations satisfy certain closure properties, and is actually a sublocale of *vectorspace*.

```
  sublocale vectorspace F V . . .
end
```

Here *F* and *V* are records that specify field operations and vector operations, respectively. Now we obtain the formalizations of *span*, *lin_dep*, *lincomb*, etc., and the correctness of the Gram-Schmidt algorithm is formalized as follows:

```
locale cof_vec_space = vec_space f_ty
  for f_ty :: α :: conjugatable_ordered_field itself
begin
  theorem gram_schmidt:
    assumes set ws ⊆ carrier_V n and distinct ws
      and ¬ lin_dep (set ws)
      and us = gram_schmidt n ws
    shows span (set ws) = span (set us)
      and corthogonal us
      and set us ⊆ carrier_V n
      and length us = length ws
end
```

6. Schur Decomposition (`Schur_Decomposition`)

In this section, we describe a formalization of the *Schur decomposition*. The main result is stated as follows, where C is an arbitrary field of the class *conjugatable_ordered_field*, e. g., $\mathbb{Q}$, $\mathbb{R}$, or $\mathbb{C}$.

THEOREM 5 (Schur). *For every matrix $A \in C^{n\times n}$ with n eigenvalues (counting multiplicities), there exists an upper-triangular matrix B of the following form:*

$$B = \begin{bmatrix} \lambda_1 & * & \cdots & * \\ & \lambda_2 & \ddots & \vdots \\ & & \ddots & * \\ & & & \lambda_n \end{bmatrix}$$

that is similar to A. Moreover, the diagonal elements $\lambda_1, \ldots, \lambda_n$ of B are the eigenvalues of A.

The proof is done constructively by actually formulating Algorithm 2 in Isabelle. It takes a square matrix A and its eigenvalues, and computes an upper-triangular matrix that is similar to A.

```
Algorithm 2: Schur Decomposition
  Input: A matrix $A \in C^{n\times n}$ and the list of its eigenvalues
         $\lambda_1, \ldots, \lambda_n$ (repeated according to their multiplicity).
  Output: A pair $\langle B, P\rangle$ of matrices, such that $B$ is
          upper-triangular, $P$ is invertible, and $A = PBP^{-1}$.
  1 If $n = 0$ then return $\langle A, I\rangle$.
  2 Find an eigenvector $\boldsymbol{v}_1$ corresponding to $\lambda$.
  3 Extend $\boldsymbol{v}_1$ to an orthogonal basis $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_n$ using the
    Gram-Schmidt algorithm.
  4 Let $V = [\boldsymbol{v}_1\ \boldsymbol{v}_2\ \ldots\ \boldsymbol{v}_n]$ and compute $V^{-1}$.
  5 Compute $V^{-1}AV = \begin{bmatrix} \lambda_1 & b_{12} & \cdots & b_{1n} \\ 0 & & & \\ \vdots & & A' & \\ 0 & & & \end{bmatrix}$.
  6 Let $\langle B', P'\rangle = \text{Schur\_Decomposition}(A', \lambda_2, \ldots, \lambda_n)$.
  7 Return $\langle \begin{bmatrix} \lambda_1 & b_{12} & \cdots & b_{1n} \\ 0 & & & \\ \vdots & & B' & \\ 0 & & & \end{bmatrix}, V \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & & & \\ \vdots & & P' & \\ 0 & & & \end{bmatrix} \rangle$.
```

For finding eigenvectors w. r. t. some eigenvalue λ, we take the standard approach, i. e., we simplify the matrix $A - \lambda I$ via Gauss-Jordan elimination into row-echelon form and then extract the first basis vector of the kernel of this simplified matrix.

In step 3, clearly the Gram-Schmidt orthogonalization can be employed, but before that we must have a basis that contains $\boldsymbol{v}_1$. Intuitively, this is easy by properly choosing the unit vectors. However, the formalization requires tedious reasonings which are now available in theory `Missing_VectorSpace`.

LEMMA 6 (Step 3). *Any non-zero vector $\boldsymbol{v}_1 \in C^n$ can be extended to an orthogonal basis $\{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_n\}$.*

Textbook algorithms compute *orthonormal* (i. e., orthogonal and normalized) bases in step 3. In our version, we omit normalizing the orthogonal basis, in order to avoid introducing unnecessary square roots: the execution of real numbers in Isabelle [24] does neither support nested roots as in $\sqrt{1+\sqrt{2}}$ nor does it support the combination of different roots as in $\sqrt{2}+\sqrt{3}$.

EXAMPLE 1. *Consider the randomly generated matrix*

$$\begin{bmatrix} 3 & 2 & 7 \\ 4 & 1 & 8 \\ 1 & 4 & 5 \end{bmatrix}$$

Our version of the Schur decomposition delivers

$$\begin{bmatrix} 0 & \frac{1}{5612}(-15705 - 45\sqrt{177}) & a + b\sqrt{177} \\ 0 & \frac{1}{2}(9+\sqrt{177}) & c + d\sqrt{177} \\ 0 & 0 & \frac{1}{2}(9-\sqrt{177}) \end{bmatrix}$$

as a result with a, b, c, d being large but rational numbers, e. g., $a = -\frac{1136075823410445}{737949547849666}$. In contrast, a manually performed computation within Mathematica[6] which uses orthonormal basis in step 3

[6] All statements about Wolfram Mathematica in this paper are based on version 10.2.

results in the following matrix:

$$\begin{bmatrix} 0 & -(7\sqrt{3}+19\sqrt{59})\sqrt{\frac{2}{2771-107\sqrt{177}}} & \frac{-1283\sqrt{183}+145\sqrt{3599}}{61\sqrt{2771-107\sqrt{177}}} \\ 0 & \frac{1}{2}(9+\sqrt{177}) & 11\sqrt{\frac{2}{61}} \\ 0 & 0 & \frac{1}{2}(9-\sqrt{177}) \end{bmatrix}$$

This matrix contains nested roots and would lead to a runtime exception.

A benefit of normalizing the basis is that V in step 4 becomes *unitary*, i. e., $V^{-1} = V^{\mathrm{H}}$. However, even if the orthogonal basis is not normalized, V^{-1} is easily computed.

LEMMA 7. *Let $\boldsymbol{v}_1, \dots, \boldsymbol{v}_n \in C^n$ be orthogonal vectors. Then $V = [\boldsymbol{v}_1 \; \cdots \; \boldsymbol{v}_n]$ is invertible with the following inverse:*

$$V^{-1} = \begin{bmatrix} \dfrac{\boldsymbol{v}_1}{\boldsymbol{v}_1^{\mathrm{H}}\boldsymbol{v}_1} & \cdots & \dfrac{\boldsymbol{v}_n}{\boldsymbol{v}_n^{\mathrm{H}}\boldsymbol{v}_n} \end{bmatrix}^{\mathrm{H}}$$

The following lemma is the key for having upper-triangular matrix as output.

LEMMA 8. *In step 5, the first column of $V^{-1}AV$ is $[\lambda_1 \; 0 \; \cdots \; 0]^{\mathrm{T}}$.*

In order to inductively apply the algorithm to the submatrix A', we also require the following lemma:

LEMMA 9. *The eigenvalues of A' are $\lambda_2, \dots, \lambda_n$.*

Finally, we present the formalized correctness result for Algorithm 2. Here, the algorithm is formalized as *schur_decomposition*, which returns P^{-1} (as *P'*) in addition to B and P.

THEOREM 6.
```
fixes A :: α :: conjugatable_ordered_field mat
  assumes A ⊆ carrier_m n n
  and char_poly A = (∏(e :: α) ← es. [: −e, 1:])
  and schur_decomposition A es = (B, P, P')
 shows A = P ⊗m B ⊗m P'
  and P' ⊗m P = 1m n
  and upper_triangular B
  and mat_diag B = es
```

7. Triangular Matrices to Jordan Normal Forms (Jordan_Normal_Form_Existence)

In the previous section, we formalized the Schur decomposition that delivers upper-triangular matrices from arbitrary square matrices. In this section, we describe an algorithm that transforms such an upper-triangular matrix into a JNF.

The following algorithm is derived and improved from the textbook [19, Sect. 11.1.4], where the algorithm is not explicitly specified but only applied on an example. It only performs elementary row and column operations which preserve similarity. For instance, *add_col_sub_row* adds a multiple of one column of A to another one, and at the same time performs the inverse operation – subtraction of a multiple – as a row operation. Hence, the whole operation corresponds to $P^{-1}AP$ where P is the elementary matrix corresponding to the addition of a column. In the same way there are similarity preserving operations for multiplication and swapping (*mult_col_div_row* and *swap_cols_rows*).

For all of these elementary operations we once prove that they preserve similarity which is then used to show that the overall algorithm preserves similarity. However, for the more difficult task – that the resulting matrix is in JNF – we provide lemmas that describe the resulting matrix after applying an elementary matrix. For instance, the following lemma characterizes the effect of *mult_col_div_row*.

LEMMA 10.
```
assumes i < dim_r A and i < dim_c A
   and j < dim_r A and j < dim_c A and a ≠ 0
 shows (mult_col_div_row a k A)_(i, j) =
   (if i = k ∧ j ≠ i then inverse a * A_(i, j)
    else if j = k ∧ j ≠ i then a * A_(i, j) else A_(i, j))
```

In the following, we mainly present the textual description of the algorithm that we extracted from the example calculation in the textbook. For the reconstruction, we often used Mathematica to execute and test prototypes of the algorithm on individual examples.

1. Let us start with an upper-triangular matrix:

$$A = \begin{bmatrix} \lambda_1 & a_{12} & \cdots & a_{1n} \\ & \lambda_2 & \ddots & \vdots \\ & & \ddots & a_{n-1,n} \\ & & & \lambda_n \end{bmatrix}$$

   For every i and j ($i < j$) such that $\lambda_i \neq \lambda_j$, eliminate the i-th row j-th column element a_{ij}, by

   add_col_sub_row $(a_{ij}/(\lambda_j - \lambda_i))\; i\; j$

   The iteration should be first by increasing j and then by decreasing i in the inner loop.

   After this first step, it is ensured that whenever $\lambda_i \neq \lambda_j$ then $a_{ij} = 0$.

2. Consider an index $j > 2$. If there is an index $i < j-1$ such that $\lambda_i = \lambda_j$ but $\lambda_{i+1}, \dots, \lambda_{j-1}$ are different from λ_j, then move λ_j next to λ_i, by

   swap_cols_rows $(i+1)\; j$
   swap_cols_rows $(i+2)\; j$
   $\vdots$
   swap_cols_rows $(j-1)\; j$

   Afterwards λ_j will be moved to $(i{+}1)$-th row $(i{+}1)$-th column, and the rows and columns in between are shifted one row down and one column right. Repeating this operation by increasing j, the diagonal elements will be grouped as in the following shape:

$$A' = \begin{bmatrix} \begin{bmatrix} \lambda'_1 & * & \cdots & * \\ & \ddots & \ddots & \vdots \\ & & \ddots & * \\ & & & \lambda'_1 \end{bmatrix} & & \\ & \ddots & \\ & & \begin{bmatrix} \lambda'_m & * & \cdots & * \\ & \ddots & \ddots & \vdots \\ & & \ddots & * \\ & & & \lambda'_m \end{bmatrix} \end{bmatrix}$$

   This second step does not occur in the textbook description. We added it since we do not assume that the eigenvalues on the diagonal are grouped together. Note that it is important to execute step 2 after step 1 since changing the order would destroy the triangular property of the matrix.

3. Transform each diagonal block

$$B = \begin{bmatrix} \lambda & * & \cdots & * \\ & \lambda & \ddots & \vdots \\ & & \ddots & * \\ & & & \lambda \end{bmatrix}$$

   into JNF. Note that the result is not necessarily a single Jordan block. The transformation is done column-by-column: Suppose that the top-left $(k-1) \times (k-1)$ submatrix of B is already in

JNF. Hence for every $i < k$, the i-th row is of form

$$[\overbrace{0\ \cdots\ 0\ \lambda\ 0\ \cdots\ 0\ b_{ik}}^{k \text{ elements}}\ *\ \cdots\ *] \qquad (1)$$

if it is the bottom of a Jordan block, or is of form

$$[\overbrace{0\ \cdots\ 0\ \lambda\ 1\ 0\ \cdots\ 0\ b_{ik}}^{k \text{ elements}}\ *\ \cdots\ *] \qquad (2)$$

otherwise. The k-th column is treated as follows:

a) For rows of form (2), eliminate b_{ik} by

$$\mathit{add\_col\_sub\_row}\ (-b_{ik})\ (i+1)\ k$$

b) Find a largest Jordan block (in the $(k-1) \times (k-1)$ submatrix) whose bottom row is of form (1) with $b_{ik} \neq 0$.

c) If such a Jordan block does not exist, then the $k \times k$ submatrix is already in JNF. Otherwise, let l be the row index of the bottom of the Jordan block. Eliminate every other nonzero element b_{ik} as follows:

$$\begin{gathered}\mathit{add\_col\_sub\_row}\ (b_{ik}/b_{lk})\ i\ l \\ \mathit{add\_col\_sub\_row}\ (b_{ik}/b_{lk})\ (i-1)\ (l-1) \\ \mathit{add\_col\_sub\_row}\ (b_{ik}/b_{lk})\ (i-2)\ (l-2) \\ \vdots\end{gathered}$$

where the number of steps is determined by the size of the Jordan block left-above of the i-th row i-th column.

d) Normalize the l-th row k-th column value to 1:

$$\mathit{mult\_col\_div\_row}\ b_{lk}^{-1}\ k$$

e) Move the 1 down from row l to row $k-1$:

$$\begin{gathered}\mathit{swap\_cols\_rows}\ (l+1)\ k \\ \mathit{swap\_cols\_rows}\ (l+2)\ k \\ \vdots \\ \mathit{swap\_cols\_rows}\ (k-1)\ k\end{gathered}$$

The full definition of the algorithm requires roughly 140 lines of Isabelle. It contains several auxiliary functions, e. g., to identify the Jordan blocks in a matrix in step 3 (b), but there are no serious deviations between the textual description and the formalization.

In order to prove the soundness of the algorithm, i. e., that the result is in JNF, we defined several invariants that are established *and maintained* throughout the algorithm. Examples are that A is triangular, A has the eigenvalues grouped together, A has only a single eigenvalue, a submatrix of A is in JNF, etc.

Most of the correctness proofs are of the form "if A satisfies invariants X and Y then *step* α A satisfies invariants X and Z". This approach admits separate proofs for each of the steps which solely reason about the invariants.

However, some of the intermediate steps do not preserve certain invariants, and in this case a lemma of the above form does not work. In order to prove the soundness of such steps we must investigate explicit descriptions for the intermediate matrices.

As an example we would like to mention step 3 (c).

EXAMPLE 2. *Consider starting step 3 (c) on matrix*

$$A = \begin{bmatrix} 3 & 1 & 0 & & & 0 \\ 0 & 3 & 1 & & & 0 \\ 0 & 0 & 3 & & & 5 \\ & & & 3 & 1 & 0 \\ & & & 0 & 3 & \boxed{8} \\ & & & & & 3 \end{bmatrix}$$

The bottom of the largest Jordan block is row 3; so we have to delete the 8 *in row 5. To this end, we first execute* add_col_sub_row $\frac{8}{5}$ *on rows 4 and 2 which results in matrix B below, which is not triangular. However, after executing* add_col_sub_row $\frac{8}{5}$ *on rows 3 and 1, we get matrix C which again satisfies all previous invariants and has additionally deleted the* 8*.*

$$B = \begin{bmatrix} 3 & 1 & 0 & & & 0 \\ 0 & 3 & 1 & & & 0 \\ 0 & 0 & 3 & & & 5 \\ & & \boxed{\frac{8}{5}} & 3 & 1 & 0 \\ & & & 0 & 3 & 0 \\ & & & & & 3 \end{bmatrix} \qquad C = \begin{bmatrix} 3 & 1 & 0 & & & 0 \\ 0 & 3 & 1 & & & 0 \\ 0 & 0 & 3 & & & 5 \\ & & & 3 & 1 & 0 \\ & & & 0 & 3 & \boxed{0} \\ & & & & & 3 \end{bmatrix}$$

In contrast to our preliminary version of the algorithm (AFP submission [26]) that always works on the whole matrix, step 3 of the current version works modularly on submatrices (as in the textbook). The advantage of this change is that less invariants have to be ensured throughout step 3. For instance, the invariant $\lambda_i \neq \lambda_j \implies a_{ij} = 0$ that is ensured by step 1 has no longer to be maintained in step 3, as we know the diagonal elements are identical. The switch to submatrices required some further results on similarity of block matrices which have not been available when initially formalizing the algorithm.

We also deviate from [19] in step 3(e). Here, the textbook suggests that a single swapping would suffice to move the 1 down from row l to row $k-1$, whereas we use several swappings. Actually, this is a mistake in the textbook: the result of the calculation $P(37)D_7(b)T_2D_7(b^{-1})P(37)$ on page 401 is not the claimed JNF on the left below, but a non JNF on the right.

$$\begin{bmatrix} \lambda & 1 & & & & & \\ & \lambda & 1 & & & & \\ & & \lambda & 1 & & & \\ & & & \lambda & 1 & & \\ & & & & \lambda & 1 & \\ & & & & & \lambda & \\ & & & & & & \lambda \end{bmatrix} \qquad \begin{bmatrix} \lambda & 1 & & & & & \\ & \lambda & 1 & & & & \\ & & \lambda & & & & \\ & & & \lambda & 1 & & \\ & & & & \lambda & 1 & \\ & & & & & \lambda & \\ & & & 1 & & & \lambda \end{bmatrix}$$

Connecting the different steps yields the algorithm *triangular_to_jnf_vector*. It additionally converts the final JNF from a matrix into a vector representation. The main soundness theorem states that every upper triangular matrix has a JNF.

THEOREM 7. **fixes** *A :: α :: field mat*
assumes *A* $\in$ *carrier$_m$ n n* **and** *upper_triangular A*
shows *jordan_nf A* (*triangular_to_jnf_vector A*)

Combining this theorem with the Schur decomposition yields the characterization of existence of JNFs.

THEOREM 8. **fixes** *A :: α :: conjugatable_ordered_field mat*
assumes *A* $\in$ *carrier$_m$ n n*
shows ($\exists$ *n_as. jordan_nf A n_as*) $\longleftrightarrow$
($\exists$ *as. char_poly A* = ($\prod$ *a* $\leftarrow$ *as.* [: $-$*a*, *1* :]))

In combination with the fundamental theorem of algebra we easily derive that every complex matrix has a JNF.

8. Polynomial Factorization (Square_Free_Factorization)

In order to automatically determine the eigenvalues of a matrix, we have to factor the characteristic polynomial. To this end, we implement the following heuristic:

1. Apply Yun's algorithm to get a square-free representation of p, i. e., $p = c \cdot p_1^1 \cdot p_2^2 \cdot \ldots \cdot p_k^k$ where c is a constant, no p_i has multiple roots, and two different p_i's share no common root.

2. Try to factor each p_i further, with the help of a set of potential factors.
3. If any of the resulting polynomials is quadratic, then use the well-known formula to compute the roots.[7]

The heuristic delivers either a fully factored polynomial, or a partially factored one. In the latter case for polynomials over $\mathbb{C}$, some non-factored polynomial q of degree 3 or higher remains.

In step 2, we choose the potential factors as follows. For matrices used for complexity analysis via matrix interpretations, we know that 1 is often an eigenvalue. Thus, we always add $x - 1$ to the set of potential factors. Moreover, we add $x - a$ to the set of potential factors for every matrix element a of A. We conclude that the algorithm always succeeds if at most two eigenvalues of A do not occur in A. We further know, by the Perron-Frobenius theorem, that if the spectral radius of a non-negative matrix A is 1 then all eigenvalues with norm 1 are n-th roots of 1, where n is smaller than the dimension of A. So we always test for the square roots, cube roots, and quartic roots of 1, which are $\{\pm 1, \pm \mathrm{i}, \frac{-1 \pm \mathrm{i}\sqrt{3}}{2}\}$. Introducing $\sqrt{3}$ is not welcome again for the Isabelle implementation of the reals, but this can be avoided. To be more precise, we replace the two potential roots $\frac{-1 \pm \mathrm{i}\sqrt{3}}{2}$ by one potential factor $x^2 + x + 1$, which gives the same precision: For real valued polynomials, the conjugate of a complex root is also a root. So, if one of $\frac{-1 \pm \mathrm{i}\sqrt{3}}{2}$ is a root, then the other is too, and thus $x^2 + x + 1$ is a factor.

In the remainder of this section we describe the most challenging task in the factorization algorithm above, namely the formalization of Yun's algorithm in Isabelle.

The algorithm is based on computing derivatives (denoted by p') and greatest common divisors (denoted by $\gcd(p, q)$) of polynomials. Both operations are available in Isabelle. Nevertheless, for derivatives we made a local copy, since the existing definition is restricted on polynomials over *real_normed_field*, whereas Yun's algorithm also works on other carriers, e. g., $\mathbb{Q}$. Our algorithm works for arbitrary fields of characteristic 0 (*field_char_0* in Isabelle).

Instead of presenting the Isabelle definition here, we just give the defining equations of Yun's algorithm. Here we further assume that p is monic, i. e., the leading coefficient of p is 1.

$$\begin{aligned} b_0 &= \frac{p}{\gcd(p, p')} & b_{n+1} &= \frac{b_n}{a_n} \\ c_0 &= \frac{p'}{\gcd(p, p')} & c_{n+1} &= \frac{d_n}{a_n} \\ d_n &= c_n - b'_n & a_n &= \gcd(b_n, d_n) \end{aligned}$$

The algorithm stops if $b_n = 1$, and then returns $a_0^1 a_1^2 \dots a_{n-1}^n$ as the desired factorization.

In the formalization, we used a command to define *partial* functions, for which termination is not guaranteed. The reason is that the algorithm, which takes two arbitrary polynomials as input, actually does not terminate if it is started with wrong values; it should be started with b_0 and c_0. Moreover, the termination argument relies on the fact that the algorithm actually computes a square-free factorization, so a termination proof is only possible after having proved soundness.

In order to prove soundness, let $q_0^1 \dots q_n^{n+1}$ be a square-free factorization of p – we also formalized that such a factorization always exists. We prove the following closed-form expressions for a_k, b_k, c_k, and d_k in a large mutual induction on k. Here, we write $q_k \dots q'_i \dots q_n$ meaning $q_k \dots q_{i-1}\, q'_i\, q_{i+1} \dots q_n$.

$$\begin{aligned} a_k &= q_k \\ b_k &= q_k \dots q_n \\ c_k &= \sum_{i=k}^{n} (i + 1 - k)\, q_k \dots q'_i \dots q_n \\ d_k &= q_k \sum_{i=k+1}^{n} (i - k)\, q_{k+1} \dots q'_i \dots q_n \end{aligned}$$

The formulas for b_{k+1}, c_{k+1} and d_k only need straightforward calculations. For instance, we deduce the formula for d_k, assuming b_k and c_k are done, as follows:[8]

$$\begin{aligned} d_k &= c_k - b'_k \\ &= \left(\sum_{i=k}^{n} (i + 1 - k)\, q_k \dots q'_i \dots q_n \right) - (q_k \dots q_n)' \\ &= \sum_{i=k}^{n} (i - k)\, q_k \dots q'_i \dots q_n \\ &= (k - k)\, q'_k\, q_{k+1} \dots q_n + \sum_{i=k+1}^{n} (i - k)\, q_k\, q_{k+1} \dots q'_i \dots q_n \\ &= 0 + q_k \cdot \sum_{i=k+1}^{n} (i - k)\, q_{k+1} \dots q'_i \dots q_n \end{aligned}$$

Formalization becomes more involved for a_k, b_0, and c_0; here we show the result for a_k.

$$\begin{aligned} a_k &= \gcd(b_k, d_k) \\ &= \gcd\left(q_k \dots q_n,\; q_k \cdot \sum_{i=k+1}^{n} (i - k)\, q_{k+1} \dots q'_i \dots q_n \right) \\ &\overset{(1)}{=} q_k \cdot \gcd\left(q_{k+1} \dots q_n,\; \sum_{i=k+1}^{n} (i - k)\, q_{k+1} \dots q'_i \dots q_n \right) \\ &\overset{(2)}{=} q_k \cdot 1 \end{aligned}$$

In equality (1), we just extract the common factor q_k out of the greatest common divisor, but formalizing this single step required roughly 250 lines. We needed the following lemma which was not found in the Isabelle distribution:

LEMMA 11. *monic p ⟹ gcd (p ∗ q) (p ∗ r) = p ∗ gcd q r*

The lemma is proved using results from HOL-Algebra [3]; we formalize that polynomials over fields form a *unique factorization domain*, and then apply results from HOL-Algebra that connect divisibility with the set of irreducible factors.

Also for equality (2) we use the connection to HOL-Algebra to prove the following lemma, where *dvd* is Isabelle's predicate for "divides".

LEMMA 12. *irreducible p ⟹ p dvd q ∗ r ⟹ p dvd q ∨ p dvd r*

We prove equality (2) as follows. Consider an irreducible and non-constant factor r that divides $q_{k+1} \dots q_n$. Using Lemma 12, we obtain some $l \in \{k + 1, \dots, n\}$ such that r divides q_l. Next, suppose that r divides the sum:

$$\sum_{i=k+1}^{n} (i - k)\, q_{k+1} \dots q'_i \dots q_n$$

[7] On purpose we did not integrate the solved forms for cubic or quartic polynomials, which require nested square/cube/quartic roots: these operations are not supported in Isabelle's current implementation of real numbers.

[8] Similar reasoning was done in the formalization, although with a slight difference: the q_i's are not provided as single polynomials, but as products of irreducible polynomials.

Since r divides q_l, it divides every $q_{k+1} \dots q_i' \dots q_n$ with $i \neq l$. Thus, r must also divide $(l-k)\, q_{k+1} \dots q_l' \dots q_n$.

Recall that we are assuming $q_0^1 \dots q_n^{n+1}$ to be a square-free factorization. That is, as r divides q_l, it cannot divide other q_i's. Thus, again using Lemma 12, we conclude that r divides q_l'.

Now write q_l as the product $r_1 \dots r_m$ of irreducible factors. As we are assuming r is irreducible, we obtain $r = r_j$ for some $j \leq m$. We know that r divides q_l', which is

$$q_l' = \sum_{i=1}^{m} r_1 \dots r_i' \dots r_m$$

We do a similar reasoning as above, concluding r_j divides r_j'. This is possible only if r_j is a constant, deriving contradiction.

9. Certifying Complexity Analysis (`Jordan_Normal_Form_Uniqueness`)

In principle, now we have all ingredients to ensure polynomial growth rates of A^k of a given matrix A. First we consider a naive approach that tries to compute a JNF of A. Unfortunately, it turns out that the approach easily fail into runtime errors, due to the limited support for irrational numbers in Isabelle implementation. Hence, we provide another algorithm that do not compute the entire JNF. The second approach will not cause the runtime error, if the input matrix does not have already irrational elements, and eigenvalues fall in the supported form.

9.1 Naive Approach

First, we consider Algorithm 3 which naively combines the previously formalized algorithms to compute JNFs.

```
Algorithm 3: Naive Certification Algorithm
  Input: A square matrix $A$ and a degree $N$.
  Output: Accept if $A^k \in \mathcal{O}(k^N)$, and reject otherwise.
1 Invoke the factorization algorithm on the characteristic
  polynomial to determine the eigenvalues.
2 Ensure that the norms of all eigenvalues is at most 1.
3 Compute the JNF by the algorithms in Sect. 5–7
4 Figure out the size $n$ of the largest Jordan block which has an
  eigenvalue with norm 1.
5 Accept if $n - 1 \leq N$ and reject otherwise.
```

The soundness of Algorithm 3 easily follows Theorem 2. On the other hand, it is not complete in the sense that the eigenvalues are not always computed. Moreover, due to the limits of the implementation of irrational numbers, an execution of the algorithm may cause runtime errors even if all eigenvalues have been determined.

As testing the algorithm on examples, we realized that the latter problem is significant.

EXAMPLE 3. *Take the following matrix*

$$A = \begin{bmatrix} 0 & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 1 & 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{3} & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 8 & 0 & 9 & 0 & \frac{1}{3} \\ 2 & 0 & 0 & 4 & 1 & 0 \end{bmatrix}$$

as input. The factorization algorithm automatically figures out the six eigenvalues: $\pm\frac{1}{3}\sqrt{3}, \pm\frac{1}{3}\sqrt{3}, \pm\frac{1}{2}\sqrt{2}$. *After successfully checking step 2, we continue with step 3, where after the first four iterations of the Schur decomposition (with eigenvalues* $\pm\frac{1}{3}\sqrt{3}$*), the remaining submatrix is*

$$B = \begin{bmatrix} a\sqrt{3} & b \\ c & d\sqrt{3} \end{bmatrix}$$

with $a, b, c, d \in \mathbb{Q}$. *In order to compute an eigenvector for eigenvalue* $\frac{1}{2}\sqrt{2}$ *of* B *we need to compute the matrix* $B - \frac{1}{2}\sqrt{2}I$. *Unfortunately, at this point Isabelle's implementation of real number raises a runtime error, as it does not support computation involving different square roots as in* $a\sqrt{3} - \frac{1}{2}\sqrt{2}$.

To circumvent the problem, we next formalize another algorithm which determines the sizes of Jordan blocks, without explicitly computing the JNF via Schur decomposition and the algorithm for upper-triangular matrices. Nevertheless, the results of Sect. 6–8 are essential since the soundness of the upcoming algorithm relies upon the existence of a JNF.

9.2 Avoiding Multiple Square Roots

We provide an algorithm that, given a matrix A and a potential Jordan block (n, λ), returns the number of occurrences of this block in a JNF of A. This number is valid for *any* JNF of A, indicating the uniqueness of JNFs up to permutation of the blocks.

At this point we follow the proofs based on generalized eigenspaces, a topic that we would have liked to avoid in the beginning.

Internally, the algorithm uses a function $d(n)$ that computes the dimension of the kernel of $(A - \lambda I)^n$, where it can be shown that the number of occurrences of Jordan block (n, λ) is exactly $2d(n) - d(n+1) - d(n-1)$. To achieve this result, we formalize several results on the kernel of a matrix, for instance that similarity transformations do not change the dimension of the kernel, or that the kernel dimension of a diagonal block matrix is exactly the sum of the kernel dimensions of each block.

Using this new algorithm for determining the sizes of Jordan blocks, we end up in an improved certification algorithm, Algorithm 4, which is available as *mat_estimate_complexity_jb* in theory `Jordan_Normal_Form_Complexity_Approximation`. The new version never raises runtime errors (unless the input matrix is already irrational), and can even succeed without figuring out all eigenvalues. To this end, it utilizes Sturm's method [7, 23] to determine the number of real roots of a polynomial in a given interval (ignoring multiplicities).

```
Algorithm 4: Certification Algorithm for Matrix Growth
  Input: A matrix $A \in C^{n \times n}$, $C \in \{\mathbb{Z}, \mathbb{Q}, \mathbb{R}\}$, a degree $N$.
  Output: Accept if $A^k \in \mathcal{O}(k^N)$, and reject otherwise.
1 Convert $A$ into a matrix over complex numbers.
2 Invoke the factorization algorithm on the characteristic
  polynomial, resulting in a set of eigenvalues $\lambda_i$ and a list of
  real valued polynomials $q_j$ with degrees $d_j \geq 3$ which
  contain the remaining eigenvalues as roots.
3 For each eigenvalue $\lambda_i$, ensure $|\lambda_i| \leq 1$; if $|\lambda_i| = 1$, then
  figure out the size $n$ of the largest Jordan block for $\lambda_i$ and
  ensure $n - 1 \leq N$.
4 For each $q_j$ with degree $d_j$ determine the number of different
  real roots in the open interval $(-1, 1)$ with the help of
  Sturm's method. Ensure that this number equals $d_j$.
```

Collecting all results of the previous sections it is actually not difficult to prove soundness of Algorithm 4. Still, there is some tedious reasoning involved, namely in performing all the conversion between different types. This is required in step 1 (for matrices), and in step 4 where the real-valued polynomial of type *complex poly* needs to be converted into a polynomial of type *real poly*, so that Sturm's method can be applied.

The fact that the algorithm does not raise runtime errors on rational matrices as input is proven as follows: The factorization algorithm has been defined in a way that it does not introduce irrational numbers except in the last step where the explicit roots for polynomials of degree 2 are calculated. Hence, each q_j is a rational polynomial and each λ_i is of the form $a + b\sqrt{c}$ for rational numbers a, b, c. If a λ_i has this form, then also the computation of the largest Jordan block will not raise runtime errors in step 3, since all intermediate values will have the form $a' + b'\sqrt{c}$ for varying $a', b' \in \mathbb{Q}$ and for the same fixed c (this form is fully supported by the implementation of the real numbers). Finally, the input to Sturm's algorithm are just rational polynomials, where also a runtime error cannot occur.

9.3 Completeness

By the completeness of Theorem 2, Algorithm 4 is complete if all eigenvalues are determined. This covers, for instance, upper- or lower-triangular matrices, since then the eigenvalues appear as elements in the matrix, which are tested in the factorization.

We further argue that the algorithm is complete, if all eigenvalues are real. So let the growth rate of A^k be within $\mathcal{O}(k^N)$ and execute the algorithm on A and N. Clearly steps 1 and 2 cannot fail. Step 3 cannot fail either, since any eigenvalue λ_i that is refused in step 3 would also violate the assumption on the growth rate. Finally, consider some polynomial q_j in step 4. By the preconditions all eigenvalues of A and hence, all roots of q_j are real. Note that there cannot be any eigenvalue with norm larger than 1 as this would violate the assumption on the growth rate of A. Moreover, also the values ± 1 cannot be roots of q_j since the factorization algorithm explicitly tested these values. Hence, all the roots of q_j are indeed in the interval $(-1, 1)$. And as q_j is a result of a square-free factorization, none of the roots occurs multiple times. Hence, the number of different real roots in $(-1, 1)$ must be exactly the degree d_j of q_j.

We also show that the algorithm is complete, if the dimension of A and the degree of the growth rate differ by at most 3, and the dimension is at least 4.

For a dimension of 5 or above this can be argued as follows. Let $A \in C^{n \times n}$, $n \geq 5$, and let the growth rate of A^k be $\Theta(k^N)$ for $N \geq n - 3$. Then there exist some eigenvalue λ whose Jordan block has size $N + 1$. The size of this block is at least $n - 2$ and since $n \geq 5$, the multiplicity of λ is strictly larger than $\frac{n}{2}$. Hence, λ is the unique eigenvalue with such a high multiplicity and is thus extracted by the square-free factorization algorithm. Hence, the remaining polynomial is at most quadratic and can also be factored. But then all eigenvalues are determined and as argued above, then the algorithm is precise.

The argumentation for dimension 4 works along the same line, where as a new case it can happen that the square-free factorization produces two eigenvalues with multiplicity 2. But then the result are two quadratic polynomials which both can be factored.

We end this section by a short example where the integration of Sturm's method is essential.

EXAMPLE 4. *Consider*

$$A = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{5} \\ 0 & \frac{1}{10} & \frac{1}{7} & \frac{1}{8} \\ 0 & \frac{1}{5} & \frac{1}{3} & \frac{1}{5} \\ 0 & 1 & 0 & \frac{1}{3} \end{bmatrix}$$

We prove a constant bound on A^k, i. e., we choose $N = 0$. First, the characteristic polynomial is computed and factored into

$$(x-1)\left(x^3 - \frac{23}{30}x^2 + \frac{61}{2520}x + \frac{29}{2520}\right)$$

where no roots of the cubic polynomial could be inferred. Step 3 clearly succeeds since the multiplicity of the eigenvalue 1 is a bound on the size of the Jordan blocks for that eigenvalue. Finally, step 4 succeeds since all roots of the cubic polynomial are real values in the range $(-1, 1)$. Note that although Mathematica is able to symbolically compute with cubic roots, it is not able to figure out that all roots are real. For instance the largest one is

$$\lambda = \frac{1}{90}\left(23 + \frac{(1120637 + 135\mathrm{i}\sqrt{141178289})^{1/3}}{14^{\frac{2}{3}}} + \frac{6491}{(15688918 + 1890\mathrm{i}\sqrt{141178289})^{\frac{1}{3}}}\right).$$

and the numerical approximation of λ yields an imaginary part of around 10^{-16}. Furthermore, Mathematica cannot symbolically simplify the expression $\mathsf{Im}(\lambda)$ to 0 where Im denotes the imaginary part of a complex number.

10. Summary

We evaluate the significance of our contributions w. r. t. the power of CeTA via experiments with different complexity tools.

The results are shown in Table 1. It does not contain detailed statistics on the number of accepted proofs; actually, all the proofs have been successfully validated.

Each row indicates the number of benchmarks each tool/strategy claimed to be in the complexity class. For each tool we test two strategies: the first strategy (CaT-old and TCT-old) restricts to the technique [16] which was already supported by the old version of CeTA, and the second strategy (CaT-new and TCT-new) allows the tool's best to infer sharp complexity bounds, involving non-triangular matrices (in case of CaT) and multiplicity analysis (in both tools). In both strategies the base technique is restricted to matrix interpretations.

In addition, for each strategy we also test the best of CeTA ("++"): we reduce the claimed complexity bound as long as CeTA accepts, without changing the proof. The last two columns summarizes the result: column "Old" corresponds to the union of CaT-old and TCT-old, whereas "New++" represents the union of all tools and strategies.

Only looking at the last two columns one can clearly see the improvements gained by our developments: We could more than double the number of certified complexity proofs for a linear bound.

Interestingly, even for the new strategies (CaT-new and TCT-new), the concluded complexity bound could be optimized with CeTA as post-processor. For instance, consider the benchmark `Transformed_CSR_04/Ex4_7_77_Bor03_iGM`. On this example, all tools and strategies could conclude either $\mathcal{O}(k^4)$ or $\mathcal{O}(k^5)$, but post-processing revealed that from any of their proofs one can actually infer $\mathcal{O}(k^3)$. Sometimes the post-processing also reduces the degree by two; for instance, on benchmark `ICFP_2010/96086` the best bound claimed by the tools was $\mathcal{O}(k^4)$ whereas post-processing yields $\mathcal{O}(k^2)$.

Finally, also the total number of certifiable complexity proofs could be improved, since some of the benchmarks require matrix interpretations which are not upper triangular, e. g., `Mixed_SRS/3`.

More details on the experiments are available online.[9] The website also contains experiments with AProVE; however, the results for AProVE are not so interesting for this paper, since it supports only the basic result [16] that is already completely covered by the old version of CeTA.

Although the experiments are quite satisfying, there is ample opportunity for future work. For instance, one can formalize and

[9] `http://cl-informatik.uibk.ac.at/software/ceta/experiments/jnf/`.

	CaT-old	++	CaT-new	++	TCT-old	++	TCT-new	++	Old	New++
$\mathcal{O}(1)$	0	0	1	1	0	1	1	1	0	1
$\mathcal{O}(k)$	51	83	99	105	46	68	68	68	51	106
$\mathcal{O}(k^2)$	178	202	212	215	160	173	173	174	178	215
$\mathcal{O}(k^3)$	204	219	223	227	177	185	185	186	206	227
$\mathcal{O}(k^4)$	224	224	232	232	184	186	187	187	224	232
$\mathcal{O}(k^5)$	224	224	232	232	186	186	187	187	224	232

Table 1. Experiments with CaT and TCT

integrate the Perron-Frobenius theorem for non-negative real matrices. This would allow to develop a decision procedure for the property "spectral radius ≤ 1", and hence one could improve Algorithm 4 in the case that the eigenvalues cannot be completely determined. Moreover, one can switch from the spectral radius to the *joint spectral radius*, which investigates not only the power A^k of a single matrix, but arbitrary products of k matrices in a given set. This approach is implemented in both CaT and TCT and results in even more precise bounds for matrix interpretations, as illustrated in [15]. Finally, this work would profit from a full formalization of the algebraic complex numbers in Isabelle; then one can always compute the eigenvalues precisely and as a consequence have a decision procedure for the property $A^k \in \mathcal{O}(k^N)$, provided that the values in A are algebraic.

Acknowledgments

We would like to thank Michael Schaper and Harald Zankl for their help on configuring TCT and CaT, and the anonymous referees for their helpful feedback. This research was supported by the Austrian Science Fund (FWF) project Y757.

References

[1] M. Avanzini and G. Moser. Tyrolean Complexity Tool: Features and usage. In *Proc. RTA 2013*, volume 21 of *LIPIcs*, pages 71–80, 2013.

[2] M. Avanzini, C. Sternagel, and R. Thiemann. Certification of complexity proofs using CeTA. In *Proc. RTA 2015*, LIPIcs 36, pages 23–39, 2015.

[3] C. Ballarin. Reading an algebra textbook. In *Proc. CICM Workshops 2013*, volume 1010 of *CEUR Workshop Proceedings*, 2013.

[4] J. Divasón and J. Aransay. Formalization and execution of linear algebra: from theorems to algorithms. In *Proc. LOPSTR 2013*, volume 8901 of *LNCS*, 2013.

[5] J. Divasón and J. Aransay. Gauss-Jordan algorithm and its applications. *Archive of Formal Proofs*, Sept. 2014. URL `http://afp.sf.net/entries/Gauss_Jordan.shtml`.

[6] J. Divasón and J. Aransay. QR decomposition. *Archive of Formal Proofs*, 2015. URL `http://afp.sourceforge.net/entries/QR_Decomposition.shtml`.

[7] M. Eberl. A decision procedure for univariate real polynomials in Isabelle/HOL. In *Proc. CPP 2015*, pages 75–83. ACM, 2015.

[8] J. Endrullis, J. Waldmann, and H. Zantema. Matrix interpretations for proving termination of term rewriting. *Journal of Automated Reasoning*, 40(2-3):195–220, 2008.

[9] J. Giesl, M. Brockschmidt, F. Emmes, F. Frohn, C. Fuhs, C. Otto, M. Plücker, P. Schneider-Kamp, T. Ströder, S. Swiderski, and R. Thiemann. Proving termination of programs automatically with AProVE. In *Proc. IJCAR 2014*, volume 8562 of *LNCS*, pages 184–191, 2014.

[10] J. Giesl, F. Mesnard, A. Rubio, R. Thiemann, and J. Waldmann. Termination competition (termCOMP 2015). In *Proc. CADE-25*, volume 9195 of *LNCS*, pages 105–108, 2015.

[11] J. Harrison. The HOL light theory of Euclidean space. *J. Autom. Reasoning*, 50(2):173–190, 2013.

[12] B. Huffman and O. Kuncar. Lifting and transfer: A modular design for quotients in Isabelle/HOL. In *Proc. CPP 2013*, volume 8307 of *LNCS*, pages 131–146, 2013.

[13] C. Jordan. *Traité des substitutions et des équations algébriques*. Gauthier-Villars, 1870.

[14] H. Lee. Vector spaces. *Archive of Formal Proofs*, 2014. URL `http://afp.sourceforge.net/entries/VectorSpace.shtml`.

[15] A. Middeldorp, G. Moser, F. Neurauter, J. Waldmann, and H. Zankl. Joint spectral radius theory for automated complexity analysis of rewrite systems. In *Proc. CAI 2011*, volume 6742 of *LNCS*, pages 1–20, 2011.

[16] G. Moser, A. Schnabl, and J. Waldmann. Complexity analysis of term rewriting based on matrix and context dependent interpretations. In *Proc. FSTTCS 2008*, volume 2 of *LIPIcs*, pages 304–315, 2008.

[17] F. Neurauter, H. Zankl, and A. Middeldorp. Revisiting matrix interpretations for polynomial derivational complexity of term rewriting. In *Proc. LPAR-17*, volume 6397 of *LNCS*, pages 550–564, 2010.

[18] T. Nipkow, L. Paulson, and M. Wenzel. *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. 2002.

[19] R. Piziak and P. L. Odell. *Matrix theory: from generalized inverses to Jordan form*. CRC Press, 2007.

[20] C. Sternagel and R. Thiemann. Executable matrix operations on matrices of arbitrary dimensions. *Archive of Formal Proofs*, June 2010. URL `http://afp.sf.net/entries/Matrix.shtml`.

[21] C. Sternagel and R. Thiemann. Formalizing monotone algebras for certification of termination and complexity proofs. In *Proc. RTA-TLCA 2014*, volume 8560 of *LNCS (ARCoSS)*, pages 441–455, 2014.

[22] V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13:354–356, 1969.

[23] J. C. F. Sturm. Mémoire sur la résolution des équations numériques. *Bulletin des Sciences de Férussac*, 11:419–425, 1829.

[24] R. Thiemann. Implementing field extensions of the form $\mathbb{Q}[\sqrt{b}]$. *Archive of Formal Proofs*, Feb. 2014. URL `http://afp.sf.net/entries/Real_Impl.shtml`.

[25] R. Thiemann and C. Sternagel. Certification of termination proofs using CeTA. In *Proc. TPHOLs'09*, volume 5674 of *LNCS*, pages 452–468, 2009.

[26] R. Thiemann and A. Yamada. Matrices, Jordan normal forms, and spectral radius theory. *Archive of Formal Proofs*, Aug. 2015. URL `http://afp.sf.net/entries/Jordan_Normal_Form.shtml`.

[27] J. Waldmann. Polynomially Bounded Matrix Interpretations. In *Proc. RTA 2010*, volume 6 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 357–372, Dagstuhl, Germany, 2010.

[28] D. Yun. On square-free decomposition algorithms. In *Proc. the third ACM symposium on Symbolic and Algebraic Computation*, pages 26–35, 1976.

[29] H. Zankl and M. Korp. Modular complexity analysis for term rewriting. *Logical Methods in Computer Science*, 10(1:19):1–34, 2014.