Università di Firenze, Università di Perugia, INdAM consorziate nel CIAFM

DOTTORATO DI RICERCA
IN MATEMATICA, INFORMATICA, STATISTICA

CURRICULUM IN INFORMATICA
CICLO XXXI

Sede amministrativa Università degli Studi di Firenze
Coordinatore Prof. Graziano Gentili

An algebraic and combinatorial study of some infinite sequences of numbers supported by symbolic and logic computation

Settore Scientifico Disciplinare INF/01

Dottorando
Massimo Nocentini

Tutore
Prof. Donatella Merlini

Coordinatore
Prof. Graziano Gentili

Anni 2015/2018

La legge del Signore è perfetta, rinfranca l'anima;
    la testimonianza del Signore è stabile,
    rende saggio il semplice.
I precetti del Signore sono retti, fanno gioire il cuore;
    il comando del Signore è limpido,
    illumina gli occhi.
Il timore del Signore è puro, rimane per sempre;
    i giudizi del Signore sono fedeli,
    sono tutti giusti.
Ti siano gradite le parole della mia bocca;
    davanti a te i pensieri del mio cuore,
    Signore, mia roccia e mio redentore.

— Sal 18

Anyone who cannot understand that a useful science can be built on stunt hacking will not understand this book, either.

— Manul Laphroaig

I spent many hours trying to do things that were well beyond the intention of the kit designer.

— Jan Addis

Per vedere il sottile cuore delle cose liberati dei nomi, dei concetti, delle aspettative, delle ambizioni e delle differenze.

— Lao Tzu

La prima regola che mi ha insegnato il mio Maestro Yoshi era: "Coltiva sempre il giusto pensiero. Solo così potrai conquistare il dono della forza della conoscenza e della pace". Ho tentato di insegnarti a dominare la rabbia, Raffaello ...Ma non è ancora sufficiente. La rabbia ottenebra la mente, e se non è completamente controllata è un nemico invisibile. Tu sei l'unico fra tutti i tuoi fratelli che è in grado di affrontare questo terribile nemico da solo ... Ma mentre lo combatti e lo fronteggi, non dimenticarti di loro, e sopratutto non dimenticarti di me... Io sono qui, figliolo.

— Maestro Splinter

Abstract

The subject of the thesis concerns the study of infinite sequences, in one or two dimensions, supporting the theoretical aspects with systems for symbolic and logic computation. In particular, in the thesis some sequences related to Riordan arrays are examined from both an algebraic and combinatorial points of view and also by using approaches usually applied in numerical analysis.

Another part concerns sequences that enumerate particular combinatorial objects, such as trees, polyominoes, and lattice paths, generated by symbolic and certified computations; moreover, tiling problems and backtracking techniques are studied in depth and enumeration of recursive structures are also given.

We propose a preliminary suite of tools to interact with the Online Encyclopedia of Integer Sequences, providing a crawling facility to download sequences recursively according to their cross references, pretty-printing them and, finally, drawing graphs representing their connections.

In the context of automatic proof derivation, an extension to an automatic theorem prover is proposed to support the relational programming paradigm. This allows us to encode facts about combinatorial objects and to enumerate the corresponding languages by producing certified theorems at the same time.

As a concrete illustration, we provide many chunks of code written using functional programming languages; our focus is to support theoretical derivations using sound, clear and elegant implementations to check their validity.

Contents

List of Tables

Introduction

Combinatorics. Logic. Programming. This dissertation is an attempt to explore how each entity relates to the others. Seen as a set of rules that characterize our methodology, we apply and use them to study the topic of *infinite sequences.* Our approach is two-fold, it allows us, first, to prove theoretical results about them; second, to built an orbit around them composed of side-track techniques that helps our main goal, such as programming techniques and logical reasoning.

In particular, the research field of interest concerns (i) the manipulation of a set of matrices that can be algebraically defined, which deserve interest both as standalone objects and as tools to study other combinatorial structures; (ii) the *practice of programming* that exposes our way of thinking to its paradigms, functional and relational in particular, and (iii) the rigor and power of *mechanized logic.*

In normal conditions, it is hard to tackle a problem in our context with both entities present at the same time; however, we sacrifice a direct approach to solve the given questions to get the most out of the process that uses the three tenets together. In this philosophy the constant delay, needed to sharpen our knowledge in each individual field, is balanced by the discovery of relations among apparently unrelated subjects that when mixed together yield nicer, more elegant and possibly unexpected solutions.

For this reason we spread our focus over many topics of mathematics and computer science instead of composing a mono-theme discussion; pairwise, we deepen into

Combinatorics and Programming the implementation of enumeration techniques for classes of combinatorial objects from both the algebraic and applicative points of view;

Programming and Logic the study of a family of languages designed for relational programming, using a general purpose inference engine to perform deductions over domain specific objects;

Logic and Combinatorics the formalization of proofs to which corresponds certified enumerations of classes of objects, using an extended theorem prover based on Higher Order

Logic.

Even though abstract reasoning took place most of the time, we make some room for practical stuff; in particular, we would like to provide a suite of tools that helps interacting with the Online Encyclopedia of Integer Sequences, to automate fetching, printing and graphing the networks they compose. Moreover, some practice with bitmasking and backtracking techniques is done to show some interesting tiling and enumeration problems.

Another force that drove our work is a pedagogical approach to problem solving, hence we preferred to refine a first, naive solution over and over to cut away unnecessary details and complexities; moreover, we don't seek for efficiency in all cases in favor of simple and beauty definitions. This methodology allows us to use many programming languages and environments to support theoretical derivations and this dissertation collects this heterogeneous pool of techniques.

Structure

This dissertation has 5 main chapters that rely on the first one, which introduces basic definitions needed by the following ones; so, except for the first, the others can be read in any order,

- Chapter 1 quickly recalls the theory characterizing the Riordan group and provides a set of constructors to define those matrices programmatically, after the introduction of the symbolic module Sympy implemented on top of the Python programming language. Moreover, it shows our programming style which is based on consecutive manipulation of symbolic equations, as mush as possible close to paper and pencil derivations.

- Chapter 2 presents a theoretical and practical framework that lifts a scalar function to a matrix function, toward application to Riordan matrices. To the best of our knowledge, this is the first implementation that allows the user to perform its computations fully symbolically, postponing to the end the substitution of ground values to fill matrices with numbers. Moreover, the Jordan Canonical Form of Riordan matrices is also studied.

- Chapter 3 studies the enumeration of languages of binary words avoiding a given pattern, provided that some contraints over the structure of each word are taken into account. In particular, when the pattern to avoid is a Riordan pattern then the problem can be solved using Riordan arrays, deriving new series developments about enumerations with respect to the number of 1-bits and to the length. Finally, some combinatorial interpretations are shown, at least for simpler families of languages.

- Chapter 4 implements a suite of tools that interact with the Online Encyclopedia of Integer Sequences; in particular, a (i) crawler fetches sequences and their cross references recursively, using asynchronous primitive to optimize network delays, a (ii) (pretty) printer for the fetched sequences that allows the user to filter the sections to be rendered both in the terminal and in web interfaces and, finally, a (iii) grapher that draw graphs where vertices are sequences and edges are references among them.
- Chapter 5 is an exercise in backtracking techniques to solve tiling and placement problems; for the sake of efficiency, bit masking manipulation and encodings are used. Moreover, it provides an implementation of a classic enumeration methodology that allows us to clearly show generations of combinatorial objects starting from concise and recursive symbolic definitions; finally, counting all of them is a check of the correctness of our implementation.
- Chapter 6 proposes an extension to the tactic mechanism actually present in the HOL Light theorem prover inspired by the relational paradigm. This prototype makes explicit use of meta-variables to support substitutions and allows backtracking facilities during the interactive proof process; nonetheless, the proposed generalization offers compositionality just as the current system does. So this extension is stressed against an evaluator for a subset of the Lisp language to find a *quine* program.

Moreover, a final paragraph concludes with a quick summary of our main results and the bibliography ends this dissertation.

Typographical and typesetting conventions

This dissertation was typeset using the LaTeX 's style `tufte-book`, which splits each page into two columns: the left-most is greater in width and holds the main body, while the right-most is lesser and holds captions, contextual data and code comments; in particular, code chunks are printed in `verbatim` and highlighted wherever possible, while outputs that denote mathematical objects are printed in math style, as usual – when their size overflows over the right column we prefer to allow it to take the entire page width for the sake of clarity.

Acknowledgments

First, I want to thank my family who always supported me, I dedicate my dissertation to my father and my mother, with love and admiration; I also thank Angela, I can't imagine a better sister.

Second, I want to thank my advisor Donatella Merlini and friends (Professors also) Giovanni Maria Marchetti and Marco Maggesi.

Third, I want to thank Professors Paul Barry and Neil J. Sloane who read carefully and proposed constructive criticism and comments to make this work better.

I remember and keep within me my friends and those people who make and have made my life beautiful; *Glazie a tutti.*

for Angela

1

Backgrounds

In this introductory section we review theoretical concepts about the *Riordan group* that will be useful in subsequent chapters; additionally, we provide a very short introduction to symbolic computation using the `sympy` module implemented using the Python programming language, giving a taste of our programming style.

1.1 Riordan Arrays, formally

A *Riordan array* is an infinite lower triangular array $(d_{n,k})_{n,k\in\mathbb{N}}$, defined by a pair of formal power series $(d(t), h(t))$ such that $d(0) \neq 0$, $h(0) = 0$ and $h'(0) \neq 0$; furthermore, each element

$$d_{n,k} = [t^n]d(t)h(t)^k, \qquad n, k \geq 0,$$

is the coefficient of monomial t^n in the series expansion of $d(t)h(t)^k$ and the bivariate generating function

$$R(t, w) = \sum_{n,k\in\mathbb{N}} d_{n,k}t^n w^k = \frac{d(t)}{1 - wh(t)} \tag{1.1}$$

enumerates the sequence $(d_{n,k})_{n,k\in\mathbb{N}}$.

These arrays were introduced in [Shapiro et al., 1991], with the aim of defining a class of infinite, lower triangular arrays and since then they have attracted, and continue to attract, a lot of attention in the literature and recent applications can be found in [Luzon et al., 2014].

Example 1. *The most simple Riordan matrix could be the Pascal triangle*

$$\mathcal{R}\left(\frac{1}{1-t}, \frac{t}{1-t}\right) \quad \textit{where} \quad d_{n,k} = \binom{n}{k};$$

moreover, another remarkable matrix is the Catalan triangle

$$\mathcal{R}\left(\frac{1-\sqrt{1-4t}}{2t}, \frac{1-\sqrt{1-4t}}{2t}\right)$$

where $d_{n,k} = \binom{2n-k}{n-k} - \binom{2n-k}{n-k-1}$.

An important property of Riordan arrays concerns the computation of combinatorial sums; precisely, it is encoded by the identity

$$\sum_{k=0}^{n} d_{n,k} f_k = [t^n] d(t) f(h(t)) \tag{1.2}$$

and it is extensively commented in [Luzon et al., 2012, Merlini et al., 2009, Sprugnoli, 1994]. It states that every combinatorial sum involving a Riordan array can be computed by extracting the coefficient of t^n from the series expansion of $d(t)f(h(t))$, where $f(t) = \mathcal{G}(f_k) = \sum_{k\geq 0} f_k t^k$ denotes the generating function of the sequence $(f_k)_{k\in\mathbb{N}}$ and the symbol $\mathcal{G}$ denotes the generating function operator. Due to its importance, relation (1.2) is commonly known as the *fundamental rule* of Riordan arrays. For short and when no confusion arises, the notation $(f_k)_k$ will be used as an abbreviation of $(f_k)_{k\in\mathbb{N}}$.

As it is well-known (see, e.g., [Luzon et al., 2014, Merlini et al., 1997, Shapiro et al., 1991]), Riordan arrays constitute an *algebraic group* with respect to the usual row-by-column product between matrices; formally, the group operation $\cdot$ applied to two Riordan arrays $D_1(d_1(t),\ h_1(t))$ and $D_2(d_2(t),\ h_2(t))$ is carried out as

$$D_1 \cdot D_2 = (d_1(t) d_2(h_1(t)),\ h_2(h_1(t))). \tag{1.3}$$

Moreover, the Riordan array $I = (1,\ t)$ acts as the identity element and the inverse of $D = (d(t), h(t))$ is the Riordan array

$$D^{-1} = \left(\frac{1}{d(\overline{h}(t))}, \overline{h}(t)\right) \tag{1.4}$$

where $\overline{h}(t)$ denotes the compositional inverse of $h(t)$.

An equivalent characterization of a each matrix $\mathcal{R}(d(t), h(t))$ in the Riordan group is given by two sequences of coefficients $(a_n)_{n\in\mathbb{N}}$ and $(z_n)_{n\in\mathbb{N}}$ called A-sequence and Z-sequence, respectively. The former one can be used to define every coefficient $d_{n,k}$ with $k > 0$,

$$d_{n+1,k+1} = a_0 d_{n,k} + a_1 d_{n,k+1} + a_2 d_{n,k+2} + \ldots + a_j d_{n,k+j} + \ldots$$

where the sum is finite because exists $j \in \mathbb{N}$ such that $n = k + j$; on the other hand, the latter one can be used to define every coefficient $d_{n,0}$ lying on the first column,

$$d_{n+1,0} = z_0 d_{n,0} + z_1 d_{n,1} + z_2 d_{n,2} + \ldots + z_n d_{n,n}$$

where the sum is finite because $d_{n,k+j} = 0$ for $j > n - k$.

Moreover, let $A(t)$ and $Z(t)$ be the generating functions of the A-sequence and Z-sequence respectively, then relations

$$h(t) = tA(h(t)) \quad \text{and} \quad d(t) = \frac{d_{0,0}}{1 - tZ(h(t))}$$

connect them with functions $d(t)$ and $h(t)$, where $d_{0,0}$ is the very first element of $\mathcal{R}$; for the sake of completeness, [Merlini et al., 1997] collects more alternative characterizations.

1.2 Symbolic computation

The main part of the symbolic computations supporting the topics discussed in this dissertation has been coded using the Python language, relying on the module `sympy` for what concerns mathematical stuff. Quoting from `http://www.sympy.org/`:

"SymPy is a Python library for symbolic mathematics. It aims to become a full-featured computer algebra system (CAS) while keeping the code as simple as possible in order to be comprehensible and easily extensible."

The paper [Meurer et al., 2017] explains it and many other resources can be found online; for example, a comprehensive documentation is [SymPy, a] and a well written, understandable tutorial [SymPy, b] is provided by the original development team.

Here we avoid to duplicate the tutorial with similar examples, instead we state the methodology used while coding our definitions. Python is a very expressive language allowing programmers to use both the object-oriented and functional paradigms. It can tackle different application domains by means of *modules* and `sympy` is an example that targets the manipulation of symbolic terms. Contrary to other proprietary software like Maple and Mathematica which ship their own languages, `sympy` is implemented entirely in Python, allowing a transparent and easy integration in other Python programs, as we will see in later chapters.

The main point to grasp in our opinion is the difference between the *meta language*, which is Python, and the *object language*, which is the mathematical expressions denoted by `sympy` objects.

Example 2. `Symbol` *is a fundamental class of objects that introduces arbitrary mathematical symbols.*

```
>>> from sympy import Symbol
>>> a_sym = Symbol('a')
>>> a_sym
a
```

The previous snippet allows us to clarify the duality among meta and object languages; precisely, the mathematical expression a *is denoted by the Python object* `a_sym`*.*

The above example is the first one found by the reader and it shows a common pattern used through this document to illustrate *computations*; in particular, when a line starts with (i) >>> then it is an *input* line holding code to be executed, (ii) `...` then it is a *continuation* line holding an unfinished code expression, otherwise (iii) it is an *output* line reporting the result of the evaluation.

A second fundamental methodology that we embrace in our symbolic manipulations is *equational reasoning*, namely we use equations denoted by `Eq` objects to express identities to reason about, used both to define things and to be solved with respect to a desired symbol.

Example 3. *Introduction of* `Eq`, `solve` *and* `symbols` *functions:*

```
>>> from sympy import Eq, solve, symbols
>>> a, t = symbols('a t')
>>> a_def = Eq(a, 3)
>>> at_eq = Eq(a+5*t, 1/(1-t))
>>> a_def, at_eq
```

$$\left(a = 3, \quad a + 5t = \frac{1}{-t+1}\right)$$

```
>>> sols = [Eq(t, s) for s in solve(at_eq, t)]
>>> sols
```

$$\left[t = -\frac{a}{10} - \frac{1}{10}\sqrt{a^2 + 10a + 5} + \frac{1}{2},\right.$$
$$\left. t = -\frac{a}{10} + \frac{1}{10}\sqrt{a^2 + 10a + 5} + \frac{1}{2}\right]$$

Due to the importance of equations in our code, we introduce two helper functions. First, `define` builds a definition:

```
def define(let, be, ctor=Eq, **kwds,):

    if 'evaluate' not in kwds:       # If `evaluate` is already given, use it # as it is,
        kwds['evaluate'] = False     # otherwise set to `False` to preevent evaluation
                                     # by `Eq`, which implicitly do simplifications;
    return ctor(let, be, **kwds)     # finally, return an equation object.
```

Example 4. *Introduction of* `Function` *objects:*

```
>>> from sympy import Function, sqrt
>>> f = Function('f')
>>> f(3)
```

```
f(3)
>>> t = symbols('t')
>>> define(let=f(t), be=(1-sqrt(1-4*t))/(2*t), ctor=FEq)
```

$$f(t) = \frac{1}{2t}\left(-\sqrt{-4t+1}+1\right)$$

The keyword argument `ctor=FEq` asks `define` to promote the equation we are defining as a `callable` object by means of the `FEq` class,

```
class FEq(Eq):

    def __call__(self, *args, **kwds):
        with lift_to_Lambda(self, **kwds) as feq:
            applied = feq(*args)
            if isinstance(applied, Eq):
                subs = Subs(applied, self.lhs, self.rhs)
                setattr(subs, 'substitution', partial(
                    lambda inner_self, *args, **kwds: self, subs))
                return subs
            else:
                return applied

    def swap(self):
        return self.__class__(self.rhs, self.lhs)

    def __iter__(self):
        yield self.lhs, self.rhs

    def as_substitution(self):
        return dict(self)

    def __mod__(self, arg):

        def S(term):
            return term.subs(self, simultaneous=True)

        if isinstance(arg, Eq):
            lhs = self.lhs if self.lhs == arg.lhs else S(arg.lhs)
            rhs = S(arg.rhs)
            return arg.__class__(lhs, rhs, evaluate=False)
        else:
            return S(arg)
```

which provides methods that allows it to be used as a substitution too; in turn, it depends on

```
@contextmanager
def lift_to_Lambda(eq, return_eq=False, lhs_handler=lambda args: []):

    lhs = eq.lhs
    args = (lhs.args[1:] if isinstance(lhs, Indexed) else    # get arguments wrt the type of `lhs` object;
            lhs.args     if isinstance(lhs, Function) else   # here we handle both function and subscript
            lhs_handler(lhs))                                # notations. Finally, `Lambda` is the
    yield Lambda(args, eq if return_eq else eq.rhs)          # class of callable objects in SymPy.
```

Example 5. *Introduction of* `IndexedBase` *objects:*

```
>>> from commons import lift_to_Lambda
>>> from sympy import IndexedBase
>>> a = IndexedBase('a')
>>> aeq = Eq(a[n], n+a[n-1])
>>> aeq(n+1)
```

$$a_{n+1} = n + a_n + 1$$

```
>>> b = Function('b')
>>> beq = Eq(b(n), n+b(n-1))
>>> beq(n+1)
```

$$b(n+1) = n + b(n) + 1$$

1.3 Riordan Arrays, computationally

In this section we describe a little framework that implements parts of the concepts seen in the previous section; in particular, we provide some strategies to build Riordan arrays, to find corresponding production matrices and their group inverse elements, respectively.

First of all we introduce (i) *function symbols* `d_fn` and `h_fn` to denote arbitrary symbolic functions d and h, (ii) *indexed symbols* `d` and `h` to denote arbitrary symbolic and indexed coefficients

```
>>> d_fn, h_fn = Function('d'), Function('h')
>>> d, h = IndexedBase('d'), IndexedBase('h')
```

respectively. To build Riordan matrices we use `Matrix` objects; in particular, the expression `Matrix(r, c, ctor)` denotes a matrix with `r` rows and `c` columns where each coefficient $d_{n,k}$ in the matrix is defined according to `ctor` which is a *callable* object consuming two arguments n and k, its row and column coordinates. We call it `ctor` as abbreviation for *constructor*, because it allows us to code the definition of each coefficient with a Python callable object.

From the official doc at `https://docs.python.org/3/library/functions.html#callable`: `callable(object)` return `True` if the `object` argument appears callable, `False` if not. If this returns true, it is still possible that a call fails, but if it is false, calling object will never succeed. Note that classes are callable (calling a class returns a new instance); instances are callable if their class has a `__call__` method.

Here we show how to build a pure symbolic matrix:

```
>>> from sympy import Matrix
>>> rows, cols = 5, 5
>>> ctor = lambda i,j: d[i,j]
>>> Matrix(rows, cols, ctor)
```

$$\begin{bmatrix} d_{0,0} & d_{0,1} & d_{0,2} & d_{0,3} & d_{0,4} \\ d_{1,0} & d_{1,1} & d_{1,2} & d_{1,3} & d_{1,4} \\ d_{2,0} & d_{2,1} & d_{2,2} & d_{2,3} & d_{2,4} \\ d_{3,0} & d_{3,1} & d_{3,2} & d_{3,3} & d_{3,4} \\ d_{4,0} & d_{4,1} & d_{4,2} & d_{4,3} & d_{4,4} \end{bmatrix}$$

In the following sections we show a collection of such `ctor`s, each one of them implements one theoretical characterization used to denote Riordan arrays and corresponding examples are given.

1.3.1 Convolution ctor

The following definition implements a ctor that allows us to build Riordan arrays by convolution of their d and h functions; here it is,

```
def riordan_matrix_by_convolution(dim, d, h):

    t = symbols('t')                                    # Local symbol to denote a formal variable `t`.

    with lift_to_Lambda(d, return_eq=True) as D:        # Lift equations `d` and `h` to become callables
      with lift_to_Lambda(h, return_eq=True) as H:      # objects, returning equations as well, in order
        d_eq, h_eq = D(t), H(t)                         # to let both of them depend on variable `t`.

    @lru_cache(maxsize=None)
    def column(j):                                      # Columns are memoized for the sake of efficiency.
        if not j: return d_eq                           # Base case.
        lhs = column(j-1).lhs * h_eq.lhs                # Otherwise, use already computed column to build
        rhs = column(j-1).rhs * h_eq.rhs                # the current `lhs` and `rhs`.
        return Eq(lhs, rhs)

    @lru_cache(maxsize=None)
    def C(j):                                           # Local function that performs Taylor expansion
        return column(j).rhs.series(t, n=dim).removeO() # of columns, which are symbolic terms up to now.

    return lambda i, j: C(j).coeff(t, i).expand()       # Return a lambda to be plugged in a `Matrix` ctor.
```

Example 6. *Symbolic Riordan array built by two polynomials with symbolic coefficients:*

```
>>> d_series = Eq(d_fn(t), 1+sum(d[i]*t**i for i in range(1,m)))
>>> h_series = Eq(h_fn(t), t*(1+sum(h[i]*t**i for i in range(1,m-1)))).expand()
>>> d_series, h_series
```

$$\left(d(t) = t^4 d_4 + t^3 d_3 + t^2 d_2 + t d_1 + 1, \quad h(t) = t^4 h_3 + t^3 h_2 + t^2 h_1 + t\right)$$

```
>>> R = Matrix(m, m, riordan_matrix_by_convolution(m, d_series, h_series))
>>> R
```

$$\begin{bmatrix} 1 & & & & \\ d_1 & 1 & & & \\ d_2 & d_1 + h_1 & 1 & & \\ d_3 & d_1 h_1 + d_2 + h_2 & d_1 + 2h_1 & 1 & \\ d_4 & d_1 h_2 + d_2 h_1 + d_3 + h_3 & 2d_1 h_1 + d_2 + h_1^2 + 2h_2 & d_1 + 3h_1 & 1 \end{bmatrix}$$

Example 7. *The Pascal triangle built using closed generating functions:*

```
>>> d_series = Eq(d_fn(t), 1/(1-t))
>>> h_series = Eq(h_fn(t), t*d_series.rhs)
>>> d_series, h_series
```

$$\left(d(t) = \frac{1}{1-t}, \quad h(t) = \frac{t}{1-t}\right)$$

```
>>> R = Matrix(10, 10, riordan_matrix_by_convolution(10, d_series, h_series))
>>> R
```

$$\begin{bmatrix} 1 &&&&&&&&& \\ 1 & 1 &&&&&&&& \\ 1 & 2 & 1 &&&&&&& \\ 1 & 3 & 3 & 1 &&&&&& \\ 1 & 4 & 6 & 4 & 1 &&&&& \\ 1 & 5 & 10 & 10 & 5 & 1 &&&& \\ 1 & 6 & 15 & 20 & 15 & 6 & 1 &&& \\ 1 & 7 & 21 & 35 & 35 & 21 & 7 & 1 && \\ 1 & 8 & 28 & 56 & 70 & 56 & 28 & 8 & 1 & \\ 1 & 9 & 36 & 84 & 126 & 126 & 84 & 36 & 9 & 1 \end{bmatrix}$$

1.3.2 Recurrence ctor

The following definition implements a ctor that allows us to build Riordan arrays by a recurrence relation over coefficients $d_{n+1,k+1}$; here it is,

```
def riordan_matrix_by_recurrence(dim, rec, init={(0,0):1},
                                 ctor=zeros, post=expand,
                                 lattice=None):

    if not lattice:
        lattice = [(n, k) for n in range(1, dim)    # `lattice` denotes the order in which coeffs in
                          for k in range(n+1)]      # the array will be computed; it can be plugged in.

    R = ctor(dim)                                   # Initial array as base for recursive construction.

    for cell, i in init.items(): R[cell] = i        # Set boundary values for the recurrence `rec`.

    for cell in lattice:                            # Visit cells as they appear in the order `lattice`;
        for comb_cell, v in rec(*cell).items():     # then, get dependencies with respect the each cell.

            try:
                comb = (1 if cell == comb_cell
                        else R[comb_cell])          # If it is possible to access the dependee cell,
                combined = v * comb                 # then perform the combination using the given `v`.
            except IndexError:
                combined = 0                        # Otherwise, take `0` because combination fails.

            R[cell] += combined                     # Finally, accumulate the current combination.

    if callable(post): R = R.applyfunc(post)        # If some post processing is desired, do it.

    return lambda n, k: R[n, k]                     # Return a lambda that uses the computed `R`.
```

Example 8. *Symbolic Riordan Array built according to the recurrence:*

$$\begin{aligned} d_{n+1,0} &= \bar{b}\, d_{n,0} + c\, d_{n,1}, \quad n \in \mathbb{N} \\ d_{n+1,k+1} &= a\, d_{n,k} + b\, d_{n,k} + c\, d_{n,k+1}, \quad n,k \in \mathbb{N} \end{aligned}$$

```
>>> dim = 5
>>> a, b, b_bar, c = symbols(r'a b \bar{b} c')
>>> M = Matrix(dim, dim,
...             riordan_matrix_by_recurrence(
...                dim, lambda n, k: {(n-1, k-1):a,
...                                   (n-1, k): b if k else b_bar,
...                                   (n-1, k+1):c}))
>>> M
```

$$\left[\begin{array}{ccccc}1 & & & & \\ \bar{b} & a & & & \\ \bar{b}^2+ac & \bar{b}a+ab & a^2 & & \\ \bar{b}^3+2\bar{b}ac+abc & \bar{b}^2a+\bar{b}ab+2a^2c+ab^2 & \bar{b}a^2+2a^2b & a^3 & \\ \bar{b}^4+3\bar{b}^2ac+2\bar{b}abc+2a^2c^2+ab^2c & \bar{b}^3a+\bar{b}^2ab+3\bar{b}a^2c+\bar{b}ab^2+5a^2bc+ab^3 & \bar{b}^2a^2+2\bar{b}a^2b+3a^3c+3a^2b^2 & \bar{b}a^3+3a^3b & a^4\end{array}\right]$$

```
>>> production_matrix(M)
```

$$\begin{bmatrix}\bar{b} & a & & \\ c & b & a & \\ & c & b & a\\ & & c & b\end{bmatrix}$$

Forcing $a = 1$ *and* $\bar{b} = b$ *yield the easier matrix* `Msubs`

```
>>> Msubs = M.subs({a:1, b_bar:b})
>>> Msubs, production_matrix(Msubs)
```

$$\left(\begin{bmatrix}1 & & & & \\ b & 1 & & & \\ b^2+c & 2b & 1 & & \\ b^3+3bc & 3b^2+2c & 3b & 1 & \\ b^4+6b^2c+2c^2 & 4b^3+8bc & 6b^2+3c & 4b & 1\end{bmatrix},\ \begin{bmatrix}b & 1 & & \\ c & b & 1 & \\ & c & b & 1\\ & & c & b\end{bmatrix}\right)$$

and the correspoding production matrix checks the substitution.

Previous examples uses the function `production_matrix` to compute the *production matrix* [Deutsch et al., 2005, 2009] of a Riordan array, here is its definition with two helper `ctors`:

```
def columns_symmetry(M):
    return lambda i, j: M[i, i-j]

def rows_shift_matrix(by):
    return lambda i, j: 1 if i + by == j else 0

def diagonal_func_matrix(f):
    return lambda i, j: f(i) if i == j else 0

def production_matrix(M, exp=False):

    f = factorial if exp else one
    U = Matrix(M.rows, M.cols, rows_shift_matrix(by=1))
    F = Matrix(M.rows, M.cols, diagonal_func_matrix(f))
    F_inv = F**(-1)
    V = F_inv * U * F
    O = F_inv * M * F
    O_inv = O**(-1)
    PM = F * O_inv * V * O * F_inv
    PM = F_inv * PM * F if exp else PM
    PM = PM[:-1, :-1]
    return PM.applyfunc(simplify)
```

implemented according to [Barry, 2017, page 215].

1.3.3 *A and Z sequences ctor*

The following definition implements a ctor that allows us to build Riordan arrays by their Z and A sequences; here it is,

```
def riordan_matrix_by_AZ_sequences(dim, seqs, init={(0,0):1},
                                   ctor=zeros, post=expand,
                                   lattice=None):

    if not lattice: lattice = [(n, k) for n in range(1, dim)
                                      for k in range(n+1)]

    Zseq, Aseq = seqs
    t = symbols('t')

    with lift_to_Lambda(Zseq) as Z, lift_to_Lambda(Aseq) as A:
        Z_series = Z(t).series(t, n=dim).removeO()
        A_series = A(t).series(t, n=dim).removeO()

    R = ctor(dim)

    for cell, i in init.items(): R[cell] = i

    for n, k in lattice:

        if k:
            v = sum(R[n-1, j] * A_series.coeff(t, n=i)
                    for i, j in enumerate(range(k-1, dim)))
        else:
            v = sum(R[n-1, j] * Z_series.coeff(t, n=j)
                    for j in range(dim))

        R[n, k] = v

    if callable(post): R = R.applyfunc(post)

    return lambda n, k: R[n, k]
```

in words, (i) it deconstructs `seqs` into objs denoting Z and A seqs, promoting both of them as callables, (ii) it introduces the formal variable `t`, (iii) it performs two Taylor expansions with respect to `t`. In order to build the resulting matrix, it visits each cell according to the order given by `lattice` then if the cell lies not on the first column, it combines using `A_series` coefficients up to the cell in position `(k-1, dim-1)`; otherwise, it combine using `Z_series` coefficients up to the cell in position `(k-1, dim-1)`. Finally, it stores the combination and if some post processing is desired, it does so; at last, it returns a lambda that uses the computed matrix `R`.

Example 9. *Again the Pascal triangle built using A and Z sequences*

```
>>> A, Z = Function('A'), Function('Z')
>>> A_eq = Eq(A(t), 1 + t)
>>> Z_eq = Eq(Z(t),1)
>>> A_eq, Z_eq
```

$$\left(A(t) = t + 1, \quad Z(t) = 1\right)$$

```
>>> R = Matrix(10, 10, riordan_matrix_by_AZ_sequences(10, (Z_eq, A_eq)))
>>> R, production_matrix(R)
```

$$\left(\begin{bmatrix}1&&&&&&&&&\\1&1&&&&&&&&\\1&2&1&&&&&&&\\1&3&3&1&&&&&&\\1&4&6&4&1&&&&&\\1&5&10&10&5&1&&&&\\1&6&15&20&15&6&1&&&\\1&7&21&35&35&21&7&1&&\\1&8&28&56&70&56&28&8&1&\\1&9&36&84&126&126&84&36&9&1\end{bmatrix}, \begin{bmatrix}1&1&&&&&&&\\&1&1&&&&&&\\&&1&1&&&&&\\&&&1&1&&&&\\&&&&1&1&&&\\&&&&&1&1&&\\&&&&&&1&1&\\&&&&&&&1&1\\&&&&&&&&1\end{bmatrix}\right)$$

Example 10. *Catalan triangle built using A and Z sequences,*

```
>>> A_ones = Eq(A(t), 1/(1-t)) # A is defined as in the previous example
>>> R = Matrix(10, 10, riordan_matrix_by_AZ_sequences(10, (A_ones, A_ones)))
>>> R, production_matrix(R)
```

$$\left(\begin{bmatrix}1&&&&&&&&&\\1&1&&&&&&&&\\2&2&1&&&&&&&\\5&5&3&1&&&&&&\\14&14&9&4&1&&&&&\\42&42&28&14&5&1&&&&\\132&132&90&48&20&6&1&&&\\429&429&297&165&75&27&7&1&&\\1430&1430&1001&572&275&110&35&8&1&\\4862&4862&3432&2002&1001&429&154&44&9&1\end{bmatrix}, \begin{bmatrix}1&1&&&&&&&\\1&1&1&&&&&&\\1&1&1&1&&&&&\\1&1&1&1&1&&&&\\1&1&1&1&1&1&&&\\1&1&1&1&1&1&1&&\\1&1&1&1&1&1&1&1&\\1&1&1&1&1&1&1&1&1\\1&1&1&1&1&1&1&1&1\end{bmatrix}\right)$$

Example 11. *Symbolic Riordan arrays built using A and Z sequences,*

```
>>> dim = 5
>>> a = IndexedBase('a')
>>> A_gen = Eq(A(t), sum((a[j] if j else 1)*t**j for j in range(dim)))
>>> R = Matrix(dim, dim, riordan_matrix_by_AZ_sequences(dim, (A_gen, A_gen)))
>>> R
```

$$\begin{bmatrix}1&&&&\\1&1&&&\\a_1+1&a_1+1&1&&\\a_1^2+2a_1+a_2+1&a_1^2+2a_1+a_2+1&2a_1+1&1&\\a_1^3+3a_1^2+3a_1a_2+3a_1+2a_2+a_3+1&a_1^3+3a_1^2+3a_1a_2+3a_1+2a_2+a_3+1&3a_1^2+3a_1+2a_2+1&3a_1+1&1\end{bmatrix}$$

```
>>> z = IndexedBase('z')
>>> A_gen = Eq(A(t), sum((a[j] if j else 1)*t**j for j in range(dim)))
>>> Z_gen = Eq(Z(t), sum((z[j] if j else 1)*t**j for j in range(dim)))
>>> Raz = Matrix(dim, dim, riordan_matrix_by_AZ_sequences(dim, (Z_gen, A_gen)))
>>> Raz
```

$$\begin{bmatrix} 1 & & & & \\ 1 & 1 & & & \\ z_1+1 & a_1+1 & 1 & & \\ a_1z_1+2z_1+z_2+1 & a_1^2+a_1+a_2+z_1+1 & 2a_1+1 & 1 & \\ \left(\begin{array}{c} a_1^2z_1+2a_1z_1+2a_1z_2+a_2z_1+ \\ z_1^2+3z_1+2z_2+z_3+1 \end{array}\right) & a_1^3+a_1^2+3a_1a_2+2a_1z_1+a_1+a_2+a_3+2z_1+z_2+1 & 3a_1^2+2a_1+2a_2+z_1+1 & 3a_1+1 & 1 \end{bmatrix}$$

```
>>> production_matrix(R), production_matrix(Raz)
```

$$\left(\begin{bmatrix} 1 & 1 & & \\ a_1 & a_1 & 1 & \\ a_2 & a_2 & a_1 & 1 \\ a_3 & a_3 & a_2 & a_1 \end{bmatrix}, \quad \begin{bmatrix} 1 & 1 & & \\ z_1 & a_1 & 1 & \\ z_2 & a_2 & a_1 & 1 \\ z_3 & a_3 & a_2 & a_1 \end{bmatrix}\right)$$

1.3.4 *Exponential ctor*

The following definition implements a ctor that allows us to build an exponential Riordan array; here it is,

```
def riordan_matrix_exponential(RA):
    return lambda i,j: factorial(i)*RA(i,j)/factorial(j)
```

Example 12. *Build the triangle of Stirling numbers of the II kind:*

```
>>> d_series = Eq(d_fn(t), 1)
>>> h_series = Eq(h_fn(t), exp(t)-1)
>>> d_series, h_series
```

$$\left(d(t) = 1, \quad h(t) = e^t - 1\right)$$

```
>>> R = matrix(10, 10, riordan_matrix_exponential(
...                     riordan_matrix_by_convolution(
...                       10, d_series, h_series)))
>>> R
```

$$\begin{bmatrix} 1 & & & & & & & & & \\ & 1 & & & & & & & & \\ & 1 & 1 & & & & & & & \\ & 1 & 3 & 1 & & & & & & \\ & 1 & 7 & 6 & 1 & & & & & \\ & 1 & 15 & 25 & 10 & 1 & & & & \\ & 1 & 31 & 90 & 65 & 15 & 1 & & & \\ & 1 & 63 & 301 & 350 & 140 & 21 & 1 & & \\ & 1 & 127 & 966 & 1701 & 1050 & 266 & 28 & 1 & \\ & 1 & 255 & 3025 & 7770 & 6951 & 2646 & 462 & 36 & 1 \end{bmatrix}$$

```
>>> production_matrix(R), production_matrix(R, exp=True)
```

$$\left(\begin{bmatrix} 0 & 1 & & & & & & & \\ & 1 & 1 & & & & & & \\ & & 2 & 1 & & & & & \\ & & & 3 & 1 & & & & \\ & & & & 4 & 1 & & & \\ & & & & & 5 & 1 & & \\ & & & & & & 6 & 1 & \\ & & & & & & & 7 & 1 \\ & & & & & & & & 8 \end{bmatrix}, \begin{bmatrix} 0 & 1 & & & & & & & \\ & 1 & 2 & & & & & & \\ & & 2 & 3 & & & & & \\ & & & 3 & 4 & & & & \\ & & & & 4 & 5 & & & \\ & & & & & 5 & 6 & & \\ & & & & & & 6 & 7 & \\ & & & & & & & 7 & 8 \\ & & & & & & & & 8 \end{bmatrix}\right)$$

```
>>> inspect(R)
nature(is_ordinary=False, is_exponential=True)
```

In the above example we introduced another function `inspect` that studies the type of array it consumes. Before reporting its definition we remark that the matrix on the left is an usual production matrix (which tells us that $d_{6,4} = d_{5,3} + 4d_{5,4} = 25 + 4 \cdot 10 = 65$, for example); on the other hand, the matrix on right helps to decide if the array is an exponential one by proving that each diagonal is an *arithmetic progression*, for more on this see [Barry, 2017].

```
def inspect(M):

    P = production_matrix(M, exp=False)                 # Ordinary production matrix.
    C = production_matrix(M, exp=True)                  # "Exponential" production matrix.

    is_ord = all(P[:1-i, 1] == P[i-1:, i]               # A RA is ordinary if columns of its PM
                 for i in range(2, P.cols))             # are shifted but equals.

    diagonals = {d: [C[j+d,j] for j in range(1,C.rows-d)]  # Fetch matrix diagonals for the
                 for d in range(C.rows-3) }             # exponential case.

    is_exp = all(map(is_arithmetic_progression,         # A Riordan array is exponential if `diagonals`
                     diagonals.values()))               # are arithmetic progressions, all of them.

    return nature(is_ord, is_exp)


def is_arithmetic_progression(prog):

    steps = len(prog)-1
    for _ in range(steps):
        prog = [(b-a).simplify()                # Reduce the list by consecutive differences,
                for a, b in zip(prog, prog[1:])]   # till a single object survives.

    assert len(prog) == 1                       # Consistency check of the last comment.

    return prog.pop() == 0                      # Finally, the reduced object has to vanish.
```

Example 13. *In this example we explore an exponential Riordan array starting from the generating functions of the Pascal triangle. Surprisingly the array we get back is known in the OEIS (`https://oeis.org/A021009`) and looking for some comments we quote the observation*

> *"the generalized Riordan array (e^x, x) with respect to the sequence $n!$ is Pascal's triangle A007318"*

by Peter Bala.

```
>>> d_series, h_series = Eq(d_fn(t), 1/(1-t)), Eq(h_fn(t), t/(1-t))
>>> d_series, h_series
```

$$\left(d(t) = \frac{1}{1-t}, \quad h(t) = \frac{t}{1-t}\right)$$

```
>>> R = matrix(10, 10, riordan_matrix_exponential(
...                      riordan_matrix_by_convolution(
...                        10, d_series, h_series)))
>>> R
```

$$\left[\begin{array}{cccccccccc}
1 & & & & & & & & & \\
1 & 1 & & & & & & & & \\
2 & 4 & 1 & & & & & & & \\
6 & 18 & 9 & 1 & & & & & & \\
24 & 96 & 72 & 16 & 1 & & & & & \\
120 & 600 & 600 & 200 & 25 & 1 & & & & \\
720 & 4320 & 5400 & 2400 & 450 & 36 & 1 & & & \\
5040 & 35280 & 52920 & 29400 & 7350 & 882 & 49 & 1 & & \\
40320 & 322560 & 564480 & 376320 & 117600 & 18816 & 1568 & 64 & 1 & \\
362880 & 3265920 & 6531840 & 5080320 & 1905120 & 381024 & 42336 & 2592 & 81 & 1
\end{array}\right]$$

```
>>> production_matrix(R), production_matrix(R, exp=True)
```

$$\left(\left[\begin{array}{ccccccccc}
1 & 1 & & & & & & & \\
1 & 3 & 1 & & & & & & \\
 & 4 & 5 & 1 & & & & & \\
 & & 9 & 7 & 1 & & & & \\
 & & & 16 & 9 & 1 & & & \\
 & & & & 25 & 11 & 1 & & \\
 & & & & & 36 & 13 & 1 & \\
 & & & & & & 49 & 15 & 1 \\
 & & & & & & & 64 & 17
\end{array}\right], \left[\begin{array}{ccccccccc}
1 & 1 & & & & & & & \\
1 & 3 & 2 & & & & & & \\
 & 2 & 5 & 3 & & & & & \\
 & & 3 & 7 & 4 & & & & \\
 & & & 4 & 9 & 5 & & & \\
 & & & & 5 & 11 & 6 & & \\
 & & & & & 6 & 13 & 7 & \\
 & & & & & & 7 & 15 & 8 \\
 & & & & & & & 8 & 17
\end{array}\right]\right)$$

```
>>> inspect(R)
nature(is_ordinary=False, is_exponential=True)
```

1.3.5 Group inverse elements

In this final section we show how to compute the compositional inverse of a function and then apply this procedure to find the inverse of a given Riordan array. By small steps, your task is to find the compositional inverse of Pascal array's h function

$$h(t) = \frac{t}{1-t},$$

namely you want to find a function $\bar{h}$ such that $\bar{h}(h(t)) = t$. Starting from this very last identity we use the substitution notation

$$\bar{h}(h(t)) = t \leftrightarrow \left[\bar{h}(y) = t \,|\, y = h(t)\right]$$

that allows us to reduce the original problem to solve $y = h(t)$ with respect to t; formally, using the definition of h we rewrite

$$y = \frac{t}{1-t} \quad \text{that implies} \quad t = \frac{y}{1+y}.$$

The latter identity can be used back in $\bar{h}(y) = t$ as substitution for t yielding $\bar{h}(y) = \dfrac{y}{1+y}$ as required; this procedure is promptly implemented as

```python
def compositional_inverse(h_eq, y=symbols('y'), check=True):

    spec, body = h_eq.lhs, h_eq.rhs                     # Destructuring function h denoted by `h_eq`,
    t, = spec.args                                      # let `t` be the formal var of function h.
                                                        # h̄(h(t)) = t ↔ [h̄(y) = t | y = h(t)] allows us
    sols = solve(Eq(y, body), t)                        # to solve y = h(t) with respect to t because
    for sol in sols:                                    # h(t) is known, denoted by `body`. For each
        L = Lambda(y, sol)                              # solution `sol`, which depends on y, we build
        if L(0) == 0:                                   # a callable object `L`. If it vanishes in 0 and
                                                        # h̄(h(t)) = t, then it is a compositional
            if check: assert L(body).simplify() == t    # inverse of h.

            h_bar = Function(r'\bar{{ {} }}'.format(    # Prepare the name for function h̄ and
                str(spec.func)))
            eq = Eq(h_bar(y), sol.factor())             # build the corresponding equation that defines
            return eq                                   # h̄(y), compositional inverse of h(t).

    raise ValueError                                    # If the above code fails to return, raise an error.

def group_inverse(d_eq, h_eq, post=identity, check=True):

    t, y = symbols('t y')                                     # Formal symbols for symbolic functions f and g.
    g_fn, f_fn = Function('g'), Function('f')

    with lift_to_Lambda(d_eq, return_eq=False) as D:          # Promote equations `d` and `h` to become
      with lift_to_Lambda(h_eq, return_eq=True) as H:         # callables objects, returning equations as well.
        f_eq = compositional_inverse(H(t), y, check)          # Let function f be the compositional inverse of
        with lift_to_Lambda(f_eq, return_eq=False) as F:      # function h, then promote it as callable and
            F_t = F(t)                                        # (i) evaluate it at t;
            g = post(1/D(F_t))                                # (ii) build and refine g, the first function of
            f = post(F_t)                                     # the new Ra; (iii) refine f, the snd function.

    g_eq = Eq(g_fn(t), g.simplify())                          # Build corresponding equation objects with
    f_eq = Eq(f_fn(t), f.simplify())                          # simplified expressions.

    if check:                                                 # If it is required to certify the computation,
        with lift_to_Lambda(g_eq, return_eq=False) as G:      # then promote the just built equations `g` and
          with lift_to_Lambda(f_eq, return_eq=False) as F:    # `f` to perform group operation in order to
            H_rhs = H(t).rhs                                  # check that, it yields the identity element,
            assert (D(t)*G(H_rhs)).simplify() == 1            # which has (i) 1 as first component and
            assert F(H_rhs).simplify() == t                   # (ii) t as second component.

    return g_eq, f_eq
```

Example 14. *Compositional inverse of Catalan tringle's* $\mathfrak{h}$ *generating function:*

```
>>> catalan_term = (1-sqrt(1-4*t))/(2*t)
>>> d_series = Eq(d_fn(t), catalan_term)
>>> h_series = Eq(h_fn(t), t*catalan_term)
>>> h_series, compositional_inverse(h_series)
```

$$\left(\mathfrak{h}(t) = -\frac{1}{2}\sqrt{-4t+1} + \frac{1}{2}, \quad \bar{\mathfrak{h}}(y) = -y\left(y-1\right)\right)$$

```
>>> C_inverse = group_inverse(d_series, h_series, post=radsimp)
>>> C_inverse
```

$$\left(g(t) = \frac{1}{2}\sqrt{4t^{2}-4t+1} + \frac{1}{2}, \quad f(t) = t\left(-t+1\right)\right)$$

```
>>> R = Matrix(10, 10, riordan_matrix_by_convolution(
...                     10, C_inverse[0], C_inverse[1]))
>>> R
```

$$\begin{bmatrix}
1 & & & & & & & & & \\
-1 & 1 & & & & & & & & \\
 & -2 & 1 & & & & & & & \\
 & 1 & -3 & 1 & & & & & & \\
 & & 3 & -4 & 1 & & & & & \\
 & & -1 & 6 & -5 & 1 & & & & \\
 & & & -4 & 10 & -6 & 1 & & & \\
 & & & 1 & -10 & 15 & -7 & 1 & & \\
 & & & & 5 & -20 & 21 & -8 & 1 & \\
 & & & & -1 & 15 & -35 & 28 & -9 & 1
\end{bmatrix}$$

Conclusions

This chapter offers to the reader a concise review of the theory of Riordan Arrays by recalling definitions, characterizations and their fundamental properties; moreover, we pair these formal arguments with a set of software abstractions that allow us to mimic the theory with objects living in a computer. Coding our definitions using the Python programming language and taking advantage of the symbolic module *Sympy*, we provide a coherent and unified environment to experiment in.

2

Functions and Jordan canonical forms of Riordan matrices

This chapter is an extended version of the recently published paper [Merlini and Nocentini, 2019] which collects results about Riordan arrays in the framework of *matrix functions*; actually, the following methodology applies to any square matrix $m \times m$ with *exactly one* eigenvalue λ of *algebraic* multiplicity $m \in \mathbb{N}$. Generalized Lagrange bases are used to construct Hermite polynomials that interpolate a family of functions; moreover, we show a parallel application of such functions via Jordan canonical forms and case studies are given.

2.1 Introduction

This work started as an educational effort to construct a practical framework that allows us to lift a scalar function $f : \mathbb{R} \rightarrow \mathbb{R}$ to a matrix function $g_f : \mathbb{R}^{m\times m} \rightarrow \mathbb{R}^{m\times m}$, $m \in \mathbb{N}$. Although many books [Gantmacher, 1959, Golub and Loan, 1996, Horn and Johnson, 1991, Lancaster and Tismenetsky, 1985] study this argument, our approach is in the spirit of [Higham, 2008], thus it does not include elementwise operations, functions producing a scalar result (such as the trace, the determinant, the spectral radius, the condition number), or matrix transformations (such as the transpose, the adjugate, the slice of a submatrix).

We provide two equivalent characterizations of the lifting process: let f be the function to be applied to a square matrix A, then the former is based on A's eigenvalues, its *algebraic* multiplicities and f's derivatives, according to [Runckel and Pittelkow, 1983, Verde-Star, 2005]; the latter is based on A's *Jordan Canonical Form*, an established approach to apply a function to a matrix.

We restrict ourselves to a class of matrices belonging to the *Riordan group* [Merlini et al., 1997, Shapiro et al., 1991, Sprugnoli, 1994, He, 2015], namely lower triangular infinite matrices that can be also manipulated algebraically using generating functions. Riordan arrays are powerful tools in combinatorics and in the analysis of algorithms, but here we focus on common properties arising from their structure to build polynomials interpolating desired functions; in fact, each minor $m \times m$ of a Riordan array $\mathcal{R}$ shares the *same and unique* eigenvalue λ_1 with algebraic multiplicity m.

We report application of a class of differentiable functions to the matrices of binomial coefficients, Catalan and Stirling numbers; for example, starting with 8×8 minors of the Pascal and Catalan triangles

$$\mathcal{P}_8 = \begin{bmatrix} 1 &&&&&&& \\ 1 & 1 &&&&&& \\ 1 & 2 & 1 &&&&& \\ 1 & 3 & 3 & 1 &&&& \\ 1 & 4 & 6 & 4 & 1 &&& \\ 1 & 5 & 10 & 10 & 5 & 1 && \\ 1 & 6 & 15 & 20 & 15 & 6 & 1 & \\ 1 & 7 & 21 & 35 & 35 & 21 & 7 & 1 \end{bmatrix} \quad \text{and}$$

$$\mathcal{C}_8 = \begin{bmatrix} 1 &&&&&&& \\ 1 & 1 &&&&&& \\ 2 & 2 & 1 &&&&& \\ 5 & 5 & 3 & 1 &&&& \\ 14 & 14 & 9 & 4 & 1 &&& \\ 42 & 42 & 28 & 14 & 5 & 1 && \\ 132 & 132 & 90 & 48 & 20 & 6 & 1 & \\ 429 & 429 & 297 & 165 & 75 & 27 & 7 & 1 \end{bmatrix}$$

respectively, which are two of the most commonly known Riordan arrays, we find matrices

$$\sqrt[3]{\mathcal{P}_8} = \begin{bmatrix} 1 &&&&&&& \\ \frac{1}{3} & 1 &&&&&& \\ \frac{1}{9} & \frac{2}{3} & 1 &&&&& \\ \frac{1}{27} & \frac{1}{3} & 1 & 1 &&&& \\ \frac{1}{81} & \frac{4}{27} & \frac{2}{3} & \frac{4}{3} & 1 &&& \\ \frac{1}{243} & \frac{5}{81} & \frac{10}{27} & \frac{10}{9} & \frac{5}{3} & 1 && \\ \frac{1}{729} & \frac{2}{81} & \frac{5}{27} & \frac{20}{27} & \frac{5}{3} & 2 & 1 & \\ \frac{1}{2187} & \frac{7}{729} & \frac{7}{81} & \frac{35}{81} & \frac{35}{27} & \frac{7}{3} & \frac{7}{3} & 1 \end{bmatrix} \quad \text{and}$$

$$e^{\mathcal{C}_8} = e\begin{bmatrix} 1 & & & & & & & \\ 1 & 1 & & & & & & \\ 3 & 2 & 1 & & & & & \\ \frac{23}{2} & 8 & 3 & 1 & & & & \\ \frac{154}{3} & 37 & 15 & 4 & 1 & & & \\ \frac{127}{4} & \frac{572}{3} & \frac{163}{2} & 24 & 5 & 1 & & \\ \frac{746}{5} & \frac{6439}{6} & 478 & 15 & 35 & 6 & 1 & \\ \frac{52481}{6} & \frac{39899}{6} & \frac{125}{4} & \frac{2965}{3} & \frac{495}{2} & 48 & 7 & 1 \end{bmatrix}$$

such that $\sqrt[3]{\mathcal{P}_8} \cdot \sqrt[3]{\mathcal{P}_8} \cdot \sqrt[3]{\mathcal{P}_8} = \mathcal{P}_8$ and $L_8\left(e^{\mathcal{C}_8}\right) = \mathcal{C}_8$, where

$$L_8(z) = \frac{z^7}{7e^7} - \frac{7z^6}{6e^6} + \frac{21z^5}{5e^5} - \frac{35z^4}{4e^4} + \frac{35z^3}{3e^3} - \frac{21z^2}{2e^2} + \frac{7z}{e} - \frac{223}{140}$$

is a polynomial that interpolates the $\log$ function; moreover, matrices $\sin(\mathcal{P}_8)$ and $\cos(\mathcal{P}_8)$ satisfying the classic identity

$$\sin(\mathcal{P}_8) \cdot \sin(\mathcal{P}_8) + \cos(\mathcal{P}_8) \cdot \cos(\mathcal{P}_8) = I_8, \tag{2.1}$$

where I is the identity matrix, the r-th power with $r \in \mathbb{Q}$ and the $\log$ functions are studied in details.

Moreover, we show how to build matrices X and Y to factor pairs of Riordan matrices $\mathcal{A}$ and $\mathcal{B}$ in Jordan canonical forms $\mathcal{A}\,X = X\,J$ and $\mathcal{B}\,Y = Y\,J$ respectively, both sharing matrix J which has a simple and interesting structure. First, we study the application of a function f to matrix J to ease the computation of $f(\mathcal{A})$ and $f(\mathcal{B})$; second, we prove that it is always possible to write a Riordan array $\mathcal{A}$ as a linear transformation of any other Riordan array $\mathcal{B}$ by means of matrices X and Y appearing in their Jordan canonical forms (in particular, there are *uncountably many* such transformations since X and Y are defined on top of arbitrary vectors $\boldsymbol{v}, \boldsymbol{w} \in \mathbb{R}^m$).

Finally, to compare and contrast the study of a matrix with a single eigenvalue with the study of a matrix with at least two different eigenvalues, we show an example concerning the Fibonacci numbers' generator matrix. All theorems and facts have been tested and confirmed by reproducible artifacts using a symbolic module on top of the Python programming language, fully available online in [Nocentini].

2.2 *Basic definitions and notations*

Let $A \in \mathbb{R}^{m \times m}$ be a matrix and denote with $\sigma(A)$ the spectrum of A, namely the set of A's eigenvalues $\sigma(A) = \{\lambda_i : A\boldsymbol{v}_i = \lambda_i \boldsymbol{v}_i, \boldsymbol{v}_i \in \mathbb{R}^m\}$ with corresponding multiplicities m_i such that $\sum_{i=1}^{\nu} m_i = m$.

Let $\nu = |\sigma(A)|$ and define the *characteristic polynomial* $p(\lambda) = \det(A - \lambda I) = \prod_{i=1}^{\nu} (\lambda - \lambda_i)^{m_i}$ of matrix A. The degree of p is m and any polynomial h of degree greater than m can be divided as $h(\lambda) = q(\lambda)p(\lambda) + r(\lambda)$ where $\deg r(\lambda) < m$; by the Cayley-Hamilton theorem $p(A) = O$ where O is the zero matrix, therefore $h(A) = r(A)$ holds, namely polynomials h and r (possibly of *different degrees*) yield the same matrix when applied to A. Moreover, $\left.\frac{\partial^{(j)}p}{\partial\lambda^j}\right|_{\lambda=\lambda_i} = 0$ implies

$$\left.\frac{\partial^{(j)}\left(h(\lambda) - r(\lambda)\right)}{\partial\lambda^j}\right|_{\lambda=\lambda_i} = \left.\frac{\partial^{(j)}\left(q(\lambda)p(\lambda)\right)}{\partial\lambda^j}\right|_{\lambda=\lambda_i} = 0,$$

so polynomials h and r satisfy $h(A) = r(A)$ if and only if

$$\left.\frac{\partial^{(j)}h}{\partial\lambda^j} = \frac{\partial^{(j)}r}{\partial\lambda^j}\right|_{\lambda=\lambda_i}, \quad \begin{array}{l} i \in \{1, \dots, \nu\} \\ j \in \{0, \dots, m_i - 1\} \end{array};$$

in words, *polynomials* h *and* r *take the same values on* $\sigma(A)$.

Let $f : \mathbb{R} \to \mathbb{R}$ be a function on the formal variable z; we say that f *is defined on* $\sigma(A)$ if exists

$$\left.\frac{\partial^{(j)}f}{\partial z^j}\right|_{z=\lambda_i}, \quad \begin{array}{l} i \in \{1, \dots, \nu\} \\ j \in \{0, \dots, m_i - 1\} \end{array}.$$

Given a function f defined on $\sigma(A)$, a polynomial g can be defined such that f and g take the same values on $\sigma(A)$; in particular, g can be written using the base of *generalized Lagrange polynomials* $\Phi_{i,j} \in \prod_{m-1}$, where $\prod_r$ denotes the set of polynomials of degree $r \in \mathbb{N}$. Coefficients of each polynomial $\Phi_{i,j}$ are implicitly defined to be the solutions of the system with m constraints

$$\left.\frac{\partial^{(r-1)}\Phi_{i,j}}{\partial z^{r-1}}\right|_{z=\lambda_l} = \delta_{i,l}\delta_{j,r}, \quad \begin{array}{l} l \in \{1, \dots, \nu\} \\ r \in \{1, \dots, m_l\} \end{array}, \tag{2.2}$$

δ being the Kroneker delta, defined as $\delta_{i,j} = 1$ if and only if $i = j$, otherwise 0; finally, polynomial g is called an *Hermite interpolating polynomial* and is formally defined as

$$g(z) = \sum_{i=1}^{\nu}\sum_{j=1}^{m_i} \left.\frac{\partial^{(j-1)}f}{\partial z^{j-1}}\right|_{z=\lambda_i} \Phi_{i,j}(z). \tag{2.3}$$

Remark 15. *Observe that if* $m_i = 1$ *for all* $i \in \{1, \dots, \nu\}$ *then* $m = \nu$ *and polynomials* $\Phi_{i,1}$ *reduce to the usual Lagrange base; let* $\nu = 4$*, then polynomials* $\Phi_{i,1}, \Phi_{i,2}, \Phi_{i,3}, \Phi_{i,4} \in \prod_3$

defined as

$$\begin{aligned}
\Phi_{1,1}(z) &= \frac{(z-\lambda_2)(z-\lambda_3)(z-\lambda_4)}{(\lambda_1-\lambda_2)(\lambda_1-\lambda_3)(\lambda_1-\lambda_4)},\\
\Phi_{2,1}(z) &= -\frac{(z-\lambda_1)(z-\lambda_3)(z-\lambda_4)}{(\lambda_1-\lambda_2)(\lambda_2-\lambda_3)(\lambda_2-\lambda_4)},\\
\Phi_{3,1}(z) &= \frac{(z-\lambda_1)(z-\lambda_2)(z-\lambda_4)}{(\lambda_1-\lambda_3)(\lambda_2-\lambda_3)(\lambda_3-\lambda_4)} \quad \textit{and}\\
\Phi_{4,1}(z) &= -\frac{(z-\lambda_1)(z-\lambda_2)(z-\lambda_3)}{(\lambda_1-\lambda_4)(\lambda_2-\lambda_4)(\lambda_3-\lambda_4)}
\end{aligned}$$

are a Lagrange base with respect to eigenvalues $\lambda_1, \lambda_2, \lambda_3$ *and* λ_4*, respectively. On the other hand, if* $\nu = 1$ *then there is only one eigenvalue* λ_1 *with algebraic multiplicity* $m_1 = m$*; let* $m = 8$*, then polynomials* $\Phi_{1,1}, \Phi_{1,2}, \Phi_{1,3}, \Phi_{1,4}, \Phi_{1,5}, \Phi_{1,6}, \Phi_{1,7}, \Phi_{1,8} \in \prod_7$ *defined as*

$$\begin{aligned}
\Phi_{1,1}(z) &= 1,\\
\Phi_{1,2}(z) &= z-\lambda_1,\\
\Phi_{1,3}(z) &= \frac{z^2}{2} - z\lambda_1 + \frac{\lambda_1^2}{2},\\
\Phi_{1,4}(z) &= \frac{z^3}{6} - \frac{z^2\lambda_1}{2} + \frac{z\lambda_1^2}{2} - \frac{\lambda_1^3}{6},\\
\Phi_{1,5}(z) &= \frac{z^4}{24} - \frac{z^3\lambda_1}{6} + \frac{z^2\lambda_1^2}{4} - \frac{z\lambda_1^3}{6} + \frac{\lambda_1^4}{24},\\
\Phi_{1,6}(z) &= \frac{z^5}{120} - \frac{z^4\lambda_1}{24} + \frac{z^3\lambda_1^2}{12} - \frac{z^2\lambda_1^3}{12} + \frac{z\lambda_1^4}{24} - \frac{\lambda_1^5}{120},\\
\Phi_{1,7}(z) &= \frac{z^6}{720} - \frac{z^5\lambda_1}{120} + \frac{z^4\lambda_1^2}{48} - \frac{z^3\lambda_1^3}{36} + \frac{z^2\lambda_1^4}{48} - \frac{z\lambda_1^5}{120} + \frac{\lambda_1^6}{720},\\
\Phi_{1,8}(z) &= \frac{z^7}{5040} - \frac{z^6\lambda_1}{720} + \frac{z^5\lambda_1^2}{240} - \frac{z^4\lambda_1^3}{144} + \frac{z^3\lambda_1^4}{144} - \frac{z^2\lambda_1^5}{240} + \frac{z\lambda_1^6}{720} - \frac{\lambda_1^7}{5040}
\end{aligned} \tag{2.4}$$

are a generalized Lagrange base with respect to the unique eigenvalue λ_1.

Now we apply this framework to the Riordan group.

2.3 Riordan matrices

From here on, $\mathcal{R}_m \in \mathbb{R}^{m\times m}$ denotes a *finite Riordan matrix*, namely a chunk of the infinite matrix $\mathcal{R}$ composed of the first m rows and the first m columns, see [Luzon et al., 2016] for a study of finite Riordan matrices. Due to its triangular shape, $\mathcal{R}_m$ admits the characteristic polynomial $p(\lambda) = \det(\mathcal{R}_m - \lambda\, I_m) = (\lambda_1 - \lambda)^m$, so $\sigma(\mathcal{R}_m) = \{\lambda_1\}$ entails $\nu = 1$ and eigenvalue λ_1 gets multiplicity $m_1 = m$; usually, functions d and h satisfy $d(0) = 1$ and $h'(0) = 1$ respectively, therefore $\lambda_1 = 1$. We relax the condition $\lambda_1 = 1$ in order to use λ_1 as a pure symbol to spot structures with respect to λ_1 and, lately, perform the substitution to specialize non-ground terms.

Lemma 16. *Let $\mathcal{R}$ be a Riordan array and $m_1 \in \mathbb{N}$, then a base of generalized Lagrange polynomials $\Phi_{1,j} \in \prod_{m_1-1}$ for the finite Riordan matrix $\mathcal{R}_{m_1}$ is*

$$\Phi_{1,j}(z) = \frac{(z-\lambda_1)^{j-1}}{(j-1)!}, \quad j \in \{1, \dots, m_1\}. \tag{2.5}$$

Proof. Reasoning on Equation 2.4 we write polynomials $\Phi_{i,j}$ in matrix notation

$$\begin{bmatrix} 1 \\ -\lambda_1 & 1 \\ \frac{\lambda_1^2}{2} & -\lambda_1 & 1 \\ -\frac{\lambda_1^3}{6} & \frac{\lambda_1^2}{2} & -\lambda_1 & 1 \\ \frac{\lambda_1^4}{24} & -\frac{\lambda_1^3}{6} & \frac{\lambda_1^2}{2} & -\lambda_1 & 1 \\ -\frac{\lambda_1^5}{120} & \frac{\lambda_1^4}{24} & -\frac{\lambda_1^3}{6} & \frac{\lambda_1^2}{2} & -\lambda_1 & 1 \\ \frac{\lambda_1^6}{720} & -\frac{\lambda_1^5}{120} & \frac{\lambda_1^4}{24} & -\frac{\lambda_1^3}{6} & \frac{\lambda_1^2}{2} & -\lambda_1 & 1 \\ -\frac{\lambda_1^7}{5040} & \frac{\lambda_1^6}{720} & -\frac{\lambda_1^5}{120} & \frac{\lambda_1^4}{24} & -\frac{\lambda_1^3}{6} & \frac{\lambda_1^2}{2} & -\lambda_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \begin{bmatrix} 1 \\ z \\ \frac{z^2}{2!} \\ \frac{z^3}{3!} \\ \frac{z^4}{4!} \\ \frac{z^5}{5!} \\ \frac{z^6}{6!} \\ \frac{z^7}{7!} \\ \vdots \end{bmatrix} = \begin{bmatrix} \Phi_{1,1}(z) \\ \Phi_{1,2}(z) \\ \Phi_{1,3}(z) \\ \Phi_{1,4}(z) \\ \Phi_{1,5}(z) \\ \Phi_{1,6}(z) \\ \Phi_{1,7}(z) \\ \Phi_{1,8}(z) \\ \vdots \end{bmatrix} \tag{2.6}$$

where the generic coefficient $d_{n,k}$ has the closed form

$$d_{n,k} = \frac{(-\lambda_1)^{n-k}}{(n-k)\,!}, \quad k \leq n;$$

therefore, we define

$$\begin{aligned} \Phi_{1,j}(z) &= \sum_{k=0}^{j-1} \frac{(-\lambda_1)^{j-1-k}}{(j-1-k)!} \frac{z^k}{k!} \\ &= \frac{1}{(j-1)!} \sum_{k=0}^{j-1} \binom{j-1}{k} z^k (-\lambda_1)^{j-1-k} = \frac{(z-\lambda_1)^{j-1}}{(j-1)!} \end{aligned}$$

which are required to satisfy the set of constraints

$$\left.\frac{\partial^{(r-1)} \Phi_{1,j}}{\partial z}\right|_{z=\lambda_1} = \delta_{j,r} \quad \text{where} \quad r \in \{1, \dots, m_1\},$$

obtained by instantiating Equation 2.2. We proceed by cases, (i) if $j < r$ then it holds because the derivative vanishes, (ii) if $j = r$ then it holds because the derivative equals 1; otherwise,

(iii) if $j > r$ then

$$\left.\frac{\partial^{(r-1)}\Phi_{1,j}}{\partial z^{r-1}}\right|_{z=\lambda_1} = \left.\frac{(r-1)!}{(j-1)!}(z-\lambda_1)^{j-r}\right|_{z=\lambda_1} = 0$$

as required. □

Observing that the outer sum in Equation 2.3 does exactly *one* iteration because $\nu = 1$ and by using polynomials in Equation 2.5 we state the following

Theorem 17. *Let $\mathcal{R}$ be a Riordan array, $m \in \mathbb{N}$ and $f : \mathbb{R} \to \mathbb{R}$; then the polynomial*

$$g_m(z) = \sum_{j=1}^{m} \left.\frac{\partial^{(j-1)} f}{\partial z^{j-1}}\right|_{z=\lambda_1} \frac{(z-\lambda_1)^{j-1}}{(j-1)!} \tag{2.7}$$

is a Hermite interpolating polynomial of function f defined on $\sigma(\mathcal{R}_m)$.

Remark 18. *For any Riordan array $\mathcal{R}$, the polynomial*

$$\begin{aligned}
g_8(z) = {} & \left.\frac{1}{5040}\frac{d^7}{dz^7}f(z)\right|_{z=1} z^7 \\
& + \left.\left(\frac{1}{720}\frac{d^6}{dz^6}f(z) - \frac{1}{720}\frac{d^7}{dz^7}f(z)\right)\right|_{z=1} z^6 \\
& + \left.\left(\frac{1}{120}\frac{d^5}{dz^5}f(z) - \frac{1}{120}\frac{d^6}{dz^6}f(z) + \frac{1}{240}\frac{d^7}{dz^7}f(z)\right)\right|_{z=1} z^5 \\
& + \left.\left(\frac{1}{24}\frac{d^4}{dz^4}f(z) - \frac{1}{24}\frac{d^5}{dz^5}f(z) + \frac{1}{48}\frac{d^6}{dz^6}f(z) - \frac{1}{144}\frac{d^7}{dz^7}f(z)\right)\right|_{z=1} z^4 \\
& + \left.\left(\frac{1}{6}\frac{d^3}{dz^3}f(z) - \frac{1}{6}\frac{d^4}{dz^4}f(z) + \frac{1}{12}\frac{d^5}{dz^5}f(z) - \frac{1}{36}\frac{d^6}{dz^6}f(z) + \frac{1}{144}\frac{d^7}{dz^7}f(z)\right)\right|_{z=1} z^3 \\
& + \left.\left(\frac{1}{2}\frac{d^2}{dz^2}f(z) - \frac{1}{2}\frac{d^3}{dz^3}f(z) + \frac{1}{4}\frac{d^4}{dz^4}f(z) - \frac{1}{12}\frac{d^5}{dz^5}f(z) + \frac{1}{48}\frac{d^6}{dz^6}f(z) - \frac{1}{240}\frac{d^7}{dz^7}f(z)\right)\right|_{z=1} z^2 \\
& + \left.\left(\frac{d}{dz}f(z) - \frac{d^2}{dz^2}f(z) + \frac{1}{2}\frac{d^3}{dz^3}f(z) - \frac{1}{6}\frac{d^4}{dz^4}f(z) + \frac{1}{24}\frac{d^5}{dz^5}f(z) - \frac{1}{120}\frac{d^6}{dz^6}f(z) + \frac{1}{720}\frac{d^7}{dz^7}f(z)\right)\right|_{z=1} z \\
& + \left.\left(f(z) - \frac{d}{dz}f(z) + \frac{1}{2}\frac{d^2}{dz^2}f(z) - \frac{1}{6}\frac{d^3}{dz^3}f(z) + \frac{1}{24}\frac{d^4}{dz^4}f(z) - \frac{1}{120}\frac{d^5}{dz^5}f(z) + \frac{1}{720}\frac{d^6}{dz^6}f(z) - \frac{1}{5040}\frac{d^7}{dz^7}f(z)\right)\right|_{z=1}
\end{aligned}$$

interpolates a function f defined on $\sigma(\mathcal{R}_8)$.

2.4 Functions and polynomials

In this section we instantiate the abstract framework just described to functions

$$\begin{gathered}
f(z) = z^r,\ f(z) = \frac{1}{z},\ f(z) = \sqrt{z},\ f(z) = e^{\alpha z}, \\
f(z) = \log z,\ f(z) = \sin z \quad \text{and} \quad f(z) = \cos z,
\end{gathered}$$

where $r, \alpha \in \mathbb{R}$; in parallel, we construct and show corresponding Hermite interpolating polynomials in a sequence of theorems, respectively. From now on, we use m and λ instead of m_1 and λ_1 to simplify the notation; moreover, we instantiate $\lambda = 1$ which is the natural eigenvalue for Riordan arrays.

We start by generalizing the r-th power A^r, usually carried out as $\underbrace{A \cdots A}_{r \text{ times}}$, to *rational* powers $r \in \mathbb{Q}$.

Theorem 19. *Let* $f(z) = z^r$, *where* $r \in \mathbb{Q}$, *and* $\mathcal{R}$ *be a Riordan array; then*

$$\begin{aligned} P_m(z) &= \sum_{j=0}^{m-1} \binom{r}{j} (z-1)^j \quad \textit{and, explicitly,} \\ P_m(z) &= \sum_{k=0}^{m-1} \left(\sum_{j=k}^{m-1} (-1)^j \binom{r}{j} \binom{j}{k} \right) (-z)^k \end{aligned} \tag{2.8}$$

are both Hermite interpolating polynomials of the r*-th power function for the minor* $\mathcal{R}_m$, $m \in \mathbb{N}$.

Proof. The closed form of the j-th derivative of function f is

$$\frac{\partial^{(j)} f(z)}{\partial z} = (r)_{(j)} z^{r-j}, \quad j \in \mathbb{N}$$

where $(r)_{(j)} = r(r-1)\cdots(r-j+1)$ denotes the falling factorial; therefore,

$$\begin{aligned} P_m(z) &= \sum_{j=1}^{m} (r)_{(j-1)} z^{r-j+1} \Big|_{z=1} \Phi_{1,j}(z) \\ &= \sum_{j=1}^{m} \frac{(r)_{(j-1)}}{(j-1)_{(j-1)}} (z-\lambda_1)^{j-1} = \sum_{j=0}^{m-1} \binom{r}{j} (z-\lambda_1)^j \end{aligned}$$

restoring $\lambda_1 = 1$ proves the first identity. On the other hand,

$$\begin{aligned} P_m(z) &= \sum_{j=1}^{m} \sum_{k=0}^{j-1} \frac{(r)_{(j-1)}}{(j-1)_{(j-1)}} \frac{(j-1)!(-1)^{j-1-k}}{(j-1-k)!} \frac{z^k}{k!} \\ &= \sum_{j=1}^{m} \sum_{k=0}^{j-1} (-1)^{j-1} \binom{r}{j-1} \binom{j-1}{k} (-z)^k \\ &= \sum_{k=0}^{m-1} \left(\sum_{j=k+1}^{m} (-1)^{j-1} \binom{r}{j-1} \binom{j-1}{k} \right) (-z)^k \\ &= \sum_{k=0}^{m-1} \left(\sum_{j=k}^{m-1} (-1)^j \binom{r}{j} \binom{j}{k} \right) (-z)^k \end{aligned}$$

proves the explicit one. □

Instantiation $r = -1$ in the previous theorem yields a Hermite interpolating polynomial for the inverse function which, in the explicit form, reduces to a binomial transform.

Theorem 20. *Let* $f(z) = \frac{1}{z}$ *and* $\mathcal{R}$ *be a Riordan array; then*

$$\begin{aligned} I_m(z) &= \sum_{j=0}^{m-1} (-1)^j \, (z-1)^j \quad \textit{and, explicitly,} \\ I_m(z) &= \sum_{k=0}^{m-1} \binom{m}{k+1} (-z)^k \end{aligned} \tag{2.9}$$

are both Hermite interpolating polynomials of the inverse function for the minor $\mathcal{R}_m$, $m \in \mathbb{N}$.

Proof. The closed form of the j-th derivative of function f is

$$\frac{\partial^{(j)} f(z)}{\partial z^j} = \frac{(-1)^j j!}{z^{j+1}}, \quad j \in \mathbb{N};$$

therefore, restoring $\lambda_1 = 1$ in

$$\begin{aligned} I_m(z) &= \sum_{j=1}^{m} \left. \frac{(-1)^{j-1}(j-1)!}{z^j} \right|_{z=1} \Phi_{1,j}(z) \\ &= \sum_{j=1}^{m} (-1)^{j-1} \, (z-\lambda_1)^{j-1} = \sum_{j=0}^{m-1} (-1)^j \, (z-\lambda_1)^j \end{aligned}$$

proves the first identity. On the other hand, in

$$\begin{aligned} I_m(z) &= \sum_{j=1}^{m} \sum_{k=0}^{j-1} \binom{j-1}{k} (-z)^k \\ &= \sum_{k=0}^{m-1} \left(\sum_{j=k+1}^{m} \binom{j-1}{k} \right) (-z)^k \\ &= \sum_{k=0}^{m-1} \left(\sum_{j=k}^{m-1} \binom{j}{k} \right) (-z)^k \end{aligned}$$

the inner sum admits the closed expression $\binom{m}{k+1}$, proving the explicit one. □

Instantiation $r = \frac{1}{2}$ yields the interpolation of the square root function, we report its derivation for completeness.

Theorem 21. *Let* $f(z) = \sqrt{z}$ *and* $\mathcal{R}$ *be a Riordan array and* $\binom{\frac{1}{2}}{j} = \frac{(-1)^{j-1}}{4^j(2j-1)}\binom{2j}{j}$*; then,*

$$
\begin{aligned}
R_m(z) &= \sum_{j=0}^{m-1} \binom{\frac{1}{2}}{j} (z-1)^j \quad \textit{and, explicitly,} \\
R_m(z) &= \sum_{k=0}^{m-1} \left(\sum_{j=k}^{m-1} (-1)^j \binom{\frac{1}{2}}{j} \binom{j}{k} \right) (-z)^k
\end{aligned}
\tag{2.10}
$$

are both Hermite interpolating polynomials of the square root function for the minor $\mathcal{R}_m$, $m \in \mathbb{N}$.

Proof. The closed form of the j-th derivative of function f is

$$
\frac{\partial^{(j)} f(z)}{\partial z^j} = \frac{(-1)^{j-1}}{2} \frac{(j-1)!}{4^{j-1}} \binom{2(j-1)}{j-1} \frac{1}{z^{\frac{2(j-1)+1}{2}}}, \quad 0 < j \in \mathbb{N};
$$

therefore, first observing that $f(1)\Phi_{1,1}(z) = 1$ entails

$$
\begin{aligned}
R_m(z) &= \sum_{j=0}^{m-1} \left.\frac{\partial^{(j)} f}{\partial z^j}\right|_{z=1} \Phi_{1,j+1}(z) \\
&= 1 + \sum_{j=1}^{m-1} \frac{(-1)^{j-1}}{2} \frac{(j-1)!}{4^{j-1}} \binom{2(j-1)}{j-1} \left.\frac{1}{z^{\frac{2(j-1)+1}{2}}}\right|_{z=1} \Phi_{1,j+1}(z);
\end{aligned}
$$

second, identities $\binom{v}{w} = \frac{v}{w}\binom{v-1}{w-1}$ and $\binom{-\frac{1}{2}}{j} = \frac{(-1)^j}{4^j}\binom{2j}{j}$ allow us to rewrite

$$
\begin{aligned}
R_m(z) &= 1 + \frac{1}{2} \sum_{j=1}^{m-1} \frac{(-1)^{j-1}}{j\,4^{j-1}} \binom{2(j-1)}{j-1} (z-1)^j \\
&= 1 + \frac{1}{2} \sum_{j=1}^{m-1} \frac{1}{j} \binom{-\frac{1}{2}}{j-1} (z-1)^j = 1 + \sum_{j=1}^{m-1} \binom{\frac{1}{2}}{j} (z-1)^j;
\end{aligned}
$$

finally, sum's coefficient equals 1 for $j = 0$, hence summation can be extended to start from index 0 incorporating the outer value 1, proving the first identity. On the other hand,

$$
\begin{aligned}
R_m(z) &= \sum_{j=0}^{m-1} \binom{\frac{1}{2}}{j} (z-1)^j = \sum_{j=0}^{m-1} \sum_{k=0}^{j} (-1)^j \binom{\frac{1}{2}}{j} \binom{j}{k} (-z)^k \\
&= \sum_{k=0}^{m-1} \left(\sum_{j=k}^{m-1} (-1)^j \binom{\frac{1}{2}}{j} \binom{j}{k} \right) (-z)^k
\end{aligned}
$$

proves the explicit one. □

Matrix exponentiation is a well studied problem [Moler and Loan, 2003], here we show another way in the Riordan arrays domain.

Theorem 22. *Let* $f(z) = e^{\alpha z}$*, where* $\alpha \in \mathbb{Q}$*, and* $\mathcal{R}$ *be a Riordan array; then*

$$\begin{aligned} E_m(z) &= e^{\alpha} \sum_{j=0}^{m-1} \frac{\alpha^j}{j!} (z-1)^j \quad \textit{and, explicitly,} \\ E_m(z) &= e^{\alpha} \sum_{k=0}^{m-1} \left(\sum_{j=k}^{m-1} \frac{(-\alpha)^j}{j!} \binom{j}{k} \right) (-z)^k \end{aligned} \tag{2.11}$$

are both Hermite interpolating polynomials of the exponential function for the minor $\mathcal{R}_m$, $m \in \mathbb{N}$.

Proof. The closed form of jth derivative of function f is

$$\frac{\partial^{(j)} f(z)}{\partial z^j} = \alpha^j e^{\alpha z}, \quad j \in \mathbb{N};$$

therefore, restoring $\lambda_1 = 1$ in

$$\begin{aligned} E_m(z) &= \sum_{j=1}^{m} \alpha^{j-1} e^{\alpha z} \Big|_{z=1} \Phi_{1,j}(z) \\ &= e^{\alpha} \sum_{j=1}^{m} \frac{\alpha^{j-1}}{(j-1)!} (z-\lambda_1)^{j-1} = e^{\alpha} \sum_{j=0}^{m-1} \frac{\alpha^j}{j!} (z-\lambda_1)^j \end{aligned}$$

proves the first identity. On the other hand,

$$\begin{aligned} E_m(z) &= e^{\alpha} \sum_{j=1}^{m} \sum_{k=0}^{j-1} \frac{(-\alpha)^{j-1}}{(j-1)!} \binom{j-1}{k} (-z)^k \\ &= e^{\alpha} \sum_{k=0}^{m-1} \left(\sum_{j=k+1}^{m} \frac{(-\alpha)^{j-1}}{(j-1)!} \binom{j-1}{k} \right) (-z)^k \end{aligned}$$

and moving the index j in the inner summation backward by 1 closes the proof. □

We show a dual theorem of the previous one concerning the interpolation of the logarithm function.

Theorem 23. *Let* $f(z) = \log z$ *and* $\mathcal{R}$ *be a Riordan array; let* H_n *be the* n*-th harmonic number, then*

$$\begin{aligned} L_m(z) &= \sum_{j=1}^{m-1} \frac{(-1)^{j-1}}{j}(z-1)^j \quad \textit{and, explicitly,} \\ L_m(z) &= -\sum_{k=1}^{m-1} \frac{1}{k}\binom{m-1}{k}(-z)^k - H_{m-1} \end{aligned} \tag{2.12}$$

are both Hermite interpolating polynomials of the logarithm function for the minor $\mathcal{R}_m$, $m \in \mathbb{N}$.

Proof. The closed form of the j-th derivative of function f is

$$\frac{\partial^{(j)} f(z)}{\partial z^j} = \frac{(-1)^{j-1}(j-1)!}{z^j}, \quad 0 < j \in \mathbb{N};$$

therefore, observing that $f(1)\Phi_{1,1}(z) = 0$ entails

$$\begin{aligned} L_m(z) &= \sum_{j=0}^{m-1} \left.\frac{\partial^{(j)} f}{\partial z^j}\right|_{z=1} \Phi_{1,j+1}(z) \\ &= \sum_{j=1}^{m-1} \left.\frac{(-1)^{j-1}(j-1)!}{z^j}\right|_{z=1} \Phi_{1,j+1}(z) \\ &= \sum_{j=1}^{m-1} \frac{(-1)^{j-1}}{j}(z-1)^j, \end{aligned}$$

proving the first identity. On the other hand,

$$\begin{aligned} L_m(z) &= -\sum_{j=1}^{m-1} \sum_{k=0}^{j} \frac{1}{j}\binom{j}{k}(-z)^k \\ &= -\sum_{k=1}^{m-1} \left(\sum_{j=k}^{m-1} \frac{1}{j}\binom{j}{k} \right)(-z)^k - \sum_{j=1}^{m-1} \frac{1}{j} \\ &= -\sum_{k=1}^{m-1} \frac{1}{k}\binom{m-1}{k}(-z)^k - H_{m-1} \end{aligned}$$

proves the explicit one. □

Remark 24. *For the sake of completeness, a Hermite interpolating polynomial* g *could also be studied by relaxing the condition* $\lambda = 1$ *thus considering* $\hat{g}(z, \lambda)$ *which subsumes* $g(z) = \hat{g}(z, 1)$. *Here are one of these augmented polynomials* *interpolating the* $\log$ *function*

$$\begin{aligned}\hat{L}_8(z,\lambda) = {} & \frac{z^7}{7\lambda^7} \\ & + z^6\left(-\frac{1}{6\lambda^6}-\frac{1}{\lambda^7}\right) \\ & + z^5\left(\frac{1}{5\lambda^5}+\frac{1}{\lambda^6}+\frac{3}{\lambda^7}\right) \\ & + z^4\left(-\frac{1}{4\lambda^4}-\frac{1}{\lambda^5}-\frac{5}{2\lambda^6}-\frac{5}{\lambda^7}\right) \\ & + z^3\left(\frac{1}{3\lambda^3}+\frac{1}{\lambda^4}+\frac{2}{\lambda^5}+\frac{10}{3\lambda^6}+\frac{5}{\lambda^7}\right) \\ & + z^2\left(-\frac{1}{2\lambda^2}-\frac{1}{\lambda^3}-\frac{3}{2\lambda^4}-\frac{2}{\lambda^5}-\frac{5}{2\lambda^6}-\frac{3}{\lambda^7}\right) \\ & + z\left(\frac{1}{\lambda}+\frac{1}{\lambda^2}+\frac{1}{\lambda^3}+\frac{1}{\lambda^4}+\frac{1}{\lambda^5}+\frac{1}{\lambda^6}+\frac{1}{\lambda^7}\right) \\ & + \log(\lambda)-\frac{1}{\lambda}-\frac{1}{2\lambda^2}-\frac{1}{3\lambda^3}-\frac{1}{4\lambda^4}-\frac{1}{5\lambda^5}-\frac{1}{6\lambda^6}-\frac{1}{7\lambda^7}.\end{aligned}$$

Theorem 25. *Let* $f_1(z) = \sin z$, $f_2(z) = \cos z$ *and* $\mathcal{R}$ *be a Riordan array; then*

$$\begin{aligned}S_m(z) = {} & \sin 1 \sum_{k=0}^{2\lceil\frac{m}{2}\rceil-2}\left(\sum_{j=\lceil\frac{k}{2}\rceil}^{\lceil\frac{m}{2}\rceil-1}\frac{(-1)^{3j}}{(2j)!}\binom{2j}{k}\right)(-z)^k \\ & + \cos 1 \sum_{k=0}^{2\lfloor\frac{m}{2}\rfloor-1}\left(\sum_{j=\lfloor\frac{k}{2}\rfloor}^{\lfloor\frac{m}{2}\rfloor-1}\frac{(-1)^{3j+1}}{(2j+1)!}\binom{2j+1}{k}\right)(-z)^k\end{aligned} \tag{2.13}$$

and

$$\begin{aligned}C_m(z) = {} & \cos 1 \sum_{k=0}^{2\lceil\frac{m}{2}\rceil-2}\left(\sum_{j=\lceil\frac{k}{2}\rceil}^{\lceil\frac{m}{2}\rceil-1}\frac{(-1)^{3j}}{(2j)!}\binom{2j}{k}\right)(-z)^k \\ & + \sin 1 \sum_{k=0}^{2\lfloor\frac{m}{2}\rfloor-1}\left(\sum_{j=\lfloor\frac{k}{2}\rfloor}^{\lfloor\frac{m}{2}\rfloor-1}\frac{(-1)^{3j+2}}{(2j+1)!}\binom{2j+1}{k}\right)(-z)^k\end{aligned} \tag{2.14}$$

are Hermite interpolating polynomials, explicitly written, of the sine and cosine functions for the minor $\mathcal{R}_m$, $m \in \mathbb{N}$, *respec-*

tively.

Proof. The closed form of the j-th derivative of function f_1 is

$$\frac{\partial^{(j)} f(z)}{\partial z^j} = \alpha_j \sin z + \alpha_{j-1} \cos z, \quad 0 < j \in \mathbb{N},$$

where $\alpha_{2k} = (-1)^k$ and $\alpha_{2k+1} = 0$ for $k \in \mathbb{N}$, with $\alpha_{-1} = 0$ required when $j = 0$. We rewrite

$$\begin{aligned}
S_m(z) &= \sum_{j=1}^{m} \left(\alpha_{j-1} \sin z + \alpha_{j-2} \cos z\right)\Big|_{z=1} \Phi_{1,j}(z) \\
&= \sin 1 \sum_{j=1}^{m} \alpha_{j-1} \Phi_{1,j}(z) + \cos 1 \sum_{j=1}^{m} \alpha_{j-2} \Phi_{1,j}(z) \\
&= \sin 1 \sum_{j=1}^{\lceil \frac{m}{2} \rceil} (-1)^{j-1} \Phi_{1,2j-1}(z) + \cos 1 \sum_{j=1}^{\lfloor \frac{m}{2} \rfloor} (-1)^{j-1} \Phi_{1,2j}(z) \\
&= \sin 1 \sum_{j=1}^{\lceil \frac{m}{2} \rceil} \sum_{k=0}^{2j-2} \frac{(-1)^{3j-3}}{k!(2j-2-k)!} (-z)^k \\
&+ \cos 1 \sum_{j=1}^{\lfloor \frac{m}{2} \rfloor} \sum_{k=0}^{2j-1} \frac{(-1)^{3j-2}}{k!(2j-1-k)!} (-z)^k.
\end{aligned}$$

Then, by swapping the sums and moving indices backwards in the inner sums we finally get

$$\begin{aligned}
S_m(z) = \sin 1 &\sum_{k=0}^{2\lceil \frac{m}{2} \rceil - 2} \left(\sum_{j=1+\lceil \frac{k}{2} \rceil}^{\lceil \frac{m}{2} \rceil} \frac{(-1)^{3j-3}}{(2j-2)!} \binom{2j-2}{k} \right) (-z)^k \\
+ \cos 1 &\sum_{k=0}^{2\lfloor \frac{m}{2} \rfloor - 1} \left(\sum_{j=1+\lfloor \frac{k}{2} \rfloor}^{\lfloor \frac{m}{2} \rfloor} \frac{(-1)^{3j-2}}{(2j-1)!} \binom{2j-1}{k} \right) (-z)^k \\
= \sin 1 &\sum_{k=0}^{2\lceil \frac{m}{2} \rceil - 2} \left(\sum_{j=\lceil \frac{k}{2} \rceil}^{\lceil \frac{m}{2} \rceil - 1} \frac{(-1)^{3j}}{(2j)!} \binom{2j}{k} \right) (-z)^k \\
+ \cos 1 &\sum_{k=0}^{2\lfloor \frac{m}{2} \rfloor - 1} \left(\sum_{j=\lfloor \frac{k}{2} \rfloor}^{\lfloor \frac{m}{2} \rfloor - 1} \frac{(-1)^{3j+1}}{(2j+1)!} \binom{2j+1}{k} \right) (-z)^k.
\end{aligned}$$

□

Proof. The closed form of the j-th derivative of function f_2 is

$$\frac{\partial^{(j)} f(z)}{\partial z^j} = \alpha_{j+1} \sin z + \alpha_j \cos z, \quad j \in \mathbb{N},$$

where coefficients α_i are defined in the sine function's proof, hence

$$\begin{aligned}
C_m(z) &= \sum_{j=1}^{m} \left(\alpha_j \sin z + \alpha_{j-1} \cos z\right)\Big|_{z=1} \Phi_{1,j}(z) \\
&= \sin 1 \sum_{j=1}^{m} \alpha_j \Phi_{1,j}(z) + \cos 1 \sum_{j=1}^{m} \alpha_{j-1} \Phi_{1,j}(z) \\
&= \cos 1 \sum_{j=1}^{\lceil \frac{m}{2} \rceil} (-1)^{j-1} \Phi_{1,2j-1}(z) + \sin 1 \sum_{j=1}^{\lfloor \frac{m}{2} \rfloor} (-1)^{j} \Phi_{1,2j}(z) \\
&= \cos 1 \sum_{j=1}^{\lceil \frac{m}{2} \rceil} \sum_{k=0}^{2j-2} \frac{(-1)^{3j-3}}{k!(2j-2-k)!} (-z)^k \\
&+ \sin 1 \sum_{j=1}^{\lfloor \frac{m}{2} \rfloor} \sum_{k=0}^{2j-1} \frac{(-1)^{3j-1}}{k!(2j-1-k)!} (-z)^k;
\end{aligned}$$

finally, repeating manipulations done in the previous proof we rewrite

$$\begin{aligned}
C_m(z) = \cos 1 &\sum_{k=0}^{2\lceil \frac{m}{2} \rceil - 2} \left(\sum_{j=1+\lceil \frac{k}{2} \rceil}^{\lceil \frac{m}{2} \rceil} \frac{(-1)^{3j-3}}{(2j-2)!} \binom{2j-2}{k} \right) (-z)^k \\
+ \sin 1 &\sum_{k=0}^{2\lfloor \frac{m}{2} \rfloor - 1} \left(\sum_{j=1+\lfloor \frac{k}{2} \rfloor}^{\lfloor \frac{m}{2} \rfloor} \frac{(-1)^{3j-1}}{(2j-1)!} \binom{2j-1}{k} \right) (-z)^k \\
= \cos 1 &\sum_{k=0}^{2\lceil \frac{m}{2} \rceil - 2} \left(\sum_{j=\lceil \frac{k}{2} \rceil}^{\lceil \frac{m}{2} \rceil - 1} \frac{(-1)^{3j}}{(2j)!} \binom{2j}{k} \right) (-z)^k \\
+ \sin 1 &\sum_{k=0}^{2\lfloor \frac{m}{2} \rfloor - 1} \left(\sum_{j=\lfloor \frac{k}{2} \rfloor}^{\lfloor \frac{m}{2} \rfloor - 1} \frac{(-1)^{3j+2}}{(2j+1)!} \binom{2j+1}{k} \right) (-z)^k.
\end{aligned}$$

$\square$

2.4.1 Case studies

Before showing explicit Hermite interpolating polynomials, we point out that evaluation of a polynomial $\Phi_{i,j} \in \prod_{m-1}$ belonging to a generalized Lagrange base will be carried out using the Horner algorithm for the sake of efficiency. Let $m = 8$, each polynomial can be written in abstract form as

$$\begin{aligned}\Phi_{i,j}(z) &= z^7\phi_{i,j,0} + z^6\phi_{i,j,1} + z^5\phi_{i,j,2} + z^4\phi_{i,j,3}\\ &\quad + z^3\phi_{i,j,4} + z^2\phi_{i,j,5} + z\phi_{i,j,6} + \phi_{i,j,7}\end{aligned}$$

and can be computed as

$$\Phi_{i,j}(z) = z\left(z\left(z\left(z\left(z\left(z\left(z\phi_{i,j,0} + \phi_{i,j,1}\right) + \phi_{i,j,2}\right) + \phi_{i,j,3}\right) + \phi_{i,j,4}\right) + \phi_{i,j,5}\right) + \phi_{i,j,6}\right) + \phi_{i,j,7},$$

where each coefficient $\phi_{i,j,k} \in \mathbb{R}$ has to be interpreted as $\phi_{i,j,k}\, I$, namely a 0-filled matrix with $\phi_{i,j,k}$ on the main diagonal. Such approach requires $m-2$ matrix products and $m-1$ additions. We use this scheme in all subsequent polynomial evaluations to a Riordan matrix.

In order to apply the functions described in the previous section to Riordan arrays $\mathcal{P}_8, \mathcal{C}_8$ and $\mathcal{S}_8$ concerning binomial coefficients, Catalan and Stirling numbers, we list the corresponding Hermite interpolating polynomials for the

r-th power function

$$\begin{aligned}P_8(z) &= (z-1)^7\binom{r}{7} + (z-1)^6\binom{r}{6} + (z-1)^5\binom{r}{5} + (z-1)^4\binom{r}{4}\\
&\quad + (z-1)^3\binom{r}{3} + (z-1)^2\binom{r}{2} + (z-1)\binom{r}{1} + \binom{r}{0}\\
&= z^7\binom{r}{7}\\
&\quad + z^6\left(\binom{r}{6} - 7\binom{r}{7}\right)\\
&\quad + z^5\left(\binom{r}{5} - 6\binom{r}{6} + 21\binom{r}{7}\right)\\
&\quad + z^4\left(\binom{r}{4} - 5\binom{r}{5} + 15\binom{r}{6} - 35\binom{r}{7}\right)\\
&\quad + z^3\left(\binom{r}{3} - 4\binom{r}{4} + 10\binom{r}{5} - 20\binom{r}{6} + 35\binom{r}{7}\right)\\
&\quad + z^2\left(\binom{r}{2} - 3\binom{r}{3} + 6\binom{r}{4} - 10\binom{r}{5} + 15\binom{r}{6} - 21\binom{r}{7}\right)\\
&\quad + z\left(\binom{r}{1} - 2\binom{r}{2} + 3\binom{r}{3} - 4\binom{r}{4} + 5\binom{r}{5} - 6\binom{r}{6} + 7\binom{r}{7}\right)\\
&\quad - \binom{r}{1} + \binom{r}{2} - \binom{r}{3} + \binom{r}{4} - \binom{r}{5} + \binom{r}{6} - \binom{r}{7} + 1;\end{aligned}$$

inverse function

$$\begin{aligned}I_8(z) &= -(z-1)^7 + (z-1)^6 - (z-1)^5 + (z-1)^4 - (z-1)^3 + (z-1)^2 - (z-1) + 1\\ &= -z^7 + 8z^6 - 28z^5 + 56z^4 - 70z^3 + 56z^2 - 28z + 8;\end{aligned}$$

square root function

$$\begin{aligned}R_8(z) &= \frac{33}{2048}(z-1)^7 - \frac{21}{1024}(z-1)^6 + \frac{7}{256}(z-1)^5 - \frac{5}{128}(z-1)^4\\ &+ \frac{1}{16}(z-1)^3 - \frac{1}{8}(z-1)^2 + \frac{1}{2}(z-1) + 1\\ &= \frac{33z^7}{2048} - \frac{273z^6}{2048} + \frac{1001z^5}{2048} - \frac{2145z^4}{2048} + \frac{3003z^3}{2048} - \frac{3003z^2}{2048} + \frac{3003z}{2048} + \frac{429}{2048};\end{aligned}$$

exponential function

$$\begin{aligned}E_8(z) &= e^{\alpha}\left(\frac{\alpha^7}{5040}(z-1)^7 + \frac{\alpha^6}{720}(z-1)^6 + \frac{\alpha^5}{120}(z-1)^5 + \frac{\alpha^4}{24}(z-1)^4\right.\\ &\quad\left.+ \frac{\alpha^3}{6}(z-1)^3 + \frac{\alpha^2}{2}(z-1)^2 + \alpha(z-1) + 1\right)\\ &= e^{\alpha}\left(\frac{\alpha^7 z^7}{5040}\right.\\ &\quad+ \frac{\alpha^6 z^6}{720}(-\alpha+1)\\ &\quad+ \frac{\alpha^5 z^5}{240}\left(\alpha^2 - 2\alpha + 2\right)\\ &\quad+ \frac{\alpha^4 z^4}{144}\left(-\alpha^3 + 3\alpha^2 - 6\alpha + 6\right)\\ &\quad+ \frac{\alpha^3 z^3}{144}\left(\alpha^4 - 4\alpha^3 + 12\alpha^2 - 24\alpha + 24\right)\\ &\quad+ \frac{\alpha^2 z^2}{240}\left(-\alpha^5 + 5\alpha^4 - 20\alpha^3 + 60\alpha^2 - 120\alpha + 120\right)\\ &\quad+ \frac{\alpha z}{720}\left(\alpha^6 - 6\alpha^5 + 30\alpha^4 - 120\alpha^3 + 360\alpha^2 - 720\alpha + 720\right)\\ &\quad\left.- \frac{\alpha^7}{5040} + \frac{\alpha^6}{720} - \frac{\alpha^5}{120} + \frac{\alpha^4}{24} - \frac{\alpha^3}{6} + \frac{\alpha^2}{2} - \alpha + 1\right),\\ E_8(z)|_{\alpha=1} &= e\left(\frac{z^7}{5040} + \frac{z^5}{240} + \frac{z^4}{72} + \frac{z^3}{16} + \frac{11z^2}{60} + \frac{53z}{144} + \frac{103}{280}\right) \quad \text{and}\\ E_8(z)|_{\alpha=-1} &= \frac{1}{e}\left(-\frac{z^7}{5040} + \frac{z^6}{360} - \frac{z^5}{48} + \frac{z^4}{9} - \frac{65z^3}{144} + \frac{163z^2}{120} - \frac{1957z}{720} + \frac{685}{252}\right);\end{aligned} \tag{2.15}$$

logarithm function

$$\begin{aligned}
L_8(z) &= \frac{1}{7}(z-1)^7 - \frac{1}{6}(z-1)^6 + \frac{1}{5}(z-1)^5 - \frac{1}{4}(z-1)^4 + \frac{1}{3}(z-1)^3 - \frac{1}{2}(z-1)^2 + (z-1) \\
&= \frac{z^7}{7} - \frac{7z^6}{6} + \frac{21z^5}{5} - \frac{35z^4}{4} + \frac{35z^3}{3} - \frac{21z^2}{2} + 7z - \frac{363}{140};
\end{aligned}$$

sine function

$$\begin{aligned}
S_8(z) &= -\frac{1}{5040}(z-1)^7\cos(1) - \frac{1}{720}(z-1)^6\sin(1) + \frac{1}{120}(z-1)^5\cos(1) + \frac{1}{24}(z-1)^4\sin(1) \\
&\quad - \frac{1}{6}(z-1)^3\cos(1) - \frac{1}{2}(z-1)^2\sin(1) + (z-1)\cos(1) + \sin(1) \\
&= \frac{1}{720}\left(-z^6 + 6z^5 + 15z^4 - 100z^3 - 195z^2 + 606z + 389\right)\sin(1) \\
&\quad + \frac{1}{5040}\left(-z^7 + 7z^6 + 21z^5 - 175z^4 - 455z^3 + 2121z^2 + 2723z - 4241\right)\cos(1) \\
&= -\frac{z^7}{5040}\cos(1) + z^6\left(-\frac{1}{720}\sin(1) + \frac{1}{720}\cos(1)\right) + z^5\left(\frac{1}{120}\sin(1) + \frac{1}{240}\cos(1)\right) \\
&\quad + z^4\left(\frac{1}{48}\sin(1) - \frac{5}{144}\cos(1)\right) + z^3\left(-\frac{5}{36}\sin(1) - \frac{13}{144}\cos(1)\right) \\
&\quad + z^2\left(-\frac{13}{48}\sin(1) + \frac{101}{240}\cos(1)\right) + z\left(\frac{101}{120}\sin(1) + \frac{389}{720}\cos(1)\right) \\
&\quad + \frac{389}{720}\sin(1) - \frac{4241}{5040}\cos(1);
\end{aligned}$$

cosine function

$$\begin{aligned}
C_8(z) &= \frac{1}{5040}(z-1)^7\sin(1) - \frac{1}{720}(z-1)^6\cos(1) - \frac{1}{120}(z-1)^5\sin(1) + \frac{1}{24}(z-1)^4\cos(1) \\
&\quad + \frac{1}{6}(z-1)^3\sin(1) - \frac{1}{2}(z-1)^2\cos(1) - (z-1)\sin(1) + \cos(1) \\
&= \frac{1}{720}\left(-z^6 + 6z^5 + 15z^4 - 100z^3 - 195z^2 + 606z + 389\right)\cos(1) \\
&\quad + \frac{1}{5040}\left(z^7 - 7z^6 - 21z^5 + 175z^4 + 455z^3 - 2121z^2 - 2723z + 4241\right)\sin(1) \\
&= \frac{z^7}{5040}\sin(1) + z^6\left(-\frac{1}{720}\sin(1) - \frac{1}{720}\cos(1)\right) + z^5\left(-\frac{1}{240}\sin(1) + \frac{1}{120}\cos(1)\right) \\
&\quad + z^4\left(\frac{5}{144}\sin(1) + \frac{1}{48}\cos(1)\right) + z^3\left(\frac{13}{144}\sin(1) - \frac{5}{36}\cos(1)\right) \\
&\quad + z^2\left(-\frac{101}{240}\sin(1) - \frac{13}{48}\cos(1)\right) + z\left(-\frac{389}{720}\sin(1) + \frac{101}{120}\cos(1)\right) \\
&\quad + \frac{4241}{5040}\sin(1) + \frac{389}{720}\cos(1).
\end{aligned}$$

Example 26. *Let $\mathcal{P}$ be the matrix of binomial coefficients, also known as the Pascal matrix,*

$$\mathcal{P}_8 = \begin{bmatrix} 1 &&&&&&& \\ 1 & 1 &&&&&& \\ 1 & 2 & 1 &&&&& \\ 1 & 3 & 3 & 1 &&&& \\ 1 & 4 & 6 & 4 & 1 &&& \\ 1 & 5 & 10 & 10 & 5 & 1 && \\ 1 & 6 & 15 & 20 & 15 & 6 & 1 & \\ 1 & 7 & 21 & 35 & 35 & 21 & 7 & 1 \end{bmatrix}$$

where $\mathcal{P} = \left(\frac{1}{1-t}, \frac{t}{1-t}\right)$. Then, the application of Hermite interpolating polynomials yields the following matrices:

$$\mathcal{P}_8^r = P_8\left(\mathcal{P}_8\right) = \begin{bmatrix} 1 &&&&&&& \\ r & 1 &&&&&& \\ r^2 & 2r & 1 &&&&& \\ r^3 & 3r^2 & 3r & 1 &&&& \\ r^4 & 4r^3 & 6r^2 & 4r & 1 &&& \\ r^5 & 5r^4 & 10r^3 & 10r^2 & 5r & 1 && \\ r^6 & 6r^5 & 15r^4 & 20r^3 & 15r^2 & 6r & 1 & \\ r^7 & 7r^6 & 21r^5 & 35r^4 & 35r^3 & 21r^2 & 7r & 1 \end{bmatrix}$$

the special cases $r = \frac{1}{2}$ and $r = \frac{1}{3}$ have been illustrated in Section 6.2 while $r = 2$ and $r = -1$ yield

$$\mathcal{P}_8^2 = \begin{bmatrix} 1 &&&&&&& \\ 2 & 1 &&&&&& \\ 4 & 4 & 1 &&&&& \\ 8 & 12 & 6 & 1 &&&& \\ 16 & 32 & 24 & 8 & 1 &&& \\ 32 & 80 & 80 & 40 & 10 & 1 && \\ 64 & 192 & 240 & 160 & 60 & 12 & 1 & \\ 128 & 448 & 672 & 560 & 280 & 84 & 14 & 1 \end{bmatrix}$$

where $\mathcal{P}^2 = \mathcal{R}\left(\frac{1}{1-2t}, \frac{t}{1-2t}\right)$, *and*

$$\mathcal{P}_8^{-1} = I_8\left(\mathcal{P}_8\right) = \begin{bmatrix} 1 &&&&&&& \\ -1 & 1 &&&&&& \\ 1 & -2 & 1 &&&&& \\ -1 & 3 & -3 & 1 &&&& \\ 1 & -4 & 6 & -4 & 1 &&& \\ -1 & 5 & -10 & 10 & -5 & 1 && \\ 1 & -6 & 15 & -20 & 15 & -6 & 1 & \\ -1 & 7 & -21 & 35 & -35 & 21 & -7 & 1 \end{bmatrix}$$

where $\mathcal{P}^{-1} = \mathcal{R}\left(\frac{1}{1+t}, \frac{t}{1+t}\right)$, *corresponding to the product and inverse operations in the Riordan group defined in Equations 1.3 and 1.4, respectively. Additionally, matrices* $e^{\mathcal{P}_8} = E_8\left(\mathcal{P}_8\right)$, *which is known as* A056857 *in the Online Encyclopedia of Integer Sequences [Sloane], and* $\log\mathcal{P}_8 = L_8\left(\mathcal{P}_8\right)$ *defined by*

$$e^{\mathcal{P}_8} = e\begin{bmatrix} 1 &&&&&&& \\ 1 & 1 &&&&&& \\ 2 & 2 & 1 &&&&& \\ 5 & 6 & 3 & 1 &&&& \\ 15 & 20 & 12 & 4 & 1 &&& \\ 52 & 75 & 50 & 20 & 5 & 1 && \\ 203 & 312 & 225 & 100 & 30 & 6 & 1 & \\ 877 & 1421 & 1092 & 525 & 175 & 42 & 7 & 1 \end{bmatrix}$$

$$\textit{and} \quad \log\mathcal{P}_8 = \begin{bmatrix} 0 &&&&&&& \\ 1 & 0 &&&&&& \\ & 2 & 0 &&&&& \\ && 3 & 0 &&&& \\ &&& 4 & 0 &&& \\ &&&& 5 & 0 && \\ &&&&& 6 & 0 & \\ &&&&&& 7 & 0 \end{bmatrix}$$

have eigenvalues e *and* 0; *therefore, in order to check the (expected) identities* $\log\left(e^{\mathcal{P}_8}\right) = e^{\log\mathcal{P}_8} = \mathcal{P}_8$ *it is required to compute new Hermite interpolating polynomials using Theorem 17 on eigenvalues* $\lambda_1 = 0$ *and* $\lambda_1 = e$, *in place of* $L_8(z)$ *and* $E_8(z)$ *which depend on eigenvalue* $\lambda = 1$ *instead.*

Remark 27. *For the sake of completeness, in order to recover $\mathcal{P}_8$ back from $\log\mathcal{P}_8$ we have to (i) to find its spectrum*

$$\sigma(L_8(\mathcal{P}_8)) = (\{1 : (\lambda_1, \quad m_1)\}, \quad \{\lambda_1 : 0\}, \quad \{m_1 : 8\}),$$

(ii) to compute the generalized Lagrange base

$$\Phi_{1,1}(z) = 1, \Phi_{1,2}(z) = z, \Phi_{1,3}(z) = \frac{z^2}{2}, \Phi_{1,4}(z) = \frac{z^3}{6},$$
$$\Phi_{1,5}(z) = \frac{z^4}{24}, \Phi_{1,6}(z) = \frac{z^5}{120}, \Phi_{1,7}(z) = \frac{z^6}{720}, \Phi_{1,8}(z) = \frac{z^7}{5040}$$

and (iii) to build the Hermite interpolating polynomial

$$E_8(z) = \frac{\alpha^7 z^7}{5040} + \frac{\alpha^6 z^6}{720} + \frac{\alpha^5 z^5}{120} + \frac{\alpha^4 z^4}{24} + \frac{\alpha^3 z^3}{6} + \frac{\alpha^2 z^2}{2} + \alpha z + 1$$

that interpolates the function $f(z) = e^{\alpha z}$, which is different from the corresponding polynomials show in Equation 2.15 ; finally, $\alpha = 1$ closes.

Example 28. *Let $\mathcal{C}$ be the matrix of Catalan numbers,*

$$\mathcal{C}_8 = \begin{bmatrix} 1 & & & & & & & \\ 1 & 1 & & & & & & \\ 2 & 2 & 1 & & & & & \\ 5 & 5 & 3 & 1 & & & & \\ 14 & 14 & 9 & 4 & 1 & & & \\ 42 & 42 & 28 & 14 & 5 & 1 & & \\ 132 & 132 & 90 & 48 & 20 & 6 & 1 & \\ 429 & 429 & 297 & 165 & 75 & 27 & 7 & 1 \end{bmatrix}$$

where $\mathcal{C} = \left(\frac{1-\sqrt{1-4t}}{2t}, \frac{1-\sqrt{1-4t}}{2}\right)$. Then, the application of Hermite interpolating polynomials yields matrices

$$\mathcal{C}_8^r \boldsymbol{e}_1 = P_8(\mathcal{C}_8)\, \boldsymbol{e}_1 = \begin{bmatrix} 1 \\ r \\ r(r+1) \\ \frac{r}{2}\left(2r^2 + 5r + 3\right) \\ \frac{r}{3}\left(3r^3 + 13r^2 + 18r + 8\right) \\ \frac{r}{12}\left(12r^4 + 77r^3 + 178r^2 + 175r + 62\right) \\ \frac{r}{30}\left(30r^5 + 261r^4 + 875r^3 + 1405r^2 + 1075r + 314\right) \\ \frac{r}{60}\left(60r^6 + 669r^5 + 3002r^4 + 6900r^3 + 8510r^2 + 5301r + 1298\right) \end{bmatrix},$$

$$\mathcal{C}_8^{-1} = I_8\left(\mathcal{C}_8\right) = \begin{bmatrix} 1 & & & & & & & \\ -1 & 1 & & & & & & \\ & -2 & 1 & & & & & \\ & 1 & -3 & 1 & & & & \\ & & 3 & -4 & 1 & & & \\ & & -1 & 6 & -5 & 1 & & \\ & & & -4 & 1 & -6 & 1 & \\ & & & 1 & -1 & 15 & -7 & 1 \end{bmatrix},$$

$$\sqrt{\mathcal{C}_8} = R_8\left(\mathcal{C}_8\right) = \begin{bmatrix} 1 & & & & & & & \\ \frac{1}{2} & 1 & & & & & & \\ \frac{3}{4} & 1 & 1 & & & & & \\ \frac{3}{2} & \frac{7}{4} & \frac{3}{2} & 1 & & & & \\ \frac{55}{16} & \frac{15}{4} & 3 & 2 & 1 & & & \\ \frac{545}{64} & \frac{143}{16} & \frac{55}{8} & \frac{9}{2} & \frac{5}{2} & 1 & & \\ \frac{79}{32} & \frac{727}{32} & \frac{273}{16} & 11 & \frac{25}{4} & 3 & 1 & \\ \frac{15249}{256} & \frac{3855}{64} & \frac{2853}{64} & \frac{455}{16} & \frac{65}{4} & \frac{33}{4} & \frac{7}{2} & 1 \end{bmatrix} \quad \textit{and}$$

$$\log\mathcal{C}_8 = L_8\left(\mathcal{C}_8\right) = \begin{bmatrix} 0 & & & & & & & \\ 1 & 0 & & & & & & \\ 1 & 2 & 0 & & & & & \\ \frac{3}{2} & 2 & 3 & 0 & & & & \\ \frac{8}{3} & 3 & 3 & 4 & 0 & & & \\ \frac{31}{6} & \frac{16}{3} & \frac{9}{2} & 4 & 5 & 0 & & \\ \frac{157}{15} & \frac{31}{3} & 8 & 6 & 5 & 6 & 0 & \\ \frac{649}{3} & \frac{314}{15} & \frac{31}{2} & \frac{32}{3} & \frac{15}{2} & 6 & 7 & 0 \end{bmatrix},$$

where the r*-th power* $\mathcal{C}_8^r$ *is a rather complex matrix of which we report the first column only, formally multiplying on the right by indicator vector* $\mathbf{e}_1 = [1, 0, \ldots, 0]$.

Example 29. *Let* $\mathcal{S}$ *be the matrix of Stirling numbers of the second kind,*

$$\mathcal{S}_8 = \begin{bmatrix} 1 & & & & & & & \\ 1 & 1 & & & & & & \\ 1 & 3 & 1 & & & & & \\ 1 & 7 & 6 & 1 & & & & \\ 1 & 15 & 25 & 10 & 1 & & & \\ 1 & 31 & 90 & 65 & 15 & 1 & & \\ 1 & 63 & 301 & 350 & 140 & 21 & 1 & \\ 1 & 127 & 966 & 1701 & 1050 & 266 & 28 & 1 \end{bmatrix}$$

where $d_{n,k} \in \mathcal{S} \leftrightarrow d_{n,k} = \dfrac{n!}{k!}[t^n]e^t(e^t - 1)^k$. *Then, the appli-*

cation of Hermite interpolating polynomials yields matrices

$$\mathcal{S}_8^r \boldsymbol{e}_1 = P_8(\mathcal{S}_8)\boldsymbol{e}_1 = \begin{bmatrix} 1 \\ r \\ \frac{r}{2}(3r-1) \\ \frac{r}{2}\left(6r^2-5r+1\right) \\ \frac{r}{6}\left(45r^3-65r^2+30r-4\right) \\ \frac{r}{24}\left(540r^4-1155r^3+890r^2-273r+22\right) \\ \frac{r}{24}\left(1890r^5-5481r^4+6125r^3-3129r^2+637r-18\right) \\ \frac{r}{12}\left(3780r^6-14049r^5+21014r^4-15540r^3+5474r^2-645r-22\right) \end{bmatrix},$$

$$\mathcal{S}_8^{-1} = I_8(\mathcal{S}_8) = \begin{bmatrix} 1 \\ -1 & 1 \\ 2 & -3 & 1 \\ -6 & 11 & -6 & 1 \\ 24 & -50 & 35 & -10 & 1 \\ -120 & 274 & -225 & 85 & -15 & 1 \\ 720 & -1764 & 1624 & -735 & 175 & -21 & 1 \\ -5040 & 13068 & -13132 & 6769 & -1960 & 322 & -28 & 1 \end{bmatrix},$$

$$\sqrt{\mathcal{S}_8} = R_8(\mathcal{S}_8) = \begin{bmatrix} 1 \\ \frac{1}{2} & 1 \\ \frac{1}{8} & \frac{3}{2} & 1 \\ 0 & \frac{5}{4} & 3 & 1 \\ \frac{1}{32} & \frac{5}{8} & 5 & 5 & 1 \\ -\frac{7}{128} & \frac{11}{32} & \frac{45}{8} & \frac{55}{4} & \frac{15}{2} & 1 \\ \frac{1}{128} & -\frac{7}{128} & \frac{161}{32} & \frac{105}{4} & \frac{245}{8} & \frac{21}{2} & 1 \\ \frac{159}{256} & -\frac{31}{64} & \frac{105}{32} & \frac{623}{16} & \frac{175}{2} & \frac{119}{2} & 14 & 1 \end{bmatrix},$$

$$e^{\mathcal{S}_8} = E_8\left(\mathcal{S}_8\right) = e \begin{bmatrix} 1 \\ 1 & 1 \\ \frac{5}{2} & 3 & 1 \\ \frac{21}{2} & 16 & 6 & 1 \\ \frac{203}{3} & \frac{235}{2} & 55 & 10 & 1 \\ \frac{14681}{24} & 1176 & \frac{1245}{2} & 140 & 15 & 1 \\ \frac{22018}{3} & \frac{367745}{24} & 8911 & \frac{4515}{2} & \frac{595}{2} & 21 & 1 \\ \frac{1348799}{12} & \frac{3014485}{12} & \frac{946043}{6} & \frac{131173}{3} & 6475 & 560 & 28 & 1 \end{bmatrix} \quad \textit{and}$$

$$\log \mathcal{S}_8 = L_8(\mathcal{S}_8) = \begin{bmatrix} 0 & & & & & & & \\ 1 & 0 & & & & & & \\ -\frac{1}{2} & 3 & 0 & & & & & \\ \frac{1}{2} & -2 & 6 & 0 & & & & \\ -\frac{2}{3} & \frac{5}{2} & -5 & 10 & 0 & & & \\ \frac{11}{12} & -4 & \frac{15}{2} & -10 & 15 & 0 & & \\ -\frac{3}{4} & \frac{77}{12} & -14 & \frac{35}{2} & -\frac{35}{2} & 21 & 0 & \\ -\frac{11}{6} & -6 & \frac{77}{3} & -\frac{112}{3} & 35 & -28 & 28 & 0 \end{bmatrix}.$$

The matrix $\mathcal{S}_8$ is related to matrix $e^{\mathcal{P}_8}$ by the identity $e^{\mathcal{P}_8} = e \cdot \left(\mathcal{S}_8 \cdot \mathcal{P}_8 \cdot \mathcal{S}_8^{-1}\right)$ and even more connections involving these matrices can be found in [Cheon and Kim, 2001].

Finally, we report sine and cosine function applications in Table 2.1; finally, Equation 2.1 shows that the identity $\sin^2 z + \cos^2 z = 1$ is preserved by the framework of matrices functions and even more trigonometric identities can be checked; for example, the polynomial

$$\begin{aligned} SS_8(z) = &-\frac{z^7}{5040}\cos(2) + z^6\left(-\frac{1}{720}\sin(2) + \frac{1}{360}\cos(2)\right) \\ &+ z^5\left(-\frac{1}{120}\cos(2) + \frac{1}{60}\sin(2)\right) + z^4\left(-\frac{1}{24}\sin(2) - \frac{1}{36}\cos(2)\right) \\ &+ z^3\left(-\frac{1}{9}\sin(2) + \frac{1}{18}\cos(2)\right) + z^2\left(\frac{7}{15}\cos(2) + \frac{1}{6}\sin(2)\right) \\ &+ z\left(-\frac{19}{45}\cos(2) + \frac{14}{15}\sin(2)\right) - \frac{19}{45}\sin(2) - \frac{286}{315}\cos(2) \end{aligned}$$

interpolates the function $f(\theta) = \sin(2\theta)$ which allows us to check the identity $\sin(2\theta) = 2\sin\theta\cos\theta$ for a Riordan matrix θ, formally $SS(\theta) = 2S(\theta)C(\theta)$.

2.5 *Jordan canonical form*

We begin this section with necessary definitions about Jordan canonical forms to help the computation of matrices functions.

Let $A \in \mathbb{R}^{m\times m}$ be a square matrix and $\Phi_{i,j} \in \prod_{m-1}$ a generalized Lagrange base, so $Z_{i,j}^{[A]} = \Phi_{i,j}(A)$ denotes the j-th *component matrix* of A relative to its i-th eigenvalue (from here on, we just write $Z_{i,j}$ to keep clean the notation when no confusion arises).

Component matrices enjoy the properties

- they are linearly independent and don't depend on function f,
- they commute respect the product, $Z_{ij}Z_{kr} = Z_{kr}Z_{ij}$,

$$\sin\mathcal{P}_8 = S_8(\mathcal{P}_8) = \begin{bmatrix}
\sin 1 &&&&&&& \\
\cos 1 & \sin 1 &&&&&& \\
-\sin 1+\cos 1 & 2\cos 1 & \sin 1 &&&&& \\
-3\sin 1 & 3\sqrt{2}\cos\left(\frac{\pi}{4}+1\right) & 3\cos 1 & \sin 1 &&&& \\
-6\sin 1-5\cos 1 & -12\sin 1 & 6\sqrt{2}\cos\left(\frac{\pi}{4}+1\right) & 4\cos 1 & \sin 1 &&& \\
-23\cos 1-5\sin 1 & -30\sin 1-25\cos 1 & -30\sin 1 & 10\sqrt{2}\cos\left(\frac{\pi}{4}+1\right) & 5\cos 1 & \sin 1 && \\
-74\cos 1+33\sin 1 & -138\cos 1-30\sin 1 & -90\sin 1-75\cos 1 & -60\sin 1 & 15\sqrt{2}\cos\left(\frac{\pi}{4}+1\right) & 6\cos 1 & \sin 1 & \\
-161\cos 1+266\sin 1 & -518\cos 1+231\sin 1 & -483\cos 1-105\sin 1 & -210\sin 1-175\cos 1 & -105\sin 1 & 21\sqrt{2}\cos\left(\frac{\pi}{4}+1\right) & 7\cos 1 & \sin 1
\end{bmatrix}$$

$$\cos\mathcal{P}_8 = C_8(\mathcal{P}_8) = \begin{bmatrix}
\cos 1 &&&&&&& \\
-\sin 1 & \cos 1 &&&&&& \\
-\sin 1-\cos 1 & -2\sin 1 & \cos 1 &&&&& \\
-3\cos 1 & -3\sqrt{2}\sin\left(\frac{\pi}{4}+1\right) & -3\sin 1 & \cos 1 &&&& \\
-6\cos 1+5\sin 1 & -12\cos 1 & -6\sqrt{2}\sin\left(\frac{\pi}{4}+1\right) & -4\sin 1 & \cos 1 &&& \\
-5\cos 1+23\sin 1 & -30\cos 1+25\sin 1 & -30\cos 1 & -10\sqrt{2}\sin\left(\frac{\pi}{4}+1\right) & -5\sin 1 & \cos 1 && \\
33\cos 1+74\sin 1 & -30\cos 1+138\sin 1 & -90\cos 1+75\sin 1 & -60\cos 1 & -15\sqrt{2}\sin\left(\frac{\pi}{4}+1\right) & -6\sin 1 & \cos 1 & \\
161\sin 1+266\cos 1 & 231\cos 1+518\sin 1 & -105\cos 1+483\sin 1 & -210\cos 1+175\sin 1 & -105\cos 1 & -21\sqrt{2}\sin\left(\frac{\pi}{4}+1\right) & -7\sin 1 & \cos 1
\end{bmatrix}$$

$$\sin\mathcal{C}_8 = S_8(\mathcal{C}_8) = \begin{bmatrix}
\sin 1 &&&&&&& \\
\cos 1 & \sin 1 &&&&&& \\
-\sin 1+2\cos 1 & 2\cos 1 & \sin 1 &&&&& \\
-\frac{11}{2}\sin 1+4\cos 1 & -3\sin 1+5\cos 1 & 3\cos 1 & \sin 1 &&&& \\
-25\sin 1+\frac{11}{3}\cos 1 & -19\sin 1+10\cos 1 & -6\sin 1+9\cos 1 & 4\cos 1 & \sin 1 &&& \\
-\frac{1231}{12}\sin 1-\frac{106}{3}\cos 1 & -93\sin 1-\frac{11}{3}\cos 1 & -\frac{87}{2}\sin 1+18\cos 1 & -10\sin 1+14\cos 1 & 5\cos 1 & \sin 1 && \\
-\frac{2171}{6}\sin 1-\frac{11209}{30}\cos 1 & -\frac{775}{2}\sin 1-\frac{698}{3}\cos 1 & -231\sin 1-37\cos 1 & -82\sin 1+28\cos 1 & -15\sin 1+20\cos 1 & 6\cos 1 & \sin 1 & \\
-\frac{156113}{60}\cos 1-\frac{12301}{15}\sin 1 & -\frac{61583}{30}\cos 1-\frac{14863}{12}\sin 1 & -\frac{3953}{4}\sin 1-\frac{1595}{2}\cos 1 & -472\sin 1-\frac{349}{3}\cos 1 & -\frac{275}{2}\sin 1+40\cos 1 & -21\sin 1+27\cos 1 & 7\cos 1 & \sin 1
\end{bmatrix}$$

$$\cos\mathcal{C}_8 = C_8(\mathcal{C}_8) = \begin{bmatrix}
\cos 1 &&&&&&& \\
-\sin 1 & \cos 1 &&&&&& \\
-2\sin 1-\cos 1 & -2\sin 1 & \cos 1 &&&&& \\
-4\sin 1-\frac{11}{2}\cos 1 & -5\sin 1-3\cos 1 & -3\sin 1 & \cos 1 &&&& \\
-25\cos 1-\frac{11}{3}\sin 1 & -19\cos 1-10\sin 1 & -9\sin 1-6\cos 1 & -4\sin 1 & \cos 1 &&& \\
-\frac{1231}{12}\cos 1+\frac{106}{3}\sin 1 & -93\cos 1+\frac{11}{3}\sin 1 & -\frac{87}{2}\cos 1-18\sin 1 & -14\sin 1-10\cos 1 & -5\sin 1 & \cos 1 && \\
-\frac{2171}{6}\cos 1+\frac{11209}{30}\sin 1 & -\frac{775}{2}\cos 1+\frac{698}{3}\sin 1 & -231\cos 1+37\sin 1 & -82\cos 1-28\sin 1 & -20\sin 1-15\cos 1 & -6\sin 1 & \cos 1 & \\
-\frac{12301}{15}\cos 1+\frac{156113}{60}\sin 1 & -\frac{14863}{12}\cos 1+\frac{61583}{30}\sin 1 & -\frac{3953}{4}\cos 1+\frac{1595}{2}\sin 1 & -472\cos 1+\frac{349}{3}\sin 1 & -\frac{275}{2}\cos 1-40\sin 1 & -27\sin 1-21\cos 1 & -7\sin 1 & \cos 1
\end{bmatrix}$$

$$\sin\mathcal{S}_8 = S_8(\mathcal{S}_8) = \begin{bmatrix}
\sin 1 &&&&&&& \\
\cos 1 & \sin 1 &&&&&& \\
-\frac{3}{2}\sin 1+\cos 1 & 3\cos 1 & \sin 1 &&&&& \\
-\frac{13}{2}\sin 1-2\cos 1 & -9\sin 1+7\cos 1 & 6\cos 1 & \sin 1 &&&& \\
-\frac{199}{6}\cos 1-\frac{35}{2}\sin 1 & -\frac{145}{2}\sin 1-15\cos 1 & -30\sin 1+25\cos 1 & 10\cos 1 & \sin 1 &&& \\
-\frac{862}{3}\cos 1+\frac{611}{8}\sin 1 & -\frac{725}{2}\sin 1-\frac{1053}{2}\cos 1 & -\frac{765}{2}\sin 1-60\cos 1 & -75\sin 1+65\cos 1 & 15\cos 1 & \sin 1 && \\
-\frac{14601}{8}\cos 1+\frac{61775}{24}\sin 1 & -\frac{43337}{6}\cos 1+\frac{7399}{8}\sin 1 & -\frac{5915}{2}\sin 1-\frac{7553}{2}\cos 1 & -\frac{2765}{2}\sin 1-175\cos 1 & -\frac{315}{2}\sin 1+140\cos 1 & 21\cos 1 & \sin 1 & \\
\frac{24757}{12}\cos 1+\frac{128564}{3}\sin 1 & -\frac{145395}{2}\cos 1+\frac{921235}{12}\sin 1 & -\frac{221977}{3}\cos 1+\frac{8211}{2}\sin 1 & -15120\sin 1-\frac{53557}{3}\cos 1 & -3955\sin 1-420\cos 1 & -294\sin 1+266\cos 1 & 28\cos 1 & \sin 1
\end{bmatrix}$$

$$\cos\mathcal{S}_8 = C_8(\mathcal{S}_8) = \begin{bmatrix}
\cos 1 &&&&&&& \\
-\sin 1 & \cos 1 &&&&&& \\
-\sin 1-\frac{3}{2}\cos 1 & -3\sin 1 & \cos 1 &&&&& \\
-\frac{13}{2}\cos 1+2\sin 1 & -7\sin 1-9\cos 1 & -6\sin 1 & \cos 1 &&&& \\
-\frac{35}{2}\cos 1+\frac{199}{6}\sin 1 & -\frac{145}{2}\cos 1+15\sin 1 & -25\sin 1-30\cos 1 & -10\sin 1 & \cos 1 &&& \\
\frac{611}{8}\cos 1+\frac{862}{3}\sin 1 & -\frac{725}{2}\cos 1+\frac{1053}{2}\sin 1 & -\frac{765}{2}\cos 1+60\sin 1 & -65\sin 1-75\cos 1 & -15\sin 1 & \cos 1 && \\
\frac{61775}{24}\cos 1+\frac{14601}{8}\sin 1 & \frac{7399}{8}\cos 1+\frac{43337}{6}\sin 1 & -\frac{5915}{2}\cos 1+\frac{7553}{2}\sin 1 & -\frac{2765}{2}\cos 1+175\sin 1 & -140\sin 1-\frac{315}{2}\cos 1 & -21\sin 1 & \cos 1 & \\
-\frac{24757}{12}\sin 1+\frac{128564}{3}\cos 1 & \frac{921235}{12}\cos 1+\frac{145395}{2}\sin 1 & \frac{8211}{2}\cos 1+\frac{221977}{3}\sin 1 & -15120\cos 1+\frac{53557}{3}\sin 1 & -3955\cos 1+420\sin 1 & -266\sin 1-294\cos 1 & -28\sin 1 & \cos 1
\end{bmatrix}$$

Table 2.1: $\sin\mathcal{P}_8$,$\cos\mathcal{P}_8$,$\sin\mathcal{C}_8$,$\cos\mathcal{C}_8$, $\sin\mathcal{S}_8$ and $\cos\mathcal{S}_8$

- $Z_{ij}Z_{kr} = O$ if $i \neq k$,
- $Z_{i1}Z_{ij} = Z_{ij}$ if $j > 0$,
- $Z_{i2}Z_{ij} = jZ_{i,j+1}$ if $j > 0$,
- $Z_{ij} = \frac{Z_{i2}^{j-1}}{(j-1)!}$ if $j > 1$,
- $Z_{i2}^{m_i} = O$ for $i \in \{1, \dots, \nu\}$,
- $I = \sum_{i=1}^{\nu} Z_{i1}$ from $f(z) = 1$, and
- $Z_{i2} = Z_{i1}(A - \lambda_i I)$ from $f(z) = z - \lambda_i$, for $i \in \{1, \dots, \nu\}$,

whose proofs can be found in [Brugnano and Trigiante, 1998, Lakshmikantham and Trigiante, 2002] together with more arguments and applications.

Example 30. *Component matrices relative to $\mathcal{P}_8$ are*

$$Z_{1,1} = \begin{bmatrix} 1 & & & & & & & \\ & 1 & & & & & & \\ & & 1 & & & & & \\ & & & 1 & & & & \\ & & & & 1 & & & \\ & & & & & 1 & & \\ & & & & & & 1 & \\ & & & & & & & 1 \end{bmatrix},$$

$$Z_{1,2} = \begin{bmatrix} & & & & & & & \\ 1 & & & & & & & \\ 1 & 2 & & & & & & \\ 1 & 3 & 3 & & & & & \\ 1 & 4 & 6 & 4 & & & & \\ 1 & 5 & 10 & 10 & 5 & & & \\ 1 & 6 & 15 & 20 & 15 & 6 & & \\ 1 & 7 & 21 & 35 & 35 & 21 & 7 & \end{bmatrix},$$

$$Z_{1,3} = \begin{bmatrix} & & & & & & & \\ & & & & & & & \\ 1 & & & & & & & \\ 3 & 3 & & & & & & \\ 7 & 12 & 6 & & & & & \\ 15 & 35 & 30 & 10 & & & & \\ 31 & 90 & 105 & 60 & 15 & & & \\ 63 & 217 & 315 & 245 & 105 & 21 & & \end{bmatrix},$$

$$Z_{1,4} = \begin{bmatrix} &&&&&&& \\ &&&&&&& \\ &&&&&&& \\ 1 &&&&&&& \\ 6 & 4 &&&&&& \\ 25 & 30 & 10 &&&&& \\ 90 & 150 & 90 & 20 &&&& \\ 301 & 630 & 525 & 210 & 35 &&& \end{bmatrix},$$

$$Z_{1,5} = \begin{bmatrix} &&&&&&& \\ &&&&&&& \\ &&&&&&& \\ &&&&&&& \\ 1 &&&&&&& \\ 10 & 5 &&&&&& \\ 65 & 60 & 15 &&&&& \\ 350 & 455 & 210 & 35 &&&& \end{bmatrix},$$

$$Z_{1,6} = \begin{bmatrix} &&&&&&& \\ &&&&&&& \\ &&&&&&& \\ &&&&&&& \\ &&&&&&& \\ 1 &&&&&&& \\ 15 & 6 &&&&&& \\ 140 & 105 & 21 &&&&& \end{bmatrix},$$

$$Z_{1,7} = \begin{bmatrix} &&&&&&& \\ &&&&&&& \\ &&&&&&& \\ &&&&&&& \\ &&&&&&& \\ &&&&&&& \\ 1 &&&&&&& \\ 21 & 7 &&&&&& \end{bmatrix} \quad \textit{and} \quad Z_{1,8} = \begin{bmatrix} &&&&&&& \\ &&&&&&& \\ &&&&&&& \\ &&&&&&& \\ &&&&&&& \\ &&&&&&& \\ &&&&&&& \\ 1 &&&&&&& \end{bmatrix};$$

in parallel, corresponding component matrices for $\mathcal{C}_8$ and $\mathcal{S}_8$ can be computed in a similar way.

Let $v \in \mathbb{R}^m$ be a *non-zero* vector to define a set of subspaces

$$\mathcal{M}_i = \left\{ x_{i,j} = Z_{i,2}^{j-1}\, Z_{i,1}\, v,\ j \in \{1, \ldots, m_i\} \right\}, \quad i \in \{1, \ldots, \nu\},$$

where $\mathrm{dim}(\mathcal{M}_i) = m_i$; moreover, vectors $x_{i,j}$ are linearly independent, therefore $\mathcal{M}_q \cap \mathcal{M}_w = \emptyset$ if $q \neq w$.

Lemma 31. *Let $\lambda_i \in \sigma(A)$, then vectors $x_{i,j} \in \mathcal{M}_i$ satisfy the recurrences*

$$\begin{aligned} A\, x_{i,j} &= \lambda_i\, x_{i,j} + x_{i,j+1}, \quad j \in \{1, \ldots, m_i - 1\} \\ A\, x_{i,m_i} &= \lambda_i\, x_{i,m_i}; \end{aligned}$$

for this reason, they are also called generalized eigenvectors.

Proof. Recall that $Z_{i2} = Z_{i,1}\,(A - \lambda_i I)$, then let $j \in \{1, \ldots, m_i - 1\}$ in

$$\begin{aligned} x_{i,j+1} &= Z_{i,2}^{j}\, Z_{i,1}\, v = Z_{i,2}^{j-1}\, Z_{i,2}\, Z_{i,1}\, v \\ &= Z_{i,2}^{j-1}\, Z_{i,1}\, Z_{i,2}\, v = Z_{i,2}^{j-1}\, Z_{i,1}\, Z_{i,1}\,(A - \lambda_i I)\, v \\ &= Z_{i,2}^{j-1}\, Z_{i,1}\,(A - \lambda_i I)\, v = (A - \lambda_i I)\, Z_{i,2}^{j-1}\, Z_{i,1}\, v \\ &= (A - \lambda_i I)\, x_{i,j} \end{aligned}$$

where the facts (i) $Z_{ij}Z_{kr} = Z_{kr}Z_{ij}$, (ii) $Z_{i1}Z_{ij} = Z_{ij}$ and (iii) the linear independence of component matrices justify the derivation steps in the branches above, respectively. This proves the first recurrence; on the other hand, consider

$$Z_{i,2}\, x_{i,m_i} = Z_{i,2}\, Z_{i,2}^{m_i-1}\, Z_{i,1}\, v = Z_{i,2}^{m_i}\, Z_{i,1}\, v = 0$$

which holds because $Z_{i,2}^{m_i} = O$, proving the second recurrence. □

Example 32. *$\mathcal{P}_8$'s component matrices are used to build the set of generalized eigenvectors*

$$x_{1,1} = \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \\ \alpha_6 \\ \alpha_7 \end{bmatrix}, \quad x_{1,2} = \begin{bmatrix} 0 \\ \alpha_0 \\ \alpha_0 + 2\alpha_1 \\ \alpha_0 + 3\alpha_1 + 3\alpha_2 \\ \alpha_0 + 4\alpha_1 + 6\alpha_2 + 4\alpha_3 \\ \alpha_0 + 5\alpha_1 + 10\alpha_2 + 10\alpha_3 + 5\alpha_4 \\ \alpha_0 + 6\alpha_1 + 15\alpha_2 + 20\alpha_3 + 15\alpha_4 + 6\alpha_5 \\ \alpha_0 + 7\alpha_1 + 21\alpha_2 + 35\alpha_3 + 35\alpha_4 + 21\alpha_5 + 7\alpha_6 \end{bmatrix},$$

$$x_{1,3} = \begin{bmatrix} 0 \\ 0 \\ 2\alpha_0 \\ 6\alpha_0 + 6\alpha_1 \\ 14\alpha_0 + 24\alpha_1 + 12\alpha_2 \\ 30\alpha_0 + 70\alpha_1 + 60\alpha_2 + 20\alpha_3 \\ 62\alpha_0 + 180\alpha_1 + 210\alpha_2 + 120\alpha_3 + 30\alpha_4 \\ 126\alpha_0 + 434\alpha_1 + 630\alpha_2 + 490\alpha_3 + 210\alpha_4 + 42\alpha_5 \end{bmatrix},$$

$$x_{1,4} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 6\alpha_0 \\ 36\alpha_0 + 24\alpha_1 \\ 150\alpha_0 + 180\alpha_1 + 60\alpha_2 \\ 540\alpha_0 + 900\alpha_1 + 540\alpha_2 + 120\alpha_3 \\ 1806\alpha_0 + 3780\alpha_1 + 3150\alpha_2 + 1260\alpha_3 + 210\alpha_4 \end{bmatrix},$$

$$x_{1,5} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 24\alpha_0 \\ 240\alpha_0 + 120\alpha_1 \\ 1560\alpha_0 + 1440\alpha_1 + 360\alpha_2 \\ 8400\alpha_0 + 10920\alpha_1 + 5040\alpha_2 + 840\alpha_3 \end{bmatrix},$$

$$x_{1,6} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 120\alpha_0 \\ 1800\alpha_0 + 720\alpha_1 \\ 16800\alpha_0 + 12600\alpha_1 + 2520\alpha_2 \end{bmatrix},$$

$$x_{1,7} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 720\alpha_0 \\ 15120\alpha_0 + 5040\alpha_1 \end{bmatrix} \quad \textit{and} \quad x_{1,8} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 5040\alpha_0 \end{bmatrix},$$

where $\boldsymbol{v} = [\alpha_0, \ldots, \alpha_7]^T \in \mathbb{C}^8$.

The recurrences can be merged in matrix notation as $A\,X_i = X_i\,J_i$ where $X_i = \left[\boldsymbol{x}_{i,1}, \ldots, \boldsymbol{x}_{i,m_i}\right] \in \mathbb{R}^{m \times m_i}$ and

$$J_i = \begin{bmatrix} \lambda_i & & & \\ 1 & \lambda_i & & \\ & \ddots & \ddots & \\ & & 1 & \lambda_i \end{bmatrix} \in \mathbb{R}^{m_i \times m_i};$$

at last, each J_i is called the *Jordan block* of the eigenvalue λ_i. Collecting matrices X_i and J_i for $i \in \{1, \ldots, \nu\}$, the *Jordan canonical form* of A is defined by the relation $A\,X = X\,J$, where $X = [X_1, \ldots, X_\nu] \in \mathbb{R}^{m \times m}$ and

$$J = \begin{bmatrix} J_1 & & \\ & \ddots & \\ & & J_\nu \end{bmatrix} \in \mathbb{R}^{m \times m}$$

with respect to an arbitrary vector $\boldsymbol{v} \in \mathbb{R}^m$; finally, if X is non-singular then matrices A and $X^{-1}\,A\,X = J$ are *similar*, $A \sim_X J$ in symbols.

Remark 33. *Column composition of vectors in 32 yields the matrix* $X = \begin{bmatrix}\boldsymbol{x}_{1,1} & \boldsymbol{x}_{1,2} & \boldsymbol{x}_{1,3} & \boldsymbol{x}_{1,4} & \boldsymbol{x}_{1,5} & \boldsymbol{x}_{1,6} & \boldsymbol{x}_{1,7} & \boldsymbol{x}_{1,8}\end{bmatrix}$ *which is the most abstract matrix such that* $\mathcal{P}_8 \sim_X J$*. In fact, it shows that there are uncountably many matrices* $\hat{X}$ *such that* $\mathcal{P}_8 \sim_{\hat{X}} J$ *by choosing different real values for the components of the vector* $\boldsymbol{v}$*.*

This derivations allow us to compute functions of matrices in a easier way, with the help of the following

Lemma 34. *Let* f *be a function defined on* $\sigma(A)$ *and* g *the corresponding Hermite interpolating polynomial. Then*

$$A \sim_X J \rightarrow g(A) \sim_X g(J),$$

for a matrix X *which depends on an arbitrary vector* $\boldsymbol{v} \in \mathbb{R}^m$*.*

Proof. By definition of similarity relation $X^{-1}\,A\,X = J$, application of g to both members preserves the identity $g(X^{-1}\,A\,X) = g(J)$; finally, since g is a linear combination of powers being a polynomial, $\left(X^{-1}\,A\,X\right)^i = X^{-1}\,A^i\,X$ entails $X^{-1}\,g(A)\,X = g(J)$, as required. □

Previous lemma ensures that $A \sim_X J \rightarrow g(A) = X\,g(J)\,X^{-1}$ and allows us to compute $f(A)$: in words, the procedure consists of, first, finding matrices X and J; second, compute $g(J)$;

third, multiply it by X on the left side and by X^{-1} on the right side. Now, to study the application of f to J we can focus on the application of f to the Jordan block J_i due to the block-wise structure of matrix J and, lately, compose results block-wise as well.

Remark 35. *Since the Jordan block* J_i *is a* m_i*-minor of the Riordan array* $(\lambda_i + t, t)$ *then it shares the same base of polynomials shown in Equation 2.5, hence for a function* f *defined on* $\sigma(J_i)$*, the application* $f(J_i)$ *yields the Toeplitz matrix*

$$\begin{bmatrix}
f(\lambda_i) & & & & & & & \\
\frac{d}{d\lambda_i} f(\lambda_i) & f(\lambda_i) & & & & & & \\
\frac{1}{2}\frac{d^2}{d\lambda_i^2} f(\lambda_i) & \frac{d}{d\lambda_i} f(\lambda_i) & f(\lambda_i) & & & & & \\
\frac{1}{6}\frac{d^3}{d\lambda_i^3} f(\lambda_i) & \frac{1}{2}\frac{d^2}{d\lambda_i^2} f(\lambda_i) & \frac{d}{d\lambda_i} f(\lambda_i) & f(\lambda_i) & & & & \\
\frac{1}{24}\frac{d^4}{d\lambda_i^4} f(\lambda_i) & \frac{1}{6}\frac{d^3}{d\lambda_i^3} f(\lambda_i) & \frac{1}{2}\frac{d^2}{d\lambda_i^2} f(\lambda_i) & \frac{d}{d\lambda_i} f(\lambda_i) & f(\lambda_i) & & & \\
\frac{1}{120}\frac{d^5}{d\lambda_i^5} f(\lambda_i) & \frac{1}{24}\frac{d^4}{d\lambda_i^4} f(\lambda_i) & \frac{1}{6}\frac{d^3}{d\lambda_i^3} f(\lambda_i) & \frac{1}{2}\frac{d^2}{d\lambda_i^2} f(\lambda_i) & \frac{d}{d\lambda_i} f(\lambda_i) & f(\lambda_i) & & \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \\
\frac{1}{(m_i-1)!}\frac{d^{m_i-1}}{d\lambda_i^{m_i-1}} f(\lambda_i) & \frac{1}{(m_i-2)!}\frac{d^{m_i-2}}{d\lambda_i^{m_i-2}} f(\lambda_i) & \cdots & \cdots & \cdots & \cdots & \frac{d}{d\lambda_i} f(\lambda_i) & f(\lambda_i)
\end{bmatrix}.$$

We show columns for the family of functions studied in previous sections for a minor 8×8:

$$J_i^r \boldsymbol{e}_1 = \begin{bmatrix} \frac{(r)_1 \lambda_i^r}{0!} \\ \frac{(r)_i}{1!}\lambda_i^{r-1} \\ \frac{(r)_2}{2!}\lambda_i^{r-2} \\ \frac{(r)_3}{3!}\lambda_i^{r-3} \\ \frac{(r)_4}{4!}\lambda_i^{r-4} \\ \frac{(r)_5}{5!}\lambda_i^{r-5} \\ \frac{(r)_6}{6!}\lambda_i^{r-6} \\ \frac{(r)_7}{7!}\lambda_i^{r-7} \end{bmatrix}, \quad \frac{\boldsymbol{e}_1}{J_i} = \begin{bmatrix} \frac{1}{\lambda_i} \\ -\frac{1}{\lambda_i^2} \\ \frac{1}{\lambda_i^3} \\ -\frac{1}{\lambda_i^4} \\ \frac{1}{\lambda_i^5} \\ -\frac{1}{\lambda_i^6} \\ \frac{1}{\lambda_i^7} \\ -\frac{1}{\lambda_i^8} \end{bmatrix}, \quad \sqrt{J_i}\boldsymbol{e}_1 = \begin{bmatrix} \sqrt{\lambda_i} \\ \frac{1}{2\sqrt{\lambda_i}} \\ -\frac{1}{8\lambda_i^{\frac{3}{2}}} \\ \frac{1}{16\lambda_i^{\frac{5}{2}}} \\ -\frac{5}{128\lambda_i^{\frac{7}{2}}} \\ \frac{7}{256\lambda_i^{\frac{9}{2}}} \\ -\frac{21}{1024\lambda_i^{\frac{11}{2}}} \\ \frac{33}{2048\lambda_i^{\frac{13}{2}}} \end{bmatrix},$$

$$e^{J_i\alpha}\boldsymbol{e}_1 = \begin{bmatrix} e^{\alpha\lambda_i} \\ \alpha e^{\alpha\lambda_i} \\ \frac{\alpha^2}{2}e^{\alpha\lambda_i} \\ \frac{\alpha^3}{6}e^{\alpha\lambda_i} \\ \frac{\alpha^4}{24}e^{\alpha\lambda_i} \\ \frac{\alpha^5}{120}e^{\alpha\lambda_i} \\ \frac{\alpha^6}{720}e^{\alpha\lambda_i} \\ \frac{\alpha^7}{5040}e^{\alpha\lambda_i} \end{bmatrix}, \quad \log(J_i)\boldsymbol{e}_1 = \begin{bmatrix} \log(\lambda_i) \\ \frac{1}{\lambda_i} \\ -\frac{1}{2\lambda_i^2} \\ \frac{1}{3\lambda_i^3} \\ -\frac{1}{4\lambda_i^4} \\ \frac{1}{5\lambda_i^5} \\ -\frac{1}{6\lambda_i^6} \\ \frac{1}{7\lambda_i^7} \end{bmatrix},$$

$$\sin(J_i)\boldsymbol{e}_1 = \begin{bmatrix} \sin(\lambda_i) \\ \cos(\lambda_i) \\ -\frac{1}{2}\sin(\lambda_i) \\ -\frac{1}{6}\cos(\lambda_i) \\ \frac{1}{24}\sin(\lambda_i) \\ \frac{1}{120}\cos(\lambda_i) \\ -\frac{1}{720}\sin(\lambda_i) \\ -\frac{1}{5040}\cos(\lambda_i) \end{bmatrix} \quad \text{and} \quad \cos(J_i)\boldsymbol{e}_1 = \begin{bmatrix} \cos(\lambda_i) \\ -\sin(\lambda_i) \\ -\frac{1}{2}\cos(\lambda_i) \\ \frac{1}{6}\sin(\lambda_i) \\ \frac{1}{24}\cos(\lambda_i) \\ -\frac{1}{120}\sin(\lambda_i) \\ -\frac{1}{720}\cos(\lambda_i) \\ \frac{1}{5040}\sin(\lambda_i) \end{bmatrix};$$

moreover, observe that if A is a Riordan array then its Jordan canonical form reduces to matrices $X = X_1$ and $J = J_1$ because of the unique eigenvalue λ_1 of algebraic multiplicity $m_1 = m$.

Example 36. *Let $\mathcal{P} \sim_X J$, then Pascal triangle's inverse $\mathcal{P}^{-1}$ can be computed by $\mathcal{P}^{-1} = X\,J^{-1}\,X^{-1}$, where*

$$X = \alpha_0 \begin{bmatrix} 1 & & & & & & & \\ 0 & 1 & & & & & & \\ 0 & 1 & 2 & & & & & \\ 0 & 1 & 6 & 6 & & & & \\ 0 & 1 & 14 & 36 & 24 & & & \\ 0 & 1 & 30 & 150 & 240 & 120 & & \\ 0 & 1 & 62 & 540 & 1560 & 1800 & 720 & \\ 0 & 1 & 126 & 1806 & 8400 & 16800 & 15120 & 5040 \end{bmatrix}$$

depends on $\boldsymbol{v} = \begin{bmatrix} \alpha_0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$, $\alpha_0 \in \mathbb{R}$.

Considering any two Riordan arrays A and B such that they share the same matrix J in their Jordan canonical forms, the next theorem states that A is a linear transformation of B and viceversa; the same transformation is preserved even for matrices $f(A)$ and $f(B)$ where f is a function defined on both $\sigma(A)$ and $\sigma(B)$.

Theorem 37. *Let A and B be two Riordan matrices and let $A\,X = X\,J$ and $B\,Y = Y\,J$ be their Jordan canonical forms, respectively, where matrices X and Y depend on complex vectors $\boldsymbol{v}$ and $\boldsymbol{w}$; then, $A \sim_{XY^{-1}} B$. Moreover, $f(A) \sim_{XY^{-1}} f(B)$ also holds, for any function f defined on both $\sigma(A)$ and $\sigma(B)$.*

Proof. By transitivity of the similarity relation, $X^{-1}\,A\,X = Y^{-1}\,B\,Y$ entails $Y\,X^{-1}\,A\,X\,Y^{-1} = B$. Finally, let g be the Hermite interpolating polynomial of f, then $g(Y\,X^{-1}\,A\,X\,Y^{-1}) = g(B)$ implies $Y\,X^{-1}\,g(A)\,X\,Y^{-1} = g(B)$, as required. □

Example 38. *Pascal and Catalan triangles are similar with respect to $\mathcal{P} \sim_{XY^{-1}} \mathcal{C}$ and $\mathcal{C} \sim_{YX^{-1}} \mathcal{P}$, where*

$$Y = \beta_0 \begin{bmatrix} 1 & & & & & & & \\ 0 & 1 & & & & & & \\ 0 & 2 & 2 & & & & & \\ 0 & 5 & 11 & 6 & & & & \\ 0 & 14 & 52 & 62 & 24 & & & \\ 0 & 42 & 238 & 470 & 394 & 120 & & \\ 0 & 132 & 1084 & 3176 & 4348 & 2844 & 720 & \\ 0 & 429 & 4956 & 20323 & 40562 & 42874 & 23148 & 5040 \end{bmatrix}$$

depends on $w = \begin{bmatrix} \beta_0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$, $\beta_0 \in \mathbb{R}$, given $\mathcal{C} \sim_Y J$ and $\mathcal{P} \sim_X J$, as before.

Finally, since the product of a Riordan matrix $\mathcal{R}(d(t), h(t))$ and an infinite vector $\mathbf{b} = (b_i)_{i\in\mathbb{N}}$, where $b(t) = \sum_{i\in\mathbb{N}} b_i t^i$, yields $\mathcal{R} \cdot \mathbf{b} = d(t)b(h(t))$ by the fundamental theorem of Riordan arrays, in the next theorem we show a connection to this result.

Theorem 39. *Let A be a Riordan matrix, $\mathbf{b}$ a vector and $AX = XJ$ be the A's Jordan canonical form built on matrices J and X depending on $\mathbf{b}$. Let f be a function defined on $\sigma(A)$, then $f(A) \cdot \mathbf{b} = X f(J)\, \mathbf{e}_0$.*

Proof. Observe that $(X_{\mathbf{b}})^{-1}\, \mathbf{b} = \mathbf{e}_0$ holds because $X_{\mathbf{b}}\, \mathbf{e}_0 = \mathbf{x}_{1,1} = Z_{1,2}^0\, Z_{1,1}\mathbf{b} = \mathbf{b}$. Let g be the Hermite interpolating polynomial of function f, then $f(A) = X\, g(J)\, X^{-1}$ entails $f(A) \cdot \mathbf{b} = X\, g(J)\, X^{-1} \cdot \mathbf{b}$, provided that X depends on $\mathbf{b}$. □

Example 40. *For the sake of clarity, here we lift the power function for application to the generation matrix of Fibonacci numbers, which isn't a Riordan array; on the other hand, its two eigenvalues are distinct and its shape is simple enough to compare and contrast $\Phi_{i,j}$ polynomials, component matrices and Jordan Canonical Form with respect to the main track.*

Let $\mathcal{F}$ be a matrix having two eigenvalues $\lambda_1 \neq \lambda_2$ defined as

$$\mathcal{F} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}, \quad \lambda_1 = \frac{1}{2} - \frac{\sqrt{5}}{2} \quad \text{and} \quad \lambda_2 = \frac{1}{2} + \frac{\sqrt{5}}{2},$$

respectively; we need to use the generalized Lagrange base com-

posed of

$$\Phi_{1,1}(z) = \frac{z}{\lambda_1 - \lambda_2} - \frac{\lambda_2}{\lambda_1 - \lambda_2} \quad \textit{and} \quad \Phi_{2,1}(z) = -\frac{z}{\lambda_1 - \lambda_2} + \frac{\lambda_1}{\lambda_1 - \lambda_2}$$

to define the polynomial

$$g(z) = z\left(\frac{\lambda_1^r}{\lambda_1 - \lambda_2} - \frac{\lambda_2^r}{\lambda_1 - \lambda_2}\right) + \frac{\lambda_1\lambda_2^r}{\lambda_1 - \lambda_2} - \frac{\lambda_1^r\lambda_2}{\lambda_1 - \lambda_2}$$

interpolating $f(z) = z^r$. *Therefore* $\mathcal{F}^r = g(\mathcal{F})$, *in matrix notation*

$$\mathcal{F}^r = \begin{bmatrix} f_{r+1} & f_r \\ f_r & f_{r-1} \end{bmatrix} = \begin{bmatrix} \frac{1}{\lambda_1-\lambda_2}\left(\lambda_1\lambda_2^r - \lambda_1^r\lambda_2 + \lambda_1^r - \lambda_2^r\right) & \frac{\lambda_1^r-\lambda_2^r}{\lambda_1-\lambda_2} \\ \frac{\lambda_1^r-\lambda_2^r}{\lambda_1-\lambda_2} & \frac{\lambda_1\lambda_2^r-\lambda_1^r\lambda_2}{\lambda_1-\lambda_2} \end{bmatrix}$$

where f_n *is the* n*-th Fibonacci number within sequence* $A000045$ *in the OEIS; choosing* $r = 8$ *yields*

$$\mathcal{F}^8 = \begin{bmatrix} f_9 & f_8 \\ f_8 & f_7 \end{bmatrix} = \begin{bmatrix} 34 & 21 \\ 21 & 13 \end{bmatrix}.$$

In order to find the Jordan normal form, we use the following component matrices

$$Z_{1,1} = \begin{bmatrix} -\frac{\lambda_2-1}{\lambda_1-\lambda_2} & \frac{1}{\lambda_1-\lambda_2} \\ \frac{1}{\lambda_1-\lambda_2} & -\frac{\lambda_2}{\lambda_1-\lambda_2} \end{bmatrix}, \quad Z_{2,1} = \begin{bmatrix} \frac{\lambda_1-1}{\lambda_1-\lambda_2} & -\frac{1}{\lambda_1-\lambda_2} \\ -\frac{1}{\lambda_1-\lambda_2} & \frac{\lambda_1}{\lambda_1-\lambda_2} \end{bmatrix}$$

which, in turn, generates subspaces $\mathcal{M}_1$ *and* $\mathcal{M}_2$ *of generalized eigenvectors*

$$x_{1,1} = \begin{bmatrix} -\frac{(\lambda_2-1)\alpha_0}{\lambda_1-\lambda_2} + \frac{\alpha_1}{\lambda_1-\lambda_2} \\ \frac{\alpha_0}{\lambda_1-\lambda_2} - \frac{\alpha_1\lambda_2}{\lambda_1-\lambda_2} \end{bmatrix}, \quad x_{2,1} = \begin{bmatrix} \frac{(\lambda_1-1)\alpha_0}{\lambda_1-\lambda_2} - \frac{\alpha_1}{\lambda_1-\lambda_2} \\ -\frac{\alpha_0}{\lambda_1-\lambda_2} + \frac{\alpha_1\lambda_1}{\lambda_1-\lambda_2} \end{bmatrix}$$

respectively, both depending on vector $v = \begin{bmatrix} \alpha_0 \\ \alpha_1 \end{bmatrix}$*; so* $\mathcal{F}X = XJ$ *is the Jordan normal form of matrix* $\mathcal{F}$*, where*

$$X = \begin{bmatrix} -\frac{(\lambda_2-1)\alpha_0}{\lambda_1-\lambda_2} + \frac{\alpha_1}{\lambda_1-\lambda_2} & \frac{(\lambda_1-1)\alpha_0}{\lambda_1-\lambda_2} - \frac{\alpha_1}{\lambda_1-\lambda_2} \\ \frac{\alpha_0}{\lambda_1-\lambda_2} - \frac{\alpha_1\lambda_2}{\lambda_1-\lambda_2} & -\frac{\alpha_0}{\lambda_1-\lambda_2} + \frac{\alpha_1\lambda_1}{\lambda_1-\lambda_2} \end{bmatrix} \quad \textit{and} \quad J = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}.$$

Let $v = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ *in* $\mathcal{F}^r = \left(XJX^{-1}\right)^r = XJ^rX^{-1} = X\begin{bmatrix} \lambda_1^r & 0 \\ 0 & \lambda_2^r \end{bmatrix}X^{-1}$ *where* $X = \begin{bmatrix} \frac{-\lambda_2+2}{\lambda_1-\lambda_2} & \frac{\lambda_1-2}{\lambda_1-\lambda_2} \\ \frac{-\lambda_2+1}{\lambda_1-\lambda_2} & \frac{\lambda_1-1}{\lambda_1-\lambda_2} \end{bmatrix}$, *so matrices*

$\mathcal{F}^r$ and $X\,J^r\,X^{-1}$, whose columns are

$$X\,J^r\,X^{-1}\mathbf{e}_0 = \begin{bmatrix} \frac{2^{-r}\left(\left(1+\sqrt{5}\right)^r(\lambda_1-2)(\lambda_2-1)-\left(-\sqrt{5}+1\right)^r(\lambda_1-1)(\lambda_2-2)\right)}{(\lambda_1-2)(\lambda_2-1)-(\lambda_1-1)(\lambda_2-2)} \\ \frac{2^{-r}\left(\left(1+\sqrt{5}\right)^r-\left(-\sqrt{5}+1\right)^r\right)(\lambda_1-1)(\lambda_2-1)}{(\lambda_1-2)(\lambda_2-1)-(\lambda_1-1)(\lambda_2-2)} \end{bmatrix} \quad \textit{and}$$

$$X\,J^r\,X^{-1}\mathbf{e}_1 = \begin{bmatrix} \frac{2^{-r}\left(-\left(1+\sqrt{5}\right)^r+\left(-\sqrt{5}+1\right)^r\right)(\lambda_1-2)(\lambda_2-2)}{(\lambda_1-2)(\lambda_2-1)-(\lambda_1-1)(\lambda_2-2)} \\ \frac{2^{-r}\left(-\left(1+\sqrt{5}\right)^r(\lambda_1-1)(\lambda_2-2)+\left(-\sqrt{5}+1\right)^r(\lambda_1-2)(\lambda_2-1)\right)}{(\lambda_1-2)(\lambda_2-1)-(\lambda_1-1)(\lambda_2-2)} \end{bmatrix},$$

are similar; by the way, substituting $r = 8$ yields

$$XJ^8X^{-1} = \begin{bmatrix} 34 & 21 \\ 21 & 13 \end{bmatrix}$$

as required.

Conclusions

In this chapter we studied Hermite interpolating polynomials for functions $f(z) = z^r$, $f(z) = \frac{1}{z}$, $f(z) = \sqrt{z}$, $f(z) = e^{\alpha z}$, $f(z) = \log z$, $f(z) = \sin z$ and $f(z) = \cos z$ for application to a well known class of matrices, namely Riordan arrays: in this context, the submatrix $m \times m$ of the array $\mathcal{R}$ has a unique eigenvalue λ of algebraic multiplicity m, which simplify derivations sensibly. Other functions could be studied provided that they are defined on $\sigma(\mathcal{R}_m)$; for example, the normal density function $f(z) = \frac{\sqrt{2}e^{-\frac{z^2}{2}}}{2\sqrt{\pi}}$ admits the interpolating polynomial, for any Riordan matrix $\mathcal{R}_8$,

$$\begin{aligned} N_8(z) = {} & \frac{\sqrt{2}z^7}{504\sqrt{\pi e}} - \frac{\sqrt{2}z^6}{360\sqrt{\pi e}} - \frac{\sqrt{2}z^5}{20\sqrt{\pi e}} + \frac{13\sqrt{2}z^4}{72\sqrt{\pi e}} \\ & - \frac{5\sqrt{2}z^3}{72\sqrt{\pi e}} - \frac{3\sqrt{2}z^2}{8\sqrt{\pi e}} - \frac{\sqrt{2}z}{90\sqrt{\pi e}} + \frac{2081\sqrt{2}}{2520\sqrt{\pi e}}. \end{aligned}$$

Another aspect that could be of interest concerns examination of functions that, once applied to Riordan arrays, produces matrices that are Riordan arrays themselves; the Pascal triangle is a witness for the r-th power function, namely $\mathcal{P}_m^r$ is a Riordan array, where $r \in \mathbb{Q}$. To this purpose, we could approach the problem from an analytic point of view in terms of functions $d(t)$ and $h(t)$ defining the Riordan array under investigation.

3

Algebraic generating functions for languages avoiding Riordan patterns

This chapter is an extended version of our paper [Merlini and Nocentini, 2018] that studies languages $\mathfrak{L}^{[\mathfrak{p}]} \subset \{0, 1\}^*$ of binary words w avoiding a given pattern $\mathfrak{p}$ such that $|w|_0 \leq |w|_1$ for any $w \in \mathfrak{L}^{[\mathfrak{p}]}$, where $|w|_0$ and $|w|_1$ correspond to the number of 1-bits and 0-bits in the word w, respectively. In particular, we concentrate on patterns $\mathfrak{p}$ related to the concept of Riordan arrays. These languages are not regular and can be enumerated by algebraic generating functions corresponding to many integer sequences which are unknown in the OEIS. We give explicit formulas for these generating functions expressed in terms of the autocorrelation polynomial of $\mathfrak{p}$ and also give explicit formulas for the coefficients of some particular patterns, algebraically and combinatorially.

3.1 Introduction

The notion of a pattern can be formalized in several ways and in this paper we consider *factor patterns*, that is, patterns whose letters must appear in an exact order and contiguously in the sequence under observation. The set of binary words avoiding a pattern, without the restriction $|w|_0 \leq |w|_1$, is defined by a regular language and can be enumerated in terms of the number of 1-bits and 0-bits by using classical results (see, e.g., [Guibas and Odlyzko, 1980, 1981, Sedgewick and Flajolet, 1996]). However, when we consider the additional restriction that the words have no more 0-bits than 1-bits, the language is no longer regular and enumerating it is a harder problem.

In this paper we are interested in *Riordan patterns*, a concept which has been defined in [Merlini and Sprugnoli, 2011] in terms of the *autocorrelation polynomial* $C^{[\mathfrak{p}]}(x, y)$ of pattern $\mathfrak{p} = p_0 \cdots p_{h-1}$. The coefficients of this polynomial are given by the *autocorrelation vector* associated to $\mathfrak{p}$, that is, the vector $c = (c_0, \ldots, c_{h-1})$ of bits defined in terms of Iverson's bracket notation (for a predicate P, the expression $[\![P]\!]$ has value 1 if P is true and 0 otherwise) as follows:

$$c_i = [\![p_0 p_1 \cdots p_{h-1-i} = p_i p_{i+1} \cdots p_{h-1}]\!];$$

in words, the bit c_i is determined by shifting $\mathfrak{p}$ to the right by i positions, setting $c_i = 1$ if and only if the remaining letters match the original. For example, when $\mathfrak{p} = 10101$ the

1	0	1	0	1	Tails	c_i
1	0	1	0	1		1
	1	0	1	0	1	0
		1	0	1	0 1	1
			1	0	1 0 1	0
				1	0 1 0 1	1

Table 3.1: The autocorrelation vector for the pattern $\mathfrak{p} = 10101$.

autocorrelation vector is $c = (1, 0, 1, 0, 1)$, as illustrated in Table 3.1, and $C^{[\mathfrak{p}]}(x, y) = 1 + xy + x^2y^2$, namely we add a term $x^j y^i$ for each tail of the pattern with j 1-bits and i 0-bits, where $c_{j+i} = 1$. For each pattern $\mathfrak{p}$, we can compute the *complement* pattern $\bar{\mathfrak{p}}$ by changing every 1 to 0 and every 0 to 1; for example, if $\mathfrak{p} = 10101$ then $\bar{\mathfrak{p}} = 01010$, therefore $C^{[\mathfrak{p}]}(x, y) = C^{[\bar{\mathfrak{p}}]}(y, x)$.

Addition of constraints to the nature of a pattern $\mathfrak{p}$ yields the following

Definition 41 (Riordan pattern). *We say that $\mathfrak{p} = p_0 \cdots p_{h-1}$ is a Riordan pattern if and only if*

$$C^{[\mathfrak{p}]}(x, y) = C^{[\bar{\mathfrak{p}}]}(y, x) = \sum_{i=0}^{\lfloor (h-1)/2 \rfloor} c_{2i} x^i y^i,$$

with $|n_1^{[\mathfrak{p}]} - n_0^{[\mathfrak{p}]}| \in \{0, 1\}$, where $n_1^{[\mathfrak{p}]}$ and $n_0^{[\mathfrak{p}]}$ correspond to the number of 1-bits and 0-bits in the pattern, respectively.

For example, Table 3.1 corresponds to a Riordan pattern and $\mathfrak{p} = 1100110110011000$ is another Riordan pattern having $n_1^{[\mathfrak{p}]} = n_0^{[\mathfrak{p}]} = 8$ and $C^{[\mathfrak{p}]}(x, y) = 1$. Moreover, in Table 3.2 we give all the Riordan patterns of length 7 with first bit equal to 1 and their correlation polynomials, the corresponding complement patterns can be easily determined.

The name *Riordan* in the above definition is due to the connection with the well-known concept of *Riordan arrays*, introduced in Chapter 1. Consider the languages $\mathfrak{L}^{[\mathfrak{p}]} \subset \{0, 1\}^*$ of binary words avoiding a pattern $\mathfrak{p}$ and let $R_{n,k}^{[\mathfrak{p}]}$ be the number of words avoiding $\mathfrak{p}$ and having n 1-bits and $n - k$ 0-bits; additionally, let $\mathcal{R}^{[\mathfrak{p}]} = \left(R_{n,k}^{[\mathfrak{p}]}\right)_{n,k \in \mathbb{N}}$ the enclosing matrix. The

Table 3.2: The Riordan patterns of length 7 with first bit equal to 1 and their correlation polynomials.

$\mathfrak{p}$	$C^{[\mathfrak{p}]}(x,y)$
1010100, 1011000 1011100, 1100010 1100100, 1101000 1101010, 1101100 1110000, 1110010 1110100, 1111000	1
1001100, 1100110	$1+x^2y^2$
1000111, 1001011 1001101, 1010011 1011001, 1100101 1101001, 1110001	$1+x^3y^3$
1010101	$1+xy+x^2y^2+x^3y^3$

following theorem, which is proved in [Merlini and Sprugnoli, 2011], shows the importance of Riordan patterns:

Theorem 42. *Matrices $\mathcal{R}^{[\mathfrak{p}]}$ and $\mathcal{R}^{[\bar{\mathfrak{p}}]}$ are Riordan arrays if and only if $\mathfrak{p}$ is a Riordan pattern.*

By previous theorem, matrices $\mathcal{R}^{[\mathfrak{p}]}$ and $\mathcal{R}^{[\bar{\mathfrak{p}}]}$ can be defined as

$$\mathcal{R}^{[\mathfrak{p}]} = (d^{[\mathfrak{p}]}(t), h^{[\mathfrak{p}]}(t)) \text{ and } \mathcal{R}^{[\bar{\mathfrak{p}}]} = (d^{[\bar{\mathfrak{p}}]}(t), h^{[\bar{\mathfrak{p}}]}(t))$$

for the appropriate $d^{[\mathfrak{p}]}$, $h^{[\mathfrak{p}]}$, $d^{[\bar{\mathfrak{p}}]}$, $h^{[\bar{\mathfrak{p}}]}$, given a Riordan pattern $\mathfrak{p}$; moreover, they represent the lower and upper part of the array $\mathcal{F}^{[\mathfrak{p}]} = (F^{[\mathfrak{p}]}_{n,k})_{n,k\in\mathbb{N}}$, where $F^{[\mathfrak{p}]}_{n,k}$ denotes the number of words avoiding pattern $\mathfrak{p}$ and having n 1-bits and k 0-bits .

Remark 43. *Riordan patterns are not the only patterns related to Riordan arrays; for example, given the pattern $\mathfrak{p} = 0100100$ corresponding to $C^{[\mathfrak{p}]}(x,y) = 1 + xy^2 + x^2y^4$, matrix $\mathcal{R}^{[\mathfrak{p}]}$ is still a Riordan array but $\mathcal{R}^{[\bar{\mathfrak{p}}]}$ is not, as illustrated in Example 5.4 of [Baccherini et al., 2007]. However, in these situations it is not easy to find functions $d^{[\mathfrak{p}]}(t)$ and $h^{[\mathfrak{p}]}(t)$ while for Riordan patterns it is always possible, as shown in Theorems 42 and 45.*

As already observed, the enumeration of the set of binary words avoiding a pattern, without the restriction about the number of 1-bits and 0-bits can be done by using classical results and gives the following rational bivariate generating function for the sequence $(F^{[\mathfrak{p}]}_{n,k})_{n,k\in\mathbb{N}}$:

$$F^{[\mathfrak{p}]}(x,y) = \frac{C^{[\mathfrak{p}]}(x,y)}{(1-x-y)C^{[\mathfrak{p}]}(x,y) + x^{n_1^{[\mathfrak{p}]}} y^{n_0^{[\mathfrak{p}]}}},$$

where $n_1^{[\mathfrak{p}]}$ and $n_0^{[\mathfrak{p}]}$ correspond to the number of 1-bits and 0-bits, respectively, and $C^{[\mathfrak{p}]}(x, y)$ is the autocorrelation polynomial, all relative to pattern $\mathfrak{p}$. Consequently, $F^{[\mathfrak{p}]}(t, 1)$ and $F^{[\mathfrak{p}]}(t, t)$ count the words avoiding $\mathfrak{p}$ according to the number of 1-bits and to length of each word, respectively.

Using the theory of Riordan arrays and the results in [Merlini and Sprugnoli, 2011], we give explicit algebraic generating functions enumerating the set of binary words avoiding a Riordan pattern with the restriction $|w|_0 \leq |w|_1$ according to various parameters, in particular to the number of 1-bits and to the words length. Most of the corresponding sequences are new in the On-Line Encyclopedia of Integer Sequences (*OEIS* for short) [Sloane], which uses the convention to identify each sequence with a label of the form $Ax_1x_2x_3x_4x_5x_6$, where $x_i \in \{0, \ldots, 9\}$; moreover, we also give explicit formulas for the coefficients of some particular patterns by providing algebraic and combinatorial proofs.

Finally, our results can be interpreted in the theory of paths and codes in light of the bijection among binary words and paths, which maps a 0-bit to a south-east step $\diagdown$ and a 1-bit to a north-east step $\diagup$. From this point of view, a coefficient $R_{n,k}^{[\mathfrak{p}]} \in \mathcal{R}^{[\mathfrak{p}]}$ counts the number of paths containing n steps of $\diagup$ kind and $n-k$ steps of $\diagdown$ kind, avoiding the subpath corresponding to pattern $\mathfrak{p}$, allowed to cross the x axis but required to end at coordinate $(2n-k, k)$ such that $0 \leq k \leq n$. In particular, $d^{[\mathfrak{p}]}(t)$ is the generating function of paths which avoid $\mathfrak{p}$ and end on the x axis.

Example 44. *The Riordan pattern* $\mathfrak{p} = 10101$ *entails the matrices shown in Table 3.3.*

3.2 Riordan arrays for Riordan patterns

We start with a result which is a direct consequence of Theorems 2.3 and 3.3 in [Merlini and Sprugnoli, 2011]:

Theorem 45. *Let* $R_{n,k}^{[\mathfrak{p}]}$ *be the number of binary words with* n *1-bits and* $n-k$ *0-bits, avoiding a Riordan pattern* $\mathfrak{p}$*. Then the triangle* $\mathcal{R}^{[\mathfrak{p}]} = (R_{n,k}^{[\mathfrak{p}]})$ *is a Riordan array* $\mathcal{R}^{[\mathfrak{p}]} = (d^{[\mathfrak{p}]}(t), h^{[\mathfrak{p}]}(t))$*. In particular, if* $n_1^{[\mathfrak{p}]}$ *and* $n_0^{[\mathfrak{p}]}$ *correspond to the number of 1-bits and 0-bits in the pattern,* $C^{[\mathfrak{p}]}(x, y)$ *is the autocorrelation polynomial of* $\mathfrak{p}$ *and* $C^{[\mathfrak{p}]}(t) = C^{[\mathfrak{p}]}(\sqrt{t}, \sqrt{t})$*, then:*

- *if* $n_1^{[\mathfrak{p}]} - n_0^{[\mathfrak{p}]} = 1$ *we have:*

$$d^{[\mathfrak{p}]}(t) = \frac{C^{[\mathfrak{p}]}(t)}{\sqrt{C^{[\mathfrak{p}]}(t)^2 - 4tC^{[\mathfrak{p}]}(t)(C^{[\mathfrak{p}]}(t) - t^{n_0^{[\mathfrak{p}]}})}},$$

Table 3.3: Matrices $\mathcal{F}^{[\mathfrak{p}]}_{n,k} = \mathcal{R}^{[\bar{\mathfrak{p}}]}_{n,n-k}$ if $k \leq n$ and $\mathcal{F}^{[\mathfrak{p}]}_{n,k} = \mathcal{R}^{[\mathfrak{p}]}_{k,k-n}$ if $n \leq k$ for the Riordan pattern $\mathfrak{p} = 10101$.

$$\mathcal{F}^{[\mathfrak{p}]} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 3 & 6 & 10 & 15 & 21 & 28 & 36 \\ 1 & 4 & 9 & 18 & 32 & 52 & 79 & 114 \\ 1 & 5 & 13 & 30 & 60 & 109 & 184 & 293 \\ 1 & 6 & 18 & 46 & 102 & 204 & 377 & 654 \\ 1 & 7 & 24 & 67 & 163 & 354 & 708 & 1324 \\ 1 & 8 & 31 & 94 & 248 & 580 & 1245 & 2490 \end{bmatrix}$$

$$\mathcal{R}^{[\mathfrak{p}]} = \begin{bmatrix} 1 & & & & & & & \\ 2 & 1 & & & & & & \\ 6 & 3 & 1 & & & & & \\ 18 & 9 & 4 & 1 & & & & \\ 60 & 30 & 13 & 5 & 1 & & & \\ 204 & 102 & 46 & 18 & 6 & 1 & & \\ 708 & 354 & 163 & 67 & 24 & 7 & 1 & \\ 2490 & 1245 & 580 & 248 & 94 & 31 & 8 & 1 \end{bmatrix}$$

$$\mathcal{R}^{[\bar{\mathfrak{p}}]} = \begin{bmatrix} 1 & & & & & & & \\ 2 & 1 & & & & & & \\ 6 & 3 & 1 & & & & & \\ 18 & 10 & 4 & 1 & & & & \\ 60 & 32 & 15 & 5 & 1 & & & \\ 204 & 109 & 52 & 21 & 6 & 1 & & \\ 708 & 377 & 184 & 79 & 28 & 7 & 1 & \\ 2490 & 1324 & 654 & 293 & 114 & 36 & 8 & 1 \end{bmatrix}$$

$$h^{[\mathfrak{p}]}(t) = \frac{C^{[\mathfrak{p}]}(t) - \sqrt{C^{[\mathfrak{p}]}(t)^2 - 4tC^{[\mathfrak{p}]}(t)(C^{[\mathfrak{p}]}(t) - t^{n_0^{[\mathfrak{p}]}})}}{2C^{[\mathfrak{p}]}(t)};$$

- *if* $n_1^{[\mathfrak{p}]} - n_0^{[\mathfrak{p}]} = 0$ *we have:*

$$d^{[\mathfrak{p}]}(t) = \frac{C^{[\mathfrak{p}]}(t)}{\sqrt{(C^{[\mathfrak{p}]}(t) + t^{n_0^{[\mathfrak{p}]}})^2 - 4tC^{[\mathfrak{p}]}(t)^2}},$$

$$h^{[\mathfrak{p}]}(t) = \frac{C^{[\mathfrak{p}]}(t) + t^{n_0^{[\mathfrak{p}]}} - \sqrt{(C^{[\mathfrak{p}]}(t) + t^{n_0^{[\mathfrak{p}]}})^2 - 4tC^{[\mathfrak{p}]}(t)^2}}{2C^{[\mathfrak{p}]}(t)};$$

- *if* $n_0^{[\mathfrak{p}]} - n_1^{[\mathfrak{p}]} = 1$ *we have:*

$$d^{[\mathfrak{p}]}(t) = \frac{C^{[\mathfrak{p}]}(t)}{\sqrt{C^{[\mathfrak{p}]}(t)^2 - 4tC^{[\mathfrak{p}]}(t)(C^{[\mathfrak{p}]}(t) - t^{n_1^{[\mathfrak{p}]}})}},$$

$$h^{[\mathfrak{p}]}(t) = \frac{C^{[\mathfrak{p}]}(t) - \sqrt{C^{[\mathfrak{p}]}(t)^2 - 4tC^{[\mathfrak{p}]}(t)(C^{[\mathfrak{p}]}(t) - t^{n_1^{[\mathfrak{p}]}})}}{2(C^{[\mathfrak{p}]}(t) - t^{n_1^{[\mathfrak{p}]}})}.$$

If $R^{[\mathfrak{p}]}(t, w)$ denotes the bivariate generating function of the Riordan array $\mathcal{R}^{[\mathfrak{p}]}$, as already mentioned in Section 3.1, we have:

$$R^{[\mathfrak{p}]}(t, w) = \sum_{n,k \in \mathbb{N}} R^{[\mathfrak{p}]}_{n,k} t^n w^k = \frac{d^{[\mathfrak{p}]}(t)}{1 - wh^{[\mathfrak{p}]}(t)}$$

and Theorem 45 allows us to state the following results.

Theorem 46. *Let* $\mathfrak{p}$ *be a Riordan pattern and* $S^{[\mathfrak{p}]}(t) = \sum_{n\geq 0} S_n^{[\mathfrak{p}]} t^n$ *the generating function enumerating the set of binary words* $\{w \in \{0,1\}^* : |w|_0 \leq |w|_1\}$ *avoiding a Riordan pattern* $\mathfrak{p}$ *according to the number of* 1*-bits. Then we have:*

- *if* $n_1^{[\mathfrak{p}]} = n_0^{[\mathfrak{p}]} + 1$:

$$S^{[\mathfrak{p}]}(t) = \frac{2C^{[\mathfrak{p}]}(t)}{\sqrt{Q(t)}\left(\sqrt{C^{[\mathfrak{p}]}(t)} + \sqrt{Q(t)}\right)}$$

where $Q(t) = (1 - 4t)C^{[\mathfrak{p}]}(t)^2 + 4t^{n_1^{[\mathfrak{p}]}}$;

- *if* $n_0^{[\mathfrak{p}]} = n_1^{[\mathfrak{p}]} + 1$:

$$S^{[\mathfrak{p}]}(t) = \frac{2C^{[\mathfrak{p}]}(t)(C^{[\mathfrak{p}]}(t) - t^{n_1^{[\mathfrak{p}]}})}{\sqrt{Q(t)}\left(C^{[\mathfrak{p}]}(t) - 2t^{n_1^{[\mathfrak{p}]}} + \sqrt{Q(t)}\right)}$$

where $Q(t) = (1-4t)C^{[\mathfrak{p}]}(t)^2 + 4t^{n_0^{[\mathfrak{p}]}}C^{[\mathfrak{p}]}(t)$;

- *if* $n_1^{[\mathfrak{p}]} = n_0^{[\mathfrak{p}]}$:

$$S^{[\mathfrak{p}]}(t) = \frac{2C^{[\mathfrak{p}]}(t)^2}{\sqrt{Q(t)}\left(C^{[\mathfrak{p}]}(t) - t^{n_0^{[\mathfrak{p}]}} + \sqrt{Q(t)}\right)}$$

where $Q(t) = (1-4t)C^{[\mathfrak{p}]}(t)^2 + 2t^{n_0^{[\mathfrak{p}]}}C^{[\mathfrak{p}]}(t) + t^{2n_0^{[\mathfrak{p}]}}$.

Proof. For the proof we can observe that $S^{[\mathfrak{p}]}(t) = \sum_{n\geq 0} S_n^{[\mathfrak{p}]}t^n = R^{[\mathfrak{p}]}(t,1)$, or, equivalently, that $S_n^{[\mathfrak{p}]} = \sum_{k=0}^{n} R_{n,k}^{[\mathfrak{p}]}$ and apply the fundamental rule (1.2) with $f_k = 1$. The statement of the theorem can be found after some algebraic simplification. □

Theorem 47. *Let* $\mathfrak{p}$ *be a Riordan pattern and* $L^{[\mathfrak{p}]}(t) = \sum_{n\geq 0} L_n^{[\mathfrak{p}]}t^n$ *the generating function enumerating the set of binary words* $\{w \in \{0,1\}^* : |w|_0 \leq |w|_1\}$ *avoiding a Riordan pattern* $\mathfrak{p}$ *according to the length. Then we have:*

- *if* $n_1^{[\mathfrak{p}]} = n_0^{[\mathfrak{p}]} + 1$:

$$L^{[\mathfrak{p}]}(t) = \frac{2tC^{[\mathfrak{p}]}(t^2)^2}{\sqrt{Q(t)}\left((2t-1)C(t^2) + \sqrt{Q(t)}\right)},$$

where $Q(t) = C^{[\mathfrak{p}]}(t^2)\left((1-4t^2)C^{[\mathfrak{p}]}(t^2) + 4t^{2n_1^{[\mathfrak{p}]}}\right)$;

- *if* $n_0^{[\mathfrak{p}]} = n_1^{[\mathfrak{p}]} + 1$:

$$L^{[\mathfrak{p}]}(t) = \frac{2t\sqrt{C^{[\mathfrak{p}]}(t^2)}(t^{2n_1^{[\mathfrak{p}]}} - C^{[\mathfrak{p}]}(t^2))}{\sqrt{Q(t)}\left((1-2t)C^{[\mathfrak{p}]}(t^2) + 2t^{n_0^{[\mathfrak{p}]}+n_1^{[\mathfrak{p}]}} - \sqrt{C^{[\mathfrak{p}]}(t^2)Q(t)}\right)},$$

where $Q(t) = (1-4t^2)C^{[\mathfrak{p}]}(t^2) + 4t^{2n_0^{[\mathfrak{p}]}}$;

- *if* $n_1^{[\mathfrak{p}]} = n_0^{[\mathfrak{p}]}$:

$$L^{[\mathfrak{p}]}(t) = \frac{2tC^{[\mathfrak{p}]}(t^2)^2}{\sqrt{Q(t)}\left((2t-1)C(t^2) - t^{2n_0^{[\mathfrak{p}]}} + \sqrt{Q(t)}\right)},$$

where $Q(t) = (1-4t^2)C^{[\mathfrak{p}]}(t^2)^2 + 2t^{2n_0^{[\mathfrak{p}]}}C^{[\mathfrak{p}]}(t^2) + t^{4n_0^{[\mathfrak{p}]}}$.

Proof. For the proof we can observe that the application of generating function $R^{[\mathfrak{p}]}(t,w)$ as

$$R^{[\mathfrak{p}]}\left(tw, \frac{1}{w}\right) = \sum_{n,k\in\mathbb{N}} R_{n,k}^{[\mathfrak{p}]}t^n w^{n-k}$$

entails that $[t^r w^s]R^{[\mathfrak{p}]}\left(tw, \frac{1}{w}\right) = R^{[\mathfrak{p}]}_{r,r-s}$ which is the number of binary words with r 1-bits and s 0-bits. To enumerate according to the length let $t = w$, therefore $L^{[\mathfrak{p}]}(t) = \sum_{n\geq 0} L^{[\mathfrak{p}]}_n t^n = R^{[\mathfrak{p}]}(t^2, 1/t)$. The statement of the theorem can be found after some algebraic simplification. □

Theorems 46 and 47 allows us to find the generating functions $S^{[\mathfrak{p}]}(t)$ and $L^{[\mathfrak{p}]}(t)$ in terms of the autocorrelation polynomial for any Riordan pattern $\mathfrak{p}$. In what follows, we study some special classes of patterns characterized by an autocorrelation polynomial which can be easily computed, as in the case $C^{[\mathfrak{p}]}(x, y) = 1$. For such particular patterns, Theorem 45 simplifies as follows.

Corollary 48. *Let $\mathcal{R}^{[\mathfrak{p}]} = (R^{[\mathfrak{p}]}_{n,k})_{n,k\in\mathbb{N}} = (d^{[\mathfrak{p}]}(t), h^{[\mathfrak{p}]}(t))$ be the Riordan array corresponding to the number of binary words with n 1-bits and $n-k$ 0-bits which avoid the Riordan pattern $\mathfrak{p}$. Then we have the following particular cases:*

- *for $\mathfrak{p} = 1^{j+1}0^j$ we have the Riordan array:*

$$d^{[\mathfrak{p}]}(t) = \frac{1}{\sqrt{1-4t+4t^{j+1}}}, \quad h^{[\mathfrak{p}]}(t) = \frac{1-\sqrt{1-4t+4t^{j+1}}}{2};$$

- *$\mathfrak{p} = 0^{j+1}1^j$ we have the Riordan array:*

$$d^{[\mathfrak{p}]}(t) = \frac{1}{\sqrt{1-4t+4t^{j+1}}}, \quad h^{[\mathfrak{p}]}(t) = \frac{1-\sqrt{1-4t+4t^{j+1}}}{2(1-t^j)};$$

- *$\mathfrak{p} = 1^j0^j$ and $\mathfrak{p} = 0^j1^j$ we have the Riordan array:*

$$d^{[\mathfrak{p}]}(t) = \frac{1}{\sqrt{1-4t+2t^j+t^{2j}}},$$
$$h^{[\mathfrak{p}]}(t) = \frac{1+t^j-\sqrt{1-4t+2t^j+t^{2j}}}{2};$$

- *$\mathfrak{p} = (10)^j1$ we have the Riordan array:*

$$d^{[\mathfrak{p}]}(t) = \frac{\sum_{i=0}^{j} t^i}{\sqrt{1-2\sum_{i=1}^{j} t^i - 3\left(\sum_{i=1}^{j} t^i\right)^2}},$$
$$h^{[\mathfrak{p}]}(t) = \frac{\sum_{i=0}^{j} t^i - \sqrt{1-2\sum_{i=1}^{j} t^i - 3\left(\sum_{i=1}^{j} t^i\right)^2}}{2\sum_{i=0}^{j} t^i};$$

- $\mathfrak{p} = (01)^j 0$ *we have the Riordan array:*

$$d^{[\mathfrak{p}]}(t) = \frac{\sum_{i=0}^{j} t^i}{\sqrt{1 - 2\sum_{i=1}^{j} t^i - 3\left(\sum_{i=1}^{j} t^i\right)^2}},$$

$$h^{[\mathfrak{p}]}(t) = \frac{\sum_{i=0}^{j} t^i - \sqrt{1 - 2\sum_{i=1}^{j} t^i - 3\left(\sum_{i=1}^{j} t^i\right)^2}}{2\sum_{i=0}^{j-1} t^i}.$$

As a peculiar instance, observe that when we instantiate a pattern from family $\mathfrak{p} = 1^j 0^j$ with $j = 1$ we get a Riordan array $\mathcal{R}^{[10]} = \left(d^{[10]}(t), h^{[10]}(t)\right)$ such that

$$d^{[10]}(t) = \frac{1}{1-t} \quad \text{and} \quad h^{[10]}(t) = t,$$

so the number $R^{[10]}_{n,0}$ of words containing n 1-bits and n 0-bits, avoiding pattern $\mathfrak{p} = 10$, is $[t^n]d^{[10]}(t) = 1$ for $n \in \mathbb{N}$.
If we consider the combinatorial interpretation of $R^{[\mathfrak{p}]}_{n,0}$ as lattice paths as illustrated in the last paragraph of Section 3.1, this corresponds to the fact that there is exactly one *valley*-shaped path having n steps of both kinds $\diagup$ and $\diagdown$, avoiding $\mathfrak{p} = 10$ and terminating at coordinate $(2n, 0)$ for each $n \in \mathbb{N}$, formally the path $0^n 1^n$.

By applying Theorem 46 to the same patterns as before we state the

Corollary 49. *Let* $S^{[\mathfrak{p}]}(t) = \sum_{n \geq 0} S^{[\mathfrak{p}]}_n t^n$ *the generating function enumerating the set of binary words* $\{w \in \{0,1\}^* : |w|_0 \leq |w|_1\}$ *avoiding a Riordan pattern* $\mathfrak{p}$ *according to the number of* 1*-bits. We have the following particular cases:*

- *for* $\mathfrak{p} = 1^{j+1}0^j$ *we have:*

$$S^{[\mathfrak{p}]}(t) = \frac{2}{\sqrt{1-4t+4t^{j+1}}\left(1 + \sqrt{1-4t+4t^{j+1}}\right)}$$

- *for* $\mathfrak{p} = 0^{j+1}1^j$ *we have:*

$$S^{[\mathfrak{p}]}(t) = \frac{2(1-t^j)}{\sqrt{1-4t+4t^{j+1}}\left(1 - 2t^j + \sqrt{1-4t+4t^{j+1}}\right)}$$

- *for* $\mathfrak{p} = 1^j 0^j$ *and* $\mathfrak{p} = 0^j 1^j$ *we have:*

$$S^{[\mathfrak{p}]}(t) = \frac{2}{\sqrt{1-4t+2t^j+t^{2j}}\left(1 - t^j + \sqrt{1-4t+2t^j+t^{2j}}\right)}$$

- *for* $\mathfrak{p} = (10)^j1$ *we have:*

$$S^{[\mathfrak{p}]}(t) = \frac{2(1-t^{j+1})}{1-4t+3t^{j+1}+\sqrt{1-4t+2t^{j+1}+4t^{j+2}-3t^{2j+2}}};$$

- *for* $\mathfrak{p} = (01)^j0$ *we have:*

$$S^{[\mathfrak{p}]}(t) = \frac{2(1-t^j-t^{j+1}+t^{2j+1})}{\sqrt{Q(t)}\left(1-2t^j+t^{j+1}+\sqrt{Q(t)}\right)}$$

where $Q(t) = 1-4t+2t^{j+1}+4t^{j+2}-3t^{2j+2}$.

We observe that the case $\mathfrak{p} = (10)^j1$ in Corollary 49 corresponds to the sequence studied in [Bilotta et al., 2013]; moreover, in Table 3.4, Table 3.5, Table 3.6, Table 3.7 and Table 3.8 we report some series developments related to $S^{[\mathfrak{p}]}(t)$ functions just defined, respectively.

j/n	0	1	2	3	4	5	6	7	8	9	10	11
0	1	0	0	0	0	0	0	0	0	0	0	0
1	1	3	7	15	31	63	127	255	511	1023	2047	4095
2	1	3	10	32	106	357	1222	4230	14770	51918	183472	651191
3	1	3	10	35	123	442	1611	5931	22010	82187	308427	1162218
4	1	3	10	35	126	459	1696	6330	23806	90068	342430	1307138
5	1	3	10	35	126	462	1713	6415	24205	91874	350406	1341782
6	1	3	10	35	126	462	1716	6432	24290	92273	352212	1349768
7	1	3	10	35	126	462	1716	6435	24307	92358	352611	1351574
8	1	3	10	35	126	462	1716	6435	24310	92375	352696	1351973

Table 3.4: Series developments for $S^{[1^{j+1}0^j]}(t)$ where $j \in \{0,\ldots,8\}$, avoiding pattern $\mathfrak{p} = 110$; moreover, when $j = 2$ the sequence corresponds to A261058.

j/n	0	1	2	3	4	5	6	7	8	9	10	11
0	1	1	1	1	1	1	1	1	1	1	1	1
1	1	3	8	20	48	112	256	576	1280	2816	6144	13312
2	1	3	10	33	111	378	1302	4525	15841	55783	197389	701286
3	1	3	10	35	124	447	1632	6015	22336	83439	313216	1180511
4	1	3	10	35	126	460	1701	6351	23890	90398	343713	1312108
5	1	3	10	35	126	462	1714	6420	24226	91958	350736	1343069
6	1	3	10	35	126	462	1716	6433	24295	92294	352296	1350098
7	1	3	10	35	126	462	1716	6435	24308	92363	352632	1351658
8	1	3	10	35	126	462	1716	6435	24310	92376	352701	1351994

Table 3.5: Series developments for $S^{[0^{j+1}1^j]}(t)$ where $j \in \{0,\ldots,8\}$, avoiding pattern $\mathfrak{p} = 001$; moreover, when $j = 1$ the sequence corresponds to A001792.

Finally, by applying Theorem 47 to the pattern families already examined, we find the following result.

j/n	0	1	2	3	4	5	6	7	8	9	10	11
0	1	3	10	35	126	462	1716	6435	24310	92378	352716	1352078
1	1	2	3	4	5	6	7	8	9	10	11	12
2	1	3	9	27	82	253	791	2499	7960	25520	82248	266221
3	1	3	10	34	118	417	1493	5400	19684	72196	266122	985003
4	1	3	10	35	125	454	1671	6211	23261	87641	331821	1261398
5	1	3	10	35	126	461	1708	6390	24086	91328	347965	1331072
6	1	3	10	35	126	462	1715	6427	24265	92154	351666	1347326
7	1	3	10	35	126	462	1716	6434	24302	92333	352492	1351028
8	1	3	10	35	126	462	1716	6435	24309	92370	352671	1351854

Table 3.6: Series developments for $S^{[0^j 1^j]}(t)$ (or, equivalently, $S^{[1^j 0^j]}(t)$) where $j \in \{0, \dots, 8\}$, avoiding pattern $\mathfrak{p} = 01$ (or, equivalently, $\mathfrak{p} = 10$); moreover, when $j = 0$ the sequence corresponds to A001700.

j/n	0	1	2	3	4	5	6	7	8	9	10	11
0	1	0	0	0	0	0	0	0	0	0	0	0
1	1	3	7	18	48	131	363	1017	2873	8169	23349	67024
2	1	3	10	32	109	377	1324	4697	16795	60425	218485	793259
3	1	3	10	35	123	445	1631	6036	22511	84460	318438	1205457
4	1	3	10	35	126	459	1699	6350	23911	90572	344737	1317397
5	1	3	10	35	126	462	1713	6418	24225	91979	350910	1344092
6	1	3	10	35	126	462	1716	6432	24293	92293	352317	1350272
7	1	3	10	35	126	462	1716	6435	24307	92361	352631	1351679
8	1	3	10	35	126	462	1716	6435	24310	92375	352699	1351993

Table 3.7: Series developments for $S^{[(10)^j 1]}(t)$ where $j \in \{0, \dots, 8\}$, avoiding pattern $\mathfrak{p} = 101$; moreover, when $j = 1$ the sequence corresponds to A225034.

j/n	0	1	2	3	4	5	6	7	8	9	10	11
0	1	1	1	1	1	1	1	1	1	1	1	1
1	1	3	8	22	61	171	483	1373	3923	11257	32418	93644
2	1	3	10	33	113	393	1384	4920	17618	63456	229642	834342
3	1	3	10	35	124	449	1647	6099	22754	85394	322022	1219205
4	1	3	10	35	126	460	1703	6366	23974	90818	345691	1321092
5	1	3	10	35	126	462	1714	6422	24241	92042	351156	1345049
6	1	3	10	35	126	462	1716	6433	24297	92309	352380	1350518
7	1	3	10	35	126	462	1716	6435	24308	92365	352647	1351742
8	1	3	10	35	126	462	1716	6435	24310	92376	352703	1352009

Table 3.8: Series developments for $S^{[(01)^j 0]}(t)$ where $j \in \{0, \dots, 8\}$, avoiding pattern $\mathfrak{p} = 010$; moreover, when $j = 1$ the sequence corresponds to A025566.

Table 3.9: Set of words with 3, 3, 8, 3 and 3 occurrences of 1-bits avoiding patterns 110, 001, 01, 101 and 010, respectively.

$[t^3]S^{[110]}(t)$	111, 0111, 1011, 00111, 01011, 10011, 10101, 000111, 001011, 010011, 010101, 100011, 100101, 101001, 101010
$[t^3]S^{[001]}(t)$	111, 0111, 1011, 1101, 1110, 01011, 01101, 01110, 10101, 10110, 11010, 11100, 010101, 010110, 011010, 011100, 101010, 101100, 110100, 111000
$[t^8]S^{[01]}(t)$	11111111, 111111110, 1111111100, 11111111000, 111111110000, 1111111100000, 11111111000000, 111111110000000, 1111111100000000
$[t^3]S^{[101]}(t)$	111, 0111, 1110, 00111, 01110, 10011, 11001, 11100, 000111, 001110, 010011, 011001, 011100, 100011, 100110, 110001, 110010, 111000
$[t^3]S^{[010]}(t)$	111, 0111, 1011, 1101, 1110, 00111, 01101, 01110, 10011, 10110, 11001, 11100, 000111, 001101, 001110, 011001, 011100, 100011, 100110, 101100, 110001, 111000

Corollary 50. *Let* $L^{[\mathfrak{p}]}(t) = \sum_{n\geq 0} L_n^{[\mathfrak{p}]} t^n$ *the generating function enumerating the set of binary words* $\{w \in \{0,1\}^* : |w|_0 \leq |w|_1\}$ *avoiding a Riordan pattern* $\mathfrak{p}$ *according to the length. We have the following particular cases:*

- *for* $\mathfrak{p} = 1^{j+1}0^j$ *we have:*

$$L^{[\mathfrak{p}]}(t) = \frac{2t}{\sqrt{1-4t^2+4t^{2(j+1)}}\left(2t-1+\sqrt{1-4t^2+4t^{2(j+1)}}\right)}$$

- *for* $\mathfrak{p} = 0^{j+1}1^j$ *we have:*

$$L^{[\mathfrak{p}]}(t) = \frac{2t(t^{2j}-1)}{\sqrt{1-4t^2+4t^{2(j+1)}}\left(1-2t+2t^{2j+1}-\sqrt{1-4t^2+4t^{2(j+1)}}\right)}$$

- *for* $\mathfrak{p} = 1^j0^j$ *and* $\mathfrak{p} = 0^j1^j$ *we have:*

$$L^{[\mathfrak{p}]}(t) = \frac{2t}{\sqrt{1-4t^2+2t^{2j}+t^{4j}}\left(-1+2t-t^{2j}+\sqrt{1-4t^2+2t^{2j}+t^{4j}}\right)}$$

- *for* $\mathfrak{p} = (10)^j 1$ *we have:*

$$L^{[\mathfrak{p}]}(t) = \frac{2t(t^{2j+2}-1)}{1-4t^2+3t^{2j+2}+(2t-1)\sqrt{Q(t)}}$$

- *for* $\mathfrak{p} = (01)^j 0$ *we have:*

$$L^{[\mathfrak{p}]}(t) = \frac{2t(t^{2j+2}-1)(t^{2j}-1)}{\sqrt{Q(t)}(t^{2j+2}-2t^{2j+1}+2t-1+\sqrt{Q(t)})}$$

where $Q(t) = 1-4t^2+2t^{2j+2}+4t^{2j+4}-3t^{4j+4}$.

In Table 3.10, Table 3.11, Table 3.12, Table 3.13 and Table 3.14 we report some series developments related to $L^{[\mathfrak{p}]}(t)$ functions just defined, respectively.

Table 3.10: Series developments for $L^{[1^{j+1}0^j]}(t)$ where $j \in \{0,\dots,7\}$.

j/n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	3	3	7	7	15	15	31	31	63	63	127	127	255
2	1	1	3	4	11	15	38	55	135	201	483	736	1742	2699	6313
3	1	1	3	4	11	16	42	63	159	247	610	969	2354	3802	9117
4	1	1	3	4	11	16	42	64	163	255	634	1015	2482	4041	9752
5	1	1	3	4	11	16	42	64	163	256	638	1023	2506	4087	9880
6	1	1	3	4	11	16	42	64	163	256	638	1024	2510	4095	9904
7	1	1	3	4	11	16	42	64	163	256	638	1024	2510	4096	9908

Table 3.11: Series developments for $L^{[0^{j+1}1^j]}(t)$ where $j \in \{0,\dots,7\}$.

j/n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	3	4	9	13	26	39	73	112	201	313	546	859	1469
2	1	1	3	4	11	16	40	61	147	231	542	870	2004	3269	7423
3	1	1	3	4	11	16	42	64	161	253	622	999	2414	3942	9396
4	1	1	3	4	11	16	42	64	163	256	636	1021	2494	4071	9812
5	1	1	3	4	11	16	42	64	163	256	638	1024	2508	4093	9892
6	1	1	3	4	11	16	42	64	163	256	638	1024	2510	4096	9906
7	1	1	3	4	11	16	42	64	163	256	638	1024	2510	4096	9908

Table 3.12: Series developments for $L^{[0^j 1^j]}(t)$ (or, equivalently, $L^{[1^j 0^j]}(t)$) where $j \in \{0, \ldots, 7\}$.

j/n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	1	1	3	4	11	16	42	64	163	256	638	1024	2510	4096	9908
1	1	1	2	2	3	3	4	4	5	5	6	6	7	7	8
2	1	1	3	4	10	14	33	48	109	163	362	552	1207	1868	4036
3	1	1	3	4	11	16	41	62	154	240	583	928	2217	3587	8459
4	1	1	3	4	11	16	42	64	162	254	629	1008	2455	4000	9614
5	1	1	3	4	11	16	42	64	163	256	637	1022	2501	4080	9853
6	1	1	3	4	11	16	42	64	163	256	638	1024	2509	4094	9899
7	1	1	3	4	11	16	42	64	163	256	638	1024	2510	4096	9907

Table 3.13: Series developments for $L^{[(10)^j 1]}(t)$ where $j \in \{0, \ldots, 7\}$.

j/n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	3	3	7	8	19	23	53	66	150	191	429	555	1235
2	1	1	3	4	11	15	38	56	139	210	511	790	1892	2973	7034
3	1	1	3	4	11	16	42	63	159	248	614	978	2382	3857	9273
4	1	1	3	4	11	16	42	64	163	255	634	1016	2486	4050	9780
5	1	1	3	4	11	16	42	64	163	256	638	1023	2506	4088	9884
6	1	1	3	4	11	16	42	64	163	256	638	1024	2510	4095	9904
7	1	1	3	4	11	16	42	64	163	256	638	1024	2510	4096	9908

Table 3.14: Series developments for $L^{[(01)^j 0]}(t)$ where $j \in \{0, \ldots, 7\}$.

j/n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	3	4	9	13	28	42	87	134	271	425	844	1342	2628
2	1	1	3	4	11	16	40	61	149	234	558	895	2098	3420	7909
3	1	1	3	4	11	16	42	64	161	253	624	1002	2430	3967	9492
4	1	1	3	4	11	16	42	64	163	256	636	1021	2496	4074	9828
5	1	1	3	4	11	16	42	64	163	256	638	1024	2508	4093	9894
6	1	1	3	4	11	16	42	64	163	256	638	1024	2510	4096	9906
7	1	1	3	4	11	16	42	64	163	256	638	1024	2510	4096	9908

3.3 Code for enumeration and construction of Riordan arrays

After the initial imports

```
from collections import namedtuple
from functools import wraps

import sympy

J_index = namedtuple('J_index', ['j'])
W_index = namedtuple('W_index', ['w_1', 'w_0'])
```

we introduce the definition `words` to be a simple implementation of a word generator, in the sense of Python generator, inspired by a tableau schema. If exausted, `words` yields all binary words that don't contain pattern `avoiding`, using the given occurrences of `ones` and `zeros`, respectively.

```
def words(avoiding, ones, zeros):

    a = len(avoiding)

    def W(word, ones, zeros):

        if ones:
            one = ones[0]
            new_word = word + [one]
            if new_word[-a:] != avoiding:
                yield from W(new_word, ones[1:], zeros)

        if zeros:
            zero = zeros[0]
            new_word = word + [zero]
            if new_word[-a:] != avoiding:
                yield from W(new_word, ones, zeros[1:])

        if not ones and not zeros: yield word

    return W([], ones, zeros)
```

```
>>> w_1, w_0 = 3, 2
>>> binary_words = words(avoiding=[1,1,0],
...                      ones=[1]*w_1,
...                      zeros=[0]*w_0)
>>> sorted(binary_words)
[[0, 0, 1, 1, 1], [0, 1, 0, 1, 1],
 [1, 0, 0, 1, 1], [1, 0, 1, 0, 1]]

>>> words_as_strings(binary_words)
['00111', '01011', '10011', '10101']

>>> {w_0: sorted(words(avoiding=[1,1,0],
...                    ones=[1]*w_1,
...                    zeros=[0]*w_0))
...  for w_0 in range(w_1+1)}
{0: [[1, 1, 1]],
 1: [[0, 1, 1, 1], [1, 0, 1, 1]],
 2: [[0, 0, 1, 1, 1], [0, 1, 0, 1, 1],
     [1, 0, 0, 1, 1], [1, 0, 1, 0, 1]],
 3: [[0, 0, 0, 1, 1, 1], [0, 0, 1, 0, 1, 1],
     [0, 1, 0, 0, 1, 1], [0, 1, 0, 1, 0, 1],
     [1, 0, 0, 0, 1, 1], [1, 0, 0, 1, 0, 1],
     [1, 0, 1, 0, 0, 1], [1, 0, 1, 0, 1, 0]]}
```

Together with the decorator `avoiding`

```
class avoiding:

    def __init__(self, pattern, G):
        self.pattern = pattern
        self.G = G # `G` stands for *generator*

    def __call__(self, post):

        @wraps(post)
        def A(n_domain, j_domain, *args, **kwds):

            T = {J_index(j):
                    {W_index(w_1, w_0): sorted(words)
                     for w_1 in n_domain
                     for w_0 in range(w_1+1)
                     for words in [self.G(pattern,
                                          ones=[1]*w_1,
                                          zeros=[0]*w_0)]}
```

A *decorator* is a function returning another function, usually applied as a function transformation using the `@wrapper` syntax.

The decorator syntax is merely syntactic sugar, the following two function definitions are semantically equivalent with respect to the `staticmethod` decorator:

```
>>> def f(*args):
...     pass
>>> f = staticmethod(f)

>>> @staticmethod
... def f(*args):
...     pass
```

Decorators provides metaprogramming capabilities and modularity; for example, a *memoized* version of Fibonacci numbers recursion can be coded using the `lru_cache` decorator:

```
from functools import lru_cache

@lru_cache(maxsize=None)
def fibonacci(n):
    '''
    Fibonacci's numbers, memoized.

    >>> fibonacci(100)
    354224848179261915075

    >>> list(map(fibonacci, range(20)))
    ... # doctest: +NORMALIZE_WHITESPACE
    [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89,
     144, 233, 377, 610, 987, 1597, 2584, 4181]

    '''
    if n < 2:
        return n

    return fibonacci(n-1) + fibonacci(n-2)
```

```
                for j in j_domain
                for pattern in [self.pattern(j)]} # implicit `let`

            res = post(T, *args, **kwds)
            return res if res else T

        return A
```

with the two helper definitions `words_as_strings` and `rows_sums`

```
def words_as_strings(bws):
    J = lambda w: ''.join(map(str, w))
    return sorted(map(J, bws), key=len)

def rows_sums(M):
    return list(M * sympy.ones(M.rows, 1))
```

it allows us to enumerate words in our families with

```
@avoiding(pattern=lambda j: [1]*(j+1) + [0]*j, G=words)
def ones_then_zeros_words(table): pass

@avoiding(pattern=lambda j: [0]*(j+1) + [1]*j, G=words)
def zeros_then_ones_words(table): pass

@avoiding(pattern=lambda j: [0]*j + [1]*j, G=words)
def zeros_equals_ones_words(table): pass

@avoiding(pattern=lambda j: [1,0]*j + [1], G=words)
def one_zero_star_one_words(table): pass

@avoiding(pattern=lambda j: [0,1]*j + [0], G=words)
def zero_one_star_zero_words(table): pass
```

over the domains for the number of 1-bits, the length and j params

```
>>> n_domain, l_domain, j_domain = (list(range(11)),
                                    list(range(13)),
                                    list(range(3)))
```

respectively. Counting words grouped by the number of 1-bits and by their length by definitions

```
def group_by_ones(n_domain, words):

    Z = sympy.zeros(len(n_domain), len(n_domain))

    for o in n_domain:
        for z in range(o+1):
            Z[o, o-z] += len(words[W_index(o, z)])

    return Z

def group_by_length(n_domain, words):

    rows = len(n_domain)
    Z = sympy.zeros(2 * rows, 2 * rows)

    for o in n_domain:
        for z in range(o+1):
            Z[o+z,o-z] += len(words[W_index(o, z)])

    return Z[:rows,:rows]
```

respectively, we justify the content of previous tables with the following computations

```
>>> binary_words = ones_then_zeros_words(n_domain, j_domain)
>>> words_as_strings([w for w_1 in [3]
...                     for w_0 in range(w_1+1)
...                     for J,W in [(J_index(1),W_index(w_1, w_0))]
...                     for w in binary_words[J][W]])
['111', '0111', '1011', '00111', '01011', '10011', '10101',
 '000111', '001011', '010011', '010101', '100011', '100101',
 '101001', '101010']
```

Proves the set of words already enumerated in the *first* row of Table 3.9.

```
>>> M = group_by_ones(n_domain, binary_words[J_index(1)])
>>> rows_sums(M)
[1, 3, 7, 15, 31, 63, 127, 255, 511, 1023, 2047]
>>> M
```

Proves the series development in row $j = 1$ of Table 3.4 as row summation $\mathtt{M} \cdot \mathbf{1}$.

$$\left[\begin{array}{ccccccccccc}
1 &&&&&&&&&& \\
2 & 1 &&&&&&&&& \\
4 & 2 & 1 &&&&&&&& \\
8 & 4 & 2 & 1 &&&&&&& \\
16 & 8 & 4 & 2 & 1 &&&&&& \\
32 & 16 & 8 & 4 & 2 & 1 &&&&& \\
64 & 32 & 16 & 8 & 4 & 2 & 1 &&&& \\
128 & 64 & 32 & 16 & 8 & 4 & 2 & 1 &&& \\
256 & 128 & 64 & 32 & 16 & 8 & 4 & 2 & 1 && \\
512 & 256 & 128 & 64 & 32 & 16 & 8 & 4 & 2 & 1 & \\
1024 & 512 & 256 & 128 & 64 & 32 & 16 & 8 & 4 & 2 & 1
\end{array}\right]$$

```
>>> binary_words = ones_then_zeros_words(l_domain, j_domain)
>>> M = group_by_length(l_domain, binary_words[J_index(1)])
>>> rows_sums(M)
[1, 1, 3, 3, 7, 7, 15, 15, 31, 31, 63, 63, 127]
>>> M
```

Proves the series development in row $j = 1$ of Table 3.10 as row summation $\mathtt{M} \cdot \mathbf{1}$.

$$\left[\begin{array}{ccccccccccccc}
1 &&&&&&&&&&&& \\
& 1 &&&&&&&&&&& \\
2 && 1 &&&&&&&&&& \\
& 2 && 1 &&&&&&&&& \\
4 && 2 && 1 &&&&&&&& \\
& 4 && 2 && 1 &&&&&&& \\
8 && 4 && 2 && 1 &&&&&& \\
& 8 && 4 && 2 && 1 &&&&& \\
16 && 8 && 4 && 2 && 1 &&&& \\
& 16 && 8 && 4 && 2 && 1 &&& \\
32 && 16 && 8 && 4 && 2 && 1 && \\
& 32 && 16 && 8 && 4 && 2 && 1 & \\
64 && 32 && 16 && 8 && 4 && 2 && 1
\end{array}\right]$$

```
>>> M = group_by_length(l_domain, binary_words[J_index(2)])
>>> rows_sums(M)
[1, 1, 3, 4, 11, 15, 38, 55, 135, 201, 483, 736, 1742]
>>> M
```

Proves the series development in row $j = 2$ of Table 3.10 as row summation $M \cdot \mathbb{1}$.

$$\begin{bmatrix}
1 &&&&&&&&&&&& \\
& 1 &&&&&&&&&&& \\
2 && 1 &&&&&&&&&& \\
& 3 && 1 &&&&&&&&& \\
6 && 4 && 1 &&&&&&&& \\
& 9 && 5 && 1 &&&&&&& \\
18 && 13 && 6 && 1 &&&&&& \\
& 29 && 18 && 7 && 1 &&&&& \\
58 && 44 && 24 && 8 && 1 &&&& \\
& 96 && 64 && 31 && 9 && 1 &&& \\
192 && 151 && 90 && 39 && 10 && 1 && \\
& 325 && 228 && 123 && 48 && 11 && 1 & \\
650 && 524 && 333 && 164 && 58 && 12 && 1
\end{bmatrix}$$

3.4 *Some combinatorial interpretations*

In the previous section we proved results about the enumeration of words avoiding patterns from an algebraic point of view. The aim of this section is to analyze in more details some particular cases of the various pattern families. We approach these problems either combinatorially by providing an interpretation, or algebraically by computing the coefficients of the involved generating functions explicitly.

3.4.1 *Enumeration with respect to the number of* 1*-bits*

Corollary 51. *Consider pattern* $\mathfrak{p} = 1^{j+1}0^{j}$*, there is only one word in* $\mathfrak{L}^{[\mathfrak{p}]}$ *for* $j = 0$*; on the other hand, there are* $S_{n}^{[\mathfrak{p}]} = 2^{n+1} - 1$ *words for* $j = 1$.

Proof. When $j = 0$ the pattern to avoid is $\mathfrak{p} = 1$, therefore only words w in $\{\varepsilon\} \cup \{0\}^{+}$ are suitable choices; however, the constraint $|w|_0 \leq |w|_1$ makes $w = \varepsilon$ the only one.

When $j = 1$ the pattern to avoid is $\mathfrak{p} = 110$ and we observe that the generic binomial $\binom{r}{k}$ can be interpreted as the number of binary words with r 0-bits containing k occurrences of the substring 10, which we call an *inversion* with respect to pattern $\mathfrak{p} = 110$. In order to build a word in the language we start from the substring 0^{r} for $r \in \{0, \cdots, n\}$ and select $k \in \{0, \ldots, r\}$ 0-bits, transforming each one using the mapping $0 \mapsto 10$, while preventing the transformation of the 0-bit in the 10 just introduced. This maneuver introduces k inversions and the selection can be done in $\binom{r}{k}$ ways; finally, we pad on the right with a strip 1^{n-k}, because it is mandatory for a word to have n 1-bits, hence there is one padding for each set of inversions and there is no other way to avoid $\mathfrak{p}$. Therefore

$$\sum_{r=0}^{n}\sum_{k=0}^{r}\binom{r}{k} = 2^{n+1}-1 = S_n^{[\mathfrak{p}]},$$

as can be verified algebraically by extracting the coefficient of the generating function $S^{[\mathfrak{p}]}(t) = \frac{1}{1-3t+2t^2} = \frac{2}{1-2t} - \frac{1}{1-t}$, as required. □

The same argument can be rewritten in term of sets which allows us give a constructive approach. Let $\mathcal{S}_{n,k,i}$ be the set of binary words containing n and k occurrences of 1-bits and 0-bits, respectively, with i inversions, namely an occurrence of the subsequence 10. By union respect to i and k we get sets $\mathcal{S}_{n,k}^{[\mathfrak{p}]}$ and $\mathcal{S}_{n}^{[\mathfrak{p}]}$, formally:

$$\mathcal{S}_n^{[\mathfrak{p}]} = \bigcup_{k\in\{0,\dots,n\}} \mathcal{S}_{n,k}^{[\mathfrak{p}]} = \bigcup_{i\in\{0,\dots,k\}} \mathcal{S}_{n,k,i}^{[\mathfrak{p}]} = \left(\bigcup_{i\in\{0,\dots,k\}} \mathcal{S}_{k,k,i}^{[\mathfrak{p}]}\right) \times \left\{1^{n-k}\right\}$$

For the sake of clarity, we enumerate all binary words avoiding $\mathfrak{p} = 110$ containing $n = 3$ 1-bits, formally we partition $\mathcal{S}_3^{[\mathfrak{p}]}$ as follows

$$\begin{aligned}\mathcal{S}_3^{[\mathfrak{p}]} &= \mathcal{S}_{0,0,0}^{[\mathfrak{p}]} \times \{111\}\\ &\cup \left(\mathcal{S}_{1,1,0}^{[\mathfrak{p}]} \cup \mathcal{S}_{1,1,1}^{[\mathfrak{p}]}\right) \times \{11\}\\ &\cup \left(\mathcal{S}_{2,2,0}^{[\mathfrak{p}]} \cup \mathcal{S}_{2,2,1}^{[\mathfrak{p}]} \cup \mathcal{S}_{2,2,2}^{[\mathfrak{p}]}\right) \times \{1\}\\ &\cup \left(\mathcal{S}_{3,3,0}^{[\mathfrak{p}]} \cup \mathcal{S}_{3,3,1}^{[\mathfrak{p}]} \cup \mathcal{S}_{3,3,2}^{[\mathfrak{p}]} \cup \mathcal{S}_{3,3,3}^{[\mathfrak{p}]}\right) \times \{\varepsilon\}\end{aligned}$$

where

$$\begin{aligned}\mathcal{S}_{3,0}^{[\mathfrak{p}]} &= \mathcal{S}_{0,0,0}^{[\mathfrak{p}]} \times \{111\} = \{\varepsilon\} \times \{111\} = \{111\}\\ \mathcal{S}_{3,1}^{[\mathfrak{p}]} &= \left(\mathcal{S}_{1,1,0}^{[\mathfrak{p}]} \cup \mathcal{S}_{1,1,1}^{[\mathfrak{p}]}\right) \times \{11\} = \{0111\} \cup \{1011\}\\ \mathcal{S}_{3,2}^{[\mathfrak{p}]} &= \left(\mathcal{S}_{2,2,0}^{[\mathfrak{p}]} \cup \mathcal{S}_{2,2,1}^{[\mathfrak{p}]} \cup \mathcal{S}_{2,2,2}^{[\mathfrak{p}]}\right) \times \{1\} = \{00111\} \cup \{10011, 01011\} \cup \{10101\}\\ \mathcal{S}_{3,3}^{[\mathfrak{p}]} &= \left(\mathcal{S}_{3,3,0}^{[\mathfrak{p}]} \cup \mathcal{S}_{3,3,1}^{[\mathfrak{p}]} \cup \mathcal{S}_{3,3,2}^{[\mathfrak{p}]} \cup \mathcal{S}_{3,3,3}^{[\mathfrak{p}]}\right) \times \{\varepsilon\} = \{000111\} \cup \{001011, 100011, 010011\}\\ &\qquad \cup \{101001, 100101, 010101\} \cup \{101010\}\end{aligned}$$

the same set of words shown in Table 3.4.

Corollary 52. *Consider pattern* $\mathfrak{p} = 0^{j+1}1^{j}$*, there is one word* $S_n^{[\mathfrak{p}]} = 1$ *for each* $n \in \mathbb{N}$ *in* $\mathfrak{L}^{[\mathfrak{p}]}$ *when* $j = 0$*; on the other hand, there are* $(n+2)2^{n-1}$ *words for* $j = 1$.

Proof. When $j = 0$ the pattern to avoid is $\mathfrak{p} = 0$, therefore only words $w = 1^n$ are suitable, hence there is one of them for each $n \in \mathbb{N}$.

When $j = 1$ the pattern to avoid is $\mathfrak{p} = 001$, therefore we extract the n-th coefficient after instantiation of the corresponding generating function $[t^n]S_n^{[\mathfrak{p}]}(t) = [t^n]\frac{1-t}{(1-2t)^2} = (n+2)2^{n-1}$, as required.

We also provide a combinatorial interpretation of the theorem; first of all, we observe that sequence $S_n^{[\mathfrak{p}]}$ is the binomial transform of the sequence of the positive integers $(n+1)_{n\in\mathbb{N}}$, formally

$$S_n^{[\mathfrak{p}]} = (n+2)2^{n-1} = \sum_{k=0}^{n} \binom{n}{k}(k+1)$$

where the generic summand $\binom{n}{k}(k+1)$ can be interpreted as the number of binary words with n 1-bits containing $n-k$ occurrences of the substring 01, which we call an *inversion* respect to pattern $\mathfrak{p} = 001$. We construct the set of words avoiding $\mathfrak{p}$ to show the bijection with the previous assert as follows: if in a word w there are $n-k$ occurrences of the substring 01 then w contains $2n-2k$ bits in total, $n-k$ of both kind. Since it is mandatory that the number of 1 is n, we add k 1-bits to it, resulting in a new word w' of length $2n-k$ which can be augmented with at most k additional 0-bits, according to the constraint $|w'|_0 \leq |w'|_1$. In order to build a word with the structure of w', we start from the substring 1^n and select $n-k$ 1-bits, transforming each one using the mapping $1 \mapsto 01$, simultaneously to prevent transforming 1-bit in 01 just introduced. This maneuver introduces $n-k$ inversions and the selection can be done in $\binom{n}{k}$ ways; moreover, we are free to pad on the right with 0^i strips, for $i \in \{0, \cdots, k\}$, hence there are $k+1$ paddings for each set of inversions. Therefore, since there can be up to n inversions,

$$\sum_{k=0}^{n} \binom{n}{n-k}(k+1) = (n+2)2^{n-1}$$

concludes the proof by symmetry of binomial coefficients.

□

Corollary 53. *Consider pattern* $\mathfrak{p} = 0^j1^j$ *(or, equivalently, $\mathfrak{p} = 1^j0^j$), there are*

$$S_n^{[\mathfrak{p}]} = \sum_{k=0}^{n} \binom{n+k}{k} = \binom{2n+1}{n}$$

words in $\mathfrak{L}^{[\mathfrak{p}]}$ *for* $j = 0$*; on the other hand, there are* $S_n^{[\mathfrak{p}]} = n+1$ *words for* $j = 1$.

Proof. When $j = 0$ there is no pattern to avoid and this situation corresponds to the enumeration of binary words $\{w \in \{0,1\}^* : |w|_0 \leq |w|_1 = n\}$. After instantiating the generating function $S^{[\mathfrak{p}]}(t)$, we extract the n-th coefficient

$$[t^n]S_n^{[\mathfrak{p}]}(t) \;=\; [t^n]\frac{1-\sqrt{1-4t}}{2t\sqrt{1-4t}}$$

obtaining $\frac{1}{2}\binom{2(n+1)}{n+1} = \binom{2n+1}{n+1} = \binom{2n+1}{n}$ which simplifies by the identity $\binom{r+s+1}{s} = \sum_{q=0}^{s}\binom{r+q}{q}$ as desired. It is possible to state the following combinatorial interpretation: since the maximum number of 0-bits is n, we reserve n boxes for them. To the left of each box reserve one more box and, finally, another one to the right of the very last box. In this way we have reserved $2n+1$ boxes where we can put n 1-bits in $\binom{2n+1}{n}$ ways, as required.

When $j = 1$ the pattern to avoid is $\mathfrak{p} = 01$ (or, equivalently, $\mathfrak{p} = 10$), therefore only words $w \in \{1^n\} \times \bigcup_{s\in\{0,\ldots,n\}}\{0^s\}$ are suitable, which are $n+1$, one for each value that s can take. □

Last two patterns $\mathfrak{p} = (10)^j1$ and $\mathfrak{p} = (01)^j0$ are harder to study: for $j = 0$ there are $S_n^{[\mathfrak{p}]} = [\![n = 0]\!]$ and $S_n^{[\mathfrak{p}]} = 1$ words, respectively. On the other hand, when $j = 1$ we report only the instantiated generating functions

$$S^{[101]}(t) = \frac{(1+t)\left(1-3t-\sqrt{1-2t-3t^2}\right)}{2t(3t-1)},$$

$$S^{[010]}(t) = \frac{1-2t-3t^2-(1-t)\sqrt{1-2t-3t^2}}{2t^2(3t-1)}.$$

As pointed out by an anonymous referee, previous functions can be rewritten as

$$S^{[101]}(t) = \frac{(1+t)(1-tM(t))}{1-3t},$$
$$S^{[010]}(t) = \frac{(1+tM(t))(1-tM(t))}{1-3t}$$

respectively, where $M(t) = \frac{1-t-\sqrt{1-2t-3t^2}}{2t^2}$ is the Motzkin numbers' generating function. Such rewriting shows a relation among Motzkin numbers and powers of 3; however, a combinatorial argument is not easy to state to the best of our knowledge.

3.4.2 *Enumeration with respect to the length*

Corollary 54. *Consider pattern $\mathfrak{p} = 1^{j+1}0^j$, there is one word in $\mathfrak{L}^{[\mathfrak{p}]}$ for $j = 0$; on the other hand, there are $2^{m+1} - 1$ words, where $n = 2m + [\![n \text{ is odd}]\!]$, for $j = 1$.*

Proof. When $j = 0$ the pattern to avoid is $\mathfrak{p} = 1$, therefore instantiating the generating function we have $L^{[\mathfrak{p}]}(t) = 1$, as required.

When $j = 1$ the pattern to avoid is $\mathfrak{p} = 110$, therefore we instantiate and extract the n-th coefficient

$$L^{[\mathfrak{p}]}_n = [t^n]\frac{2}{1-2t^2} + [t^{n-1}]\frac{2}{1-2t^2} - [t^n]\frac{1}{1-t}$$

and proceed by cases on the parity of n. If $n = 2m$ then the second term in the rhs disappears, otherwise if $n = 2m+1$ the first term disappears; in both cases it is required to perform $[u^m]\frac{2}{1-2u} = 2^{m+1}$ where $u = t^2$, as required.

It is possible to state a combinatorial interpretation using an argument similar to the one given in the proof of Corollary 51. Let $n = 2m$, therefore a word w needs to have $m + j$ 1-bits, where $j \in \{0, \ldots, m\}$; conversely, w needs to have $n - m - j = m - j$ 0-bits. Fixing j in the given range, from the substring 0^{m-j} we select $i \in \{0, \ldots, m-j\}$ 0-bits to introduce i inversions, namely i occurrences of 10, applying the mapping $0 \mapsto 10$ simultaneously. This maneuver keeps the original 0-bits and introduces at most $m - j$ 1-bits, so we pad with 1-bits on the right in order to have the required $m + j$ 1-bits in the entire word; finally, selections can be done in

$$\sum_{j=0}^{m}\sum_{i=0}^{m-j}\binom{m-j}{i} = \sum_{j=0}^{m} 2^{m-j} = 2^{m+1} - 1$$

ways, because padding can be done in only one way, completing the case for n even.

Let $n = 2m+1$, therefore a word w needs to have $m+1+j$ 1-bits, where $j \in \{0, \ldots, m\}$; conversely, w needs to have $n-m-1-j = m-j$ 0-bits. Fixing j in the given range, from the substring 0^{m-j} we select $i \in \{0, \ldots, m-j\}$ 0-bits to introduce i inversions as done in the even case, introducing at most $m-j$ 1-bits, and padding as necessary to have $m+1+j$ 1-bits, the total number of selections equals the one given for the even case, completing the case for n odd. □

Corollary 55. *Consider pattern $\mathfrak{p} = 0^{j+1}1^j$, there is one word $L_n^{[\mathfrak{p}]} = 1$ for each $n \in \mathbb{N}$ in $\mathfrak{L}^{[\mathfrak{p}]}$ when $j = 0$; on the other hand, there are $L_n^{[\mathfrak{p}]} = F_{n+3} - 2^m$ words if $n = 2m$ else $L_n^{[\mathfrak{p}]} = F_{n+3} - 2^{m+1}$ words if $n = 2m+1$, for $j = 1$, where F_n is the n-th Fibonacci number.*

Proof. When $j = 0$ the pattern to avoid is $\mathfrak{p} = 0$, therefore suitable words of length n are of the form $w = 1^n$, hence $L_n^{[\mathfrak{p}]} = 1$ for each $n \in \mathbb{N}$.

When $j = 1$ the pattern to avoid is $\mathfrak{p} = 001$, therefore we instantiate and extract the n-th coefficient

$$\begin{aligned} L_n^{[\mathfrak{p}]} &= 2[t^{n+1}]\frac{t}{1-t-t^2} + [t^n]\frac{t}{1-t-t^2} \\ &\quad - [t^n]\frac{1}{1-2t^2} - 2[t^{n-1}]\frac{1}{1-2t^2} \end{aligned}$$

in order to have $L_n^{[\mathfrak{p}]} = 2F_{n+1} + F_n - a_n = F_{n+3} - a_n$, where $a_{2m} = 2^m$ and $a_{2m+1} = 2^{m+1}$.

It is possible to state a combinatorial interpretation using an argument similar to the one given in the proof of Corollary 52. Let $n = 2m$, therefore a word w needs to have $m+j$ 1-bits, where $j \in \{0, \ldots, m\}$; conversely, w needs to have $n-m-j = m-j$ 0-bits. Fixing j in the given range, from the substring 1^{m+j} we select $i \in \{0, \ldots, m-j\}$ 1-bits to introduce i inversions, namely i occurrences of 01, applying the mapping $1 \mapsto 01$ simultaneously. This maneuver keeps the original 1-bits and introduces at most $m-j$ 0-bits; finally, selections can be done in $\sum_{j=0}^{m}\sum_{i=0}^{m-j}\binom{m+j}{i}$ ways. In order to find a closed form for the double summation, we inspect the region of the Pascal triangle taken into account; marking with ○ the involved binomials

n/j	0	1	$\dots$	$m-1$	m	$m+1$	$\dots$	$2m-1$	$2m$	$\dots$
0										
$\vdots$										
$m-1$										
m	○	○	$\dots$	○	○					
$m+1$	○	○	$\dots$	○						
$\vdots$	$\vdots$	$\vdots$	$\iddots$							
$2m-1$	○	○								
$2m$	○									
$2m+1$										
$\vdots$										

and using identity $\binom{r+1}{k+1} - \binom{s}{k+1} = \sum_{i=s}^{r} \binom{i}{k}$ for rearranging the summation and identities $2^n = \sum_{i=0}^{n} \binom{n}{i}$ and $F_{n+1} = \sum_{i=0}^{n} \binom{n-i}{i}$ to collect terms, we obtain

$$\sum_{j=0}^{m} \sum_{i=0}^{m-j} \binom{m+j}{i} = \sum_{k=0}^{m} \binom{2m+1-k}{k+1} - \binom{m}{k+1}$$

which equals $F_{2m+3} - 2^m = L_{2m}^{[\mathfrak{p}]}$, closing the case for n even.

Let $n = 2m+1$, therefore a word w needs to have $m + 1 + j$ 1-bits, where $j \in \{0, \dots, m\}$; conversely, w needs to have $n - m - 1 - j = m - j$ 0-bits. Fixing j in the given range, from the substring 1^{m+1+j} select $i \in \{0, \dots, m-j\}$ 1-bits to introduce i inversions as done for the even case; in parallel, selections can be done in $\sum_{j=0}^{m} \sum_{i=0}^{m-j} \binom{m+1+j}{i}$ ways. The involved region in the Pascal triangle has the same shape as the one shown for the even case translated one row to the bottom, so binomials lying on row m are excluded and binomials $\binom{2m+1-k}{k}$ are included, for $k \in \{0, \dots, m\}$. Therefore we rewrite

$$\sum_{j=0}^{m} \sum_{i=0}^{m-j} \binom{m+1+j}{i} = \sum_{k=0}^{m} \binom{2m+2-k}{k+1} - \binom{m+1}{k+1}$$

which equals $F_{2m+4} - 2^{m+1} = L_{2m+1}^{[\mathfrak{p}]}$, closing the case for n odd. $\square$

Corollary 56. *Consider pattern* $\mathfrak{p} = 0^j 1^j$ *(or, equivalently,* $\mathfrak{p} = 1^j 0^j$*), there are* 2^{n-1} *words in* $\mathfrak{L}^{[\mathfrak{p}]}$ *if* n *is odd else* $2^{n-1} +$

$\frac{1}{2}\binom{2m}{m}$ *where* $n = 2m$, *for* $j = 0$; *on the other hand, there are* $L^{[p]}_{n} = m + 1$ *words, where* $n = 2m + [\![n \text{ is odd}]\!]$, *for* $j = 1$.

Proof. When $j = 0$ the pattern to avoid is $p = \varepsilon$, namely the empty word, therefore instantiating the generating function we have

$$L^{[p]}(t) = \frac{1}{2(1-2t)} + \frac{1}{2\sqrt{1-4t^2}}$$

we extract the coefficient $L^{[p]}_{n} = 2^{n-1} + \frac{a_n}{2}$, where $a_{2m+1} = 0$ and $a_{2m} = \binom{2m}{m}$, as required. We observe that these values correspond to the summation $\sum_{i=0}^{m}\binom{n}{i}$ for $n = 2m, 2m + 1, \cdots$, where the binomial coefficient computes the number of ways to choose i 0-bits among n bits, and this gives the combinatorial interpretation.

When $j = 1$ the pattern to avoid is $p = 01$ (or, equivalently, $p = 10$), therefore after instantiation

$$L^{[p]}(t) = \frac{1}{4(1-t)} + \frac{1}{2(1-t)^2} + \frac{1}{4(1+t)}$$

we extract the n-th coefficient $L^{[p]}_{n} = \frac{1}{4} + \frac{(-1)^n}{4} + \frac{n+1}{2}$, so either $n = 2m$ or $n = 2m + 1$ entails $L^{[p]}_{n} = m + 1$, as required.

A combinatorial interpretation can be given as follows. If $n = 2m$ then suitable words have structure $1^m 1^j 0^{m-j}$ for $j \in \{0, \dots, m\}$, and there are $m + 1$ of them. On the contrary, if $n = 2m + 1$ holds then suitable words have structure $1^{m+1} 1^j 0^{m-j}$ for $j \in \{0, \dots, m\}$, they are $m + 1$ in number again, as required. □

As before, last two patterns $p = (01)^j 0$ and $p = (10)^j 1$ are harder to study and we avoid to report formulas about $L^{[p]}(t)$ functions because we have not a meaningful combinatorial interpretation: we only point out that these functions can be expressed in terms of $M(t^2)$, where $M(t)$ is the generating function of Motzkin numbers, similarly to the corresponding $S^{[p]}(t)$ functions.

Conclusions

As a final remark, we observe a structural properties of matrices $\mathcal{R}^{[p]}$ against the studied families of patterns. Recall that the Pascal triangle and its inverse correspond to the Riordan arrays

$$P = \left(\frac{1}{1-t}, \frac{t}{1-t}\right) \quad \text{and} \quad P^{-1} = \left(\frac{1}{1+t}, \frac{t}{1+t}\right),$$

respectively; therefore, for any Riordan array $\mathcal{R}^{[\mathfrak{p}]}$ we can compute $B^{[\mathfrak{p}]} = P^{-1} * \mathcal{R}^{[\mathfrak{p}]}$, which is equivalent to say that $\mathcal{R}^{[\mathfrak{p}]}$ is the binomial transform of $B^{[\mathfrak{p}]}$, or $\mathcal{R}^{[\mathfrak{p}]} = P * B^{[\mathfrak{p}]}$.

For the sake of clarity, consider the pattern family $\mathfrak{p} = 1^{j+1}0^{j}$, so for $j = 1$ we have the minor

$$\mathcal{R}^{[110]} = \begin{bmatrix} 1 &&&&&&&&&& \\ 2 & 1 &&&&&&&&& \\ 4 & 2 & 1 &&&&&&&& \\ 8 & 4 & 2 & 1 &&&&&&& \\ 16 & 8 & 4 & 2 & 1 &&&&&& \\ 32 & 16 & 8 & 4 & 2 & 1 &&&&& \\ 64 & 32 & 16 & 8 & 4 & 2 & 1 &&&& \\ 128 & 64 & 32 & 16 & 8 & 4 & 2 & 1 &&& \\ 256 & 128 & 64 & 32 & 16 & 8 & 4 & 2 & 1 && \\ 512 & 256 & 128 & 64 & 32 & 16 & 8 & 4 & 2 & 1 & \\ 1024 & 512 & 256 & 128 & 64 & 32 & 16 & 8 & 4 & 2 & 1 \end{bmatrix}$$

which corresponds to

$$B^{[110]} = \begin{bmatrix} 1 &&&&&&&&&& \\ 1 & 1 &&&&&&&&& \\ 1 & 0 & 1 &&&&&&&& \\ 1 & 1 & -1 & 1 &&&&&&& \\ 1 & 0 & 2 & -2 & 1 &&&&&& \\ 1 & 1 & -2 & 4 & -3 & 1 &&&&& \\ 1 & 0 & 3 & -6 & 7 & -4 & 1 &&&& \\ 1 & 1 & -3 & 9 & -13 & 11 & -5 & 1 &&& \\ 1 & 0 & 4 & -12 & 22 & -24 & 16 & -6 & 1 && \\ 1 & 1 & -4 & 16 & -34 & 46 & -40 & 22 & -7 & 1 & \\ 1 & 0 & 5 & -20 & 50 & -80 & 86 & -62 & 29 & -8 & 1 \end{bmatrix}$$

defined by functions $d^{[110]}(t) = \frac{1}{1-t}$ and $h^{[110]}(t) = \frac{t}{1+t}$.

On the other hand, the Riordan array $\mathcal{R}^{[11100]}$, that is $j = 2$

in the family, is the binomial transform of

$$B^{[11100]} = \begin{bmatrix} 1 \\ 1 & 1 \\ 3 & 1 & 1 \\ 5 & 3 & 1 & 1 \\ 15 & 7 & 3 & 1 & 1 \\ 31 & 16 & 9 & 3 & 1 & 1 \\ 87 & 43 & 17 & 11 & 3 & 1 & 1 \\ 201 & 101 & 55 & 18 & 13 & 3 & 1 & 1 \\ 543 & 271 & 119 & 67 & 19 & 15 & 3 & 1 & 1 \\ 1331 & 666 & 341 & 141 & 79 & 20 & 17 & 3 & 1 & 1 \\ 3533 & 1766 & 826 & 411 & 167 & 91 & 21 & 19 & 3 & 1 & 1 \end{bmatrix}$$

defined by functions

$$d^{[11100]}(t) = \sqrt{\frac{1+t}{1-t-5t^2+t^3}} \quad \text{and}$$

$$h^{[11100]}(t) = \frac{1+2t+t^2-\sqrt{(1-t-5t^2+t^3)(1+t)}}{2(1+t)^2};$$

in particular, the latter expands to

$$h^{[11100]}(t) = t+2t^4-t^5+7t^6+24t^8+17t^9+98t^{10}+\mathcal{O}\left(t^{11}\right).$$

Riordan arrays $B^{[p]}$ can be completely defined by using the results of Theorem 45 and the product rule of the Riordan group. Doing so, for each pattern family studied in this work with $j > 1$, the Riordan array $\mathcal{R}^{[p]}$ appears to be the binomial transform of another Riordan array $B^{[p]}$ with non-negative integer coefficients, although it is not easy to spot this property looking at the corresponding h functions because their series expansions might contain negative coefficients, as shown for matrices $B^{[110]}$ and $B^{[11100]}$. This fact could be further investigated from an algebraic and combinatorial point of view and possibly yield interesting combinatorial interpretations also in the case $j > 1$.

4

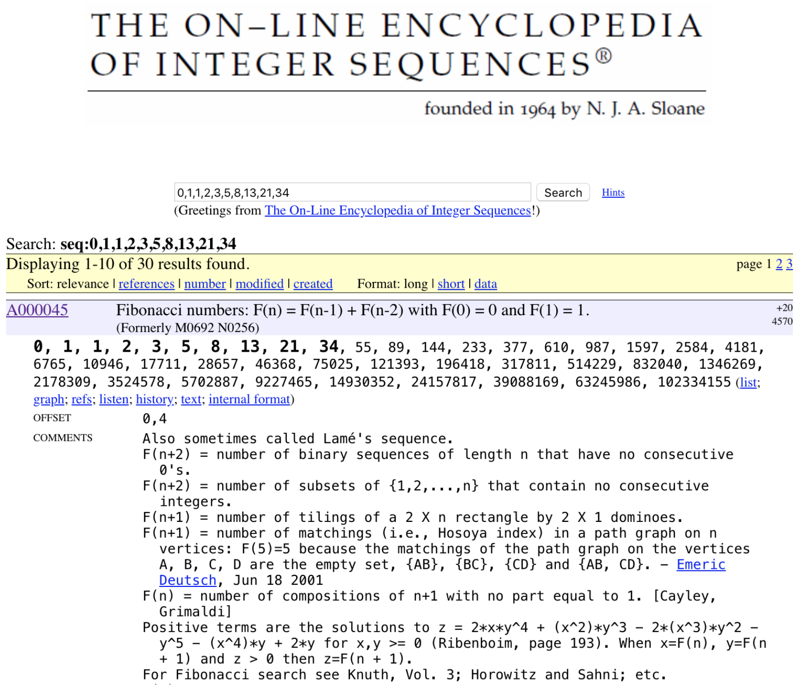
Crawling, (pretty) printing and graphing the OEIS

In this chapter we present a suite of software tools that allows us to interact with the *Online Encyclopedia of Integer Sequences*; in particular, (i) a *crawler* fetches sequences recursively and asynchronously, (ii) a *pretty printer* represents the same data stored in the online archive using two different formats, namely the old UNIX console and modern Jupyter notebooks, (iii) a *grapher* shows connections among sequences by using graph structures.

4.1 Introduction

The *Online Encyclopedia of Integer Sequences* [Sloane] is an online database of sequences of numbers that collects any kind of data regarding them, available at `https://oeis.org/`. It was founded by N. J. A. Sloane in 1964 and since then has been, and continue to be, updated constantly by contributions of many users. Despite of its powerful searching mechanisms, shown in Figure 4.1, we design a parallel *suite of software tools* that satisfies the necessities (i) to search the OEIS offline by automating repeated searches, (ii) to work in a UNIX console in order to use its programming facilities for a more efficient manipulation of textual contents and (iii) to interface with third-party libraries to visualize networks encoding connections among sequences.

A similar approach in the recent literature is [Nguyen and Taggart, 2013] that mines the OEIS for new mathematical identities, discussing how to store, compare and match integer sequences toward the formalization of some conjectures; on the other hand, searching the word "*oeis*" in GitHub returns one hundred repositories, the majority of them (i) host simple implementations of scripts that download data about a desired sequence targeting all major programming languages. Moreover, [Weidmann] is a project that tries to deduce closed


THE ON–LINE ENCYCLOPEDIA OF INTEGER SEQUENCES®

founded in 1964 by N. J. A. Sloane

0,1,1,2,3,5,8,13,21,34 Search Hints

(Greetings from The On-Line Encyclopedia of Integer Sequences!)

Search: **seq:0,1,1,2,3,5,8,13,21,34**

Displaying 1-10 of 30 results found. page 1 2 3

Sort: relevance | references | number | modified | created Format: long | short | data

A000045 Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1. +20 4570
(Formerly M0692 N0256)

```
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181,
6765, 10946, 17711, 28657, 46368, 75025, 121393, 196418, 317811, 514229, 832040, 1346269,
2178309, 3524578, 5702887, 9227465, 14930352, 24157817, 39088169, 63245986, 102334155 (list;
graph; refs; listen; history; text; internal format)
OFFSET          0,4
COMMENTS        Also sometimes called Lamé's sequence.
                F(n+2) = number of binary sequences of length n that have no consecutive
                  0's.
                F(n+2) = number of subsets of {1,2,...,n} that contain no consecutive
                  integers.
                F(n+1) = number of tilings of a 2 X n rectangle by 2 X 1 dominoes.
                F(n+1) = number of matchings (i.e., Hosoya index) in a path graph on n
                  vertices: F(5)=5 because the matchings of the path graph on the vertices
                  A, B, C, D are the empty set, {AB}, {BC}, {CD} and {AB, CD}. - Emeric
                  Deutsch, Jun 18 2001
                F(n) = number of compositions of n+1 with no part equal to 1. [Cayley,
                  Grimaldi]
                Positive terms are the solutions to z = 2*x*y^4 + (x^2)*y^3 - 2*(x^3)*y^2 -
                  y^5 - (x^4)*y + 2*y for x,y >= 0 (Ribenboim, page 193). When x=F(n), y=F(n
                  + 1) and z > 0 then z=F(n + 1).
                For Fibonacci search see Knuth, Vol. 3; Horowitz and Sahni; etc.
```

Figure 4.1: The OEIS search page and results for a query concerning the sequence $(0, 1, 1, 2, 3, 5, 8, 13, 21, 34)$.

formulae that generates a given list of numbers.

Our approach complements the existing ones by providing a *recursive* and *asynchronous* fetching process, vanilla data storage in JSON files and visualization of relations among sequences; the description of each tool is addressed in the following sections, respectively.

The present suite of tools had been shown at an open school on Combinatorial Method in the analysis of Algorithms and Data Structures in Korea [Nocentini, 2017]; moreover, all the sources that implements the applications can be found online in the repository `https://github.com/massimo-nocentini/oeis-tools`.

4.2 The Crawler

The script `crawling.py` implements a bot that given a sequence identifier in the form Axxxxxx, where xs are digits, it issues an HTTP request to the main OEIS server and waits for a response; once it is received, the bot stores data locally and, looking into the response's `xref` section that contains a set of other sequences identifiers, repeats its behaviour on each one of them, recursively. Such a bot is commonly known as *crawler*.

Our implementation features neither threads nor race conditions nor data sync; on the contrary, it targets *pure asynchronous computation* by using *async/await* Python primitives only. The approach is educational and we strive to create a simple but elegant codebase which boils down to 300 lines of Python code; eventually, it allows us to cache portions of the OEIS to speed up repeated lookups and to restart the fetching process from the cache already downloaded.

The script presents a help message to explain itself:

```
$ python3.6 crawling.py -h
usage: crawling.py [-h] [--clear-cache] [--restart] [--workers WORKERS]
                   [--log-level {DEBUG,INFO,WARNING,ERROR,CRITICAL}]
                   [--cache-dir CACHE_DIR] [--progress-mark PROGRESS_MARK]
                   [S [S ...]]

OEIS Crawler.

positional arguments:
  S                     Sequence to fetch, given in the form Axxxxxx

optional arguments:
  -h, --help            show this help message and exit
  --clear-cache         Clear cache of sequences, according to --cache-dir
  --restart             Build fringe from cached sequences (defaults to False)
  --workers WORKERS     Degree of parallelism (defaults to 10)
  --log-level {DEBUG,INFO,WARNING,ERROR,CRITICAL}
                        Logger verbosity (defaults to ERROR)
  --cache-dir CACHE_DIR
                        Cache directory (defaults to ./fetched/)
  --progress-mark PROGRESS_MARK
                        Symbol for fetched event (defaults to ●)
```

Example 57. *We illustrate a typical session where we start from scratch. First of all, we want to download the OEIS content about two important and nice sequences, namely those corresponding to the Fibonacci and Catalan numbers, respectively identified by labels* A000045 *and* A000108*:*

```
$ python3.6 crawling.py A000045 A000108
●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●^C
fetched 30 new sequences:
{'A006480', 'A000045', 'A032443', 'A046161', 'A002390',
 'A137697', 'A000142', 'A000736', 'A094639', 'A176137',
 'A120303', 'A000344', 'A032357', 'A000753', 'A048990',
 'A003519', 'A000108', 'A009766', 'A211611', 'A120274',
 'A008276', 'A129763', 'A094216', 'A106566', 'A003518',
 'A098597', 'A099039', 'A014137', 'A000245', 'A264663'}
```

After stopping the crawler, we check the content of the cache with the commands

```
$ python3.6 crawling.py
30 sequences in cache ./fetched/
289 sequences in fringe for restarting
$ ls fetched/
A000045.json    A000245.json    A000753.json    A003519.json
A009766.json    A032443.json    A094216.json    A099039.json
A120303.json    A176137.json    A000108.json    A000344.json
A002390.json    A006480.json    A014137.json    A046161.json
A094639.json    A106566.json    A129763.json    A211611.json
A000142.json    A000736.json    A003518.json    A008276.json
A032357.json    A048990.json    A098597.json    A120274.json
A137697.json    A264663.json
```

which tell us that 30 *sequences had been fetched and stored in the default directory* `./fetched/`. *Moreover, we can restart the crawler from where it was interrupted with the command*

```
$ python3.6 crawling.py --restart
●●●●●●●●●●●●●●●●●●●●^C
fetched 20 new sequences:
{'A000165', 'A001044', 'A003517', 'A027914', 'A214292', 'A152063',
 'A014138', 'A062103', 'A003422', 'A238717', 'A045520', 'A064062',
 'A144107', 'A045525', 'A007004', 'A002057', 'A244230', 'A099731',
 'A033552', 'A121839'}
```

and we check that new sequences are actually collected,

```
$ python3.6 crawling.py
50 sequences in cache ./fetched/
354 sequences in fringe for restarting
```

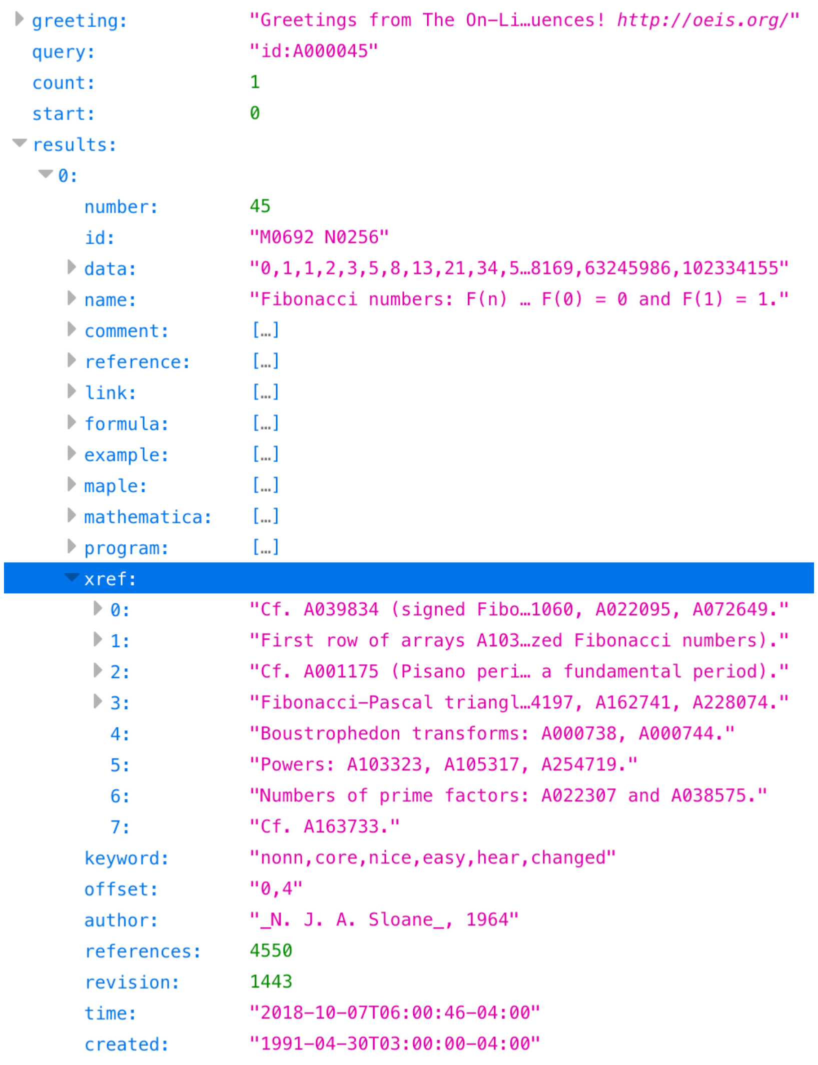
as desired.

Having contents stored in JSON files, whose structure is presented in Figure 4.2, allows us to inspect and manipulate them using every tool available in our working environment, as the next example shows.

Example 58. *Combining the* `cat` *command with the Python module* `json.tool`*, that prints JSON files with respect to indentation, we can visualize data about the sequence of Fibonacci numbers as follows*

```
$ cat fetched/A000045.json | python3.6 -m json.tool
{
    "greeting": "Greetings from The On-Line Encyclopedia of Integer Sequences! http://oeis.org/",
    "query": "id:A000045",
    "count": 1,
    "start": 0,
    "results": [
        {
            "number": 45,
            "id": "M0692 N0256",
            "data": "0,1,1,2,3,5,8,13,21,34,55,89,144,233,377,610,987,1597,2584,4181,6765,10946,
                     17711,28657,46368,75025,121393,196418,317811,514229,832040,1346269,2178309,
                     3524578,5702887,9227465,14930352,24157817,39088169,63245986,102334155",
            "name": "Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.",
            "comment": [
                "Also sometimes called Lam\u00e9's sequence.",
                "F(n+2) = number of binary sequences of length n that have no consecutive 0's.",
                "F(n+2) = number of subsets of {1,2,...,n} that contain no consecutive integers.",
                "F(n+1) = number of tilings of a 2 X n rectangle by 2 X 1 dominoes.",
                ... # more comments here
            ]
            ... # more sections here
        }
    ]
}
```

Our implementation takes strong inspiration from [van Rossum and Davis] and provides the following abstractions:



```
▸ greeting:            "Greetings from The On-Li…uences! http://oeis.org/"
  query:               "id:A000045"
  count:               1
  start:               0
▾ results:
  ▾ 0:
      number:          45
      id:              "M0692 N0256"
    ▸ data:            "0,1,1,2,3,5,8,13,21,34,5…8169,63245986,102334155"
    ▸ name:            "Fibonacci numbers: F(n) … F(0) = 0 and F(1) = 1."
    ▸ comment:         […]
    ▸ reference:       […]
    ▸ link:            […]
    ▸ formula:         […]
    ▸ example:         […]
    ▸ maple:           […]
    ▸ mathematica:     […]
    ▸ program:         […]
    ▾ xref:
      ▸ 0:             "Cf. A039834 (signed Fibo…1060, A022095, A072649."
      ▸ 1:             "First row of arrays A103…zed Fibonacci numbers)."
      ▸ 2:             "Cf. A001175 (Pisano peri… a fundamental period)."
      ▸ 3:             "Fibonacci-Pascal triangl…4197, A162741, A228074."
        4:             "Boustrophedon transforms: A000738, A000744."
        5:             "Powers: A103323, A105317, A254719."
        6:             "Numbers of prime factors: A022307 and A038575."
        7:             "Cf. A163733."
      keyword:         "nonn,core,nice,easy,hear,changed"
      offset:          "0,4"
      author:          "_N. J. A. Sloane_, 1964"
      references:      4550
      revision:        1443
      time:            "2018-10-07T06:00:46-04:00"
      created:         "1991-04-30T03:00:00-04:00"
```

Figure 4.2: The complete structure of the JSON encoding of OEIS content about the sequence A000045; in particular, the property `xref`, highlighted in blue, is left expanded because of its importance in the recursive behaviour of the crawler, namely every label in this section bacomes a candidate sequence for the fetching process.

`reader` objects, that have the responsibility to be *asynchronous iterators* in the sense that they have to respond to the message `__anext__`, where the computation waits asynchronously for incoming data from the `self.read` coroutine. The following code implements the description precisely,

```
class reader:

    def __init__(self, read):
        self.read = read

    def __aiter__(self):
        return self

    async def __anext__(self):
        chunk = await self.read()
        if chunk:   return chunk
        else:       raise StopAsyncIteration
```

`fetcher` objects have the responsibilities (i) to create a socket with OEIS server, (ii) to establish a working connection, (iii) to send an HTTP `GET` request for the desired sequence, (iv) to wait for the fetching process completes and (v) to close the socket and signal that the it ends successfully. A literal translation follows,

```
class fetcher:

    def __init__(   self, url,
                    resource_key=lambda resource: resource,
                    done=lambda url, content: print(content)):

        self.url = url
        self.response = b''
        self.sock = None
        self.done = done
        self.resource_key=lambda: resource_key(self.url.resource)

    def encode_request(self, encoding='utf8'):

        request = 'GET {} HTTP/1.0\r\nHost: {}\r\n\r\n'.format(
                self.resource_key(), self.url.host)

        return request.encode(encoding)

    async def fetch(self):

        self.sock = socket.socket()
        self.sock.setblocking(False)

        await loop.sock_connect(self.sock, address=(self.url.host, self.url.port))

        logger.info('Connection established with {} asking resource {}'.format(
                self.url.host, self.url.resource))

        await loop.sock_sendall(self.sock, self.encode_request())

        self.response = await self.read_all()
        self.sock.close()

        return self.done(self.url, self.response.decode('utf8'))

    async def read(self, nbytes=4096):

        chunk = await loop.sock_recv(self.sock, nbytes)
        return chunk

    async def read_all(self):

        response = [chunk async for chunk in reader(self.read)]
        return b''.join(response)
```

`crawler` objects have the responsibilities (i) to keep a queue of task, one for each candidate sequence, (ii) to put each ready task into the scheduling process and (iii) to reclaim memory for the completed ones and (iv) to deque them, eventually; again, its code follows

```
class crawler:

    def __init__(self, resources, fetcher_factory, max_tasks):

        self.resources = resources
        self.max_tasks = max_tasks
        self.fetcher_factory = fetcher_factory
        self.q = asyncio.Queue()

    async def crawl(self):

        for res in self.resources: self.q.put_nowait(res)

        tasks = [loop.create_task(coro=self.work()) for _ in range(self.max_tasks)]

        await self.q.join()

        for t in tasks: t.cancel()


    async def work(self):

        while True:

            resource = await self.q.get()

            await self.fetcher_factory(resource, appender=self.q.put_nowait).fetch()

            self.q.task_done()
```

4.3 The (Pretty) Printer

The script `pprinting.py` provides a proxy for searching into the OEIS, therefore it shows exactly the same contents you see from usual web interface on `http://oeis.org`; additionally, it provides (i) tabular representations of `data` sections in *one* and *two* dimensions using *list* and *matrix* notations, respectively, (ii) filtering capabilities on most response's sections and (iii) interoperability with the crawler tool by taking advantage of cached sequences.

The script presents a help message to explain itself:

```
$ python3.6 pprinting.py -h
usage: pprinting.py [-h]
                    (--id ID | --seq SEQ | --query QUERY | --most-recents M)
                    [--force-fetch] [--cache-dir CACHE_DIR] [--tables-only]
                    [--start-index S] [--max-results R] [--data-only]
                    [--upper-limit U] [--comment-filter C]
                    [--formula-filter F] [--xrefs-filter X] [--link-filter L]
                    [--cite-filter R] [--console-width W]

OEIS Pretty Printer.

optional arguments:
  -h, --help            show this help message and exit
  --id ID               Sequence id, given in the form Axxxxxx
  --seq SEQ             Literal sequence, ordered '[...]' or presence '{...}'
  --query QUERY         Open query for plain search, in the form '...'
  --most-recents M      Print the most recent sequences ranking by M in ACCESS
                        or MODIFY, looking into --cache-dir, at most --max-
                        results (defaults to None)
  --force-fetch         Bypass cache fetching again, according to --cache-dir
                        (defaults to False)
  --cache-dir CACHE_DIR
                        Cache directory (defaults to ./fetched/)
  --tables-only         Print matrix sequences only (defaults to False)
  --start-index S       Start from result at rank position S (defaults to 0)
  --max-results R       Pretty print the former R <= 10 results (defaults to 10)
  --data-only           Show only data repr and preamble (defaults to False)
  --upper-limit U       Upper limit for data repr: U is a dict '{"list":i,
                        "table":(r, c)}' where i, r and c are ints (defaults
                        to i=15, r=10 and c=10), respectively)
  --comment-filter C    Apply filter C to comments, where C is Python `lambda`
                        predicate 'lambda i,c: ...' referring to i-th comment c
  --formula-filter F    Apply filter F to formulae, where F is Python `lambda`
                        predicate 'lambda i,f: ...' referring to i-th formula f
  --xrefs-filter X      Apply filter X to cross refs, where X is Python
                        `lambda` predicate 'lambda i,x: ...' referring to i-th xref x
  --link-filter L       Apply filter L to links, where L is Python `lambda`
                        predicate 'lambda i,l: ...' referring to i-th link l
  --cite-filter R       Apply filter R to citation, where R is Python `lambda`
                        predicate 'lambda i,r: ...' referring to i-th citation r
  --console-width W     Console columns (defaults to 72)
```

In the next examples we show how `pprinting`'s facilities can be used to apply filters, to print data-only visualization and to search by an open query, respectively.

Example 59. *Typing the following command into a shell, it outputs on the* `stdout` *the pretty-printed contents about the sequence of Fibonacci numbers, with two filters applied that show comments made by prof. Barry and the first* 5 *formulae only,*

```
$ python3.6 pprinting.py                        \
    --id A000045                                \
    --comment-filter 'lambda i,c: "Barry" in c' \
    --formula-filter 'lambda i,f: i < 5'

 A000045 - Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and
  F(1) = 1.

by _N. J. A. Sloane_, 1964

_Keywords_: `nonn,core,nice,easy,hear,changed`

_Data_:
[0  1  1  2  3  5  8  13  21  34  55  89  144  233  377]

_Comments_:
    ● F(n+2) = Sum_{k=0..n} binomial(floor((n+k)/2),k), row sums of
      A046854. - _Paul Barry_, Mar 11 2003

_Formulae_:
    ● G.f.: x / (1 - x - x^2).
    ● G.f.: Sum_{n>=0} x^n * Product_{k=1..n} (k + x)/(1 + k*x). - _Paul
      D. Hanna_, Oct 26 2013
    ● F(n) = ((1+sqrt(5))^n - (1-sqrt(5))^n)/(2^n*sqrt(5)).
    ● Alternatively, F(n) = ((1/2+sqrt(5)/2)^n -
      (1/2-sqrt(5)/2)^n)/sqrt(5).
    ● F(n) = F(n-1) + F(n-2) = -(-1)^n F(-n).

_Cross references_:
    ● Cf. A039834 (signed Fibonacci numbers), A001690 (complement),
      A000213, A000288, A000322, A000383, A060455, A030186, A020695,
      A020701, A071679, A099731, A100492, A094216, A094638, A000108,
      A101399, A101400, A001611, A000071, A157725, A001911, A157726,
      A006327, A157727, A157728, A157729, A167616, A059929, A144152,
      A152063, A114690, A003893, A000032, A060441, A000930, A003269,
      A000957, A057078, A007317, A091867, A104597, A249548, A262342,
      A001060, A022095, A072649.
    ● First row of arrays A103323, A234357. Second row of arrays
      A099390, A048887, and A092921 (k-generalized Fibonacci numbers).
    ● a(n) = A094718(4, n). a(n) = A101220(0, j, n).
    ● a(n) = A090888(0, n+1) = A118654(0, n+1) = A118654(1, n-1) =
      A109754(0, n) = A109754(1, n-1), for n > 0.
    ● Fibonacci-Pascal triangles: A027926, A036355, A037027, A074829,
      A105809, A109906, A111006, A114197, A162741, A228074.
    ● Boustrophedon transforms: A000738, A000744.
    ● Powers: A103323, A105317, A254719.
    ● Numbers of prime factors: A022307 and A038575.
    ● Cf. A163733.
```

other sections, such as `reference` *and* `link`*, are hidden by default to provide a cleaner output.*

Example 60. *The following command pretty prints (i) the first 3 sequences from our current cache –the result may vary if you try on your own machine–, (ii) ranking them according to the most recent access time, (iii) reporting data only and (iv) limiting up to* 10 *coefficients for linear sequences:*

```
$ python3.6 pprinting.py        \
    --most-recent ACCESS        \
    --data-only                 \
    --max-results 3             \
    --upper-limit '{"list":10}'

 A001044 - a(n) = (n!)^2.

by _N. J. A. Sloane_, _R. K. Guy_

_Keywords_: `nonn,easy,nice`

_Data_:
[1  1  4  36  576  14400  518400  25401600  1625702400  131681894400]
```

```
 A048990 - Catalan numbers with even index (A000108(2*n), n >= 0): a(n)
  = binomial(4*n, 2*n)/(2*n+1).

by _Wolfdieter Lang_

_Keywords_: `easy,nonn`

_Data_:
[1  2  14  132  1430  16796  208012  2674440  35357670  477638700]
```

```
 A014138 - Partial sums of (Catalan numbers starting 1, 2, 5, ...).

by _N. J. A. Sloane_

_Keywords_: `nonn,nice`

_Data_:
[0  1  3  8  22  64  196  625  2055  6917]
```

Example 61. *The following command pretty prints the responses about the open query "*`pascal triangle`*", using* 2*-dimension representation for matrices in* `data` *sections, and reports the first* 2 *sequences only,*

```
$ python3.6 pprinting.py
    --query 'pascal triangle'   \
    --tables-only               \
    --data-only                 \
    --max-results 2

 A007318 - Pascal's triangle read by rows: C(n,k) = binomial(n,k) =
  n!/(k!*(n-k)!), 0 <= k <= n.

by _N. J. A. Sloane_ and _Mira Bernstein_, Apr 28 1994

_Keywords_: `nonn,tabl,nice,easy,core,look,hear,changed`

_Data_:
⎡1  0  0   0    0    0   0   0   0  0⎤
⎢1  1  0   0    0    0   0   0   0  0⎥
⎢1  2  1   0    0    0   0   0   0  0⎥
⎢1  3  3   1    0    0   0   0   0  0⎥
```

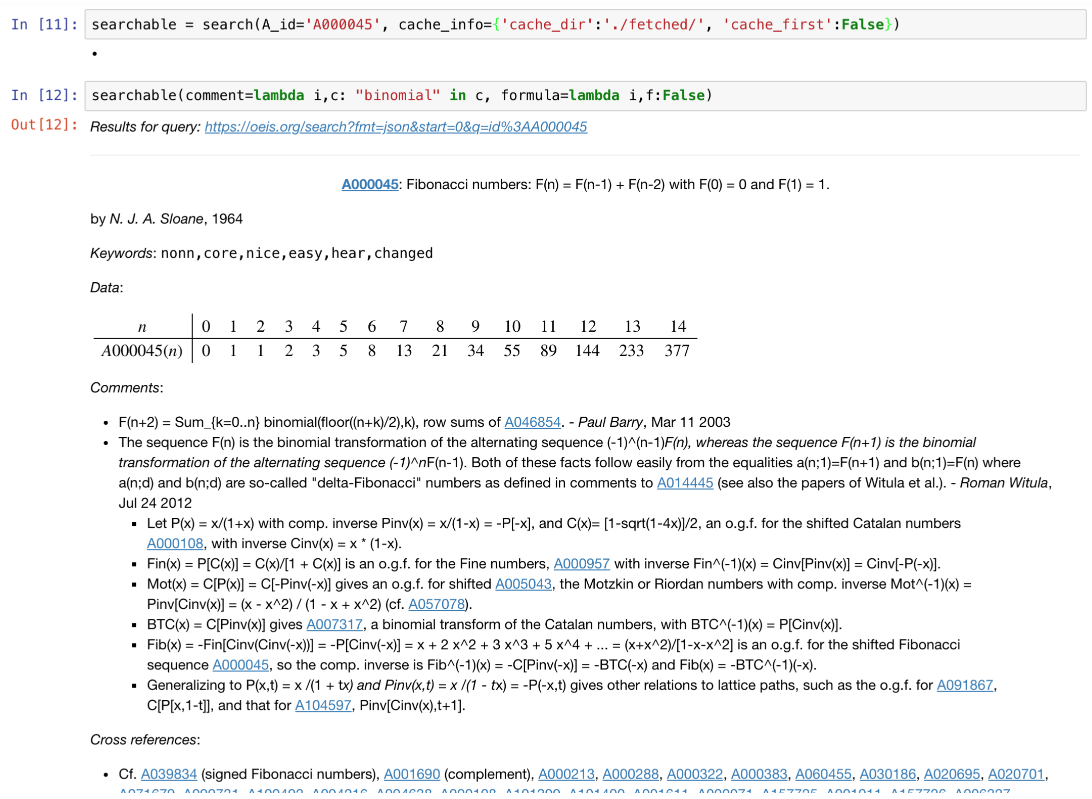
```
⎢1  4  6   4    1    0    0   0   0  0⎥
⎢1  5  10  10   5    1    0   0   0  0⎥
⎢1  6  15  20  15    6    1   0   0  0⎥
⎢1  7  21  35  35   21    7   1   0  0⎥
⎢1  8  28  56  70   56   28  8   1  0⎥
⎣1  9  36  84  126  126  84  36  9  1⎦

________________________________________________________________________

 A047999 - Sierpiński's [Sierpinski's] triangle (or gasket): triangle,
  read by rows, formed by reading Pascal's triangle mod 2.

by _N. J. A. Sloane_

_Keywords_: `nonn,tabl,easy,nice`

_Data_:
⎡1  0  0  0  0  0  0  0  0  0⎤
⎢1  1  0  0  0  0  0  0  0  0⎥
⎢1  0  1  0  0  0  0  0  0  0⎥
⎢1  1  1  1  0  0  0  0  0  0⎥
⎢1  0  0  0  1  0  0  0  0  0⎥
⎢1  1  0  0  1  1  0  0  0  0⎥
⎢1  0  1  0  1  0  1  0  0  0⎥
⎢1  1  1  1  1  1  1  1  0  0⎥
⎢1  0  0  0  0  0  0  0  1  0⎥
⎣1  1  0  0  0  0  0  0  1  1⎦
```

In parallel of the terminal interface, we develop pretty printing functions that integrates in Jupyter notebooks. The aim remains the same, namely to present contents taken from the OEIS targeting a different environment that accepts their representation; this is the time of a dynamic web interface that allows us to evaluate Python code on the fly. Using the Markdown language (`https://daringfireball.net/projects/markdown/`) to write textual content, we propose another view of the same data, as shown in Figures 4.3, 4.4 and 4.5; in particular, we take advantage of (i) hyper-references to make labels of sequences clickable to quickly visit them, (ii) font styles to emphasize words in italics and bold-face and (iii) to render math expressions properly, such as 2-dimensional array representation for matrices.

4.4 *The Grapher*

The script `graphing.py` allows us to represent networks where vertices are sequences and edges are connections among them, according to `xref` sections in their JSON encodings. It integrates with the crawler tool by parsing the fetched files and creates `Graph` objects, defined in the Python module `networkx`, having different layouts according to a set of drawing algorithms.

It presents a help message to explain itself:



```
In [11]: searchable = search(A_id='A000045', cache_info={'cache_dir':'./fetched/', 'cache_first':False})
         •

In [12]: searchable(comment=lambda i,c: "binomial" in c, formula=lambda i,f:False)
Out[12]:
```

Results for query: https://oeis.org/search?fmt=json&start=0&q=id%3AA000045

A000045: Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.

by *N. J. A. Sloane*, 1964

Keywords: `nonn,core,nice,easy,hear,changed`

Data:

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A000045(n)	0	1	1	2	3	5	8	13	21	34	55	89	144	233	377

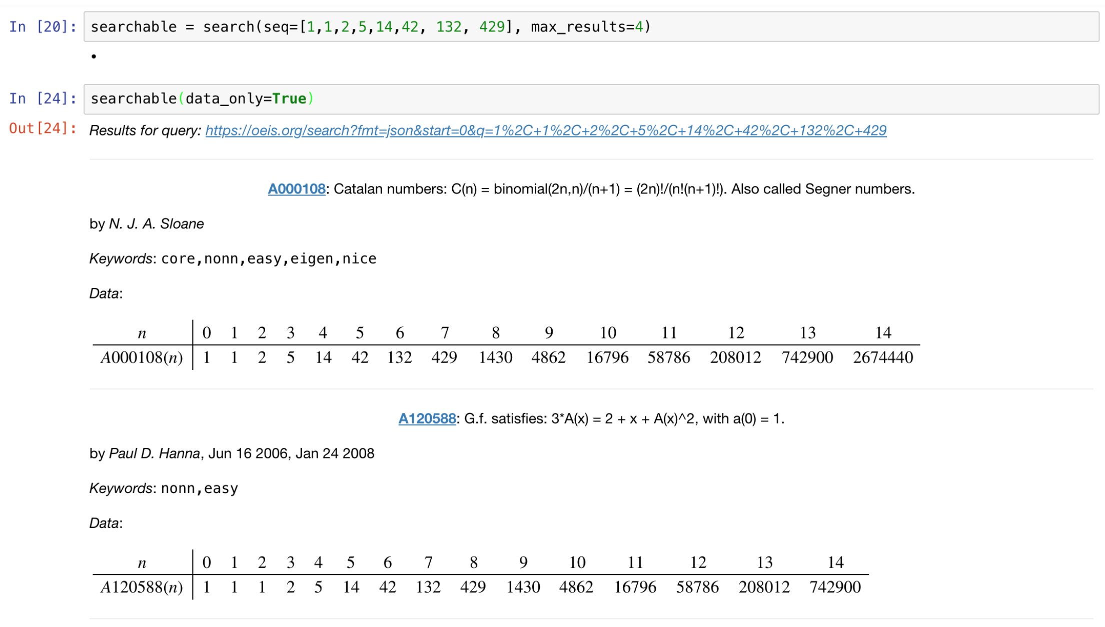
Comments:

- F(n+2) = Sum_{k=0..n} binomial(floor((n+k)/2),k), row sums of A046854. - *Paul Barry*, Mar 11 2003
- The sequence F(n) is the binomial transformation of the alternating sequence (-1)^(n-1)*F(n), whereas the sequence F(n+1) is the binomial transformation of the alternating sequence (-1)*^nF(n-1). Both of these facts follow easily from the equalities a(n;1)=F(n+1) and b(n;1)=F(n) where a(n;d) and b(n;d) are so-called "delta-Fibonacci" numbers as defined in comments to A014445 (see also the papers of Witula et al.). - *Roman Witula*, Jul 24 2012
  - Let P(x) = x/(1+x) with comp. inverse Pinv(x) = x/(1-x) = -P[-x], and C(x)= [1-sqrt(1-4x)]/2, an o.g.f. for the shifted Catalan numbers A000108, with inverse Cinv(x) = x * (1-x).
  - Fin(x) = P[C(x)] = C(x)/[1 + C(x)] is an o.g.f. for the Fine numbers, A000957 with inverse Fin^(-1)(x) = Cinv[Pinv(x)] = Cinv[-P(-x)].
  - Mot(x) = C[P(x)] = C[-Pinv(-x)] gives an o.g.f. for shifted A005043, the Motzkin or Riordan numbers with comp. inverse Mot^(-1)(x) = Pinv[Cinv(x)] = (x - x^2) / (1 - x + x^2) (cf. A057078).
  - BTC(x) = C[Pinv(x)] gives A007317, a binomial transform of the Catalan numbers, with BTC^(-1)(x) = P[Cinv(x)].
  - Fib(x) = -Fin[Cinv(Cinv(-x))] = -P[Cinv(-x)] = x + 2 x^2 + 3 x^3 + 5 x^4 + ... = (x+x^2)/[1-x-x^2] is an o.g.f. for the shifted Fibonacci sequence A000045, so the comp. inverse is Fib^(-1)(x) = -C[Pinv(-x)] = -BTC(-x) and Fib(x) = -BTC^(-1)(-x).
  - Generalizing to P(x,t) = x /(1 + t*x) and Pinv(x,t) = x /(1 - t*x) = -P(-x,t) gives other relations to lattice paths, such as the o.g.f. for A091867, C[P[x,1-t]], and that for A104597, Pinv[Cinv(x),t+1].

Cross references:

- Cf. A039834 (signed Fibonacci numbers), A001690 (complement), A000213, A000288, A000322, A000383, A060455, A030186, A020695, A020701,

Figure 4.3: This screenshot shows search results about the Fibonacci numbers where (i) the section about comments is filtered such that the word "*binomial*" has to appear in their text and (ii) the section about formulae is hidden.



```
In [20]: searchable = search(seq=[1,1,2,5,14,42, 132, 429], max_results=4)
         •

In [24]: searchable(data_only=True)
Out[24]:
```

Results for query: https://oeis.org/search?fmt=json&start=0&q=1%2C+1%2C+2%2C+5%2C+14%2C+42%2C+132%2C+429

A000108: Catalan numbers: C(n) = binomial(2n,n)/(n+1) = (2n)!/(n!(n+1)!). Also called Segner numbers.

by *N. J. A. Sloane*

Keywords: `core,nonn,easy,eigen,nice`

Data:

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A000108(n)	1	1	2	5	14	42	132	429	1430	4862	16796	58786	208012	742900	2674440

A120588: G.f. satisfies: 3*A(x) = 2 + x + A(x)^2, with a(0) = 1.

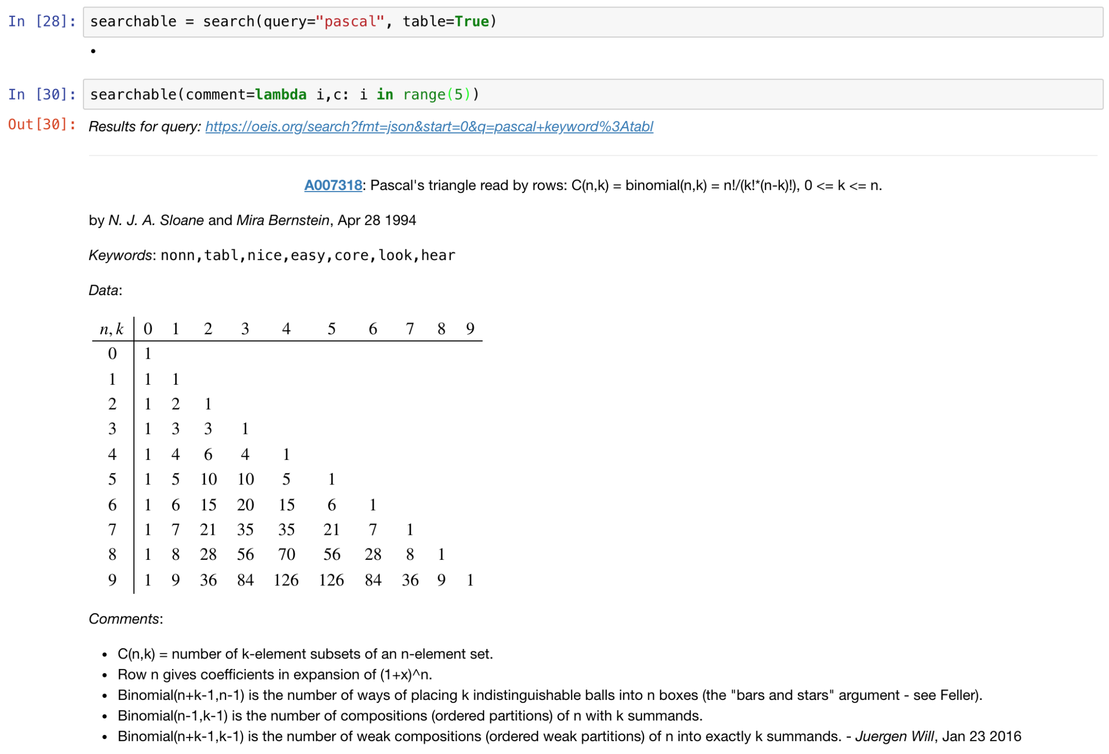
by *Paul D. Hanna*, Jun 16 2006, Jan 24 2008

Keywords: `nonn,easy`

Data:

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A120588(n)	1	1	1	2	5	14	42	132	429	1430	4862	16796	58786	208012	742900

Figure 4.4: This screenshot shows search results of a query using a subsequence, showing *data* sections only.



```
In [28]: searchable = search(query="pascal", table=True)
         .

In [30]: searchable(comment=lambda i,c: i in range(5))
Out[30]: Results for query: https://oeis.org/search?fmt=json&start=0&q=pascal+keyword%3Atabl
```

[A007318](https://oeis.org/A007318): Pascal's triangle read by rows: C(n,k) = binomial(n,k) = n!/(k!*(n-k)!), 0 <= k <= n.

by *N. J. A. Sloane* and *Mira Bernstein*, Apr 28 1994

Keywords: `nonn,tabl,nice,easy,core,look,hear`

Data:

n, k	0	1	2	3	4	5	6	7	8	9
0	1									
1	1	1								
2	1	2	1							
3	1	3	3	1						
4	1	4	6	4	1					
5	1	5	10	10	5	1				
6	1	6	15	20	15	6	1			
7	1	7	21	35	35	21	7	1		
8	1	8	28	56	70	56	28	8	1	
9	1	9	36	84	126	126	84	36	9	1

Comments:

- C(n,k) = number of k-element subsets of an n-element set.
- Row n gives coefficients in expansion of (1+x)^n.
- Binomial(n+k-1,n-1) is the number of ways of placing k indistinguishable balls into n boxes (the "bars and stars" argument - see Feller).
- Binomial(n-1,k-1) is the number of compositions (ordered partitions) of n with k summands.
- Binomial(n+k-1,k-1) is the number of weak compositions (ordered weak partitions) of n into exactly k summands. - *Juergen Will*, Jan 23 2016

Figure 4.5: This screenshot shows search results of an open query using the ”*pascal*” keyword, representing the *data* section as a 2-dimensional array.

```
$ python3.6 graphing.py -h
usage: graphing.py [-h] [--directed] [--cache-dir CACHE_DIR]
                   [--graphs-dir GRAPHS_DIR] [--dpi DPI] [--layout LAYOUT]
                   F

OEIS grapher.

positional arguments:
  F                     Save image in file F.

optional arguments:
  -h, --help            show this help message and exit
  --directed            Draw directed edges
  --cache-dir CACHE_DIR
                        Cache directory (defaults to ./fetched/)
  --graphs-dir GRAPHS_DIR
                        Graphs directory (defaults to ./graphs/)
  --dpi DPI             Resolution in DPI (defaults to 600)
  --layout LAYOUT       Graph layout, choose from: {RANDOM, CIRCULAR, SHELL,
                        FRUCHTERMAN-REINGOLD, SPRING, SPECTRAL} (defaults to
                        SHELL)
```

Example 62. *The following command draws the graph shown in Figure 4.6, where the width of each vertex grows according to the number of its incoming connections,*

```
$ python3.6 graphing.py --layout FRUCHTERMAN-REINGOLD graph.png
```

in order to emphasize most referenced sequences.

Moreover, it can extract essential data from the whole set of JSON files, such as the list of vertices and edges, to inter-

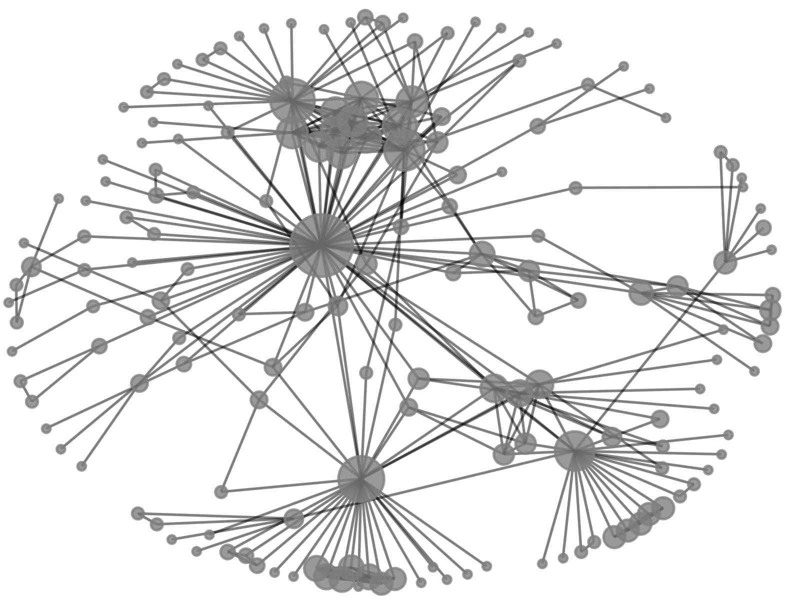


Figure 4.6: Sequences network where vertices are emphasized according to the number of incoming connections.

face with third-party software tools that provide different visualizations; in particular, libraries using the *Javascript* programming language are very powerful and the output they produce are very expressive. For our purposes, we use the `arborjs` library (freely available at `http://arborjs.org/`) to display two additional graphs described in the next two examples, respectively.

Example 63. *Figure 4.7 reports a new unlabeled graph that shows the underlying structure of sequences connections. Here, the layout spreads vertices such that the ones having many outgoing connections are centered, while those having poor connectivity are left on borders.*

Under the hood, the Fibonacci and Catalan numbers are the two central sequences and both of them have an orbit which contains a set of highly connected sequences.

Example 64. *On the other hand, Figure 4.8 adds labels and colors to vertices in order to spot their identity and their relevance according to a combination of their properties. In particular, each color is represented by an RGB tuple that gets weigths (i) the number of comments and formulae for red, (ii) the number of references and links for green and (iii) the number of incoming and outgoing connections for blue, respectively. Moreover, we get the complement to* 255 *of each component because many sequences have not so many details and this manipulation allows us to obtain cleaner and more expressive graphs.*

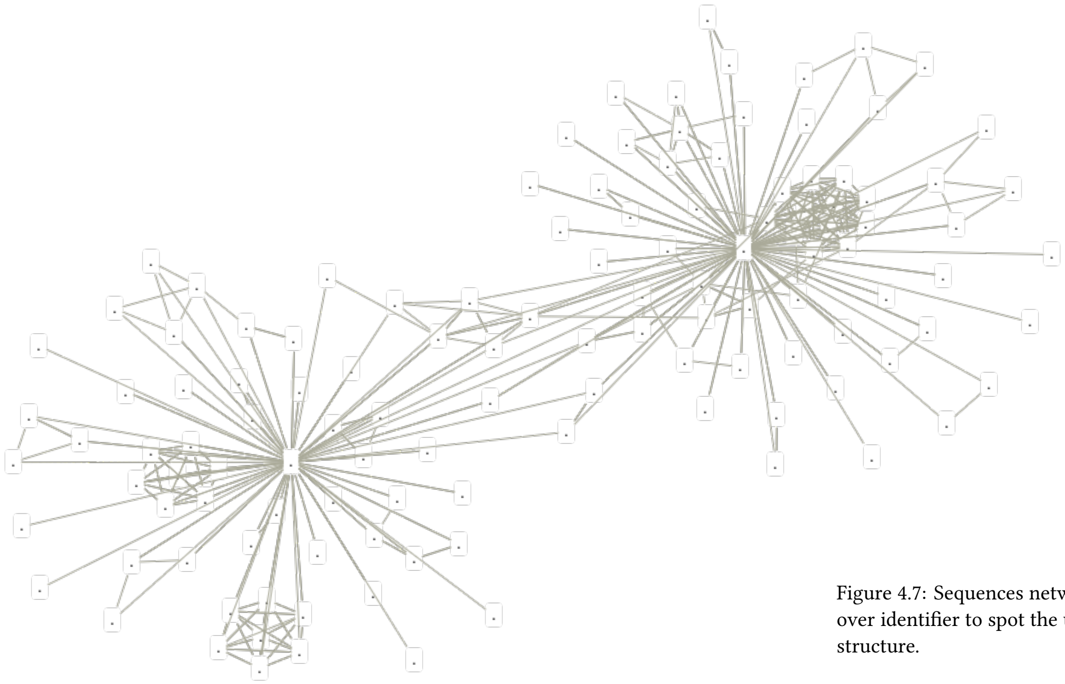


Figure 4.7: Sequences network abstracting over identifier to spot the underlying structure.

For the sake of clarity, the two sequences in evidence are the Fibonacci and Catalan numbers, the former has the color $(006100)_{16}$ *and its complement* $(\mathrm{FFFFFF})_{16} - (006100)_{16} = (\mathrm{FF9EFF})_{16}$ *means that it has many comments, formulae and connections; the latter has the color* $(7\mathrm{C00E5})_{16}$ *and its complement* $(\mathrm{FFFFFF})_{16} - (7\mathrm{C00E5})_{16} = (83\mathrm{FF1A})_{16}$ *means that it has lots of comments, links and references.*

Remark 65. *Recall that the interpretations given in the previous examples concern a subset of the OEIS only, in particular the one fetched in our session; finally, the more we crawl, the more graphs are effective and accurate.*

Example 66. *Finally, crawling for a while to get more sequences, we represent their connections in Figure 4.9, arranging them using a circular layout and we emphasize vertices in the dominating set using the red color.*

Conclusions

This chapter presents a suite of tools that interacts with the *Online Encyclopedia of Integer Sequences*, whose primary goal is to automate simple and repetitive operations such

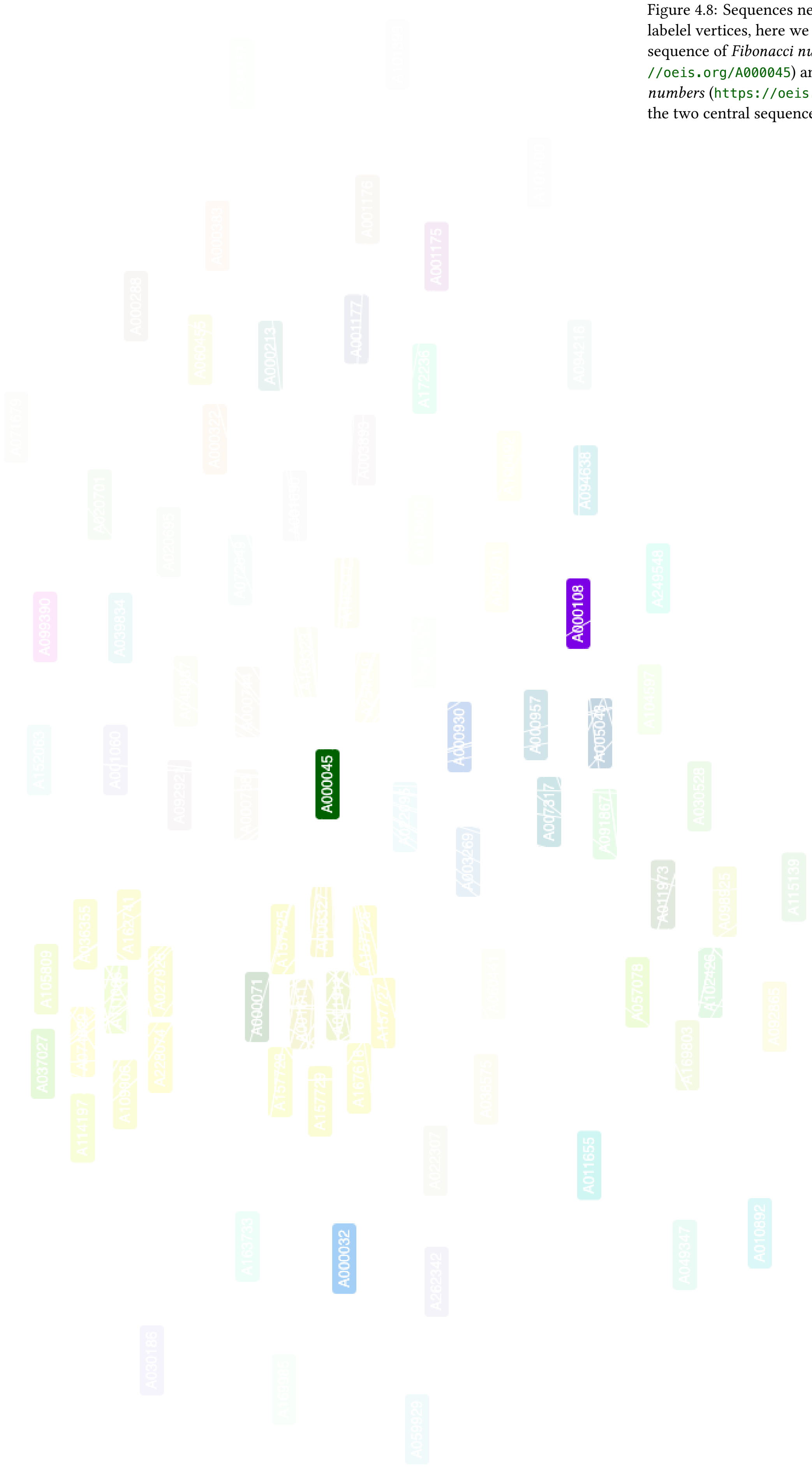


Figure 4.8: Sequences network with labelel vertices, here we see that the sequence of *Fibonacci numbers* (https://oeis.org/A000045) and of *Catalan numbers* (https://oeis.org/A000108) are the two central sequences, respectively.

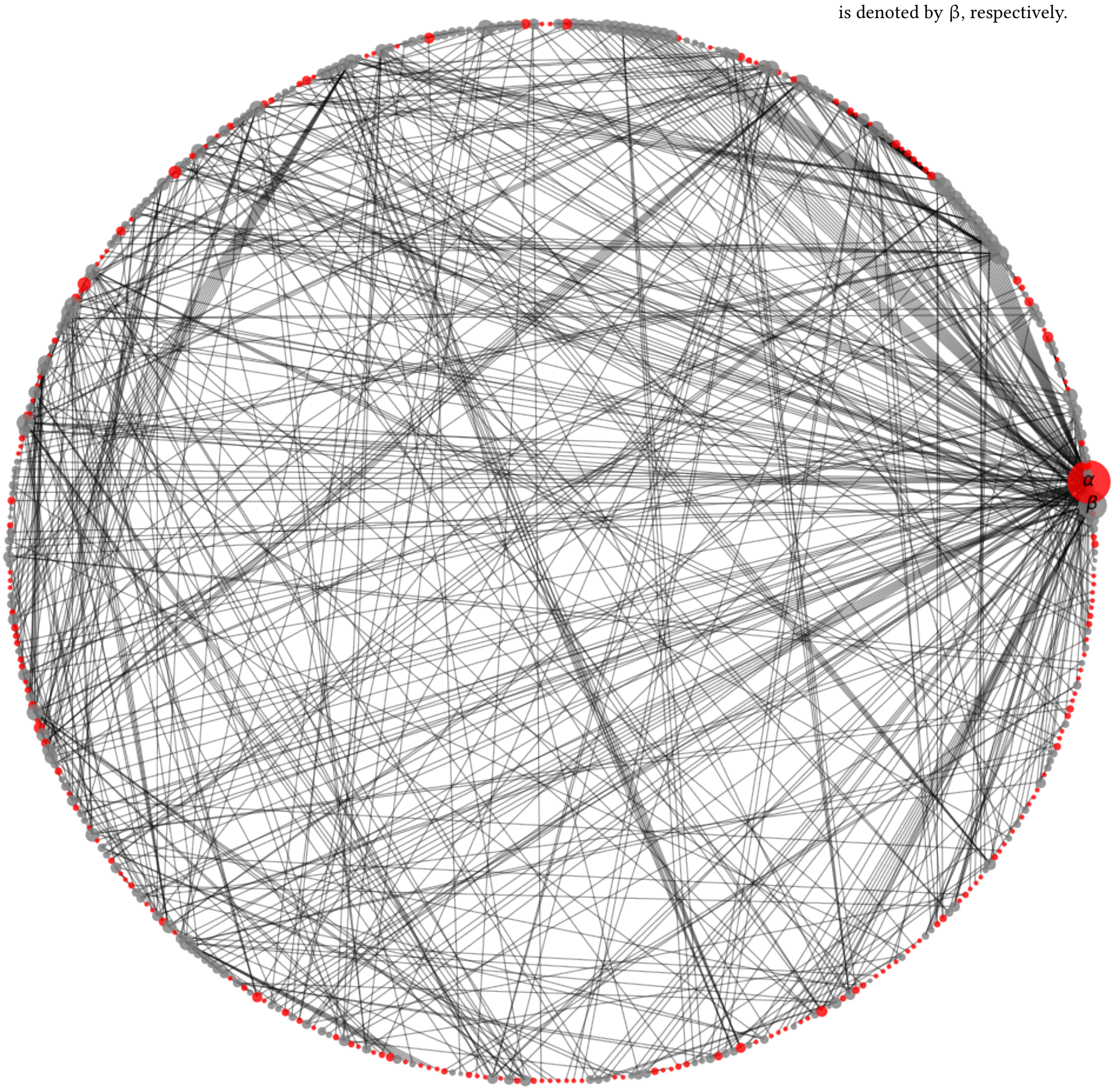


Figure 4.9: A bigger sequences network composed of 419 sequences; here the sequence of *Fibonacci numbers* is denoted by α and the sequence of *Catalan numbers* is denoted by β, respectively.

as (i) crawling sequences to hold a local copy stored in JSON files, (ii) pretty printing data with filtering capabilities, both in the terminal and in Jupyter (`http://jupyter.org/`) notebooks and (iii) to visualize connections among sequences using graphs.

In parallel, this suite has been though to be open to extension and to interface with the hosting environment, UNIX in particular. For instance, the printer can be used in pipe with the `less` command to gain scroll and search features for free or the grapher can be augmented to generate more detailed graph descriptions to be processed by visualization tools.

An additional work direction is to make graphs interactive, namely to tie together the crawler and the grapher in a web-browser interface such that a click on a vertex triggers the execution of the fetching process (unless it has been downloaded already) and the new connections are added to the network dynamically.

5

Queens, tilings, ECO and polyominoes

This chapter solves placement and tiling problems by means of the *backtracking* programming technique. Our approach has an educational component in the sense that we aim to code as clean as possible, relying on *bitmasking* manipulation to balance efficiency drawback due to vanilla implementations. We will tackle the 8-Queens problem, tilings using *pentominoes* and *parallelogram polyominoes*; for what concerns placement problems, we will show an implementation of the *ECO* methodology in order to enumerate classes of objects that obeys particular symbolic equations.

First of all, we introduce basic bitwise tricks and programming idioms that will be useful for the understanding of the upcoming content, which heavily lies on those techniques for the sake of efficency:

```
def is_on(S, j):
    return (S & (1 << j)) >> j

def set_all(n):
    return (1 << n) - 1

def low_bit(S):
    return (S & (-S)).bit_length() - 1

def clear_bit(S, j):
    return S & ~(1 << j)
```

```
>>> S = 0b101010
>>> is_on(S, 3), is_on(S, 2)
(1, 0)

>>> bin(set_all(10))
'0b1111111111'

>>> low_bit(0b100101)
0
>>> low_bit(0b100100)
2
>>> low_bit(1 << 5)
5

>>> S = 0b101010
>>> bin(clear_bit(S, 1))
'0b101000'
```

Table 5.1: Uses of bitmasking functions.

Many other techniques can be found in [Warren, 2012].

5.1 The n-Queens problem

The n-Queens problem is a well known problem in computer science and it is often used as a ”benchmark” to test efficiency of new heuristics and approaches; many resources talks about it, see the survey [Bell and Stevens, 2009]. In this section we provide a pythonic implementation of an algorithm using the idea described in Chapter 3 of [Ruskey, 2003].

We use three *bit masks*, namely integers, to represent whether a row, a raising $\nearrow$ and a falling $\searrow$ diagonals are *under attack* by an already placed queen, instead of three boolean arrays. It is sufficient to use *one* bit only to represent that a cell on a diagonal is under attack, hence to each diagonal is associated one bit according to:

- if such diagonal is *raising*, call it $d_{\nearrow}$, then

  $$a_{r_1,c_1} \in d_{\nearrow} \land a_{r_2,c_2} \in d_{\nearrow} \quad \text{if and only if} \quad r_1 + c_1 = r_2 + c_2;$$

  in words, the sum of the row and column indices is constant along raising diagonals. Therefore diagonal $d_{\nearrow}$ is associated to the bit in position $r_1 + c_1$ of a suitable bitmask p;

- otherwise, if such diagonal is *falling*, call it $d_{\searrow}$, then

  $$a_{r_1,c_1} \in d_{\searrow} \land a_{r_2,c_2} \in d_{\searrow} \quad \text{if and only if} \quad c_1 - r_1 = c_2 - r_2;$$

  in words, the difference of the column and row indices is constant along falling diagonals. Therefore diagonal $d_{\searrow}$ is associated to the bit in position $c_1 - r_1$ of a suitable bitmask p. In order to be consistent, if $c_1 - r_1 < 0$ then take the difference modulo $2n - 1$, where n is the number of rows (and columns), formally

  $$p_{-1}\, p_{-2}\, \ldots\, p_{-(n-1)} \rightarrow_{\equiv_{2n-1}} p_{2n-2}\, p_{2n-3}\, \ldots\, p_n$$

  which entails that

  $$\left(p_{n-1}\, p_{n-2}\, \ldots\, p_0 p_{-1}\, p_{-2}\, \ldots\, p_{-(n-1)}\right)_2$$

  equals

  $$\left(p_{2n-2}\, p_{2n-3}\, \ldots\, p_n p_{n-1}\, p_{n-2}\, \ldots\, p_0\right)_2,$$

where rows and cols indexes range in $\{0, \ldots, n-1\}$; in both cases, it is necessary a bitmask $2n - 1$ bits long.

The function queens is a Python generator of solutions for the n-Queens problem,

```
def queens(n):

    sol = [0] * n                                          # Initialize the permutation that has
                                                           # *columns* indices and *rows* as values.
    def gen(c, rows, raises, falls):

        for r in range(n):                                 # For each row index:

            raising = c + r                                # when r > c negative positions appear in the
            falling = (c - r) % (2*n-1)                    # *most significant part* of `falling`.

            if (is_on(rows, r)                             # if there is no queen on the same row and on
                and is_on(raises, raising)                 # the same raising and falling diagonals,
                and is_on(falls, falling)):                # then `r,c` is a candidate position.

                sol[c] = r                                 # remember the choice of `r`.

                if c == n-1:                               # if this recursive call concerns the last
                    yield sol                              # column then no more work has to be done,
                else:                                      # yield a result. Otherwise, recurs looking
                    yield from gen(c+1,                    # for a location in the remaining columns,
                                   clear_bit(rows, r),     # propagating the fact that row `r` and
                                   clear_bit(raises, raising), # diagonals `raising` and `falling`
                                   clear_bit(falls, falling))  # are under attack and no more selectable.

    return gen(0, set_all(n), set_all(2*n-1), set_all(2*n-1))  # start the enumeration of solutions.
```

and it returns *all* the solution to the given problem when required to do so, as the next example shows.

Example 67. *Solutions to the* 5*-Queens problem are*

```
>>> for s in queens(5):
...     print(pretty(s))

|Q| | | | |  |Q| | | | |  | | |Q| | |  | | | |Q| |  | | | |Q| |
| | | |Q| |  | | |Q| | |  |Q| | | | |  |Q| | | | |  | |Q| | | |
| |Q| | | |  | | | | |Q|  | | | |Q| |  | | |Q| | |  | | | | |Q|
| | | | |Q|  | |Q| | | |  | |Q| | | |  | | | | |Q|  | | |Q| | |
| | |Q| | |  | | | |Q| |  | | | | |Q|  | |Q| | | |  |Q| | | | |
| |Q| | | |  | | | | |Q|  | |Q| | | |  | | | | |Q|  | | |Q| | |
| | | |Q| |  | | |Q| | |  | | | | |Q|  | |Q| | | |  | | | | |Q|
|Q| | | | |  |Q| | | | |  | | |Q| | |  | | | |Q| |  | |Q| | | |
| | |Q| | |  | | | |Q| |  |Q| | | | |  |Q| | | | |  | | | |Q| |
| | | | |Q|  | |Q| | | |  | | | |Q| |  | | |Q| | |  |Q| | | | |
```

In these examples the following pretty printer is used to represent solutions drawing them in bare minimal ASCII,

```
def pretty(sol):
    n = len(sol)
    s = ""
    for r in range(n):
        pos = sol.index(r)
        row = "|".join('Q' if c == pos else ' '
                       for c in range(n))
        s += "|{}|\n".format(row)

    return s
```

Enumerating all solutions for different integers n we get the known sequence `http://oeis.org/A000170`,

```
>>> [len(list(queens(i))) for i in range(1,13)]
[1, 0, 0, 2, 10, 4, 40, 92, 352, 724, 2680, 14200]
```

Example 68. *We tackle the more complex* 24*-Queens problem, whose first solution is about* 3 *seconds away*

```
>>> more_queens = queens(24)
>>> print(pretty(next(more_queens)))
```

```
|Q| | | | | | | | | | | | | | | | | | | | | | | |
| | | |Q| | | | | | | | | | | | | | | | | | | | |
| |Q| | | | | | | | | | | | | | | | | | | | | | |
| | | | |Q| | | | | | | | | | | | | | | | | | | |
| | |Q| | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |Q| | | | | | | |
| | | | | | | | | | | | | | | | | | | | | |Q| | |
| | | | | | | | | | | | | | | | | |Q| | | | | | |
| | | | | |Q| | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | |Q| | | | | | | | | |
| | | | | | |Q| | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | |Q| | | | | |
| | | | | | | | | | | | | | | | | | | | |Q| | | |
| | | | | | | |Q| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | |Q|
| | | | | | | | | | | | | | | | | | | |Q| | | | |
| | | | | | | | | | | | | | | | | | | | | | |Q| |
| | | | | | | | |Q| | | | | | | | | | | | | | | |
| | | | | | | | | | |Q| | | | | | | | | | | | | |
| | | | | | | | | | | | |Q| | | | | | | | | | | |
| | | | | | | | | | | | | | | |Q| | | | | | | | |
| | | | | | | | | |Q| | | | | | | | | | | | | | |
| | | | | | | | | | | |Q| | | | | | | | | | | | |
| | | | | | | | | | | | | |Q| | | | | | | | | | |
```

5.2 *Polyominoes*

In this section we play with some problems concerning *polyominoes*, formalized and introduced by prof. Solomon Golomb in [Golomb, 1996]. Our aim is to provide a *generic* algorithm that consumes a board where it is possible to place some pieces and produces a collection of (possibly *incomplete*) tilings of the board. Therefore we describe (i) how boards are encoded, (ii) how shapes can be defined and (iii) the fundamental concept of *anchor* that allows us to bookkeep the next free cell in the board. Our implementation is coded in Python and has an educational flavor; additionally, for more puzzles and problems solved using the same language see [Goodger, 2015], while [Knuth, 2000] popularize the idea of [Hitotumatu and Noshita, 1979] about a clever use of doubly linked lists to tackle combinatorial enumerations via depth-first searches and backtracking.

5.2.1 *Boards, shapes, anchors for backtracking*

Maybe the hardest part in the understanding concerns the representations of both the board and the state (free or occupied) of each cell; moreover, the same difficulty arises for shapes and their orientations as well. We answer to each question in turn:

- a *board* with `r` rows and `c` columns is represented by an *integer* with `rc` bits; this is because we want to use bit masking techniques and it is efficient to find the *next free* cell (using the utility function `low_bit`), which correspond to the position of the first bit `1` from the right, namely the right-most 1 in its least significant part. Here it is,

0	r	$2r$	$\dots$	$(c-1)r$
1	$r+1$	$2r+1$	$\dots$	$(c-1)r+1$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$r-1$	$2r-1$	$3r-1$	$\dots$	$rc-1$

- a *shape* is a collection of cells, usually sharing an edge pairwise. We choose to represent a shape as a `namedtuple` object: it has an *hashable* component and a collection of *isomorphisms* to represent rotations and mirroring, coded as a lambda expression which consumes the *anchor* position as a pair of row and column indices, and returns a list of isomorphic shapes, namely positions coding symmetry, reflection or rotation of the shape; therefore, *each isomorphism is a sequence of positions too.*

- an *anchor* is the position in which the top-left cell of a shape orientation will be placed in the next *free* cell of the board and every orientation should be relative to the anchor provided. The anchor is *always* given with respect to position `(r,c)`:

```
      *                    (r-2,c+2)
      *   ->               (r-1,c+2)
  * * *       (r,c) (r,c+1) (r, c+2)
```

  so the orientation is coded as the *tuple*

```
((r,c), (r,c+1), (r-2,c+2), (r-1,c+2), (r, c+2))
```

  in the given order, where pairs are listed according to the order *top to bottom* then *left to right*, namely when rows are exhausted repeat from to the top of the next column.

In order to structure our thoughts, we start with the definition of the shape concept as a `namedtuple` object

```python
from collections import namedtuple

shape_spec = namedtuple('shape_spec', ['name', 'isomorphisms',])
```

that allows us to define the backtracking algorithm

```python
def polyominoes(dim, shapes, availables='ones',
                max_depth_reached=None, forbidden=[],
                pruning=lambda coord, positions, shapes: False):

    rows, cols = dim
    sol = []

    if not availables or availables == 'ones':
        availables = {s.name:1 for s in shapes}
    elif availables == 'inf':
        availables = {s.name:-1 for s in shapes}
    def place(S, positions):
        for r, c in positions:
            S = clear_bit(S, r + rows*c)
        return S

    def shapes_available():
        return {s for s in shapes if availables[s.name]}

    def gen(positions, attempts):

        p = low_bit(positions)
        c, r = divmod(p, rows)

        if pruning((r,c), positions, shapes_available()):
            raise StopIteration()

        for i, s in enumerate(shapes):

            if not availables[s.name]: continue

            for j, iso in enumerate(s.isomorphisms(r, c)):

                if all(0 <= rr < rows
                       and 0 <= cc < cols
                       and is_on(positions, rr + rows*cc)
                       for rr, cc in iso):

                    fewer_positions = place(positions, iso)
                    availables[s.name] -= 1
                    sol.append((s, positions, (r,c), iso),)

                    if not (fewer_positions and attempts):
                        yield sol
                    else:
                        yield from gen(fewer_positions, attempts-1)

                    sol.pop()
                    availables[s.name] += 1

    return gen(place(set_all(rows*cols), forbidden),
               max_depth_reached or -1)
```

5.2.2 *Pentominoes*

We start with a relatively simple set of shapes, those composed of 5 unit cells and commonly known as *pentominoes*. According to our encoding, we introduce shapes with their orientations; for example, here is the definition of the `V_shape`:

```
"""
*       * * *       *   * * *
*           *       *   *
* * *       *   * * *   *
"""
V_shape = shape_spec(
    name='V',
    isomorphisms=lambda r, c: [
        ((r,c), (r+1,c),  (r+2,c),   (r+2, c+1), (r+2, c+2)),
        ((r,c), (r, c+1), (r,c+2),   (r+1, c+2), (r+2, c+2)),
        ((r,c), (r,c+1),  (r-2,c+2), (r-1,c+2),  (r, c+2)),
        ((r,c), (r+1,c),  (r+2,c),   (r,c+1),    (r, c+2))
    ])
```

With the current setup we can define the complete set of shapes and, consequently, the generator over the solution space for the tilings of a board 6 × 10 having 1 piece of each shape, respectively.

```
>>> '''
... X:      I:  V:      U:    W:      T:
...   *     *   *       * *   *       * * *
... * * *   *   *       *     * *       *
...   *     *   * * *   * *     * *     *
...         *
...         *
...
... Z:      N:    L:    Y:    F:      P:
... *       *     *     *     *       *
... * * *   * *   *     * *   * * *   * *
...     *     *   *     *       *     * *
...           *   * *   *
... '''
>>> shapes = [X_shape, I_shape, V_shape, U_shape, W_shape, T_shape,
...           Z_shape, N_shape, L_shape, Y_shape, F_shape, P_shape]
>>> dim = (6,10)
>>> tilings = polyominoes(dim, shapes, availables="ones")
```

In Table 5.2 we report an application of our implementation; in particular, it shows our choice to represent tilings, drawing shapes as collections of multiple occurrences of the same greek letters.

```
>>> for i in range(6):
...     print(pretty(next(tilings)))
┌─────────────────────┐
│ β δ δ δ ε ε ι ι ι ι │
│ β δ θ δ α ε ε λ λ ι │
│ β θ θ α α α ε η λ λ │
│ β θ γ μ α η η η λ ζ │
│ β θ γ μ μ η κ ζ ζ ζ │
│ γ γ γ μ μ κ κ κ κ ζ │
└─────────────────────┘
┌─────────────────────┐
│ β δ δ δ η η α ζ ζ ζ │
│ β δ θ δ η α α α ζ κ │
│ β θ θ η η λ α ε ζ κ │
│ β θ γ λ λ λ ε ε κ κ │
│ β θ γ ι λ ε ε μ μ κ │
│ γ γ γ ι ι ι ι μ μ μ │
└─────────────────────┘
┌─────────────────────┐
│ β δ δ δ η η ι ι ι ι │
│ β δ θ δ η ε ε λ λ ι │
│ β θ θ η η α ε ε λ λ │
│ β θ γ μ α α α ε λ ζ │
│ β θ γ μ μ α κ ζ ζ ζ │
│ γ γ γ μ μ κ κ κ κ ζ │
└─────────────────────┘
┌─────────────────────┐
│ β ε ε ζ ζ ζ ι ι ι ι │
│ β κ ε ε ζ λ θ θ θ ι │
│ β κ κ ε ζ λ λ λ θ θ │
│ β κ γ δ δ α λ η η μ │
│ β κ γ δ α α α η μ μ │
│ γ γ γ δ δ α η η μ μ │
└─────────────────────┘
┌─────────────────────┐
│ β ε ε ζ ζ ζ ι ι ι ι │
│ β κ ε ε ζ λ θ θ θ ι │
│ β κ κ ε ζ λ λ λ θ θ │
│ β κ γ μ η η λ α δ δ │
│ β κ γ μ μ η α α α δ │
│ γ γ γ μ μ η η α δ δ │
└─────────────────────┘
┌─────────────────────┐
│ β ε ε μ μ μ ζ δ δ δ │
│ β κ ε ε μ μ ζ δ θ δ │
│ β κ κ ε α ζ ζ ζ θ θ │
│ β κ γ α α α λ η η θ │
│ β κ γ ι α λ λ λ η θ │
│ γ γ γ ι ι ι ι λ η η │
└─────────────────────┘
```

Table 5.2: The first 6 tilings enumerated by generator `polyominoes` using the `shapes` collection of pieces.

With forbidden cells and limited shapes availability

It is possible to taylor the enumeration with respect to (i) the number of available pieces for each shape and to (ii) forbidden placements on the board. Both of them can be easy achieved by tuning the application of `polyominoes` providing the *keyword* arguments `availables` and `forbidden`, respectively. For the former, provide a dictionary of `(k, v)` objects, where `k` denotes the shape's name and `v` denotes its pieces availability; for the latter, provide a list of positions that should be avoided in the placement process. An example follows,

```
>>> dim = (6,10)
>>> tilings = polyominoes(
...     dim, shapes,
...     availables={s.name:3 for s in shapes},
...     forbidden=[(0,0), (1,0), (2,0), (3,0), (4,0),
...                (1,9), (2,9), (3,9), (4,9), (5,9),
...                (1,5), (2,4), (2,5), (3,4), (3,5)])
```

and some tilings are shown in Table 5.3.

5.2.3 *Polyomino's order*

In the exercise 7 of his book, Ruskey asks to find the *order* of some polyomino, defined according to

Definition 69. *The order of a polyomino* P *is the smallest number of* P *copies that will perfectly fit into a rectangle board, where rotations and reflections of* P *are allowed.*

We take into account the Y polyomino and we check that its order is actually 10 in tailing a board 5×10; in order to show this fact, we give 10 copies of the `Y_shape` object, each one with one piece available, respectively; although there are many other solution,

```
>>> Y_shapes = [
...     shape_spec(name="{}_{}".format(Y_shape.name, i),
...                isomorphisms=Y_shape.isomorphisms)
...     for i in range(10)
... ]
>>> dim = (5,10)
>>> Y_tilings = polyominoes(dim, Y_shapes, availables='ones')
>>> print(pretty(next(Y_tilings)))
```

```
┌─────────────────────┐
│                     │
│ α γ γ γ γ ζ ι ι ι ι │
│ α α δ γ ζ ζ ζ ζ ι κ │
│ α δ δ δ δ η η η η κ │
│ α β ε ε ε ε θ η κ κ │
│ β β β β ε θ θ θ θ κ │
│                     │
└─────────────────────┘
```

the one shown above solves the problem.

5.2.4 *Fibonacci's tilings*

In this section we take into account a smaller set of shapes, composed of *squares* and *dominos* pieces, in order to tile boards with an increasing number of rows; eventually, enumerations of tilings for greater boards are counted by known sequences in the OEIS.

```
"""
*
"""
square_shape = shape_spec(
    name='square',
    isomorphisms=lambda r, c: [((r, c),)])

"""
* *    *
       *
"""
domino_shape = shape_spec(
    name='domino',
    isomorphisms=lambda r, c: [ ((r, c), (r, c+1)),
                                ((r, c), (r+1, c))])

fibonacci_shapes = [square_shape, domino_shape]
```

```
>>> for i in range(6):
...     print(pretty(next(tilings)))
```

```
┌─────────────────────┐
│                     │
│   γ γ γ δ δ δ ζ ζ ζ │
│   ι ι γ δ   δ λ ζ   │
│   ι μ γ     λ λ ζ   │
│   ι μ μ     η λ λ   │
│   ι μ μ θ θ η η η   │
│ β β β β β θ θ θ η   │
│                     │
└─────────────────────┘
┌─────────────────────┐
│                     │
│   γ γ γ ι ι ι ι λ λ │
│   μ μ γ ι   ε λ λ   │
│   μ μ γ     ε ε λ   │
│   δ μ δ     η ε ε   │
│   δ δ δ θ θ η η η   │
│ β β β β β θ θ θ η   │
│                     │
└─────────────────────┘
┌─────────────────────┐
│                     │
│   γ γ γ ι ι ι ι λ λ │
│   γ μ μ ι   ε λ λ   │
│   γ μ μ     ε ε λ   │
│   δ μ δ     η ε ε   │
│   δ δ δ θ θ η η η   │
│ β β β β β θ θ θ η   │
│                     │
└─────────────────────┘
┌─────────────────────┐
│                     │
│   γ γ γ θ θ δ δ λ λ │
│   γ θ θ θ   δ λ λ   │
│   γ ε ε     δ δ λ   │
│   ε ε κ     ζ μ μ   │
│   ε κ κ κ κ ζ μ μ   │
│ β β β β β ζ ζ ζ μ   │
│                     │
└─────────────────────┘
┌─────────────────────┐
│                     │
│   γ γ γ θ θ δ δ λ λ │
│   γ θ θ θ   δ λ λ   │
│   γ ε ε     δ δ λ   │
│   ε ε κ     η η μ   │
│   ε κ κ κ κ η μ μ   │
│ β β β β β η η μ μ   │
│                     │
└─────────────────────┘
┌─────────────────────┐
│                     │
│   γ γ γ θ θ δ δ λ λ │
│   γ θ θ θ   δ λ λ   │
│   γ ε ε     δ δ λ   │
│   ε ε κ     μ μ μ   │
│   ε κ κ κ κ μ μ ι   │
│ β β β β β ι ι ι ι   │
│                     │
└─────────────────────┘
```

Table 5.3: The first 6 tilings enumerated by generator `polyominoes` using the `shapes` collection of pieces under the restriction to have 3 pieces for each shape and forbidden cells should be left blank.

Tiling greater boards, ascending ordered according to the number of rows, we enumerate the sequences

```
>>> [ [len(list(polyominoes(dim=(j,i),
...                          shapes=fibonacci_shapes,
...                          availables='inf')))
...     for i in range(n)]
...   for j, n in [(1, 13),  # https://oeis.org/A000045
...                (2, 13),  # https://oeis.org/A030186
...                (3, 7),   # https://oeis.org/A033506
...                (4, 7),   # https://oeis.org/A033507
...                (5, 6)]]  # https://oeis.org/A033508
[[0, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233],
 [0, 2, 7, 22, 71, 228, 733, 2356, 7573, 24342, 78243, 251498, 808395],
 [0, 3, 22, 131, 823, 5096, 31687],
 [0, 5, 71, 823, 10012, 120465, 1453535],
 [0, 8, 228, 5096, 120465, 2810694]]
```

which counts Fibonacci's numbers, the number of matchings in graphs $P_2 \times P_n$, $P_3 \times P_n$, $P_4 \times P_n$ and $P_5 \times P_n$, respectively. For the sake of clarity, the $f_6 = 13$ ways to tile a simple board 1×6 are:

```
┌─────────────┐  ┌─────────────┐  ┌─────────────┐  ┌─────────────┐
│ α α α α α α │  │ α α α α β β │  │ α α α β β α │  │ α α β β α α │
└─────────────┘  └─────────────┘  └─────────────┘  └─────────────┘
┌─────────────┐  ┌─────────────┐  ┌─────────────┐  ┌─────────────┐
│ α α β β β β │  │ α β β α α α │  │ β β α α α α │  │ β β α α β β │
└─────────────┘  └─────────────┘  └─────────────┘  └─────────────┘
┌─────────────┐  ┌─────────────┐  ┌─────────────┐  ┌─────────────┐
│ α β β α β β │  │ α β β β β α │  │ β β α β β α │  │ β β β β α α │
└─────────────┘  └─────────────┘  └─────────────┘  └─────────────┘
┌─────────────┐
│ β β β β β β │
└─────────────┘
```

5.3 *Parallelogram Polyominoes*

In this section we study a collection of polyominoes where their shapes are subject to a constraint; precisely, a *parallelogram polyomino* is defined by two paths that only intersect at their origin and extremity, composed of East and South steps only. They are counted by Catalan numbers according to their *semiperimeter*, which equals the sum of their heights and widths; for the sake of clarity, both [Delest and Fedou, 1993, Delest et al., 1987] are extensive studies.

Example 70. *The set of* 42 *parallelogram polyominoes shown in Table 5.4 can be used to tile a board* 16×16*; precisely, using the enumerator* `polyominoes` *again, we show the first incomplete and the first complete tilings,*

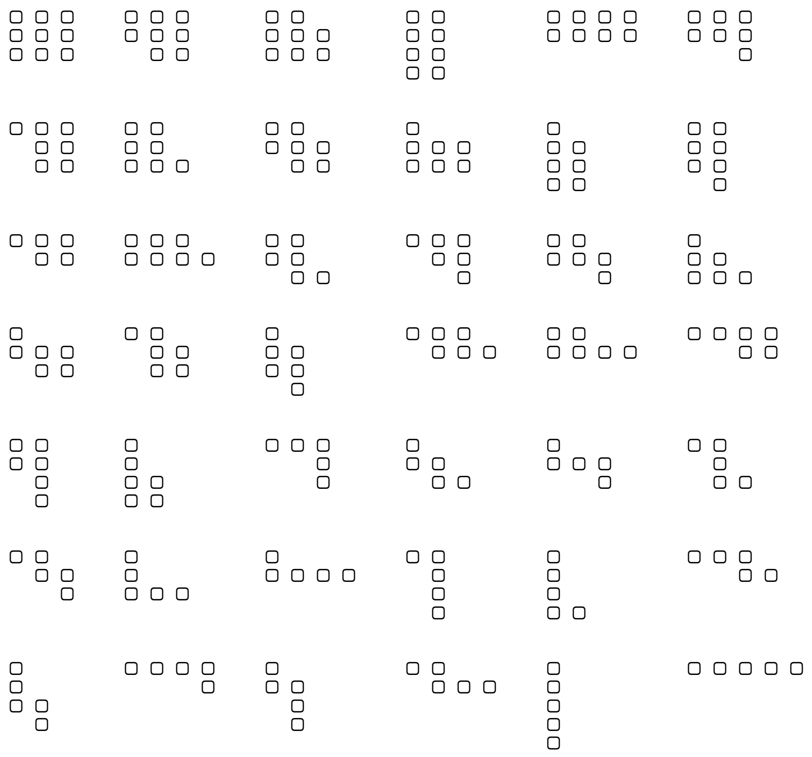
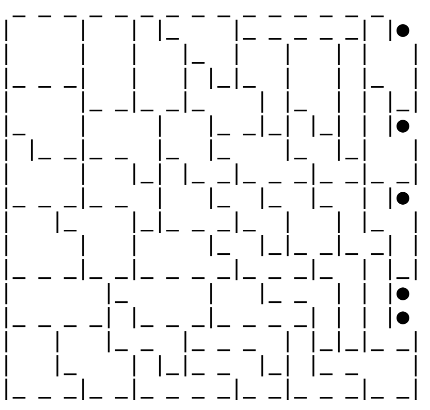
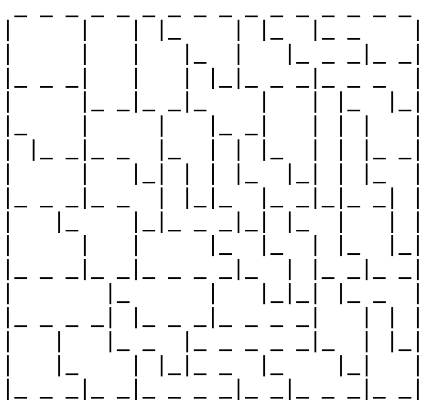


Table 5.4: Parallelogram Polyominoes with semiperimeter 6, which are 42 in total, the 6th Catalan number.

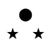
respectively. The solution space of this problem is very sparse and the enumeration is pretty hard; despite the vanilla approach used, our implementation allows us to provide an heuristic function (to prune any attempt to insert a polyomino on last row or last column, for example) but we get no gain in efficiency. On the contrary, we note that the order in which polyominoes are choosen for placement leverage the execution time, even for smaller boards; however, for greater boards the problem remains open.

5.4 *An implementation of the ECO method*

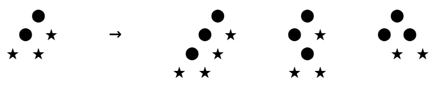
The *ECO methodology* is introduced in [Barcucci et al., 1998b] and refined in [Barcucci et al., 1999] in order to enumerate classes of combinatorial objects; relying on the idea to perform *local expansion* on objects' *active sites* by means of an *operator*, the ECO method gives a recursive construction of the class that objects belong to.

In the spirit of [Bernini et al., 2007, Bacchelli et al., 2004], we provide a implementation of the ECO method which allows us to *build* classes of combinatorial objects; consequently, enumeration comes for free. Moreover, defining a *recursive shape* as the combination of symbols denoting the objects' structure with symbols denoting the objects'*active sites*, namely positions where it is possible to perform a local expansion, we have a data structure to be manipulated by a Python function, which reifies the *operator* concept.

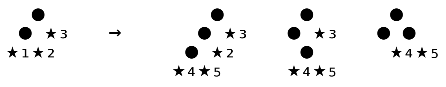
For the sake of clarity, let $\star$ be an *active site* that accepts to be replaced by



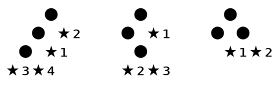
hence, operator $\rightarrow$ performs replacements for each site; for example,



In order to understand how $\rightarrow$ works, we label each $\star$ with a integer subscript that denotes the discrete time in which it will be replaced,



and we normalize discrete times of produced objects in order to restart the application of operator $\rightarrow$ to each one of them,



From this characterization it is possible to recover the correlated concepts of *generating tree* and *succession rule* [Chung et al., 1978]. The former is straightforward encoded within the application of the operator $\rightarrow$, the latter is $(3) \hookrightarrow (4)(3)(2)$ because (i) on the left hand side of $\rightarrow$ there are 1 object with 3 active sites and (ii) on the right hand side of $\rightarrow$ there are 3 objects with $4, 3$ and 2 active sites, respectively.

In the following examples we apply this methodology to build and enumerate known classes of combinatorial objects.

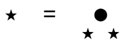
Example 71 (Binary trees). *Their class is encoded by*

```
binary_tree_shapes = {
    'bintree': lambda r, c: Anchor(symbol='●', stars=[
        Star(row=r+1, col=c-1, offsets=None,
             link='bintree', symbol='★'),
        Star(row=r+1, col=c+1, offsets=None,
             link='bintree', symbol='★'),
    ]),
}
```

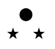
and enumerated in Table 5.5.

The previous example shows our way to encode the recursive shape of binary trees; in particular, shapes are vanilla Python dictionaries, where each one of them contains key-value pairs `(k, v)` where `k` denotes the shape label and `v` denotes a function that consumes a coordinate `(r,c)` and produces an `Anchor` object that carries information about the structure symbol and the collection of shape's *active sites*.

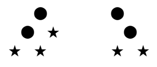
According to the symbolic equation



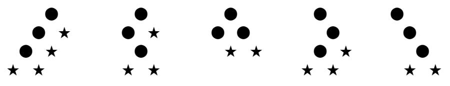
symbol ★ enumerates the (i) generation



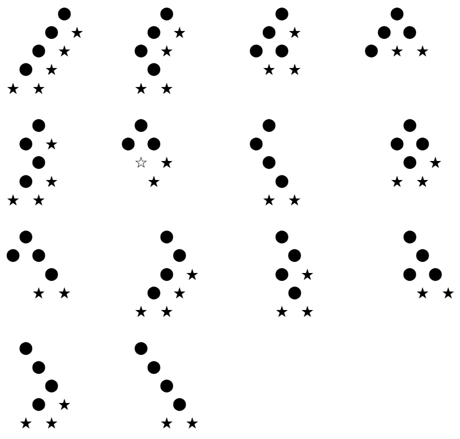
the (ii) generation



the (iii) generation



the (iv) generation



where the symbol ☆ means the sovrapposition of symbols ● and ★ in back and fore ground, respectively.

Table 5.5: Enumerations up to the 5th generation of binary trees.

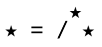
Example 72 (Dyck paths). *Their class is encoded by*

```
dyck_path_shapes = {
    'dick': lambda r, c: Anchor(symbol='/', stars=[
        Star(row=r-1, col=c+1, offsets=(0, 2),
             link='dick', symbol='★'),
        Star(row=r,   col=c+2, offsets=None,
             link='dick', symbol='★'),
    ]),
}
```

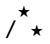
and enumerated in Table 5.6.

Previous example spots a feature provided by `Star` objects, namely the capability to shift part of the structure in order to make room for local expansions; this is achieved by the keyword argument `offset`, which has to be a pair `(or, oc)` where components are integers that denote row and column offsets, respectively. For the Dyck paths shape, we provide `offset=(0,2)` for the topmost active site because when the structure is expanded there, the rightmost path already generated should be shifted by 2 columns while remaining at the same distance from the x axis.

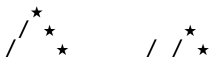
According to the symbolic equation



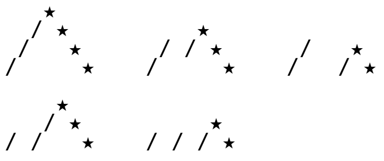
symbol ★ enumerates the (i) generation



the (ii) generation



the (iii) generation



the (iv) generation

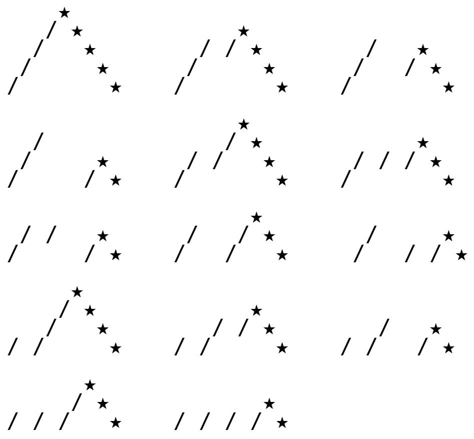


Table 5.6: Enumerations up to the 5th generation of Dyck paths.

Example 73 (Balanced Parens). *Their class is encoded by*

```
balanced_parens_shapes = {
    'parens': lambda r, c: Anchor(symbol='(', stars=[
        Star(row=r, col=c+1, offsets=(0, 3),
             link='parens', symbol='★'),
        Star(row=r, col=c+3, offsets=None,
             link='parens', symbol='★'),
    ]),
}
```

and enumerated in Table 5.7.

Both the previous examples and the next one enumerate classes of objects counted by *Catalan numbers* and each class obeys to the succession rule $(1) \hookrightarrow (2)$ and $(k) \hookrightarrow (2)\cdots(k+1)$, where $k > 1$.

Example 74 (Steep parallelograms polyominoes)**.** *They are a refinement of parallelogram polyominoes because the lower border of those polyominoes has no pair of consecutive horizontal steps. Their class is encoded by*

```
steep_shapes = {
    'one_star': lambda r, c: Anchor(symbol='▢', stars=[
        Star(row=r-1, col=c, offsets=None,
             link='two_stars', symbol='☆'),
    ]),
    'two_stars': lambda r, c: Anchor(symbol='▢', stars=[
        Star(row=r-1, col=c,   offsets=None,
             link='two_stars', symbol='☆'),
        Star(row=r,   col=c+1, offsets=None,
             link='one_star',  symbol='★'),
    ]),
}
```

and enumerated in Table 5.8.

In [Barcucci et al., 1998a] it is proved that steep parallelogram polyominoes are counted by the *Motzkin numbers*, in particular the set of those polyominoes having semiperimeter $n+1$ has $\mathcal{M}_n$ objects – the n-th Motzkin number. Moreover, the enumeration verifies that steep parallelogram polyominoes obey the succession rule $(1) \hookrightarrow (2)$ and $(k) \hookrightarrow (1)\cdots(k-1)(k+1)$, where $k > 1$.

Example 75 (Rabbits)**.** *These shapes encode that (i) a couple of young rabbits* ○ *gets old and becomes* ● *which, in turn, (ii) gives birth to a couple of young rabbits and gets older.*

```
rabbits_shapes = {
    'young': lambda r, c: Anchor(symbol='○', stars=[
        Star(row=r-1, col=c+1, offsets=None,
             link='senior', symbol='★'),
    ]),
    'senior': lambda r, c: Anchor(symbol='●', stars=[
        Star(row=r,   col=c+1, offsets=None,
             link='young',  symbol='▲'),
        Star(row=r+1, col=c+1, offsets=None,
             link='senior', symbol='★'),
    ]),
}
```

The complete enumeration contains doubles, so we report the set of those structures that is actually counted by the *Fibonacci numbers*; according to the mutually symbolic equations,

According to the symbolic equation

```
★ = (★)★
```

symbol ★ enumerates the (i) generation

```
(★ ★
```

the (ii) generation

```
((★ ★ ★
(  (★ ★
```

the (iii) generation

```
(((★ ★ ★ ★
((  (★ ★ ★
((    (★ ★
(  ((★ ★ ★
(  (  (★ ★
```

the (iv) generation

```
((((★ ★ ★ ★ ★
(((  (★ ★ ★ ★
(((    (★ ★ ★
(((      (★ ★
((  ((★ ★ ★ ★
((  (  (★ ★ ★
((  (    (★ ★
((    ((★ ★ ★
((    (  (★ ★
(  (((★ ★ ★ ★
(  ((  (★ ★ ★
(  ((    (★ ★
(  (  ((★ ★ ★
(  (  (  (★ ★
```

Table 5.7: Enumerations up to the 5th generation of balanced parens.

According to the mutually symbolic equations

```
    ☆          ☆
★ = □      ☆ = □ ★
```

symbol ★ enumerates the (i) generation

```
☆
□
```

the (ii) generation

```
☆
□ ★
□
```

the (iii) generation

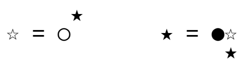
```
☆
□ ★      ☆
□ ★    □ □
□      □
```

the (iv) generation

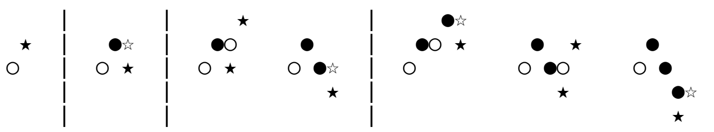
```
☆
□ ★      ☆             ☆
□ ★    □ □    □ ☆      □ ★
□ ★    □ ★    □ □    □ □
□      □      □      □

☆
□ ★      ☆
□ ★    □ □    □ ☆    □
□ ★    □ ★    □ □    □ ☆
□ ★    □ ★    □ ★    □ □
□      □      □      □

  ☆                     ☆
  □ ★           ☆       □ ★
□ □     □ ◐   □ □ ★     □ ★
□ ★     □ □   □ □     □ □
□       □     □       □

    ☆
  □ □
□ □
□
```

where the symbol ◐ means the sovrapposition of symbols ▢ and ☆ in back and fore ground, respectively.

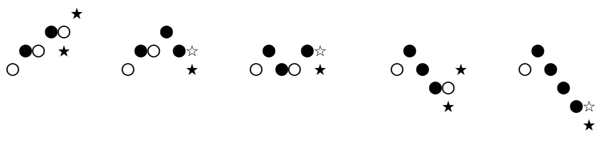
Table 5.8: Enumerations up to the 5th generation of steep parallelograms.



the enumeration process starts with ☆ and enumerates the (i), (ii), (iii) and (iv) generations



and, finally, the (v) generation



as required.

Conclusions

This chapter has presented an extensive exercise in coding, practicing with backtracking programming techniques; in particular, vanilla implementations pair with bitmasking techniques to speed up computation, keeping elegance and clarity at the same time.

We strive to write generic algorithms that work in different contexts ranging from the placement problems, such as the n-Queens problem, to tiling problems using different shapes; moreover, the same minimal and open approach allows us to tackle enumeration tasks starting from simple but powerful specifications of classes of combinatorial objects and a well known enumeration method called ECO has been fully implemented.

6 Semi-Certified Interactive Logic Programming

This chapter studies an embedded Domain Specific Language for logic programming. First, we give a quick introduction of μ*Kanren*, a purely functional implementation of this language and, second, we extend the HOL Light theorem prover in order to introduce the relational paradigm in its tactics mechanism.

6.1 μKanren and relational programming

The central tenet of relational programming is that *programs corresponds to relations that generalize mathematical functions*; our interest here is to deepen our understanding of the underlying concepts and data structures of languages in the *miniKanren* family. The main reference that drives our work is [Friedman et al., 2018] and advanced topics are discussed in Byrd's dissertation [Byrd, 2009].

The heavy use of higher order functions, infinite streams of objects and unification à-la Robinson makes possible to implement μKanren [Hemann and Friedman, 2013], a purely functional core of miniKanren; we repeat the exercise of coding it using different programming languages, in particular

Python we provide both a complete implementation of the abstract definition and a test suite that stresses our functions against *all* questions in the reference book. Moreover, we characterize our code with a *fair* enumeration strategy based on the *dovetail* techniques used in the enumeration of the Rationals numbers; precisely, the monadic function `mplus(streams, interleaving)` enumerates the states space `streams`, using different strategies according to the argument `interleaving`.

In order to understand states enumeration can be helpful to use a ”matrix notation”, that associates a row to each stream α of states in `streams`, which is an *iterable* object over a *countably*, possibly infinite, set of *states streams*, so the matrix could have infinite rows. In parallel, since each states stream α lying on a row is an *iterable* object over a *countably*, possibly infinite, set of *satisfying states*, the matrix could have infinite columns too; therefore, the matrix we are building could be infinite in both dimensions. So, let `streams` be represented as

$$\begin{pmatrix} s_{00} & s_{01} & s_{02} & s_{03} & \dots \\ s_{10} & s_{11} & s_{12} & \dots & \\ s_{20} & s_{21} & \dots & & \\ s_{30} & \dots & & & \\ \dots & & & & \end{pmatrix}$$

where each $s_{i,j}$ is a state that carries a substitution which satisfies the relation under study. Such states are visited according to the *dovetail* techniques which enumerates by interleaving `state` objects lying on the same *rising diagonal*, resulting in a *fair, complete scheduler* in the sense that *every* satisfying `state` object will be reached, eventually. For the sake of clarity, enumeration proceeds as follows

$$s_{00}, s_{10}, s_{01}, s_{20}, s_{11}, s_{02}, s_{30}, s_{21}, s_{12}, s_{03}, \dots$$

with respect to its implementation

```python
def mplus(streams, interleaving):

    if interleaving:

        try: α = next(streams)
        except StopIteration: return
        else: S = [α]

        while S:

            for j in reversed(range(len(S))):
                β = S[j]
                try: s = next(β)
                except StopIteration: del S[j]
                else: yield s

            try: α = next(streams)
            except StopIteration: pass
            else: S.append(α)

    else:

        for α in streams: yield from α
```

Scheme we provide our implementation using the same language that original authors use for their canonical version. We diverge from them in the way we represent substitutions, choosing an *union-find* data structure that allows us to maintain a balanced tree to track associations. The overhead work that was necessary to implement a fully-flagged μKanren yield a Scheme library to define and manipulate infinite streams of objects, and this allows us to have another way to define Riordan arrays for free, such as

```
(test '((1)
        (1 1)
        (1 2 1)
        (1 3 3 1)
        (1 4 6 4 1)
        (1 5 10 10 5 1)
        (1 6 15 20 15 6 1)
        (1 7 21 35 35 21 7 1)
        (1 8 28 56 70 56 28 8 1)
        (1 9 36 84 126 126 84 36 9 1))
 ((list∘take 10) (riordan-array stream:1s stream:1s)))
```

lying on the procedural abstraction `riordan-array` that consumes two formal power series and produces a stream of lists, each one denoting a triangle's row; its definition is clear and elegant in our opinion,

```
(define riordan-array
 (Λ (d h)
  (stream:dest/car+cdr (d ∅)
   ((dcar dcdr) (stream:cons
                 (list dcar)
                 ((stream:zip-with cons)
                  dcdr (riordan-array (series:× d h) h)))))))
```

where `stream:×` denotes the *series convolution operator* and the syntactic abstraction Λ is defined as an ”augmented lambda” form that allows us to define delayed by means of `delay-force`,

```
(define-syntax Λ
 (syntax-rules ()
  ((Λ args body ...)
   (lambda args (delay-force (begin body ...))))))
```

Smalltalk this implementation is an exercise in object-oriented programming and its coding has been driven by a test-first approach [Beck, 2002] and it is a literal port of the canonical one.

OCaml finally, this version is preparatory for the extension of the HOL Light theorem prover which we are going to describe in the rest of this chapter.

All these prototypes can be found in [Nocentini, 2018] and some examples follows using the Scheme implementation to show the power of the present paradigm.

Example 76. *The context free grammar $\mathcal{D}$ that defines the set of Dyck paths*

$$\mathcal{D} = \varepsilon \mid \circ\ \mathcal{D} \bullet \mathcal{D}$$

is encoded in the relation `dyckº` *defined by the* μ*Kanren goal*

```
(define dyckº
 (lambda (α)
  (condº/§
   ((nullº α))
   ((fresh (β γ) (∧
                  (dyckº β)
                  (dyckº γ)
                  (appendº `(○ . ,β) `(● . ,γ) α)))))))
```

and an enumeration is reported in Table 6.1.

```
(()
 (○ ●)
 (○ ○ ● ●)
 (○ ● ○ ●)
 (○ ○ ○ ● ● ●)
 (○ ● ○ ○ ● ●)
 (○ ○ ● ● ○ ●)
 (○ ● ○ ● ○ ●)
 (○ ○ ● ○ ● ●)
 (○ ● ○ ○ ○ ● ● ●)
 (○ ○ ● ● ○ ○ ● ●)
 (○ ● ○ ● ○ ○ ● ●)
 (○ ○ ○ ● ● ● ○ ●)
 (○ ● ○ ○ ● ● ○ ●)
 (○ ○ ● ● ○ ● ○ ●)
 (○ ● ○ ● ○ ● ○ ●)
 (○ ○ ○ ○ ● ● ● ●)
 (○ ● ○ ○ ● ○ ● ●)
 (○ ○ ● ● ○ ○ ○ ● ● ●)
 (○ ● ○ ● ○ ○ ○ ● ● ●)
 (○ ○ ○ ● ● ● ○ ○ ● ●)
 (○ ● ○ ○ ● ● ○ ○ ● ●)
 (○ ○ ● ● ○ ● ○ ○ ● ●)
 (○ ● ○ ● ○ ● ○ ○ ● ●)
 (○ ○ ● ○ ● ● ○ ●)
 (○ ● ○ ○ ○ ● ● ● ○ ●)
 (○ ○ ● ● ○ ○ ● ● ○ ●)
 (○ ● ○ ● ○ ○ ● ● ○ ●)
 (○ ○ ○ ● ● ● ○ ● ○ ●)
 (○ ● ○ ○ ● ● ○ ● ○ ●)
 (○ ○ ● ● ○ ● ○ ● ○ ●)
 (○ ● ○ ● ○ ● ○ ● ○ ●)
 (○ ○ ● ○ ○ ● ● ●)
 (○ ● ○ ○ ○ ○ ● ● ● ●)
 (○ ○ ● ● ○ ○ ● ○ ● ●)
 (○ ● ○ ● ○ ○ ● ○ ● ●)
 (○ ○ ○ ● ● ● ○ ○ ○ ● ● ●)
 (○ ● ○ ○ ● ● ○ ○ ○ ● ● ●)
 (○ ○ ● ● ○ ● ○ ○ ○ ● ● ●)
 (○ ● ○ ● ○ ● ○ ○ ○ ● ● ●)
 (○ ○ ● ○ ● ● ○ ○ ● ●)
 (○ ● ○ ○ ○ ● ● ● ○ ○ ● ●))
```

Table 6.1: First 42 Dyck paths enumerated by relation dyckº.

Example 77. *The recurrence relation for the Fibonacci numbers*

$$f_{n+2} = f_{n+1} + f_n, \quad n \geq 0$$

is encoded in the relation `fibonacciº` *defined by the goal*

```
(define fibonacciº
 (lambda (depth n α)
  (cond
   ((zero? depth) (≡ α (list n)))
   (else (fresh (β γ)
          (∧
           (fibonacciº (sub1 depth) (sub1 n) β)
           (fibonacciº (sub1 depth) (sub2 n) γ)
           (appendº β γ α)))))))
```

which enumerates the following identities

$$\begin{aligned} f_{n+2} &= f_n + f_{n+1} \\ f_{n+2} &= f_{n-2} + 2\,f_{n-1} + f_n \\ f_{n+2} &= f_{n-4} + 3\,f_{n-3} + 3\,f_{n-2} + f_{n-1} \\ f_{n+2} &= f_{n-6} + 4\,f_{n-5} + 6\,f_{n-4} + 4\,f_{n-3} + f_{n-2} \\ f_{n+2} &= f_{n-8} + 5\,f_{n-7} + 10\,f_{n-6} + 10\,f_{n-5} + 5\,f_{n-4} + f_{n-3} \\ f_{n+2} &= f_{n-10} + 6\,f_{n-9} + 15\,f_{n-8} + 20\,f_{n-7} + 15\,f_{n-6} + 6\,f_{n-5} + f_{n-4} \end{aligned}$$

compacted in $f_n = \sum_{i=0}^{j} \binom{j}{i} f_{n-2j+i}$ *where* $j \leq \frac{n}{2}$.

Example 78. *The recurrence relation for the Pascal triangle*

$$\begin{aligned} d_{0,0} &= 1, \\ d_{n+1,0} &= d_{n,0}, \quad n \geq 0 \\ d_{n+1,k+1} &= d_{n,k} + d_{n,k+1}, \quad n, k \geq 0 \end{aligned}$$

is encoded in the relation `tartagliaº` *defined by the goal*

```
(define tartagliaº
 (lambda (depth n k α)
  (cond
   ((zero? depth) (≡ α (list (list n k))))
   (else (fresh (β γ)
          (∧
           (tartagliaº (sub1 depth) (sub1 n) (sub1 k) β)
```

```
        (tartagliaº (sub1 depth) (sub1 n) k γ)
        (appendº β γ α)))))))
```

which enumerates the following identities

$$
\begin{aligned}
d_{n+1,k+1} &= d_{n,k+1} + d_{n,k} \\
d_{n+1,k+1} &= d_{n-1,k+1} + 2\,d_{n-1,k} + d_{n-1,k-1} \\
d_{n+1,k+1} &= d_{n-2,k+1} + 3\,d_{n-2,k} + 3\,d_{n-2,k-1} + d_{n-2,k-2} \\
d_{n+1,k+1} &= d_{n-3,k+1} + 4\,d_{n-3,k} + 6\,d_{n-3,k-1} + 4\,d_{n-3,k-2} + d_{n-2,k-3} \\
d_{n+1,k+1} &= d_{n-4,k+1} + 5\,d_{n-4,k} + 10\,d_{n-4,k-1} + 10\,d_{n-4,k-2} + 5\,d_{n-4,k-3} + d_{n-4,k-4} \\
d_{n+1,k+1} &= d_{n-5,k+1} + 6\,d_{n-5,k} + 15\,d_{n-5,k-1} + 20\,d_{n-5,k-2} + 15\,d_{n-5,k-3} + 6\,d_{n-5,k-4} + d_{n-5,k-5}
\end{aligned}
$$

compacted in $\binom{p+m}{r+m} = \sum_{j=0}^{m-1} \binom{m-1}{j}\binom{p-1+m}{r-j+m}$ *for* $p, r, m \in \mathbb{N}$ – *recall that* $d_{n,k} = \binom{n}{k}$.

6.2 *Toward certified computation*

Theorem provers are employed to construct logically verified truths. In this work, we propose an extended language of tactics which support the derivation of formally verified theorems in the spirit of the logic programming paradigm.

Our setup, is based on the HOL Light theorem prover, in which we extend the currently available tactics mechanism with three basic features: (i) the explicit use of metavariables, (ii) the ability to backtrack during the proof search, (iii) a layer of tools and facilities to interface with the underlying proof mechanism.

The basic building block of our framework are ML procedures that we call *solvers*, which are a generalization of HOL tactics and are –as well as tactics– meant to be used compositionally to define arbitrarily complex proof search strategies.

We say that our approach is *semi-certified* because

- on one hand, the produced solutions are formally proved theorems, hence their validity is guaranteed by construction;
- on the other hand, the completeness of the search procedure cannot be enforced in our framework and consequently has to be ensured by a meta-reasoning.

At the present stage, our implementation [Maggesi and Nocentini, 2018] is intended to be a test bed for experiments and further investigation on this reasoning paradigm.

6.3 A simple example

To give the flavor of our framework, we show how to perform simple computations on lists.

Consider first the problem of computing the concatenation of two lists `[1; 2]` and `[3]`. One natural way to approach this problem is by using rewriting. In HOL Light, this can be done by using *conversions* with the command

```
# REWRITE_CONV [APPEND] `APPEND [1;2] [3]`;;
```

where the theorem

```
# APPEND;;
val it : thm =
  |- (!l. APPEND [] l = l) /\
     (!h t l. APPEND (h :: t) l = h :: APPEND t l)
```

gives the recursive equations for the operator `APPEND`.

Our implementation allows us to address the same problem from a logical point of view. We start by proving two theorems

```
# APPEND_NIL;;
val it : thm = |- !l. APPEND [] l = l

# APPEND_CONS;;
val it : thm =
  |- !x xs ys zs. APPEND xs ys = zs
                  ==> APPEND (x :: xs) ys = x :: zs
```

that gives the logical rules that characterize the `APPEND` operator. Then we define a *solver*

```
let APPEND_SLV : solver =
  REPEAT_SLV (CONCAT_SLV (ACCEPT_SLV APPEND_NIL)
                         (RULE_SLV APPEND_CONS));;
```

which implements the most obvious strategy for proving a relation of the form `` `APPEND x y = z` `` by structural analysis on the list `` `x` ``. The precise meaning of the above code will be clear later in this note; however, this can be seen as the direct translation of the Prolog program

```
append([],X,X).
append([X|Xs],Ys,[X|Zs]) :- append(Xs,Ys,Zs).
```

Then, the problem of concatenating the two lists is described by the term

```
`??x. APPEND [1;2] [3] = x`
```

where the binder `` `(??)` `` is a syntactic variant of the usual existential quantifier `` `(?)` ``, which introduces the *metavariables* of the *query.*

The following command

```
list_of_stream
  (solve APPEND_SLV
         `??x. APPEND [1; 2] [3] = x`);;
```

runs the search process where the `solve` function starts the proof search and produces a stream (i.e., a lazy list) of *solutions* and the outermost `list_of_stream` transform the stream into a list.

The output of the previous command is a single solution which is represented by a pair where the first element is the instantiation for the meta-variable `` `x` ``and the second element is a HOL theorem

```
val it : (term list * thm) list =
  [([`x = [1; 2; 3]`], |- APPEND [1; 2] [3] = [1; 2; 3])]
```

Now comes the interesting part: as in logic programs, our search strategy (i.e., the `APPEND_SLV` solver) can be used for backward reasoning.

Consider the variation of the above problem where we want to enumerate all possible splits of the list `[1; 2; 3]`. This can be done by simply changing the goal term in the previous query:

```
# list_of_stream
    (solve APPEND_SLV
           `??x y. APPEND x y = [1;2;3]`);;

val it : (term list * thm) list =
  [([`x = []`; `y = [1; 2; 3]`],
    |- APPEND [] [1; 2; 3] = [1; 2; 3]);
   ([`x = [1]`; `y = [2; 3]`],
    |- APPEND [1] [2; 3] = [1; 2; 3]);
   ([`x = [1; 2]`; `y = [3]`],
    |- APPEND [1; 2] [3] = [1; 2; 3]);
   ([`x = [1; 2; 3]`; `y = []`],
    |- APPEND [1; 2; 3] [] = [1; 2; 3])]
```

6.4 A library of solvers

Our framework is based on ML procedures called *solvers*. Solvers generalizes classical HOL tactics in two ways, (i) they facilitate the manipulation of meta-variables in the goal and (ii) they allows us to backtrack during the proof search. We observe that the tactics mechanism currently implemented in HOL Light already provides basic support for meta-variables in goals; however, it seems to be used only internally in the implementation of the intuitionistic tautology prover `ITAUT_TAC`.

We provide a library of basic solvers with the convention that their names end in `_SLV` as in `REFL_SLV`, for instance.

Every HOL tactic can be 'promoted' into a solver using the ML function

```
TACTIC_SLV : tactic -> solver
```

A partial list of solvers approximately corresponding to classical HOL tactics are `ACCEPT_SLV`, `NO_SLV`, `REFL_SLV`, `RULE_SLV` (corresponding to `MATCH_MP_TAC`).

Notice that these solvers are different from their corresponding tactics because either

1. use the stream mechanism instead of OCaml exceptions to handle the control flow; or
2. perform some kind of unification.

For (1), a very basic example is the solver `NO_SLV` which, instead of raising an exception, it returns the empty stream of solutions.

One example of (2) is the `REFL_SLV` solver: when it is applied to the goal

```
?- x + 1 = 1 + x
```

where `x` is a meta-variable, closes the goal by augmenting the instantiation with the substitution `1/x` and producing the theorem `|- 1 + 1 = 1 + 1`. Observe that the corresponding `REFL_TAC` fails in this case.

As for tactics, we have a collection of higher-order solvers. Some of them, are the analogous of the corresponding tacticals: `ASSUM_LIST_SLV`, `CHANGED_SLV`, `EVERY_SLV`, `MAP_EVERY_SLV`, `POP_ASSUM_LIST_SLV`, `POP_ASSUM_SLV`, `REPEAT_SLV`, `THENL_SLV`, `THEN_SLV`, `TRY_SLV`, `UNDISCH_THEN_SLV`.

Given two solvers s_1 and s_2 the solver combinator `CONCAT_SLV` make a new solver that collect sequentially all solutions of s_1 followed by all solutions of s_2. This is the most basic construction for introducing backtracking into the proof strategy.

From `CONCAT_SLV`, a number of derived combinators are defined to capture the most common enumeration patterns, here we give a brief list of those combinators without an explicit description. However, we hope that the reader can guess the actual behaviour from both their name and their ML type:

```
COLLECT_SLV : solver list -> solver
MAP_COLLECT_SLV : ('a->solver) -> 'a list -> solver
COLLECT_ASSUM_SLV : thm_solver -> solver
COLLECT_X_ASSUM_SLV : thm_solver -> solver
```

Solvers can be used interactively. Typically, we can start a new goal with the command `gg` and execute solvers with `ee`. The command `bb` restore the previous proof state and `pp` prints the current goal state. The stream of results is produced by a call to `top_thms()`.

Here is an example of interaction. We first introduce the goal, notice the use of the binder `(??)` for the meta-variable `x`:

```
# gg `??x. 2 + 2 = x`;;
val it : mgoalstack =
`2 + 2 = x`
```

one possible solution is by using reflexivity, closing the proof

```
# ee REFL_SLV;;
val it : mgoalstack =
```

we can now form the resulting theorem

```
# list_of_stream(top_thms());;
val it : thm list = [|- 2 + 2 = 2 + 2]
```

Now, if one want to find a different solution, we can restore the initial state

```
# bb();;
val it : mgoalstack =
`2 + 2 = x`
```

then we use a different solver that allows us to unify with the equation `|- 2 + 2 = 4`

```
# ee (ACCEPT_SLV(ARITH_RULE `2 + 2 = 4`));;
val it : mgoalstack =
```

and again, we take the resulting theorem

```
# list_of_stream(top_thms());;
val it : thm list = [|- 2 + 2 = 4]
```

Finally, we can change the proof strategy to find both solutions by using backtracking

```
# bb();;
val it : mgoalstack =
`2 + 2 = x`

# ee (CONCAT_SLV REFL_SLV (ACCEPT_SLV(ARITH_RULE `2 + 2 = 4`)));;
val it : mgoalstack =
# list_of_stream(top_thms());;
val it : thm list = [|- 2 + 2 = 2 + 2; |- 2 + 2 = 4]
```

The function

```
solve : solver -> term -> (term list * thm) stream
```

runs the proof search non interactively and produces a list of solutions as already shown in Section 6.3. In this last case it would be

```
# list_of_stream
    (solve (CONCAT_SLV REFL_SLV (ACCEPT_SLV(ARITH_RULE `2 + 2 = 4`)))
           `??x. 2 + 2 = x`);;
val it : (term list * thm) list =
  [([`x = 2 + 2`], |- 2 + 2 = 2 + 2);
   ([`x = 4`], |- 2 + 2 = 4)]
```

6.5 Case study: Evaluation for a lisp-like language

The material in this section is strongly inspired from the ingenious work of Byrd, Holk and Friedman about the miniKanren system [Byrd et al., 2012], where the authors work with the semantics of the Scheme programming language. Here, we target a lisp-like language, implemented as an object language inside the HOL prover. Our language is substantially simpler than Scheme; in particular, it uses dynamic (instead of lexical) scope for variables. Nonetheless, we believe that this example can suffice to illustrate the general methodology.

First, we need to extend our HOL Light environment with an object datatype `sexp` for encoding S-expressions.

```
let sexp_INDUCT,sexp_RECUR = define_type
  "sexp = Symbol string
        | List (sexp list)";;
```

For instance the sexp `(list a (quote b))` is represented as HOL term with

```
`List [Symbol "list";
       Symbol "a";
       List [Symbol "quote";
             Symbol "b"]]`
```

This syntactic representation can be hard to read and gets quickly cumbersome as the size of the terms grows. Hence, we also introduce a notation for concrete sexp terms, which is activated by the syntactic pattern `'(...)`. For instance, the above example is written in the HOL concrete syntax for terms as

```
`'(list a (quote b))`
```

With this setup, we can easily specify the evaluation rules for our minimal lisp-like language. This is an inductive predicate with rules for: (i) quoted expressions; (ii) variables; (iii) lambda abstractions; (iv) lists; (v) unary applications. We define a ternary predicate `` `EVAL e x y` ``, where e is a variable environment expressed as associative list, x is the input program and y is the result of the evaluation.

```
let EVAL_RULES,EVAL_INDUCT,EVAL_CASES = new_inductive_definition
  `(!e q. EVAL e (List [Symbol "quote"; q]) q) /\
   (!e a x. RELASSOC a e x ==> EVAL e (Symbol a) x) /\
   (!e l. EVAL e (List (CONS (Symbol "lambda") l))
                 (List (CONS (Symbol "lambda") l))) /\
   (!e l l'. ALL2 (EVAL e) l l'
             ==> EVAL e (List (CONS (Symbol "list") l)) (List l')) /\
   (!e f x x' v b y.
      EVAL e f (List [Symbol "lambda"; List[Symbol v]; b]) /\
      EVAL e x x' /\ EVAL (CONS (x',v) e) b y
      ==> EVAL e (List [f; x]) y)`;;
```

We now use our framework for running a certified evaluation process for this language. First, we define a solver for a single step of computation.

```
let STEP_SLV : solver =
  COLLECT_SLV
    [CONJ_SLV;
     ACCEPT_SLV EVAL_QUOTED;
     THEN_SLV (RULE_SLV EVAL_SYMB) RELASSOC_SLV;
     ACCEPT_SLV EVAL_LAMBDA;
     RULE_SLV EVAL_LIST;
     RULE_SLV EVAL_APP;
     ACCEPT_SLV ALL2_NIL;
     RULE_SLV ALL2_CONS];;
```

In the above code, we collect the solutions of several different solvers. Other than the five rules of the `EVAL` predicate, we include specific solvers for conjunctions and for the two predicates `REL_ASSOC` and `ALL2`.

The top-level recursive solver for the whole evaluation predicate is now easy to define:

```
let rec EVAL_SLV : solver =
   fun g -> CONCAT_SLV ALL_SLV (THEN_SLV STEP_SLV EVAL_SLV) g;;
```

Let us make a simple test. The evaluation of the expression

```
((lambda (x) (list x x x)) (list))
```

can be obtained as follows:

```
# get (solve EVAL_SLV
             `??ret. EVAL []
                          '((lambda (x) (list x x x)) (list))
                          ret`);;

val it : term list * thm =
  ([`ret = '(() () ())`],
   |- EVAL [] '((lambda (x) (list x x x)) (list)) '(() () ()))
```

Again, we can use the declarative nature of logic programs to run the computation backwards. For instance, one intriguing exercise is the generation of *quine* programs, that is, programs that evaluates to themselves. In our formalization, they are those terms q satisfying the relation `` `EVAL [] q q` ``. The following command computes the first two quines found by our solver.

```
# let sols = solve EVAL_SLV `??q. EVAL [] q q`);;
# take 2 sols;;

val it : (term list * thm) list =
  [([`q = List (Symbol "lambda" :: _3149670)`],
    |- EVAL [] (List (Symbol "lambda" :: _3149670))
       (List (Symbol "lambda" :: _3149670)));
   ([`q =
      List
      [List
       [Symbol "lambda"; List [Symbol _3220800];
        List [Symbol "list"; Symbol _3220800; Symbol _3220800]];
       List
       [Symbol "lambda"; List [Symbol _3220800];
        List [Symbol "list"; Symbol _3220800; Symbol _3220800]]]`],
    |- EVAL []
       (List
       [List
        [Symbol "lambda"; List [Symbol _3220800];
         List [Symbol "list"; Symbol _3220800; Symbol _3220800]];
        List
        [Symbol "lambda"; List [Symbol _3220800];
         List [Symbol "list"; Symbol _3220800; Symbol _3220800]]])
       (List
       [List
        [Symbol "lambda"; List [Symbol _3220800];
         List [Symbol "list"; Symbol _3220800; Symbol _3220800]];
        List
        [Symbol "lambda"; List [Symbol _3220800];
         List [Symbol "list"; Symbol _3220800; Symbol _3220800]]]))]
```

One can easily observe that any lambda expression is trivially a quine for our language. This is indeed the first solution found by our search:

```
([`q = List (Symbol "lambda" :: _3149670)`],
 |- EVAL []
        (List (Symbol "lambda" :: _3149670))
        (List (Symbol "lambda" :: _3149670)))
```

The second solution is more interesting. Unfortunately it is presented in a form that is hard to decipher. A simple trick can help us to present this term as a concrete sexp term: it is enough to replace the HOL generated variable (`` `_3149670` ``) with a concrete string. This can be done by an ad hoc substitution.

```
# let [_; i2,s2] = take 2 sols;;
# vsubst [`"x"`,hd (frees (rand (hd i2)))] (hd i2);;

val it : term =
  `q = '((lambda (x) (list x x)) (lambda (x) (list x x)))`
```

If we take one more solution from `sols` stream, we get a new quine, which, interestingly enough, is precisely the one obtained in [Byrd et al., 2012].

```
val it : term =
  `q =
   '((quote (lambda (x) (list x (list (quote quote) x))))
     (quote (quote (lambda (x) (list x (list (quote quote) x))))))`
```

Conclusions

We presented a rudimentary framework implemented on top of the HOL Light theorem prover that enables a logic programming paradigm for proof searching. More specifically, it facilitates the use of meta-variables in HOL goals and permits backtracking during the proof construction.

It would be interesting to enhance our framework with more features:

- Implement higher-order unification such as Miller's higher-order patterns, so that our system can enable higher-order logic programming in the style of λProlog.
- Support constraint logic programming, e.g., by adapting the data structure that represent goals.

Despite the simplicity of the present implementation, we have already shown the implementation of some paradigmatic examples of logic-oriented proof strategies. In the code base, some further examples are included concerning a quicksort implementation and a simple example of a logical puzzle.

7
Bibliography

S. Bacchelli, E. Barcucci, E. Grazzini, and E. Pergola. Exhaustive generation of combinatorial objects by ECO. *Acta Informatica*, 40(8):585–602, Jul 2004. ISSN 1432-0525. DOI: 10.1007/s00236-004-0139-x.

D. Baccherini, D. Merlini, and R. Sprugnoli. Binary words excluding a pattern and proper Riordan arrays. *Discrete Mathematics*, 307:1021–1037, 2007.

E. Barcucci, A. Del Lungo, J.M. Fédou, and R. Pinzani. Steep polyominoes, q-Motzkin numbers and q-Bessel functions. *Discrete Mathematics*, 189(1):21 – 42, 1998a. ISSN 0012-365X. DOI: https://doi.org/10.1016/S0012-365X(97)00275-6. URL `http://www.sciencedirect.com/science/article/pii/S0012365X97002756`.

E. Barcucci, A. Del Lungo, E. Pergola, and R. Pinzani. A methodology for plane tree enumeration. *Discrete Mathematics*, 180(1):45 – 64, 1998b. ISSN 0012-365X. DOI: https://doi.org/10.1016/S0012-365X(97)00122-2. URL `http://www.sciencedirect.com/science/article/pii/S0012365X97001222`. Proceedings of the 7th Conference on Formal Power Series and Algebraic Combinatorics.

E. Barcucci, A. Del Lungo, E. Pergola, and R. Pinzani. ECO:a methodology for the enumeration of combinatorial objects. *Journal of Difference Equations and Applications*, 5(4-5): 435–490, 1999. DOI: 10.1080/10236199908808200.

P. Barry. *Riordan Arrays: A Primer*. Lulu.com, 2017. ISBN 9781326855239. URL `https://books.google.it/books?id=7fGBDgAAQBAJ`.

K. Beck. *Test-Driven Development: By Example*. Pearson Education, 2002.

J. Bell and B. Stevens. A survey of known results and research areas for n-queens. *Discrete Mathematics*, 309(1):1 – 31, 2009. ISSN 0012-365X. DOI:

https://doi.org/10.1016/j.disc.2007.12.043. URL http://www.sciencedirect.com/science/article/pii/S0012365X07010394.

A. Bernini, I. Fanti, and E. Grazzini. An exhaustive generation algorithm for Catalan objects and others. 2007.

S. Bilotta, E. Pergola, and E. Grazzini. Counting Binary Words Avoiding Alternating Patterns. *Journal of Integer Sequences*, 16, 2013.

L. Brugnano and D. Trigiante. *Solving Differential Problems by Multistep Initial and Boundary Value Methods*. Australia Etc. : Gordon & Breach, 1998.

W. E. Byrd. Relational Programming in miniKanren: Techniques, Applications, and Implementations. 2009. PhD dissertation.

W. E. Byrd, E. Holk, and D. P. Friedman. miniKanren, Live and Untagged: Quine Generation via Relational Interpreters (Programming Pearl). In *Proceedings of the 2012 Annual Workshop on Scheme and Functional Programming*, Scheme '12, pages 8–29, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1895-2. DOI: 10.1145/2661103.2661105. URL http://doi.acm.org/10.1145/2661103.2661105.

G.-S. Cheon and J.-S. Kim. Stirling matrix via Pascal matrix. *Linear Algebra and its Applications*, 329:49 – 59, 2001.

F.R.K. Chung, R.L. Graham, V.E. Hoggatt, and M. Kleiman. The number of baxter permutations. *Journal of Combinatorial Theory, Series A*, 24(3):382 – 394, 1978. ISSN 0097-3165. DOI: https://doi.org/10.1016/0097-3165(78)90068-7. URL http://www.sciencedirect.com/science/article/pii/0097316578900687.

M. Delest, D. Gouyou-Beauchamps, and B. Vauquelin. Enumeration of parallelogram polyominoes with given bond and site perimeter. 3:325–339, 01 1987.

M.-P. Delest and J.-M. Fedou. Enumeration of skew Ferrers diagrams. *Discrete Mathematics*, 112(1-3):65–79, 1993.

E. Deutsch, L. Ferrari, and S. Rinaldi. Production matrices. *Advances in Applied Mathematics*, 34(1):101 – 122, 2005. ISSN 0196-8858. DOI: https://doi.org/10.1016/j.aam.2004.05.002. URL http://www.sciencedirect.com/science/article/pii/S0196885804000673.

E. Deutsch, L. Ferrari, and S. Rinaldi. Production Matrices and Riordan Arrays. *Annals of Combinatorics*, 13(1):65–85, Jul 2009. ISSN 0219-3094. DOI: 10.1007/s00026-009-0013-1. URL https://doi.org/10.1007/s00026-009-0013-1.

D. Friedman, W. E. Byrd, and O. Kiselyov. *The Reasoned Schemer, Second Edition.* MIT Press, 2018.

F. R. Gantmacher. *The Theory of Matrices, volume one.* Chelsea, New York, USA., 1959.

S. Golomb. *Polyominoes.* Princeton University Press, 2nd edition, 1996. ISBN 9780691024448.

G. H. Golub and C. F. Van Loan. *Matrix Computations.* Johns Hopkins University Press, Baltimore, MD, USA, third edition, 1996., 1996.

D. Goodger. Polyominoes: Puzzles & Solutions, 2015. URL `http://puzzler.sourceforge.net`.

L. J. Guibas and A. M. Odlyzko. Long repetitive patterns in random sequences. *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete*, 53(3):241–262, Jan 1980. ISSN 1432-2064. DOI: 10.1007/BF00531434. URL `https://doi.org/10.1007/BF00531434`.

L. J. Guibas and A. M. Odlyzko. String overlaps, pattern matching, and nontransitive games. *Journal of Combinatorial Theory, Series A*, 30(2):183 – 208, 1981. ISSN 0097-3165. DOI: https://doi.org/10.1016/0097-3165(81)90005-4. URL `http://www.sciencedirect.com/science/article/pii/0097316581900054`.

T.-X. He. Matrix characterizations of Riordan arrays. *Linear Algebra and its Applications*, 465:15 – 42, 2015.

J. Hemann and D. P. Friedman. μKanren: a Minimal Functional Core for Relational Programming. 2013. Scheme2013, Alexandria.

N. J. Higham. *Functions of Matrices.* Siam, 2008.

H. Hitotumatu and K. Noshita. A technique for implementing backtrack algorithms and its application. *Information Processing Letters*, 8(4):174 – 175, 1979. ISSN 0020-0190. DOI: https://doi.org/10.1016/0020-0190(79)90016-4. URL `http://www.sciencedirect.com/science/article/pii/0020019079900164`.

R. A. Horn and C. R. Johnson. *Topics in Matrix Analysis.* Cambridge University Press, 1991.

D. E. Knuth. Dancing links. *eprint arXiv:cs/0011047*, November 2000.

V. Lakshmikantham and D. Trigiante. *Theory of Difference Equations : Numerical Methods and Applications.* New York: Marcel Dekker, 2002.

P. Lancaster and M. Tismenetsky. *The Theory of Matrices, second edition.* Academic Press, London, 1985.

A. Luzon, D. Merlini, M. A. Moron, and R. Sprugnoli. Identities induced by Riordan arrays. *Linear Algebra and its Applications*, 436(3):631 – 647, 2012. ISSN 0024-3795. DOI: https://doi.org/10.1016/j.laa.2011.08.007. URL http://www.sciencedirect.com/science/article/pii/S0024379511005805.

A. Luzon, D. Merlini, M. A. Moron, and R. Sprugnoli. Complementary Riordan arrays. *Discrete Applied Mathematics*, 172:75 – 87, 2014. ISSN 0166-218X. DOI: https://doi.org/10.1016/j.dam.2014.03.005. URL http://www.sciencedirect.com/science/article/pii/S0166218X14001280.

A. Luzon, D. Merlini, M. A. Moron, L. F. Prieto-Martinez, and R. Sprugnoli. Some inverse limit approaches to the Riordan group. *Linear Algebra and its Applications*, 491:239 – 262, 2016.

M. Maggesi and M. Nocentini. Kanren Light, 2018. URL https://github.com/massimo-nocentini/kanren-light.

D. Merlini and M. Nocentini. Algebraic Generating Functions for Languages Avoiding Riordan Patterns. *Journal of Integer Sequences*, 21, 2018. URL https://cs.uwaterloo.ca/journals/JIS/VOL21/Merlini/merlini5.html.

D. Merlini and M. Nocentini. Functions and jordan canonical forms of riordan matrices. *Linear Algebra and its Applications*, 565:177 – 207, 2019. DOI: https://doi.org/10.1016/j.laa.2018.12.011.

D. Merlini and R. Sprugnoli. Algebraic aspects of some Riordan arrays related to binary words avoiding a pattern. *Theoretical Computer Science*, 412(27):2988 – 3001, 2011. ISSN 0304-3975. DOI: https://doi.org/10.1016/j.tcs.2010.07.019. URL http://www.sciencedirect.com/science/article/pii/S0304397510004056. Combinatorics on Words (WORDS 2009).

D. Merlini, D. G. Rogers, R. Sprugnoli, and M. C. Verri. On Some Alternative Characterizations of Riordan Arrays. *Canadian Journal of Mathematics*, 49:301 – 320, 08 1997.

D. Merlini, R. Sprugnoli, and M. C. Verri. Combinatorial Sums and Implicit Riordan Arrays. *Discrete Math.*, 309(2):475–486, January 2009. ISSN 0012-365X. DOI: 10.1016/j.disc.2007.12.039. URL http://dx.doi.org/10.1016/j.disc.2007.12.039.

A. Meurer, C. P. Smith, M. Paprocki, O. Čertík, S. B. Kirpichev, M. Rocklin, A. Kumar, S. Ivanov, J. K. Moore, S. Singh, T. Rathnayake, S. Vig, B. E. Granger, R. P. Muller, F. Bonazzi, H. Gupta, S. Vats, F. Johansson, F. Pedregosa, M. J. Curry, A. R. Terrel, Š. Roučka, A. Saboo, I. Fernando, S.h Kulal, R. Cimrman, and A. Scopatz. SymPy: symbolic computing in Python. *PeerJ Computer Science*, 3:e103, January 2017. ISSN 2376-5992. DOI: 10.7717/peerj-cs.103. URL `https://doi.org/10.7717/peerj-cs.103`.

C. Moler and C. Van Loan. Nineteen Dubious Ways to Compute the Exponential of a Matrix, Twenty-Five Years Later. *SIAM review*, 45:3 – 49, 2003.

H. D. Nguyen and D. Taggart. Mining the Online Encyclopedia of Integer Sequences, 2013. Preprint.

M. Nocentini. Simulation Methods. URL `https://massimo-nocentini.github.io/simulation-methods/build/html/index.html`.

M. Nocentini. *OEIS Tools*. Open School on Combinatorial Method in the analysis of Algorithms and Data Structures, SKKU University, Korea, 2017. URL `http://massimo-nocentini.github.io/PhD/skku-aorc-2017/oeistools.html`.

M. Nocentini. μKanren Implementations, 2018. In the following repositories, indexed by programming languages:

Python `https://github.com/massimo-nocentini/microkanrenpy`

Scheme `https://github.com/massimo-nocentini/on-scheme/blob/master/src/microkanren.scm`

Smalltalk `https://github.com/massimo-nocentini/microkanrenst`

OCaml `https://github.com/massimo-nocentini/kanren-light`

respectively.

H. Runckel and U. Pittelkow. Practical computation of matrix functions. *Linear Algebra and its Applications*, 49:161 – 178, 1983.

F. Ruskey. *Combinatorial Generation*. 2003. URL `http://www.1stworks.com/ref/RuskeyCombGen.pdf`.

R. Sedgewick and P. Flajolet. *An Introduction to the Analysis of Algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1996. ISBN 0-201-40009-X.

L. W. Shapiro, S. G., W.-J. Woan, and L. C. Woodson. The Riordan group. *Discrete Applied Mathematics*, 34(1):229 – 239, 1991. ISSN 0166-218X. DOI: https://doi.org/10.1016/0166-218X(91)90088-E. URL http://www.sciencedirect.com/science/article/pii/0166218X9190088E.

N. J. A. Sloane. The Encyclopedia of Integer Sequences. URL http://oeis.org/.

R. Sprugnoli. Riordan arrays and combinatorial sums. *Discrete Mathematics*, 132(1):267 – 290, 1994. ISSN 0012-365X. DOI: https://doi.org/10.1016/0012-365X(92)00570-H. URL http://www.sciencedirect.com/science/article/pii/0012365X9200570H.

SymPy. SymPy's documentation, a. URL http://docs.sympy.org/latest/index.html.

SymPy. SymPy's tutorial, b. URL http://docs.sympy.org/latest/tutorial/index.html.

G. van Rossum and J.J. Davis. A Web Crawler With asyncio Coroutines. URL http://www.aosabook.org/en/500L/a-web-crawler-with-asyncio-coroutines.html.

L. Verde-Star. Functions of matrices. *Linear Algebra and its Applications*, 406:285 – 300, 2005.

H. S. Warren. *Hacker's Delight.* Addison-Wesley Professional, 2nd edition, 2012. ISBN 0321842685, 9780321842688.

P. E. Weidmann. Sequencer. URL https://github.com/p-e-w/sequencer.