JACOBS UNIVERSITY

Formula Semantification and Automated Relation Finding in the OEIS

by

Enxhell Luzhnica

Bachelor Thesis in Computer Science

Prof. Dr. Michael Kohlhase
Name and title of the supervisor

Date of Submission: May 8, 2016

Jacobs University — School of Engineering and Science

With my signature, I certify that this thesis has been written by me using only the indicated resources and materials. Where I have presented data and results, the data and results are complete, genuine, and have been obtained by me unless otherwise acknowledged; where my results derive from computer programs, these computer programs have been written by me unless otherwise acknowledged. I further confirm that this thesis has not been submitted, either in part or as a whole, for any other academic degree at this or another institution. Parts of this thesis have been pre-published [LIK15] and are copied verbatim. The contribution of the other authors will be explicitly mentioned.

Enxhell Luzhnica　　　　　　　　Bremen, 07.05.2016

Contents

Abstract

The On-line Encyclopedia of Integer Sequences (OEIS) is the largest database of its kind and an important resource for mathematicians. The database is well-structured and rich in mathematical content but is informal in nature, so knowledge management services are not directly applicable. In this paper we provide a partial parser for the OEIS that leverages the fact that, in practice, the syntax used in its formulas is fairly regular. Then, we import the result into OMDoc to make the OEIS accessible to OMDoc-based knowledge management applications. We exemplify this with a formula search application based on the MathWebSearch system and a program that finds relations between the OEIS sequences.

1 Introduction

Integer sequences are important mathematical objects that appear in many areas of mathematics and science and are studied in their own right. The On-line Encyclopedia of Integer Sequences (OEIS) [Inc15b] is a publicly accessible, searchable database documenting such sequences and collecting knowledge about them. The effort was started in 1964 by N. J. A. Sloane and led to a book [NJ 73] describing 2372 sequences which was later extended to over 5000 in [NS95]. The online version [N J94] started in 1994 and currently contains over 250000 documents from thousands of contributors with 15000 new entries being added each year [N J12]. Documents contain varied information about each sequence such as the beginning of the sequence, its name or description, formulas describing it, or computer programs in various languages for generating it.

The OEIS library is an important resource for mathematicians. It helps to identify and reference sequences encountered in their work and there are currently over 4000 books and articles that reference it. Sequences can be looked up using a text-based search functionality that OEIS provides, most notably by giving the name (e.g. "Fibonacci") or starting values (e.g. "$1, 2, 3, 5, 8, 13, 21$"). However, given that the source documents describing the sequences are mostly informal text, more semantic methods of knowledge management and information retrieval are limited.

In this paper we tackle this problem by building a (partial) parser for the source documents and importing the OEIS library into the OMDoc/MMT format which is designed for better machine support and interoperability. This opens up the OEIS library to OMDoc-based knowledge management applications, which we exemplify by a semantic search application based on the MathWebSearch [KP13] system that permits searching for text and formulas and by a relation finder.

This paper is organized as follows: in Section 2 we briefly introduce the OMDoc/MMT language, OEIS and the theory of generating functions. In Section 3 we discuss related work and in Section 4 we describe our import of the OEIS library into OMDoc. In Section 5 we show an initial application of our import by providing formula search for the OEIS library. Then, in Section 6 we present the relation finder and in Section 7 we discuss future work and conclude.

Acknowledgements The author would like to thank Prof. Dr. Michael Kohlhase for his guidance in this research project and Mihnea Iancu for the valuable discussions and support.

2 Preliminaries

2.1 The OEIS document format

The *internal format* [OEI16] is line-based in the sense that each line starts with a marker that represents the kind of content found in that line. We briefly introduce the relevant kinds below but refer to [OEI16] for details.

1. The *identification* line gives the unique ID of the sequence declared in that document.
2. The *start values* lines give the beginning of the sequence. It is used for searching as well as lexicographic ordering of the sequences.
3. The *name* line gives the name, a brief description or the definition of the sequence.
4. The *references* lines contain references to papers which in turn refer to or are concerned with the sequence described in the current document.
5. The *formula* lines give formulas that define or hold for the sequence described in the current document. The formulas are in plain text ASCII syntax that is similar to LaTeX math markup and can contain text as part of the formula or as comments. Among others, here we find the representing functions and the generating functions (starting with `G.f.`) of the sequence in hand.
6. The *author* line gives the document author typically containing the name and e-mail address.
7. The *examples* lines give examples and additional information about the initial terms of the sequences

The actual document-level syntax of the internal format is presented below in Figure 1 where we use $\underline{\circ}$ for required lines, $[\,\circ\,]$ for optional parts, $\circ^*$ for repetitions and $\circ_1 \mid \circ_2$ for alternatives. Moreover we use *italic* for grammar productions, `typewriter font` for fixed syntax and normal font for informal descriptions.

Ident.	::=	%I *ID* $[M_n]$ $[N_n]$
ID	::=	Identification number in [Inc15b]
M_n	::=	Identification number in [NS95]
N_n	::=	Identification number in [NJ 73]
Values	::=	%S *ID* nr* [%T ID nr* [%U *ID* nr*]]
Name	::=	%N *ID* text
References	::=	%D *ID* text
Formula	::=	%F *ID* formula
Author	::=	%A *ID* text
Examples	::=	%E *ID* text

Figure 1: Grammar of OEIS Internal Format

[Fibonacci numbers] The article for Fibonacci numbers [Inc15a] was one of the first entries in the OEIS and is one of the most comprehensive. We will use it as a running example throughout the paper, although we will heavily trim the document for simplicity by presenting here only a few sanitized lines. Listing 2.1 shows the document with identification, values, name and reference lines, followed by three formula lines and the author line.

```
%I A000045 M0692 N0256
%S A000045 0,1,1,2,3,5,8,13,21,34,55,89,144,233,377,610,987
%N A000045 Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.
%D A000045 V. E. Hoggatt, Jr., Fibonacci and Lucas Numbers. Houghton, Boston, MA,
     1969.
%F A000045 F(n) = ((1+sqrt(5))^n-(1-sqrt(5))^n)/(2^n*sqrt(5))
%F A000045 G.f.: Sum_{n>=0} x^n * Product_{k=1..n} (k + x)/(1 + k*x). - _Paul D.
    Hanna_, Oct 26 2013
%F A000045 This is a divisibility sequence; that is, if n divides m, then a(n)
    divides a(m)
```

```
%A A000045 _N. J. A. Sloane_, Apr 30 1991
```

2.2 OMDoc/MMT

The MMT language [Rab11] is designed to be applicable to a large collection of declarative formal base languages and all MMT notions are fully abstract in the choice of the base language. For clarification, typical base languages are logics, type theories, or programming languages. In MMT, mathematical knowledge is defined in terms of four important units: documents, modules, symbols and objects. The biggest unit of mathematical knowledge, the document, is represented in MMT as a theory graph. **Theory Graph** is one of the central notions of MMT which contains *theories* or modules and *views* (morphisms between theories). To understand this better we will define some additional notions.
Theories are defined by a set of typed **symbols** (the signature) and a set of **axioms** describing the properties of the symbols.
Symbols are either constant declarations (declaring constants, type axioms, lemmas, definitions) or a structure declaration (which declares an import from another theory). In the latter case, all the symbols from the imported theory become available. In the theory graph this is represented by an arrow from the imported theory to the importing one. A **signature morphism** σ from a source theory *S* to a target theory *T* translates or interprets the symbols of *S* in *T*.
If we have entailment relations for the formulas of S and T, a signature morphism that translates all theorems of *S* to theorems of *T* is called a **view**. So, in MMT a **view** from *S* to *T* is a list of assignments $c \mapsto \omega$ where c is an *S*-constant (axiom) and ω is a *T*-term (proof). This list of assignments induces a homomorphic translation of S-terms to T-terms by replacing every c with the corresponding ω.
The object level provides the structure of the symbols inside a theory; objects are terms produced by a grammar in which the basic units are constants, variables, applications and bindings.

Let us consider the theories (Fig. 2) of **group, monoid** and **unital set** of a monoid and one view **m** which shows that the unital set of a monoid is a group.

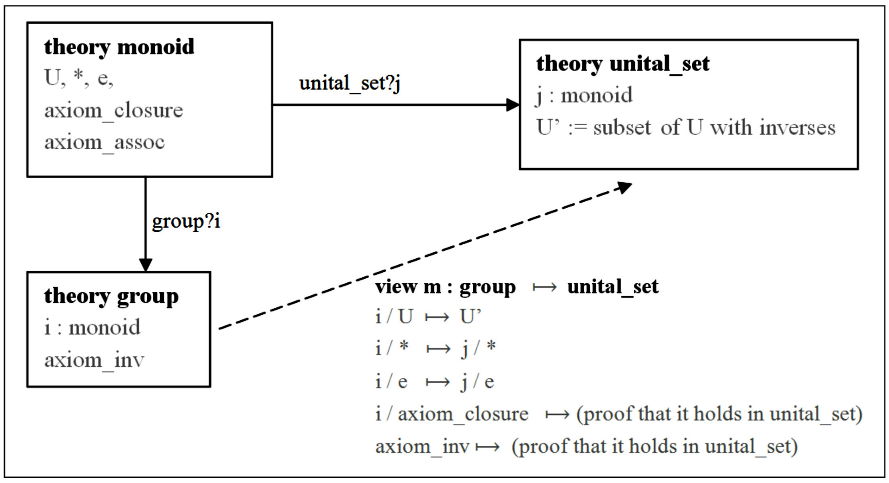

theory monoid
U, *, e,
axiom_closure
axiom_assoc
unital_set?j
theory unital_set
j : monoid
U' := subset of U with inverses
group?i
theory group
i : monoid
axiom_inv
view m : group ⟼ unital_set
i / U ⟼ U'
i / * ⟼ j / *
i / e ⟼ j / e
i / axiom_closure ⟼ (proof that it holds in unital_set)
axiom_inv ⟼ (proof that it holds in unital_set)

Figure 2: An Example Theory Graph Comprising Three Theories

The imports (**unital_set?j** and **group?i**) are marked by solid arrows. Each import carries all the symbols from the source theory to the destination. In theory **monoid** the declared constants are **U**-underlying set, $*$-monoid operation, **e**-unity element and **axiom_closure, axiom_assoc**.

Theory **group** imports from monoid via the structure **i** and formulates the axiom of invertibility. Theory **unital_set** imports from monoid via **j** and declares **U'**-the set of invertible elements in **U**. The view **m** is mapping all the symbols and proving the axioms still hold with the new mapping in **unital_set**.

OMDoc [Koh06] is a content markup format and data model for mathematical documents. It models mathematical content using three levels.

Object Level: uses OpenMath and MathML as established standards for the markup of *formulae*.

Statement Level: supplies original markup for explicitly representing the various kinds of mathematical statements including *symbol declarations* and *definitions* (which introduce a new symbol names), *assertions* (which can represent theorems, lemmas or corollaries), and *examples*.

Theory Level: offers original markup that allows for clustering sets of statements into *theories* as well as specifying relations between them (inclusions, morphisms).

The Mmt [RK13] language can be seen as a restricted version of OMDoc but with a fully specified semantics. Additionally, for Mmt there is an Mmt system [Rab13] which is a Scala-based [EPF] open source implementation of the Mmt language. The key features of the Mmt system for this paper are that it provides an infrastructure for writing importers from any compatible format into Mmt as well as exporters from Mmt, most notably into (MathML-enriched) HTML for both local and web-based presentation.

For the sake of simplicity, we often do not differentiate between Mmt and OMDoc languages in the following and refer to [Koh06] and, respectively, [RK13] for details on each language. Throughout this paper we will use OMDoc/MMT to refer to both OMDoc and Mmt.

2.3 Theory of Generating Functions

Generating functions are used to represent sequences efficiently by coding the terms of a sequence as coefficients of powers of a variable x in a formal power series. Unlike an ordinary series, this formal series is allowed to diverge, meaning that the generating function is not always a true function and the "variable" is actually an indeterminate.

Generating functions are used for solving combinatorial problems, recurrences and most importantly in our context, analyzing various properties of sequences. There are different types of generating functions, including **ordinary generating functions, exponential generating functions, Lambert series, Bell series**, and **Dirichlet series**.

The **ordinary generating function** (or just **generating function**) of a sequence $a_0, a_1, a_2 \ldots, a_{n-1}, a_n, \ldots$ is the infinite series:

$$GF(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-1} x^{n-1} + a_n x^n + \ldots = \sum_{i=0}^{\infty} a_i x^i \qquad (1)$$

For example, the sequence $(1, 1, 1, 1, 1, \ldots)$ can be represented as:

$$1 + x + x^2 + \ldots = \frac{1}{1-x} \tag{2}$$

We know that the equation above only holds for $|x| < 1$ but we ignore the issues of convergence, as already mentioned. Thus, the ordinary generating function of this sequence can be written as $\frac{1}{1-x}$.

Due to this representation, generating functions have some interesting properties which we are going to discuss shortly. Below are some properties of the ordinary generating functions.

Let $f(x) = \sum_{i=0}^{\infty} a_i x^i$, $g(x) = \sum_{i=0}^{\infty} b_i x^i$, $c \in \mathbb{R}$ and $k \in \mathbb{Z}$, then

$$f(x) + g(x) = \sum_{i=0}^{\infty} (a_i + b_i) x^i \tag{3}$$

$$cf(x) = \sum_{i=0}^{\infty} c a_i x^i \tag{4}$$

$$x^k f(x) = \sum_{i=0}^{\infty} a_i x^{i+k} \tag{5}$$

$$\frac{d}{dx} f(x) = \sum_{i=0}^{\infty} (i+1) a_{i+1} x^i \tag{6}$$

$$\int f(x)dx = \sum_{i=0}^{\infty} \frac{1}{i+1} a_i x^{i+1} \tag{7}$$

From equation 3 the we understand that adding two generating functions is the same as adding the sequences they represent term by term. From equation 4, multiplying a generating function by some constant is the same as multiplying each term of the sequence by that same constant. From equation 5, multiplying the sequence by x^k is the same as shifting the sequence to the left ($k < 0$) or right ($k > 0$). If shifted to the right, the new terms added in the beginning are 0s. From equation 6, taking the derivative of a generating function is the same as multiplying each term by its index and shifting the sequence to the left by one. From equation 7, taking the integral is the same as dividing each term by its index and shifting the sequence to the right by one.

3 Related Work and Comparisons

The content representation of mathematical expressions has been widely researched from different perspectives. The main problem of processing natural language along a complex mathematical symbolism has been mainly approached in two ways [Gin11]. The first approach tries to make a controlled natural language and make the language incrementally more powerful while disallowing ambiguities. The ambiguity resolution here is usually based on an explicit context and/or type inference. The top-down approach tries to model the existing discourse of mathematics. The parsing of mathematical expressions in done mainly using context-free grammars aided with a type system or special purpose grammars that are based on type systems. The work done using these approaches generally aims to parse a large set of topics in the mathematical discourse. In our case we have a more specific mathematical topic to parse and the discourse is only limited to mathematical expressions. Our current approach is a top-down approach as we will see in Section 4.

On the other hand, the work in finding relations between sequences has been centered around numerical based approaches, mainly due to the representation of OEIS. Generally, the high level approach is to apply different numerical transformations on the available starting values of the integer sequences and then check for matches. The checking for matches is usually done with exact matches on either all starting values of a sequence or only a subset of it, or even analyzing the sequence that comes out of some predefined error function. Transformations are tuned for relations that the author is interested in, but keeping in mind the search costs. Two approaches of this nature have been compiled in Appendix A.

An immediate advantage of the numerical approach is that one can, in theory, investigate for all kinds of relations between sequences. One does not need to be restricted by linear or polynomial relations but can formulate arbitrarily complex relations to search for. In practice that needs special care since it quickly becomes intractable because of the high cost algorithms. For instance, with a naive algorithm, when searching for a linear relation, $P = a * X + b * Y + c$, for sequences P, X, Y, we have to find constants a, b, c in addition to the sequences X, Y, Z.
A disadvantage is that the found relations are only conjectures since the matching is done on a finite subsequence, which usually is less than 30 terms. An extreme example of two sequences that match for a long time but are not equal, is the following:

$$\left\lfloor \frac{2n}{log(2)} \right\rfloor \quad \text{and} \quad \left\lceil \frac{2}{2^{1/n} - 1} \right\rceil$$

They agree for the first 777451915729367 terms [N J12]!

In fact, a similar approach from Ralf Stephan [Ste04a] yielded 117 conjectures from which 17 of them are still conjectures[Ste04b], which shows that these conjectures are not always easy to prove or disprove.

4 Parsing the OEIS Formula Format

We built a partial parser for OEIS formulas by identifying and analyzing well-behaved formulas to produce a workable grammar. We leverage the fact that, although there is no standardized format for OEIS formulas, many of them use a sufficiently regular syntax.

OEIS is known for the human-readable mathematical terms, so a variety of syntactic rules are encountered when forming these mathematical terms. We use the following classification for notations, inspired by [CKS11] and further motivated by our analysis of the OEIS formulas:

1. **infix operators** are used to combine two terms to one complex term, e.g the `+` symbol in `m+n`.
2. **suffix operators** are added after a term to form another term, e.g. the `!` symbol in `n!`.

3. **prefix operators** (with or without bracketed application) are added in front of a term to form another term, e.g. `sin` in `sin(x)` or `sin x`, respectively.
4. **infix relation symbols** are used to construct a formula out of two terms, e.g. the < symbol in `x<2`.
5. **binding operators** that bind a context to a body to construct a term, e.g. the $\forall$ symbol in $\forall$`x. x^2 > 0`

The classification presented above guides our grammar and, in principle, covers virtually all important notations used in OEIS formulas. However, in practice, we encountered several important challenges which we discuss individually below.

Open Set of Primitives Since the formulas are not standardized, not only is the syntax flexible, but so is the set of primitive operators that are used. For instance, the formulas in Listing 2.1 (on lines 5-6) use square root, power, as well as the sum (Σ) and product (Π) binders. The challenges arise because of the many different notations used for such primitives. For instance, in line 6 of Listing 2.1 the range for sum and product is given in two different ways. Similar problems appear with limits and integrals as well as numerous atypical infix and suffix operators. In order to parse these correctly, we investigate the documents and the grammar failures manually and incrementally extend the grammar.

Ambiguity As it is often the case with informal, presentation-oriented formulas, there can be ambiguity in the parsing process when there exist several reasonable interpretations. Since the OEIS syntax is not fixed, this is quite common, so we do additional disambiguation during parsing to resolve most of the ambiguities. Here we discuss a few of the many ambiguities that arise.

The multiplication sign is usually implicit so, instead of `a*(x+y)`, we encounter `a(x+y)` which could represent either a function application or a multiplication depending on whether `a` is a function or an individual (constant or variable). There is no general way to solve this, so we rely on several heuristics. First, we check if the symbol in question is used somewhere else in the same formula with an unambiguous meaning. Specifically, we default to function application unless the same symbol is used as an individual somewhere else in the formula. This already disambiguates most such cases in OEIS but we use several additional heuristics. For instance, having `name(`arg`)` will result in marking `name` as function since it is unlikely to be a multiplication between two individuals. Similarly, having `name(`arg_1`,...,`arg_n`)` results in marking `name` as a function.

The natural way of using the power operator also leads to ambiguities. For example, `T^2(y)` is used for $(T(y))^2$, however `T^y(x^2+2)` is ambiguous. We solve this using similar heuristics as for the implicit multiplication.

For unbracketed function application as in `sin x`, we rely on the heuristic that this form of function application is used only in well known functions. Therefore, we code these notations for well known functions in the grammar itself. This form of function application can also mean multiplication, for instance `Pi x`. One can already see that parsing and disambiguating the mathematical expressions in this context has a lot of aspects. Additional cases of ambiguities are handled in similar ways and we omit the details for brevity.

Delineating formulas OEIS formula lines freely mix text and formulas so it is required to correctly distinguish between text and formula parts within the lines in order to accurately parse each line. For instance, line 6 in Listing 2.1 starts with the text `G.f.:` (meaning "Generating function:") and continues with the formula. The line then has the author and date, separated from the formula by a dash (`-`) which could also be interpreted as a minus and, therefore, a continuation of the formula. In the extraction of the formulas we use the help of a dictionary. The text in the OEIS documents has words that are not found in the dictionaries since it contains many technical terms so we first run a pre-parsing procedure which enriches the dictionary. The final grammar tries to parse words until it fails and then tries to parse formulas; this process repeats.

4.1 Importing into OMDoc/MMT

For each OEIS document we create a corresponding OMDoc/MMT document that contains a single theory. Then, OEIS lines roughly correspond to OMDoc/MMT declarations inside that theory. We use the OEIS sequence ID as the name of the OMDoc theory. Then, the identification line produces an OMDoc symbol declaration representing the sequence (as a function from integers to values). The start values and example lines are both represented as OMDoc/MMT examples. Specifically, the starting values are considered as examples of sequence elements. Formula lines are represented as OMDoc assertions (about the sequence symbol). Finally, name, reference and author lines are represented as metadata using the Dublin Core standard.

[Fibonacci Numbers] The corresponding OMDoc/MMT document for the Fibonacci numbers article from Example 2.1 is shown in Listing 4.1. We omit most formulas and some XML boilerplate for conciseness and simplicity.

```
<omdoc xmlns:dc="http://purl.org/dc/elements/1.1/">
  <theory id="A000045">
  <metadata>
    <dc:creator>N. J. A. Sloane</dc:creator>
    <dc:title>Fibonacci numbers</dc:title>
  </metadata>
  <symbol name="seq"/>
  <assertion>
    <!-- OpenMath for ∀n.seq(n) = ((1+√5)^n−(1−√5)^n)/(2^n√5) -->
    <OMBIND>
      <OMS cd="arith" name="forall"/>
      <OMBVAR> <OMV name="n"/> </OMBVAR>
      <OMA>
        <OMS cd="arith" name="equal"/>
        <OMA><OMS name="seq"/><OMV name="n"></OMA>
        ⋮
      </OMA>
    </OMBIND>
  </assertion>
  ⋮
  </theory>
</omdoc>
```

4.2 Implementation and Evaluation

The importer is implemented in Scala as an extension for the MMT system and consists of about 2000 lines of code. It is available at https://svn.kwarc.info/repos/MMT/src/mmt-oeis/. The implementation is mostly straightforward, other than the formula parser which we discuss separately below.

There are 257654 documents in OEIS totaling over 280MB of data. The OMDoc/MMT import expands it to around 9GB, partly due to the verbosity of XML and partly due to producing the semantic representation of formulas. The total running time is around 1h40m using an Intel Core i5, 16GB of RAM and a SATA hard drive.

Formula parsing The formula parser is implemented using the Packrat Parser [For02] for which Scala provides a standard implementation. Packrat parsers allow us to write left recursive grammars while guaranteeing a linear time worst case which is important for scaling to the OEIS.

There are 223866 formula lines in OEIS and the formula parser succeeds on 201384 (or 90%) of them. Out of that, 196515 (or 97.6%) contain mathematical expressions. Based on a manual inspection of selected formulas we determined that most parser fails occur because of logical connectives since those are not yet supported. Other failures include wrong formula delineation because of unusual mix of formulas and text.

The statistics above refer just to the successful parses, but we cannot automatically evaluate if the result returned by the parser is actually the expected one. For this, we did a manual evaluation of the parsing result for 40 randomly selected OEIS documents and evaluated 85% of successfully parsed formulas as semantically correct. The main contributor of incorrect formula parses was badly delineated formulas, which causes text to be wrongly parsed as part of a formula.

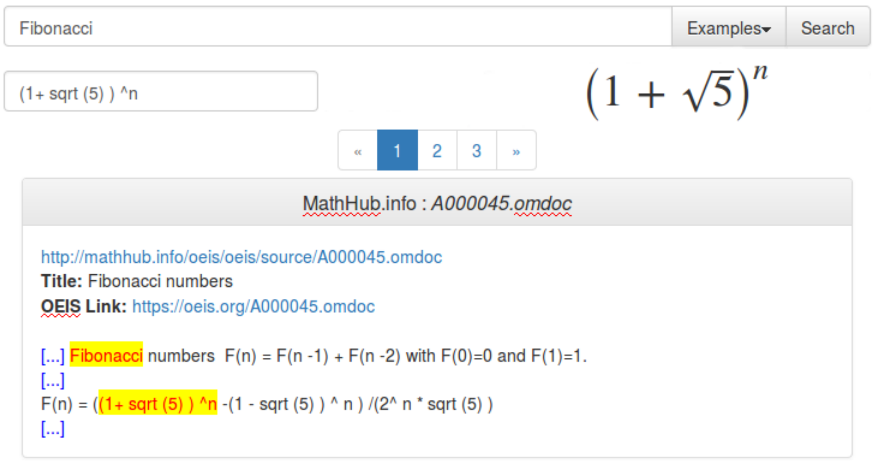
5 Search

[Acknowledgements: The author gratefully acknowledges the practical help of Mihnea Iancu on making the MathWebSearch *instance]*

MathWebSearch (MWS)[KP13] is an open-source, open-format, content-oriented search engine for mathematical expressions. We refer to [KP13] for details.

To realize the search instance in MWS we need to provide two things:

1. A *harvest* of MathML-enriched HTML files that the search system can resolve queries against. The content-MathML from the files will be used to resolve the formula part of the query while the rest of the HTML will be used for the text part. The harvest additionally requires a configuration file that defines the location in the HTML files of MWS-relevant metadata such as the title, author or URL of the original article. This, together with the HTML itself is used when presenting the query results.
2. A *formula converter* that converts a text-based formula format into MathML. This will be used so that we can input formulas for searching in a text format (in our case OEIS-inspired ASCII math syntax) rather than writing MathML directly.



MathHub.info : A000045.omdoc

http://mathhub.info/oeis/oeis/source/A000045.omdoc
Title: Fibonacci numbers
OEIS Link: https://oeis.org/A000045.omdoc

[...] Fibonacci numbers F(n) = F(n -1) + F(n -2) with F(0)=0 and F(1)=1.
[...]
F(n) = ((1+ sqrt (5)) ^n -(1 - sqrt (5)) ^ n) /(2^ n * sqrt (5))
[...]

Figure 3: Text and Formula Search for OEIS

To produce the harvest of the OEIS library for MWS we export the HTML from the content imported into MMT. We reuse the MMT presentation framework and only enhance it with OEIS-specific technicalities such as sequence name or OEIS link. For the formula converter we use the same parser used for OEIS formulas and described above, except extended with one grammar rule for MWS *query variables*. We then forward the resulting formula in MMT to produce the presentation (MATHML) and return it to the MWS frontend. The web-server infrastructure, needed to communicate with MWS, is provided by MMT and we just extend it. Figure 3 shows (a part of) the current interface answering a query about Fibonacci numbers. The search system is available at `http://ash.eecs.jacobs-university.de:9999/`.

6 Enriching the OEIS with a Theory Graph

In the proceeding subsections we first propose a way of representing the OEIS Theory Graph. Then, we discuss an approach of finding views and discuss the results we get from it.

6.1 Representing OEIS in Theory Graph

First, we need to define the *theory* in the context of OEIS, hence we need to define all the notions related to the *theory* in this context. We propose the following way to define the theory of a sequence.
A sequence S has a corresponding theory which has a typed symbol, $S : nat \mapsto int$ to represent the function symbol and all the functions are expressed as axioms. Assuming the sequence S contains two functions $f(n)$ and $g(n)$ the axioms would be:

$$\begin{aligned} S_f &: S = \lambda n.\ f(n) \\ S_g &: S = \lambda n.\ g(n) \end{aligned} \tag{8}$$

Now that we have a way to define the theory, we will show the other notions related to it. The theory inclusions are practically the same as in other contexts. The theory T_i

that includes theory T_j imports all the symbols of theory T_j and they become available in theory T_i. An implication of this theory representation is that we can represent relations between sequences as views between their corresponding theories.
Assume that we have two sequences S_i and S_j and that their respective theories are T_i and T_j. If their corresponding functions are f and g, respectively, then theory T_i and T_j can be represented as:

$$T_i = \begin{cases} S_i : nat \mapsto int \\ S_f : S_i = f \end{cases}$$

and

$$T_j = \begin{cases} S_j : nat \mapsto int \\ S_g : S_j = g \end{cases}$$

If f and g are related through a function $h : (nat \mapsto int) \mapsto (nat \mapsto int)$ such that, $f = h(g)$ then one can construct a view ϕ from T_i to T_j, and that is:

$$\phi = \begin{bmatrix} S_i \mapsto h\ S_j \\ S_f \mapsto \text{proof that the axiom holds in } T_j \end{bmatrix} \tag{9}$$

One can easily construct the proof given that we know the function h.

A more general case is when the relation occurs between two functions of different sequences but one of the sequences has more than one representing function. Let S_i be a sequence which has m functions and the corresponding theory T_i. Since all the functions represent the same sequence then they must be equal. We take these as theorems stating the equality of the m functions with each other. These are then contained in a special theory U, included by all the corresponding theories of sequences.
To show this general case, let us have the sequence S_i as above, and a sequence S_j with k functions. Let there be a relation $h : (nat \mapsto int) \mapsto (nat \mapsto int)$ that maps one of the functions of S_j, say g_1 to function f from S_i. A view θ, from T_i to T_j needs to have proofs for all the axioms of T_i under the symbol assignments in θ. The symbol assignment is the same as in the case when S_j had only one function. From the available theorems in theory U we get $g_1 = g_2 = ... = g_k$. Thus, from $f = h(g_1)$ we get $f = h(g_i)$ for i in $\{1, \ldots, k\}$, which simplifies the matter to the already explained initial case. We can encode similarly the case when the view is based on the generating functions.

6.2 Algorithm

The current representation of the library allows us to access the formulas present in the OEIS documents. Although the representing functions are available, we do relation searching on the level of the ordinary generating functions. The approach we follow is mathematically simple. We will show three methods, each building on top of the previous one.

Method 1 In this case, we *normalize* the ordinary generating functions of the sequences and check for equality between the normalized expressions. The normalization rules are defined as follows:

$$\frac{cG \quad c\text{ - constant} \quad G\text{ - generating function}}{G}\ (\textsc{Const})$$

$$\frac{x^n G \quad x\text{ - the indeterminate of } G \quad G\text{ - generating function}}{G}\ (\textsc{Unshift})_n$$

$$\frac{P/Q \quad P, Q\text{ - polynomials}}{(\sum_{i=0}^{n} p_i x^i)/(\sum_{i=1}^{m} q_i x^i) \quad p_n > 0 \quad q_n > 0}\ (\textsc{Sort})$$

The rules are composed in the same order as defined above to form the method *normalize*. The first two rules are self-evident in what they are supposed to achieve. Rule (SORT) is used for sorting the polynomials based on ascending powers of x. We add an extra condition on the sign of the highest degree term to normalize the sign of the polynomial.

The normalization rules entail easily interpretable relations with the pre-normalized formula. Rule (CONST) is removing the multiplying constant. Multiplying an ordinary generating function by a constant is equivalent to multiplying each term of the sequence by that number. Since multiplying by x^n is the same as shifting the sequence the (UNSHIFT)$_n$ is normalizing the shift.

Let B (before) be the pre-normalized ordinary generating function, and A (after) the normalized one. Also, let $\hat{B}(n), \hat{A}(n)$ be the representing functions of the sequence represented by B, A, respectively. Then, the relations that the normalizing rules entail are:

1. Rule (CONST): $\hat{A}(n) = \frac{1}{c}\hat{B}(n)$.
2. Rule (UNSHIFT)$_k$: $\hat{A}(n) = \hat{B}(n+k)$ given that x^k was removed.
3. Rule (SORT): $\hat{A}(n) = \hat{B}(n)$.

Intuitively, in this case we are checking if sequences are scaled and/or shifted versions of each other. These relations are not meant to be interesting or new.

Method 2 In this case we check if a sequence can be expressed as a sum of other sequences existing in the OEIS, possibly *normalized* and *transformed*.

A simplified algorithm roughly follows this pseudocode:

```
foreach sequence
    foreach ogf in ordinaryGeneratingFunction(sequence)
        add normalize(ogf) to hashSet

foreach sequence
    foreach ogf in ordinaryGeneratingFunction(sequence)
        pdf = partialFractionDecomposition(ogf)
        partialFractions = decompose(partialFractions(pfd))
        relationsExists =
        forall pgf in partialFractions
                transformedPartialFractions = normalize(applyTransformations(pgf))
                transformedPartialFractions.intersection(hashSet).length > 0
        if (relationsExists)
            print relations
```

We will now explain the functions that we are using above.

Let GF_n be one of the ordinary generating functions of sequence A_n. The partial fraction decomposition (*partialFractionDecomposition*) would leave us with:

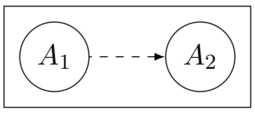

$$\begin{tikzcd} A_1 \arrow[r, dashed] & A_2 \end{tikzcd}$$

Figure 4: Relations of *Method 2*

$$GF_n = \sum_{j=1}^{n} G_j \tag{10}$$

where G_j is also an ordinary generating function. The function *partialFractions* extracts the summands, in this case, the partial fractions (ordinary generating functions) themselves. The function *decompose* does a further step of decomposition. If $G_j = \frac{P}{Q}$ where P, Q are polynomials ($P = \sum_{i=0}^{n} a_i x^i$) then it rewrites $G_j = \sum_{i=0}^{n} \frac{a_i x^i}{Q}$. These summands are then considered partial fractions too. The reason we allow this is because $a_i x^i$ is just doing scaling and shifting on the sequence represented by the ordinary generating function $1/Q$.

The transformations are *integration, differentiation* and *unit*. The transformations are selected such that expressions that match under these transformations can be easily related both mathematically and semantically.

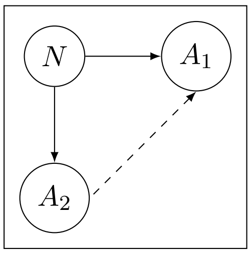

$$\begin{tikzcd} N \arrow[r] \arrow[d] & A_1 \\ A_2 \arrow[ur, dashed] & \end{tikzcd}$$

Figure 5: Relations of *Method 3*

Similarly as above, let B be the pre-transformed ordinary generating function, and A the transformed one. Let $\hat{B}(n), \hat{A}(n)$ be the representing function of the sequence represented by B, A, respectively. Then, the relations that the transformation rules entail are:

1. Integration: $\hat{A}(n) = \frac{1}{n}\hat{B}(n-1)$.
2. Differentiation: $\hat{A}(n) = n\hat{B}(n+1)$.
3. Unit: $\hat{A}(n) = \hat{B}(n)$.

If GF_n is the ordinary generating function of sequence A_n, c_i a real constant, $p \in \{-1, 0, 1\}$ and k_i an integer, the algorithm would try to find relations of the form:

$$A_j(n) = \sum_{i=0}^{n} c_i n^p A_i(n - k_i) \tag{11}$$

In this case we will be finding what we call *direct relations*. So there is a direct *view* between the sequences (Fig. 4).

Method 3 This is a more generalized case of the second method. In this case we do not restrict the generating functions to be representing existing OEIS sequences. Instead, we search if two ordinary generating functions have a common partial fraction (generating function) (Fig. 5).

The simplified algorithm is given by the pseudocode below.

```
foreach sequence
    foreach ogf in ordinaryGeneratingFunction(sequence)
        pfd = partialFractionDecomposition(ogf)
        partialFractions = decompose(partialFractions(pfd))
        foreach pgf in partialFraction
            transformations = normalize(applyTransformations(pgf))
            foreach transformed in transformations
            if transformed exists in hashSet
                print relation
            add pgf to hashSet
```

As an example, let us take two sequences A_1 and A_2 and their corresponding ordinary generating functions GF_1 and GF_2. Let N be a partial fraction and M, S sums of partial fractions. Moreover, for any G, if G is an ordinary generating function, let $\hat{G}$ denote the representing function of the same sequence that G represents. If

$$GF_1 = M + \int N \tag{12}$$

$$GF_2 = S + N \tag{13}$$

then from the relations defined above and assuming that the representing functions exist, we can carry over the relation to the representing functions:

$$\hat{GF_1}(n) = \hat{M}(n) + \frac{1}{n}\hat{N}(n-1) \tag{14}$$

$$\hat{GF_2}(n) = \hat{S}(n) + \hat{N}(n) \tag{15}$$

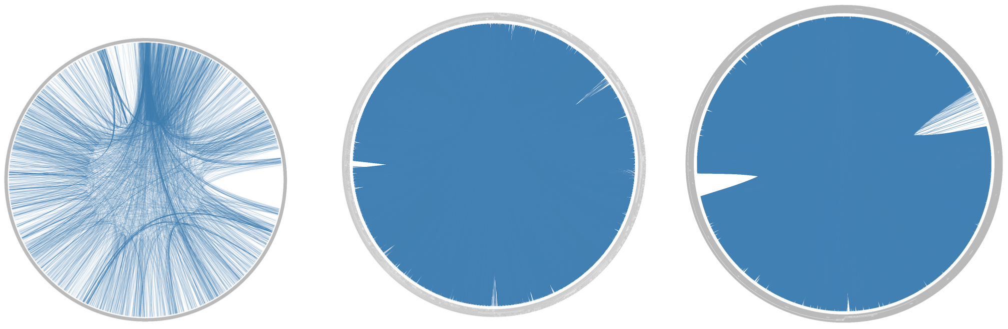
Which then can be solved to give:

$$\hat{GF_2}(n) = \hat{S}(n) + (n+1)(\hat{GF_1}(n+1) - \hat{M}(n+1)) \tag{16}$$

Notice that the step of carrying over the relation between generating functions, to the corresponding representing functions assumes that the representing functions exist, which is not always the case. Nonetheless, it is worth noticing that the last equation is independent of $\hat{N}$.

6.3 Implementation and Evaluation

The relation finder is implemented in Scala and is available at `https://github.com/eluzhnica/OEIS`. The page will be kept up to date with results. The implementation of the


(a) OEIS Theory Graph of Explicit Views (b) OEIS Theory Graph of Inferred Views using Method 3 (c) OEIS Theory Graph of Inferred Views using Method 2

The points around the circle represent the theories and the blue lines views between them. The theories presented here are only the ones for which we have parsed the generating functions.

Figure 6: OEIS Theory Graphs

normalization rules makes use of the parsing tree of the expression. The *transformations* are done using SageMath as a math engine. Our Scala code communicates with a local SageMath server using a REST API.

We show below some examples of the relations found from each method.

Method 1 This is more of a sanity check of the algorithm. Due to the nature of these relations, these are self-evident relations. Additionally, these relations can be effectively searched utilizing a numerical method. An instance that our algorithm finds is that sequence $A001478(n) = -A000012(n)$. Sequence $A001478$ is the sequence of the negative integers, while $A000012$ is the sequence of positive integers.

Method 2 An example of this method, which we have submitted and it is accepted in the OEIS (https://oeis.org/A037532), is as follows. Sequence $A037532$ has an ordinary generating function:

$$\frac{x(1+x+2x^2)}{(x-1)(7x-1)(1+x+x^2)} \overset{\text{PFD}}{=} \frac{1}{57}\frac{5x+3}{x^2+x+1} - \frac{\frac{29}{171}}{7x-1} + \frac{\frac{2}{9}}{x-1} \tag{17}$$

$$\overset{\text{Decomp.+(Sort)}}{=} \frac{1}{57}\frac{5x}{1+x+x^2} + \frac{1}{57}\frac{3}{1+x+x^2} - \frac{\frac{29}{171}}{-1+7x} + \frac{\frac{2}{9}}{-1+x} \tag{18}$$

Sequence $A049347$:

$$\frac{1}{1+x+x^2} \tag{19}$$

Sequence $A000420$:

$$\frac{1}{1-7x} \overset{\text{(Sort)}}{=} -\frac{1}{-1+7x} \tag{20}$$

Sequence $A000012$:

$$\frac{1}{1-x} \overset{\text{(Sort)}}{=} -\frac{1}{-1+x} \tag{21}$$

It is clear that the relation that can be derived here is:

$$A037532(n) = \frac{5}{57}A049347(n-1) + \frac{3}{57}A049347(n) + \frac{29}{171}A000420(n) - \frac{2}{9} \tag{22}$$

There is one subtlety that needs to be explained. The sequence with ordinary generating function $\frac{1}{1-x}$ is the sequence $(1, 1, 1, \ldots)$. However, for simplicity we write down $\frac{2}{9}$ instead of $\frac{2}{9}A000012(n)$.

Method 3 An example relation is shown below.

Sequence $A257890$ has a generating function:

$$\frac{(x^2-x+1)(x^2-3x+3)}{(x-1)^6} = \frac{1}{(x-1)^2} + \frac{1}{(x-1)^4} + \frac{1}{(x-1)^6} \tag{23}$$

Sequence $A257199$ has a generating function:

$$\frac{x(1-x+x^2)}{(1-x)^6} = \frac{1}{(x-1)^3} + \frac{2}{(x-1)^4} + \frac{2}{(x-1)^5} + \frac{1}{(x-1)^6} \tag{24}$$

Clearly, $\frac{1}{(x-1)^6}$ is in common. But also, $-5\int \frac{1}{(x-1)^6}dx = \frac{1}{(x-1)^5}$, which makes it possible to express a relation through $\frac{1}{(x-1)^5}$, too. In this case we can use the technique used in Equation 16 to carry over the relation to the representing functions since the representing functions of the corresponding sequences are known. Hence, we get the following relation:

$$\begin{aligned} &A257890(n) = A000027(n+1) + A000292(n+1) \\ &- \frac{n+1}{10}(A257199(n+1) + A000217(n+2) - 2 * A000292(n+2) - A000389(n+6)) \end{aligned} \tag{25}$$

Nonetheless, in the general case the relation cannot be always carried over to the representing function, so the relation would only be in the level of the generating function. Also, as for the current implementation, the step of getting the relation to the representing functions is done manually.

Since our parser runs over all the formulas of OEIS, we have extracted the existing explicit relations in OEIS and made a graph (Figure 6a) showing the existing connections between sequences . The second method enriches the theory graph as shown in Figure 6c and the third method as shown in Figure 6b).

Parsed Generating Functions	43005
SageMath verified Generating Functions	16065
Parsed Ordinary Generating Functions	35953
SageMath verified Ordinary Generating Functions	13400
Method 1 relations	4859
Sequences in Method 1 relations	853
Method 2 relations	297284646
Method 2 relations without normalization	66427
Method 3 relations	24615176
Method 3 relations without normalization	1866654

We converted the generating functions to a SageMath syntax and checked if SageMath accepts the expressions. From manual inspection we found out that most of the expressions that were not being accepted were because they referred to some other functions. For instance, $1 + Q(0)$ where the function $Q(n)$ is defined later on in the sequence document. We currently do not resolve these references.

Method 1 reports 4859 relations of that kind. However, in total only 853 sequences can be normalized to other existing sequences.

It is noticeable that there are a lot of relations found using the third method. However, one needs to keep in mind that for each common partial fraction there is a relation generated. Moreover, if one partial fraction without any transformation is in common in two generating functions, then all of the transformed versions of that partial fraction will also be in common which then would count as extra relations. This adds to the number of relations found. Another thing to consider is that not all of relations can be carried over to the representing functions, as already mentioned. One must interpret the result presented having this in mind. Nonetheless, we exclude the relations found by having a constant in common or any transformation of them. These relations are not interesting since a lot of sequences have a constant generating function in their partial decomposition. We also remove relations that emerge from the intersection of ordinary generating functions of the form x^n or $\frac{1}{1-x}$ or transformations of these. For this method, further consideration must go into building a set of heuristics that remove trivial and uninteresting relations.

One can also notice that the number of possible relations generated from the second method is also big. This is due to the number of relations found using the normalization rules (Method 1). Take for instance, $G = A + B + C$, and say that each of A, B, C is an OEIS sequence and is related with 3 other sequences under the normalization rules. Then the number of relations that we can form is actually 4^3. For this reason, we also report the number of relations when we remove the relations that come due to the normalization. So, in the example above the relation would count only once, instead of 4^3.

Out of three submissions, two relations are already accepted in the OEIS under the author's name. One of them has already been presented in Equation 22 and the other relation is $A001787(n) = A007283(n)\frac{n}{6}$ which can be found at `https://oeis.org/A001787`. The unaccepted submission was not perceived to add new information since a similar relation was already present. The submitted relations were selected randomly.

7 Conclusions and Future Work

In this thesis we improved the digitalization of the OEIS followed by two applications that this representation opens. First, the MathWebSearch instance on OEIS which allows the users to make text and formula queries. Second, a way of generating knowledge from OEIS, specifically, relations between sequences.

The OEIS formula syntax is excessively flexible and the parser uses certain heuristics that in some cases fail. Besides, there are cases that we know we cannot parse and we do not plan to parse through the parser since adding that flexibility to the language makes the parser too permissive. We plan to make a multi-step transformation of OEIS. First, we make a parser which normalizes the formula language of OEIS to an unambiguous one. Second, our current parser transforms the presently parsed expressions to the unambiguous language. Third, the formulas that are left uncovered are transformed manually. This allows to have higher confidence in the validity of the parsed expressions.

Having a representation like this can be helpful in maintaining OEIS, too. First of all, this would avoid mistakes in the formula language. One can also integrate a math engine to automatically check the correctness of the submissions. For instance, when a new generating function is submitted the program can make an initial check if the generating function produces the already known terms of the sequences in hand.

OEIS has made a long journey from publishing sequences in books to a fast changing web-based knowledge source. However, programs that derive from OEIS would benefit from a friendlier API. An important point is that OEIS can help the clients keep up with the updates. The clients simply needs to update the reliant system on the archives that changed and not re-run everything over the whole corpus. In the case of our experiment we would have also appreciated a friendlier API which would allow the user to submit new relations.

References

[1] Marcos Cramer, Peter Koepke, and Bernhard Schröder. “Parsing and Disambiguation of Symbolic Mathematics in the Naproche System”. In: *Intelligent Computer Mathematics - 18th Symposium, Calculemus 2011, and 10th International Conference, MKM 2011, Bertinoro, Italy, July 18-23, 2011. Proceedings*. Ed. by James H. Davenport et al. Vol. 6824. Lecture Notes in Computer Science. Springer, 2011, pp. 180–195. ISBN: 978-3-642-22672-4. DOI: 10.1007/978-3-642-22673-1_13. URL: http://dx.doi.org/10.1007/978-3-642-22673-1_13.

[2] Bryan Ford. “Packrat Parsing:: Simple, Powerful, Lazy, Linear Time, Functional Pearl”. In: *Proceedings of the Seventh ACM SIGPLAN International Conference on Functional Programming*. ICFP ’02. Pittsburgh, PA, USA: ACM, 2002, pp. 36–47. ISBN: 1-58113-487-8. DOI: 10.1145/581478.581483. URL: http://doi.acm.org/10.1145/581478.581483.

[3] Deyan Ginev. “The Structure of Mathematical Expressions”. Master’s Thesis. Bremen, Germany: Jacobs University Bremen, Aug. 2011. URL: http://kwarc.info/people/dginev/publications/DeyanGinev_Mastersthesis.pdf.

[4] Mihnea Iancu and Michael Kohlhase. “Math Literate Knowledge Management via Induced Material”. In: *Intelligent Computer Mathematics*. LNCS. submitted. Springer, 2015, pp. 187–202. URL: http://kwarc.info/kohlhase/papers/cicm15-induced.pdf.

[5] The OEIS Foundation Inc. *Fibonacci Numbers, The On-Line Encyclopedia of Integer Sequences*. http://oeis.org/A000045. 2015.

[6] The OEIS Foundation Inc. *The On-Line Encyclopedia of Integer Sequences*. http://oeis.org/. 2015.

[7] Michael Kohlhase. OMDoc – *An open markup format for mathematical documents [Version 1.2]*. LNAI 4180. Springer Verlag, Aug. 2006. URL: http://omdoc.org/pubs/omdoc1.2.pdf.

[8] Michael Kohlhase and Corneliu Prodescu. “MathWebSearch at NTCIR-10”. In: *NTCIR Workshop 10 Meeting*. Ed. by Noriko Kando and Kazuaki Kishida. Tokyo, Japan: NII, Tokyo, 2013, pp. 675–679. URL: http://research.nii.ac.jp/ntcir/workshop/OnlineProceedings10/pdf/NTCIR/MATH/04-NTCIR10-MATH-KohlhaseM.pdf.

[9] Peter Liu. *Master thesis essay*. https://cs.uwaterloo.ca/~shallit/peter-liu-masters.pdf. 1994.

[10] Enxhell Luzhnica, Mihnea Iancu, and Michael Kohlhase. “Importing the OEIS Library Into OMDoc”. In: *LWA*. Vol. 1458. CEUR Workshop Proceedings. CEUR-WS.org, 2015, pp. 296–303.

[11] Florian Rabe. *The MMT Language and System*. Oct. 11, 2011. URL: https://svn.kwarc.info/repos/MMT (visited on 10/11/2011).

[12] N. J. A. Sloane. *An On-Line Version of the Encyclopedia of Integer Sequences*. http://www3.combinatorics.org/Volume_1/PDF/v1i1f1.pdf. 1994.

[13] N. J. A. Sloane. *The On-Line Encyclopedia of Integer Sequences*. http://neilsloane.com/doc/eger.pdf. 2012.

[14] N.J. A. Sloane. *A Handbook of Integer Sequences*. Academic Press, 1973.

[15] N.J. A. Sloane and Simon Plouffe. *The Encyclopedia of Integer Sequences*. Academic Press, 1995.

[16] OEIS. *OEIS Help*. http://oeis.org/oeishelp1.html. 2016.

[17] Florian Rabe. “The MMT API: A Generic MKM System”. In: *Intelligent Computer Mathematics*. Conferences on Intelligent Computer Mathematics. (Bath, UK, July 8–12, 2013). Ed. by Jacques Carette et al. Lecture Notes in Computer Science 7961. Springer, 2013, pp. 339–343. ISBN: 978-3-642-39319-8. DOI: 10.1007/978-3-642-39320-4.

[18] Florian Rabe and Michael Kohlhase. “A Scalable Module System”. In: *Information & Computation* 230 (2013), pp. 1–54. URL: http://kwarc.info/frabe/Research/mmt.pdf.

[19] École polytechnique fédérale de Lausanne. *The Scala Programming Language*. URL: http://www.scala-lang.org (visited on 10/22/2009).

[20] Ralf Stephan. *Prove or Disprove 100 Conjectures From The OEIS*. http://arxiv.org/pdf/math/0409509v4.pdf. 2004.

[21] Ralf Stephan. *State of 100 Conjectures From The OEIS*. http://www.ark.in-berlin.de/conj.txt. 2004.

Appendices

A Appendix A

OEIS has the mail server called Superseeker which tries over 120 transformation on the OEIS sequences to match a given sequence. This has also been used to generate relations between the current sequences by running it over the whole OEIS corpus. The steps involved in the Superseeker for a given sequence S are as follows [Liu94]:

1. Do an exact match on the current OEIS sequences.
2. Test if $S[n]$ is a polynomial in n.
3. Test if the difference of some order $S_d[n]$ are periodic.
4. Test if any row of the difference table of some depth is essentially constant.
5. Form some generating functions for the sequence for some predefined types (ie. ordinary gf, exponential gf etc.)
6. Look for a linear recurrence with polynomial coefficients for the coefficients of the above 6 types of generating functions.
7. Apply transformations to the sequence and lookup the result in the table.
8. If the original sequence is not in the table, find the 3 closest sequences in the table using the $L1$ metric. Only those with $L1 \leq 3$ are reported.

An approach from Peter Liu [Liu94] extends the search space of finding relations. While Superseeker uses the given sequence or its transformation sequences to query the database and the queries are exact match searches, his approach is to use the given sequence itself to query the database and the queries are not just exact matches. They could be the linear combination of two sequences in the table, or the affine transformation of a sequence or the polynomial of a sequence bounded by a constant. More specifically, if S is a given sequence and D the sequence in database, the searched relations are as follows [Liu94]:

1. Is there an exact match between S and $T_i \in D$?
2. Is S simply some sequence in the D shifted?
3. Are all member of S contained in some sequence $T_i \in D$?
4. Is S an affine transformation of some sequence $T_i \in D$?
5. Can S be written as a polynomial of degree bounded by a constant?
6. Can S be written as the linear combination of two T_i and $T_j \in D$?
7. Is the S close to a T_i?
8. Does S leave a constant remainder when divided by a T_i?
9. Does affine transformation of S give a subsequence of some $T_i \in D$?

Of course, these queries have to be efficient enough to run over the corpus and the author realizes that through pre-processing the database, use of hashing techniques and number theory. His approach yielded some new relations that the Superseeker could not find, speaking of the time that Peter Liu's work has been published. From query 7, the author found relations of the form $T_{1923} = 2 * T_{825} + T_{1428}$ or $T_{1746} = 2 * T_{1205}$.

While from query 4, there were more relations found and one of those is $T_{1567} = 2 * T_{1049} - 1 = 2 * T_{616} + 1 = 4 * T_{391} - 3$.