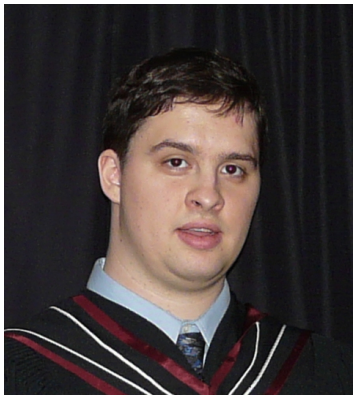
Recent results in combinatorics on words using an automatic prover

Jeffrey Shallit

School of Computer Science, University of Waterloo
Waterloo, Ontario N2L 3G1, Canada
shallit@cs.uwaterloo.ca
https://cs.uwaterloo.ca/~shallit

Some of my co-authors


Luke Schaeffer


Hamoon Mousavi


Narad Rampersad


Daniel Goč


Chen Fei Du


Émilie Charlier


Eric Rowland


Jean-Paul Allouche

This talk — summarized

1. There is a decision procedure for **deducing** and **proving** theorems about automatic sequences and related sequences based on first-order logic
2. Many properties of sequences that have long been studied in the literature can be phrased in first-order logic (including some for which this is not obvious!)
3. The decision procedure is relatively easy to implement and often runs remarkably quickly, despite its formidable worst-case complexity — and we have an implementation that is publicly available (`Walnut`)
4. The method can also be used to not simply decide, but also *enumerate*, many aspects of sequences

Seven Points of the Talk

5. Many results already in the literature (in dozens of papers and Ph. D. theses) can be reproved by our program in a matter of seconds (including fixing at least one that was wrong!)
6. Many new results can be proved
7. However, there are some well-defined limits to what we can do with the method because either
   - the property is not expressible in first-order logic; or
   - the underlying sequence leads to undecidability.

Some successes of the method

Unbordered factors

- A word w is said to be *bordered* if it can be written in the form xyx, with x nonempty.
- Otherwise w is *unbordered*.
- For example, the French word `lesquelles` is bordered, with $x = \texttt{les}$ and $y = \texttt{quel}$.

Improving a result of Currie and Saari

- Currie and Saari proved that if $n \not\equiv 1 \pmod 6$, then there is an unbordered factor of length n in the Thue-Morse sequence $\mathbf{t}$.
- However, this is not an iff: the length-31 factor 0011010010110100110010110100101 occurs in $\mathbf{t}$ but is not bordered.
- Raises the natural question: what is a precise characterization of the lengths of unbordered factors of $\mathbf{t}$?
- The logical method gives: $\mathbf{t}$ has an unbordered factor of length n iff $(n)_2 \notin 1(01^*0)^*(10^*1)$.
- We can also give a formula for the number of unbordered factors of each length.

Avoiding the pattern xxx^R

- Let x^R denote the reversal (*image miroir*) of the word x.
  - Example: if $x = \texttt{engager}$, then $x^R = \texttt{regagne}$.
- We are interested in the pattern xxx^R.
- An instance of the pattern xxx^R in French appears in the word choisissiez, where $x = \texttt{is}$.
- Can all instances of the pattern xxx^R be avoided in an infinite aperiodic binary word?
- We proved that it can.

Avoiding the pattern xxx^R

Consider the following Fibonacci-automaton:

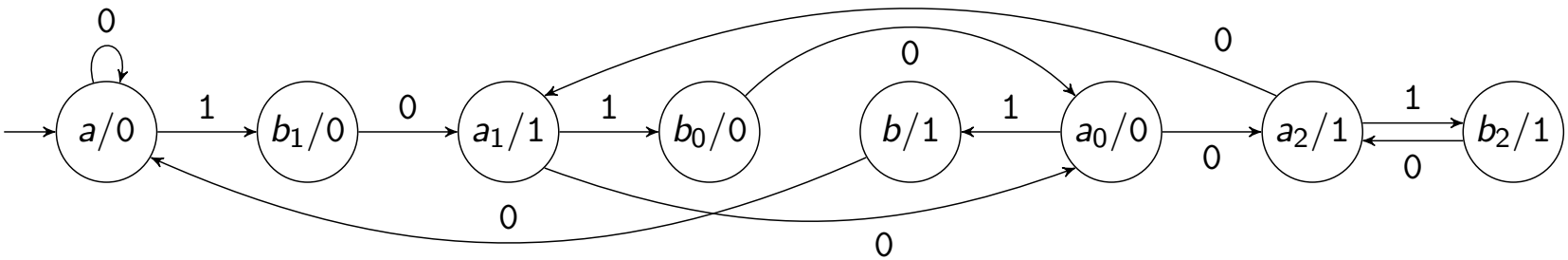


Figure : Fibonacci automaton generating the sequence **R**

It generates the sequence

$\mathbf{R} = 0010011011011001001101101100100100110110010010011011001\cdots$

Claim: **R** is aperiodic and avoids the patterns xxx^R and also xx^Rx^R.

Note: Narad Rampersad will speak more on this subject on Friday.

Repetitions in words

- A **square** is a word of the form xx, where x is a nonempty word
  - Examples in French include `rentrent` and `froufrou`
- An **overlap** is a word over the form $axaxa$, where a is a single letter and x is a possibly empty word
  - Examples in French include `entente` and `tentent`

Correcting an error of Fraenkel-Simpson

- Define the *finite Fibonacci words* X_n by $X_1 = 1$, $X_2 = 0$, and $X_n = X_{n-1}X_{n-2}$ for $n \geq 3$.
- Fraenkel and Simpson (1999) computed the number of occurrences $B(n)$ of squares in X_n:

$$B(n+1) = \frac{4}{5}nF_{n+1} - \frac{2}{5}(n+6)F_n - F_{n-1} + n + 1$$

for $n \geq 3$.

- But this is slightly wrong: the real formula is

$$B(n+1) = \frac{4}{5}nF_{n+1} - \frac{2}{5}(n+6)F_n - 4F_{n-1} + n + 1.$$

- The logical method can obtain the correct formula (almost) purely mechanically.

Bounds on pattern occurrence

The logical approach allows one to prove bounds on various quantities about automatic sequences.

Example: suppose $\mathbf{s} = (s_n)_{n \geq 0}$ is a k-automatic sequence.

We can prove theorems such as

Theorem. Let $f(t) = 2^{8t^2}$. If $\mathbf{s}$ is generated by an automaton with t states, and if $\mathbf{s}$ contains a square, then there is a square of order $\leq f(t)$ in the prefix of $\mathbf{s}$ of length $\leq f(t)$.

Factors in common

Theorem. If x is a factor of both the Thue-Morse sequence on $0, 1$ and the Rudin-Shapiro sequence on $0, 1$, then $|x| \leq 8$.

Theorem. If $\mathbf{f}$ and $\mathbf{g}$ specify two different paperfolding sequences $P_{\mathbf{f}}$ and $P_{\mathbf{g}}$, and ℓ is the smallest index for which $\mathbf{f}[\ell] \neq \mathbf{g}[\ell]$, then $P_{\mathbf{f}}$ and $P_{\mathbf{g}}$ have no factors of length $\geq 14 \cdot 2^{\ell}$ in common.

Theorem. There exists an computable function $f = f(k, m, n)$ with the property that if two k-automatic sequences, generated by automata with m (resp., n states), have a factor of length $\geq f(k, m, n)$ in common, then they have infinitely many factors in common.

An application to algebra

Theorem. There is an algorithm that, given n algebraic formal power series $f_1(X), \ldots, f_n(X)$ in $GF(q)[[X]]$ and a degree bound d, decides if there exist n polynomials $g_1(X), \ldots, g_n(X)$ in $GF(q)[X]$, not all 0, each of degree $\leq d$, such that

$$g_1 f_1 + \cdots + g_n f_n = 0.$$

Theorem. There is an algorithm that, given two algebraic formal power series $f(X)$ and $g(X)$ in $GF(q)[[X]]$, decides whether there exists some integer $t \geq 0$ such that $f(X)$ and $X^t g(X)$ agree on infinitely many coefficients.

First-order logic

By *first-order logic*, we mean the set of all formulas formed from

- any finite number of variables that can take values in some domain;
- equality defined on variables;
- possibly other comparison operators that can be applied to variables, such as less than, greater than, etc., depending on domain;
- possibly other functions applied to the variables, such as addition or multiplication;
- logical operations such as `and` ($\wedge$), `or` ($\vee$), `logical implication` ($\implies$), `iff` ($\iff$), and `not` ($\neg$);
- quantifiers, such as `for all` ($\forall$) and `there exists` ($\exists$).

First-order logic

- Variables can be either bound (by a quantifier) or unbound.
- If all variables are bound, then we can assign a truth value to the formula.
- If some variables are unbound, then we can consider the set of all values of the variables for which the formula is true.

A first-order logical theory is *decidable* if there is an algorithm that, given a well-formed formula with all variables bound, will decide its truth.

In the case of unbound variables, we'd like to algorithmically construct the representations of all integers for which the formula is true.

Presburger arithmetic

Presburger arithmetic is $\mathrm{Th}(\mathbb{N}, +)$, the first-order theory of the natural numbers $\mathbb{N} = \{0, 1, 2, \ldots\}$ with addition.

Mojżesz Presburger (1904–1943)
(died in the Holocaust)

Presburger arithmetic

- Sometimes Presburger arithmetic is written to include $<$, the "less-than" operator
- But it is not really needed, since the assertion $x < y$ is equivalent to $\exists z \ (z \neq 0) \ \wedge \ y = x + z$.

Example: The Chicken McNuggets Problem

A famous problem in elementary arithmetic books in the US:

At McDonald's, Chicken McNuggets are available in packs of either 6, 9, or 20 nuggets. What is the largest number of McNuggets that one cannot purchase?

Presburger arithmetic

In Presburger arithmetic we can express the "Chicken McNuggets theorem" that 43 is the largest integer that cannot be represented as a non-negative integer linear combination of $6, 9$, and 20, as follows:

$$(\forall n > 43\ \exists x, y, z \geq 0 \text{ such that } n = 6x + 9y + 20z) \wedge \neg(\exists x, y, z \geq 0 \text{ such that } 43 = 6x + 9y + 20z). \quad (1)$$

Here, of course, "$6x$" is shorthand for the expression "$x + x + x + x + x + x$", and similarly for $9y$ and $20z$.

Presburger's theorem

Presburger proved that $\mathrm{Th}(\mathbb{N}, +, 0, 1)$ is *decidable*: that is, there exists an algorithm that, given a well-formed formula in the theory, will decide its truth.

He used quantifier elimination.

Decidability of Presburger arithmetic: Büchi's proof

Julius Richard Büchi (1924–1984) found a much simpler proof of Presburger's result, based on automata. It gives us automata for the unbound variable case, too!

Ideas:

- represent integers in an integer base $k \geq 2$ using the alphabet $\Sigma_k = \{0, 1, \ldots, k-1\}$.
- represent n-tuples of integers as words over the alphabet Σ_k^n, padding with leading zeroes, if necessary. This corresponds to reading the base-k representations of the n-tuples *in parallel*.
- For example, the pair $(21, 7)$ can be represented in base 2 by the word

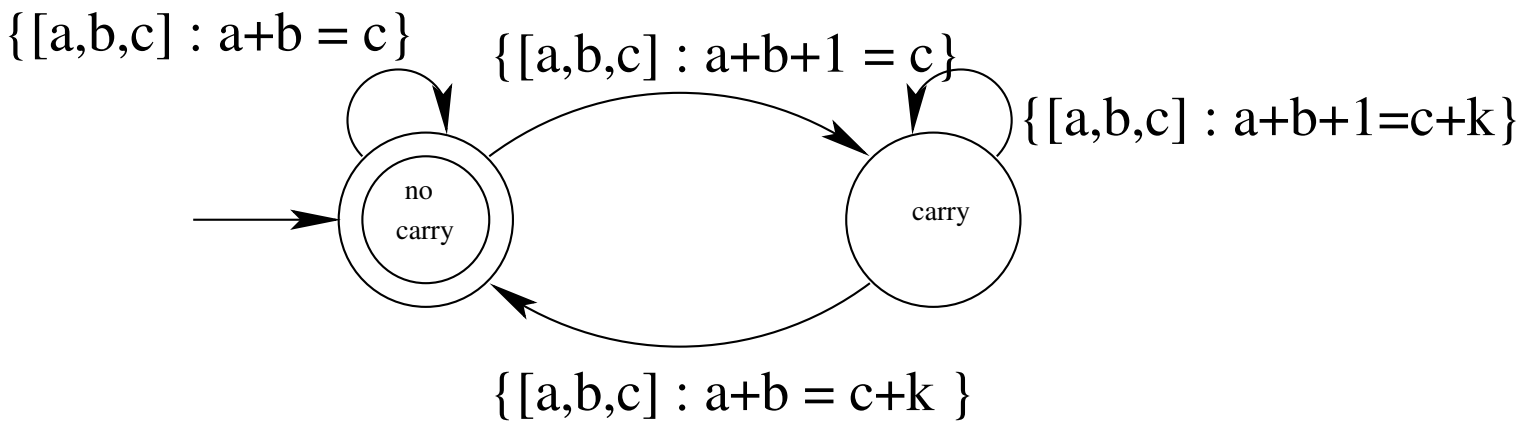
$$[1, 0][0, 0][1, 1][0, 1][1, 1].$$

Büchi's proof

- Automata will accept words over the alphabet Σ_k^n representing n-tuples of integers
- The language accepted is the set of all n-tuples of integers for which the formula (or subformula) is true
- Parsing the formula corresponds to performing operations on automata
- For example, if automaton M corresponds to some formula φ, then $\neg\varphi$ can be obtained by changing the "finality" of M's states: a final state becomes non-final and vice-versa
- Care is needed to handle the "leading zeroes" problem

Decidability of Presburger arithmetic

- The relation $x + y = z$ can be checked by a simple 2-state automaton depicted below, where transitions not depicted lead to a nonaccepting "dead state".



Decidability of Presburger arithmetic: proof sketch

- Relations like $x = y$ and $x < y$ can be checked similarly. (exercise)
- Given a formula with free variables $x_1, x_2, \ldots, x_n$, we construct an automaton accepting the base-k expansion of those n-tuples $(x_1, \ldots, x_n)$ for which the proposition holds.
- If a formula is of the form $\exists x_1, x_2, \ldots x_n \; p(x_1, \ldots, x_n)$, then we use nondeterminism to "guess" the x_i and check them.
- If the formula is of the form $\forall p$, we use the equivalence $\forall p \equiv \neg\exists\neg p$; this may require using the subset construction to convert an NFA to a DFA and then flipping the "finality" of states.
- Ultimately, if all variables are bound, we are left with a single state machine that either accepts (formula is true) or rejects (formula is false)

The bad news

- The worst-case running time of the algorithm above is bounded above by

$$2^{2^{\cdot^{\cdot^{\cdot^{2^{p(N)}}}}}},$$

  where the number of 2's in the exponent is equal to the number of quantifier alternations, p is a polynomial, and N is the number of states needed to describe the underlying automatic sequence.
- The bound for Presburger arithmetic can be improved to double-exponential.

The proof using automata

A couple of additional tricks: if the last quantifiers are $\exists$, all we need to do is check to see if the resulting automaton accepts some word.

In this case, we do not need to convert an NFA to a DFA.

We can check acceptance with depth-first search, by seeing if there is a path in the automaton from the initial state q_0 to a state of F. This can be done in time linear in the size of the automaton.

Similarly, if we want to know if there are infinitely many integers for which some formula holds (which is sometimes written $\exists_\infty$) we just need to check for which states q there is a nonempty cycle beginning and ending at q (which can be done using depth-first search), and then check to see if there is a path from q_0 to q and q to a final state. Again, linear time.

Some subtleties

Every integer has infinitely many representations!

For example, 5 in base 2 can be written as 101, 0101, 00101, and so forth.

It is best to allow all possible representations in our automata.

(If we do not, then we can run into problems working with k-tuples of integers where one integer has a larger representation than other.)

Augmenting Presburger arithmetic

As described, Presburger arithmetic isn't so interesting (although used, e.g., in system verification).

But if we add DFAO's to the mix, using the same decision procedure, we suddenly can prove theorems people actually want to prove.

For example, we can start with a 2-DFAO M for the Thue-Morse sequence $\mathbf{t}$, write a predicate for $\mathbf{t}$ having an overlap, and use the decision procedure to decide it — thus reproving Thue's 1912 result by machine.

But what is the logical theory corresponding to starting with a DFAO?

Büchi's mistake

Büchi was apparently the first to consider this question.

He thought one should add, to Presburger arithmetic, the function $\nu_k(n)$, which is the function computing the *exponent* of the highest power of k dividing n. For example, $\nu_2(24) = 3$.

This was a mistake.

The correct function to add is $V_k(n)$, the function computing the highest power of k, say k^e, dividing n. For example, $V_2(24) = 8$.

Presburger arithmetic augmented

Theorem. A set of integers is definable in $\mathrm{Th}(\mathbb{N}, +, V_k)$ if and only if its characteristic sequence is k-automatic.

Corollary. The theory $\mathrm{Th}(\mathbb{N}, +, V_k)$ is decidable.

Proof. We can decide if a formula in $\mathrm{Th}(\mathbb{N}, +, V_k)$ is true, just as with Presburger arithmetic, by creating the automaton associated with the formula and checking if it accepts.

(See work of Bruyère, Michaux, Villemaire, Hansel, Hodgson, etc.)

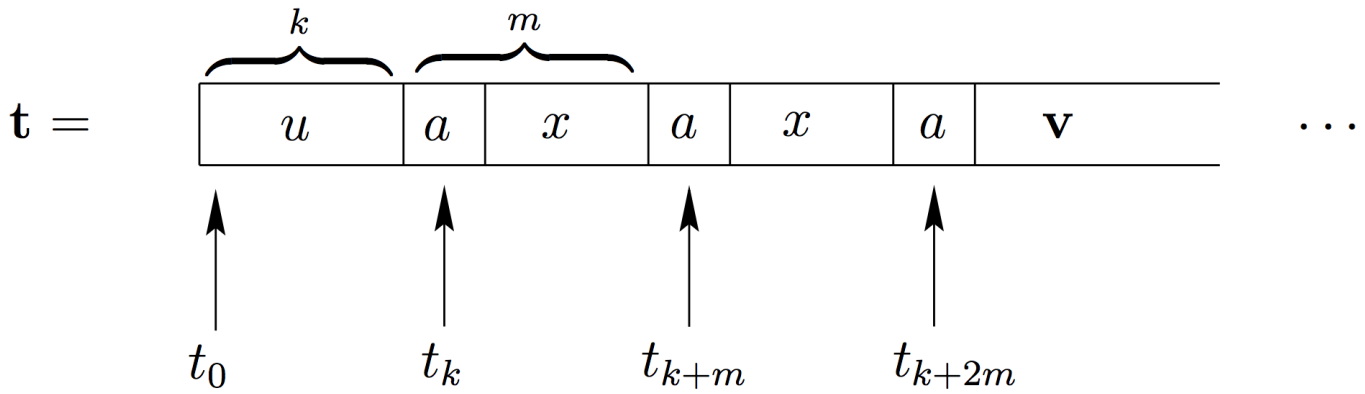
Theorem. There is an algorithm that, given a proposition phrased using only the universal and existential quantifiers, indexing into one or more k-automatic sequences, addition, subtraction, logical operations, and comparisons, will decide the truth of that proposition.

Walnut

- My former student Hamoon Mousavi implemented the decision procedure in Java in freely available software.
- It is downloadable from `https://github.com/hamoonmousavi/Walnut`
- You can
  - define automata as text files
  - enter predicates in terms of these automata
  - if there are no unbound variables, you can evaluate the truth/falsity of these predicates
  - if there are unbound variables, you can obtain automata accepting those $(n)_k$ for which the predicate is true .

An example

Let's write a predicate for the assertion that a sequence $\mathbf{t}$ has an overlap:



So our predicate is

$$\exists k\ \exists m\ (m \geq 1)\ \wedge\ \forall i\ (i \leq m)\implies \mathbf{t}[k+i] = \mathbf{t}[k+i+m].$$

Evaluating the predicate in Walnut

- The predicate was

$$\exists k\ \exists m\ (m \geq 1)\ \wedge\ \forall i\ (i \leq m)\ \Longrightarrow\ \mathbf{t}[k+i] = \mathbf{t}[k+i+m]$$

- To evaluate its truth/falsity we type the following command into Walnut

```
eval t2 "Ek Em (m >= 1) & Ai (i <= m) => T[k+i] = T[k+i+m]":
```

and Walnut returns the value "false".

Complexity

Theorem. If **s** is a k-automatic sequence, then its complexity function $p(n)$ (counting the number of distinct factors of length n in **s**) is k-regular. Furthermore, an explicit representation for p is computable from the automaton for **s**.

The complexity of **t** is well-known.

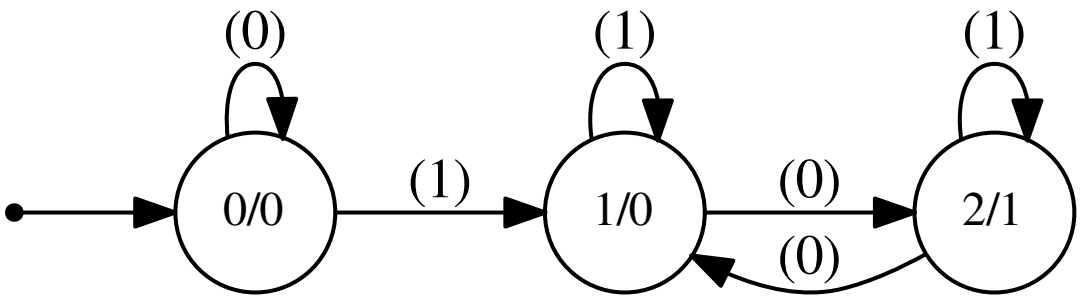
So let's compute it for a lesser-known sequence, the *twisted Thue-Morse sequence*

$$\mathbf{t}' = 0010011010010110011010011001011\cdots$$

that counts the number of occurrences of 0, mod 2, in the binary representation of n.

Determining the complexity of $\mathbf{t}'$

First, we find an automaton for $\mathbf{t}'$:



The first difference of the complexity function $d(n) = p(n+1) - p(n)$ is just the number of length-n factors x for which both $x0$ and $x1$ are factors.

These are the so-called *right-special factors*.

A predicate for right-special factors

We can create a predicate for the assertion that the factor of length n beginning at position i is right-special:

$$\mathrm{spec}(i, n) :=$$

$$(\exists k_0\ (\mathbf{t}'[k_0 + n] = 0)\ \wedge\ (\forall \ell\ (\ell < n)\implies \mathbf{t}'[i + \ell] = \mathbf{t}'[k_0 + \ell]))$$

$$\wedge\ (\exists k_1\ (\mathbf{t}'[k_1 + n] = 1)\ \wedge\ (\forall \ell\ (\ell < n)\implies \mathbf{t}'[i + \ell] = \mathbf{t}'[k_1 + \ell])).$$

Finding first occurrence of a special factor

Next, we modify this modified predicate to include the assertion that the factor of length n beginning at position i is right-special **and** also is the first occurrence of that factor:

$$\mathrm{spec}(i,n) \ \wedge \ \forall k \ (k < i) \implies \exists t \ (t < n) \ \mathbf{t}'[k+t] \neq \mathbf{t}'[i+t].$$

From the automaton A accepting those pairs $(i,n)_2$ for which the predicate is true, we can form the matrices M_0 (resp., M_1) counting the number of transitions from state p to state q for which the second component is labeled 0 (resp., 1).

This gives a linear representation for $d(n)$:

$$vM_{a_1} \cdots M_{a_i} w,$$

where $(n)_2 = a_1 \cdots a_i$.

Linear representation for complexity

Here it is:

$$v = [1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0] \qquad w = [1\ 0\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ 1\ 0\ 0\ 0\ 1]^T$$

$$M_0 = \begin{bmatrix}
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

$$M_1 = \begin{bmatrix}
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

Complexity of $\mathbf{t}'$

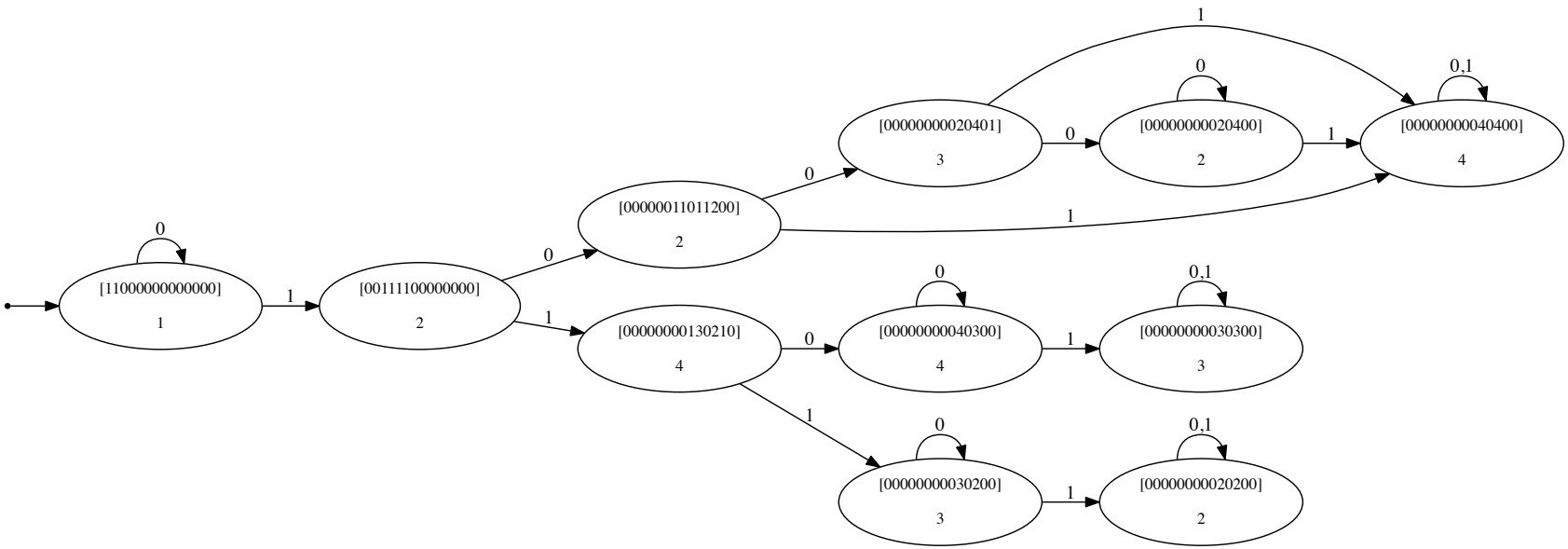
We can obtain an exact description of the first difference of the subword complexity of $\mathbf{t}'$:

Theorem.

$$d(n) = \begin{cases} 1, & \text{if } n = 0; \\ 2, & \text{if } n = 1, 2 \text{ or } 7 \cdot 2^{k-2} < n \leq 2^{k+1} \text{ for } k \geq 2; \\ 3, & \text{if } n = 4 \text{ or } 3 \cdot 2^{k-1} < n \leq 7 \cdot 2^{k-2} \text{ for } k \geq 2; \\ 4, & \text{if } 2^k < n \leq 3 \cdot 2^{k-1} \text{ for } k \geq 1. \end{cases}$$

Proof. Now we can make an automaton out of the states, which are the vectors v dotted with the semigroup S generated by the matrices M_0, M_1. The output of each state is the name of the state dotted with w.
When we do this for $\mathbf{t}'$ we get the following automaton.

Complexity



From this the theorem about complexity follows.

A new complexity result suggested by computation

Arseny Shur and JS just proved:

Theorem. If $\mathbf{s}$ is an overlap-free binary sequence, then

$$p_{\mathbf{t}}(n) \leq p_{\mathbf{s}}(n) \leq p_{\mathbf{t}'}(n)$$

where

- $\mathbf{t}$ is the Thue-Morse sequence; and
- $\mathbf{t}'$ is the twisted Thue-Morse sequence.

Avoiding the pattern xxx^R

- We start by trying depth-first search.
- It gives the lexicographically least such sequence.
- This gives the word

$$(001)^3(10)^\omega = 001001001101010\cdots .$$

- So in particular the word $(10)^\omega = 101010\cdots$ avoids the pattern. (Easy proof!)
- This suggests: are there any other periodic infinite words avoiding xxx^R?
- Also: are there any aperiodic infinite words avoiding xxx^R?

An extended example: avoiding the pattern xxx^R

When we search for other primitive words z such that z^ω avoids the pattern, we find there are some of length 10:

0010011011 0011011001 0100110110 0110010011 0110110010
1001001101 1001101100 1011001001 1100100110 1101100100

- We notice that each of these words is of the form $w\overline{w}$.
- This suggests looking at words of this form.
- The next ones are $w = 00100100110110100100$, and its shifts and complements.

An extended example: avoiding the pattern xxx^R

- To summarize, here are the solutions we've found so far:

w	$\lvert w \rvert$
0	1
00100	5
00100100110110100100	21

- The presence of the numbers 1,5,21 suggests some connection with the Fibonacci numbers.

- They are F_2, F_5, and F_8.

An aperiodic word avoiding xxx^R

- Suppose we take the run-length encodings of the strings of length 21. One of them looks familiar: 212212212212 1. This is a prefix of the infinite Fibonacci word generated by $2 \to 21$, $1 \to 2$.
- This suggests the construction of an *infinite* aperiodic word avoiding xxx^R: take the infinite Fibonacci word, and use it as "repetition factors" for 0 and 1 alternating. This gives the word

$$\mathbf{R} = 0010011011011001001 10 \cdots$$

  which we conjecture avoids xxx^R.
- Can we find an automaton generating this sequence? Yes, but now it is not based on base-2 representations, but rather Fibonacci (or "Zeckendorf") representations.
- "Guess" the automaton and verify it with `Walnut`.

Reproving (and fixing) a result of Fraenkel and Simpson

We turn to a result of Fraenkel and Simpson (1999). They computed the exact number of occurrences of all squares appearing in the finite Fibonacci words X_n.
To solve this using the logical approach, we generalize the problem to consider *any* length-n prefix of $\mathbf{f}$.
The total number of square occurrences in $\mathbf{f}[0..n-1]$:

$$L_{\text{dos}} := \{(n,i,j)_F \ : \ i+2j \leq n \text{ and } \mathbf{f}[i..i+j-1] = \mathbf{f}[i+j..i+2j-1]\}.$$

Let $b(n)$ denote the number of occurrences of squares in $\mathbf{f}[0..n-1]$. First, we use the logical method to find a DFA M accepting L_{dos}. This (incomplete) DFA has 27 states.

Reproving (and fixing) a result of Fraenkel and Simpson

Next, we compute matrices M_0 and M_1 as we did before. We get a linear representation of the sequence $b(n)$:
if $x = a_1 a_2 \cdots a_t$ is the Fibonacci representation of n, then there are matrices M_0, M_1 and vectors v, w such that

$$b(n) = v M_{a_1} \cdots M_{a_t} w^T. \tag{2}$$

Reproving (and fixing) a result of Fraenkel and Simpson

Now let $B(n)$ denote the number of square occurrences in the finite Fibonacci word X_n.
This corresponds to considering the Fibonacci representation of the form 10^{n-1}; that is, $B(n+1) = b([10^n]_F)$.
The matrix M_0 is the following 27×27 array

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

Reproving (and fixing) a result of Fraenkel and Simpson

- M_0 has minimal polynomial

$$X^4(X-1)^2(X+1)^2(X^2-X-1)^2.$$

- It follows from the theory of linear recurrences that there are constants $c_1, c_2, \ldots, c_8$ such that

$$B(n+1) = (c_1 n + c_2)\alpha^n + (c_3 n + c_4)\beta^n + c_5 n + c_6 + (c_7 n + c_8)(-1)^n$$

for $n \geq 3$, where $\alpha = (1+\sqrt{5})/2$, $\beta = (1-\sqrt{5})/2$ are the roots of $X^2 - X - 1$.

- We can find these constants by computing $B(4), B(5), \ldots, B(11)$ and then solving for the values of the constants $c_1, \ldots, c_8$.

Reproving (and fixing) a result of Fraenkel and Simpson

When we do so, we find

$$c_1 = \frac{2}{5} \qquad c_2 = -\frac{2}{25}\sqrt{5} - 2 \qquad c_3 = \frac{2}{5}$$
$$c_4 = \frac{2}{25}\sqrt{5} - 2 \qquad c_5 = 1 \qquad c_6 = 1$$
$$c_7 = 0 \qquad c_8 = 0$$

A little simplification, using the fact that $F_n = (\alpha^n - \beta^n)/(\alpha - \beta)$, leads to

Theorem

Let $B(n)$ denote the number of square occurrences in X_n. Then

$$B(n+1) = \frac{4}{5}nF_{n+1} - \frac{2}{5}(n+6)F_n - 4F_{n-1} + n + 1$$

for $n \geq 3$.

This statement corrects a small error in their paper.

Counting cube occurrences in finite Fibonacci words

In a similar way, we can count the cube occurrences in X_n. Using analysis exactly like the square case, we easily find

Theorem

Let $C(n)$ denote the number of cube occurrences in the Fibonacci word X_n. Then for $n \geq 3$ we have

$$C(n) = (d_1 n + d_2)\alpha^n + (d_3 n + d_4)\beta^n + d_5 n + d_6$$

where

$$d_1 = \frac{3 - \sqrt{5}}{10} \qquad d_2 = \frac{17}{50}\sqrt{5} - \frac{3}{2}$$

$$d_3 = \frac{3 + \sqrt{5}}{10} \qquad d_4 = -\frac{17}{50}\sqrt{5} - \frac{3}{2}$$

$$d_5 = 1 \qquad d_6 = -1.$$

Limits to the approach

- A difficult candidate: abelian properties
- We say that a nonempty word x is an *abelian square* if it is of the form ww' with $|w| = |w'|$ and w' a permutation of w. (An example in English is the word `reappear`.)
- Luke Schaeffer showed that the predicate for abelian squarefreeness is indeed inexpressible in $\mathrm{Th}(\mathbb{N}, +, 0, 1, V_k)$.
- However, for some sequences (e.g., Thue-Morse, Fibonacci) many abelian properties are decidable

Other limits to the approach

- Consider the morphism $a \to abcc$, $b \to bcc$, $c \to c$.
- The fixed point of this morphism is

$$\mathbf{s} = abccbccccbccccccbccccccccb\cdots$$

- It encodes, in the positions of the b's, the characteristic sequence of the squares.
- So the first-order theory $\mathrm{Th}(\mathbb{N}, +, 0, 1, n \to \mathbf{s}[n])$ is powerful enough to express the assertion that "n is a square"
- With that, one can express multiplication, and so the theory is undecidable.

Going even further

- The first-order theory of the paperfolding words (uncountably many!) is decidable.
- Generalizing to "visibly pushdown automata" (in progress)
- Not quite achieved yet (but soon!): the first-order theory of the overlap-free binary words is decidable.
- We hope to show: the first-order theory of the characteristic (and even Sturmian?) words is decidable.