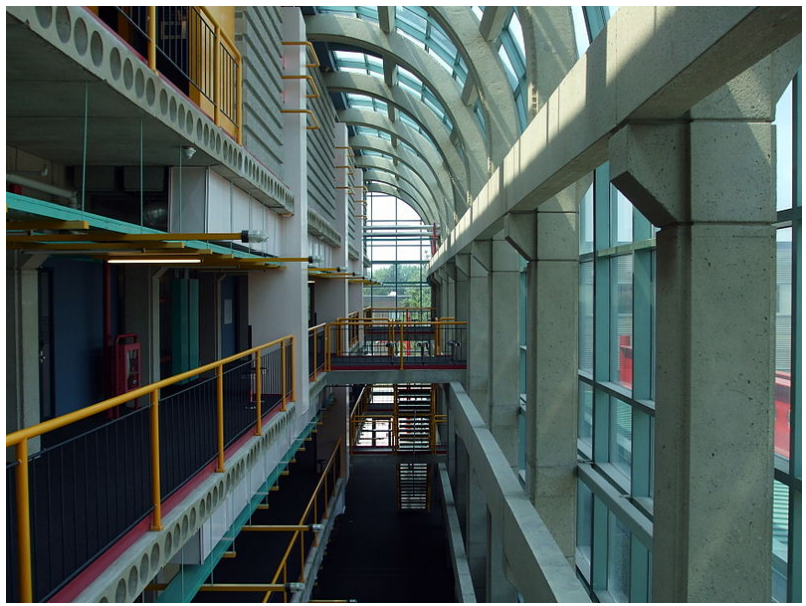
Using Automata and a Decision Procedure to Prove Results in Pattern Matching

Jeffrey Shallit

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



Joint work with Luke Schaeffer, Institute for Quantum Computing, University of Waterloo, Waterloo, ON N2L 3G1, Canada.

The role of famous infinite words

Famous infinite words such as the Thue-Morse word

$$\mathbf{t} = 01101001100101101\cdots$$

and the Fibonacci word

$$\mathbf{f} = 01001010010010100101\cdots$$

have played a role in combinatorial pattern matching from the very beginning of the field.

Example: in the Knuth-Morris-Pratt string-matching algorithm (1977), finite prefixes of the infinite Fibonacci word play a special role as worst-cases of the algorithm.

Excerpt from Knuth-Morris-Pratt, 1977

As a warmup for our theoretical discussion, let us consider the *Fibonacci strings* [14, exercise 1.2.8–36], which turn out to be especially pathological patterns for the above algorithm. The definition of Fibonacci strings is

(1) $$\phi_1 = b, \quad \phi_2 = a; \quad \phi_n = \phi_{n-1}\phi_{n-2} \quad \text{for } n \geqq 3.$$

For example, $\phi_3 = a\,b$, $\phi_4 = a\,b\,a$, $\phi_5 = a\,b\,a\,a\,b$. It follows that the length $|\phi_n|$ is the nth Fibonacci number F_n, and that ϕ_n consists of the first F_n characters of an infinite string ϕ_∞ when $n \geqq 2$.

Consider the pattern ϕ_8, which has the functions $f[j]$ and $next[j]$ shown in Table 1.

TABLE 1

j	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
$pattern[j]$	a	b	a	a	b	a	b	a	a	b	a	a	b	a	b	a	a	b	a	b	a
$f[j]$	0	1	1	2	2	3	4	3	4	5	6	7	5	6	7	8	9	10	11	12	8
$next[j]$	0	1	0	2	1	0	4	0	2	1	0	7	1	0	4	0	2	1	0	12	0

Thue-Morse and Fibonacci in CPM Papers

The Thue-Morse word and Fibonacci word appear, for example, in the following CPM papers:

Karpinski et al., Pattern-matching for strings with short descriptions, CPM 1995
Rytter, Application of Lempel-Ziv factorization..., CPM 2002
Kolpakov and Kucherov, Searching for gapped palindromes, CPM 2008
Amir et al., Quasi-distinct parsing and optimal compression methods, CPM 2009
Xu, A minimal periods algorithm with applications, CPM 2010
Kärkkäinen et al., Linear time Lempel-Ziv factorization, CPM 2013
Belazzougui et al., Composite repetition-aware data structures, CPM 2015
Alamro et al., Computing the antiperiod(s) of a string, CPM 2019
Pape-Lange, On maximal repeats in compressed strings, CPM 2019
👉 Mieno et al., RePair grammars are the smallest grammars..., CPM 2022

Proving properties of words like **f** and **t**: the usual method

In CPM we often want to illustrate our ideas by proving properties of words like **f** and **t** as examples.

Doing so often involves a long case-based argument, often involving induction.

These kinds of long arguments are rarely enlightening and are prone to error.

It would be nice to have a hard-working assistant to help create the proofs for these examples!

A new approach to proving properties of words like **f** and **t**

In this talk I will speak about a method for *mechanically* proving properties of words like **f** and **t**, and their finite prefixes, with applications to combinatorial pattern-matching.

The method itself is not new—the ideas go back to Presburger (1929) and Büchi (1960), and refined more recently by Bruyère et al. (1994)—but modern computers make it possible to actually implement and run the method on many examples.

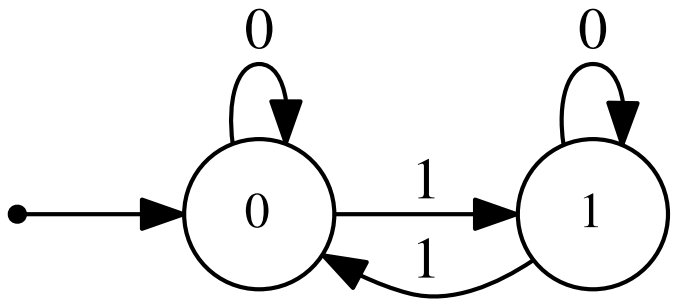
The method involves a decision procedure—an algorithm—that takes an assertion φ phrased in a logical language, and produces a rigorous proof or disproof of the statement.

The ideas are applicable to a class of sequences called *automatic*.

What are automatic sequences?

A sequence $\mathbf{x} = (a_n)_{n \geq 0}$ is *automatic* if there exists a DFAO (deterministic finite automaton with output) that, given a representation of n in a suitable numeration system as input, outputs a_n as a function of the last state reached.

Example #1: The Thue-Morse sequence $\mathbf{t}$. Here the numeration system is base 2, and the automaton is as follows:



Here the output is the name of the state.

What are automatic sequences?

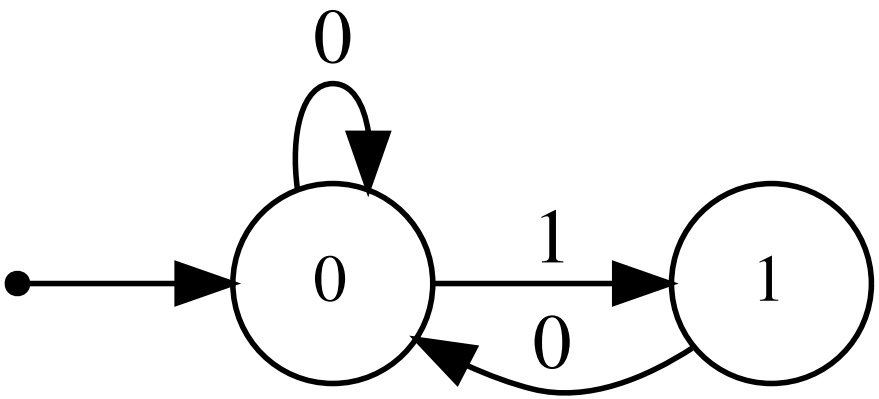
Example #2: The Fibonacci sequence **f**. Here we have to use an "exotic" numeration system, the so-called Zeckendorf representation.

In this system we represent a natural number n as a sum of Fibonacci numbers with coefficients $0, 1$, namely

$$n = \sum_{2 \le i \le t} e_i F_i,$$

subject to the requirement that $e_i e_{i+1} \neq 1$.

Then **f** is generated by the following automaton:



The main ideas

The first main theorem:

Theorem. *There is a decision procedure that, given an automatic sequence* **x** *and* φ*, a first-order logical statement about* **x***, with no free variables, over the natural numbers with addition, will terminate and answer* `TRUE` *if* φ *holds and* `FALSE` *otherwise.*

free = not bound to any quantifier

allowed operations: the usual logical operations, comparisons of integers, addition, subtraction, indexing into **x**, multiplication and integer division by constants.

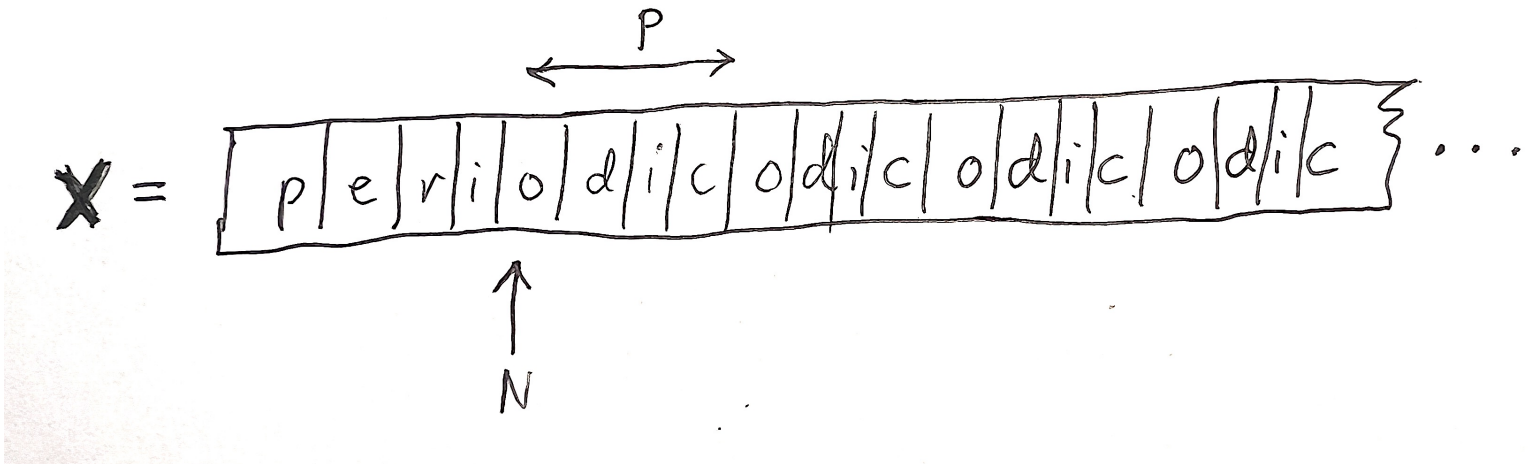
not allowed: multiplication and division of two variables, recursive definitions, counting.

Some examples of what can be decided

Examples of the kinds of things we can phrase in first-order logic:

x *is eventually periodic:*

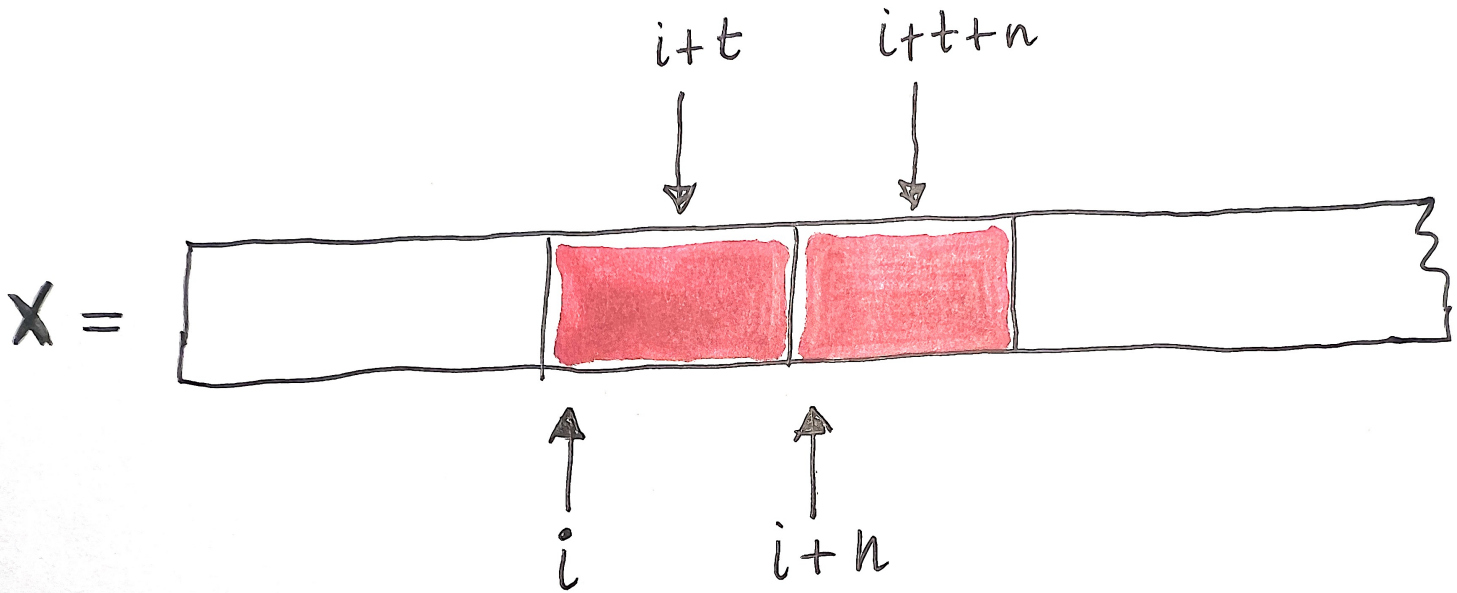
$$\exists N, p\ (p \geq 1) \wedge \forall i\ (i \geq N) \implies \mathbf{x}[i] = \mathbf{x}[i+p].$$



Some examples

x *contains a square (aka repetition, aka tandem repeat—a nonempty block of the form yy):*

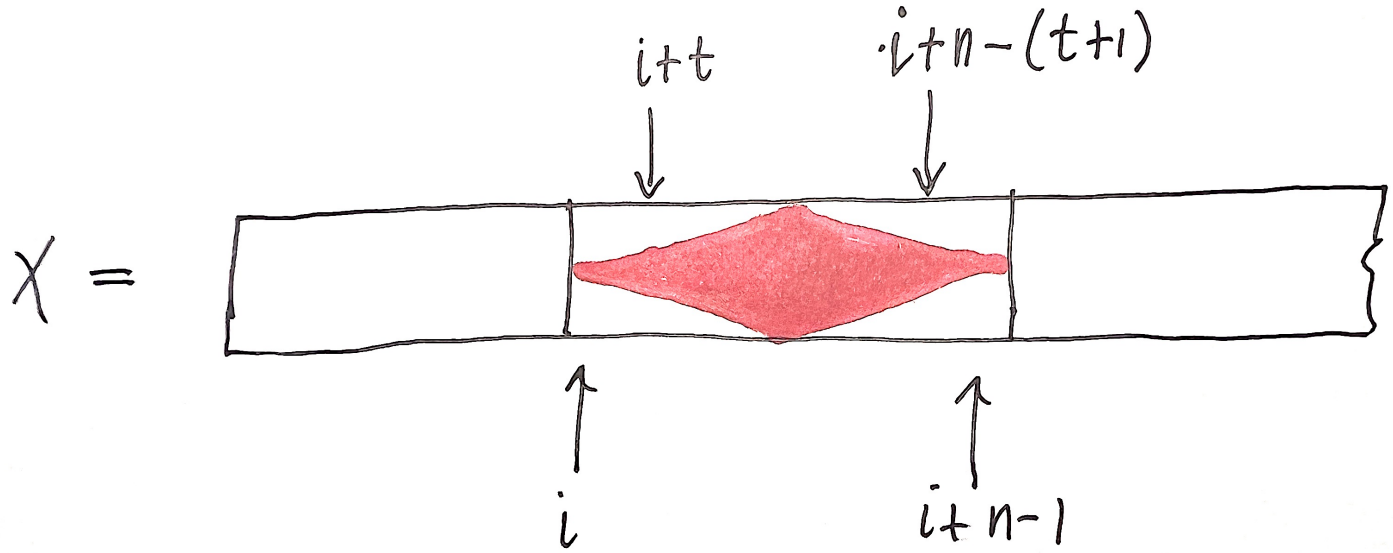
$$\exists i, n\ (n \geq 1) \wedge \forall t\ (t < n) \implies \mathbf{x}[i + t] = \mathbf{x}[i + t + n].$$



Some examples

x *contains arbitrarily large palindromes:*

$$\forall M\ \exists i, n\ (n \geq M) \wedge \forall t\ (t < n) \implies \mathbf{x}[i+t] = \mathbf{x}[(i+n)-(t+1)].$$



Alternate formulas

This last formula for palindromes could also have been expressed

$$\forall M\ \exists i, n\ (n \geq M) \wedge \forall t\ (t < n/2) \implies \mathbf{x}[i+t] = \mathbf{x}[(i+n)-(t+1)].$$

- There are often many different ways to express the same assertion.
- Some may run more quickly than others in the decision procedure.
- It is hard to determine ahead of time which query is likely to run fastest.

The second main result

The second main theorem:

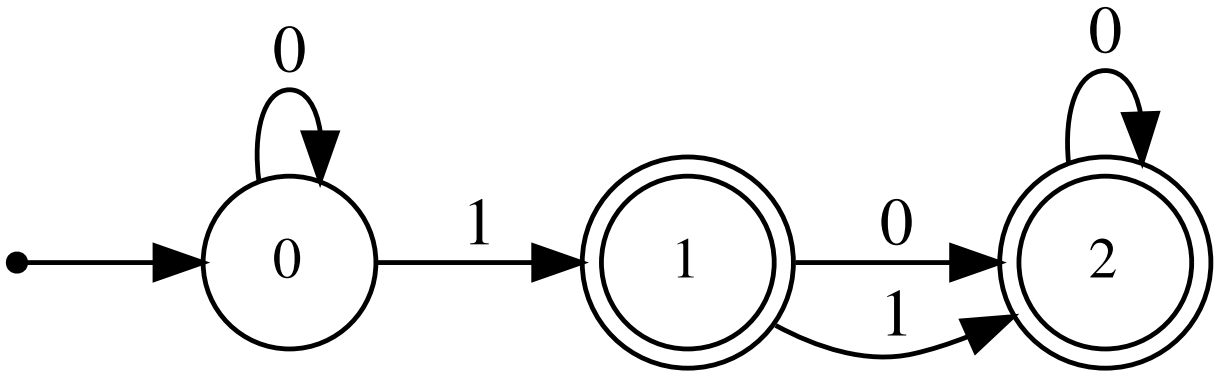
Theorem. *There is a procedure that, given an automatic sequence* $\mathbf{x}$ *and first-order logical statement* φ *about* $\mathbf{x}$ *with some free variables* $z_1, \ldots, z_n$*, over the natural numbers with addition, produces a finite automaton that accepts, in parallel, the values of the free variables* $z_1, \ldots, z_n$ *that make the formula* φ *true.*

An example of the second theorem

Orders of squares in the Thue-Morse sequence:

$$(n \geq 1) \wedge \exists i \; \forall t \; (t < n) \implies \mathbf{x}[i+t] = \mathbf{x}[i+t+n].$$

The method produces the following automaton:



Notice that it accepts the base-2 representation of 2^i for $i \geq 0$ and $3 \cdot 2^i$ for $i \geq 0$.

How the two theorems are proved

Main ideas:

Compile the first-order logic statement into a series of transformations on automata.

Start with the automaton for the sequence **x** itself; use basic theorems of automata theory to carry out "and" (intersection), "or" (union), etc.

An "adder automaton" is needed to check the relation $x + y = z$.

How the two theorems are proved

The existential quantifier $\exists$ is carried out by projection of transitions, followed by determinization and minimization.

Projection can cause the automaton to become nondeterministic.

Such an automaton needs to be determinized, which can blow up the number of states exponentially.

The universal quantifier $\forall$ is handled by $\forall x\ p(x)$ replaced by $\neg\exists x\ \neg p(x)$.

From infinite words to finite words

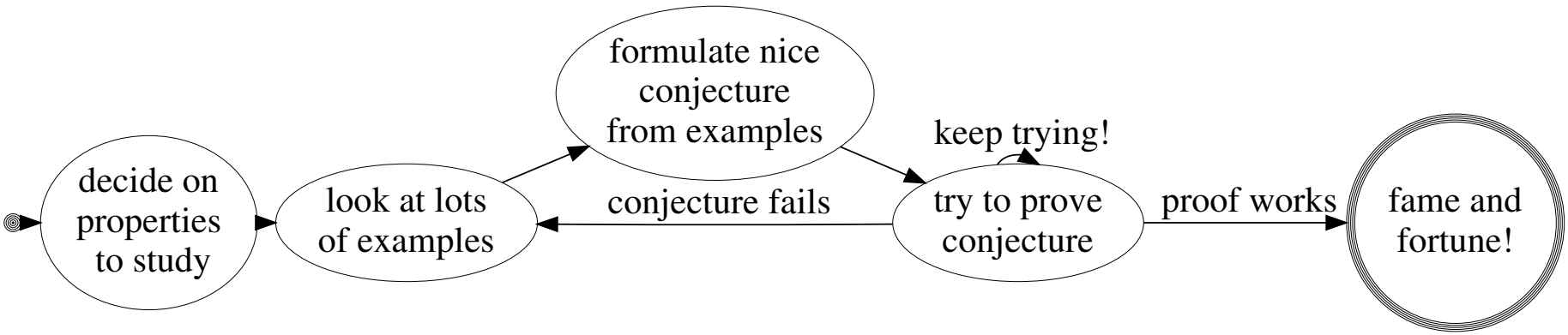
Most theorems in combinatorial pattern-matching concern finite strings, not infinite ones.

Typically in CPM one is interested in special prefixes of Thue-Morse (of length 2^n) or Fibonacci (of length F_n).

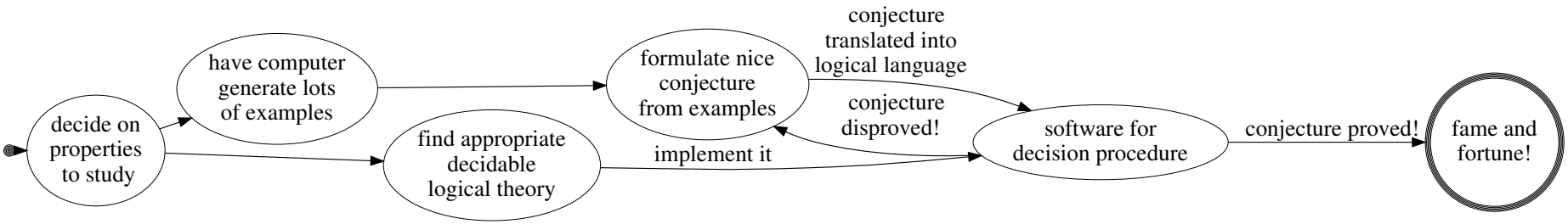
We can use the logical approach to prove results about finite strings if they are (i) prefixes of an appropriate automatic sequence or (ii) factors (contiguous substrings) of an appropriate automatic sequence or (iii) simple modifications of these.

In particular, we can prove results for *all* prefixes of **t** and **f**, not just those of special lengths.

The old paradigm for research in discrete mathematics



A new paradigm for research in discrete mathematics



String Attractors

Recently, the Thue-Morse word played a role as an example in the study of "string attractors".

Definition. (Kempa & Prezza, 2018) Let $w = w[0..n-1]$ be a finite word. A *string attractor* of w is a subset $S \subseteq \{0, 1, \ldots, n-1\}$ such that every nonempty factor f of w has an occurrence in w that touches one of the indices of S.

For example, $\{2, 3, 4\}$ is a string attractor for the word $w = \texttt{ALFALFA}$, and there is no smaller string attractor for w.

A	L	F	A	L	F	A
0	1	2	3	4	5	6

String attractors and automatic sequences

Applying the logical decision procedure to string attractors gives us the following results:

Theorem.
(a) It is decidable, given an automatic sequence $\mathbf{x}$ *and a constant* c*, whether all prefixes of* $\mathbf{x}$ *have a string attractor of size at most* c*.*

(b) Furthermore, if this is the case, we can construct a finite automaton that, for each n*, provides the lexicographically least, minimal string attractor for the length-*n *prefix of* $\mathbf{x}$*.*

Proof. It suffices to provide first-order logical formulas for (a) and (b).

A logical formula for string attractors

Let us create a first-order formula φ asserting that the length-n prefix of $\mathbf{x}$ has a string attractor of size at most c.

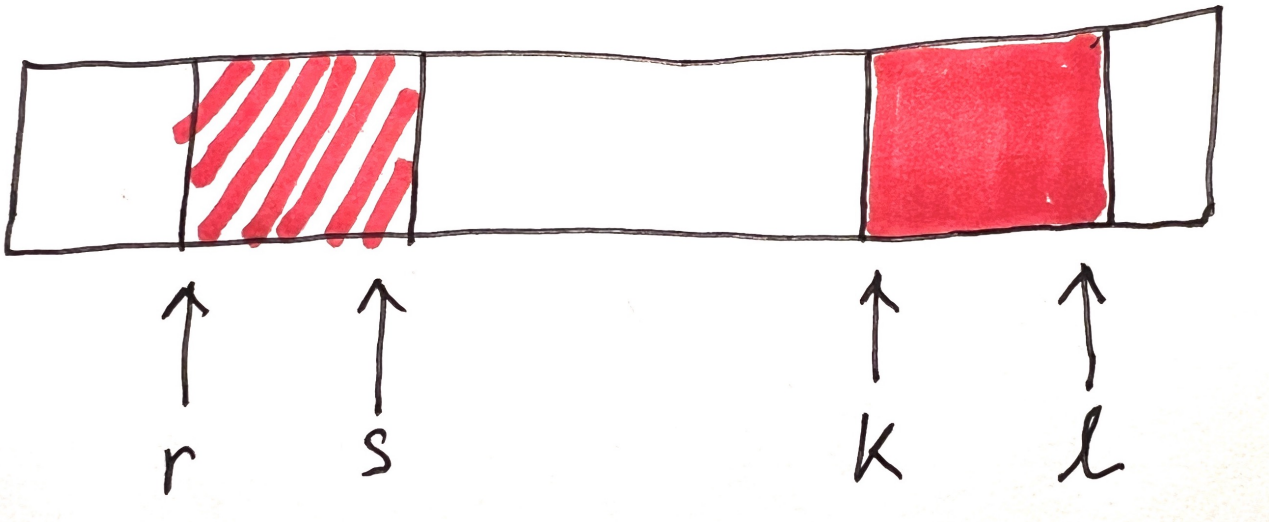
We construct φ in several stages.

First, a formula comparing two different factors of $\mathbf{x}$.

Next, a formula asserting that position i is touched by some occurrence of a given factor.

Finally, a formula asserting that all factors are touched by some i in the string attractor set S.

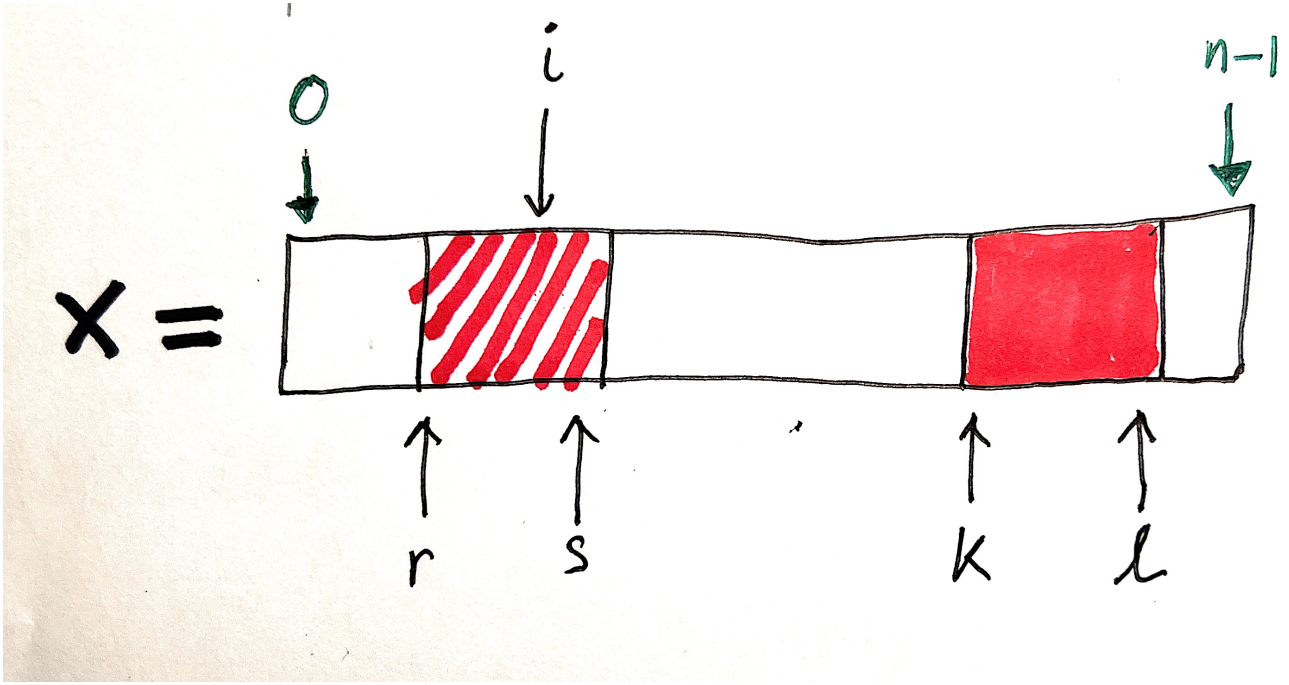
A logical formula for string attractors



First, we need a formula asserting that the factors $\mathbf{x}[k..\ell]$ and $\mathbf{x}[r..s]$ coincide. We can do this as follows:

$$\begin{aligned}\mathsf{faceq}(k,\ell,r,s) := (\ell - k = s - r) \wedge \\ \forall j\ (r + j \leq s) \implies \mathbf{x}[r+j] = \mathbf{x}[k+j].\end{aligned}$$

A logical formula for string attractors



Next, we need a formula asserting that the factor $\mathbf{x}[k..\ell]$ has some occurrence $\mathbf{x}[r..s]$ in the prefix $\mathbf{x}[0..n-1]$ that touches position i:

$$\text{stringa}(i,k,\ell,n) := \exists r,s\ (s<n) \wedge \text{faceq}(k,\ell,r,s) \wedge r \leq i \wedge i \leq s.$$

A logical formula for string attractors

Next, let us create a set of formulas, indexed by c, asserting that $i_1 \leq i_2 \leq i_3 \leq \cdots \leq i_c < n$ is a string attractor for the length-n prefix of $\mathbf{x}$. We can do this as follows:

$$\begin{aligned}\mathsf{sa}_c(i_1, i_2, \ldots, i_c, n) := {} & (i_1 \leq i_2) \wedge (i_2 \leq i_3) \wedge \cdots \wedge (i_{c-1} \leq i_c) \wedge (i_c < n) \wedge \\ & \forall k, \ell \; (k \leq \ell \wedge \ell < n) \implies \\ & (\mathsf{stringa}(i_1, k, \ell, n) \vee \mathsf{stringa}(i_2, k, \ell, n) \vee \cdots \vee \mathsf{stringa}(i_c, k, \ell, n)).\end{aligned}$$

Notice here that we do not demand that the i_j be distinct, which explains why this formula is designed to check that the string attractor size is $\leq c$, rather than just equal to c.

A logical formula for string attractors

Finally, we can create a formula with no free variables asserting that every nonempty prefix of $\mathbf{x}$ has a string attractor of cardinality $\leq c$ as follows:

$$\forall n\ (n \geq 1) \implies \exists i_1, i_2, \ldots, i_c\ \ \mathsf{sa}_c(i_1, i_2, \ldots, i_c, n).$$

Suppose we have found a c such that there exists a string attractor of size c. By checking successively the truth of the formulas

$$\begin{aligned}
\forall n\ (n \geq 1) &\implies \exists i_1\ \ \mathsf{sa}_1(i_1, n)\\
\forall n\ (n \geq 1) &\implies \exists i_1, i_2\ \ \mathsf{sa}_2(i_1, i_2, n)\\
&\ \ \vdots\\
\forall n\ (n \geq 1) &\implies \exists i_1, i_2, \ldots, i_c\ \ \mathsf{sa}_c(i_1, i_2, \ldots, i_c, n)
\end{aligned}$$

we can find the minimum such $c = c_0$.

A logical formula for string attractors

In order to actually find the string attractors, we create a formula asserting that $i_1 \leq i_2 \leq \cdots \leq i_c$ is the lexicographically least tuple that is a string attractor for $\mathbf{x}[0..n-1]$:

$$
\begin{aligned}
&\mathsf{lexleast}(i_1, i_2, \ldots, i_c, n) := \mathsf{sa}_c(i_1, i_2, \ldots, i_c, n) \wedge \\
&\quad \forall j_1, j_2, \ldots, j_c \ (\mathsf{sa}_c(j_1, j_2, \ldots, j_c, n) \implies \\
&\quad ((i_c \leq j_c) \wedge \\
&\quad ((i_c = j_c) \implies (i_{c-1} \leq j_{c-1})) \wedge \\
&\quad ((i_{c-1} = j_{c-1} \wedge i_c = j_c) \implies (i_{c-2} \leq j_{c-2})) \wedge \\
&\quad \cdots \wedge \\
&\quad ((i_2 = j_2 \wedge i_3 = j_3 \wedge \cdots \wedge i_c = j_c) \implies i_1 \leq j_1))).
\end{aligned}
$$

Walnut

Walnut is a free software package for deciding the truth of logical assertions about automatic sequences and for creating the automaton accepting satisfying values of the free variables (if there are any).

It is available at https://cs.uwaterloo.ca/~shallit/walnut.html .

When Walnut halts and produces a result TRUE or FALSE, then the answer is *guaranteed* to be correct.

Queries are given in first-order logic, basically the same as given previously, with some translations.

Walnut syntax

Translation:

$\forall$ is represented by `A`

$\exists$ is represented by `E`

$\Longrightarrow$ is represented by `=>`

$\wedge$ is represented by `&`

$\vee$ is represented by `|`

$\neg$ is represented by `~`

`eval` – evaluates a formula as TRUE/FALSE

`def` – defines an automaton that can be reused

`T` – the Thue-Morse sequence **t**

`F` – the Fibonacci sequence **f**

Examples of the two theorems in Walnut

Example: does Thue-Morse contain arbitrarily large palindromes?

logical formula:

$$\forall M\ \exists i, n\ (n \geq M) \wedge \forall t\ (t < n) \implies \mathbf{x}[i+t] = \mathbf{x}[(i+n)-(t+1)].$$

translation into Walnut:

```
eval tmpal "Am Ei,n n>=m & At (t<n) =>
   T[i+t]=T[(i+n)-(t+1)]":
```

Returns TRUE.

Examples of the two theorems in Walnut

Example: what are the orders of squares in the Fibonacci sequence?

Logical formula:

$$(n \geq 1) \wedge \exists i \; \forall t \; (t < n) \implies \mathbf{f}[i+t] = \mathbf{f}[i+t+n].$$

translation into Walnut:

```
def fibsquares "?msd_fib (n>=1) &
   Ei At (t<n) => F[i+t]=F[i+t+n]":
```

Walnut returns a 2-state automaton accepting the orders of squares, expressed in the Zeckendorf numeration system: namely, it accepts 10^*, so the orders of squares are F_n for $n \geq 2$.

Example: the period-doubling sequence

Now let's attack the string attractor problem.

The so-called *period-doubling sequence* is an infinite binary sequence

$$\mathbf{pd} = 10111010101110111\cdots$$

that is the fixed point of the morphism

$$1 \to 10, \qquad 0 \to 11.$$

It can be obtained from the Thue-Morse sequence by taking the xor (or sum (mod 2)) of a window of size 2 moving through **t**:

$$0110100110010110 \to 10111010101110111\cdots.$$

Let us use `Walnut` to determine the string attractors for all prefixes of **pd**.

Determining the string attractors for the period-doubling sequence with Walnut

```
def faceq "s+k=r+l & Aj (r+j<=s) => PD[r+j]=PD[k+j]"
# 19 states, 433 ms

def stringa "Er,s (s<n) & $faceq2(k,l,r,s) & r<=i & i<=s":
# 57 states, 82 ms

def sa1 "i1<n & Ak,l (k<=l & l<n) => $stringa(i1,k,l,n)":
# 2 states, 104 ms

eval test1 "An (n>=1) => Ei1 $sa1(i1,n)":
# 273 ms, returns FALSE,
# so no string attractor of size 1 (of course)
```

Determining the string attractors for the period-doubling sequence with Walnut

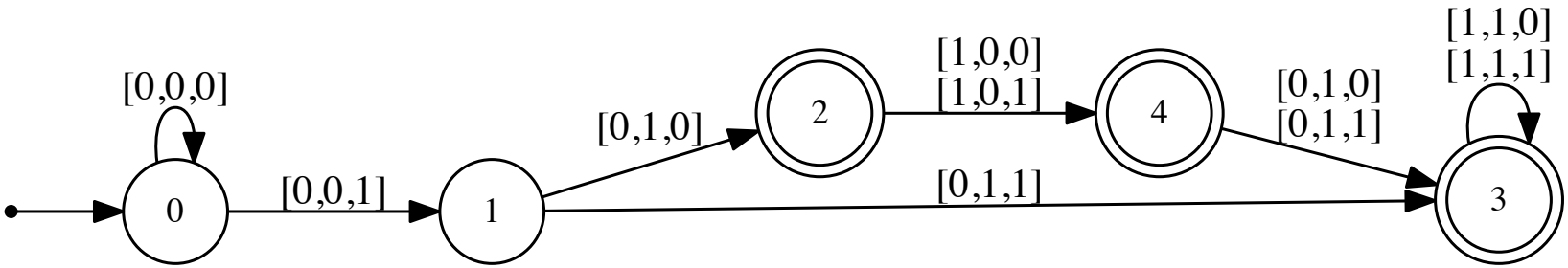
```
def sa2 "i1<=i2 & i2<n & Ak,l (k<=l & l<n) =>
   ($stringa(i1,k,l,n)|$stringa(i2,k,l,n))":
# 14 states, 3971372ms
# involved minimizing a DFA of 6091705 states

eval test2 "An (n>=1) => Ei1,i2 $sa2(i1,i2,n)":
# 1 ms, returns TRUE

eval lexleast "$sa2(i1,i2,n) & Aj1,j2 $sa2(j1,j2,n) =>
   (i2<=j2 & (i2=j2 => i1<=j1))":
# 5 states, 9 ms
```

Determining the string attractors for the period-doubling sequence with Walnut

The `lexleast` command gives the following automaton:



For example, let's find the lexicographically least string attractor for the prefix of length $n = 8$.

An accepting path is labeled

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

corresponding to the triple $(i_1, i_2, n) = (2, 5, 8)$.

Determining the string attractors for the period-doubling sequence with Walnut

An accepting path is labeled

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

corresponding to the triple $(i_1, i_2, n) = (2, 5, 8)$ and the string attractor $\{2, 5\}$:

1	0	$\dot{1}$	1	1	$\dot{0}$	1	0
0	1	2	3	4	5	6	7

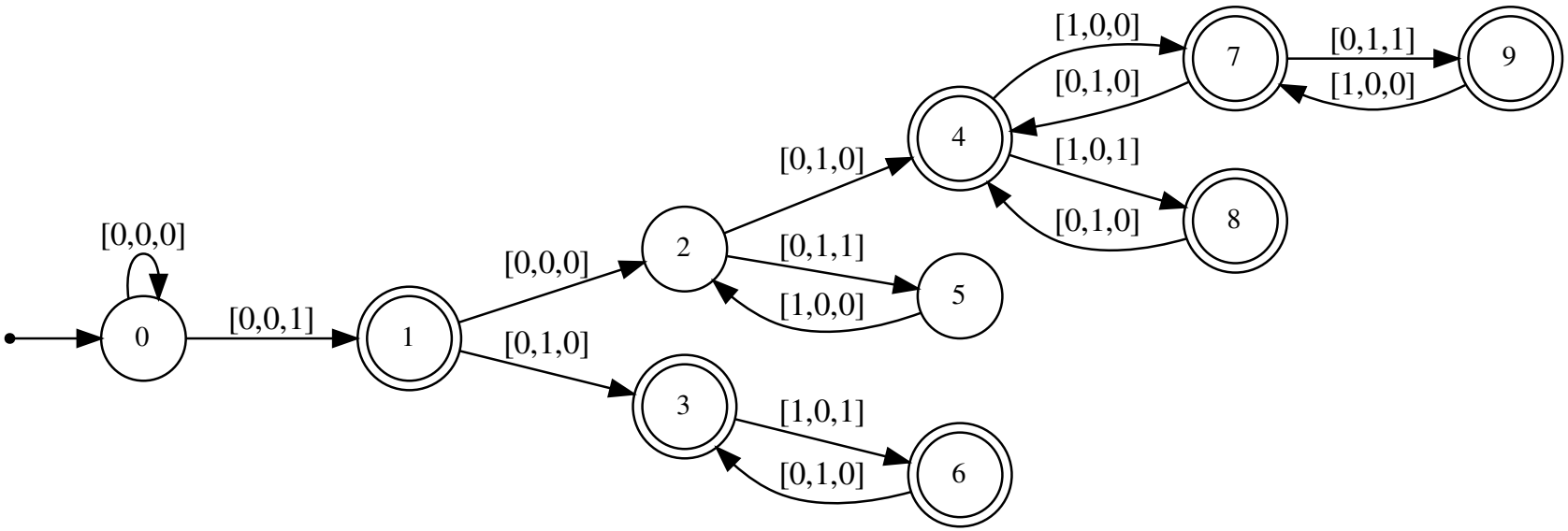
Determining the string attractors for the period-doubling sequence with `Walnut`

From inspection of the automaton we can obtain an explicit formula for the string attractors of **pd**.

Theorem. *Let $n \geq 6$. Then a string attractor of minimum size for* $\mathbf{pd}[0..n-1]$ *is given by*

$$\begin{cases} \{3 \cdot 2^{i-3} - 1, \quad 3 \cdot 2^{i-2} - 1\}, & \text{if } 2^i \leq n < 3 \cdot 2^i; \\ \{2^i - 1, \quad 2^{i+1} - 1\}, & \text{if } 3 \cdot 2^i \leq n < 2^{i+1}. \end{cases}$$

Determining the string attractors for the infinite Fibonacci word with `Walnut`



This was a big calculation for `Walnut`, using 12229013 ms and 356 Gigs of storage. It involved minimizing a DFA of 41532816 states.

Bad news and good news

We have seen how powerful this logical approach is.

However, it's now time for the bad news: the worst-case running time of this method is enormous.

It is

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

where the number of 2's in the exponent is equal to the number of quantifier alternations, p is a polynomial in the length of the particular statement being decided, and N is the number of automaton states needed to describe the underlying sequence.

Good news: even so, we have been successful on something like 90% of the queries we've tried, even with as many as 6 quantifier alternations.

Thue-Morse

When we try to apply the logical method directly to the Thue-Morse sequence, we run up against exponential blowup, and `Walnut` doesn't terminate in a reasonable length of time.

Nevertheless, we can still use it!

The idea is to *guess* what a string attractor would look like (by looking at examples) and then use `Walnut` to *verify* the guess.

Thue-Morse

With this, we can prove the following result:

Theorem.

(a) $\{2^i - 1, \quad 3 \cdot 2^{i-1} - 1, \quad 2^{i+1} - 1, \quad 3 \cdot 2^i - 1\}$ *is a string attractor for* $\mathbf{t}[0..n-1]$ *for* $13 \cdot 2^{i-2} - 1 \leq n \leq 5 \cdot 2^i$ *and* $i \geq 2$;

(b) $\{3 \cdot 2^{i-1} - 1, \quad 2^{i+1} - 1, \quad 3 \cdot 2^i - 1, \quad 2^{i+2} - 1\}$ *is a string attractor for* $\mathbf{t}[0..n-1]$ *for* $9 \cdot 2^{i-1} - 1 \leq n \leq 3 \cdot 2^{i+1}$ *and* $i \geq 1$;

(c) $\{3 \cdot 2^{i-1} - 1, \quad 2^{i+1} - 1, \quad 2^{i+2} - 1, \quad 5 \cdot 2^i - 1\}$ *is a string attractor for* $\mathbf{t}[0..n-1]$ *for* $3 \cdot 2^{i+1} - 1 \leq n \leq 13 \cdot 2^{i-1}$ *and* $i \geq 1$.

Corollary. $\mathbf{t}[0..n-1]$ *has a string attractor of size* 4 *for* $n \geq 25$.

(Previously only known for n a power of 2.)

Going further: span

The advantage to this formulation of string attractors is that it is easy to explore other aspects, simply by defining new formulas.

For example, Marinella Sciortino defined the notion of *span* of a string attractor: it is the difference between the smallest and largest element of a string attractor of minimum size for n. We can then investigate the minimum and maximum possible span of string attractors for $\mathbf{x}[0..n-1]$.

We can create formulas for these as follows:

$$\begin{aligned}
\mathsf{span}(n, r) &:= \exists i_1, i_2, \ldots, i_c \ \mathsf{sa}(i_1, i_2, \ldots, i_c, n) \wedge i_c = i_1 + r \\
\mathsf{minspan}(n, r) &:= \mathsf{span}(n, r) \wedge \forall s \ (s < r) \implies \neg \mathsf{span}(n, s) \\
\mathsf{maxspan}(n, r) &:= \mathsf{span}(n, r) \wedge \forall s \ (s > r) \implies \neg \mathsf{span}(n, s)
\end{aligned}$$

Going further: span

Using `Walnut` we can show:

Theorem. *For the period-doubling sequence the* minspan *of the length-n prefix equals*

$$\begin{cases} 0, & \text{for } n = 1; \\ 1, & \text{for } 2 < n < 5; \\ 2^i, & \text{for } 3 \cdot 2^i \leq n < 3 \cdot 2^{i+1} \text{ and } i \geq 1 \ . \end{cases}$$

The maxspan *of the length-n prefix equals*

$$\begin{cases} 0, & \text{for } n = 1; \\ 1, & \text{for } 2 \leq n \leq 3; \\ 2^i, & \text{if } 5 \cdot 2^{i-1} - 1 \leq n < 6 \cdot 2^{i-1} - 2 \text{ and } i \geq 1; \\ 3 \cdot 2^i, & \text{if } 6 \cdot 2^i - 1 \leq n \leq 5 \cdot 2^{i+1} - 2 \text{ and } i \geq 0. \end{cases}$$

Other kinds of properties

`Walnut` can be used to check dozens of other combinatorial properties:

- *recurrence:* whether a given block occurs finitely many times or infinitely often
- *recurrence constant:* minimum and maximum distance separating two consecutive occurrences of a length-n factor
- *critical exponent:* largest degree of repetition of a factor
- and many many others...

Enumeration

`Walnut` can also be used to *enumerate* various kinds of properties:

- Count the number of distinct factors of length n (aka *subword complexity*)
- Count the number of squares (or cubes or palindromes) in a prefix of length n
- and many others...

It produces a *linear representation* for the enumeration function.

Conclusions

- The decision algorithm implemented in `Walnut` can often "automatically" prove interesting and novel combinatorial claims about famous sequences like Thue-Morse or Fibonacci
- `Walnut` now used in more than 50 papers on combinatorics on words, often to prove entirely new results, sharpen existing results, and sometimes to correct incorrect published results.
- An example of a false claim in the literature found and corrected by `Walnut`: "Every length k factor of Thue-Morse appears as a factor of every length $8k - 1$ factor of Thue-Morse."

Conclusions

- `Walnut` removes the drudgery of long case-based proofs
- Although the worst-case running time is astronomical, in many (most?) cases it runs rather quickly.
- But there are limitations on what sequences can be investigated and what kinds of properties can be proved.
- For a complete list of papers using it and to download `Walnut`, see `https://cs.uwaterloo.ca/~shallit/walnut.html`

For further reading

- L. Schaeffer and J. Shallit, Arxiv preprint 2012.06840 [cs.FL], January 7 2021. Available at https://arxiv.org/abs/2012.06840.
- S. Mantaci, A. Restivo, G. Romana, G. Rosone, and M. Sciortino, A combinatorial view on string attractors, *Theoret. Comput. Sci.* **850** (2021), 236–248.
- K. Kutsukake, T. Matsumoto, Y. Nakashima, S. Inenaga, H. Bannai, and M. Takeda. On repetitiveness measures of Thue-Morse words. In C. Boucher and S. V. Thankachan, editors, *SPIRE 2020*, Vol. 12303 of Lecture Notes in Computer Science, pp. 213–220. Springer-Verlag, 2020.

For further reading

Coming soon
to a fine bookstore
near you!