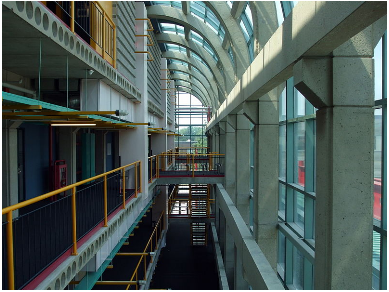
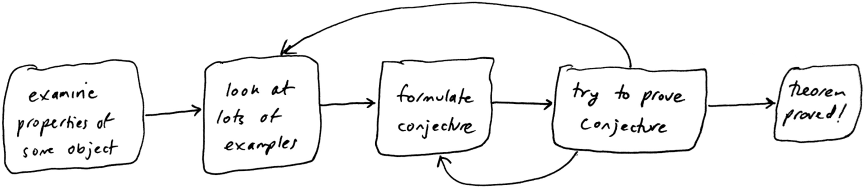
Proving Results in Combinatorics on Words and Number Theory using a Decidable Logic Theory

Jeffrey Shallit

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



The familiar research methodology for mathematics



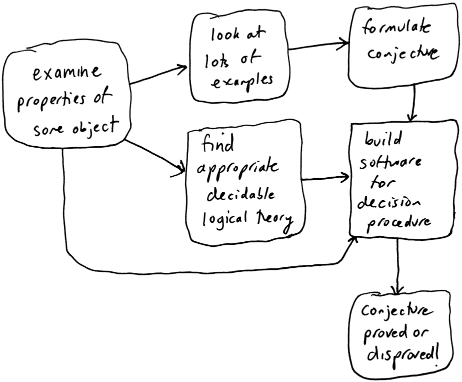
Advantages:

- it's familiar
- it's worked for centuries

Disadvantages:

- need to be clever
- might be preposterously difficult if sought description is complicated

A new (?) research methodology for mathematics



Advantages:

- don't need to be very clever
- sometimes it automatically generates the conjecture for you

Disadvantages:

- decision procedures don't exist for most of mathematics
- sometimes they take ridiculous amounts of space and time

Examples of this new approach

- The Wilf-Zeilberger (WZ) approach to automatically prove combinatorial identities, such as

$$\sum_{-n\leq k\leq n} (-1)^k \binom{2n}{n+k}^3 = \frac{(3n)!}{n!^3}.$$

- Use of SAT solvers (e.g., the recent solution of the Boolean pythagorean triples problem: is there a 2-coloring of the positive integers so that there is no monochromatic triple (a, b, c) with $a^2 + b^2 = c^2$?)
- Proof assistants like Isabelle and Coq

Hilbert's dreams

David Hilbert
(1862–1943)
German mathematician.

- To show that every true statement is provable (killed by Gödel)
- To provide an algorithm to decide if a given statement is provable (killed by Turing)
- Nevertheless, some *subclasses* of problems are decidable — i.e., an algorithm exists guaranteed to prove or disprove any statement in the class

First-order logic

- Let $\langle \mathbb{N}, + \rangle$ denote the set of all first-order logical formulas in the natural numbers with addition.
- Here we are allowed to use any number of variables, logical connectives like "and", "or", "not", etc., addition of natural numbers, comparison of natural numbers, and quantifiers like "there exists" ($\exists$) and "for all" ($\forall$).
- This is sometimes called *Presburger arithmetic*.
- Example: $\forall x, y \; x + y = y + x$. What does this assert?

Another example: the Chicken McNuggets problem

A famous problem in elementary arithmetic books:

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:

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

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

Presburger's theorem

Mojżesz Presburger
(1904–1943)
Murdered by the Nazis.

Presburger proved that $\mathrm{FO}(\mathbb{N}, +)$ is *decidable*: that is, there exists an algorithm that, given a sentence in $\langle \mathbb{N}, + \rangle$ with no free variables, will decide its truth.

He used quantifier elimination.

His master's thesis was one of the most influential of all time in mathematics.

Büchi's proof of Presburger's theorem

Julius Richard Büchi
(1924–1984)
Swiss logician

Büchi found a completely different proof of Presburger's theorem.

Numbers are represented in base-k for some integer $k \geq 2$.

And logical formulas are implemented by means of *finite automata*.

What are finite automata?

Finite automata are a model of a very simple kind of computing machine, having only finite memory (called the "states").

Their inputs are strings of symbols ("words") chosen from a finite alphabet Σ.

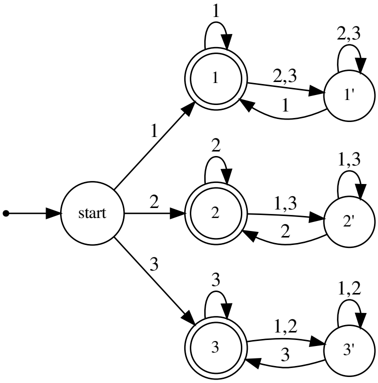
As each letter is processed from left to right, the automaton looks up in a table (the "transition function") which state to go to, based on the current state and current input letter.

Certain states are called "final". If, after reading the entire input, the automaton ends in a final state, the input is *accepted*; otherwise it is *rejected*. This is the basic automaton model.

In a variation of the model, we associate an output letter with each state, and then the output corresponding to an input is the output associated with the last state reached.

Example of an automaton

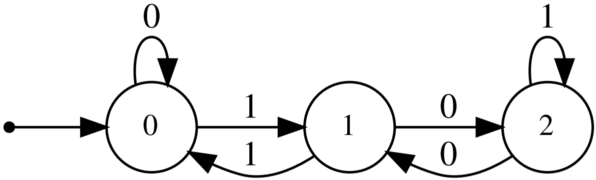
Here is an automaton accepting those words over $\{1, 2, 3\}$ whose first symbol is the same as the last symbol:



Double circle: indicates final state.

Second example of an automaton

This automaton computes n modulo 3, where n is expressed in base 2:



Here the output associated with each state is the number of that state. The meaning of state i is "number represented in binary by the word seen so far is congruent to i (mod 3)".

Decidability of Presburger arithmetic: Büchi's proof

Ideas:

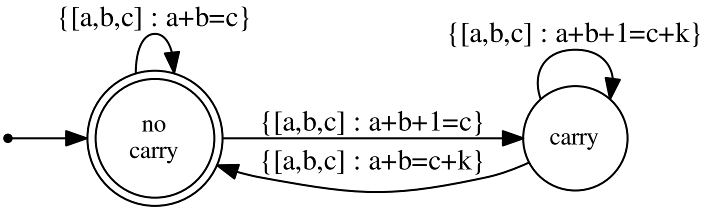
- represent integers in an integer base $k \geq 2$ using the alphabet $\Sigma_k = \{0, 1, \ldots, k-1\}$, most-significant-digit first.
- represent t-tuples of integers as words over the alphabet Σ_k^t, padding with leading zeroes, if necessary. This corresponds to reading the base-k representations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

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

  - First component spells out 10101, which is 21 in base 2
  - Second component spells out 00111, which is 7 in base 2

Decidability of Presburger arithmetic: proof sketch

- Given a formula φ with free variables $x_1, x_2, \ldots, x_t$, we inductively construct an automaton accepting the base-k expansions of those t-tuples $(x_1, \ldots, x_t)$ for which the formula evaluates to true.
- For example, 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.
- If a formula is of the form $\exists x_1, x_2, \ldots x_t \; p(x_1, \ldots, x_t, \ldots, x_u)$, then we use nondeterminism to "guess" the x_i for $1 \leq i \leq t$ and check them.
- This is done by "projecting" away the first t components of the transitions.
- If the formula is of the form $\forall p$, we use the equivalence

  $$\forall p \equiv \neg \exists \neg p;$$

  this may require using something called the "subset construction", which can produce exponential blow-up in the size of the automaton each time.

Decidability of Presburger arithmetic: proof sketch

- We now parse the formula φ, applying well-known constructions on automata to implement the operations in the formula.
- At the end, if there are no free variables, eventually we get a 1-state automaton that either accepts everything ("true") or rejects everything ("false").
- If there are t free variables left, at the end we get an automaton taking t-tuples as input, and accepting those t-tuples of natural numbers making the formula evaluate to "true".

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 size of the logical formula.
- This bound can be improved to double-exponential.

The good news

- With a small extension to Presburger's logical theory — adding the function $V_k(n)$, the largest power of k dividing n — one can also verify statements that are much more interesting! But then the worst-case time bound returns to

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

- Based on a beautiful logical theory due to Büchi, Bruyère, Hansel, Michaux, Villemaire, etc.
- Despite the awful worst-case bound on running time, an implementation often succeeds in verifying statements in the theory in a reasonable amount of time and space.
- Many old results from the literature can been verified with this technique, and many new ones can be proved.

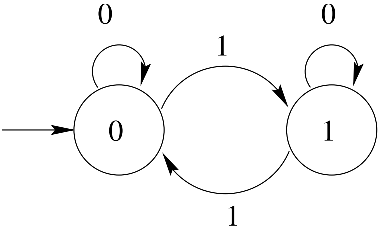
What can we prove things about?

- One large class of objects: the class of *k-automatic sequences*
- These are infinite sequences

$$\mathbf{a} = a_0 a_1 a_2 \cdots$$

  over a finite alphabet of letters, generated by a finite-state machine (automaton)
- The automaton, given n as input, computes a_n as follows:
  - n is represented in some fixed integer base $k \geq 2$
  - The automaton moves from state to state according to this input
  - Each state has an output letter associated with it
  - The output on input n is the output associated with the last state reached

The canonical example of automatic sequence: the Thue-Morse sequence



By determining the parity of the number of 1's in the base-2 expansion of the input n, this automaton generates the *Thue-Morse sequence*

$$\mathbf{t} = (t_n)_{n \geq 0} = 0110100110010110 \cdots .$$

Thue and Morse

Axel Thue (1863–1922)
Norwegian number theorist

Marston Morse (1892–1977)
American mathematician

What's next?

My students built some free software, called `Walnut`, that can prove or disprove statements in the logical theory $\langle \mathbb{N}, +, V_k \rangle$.

Now that we have such a decision procedure that works on a class of mathematically-interesting objects, what can we* do with it?

1. Give new, almost trivial proofs of famous old results for which only complicated and/or case-based proofs exist.
2. Check existing claims in the literature and fix wrong ones.
3. Improve previously-known results (e.g., turn an "if" into an "if and only if").
4. Explore new claims (and obtain new results "purely mechanically", just by stating the properties of the object we want!).

* That is, I and my co-authors (Émilie Charlier, Narad Rampersad, Hamoon Mousavi, Daniel Gabric, Jason Bell, Aseem Baranwal, Thomas Lidbetter, Lucas Mol, Ramin Zarifi, ...) .

Proving a famous old result: $\mathbf{t}$ is overlap-free

Probably the most famous result about the Thue-Morse sequence

$$\mathbf{t} = t_0 t_1 t_2 \cdots = 0110100110010110 \cdots$$

is Thue's 1912 theorem that $\mathbf{t}$ is *overlap-free*.

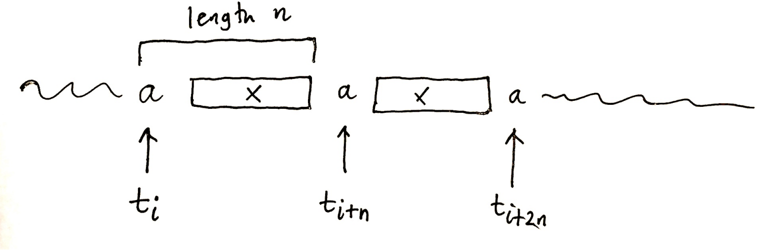
An *overlap* is a word of the form *axaxa*, where *a* is a single letter and *x* is a (possibly empty) word, like the English word `alfalfa`.

When we say $\mathbf{t}$ is overlap-free, we mean it contains no contiguous block that is an overlap.

Let us try to prove this by phrasing the existence of an overlap as a first-order logic statement.

Existence of overlap

If **t** had an overlap, this is what it would look like:



In other words, there would exist integers i, n with $n \geq 1$, such that

$$t_i t_{i+1} \cdots t_{i+n} = t_{i+n} t_{i+n+1} \cdots t_{i+2n}.$$

We can express this in the logical system $\langle \mathbb{N}, +, n \to t_n \rangle$ as follows:

$$\exists i, n \ (n \geq 1) \wedge \forall j \ (j \leq n) \implies t_{i+j} = t_{i+j+n}.$$

The Thue-Morse word is overlap-free

Now we can evaluate the assertion

$$\exists i, n\ (n \geq 1) \wedge \forall j\ (j \leq n) \implies t_{i+j} = t_{i+j+n}$$

using our decision procedure, by translating it into the syntax of `Walnut`:

```
eval tmhasover "Ei,n (n>=1) & Aj (j<=n) => T[i+j]=T[i+j+n]":
```

Here

- `eval` tells `Walnut` to evaluate the statement that follows
- `tmhasover` is just a filename where results will be stored
- `E` means "$\exists$" and `A` means "$\forall$"
- `&` means "logical and"; `=>` means "logical implication"
- `T` is `Walnut`'s way of writing the Thue-Morse sequence

The output of `Walnut`

```
eval tmhasover "Ei,n (n>=1) & Aj (j<=n) => T[i+j]=T[i+j+n]":
computed ~:1 states - 288ms
computed ~:2 states - 3ms
n>=1:2 states - 389ms
 j<=n:2 states - 1ms
  T[(i+j)]=T[((i+j)+n)]:12 states - 15ms
   (j<=n=>T[(i+j)]=T[((i+j)+n)]):25 states - 4ms
    (A j (j<=n=>T[(i+j)]=T[((i+j)+n)])):1 states - 32ms
     (n>=1&(A j (j<=n=>T[(i+j)]=T[((i+j)+n)]))):1 states - 1ms
      (E i , n (n>=1&(A j (j<=n=>T[(i+j)]=T[((i+j)+n)])))):1 states - 0ms
Total computation time: 523ms.
-----
FALSE
```

and so we have **proven** that the Thue-Morse word **t** has no overlaps!

Another example: Dejean's ternary word

Let's see another example of a much shorter proof of an existing theorem.

Françoise Dejean (1972), in a famous paper, gave an example of an infinite ternary word with the property that it avoids $(7/4+\epsilon)$-powers, namely,

$$\mathbf{dej} = 012021201210212021012010 20 \cdots ,$$

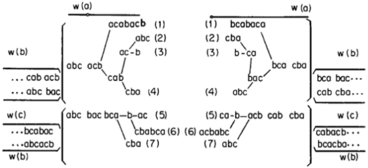
the fixed point, starting with 0, of the 19-uniform morphism

$$\begin{aligned} 0 &\to 0120212012102120210 \\ 1 &\to 1201020120210201021 \\ 2 &\to 2012101201021012102 . \end{aligned}$$

This means that the word **dej** has no factor of the form xx' with x' a prefix of x and $|xx'|/|x| > 7/4$.

Her proof was about 5 pages long and rather complicated.

Excerpts from Dejean's proof



En associant le facteur gauche (4) avec le facteur droit (4), on obtient le morphisme w défini par

$$w(a) = abc\ acb\ cab\ c\ bac\ bca\ cba,$$
$$w(b) = bca\ bac\ abc\ a\ cba\ cab\ acb,$$
$$w(c) = cab\ cba\ bca\ b\ acb\ abc\ bac.$$

Il est clair que w vérifie (i), (iii), et (iv); on s'assurera qu'il répond aussi à la condition (ii). Pour ce morphisme w, $|w(a)| = 19$. On peut s'assurer qu'il n'y a pas d'autre façon de combiner les facteurs gauches et droits obtenus, pour former une solution w' du problème telle que $|w'(a)| \leqslant 19$; en effet, les seuls couples de facteurs gauche et droit pouvant être associés pour donner un mot de longueur au plus 19 sont (1, 2), et par raison de symétrie, (2, 1); (3, 3); (4, 4); et (5, 5). Or, pour le morphisme obtenu dans le cas (1, 2), $w(ac)$ ne vérifie pas la propriété $P_{1/3}$, car ce mot admet pour facteur gauche

et par son facteur droit de longueur q', on obtient un mot h_1 tel que

$$w(h_1) \in X^r f X^{r'} \qquad (0 \leqslant r, r' < 19).$$

Montrons maintenant que si h_2 est un autre mot de X^* tel que

$$w(h_2) \in X^s f X^{s'} \qquad (0 \leqslant s, s' < 19),$$

nécessairement $h_1 = h_2$.

Si f contient un facteur caractéristique du mot $w(x)$ ($x \in X$), après un facteur gauche de longueur n, on sait (lemme 2) que h_1 contient x et que $r + n \equiv n_0 \bmod 19$, où $n_0 = 0, 6$, ou 12 suivant qu'il s'agit d'un facteur caractéristique gauche, central, ou droit de $w(x)$. De même, de $w(h_2) \in X^s f X^{s'}$, on déduit $s + n \equiv n_0 \bmod 19$. Mais alors, $r = s$, et les premières lettres x_1 et x_2 ($x_1, x_2 \in X$) de h_1 et h_2 sont telles que leurs images par w ont un facteur droit non vide commun: le facteur gauche f_0 de longueur $19 - r$ de f. D'après le lemme 3 elles sont égales. De même, $r' = s'$, et si z_1 et z_2 désignent les dernières lettres de h_1 et de h_2, $w(z_1)$ et $w(z_2)$ ont un facteur gauche commun non vide, le facteur droit f_0' de longueur $19 - r'$ de f, donc $z_1 = z_2$. Si l'on définit f_1, g_1, et g_2 par $f = f_0 f_1 f_0'$, $h_1 = x_1 g_1 z_1$, et $h_2 = x_2 g_2 z_2$, on obtient:

$$w(h_1) = w(x_1)\, w(g_1)\, w(z_1) = w(x_1)\, f_1 w(z_1),$$
$$w(h_2) = w(x_1)\, w(g_2)\, w(z_1) = w(x_1)\, f_1 w(z_1).$$

Il s'ensuit que $w(h_1) = w(h_2)$, donc $h_1 = h_2$.

LEMME 5. *Si m possède la propriété $P_{1/3}$, toute sesquipuissance fgf facteur de $w(m)$ telle que f contienne un facteur caractéristique vérifie $|g| \geqslant (1/3)|f|$.*

Another example: Dejean's ternary word

But, using `Walnut`, we can verify her construction in 79 seconds, just by building a formula asserting the existence of a $(7/4+\epsilon)$-power in **dej**:

```
eval dejean "?msd_19 Ei,n n>=1 & Aj (j>=i & 4*j<=4*i+3*n)
     => DEJ[j]=DEJ[j+n]":
```

```
eval dejean "?msd_19 Ei,n n>=1 & Aj (j>=i & 4*j<=4*i+3*n) => DEJ[j]=DEJ[j+n]":
computed ~:1 states - 2ms
computed ~:2 states - 1ms
n>=1:2 states - 36ms
 j>=i:2 states - 4ms
  (4*j)<=((4*i)+(3*n)):11 states - 10145ms
   (j>=i&(4*j)<=((4*i)+(3*n))):14 states - 1164ms
    DEJ[j]=DEJ[(j+n)]:6 states - 690ms
     ((j>=i&(4*j)<=((4*i)+(3*n)))=>DEJ[j]=DEJ[(j+n)]):79 states - 3146ms
      (A j ((j>=i&(4*j)<=((4*i)+(3*n)))=>DEJ[j]=DEJ[(j+n)])):1 states - 5996ms
       (n>=1&(A j ((j>=i&(4*j)<=((4*i)+(3*n)))=>DEJ[j]=DEJ[(j+n)]))):1 states - 2ms
        (E i , n (n>=1&(A j ((j>=i&(4*j)<=((4*i)+(3*n)))=>DEJ[j]=DEJ[(j+n)])))):1 states - 1ms
Total computation time: 79365ms.
-----
FALSE
```

Check existing claims in the literature

Example from a recent preprint:
"...it follows from Proposition 2.1 that every segment of length k of $\mathbf{t}$ is a factor of every segment of length $\ell = 8k - 1$ of $\mathbf{t}$".

Let's check this:

```
eval checkclaim "Ai,j,k (k>=1) => Ep (p>=j) &
     (p+k <= j+8*k-1) & As (s<k) => T[i+s]=T[p+s]":
```

`Walnut` returns `FALSE`, so the claim is wrong.

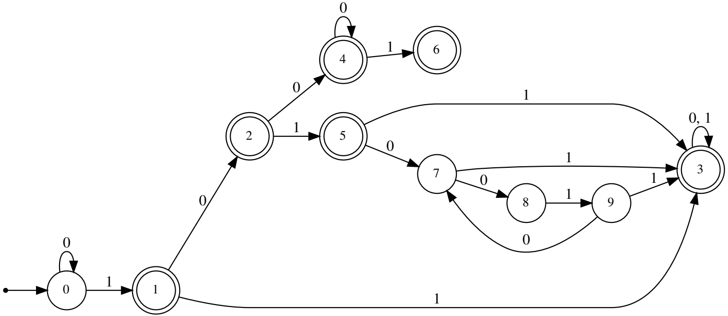
But we can do even more...

Let's find those k for which the claim is true!

Check existing claim in the literature

```
def whichright "Ai,j Ep (p>=j) & (p+k <= j+8*k-1) &
     As (s<k) => T[i+s]=T[p+s]":
```

This produces an automaton, specifying those lengths k, represented in base 2, for which the claim is true.



So the claim was wrong for (e.g.) $k = 10$ and infinitely many k.

Improving previously-known results: unbordered factors

- A word is *bordered* if it can be expressed as uvu for words u, v with u nonempty, and otherwise it is unbordered.
- Example: the English word `ionization` has border `ion`.
- James Currie and Kalle Saari proved that $\mathbf{t}$ has an unbordered factor of length n if $n \not\equiv 1 \pmod{6}$.
- However, these are not the only lengths with an unbordered factor; for example,

  $$0011010010110100110010110100101$$

  is an unbordered factor of $\mathbf{t}$ of length 31.

Unbordered factors

We can express the property that $\mathbf{t}$ has an unbordered factor of length n as follows:

$$\exists i \neg \textsc{Bordered}(i, n)$$

where

$$\textsc{Bordered}(i, n) := \exists j \ (1 \leq j \leq n/2) \ \wedge \ \forall k \ (k < j) \implies t_{i+k} = t_{i+n+k-j}$$

asserts that the length-n factor of $\mathbf{t}$, starting at position i, has a border. Let's translate this to `Walnut`:

```
def bordered "Ej (j>=1) & (j<=n/2) &
     Ak (k<j) => T[i+k]=T[(i+n+k)-j]":
     # T[i..i+n-1] is bordered
def unbordlength "Ei ~$bordered(i,n)":
```

Unbordered factors

Now we can verify the Currie-Saari theorem: *if* $n \not\equiv 1 \pmod 6$, *then* $\mathbf{t}$ *has an unbordered factor of length* 6:

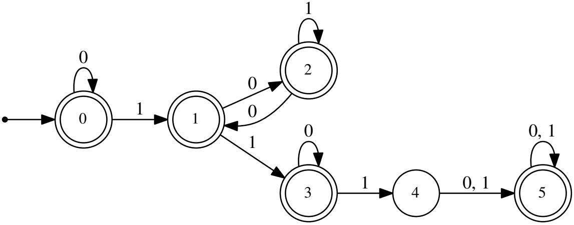
```
eval checkcs "An (1!=n-6*(n/6)) => $unbordlength(n)":
```

and `Walnut` returns `TRUE`.

Unbordered factors

However, we can do much more!

`Walnut` compiles our `unbordlength` definition into an automaton that recognizes the base-2 representation of all n for which $\mathbf{t}$ has a length-n unbordered factor:



and so (by inspection of this automaton) we have improved the Currie-Saari result as follows:

Theorem. The Thue-Morse sequence $\mathbf{t}$ has an unbordered factor of length n if and only if $(n)_2 \notin 1(01^*0)^*10^*1$.

A new example from the OEIS

Jon E. Schoenfield recently looked at the sequence A005652 in the OEIS (On-Line Encyclopedia of Integer Sequences), namely

$$1, 3, 6, 8, 9, 11, 14, 16, 17, 19, 21, 22, 24, 27, 29, 30, \ldots$$

and made the following conjecture: it is the lexicographically least increasing sequence of positive integers with the property that no two distinct terms sum to a Fibonacci number.

How can we prove this using `Walnut`?

We start with one description of the sequence: namely, an integer n appears in sequence A005652 if and only if

$$\lfloor 2\varphi n \rfloor > 2\lfloor \varphi n \rfloor \quad \text{where } \varphi = (1+\sqrt{5})/2.$$

To use it we need a different way to represent numbers, not base k ...

Fibonacci (Zeckendorf) representation

- The Fibonacci numbers: $F_0 = 0$, $F_1 = 1$, $F_n = F_{n-1} + F_{n-2}$

- In analogy with base-2 representation, we can represent every non-negative integer in the form

$$\sum_{0 \le i \le t} \epsilon_i F_{i+2} \quad \text{with} \quad \epsilon_i \in \{0, 1\}.$$

Fibonacci (Zeckendorf) representation

- But then some integers have multiple representations, e.g., $14 = 13 + 1 = 8 + 5 + 1 = 8 + 3 + 2 + 1$
- So we impose the additional condition that $\epsilon_i \epsilon_{i+1} = 0$ for all i: never use two adjacent Fibonacci numbers.
- Usually we write the representation in the form

$$\epsilon_t \epsilon_{t-1} \cdots \epsilon_0,$$

with most significant digit first. So, for example, 19 is represented by 101001. This is called *Fibonacci representation* or *Zeckendorf representation*.

Edouard Zeckendorf (1901–1983)

Fibonacci-automatic infinite words

- Consider a finite automaton that takes Fibonacci representation of n as input
- Outputs are associated with the last state reached
- Invalid inputs (those with two consecutive 1's) are rejected or not considered
- An infinite word results from feeding the canonical representation of each $n \geq 0$ into the automaton.

The Fibonacci decision procedure

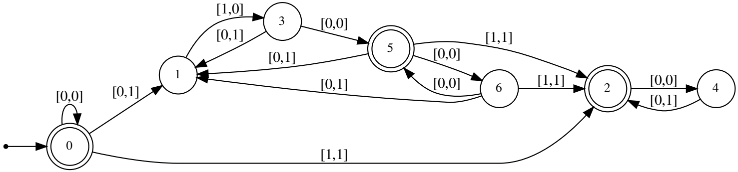
- Exactly like before, except now all integers are represented in Fibonacci representation
- Comparison is easy
- Addition is harder; need an adder
- There is a 17-state automaton that on input (x, y, z) in Fibonacci representation will determine whether $x + y = z$
- Based on ideas originally due to Jean Berstel and since elaborated by others: Frougny, Sakarovitch, etc.

A new example from the OEIS

We can check the condition

$$\lfloor 2\varphi n \rfloor > 2\lfloor \varphi n \rfloor$$

that defines A005652 in `Walnut` because the function $n \mapsto \lfloor \varphi n \rfloor$ is "Fibonacci-synchronized": there is a simple automaton that takes the Zeckendorf representation of n and x in parallel and accepts if and only if $x = \lfloor \varphi n \rfloor$.



Example: it accepts $[0,1][1,0][0,0][0,1][1,0][0,0]$, which corresponds to the pair $(10,16)$.

A new example from the OEIS

So, how do we verify Schoenfield's conjecture?

First step is to check that indeed, for our particular sequence, no two distinct elements sum to a Fibonacci number. We can do this as follows:

```
def a005652 "?msd_fib Ex,y $phin(2*n,x) & $phin(n,y) &
   x>2*y":
reg isfib msd_fib "0*10*":
eval schoenfield1 "?msd_fib Au,v (u!=v & $a005652(u) &
   $a005652(v)) => (~Ew $isfib(w) & w=u+v)":
```

A new example from the OEIS

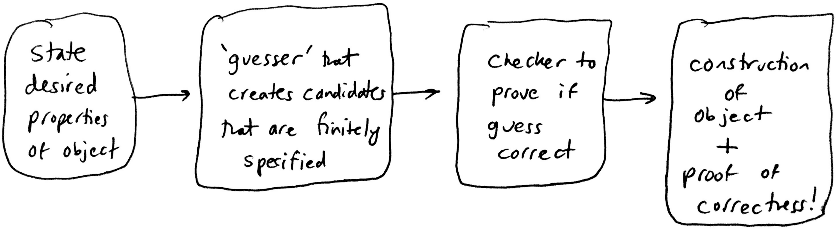
Next, we have to verify it is the lexicographically least such sequence. We can do this inductively, as follows: consider two consecutive elements of the sequence, say x and y, separated by numbers not in the sequence:

$$x, \underbrace{x+1, x+2, \ldots, y-1}, y.$$

It suffices to show that if we chose any of the numbers $x+1, \ldots, y-1$ as a term of the sequence, we would have found two distinct terms that summed to a Fibonacci number. We can do this as follows:

```
def runs "?msd_fib x<y & $a005652(x) & $a005652(y) &
   (Az (x<z & z<y) => ~$a005652(z))":
eval schoenfield2 "?msd_fib Ax,y,z ($runs(x,y) & x<z & z<y)
   => (Ew,t $a005652(w) & w<=x & $isfib(t) & w+z=t)":
```

The best possible (?) research methodology



Advantages:

- no work at all: just state the desired properties of the object, and the program finds an example and proves its correctness.

Disadvantages:

- Having to explain why you should be paid a salary for something that easy!

Heuristics + Decision Procedures Provide Constructions + Proofs

We can combine the depth-first or breadth-first search over a space with a decision procedure to (a) figure out a good candidate for a solution and then (b) prove it is correct.

Example: In 1965, Richard Dean studied the *Dean words:* squarefree words over $\{x, y, x^{-1}, y^{-1}\}$ that are not reducible (that is, there are no occurrences of $xx^{-1}, x^{-1}x, yy^{-1}, y^{-1}y$).

Heuristics Plus Decision Procedures Provide Proofs

Let us use the coding $0 \leftrightarrow x$, $1 \leftrightarrow y$, $2 \leftrightarrow x^{-1}$, $3 \leftrightarrow y^{-1}$.

So we must not allow the blocks 02, 20, 13, or 31.

We can use "automatic breadth-first search" to find a candidate for an infinite Dean word.

In automatic breadth-first search, you guess that the infinite word you want to construct is k-automatic for some integer $k \geq 2$, and generated by an automaton of $\leq \ell$ states.

You then use BFS to explore the tree of all words w obeying the particular constraints, such that the smallest automaton generating w has $\leq \ell$ states.

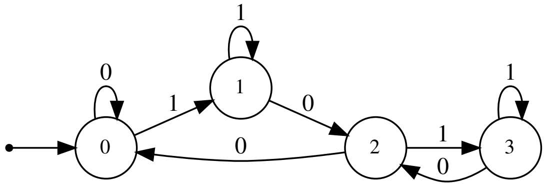
Heuristics Plus Decision Procedures Provide Proofs

If you are lucky, BFS will converge to the prefixes of a single *k*-automatic infinite word (or small number of such words).

When implemented for Dean words, breadth-first search quickly converges on the sequence

$$0121032101230321 \cdots ,$$

which (using the Myhill-Nerode theorem) we can guess as generated by the automaton below:



Heuristics Plus Decision Procedures Provide Proofs

Then we carry out the following Walnut commands:

```
morphism d "0->01 1->21 2->03 3->23":
promote DE d:
eval dean1 "Ei,n (n>=1) & At (t<n) => DE[i+t]=DE[i+n+t]":
# check if there’s a square
eval dean02 "Ei DE[i]=@0 & DE[i+1]=@2":
eval dean20 "Ei DE[i]=@2 & DE[i+1]=@0":
eval dean13 "Ei DE[i]=@1 & DE[i+1]=@3":
eval dean31 "Ei DE[i]=@3 & DE[i+1]=@1":
# check for existence of factors 02, 20, 13, 31
```

All of these return FALSE, so this word is a Dean word. We have thus proved the existence of Dean words with essentially no human intervention.

What other properties of automatic sequences are decidable?

- A difficult candidate: abelian properties
- We say that a nonempty word x is an *abelian square* if it 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 $\langle \mathbb{N}, +, V_k \rangle$
- However, for some sequences (e.g., Thue-Morse, Fibonacci, Tribonacci) many abelian properties are decidable

Tribonacci synchronization: a new result

We can write first-order formulas for properties of the Tribonacci sequence $\mathbf{tr} = 0102010\cdots$, defined as the fixed point of the morphism $0 \to 01$, $1 \to 02$, $2 \to 0$.

An *abelian cube* is a word of the form $w = x\,x'\,x''$, where x', x'' are permutations of x, like the English word `deeded`. The order of the abelian cube w is defined to be $|x|$.

What are the orders of abelian cubes appearing in $\mathbf{tr}$?

Answer: there is a Tribonacci automaton of 1169 states (!) recognizing the set of all these orders (expressed in the Tribonacci numeration system). Probably there is no simple description of what these orders are.

Additive cubes

Similarly, one can study the *additive cubes* appearing in the Tribonacci word **tr**.

These are factors of **tr** of the form $xx'x''$, where $\sum x = \sum x' = \sum x''$.

Theorem. *There is a Tribonacci automaton with 4927 states (!) recognizing the Tribonacci representation of the orders of additive cubes in* **tr**.

Once again, probably there is no simple description of these orders.

Decision methods as a kind of powerful telescope

The use of the "light-year" as a yardstick strikes one with a certain awe. This amounts to taking a distance of nearly 6,000,000,000,000 miles as the unit for the measurement of astronomical distances; and in some of his calculations which have to do with extra-galactic systems the astronomer has to apply this little measuring rod thousands of times. These vast distances and these vast numbers stagger the imagination, and yet the mathematician reaches out with his high-powered machines and his high-powered theorems and investigates the internal structure of his distant bodies much as the astronomer inquires into the structure of some distant star.

– D. N. Lehmer, "Hunting Big Game in the Theory of Numbers", 1932

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 $\mathrm{FO}(\mathbb{N}, +, n \to \mathbf{s}[n])$ is powerful enough to express the assertion that "n is a square"
- With that, one can express multiplication, and so it is undecidable.

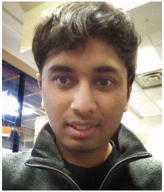
The Walnut Prover

Our publicly-available prover, originally written by Hamoon Mousavi, is called `Walnut` and can be downloaded from

https://cs.uwaterloo.ca/~shallit/walnut.html .

Designer and Implementers of Walnut

Hamoon Mousavi—Designer and Implementer



Aseem Baranwal—implementer

Laindon C. Burnett—implementer

Anatoly Zavyalov—implementer

For further reading

Available at
a fine bookstore
near you!