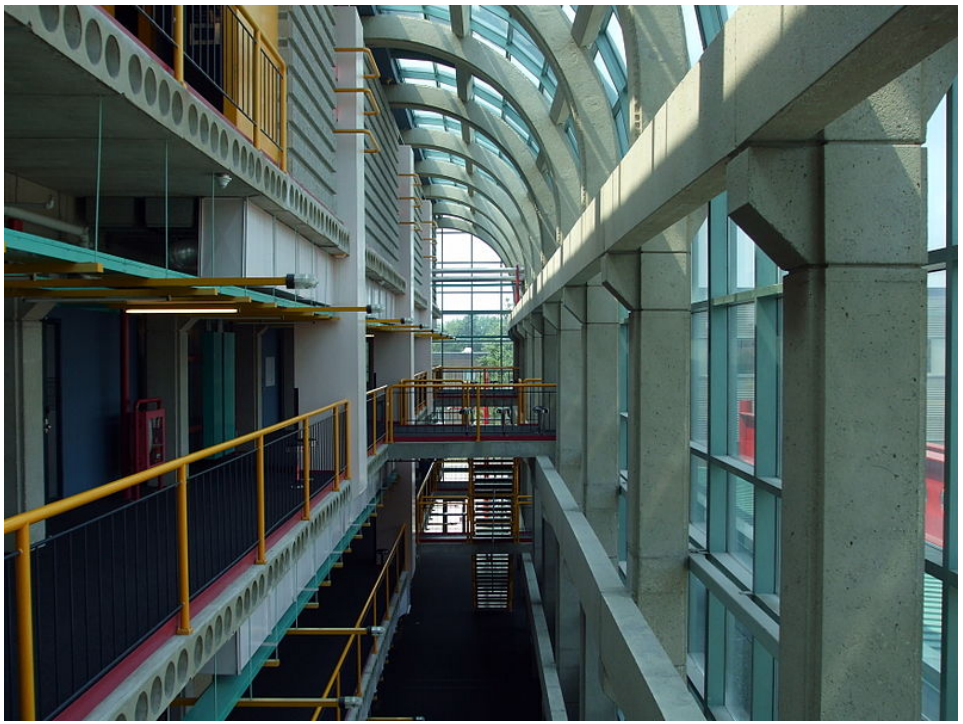
# Adventures with an Automatic Prover

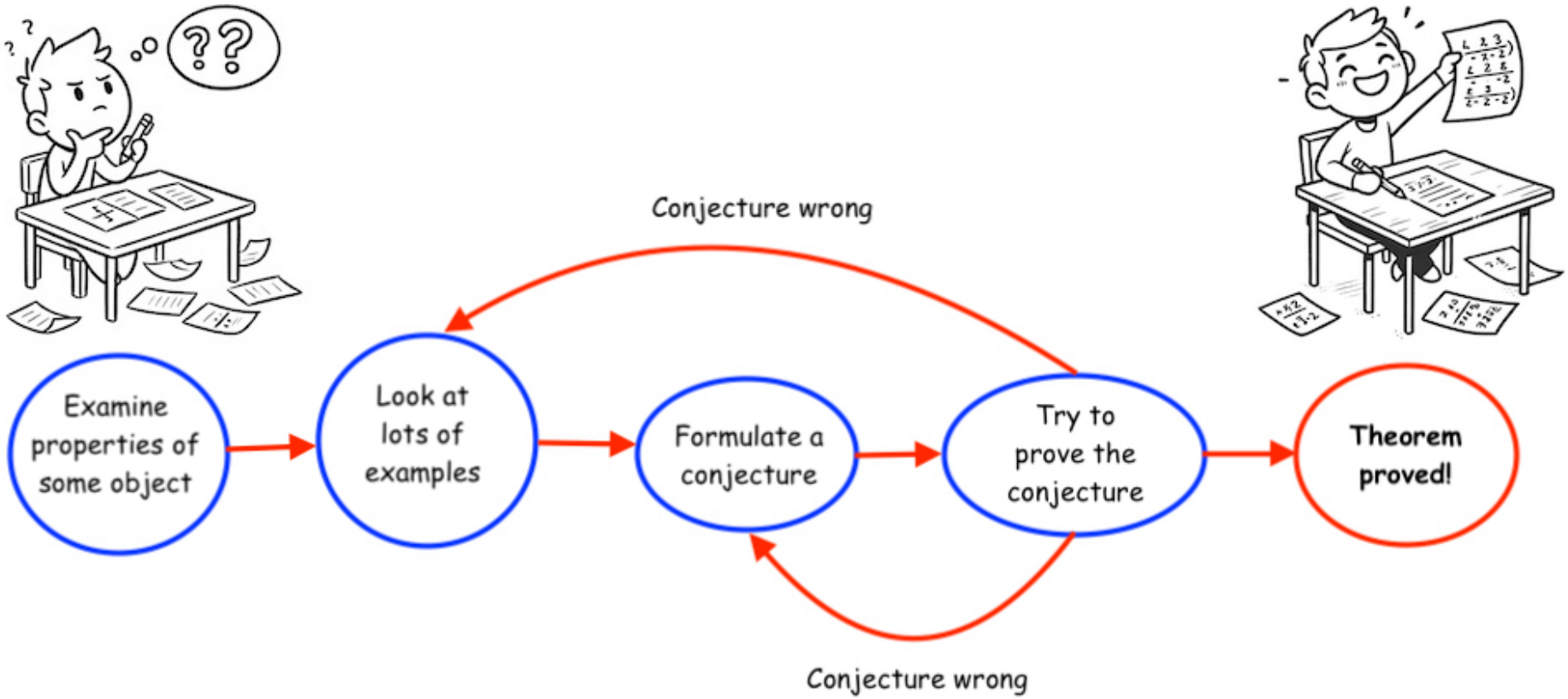
Jeffrey Shallit

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



Davis Centre, U. Waterloo

# The familiar research methodology for mathematics



# Using a proof assistant

Examine properties of some object

Look at lots of examples

Conjecture wrong

Formulate a conjecture

Try to prove the conjecture

Conjecture wrong

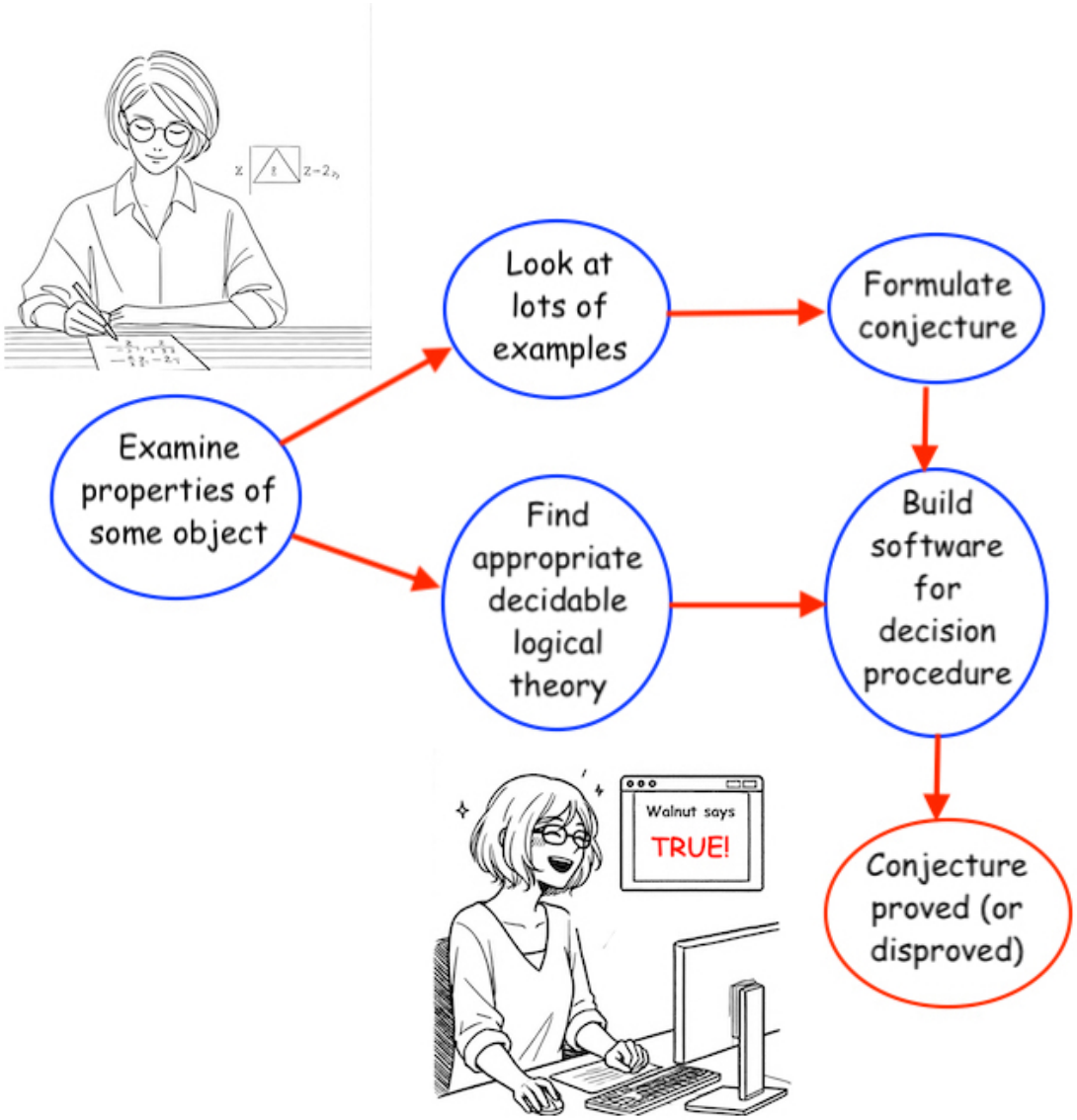
Theorem proved!

Uh oh!

Formalize proof with proof assistant

Proof verified!

# A new (?) research methodology for mathematics




Advantages:

- you don't need to be very clever
- sometimes it can even generate the "right" conjecture, automatically

Disadvantages:

- decision procedures don't exist for most of mathematics
- sometimes they take ridiculous amounts of space and time
- proof may not give much insight as to "why"

# 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

# A decidable theory: Presburger arithmetic

- 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:

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

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 $\Sigma$.

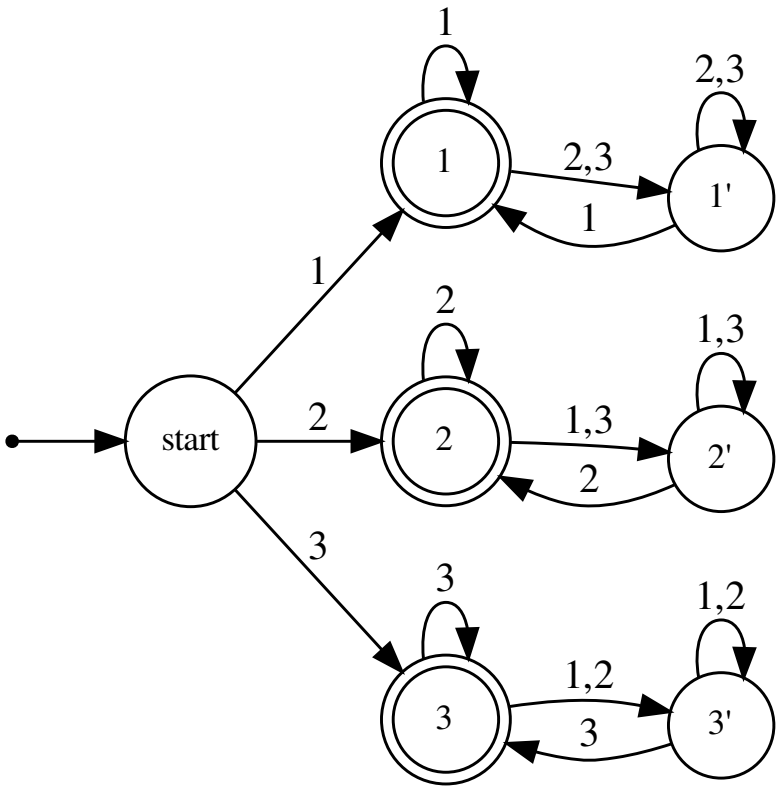
As each letter is processed from left to right, the automaton looks up in a table (the "transition function") which state to go to, based on the current state and current input letter.

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.

The set of all accepted strings is called a *regular language*.

# Example of an automaton

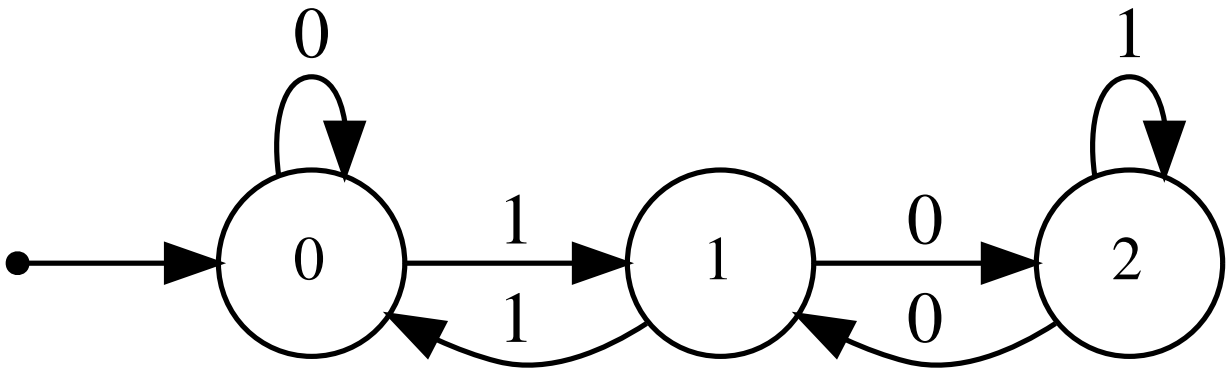
Here is an automaton accepting those words over $\{1, 2, 3\}$ whose first symbol is the same as the last symbol:




Double circle: indicates final state.

# An automaton can also compute a finite-range function

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 input seen so far is congruent to $i$ (mod 3)".

# What is Walnut?

Walnut is a free, open-source theorem prover that is based on an extension of Presburger arithmetic.

It can provide rigorous proofs or disproofs of certain claims stated in first-order logic.

It is not a general-purpose proof assistant.

It has been used in over 100 published papers and books to verify existing results (often replacing pages and pages of tedious inductions), correct false published results, solve previously-unproved conjectures, and prove new results.

For some theorems, the Walnut proofs are the only known ones.

# Example: Chicken McNuggets in Walnut

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

```
eval chicken "(A n (n>43) => E x,y,z n=6*x+9*y+20*z) &
   (~E x,y,z 43=6*x+9*y+20*z)";

computed ~:1 states - 25ms
computed ~:2 states - 2ms

----
TRUE
```

# Going beyond Presburger

- With a small **extension** to Presburger's logical theory—adding the function $V_k(n)$, the largest power of $k$ dividing $n$—we still get a decidable theory!
- And now we can also verify statements that are *much* more interesting!
- Ideas are based on a beautiful logical theory due to Büchi and Bruyère-Hansel-Michaux-Villemaire, called *Büchi arithmetic*.
- Can be extended to numeration systems other than base $k$, such as Fibonacci (Zeckendorf) representation (theory due to Christiane Frougny and her co-authors)
- Also implemented in the free software `Walnut`, originally designed by Hamoon Mousavi
- Many old results from the literature can been verified with this technique, and many new ones can be proved.

# How does the decision procedure work?

`Walnut` compiles a first-order statement into a series of basic operations on finite automata, which are then executed.

Basic operations include: Boolean operations on languages (union, intersection, negation, etc.), addition, projection of coordinates (corresponding to $\exists$), etc.

If an expression contains free variables, the automaton constructed accepts those inputs corresponding to tuples of integers making the expression evaluate to `TRUE`.

If an expression contains no free variables, the resulting automaton is a single state, with no inputs, that either accepts everything (`TRUE`) or rejects everything (`FALSE`).

# The bad news and the good news

- The worst-case running time of the automaton-based 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.
- 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.

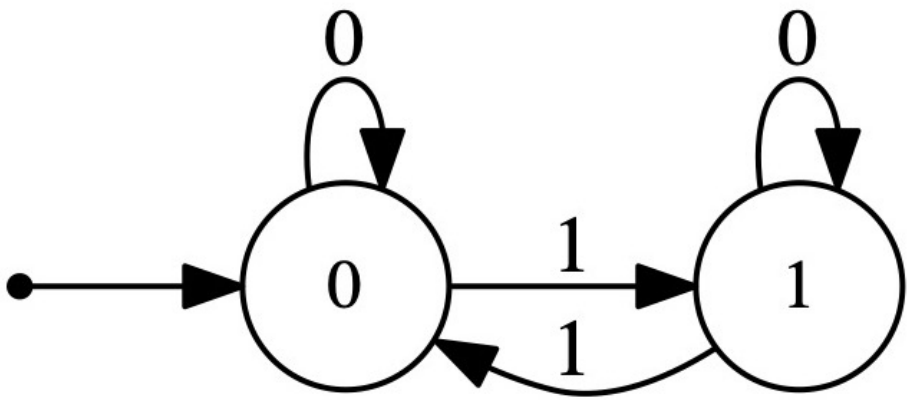
# What can we prove things about?

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

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

  taking their values in a finite set, and computed by a finite-state machine (automaton).
- The automaton is given $n$ as input, written in base $k \geq 2$, and $a_n$ equals the output provided by the last state reached.
- Examples include the Thue-Morse sequence, the Rudin-Shapiro sequence, the infinite binary Fibonacci word, Sturmian words of quadratic slope, the paperfolding words, etc.

# 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*

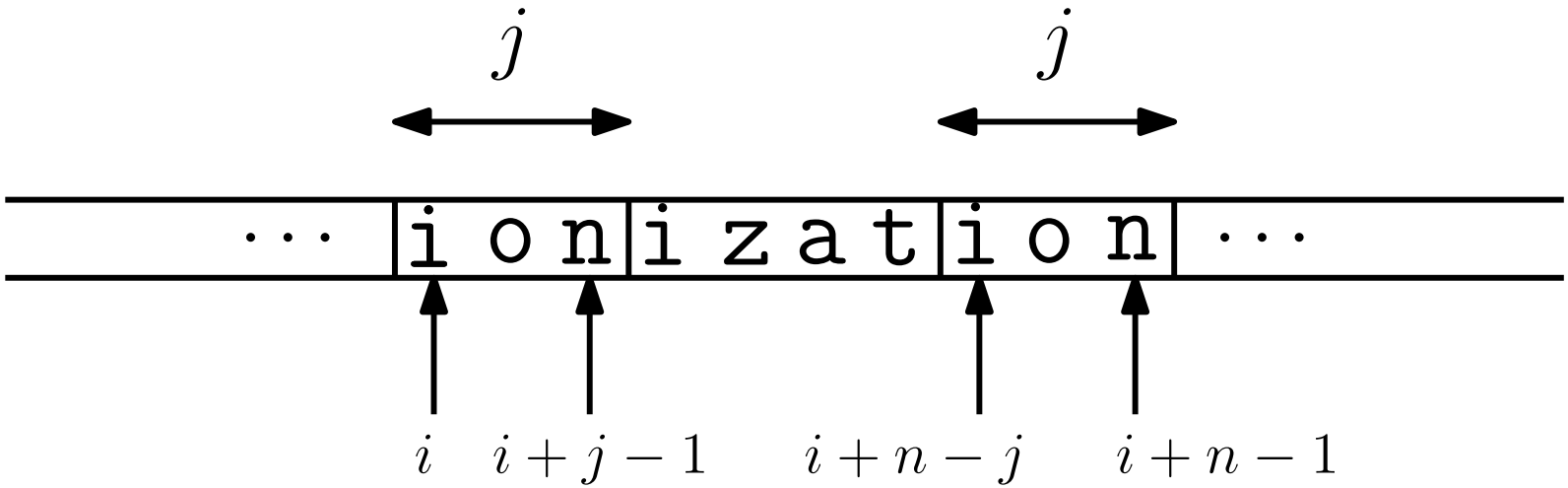
$$\mathbf{t} = (t_n)_{n \geq 0} = 0110100110010110 \cdots ,$$

a famous sequence with many amazing properties.

# A result of Currie and Saari

- 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 if $n \not\equiv 1 \pmod 6$ then **t** has an unbordered factor of length $n$.
- Let's prove this with `Walnut`.

# A result of Currie and Saari




We can express the property that the factor of $\mathbf{t}$ of length $n$, beginning at position $i$, has a border of length $j$, as follows:

$$\text{Bordered}(i, j, n) := \forall k \ (k < j) \implies t_{i+k} = t_{i+n-j+k}.$$

We can express the assertion that $\mathbf{t}[i..i+n-1]$ is an unbordered factor of $\mathbf{t}$ as follows:

$$\text{Unbordered}(i, n) := \forall j \ (j \geq 1 \wedge j \leq n/2) \implies \neg\, \text{Bordered}(i, j, n).$$

# Unbordered factors

We can express the assertion that **t** has an unbordered factor of length $n$ as follows:

$$\text{Currie}(n) := \exists i \ \text{Unbordered}(i, n).$$

Let's translate all this to `Walnut`:

$\text{Bordered}(i, j, n) := \forall k \ (k < j) \implies t_{i+k} = t_{i+n-j+k}.$

```
def bordered "A k (k<j) => T[i+k]=T[(i+n+k)-j]":
```

$\text{Unbordered}(i, n) := \forall j \ (j \geq 1 \wedge j \leq n/2) \implies \neg \text{Bordered}(i, j, n).$

```
def unbordered "A j (j>=1 & j<=n/2) => ~$bordered(i,j,n)":
```

$\text{Currie}(n) := \exists i \ \text{Unbordered}(i, n).$

```
def currie "E i $unbordered(i,n)":
```

# Unbordered factors

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

```
[Walnut]$ eval checkcs "A n (1!=n-6*(n/6)) => $currie(n)";

----
TRUE
```

`Walnut` returns `TRUE`. The theorem is proved!

However, we can do much more!

The lengths $n \not\equiv 1 \pmod 6$ are not the only lengths with an unbordered factor.

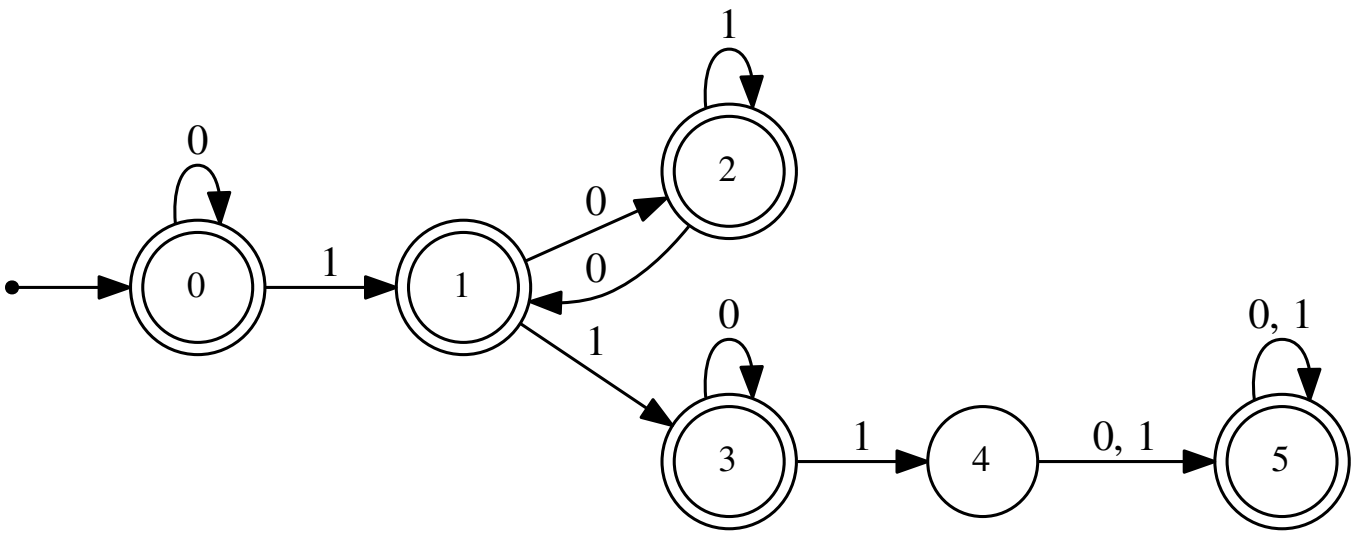
For example,

0011010010110100110010110100101

is an unbordered factor of **t** of length 31.

# Unbordered factors: `Walnut` tells you the theorem

`Walnut` compiles our `currie` definition into a 6-state automaton that recognizes the base-2 representation of all $n$ for which $\mathbf{t}$ has a length-$n$ unbordered factor.




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

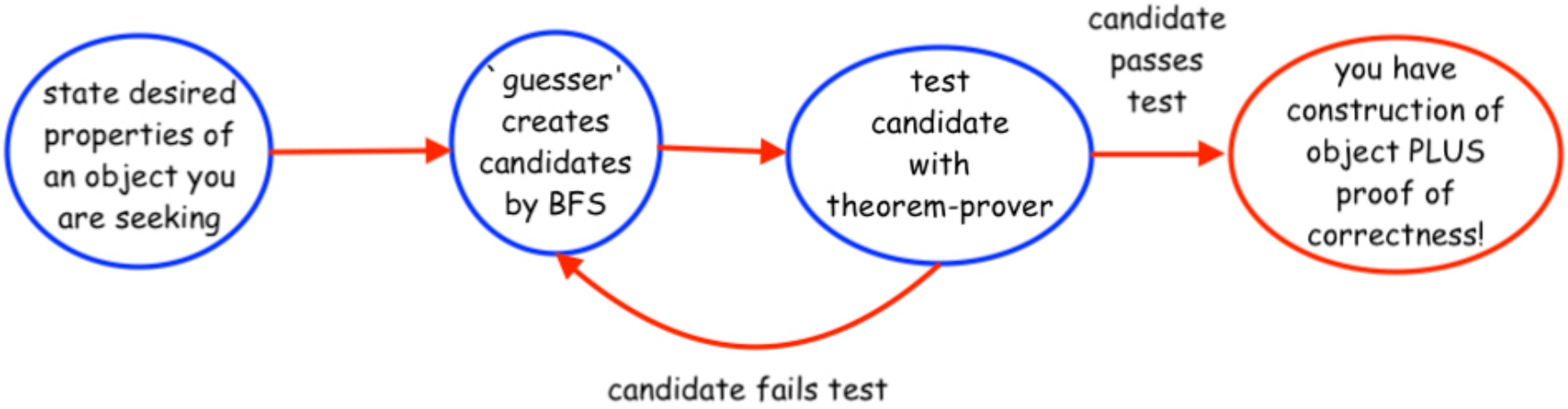
**Theorem.** The Thue-Morse sequence $\mathbf{t}$ has an unbordered factor of length $n$ if and only if $(n)_2 \notin 1(01^*0)^*10^*1$.

# Finding an object with a given property 'automatically'

We just saw an example of proving (and improving) a result from the literature.

Sometimes we can do even more: we can merely state the properties we want the desired object to have, and then find it "automatically".

# 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 for something that easy!

# Thue's problem

Probably the most famous problem in combinatorics on words: *is there an infinite word over a 3-letter alphabet containing no two consecutive identical nonempty blocks?*

Two consecutive identical blocks are called a *square*. Example: `murmur`.

First posed (and solved) by Thue in 1906.

Can't do it over a 2-letter alphabet: every word of length $\geq 4$ over $\{0, 1\}$ contains a square.

Axel Thue (1863–1922)
Norwegian
mathematician

# Solving Thue's problem

Two steps:

1. Identify a candidate infinite word.
2. Prove that the candidate word avoids squares.

A backtracking search quickly finds very long finite words over $\{0, 1, 2\}$ avoiding squares.

We'd like an infinite word where the avoidance property is easy to prove.

Let us guess that such a word $a_0 a_1 a_2 \cdots$ could be generated by a finite automaton reading inputs $n$ in base 2.

# Finding a candidate automaton

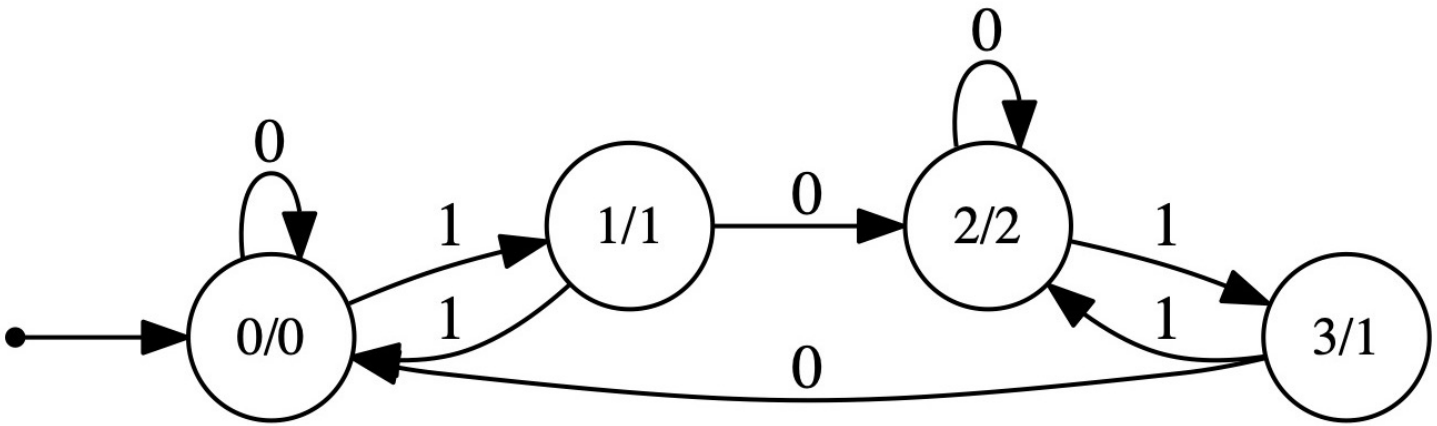
There is an algorithm that, given a finite word $a_0 a_1 \cdots a_{n-1}$, finds a smallest automaton whose outputs are consistent with that word.

We can then do a breadth-first search (BFS) of the space of all words over $\{0, 1, 2\}$, keeping only the squarefree words, and making sure that they can be generated by an automaton of at most $t$ states.

If $t = 2, 3$ then the search terminates with no automaton found.

# A candidate is found!

But for $t = 4$ we quickly hit on the following candidate automaton `TA`.




It generates the following sequence:

$$0120210121020120210201210120210121020121012021020120210121\cdots$$

which certainly appears to have no squares in it...
But we need a proof...

# Proving that the candidate works

```
eval squarefree "~E i,n n>=1 & A t (t<n)
        => TA[i+t]=TA[i+n+t]":
```

And here's the output:

```
[Walnut]$ eval squarefree "~E i,n n>=1 & A t (t<n) => TA[i+t]=TA[i+n+t]":
n>=1:2 states - 0ms
 t<n:2 states - 0ms
  TA[(i+t)]=TA[((i+n)+t)]:12 states - 4ms
   (t<n=>TA[(i+t)]=TA[((i+n)+t)]):25 states - 1ms
    (A t (t<n=>TA[(i+t)]=TA[((i+n)+t)])):1 states - 4ms
     (n>=1&(A t (t<n=>TA[(i+t)]=TA[((i+n)+t)]))):1 states - 0ms
      (E i , n (n>=1&(A t (t<n=>TA[(i+t)]=TA[((i+n)+t)])))):1 states - 0ms
       ~(E i , n (n>=1&(A t (t<n=>TA[(i+t)]=TA[((i+n)+t)])))):1 states - 0ms
Total computation time: 9ms.

TRUE
```

And so we have found an infinite squarefree sequence and proved it correct!

# Application: Additive number theory

Let $S \subseteq \mathbb{N}$ be a subset of the natural numbers.

Additive number theory is concerned with the following kinds of questions:

1. What numbers can be written as the sum of $k$ numbers chosen from $S$? (Distinct, or not necessarily distinct; exactly $k$ or at most $k$)
2. Can all elements of $\mathbb{N}$ be written in that way?
3. Can all sufficiently large elements be written in that way?

These are all first-order statements, and so `Walnut` can solve all the basic problems of additive number theory for automatic sequences.

# Additive number theory

For example, consider the numbers

$$S = \{1, 2, 4, 7, 8, 11, 13, 14, 16, 19, 21, 22, \ldots\},$$

the so-called odious numbers (those $i$ for which $t_i = 1$).

**Theorem.** Every natural number $n > 15$ is the sum of three distinct odious numbers.

*Proof.* We use `Walnut`:

```
eval sum3do "A n (n>15) => E i,j,k (i!=j) & (i!=k) &
   (j!=k) & (n=i+j+k) & (T[i]=@1) & (T[j]=@1) & (T[k]=@1)":
```

And `Walnut` returns TRUE.

# Linear representations

The automaton approach entails additional "non-logic" techniques; for example, enumeration.

Suppose we have an automaton computing a boolean function $f$ of two arguments $i$ and $n$, and we want to compute the number of $i$ for which $f(i, n)$ is true. Call this $g(n)$.

Then, directly from the automaton, we can find a linear representation for $g$ as follows:

$$g(n) = v \cdot \gamma(x) \cdot w,$$

where

(a) $v$ is a $1 \times t$ vector;

(b) $w$ is a $t \times 1$ vector;

(c) $\gamma$ is a $t \times t$-integer-matrix-valued morphism;

(d) $x$ is the base-$k$ representation of $n$.

# Linear representations

And given two linear representations, we can decide if they represent the same function.

So we can test many kinds of conjectures about the number of representations.

Example: count number $g(n)$ of unbordered factors of length $n$ of $\mathbf{t}$.

Can't just count the number of $i$ for which $\mathbf{t}[i..i+n-1]$ is unbordered, because the same unbordered factor appears infinitely often in $\mathbf{t}$.

Solution: count the number of $i$ for which $\mathbf{t}[i..i+n-1]$ is unbordered **and** $\mathbf{t}[i..i+n-1]$ never appeared at an earlier position of $\mathbf{t}$.

# Linear representations

Let us count the *novel* unbordered factors of **t**: those that never appeared before in **t**.

```
def eqfac "A t (t<n) => T[i+t]=T[j+t]":
def novel "A j $eqfac(i,j,n) => j>=i":
def unbord n "$novel(i,n) & $unbordered(i,n)":
```

This produces a linear representation for $g(n)$ of rank 22.
From this linear representation we can, for example, prove that there are

$$\frac{1}{3}(2^{n-1} + 4(-1)^n)$$

unbordered factors of **t** of length $2^n - 1$, for $n \geq 3$.
Idea involves looking at zeros of characteristic polynomial of $\gamma(1)$.

# Dombi's conjecture

Let $r(k, A, n)$ be the number of representations of $n$ as a sum of $k$ elements chosen from a set $A$.

In 2002 Gergely Dombi stated a conjecture in additive number theory: there is no subset $A \subseteq \mathbb{N}$ with $\mathbb{N} \setminus A$ infinite and $r(3, A, n)$ eventually increasing.

I found a counterexample to Dombi's conjecture, as follows:

Guess that there is a set $A$ accepted by a finite automaton.

Find a candidate $A$ via BFS with small automaton description.

Using Walnut, compute linear representations for $r(3, A, n)$ and $r(3, A, n + 1)$.

# Dombi's conjecture

Compute a linear representation for $r(3, A, n+1) - r(3, A, n)$.

From empirical data, guess a simple expression $E_n$ for this difference.

Form another linear representation for $E_n$ directly, and then check that it is indeed the same as that for $r(3, A, n+1) - r(3, A, n)$.

Using $E_n$, prove that $r(3, A, n+1) - r(3, A, n)$ is always positive.

# How can we rely on Walnut's results?

In general, there doesn't seem to be any way to get a short certificate of correctness that one can verify.

For many calculations, the only way to check them seems to be to run them again.

Walnut consists of two parts: a parser that takes the first-order statement, and compiles it into a short list of simple basic operations on finite automata.

And an automata engine that performs those simple basic operations: Cartesian product of automata, projection of transitions to smaller alphabet, subset construction, minimization, etc.

It seems likely both parts are capable of formal verification.

# How can we rely on Walnut's results?

There are already libraries for automata theory in Isabelle, for example: https://www.isa-afp.org/entries/CAVA_Automata.html.

So it should be possible to formally verify the code, or at least large parts of it.

This would allow integration of Walnut proofs with existing proof assistants.

Looking for collaborators!

# Self verification and construction

`Walnut`'s basic algorithm to compute the automaton can sometimes take a quite unreasonable amount of time and space.

For example, consider

```
def trib_eqfac "?msd_trib A t (t<n) => TR[i+t]=TR[j+t]":
```

which generates an automaton testing whether $\mathbf{tr}[i..i+n-1] = \mathbf{tr}[j..j+n-1]$. Here **tr** is the Tribonacci word, fixed point of $0 \to 01$, $1 \to 02$, $2 \to 0$.

This `Walnut` command generates an intermediate automaton with more than 250,000,000 states, even though the final result has only 26 states.

The 'eqfac' predicate is one of the most basic ones we need in order to understand a sequence. So we really need to be able to compute this efficiently.

A new idea: generate the automaton "from the ground up" using Angluin's learning algorithm and self-verification.

# New ideas

Angluin's algorithm learns the automaton for a regular language $L$ with the help of a "teacher".

When presented with a word $w$, the teacher says whether or not $w \in L$.

One can also present the teacher with a candidate automaton $A$ for $L$ and ask if it is correct. If it is correct, you are done. Otherwise, the teacher gives a counterexample, a member of $(L(A) \setminus L) \cup (L \setminus L(A))$.

# New ideas

As an analogy, consider building a house out of a pile of rubble with the help of a master builder.

At every stage you can pick up an item, like a brick, and ask if it goes in some specific place. If it does, you can put the brick there.


Furthermore, you can point to what you have built so far and ask "Is it a house yet?" If so, you are done. If not, the master builder says something like, "No, it is missing part of the roof."

# New ideas

To generate the 'eqfac' predicate for a given automatic sequence, we can use `Walnut` to be the teacher! It can teach *itself* what the right automaton is.

Here the language is

$$L = \{(i, j, n) \ : \ X[i..i+n-1] = X[j..j+n-1]\}.$$

When presented with a *specific* triple $q$ like $(23, 45, 56)$, we can evaluate the predicate

```
(t<56) & X[23+t]!=X[45+t]
```

with `Walnut` and get an automaton out of it. We can then search this automaton with breadth-first search to see if it accepts anything. It accepts something iff $q \notin L$. So `Walnut` can answer membership queries.

# New ideas

Now suppose we have a given candidate automaton $A$, and we wish to check if it is correct.

We claim 'eqfac' is *self-verifying* (aka "self-reducible"): $A$ computes 'eqfac' correctly if and only if both of the following hold:

(i) $\forall i, j \; A(i, j, 0)$;

(ii) $\forall i, j, n \; A(i, j, n+1) \iff (A(i, j, n) \wedge X[i+n] = X[j+n])$.

We can check these conditions with `Walnut`.

And if either condition fails, `Walnut` can give us the smallest counterexample.

**Theorem.** Using Angluin's algorithm we can construct an automaton for the 'eqfac' predicate in time *polynomial* in the number of states of the original automaton `X` and the minimal 'eqfac' automaton.

# Future challenges

We saw that direct translation of the `Walnut` query

```
def trib_eqfac "?msd_trib A t (t<n) => TR[i+t]=TR[j+t]":
```

results in an intermediate automaton of more than 250,000,000 states after days of computation, although the final result has only 26 states. By contrast, the equivalent statement

```
def trib_eqfac_2 "?msd_trib A u,v (u>=i & u<i+n & u+j=v+i)
   => TR[u]=TR[v]":
```

results in an intermediate of automaton of 384,498 states (and a final result of 26 states), in 30 seconds of CPU time.

This and the previous `trib_eqfac` differ only by some substitution of variables.

Why do they have such different behavior?

# Connections with the rest of this conference

- Can we predict which kinds of queries might take inordinate amounts of time and space? And if so "automatically" rewrite them into equivalent statements that don't?
- Machine learning: use of Angluin's algorithm.
- Using a SAT solver: we found minimal automata for computing the $i$'th bit of $\varphi = (1 + \sqrt{5})/2$, the golden ratio, by solving a more general problem and then specializing to a specific class of inputs.
    - We used a MinDFA solver called DFA-Inductor to generate SAT encodings, which are then passed to the CaDiCaL SAT solver.
- Integration with tools like LEAN to prove correctness of output.
- Auto-formalization: translate natural language specification directly into first-order queries `Walnut` can process.

Looking for collaborators!

# Summing up

- Decision procedures are rare but can be quite useful in limited domains.
- The decision procedures can be combined with existing techniques to resolve conjectures and simplify proofs.
- Even nonelementary worst-case running times may not reflect running time in practice! Don't be scared to try to implement them.

# Walnut demo and tutorial

4 PM in Simons Institute Room 116 today (Thursday April 10).

I would be glad to talk about Walnut any time!

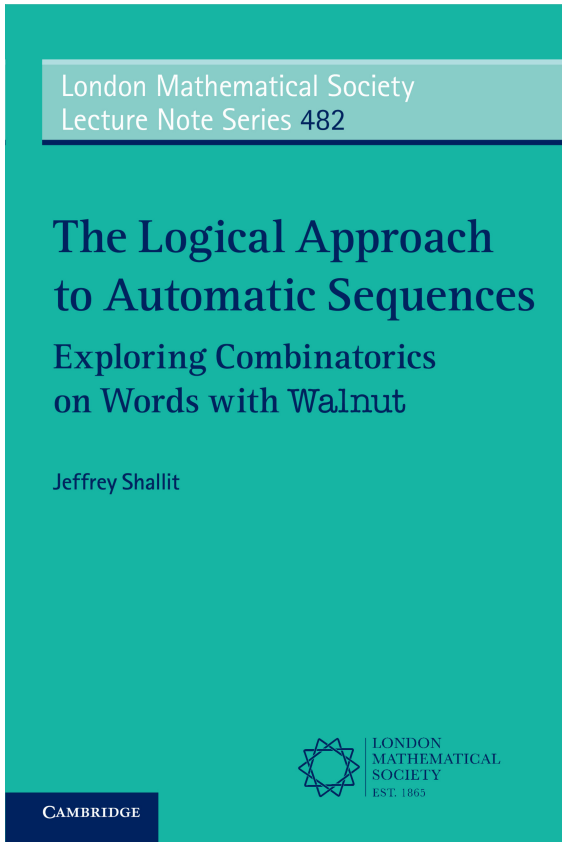
Walnut tutorial at
https://cs.uwaterloo.ca/~shallit/walnut-tutorial.html

# The Walnut community

The Walnut theorem-prover and documentation is available at

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

There is also a Discord server at https://discord.gg/c5wzTXhMzT.



# Limits to the approach

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

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

- 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.