Results on Infinite Words Obtained with an Automatic Prover

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

The outline

1. Hilbert’s dream
2. Fibonacci (Zeckendorf) representation
3. Fibonacci-automatic words
4. Logic
5. The infinite and finite Fibonacci words
6. Fibonacci-regular sequences and enumeration
7. Recent applications to avoidability
1. Hilbert’s dream

- To show that every true statement is provable (killed by Gödel)
- To provide an algorithm to prove every provable statement (killed by Turing)
- Nevertheless, some subclasses of problems are decidable - an algorithm will prove or disprove them (e.g., Presburger arithmetic, first-order theory of the reals)
- Another interesting subclass: the Wilf-Zeilberger method for proving binomial coefficient identities
2. Fibonacci (Zeckendorf) representation

- 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

\[
\sum_{0 \leq i \leq t} \epsilon_i F_{i+2} \quad \text{with} \quad \epsilon_i \in \{0, 1\}.
\]
But then some integers have multiple representations, e.g.,
\[13 = 8 + 5 = 8 + 3 + 2\]
So we impose the additional condition that \(\varepsilon_i\varepsilon_{i+1} = 0\) for all \(i\)
Usually we write the representation in the form
\[\varepsilon_t\varepsilon_{t-1} \cdots \varepsilon_0,\]
with most significant digit first. So, for example, 19 is represented by 101001. This is called Fibonacci or Zeckendorf representation.

Figure: Edouard Zeckendorf (1901–1983)
Consider a finite automaton that takes Fibonacci representation of \( n \) as input

- Outputs are associated with the last state reached
- Invalid representations are ignored
- An infinite word results from feeding the representation of each \( n \geq 0 \) into the automaton

Example: the Fibonacci infinite word

\[ f = 0100101001001 \ldots \]
Patterns in infinite words

- **Squares**: words of the form $xx$ (like the French word *couscous*)
- **Overlaps**: just a little more than a square (like the French word *entente*)
- **Cubes**: words of the form $xxx$ (like the French sort-of-word *blablabla*)
Let \( \text{Th}(\mathbb{N}, +, 0, 1, <) \) denote the set of all true first-order sentences in the logical theory of the natural numbers with addition.

Example: in this theory we can express the so-called “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:

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

Here, of course, “6x” is shorthand for the expression “\( x + x + x + x + x + x \)”, and similarly for 9y and 20z.
Presburger proved that $\text{Th}((\mathbb{N}, +, 0, 1, <))$ is *decidable*: that is, there exists an algorithm that, given a sentence in the theory, will decide its truth.
Decidability of Presburger arithmetic: proof sketch

- 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 $\Sigma_k^n$, 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

$$[1, 0][0, 0][1, 1][0, 1][1, 1].$$
Then 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”.

\[
\begin{align*}
\{[a, b, c] : a + b = c\} & \quad \{[a, b, c] : a + b + 1 = c + k\} \\
\{[a, b, c] : a + b + 1 = c\} & \quad \{[a, b, c] : a + b = c + k\}
\end{align*}
\]

**Figure:** Checking addition in base $k$
Relations like $x = y$ and $x < y$ can be checked similarly.

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.

Finally, the truth of a formula can be checked by using the usual depth-first search techniques to see if any final state is reachable from the start state.
The worst-case running time of the algorithm above is bounded above by

$$2^{2^{\ldots 2^p(N)}}$$

where the number of 2’s in the exponent is equal to the number of quantifiers, $p$ is a polynomial, and $N$ is the number of states needed to describe the underlying automatic sequence.
The good news

- With a small addition to the logical theory, we can also verify many other kinds of statements (e.g., about the Fibonacci-automatic words)
- Despite bad worst-case bound on running time, an implementation often succeeds in verifying statements in the theory
- Many old results have been verified with this technique, and many new ones proved.
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 Berstel
The Fibonacci infinite word $f$ has a square of length $2j$ beginning at position $i$:

$$\forall t \ (0 \leq t < j) \implies f[i + t] = f[i + j + t]$$

This particular word never occurred previously in $f$:

$$\forall \ell \ (\forall t ((0 \leq t < 2j) \land f[i + t] = f[\ell + t]) \implies \ell \geq i)$$

And this particular word appears in the prefix of length $n$ of $f$:

$$i + 2j < n$$

Joining these all with “AND” gives a predicate for triples $(n, i, j)$ such that $f[i..i + 2j - 1]$ is a novel square occurring in the prefix of length $n$ in $f$. 

Predicates for various properties
5. The infinite Fibonacci word

The most famous Fibonacci-automatic word is the Fibonacci word

$$f = 0100101001001010010100100101001001 \cdots ,$$

which can be defined in various ways.
One way is the fixed point of the morphism $$\varphi(0) = 01, \varphi(1) = 0.$$ Another way is the automaton
The infinite Fibonacci word

Yet another way is through the iteration

\[ X_1 = 1 \]
\[ X_2 = 0 \]
\[ X_n = X_{n-1}X_{n-2} \]

Note that \(|X_n| = F_n|.

The \((X_n)_{n \geq 1}\) are called the finite Fibonacci words and for \(n \geq 2\) they are all prefixes of \(f\).
Properties of the Fibonacci word have been widely studied.

- $f$ is not ultimately periodic
- $f$ contains no 4th powers (Karhumäki, 1983)
- All squares in $f$ are of order $F_n$ for $n \geq 2$, and squares of all these lengths exist (Séébold, 1985)
- There exist palindromes of all lengths in $f$ (Chuan, 1993)

All of these claims can easily be verified using our method.
Theorems about the finite Fibonacci words

- Since every finite Fibonacci word is a prefix of length $F_n$ of the infinite Fibonacci word, we can rephrase many claims about the finite Fibonacci words in terms of our logical language.

- There are two possible approaches: we can state these claims for length-$n$ prefixes and ask for which $n$ they are satisfied.

- Or we can additionally restrict $n$ in our logical language to have Fibonacci representation of the form $10^*$.
To illustrate this idea, consider one of the most famous properties of the Fibonacci words, the \textit{almost-commutative} property: letting 
\[
\eta(a_1a_2\cdots a_n) = a_1a_2\cdots a_{n-2}a_na_{n-1}
\]
be the map that interchanges the last two letters of a string of length at least 2, we have

\textbf{Theorem}

\[X_{n-1}X_n = \eta(X_nX_{n-1}) \text{ for } n \geq 2.\]

We can verify this, and prove even more, using our method.

\textbf{Theorem}

Let \(x = f[0..i-1]\) and \(y = f[0..j-1]\) for \(i > j > 1\). Then 
\[xy = \eta(yx) \text{ if and only if } i = F_n, j = F_{n-1} \text{ for } n \geq 3.\]
Proof of the almost-commutative property

Proof.
The idea is to check, for each $i > j > 1$, whether

$$f[0..i - 1]f[0..j - 1] = \eta(f[0..j - 1]f[0..i - 1]).$$

We can do this with the following predicate:

$$(i > j) \land (j \geq 2) \land (\forall t, j \leq t < i, f[t] = f[t - j]) \land (\forall s \leq j - 3 f[s] = f[s + i - j]) \land (f[j - 2] = f[i - 1]) \land (f[j - 1] = f[i - 2]).$$

The resulting automaton accepts $[1, 0][0, 1][0, 0]^+$, which corresponds to $i = F_n, j = F_{n-1}$ for $n \geq 4$. 

\qedsymbol
In many cases we can count the number $T(n)$ of length-$n$ factors of a Fibonacci-automatic sequence having a particular property $P$.

Here by “count” we mean, give an algorithm $A$ to compute $T(n)$ efficiently, that is, in time bounded by a polynomial in $\log n$.

Although finding the algorithm $A$ may not be particularly efficient, once we have it, we can compute $T(n)$ quickly.
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 our approach, we generalize the problem to consider any length-$n$ prefix of $f$.

The total number of square occurrences in $f[0..n-1]$:

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

Let $b(n)$ denote the number of occurrences of squares in $f[0..n-1]$. First, we use our method to find a DFA $M$ accepting $L_{dos}$. This (incomplete) DFA has 27 states.
Next, we compute matrices $M_0$ and $M_1$, indexed by states of $M$, such that $(M_a)_{k,l}$ counts the number of edges (corresponding to the variables $i$ and $j$) from state $k$ to state $l$ on the digit $a$ of $n$. We also compute a vector $u$ corresponding to the initial state of $M$ and a vector $v$ corresponding to the final states of $M$. This gives us the following linear representation of the sequence $b(n)$: if $x = a_1a_2 \cdots a_t$ is the Fibonacci representation of $n$, then

$$b(n) = uM_{a_1} \cdots M_{a_t} v,$$ \hspace{1cm} (2)

which, incidentally, gives a fast algorithm for computing $b(n)$ for any $n$. 
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 $27 \times 27$ array
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 $c_1, \ldots, c_8$. 
When we do so, we find

\[
\begin{align*}
    c_1 &= \frac{2}{5} \\
    c_2 &= -\frac{2}{25}\sqrt{5} - 2 \\
    c_4 &= \frac{2}{25}\sqrt{5} - 2 \\
    c_5 &= 1 \\
    c_6 &= 1 \\
    c_7 &= 0 \\
    c_8 &= 0
\end{align*}
\]

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

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

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

$$d_5 = 1, \quad d_6 = -1.$$
7. Avoidability

- Consider avoiding the pattern $xxx^R$.
- This pattern occurs, for example, in the French word * choisissiez*, with $x = is$.
- There are periodic infinite binary words avoiding this pattern, like
  \[ (01)\omega = 010101 \cdots \]
  and
  \[ (0010011011)\omega = 001001101100100110110010011011 \cdots . \]
- But are there infinite aperiodic words avoiding $xxx^R$?
Yes! Take the infinite Fibonacci word $f$ and run it through the following transducer:

![Transducer Diagram]

obtaining the infinite word

$$r = 001001101101100100110110010010011011001001001101100100100110 \cdots$$

Claim: it avoids the patterns $xxx^R$ and also $xx^Rxx^R$. 
Avoiding $xxx^R$

To prove this we use the predicate

$$\exists i \geq 0 \forall t, 0 \leq t < n \left( r[i+t] = r[i+t+n] \right) \land \left( r[i+t] = r[i+3n-1-t] \right),$$

which says that the first block of length $n$ equals the second block, and the first block equals the reverse of the the third block. The word $r$ itself is generated by an 8-state automaton:

When we run this predicate on the automaton, we get that only length $n = 0$ is accepted. So the pattern $xxx^R$ doesn’t occur. This takes about 8 seconds on a laptop and the largest intermediate automaton has 8304 states.
Other applications of the method

- The Tribonacci-automatic words: based on a recurrence 
  \( T_n = T_{n-1} + T_{n-2} + T_{n-3} \).

- The paperfolding words: we can prove theorems about uncountably many different sequences simultaneously!

- The Sturmian words: modulo a few details which need to be proven, Luke Schaeffer can show that there is a decidable theory for these words, too.