Runs in Paperfolding Sequences

Jeffrey Shallit*
School of Computer Science
University of Waterloo
Waterloo, ON N2L 3G1
Canada
shallit@uwaterloo.ca

December 25, 2024

Abstract

The paperfolding sequences form an uncountable class of infinite sequences over the alphabet $\{-1, 1\}$ that describe the sequence of folds arising from iterated folding of a piece of paper, followed by unfolding. In this note we observe that the sequence of run lengths in such a sequence, as well as the starting and ending positions of the n'th run, is 2-synchronized and hence computable by a finite automaton. As a specific consequence, we obtain the recent results of Bunder, Bates, and Arnold, in much more generality, via a different approach. We also prove results about the critical exponent and subword complexity of these run-length sequences.

1 Introduction

Paperfolding sequences are sequences over the alphabet $\{-1, 1\}$ that arise from the iterated folding of a piece of paper, introducing a hill $(+1)$ or valley (-1) at each fold. They are admirably discussed, for example, in [8, 9].

The formal definition of a paperfolding sequence is based on a (finite or infinite) sequence of *unfolding instructions* $\mathbf{f}$. For finite sequences $\mathbf{f}$ we define

$$P_\epsilon = \epsilon$$
$$P_{\mathbf{f}a} = (P_\mathbf{f})\ a\ (-P_\mathbf{f}^R) \quad (1)$$

for $a \in \{-1, 1\}$ and $\mathbf{f} \in \{-1, 1\}^*$. Here ϵ denotes the empty sequence of length 0, $-x$ changes the sign of each element of a sequence x, and x^R reverses the order of symbols in a sequence

*Research supported by a grant from NSERC, 2024-03725.

x. An easy induction now shows that $|P_{\mathbf{f}}| = 2^{|\mathbf{f}|} - 1$, where $|x|$ means the length, or number of symbols, of a sequence x.

Now let $\mathbf{f} = f_0 f_1 f_2 \cdots$ be an infinite sequence in $\{-1, 1\}^\omega$. It is easy to see that $P_{f_0 f_1 \cdots f_n}$ is a prefix of $P_{f_0 f_1 \cdots f_{n+1}}$ for all $n \geq 0$, so there is a unique infinite sequence of which all the $P_{f_0 f_1 \cdots f_n}$ are prefixes; we call this infinite sequence $P_{\mathbf{f}}$.

As in the previous paragraph, we always index the unfolding instructions starting at 0: $\mathbf{f} = f_0 f_1 f_2 \cdots$. Also by convention the paperfolding sequence itself is indexed starting at 1: $P_{\mathbf{f}} = p_1 p_2 p_3 \cdots$. With these conventions we immediately see that $P_{\mathbf{f}}[2^n] = p_{2^n} = f_n$ for $n \geq 0$. Since there are a countable infinity of choices between -1 and 1 for each unfolding instructions, there are uncountably many infinite paperfolding sequences.

As an example let us consider the most famous such sequence, the *regular paperfolding sequence*, where the sequence of unfolding instructions is $1^\omega = 111 \cdots$. Here we have, for example,

$$\begin{aligned} P_1 &= 1 \\ P_{11} &= 1\,1\,(-1) \\ P_{111} &= 1\,1\,(-1)\,1\,1\,(-1)\,(-1). \end{aligned}$$

The first few values of the limiting infinite paperfolding sequence $P_{1^\omega}[n]$ are given in Table 1.

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	$\cdots$
$P_{1^\omega}[n]$	1	1	-1	1	1	-1	-1	1	1	1	-1	-1	1	-1	-1	1	$\cdots$

Table 1: The regular paperfolding sequence.

The paperfolding sequences have a number of interesting properties that have been explored in a number of papers. In addition to the papers [8, 9] already cited, the reader can also see Allouche [1], Allouche and Bousquet-Mélou [4, 5], and Goč et al. [10], to name just a few.

Recently Bunder et al. [6] explored the sequence of lengths of runs of the regular paperfolding sequence, and proved some theorems about them. Here by a "run" we mean a maximal block of consecutive identical values. Runs and run-length encodings are a long-studied feature of sequences; see, for example, [11]. The run lengths R_{1111} for the finite paperfolding sequence P_{1111}, as well as the starting positions S_{1111} and ending positions E_{1111} of the n'th run, are given in Table 2.

As it turns out, however, *much* more general results, applicable to *all* paperfolding sequences, can be proven rather simply, in some cases making use of the `Walnut` theorem-prover [13]. As shown in [17], to use `Walnut` it suffices to state a claim in first-order logic, and then the prover can rigorously determine its truth or falsity.

In order to use `Walnut` to study the run-length sequences, these sequences must be computable by a finite automaton ("automatic"). Although the paperfolding sequences

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P_{1111}[n]$	1	1	−1	1	1	−1	−1	1	1	1	−1	−1	1	−1	−1
$R_{1111}[n]$	2	1	2	2	3	2	1	2							
$S_{1111}[n]$	1	3	4	6	8	11	13	14							
$E_{1111}[n]$	2	3	5	7	10	12	13	15							

Table 2: Run lengths of the regular paperfolding sequence.

themselves have this property (as shown, for example, in [10]), there is no reason, a priori, to expect that the sequence of run lengths will also have the property. For example, the sequence of runs of the Thue-Morse sequence $\mathtt{t} = 0110100110010110 \cdots$ is $12112221121 \cdots$, fixed point of the morphism $1 \rightarrow 121$, $2 \rightarrow 12221$ [3], and is known to *not* be automatic [2].

The starting and ending positions of the n'th run are integer sequences. In order to use `Walnut` to study these, we would need these sequences to be *synchronized* (see [16]); that is, there would need to be an automaton that reads the integers n and x in parallel and accepts if x is the starting (resp., ending) position of the n'th run. But there is no reason, a priori, that the starting and ending positions of the n'th run of an arbitrary automatic sequence should be synchronized. Indeed, if this were the case, and the length of runs were bounded, then the length of these runs would always be automatic, which as we have just seen is not the case for the Thue-Morse sequence.

However, as we will see, there is a single finite automaton that can compute the run sequence $R_{\mathbf{f}}$ for *all* paperfolding sequences simultaneously, and the same thing applies to the sequences $S_{\mathbf{f}}$ and $E_{\mathbf{f}}$ of starting and ending positions respectively.

In this paper we use these ideas to study the run-length sequences of paperfolding sequences, explore their critical exponent and subword complexity, and generalize the results of Bunder et al. [6] on the continued fraction of a specific real number to uncountably many real numbers.

2 Automata for the starting and ending positions of runs

We start with a basic result with a simple induction proof.

Proposition 1. *Let* $\mathbf{f}$ *be a finite sequence of unfolding instructions of length* n. *Then the corresponding run-length sequence* $R_{\mathbf{f}}$, *as well as* $S_{\mathbf{f}}$ *and* $E_{\mathbf{f}}$, *has length* 2^{n-1}.

Proof. The result is clearly true for $n = 1$. Now suppose $\mathbf{f}$ has length $n+1$ and write $\mathbf{f} = \mathbf{g}a$ for $a \in \{-1, 1\}$. For the induction step, we use Eq. (1). From it, we see that there are 2^{n-1} runs in $P_{\mathbf{g}}$ and in $-P_{\mathbf{g}}^R$. Since the last symbol of $P_{\mathbf{g}}$ is the negative of the first symbol of $-P_{\mathbf{g}}^R$, introducing a between them extends the length of one run, and doesn't affect the other. Thus we do not introduce a new run, nor combine two existing runs into one. Hence the number of runs in $P_{\mathbf{f}}$ is 2^n, as desired. □

Remark 2. Bunder et al. [6] proved the same result for the specific case of the regular paperfolding sequence.

Next, we find automata for the starting and ending positions of the runs. Let us start with the starting positions.

The desired automaton `sp` takes three inputs in parallel. The first input is a finite sequence $\mathbf{f}$ of unfolding instructions, the second is the number n written in base 2, and the third is some number x, also expressed in base 2. The automaton accepts if and only if $x = S_{\mathbf{f}}[n]$.

Normally we think of the unfolding instructions as over the alphabet $\{-1, 1\}$, but it is useful to be more flexible and also allow 0's, but only at the end; these 0's are essentially disregarded. We need this because the parallel reading of inputs requires that all three inputs be of the same length. Thus, for example, the sequences $-1, 1, 1, 0$ and $-1, 1, 1$ are considered to specify the same paperfolding sequence, while $-1, 0, 1, 1$ is not considered a valid specification.

Because we choose to let f_0 be the first symbol of the unfolding instructions, it is also useful to require that the inputs n and x mentioned above be represented with the *least-significant digit first*. In this representation, we allow an unlimited number of trailing zeros.

Finally, although we assume that $S_{\mathbf{f}}$ is indexed starting at position 1, it is useful to define $S_{\mathbf{f}}[0] = 0$ for all finite unfolding instruction sequences $\mathbf{f}$.

To find the automaton computing the starting positions of runs, we use a guessing procedure described in [17], based on a variant of the Myhill-Nerode theorem. Once a candidate automaton is guessed, we can rigorously verify its correctness with `Walnut`.

We will need one `Walnut` automaton already introduced in [17]: `FOLD`, and another one that we can define via a regular expression.

- `FOLD` takes two inputs, $\mathbf{f}$ and n. If n is in the range $1 \leq n < 2^{|\mathbf{f}|}$, then it returns the n'th term of the paperfolding sequence specified by f.

- `lnk` takes two inputs, f and x. It accepts if f is the valid code of a paperfolding sequence (that is, no 0's except at the end) and x is $2^t - 1$, where t is the length of f (not counting 0's at the end). It can be created using the `Walnut` command

  ```
  reg lnk {-1,0,1} {0,1} "([-1,1]|[1,1])*[0,0]*":
  ```

Our guessed automaton `sp` has 17 states. We must now verify that it is correct. To do so we need to verify the following things:

1. The candidate automaton `sp` computes a partial function. More precisely, for a given $\mathbf{f}$ and n, at most one input of the form $(\mathbf{f}, n, x)$ is accepted.

2. `sp` accepts $(\mathbf{f}, 0, 0)$.

3. sp accepts $(\mathbf{f}, 1, 1)$ provided $|\mathbf{f}| \geq 1$.

4. There is an x such that sp accepts $(\mathbf{f}, 2^{|\mathbf{f}|-1}, x)$.

5. sp accepts no input of the form $(\mathbf{f}, n, x)$ if $n > 2^{|\mathbf{f}|-1}$.

6. If sp accepts $(\mathbf{f}, 2^{|\mathbf{f}|-1}, x)$ then the symbols $P_{\mathbf{f}}[t]$ for $x \leq t < 2^{|\mathbf{f}|}$ are all the same.

7. Runs are nonempty: if sp accepts $(\mathbf{f}, n-1, y)$ and $(\mathbf{f}, n, z)$ then $y < z$.

8. And finally, we check that if sp accepts $(\mathbf{f}, n, x)$, then x is truly the starting position of the n'th run. This means that all the symbols from the starting position of the $(n-1)$'th run to $x-1$ are the same, and different from $P_{\mathbf{f}}[x]$.

We use the following Walnut code to check each of these. A brief review of Walnut syntax may be useful:

- ?lsd_2 specifies that all numbers are represented with the least-significant digit first, and in base 2;

- A is the universal quantifier $\forall$ and E is the existential quantifier $\exists$;

- & is logical AND, | is logical OR, W is logical NOT, => is logical implication, <=> is logical IFF, and != is inequality;

- eval expects a quoted string representing a first-order assertion with no free (unbound) variables, and returns TRUE or FALSE;

- def expects a quoted string representing a first-order assertion φ that may have free (unbound) variables, and computes an automaton accepting the representations of those tuples of variables that make φ true, which can be used later.

```
eval tmp1 "?lsd_2 Af,n ~Ex,y x!=y & $sp(f,n,x) & $sp(f,n,y)":
# check that it is a partial function
eval tmp2 "?lsd_2 Af,x $lnk(f,x) => $sp(f,0,0)":
# check that 0th run is at position 0; the lnk makes sure that
# the format of f is correct (doesn't have 0's in the middle of it.)
eval tmp3 "?lsd_2 Af,x ($lnk(f,x) & x>=1) => $sp(f,1,1)":
# check if code specifies nonempty string then first run is at position 1
eval tmp4 "?lsd_2 Af,n,z ($lnk(f,z) & z+1=2*n) => Ex $sp(f,n,x)":
# check it accepts n = 2^{|f|-1}
eval tmp5 "?lsd_2 Af,n,z ($lnk(f,z) & z+1<2*n) => ~Ex $sp(f,n,x)":
# check that it accepts no n past 2^{|f|-1}
eval tmp6 "?lsd_2 Af,n,z,x ($lnk(f,z) & 2*n=z+1 & $sp(f,n,x))
   => At (t>=x & t<z) => FOLD[f][x]=FOLD[f][t]":
# check last run is right and goes to the end of the finite
```

```
# paperfolding sequence specified by f
eval tmp7 "?lsd_2 Af,n,x,y,z ($lnk(f,z) & $sp(f,n-1,x) &
   $sp(f,n,y) & 1<=n & 2*n<=z+1) => x<y":
# check that starting positions form an increasing sequence
eval tmp8 "?lsd_2 Af,n,x,y,z,t ($lnk(f,z) & n>=2 & $sp(f,n-1,y) &
   $sp(f,n,x) & x<=z & y<=t & t<x) => FOLD[f][x]!=FOLD[f][t]":
# check that starting position code is actually right
```

`Walnut` returns `TRUE` for all of these, which gives us a proof by induction on n that indeed $x_n = S_{\mathbf{f}}[n]$.

From the automaton for starting positions of runs, we can obtain the automaton for ending positions of runs, `ep`, using the following `Walnut` code:

```
def ep "?lsd_2 Ex $lnk(f,x) & ((2*n<=x-1 & $sp(f,n+1,z+1)) |
   (2*n-1=x & z=x))":
```

Thus we have proved the following result.

Theorem 3. *There is a synchronized automaton of* 17 *states* `sp` *computing* $S_{\mathbf{f}}[n]$ *and one of* 13 *states* `ep` *computing* $E_{\mathbf{f}}[n]$*, for all paperfolding sequences simultaneously.*

Using the automaton `ep`, we are now able to prove the following new theorem. Roughly speaking, it says that the ending position of the n'th run for the unfolding instructions $\mathbf{f}$ is $2n - \epsilon_n$, where $\epsilon_n \in \{0, 1\}$, and we can compute ϵ_n by looking at a sequence of unfolding instructions closely related to $\mathbf{f}$.

Theorem 4. *Let* $\mathbf{f}$ *be a finite sequence of unfolding instructions, of length at least* 2*. Define a new sequence* $\mathbf{g}$ *of unfolding instructions as follows:*

$$\mathbf{g} := \begin{cases} 1\ (-x), & \text{if } \mathbf{f} = 11x; \\ (-1)\ (-x), & \text{if } \mathbf{f} = 1(-1)x; \\ (-1)\ x, & \text{if } \mathbf{f} = (-1)1x; \\ 1\ x, & \text{if } \mathbf{f} = (-1)(-1)x. \end{cases} \tag{2}$$

Then

$$E_{\mathbf{f}}[n] + \epsilon_n = 2n \tag{3}$$

for $1 \leq n < 2^{n-1}$*, where*

$$\epsilon_n = \begin{cases} 0, & \text{if } P_{\mathbf{g}}[n] = 1; \\ 1, & \text{if } P_{\mathbf{g}}[n] = -1. \end{cases}$$

Furthermore, if $\mathbf{f}$ *is an infinite set of unfolding instructions, then Eq. (3) holds for all* $n \geq 1$.

Proof. We prove this using `Walnut`. First, we need an automaton `assoc` that takes two inputs **f** and **g** in parallel, and accepts if **g** is defined as in Eq. (2). This automaton is depicted in Figure 1, and correctness is left to the reader. Now we use the following `Walnut` code.

```
eval thm3 "?lsd_2 Af,g,y,n,t ($lnk(g,y) & $assoc(f,g) & y>=1 &
   n<=y & n>=1 & $ep(f,n,t)) =>
   ((FOLD[g][n]=@-1 & t+1=2*n)|(FOLD[g][n]=@1 & t=2*n))":
```

And `Walnut` returns `TRUE`. □

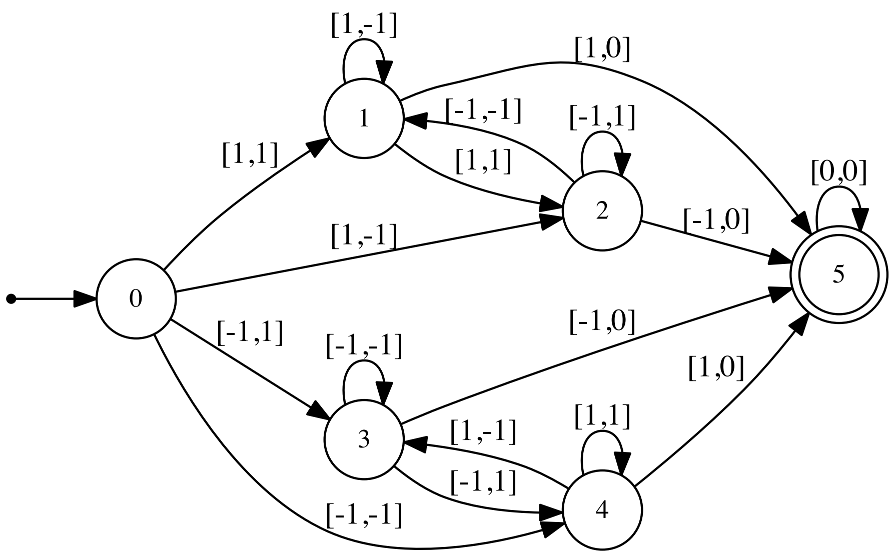

Figure 1: The automaton `assoc`.

3 Automaton for the sequence of run lengths

Next we turn to the sequence of run lengths itself. We can compute these from the automata for `ep` and `sp`.

```
def rl "?lsd_2 Ex,y $sp(f,n,x) & $ep(f,n,y) & z=1+(y-x)":
```

Proposition 5. *For all finite and infinite sequences of paperfolding instructions, the only run lengths are* 1, 2, *or* 3.

Proof. It suffices to prove this for the finite paperfolding sequences.

```
def prop4 "?lsd_2 Af,n,x,z ($lnk(f,x) & 1<=n & 2*n<=x+1
   & $rl(f,n,z)) => (z=1|z=2|z=3)":
```

And Walnut returns TRUE. □

Remark 6. Proposition 5 was proved by Bunder et al. [6] for the specific case of the regular paperfolding sequence.

We now use another feature of Walnut, which is that we can turn a synchronized automaton computing a function of finite range into an automaton returning the value of the function. The following code

```
def rl1 "?lsd_2 $rl(f,n,1)":
def rl2 "?lsd_2 $rl(f,n,2)":
def rl3 "?lsd_2 $rl(f,n,3)":
combine RL rl1=1 rl2=2 rl3=3:
```

computes an automaton RL of two inputs **f** and n, and returns the value of the run-length sequence at index n (either 1, 2, or 3) for the unfolding instructions **f**. This automaton has 31 states.

Recall that an *overlap* is a string of the form $axaxa$, where a is a single letter, and x is a possibly empty string. For example, the word entente is an overlap from French. We now prove that the sequence of run lengths in a paperfolding sequence contains no overlaps.

Theorem 7. *The sequence of run lengths corresponding to every finite or infinite paperfolding sequence is overlap-free.*

Proof. It suffices to prove the result for every finite paperfolding sequence. We can do this is as follows:

```
def chk_over "?lsd_2 ~Ef,i,n,x $lnk(f,x) & x>=1 & i>=1 & n>=1
   & i+2*n<=(x+1)/2 & At (t<=n) => RL[f][i+t]=RL[f][i+n+t]":
# asserts no overlaps
```

And Walnut returns TRUE. □

We now consider squares, that is, blocks of the form zz, where z is a nonempty sequence.

Theorem 8. *The only possible squares occurring in the run lengths of a paperfolding sequence are* 22*,* 123123*, and* 321321*.*

Proof. We start by showing that the only squares are of order 1 or 3.

```
def chk_sq1 "?lsd_2 Af,i,n,x ($lnk(f,x) & x>=1 & i>=1 & n>=1
   & i+2*n-1<=(x+1)/2 & At (t<n) => RL[f][i+t]=RL[f][i+n+t]) => (n=1|n=3)":
```

Next we check that the only square of order 1 is 22.

```
def chk_sq2 "?lsd_2 Af,x,i ($lnk(f,x) & x>=1 & i>=1 &
   i+1<=(x+1)/2 & RL[f][i]=RL[f][i+1]) => RL[f][i]=@2":
```

Finally, we check that the only squares of order 3 are 123123 and 321321.

```
def chk_sq3 "?lsd_2 Af,x,i ($lnk(f,x) & x>=1 & i>=1 &
   i+5<=(x+1)/2 & RL[f][i]=RL[f][i+3] & RL[f][i+1]=RL[f][i+4]
   & RL[f][i+2]=RL[f][i=5]) => ((RL[f][i]=@1 & RL[f][i+1]=@2
   & RL[f][i+2]=@3)|(RL[f][i]=@3 & RL[f][i+1]=@2 & RL[f][i+2]=@1))":
```

□

Proposition 9. *In every finite paperfolding sequence formed by* 7 *or more unfolding instructions, the squares* 22, 123123, *and* 321321 *are all present in the run-length sequence.*

We now turn to palindromes.

Theorem 10. *The only palindromes that can occur in the run-length sequence of a paperfolding sequence are* $1, 2, 3, 22, 212, 232, 12321,$ *and* 32123.

Proof. It suffices to check the factors of the run-length sequences of length at most 7. These correspond to factors of length at most $2+3\cdot 7 = 23$, and by the bounds on the "appearance" function given in Theorem [17, Thm 12.2.2], to guarantee we have seen all of these factors, it suffices to look at prefixes of paperfolding sequences of length at most $13 \cdot 23 = 299$. (Also see [7].) Hence it suffices to look at all 2^9 finite paperfolding sequences of length $2^9 - 1 = 511$ specified by instructions of length 9. When we do this, the only palindromes we find are those in the statement of the theorem. □

Theorem 11. *The subword complexity of the run-length sequence of an infinite paperfolding sequence is* $4n + 4$ *for* $n \geq 6$.

Proof. First we prove that if x is a factor of a run-length sequence, and $|x| \geq 2$, then xa is a factor of the same sequence for at most two different a.

```
def faceq "?lsd_2 At (t<n) => RL[f][i+t]=RL[f][j+t]":
eval three "?lsd_2 Ef,i,j,k,n n>=2 & i>=1 & RL[f][i+n]=@1 &
   RL[f][j+n]=@2 & RL[f][k+n]=@3 & $faceq(f,i,j,n) & $faceq(f,j,k,n)":
```

Next we prove that if $|x| \geq 5$, then exactly four factors of a run-length sequence are right-special (have an extension by two different letters).

```
def rtspec "?lsd_2 Ej,x $lnk(f,x) & i+n<=x & i>=1 &
   $faceq(f,i,j,n) & RL[f][i+n]!=RL[f][j+n]":
eval nofive "?lsd_2 ~Ef,i,j,k,l,m,n n>=5 & i<j & j<k & k<l
   & l<m & $rtspec(f,i,n) & $rtspec(f,j,n) & $rtspec(f,k,n) &
   $rtspec(f,l,n) & $rtspec(f,m,n)":
eval four "?lsd_2 Af,n,x ($lnk(f,x) & x>=127 & n>=6 &
   13*n<=x) => Ei,j,k,l i>=1 & i<j & j<k & k<l &
   $rtspec(f,i,n) & $rtspec(f,j,n) & $rtspec(f,k,n) & $rtspec(f,l,n)":
```

Here `nofive` shows that no length 5 or larger has five or more right-special factors of that length, and every length 6 or larger has exactly four such right-special factors. Here we have used [17, Thm. 12.2.2], which guarantees that every factor of length n of a paperfolding sequence can be found in a prefix of length $13n$.

Since there are 28 factors of every run-length sequence of length 6 (which we can check just by enumerating them, again using [17, Thm. 12.2.2]), the result now follows. □

4 The regular paperfolding sequence

In this section we specialize everything we have done so far to the case of a single infinite paperfolding sequence, the so-called regular paperfolding sequence, where the folding instructions are $1^\omega = 111\cdots$. In [6], the sequence $2122321231232212\cdots$ of run lengths $2, 3, 5, 7, 10, 12, 13, 15, 18, 19, 21, \ldots$ for the regular paperfolding sequence was called $g(n)$, and the sequence of ending positions of runs was called $h(n)$. We adopt their notation. Note that $g(n)$ forms sequence A088431 in the On-Line Encyclopedia of Integer Sequences (OEIS) [18].

In this case we can compute an automaton computing the n'th term of the run length sequence $g(n)$ as follows:

```
reg rps {-1,0,1} {0,1} "[1,1]*[0,0]*":
def runlr1 "?lsd_2 Ef,x $rps(f,x) & n>=1 & n<=x/2 & RL[f][n]=@1":
def runlr2 "?lsd_2 Ef,x $rps(f,x) & n>=1 & n<=x/2 & RL[f][n]=@2":
def runlr3 "?lsd_2 Ef,x $rps(f,x) & n>=1 & n<=x/2 & RL[f][n]=@3":
combine RLR runlr1=1 runlr2=2 runlr3=3:
```

The resulting automaton is depicted in Figure 2.

Casual inspection of this automaton immediately proves many of the results of [6], such as their multi-part Theorems 2.1 and 2.2. To name just one example, the sequence $g(n)$ takes the value 1 iff $n \equiv 2, 7 \pmod 8$. For their other results, we can use `Walnut` to prove them.

We can also specialize `sp` and `ep` to the case of the regular paperfolding sequence, as follows:

```
reg rps {-1,0,1} {0,1} "[1,1]*[0,0]*":
def sp_reg "?lsd_2 (n=0&z=0) | Ef,x $rps(f,x) & n>=1 & n<=x/2 & $sp(f,n,z)":
def ep_reg "?lsd_2 (n=0&z=0) | Ef,x $rps(f,x) & n>=1 & n<=x/2 & $ep(f,n,z)":
```

These automata are depicted in Figures 3 and 4.

Once we have these automata, we can easily recover many of the results of [6], such as their Theorem 3.2. For example they proved that if $n \equiv 1 \pmod 4$, then $h(n) = 2n$. We can prove this as follows with `Walnut`:

```
eval test32a "?lsd_2 An (n=4*(n/4)+1) => $ep_reg(n,2*n)":
```

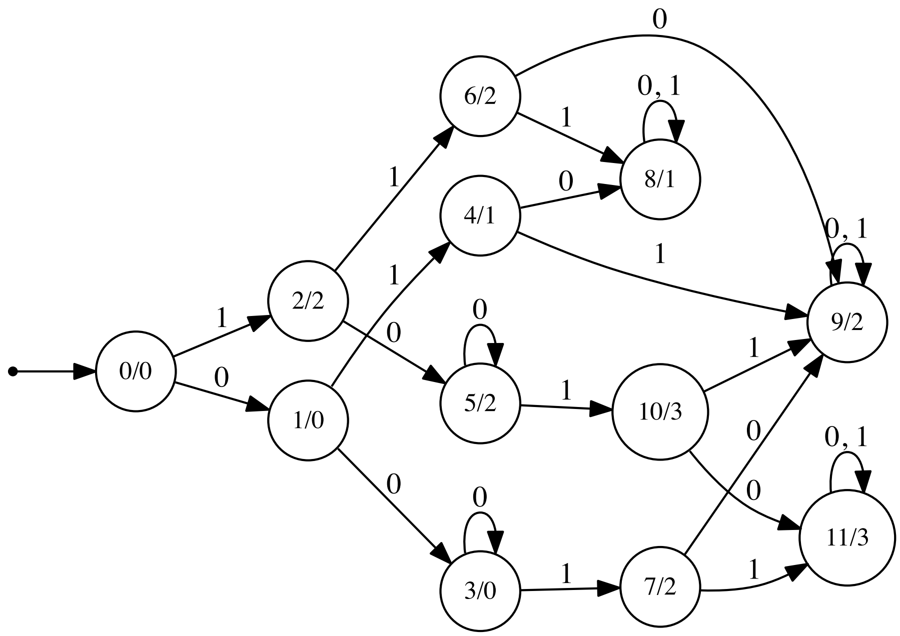

Figure 2: The lsd-first automaton `RLR`.

The reader may enjoy constructing `Walnut` expressions to check the other results of [6].

Slightly more challenging to prove is the sum property, conjectured by Hendriks, and given in [6, Thm. 4.1]. We state it as follows:

Theorem 12. *Arrange the set of positive integers not in* $H := \{h(n) + 1 \ : \ n \geq 0\}$ *in increasing order, and let* $t(n)$ *be the n'th such integer, for* $n \geq 1$*. Then*

(a) $g(h(i) + 1) = 2$ *for* $i \geq 0$*;*

(b) $g(t(2i)) = 3$ *for* $i \geq 1$*;*

(c) $g(t(2i - 1)) = 1$ *for* $i \geq 1$*.*

Proof. The first step is to create an automaton `tt` computing $t(n)$. Once again, we guess the automaton from data and then verify its correctness. It is depicted in Figure 5.

In order to verify its correctness, we need to verify that `tt` indeed computes a increasing function $t(n)$ and further that the set $\{t(n) \ : \ n \geq 1\} = \{1, 2, \ldots, \} \setminus H$. We can do this as follows:

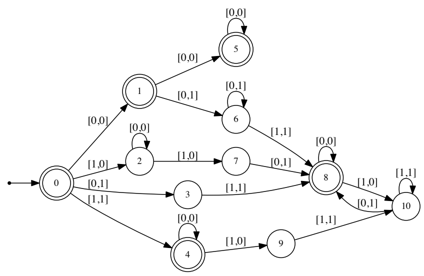

Figure 3: Synchronized automaton `sp_reg` for starting positions of runs of the regular paperfolding sequence.

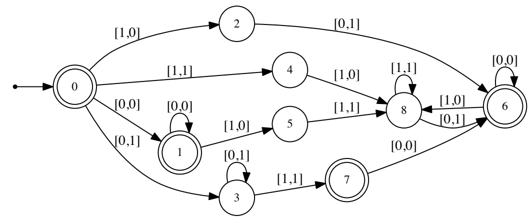

Figure 4: Synchronized automaton `ep_reg` for ending positions of runs of the regular paperfolding sequence.

```
eval tt1 "?lsd_2 An (n>=1) => Ex $tt(n,x)":
# takes a value for all n
eval tt2 "?lsd_2 ~En,x,y n>=1 & x!=y & $tt(n,x) & $tt(n,y)":
# does not take two different values for the same n
eval tt3 "?lsd_2 An,y,z (n>=1 & $tt(n,y) & $tt(n+1,z)) => y<z":
# is an increasing function
eval tt4 "?lsd_2 Ax (x>=1) =>
   ((En n>=1 & $tt(n,x)) <=> (~Em,y $ep_reg(m,y) & x=y+1))":
# takes all values not in H
```

Now we can verify parts (a)-(c) as follows:

```
eval parta "?lsd_2 Ai,x (i>=1 & $ep_reg(i,x)) => RLR[x+1]=@2":
eval partb "?lsd_2 Ai,x (i>=1 & $tt(2*i,x)) => RLR[x]=@3":
eval partc "?lsd_2 Ai,x (i>=1 & $tt(2*i-1,x)) => RLR[x]=@1":
```

And `Walnut` returns `TRUE` for all of these. This completes the proof. □

5 Connection with continued fractions

Dimitri Hendriks observed, and Bunder et al. [6] proved, a relationship between the sequence of runs for the regular paperfolding sequence, and the continued fraction for the real number $\sum_{i\geq 0} 2^{-2^i}$.

As it turns out, however, a *much* more general result holds; it links the continued fraction for uncountably many irrational numbers to runs in the paperfolding sequences.

Theorem 13. *Let $n \geq 2$ and $\epsilon_i \in \{-1, 1\}$ for $2 \leq i \leq n$. Define*

$$\alpha(\epsilon_2, \epsilon_3, \ldots, \epsilon_n) := \frac{1}{2} + \frac{1}{4} + \sum_{2 \leq i \leq n} \epsilon_i 2^{-2^i}.$$

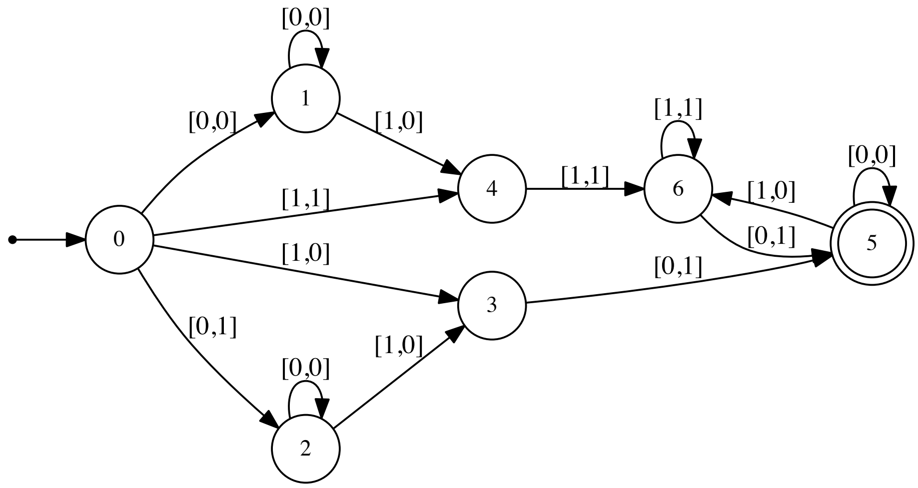

Figure 5: The automaton `tt` computing $t(n)$.

Then the continued fraction for $\alpha(\epsilon_2, \epsilon_3, \ldots, \epsilon_n)$ *is given by* $[0, 1, (2R_{1,\epsilon_2,\epsilon_3,\ldots,\epsilon_n})']$, *where the prime indicates that the last term is increased by* 1*.*

As a consequence, we get that the numbers $\alpha(\epsilon_2, \epsilon_3, \ldots)$ *have continued fraction given by* $[0, 1, 2R_{1,\epsilon_2,\epsilon_3,\ldots}]$.

Remark 14. These numbers were proved transcendental by Kempner [12]. They are sometimes erroneously called Fredholm numbers, even though Fredholm never studied them.

As an example, suppose $(\epsilon_2, \epsilon_3, \epsilon_4, \epsilon_5) = (1, -1, -1, 1)$. Then

$$x(1, -1, -1, 1) = 3472818177/2^{32} = [0, 1, 4, 4, 2, 6, 4, 2, 4, 4, 6, 4, 2, 4, 6, 2, 4, 5],$$

while $R_{1,1,-1,-1,1} = 2213212232123122$.

To prove Theorem 13, we need the "folding lemma":

Lemma 15. *Suppose* $p/q = [0, a_1, a_2, \ldots, a_t]$, t *is odd, and* $\epsilon \in \{-1, 1\}$. *Then*

$$p/q + \epsilon/q^2 = [0, a_1, a_2, \ldots, a_{t-1}, a_t - \epsilon, a_t + \epsilon, a_{t-1}, \ldots, a_2, a_1].$$

Proof. See [9, p. 177], although the general ideas can also be found in [14, 15]. □

We can now prove Theorem 13.

Proof. From Lemma 15 we see that if $\alpha(\epsilon_2, \epsilon_3, \ldots, \epsilon_n) = [0, 1, a_2, \ldots, a_t]$ then

$$\alpha(\epsilon_2, \epsilon_3, \ldots, \epsilon_n, \epsilon_{n+1}) = [0, 1, a_2, \ldots, a_{t-1}, a_t - \epsilon_{n+1}, a_t + \epsilon_{n+1}, a_{t-1}, a_{t-2}, \ldots, a_3, a_2 + 1].$$

Now $F_{1,\epsilon_2,\epsilon_3,\ldots,\epsilon_n}$ always ends in -1. Write $R_{1,\epsilon_2,\epsilon_3,\ldots,\epsilon_n} = b_1 b_2 \cdots b_t$. Then

$$R_{1,\epsilon_2,\ldots,\epsilon_n,\epsilon_{n+1}} = b_1 \cdots b_{t-1}, b_t + 1, b_{t-1}, \ldots, b_1$$

if $\epsilon_{n+1} = -1$ (because we extend the last run with one more -1) and

$$R_{1,\epsilon_2,\ldots,\epsilon_n,\epsilon_{n+1}} = b_1 \cdots b_{t-1}, b_t, b_t + 1, b_{t-1}, \ldots, b_1$$

if $\epsilon_{n+1} = 1$.

Suppose

$$\begin{aligned}\alpha(\epsilon_2, \epsilon_3, \ldots, \epsilon_n) &= [0, 1, (2R_{1,\epsilon_2,\epsilon_3,\ldots,\epsilon_n})'] \\ &= [0, 1, a_2, \ldots, a_t],\end{aligned}$$

and let $R_{1,\epsilon_2,\ldots,\epsilon_n} = b_1 b_2 \cdots b_{t-1}$. Then

$$\begin{aligned}\alpha(\epsilon_2, \epsilon_3, \ldots, \epsilon_n, \epsilon_{n+1}) &= [0, 1, a_2, \ldots, a_{t-1}, a_t - \epsilon_{n+1}, a_t + \epsilon_{n+1}, a_{t-1}, \ldots, a_3, a_2 + 1] \\ &= [0, 1, 2b_1, \ldots, 2b_{t-2}, 2b_{t-1} + 1 - \epsilon_{n+1}, 2b_{t-1} + 1 + \epsilon_{n+1}, 2b_{t-2}, \ldots, 2b_2, 2b_1, 1] \\ &= [0, 1, 2b_1, \ldots, 2b_{t-2}, 2b_{t-1} + 1 - \epsilon_{n+1}, 2b_{t-1} + 1 + \epsilon_{n+1}, 2b_{t-2}, \ldots, 2b_2, 2b_1 + 1] \\ &= [0, 1, 2R'_{1,\epsilon_2,\ldots,\epsilon_{n+1}}],\end{aligned}$$

as desired. □

6 Acknowledgments

I thank Jean-Paul Allouche and Narad Rampersad for their helpful comments.

References

[1] J.-P. Allouche. The number of factors in a paperfolding sequence. *Bull. Aust. Math. Soc.* **46** (1992), 23–32.

[2] G. Allouche, J.-P. Allouche, and J. Shallit. Kolams indiens, dessins sur le sable aux îles Vanuatu, courbe de Sierpinski, et morphismes de monoïde. *Annales Inst. Fourier* **56** (2006), 2115–2130.

[3] J.-P. Allouche, A. Arnold, J. Berstel, S. Brlek, W. Jockusch, S. Plouffe, and B. E. Sagan. A relative of the Thue-Morse sequence. *Discrete Math.* **139** (1995), 455–461.

[4] J.-P. Allouche and M. Bousquet-Mélou. Canonical positions for the factors in the paperfolding sequences. *Theoret. Comput. Sci.* **129** (1994), 263–278.

[5] J.-P. Allouche and M. Bousquet-Mélou. Facteurs des suites de Rudin-Shapiro généralisées. *Bull. Belgian Math. Soc.* **1** (1994), 145–164.

[6] M. Bunder, B. Bates, and S. Arnold. The summed paperfolding sequence. *Bull. Aust. Math. Soc.* **110** (2024), 189–198.

[7] R. Burns. The appearance function for paper-folding words. ArXiv preprint arXiv:2210.14719 [math.NT], October 22 2022. Available at https://arxiv.org/abs/2210.14719.

[8] C. Davis and D. E. Knuth. Number representations and dragon curves–I, II. *J. Recreational Math.* **3** (1970), 66–81, 133–149.

[9] F. M. Dekking, M. Mendès France, and A. J. van der Poorten. Folds! *Math. Intelligencer* **4** (1982), 130–138, 173–181, 190–195. Erratum, **5** (1983), 5.

[10] D. Goč, H. Mousavi, L. Schaeffer, and J. Shallit. A new approach to the paperfolding sequences. In A. Beckmann et al., eds., *CiE 2015*, Lecture Notes in Comput. Sci., Vol. 9136, Springer, 2015, pp. 34–43.

[11] S. W. Golomb. Run-length encodings. *IEEE Trans. Info. Theory* **IT-12** (1966), 399–401.

[12] A. J. Kempner. On transcendental numbers. *Trans. Amer. Math. Soc.* **17** (1916), 476–482.

[13] H. Mousavi. Automatic theorem proving in `Walnut`. Arxiv preprint arXiv:1603.06017 [cs.FL], available at http://arxiv.org/abs/1603.06017, 2016.

[14] J. O. Shallit. Simple continued fractions for some irrational numbers. *J. Number Theory* **11** (1979), 209–217.

[15] J. O. Shallit. Explicit descriptions of some continued fractions. *Fibonacci Quart.* **20** (1982), 77–81.

[16] J. Shallit. Synchronized sequences. In T. Lecroq and S. Puzynina, eds., *WORDS 2021*, Lecture Notes in Comp. Sci., Vol. 12847, Springer, 2021, pp. 1–19.

[17] J. Shallit. *The Logical Approach To Automatic Sequences: Exploring Combinatorics on Words with* `Walnut`, Vol. 482 of *London Math. Soc. Lecture Note Series*. Cambridge University Press, 2023.

[18] N. J. A. Sloane et al. The On-Line Encyclopedia of Integer Sequences. Available at https://oeis.org.