Proof of Irvine's Conjecture via Mechanized Guessing

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

October 24, 2023

Abstract

We prove a recent conjecture of Sean A. Irvine about a nonlinear recurrence, using mechanized guessing and verification. The theorem-prover `Walnut` plays a large role in the proof.

Key words: Irvine's conjecture, Gutkovskiy's sequence, numeration system, morphism, automaton, `Walnut`, nonlinear recurrence, automatic sequence, combinatorial game, subword complexity, critical exponent.

1 Introduction

Mathematicians have long used intelligent guessing of a problem's solution, followed by rigorous verification (for example, by induction), to prove theorems. In this note I show how to do this, at least in some cases, using a simple algorithm to infer a finite automaton from empirical data. Once a candidate automaton is inferred, a rigorous proof of its correctness can be supplied by using `Walnut`, a theorem-prover for automatic sequences [8, 10].

On May 24 2017 Ilya Gutkovskiy proposed the following nonlinear recurrence as sequence A286389 in the OEIS (On-Line Encyclopedia of Integer Sequences) [11]:

$$g_n = \begin{cases} 0, & \text{if } n = 0; \\ n - g_{\lfloor g_{n-1}/2 \rfloor}, & \text{otherwise.} \end{cases} \tag{1}$$

*Research supported by NSERC grant 2018-04118.

The first few values of this sequence, which we call Gutkovskiy's sequence, are given in Table 1. This recurrence is a variation on similar sequences originally discussed by Hofstadter [7, p. 137].

Then, on July 20 2022, Sean A. Irvine observed that this sequence seemed to be given by the partial sums of the sequence A285431, which is the fixed point of the morphism h, where $h(1) = 110$ and $h(0) = 11$. We denote the sequence A285431 by $(k_n)_{n\geq 1}$, in honor of its proposer, Clark Kimberling. The first few values of the sequence A285431 are also given in Table 1; In order to maintain the indexing given in the OEIS, we define $k_0 = 0$. More precisely, then, Irvine's conjecture is that $g_n = \sum_{1\leq i\leq n} k_i$.

n	0	1	2	3	4	5	6	7	8	9	10	11
g_n	0	1	2	2	3	4	4	5	6	7	8	8
k_n	0	1	1	0	1	1	0	1	1	1	1	0

Table 1: First few values of g_n.

In this note we prove Irvine's conjecture, as well as a number of related results.

All the needed `Walnut` code to verify the claims of the paper is available on the author's website, https://cs.uwaterloo.ca/~shallit/papers.html.

2 From a morphism to a numeration system

We start with the morphism $h : 1 \to 110,\ 0 \to 11$ that generates OEIS sequence A285431. Define $K_n = h^n(1)$, so that $K_0 = 1$, $K_1 = 110$, $K_2 = 11011011$, and so forth.

Proposition 1. *For $n \geq 2$ we have $K_n = K_{n-1}K_{n-1}K_{n-2}K_{n-2}$.*

Proof. By induction on n. The base cases of $n = 0, 1$ are trivial. Otherwise assume $n \geq 2$. Then

$$\begin{aligned} K_n &= h^n(1) = h^{n-1}(h(1)) = h^{n-1}(1)h^{n-1}(1)h^{n-1}(0) \\ &= K_{n-1}K_{n-1}h^{n-2}(11) = K_{n-1}K_{n-1}K_{n-2}K_{n-2}. \end{aligned}$$

□

Since each K_i is the prefix of K_{i+1}, it follows that there is a unique limiting infinite word $\mathbf{k} = k_1k_2k_3 \cdots = 1101101111 \cdots$ of which all the K_i are prefixes. Furthermore, Proposition 1 shows that $\mathbf{k}$ is a "generalized automatic sequence" as studied in [9], and hence there is a numeration system associated with it, where k_n can be computed by a finite automaton taking, as inputs, the representation of n in this numeration system.

We now explain how this is done. Define $\mathcal{K}_n = |K_n|$, so that $\mathcal{K}_0 = 1$, $\mathcal{K}_1 = 3$, $\mathcal{K}_2 = 8$, and in general $\mathcal{K}_n = 2\mathcal{K}_{n-1} + 2\mathcal{K}_{n-2}$. This two-term linear recurrence is sequence A028859 in the OEIS (and also A155020 shifted by one).

We now build a numeration system, which we call *K-representation*, out of the sequence $(\mathcal{K}_i)_{i\geq 0}$. We represent every natural number as a sum $\sum_{0\leq i\leq t} a_i\mathcal{K}_i$, where $a_i \in \Sigma_3 := \{0,1,2\}$. Furthermore we associate a ternary word $a_t \cdots a_0$ with the corresponding sum, as follows: $[a_t \cdots a_0]_K = \sum_{0\leq i\leq t} a_i\mathcal{K}_i$. Notice that words are written "backwards" so the most significant digit is at the left.

Evidently numbers could have multiple representations in this system as we have described it so far. For example $[22]_K = 8 = [100]_K$. In order to get a unique, canonical representation, we impose the restriction $a_i a_{i+1} \neq 22$. This is in analogy with a similar restriction for the Zeckendorf (or Fibonacci) numeration system. We let $(n)_K$ denote this canonical representation for n. Table 2 gives the first few representations in this numeration system. Notice that the canonical representation for 0 is ϵ, the empty string.

n	$(n)_K$
0	ϵ
1	1
2	2
3	10
4	11
5	12
6	20
7	21
8	100
9	101
10	102

Table 2: Representation for the first few numbers.

It is now easy to see that the greedy algorithm produces the canonical representation [4]. Furthermore, it is easy to see that there is a finite automaton that takes, as input, a string x over the alphabet Σ_3, and accepts if and only if x is a canonical representation. It is depicted in Figure 1. (We routinely omit useless states without comment.)

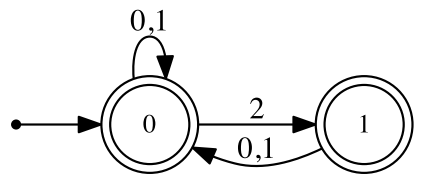

Figure 1: Automaton accepting canonical representations.

Some of the sequences we study in this paper were previously studied by Fraenkel and co-authors [5, 1], in the context of some variations on Wythoff's game. These authors already found the numeration system we described here. Also see [3]. Our main contribution is to combine the use of automata theory with the numeration system.

3 An incrementer automaton for K-representations

We claim that we can go from the K-representation of n to that of $n+1$ as follows: if the last digit is 0, add one to it. If the last digit is 1, add one to it, except in the case that the representation ends with $a(21)^i$, for $a \in \{0,1\}$, in which case the representation of $n+1$ ends in $(a+1)0^{2i}$ instead. If the last two digits are $a2$, for $a \in \{0,1\}$, then the last two digits of $n+1$ are $(a+1)0$. Verification of this is straightforward and is left to the reader.

A synchronized automaton 'incr' implementing these rules is depicted in Figure 2. The meaning of "synchronized" here is that the DFA takes the canonical K-representations of n and x in parallel as input, and accepts if $x = n+1$.

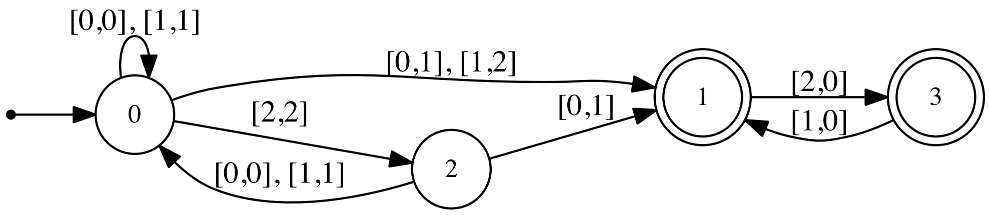

Figure 2: Incrementer automaton for K-representations.

4 An adder automaton for K-representation

The next step is to build an "adder" for K-representations. This is a synchronized automaton that takes, in parallel, the canonical K-representations of integers x, y, z, and accepts if and only if $x + y = z$. The existence of this automaton for our numeration system follows from very general results of Frougny and Solomyak [6].

However, in this case it is actually easier to just "guess" the automaton from empirical data, and then verify its correctness. The method of guessing is based on the Myhill-Nerode theorem from formal language theory, and is explained, for example, in [10].

Once we have an automaton that we believe is an adder, we can verify its correctness by induction by checking the following conditions.

(i) $\forall x, y \; \exists z \;\; \text{add}(x, y, z)$ (adder is well-defined)

(ii) $\forall x, y, z, w\ (\text{add}(x, y, z) \wedge \text{add}(x, y, w)) \implies z = w$ (adder represents a function)

(iii) $\forall x, y, z\ \ \text{add}(x, y, z) \iff \text{add}(y, x, z)$ (commutative law)

(iv) $\forall x, y, z, t\ (\exists r\ \ \text{add}(x, y, r) \wedge \text{add}(r, z, t)) \iff (\exists s\ \ \text{add}(y, z, s) \wedge \text{add}(x, s, t))$ (associative law)

(v) $\forall x\ \ \text{add}(x, 0, x)$ (base case of induction)

(vi) $\forall x, y\ \ \text{add}(x, 1, y) \iff \text{incr}(x, y)$ (induction step).

Our candidate adder had 42 states. To verify its correctness, we use the following straightforward implementation of the conditions above.

```
eval check_i "?msd_kim Ax,y Ez $add(x,y,z)":
eval check_ii "?msd_kim Ax,y,z,w ($add(x,y,z) & $add(x,y,w)) => z=w":
eval check_iii "?msd_kim Ax,y,z $add(x,y,z) <=> $add(y,x,z)":
eval check_iv "?msd_kim Ax,y,z,t (Er $add(x,y,r) & $add(r,z,t)) <=>
   (Es $add(y,z,s) & $add(x,s,t))":
eval checkv "?msd_kim Ax $add(x,0,x)":
eval checkvi "?msd_kim Ax,y $add(x,1,y) <=> $incr(x,y)":
```

and `Walnut` returns `TRUE` for all six statements. The correctness of the adder now follows.

We briefly comment on the syntax of `Walnut` commands. Here `A` and `E` represent the universal and existential quantifiers $\forall$ and $\exists$, respectively. The jargon `?msd_kim` means to interpret the statements using the K-numeration system. The symbol `&` means logical "and", `|` means logical "or", `~` is logical negation, `=>` is implication, and `<=>` represents iff. The command `def` defines an automaton, `eval` evaluates truth or falsity, and `reg` converts a regular expression to an automaton.

5 The Kimberling sequence

Define $k'_n = k_{n+1}$ for $n \geq 0$. It is now easy to create a DFAO (deterministic finite automaton with output) computing the sequence $(k'_n)_{n\geq 0}$, by associating states of the DFAO with letters of the alphabet, and transitions with images of those letters, as explained in [9]. It is depicted in Figure 3. This DFAO takes a canonical K-representation of n as input, and outputs (as the last state reached) the value of k'_n. In `Walnut` this is represented by the file `KP.txt`, as follows:

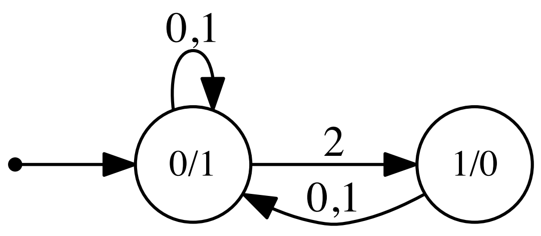

Figure 3: DFAO computing k'_n.

```
msd_kim

0 1
0 -> 0
1 -> 0
2 -> 1

1 0
0 -> 0
1 -> 0
```

Once we have this DFAO, we can get a DFAO for $(k_n)_{n\geq 0}$ simply by shifting the index.

```
def kks "?msd_kim KP[n-1]=@1":
combine K kks:
```

The resulting DFAO is depicted in Figure 4.

We can now verify that this automaton actually does compute the Kimberling sequence. We can do this by induction, by verifying that

$$\mathbf{k}[1..\mathcal{K}_n] = \mathbf{k}[1..\mathcal{K}_{n-1}]\ \mathbf{k}[1..\mathcal{K}_{n-1}]\ \mathbf{k}[1..\mathcal{K}_{n-2}]\ \mathbf{k}[1..\mathcal{K}_{n-2}].$$

To do so, we use the following `Walnut` code:

```
reg isk msd_kim "0*10*":
reg pair msd_kim msd_kim "[0,0]*[1,0][0,1][0,0]*":
eval checkk1 "?msd_kim At,x ($isk(x) & t>=1 & t<=x) => K[t+x]=K[t]":
eval checkk2 "?msd_kim At,x,y ($pair(x,y) & t>=1 & t<=y) => K[t+2*x]=K[t]":
eval checkk3 "?msd_kim At,x,y ($pair(x,y) & t>=1 & t<=y) => K[t+2*x+y]=K[t]":
```

Here isk(x) asserts that $x = \mathcal{K}_n$ for some $n \geq 1$, and pair(x, y) asserts that $x = \mathcal{K}_{n+1}$ and $y = \mathcal{K}_n$ for some $n \geq 1$.

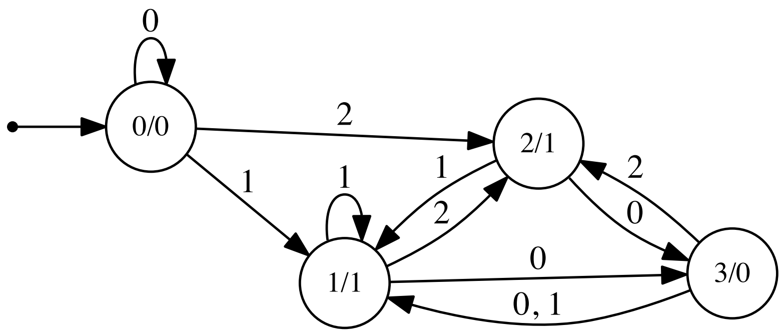

Figure 4: DFAO for the sequence $\mathbf{k}$.

6 Synchronized automaton for Gutkovskiy's sequence

The last piece of the puzzle we need is a synchronized DFA computing Gutkovskiy's sequence A286389. To find this automaton we once again guess it from empirical data, and then verify it using Eq. (1).

The guessed 17-state automaton is called 'gut'. To verify its correctness we use the following `Walnut` code:

```
eval check1 "?msd_kim An Ex $gut(n,x)":
eval check2 "?msd_kim An,x,y ($gut(n,x) & $gut(n,y)) => x=y":
eval check3 "?msd_kim $gut(0,0) & An,x,y,z (n>=1 & $gut(n,x) &
   $gut(n-1,y) & $gut(y/2,z)) => x+z=n":
```

Thus our automaton correctly computes Gutkovskiy's sequence.

7 Proof of Irvine's conjecture

We now have everything we need to prove Irvine's conjecture.

Theorem 2. *For $n \geq 0$ we have $g_n = \sum_{1 \leq i \leq n} k_i$.*

Proof. We use the following `Walnut` code:

```
eval check "?msd_kim An K[n]=@1 <=> (Ex $gut(n-1,x) & $gut(n,x+1))":
```

and `Walnut` returns `TRUE`. □

8 Some related sequences

We now turn to three related sequences; for $n \geq 1$ the first two give the n'th positions of the ones (resp., zeros) in the sequence $\mathbf{k}$. We call them A_n and B_n, respectively. The third sequence, called Q_n, has a more complicated definition:

$$Q_n = \begin{cases} n, & \text{if } n \leq 1; \\ Q_m, & \text{if } n = Q_m + 2m \text{ and there is} \\ & \text{exactly one } i < n \text{ with } Q_i = Q_m; \\ \text{least positive integer not in } Q_1, \ldots, Q_{n-1}, & \text{otherwise.} \end{cases} \tag{2}$$

It is sequence A026366 in the OEIS.

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A_n	0	1	2	4	5	7	8	9	10	12	13	15	16	17	18	20
B_n	0	3	6	11	14	19	22	25	28	33	36	41	44	47	50	55
Q_n	0	1	2	1	3	4	2	5	6	7	8	3	9	10	4	11

Table 3: First few values of A_n, B_n, and Q_n.

Once again we can guess synchronized automata computing these functions and verify that they are correct. The guessed automaton for A_n has 23 states, the guessed automaton for B_n has 24 states, and the guessed automaton for Q_n has 45 states. We call them 'aa', 'bb', and 'qq', respectively.

We now verify correctness of A and B:

```
eval check_A_1 "?msd_kim An Ex $aa(n,x)":
eval check_A_2 "?msd_kim An,x,y ($aa(n,x) & $aa(n,y)) => x=y":
eval check_A_3 "?msd_kim Ax (En n>=1 & $aa(n,x)) <=> K[x]=@1":
eval check_A_4 "?msd_kim An,x,y ($aa(n,x) & $aa(n+1,y)) => x<y":
eval check_B_1 "?msd_kim An Ex $bb(n,x)":
eval check_B_2 "?msd_kim An,x,y ($bb(n,x) & $bb(n,y)) => x=y":
eval check_B_3 "?msd_kim Ax (En  $bb(n,x)) <=> K[x]=@0":
eval check_B_4 "?msd_kim An,x,y ($bb(n,x) & $bb(n+1,y)) => x<y":
```

and `Walnut` returns `TRUE` for all of these.

To verify correctness of Q, we need to verify its definition:

```
def occurs_once_in "?msd_kim (Ei,x i>=1 & i<n & $qq(i,x) & $qq(m,x)) &
   (~Ei,j,x i>=1 & i<j & j<n & $qq(i,x) & $qq(j,x) & $qq(m,x))":
# true if Q_m occurs exactly once in Q_0, Q_1, ..., Q_{n-1}
```

```
def occurs_in "?msd_kim Ei,y i<n & $qq(i,y) & $qq(i,x)":
# true if x occurs in Q_0, ..., Q_{n-1}

def least_not_in "?msd_kim (~$occurs_in(n,x)) &
   (Az (~$occurs_in(n,z)) => z>=x)":
# true if x is the least integer not in Q_1, ..., Q_{n-1}

eval check_Q_1 "?msd_kim An Ex $qq(n,x)":
eval check_Q_2 "?msd_kim An,x,y ($qq(n,x) & $qq(n,y)) => x=y":
eval check_Q_3 "?msd_kim Am,n,y,z (1<=m & m<n & $occurs_once_in(m,n) &
   $qq(m,y) & n=y+2*m & $qq(n,z)) => y=z":
eval check_Q_4 "?msd_kim An,y ($qq(n,y) & ~(Em 1<=m & m<n &
   $occurs_once_in(m,n))) => $least_not_in(n,y)":
```

So indeed our automaton computes Q_n correctly.

If we look at OEIS sequence A026367, we see that its description says (essentially) "least t such that $Q_t = n$". This allows use to verify that A026367 is in fact A_n, as follows:

```
def check_A_5 "?msd_kim An,t $aa(n,t) => $qq(t,n) & Au (u<t) => ~$qq(u,n)":
```

Similarly, if we look at OEIS sequence A026368, we see that its description says (essentially) "greatest t such that $Q_t = n$". We can then verify that A026368 is in fact B_n, as follows:

```
def check_B_5 "?msd_kim An,t $bb(n,t) => $qq(t,n) & Au (u>t) => ~$qq(u,n)":
```

In particular, we have proved Neil Sloane's observation that "A026368 appears to be [the] complement[ary] sequence of A026367".

We can easily verify the observation of Fokkink, Ortega, and Rust [3] that $B_n = 2A_n + n$ for $n \geq 0$:

```
eval check_FOR "?msd_kim An,x,y ($aa(n,x) & $bb(n,y)) => y=2*x+n":
```

and `Walnut` returns `TRUE`.

Finally, Fokkink, Ortega, and Rust [3] left the following as an open problem, which we can turn into a theorem.

Theorem 3. *For all n we have $A_{B_n} \in \{A_n + B_n - 1, A_n + B_n\}$.*

Proof. We use the following `Walnut` code:

```
eval check_FOR_2 "?msd_kim An,t,x,y ($aa(n,t) & $bb(n,x) & $aa(x,y)) =>
   (y=t+x|y+1=t+x)":
```

and `Walnut` returns `TRUE`. □

Remark 4. Furthermore we could, if it were desired, give a DFAO that computes, for each input n, which of the two alternatives in Theorem 3 holds.

Similarly we can prove, for example, that $B_{A_n} - A_n - B_n \in \{-3, -2, -1, 0, 1\}$.

9 Subword complexity

Recall that the subword complexity function $\rho(n)$ counts the number of distinct factors of length n of an infinite word. In this section we compute this function for $\mathbf{k}$.

Call a factor w of an infinite binary word $\mathbf{x}$ *right-special* if both $w0$ and $w1$ appear in $\mathbf{x}$. For binary words we know that $\rho(n+1) - \rho(n)$ counts the number of length-n right-special factors.

`Walnut` formulas for special factors are given in [10, §8.8.6]. Adapting them to our situation, we have the following code:

```
def keqfac "?msd_kim At (t<n) => K[i+t]=K[j+t]":
def kisrs "?msd_kim Ej $keqfac(i,j,n) & K[i+n]!=K[j+n]":
eval nothree "?msd_kim Ei,j,k,n $kisrs(i,n) & $kisrs(j,n) & $kisrs(k,n) &
   ~$keqfac(i,j,n) & ~$keqfac(j,k,n) & ~$keqfac(i,k,n)":
def hastwo "?msd_kim Ei,j $kisrs(i,n) & $kisrs(j,n) & ~$keqfac(i,j,n)":
```

Here

- `keqfac` asserts that $\mathbf{k}[i..i+n-1] = \mathbf{k}[j..j+n-1]$;
- `kisrs` asserts that $\mathbf{k}[i..i+n-1]$ is a right-special factor;
- `nothree` asserts that there is no n for which $\mathbf{k}$ has three or more distinct right-special factors of length n;
- `hastwo` accepts precisely those n for which $\mathbf{k}$ has exactly two distinct right-special factors of length n.

The automaton created by 'hastwo' is displayed in Figure 5. We can now prove the

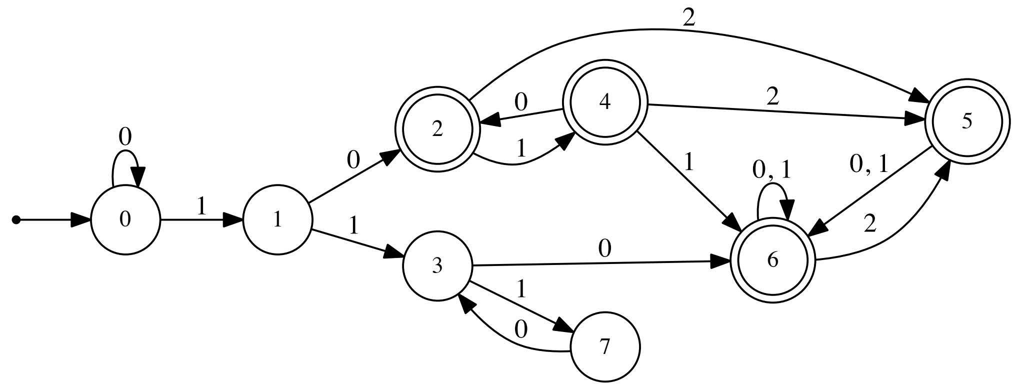

Figure 5: Automaton accepting those n for which $\mathbf{k}$ has exactly two distinct right-special factors of length n.

following theorem.

Theorem 5. *The infinite word* $\mathbf{k}$ *has exactly two distinct right-special factors of length* n *if and only if there exists* $i \geq 0$ *such that one of the following holds:*

- $x \leq n < x + \mathcal{K}_{2i}$*, where* $x = \mathcal{K}_1 + \mathcal{K}_3 + \cdots + \mathcal{K}_{2i+1}$*;*
- $y \leq n < y + \mathcal{K}_{2i+1}$*, where* $y = \mathcal{K}_0 + \mathcal{K}_2 + \cdots + \mathcal{K}_{2i+2}$*.*

Proof. We use the following `Walnut` code:

```
reg ul msd_kim msd_kim "[0,0]*[1,1][0,1]([1,1][0,0])*(()|[1,1]":
eval check_sw "?msd_kim An $hastwo(n) <=> Ex,y $ul(x,y) & x<=n & n<y":
```

and `Walnut` returns `TRUE`. □

Corollary 6. *We have* $\limsup_{n\geq 1} \rho(n)/n = (30+\sqrt{3})/23 \doteq 1.37965438$ *and* $\liminf_{n\geq 1} \rho(n)/n = (3+\sqrt{3})/4 \doteq 1.1830127$*.*

10 Critical exponents

Recall that we say $p \geq 1$ is a *period* of a finite word $x = x[1..n]$ if $x[i] = x[i+p]$ for $1 \leq i \leq n-p$. The *exponent* of a finite word x is the length of x divided by its shortest period. Finally, the *critical exponent* of an infinite word $\mathbf{z}$ is the supremum, over all finite nonempty factors x of $\mathbf{z}$, of the exponent of x.

Theorem 7. *The critical exponent of* $\mathbf{k}$ *is* $(2\sqrt{3}+12)/3 \doteq 5.1547$*.*

Proof. Since the basic ideas have already been covered elsewhere in detail [10, pp. 148–150], we just sketch them here. We create `Walnut` formulas for the shortest period of a factor of $\mathbf{k}$, and then obtain the corresponding longest words with the given period. Then we restrict to those factors of exponent at least 5. The resulting automaton, computed by 'klong5', accepts pairs of the form $(n,p) = ([121(01)^i 0]_K, [10(00)^i 0]_K)$ and $(n,p) = ([121(01)^i 02]_K, [10(00)^i 00]_K)$. Routine work with two-term linear recurrences then gives the result.

```
def kperi "?msd_kim p>0 & p<=n & Aj (j>=i & j+p<i+n) => K[j]=K[j+p]":
def klper "?msd_kim $kperi(i,n,p) & (Aq (q>=1 & q<p) => ~$kperi(i,n,q))":
def kleastp "?msd_kim Ei,n n>=1 & $klper(i,n,p)":
def klongest "?msd_kim (Ei $klper(i,n,p)) &
   (Ar,i $klper(i,r,p) => r<=n)":
def klong5 "?msd_kim $klongest(n,p) & n>5*p":
```

□

Acknowledgments

I thank Michel Dekking for telling me about his paper [2], and in particular its Remark 9.

References

[1] S. Artstein-Avidan, A. S. Fraenkel, and V. T. Sós. A two-parameter family of an extension of Beatty sequences. *Discrete Math.* **308** (2008), 4578–4588.

[2] F. M. Dekking. On Hofstadter's G-sequence. Arxiv preprint arXiv:2307.01471 [math.CO], July 25 2023. Available at https://arxiv.org/abs/2307.01471.

[3] R. Fokkink, G. F. Ortega, and D. Rust. Corner the empress. Arxiv preprint arXiv:2204.11805 [math.CO], December 8 2022. Available at https://arxiv.org/abs/2204.11805.

[4] A. S. Fraenkel. Systems of numeration. *Amer. Math. Monthly* **92** (1985), 105–114.

[5] A. S. Fraenkel. Heap games, numeration systems and sequences. *Ann. Combin.* **2** (1998), 197–210.

[6] C. Frougny and B. Solomyak. On representation of integers in linear numeration systems. In M. Pollicott and K. Schmidt, editors, *Ergodic Theory of $\mathbb{Z}^d$ Actions (Warwick, 1993–1994)*, Vol. 228 of *London Mathematical Society Lecture Note Series*, pp. 345–368. Cambridge University Press, 1996.

[7] D. R. Hofstadter. *Gödel, Escher, Bach: an Eternal Golden Braid.* Basic Books, 1979.

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

[9] J. Shallit. A generalization of automatic sequences. *Theoret. Comput. Sci.* **61** (1988), 1–16.

[10] J. Shallit. *The Logical Approach to Automatic Sequences: Exploring Combinatorics on Words with* `Walnut`, Vol. 482 of *London Math. Soc. Lecture Notes Series.* Cambridge University Press, 2022.

[11] N. J. A. Sloane et al. The On-Line Encyclopedia of Integer Sequences. Electronic resource available at https://oeis.org, 2023.