Synchronized Sequences

Jeffrey Shallit

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

Automatic sequences

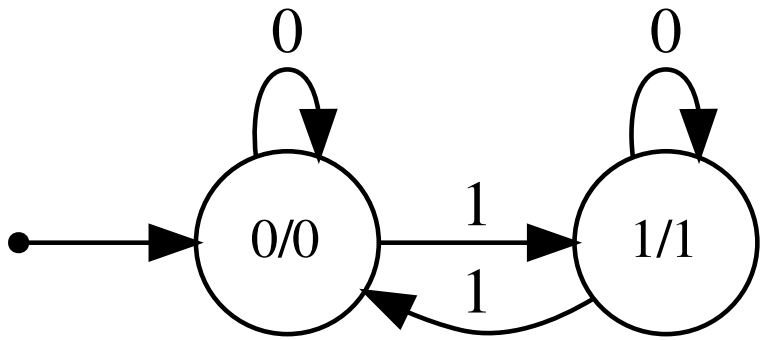
A sequence $(a(n))_{n\geq 0}$ over a finite alphabet is *k-automatic* if there is a deterministic finite automaton with output (DFAO) that, on input n in base k, reaches a state with output $a(n)$.

The most famous automatic sequence is the *Thue-Morse* sequence

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

defined by $t_n = s_2(n) \bmod 2$, where s_2 is the sum of the bits in the binary representation of n.

It is generated by the following DFAO:



Regular sequences

A sequence $(c(n))_{n\geq 0}$ taking values in $\mathbb{Z}$ is *k-regular* if there are a finite number of sequences $c_1 = c, c_2, \ldots, c_t$ such that each subsequence of the form $(c(k^j n + e))_{n\geq 0}$, $j \geq 0$, $0 \leq e < k^j$, is a $\mathbb{Q}$-linear combination of the $(c_i(n))_{n\geq 0}$.

An alternate definition is that there exists a *linear representation* (v, γ, w) where

- v is a $1 \times t$ matrix;
- γ is a $t \times t$-matrix-valued morphism; and
- w is a $t \times 1$ matrix,

such that

$$c(n) = v\gamma(x)w,$$

where x is the base-k representation of n.

An example of a 2-regular sequence is $s_2(n)$, the sum of the base-2 digits of n.

Synchronized sequences

In this talk I will speak about a class of sequences that lies strictly between the automatic sequence and the regular sequences: the *synchronized sequences*.*

* *Not* the same notion as "synchronized" in the sense of Černý's conjecture!

Like automatic sequences, the first-order logical theory of synchronized sequences is decidable, which means many assertions about them can be mechanically verified.

Like regular sequences, the synchronized sequences take their values in an infinite alphabet, namely $\mathbb{N}$.

So, combining the virtues of both classes, the synchronized sequences are a particularly nice class to study.

Representation of numbers

Let $(n)_k$ be the canonical representation of n in base k, with no leading zeros, starting with the most significant digit.

This can be generalized to pairs as follows: pad the representation of the shorter number with leading zeros so they have the same length. Then encode $(m, n)_k$ by a sequence of pairs of digits. For example,

$$(43, 17)_2 = [1, 0][0, 1][1, 0], [0, 0], [1, 0][1, 1].$$

It can also be generalized to pairs of bases: $(m, n)_{k,\ell}$ means

- m is expressed in base k, and
- n is expressed in base ℓ.

Definition of synchronized sequence

We say that a sequence $(f(n))_{n\geq 0}$ is (k,ℓ)-*synchronized* if there is a DFA recognizing the language

$$[0,0]^*\{(n,f(n))_{k,\ell} \ : \ n \geq 0\}.$$

In other words, the *graph* of the function f is a regular language.

(If $k=\ell$ we just say f is k-*synchronized*.)

Three example sequences

$$a(n) = \left(\sum_{0 \le i \le n} s_2(i) \right) \bmod 2 = \left(\sum_{0 \le i \le n} t_i \right) \bmod 2 \quad \text{(automatic)}$$

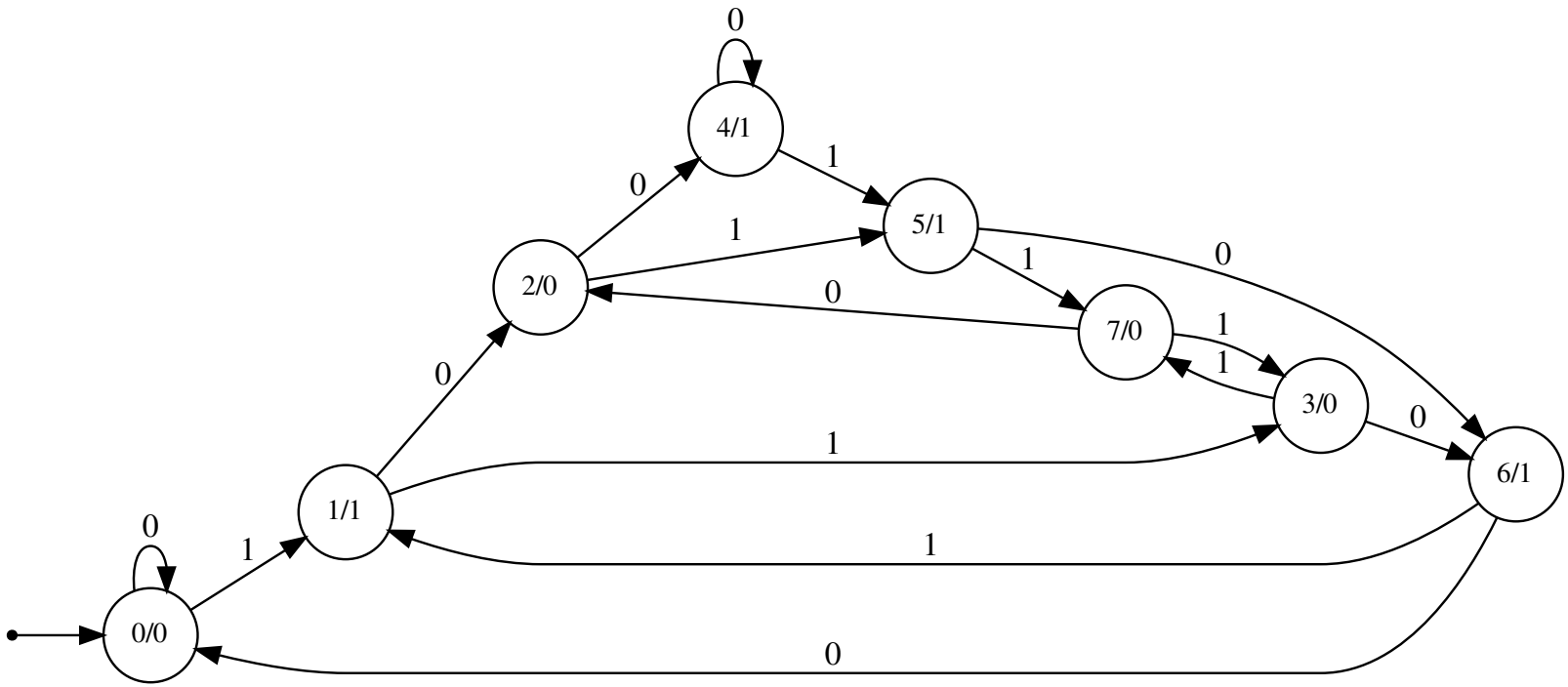
$$b(n) = \sum_{0 \le i \le n} (s_2(i) \bmod 2) = \sum_{0 \le i \le n} t_i \quad \text{(synchronized, not automatic)}$$

$$c(n) = \sum_{0 \le i \le n} s_2(i) \quad \text{(regular, not synchronized)}.$$

n	0	1	2	3	4	5	6	7	8	9	10	OEIS Seq.
$a(n)$	0	1	0	0	1	1	1	0	1	1	1	A255817
$b(n)$	0	1	2	2	3	3	3	4	5	5	5	A115384
$c(n)$	0	1	2	4	5	7	9	12	13	15	17	A000788

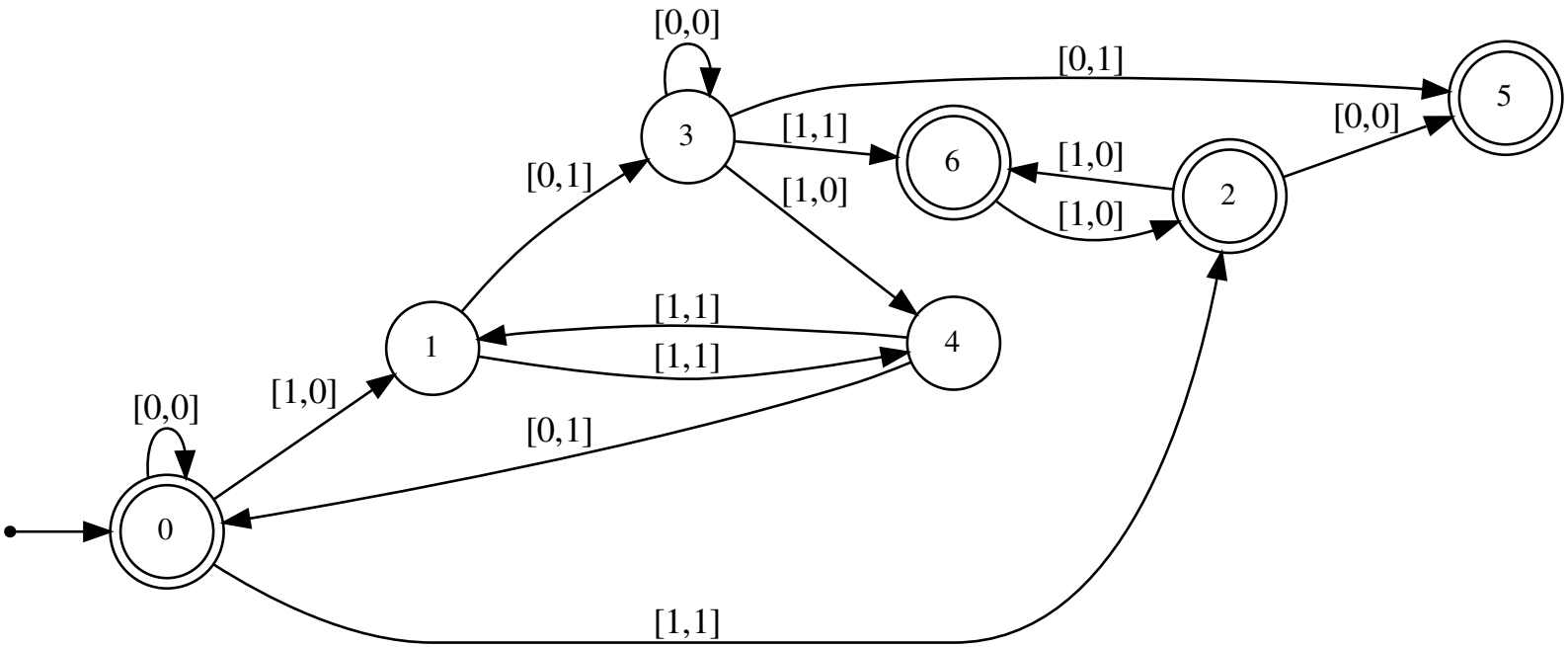
$a(n)$ is an automatic sequence

$$a(n) = \left(\sum_{0 \le i \le n} t_i \right) \bmod 2.$$



$b(n)$ is a synchronized sequence

$$b(n) = \sum_{0 \le i \le n} (s_2(i) \bmod 2).$$



This DFA accepts the input $[1,1][1,0][1,0]$, so $b(7) = 4$.

$c(n)$ is a regular sequence

$$c(n) = \sum_{0 \le i \le n} s_2(i).$$

It has linear representation

$$v = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}^T; \quad \gamma(0) = \begin{bmatrix} 2 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 1 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}; \quad \gamma(1) = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 2 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}; \quad w = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

For example,

$$c(5) = v\gamma(1)\gamma(0)\gamma(1)w = 7.$$

History of synchronized sequences

Although the idea of synchronization goes back to the 1950's (Rabin-Scott 1959; Elgot-Mezei 1965; Frougny-Sakarovitch 1993), the first paper discussing synchronized *sequences* was by Carpi and Maggi (2001).

They proved that if an aperiodic sequence is k-automatic, then its separator function ($n \rightarrow$ length of the shortest novel factor beginning at position n) is synchronized.

Here a *novel factor* means it has never occurred previously in the sequence.

Carpi and Maggi also proved a number of closure properties of synchronized sequences.

And they proved an upper bound on the growth rate of a synchronized sequence.

Connection between logic and synchronization

A theorem of Bruyère et al. implies

Theorem. *If we can write a first-order formula φ, with two free variables s and n, asserting that $s = f(n)$, in terms of k-automatic sequences, logical operations, universal and existential quantifiers, and comparisons and addition on integers, then f is a k-synchronized sequence.*

Furthermore, there is an algorithm to "compile" a formula φ into a synchronized DFA computing f.

Free software called `Walnut`, created by Hamoon Mousavi, can be used to create synchronized DFA's from first-order logic statements, and test their properties.

Automatic sequences and synchronization

Many aspects of automatic sequences are synchronized.

For example, the *uniform recurrence function* for $\mathbf{x}$: $R_{\mathbf{x}}(n)$ is the smallest integer m such that every length-m block contains all factors of $\mathbf{x}$ of length n.

This is well-defined iff $\mathbf{x}$ is uniformly recurrent.

If $\mathbf{x}$ is automatic, then we can show $R_{\mathbf{x}}(n)$ by constructing a first-order formula with free variables n and s, asserting that $s = R_{\mathbf{x}}(n)$.

We do this as follows:

The recurrence function is synchronized

$$\begin{aligned}
\text{FactorEq}(i,j,n) &:= \forall t\ (t < n) \implies \mathbf{x}[i+t] = \mathbf{x}[j+t] \\
\text{Occurs}(i,j,n,s) &:= (n \le s) \ \wedge\ \exists k\ (k+n \le s) \ \wedge\ \text{FactorEq}(i, j+k, n) \\
\text{ContainsAll}(j,n,s) &:= \forall i\ \ \text{Occurs}(i,j,n,s) \\
\text{Recur}(n,s) &:= (\forall j\ \ \text{ContainsAll}(j,n,s) \wedge \\
&\qquad (\exists k\ \neg\, \text{ContainsAll}(k,n,s-1))
\end{aligned}$$

Here

- FactorEq(i,j,n) asserts that $\mathbf{x}[i..i+n-] = \mathbf{x}[j..j+n-1]$
- Occurs(i,j,n,s) asserts that $\mathbf{x}[i..i+n-1]$ occurs as a factor of $\mathbf{x}[j..j+s-1]$
- ContainsAll(j,n,s) asserts that $\mathbf{x}[j..j+s-1]$ contains all length-n factors of $\mathbf{x}$
- Recur(n,s) asserts that every length-s block contains all length-n factors of $\mathbf{x}$, but some length-$(s-1)$ block doesn't.

Computing the recurrence function of Thue-Morse with Walnut

We use the following Walnut commands:

```
def tmfactoreq "At t<n => T[i+t]=T[j+t]":
def tmoccurs "n<=s & Ek k+n<=s & $tmfactoreq(i,j+k,n)":
def tmcontainsall "Ai $tmoccurs(i,j,n,s)":
def tmrecur "(Aj $tmcontainsall(j,n,s)) &
     (Ek ~$tmcontainsall(k,n,s-1))":
```

Walnut then outputs a 12-state synchronized automaton computing $R_{\mathbf{t}}(n)$.

Note: A means "for all", T represents the Thue-Morse sequence, E means "there exists", ~ means "not", & means "and", => means "implies".

Computing the recurrence function of Thue-Morse

With this synchronized automaton we can easily reprove a 1938 theorem of Morse and Hedlund:

Theorem. *The recurrence function for the Thue-Morse sequence is*

$$R_{\mathbf{t}}(n) = \begin{cases} 3, & \text{if } n = 1; \\ 9, & \text{if } n = 2; \\ 9 \cdot 2^j + n - 1, & \text{if } n \geq 3 \text{ and } 2^j + 2 \leq n \leq 2^{j+1} + 1. \end{cases}$$

We can verify this formula for $n \geq 3$ with `Walnut` as follows (letting the variable x stand for 2^j):

```
reg power2 msd_2 "0*10*":
eval checktmrecur "An,x,s (n>=3 & $power2(x) & x+2<=n
   & n<=2*x+1 & $tmrecur(n,s)) => s+1=9*x+n":
```

which returns `TRUE`.

Subword complexity

Recall that the subword complexity function $\rho_{\mathbf{x}}(n)$ is defined to be the number of distinct length-n factors appearing in $\mathbf{x}$.

Theorem. *If* $\mathbf{x}$ *is automatic then* $\rho_{\mathbf{x}}(n)$ *is synchronized.*

Proof. We need a nontrivial idea: in an automatic sequence, the novel factors occur consecutively in a finite number of clumps. So we can construct a first-order formula asserting that $s = \rho_{\mathbf{x}}(n)$ by

- using existential quantifiers for the starting and ending positions of clumps,
- verifying that novel factors occur inside the clumps and not outside, and
- asserting that s is the sum of the clump sizes.

Computing values of a synchronized function

Suppose we have a synchronized DFA M for $f(n)$. How do we actually evaluate f on some particular n?

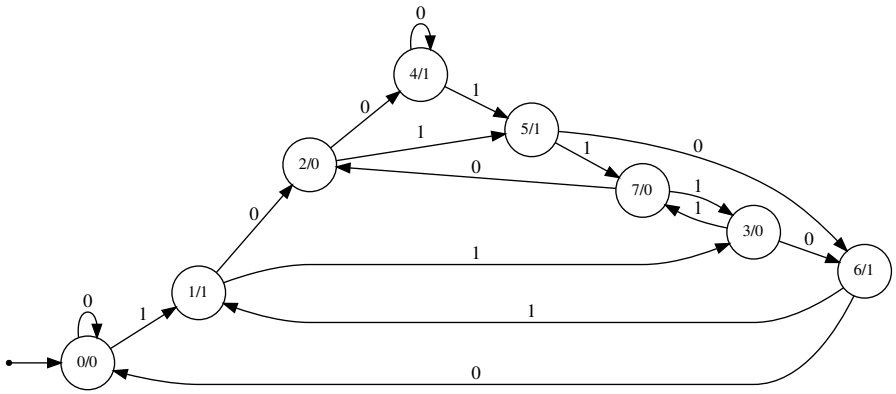
It's not hard: take the DFA M and intersect it with a DFA M_n recognizing those strings with $0^*(n)_k$ in the first coordinate, and anything in the second coordinate, obtaining M'.

This DFA M_n has $O(\log n)$ states, so M' also has $O(\log n)$ states, and only recognizes $0^*(n, f(n))_k$.

We can therefore compute $f(n)$ using breadth-first search in M' in $O(\log n)$ time.

Using Walnut to verify the automaton for $a(n)$

$$a(n) = \left(\sum_{0 \le i \le n} t_i \right) \bmod 2.$$



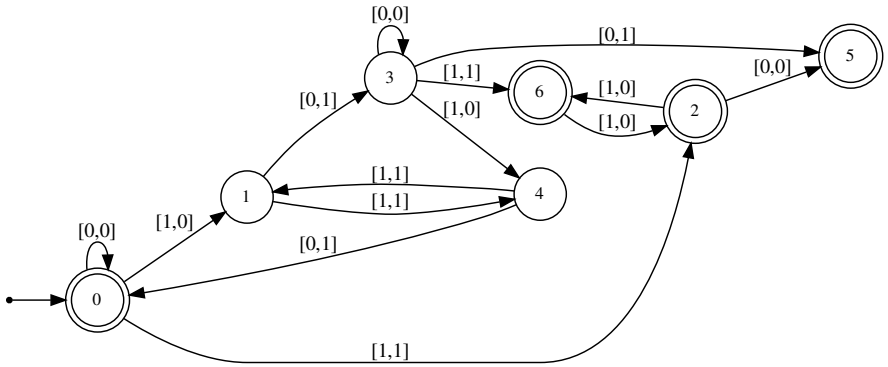
It suffices to check the assertions that $a(0) = 0$ and $a(n+1) = (a(n) + t_{n+1}) \bmod 2$ for all n. We can do this in Walnut as follows:

```
eval verifyseqa "SA[0]=@0 & (An SA[n+1]=@1 <=>
((SA[n]=@0 & T[n+1]=@1)|(SA[n]=@1 & T[n+1]=@0)))":
```

where SA is the DFAO at top right. And Walnut returns TRUE.

Verifying the synchronized automaton for $b(n)$

$$b(n) = \sum_{0 \le i \le n} (s_2(i) \bmod 2).$$



It suffices to check that $b(0) = 0$ and $b(n+1) = b(n) + t_{n+1}$.

We can do this in `Walnut` as follows:

```
eval verifyseqb "$seqb(0,0) &
An,y,z ($seqb(n,y) & $seqb(n+1,z)) =>
((z=y & T[n+1]=@0)|(z=y+1 & T[n+1]=@1))":
```

Computing the linear representation for $c(n)$

$$c(n) = \sum_{0 \le i \le n} s_2(i).$$

Then

$$\begin{aligned} c(2n+1) &= \sum_{0 \le i \le 2n+1} s_2(i) = \sum_{0 \le i \le n} s_2(2i) + \sum_{0 \le i \le n} s_2(2i+1) \\ &= \sum_{0 \le i \le n} s_2(i) + \sum_{0 \le i \le n} (s_2(i)+1) = 2c(n) + n + 1 \end{aligned}$$

and

$$\begin{aligned} c(2n) &= c(2n+1) - s_2(2n+1) \\ &= (2c(n) + n + 1) - (s_2(n) + 1) = 2c(n) + n - s_2(n). \end{aligned}$$

Linear representation for $c(n)$

This means that the 2-kernel of $(c(n))_{n\geq 0}$ is a subset of

$$\langle\ (c(n))_{n\geq 0}, (s_2(n))_{n\geq 0}, (n)_{n\geq 0}, (1)_{n\geq 0}\ \rangle.$$

We can find a linear representation for $c(n)$ by observing that

$$[c(2n) \quad s_2(2n) \quad 2n \quad 1] = [c(n) \quad s_2(n) \quad n \quad 1] \begin{bmatrix} 2 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 1 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

and

$$[c(2n+1) \quad s_2(2n+1) \quad 2n+1 \quad 1] = [c(n) \quad s_2(n) \quad n \quad 1] \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 2 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}.$$

Verifying another formula for a synchronized sequence

One big advantage to a synchronized sequence is that there is a decision procedure to verify formulas for it.

For example, if we look up sequence A115384 in the OEIS, we find the formula

$$b(n) = \left\lfloor \frac{n+1}{2} \right\rfloor + \frac{(1+(-1)^n)(1-(-1)^{t_n})}{4}.$$

As stated this is not quite amenable to verification (because of the exponents). However, note that if $x \in \{0, 1\}$ then $(-1)^x = 1 - 2x$. So simplifying, we need to verify that

$$b(n) = \left\lfloor \frac{n+1}{2} \right\rfloor + [t_n = 1 \text{ and } n \text{ even}],$$

where $[z]$ is the Iverson bracket, equal to 1 if z is true and 0 otherwise.

Verifying a formula for a synchronized sequence

We need to verify that

$$b(n) = \left\lfloor \frac{n+1}{2} \right\rfloor + [t_n = 1 \text{ and } n \text{ even}],$$

We can do this in Walnut as follows:

```
def even "Em n=2*m":
def odd "Em n=2*m+1":
eval checkform "An,b $seqb(n,b) <=>
   ((T[n]=@1 & $even(n) & b=((n+1)/2)+1)|
   ((T[n]=@0 | $odd(n)) & b=(n+1)/2))":
```

which returns `TRUE`.

Relationship between automatic, regular, and synchronized sequences

In what follows, k and ℓ are arbitrary integers ≥ 2.

Theorem. *Let $(a(n))_{n\geq 0}$ be a (k,ℓ)-synchronized sequence. Then it is k-automatic iff $a(n) = O(1)$.*

Theorem. *Let $(a(n))_{n\geq 0}$ be a (k,ℓ)-synchronized sequence. Then it is k-regular.*

Theorem. *Let $(a(n))_{n\geq 0}$ be a (k,ℓ)-synchronized sequence and let S be the range of the sequence $(a(n))_{n\geq 0}$. Then the characteristic sequence $(\chi_S(i))_{i\geq 0}$, defined to be 1 if $i \in S$ and 0 otherwise, is ℓ-automatic.*

Growth rate of synchronized sequences

Let $k, \ell \geq 2$ be integers, and define $\beta = (\log \ell)/(\log k)$.

Theorem. *Let $(f(n))_{n \geq 0}$ be a (k, ℓ)-synchronized sequence. Then*

(a) $f(n) = O(n^\beta)$;

(b) *If $f(n) = o(n^\beta)$, then $f(n) = O(1)$;*

(c) *If there exists an increasing subsequence $0 < n_1 < n_2 < \cdots$ such that $\lim_{i \to \infty} f(n_i)/n_i^\beta = 0$, then there exists a constant C such that $f(n) = C$ for infinitely many n.*

Growth rate of synchronized sequences

Proof. (a): Suppose $f \neq O(n^\beta)$, where $\beta = (\log \ell)/(\log k)$.

Then there exists an increasing subsequence $(n_i)_{i \geq 0}$ such that $f(n_i)/n_i^\beta \to \infty$.

Suppose the DFA recognizing $\{(n, f(n))_{k,\ell} : n \geq 0\}$ has t states (t is the pumping lemma constant).

Choose i such that $n_i \geq k^t$ and $f(n_i)/n_i^\beta > \ell^{t+1}$, and in the pumping lemma let $z = (n_i, f(n_i))_{k,\ell}$.

Then $|z| > t$, and furthermore we have

$$\begin{aligned} |f(n_i)_\ell| &> \log_\ell f(n_i) > \log_\ell(n_i^\beta \ell^{t+1}) = (\log_\ell n_i^\beta) + t + 1 \\ &= (\beta \log_\ell n_i) + t + 1 = (\log_k n_i) + t + 1 \geq |(n_i)_k| + t. \end{aligned}$$

Growth rate of synchronized sequences

Hence the first component of z starts with at least t 0's, while the second component starts with a nonzero digit.

When we pump (that is, write $z = uvw$ with $|uv| \leq t$ and $|v| \geq 1$ and consider uv^2w) we only add to the number of leading 0's in the first component, so its numerical value does not change.

But the second component's base-ℓ value increases in size (since it starts with a nonzero digit). This implies that f is not a function, a contradiction. ■

Parts (b) and (c) can be proved in a similar way.

Closure properties

Theorem. *Suppose* $(a(n))_{n\geq 0}$ *and* $(b(n))_{n\geq 0}$ *are* (k,ℓ)*-synchronized sequences. Then so are the sequences*

(a) $(a(n)+b(n))_{n\geq 0}$;

(b) $(a(n) \dot{-} b(n))_{n\geq 0}$, *where* $x \dot{-} y$ *is the "monus" function, defined by* $\max(0, x-y)$;

(c) $(|a(n)-b(n)|)_{n\geq 0}$;

(d) $(\lfloor \alpha a(n) \rfloor)_{n\geq 0}$, *where* α *is a non-negative rational number;*

(e) $(\max(a(n), b(n)))_{n\geq 0}$;

(f) $(\min(a(n), b(n)))_{n\geq 0}$;

(g) *running maximum, defined by* $c(n) = \max_{0\leq i\leq n} a(n)$;

(h) *running minimum, defined by* $d(n) = \min_{0\leq i\leq n} a(n)$.

Closure properties

Let's prove (c): If $a(n)$, $b(n)$ are both (k, ℓ)-synchronized then so is $|a(n) - b(n)|$.

Let A be a (k, l)-synchronized DFA computing $a(n)$ and B be a (k, l)-synchronized DFA computing $b(n)$.

It suffices to write a first-order formula with free variables n and s asserting that $s = |a(n) - b(n)|$:

$$\forall x, y \ (A(n, x) \ \wedge \ B(n, y)) \implies$$
$$((x \geq y \implies x = s + y) \ \wedge \ (x < y \implies y = s + x)).$$

Guessing a synchronized automaton

Suppose you have a sequence over $\mathbb{N}$ defined in some way, and you suspect it is k-synchronized. How can you check this?

In general, of course, there is no algorithm.

However, in some cases, the following idea works. Use the Myhill-Nerode theorem to *guess* an automaton recognizing the language $[0,0]^*\{(n, f(n))_k \ : \ n \geq 0\}$.

We can do this by computing f to hundreds or thousands of terms, and then checking the Myhill-Nerode equivalence relation just on the terms we have computed.

After a candidate has been guessed, use `Walnut` to verify that it has the desired property. This gives a rigorous *proof* that your guess was correct!

Propp's sequence

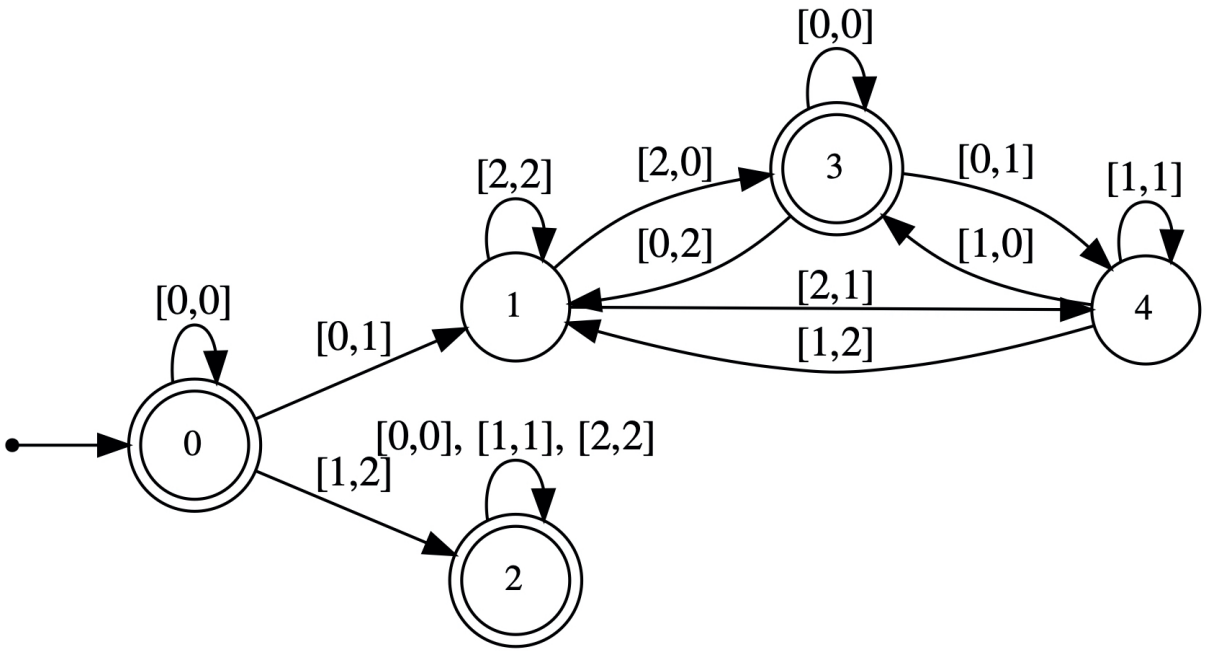
We can illustrate the "guessing" approach with *Propp's sequence*; it is the unique increasing sequence $(s(n))_{n\geq 0}$ of natural numbers with the property that $s(s(n)) = 3n$.

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$s(n)$	0	2	3	6	7	8	9	12	15	18	19	20	21	22	23

Assuming that $s(n)$ is a 3-sychronized function, we can "guess" its automaton using the procedure just described.

We get...

Propp's sequence



Now we can verify that our guess is correct: indeed $s(s(n)) = 3n$ for all n.

```
eval proppcheck "?msd_3 An,x,y
   ($prop(n,x) & $prop(x,y)) => y=3*n":
```

Propp's sequence

We can also verify yet another formula for s, namely

$$s(n) = \begin{cases} 0, & \text{if } n = 0; \\ n + 3^k, & \text{if } 3^k \le n < 2 \cdot 3^k \text{ for } k \ge 0; \\ 3(n - 3^k), & \text{if } 2 \cdot 3^k \le n < 3^{k+1} \text{ for } k \ge 0. \end{cases}$$

with the following `Walnut` code:

```
reg power3 msd_3 "0*10*":
def pow3n "?msd_3 $power3(x) & x<=n & n<3*x":
     # x is the largest power of 3 that is <= n
eval proppcheck3 "?msd_3 An,x (($pow3n(n,x) & n<2*x) =>
     $prop(n,n+x)) | (($pow3n(n,x) & n>=2*x) =>
     $prop(n,(3*n)-3*x))":
```

Fibonacci synchronization

Up until now, we have discussed synchronization with respect to representations in base k.

However, it is quite possible to have synchronization in alternative number systems, such as Fibonacci representation.

In this system, we write a natural number n in the form

$$[a_1 \cdots a_t]_F = \sum_{1 \le i \le t} a_i F_{t+2-i},$$

where the Fibonacci numbers are defined by $F_0 = 0$, $F_1 = 1$, and $F_n = F_{n-1} + F_{n-2}$ for $n \ge 2$.

Every natural number n has a unique representation $(n)_F$ like this (up to leading zeroes), provided $a_i \in \{0, 1\}$ and $a_i a_{i+1} = 0$ for all i.

Fibonacci synchronization and Wythoff sequences

Recall that the Fibonacci word $\mathbf{f} = f_0 f_1 f_2 \cdots = 01001010\cdots$ is the fixed point of the morphism $0 \to 01$, $1 \to 0$.

Our goal is to find synchronized automata for two classical number-theoretic sequences: the lower and upper Wythoff sequences:

$$L(n) = \lfloor \varphi n \rfloor$$
$$U(n) = \lfloor \varphi^2 n \rfloor,$$

where $\varphi = (1+\sqrt{5})/2$, the golden ratio.

n	0	1	2	3	4	5	6	7	8	9	10	OEIS Seq.
f_n	0	1	0	0	1	0	1	0	0	1	0	A003849
$L(n)$	0	1	3	4	6	8	9	11	12	14	16	A000201
$U(n)$	0	2	5	7	10	13	15	18	20	23	26	A001950

Fibonacci synchronization and Wythoff sequences

We start with the sequences computing the positions of the n'th 0 (resp., n'th 1) in the Fibonacci word $\mathbf{f}$:

Theorem. *$P_0(n) = [(n)_F 0]_F$ and $P_1(n) = [(n)_F 01]_F$.*

This immediately gives two simple synchronized automata for $P_0(n)$ and $P_1(n)$.

Fibonacci synchronization and Wythoff sequences

In Walnut we can define

```
reg shift {0,1} {0,1} "([0,0]|[0,1][1,0])*":
def p0 "?msd_fib $shift(n,s)":
def p1 "?msd_fib Er,t $p0(n,r) & $p0(r,t) & s=t+1":
```

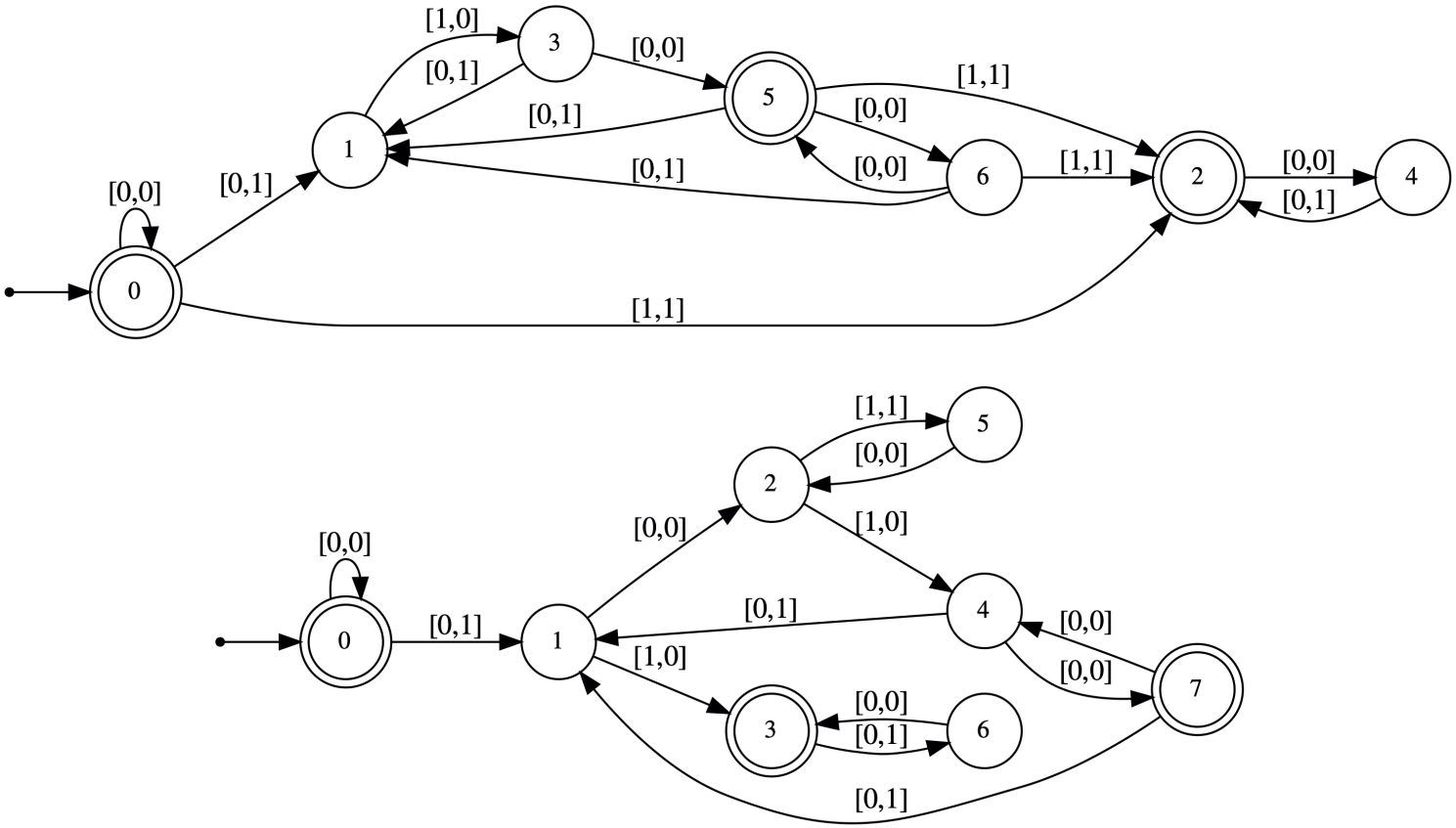
Theorem. *For $n \geq 1$ we have $L(n) = P_0(n-1) + 1$ and $U(n) = P_1(n-1) + 1$.*

In Walnut we can use this theorem to define

```
def lowerw "?msd_fib (n=0&s=0) | Et $p0(n-1,t) & s=t+1":
def upperw "?msd_fib (n=0&s=0) | Et $p1(n-1,t) & s=t+1":
```

Fibonacci synchronization and Wythoff sequences

This gives Fibonacci-synchronized automata for $L(n)$ (top) and $U(n)$ (bottom).



Fibonacci synchronization and Wythoff sequences

Using these we can easily prove recent results of Kawsumarng et al.:

Theorem.
(a) *Every integer $n > 9$ can be written as the sum of $L(i) + U(j) + U(k)$;*
(b) *Every integer $n > 26$ can be written as the sum $U(i) + U(j) + U(k)$.*

```
eval kawa "?msd_fib An (n>9) => Ei,j,k,r,s,t $lowerw(i,r) &
   $upperw(j,s) & $upperw(k,t) & n=r+s+t":
eval kawb "?msd_fib An (n>26) => Ei,j,k,r,s,t $upperw(i,r) &
   $upperw(j,s) & $upperw(k,t) & n=r+s+t":
```

Both of these return `TRUE`.

Minimal excludant and Fibonacci synchronization

We can also use our Fibonacci-synchronized automata to prove a basic characterization of the lower and upper Wythoff sequences, namely: For a set $S \subsetneq \mathbb{N}$, we define $\mathrm{mex}(S) = \min\{n : n \notin S\}$. Then

$$L_n = \mathrm{mex}\{L_i, U_i : 0 \leq i < n\}$$
$$U_n = L_n + n.$$

We can check this as follows:

```
def incl "?msd_fib Ei i<n & ($lowerw(i,s) | $upperw(i,s))":
     # s appears in {L_i, U_i : 0 <= i < n }
def mex "?msd_fib (~$incl(n,s)) & At (t<s) => $incl(n,t)":
     # s equals mex {L_i, U_i : 0 <= i < n }
eval mexchk1 "?msd_fib An,s $mex(n,s) <=> $lowerw(n,s)":
eval mexchk2 "?msd_fib An,s $upperw(n,s) <=>
   (Et $lowerw(n,t) & s=t+n)":
```

Abelian properties

A property of a word is called *abelian* if it depends only on the number of occurrences of each letter in the word.

For example, an *abelian square* is a word of the form xx' where x' is a permutation of x, like the English word `reappear`.

Although first-order formulas can't deal with abelian properties in general, if an automatic infinite word has the property that the number of occurrences of each letter in a length-n prefix is synchronized, then we can do so.

Examples of such words include the Thue-Morse word $\mathbf{t}$ and the Fibonacci word $\mathbf{f}$.

Tribonacci synchronization: a new result

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

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

What are the orders of abelian cubes appearing in **tr**?

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

Unsynchronized sequences

Not all aspects of automatic sequences are synchronized.

For example, the number of unbordered length-n factors of an automatic sequence need not be synchronized.

Here is another example: $f(n) = n^2$ cannot be (k, k^2)-synchronized.

Proof. Assume it is. Then $f(n+1)$ would be (k, k^2)-synchronized, and hence $f(n+1) - f(n) = 2n+1$ would be (k, k^2)-synchronized. But this violates the growth rate theorem we presented earlier.

Unsynchronized sequences

Sometimes we can use (the lack of) synchronization to prove that a sequence is not k-automatic for any k.

For example, consider **vn**, the fixed point of the morphism $a \to aab$, $b \to b$.

$$\mathbf{vn} = aabaabbaabaabbb\cdots$$

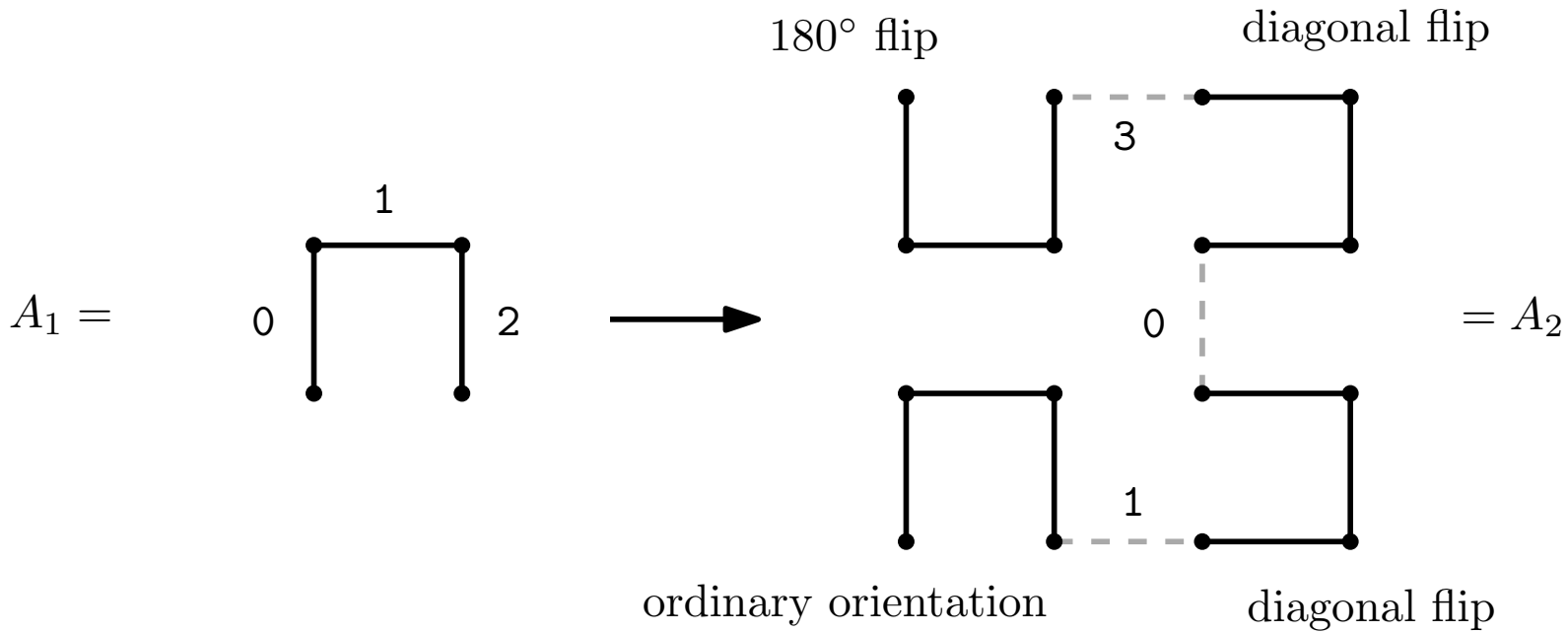
If it were k-automatic, then the starting position p_n of the first block of n b's in **vn** would be synchronized, and hence $p_n = O(n)$.

But it is easily checked that $p_n = 2^{n+1} - n - 1$, a contradiction.

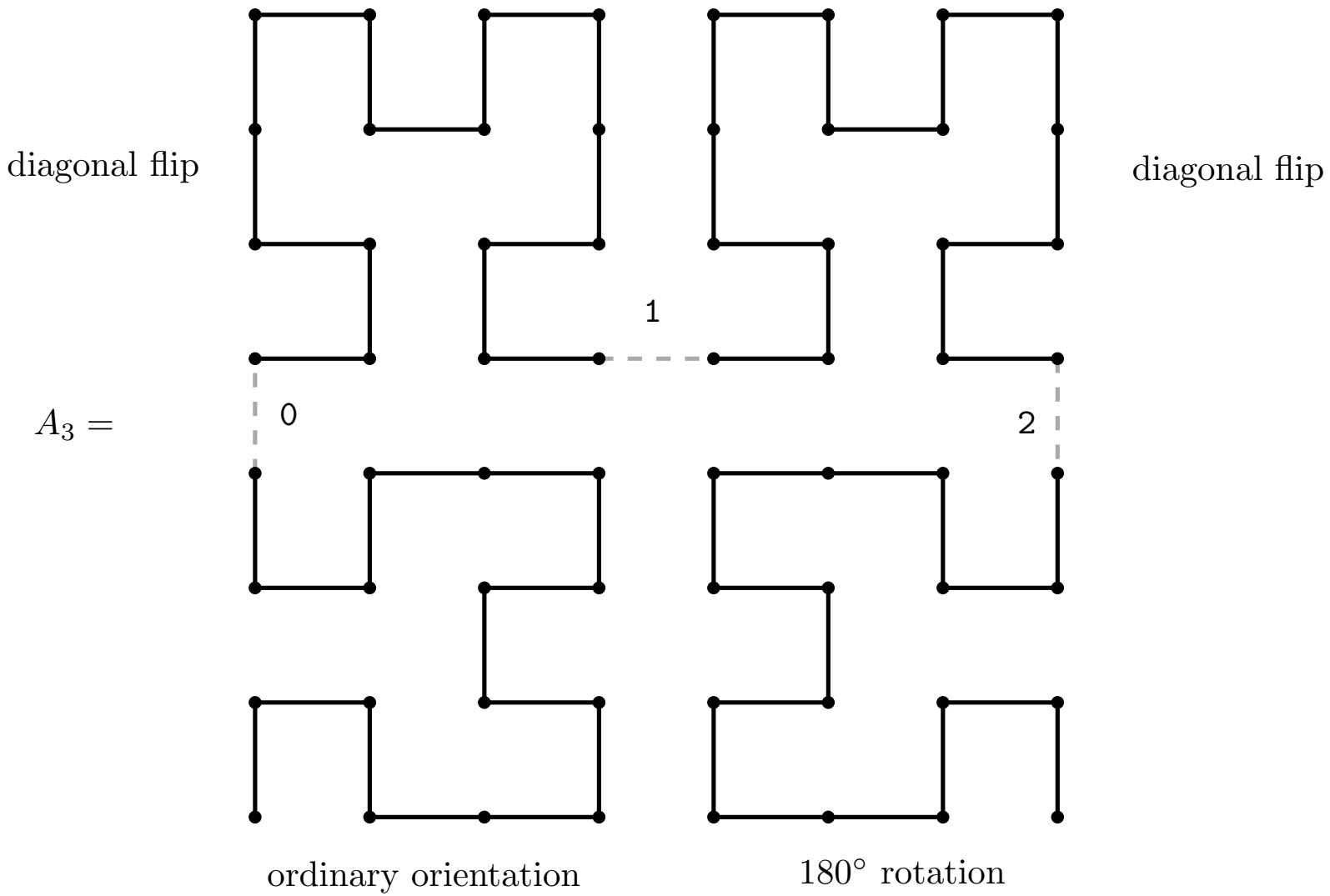
Lagniappe: Hilbert's spacefilling curve

Hilbert's spacefilling curve can be viewed as a curve that traverses all the points in $\mathbb{N} \times \mathbb{N}$, each lattice point visited exactly once.

It can be constructed by an iterative process, where we join four copies of the previous iteration.



Hilbert's spacefilling curve



Hilbert's spacefilling curve

Let us write (x_n, y_n) for the n'th lattice point in the Hilbert curve.

n	0	1	2	3	4	5	6	7	8	9	10	11	12
x_n	0	0	1	1	2	3	3	2	2	3	3	2	1
y_n	0	1	1	0	0	0	1	1	2	2	3	3	3

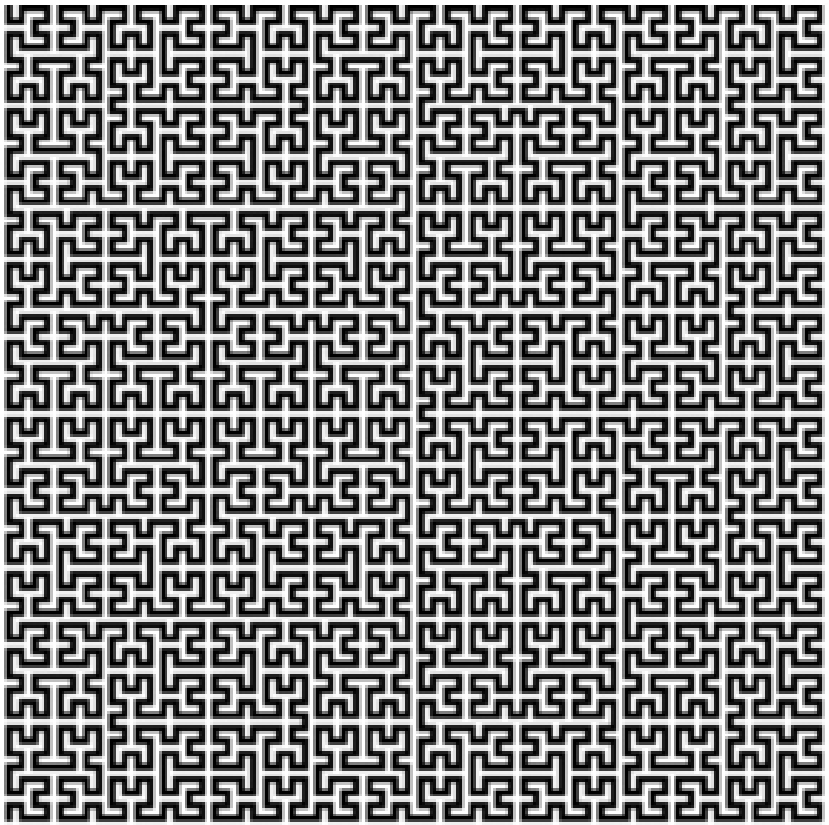
It turns out that $n \to (x_n, y_n)$ is synchronized, but only if we represent things in the right way!

We must represent n in base 4, and x_n and y_n in base 2.

In other words, (n, x_n, y_n) is $(4, 2, 2)$-synchronized, with a 10-state automaton.

Hilbert's spacefilling curve

With this synchronized automaton, we can easily construct a two-dimensional automatic sequence containing a bitmap image of the Hilbert curve:



A final word

Thanks to Arturo Carpi and his co-authors for this wonderful concept of synchronized sequence!