Dyck Words, Pattern Avoidance, and Automatic Sequences

Lucas Mol

Joint work with Narad Rampersad (Winnipeg) and Jeffrey Shallit (Waterloo)

TRU Mathematics and Statistics Seminar
September 2023

WORDS 2023 Conference, Umeå, Sweden

PLAN

Combinatorics on words

- An alphabet is a finite set of letters, treated simply as symbols, e.g.,
  - $\{\mathtt{a}, \mathtt{b}, \mathtt{c}, \ldots, \mathtt{z}\}$ (the English alphabet)
  - $\{\mathtt{0}, \mathtt{1}\}$ (the binary alphabet)
  - $\{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$ (the DNA alphabet)

COMBINATORICS ON WORDS

- An alphabet is a finite set of letters, treated simply as symbols, e.g.,
  - $\{\mathtt{a}, \mathtt{b}, \mathtt{c}, \ldots, \mathtt{z}\}$ (the English alphabet)
  - $\{\mathtt{0}, \mathtt{1}\}$ (the binary alphabet)
  - $\{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$ (the DNA alphabet)
- A word is a sequence of letters taken from some alphabet, e.g.,
  - $\mathtt{apple}, \mathtt{banana}, \mathtt{clementine}$ (English words)
  - $\mathtt{0110100110010110}$ (a binary word)
  - $\mathtt{AAGATGCCGT}$ (a DNA string)

Combinatorics on words

- An alphabet is a finite set of letters, treated simply as symbols, e.g.,
  - $\{\mathtt{a}, \mathtt{b}, \mathtt{c}, \ldots, \mathtt{z}\}$ (the English alphabet)
  - $\{\mathtt{0}, \mathtt{1}\}$ (the binary alphabet)
  - $\{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$ (the DNA alphabet)
- A word is a sequence of letters taken from some alphabet, e.g.,
  - `apple`, `banana`, `clementine` (English words)
  - `0110100110010110` (a binary word)
  - `AAGATGCCGT` (a DNA string)
- We are mostly interested in long words over small alphabets.

COMBINATORICS ON WORDS

- An alphabet is a finite set of letters, treated simply as symbols, e.g.,
  - $\{\mathtt{a}, \mathtt{b}, \mathtt{c}, \ldots, \mathtt{z}\}$ (the English alphabet)
  - $\{\mathtt{0}, \mathtt{1}\}$ (the binary alphabet)
  - $\{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$ (the DNA alphabet)
- A word is a sequence of letters taken from some alphabet, e.g.,
  - `apple`, `banana`, `clementine` (English words)
  - `0110100110010110` (a binary word)
  - `AAGATGCCGT` (a DNA string)
- We are mostly interested in long words over small alphabets.
- Which patterns can be avoided, and which patterns must inevitably occur?

Squares and square-free words

- A square is a word of the form *xx*, e.g.,
  - `murmur`, `hotshots`, `caracara`
  - `00`, `010212010212`

Squares and square-free words

- A square is a word of the form *xx*, e.g.,
  - `murmur`, `hotshots`, `caracara`
  - `00`, `010212010212`
- The factors of a word are its contiguous subwords.

SQUARES AND SQUARE-FREE WORDS

- A square is a word of the form *xx*, e.g.,
  - `murmur`, `hotshots`, `caracara`
  - `00`, `010212010212`
- The factors of a word are its contiguous subwords.
  - e.g. The word `0110` has factors:

    `0`,

Squares and square-free words

- A square is a word of the form xx, e.g.,
  - murmur, hotshots, caracara
  - 00, 010212010212
- The factors of a word are its contiguous subwords.
  - e.g. The word 0110 has factors:

    0, 1,

SQUARES AND SQUARE-FREE WORDS

- A square is a word of the form xx, e.g.,
  - murmur, hotshots, caracara
  - 00, 010212010212
- The factors of a word are its contiguous subwords.
  - e.g. The word 0110 has factors:

    0, 1, 01,

Squares and square-free words

- A square is a word of the form *xx*, e.g.,
  - `murmur`, `hotshots`, `caracara`
  - `00`, `010212010212`
- The factors of a word are its contiguous subwords.
  - e.g. The word `0110` has factors:

    `0`, `1`, `01`, `11`,

Squares and square-free words

- A square is a word of the form xx, e.g.,
  - murmur, hotshots, caracara
  - 00, 010212010212
- The factors of a word are its contiguous subwords.
  - e.g. The word 0110 has factors:

    0, 1, 01, 11, 10,

Squares and square-free words

- A square is a word of the form xx, e.g.,
  - murmur, hotshots, caracara
  - 00, 010212010212
- The factors of a word are its contiguous subwords.
  - e.g. The word 0110 has factors:

    0, 1, 01, 11, 10, 011,

SQUARES AND SQUARE-FREE WORDS

- A square is a word of the form *xx*, e.g.,
  - `murmur`, `hotshots`, `caracara`
  - `00`, `010212010212`
- The factors of a word are its contiguous subwords.
  - e.g. The word `0110` has factors:

    `0`, `1`, `01`, `11`, `10`, `011`, `110`,

SQUARES AND SQUARE-FREE WORDS

- A square is a word of the form xx, e.g.,
  - murmur, hotshots, caracara
  - 00, 010212010212
- The factors of a word are its contiguous subwords.
  - e.g. The word 0110 has factors:

    0, 1, 01, 11, 10, 011, 110, and 0110

SQUARES AND SQUARE-FREE WORDS

- A square is a word of the form xx, e.g.,
  - `murmur`, `hotshots`, `caracara`
  - `00`, `010212010212`
- The factors of a word are its contiguous subwords.
  - e.g. The word `0110` has factors:

    `0`, `1`, `01`, `11`, `10`, `011`, `110`, and `0110`, but NOT `00`.

Squares and square-free words

- A square is a word of the form xx, e.g.,
  - `murmur`, `hotshots`, `caracara`
  - `00`, `010212010212`
- The factors of a word are its contiguous subwords.
  - e.g. The word `0110` has factors:

    `0`, `1`, `01`, `11`, `10`, `011`, `110`, and `0110`, but NOT `00`.

SQUARES AND SQUARE-FREE WORDS

- A square is a word of the form *xx*, e.g.,
  - `murmur`, `hotshots`, `caracara`
  - `00`, `010212010212`
- The factors of a word are its contiguous subwords.
  - e.g. The word `0110` has factors:
    `0`, `1`, `01`, `11`, `10`, `011`, `110`, and `0110`, but NOT `00`.
- A word is square-free if it contains no squares as factors.
  - `apple`

SQUARES AND SQUARE-FREE WORDS

- A square is a word of the form xx, e.g.,
  - murmur, hotshots, caracara
  - 00, 010212010212
- The factors of a word are its contiguous subwords.
  - e.g. The word 0110 has factors:
    0, 1, 01, 11, 10, 011, 110, and 0110, but NOT 00.
- A word is square-free if it contains no squares as factors.
  - apple – not square-free

SQUARES AND SQUARE-FREE WORDS

- A square is a word of the form *xx*, e.g.,
  - `murmur`, `hotshots`, `caracara`
  - `00`, `010212010212`
- The factors of a word are its contiguous subwords.
  - e.g. The word `0110` has factors:
    `0`, `1`, `01`, `11`, `10`, `011`, `110`, and `0110`, but NOT `00`.
- A word is square-free if it contains no squares as factors.
  - `apple` – not square-free
  - `banana`

Squares and square-free words

- A square is a word of the form xx, e.g.,
  - murmur, hotshots, caracara
  - 00, 010212010212
- The factors of a word are its contiguous subwords.
  - e.g. The word 0110 has factors:
    0, 1, 01, 11, 10, 011, 110, and 0110, but NOT 00.
- A word is square-free if it contains no squares as factors.
  - apple – not square-free
  - banana – not square-free

Squares and square-free words

- A square is a word of the form xx, e.g.,
  - murmur, hotshots, caracara
  - 00, 010212010212
- The factors of a word are its contiguous subwords.
  - e.g. The word 0110 has factors:
    0, 1, 01, 11, 10, 011, 110, and 0110, but NOT 00.
- A word is square-free if it contains no squares as factors.
  - apple – not square-free
  - banana – not square-free
  - clementine

SQUARES AND SQUARE-FREE WORDS

- A square is a word of the form *xx*, e.g.,
  - `murmur`, `hotshots`, `caracara`
  - `00`, `010212010212`
- The factors of a word are its contiguous subwords.
  - e.g. The word `0110` has factors:
    `0`, `1`, `01`, `11`, `10`, `011`, `110`, and `0110`, but NOT `00`.
- A word is square-free if it contains no squares as factors.
  - `apple` – not square-free
  - `banana` – not square-free
  - `clementine` – square-free

SQUARES AND SQUARE-FREE WORDS

- A square is a word of the form *xx*, e.g.,
  - `murmur`, `hotshots`, `caracara`
  - `00`, `010212010212`
- The factors of a word are its contiguous subwords.
  - e.g. The word `0110` has factors:
    `0`, `1`, `01`, `11`, `10`, `011`, `110`, and `0110`, but NOT `00`.
- A word is square-free if it contains no squares as factors.
  - `apple` – not square-free
  - `banana` – not square-free
  - `clementine` – square-free
- Theorem (Thue,1906): There are arbitrarily long square-free words over an alphabet of just three letters!

Dyck Words

- A Dyck word is a string of balanced parentheses.
  - 0 – left paren
  - 1 – right paren

  E.g.,
  - $001011 = (()())$ is Dyck
  - $0110 = ())($ is not
- Formally, x is Dyck if
  - $x = \varepsilon$,
  - $x = 0y1$ for some Dyck word y, or
  - $x = yz$ for some Dyck words y and z.
- Dyck words have been well-studied, but NOT in the context of combinatorics on words.

BALANCE AND NESTING LEVEL

- The balance of x is defined by

$$B(x) = |x|_0 - |x|_1.$$

- The word x is Dyck if and only if

$$B(x) = 0 \text{ and } B(x') \geq 0 \text{ for all prefixes } x' \text{ of } x.$$

- The nesting level of a Dyck word x, denoted $N(x)$, is the deepest level of parenthesis nesting in x, e.g.,

$$N(001011) = 2.$$

- More generally,

$$N(x) = \max\{B(x') : \ x' \text{ is a prefix of } x\}.$$

QUESTIONS

- What repetitions must appear in long Dyck words? What repetitions can be avoided? What is the relationship between avoidable repetitions and nesting level?
- Can `Walnut` be used to prove statements about the Dyck factors of certain automatic sequences?

Plan

REPETITIONS

- The exponent of a word is its length divided by its smallest period, e.g.,
  - $\texttt{murmur} = (\texttt{mur})^2$ has exponent 2
  - $\texttt{alfalfa} = (\texttt{alf})^{7/3}$ has exponent $7/3$

Repetitions

- The exponent of a word is its length divided by its smallest period, e.g.,
  - $\texttt{murmur} = (\texttt{mur})^2$ has exponent 2
  - $\texttt{alfalfa} = (\texttt{alf})^{7/3}$ has exponent $7/3$
- Pop quiz: The word `educated` has exponent...

Repetitions

- The exponent of a word is its length divided by its smallest period, e.g.,
  - $\texttt{murmur} = (\texttt{mur})^2$ has exponent 2
  - $\texttt{alfalfa} = (\texttt{alf})^{7/3}$ has exponent $7/3$
- Pop quiz: The word `educated` has exponent... $4/3$

Repetitions

- The exponent of a word is its length divided by its smallest period, e.g.,
  - $\texttt{murmur} = (\texttt{mur})^2$ has exponent 2
  - $\texttt{alfalfa} = (\texttt{alf})^{7/3}$ has exponent $7/3$
- Pop quiz: The word `educated` has exponent... $4/3$
- A word is α-power-free if it contains no factors of exponent greater than or equal to α.
- A word is α^+-power-free if it contains no factors of exponent greater than α.

Repetitions

- The exponent of a word is its length divided by its smallest period, e.g.,
  - $\texttt{murmur} = (\texttt{mur})^2$ has exponent 2
  - $\texttt{alfalfa} = (\texttt{alf})^{7/3}$ has exponent $7/3$
- Pop quiz: The word `educated` has exponent... $4/3$
- A word is α-power-free if it contains no factors of exponent greater than or equal to α.
- A word is α^+-power-free if it contains no factors of exponent greater than α.
- Pop quiz: Is the word

$$\texttt{01101001}$$

  - 2-power-free?

Repetitions

- The exponent of a word is its length divided by its smallest period, e.g.,
  - $\texttt{murmur} = (\texttt{mur})^2$ has exponent 2
  - $\texttt{alfalfa} = (\texttt{alf})^{7/3}$ has exponent $7/3$
- Pop quiz: The word `educated` has exponent... $4/3$
- A word is α-power-free if it contains no factors of exponent greater than or equal to α.
- A word is α^+-power-free if it contains no factors of exponent greater than α.
- Pop quiz: Is the word

$$\texttt{01101001}$$

  - 2-power-free? No.

Repetitions

- The exponent of a word is its length divided by its smallest period, e.g.,
  - `murmur` $=$ (`mur`)^2 has exponent 2
  - `alfalfa` $=$ (`alf`)$^{7/3}$ has exponent $7/3$
- Pop quiz: The word `educated` has exponent... $4/3$
- A word is α-power-free if it contains no factors of exponent greater than or equal to α.
- A word is α^+-power-free if it contains no factors of exponent greater than α.
- Pop quiz: Is the word

`01101001`

  - 2-power-free? No.
  - 2^+-power-free?

Repetitions

- The exponent of a word is its length divided by its smallest period, e.g.,
  - $\texttt{murmur} = (\texttt{mur})^2$ has exponent 2
  - $\texttt{alfalfa} = (\texttt{alf})^{7/3}$ has exponent $7/3$
- Pop quiz: The word `educated` has exponent... $4/3$
- A word is α-power-free if it contains no factors of exponent greater than or equal to α.
- A word is α^+-power-free if it contains no factors of exponent greater than α.
- Pop quiz: Is the word

$$\texttt{01101001}$$

  - 2-power-free? No.
  - 2^+-power-free? Yes!

Repetitions in binary words

Q: Are there arbitrarily long 2-power-free binary words?

Repetitions in binary words

Q: Are there arbitrarily long 2-power-free binary words?

A: No!

Repetitions in binary words

Q: Are there arbitrarily long 2-power-free binary words?
A: No!

Q: Are there arbitrarily long 2^+-power-free binary words?

Repetitions in binary words

Q: Are there arbitrarily long 2-power-free binary words?
A: No!

Q: Are there arbitrarily long 2^+-power-free binary words?
A: Hmmmm...

A construction

A CONSTRUCTION

- Define μ by $\mu(\texttt{0}) = \texttt{01}$ and $\mu(\texttt{1}) = \texttt{10}$.

A CONSTRUCTION

- Define μ by $\mu(\texttt{0}) = \texttt{01}$ and $\mu(\texttt{1}) = \texttt{10}$.
- Extend μ to all words over $\{\texttt{0}, \texttt{1}\}$ in the obvious way, e.g.,

$$\mu(\texttt{010}) = \mu(\texttt{0})\mu(\texttt{1})\mu(\texttt{0}) = \texttt{011001}$$

A CONSTRUCTION

- Define μ by $\mu(\mathtt{0}) = \mathtt{01}$ and $\mu(\mathtt{1}) = \mathtt{10}$.
- Extend μ to all words over $\{\mathtt{0}, \mathtt{1}\}$ in the obvious way, e.g.,

$$\mu(\mathtt{010}) = \mu(\mathtt{0})\mu(\mathtt{1})\mu(\mathtt{0}) = \mathtt{011001}$$

- The map μ is called a morphism.

A CONSTRUCTION

- Define μ by $\mu(\mathtt{0}) = \mathtt{01}$ and $\mu(\mathtt{1}) = \mathtt{10}$.
- Extend μ to all words over $\{\mathtt{0}, \mathtt{1}\}$ in the obvious way, e.g.,

$$\mu(\mathtt{010}) = \mu(\mathtt{0})\mu(\mathtt{1})\mu(\mathtt{0}) = \mathtt{011001}$$

- The map μ is called a morphism.
- Start with $\mathtt{0}$, and repeatedly apply μ:

A CONSTRUCTION

- Define μ by $\mu(\texttt{0}) = \texttt{01}$ and $\mu(\texttt{1}) = \texttt{10}$.
- Extend μ to all words over $\{\texttt{0}, \texttt{1}\}$ in the obvious way, e.g.,

$$\mu(\texttt{010}) = \mu(\texttt{0})\mu(\texttt{1})\mu(\texttt{0}) = \texttt{011001}$$

- The map μ is called a morphism.
- Start with $\texttt{0}$, and repeatedly apply μ:

$$\mu(\texttt{0}) = \texttt{01}$$

A CONSTRUCTION

- Define μ by $\mu(\texttt{0}) = \texttt{01}$ and $\mu(\texttt{1}) = \texttt{10}$.
- Extend μ to all words over $\{\texttt{0}, \texttt{1}\}$ in the obvious way, e.g.,

$$\mu(\texttt{010}) = \mu(\texttt{0})\mu(\texttt{1})\mu(\texttt{0}) = \texttt{011001}$$

- The map μ is called a morphism.
- Start with $\texttt{0}$, and repeatedly apply μ:

$$\mu(\texttt{0}) = \texttt{01}$$
$$\mu^2(\texttt{0}) = \texttt{0110}$$

A CONSTRUCTION

- Define μ by $\mu(\mathtt{0}) = \mathtt{01}$ and $\mu(\mathtt{1}) = \mathtt{10}$.
- Extend μ to all words over $\{\mathtt{0}, \mathtt{1}\}$ in the obvious way, e.g.,

$$\mu(\mathtt{010}) = \mu(\mathtt{0})\mu(\mathtt{1})\mu(\mathtt{0}) = \mathtt{011001}$$

- The map μ is called a morphism.
- Start with $\mathtt{0}$, and repeatedly apply μ:

$$\begin{aligned} \mu(\mathtt{0}) &= \mathtt{01} \\ \mu^2(\mathtt{0}) &= \mathtt{0110} \\ \mu^3(\mathtt{0}) &= \mathtt{01101001} \end{aligned}$$

A CONSTRUCTION

- Define μ by $\mu(\texttt{0}) = \texttt{01}$ and $\mu(\texttt{1}) = \texttt{10}$.
- Extend μ to all words over $\{\texttt{0}, \texttt{1}\}$ in the obvious way, e.g.,

$$\mu(\texttt{010}) = \mu(\texttt{0})\mu(\texttt{1})\mu(\texttt{0}) = \texttt{011001}$$

- The map μ is called a morphism.
- Start with $\texttt{0}$, and repeatedly apply μ:

$$\begin{aligned} \mu(\texttt{0}) &= \texttt{01} \\ \mu^2(\texttt{0}) &= \texttt{0110} \\ \mu^3(\texttt{0}) &= \texttt{01101001} \\ \mu^4(\texttt{0}) &= \texttt{0110100110010110} \end{aligned}$$

A CONSTRUCTION

- Define μ by $\mu(\texttt{0}) = \texttt{01}$ and $\mu(\texttt{1}) = \texttt{10}$.
- Extend μ to all words over $\{\texttt{0}, \texttt{1}\}$ in the obvious way, e.g.,

$$\mu(\texttt{010}) = \mu(\texttt{0})\mu(\texttt{1})\mu(\texttt{0}) = \texttt{011001}$$

- The map μ is called a morphism.
- Start with $\texttt{0}$, and repeatedly apply μ:

$$\begin{aligned}
\mu(\texttt{0}) &= \texttt{01} \\
\mu^2(\texttt{0}) &= \texttt{0110} \\
\mu^3(\texttt{0}) &= \texttt{01101001} \\
\mu^4(\texttt{0}) &= \texttt{0110100110010110} \\
&\vdots \\
\mu^\omega(\texttt{0}) &= \texttt{0110100110010110}\cdots
\end{aligned}$$

The Thue-Morse word (TM)

$$\mu^{\omega}(0) = 0110100110010110 \cdots$$

The Thue-Morse word (TM)

$$\mu^{\omega}(\texttt{0}) = \texttt{0110100110010110}\cdots$$

- Theorem (Thue, 1912): TM is 2^+-power-free.

THE THUE-MORSE WORD (TM)

$$\mu^{\omega}(0) = 0110100110010110\cdots$$

- Theorem (Thue, 1912): TM is 2^{+}-power-free.
- Notice: TM has many squares, but every square is followed by a letter that breaks the repetition.

THE THUE-MORSE WORD (TM)

$$\mu^{\omega}(0) = 0110100110010110\cdots$$

- Theorem (Thue, 1912): TM is 2^+-power-free.
- Notice: TM has many squares, but every square is followed by a letter that breaks the repetition.

The Thue-Morse word (TM)

$$\mu^{\omega}(0) = 0110100110010110\cdots$$

- Theorem (Thue, 1912): TM is 2^{+}-power-free.
- Notice: TM has many squares, but every square is followed by a letter that breaks the repetition.

The Thue-Morse word (TM)

$$\mu^{\omega}(0) = 0110100110010110\cdots$$

- Theorem (Thue, 1912): TM is 2^{+}-power-free.
- Notice: TM has many squares, but every square is followed by a letter that breaks the repetition.

The Thue-Morse word (TM)

$$\mu^{\omega}(\mathtt{0}) = \mathtt{0110100110010110}\cdots$$

- Theorem (Thue, 1912): TM is 2^+-power-free.
- Notice: TM has many squares, but every square is followed by a letter that breaks the repetition.

The Thue-Morse word (TM)

$$\mu^\omega(0) = 0110100110010110\cdots$$

- Theorem (Thue, 1912): TM is 2^+-power-free.
- Notice: TM has many squares, but every square is followed by a letter that breaks the repetition.

The Thue-Morse word (TM)

$$\mu^{\omega}(\texttt{0}) = \texttt{0110100110010110}\cdots$$

- Theorem (Thue, 1912): TM is 2^+-power-free.
- Notice: TM has many squares, but every square is followed by a letter that breaks the repetition.

Plan

QUESTION

- What repetitions must appear in long Dyck words? What repetitions can be avoided? What is the relationship between avoidable repetitions and nesting level?

Theorem: A characterization of 2^+-power-free Dyck words.

Corollary: There are arbitrarily long 2^+-power-free Dyck words.

Sketch of Proof:

- Define g by

$$g(\mathtt{0}) = \mathtt{012}, \quad g(\mathtt{1}) = \mathtt{02}, \quad \text{and} \quad g(\mathtt{2}) = \mathtt{1},$$

  and let $\mathbf{s} = g^{\omega}(\mathbf{0}) = \mathtt{012021012102}\cdots$
- Define h by

$$h(\mathtt{0}) = \mathtt{01}, \quad h(\mathtt{1}) = \mathtt{0011}, \quad \text{and} \quad h(\mathtt{2}) = \mathtt{001011}.$$

- Let x be a prefix of $\mathbf{s}$ ending in 10.
- Then $h(x)$ and $0h(x)1$ are 2^+-power-free Dyck words.

Note: These words have nesting level at most 3.

Theorem: If w is a $\frac{7}{3}$-power-free Dyck word, then $N(w) \leq 3$.

Theorem: There are $\frac{7}{3}^{+}$-power-free Dyck words of every nesting level.

Idea of Proof:

- We sketch the simpler proof that there are *cube-free* Dyck words of every nesting level.
- Define f by $f(\texttt{0}) = \texttt{001}$ and $f(\texttt{1}) = \texttt{011}$.
- It is well-known that f preserves cube-freeness.
- Applying f preserves the Dyck property, and increases the nesting level by one.
- By induction, for all $t \geq 0$, the word $f^t(01)$ is a cube-free Dyck word of nesting level $t + 1$.

Summary: Repetitions and Dyck words

- There are arbitrarily long 2^+-power-free Dyck words, but they have small nesting level.
- Dyck words of large nesting levels only become attainable when we allow $7/3$-powers.

Plan

Introduction

Repetitions

Repetitions and Dyck words

Automatic sequences and `Walnut`

Dyck factors of some automatic sequences

AUTOMATIC SEQUENCES

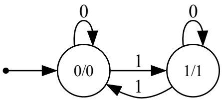
$$\mu^{\omega}(0) = 0110100110010110\cdots$$

- It turns out that the Thue-Morse word is the prototypical example of an automatic sequence.

AUTOMATIC SEQUENCES

$$\mu^\omega(0) = 0110100110010110\cdots$$

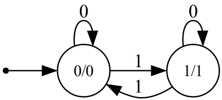
- It turns out that the Thue-Morse word is the prototypical example of an automatic sequence.
- We have seen its definition in terms of the morphism μ, but it can also be defined in terms of the following automaton.



Automatic Sequences

$$\mu^\omega(0) = 0110100110010110\cdots$$

- It turns out that the Thue-Morse word is the prototypical example of an automatic sequence.
- We have seen its definition in terms of the morphism μ, but it can also be defined in terms of the following automaton.

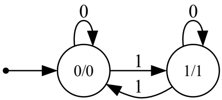


- To get the letter at position *n* in TM, just feed the binary representation of *n* into this automaton.

Automatic Sequences

$$\mu^\omega(0) = 0110100110010110\cdots$$

- It turns out that the Thue-Morse word is the prototypical example of an automatic sequence.
- We have seen its definition in terms of the morphism μ, but it can also be defined in terms of the following automaton.

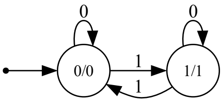


- To get the letter at position n in TM, just feed the binary representation of n into this automaton.
  - $T[0] = T[(0)_2] = 0$

AUTOMATIC SEQUENCES

$$\mu^\omega(\texttt{0}) = \texttt{0110100110010110}\cdots$$

- It turns out that the Thue-Morse word is the prototypical example of an automatic sequence.
- We have seen its definition in terms of the morphism μ, but it can also be defined in terms of the following automaton.

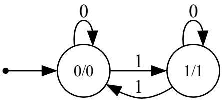


- To get the letter at position *n* in TM, just feed the binary representation of *n* into this automaton.
  - $T[0] = T[(0)_2] = \texttt{0}$
  - $T[1] = T[(1)_2] = \texttt{1}$

AUTOMATIC SEQUENCES

$$\mu^\omega(0) = 0110100110010110\cdots$$

- It turns out that the Thue-Morse word is the prototypical example of an automatic sequence.
- We have seen its definition in terms of the morphism μ, but it can also be defined in terms of the following automaton.

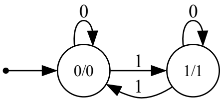


- To get the letter at position n in TM, just feed the binary representation of n into this automaton.
  - $T[0] = T[(0)_2] = 0$
  - $T[1] = T[(1)_2] = 1$
  - $T[2] = T[(10)_2] = 1$

Automatic Sequences

$$\mu^\omega(0) = 0110100110010110\cdots$$

- It turns out that the Thue-Morse word is the prototypical example of an automatic sequence.
- We have seen its definition in terms of the morphism μ, but it can also be defined in terms of the following automaton.



- To get the letter at position n in TM, just feed the binary representation of n into this automaton.
  - $T[0] = T[(0)_2] = 0$
  - $T[1] = T[(1)_2] = 1$
  - $T[2] = T[(10)_2] = 1$
  - $T[3] = T[(11)_2] = 0$

AUTOMATIC THEOREM-PROVING

- `Walnut` is a software program that can be used to prove statements, written in a certain first-order logic, about automatic sequences.

AUTOMATIC THEOREM-PROVING

- `Walnut` is a software program that can be used to prove statements, written in a certain first-order logic, about automatic sequences.
- For example, to show that TM is 2^+-power-free, we enter

```
eval TMHasOverlap "Ei,n (n>=1) & Ak (k<=n) => T[i+k]=T[i+k+n]":
```

AUTOMATIC THEOREM-PROVING

- `Walnut` is a software program that can be used to prove statements, written in a certain first-order logic, about automatic sequences.
- For example, to show that TM is 2^+-power-free, we enter

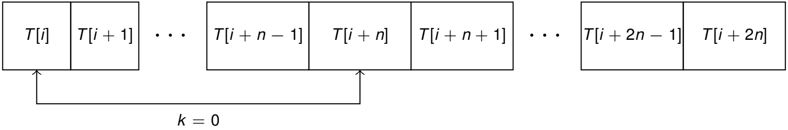
```
eval TMHasOverlap "Ei,n (n>=1) & Ak (k<=n) => T[i+k]=T[i+k+n]":
```

$T[i]$	$T[i+1]$	$\cdots$	$T[i+n-1]$	$T[i+n]$	$T[i+n+1]$	$\cdots$	$T[i+2n-1]$	$T[i+2n]$

AUTOMATIC THEOREM-PROVING

- Walnut is a software program that can be used to prove statements, written in a certain first-order logic, about automatic sequences.
- For example, to show that TM is 2^+-power-free, we enter

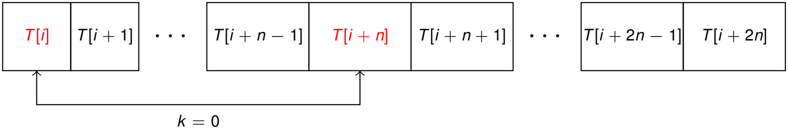
```
eval TMHasOverlap "Ei,n (n>=1) & Ak (k<=n) => T[i+k]=T[i+k+n]":
```



AUTOMATIC THEOREM-PROVING

- `Walnut` is a software program that can be used to prove statements, written in a certain first-order logic, about automatic sequences.
- For example, to show that TM is 2^+-power-free, we enter

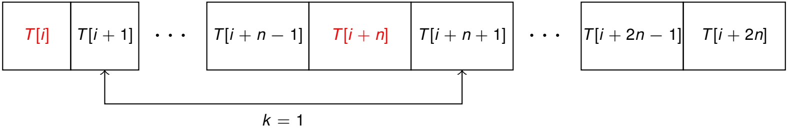
```
eval TMHasOverlap "Ei,n (n>=1) & Ak (k<=n) => T[i+k]=T[i+k+n]":
```



AUTOMATIC THEOREM-PROVING

- `Walnut` is a software program that can be used to prove statements, written in a certain first-order logic, about automatic sequences.
- For example, to show that TM is 2^{+}-power-free, we enter

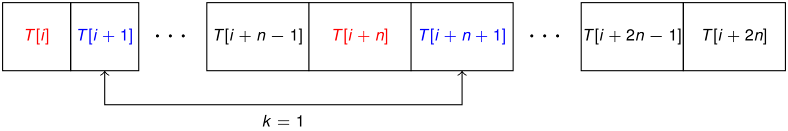
```
eval TMHasOverlap "Ei,n (n>=1) & Ak (k<=n) => T[i+k]=T[i+k+n]":
```



AUTOMATIC THEOREM-PROVING

- Walnut is a software program that can be used to prove statements, written in a certain first-order logic, about automatic sequences.
- For example, to show that TM is 2^+-power-free, we enter

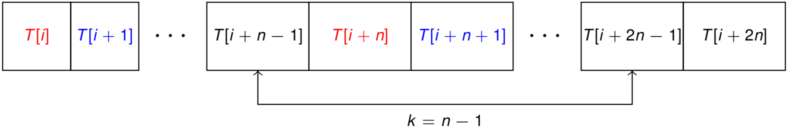
```
eval TMHasOverlap "Ei,n (n>=1) & Ak (k<=n) => T[i+k]=T[i+k+n]":
```



AUTOMATIC THEOREM-PROVING

- `Walnut` is a software program that can be used to prove statements, written in a certain first-order logic, about automatic sequences.
- For example, to show that TM is 2^+-power-free, we enter

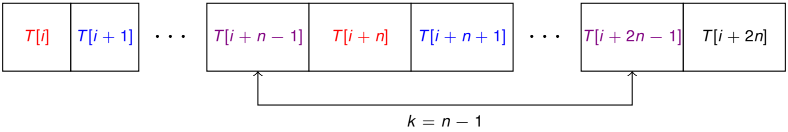
```
eval TMHasOverlap "Ei,n (n>=1) & Ak (k<=n) => T[i+k]=T[i+k+n]":
```



AUTOMATIC THEOREM-PROVING

- `Walnut` is a software program that can be used to prove statements, written in a certain first-order logic, about automatic sequences.
- For example, to show that TM is 2^+-power-free, we enter

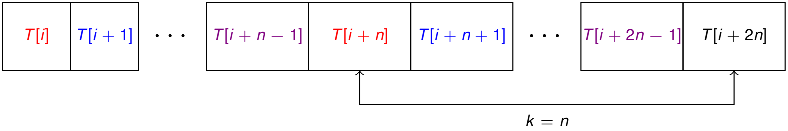
```
eval TMHasOverlap "Ei,n (n>=1) & Ak (k<=n) => T[i+k]=T[i+k+n]":
```



AUTOMATIC THEOREM-PROVING

- `Walnut` is a software program that can be used to prove statements, written in a certain first-order logic, about automatic sequences.
- For example, to show that TM is 2^+-power-free, we enter

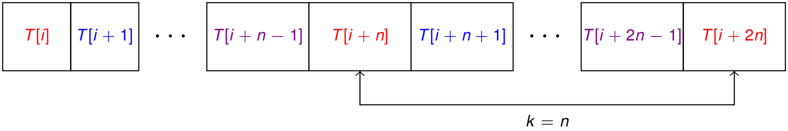
```
eval TMHasOverlap "Ei,n (n>=1) & Ak (k<=n) => T[i+k]=T[i+k+n]":
```



AUTOMATIC THEOREM-PROVING

- `Walnut` is a software program that can be used to prove statements, written in a certain first-order logic, about automatic sequences.
- For example, to show that TM is 2^+-power-free, we enter

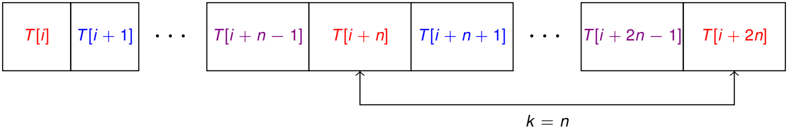
```
eval TMHasOverlap "Ei,n (n>=1) & Ak (k<=n) => T[i+k]=T[i+k+n]":
```



Automatic Theorem-Proving

- `Walnut` is a software program that can be used to prove statements, written in a certain first-order logic, about automatic sequences.
- For example, to show that TM is 2^+-power-free, we enter

```
eval TMHasOverlap "Ei,n (n>=1) & Ak (k<=n) => T[i+k]=T[i+k+n]":
```



- `Walnut` returns `FALSE`.

LENGTHS OF SQUARES IN TM

- To show that TM has a square, we enter
  ```
  eval TMHasSquare "Ei,n (n>=1) & Ak (k<n) => T[i+k]=T[i+k+n]":
  ```
  and `Walnut` returns `TRUE`.

LENGTHS OF SQUARES IN TM

- To show that TM has a square, we enter

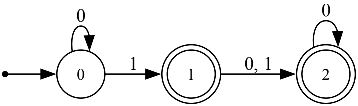
```
eval TMHasSquare "Ei,n (n>=1) & Ak (k<n) => T[i+k]=T[i+k+n]":
```

  and `Walnut` returns `TRUE`.

- When we enter

```
def TMSquareLengths "Ei (n>=1) & Ak (k<n) => T[i+k]=T[i+k+n]":
```

  `Walnut` returns an automaton that accepts all values of n for which TM has a square of length $2n$.



LENGTHS OF SQUARES IN TM

- To show that TM has a square, we enter

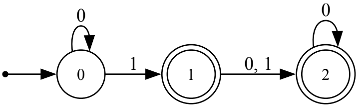
```
eval TMHasSquare "Ei,n (n>=1) & Ak (k<n) => T[i+k]=T[i+k+n]":
```

  and `Walnut` returns `TRUE`.

- When we enter

```
def TMSquareLengths "Ei (n>=1) & Ak (k<n) => T[i+k]=T[i+k+n]":
```

  `Walnut` returns an automaton that accepts all values of *n* for which TM has a square of length 2*n*.



- To show that TM has arbitrarily long squares, we enter

```
eval TMLongSquares "Ak En (n>=k) & $TMSquareLengths(n)":
```

  and `Walnut` returns `TRUE`.

Plan

QUESTION

- Can `Walnut` be used to prove statements about the Dyck factors of automatic sequences?

QUESTION

- Can `Walnut` be used to prove statements about the Dyck factors of automatic sequences?
- Remember that `Walnut` can be used to prove statements, written in a certain first-order logic, about automatic sequences.

Question

- Can `Walnut` be used to prove statements about the Dyck factors of automatic sequences?
- Remember that `Walnut` can be used to prove statements, written in a certain first-order logic, about automatic sequences.
- But the language of Dyck words is not definable in this first-order logic! (Choffrut, Malcher, Mereghetti, Palano, 2012.)

QUESTION

- Can `Walnut` be used to prove statements about the Dyck factors of automatic sequences?
- Remember that `Walnut` can be used to prove statements, written in a certain first-order logic, about automatic sequences.
- But the language of Dyck words is not definable in this first-order logic! (Choffrut, Malcher, Mereghetti, Palano, 2012.)
- So `Walnut` cannot directly handle the Dyck factors of all automatic sequences...

RUNNING-SUM SYNCHRONIZED SEQUENCES

- For an automatic sequence $\mathbf{s} = (s(n))_{n \geq 0}$, define its running-sum sequence by

$$v(n) = \sum_{0 \leq i < n} s(i).$$

- We say that $\mathbf{s}$ is running-sum synchronized if there is an automaton accepting, in parallel, the base-k representations of n and $v(n)$.

Theorem: `Walnut` can handle the Dyck factors of running-sum synchronized sequences!

An Example: Thue-Morse

The Thue-Morse sequence is running-sum synchronized.

$$0\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ \cdots$$

An Example: Thue-Morse

The Thue-Morse sequence is running-sum synchronized.

$$0\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ \cdots$$

$$0$$

An Example: Thue-Morse

The Thue-Morse sequence is running-sum synchronized.

$$0\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ \cdots$$

$$0\ 1$$

An Example: Thue-Morse

The Thue-Morse sequence is running-sum synchronized.

$$0\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ \cdots$$

$$0\ 1\ 2$$

An Example: Thue-Morse

The Thue-Morse sequence is running-sum synchronized.

$$0\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ \cdots$$

$$0\ 1\ 2\ 2$$

An Example: Thue-Morse

The Thue-Morse sequence is running-sum synchronized.

0 1 1 0 1 0 0 1 $\cdots$

0 1 2 2 3

An Example: Thue-Morse

The Thue-Morse sequence is running-sum synchronized.

$$0\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ \cdots$$

$$0\ 1\ 2\ 2\ 3\ 3$$

An Example: Thue-Morse

The Thue-Morse sequence is running-sum synchronized.

0 1 1 0 1 0 0 1 $\cdots$

0 1 2 2 3 3 3

An Example: Thue-Morse

The Thue-Morse sequence is running-sum synchronized.

$$0\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ \cdots$$

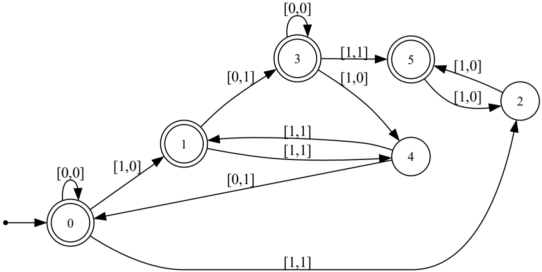
$$0\ 1\ 2\ 2\ 3\ 3\ 3\ 4\ \cdots$$

An Example: Thue-Morse

The Thue-Morse sequence is running-sum synchronized.

$$0\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ \cdots$$

$$0\ 1\ 2\ 2\ 3\ 3\ 3\ 4\ \cdots$$



- $[1,1][1,0]$ is accepted, since $v(3) = 2$.
- $[1,0][1,0]$, $[1,0][1,1]$, and $[1,1][1,1]$ are not accepted!

An Example: Thue-Morse

The automaton on the previous slide was built in `Walnut` as follows:

```
def even "Ek n=2*k":    # accepts even numbers

def odd "Ek n=2*k+1":   # accepts odd numbers

def V "($even(n) & 2*x=n) |
       ($odd(n) & 2*x+1=n & T[n-1]=@0) |
       ($odd(n) & 2*x=n+1 & T[n-1]=@1)":
# accepts n and v(n) in parallel
```

AN EXAMPLE: THUE-MORSE

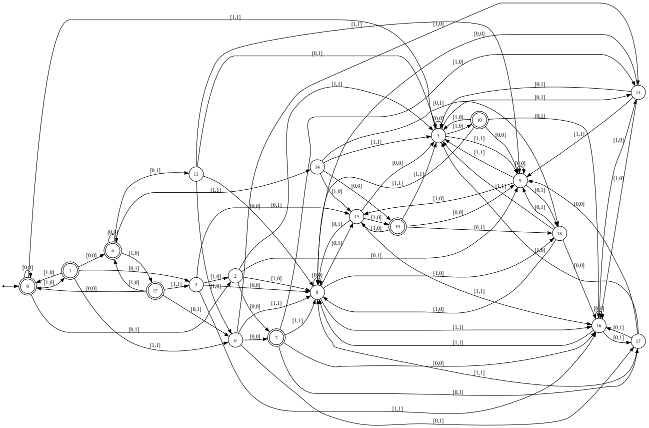
We can now build an automaton that identifies the Dyck factors of Thue-Morse:

```
def N1 "Ey,z $V(i,y) & $V(i+n,z) & x+y=z":
# accepts (i,n,x) if T[i..i+n−1] has x 1's

def N0 "Ey $N1(i,n,y) & n=x+y":
# accepts (i,n,x) if T[i..i+n−1] has x 0's

def Dyck "(Ew $N0(i,n,w) & $N1(i,n,w)) &
          At,y,z (t<n & $N0(i,t,y) & $N1(i,t,z)) => y>=z":
# accepts (i,n) if T[i..i+n−1] is Dyck
```

AN EXAMPLE: THUE-MORSE



The automaton recognizing Dyck factors of Thue-Morse!

AN EXAMPLE: THUE-MORSE

Now we can prove statements about Dyck factors of TM.

- TM has Dyck factors of all even lengths.

  We run the command

  ```
  eval AllLengths "An $even(n) => Ei $Dyck(i,n)":
  ```

  and `Walnut` returns `TRUE`.

- Every Dyck factor of TM has nesting level at most 2.

  We run the commands

  ```
  def Bal "Ey,z $N0(i,n,y) & $N1(i,n,z) &
           ((y<z & x=0) | (y>=z & y=x+z))":
  def Nest "Em (m<n) & $Bal(i,m,x) &
            Ap,y (p<n & $Bal(i,p,y)) => y<=x":
  eval MaxNest "Ai,n,x ($Dyck(i,n) & $Nest(i,n,x)) => x<=2":
  ```

  and `Walnut` returns `TRUE`.

- `Walnut` can also be used to count the Dyck factors of TM!

Summary: Automatic Sequences

- `Walnut` cannot directly handle the Dyck factors of all automatic sequences.
- `Walnut` can handle the Dyck factors of automatic sequences that are running-sum synchronized.

OUTLOOK

Some possible directions for future work:

- Extend to Dyck words with two or more types of parens.
- Develop techniques to recognize/characterize the Dyck factors of words that are not running-sum synchronized.

Jeffrey Shallit and Anatoly Zavyalov have made some progress in these directions!

Thank you!