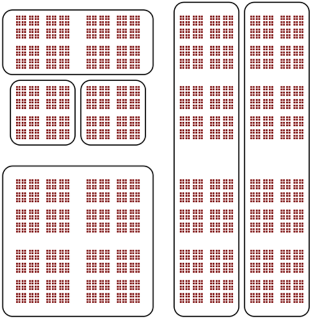
Turing Machines, Automata, and the Brin–Thompson Group $2V$



Jim Belk

Cornell University

Joint Work

Collin Bleak

University of St Andrews

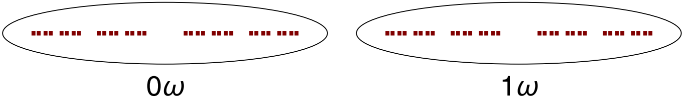
V and $2V$

Thompson's Group V

The ***Cantor set*** C is the infinite product space $\{0,1\}^{\omega}$.

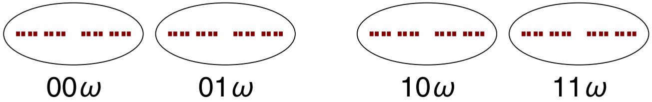
Thompson's Group V

The ***Cantor set*** C is the infinite product space $\{0, 1\}^{\omega}$.



Thompson's Group V

The ***Cantor set*** C is the infinite product space $\{0, 1\}^\omega$.



Thompson's Group V

The ***Cantor set*** C is the infinite product space $\{0, 1\}^{\omega}$.

Thompson's Group V

The ***Cantor set*** C is the infinite product space $\{0,1\}^{\omega}$.

A ***dyadic subdivision*** of C is any subdivision obtained by repeatedly cutting pieces in half.

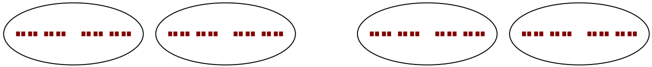
Thompson's Group V

The ***Cantor set*** C is the infinite product space $\{0,1\}^\omega$.

A ***dyadic subdivision*** of C is any subdivision obtained by repeatedly cutting pieces in half.

Thompson's Group V

The ***Cantor set*** C is the infinite product space $\{0,1\}^\omega$.



A ***dyadic subdivision*** of C is any subdivision obtained by repeatedly cutting pieces in half.

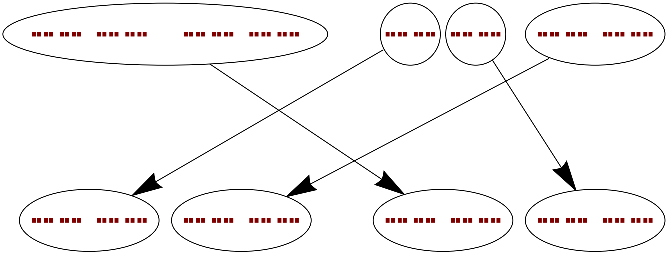
Thompson's Group V

The ***Cantor set*** C is the infinite product space $\{0, 1\}^\omega$.

A ***dyadic subdivision*** of C is any subdivision obtained by repeatedly cutting pieces in half.

Thompson's Group V

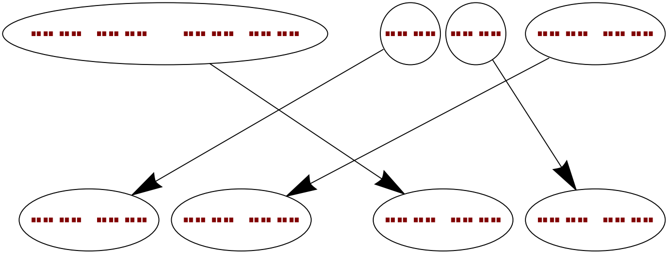
A ***dyadic rearrangement*** of C is a homeomorphism that maps "linearly" between the pieces of two dyadic subdivisions.



The group of all dyadic rearrangements of C is ***Thompson's group V.***

Thompson's Group V

We can describe an element of V using ***prefix replacement rules***

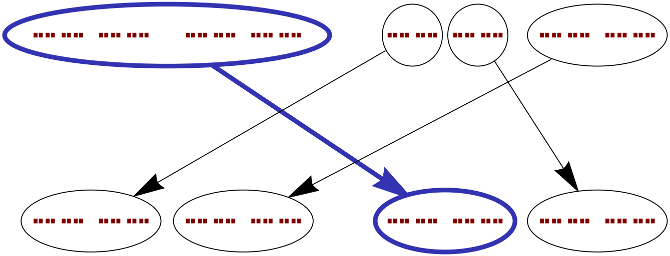


$$0\omega \mapsto 10\omega \qquad 100\omega \mapsto 00\omega$$

$$101\omega \mapsto 11\omega \qquad 11\omega \mapsto 01\omega$$

Thompson's Group V

We can describe an element of V using ***prefix replacement rules***

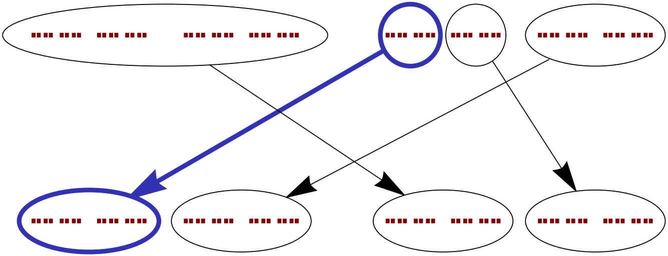


$$\mathbf{0\omega \mapsto 10\omega} \qquad 100\omega \mapsto 00\omega$$

$$101\omega \mapsto 11\omega \qquad 11\omega \mapsto 01\omega$$

Thompson's Group V

We can describe an element of V using ***prefix replacement rules***

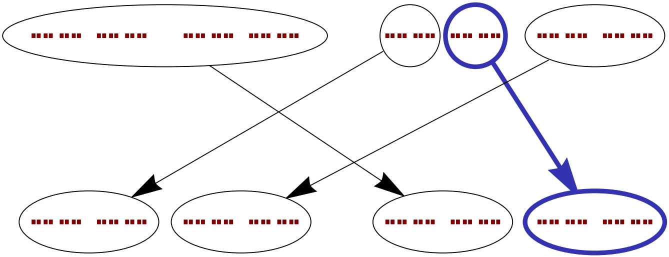


$0\omega \mapsto 10\omega$ **$100\omega \mapsto 00\omega$**

$101\omega \mapsto 11\omega$ $11\omega \mapsto 01\omega$

Thompson's Group V

We can describe an element of V using ***prefix replacement rules***

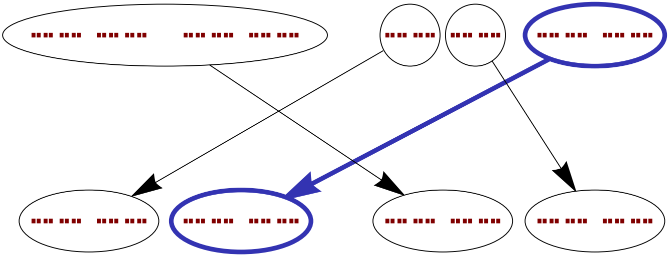


$0\omega \mapsto 10\omega$ $\quad$ $100\omega \mapsto 00\omega$

$\mathbf{101\omega \mapsto 11\omega}$ $\quad$ $11\omega \mapsto 01\omega$

Thompson's Group V

We can describe an element of V using ***prefix replacement rules***

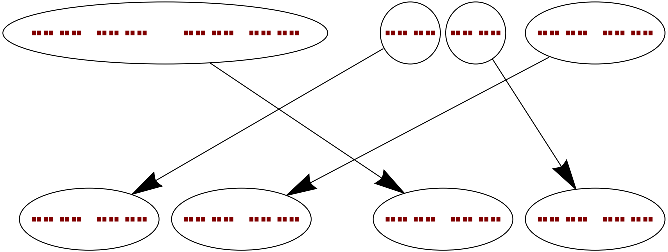


$$0\omega \mapsto 10\omega \qquad 100\omega \mapsto 00\omega$$

$$101\omega \mapsto 11\omega \qquad \mathbf{11\omega \mapsto 01\omega}$$

Thompson's Group V

We can describe an element of V using ***prefix replacement rules***



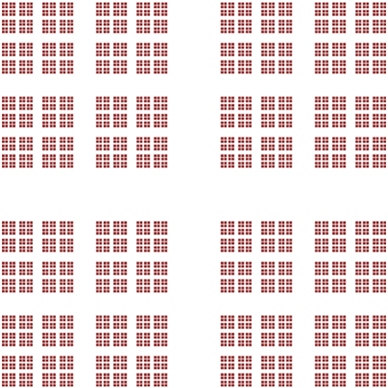
$$0\omega \mapsto 10\omega \qquad 100\omega \mapsto 00\omega$$

$$101\omega \mapsto 11\omega \qquad 11\omega \mapsto 01\omega$$

The Group $2V$

Matt Brin introduced the group $2V$ in 2004.

It acts on the ***Cantor square*** $C \times C$.

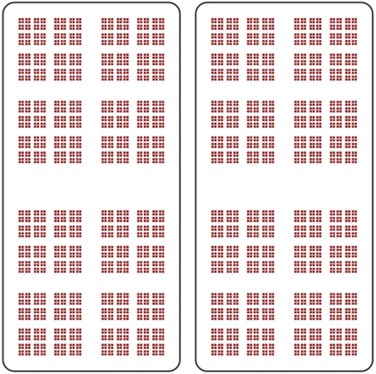


The Group $2V$

We can subdivide horizontally or vertically.

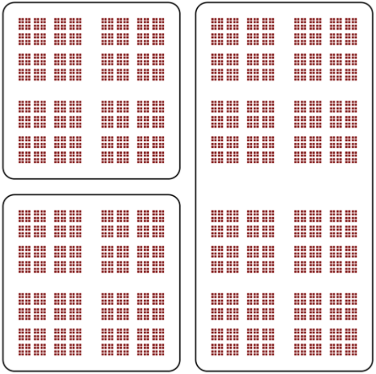
The Group $2V$

We can subdivide horizontally or vertically.



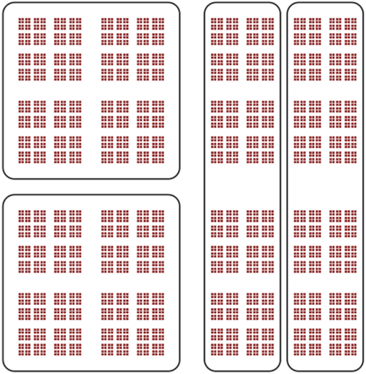
The Group $2V$

We can subdivide horizontally or vertically.



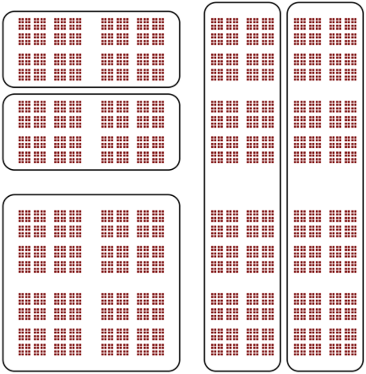
The Group $2V$

We can subdivide horizontally or vertically.



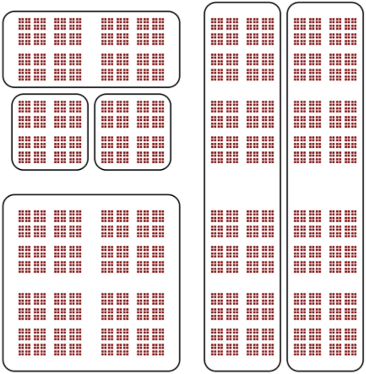
The Group $2V$

We can subdivide horizontally or vertically.



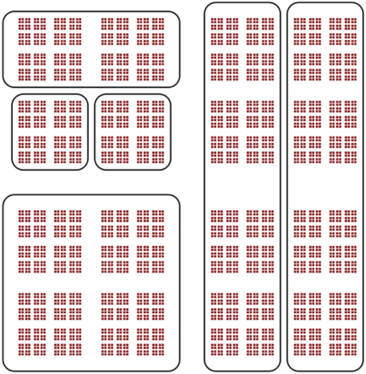
The Group $2V$

We can subdivide horizontally or vertically.



The Group $2V$

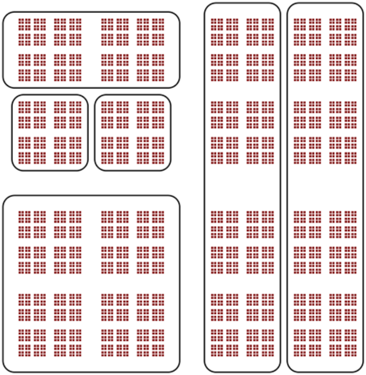
We can subdivide horizontally or vertically.



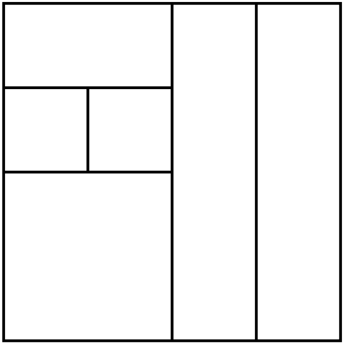
This is a ***dyadic subdivision*** of the Cantor square.

The Group $2V$

Note: It is easier to draw the ***pattern*** for a subdivision.



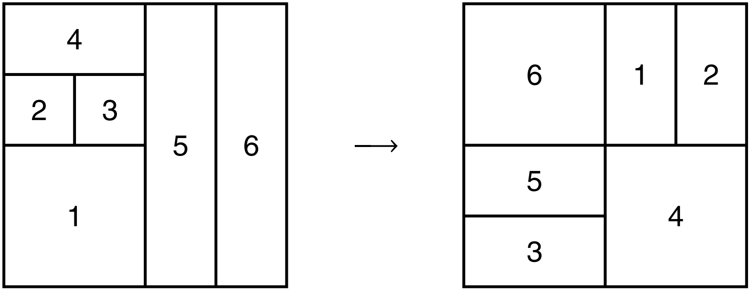
Subdivision



Pattern

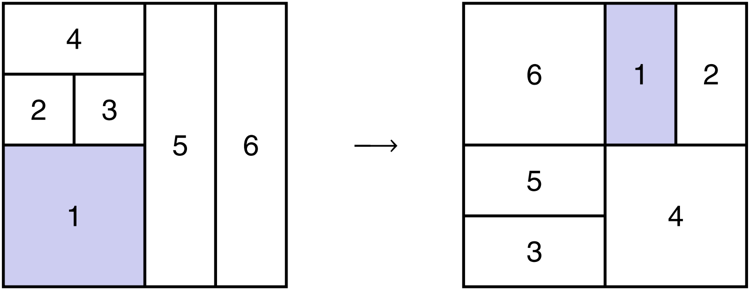
The Group $2V$

Each element of $2V$ maps "linearly" between the rectangles of two dyadic subdivisions.



The Group $2V$

Each piece is a ***prefix pair replacement***



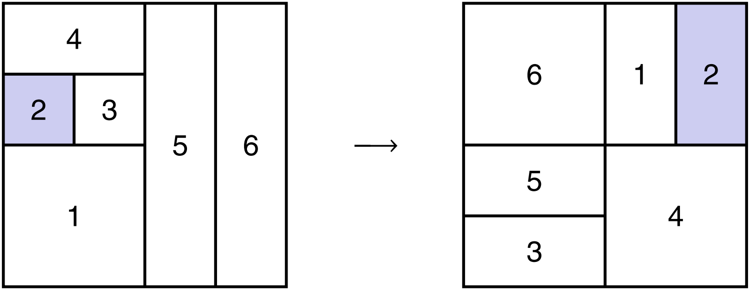
$(0\psi, 0\omega) \mapsto (10\psi, 1\omega)$ $(00\psi, 10\omega) \mapsto (11\psi, 1\omega)$

$(01\psi, 10\omega) \mapsto (0\psi, 00\omega)$ $(0\psi, 11\omega) \mapsto (1\psi, 0\omega)$

$(10\psi, \omega) \mapsto (0\psi, 01\omega)$ $(11\psi, \omega) \mapsto (0\psi, 1\omega)$

The Group $2V$

Each piece is a ***prefix pair replacement***



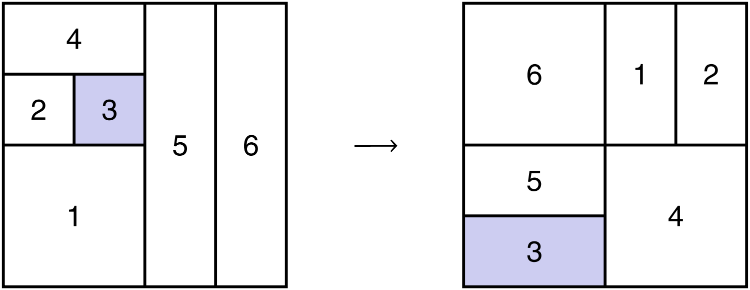
$$(0\psi, 0\omega) \mapsto (10\psi, 1\omega) \qquad (00\psi, 10\omega) \mapsto (11\psi, 1\omega)$$

$$(01\psi, 10\omega) \mapsto (0\psi, 00\omega) \qquad (0\psi, 11\omega) \mapsto (1\psi, 0\omega)$$

$$(10\psi, \omega) \mapsto (0\psi, 01\omega) \qquad (11\psi, \omega) \mapsto (0\psi, 1\omega)$$

The Group $2V$

Each piece is a ***prefix pair replacement***



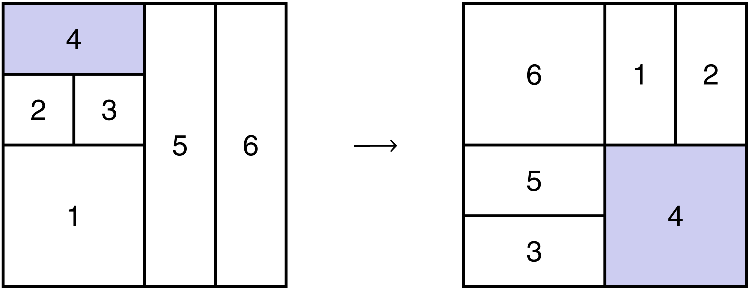
$(0\psi, 0\omega) \mapsto (10\psi, 1\omega)$ $\quad$ $(00\psi, 10\omega) \mapsto (11\psi, 1\omega)$

$(01\psi, 10\omega) \mapsto (0\psi, 00\omega)$ $\quad$ $(0\psi, 11\omega) \mapsto (1\psi, 0\omega)$

$(10\psi, \omega) \mapsto (0\psi, 01\omega)$ $\quad$ $(11\psi, \omega) \mapsto (0\psi, 1\omega)$

The Group $2V$

Each piece is a ***prefix pair replacement***



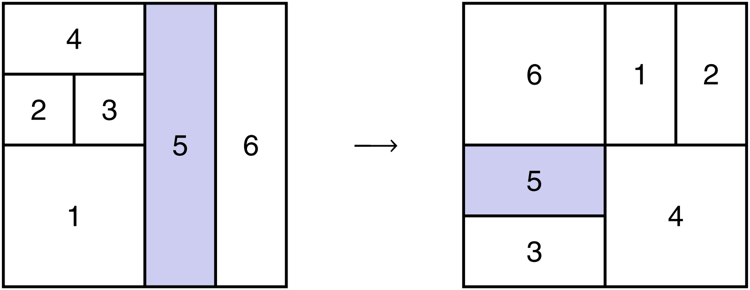
$(0\psi, 0\omega) \mapsto (10\psi, 1\omega)$ $\quad$ $(00\psi, 10\omega) \mapsto (11\psi, 1\omega)$

$(01\psi, 10\omega) \mapsto (0\psi, 00\omega)$ $\quad$ $(0\psi, 11\omega) \mapsto (1\psi, 0\omega)$

$(10\psi, \omega) \mapsto (0\psi, 01\omega)$ $\quad$ $(11\psi, \omega) \mapsto (0\psi, 1\omega)$

The Group $2V$

Each piece is a ***prefix pair replacement***



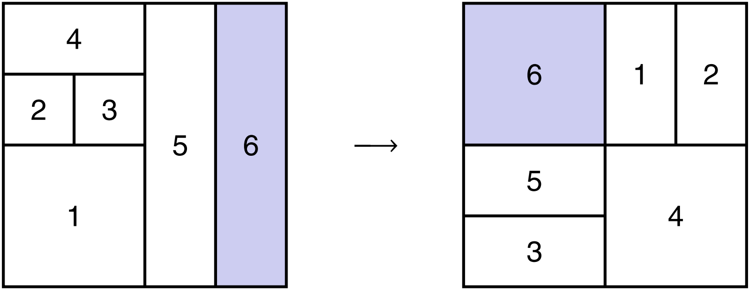
$(0\psi, 0\omega) \mapsto (10\psi, 1\omega)$ $\quad$ $(00\psi, 10\omega) \mapsto (11\psi, 1\omega)$

$(01\psi, 10\omega) \mapsto (0\psi, 00\omega)$ $\quad$ $(0\psi, 11\omega) \mapsto (1\psi, 0\omega)$

$(10\psi, \omega) \mapsto (0\psi, 01\omega)$ $\quad$ $(11\psi, \omega) \mapsto (0\psi, 1\omega)$

The Group $2V$

Each piece is a ***prefix pair replacement***



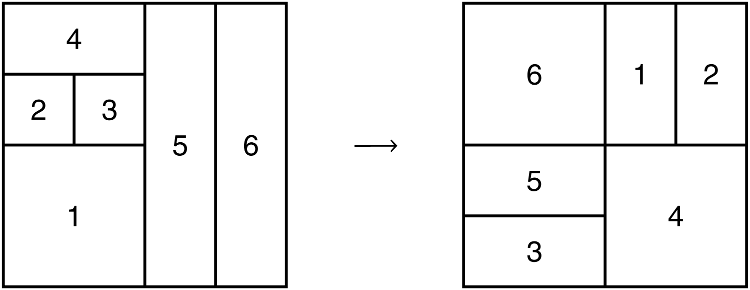
$$(0\psi, 0\omega) \mapsto (10\psi, 1\omega) \qquad (00\psi, 10\omega) \mapsto (11\psi, 1\omega)$$

$$(01\psi, 10\omega) \mapsto (0\psi, 00\omega) \qquad (0\psi, 11\omega) \mapsto (1\psi, 0\omega)$$

$$(10\psi, \omega) \mapsto (0\psi, 01\omega) \qquad (11\psi, \omega) \mapsto (0\psi, 1\omega)$$

The Group $2V$

Each piece is a ***prefix pair replacement***



$$(0\psi, 0\omega) \mapsto (10\psi, 1\omega) \qquad (00\psi, 10\omega) \mapsto (11\psi, 1\omega)$$

$$(01\psi, 10\omega) \mapsto (0\psi, 00\omega) \qquad (0\psi, 11\omega) \mapsto (1\psi, 0\omega)$$

$$(10\psi, \omega) \mapsto (0\psi, 01\omega) \qquad (11\psi, \omega) \mapsto (0\psi, 1\omega)$$

Properties of $2V$

Theorem (Brin 2004)

$2V$ is finitely presented and simple.

Theorem (Brin 2004)

Given pattern pairs for two elements $f, g \in 2V$,

1. *There is an algorithm to determine whether $f = g$.*
2. *There is an algorithm to compute the pattern pair for $f \circ g$.*

Corollary

$2V$ has solvable word problem.

Main Results

Theorem (B–Bleak 2017)

There is no algorithm to decide whether a given element of $2V$ has finite order.

Strategy: Use elements of $2V$ to simulate Turing machines.

Theorem (Kari–Ollinger 2008)

There is no algorithm to determine whether a given complete, reversible Turing machine is uniformly periodic.

Main Results

Theorem (B–Bleak 2017)

There is no algorithm to decide whether a given element of $2V$ has finite order.

Strategy: Use elements of $2V$ to simulate Turing machines.

Theorem (Kari–Ollinger 2008)

There is no algorithm to determine whether a given complete, reversible Turing machine is uniformly periodic.

Theorem (B–Bleak 2017)

There is no algorithm to determine whether a given finite-state transducer defines a mapping of finite order.

The Torsion Problem

Torsion problem for a group G:
Given a word w, does w represent an element of finite order in G?

Theorem (Baumslag–Boone–Neumann 1959)

There exists a finitely presented group with unsolvable torsion problem.

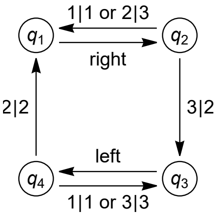
Theorem (Arzhantseva–Lafont–Minasyan 2012)

There exists a finitely presented group with solvable word problem and unsolvable torsion problem.

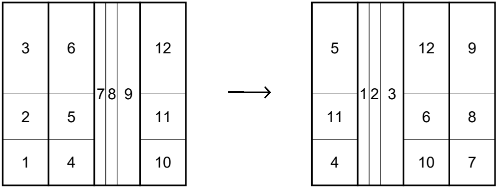
$2V$ was the first concrete example of such a group.

The Plan

Given: A complete, reversible Turing machine.



Construct: An element of $2V$ with the same dynamics.



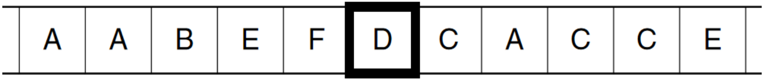
Turing Machines

Turing Machines

A ***Turing machine*** consists of:

1. A finite set Q of states (move **or** read/write),
2. A finite ***tape alphabet*** A,
3. A ***transition*** for each state.

A ***tape*** is any function $\mathbb{Z} \to A$.



Transitions will affect the tape by either moving (i.e. sliding horizontally) or reading and writing at location 0.

Transitions

Each state has a ***transition***

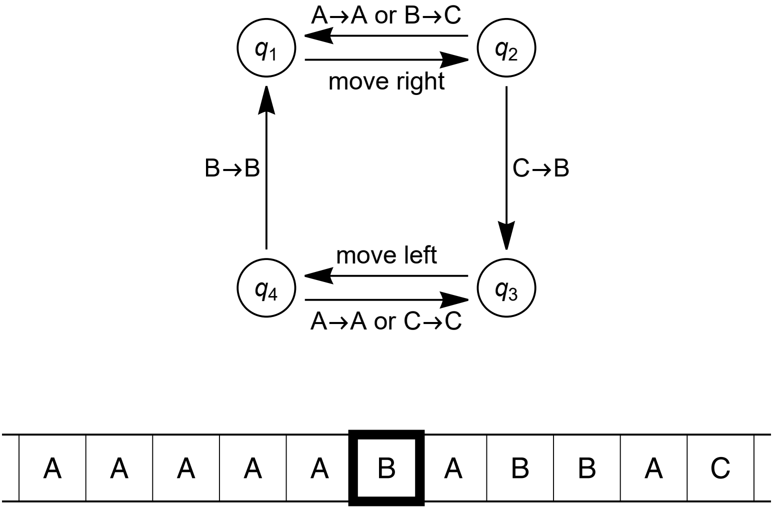
Transition for a Move State

1. Move one step in a certain direction (left or right).
2. Go to a certain state.

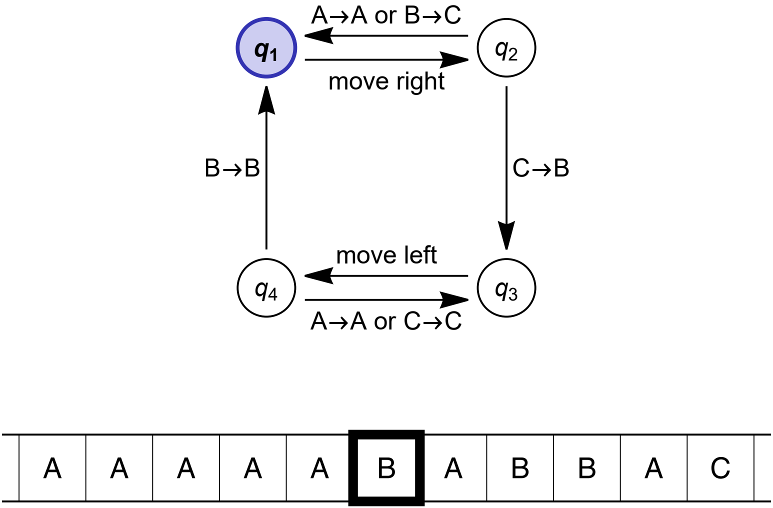
Transition for a Read/Write State

1. Read the current letter.
2. Write a letter, depending on what was read.
3. Go to a certain state, depending on what was read.

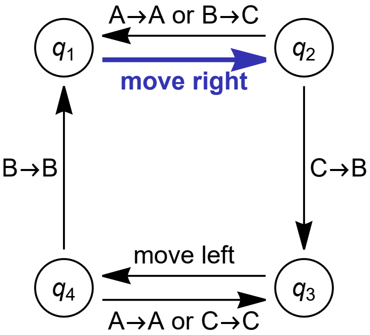
Example Turing Machine

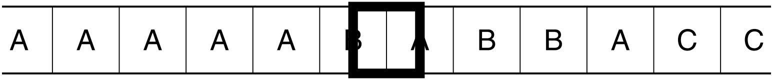


Example Turing Machine

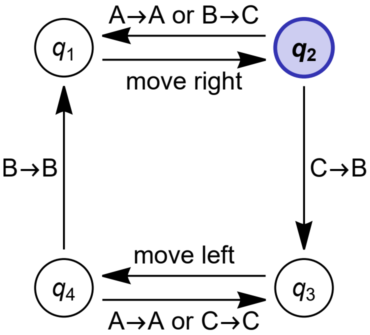


Example Turing Machine



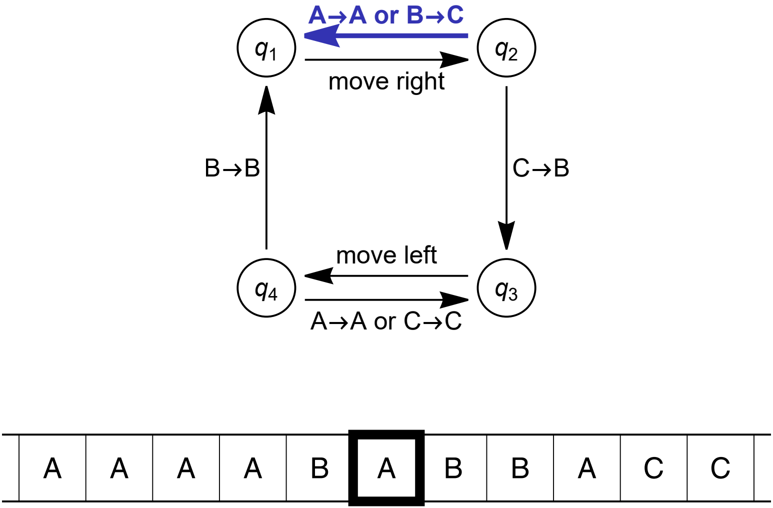


Example Turing Machine

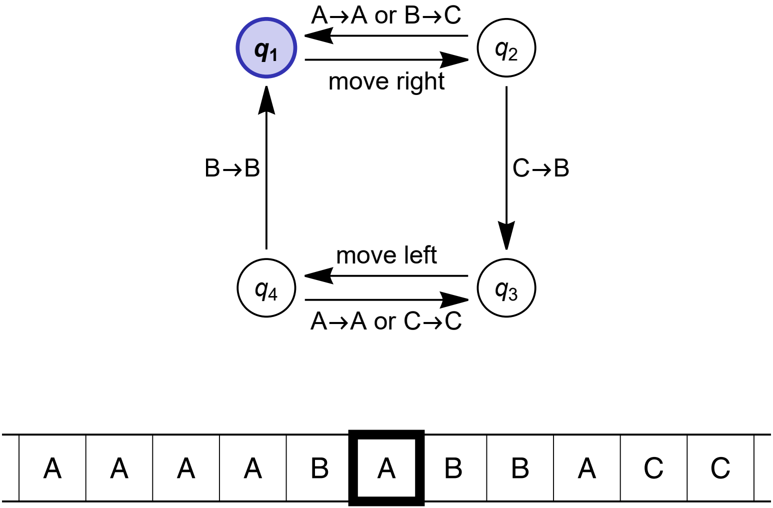


A	A	A	A	B	A	B	B	A	C	C

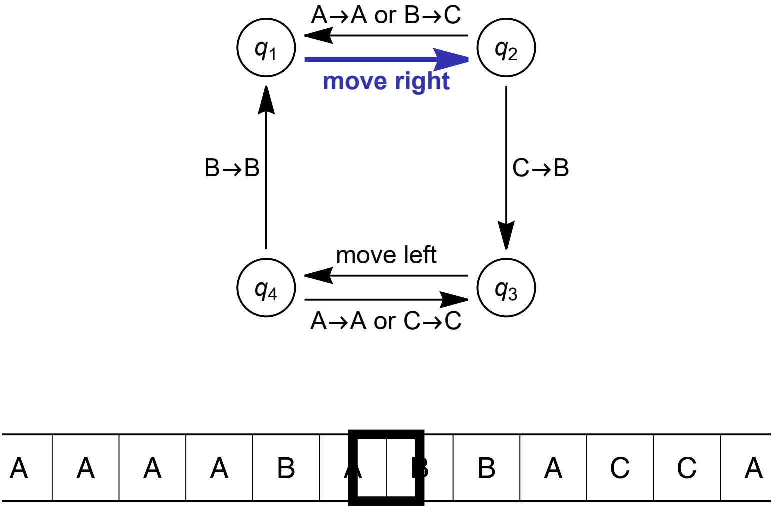
Example Turing Machine



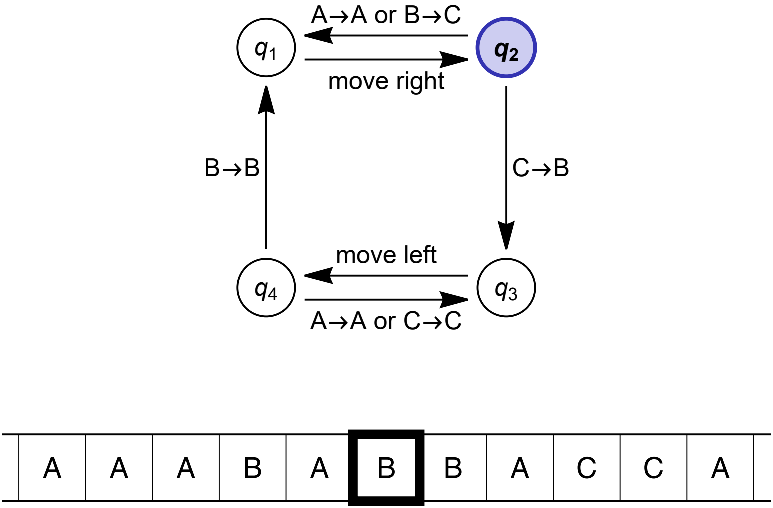
Example Turing Machine



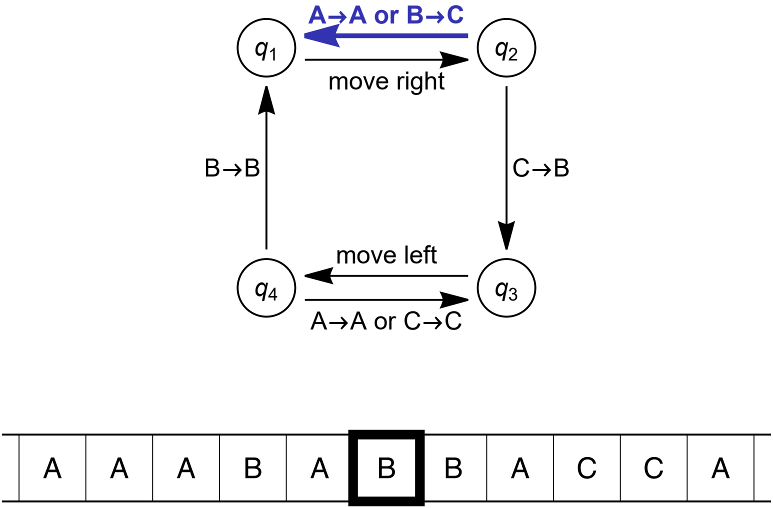
Example Turing Machine



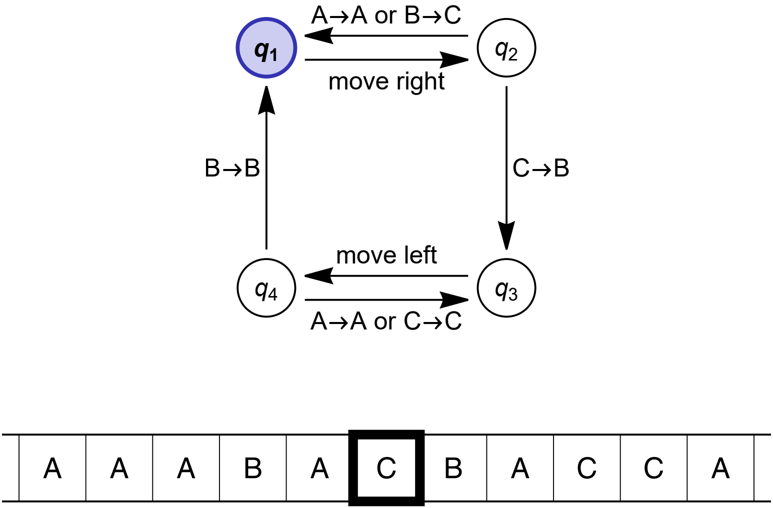
Example Turing Machine



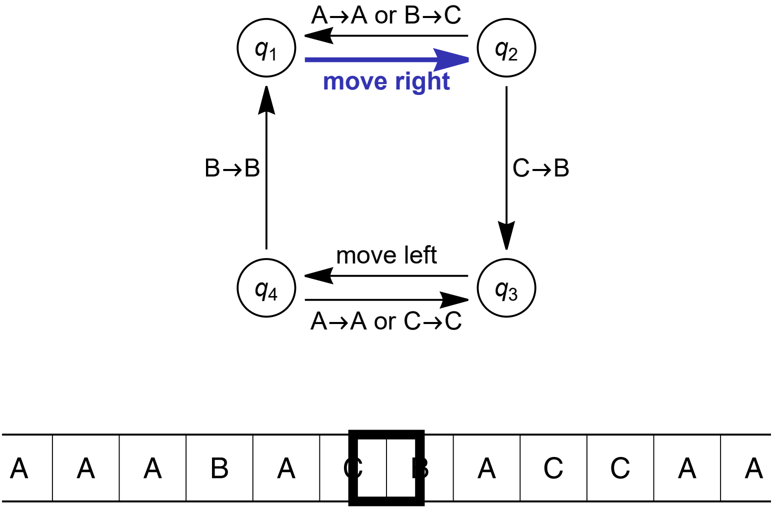
Example Turing Machine



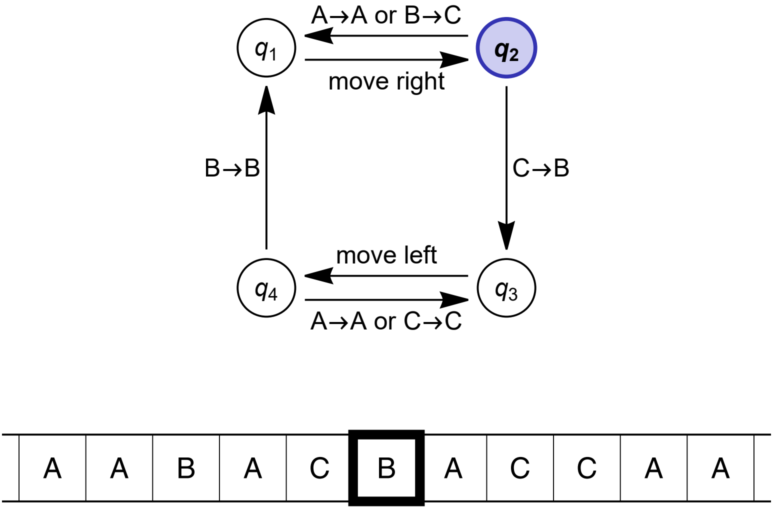
Example Turing Machine



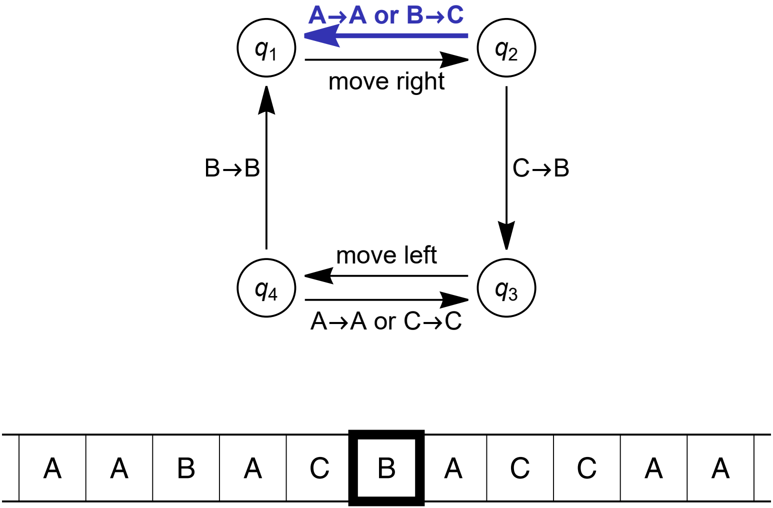
Example Turing Machine



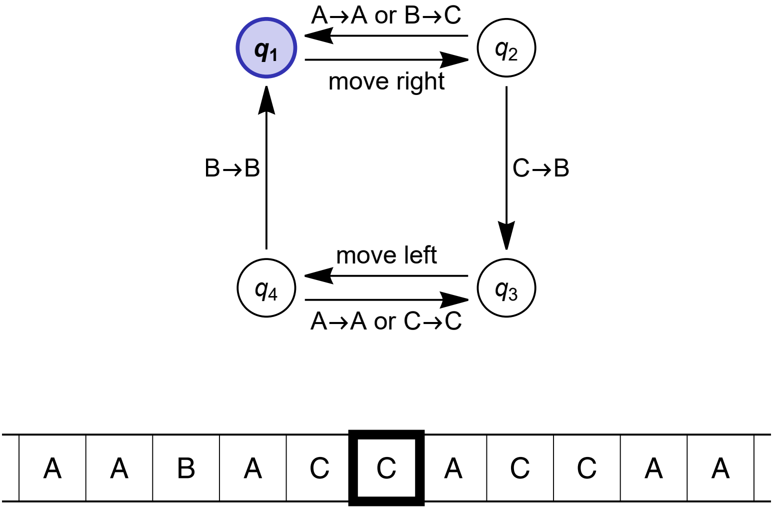
Example Turing Machine



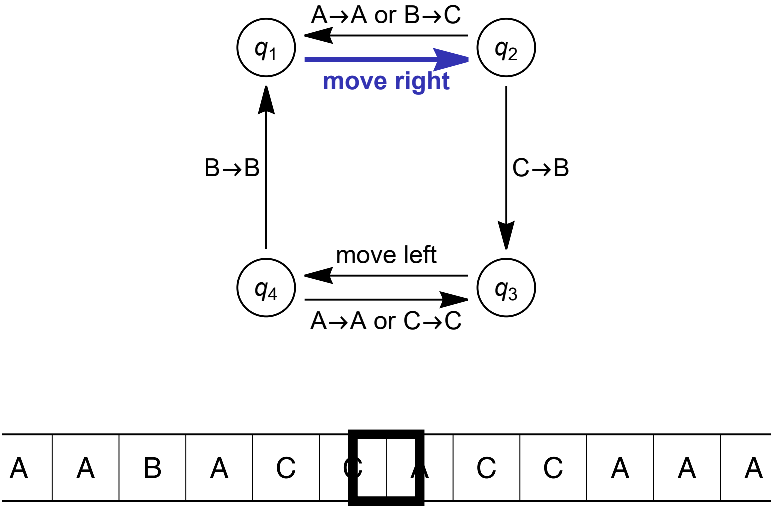
Example Turing Machine



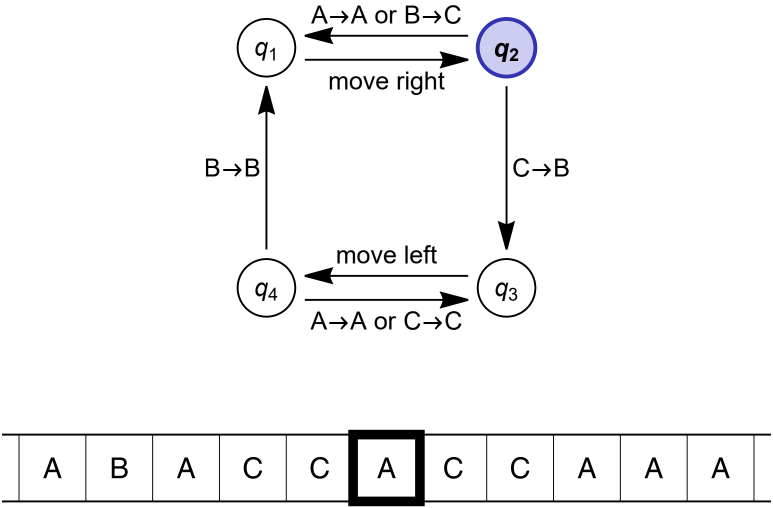
Example Turing Machine



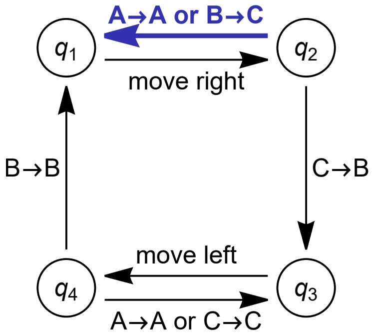
Example Turing Machine

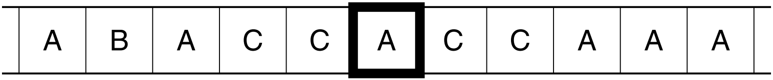


Example Turing Machine

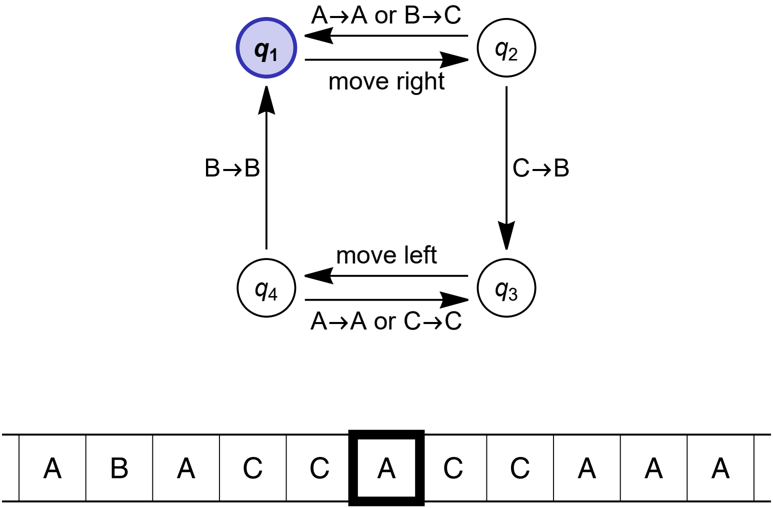


Example Turing Machine

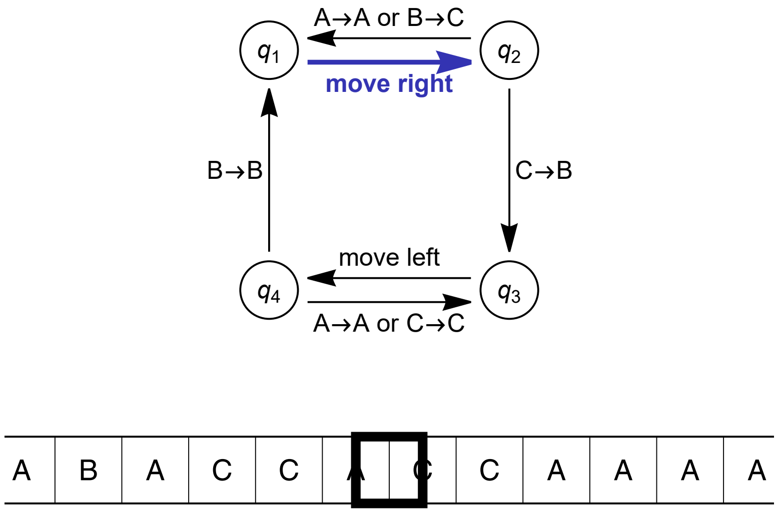




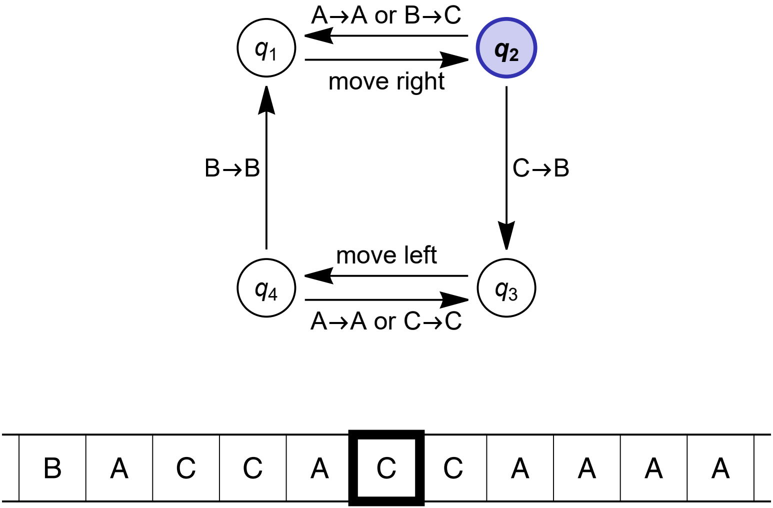
Example Turing Machine



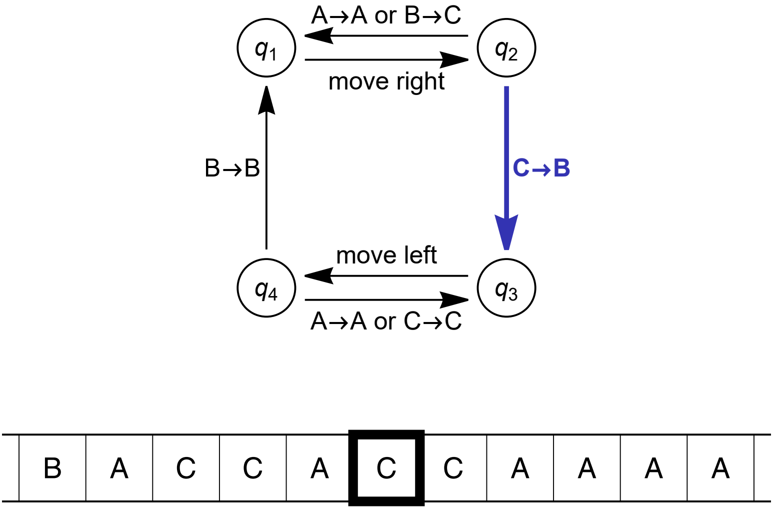
Example Turing Machine



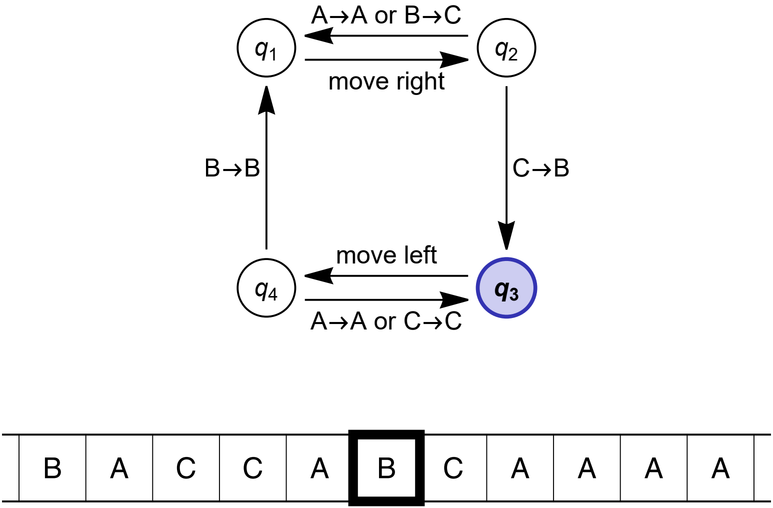
Example Turing Machine



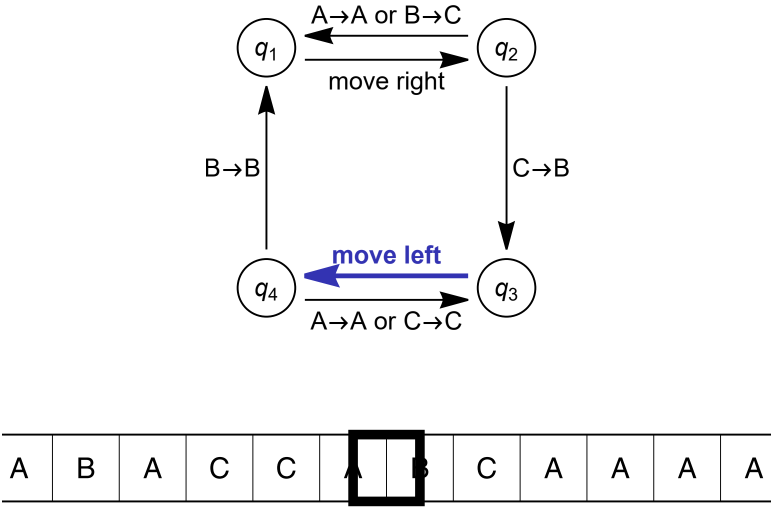
Example Turing Machine



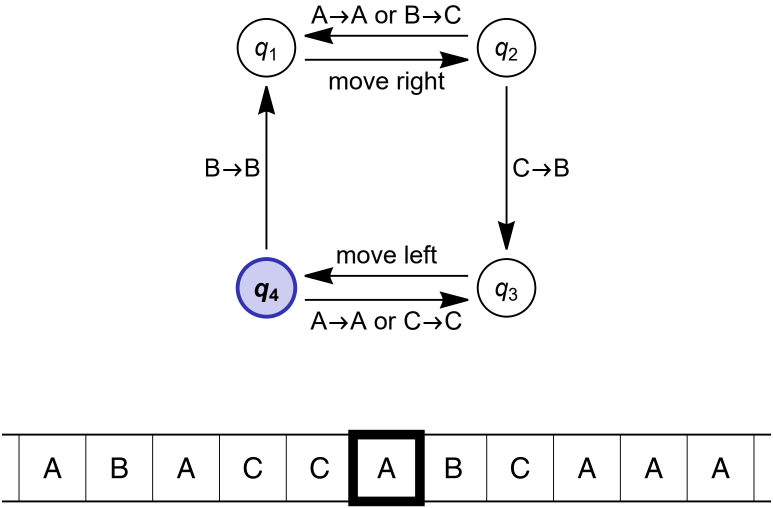
Example Turing Machine



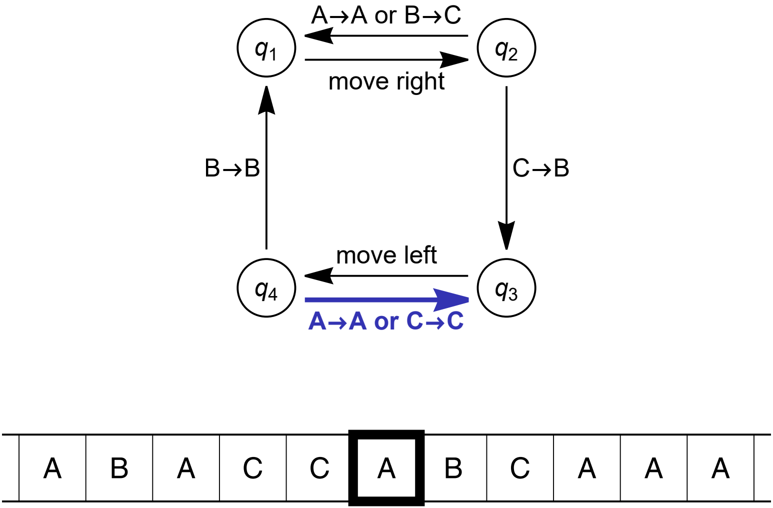
Example Turing Machine



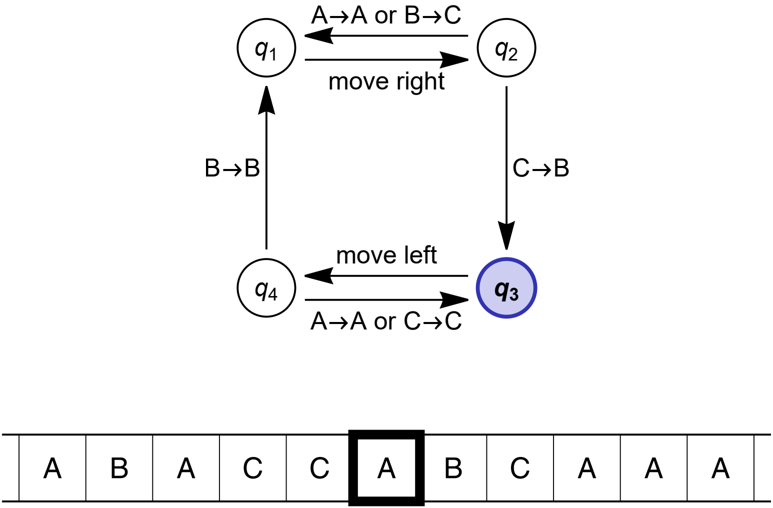
Example Turing Machine



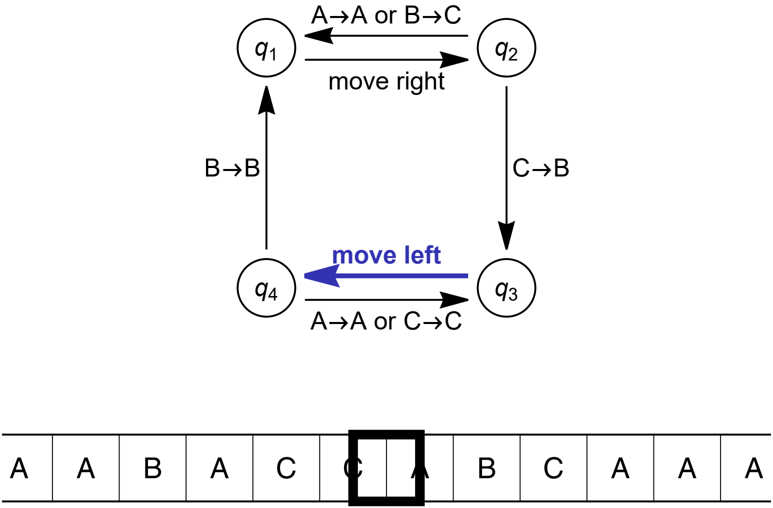
Example Turing Machine



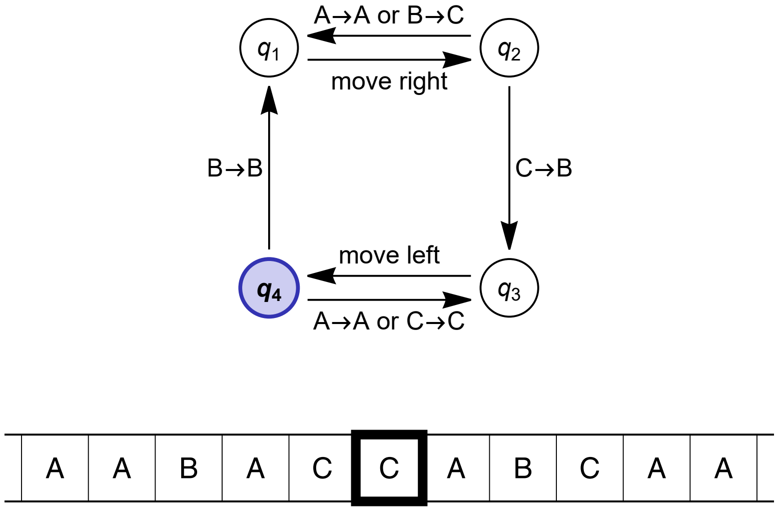
Example Turing Machine



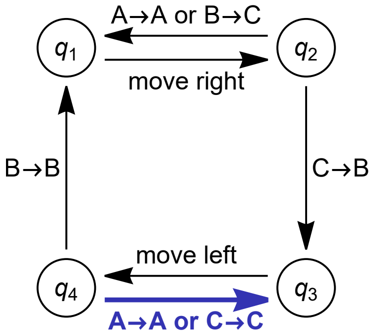
Example Turing Machine

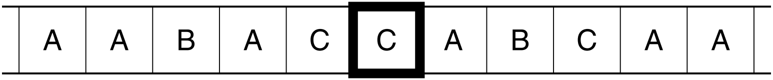


Example Turing Machine

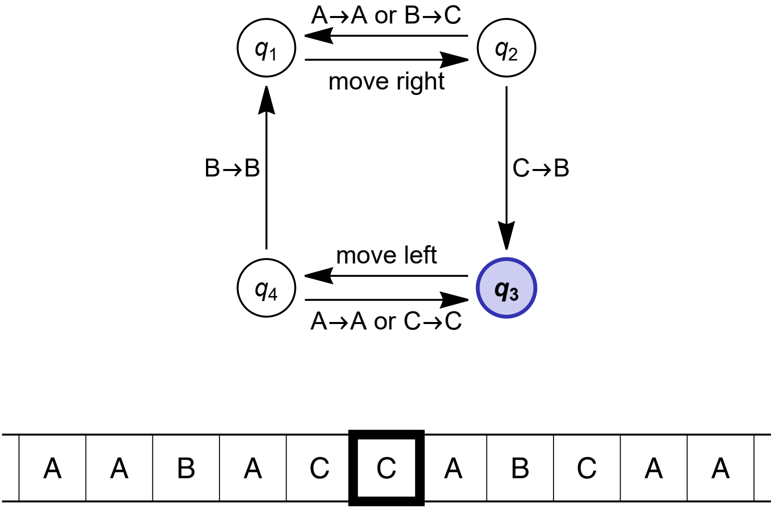


Example Turing Machine

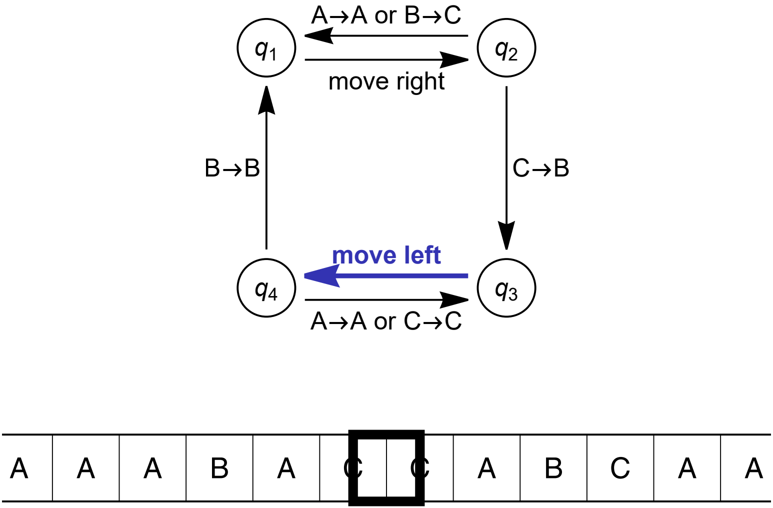




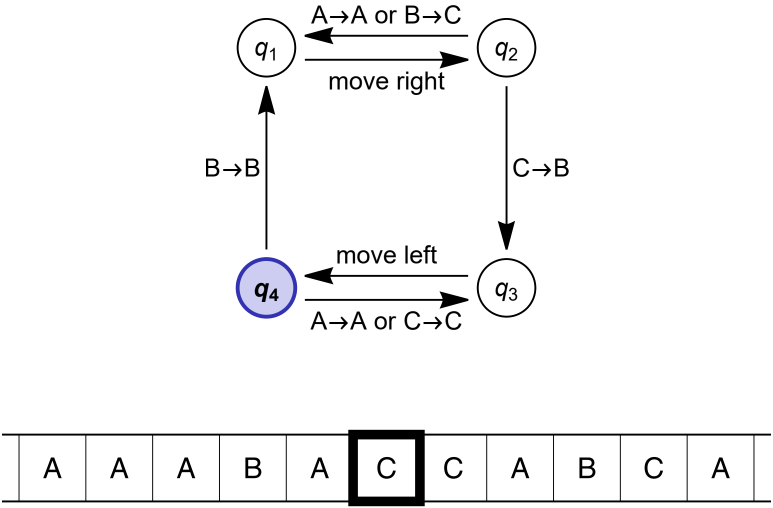
Example Turing Machine



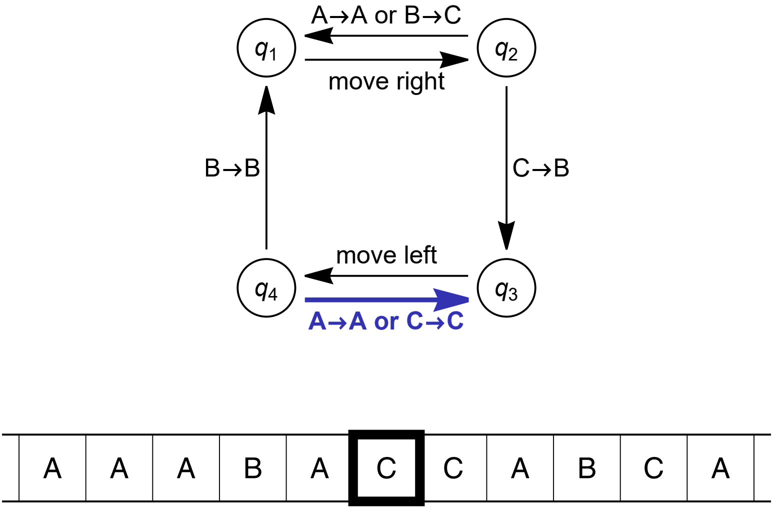
Example Turing Machine



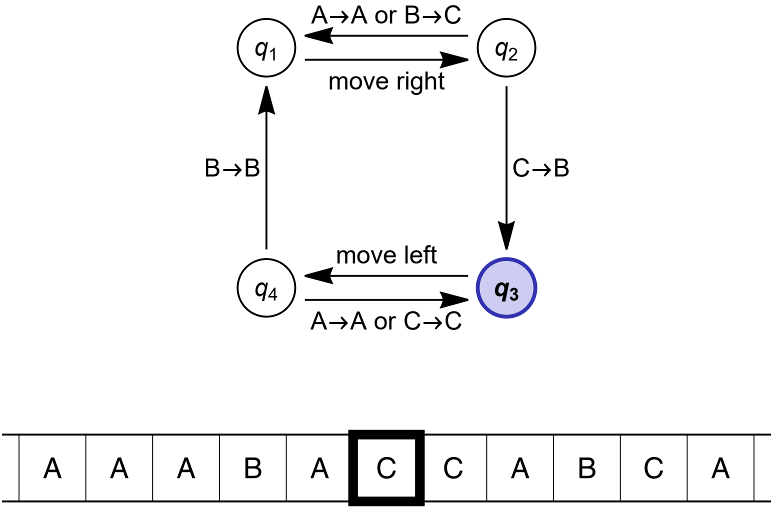
Example Turing Machine



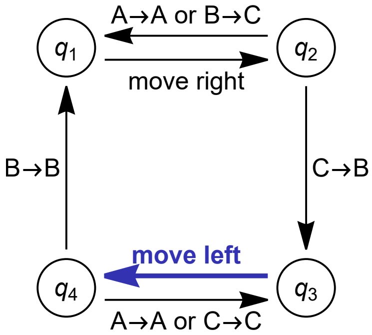
Example Turing Machine

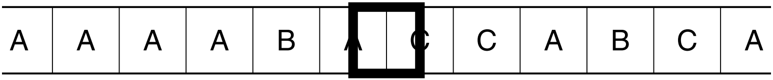


Example Turing Machine

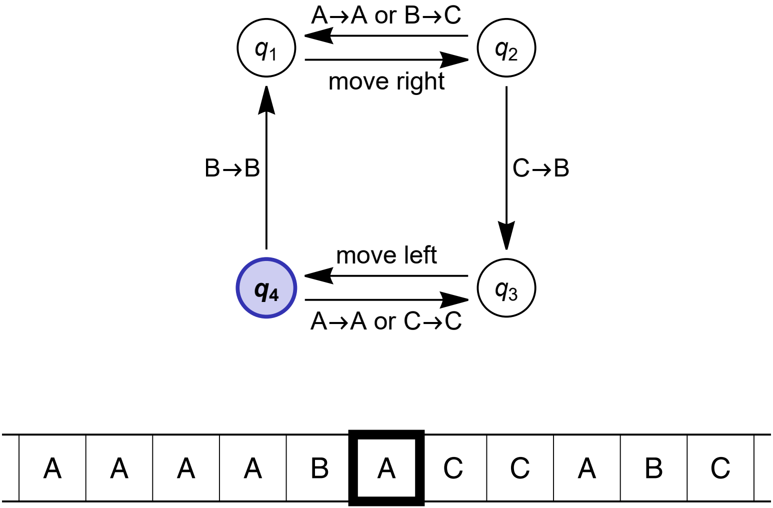


Example Turing Machine

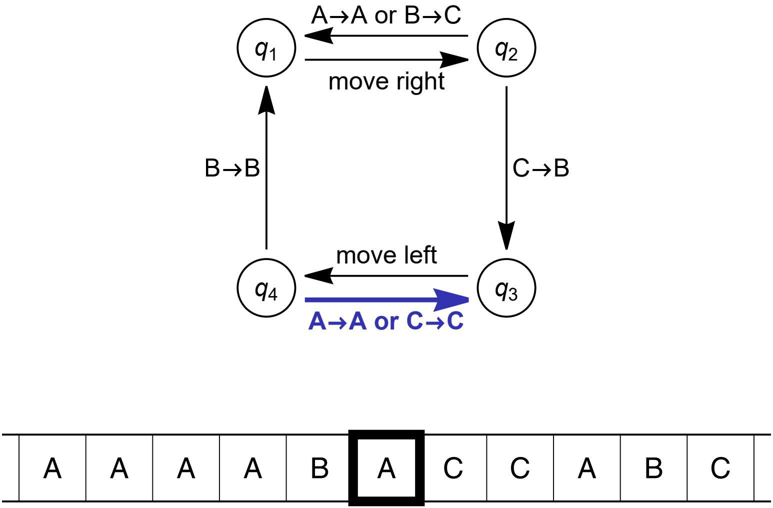




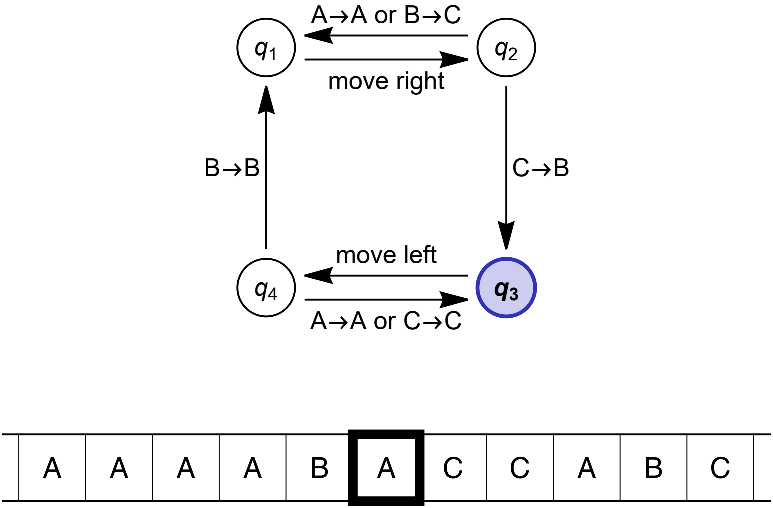
Example Turing Machine



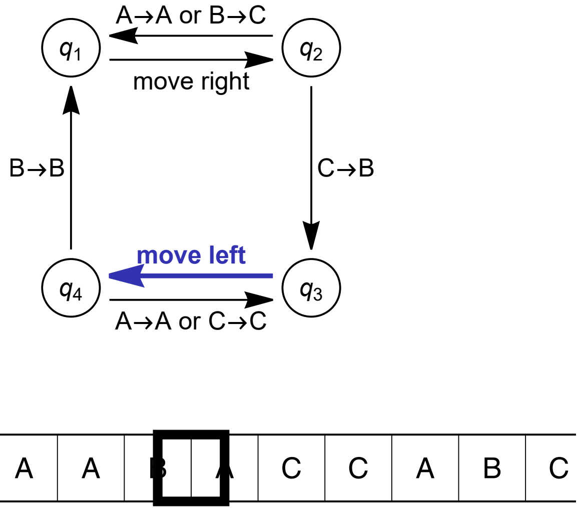
Example Turing Machine



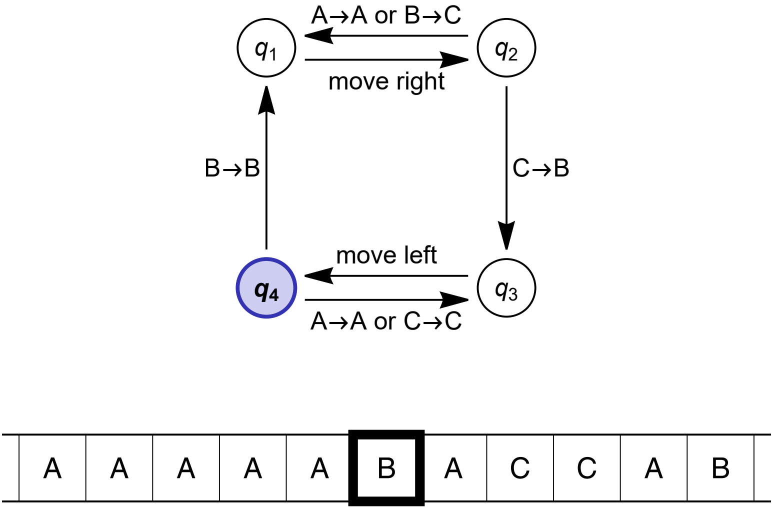
Example Turing Machine



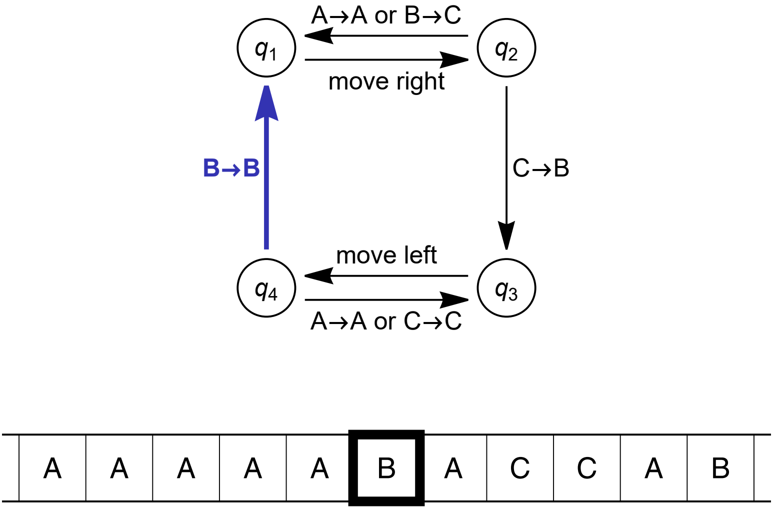
Example Turing Machine



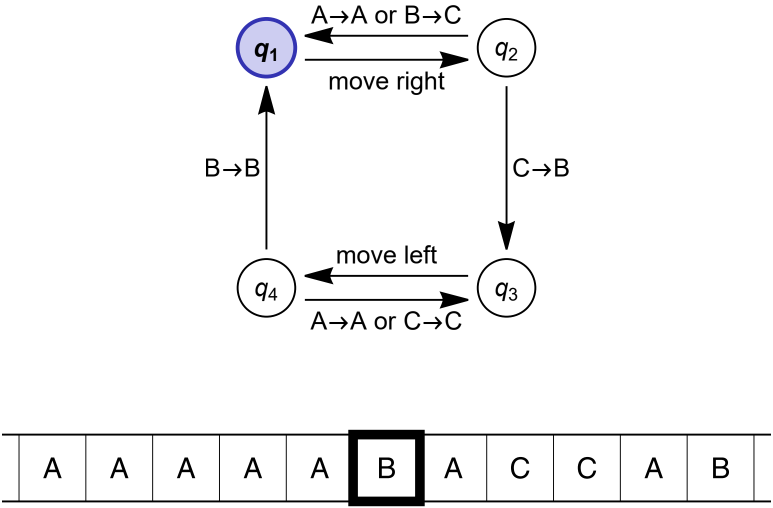
Example Turing Machine



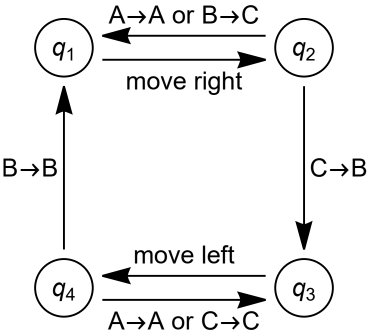
Example Turing Machine



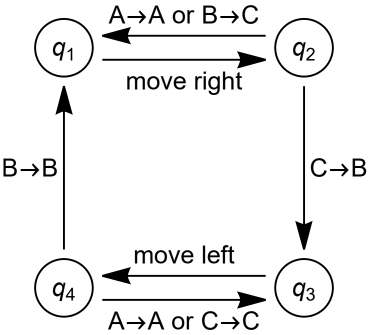
Example Turing Machine



Example Turing Machine



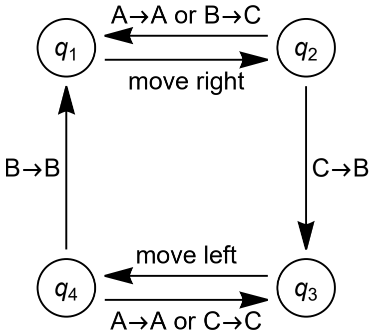
Example Turing Machine



Notes

1. This Turing machine is ***complete*** (it never halts).

Example Turing Machine



Notes

1. This Turing machine is ***complete*** (it never halts).
2. This Turing machine is ***reversible*** (it can be run backwards).

Dynamics of Turing Machines

A ***configuration*** of a Turing machine is a $(\text{state}, \text{tape})$ pair.

The ***configuration space*** is the space $Q \times A^{\mathbb{Z}}$ of all configurations.

Note: This is homeomorphic to the Cantor set.

Fact

A complete, reversible Turing machine acts as a homeomorphism of its configuration space.

Theorem (Kari–Olinger 2008)

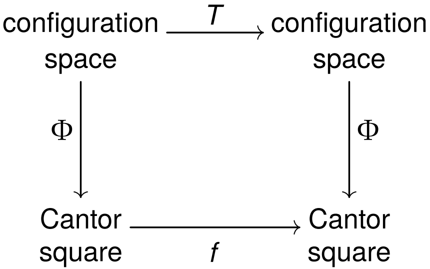
There is no algorithm to decide whether the homeomorphism defined by a given complete, reversible Turing machine has finite order.

Turing Machines in $2V$

The Plan

Given: A complete, reversible Turing machine T.

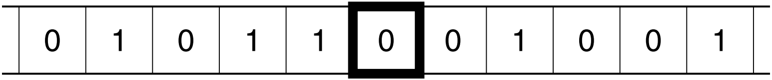
Construct: An ***encoding homeomorphism*** Φ and an element $f \in 2V$ making the following diagram commute:



Then f has the same order as T, so we can't tell whether T has finite order.

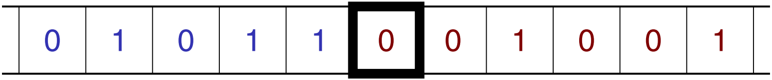
Basic Encoding Idea

A tape can be split into two infinite sequences.



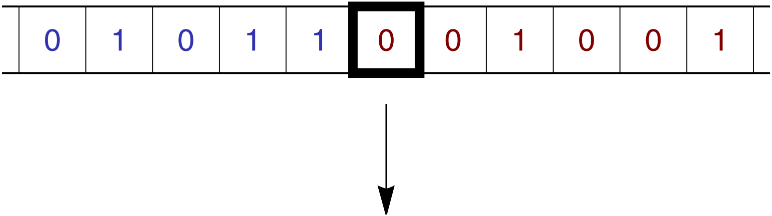
Basic Encoding Idea

A tape can be split into two infinite sequences.



Basic Encoding Idea

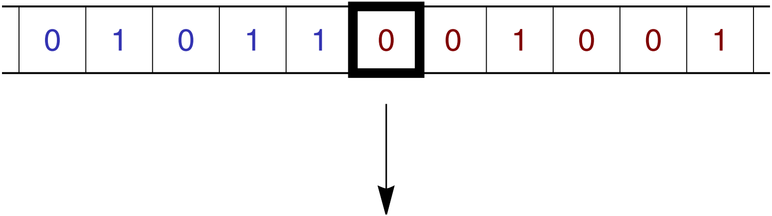
A tape can be split into two infinite sequences.



$$(1\ 1\ 0\ 1\ 0\ \cdots,\ 0\ 0\ 1\ 0\ 0\ 1\ \cdots)$$

Basic Encoding Idea

A tape can be split into two infinite sequences.



$$(1\ 1\ 0\ 1\ 0\ \cdots,\ 0\ 0\ 1\ 0\ 0\ 1\ \cdots)$$

Problems:

1. What if the alphabet isn't binary?
2. What about the state?

Encoding the Tape Alphabet

It is possible to encode any finite alphabet into binary using a ***complete binary prefix code***.

Example. The alphabet $\{A, B, C\}$.

Use the following encoding:

$$A \mapsto 00, \qquad B \mapsto 01, \qquad C \mapsto 1.$$

So

$$A \;\; B \;\; A \;\; C \;\; B \;\; C \;\; \cdots$$

becomes

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

Encoding the Tape Alphabet

It is possible to encode any finite alphabet into binary using a ***complete binary prefix code***.

Example. The alphabet $\{A, B, C\}$.

Use the following encoding:

$$A \mapsto 00, \qquad B \mapsto 01, \qquad C \mapsto 1.$$

So

$$\underline{A} \;\; B \;\; A \;\; C \;\; B \;\; C \;\; \cdots$$

becomes

$$\underline{0 \;\; 0} \;\; 0 \;\; 1 \;\; 0 \;\; 0 \;\; 1 \;\; 0 \;\; 1 \;\; 1 \;\; \cdots$$

Encoding the Tape Alphabet

It is possible to encode any finite alphabet into binary using a ***complete binary prefix code***.

Example. The alphabet $\{A, B, C\}$.

Use the following encoding:

$$A \mapsto 00, \qquad B \mapsto 01, \qquad C \mapsto 1.$$

So

$$A \;\; \underline{B} \;\; A \;\; C \;\; B \;\; C \;\; \cdots$$

becomes

$$0 \;\; 0 \;\; \underline{0 \;\; 1} \;\; 0 \;\; 0 \;\; 1 \;\; 0 \;\; 1 \;\; 1 \;\; \cdots$$

Encoding the Tape Alphabet

It is possible to encode any finite alphabet into binary using a ***complete binary prefix code***.

Example. The alphabet $\{A, B, C\}$.

Use the following encoding:

$$A \mapsto 00, \qquad B \mapsto 01, \qquad C \mapsto 1.$$

So

$$A \;\; B \;\; \underline{A} \;\; C \;\; B \;\; C \;\; \cdots$$

becomes

$$0 \;\; 0 \;\; 0 \;\; 1 \;\; \underline{0 \;\; 0} \;\; 1 \;\; 0 \;\; 1 \;\; 1 \;\; \cdots$$

Encoding the Tape Alphabet

It is possible to encode any finite alphabet into binary using a ***complete binary prefix code***.

Example. The alphabet $\{A, B, C\}$.

Use the following encoding:

$$A \mapsto 00, \qquad B \mapsto 01, \qquad C \mapsto 1.$$

So

$$A \;\; B \;\; A \;\; \underline{C} \;\; B \;\; C \;\; \cdots$$

becomes

$$0 \;\; 0 \;\; 0 \;\; 1 \;\; 0 \;\; 0 \;\; \underline{1} \;\; 0 \;\; 1 \;\; 1 \;\; \cdots$$

Encoding the Tape Alphabet

It is possible to encode any finite alphabet into binary using a ***complete binary prefix code***.

Example. The alphabet $\{A, B, C\}$.

Use the following encoding:

$$A \mapsto 00, \qquad B \mapsto 01, \qquad C \mapsto 1.$$

So

$$A \ \ B \ \ A \ \ C \ \ \underline{B} \ \ C \ \ \cdots$$

becomes

$$0 \ \ 0 \ \ 0 \ \ 1 \ \ 0 \ \ 0 \ \ 1 \ \ \underline{0 \ \ 1} \ \ 1 \ \ \cdots$$

Encoding the Tape Alphabet

It is possible to encode any finite alphabet into binary using a ***complete binary prefix code***.

Example. The alphabet $\{A, B, C\}$.

Use the following encoding:

$$A \mapsto 00, \qquad B \mapsto 01, \qquad C \mapsto 1.$$

So

$$A \;\; B \;\; A \;\; C \;\; B \;\; \underline{C} \;\; \cdots$$

becomes

$$0 \;\; 0 \;\; 0 \;\; 1 \;\; 0 \;\; 0 \;\; 1 \;\; 0 \;\; 1 \;\; \underline{1} \;\; \cdots$$

Encoding the Tape Alphabet

It is possible to encode any finite alphabet into binary using a ***complete binary prefix code***.

Example. The alphabet $\{A, B, C\}$.

Use the following encoding:

$$A \mapsto 00, \qquad B \mapsto 01, \qquad C \mapsto 1.$$

So

$$A \;\; B \;\; A \;\; C \;\; B \;\; C \;\; \cdots$$

becomes

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

This defines a homeomorphism $\{A, B, C\}^\omega \to \{0, 1\}^\omega$.

Encoding the Tape Alphabet

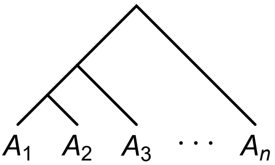
It is possible to encode any finite alphabet into binary using a ***complete binary prefix code***.

Encoding the Tape Alphabet

It is possible to encode any finite alphabet into binary using a ***complete binary prefix code***.

General Procedure. For the alphabet $\{A_1, A_2, \ldots, A_n\}$.

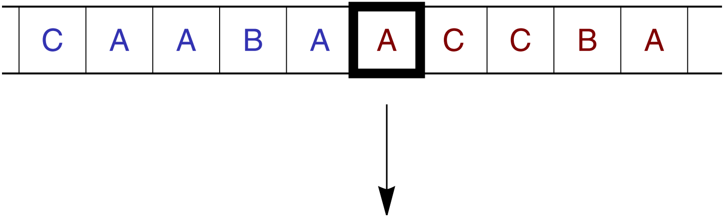
Choose a binary tree with one leaf for each letter:



The binary code for each letter is given by its position in the tree.

Progress So Far

We can now encode an arbitrary tape as a point in the Cantor square.

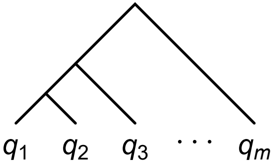


$$(0\;0\;0\;1\;0\;0\;0\;0\;1\;\cdots,\;0\;0\;1\;1\;0\;1\;0\;0\;\cdots)$$

But what about the states?

Encoding The States

Choose a complete binary prefix code for the states:



We will store the state at the beginning of the x-coordinate:

$$(\mathbf{0\ 1\ 1}\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ \cdots,\ 0\ 0\ 1\ 1\ 0\ 1\ 0\ 0\ \cdots)$$

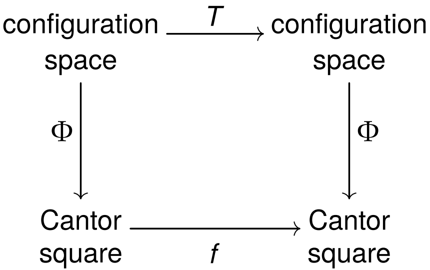
state left half of tape right half of tape

This defines the encoding homeomorphism

$$\Phi\colon \text{(configuration space)} \longrightarrow \text{(Cantor square)}$$

Progress So Far

We have now defined the encoding homeomorphism Φ.



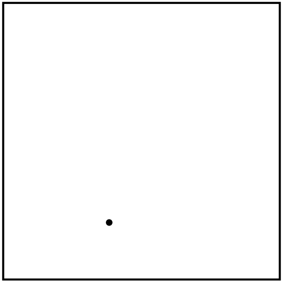
Let $f = \Phi^{-1} \circ T \circ \Phi$. We must show that $f \in 2V$.

Geometry of the Encoding

Encoding: (**0 1 1** 0 0 0 1 0 0 0 0 1 $\cdots$, 0 0 1 1 0 1 0 0 $\cdots$)

state — left half of tape — right half of tape

Each configuration corresponds to a point in the Cantor square.

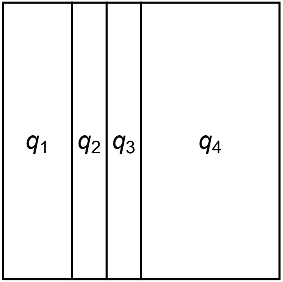


Geometry of the Encoding

Encoding: (**0 1 1** 0 0 0 1 0 0 0 0 1 $\cdots$, 0 0 1 1 0 1 0 0 $\cdots$)

state left half of tape right half of tape

States correspond to vertical rectangles.



$q_1 \mapsto 00$

$q_2 \mapsto 010$

$q_3 \mapsto 011$

$q_4 \mapsto 1$

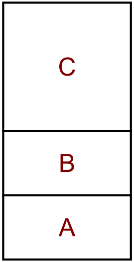
Transitions map between these rectangles.

Geometry of the Encoding

Encoding: (**0 1 1** 0 0 0 1 0 0 0 0 1 $\cdots$, 0 0 1 1 0 1 0 0 $\cdots$)

state left half of tape right half of tape

Within each state, the current letter corresponds to a horizontal subrectangle.



rectangle for q_3

A $\mapsto$ 00

B $\mapsto$ 01

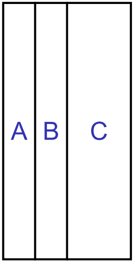
C $\mapsto$ 1

Geometry of the Encoding

Encoding: (**0 1 1** 0 0 0 1 0 0 0 0 1 $\cdots$, 0 0 1 1 0 1 0 0 $\cdots$)

state — left half of tape — right half of tape

The first letter on the left corresponds to a vertical subrectangle.



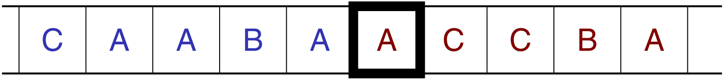
rectangle for q_3

A $\mapsto$ 00

B $\mapsto$ 01

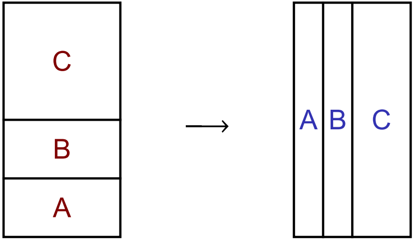
C $\mapsto$ 1

A Right Move



(**0 1 1** 0 0 0 1 0 0 0 0 1 ··· , 0 0 1 1 0 1 0 0 ···)

state left half of tape right half of tape

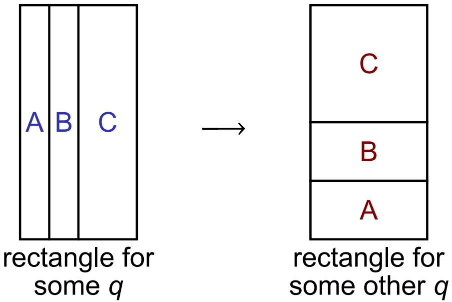


rectangle for some q → rectangle for some other q

A Left Move

Encoding: (**0 1 1** 0 0 0 1 0 0 0 0 1 ⋯ , 0 0 1 1 0 1 0 0 ⋯)

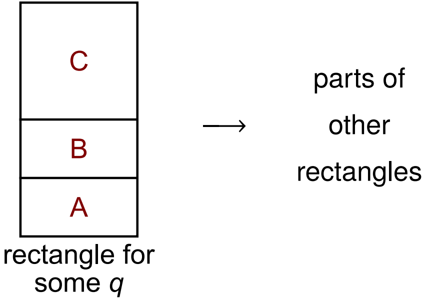
state left half of tape right half of tape



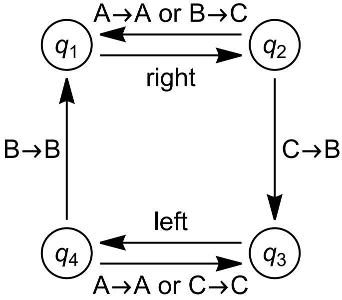
A Read/Write

Encoding: (**0 1 1** 0 0 0 1 0 0 0 0 1 $\cdots$, 0 0 1 1 0 1 0 0 $\cdots$)

state — left half of tape — right half of tape



An Example



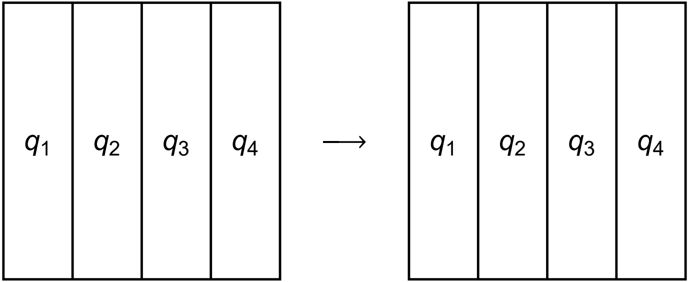
Alphabet Encoding:

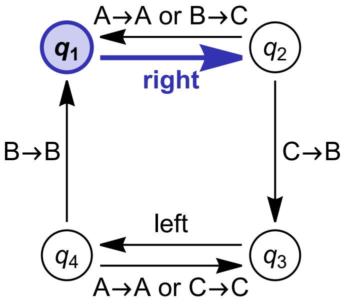
$A \mapsto 00, \quad B \mapsto 01, \quad C \mapsto 1$

State Encoding:

$q_1 \mapsto 00, \quad q_2 \mapsto 01,$

$q_3 \mapsto 10, \quad q_4 \mapsto 11$





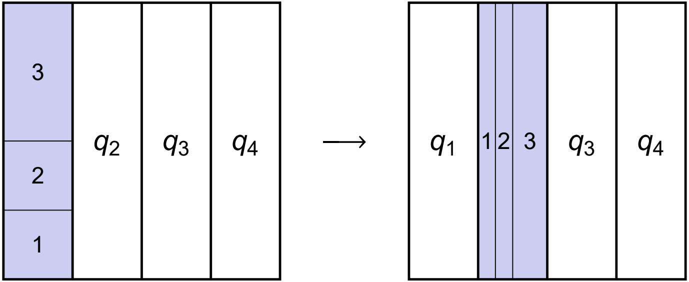
Alphabet Encoding:

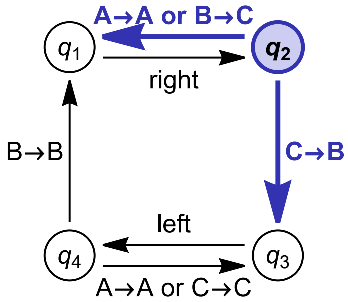
$A \mapsto 00, \quad B \mapsto 01, \quad C \mapsto 1$

State Encoding:

$q_1 \mapsto 00, \quad q_2 \mapsto 01,$

$q_3 \mapsto 10, \quad q_4 \mapsto 11$





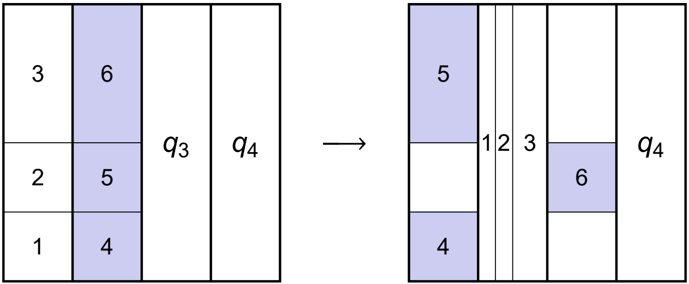
Alphabet Encoding:

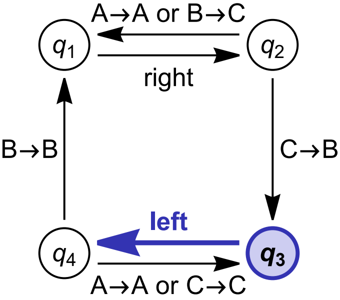
$A \mapsto 00, \quad B \mapsto 01, \quad C \mapsto 1$

State Encoding:

$q_1 \mapsto 00, \quad q_2 \mapsto 01,$

$q_3 \mapsto 10, \quad q_4 \mapsto 11$





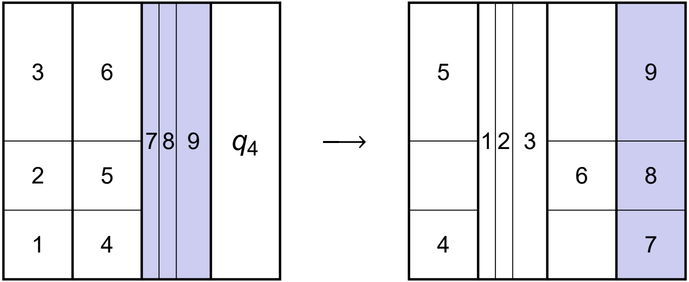
Alphabet Encoding:

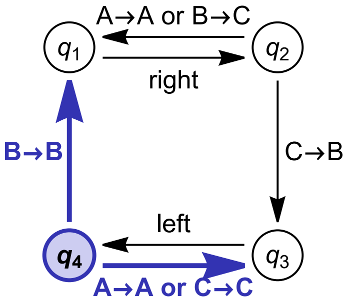
$A \mapsto 00, \quad B \mapsto 01, \quad C \mapsto 1$

State Encoding:

$q_1 \mapsto 00, \quad q_2 \mapsto 01,$

$q_3 \mapsto 10, \quad q_4 \mapsto 11$





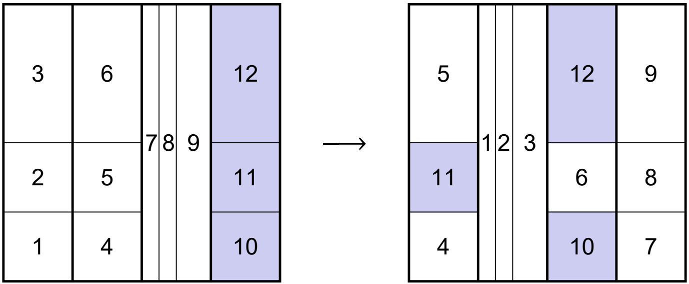
Alphabet Encoding:

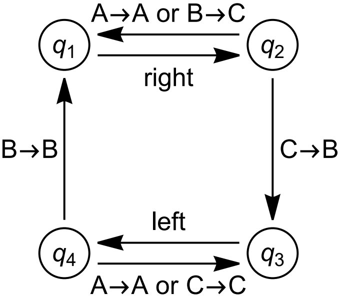
$A \mapsto 00, \quad B \mapsto 01, \quad C \mapsto 1$

State Encoding:

$q_1 \mapsto 00, \quad q_2 \mapsto 01,$

$q_3 \mapsto 10, \quad q_4 \mapsto 11$





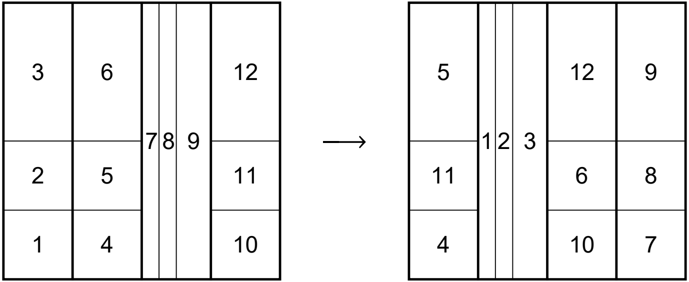
Alphabet Encoding:

A $\mapsto$ 00, B $\mapsto$ 01, C $\mapsto$ 1

State Encoding:

$q_1 \mapsto 00$, $q_2 \mapsto 01$,

$q_3 \mapsto 10$, $q_4 \mapsto 11$



Consequences

Consequences

Theorem

The group $2V$ has unsolvable torsion problem.

Define $\Omega\colon \mathbb{N} \longrightarrow \mathbb{N}$ by

$$\Omega(n) = \max\{\ |f|\ :\ f \in 2V \text{ has finite order and length} \leq n\}$$

Corollary

The function $\Omega(n)$ grows more quickly than any computable function.

Corollary

There exists an element $f \in 2V$ of infinite order such that the statement "f has infinite order" is not provable in ZFC.

Conjugacy

Question

Does $2V$ have solvable conjugacy problem?

The torsion problem seems much “easier” than the conjugacy problem, but there is no direct relationship.

Conjugacy

Question

Does $2V$ have solvable conjugacy problem?

The torsion problem seems much "easier" than the conjugacy problem, but there is no direct relationship.

Theorem (Salo 2021)

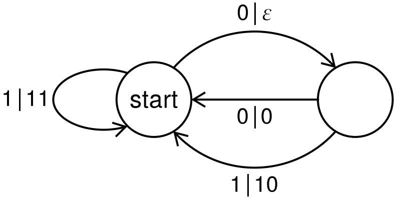
The conjugacy problem in $2V$ *is unsolvable*

Arguably $2V$ is the simplest "naturally occurring" example of a group with solvable word problem and unsolvable conjugacy problem.

Transducers

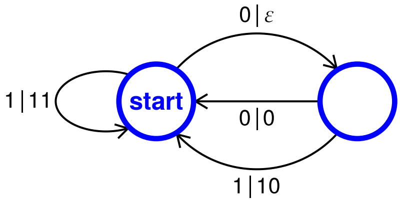
Transducers

A ***transducer*** (or ***Mealy automaton***) is a machine for processing strings over a finite alphabet A.



Transducers

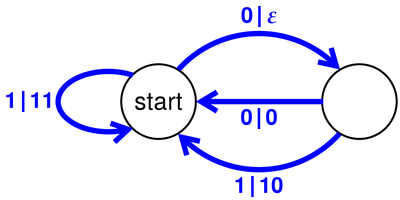
A ***transducer*** (or ***Mealy automaton***) is a machine for processing strings over a finite alphabet A.



It has finitely many ***states***, one of which is the ***start state***.

Transducers

A ***transducer*** (or ***Mealy automaton***) is a machine for processing strings over a finite alphabet A.



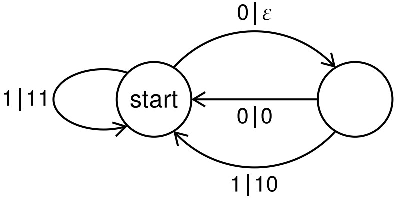
There are ***transitions*** between the states:

$$\xrightarrow{\;\;p \mid q\;\;} \quad \text{input } p \text{ and output } q.$$

The ***input*** must be 0 or 1, but the ***output*** can be any binary string.

Transducers

A ***transducer*** (or ***Mealy automaton***) is a machine for processing strings over a finite alphabet A.



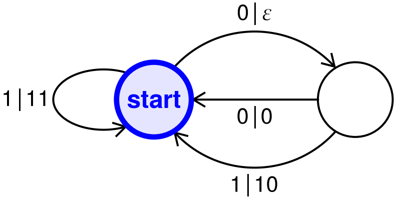
Input String:

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

Output String:

Transducers

A ***transducer*** (or ***Mealy automaton***) is a machine for processing strings over a finite alphabet A.



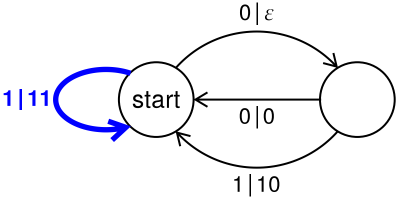
Input String:

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

Output String:

Transducers

A ***transducer*** (or ***Mealy automaton***) is a machine for processing strings over a finite alphabet A.



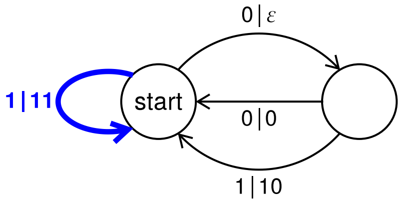
Input String:

| 1 1 0 1 0 0 1 0 0 0 ⋯

Output String:

Transducers

A ***transducer*** (or ***Mealy automaton***) is a machine for processing strings over a finite alphabet *A*.



Input String:

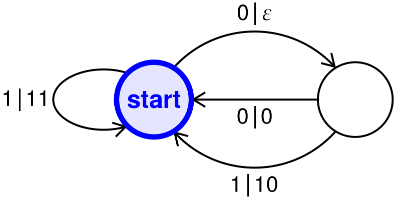
1 | 1 0 1 0 0 1 0 0 0 $\cdots$

Output String:

1 1

Transducers

A ***transducer*** (or ***Mealy automaton***) is a machine for processing strings over a finite alphabet A.



Input String:

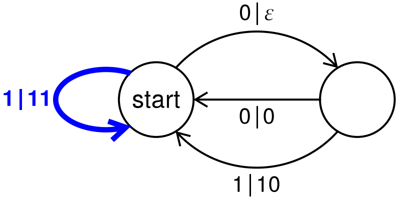
1 | 1 0 1 0 0 1 0 0 0 ⋯

Output String:

1 1

Transducers

A ***transducer*** (or ***Mealy automaton***) is a machine for processing strings over a finite alphabet A.



Input String:

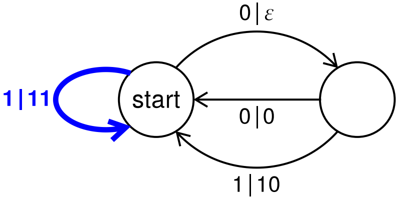
1 | 1 0 1 0 0 1 0 0 0 ⋯

Output String:

1 1

Transducers

A ***transducer*** (or ***Mealy automaton***) is a machine for processing strings over a finite alphabet A.



Input String:

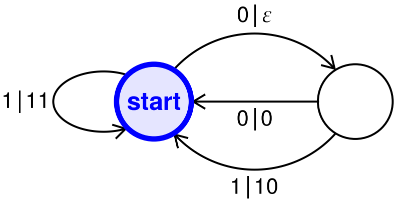
1 1 | 0 1 0 0 1 0 0 0 ⋯

Output String:

1 1 1 1

Transducers

A ***transducer*** (or ***Mealy automaton***) is a machine for processing strings over a finite alphabet A.



Input String:

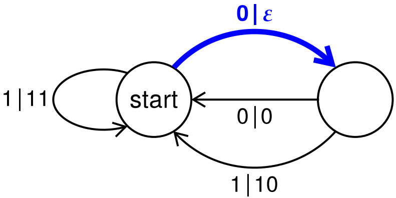
1 1 | 0 1 0 0 1 0 0 0 ⋯

Output String:

1 1 1 1

Transducers

A ***transducer*** (or ***Mealy automaton***) is a machine for processing strings over a finite alphabet A.



Input String:

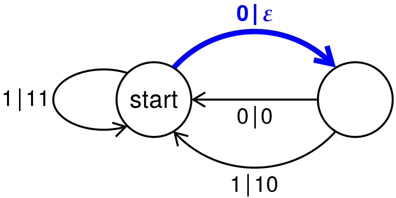
1 1 | 0 1 0 0 1 0 0 0 ···

Output String:

1 1 1 1

Transducers

A ***transducer*** (or ***Mealy automaton***) is a machine for processing strings over a finite alphabet A.



Input String:

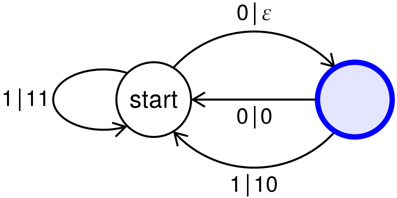
1 1 0 | 1 0 0 1 0 0 0 $\cdots$

Output String:

1 1 1 1

Transducers

A ***transducer*** (or ***Mealy automaton***) is a machine for processing strings over a finite alphabet A.



Input String:

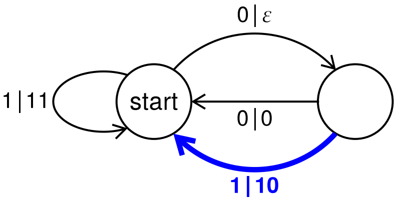
1 1 0 | 1 0 0 1 0 0 0 · · ·

Output String:

1 1 1 1

Transducers

A ***transducer*** (or ***Mealy automaton***) is a machine for processing strings over a finite alphabet *A*.



Input String:

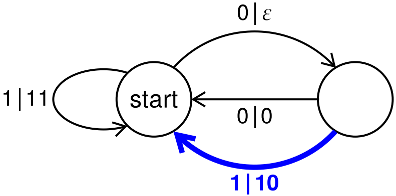
1 1 0 | 1 0 0 1 0 0 0 · · ·

Output String:

1 1 1 1

Transducers

A ***transducer*** (or ***Mealy automaton***) is a machine for processing strings over a finite alphabet A.



Input String:

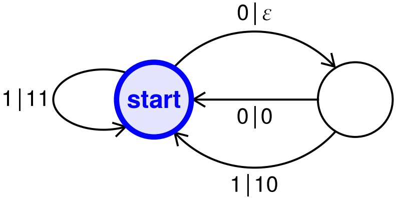
1 1 0 1 | 0 0 1 0 0 0 · · ·

Output String:

1 1 1 1 1 0

Transducers

A ***transducer*** (or ***Mealy automaton***) is a machine for processing strings over a finite alphabet A.



Input String:

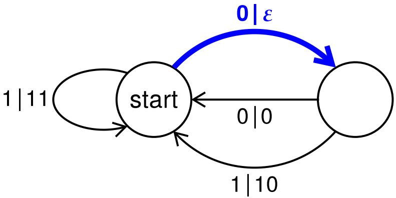
1 1 0 1 | 0 0 1 0 0 0 ⋯

Output String:

1 1 1 1 1 0

Transducers

A ***transducer*** (or ***Mealy automaton***) is a machine for processing strings over a finite alphabet A.



Input String:

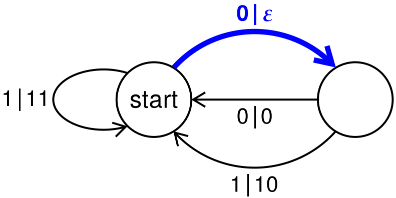
1 1 0 1 | 0 0 1 0 0 0 ...

Output String:

1 1 1 1 1 0

Transducers

A ***transducer*** (or ***Mealy automaton***) is a machine for processing strings over a finite alphabet A.



Input String:

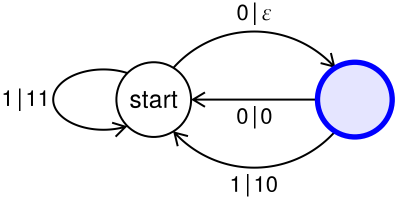
1 1 0 1 0 | 0 1 0 0 0 $\cdots$

Output String:

1 1 1 1 1 0

Transducers

A ***transducer*** (or ***Mealy automaton***) is a machine for processing strings over a finite alphabet A.



Input String:

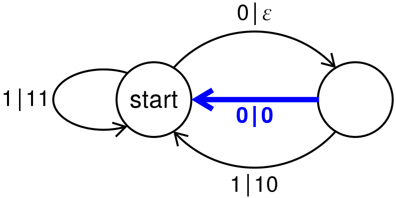
1 1 0 1 0 | 0 1 0 0 0 ⋯

Output String:

1 1 1 1 1 0

Transducers

A ***transducer*** (or ***Mealy automaton***) is a machine for processing strings over a finite alphabet A.



Input String:

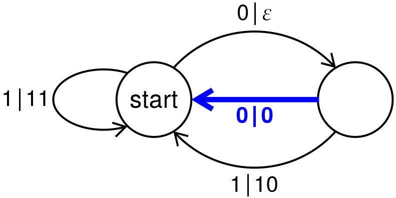
1 1 0 1 0 | 0 1 0 0 0 ⋯

Output String:

1 1 1 1 1 0

Transducers

A ***transducer*** (or ***Mealy automaton***) is a machine for processing strings over a finite alphabet A.



Input String:

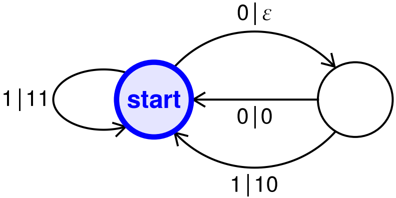
1 1 0 1 0 0 | 1 0 0 0 ⋯

Output String:

1 1 1 1 1 0 0

Transducers

A ***transducer*** (or ***Mealy automaton***) is a machine for processing strings over a finite alphabet A.



Input String:

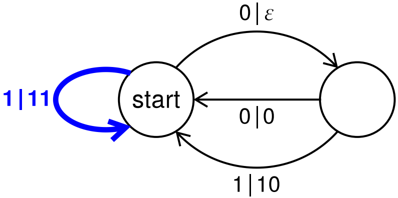
1 1 0 1 0 0 | 1 0 0 0 $\cdots$

Output String:

1 1 1 1 1 0 0

Transducers

A ***transducer*** (or ***Mealy automaton***) is a machine for processing strings over a finite alphabet A.



Input String:

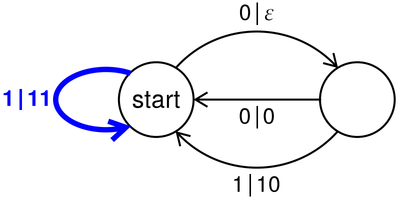
1 1 0 1 0 0 | 1 0 0 0 · · ·

Output String:

1 1 1 1 1 0 0

Transducers

A ***transducer*** (or ***Mealy automaton***) is a machine for processing strings over a finite alphabet A.



Input String:

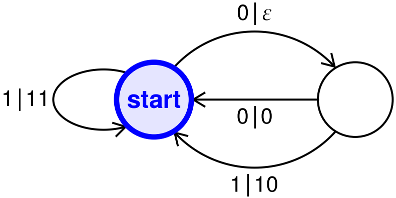
1 1 0 1 0 0 1 | 0 0 0 ···

Output String:

1 1 1 1 1 0 0 1 1

Transducers

A ***transducer*** (or ***Mealy automaton***) is a machine for processing strings over a finite alphabet A.



Input String:

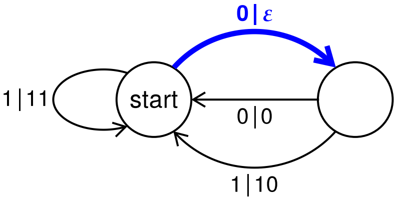
1 1 0 1 0 0 1 | 0 0 0 · · ·

Output String:

1 1 1 1 1 0 0 1 1

Transducers

A ***transducer*** (or ***Mealy automaton***) is a machine for processing strings over a finite alphabet A.



Input String:

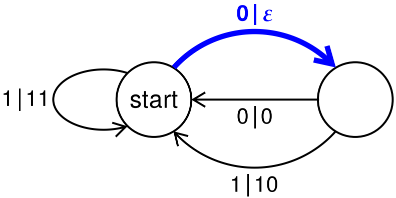
1 1 0 1 0 0 1 | 0 0 0 ···

Output String:

1 1 1 1 1 0 0 1 1

Transducers

A ***transducer*** (or ***Mealy automaton***) is a machine for processing strings over a finite alphabet A.



Input String:

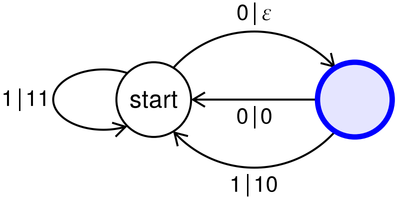
1 1 0 1 0 0 1 0 | 0 0 ⋯

Output String:

1 1 1 1 1 0 0 1 1

Transducers

A ***transducer*** (or ***Mealy automaton***) is a machine for processing strings over a finite alphabet A.



Input String:

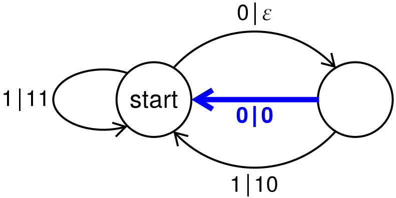
1 1 0 1 0 0 1 0 | 0 0 ...

Output String:

1 1 1 1 1 0 0 1 1

Transducers

A ***transducer*** (or ***Mealy automaton***) is a machine for processing strings over a finite alphabet A.



Input String:

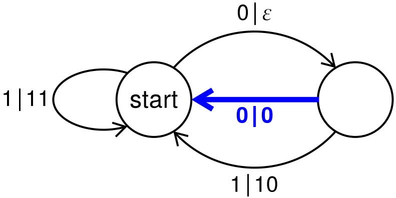
1 1 0 1 0 0 1 0 | 0 0 ···

Output String:

1 1 1 1 1 0 0 1 1

Transducers

A ***transducer*** (or ***Mealy automaton***) is a machine for processing strings over a finite alphabet A.



Input String:

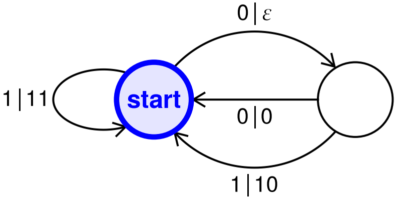
1 1 0 1 0 0 1 0 0 | 0 ···

Output String:

1 1 1 1 1 0 0 1 1 0

Transducers

A ***transducer*** (or ***Mealy automaton***) is a machine for processing strings over a finite alphabet A.



Input String:

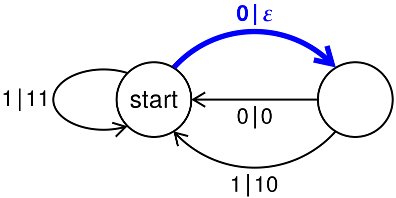
1 1 0 1 0 0 1 0 0 | 0 $\cdots$

Output String:

1 1 1 1 1 0 0 1 1 0

Transducers

A ***transducer*** (or ***Mealy automaton***) is a machine for processing strings over a finite alphabet A.



Input String:

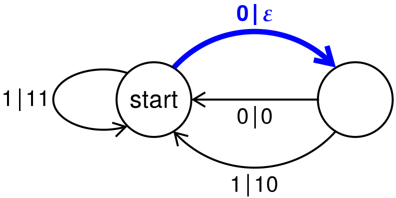
1 1 0 1 0 0 1 0 0 | 0 $\cdots$

Output String:

1 1 1 1 1 0 0 1 1 0

Transducers

A ***transducer*** (or ***Mealy automaton***) is a machine for processing strings over a finite alphabet A.



Input String:

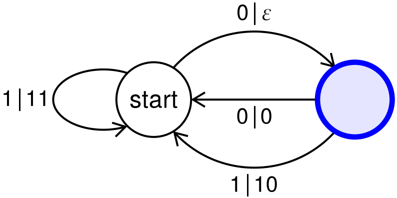
1 1 0 1 0 0 1 0 0 0 | ⋯

Output String:

1 1 1 1 1 0 0 1 1 0

Transducers

A ***transducer*** (or ***Mealy automaton***) is a machine for processing strings over a finite alphabet A.



Input String:

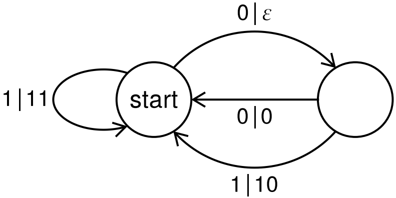
1 1 0 1 0 0 1 0 0 0 | ⋯

Output String:

1 1 1 1 1 0 0 1 1 0

Transducers

A ***transducer*** (or ***Mealy automaton***) is a machine for processing strings over a finite alphabet A.



Input String:

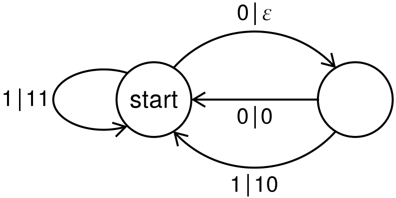
1 1 0 1 0 0 1 0 0 0 $\cdots$

Output String:

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

Transducers

A ***transducer*** (or ***Mealy machine***) is a machine for processing strings over a finite alphabet A.



So the transducer defines a function

$$A^\omega \longrightarrow A^\omega$$

Such a function is a homeomorphism as long as it is bijective.

The Finiteness Question

Question (Grigorchuk–Nekrasevych–Sushchanskii 2000)

Is it possible to decide whether a given set of transducers generate a finite group?

- Bondarenko–Bondarenko–Sidki–Zapata 2010: **Yes** for groups generated by a single “bounded” transducer
- Akhavi–Klimann–Lombardy–Mairesse–Picantin 2011: Several partial results.
- Klinman 2012: **Yes** for two-state transducers, and **yes** for invertible-reversible transducers over a two-letter alphabet.
- Gillibert 2013: **No** in the case of semigroups.

Main Result

Theorem (B–Bleak 2017)

There is no algorithm to decide whether a given transducer has finite order.

Strategy: Simulate elements of $2V$ using transducers.

Main Result

Theorem (B–Bleak 2017)

There is no algorithm to decide whether a given transducer has finite order.

Strategy: Simulate elements of $2V$ using transducers.

The transducers we used are asynchronous, but this turns out to be unnecessary:

Theorem (Gillibert 2018 and Bartholdi–Mitrofanov 2020)

There is no algorithm to decide whether a given synchronous transducer has finite order.

Sketch of Proof

Elements of $2V$ as Transducers

Given a point in the Cantor square, we can combine the two coordinates together:

$$(0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ \cdots,\ 1\ 1\ 1\ 0\ 0\ 0\ 1\ 1\ 1\ \cdots)$$

$$\downarrow$$

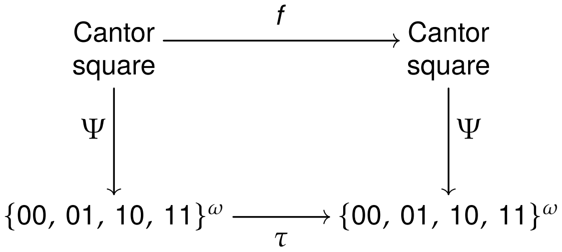
$$01,\ 11,\ 01,\ 10,\ 00,\ 10,\ 01,\ 11,\ 01,\ \ldots$$

This gives a homeomorphism

$$\Psi\colon (\text{Cantor square}) \longrightarrow \{00, 01, 10, 11\}^{\omega}$$

Elements of $2V$ as Transducers

Using this homeomorphism, any $f \in 2V$ induces a homeomorphism of $\{00, 01, 10, 11\}^\omega$.



We must show that τ is a transducer.

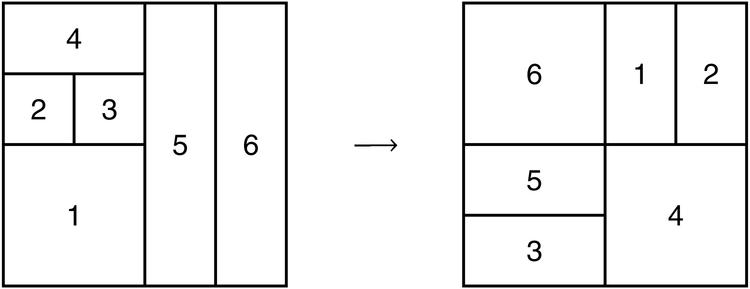
Elements of $2V$ as Transducers

Elements of $2V$ act as prefix pair replacements

$$(0\psi, 0\omega) \mapsto (10\psi, 1\omega) \qquad (00\psi, 10\omega) \mapsto (11\psi, 1\omega)$$

$$(01\psi, 10\omega) \mapsto (0\psi, 00\omega) \qquad (0\psi, 11\omega) \mapsto (1\psi, 0\omega)$$

$$(10\psi, \omega) \mapsto (0\psi, 01\omega) \qquad (11\psi, \omega) \mapsto (0\psi, 1\omega)$$



Elements of $2V$ as Transducers

Elements of $2V$ act as prefix pair replacements

$$(0\psi, 0\omega) \mapsto (10\psi, 1\omega) \qquad (00\psi, 10\omega) \mapsto (11\psi, 1\omega)$$

$$(01\psi, 10\omega) \mapsto (0\psi, 00\omega) \qquad (0\psi, 11\omega) \mapsto (1\psi, 0\omega)$$

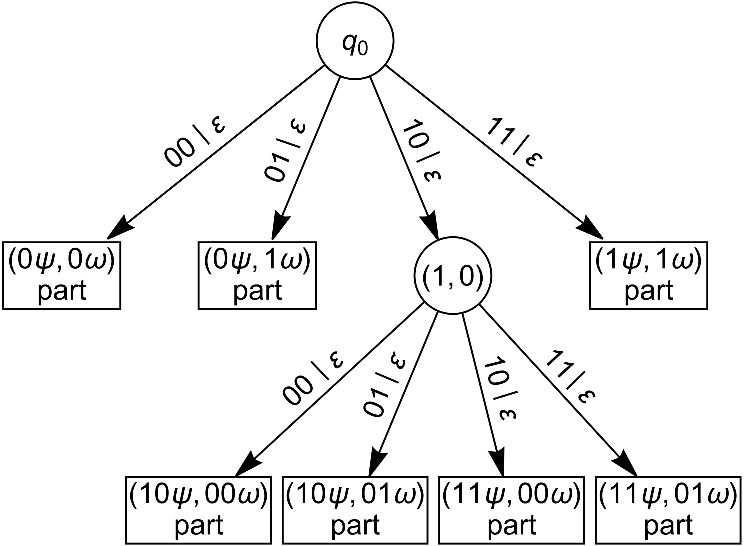
$$(10\psi, \omega) \mapsto (0\psi, 01\omega) \qquad (11\psi, \omega) \mapsto (0\psi, 1\omega)$$

So the transducer has two tasks:

1. Input digits until we recognize the prefix.
2. Output the new prefix, followed by the remaining digits.

Elements of $2V$ as Transducers

Recognizing the prefix is easy. There's a tree of possibilities.



Elements of $2V$ as Transducers

Outputting the new prefix and the remaining digits requires a trick.

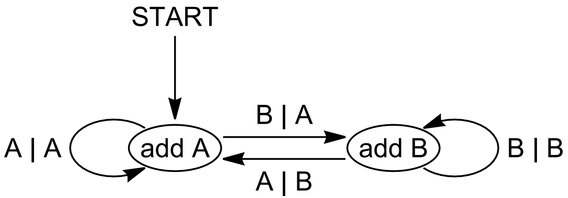
Elements of 2V as Transducers

Outputting the new prefix and the remaining digits requires a trick.

The Trick

Transducers can remember things.

Here's a synchronous transducer that adds the letter "A" to the beginning of a string:



It's always one letter behind, but it "remembers" this letter.

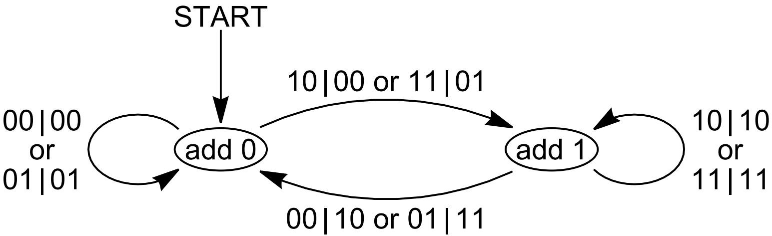
Elements of $2V$ as Transducers

Outputting the new prefix and the remaining digits requires a trick.

The Trick

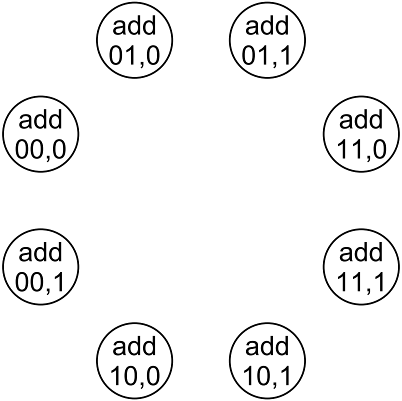
Transducers can remember things.

This transducer adds a "0" to the beginning of the *x*-coordinate.



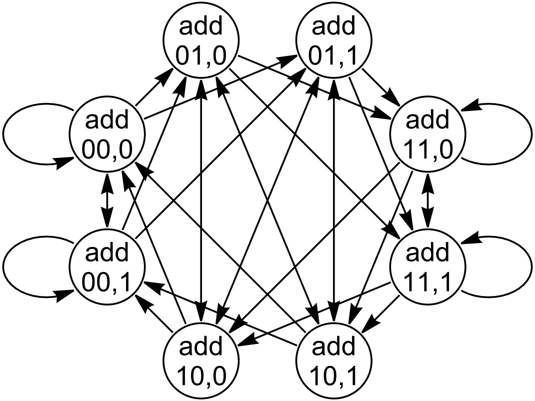
Adding the New Prefix

Using this trick, we can add a prefix to x and a prefix to y.



Adding the New Prefix

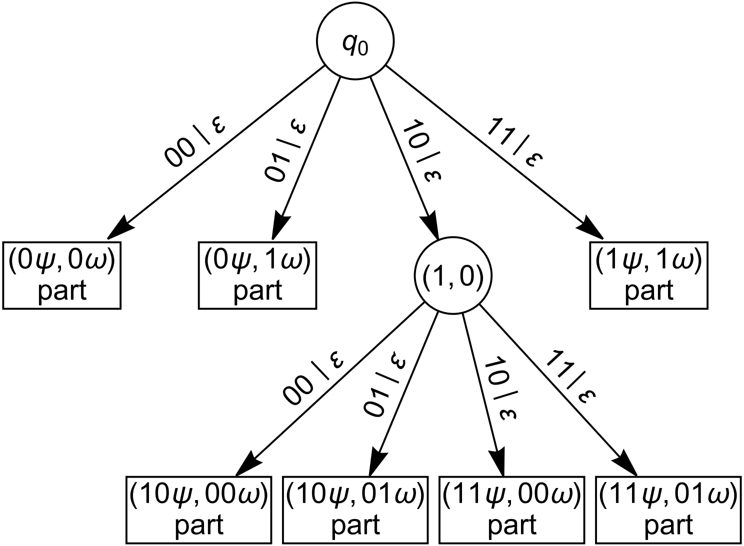
Using this trick, we can add a prefix to x and a prefix to y.



We must remember a *queue* of digits for each coordinate.

Elements of $2V$ as Transducers

We need one of these prefix-adding transducers for each part.



Elements of $2V$ as Transducers

$$\begin{array}{ccc} \text{Cantor square} & \xrightarrow{\;f \in 2V\;} & \text{Cantor square} \\ \Big\downarrow \Psi & & \Big\downarrow \Psi \\ \{00, 01, 10, 11\}^{\omega} & \xrightarrow[\;\tau\;]{} & \{00, 01, 10, 11\}^{\omega} \end{array}$$

This proves that τ is a transducer, so there is no algorithm that decides whether a transducer has finite order.

The End