Transducers

Introduction and Mealy machines

Transducers

Introduction and Mealy machines

Transducers

Introduction and Mealy machines

Transducers

Introduction and Mealy machines

Transducers

Introduction and Mealy machines

A transducer is an automaton that has non-trivial outputs

A transducer is an automaton that has non-trivial outputs

could be one-way,
two-way,
deterministic,
nondeterministic, ...

A transducer is an automaton that has non-trivial outputs

could be one-way,
two-way,
deterministic,
nondeterministic, ...

could be strings,
trees, numbers,
graphs, ...

A transducer is an automaton that has non-trivial outputs

could be one-way,
two-way,
deterministic,
nondeterministic, ...

could be strings,
trees, numbers,
graphs, ...

Example. Replace e by o: meet $\mapsto$ moot

A transducer is an automaton that has non-trivial outputs

could be one-way,
two-way,
deterministic,
nondeterministic, ...

could be strings,
trees, numbers,
graphs, ...

Example. Replace e by o: meet ↦ moot
Example. Replace ee by oo: meet ↦ moot met ↦ met

A transducer is an automaton that has non-trivial outputs

could be one-way,
two-way,
deterministic,
nondeterministic, ...

could be strings,
trees, numbers,
graphs, ...

Example. Replace e by o: meet ⟼ moot
Example. Replace ee by oo: meet ⟼ moot met ⟼ met
Example. Duplicate: meet ⟼ meetmeet

A transducer is an automaton that has non-trivial outputs

could be one-way,
two-way,
deterministic,
nondeterministic, ...

could be strings,
trees, numbers,
graphs, ...

Example. Replace e by o: meet ⟼ moot
Example. Replace ee by oo: meet ⟼ moot met ⟼ met
Example. Duplicate: meet ⟼ meetmeet

These examples are string-to-string, but there could be:
string-to-number, tree-to-string, tree-to-tree, graph-to-graph, etc.

A transducer is an automaton that has non-trivial outputs

could be one-way,
two-way,
deterministic,
nondeterministic, ...

could be strings,
trees, numbers,
graphs, ...

Example. Replace e by o: meet ⟼ moot
Example. Replace ee by oo: meet ⟼ moot met ⟼ met
Example. Duplicate: meet ⟼ meetmeet

These examples are string-to-string, but there could be:
string-to-number, tree-to-string, tree-to-tree, graph-to-graph, etc.
The usual languages are string-to-Boolean.

A transducer is an automaton that has non-trivial outputs

could be one-way,
two-way,
deterministic,
nondeterministic, ...

could be strings,
trees, numbers,
graphs, ...

Example. Replace e by o: meet ⟼ moot
Example. Replace ee by oo: meet ⟼ moot met ⟼ met
Example. Duplicate: meet ⟼ meetmeet

These examples are string-to-string, but there could be:
string-to-number, tree-to-string, tree-to-tree, graph-to-graph, etc.
The usual languages are string-to-Boolean.

If you thought that there were not enough automata models, you will be happy to hear about transducers.

A transducer is an automaton that has non-trivial outputs

could be one-way,
two-way,
deterministic,
nondeterministic, ...

could be strings,
trees, numbers,
graphs, ...

Example. Replace e by o: meet ⟼ moot
Example. Replace ee by oo: meet ⟼ moot met ⟼ met
Example. Duplicate: meet ⟼ meetmeet

These examples are string-to-string, but there could be:
string-to-number, tree-to-string, tree-to-tree, graph-to-graph, etc.
The usual languages are string-to-Boolean.

If you thought that there were not enough automata models, you will be happy to hear about transducers.

Added bonus: unlike for automata, many of the models have different expressive power.

A transducer is an automaton that has non-trivial outputs

could be one-way,
two-way,
deterministic,
nondeterministic, ...

could be strings,
trees, numbers,
graphs, ...

Example. Replace e by o: meet ↦ moot
Example. Replace ee by oo: meet ↦ moot met ↦ met
Example. Duplicate: meet ↦ meetmeet

These examples are string-to-string, but there could be:
string-to-number, tree-to-string, tree-to-tree, graph-to-graph, etc.
The usual languages are string-to-Boolean.

If you thought that there were not enough automata models, you will be happy to hear about transducers.

Added bonus: unlike for automata, many of the models have different expressive power. For example, one-way ≠ two-way.

A transducer is an automaton that has non-trivial outputs

could be one-way,
two-way,
deterministic,
nondeterministic, ...

could be strings,
trees, numbers,
graphs, ...

Example. Replace e by o: meet ↦ moot
Example. Replace ee by oo: meet ↦ moot met ↦ met
Example. Duplicate: meet ↦ meetmeet

These examples are string-to-string, but there could be:
string-to-number, tree-to-string, tree-to-tree, graph-to-graph, etc.
The usual languages are string-to-Boolean.

Not all is allowed!

If you thought that there were not enough automata models, you will be happy to hear about transducers.

Added bonus: unlike for automata, many of the models have different expressive power. For example, one-way ≠ two-way.

A transducer is an automaton that has non-trivial outputs

could be one-way,
two-way,
deterministic,
nondeterministic, ...

could be strings,
trees, numbers,
graphs, ...

Example. Replace e by o: meet ⟼ moot
Example. Replace ee by oo: meet ⟼ moot met ⟼ met
Example. Duplicate: meet ⟼ meetmeet

These examples are string-to-string, but there could be:
string-to-number, tree-to-string, tree-to-tree, graph-to-graph, etc.
The usual languages are string-to-Boolean.

If you thought that there were not enough automata models, you will be happy to hear about transducers.

Added bonus: unlike for automata, many of the models have different expressive power. For example, one-way ≠ two-way.

Not all is allowed!

Stacks are not allowed

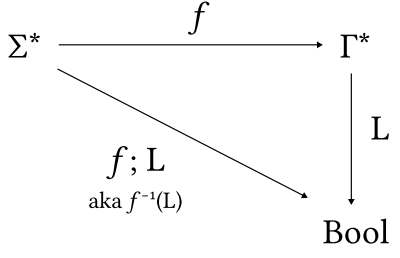
Continuity axiom

Continuity axiom

Definition. A string-to-string function is *continuous* if for every regular property of the outputs, the corresponding property of the inputs is also regular

Continuity axiom

Definition. A string-to-string function is *continuous* if for every regular property of the outputs, the corresponding property of the inputs is also regular



Example. Reversing $123 \mapsto 321$ is continuous.

Example. Reversing $123 \mapsto 321$ is continuous.

This means that if L is regular, then so is $\{ w \mid (\text{reverse of } w) \in \mathrm{L} \}$

Example. Reversing $123 \mapsto 321$ is continuous.

This means that if L is regular, then so is $\{ w \mid (\text{reverse of } w) \in \mathrm{L} \}$

This is a classic automata exercise. Solution: reverse an NFA.

Example. Reversing $123 \mapsto 321$ is continuous.

This means that if L is regular, then so is $\{ w \mid (\text{reverse of } w) \in L \}$

This is a classic automata exercise. Solution: reverse an NFA.

Example. Duplicating $123 \mapsto 123123$ is continuous.

Example. Reversing $123 \mapsto 321$ is continuous.

This means that if L is regular, then so is $\{ w \mid (\text{reverse of } w) \in \mathrm{L} \}$

This is a classic automata exercise. Solution: reverse an NFA.

Example. Duplicating $123 \mapsto 123123$ is continuous.

This means that if L is regular, then so is $\{ w \mid ww \in \mathrm{L} \}$

Example. Reversing $123 \mapsto 321$ is continuous.

This means that if L is regular, then so is $\{ w \mid (\text{reverse of } w) \in L \}$

This is a classic automata exercise. Solution: reverse an NFA.

Example. Duplicating $123 \mapsto 123123$ is continuous.

This means that if L is regular, then so is $\{ w \mid ww \in L \}$

Another classic automata exercise. Solution: guess the state between the two copies.

Example. Reversing $123 \mapsto 321$ is continuous.

This means that if L is regular, then so is $\{ w \mid (\text{reverse of } w) \in L \}$

This is a classic automata exercise. Solution: reverse an NFA.

Example. Duplicating $123 \mapsto 123123$ is continuous.

This means that if L is regular, then so is $\{ w \mid ww \in L \}$

Another classic automata exercise. Solution: guess the state between the two copies.

Example. Squaring $123 \mapsto 123123123$ is continuous.

Example. Reversing $123 \mapsto 321$ is continuous.

This means that if L is regular, then so is $\{ w \mid (\text{reverse of } w) \in L \}$

This is a classic automata exercise. Solution: reverse an NFA.

Example. Duplicating $123 \mapsto 123123$ is continuous.

This means that if L is regular, then so is $\{ w \mid ww \in L \}$

Another classic automata exercise. Solution: guess the state between the two copies.

Example. Squaring $123 \mapsto 123123123$ is continuous.

This means that if L is regular, then so is $\{ w \mid w^{|w|} \in L \}$

Example. Reversing $123 \mapsto 321$ is continuous.

This means that if L is regular, then so is $\{ w \mid (\text{reverse of } w) \in L \}$

This is a classic automata exercise. Solution: reverse an NFA.

Example. Duplicating $123 \mapsto 123123$ is continuous.

This means that if L is regular, then so is $\{ w \mid ww \in L \}$

Another classic automata exercise. Solution: guess the state between the two copies.

Example. Squaring $123 \mapsto 123123123$ is continuous.

This means that if L is regular, then so is $\{ w \mid w^{|w|} \in L \}$

Yet another classic (problem 31 in the 200 book).

Example. Reversing $123 \mapsto 321$ is continuous.

This means that if L is regular, then so is $\{ w \mid (\text{reverse of } w) \in L \}$

This is a classic automata exercise. Solution: reverse an NFA.

Example. Duplicating $123 \mapsto 123123$ is continuous.

This means that if L is regular, then so is $\{ w \mid ww \in L \}$

Another classic automata exercise. Solution: guess the state between the two copies.

Example. Squaring $123 \mapsto 123123123$ is continuous.

This means that if L is regular, then so is $\{ w \mid w^{|w|} \in L \}$

Yet another classic (problem 31 in the 200 book).

Nonexample. Binary addition $0100 + 1001 \mapsto 1101$ is not continuous.

Example. Reversing $123 \mapsto 321$ is continuous.

This means that if L is regular, then so is $\{ w \mid (\text{reverse of } w) \in L \}$

This is a classic automata exercise. Solution: reverse an NFA.

Example. Duplicating $123 \mapsto 123123$ is continuous.

This means that if L is regular, then so is $\{ w \mid ww \in L \}$

Another classic automata exercise. Solution: guess the state between the two copies.

Example. Squaring $123 \mapsto 123123123$ is continuous.

This means that if L is regular, then so is $\{ w \mid w^{|w|} \in L \}$

Yet another classic (problem 31 in the 200 book).

Nonexample. Binary addition $0100 + 1001 \mapsto 1101$ is not continuous.

If the input is not the addition of two numbers, then the output is $\perp$.

Example. Reversing $123 \mapsto 321$ is continuous.

This means that if L is regular, then so is $\{ w \mid (\text{reverse of } w) \in \mathrm{L} \}$

This is a classic automata exercise. Solution: reverse an NFA.

Example. Duplicating $123 \mapsto 123123$ is continuous.

This means that if L is regular, then so is $\{ w \mid ww \in \mathrm{L} \}$

Another classic automata exercise. Solution: guess the state between the two copies.

Example. Squaring $123 \mapsto 123123123$ is continuous.

This means that if L is regular, then so is $\{ w \mid w^{|w|} \in \mathrm{L} \}$

Yet another classic (problem 31 in the 200 book).

Nonexample. Binary addition $0100 + 1001 \mapsto 1101$ is not continuous.

If the input is not the addition of two numbers, then the output is $\perp$.

If the second argument is given in reverse, then the function could be implemented with a stack.

Example. Reversing $123 \mapsto 321$ is continuous.

This means that if L is regular, then so is $\{ w \mid (\text{reverse of } w) \in \mathrm{L} \}$

This is a classic automata exercise. Solution: reverse an NFA.

Example. Duplicating $123 \mapsto 123123$ is continuous.

This means that if L is regular, then so is $\{ w \mid ww \in \mathrm{L} \}$

Another classic automata exercise. Solution: guess the state between the two copies.

Example. Squaring $123 \mapsto 123123123$ is continuous.

This means that if L is regular, then so is $\{ w \mid w^{|w|} \in \mathrm{L} \}$

Yet another classic (problem 31 in the 200 book).

Nonexample. Binary addition $0100 + 1001 \mapsto 1101$ is not continuous.

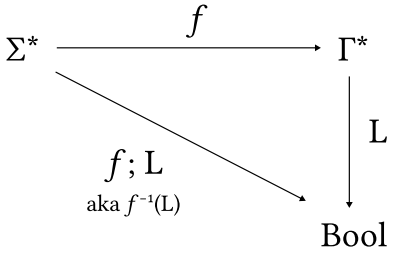
If the input is not the addition of two numbers, then the output is $\perp$.

If the second argument is given in reverse, then the function could be implemented with a stack.

The inverse image of the language 1^* is not regular.

Continuity axiom

Definition. A string-to-string function is *continuous* if for every regular property of the outputs, the corresponding property of the inputs is also regular



Example. Reversing 123 ↦ 321 is continuous.
This means that if L is regular, then so is { w | (reverse of w) ∈ L }
This is a classic automata exercise. Solution: reverse an NFA.

Example. Duplicating 123 ↦ 123123 is continuous.
This means that if L is regular, then so is { w | ww ∈ L }
Another classic automata exercise. Solution: guess the state between the two copies.

Example. Squaring 123 ↦ 123123123 is continuous.
This means that if L is regular, then so is { w | $w^{|w|} \in L$ }
Yet another classic (problem 31 in the 200 book).

Nonexample. Binary addition 0100 + 1001 ↦ 1101 is not continuous.
If the input is not the addition of two numbers, then the output is ⊥.
If the second argument is given in reverse, then the function could be implemented with a stack.
The inverse image of the language 1* is not regular.

A transducer is an automaton that has non-trivial outputs

could be one-way,
two-way,
deterministic,
nondeterministic, ...

could be strings,
trees, numbers,
graphs, ...

Example. Replace e by o: meet ⟼ moot
Example. Replace ee by oo: meet ⟼ moot met ⟼ met
Example. Duplicate: meet ⟼ meetmeet

These examples are string-to-string, but there could be:
string-to-number, tree-to-string, tree-to-tree, graph-to-graph, etc.
The usual languages are string-to-Boolean.

If you thought that there were not enough automata models, you will be happy to hear about transducers.

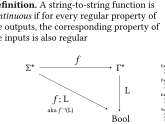
Added bonus: unlike for automata, many of the models have different expressive power. For example, one-way ≠ two-way.

Not all is allowed!

Stacks are not allowed

Continuity axiom

Definition. A string-to-string function is *continuous* if for every regular property of the outputs, the corresponding property of the inputs is also regular



We will consider three classes of transducers:

We will consider three classes of transducers:

These are string-to-string models, but maybe we will also look at trees or graphs

1. Mealy machines

1. Mealy machines

A DFA where each transition is labelled by one output letter

1. Mealy machines

A DFA where each transition is labelled by one output letter

1234567 ↦ _123456

1. Mealy machines

   A DFA where each transition is labelled by one output letter

   $1234567 \mapsto \_123456$

2. Linear regular functions

1. Mealy machines

   A DFA where each transition is labelled by one output letter

   1234567 ↦ _123456

2. Linear regular functions

   A 2-DFA where each transition is labelled by a possibly empty output string

1. Mealy machines

   A DFA where each transition is labelled by one output letter

   1234567 ↦ _123456

2. Linear regular functions

   A 2-DFA where each transition is labelled by a possibly empty output string

   1234 ↦ 1234 1234

1. Mealy machines

   A DFA where each transition is labelled by one output letter

   $1234567 \mapsto \_123456$

2. Linear regular functions

   A 2-DFA where each transition is labelled by a possibly empty output string

   $1234 \mapsto 1234\ 1234$

3. Polynomial regular functions

1. Mealy machines

   A DFA where each transition is labelled by one output letter

   1234567 ↦ _123456

2. Linear regular functions

   A 2-DFA where each transition is labelled by a possibly empty output string

   1234 ↦ 1234 1234

3. Polynomial regular functions

   A program that can do nested loops over the input string

1. Mealy machines

   A DFA where each transition is labelled by one output letter

   1234567 ↦ _123456

2. Linear regular functions

   A 2-DFA where each transition is labelled by a possibly empty output string

   1234 ↦ 1234 1234

3. Polynomial regular functions

   A program that can do nested loops over the input string

   1234 ↦ 1234 1234 1234 1234

We will consider three classes of transducers:

These are string-to-string models, but maybe we will also look at trees or graphs

1. Mealy machines
   A DFA where each transition is labelled by one output letter
   1234567 ↦ _123456

2. Linear regular functions
   A 2-DFA where each transition is labelled by a possibly empty output string
   1234 ↦ 1234 1234

3. Polynomial regular functions
   A program that can do nested loops over the input string
   1234 ↦ 1234 1234 1234 1234

We will consider three classes of transducers:

These are string-to-string models, but maybe we will also look at trees or graphs

1. Mealy machines
   A DFA where each transition is labelled by one output letter
   1234567 ↦ _123456
2. Linear regular functions
   A 2-DFA where each transition is labelled by a possibly empty output string
   1234 ↦ 1234 1234
3. Polynomial regular functions
   A program that can do nested loops over the input string
   1234 ↦ 1234 1234 1234 1234

We will be interested in results like:

We will consider three classes of transducers:

These are string-to-string models, but maybe we will also look at trees or graphs

1. Mealy machines
   A DFA where each transition is labelled by one output letter
   1234567 ↦ _123456
2. Linear regular functions
   A 2-DFA where each transition is labelled by a possibly empty output string
   1234 ↦ 1234 1234
3. Polynomial regular functions
   A program that can do nested loops over the input string
   1234 ↦ 1234 1234 1234 1234

We will be interested in results like:

Transducer models X, Y and Z define the same functions.

We will consider three classes of transducers:

These are string-to-string models, but maybe we will also look at trees or graphs

1. Mealy machines
   A DFA where each transition is labelled by one output letter
   1234567 ↔ _123456
2. Linear regular functions
   A 2-DFA where each transition is labelled by a possibly empty output string
   1234 ↔ 1234 1234
3. Polynomial regular functions
   A program that can do nested loops over the input string
   1234 ↔ 1234 1234 1234 1234

We will be interested in results like:

Transducer models X, Y and Z define the same functions.

Each transducer from class X is a composition of transducers from class Y.

We will consider three classes of transducers:

These are string-to-string models, but maybe we will also look at trees or graphs

1. Mealy machines
   A DFA where each transition is labelled by one output letter
   1234567 ↔ _123456
2. Linear regular functions
   A 2-DFA where each transition is labelled by a possibly empty output string
   1234 ↔ 1234 1234
3. Polynomial regular functions
   A program that can do nested loops over the input string
   1234 ↔ 1234 1234 1234 1234

We will be interested in results like:

Transducer models X, Y and Z define the same functions.

Each transducer from class X is a composition of transducers from class Y.

Given transducers $f, g \in$ X, one can decide if $f = g$.

Transducers

Introduction and Mealy machines

A transducer is an automaton that has non-trivial outputs

could be one-way, two-way, deterministic, nondeterministic, ... (automaton)

could be strings, trees, numbers, graphs, ... (outputs)

Example. Replace e by o: meet ↦ moot
Example. Replace ee by oo: meet ↦ moot met ↦ met
Example. Duplicate: meet ↦ meetmeet

These examples are string-to-string, but there could be: string-to-number, tree-to-string, tree-to-tree, graph-to-graph, etc. The usual languages are string-to-Boolean.

If you thought that there were not enough automata models, you will be happy to hear about transducers.

Added bonus: unlike for automata, many of the models have different expressive power. For example, one-way ≠ two-way.

Not all is allowed!
Stacks are not allowed

Continuity axiom

Definition. A string-to-string function is continuous if for every regular property of the outputs, the corresponding property of the inputs is also regular

We will consider three classes of transducers:
These are string-to-string models, but maybe we will also look at trees or graphs

1. Mealy machines
   A DFA where each transition is labelled by one output letter
2. Linear regular functions
   A 2-DFA where each transition is labelled by a possibly empty output string
3. Polynomial regular functions
   A program that can do nested loops over the input string

We will be interested in results like:

Transducer models X, Y and Z define the same functions.
Each transducer from class X is a composition of transducers from class Y.
Given transducers $f, g \in X$, one can decide if $f = g$.

Transducers

Introduction and Mealy machines

A transducer is an automaton that has non-trivial outputs

could be one-way, two-way, deterministic, nondeterministic, ...

could be strings, trees, numbers, graphs, ...

Example. Replace e by o: meet ↦ moot
Example. Replace ee by oo: meet ↦ moot met ↦ met
Example. Duplicate: meet ↦ meetmeet

These examples are string-to-string, but there could be: string-to-number, tree-to-string, tree-to-tree, graph-to-graph, etc. The usual languages are string-to-Boolean.

If you thought that there were not enough automata models, you will be happy to hear about transducers.

Added bonus: unlike for automata, many of the models have different expressive power. For example, one-way ≠ two-way.

Not all is allowed!
Stacks are not allowed

Continuity axiom

Definition. A string-to-string function is continuous if for every regular property of the outputs, the corresponding property of the inputs is also regular

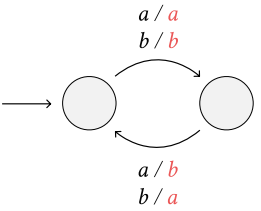
We will consider three classes of transducers:
These are string-to-string models, but maybe we will also look at trees or graphs

1. Mealy machines
2. Linear regular functions
3. Polynomial regular functions

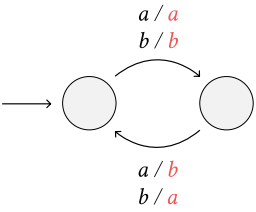
We will be interested in results like:

Transducer models X, Y and Z define the same functions.
Each transducer from class X is a composition of transducers from class Y.
Given transducers $f, g \in X$, one can decide if $f = g$.

We begin our course with Mealy machines, which are the simplest transducer model

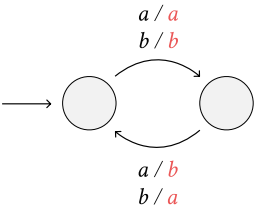

a / a
b / b
a / b
b / a

A Mealy machine is a DFA, but:



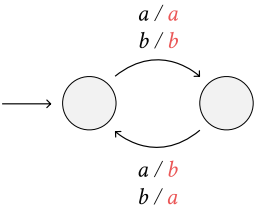
A Mealy machine is a DFA, but:

- each transition is labelled by one output letter



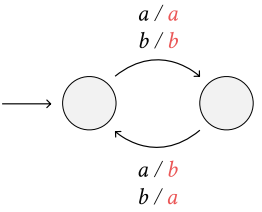
A Mealy machine is a DFA, but:

- each transition is labelled by one output letter
  and therefore it defines a letter-to-letter function (i.e. length preserving)



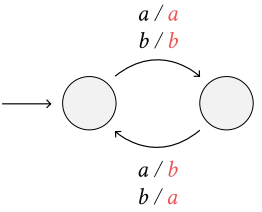
A Mealy machine is a DFA, but:

- each transition is labelled by one output letter
  and therefore it defines a letter-to-letter function (i.e. length preserving)
- we do not define accepting states



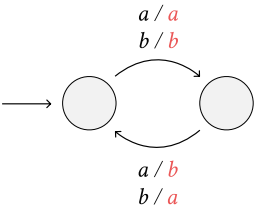
A Mealy machine is a DFA, but:

- each transition is labelled by one output letter
  and therefore it defines a letter-to-letter function (i.e. length preserving)
- we do not define accepting states
  it does not accept or reject, but only defines an output string



A Mealy machine is a DFA, but:

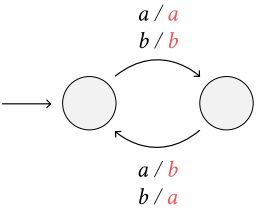
- each transition is labelled by one output letter
  and therefore it defines a letter-to-letter function (i.e. length preserving)
- we do not define accepting states
  it does not accept or reject, but only defines an output string



Formal definition.

A Mealy machine is a DFA, but:

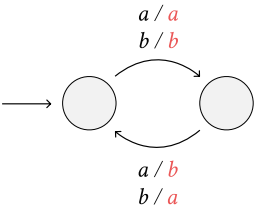
- each transition is labelled by one output letter
  and therefore it defines a letter-to-letter function (i.e. length preserving)
- we do not define accepting states
  it does not accept or reject, but only defines an output string



Formal definition. Σ input alphabet

A Mealy machine is a DFA, but:

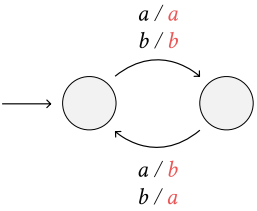
- each transition is labelled by one output letter
  and therefore it defines a letter-to-letter function (i.e. length preserving)
- we do not define accepting states
  it does not accept or reject, but only defines an output string



Formal definition. Σ input alphabet Γ output alphabet

A Mealy machine is a DFA, but:

- each transition is labelled by one output letter
  and therefore it defines a letter-to-letter function (i.e. length preserving)
- we do not define accepting states
  it does not accept or reject, but only defines an output string

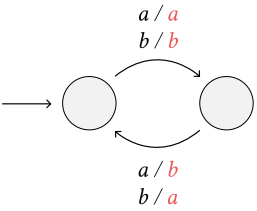


Formal definition.

Σ	Γ	Q
input alphabet	output alphabet	states

A Mealy machine is a DFA, but:

- each transition is labelled by one output letter
  and therefore it defines a letter-to-letter function (i.e. length preserving)
- we do not define accepting states
  it does not accept or reject, but only defines an output string

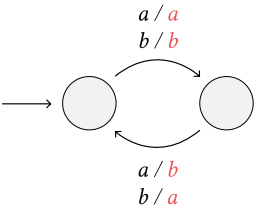


Formal definition.

Σ	Γ	Q	$q_0 \in Q$
input alphabet	output alphabet	states	initial state

A Mealy machine is a DFA, but:

- each transition is labelled by one output letter
  and therefore it defines a letter-to-letter function (i.e. length preserving)
- we do not define accepting states
  it does not accept or reject, but only defines an output string

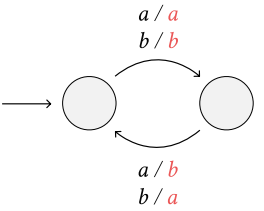


Formal definition.

Σ	Γ	Q	$q_0 \in Q$	$\delta : Q \times \Sigma \longrightarrow Q \times \Gamma$
input alphabet	output alphabet	states	initial state	transition function

A Mealy machine is a DFA, but:

- each transition is labelled by one output letter
  and therefore it defines a letter-to-letter function (i.e. length preserving)
- we do not define accepting states
  it does not accept or reject, but only defines an output string



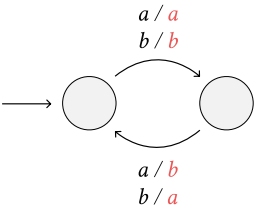
Formal definition.

Σ	Γ	Q	$q_0 \in Q$	$\delta : Q \times \Sigma \longrightarrow Q \times \Gamma$
input alphabet	output alphabet	states	initial state	transition function

Examples

A Mealy machine is a DFA, but:

- each transition is labelled by one output letter
  and therefore it defines a letter-to-letter function (i.e. length preserving)
- we do not define accepting states
  it does not accept or reject, but only defines an output string



Formal definition.

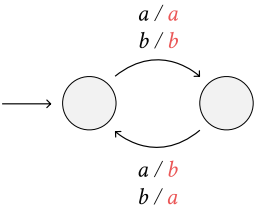
Σ	Γ	Q	$q_0 \in Q$	$\delta : Q \times \Sigma \longrightarrow Q \times \Gamma$
input alphabet	output alphabet	states	initial state	transition function

Examples

Basic Properties

A Mealy machine is a DFA, but:

- each transition is labelled by one output letter
  and therefore it defines a letter-to-letter function (i.e. length preserving)
- we do not define accepting states
  it does not accept or reject, but only defines an output string



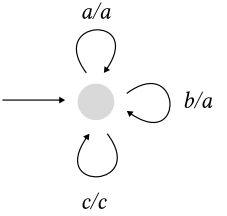
Formal definition.

Σ	Γ	Q	$q_0 \in Q$	$\delta : Q \times \Sigma \longrightarrow Q \times \Gamma$
input alphabet	output alphabet	states	initial state	transition function

Examples Basic Properties Krohn-Rhodes Theorem

Example. One-state Mealy machines are the same as letter-to-letter homomorphisms.

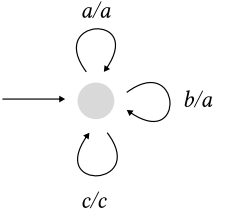
This is a function $\Sigma^* \longrightarrow \Gamma^*$ obtained by lifting some function $\Sigma \longrightarrow \Gamma$



Example. One-state Mealy machines are the same as letter-to-letter homomorphisms.

This is a function $\Sigma^* \longrightarrow \Gamma^*$ obtained by lifting some function $\Sigma \longrightarrow \Gamma$

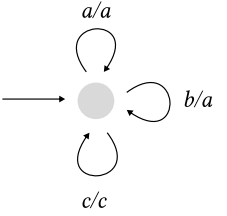
A (not necessarily letter-to-letter) homomorphism lifts a function $\Sigma \longrightarrow \Gamma^*$



Example. One-state Mealy machines are the same as letter-to-letter homomorphisms.

This is a function $\Sigma^* \longrightarrow \Gamma^*$ obtained by lifting some function $\Sigma \longrightarrow \Gamma$

A (not necessarily letter-to-letter) homomorphism lifts a function $\Sigma \longrightarrow \Gamma^*$



Example. The flip-flop machine.

Example. The flip-flop machine.

Pay attention, because it will appear in the Krohn-Rhodes Theorem

Example. The flip-flop machine.

Pay attention, because it will appear in the Krohn-Rhodes Theorem

For an alphabet Σ, define a transducer $(\Sigma + \bigcirc)^* \longrightarrow (\Sigma + \bigcirc)^*$ as follows.

Example. The flip-flop machine.

Pay attention, because it will appear in the Krohn-Rhodes Theorem
For an alphabet Σ, define a transducer $(\Sigma + \bigcirc)^* \longrightarrow (\Sigma + \bigcirc)^*$ as follows.

The n-th output letter is the last input letter $< n$ that was not $\bigcirc$

Example. The flip-flop machine.

Pay attention, because it will appear in the Krohn-Rhodes Theorem
For an alphabet Σ, define a transducer $(\Sigma + \bigcirc)^* \longrightarrow (\Sigma + \bigcirc)^*$ as follows.

The n-th output letter is the last input letter $< n$ that was not $\bigcirc$

$\bigcirc$	$\bigcirc$	a	$\bigcirc$	$\bigcirc$	b	c	$\bigcirc$	a	$\bigcirc$	input

Example. The flip-flop machine.

Pay attention, because it will appear in the Krohn-Rhodes Theorem
For an alphabet Σ, define a transducer $(\Sigma + \bigcirc)^* \longrightarrow (\Sigma + \bigcirc)^*$ as follows.

The n-th output letter is the last input letter $< n$ that was not $\bigcirc$

$\bigcirc$	$\bigcirc$	a	$\bigcirc$	$\bigcirc$	b	c	$\bigcirc$	a	$\bigcirc$	input
$\bigcirc$	$\bigcirc$	$\bigcirc$	a	a	a	b	c	c	a	a	states

Example. The flip-flop machine.

Pay attention, because it will appear in the Krohn-Rhodes Theorem
For an alphabet Σ, define a transducer $(\Sigma + \bigcirc)^* \longrightarrow (\Sigma + \bigcirc)^*$ as follows.

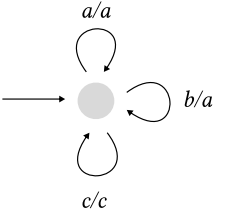
The n-th output letter is the last input letter $< n$ that was not $\bigcirc$

	○	○	*a*	○	○	*b*	*c*	○	*a*	○	input
○	○	○	*a*	*a*	*a*	*b*	*c*	*c*	*a*	*a*	states
	○	○	○	*a*	*a*	*a*	*b*	*c*	*c*	*a*	output

Example. One-state Mealy machines are the same as letter-to-letter homomorphisms.

This is a function $\Sigma^* \longrightarrow \Gamma^*$ obtained by lifting some function $\Sigma \longrightarrow \Gamma$

A (not necessarily letter-to-letter) homomorphism lifts a function $\Sigma \longrightarrow \Gamma^*$



Example. The flip-flop machine.

Pay attention, because it will appear in the Krohn-Rhodes Theorem

For an alphabet Σ, define a transducer $(\Sigma + \bigcirc)^* \longrightarrow (\Sigma + \bigcirc)^*$ as follows.

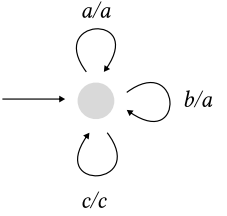
The n-th output letter is the last input letter $< n$ that was not $\bigcirc$

	○	○	*a*	○	○	*b*	*c*	○	*a*	○	input
○	○	○	*a*	*a*	*a*	*b*	*c*	*c*	*a*	*a*	states
	○	○	○	*a*	*a*	*a*	*b*	*c*	*c*	*a*	output

Example. One-state Mealy machines are the same as letter-to-letter homomorphisms.

This is a function $\Sigma^* \longrightarrow \Gamma^*$ obtained by lifting some function $\Sigma \longrightarrow \Gamma$

A (not necessarily letter-to-letter) homomorphism lifts a function $\Sigma \longrightarrow \Gamma^*$



Example. The flip-flop machine.

Pay attention, because it will appear in the Krohn-Rhodes Theorem

For an alphabet Σ, define a transducer $(\Sigma + \bigcirc)^* \longrightarrow (\Sigma + \bigcirc)^*$ as follows.

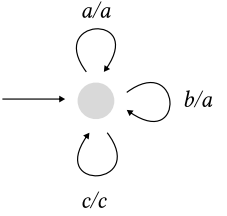
The n-th output letter is the last input letter $< n$ that was not $\bigcirc$

	○	○	*a*	○	○	*b*	*c*	○	*a*	○	input
○	○	○	*a*	*a*	*a*	*b*	*c*	*c*	*a*	*a*	states
	○	○	○	*a*	*a*	*a*	*b*	*c*	*c*	*a*	output

Example. Mealy machines with a two-letter output alphabet are the same as regular languages.

Example. One-state Mealy machines are the same as letter-to-letter homomorphisms.

This is a function $\Sigma^* \longrightarrow \Gamma^*$ obtained by lifting some function $\Sigma \longrightarrow \Gamma$
A (not necessarily letter-to-letter) homomorphism lifts a function $\Sigma \longrightarrow \Gamma^*$



Example. The flip-flop machine.

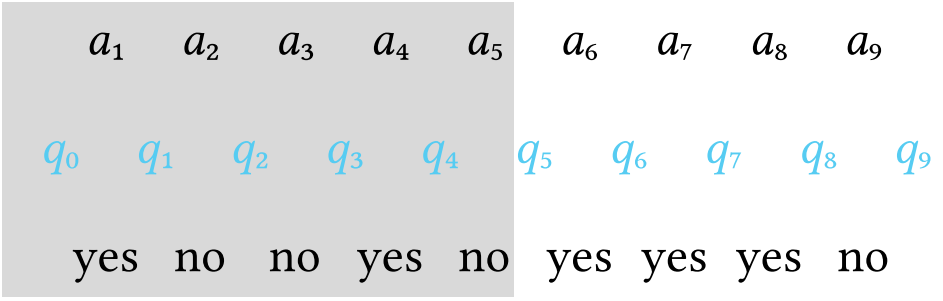
Pay attention, because it will appear in the Krohn-Rhodes Theorem
For an alphabet Σ, define a transducer $(\Sigma + \bigcirc)^* \longrightarrow (\Sigma + \bigcirc)^*$ as follows.

The n-th output letter is the last input letter $< n$ that was not $\bigcirc$

○	○	*a*	○	○	*b*	*c*	○	*a*	○	input
○	○	○	*a*	*a*	*a*	*b*	*c*	*c*	*a*	*a*	states
○	○	○	*a*	*a*	*a*	*b*	*c*	*c*	*a*	output

Example. Mealy machines with a two-letter output alphabet are the same as regular languages.

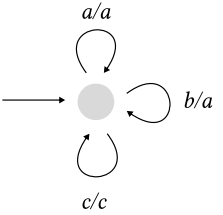
General Mealy machines the same as partitions of Σ^* into $|\Gamma|$ regular parts.



the n-th output letter can be seen as a decision for the first n input letters.

Example. One-state Mealy machines are the same as letter-to-letter homomorphisms.

This is a function $\Sigma^* \longrightarrow \Gamma^*$ obtained by lifting some function $\Sigma \longrightarrow \Gamma$
A (not necessarily letter-to-letter) homomorphism lifts a function $\Sigma \longrightarrow \Gamma^*$



Example. The flip-flop machine.

Pay attention, because it will appear in the Krohn-Rhodes Theorem
For an alphabet Σ, define a transducer $(\Sigma + \bigcirc)^* \longrightarrow (\Sigma + \bigcirc)^*$ as follows.

The n-th output letter is the last input letter $< n$ that was not $\bigcirc$

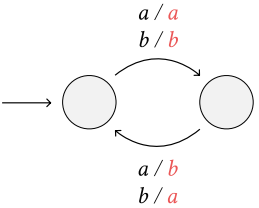
○	○	*a*	○	○	*b*	*c*	○	*a*	○	input
○	○	○	*a*	*a*	*a*	*b*	*c*	*c*	*a*	*a*	states
○	○	○	*a*	*a*	*a*	*b*	*c*	*c*	*a*	output

Example. Mealy machines with a two-letter output alphabet are the same as regular languages.

General Mealy machines the same as partitions of Σ^* into $|\Gamma|$ regular parts.

A Mealy machine is a DFA, but:

- each transition is labelled by one output letter
  and therefore it defines a letter-to-letter function (i.e. length preserving)
- we do not define accepting states
  it does not accept or reject, but only defines an output string



Formal definition.

Σ	Γ	Q	$q_0 \in Q$	$\delta : Q \times \Sigma \longrightarrow Q \times \Gamma$
input alphabet	output alphabet	states	initial state	transition function

Examples Basic Properties Krohn-Rhodes Theorem


Example. One-state Mealy machines are the same as letter-to-letter homomorphisms.
This is a function $\Sigma^* \longrightarrow \Gamma^*$ obtained by lifting some function $\Sigma \longrightarrow \Gamma$.
A (not necessarily letter-to-letter) homomorphism Πh: a function $\Sigma \longrightarrow \Gamma^*$.

Example. The flip-flop machine.

Theorem. (Functions computed by) Mealy machines are closed under composition.

Theorem. (Functions computed by) Mealy machines are closed under composition.

Proof. A product construction.


Σ*
f
Δ*
g
Γ*

a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_8 a_9

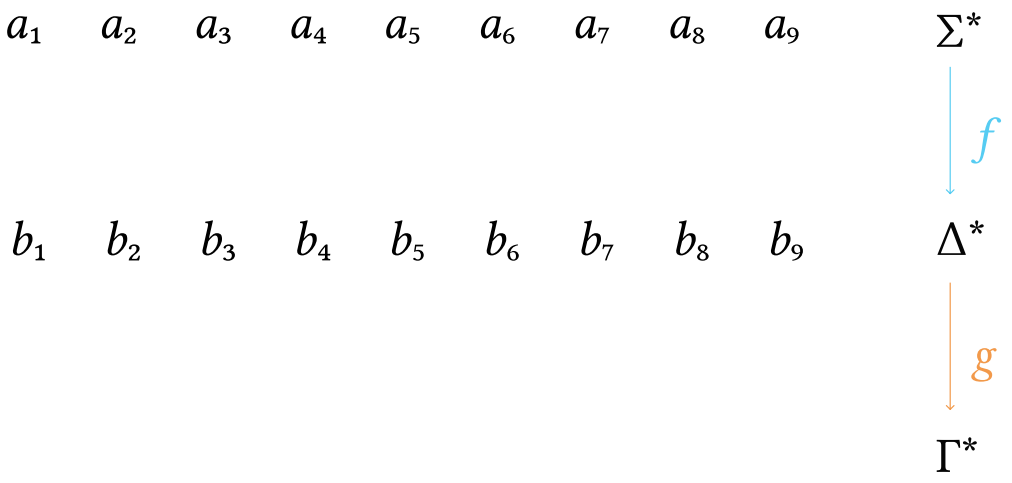
Σ^*

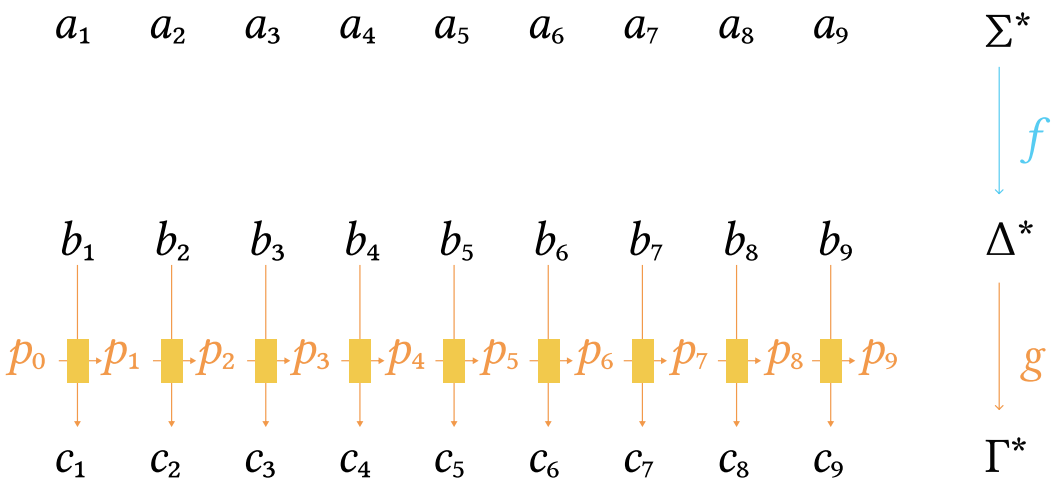
f

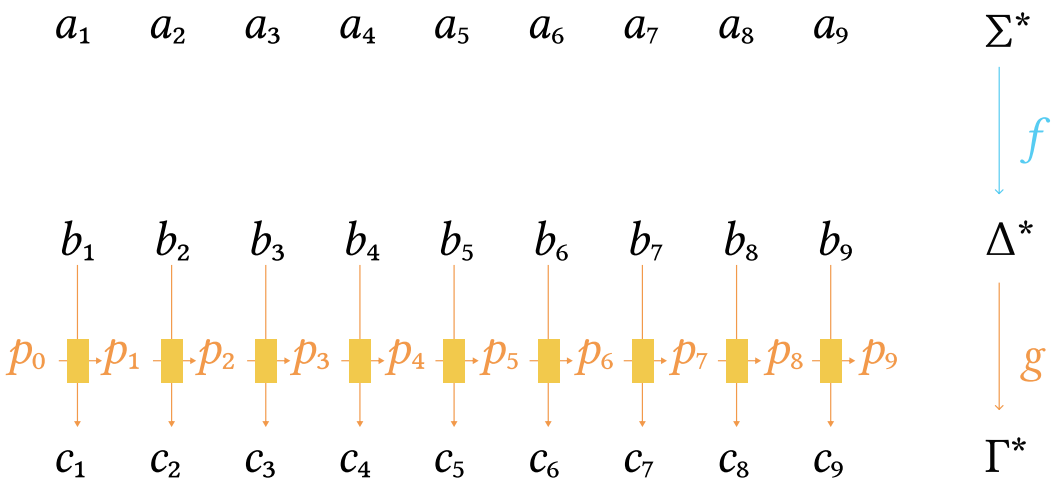
Δ^*

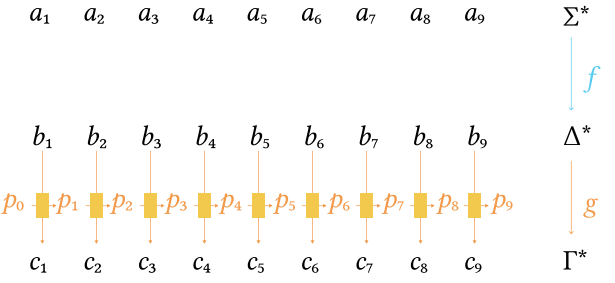
g

Γ^*


a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_8 a_9
Σ^*
f
b_1 b_2 b_3 b_4 b_5 b_6 b_7 b_8 b_9
Δ^*
g
Γ^*

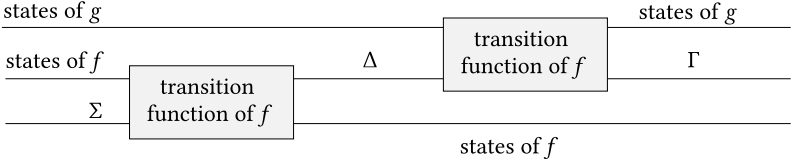

a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_8 a_9
Σ^*
f
b_1 b_2 b_3 b_4 b_5 b_6 b_7 b_8 b_9
Δ^*
p_0 p_1 p_2 p_3 p_4 p_5 p_6 p_7 p_8 p_9
g
c_1 c_2 c_3 c_4 c_5 c_6 c_7 c_8 c_9
Γ^*

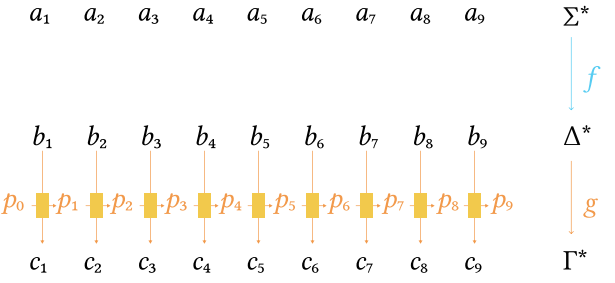

a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_8 a_9
Σ^*
f
b_1 b_2 b_3 b_4 b_5 b_6 b_7 b_8 b_9
Δ^*
p_0 p_1 p_2 p_3 p_4 p_5 p_6 p_7 p_8 p_9
g
c_1 c_2 c_3 c_4 c_5 c_6 c_7 c_8 c_9
Γ^*



The states for the composition are (states of f) × (states of g)

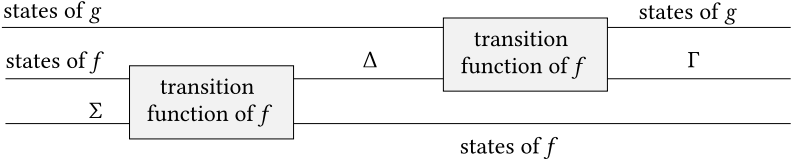
The initial state is the pair of initial states, and the transition function is

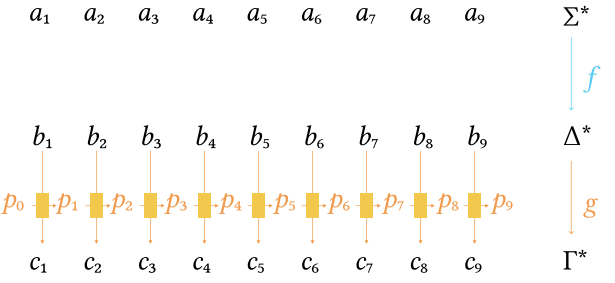




The states for the composition are (states of f) × (states of g)

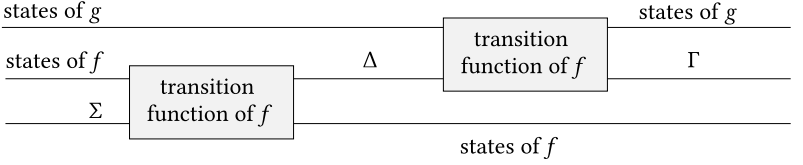
The initial state is the pair of initial states, and the transition function is

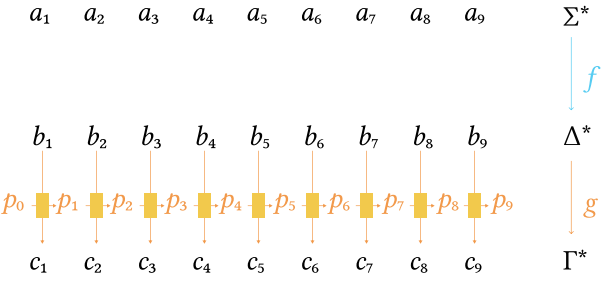




The states for the composition are (states of f) × (states of g)

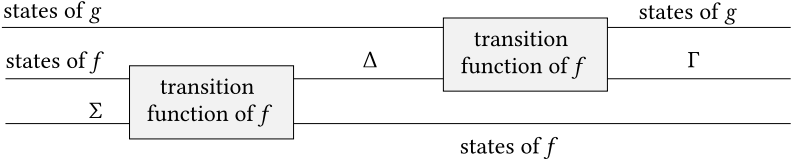
The initial state is the pair of initial states, and the transition function is





The states for the composition are (states of f) × (states of g)

The initial state is the pair of initial states, and the transition function is

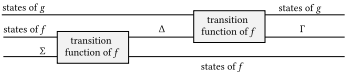


Theorem. (Functions computed by) Mealy machines are closed under composition.

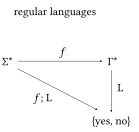
Proof. A product construction.

The states for the composition are (states of f) × (states of g)

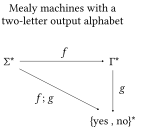
The initial state is the pair of initial states, and the transition function is



Corollary. Mealy machines are continuous, i.e. inverse images of regular languages are regular.



continuity



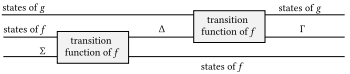
closure under composition

Theorem. (Functions computed by) Mealy machines are closed under composition.

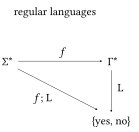
Proof. A product construction.

The states for the composition are (states of f) × (states of g)

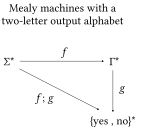
The initial state is the pair of initial states, and the transition function is



Corollary. Mealy machines are continuous, i.e. inverse images of regular languages are regular.

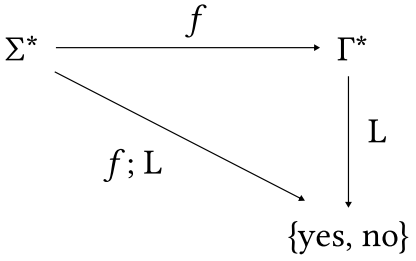


continuity



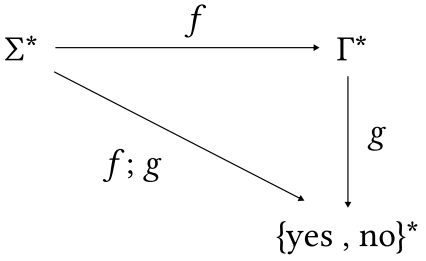
closure under composition

regular languages



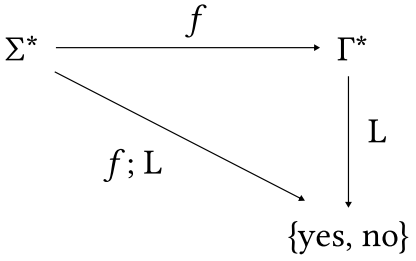
continuity

Mealy machines with a two-letter output alphabet



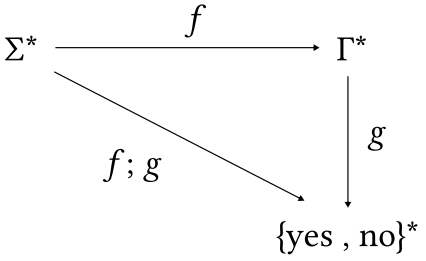
closure under composition

regular languages



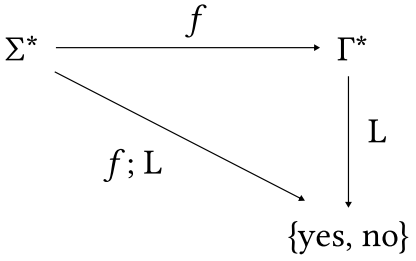
continuity

Mealy machines with a two-letter output alphabet



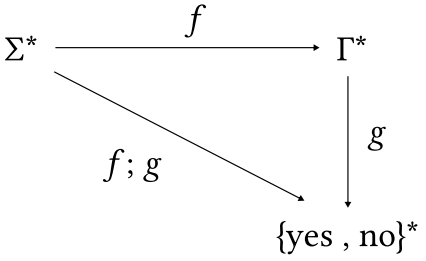
closure under composition

regular languages



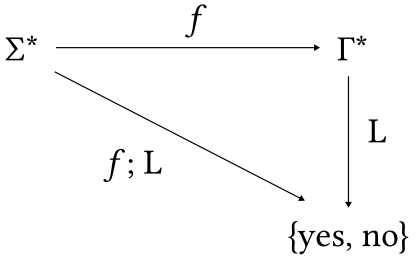
continuity

Mealy machines with a two-letter output alphabet



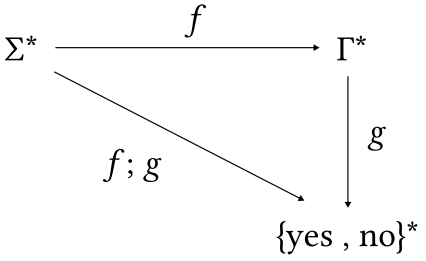
closure under composition

regular languages



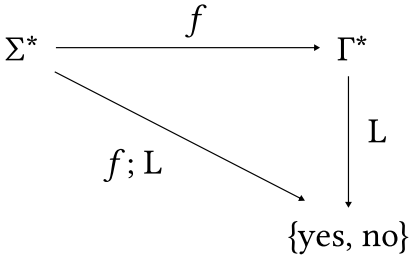
continuity

Mealy machines with a two-letter output alphabet



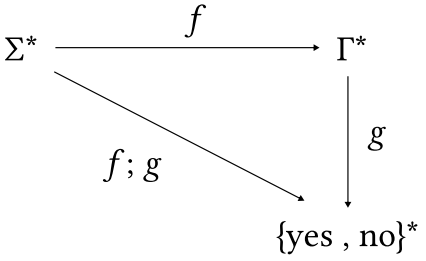
closure under composition

regular languages



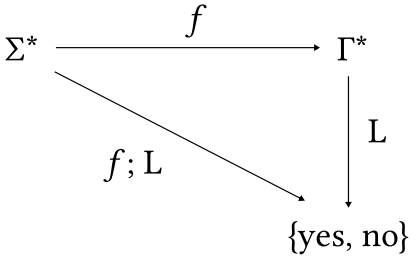
continuity

Mealy machines with a two-letter output alphabet



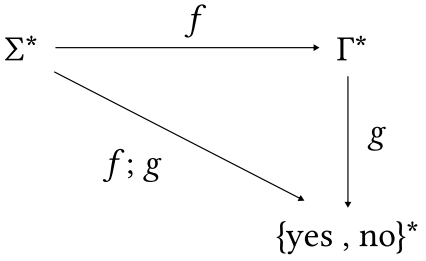
closure under composition

regular languages



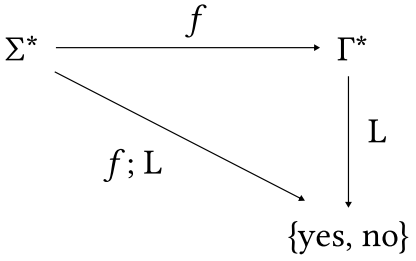
continuity

Mealy machines with a two-letter output alphabet



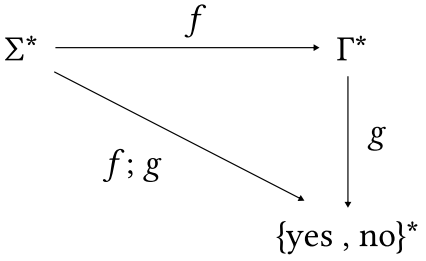
closure under composition

regular languages



continuity

Mealy machines with a two-letter output alphabet



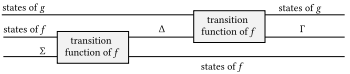
closure under composition

Theorem. (Functions computed by) Mealy machines are closed under composition.

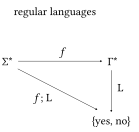
Proof. A product construction.

The states for the composition are (states of f) × (states of g)

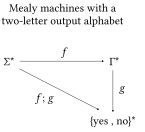
The initial state is the pair of initial states, and the transition function is



Corollary. Mealy machines are continuous, i.e. inverse images of regular languages are regular.



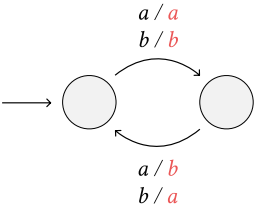
continuity



closure under composition

A Mealy machine is a DFA, but:

- each transition is labelled by one output letter
  and therefore it defines a letter-to-letter function (i.e. length preserving)
- we do not define accepting states
  it does not accept or reject, but only defines an output string



Formal definition.

Σ	Γ	Q	$q_0 \in Q$	$\delta : Q \times \Sigma \longrightarrow Q \times \Gamma$
input alphabet	output alphabet	states	initial state	transition function

Examples

Example. One-state Mealy machines are the same as letter-to-letter homomorphisms.
This is a function $\Sigma^* \longrightarrow \Gamma^*$ obtained by lifting some function $\Sigma \longrightarrow \Gamma$.
A (not necessarily letter-to-letter) homomorphism $lift_h$ a function $\Sigma \longrightarrow \Gamma^*$.

Example. The flip-flop machine.

Basic Properties

Theorem. (Functions computed by) Mealy machines are closed under composition.

Proof. A product construction.

Corollary. Mealy machines are continuous, i.e. inverse images of regular languages are regular.

Krohn-Rhodes Theorem

Theorem. [Krohn-Rhodes 1965] Every Mealy machine f can be decomposed as $f = f_1; ...; f_n$ where each Mealy machine f_i is either:

Reversible: every state transformation is bijection; or
The state transformation of a letter is defined to be the corresponding function $Q \longrightarrow Q$.
If a machine is reversible, and we know the last state, then we can reconstruct the run.

Theorem. [Krohn-Rhodes 1965] Every Mealy machine f can be decomposed as $f = f_1; \ldots; f_n$ where each Mealy machine f_i is either:

Reversible: every state transformation is bijection; or

The state transformation of a letter is defined to be the corresponding function $Q \longrightarrow Q$. If a machine is reversible, and we know the last state, then we can reconstruct the run and output in a right-to-left pass.

Flip-flop: every state transformation is the identity or constant.

In a flip-flop machine, an input letter either keeps the state unchanged, or forgets it and sets it to some new state that depends on the letter

$$\text{Mealy} = (\underbrace{\text{reversible} + \text{flip-flop}})^*$$

we use the name *prime* for these two kinds

Theorem. [Krohn-Rhodes 1965] Every Mealy machine f can be decomposed as $f = f_1; \ldots; f_n$ where each Mealy machine f_i is either:

Reversible: every state transformation is bijection; or

The state transformation of a letter is defined to be the corresponding function $Q \longrightarrow Q$. If a machine is reversible, and we know the last state, then we can reconstruct the run and output in a right-to-left pass.

Flip-flop: every state transformation is the identity or constant.

In a flip-flop machine, an input letter either keeps the state unchanged, or forgets it and sets it to some new state that depends on the letter

$$\text{Mealy} = (\underbrace{\text{reversible} + \text{flip-flop}})^*$$

we use the name *prime* for these two kinds

Theorem. [Krohn-Rhodes 1965] Every Mealy machine f can be decomposed as $f = f_1; \ldots; f_n$ where each Mealy machine f_i is either:

Reversible: every state transformation is bijection; or

The state transformation of a letter is defined to be the corresponding function $Q \longrightarrow Q$. If a machine is reversible, and we know the last state, then we can reconstruct the run and output in a right-to-left pass.

Flip-flop: every state transformation is the identity or constant.

In a flip-flop machine, an input letter either keeps the state unchanged, or forgets it and sets it to some new state that depends on the letter

$$\text{Mealy} = (\underbrace{\text{reversible} + \text{flip-flop}})^*$$

we use the name *prime* for these two kinds

Theorem. [Krohn-Rhodes 1965] Every Mealy machine f can be decomposed as $f = f_1; \ldots; f_n$ where each Mealy machine f_i is either:

Reversible: every state transformation is bijection; or

The state transformation of a letter is defined to be the corresponding function $Q \longrightarrow Q$. If a machine is reversible, and we know the last state, then we can reconstruct the run and output in a right-to-left pass.

Flip-flop: every state transformation is the identity or constant.

In a flip-flop machine, an input letter either keeps the state unchanged, or forgets it and sets it to some new state that depends on the letter

$$\text{Mealy} = (\underbrace{\text{reversible} + \text{flip-flop}})^*$$

we use the name *prime* for these two kinds

Theorem. [Krohn-Rhodes 1965] Every Mealy machine f can be decomposed as $f = f_1; \ldots; f_n$ where each Mealy machine f_i is either:

Reversible: every state transformation is bijection; or

The state transformation of a letter is defined to be the corresponding function $Q \longrightarrow Q$. If a machine is reversible, and we know the last state, then we can reconstruct the run and output in a right-to-left pass.

Flip-flop: every state transformation is the identity or constant.

In a flip-flop machine, an input letter either keeps the state unchanged, or forgets it and sets it to some new state that depends on the letter

$$\text{Mealy} = (\underbrace{\text{reversible} + \text{flip-flop}})^*$$

we use the name *prime* for these two kinds

Theorem. [Krohn-Rhodes 1965] Every Mealy machine f can be decomposed as $f = f_1; \ldots; f_n$ where each Mealy machine f_i is either:

Reversible: every state transformation is bijection; or

The state transformation of a letter is defined to be the corresponding function $Q \longrightarrow Q$. If a machine is reversible, and we know the last state, then we can reconstruct the run and output in a right-to-left pass.

Flip-flop: every state transformation is the identity or constant.

In a flip-flop machine, an input letter either keeps the state unchanged, or forgets it and sets it to some new state that depends on the letter

$$\text{Mealy} = (\underbrace{\text{reversible} + \text{flip-flop}})^*$$

we use the name *prime* for these two kinds

Theorem. [Krohn-Rhodes 1965] Every Mealy machine f can be decomposed as $f = f_1; \ldots; f_n$ where each Mealy machine f_i is either:

Reversible: every state transformation is bijection; or

The state transformation of a letter is defined to be the corresponding function $Q \longrightarrow Q$. If a machine is reversible, and we know the last state, then we can reconstruct the run and output in a right-to-left pass.

Flip-flop: every state transformation is the identity or constant.

In a flip-flop machine, an input letter either keeps the state unchanged, or forgets it and sets it to some new state that depends on the letter

$$\text{Mealy} = (\underbrace{\text{reversible} + \text{flip-flop}})^*$$

we use the name *prime* for these two kinds

Theorem. [Krohn-Rhodes 1965] Every Mealy machine f can be decomposed as $f = f_1; \ldots; f_n$ where each Mealy machine f_i is either:

Reversible: every state transformation is bijection; or

The state transformation of a letter is defined to be the corresponding function $Q \longrightarrow Q$. If a machine is reversible, and we know the last state, then we can reconstruct the run and output in a right-to-left pass.

Flip-flop: every state transformation is the identity or constant.

In a flip-flop machine, an input letter either keeps the state unchanged, or forgets it and sets it to some new state that depends on the letter

$$\text{Mealy} = (\underbrace{\text{reversible} + \text{flip-flop}})^*$$

we use the name *prime* for these two kinds

Mealy = (reversible + flip-flop)*

we use the name *primes* for these two kinds

Clearly primes* ⊆ Mealy.

Because primes ⊆ Mealy and Mealy* = Mealy.

The interesting part is Mealy ⊆ primes*.

Step 1. primes* is closed under map.

For a Mealy machine $f\colon \Sigma^* \longrightarrow \Gamma^*$, define its map to be:

$$w_1 \mid w_2 \mid \ldots \mid w_n \quad \mapsto \quad f(w_1) \mid f(w_2) \mid \ldots \mid f(w_n)$$

the input and output alphabets are extended with a separator |

This is clearly a Mealy machine. But also:

Lemma. If a function is in primes*, then so is its map.

Proof. It is enough to prove that if a function is in primes, then its map is in primes*. because the map of fg is (map f)(map g)

The map of a flip-flop is also a flip-flop.
The separator has a constant state transformation.

The interesting case is when f is reversible.

Step 2. finish the job

We decompose every Mealy machine into primes, using induction on:

1. the number of states
2. the size of the input alphabet

The parameters are ordered lexicographically, which means that the induction advances if we decrease the state space but increase the input alphabet

In the induction basis of one state, the machine is flip-flop. also, reversible.
Consider the induction step.

Either the machine is already reversible, or there is some input letter $a \in \Sigma$ such that the corresponding state transformation $Q \longrightarrow Q$ has an image that is a proper subset of Q.

Intuition: words that end with a fall under item 1 of the induction assumption, and words that do not contain a fall under item 2.

Bonus. We have also proved the Schützenberger Theorem.

A Mealy machine is called *counter-free* if it does not have a counter:

a counter is an input letter that creates a non-trivial cycle on states

non-trivial means that there are at least two different states in the cycle

Counter-free machines are connected to first-order logic.

Theorem. counter-free Mealy = (flip-flop)*.

Proof. If we apply the proof of the Krohn-Rhodes Theorem to a counter-free machine, then counters will not be created, and only flip-flop machines will be needed.

Mealy = (reversible + flip-flop)*

we use the name *primes* for these two kinds

Clearly primes* ⊆ Mealy.

Because primes ⊆ Mealy and Mealy* = Mealy.

The interesting part is Mealy ⊆ primes*.

Step 1. primes* is closed under map.

For a Mealy machine $f\colon \Sigma^* \longrightarrow \Gamma^*$, define its map to be:

$$w_1 \mid w_2 \mid \ldots \mid w_n \quad \mapsto \quad f(w_1) \mid f(w_2) \mid \ldots \mid f(w_n)$$

the input and output alphabets are extended with a separator |

This is clearly a Mealy machine. But also:

Lemma. If a function is in primes*, then so is its map.

Proof. It is enough to prove that if a function is in primes, then its map is in primes*. because the map of fg is (map f)(map g)

The map of a flip-flop is also a flip-flop.
The separator has a constant state transformation.

The interesting case is when f is reversible.

Step 2. finish the job

We decompose every Mealy machine into primes, using induction on:

1. the number of states
2. the size of the input alphabet

The parameters are ordered lexicographically, which means that the induction advances if we decrease the state space but increase the input alphabet

In the induction basis of one state, the machine is flip-flop. also, reversible.
Consider the induction step.

Either the machine is already reversible, or there is some input letter $a \in \Sigma$ such that the corresponding state transformation $Q \longrightarrow Q$ has an image that is a proper subset of Q.

Intuition: words that end with a fall under item 1 of the induction assumption, and words that do not contain a fall under item 2.

Bonus. We have also proved the Schützenberger Theorem.

A Mealy machine is called *counter-free* if it does not have a counter:

a counter is an input letter that creates a non-trivial cycle on states

non-trivial means that there are at least two different states in the cycle

Counter-free machines are connected to first-order logic.

Theorem. counter-free Mealy = (flip-flop)*.

Proof. If we apply the proof of the Krohn-Rhodes Theorem to a counter-free machine, then counters will not be created, and only flip-flop machines will be needed.

Mealy = (reversible + flip-flop)*

we use the name *primes* for these two kinds

Clearly primes* ⊆ Mealy.

Because primes ⊆ Mealy and Mealy* = Mealy.

The interesting part is Mealy ⊆ primes*.

Step 1. primes* is closed under map.

For a Mealy machine $f\colon \Sigma^* \longrightarrow \Gamma^*$, define its map to be:

$$w_1 \mid w_2 \mid \ldots \mid w_n \quad \mapsto \quad f(w_1) \mid f(w_2) \mid \ldots \mid f(w_n)$$

the input and output alphabets are extended with a separator |

This is clearly a Mealy machine. But also:

Lemma. If a function is in primes*, then so is its map.

Proof. It is enough to prove that if a function is in primes, then its map is in primes*. because the map of fg is (map f)(map g)

The map of a flip-flop is also a flip-flop.
The separator has a constant state transformation.

The interesting case is when f is reversible.

Step 2. finish the job

We decompose every Mealy machine into primes, using induction on:

1. the number of states
2. the size of the input alphabet

The parameters are ordered lexicographically, which means that the induction advances if we decrease the state space but increase the input alphabet

In the induction basis of one state, the machine is flip-flop. also, reversible.
Consider the induction step.

Either the machine is already reversible, or there is some input letter $a \in \Sigma$ such that the corresponding state transformation $Q \longrightarrow Q$ has an image that is a proper subset of Q.

Intuition: words that end with a fall under item 1 of the induction assumption, and words that do not contain a fall under item 2.

Bonus. We have also proved the Schützenberger Theorem.

A Mealy machine is called *counter-free* if it does not have a counter:

a counter is an input letter that creates a non-trivial cycle on states

non-trivial means that there are at least two different states in the cycle

Counter-free machines are connected to first-order logic.

Theorem. counter-free Mealy = (flip-flop)*.

Proof. If we apply the proof of the Krohn-Rhodes Theorem to a counter-free machine, then counters will not be created, and only flip-flop machines will be needed.

Mealy = (reversible + flip-flop)*

we use the name *primes* for these two kinds

Clearly primes* ⊆ Mealy.

Because primes ⊆ Mealy and Mealy* = Mealy.

The interesting part is Mealy ⊆ primes*.

Step 1. primes* is closed under map.

For a Mealy machine $f\colon \Sigma^* \longrightarrow \Gamma^*$, define its map to be:

$$w_1 \mid w_2 \mid \ldots \mid w_n \quad \mapsto \quad f(w_1) \mid f(w_2) \mid \ldots \mid f(w_n)$$

the input and output alphabets are extended with a separator |

This is clearly a Mealy machine. But also:

Lemma. If a function is in primes*, then so is its map.

Proof. It is enough to prove that if a function is in primes, then its map is in primes*. because the map of fg is (map f)(map g)

The map of a flip-flop is also a flip-flop.
The separator has a constant state transformation.

The interesting case is when f is reversible.

Step 2. finish the job

We decompose every Mealy machine into primes, using induction on:

1. the number of states
2. the size of the input alphabet

The parameters are ordered lexicographically, which means that the induction advances if we decrease the state space but increase the input alphabet

In the induction basis of one state, the machine is flip-flop. also, reversible.
Consider the induction step.

Either the machine is already reversible, or there is some input letter $a \in \Sigma$ such that the corresponding state transformation $Q \longrightarrow Q$ has an image that is a proper subset of Q.

Intuition: words that end with a fall under item 1 of the induction assumption, and words that do not contain a fall under item 2.

Bonus. We have also proved the Schützenberger Theorem.

A Mealy machine is called *counter-free* if it does not have a counter:

a counter is an input letter that creates a non-trivial cycle on states

non-trivial means that there are at least two different states in the cycle

Counter-free machines are connected to first-order logic.

Theorem. counter-free Mealy = (flip-flop)*.

Proof. If we apply the proof of the Krohn-Rhodes Theorem to a counter-free machine, then counters will not be created, and only flip-flop machines will be needed.

Mealy = (reversible + flip-flop)*

we use the name *primes* for these two kinds

Clearly primes* ⊆ Mealy.

Because primes ⊆ Mealy and Mealy* = Mealy.

The interesting part is Mealy ⊆ primes*.

Step 1. primes* is closed under map.

For a Mealy machine $f\colon \Sigma^* \longrightarrow \Gamma^*$, define its map to be:

$$w_1 \mid w_2 \mid \ldots \mid w_n \quad \mapsto \quad f(w_1) \mid f(w_2) \mid \ldots \mid f(w_n)$$

the input and output alphabets are extended with a separator |

This is clearly a Mealy machine. But also:

Lemma. If a function is in primes*, then so is its map.

Proof. It is enough to prove that if a function is in primes, then its map is in primes*. because the map of fg is (map f)(map g)

The map of a flip-flop is also a flip-flop.
The separator has a constant state transformation.

The interesting case is when f is reversible.

Step 2. finish the job

We decompose every Mealy machine into primes, using induction on:

1. the number of states
2. the size of the input alphabet

The parameters are ordered lexicographically, which means that the induction advances if we decrease the state space but increase the input alphabet

In the induction basis of one state, the machine is flip-flop. also, reversible.
Consider the induction step.

Either the machine is already reversible, or there is some input letter $a \in \Sigma$ such that the corresponding state transformation $Q \longrightarrow Q$ has an image that is a proper subset of Q.

Intuition: words that end with a fall under item 1 of the induction assumption, and words that do not contain a fall under item 2.

Bonus. We have also proved the Schützenberger Theorem.

A Mealy machine is called *counter-free* if it does not have a counter:

a counter is an input letter that creates a non-trivial cycle on states

non-trivial means that there are at least two different states in the cycle

Counter-free machines are connected to first-order logic.

Theorem. counter-free Mealy = (flip-flop)*.

Proof. If we apply the proof of the Krohn-Rhodes Theorem to a counter-free machine, then counters will not be created, and only flip-flop machines will be needed.

Mealy = (reversible + flip-flop)*

we use the name *primes* for these two kinds

Clearly primes* ⊆ Mealy.

Because primes ⊆ Mealy and Mealy* = Mealy.

The interesting part is Mealy ⊆ primes*.

Step 1. primes* is closed under map.

For a Mealy machine $f\colon \Sigma^* \longrightarrow \Gamma^*$, define its map to be:

$$w_1 \mid w_2 \mid \ldots \mid w_n \quad \mapsto \quad f(w_1) \mid f(w_2) \mid \ldots \mid f(w_n)$$

the input and output alphabets are extended with a separator |

This is clearly a Mealy machine. But also:

Lemma. If a function is in primes*, then so is its map.

Proof. It is enough to prove that if a function is in primes, then its map is in primes*. because the map of fg is (map f)(map g)

The map of a flip-flop is also a flip-flop.
The separator has a constant state transformation.

The interesting case is when f is reversible.

Step 2. finish the job

We decompose every Mealy machine into primes, using induction on:

1. the number of states
2. the size of the input alphabet

The parameters are ordered lexicographically, which means that the induction advances if we decrease the state space but increase the input alphabet

In the induction basis of one state, the machine is flip-flop. also, reversible.
Consider the induction step.

Either the machine is already reversible, or there is some input letter $a \in \Sigma$ such that the corresponding state transformation $Q \longrightarrow Q$ has an image that is a proper subset of Q.

Intuition: words that end with a fall under item 1 of the induction assumption, and words that do not contain a fall under item 2.

Bonus. We have also proved the Schützenberger Theorem.

A Mealy machine is called *counter-free* if it does not have a counter:

a counter is an input letter that creates a non-trivial cycle on states

non-trivial means that there are at least two different states in the cycle

Counter-free machines are connected to first-order logic.

Theorem. counter-free Mealy = (flip-flop)*.

Proof. If we apply the proof of the Krohn-Rhodes Theorem to a counter-free machine, then counters will not be created, and only flip-flop machines will be needed.

Mealy = (reversible + flip-flop)*

we use the name *primes* for these two kinds

Clearly primes* ⊆ Mealy.

Because primes ⊆ Mealy and Mealy* = Mealy.

The interesting part is Mealy ⊆ primes*.

Step 1. primes* is closed under map.

For a Mealy machine $f\colon \Sigma^* \longrightarrow \Gamma^*$, define its map to be:

$$w_1 \mid w_2 \mid \ldots \mid w_n \quad \mapsto \quad f(w_1) \mid f(w_2) \mid \ldots \mid f(w_n)$$

the input and output alphabets are extended with a separator |

This is clearly a Mealy machine. But also:

Lemma. If a function is in primes*, then so is its map.

Proof. It is enough to prove that if a function is in primes, then its map is in primes*. because the map of fg is (map f)(map g)

The map of a flip-flop is also a flip-flop.
The separator has a constant state transformation.

The interesting case is when f is reversible.

Step 2. finish the job

We decompose every Mealy machine into primes, using induction on:

1. the number of states
2. the size of the input alphabet

The parameters are ordered lexicographically, which means that the induction advances if we decrease the state space but increase the input alphabet

In the induction basis of one state, the machine is flip-flop. also, reversible.
Consider the induction step.

Either the machine is already reversible, or there is some input letter $a \in \Sigma$ such that the corresponding state transformation $Q \longrightarrow Q$ has an image that is a proper subset of Q.

Intuition: words that end with a fall under item 1 of the induction assumption, and words that do not contain a fall under item 2.

Bonus. We have also proved the Schützenberger Theorem.

A Mealy machine is called *counter-free* if it does not have a counter:

a counter is an input letter that creates a non-trivial cycle on states

non-trivial means that there are at least two different states in the cycle

Counter-free machines are connected to first-order logic.

Theorem. counter-free Mealy = (flip-flop)*.

Proof. If we apply the proof of the Krohn-Rhodes Theorem to a counter-free machine, then counters will not be created, and only flip-flop machines will be needed.

For a Mealy machine $f\colon \Sigma^* \longrightarrow \Gamma^*$, define its map to be:

$$w_1 \mid w_2 \mid \ldots \mid w_n \quad \mapsto \quad f(w_1) \mid f(w_2) \mid \ldots \mid f(w_n)$$

the input and output alphabets are extended with a separator |

This is clearly a Mealy machine. But also:

Lemma. If a function is in primes*, then so is its map.

Proof. It is enough to prove that if a function is in primes, then its map is in primes*. because the map of f;g is (map f);(map g)

The map of a flip-flop is also a flip-flop.

The separator has a constant state transformation.

The interesting case is when f is reversible.

For a Mealy machine $f\colon \Sigma^* \longrightarrow \Gamma^*$, define its map to be:

$$w_1 \mid w_2 \mid \ldots \mid w_n \quad \mapsto \quad f(w_1) \mid f(w_2) \mid \ldots \mid f(w_n)$$

the input and output alphabets are extended with a separator |

This is clearly a Mealy machine. But also:

Lemma. If a function is in primes*, then so is its map.

Proof. It is enough to prove that if a function is in primes, then its map is in primes*. because the map of f;g is (map f);(map g)

The map of a flip-flop is also a flip-flop.

The separator has a constant state transformation.

The interesting case is when f is reversible.

For a Mealy machine $f\colon \Sigma^* \longrightarrow \Gamma^*$, define its map to be:

$$w_1 \mid w_2 \mid \ldots \mid w_n \quad \mapsto \quad f(w_1) \mid f(w_2) \mid \ldots \mid f(w_n)$$

the input and output alphabets are extended with a separator |

This is clearly a Mealy machine. But also:

Lemma. If a function is in primes*, then so is its map.

Proof. It is enough to prove that if a function is in primes, then its map is in primes*. because the map of f;g is (map f);(map g)

The map of a flip-flop is also a flip-flop.

The separator has a constant state transformation.

The interesting case is when f is reversible.

For a Mealy machine $f\colon \Sigma^* \longrightarrow \Gamma^*$, define its map to be:

$$w_1 \mid w_2 \mid \ldots \mid w_n \quad \mapsto \quad f(w_1) \mid f(w_2) \mid \ldots \mid f(w_n)$$

the input and output alphabets are extended with a separator |

This is clearly a Mealy machine. But also:

Lemma. If a function is in primes*, then so is its map.

Proof. It is enough to prove that if a function is in primes, then its map is in primes*. because the map of f;g is (map f);(map g)

The map of a flip-flop is also a flip-flop.

The separator has a constant state transformation.

The interesting case is when f is reversible.

For a Mealy machine $f\colon \Sigma^* \longrightarrow \Gamma^*$, define its map to be:

$$w_1 \mid w_2 \mid \ldots \mid w_n \quad \mapsto \quad f(w_1) \mid f(w_2) \mid \ldots \mid f(w_n)$$

the input and output alphabets are extended with a separator |

This is clearly a Mealy machine. But also:

Lemma. If a function is in primes*, then so is its map.

Proof. It is enough to prove that if a function is in primes, then its map is in primes*. because the map of f;g is (map f);(map g)

The map of a flip-flop is also a flip-flop.

The separator has a constant state transformation.

The interesting case is when f is reversible.

For a Mealy machine $f\colon \Sigma^* \longrightarrow \Gamma^*$, define its map to be:

$$w_1 \mid w_2 \mid \ldots \mid w_n \quad \mapsto \quad f(w_1) \mid f(w_2) \mid \ldots \mid f(w_n)$$

the input and output alphabets are extended with a separator |

This is clearly a Mealy machine. But also:

Lemma. If a function is in primes*, then so is its map.

Proof. It is enough to prove that if a function is in primes, then its map is in primes*. because the map of f;g is (map f);(map g)

The map of a flip-flop is also a flip-flop.

The separator has a constant state transformation.

The interesting case is when f is reversible.

For a Mealy machine $f\colon \Sigma^* \longrightarrow \Gamma^*$, define its map to be:

$$w_1 \mid w_2 \mid \ldots \mid w_n \quad \mapsto \quad f(w_1) \mid f(w_2) \mid \ldots \mid f(w_n)$$

the input and output alphabets are extended with a separator |

This is clearly a Mealy machine. But also:

Lemma. If a function is in primes*, then so is its map.

Proof. It is enough to prove that if a function is in primes, then its map is in primes*. because the map of f;g is (map f);(map g)

The map of a flip-flop is also a flip-flop.

The separator has a constant state transformation.

The interesting case is when f is reversible.

For a Mealy machine $f\colon \Sigma^* \longrightarrow \Gamma^*$, define its map to be:

$$w_1 \mid w_2 \mid \ldots \mid w_n \quad \mapsto \quad f(w_1) \mid f(w_2) \mid \ldots \mid f(w_n)$$

the input and output alphabets are extended with a separator |

This is clearly a Mealy machine. But also:

Lemma. If a function is in primes*, then so is its map.

Proof. It is enough to prove that if a function is in primes, then its map is in primes*. because the map of f;g is (map f);(map g)

The map of a flip-flop is also a flip-flop.

The separator has a constant state transformation.

The interesting case is when f is reversible.

The interesting case is when f is reversible.

The interesting case is when f is reversible.

Suppose that f is reversible.

The interesting case is when f is reversible.

Suppose that f is reversible. We want to show that its map is in primes*.

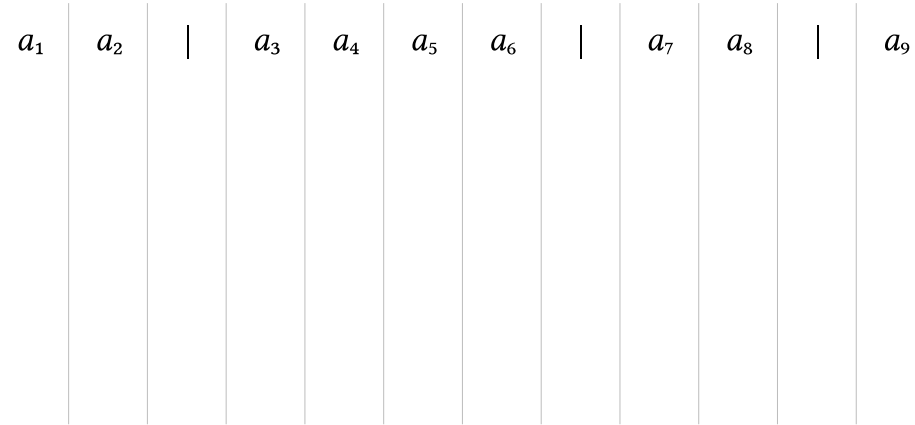
The interesting case is when f is reversible.

Suppose that f is reversible. We want to show that its map is in primes*.

$$a_1 \quad a_2 \quad | \quad a_3 \quad a_4 \quad a_5 \quad a_6 \quad | \quad a_7 \quad a_8 \quad | \quad a_9$$

The interesting case is when f is reversible.

Suppose that f is reversible. We want to show that its map is in primes*.



The interesting case is when f is reversible.

Suppose that f is reversible. We want to show that its map is in primes*.

a_1	a_2	|	a_3	a_4	a_5	a_6	|	a_7	a_8	|	a_9
δ_0	δ_1	δ_2	δ_2	δ_3	δ_4	δ_5	δ_6	δ_6	δ_7	δ_8	δ_8

The interesting case is when f is reversible.

Suppose that f is reversible. We want to show that its map is in primes*.

a_1	a_2	|	a_3	a_4	a_5	a_6	|	a_7	a_8	|	a_9
δ_0	δ_1	δ_2	δ_2	δ_3	δ_4	δ_5	δ_6	δ_6	δ_7	δ_8	δ_8

Here δ_n is the state transformation obtained by reading first n letters

The interesting case is when f is reversible.

Suppose that f is reversible. We want to show that its map is in primes*.

a_1	a_2	|	a_3	a_4	a_5	a_6	|	a_7	a_8	|	a_9
δ_0	δ_1	δ_2	δ_2	δ_3	δ_4	δ_5	δ_6	δ_6	δ_7	δ_8	δ_8

Here δ_n is the state transformation obtained by reading first n letters
By reversibility, this state transformation belongs to the group G of state permutations.

The interesting case is when f is reversible.

Suppose that f is reversible. We want to show that its map is in primes*.

a_1	a_2	|	a_3	a_4	a_5	a_6	|	a_7	a_8	|	a_9
δ_0	δ_1	δ_2	δ_2	δ_3	δ_4	δ_5	δ_6	δ_6	δ_7	δ_8	δ_8

Here δ_n is the state transformation obtained by reading first n letters
By reversibility, this state transformation belongs to the group G of state permutations.
The red word can be produced using a reversible machine with states G.

The interesting case is when f is reversible.

Suppose that f is reversible. We want to show that its map is in primes*.

a_1	a_2	|	a_3	a_4	a_5	a_6	|	a_7	a_8	|	a_9
δ_0	δ_1	δ_2	δ_2	δ_3	δ_4	δ_5	δ_6	δ_6	δ_7	δ_8	δ_8

Here δ_n is the state transformation obtained by reading first n letters
By reversibility, this state transformation belongs to the group G of state permutations.
The red word can be produced using a reversible machine with states G.

The interesting case is when f is reversible.

Suppose that f is reversible. We want to show that its map is in primes*.

a_1	a_2	|	a_3	a_4	a_5	a_6	|	a_7	a_8	|	a_9
δ_0	δ_1	δ_2	δ_2	δ_3	δ_4	δ_5	δ_6	δ_6	δ_7	δ_8	δ_8
δ_0	δ_0	δ_2	δ_2	δ_2	δ_2	δ_2	δ_6	δ_6	δ_6	δ_8	δ_8

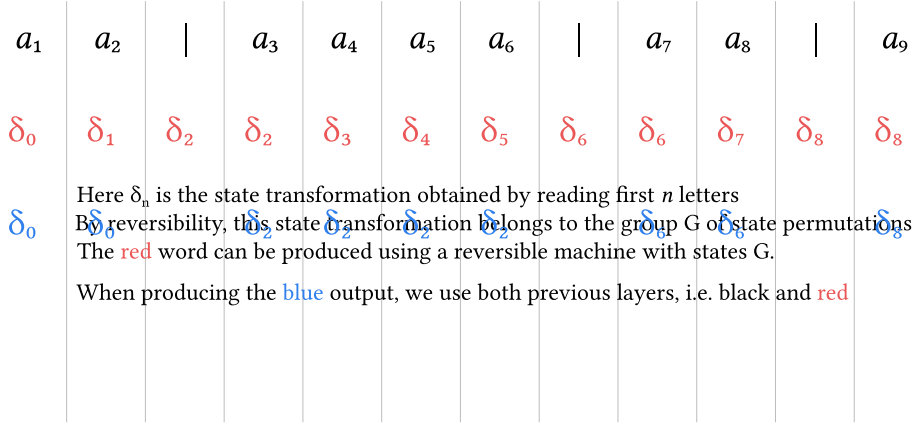
Here δ_n is the state transformation obtained by reading first n letters

By reversibility, this state transformation belongs to the group G of state permutations.

The red word can be produced using a reversible machine with states G.

The interesting case is when f is reversible.

Suppose that f is reversible. We want to show that its map is in primes*.



Here δ_n is the state transformation obtained by reading first n letters

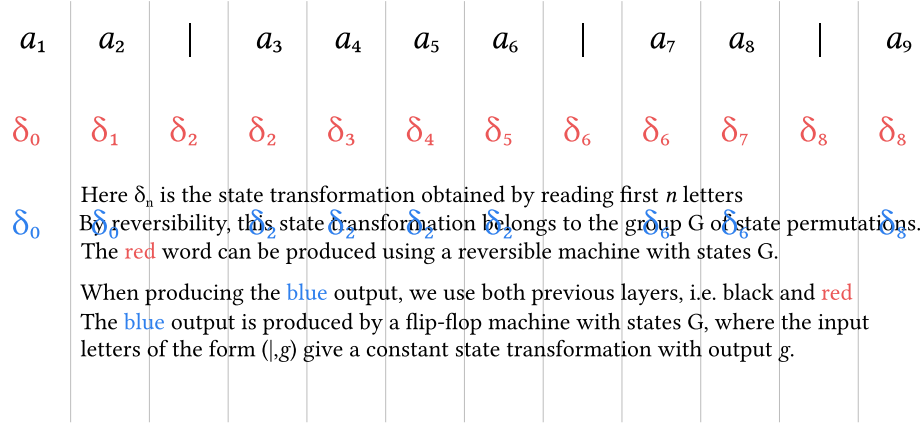
By reversibility, this state transformation belongs to the group G of state permutations.

The red word can be produced using a reversible machine with states G.

When producing the blue output, we use both previous layers, i.e. black and red

The interesting case is when f is reversible.

Suppose that f is reversible. We want to show that its map is in primes*.



Here δ_n is the state transformation obtained by reading first n letters

By reversibility, this state transformation belongs to the group G of state permutations.

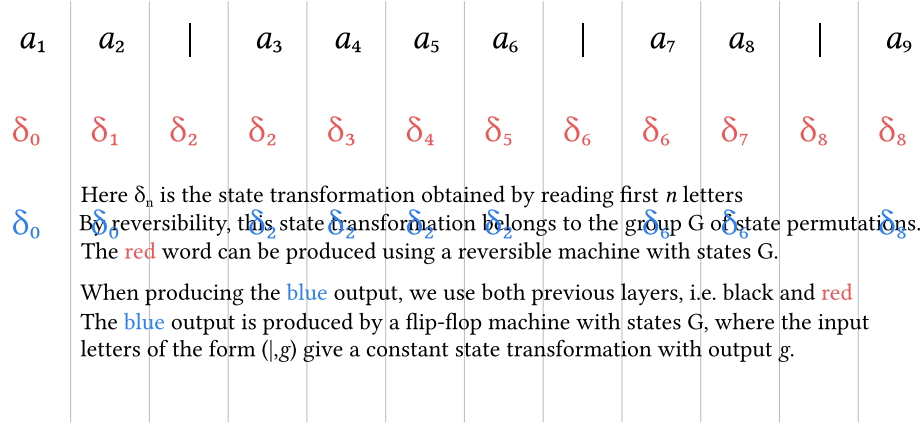
The red word can be produced using a reversible machine with states G.

When producing the blue output, we use both previous layers, i.e. black and red

The blue output is produced by a flip-flop machine with states G, where the input letters of the form (|,g) give a constant state transformation with output g.

The interesting case is when f is reversible.

Suppose that f is reversible. We want to show that its map is in primes*.



Here δ_n is the state transformation obtained by reading first n letters

By reversibility, this state transformation belongs to the group G of state permutations.

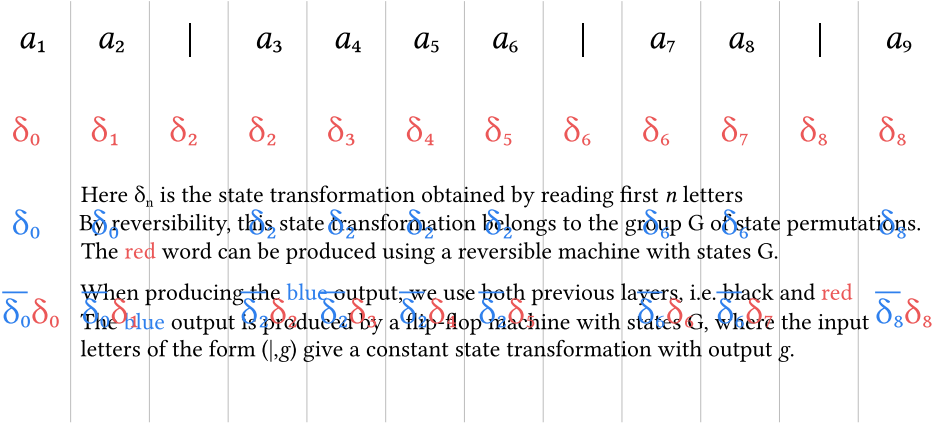
The red word can be produced using a reversible machine with states G.

When producing the blue output, we use both previous layers, i.e. black and red

The blue output is produced by a flip-flop machine with states G, where the input letters of the form (|,g) give a constant state transformation with output g.

The interesting case is when f is reversible.

Suppose that f is reversible. We want to show that its map is in primes*.



Here δ_n is the state transformation obtained by reading first n letters

By reversibility, this state transformation belongs to the group G of state permutations.

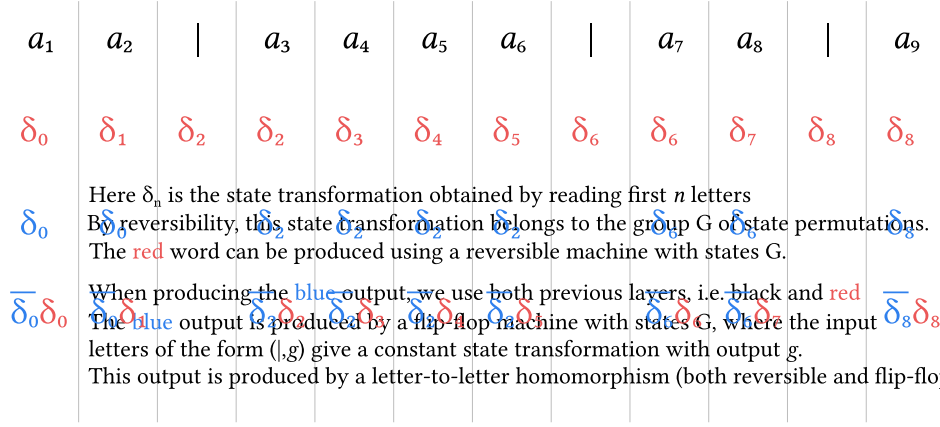
The red word can be produced using a reversible machine with states G.

When producing the blue output, we use both previous layers, i.e. black and red

The blue output is produced by a flip-flop machine with states G, where the input letters of the form $(|,g)$ give a constant state transformation with output g.

The interesting case is when f is reversible.

Suppose that f is reversible. We want to show that its map is in primes*.



Here δ_n is the state transformation obtained by reading first n letters

By reversibility, this state transformation belongs to the group G of state permutations.

The red word can be produced using a reversible machine with states G.

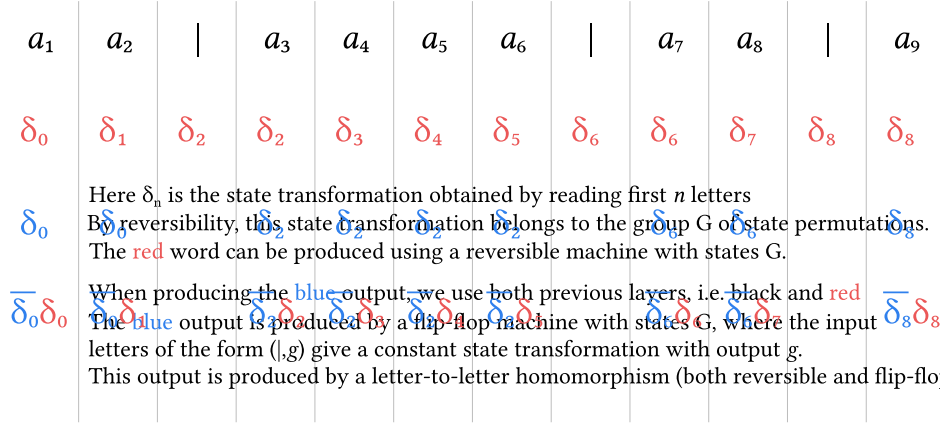
When producing the blue output, we use both previous layers, i.e. black and red

The blue output is produced by a flip-flop machine with states G, where the input letters of the form (|,g) give a constant state transformation with output g.

This output is produced by a letter-to-letter homomorphism (both reversible and flip-flop)

The interesting case is when f is reversible.

Suppose that f is reversible. We want to show that its map is in primes*.



Here δ_n is the state transformation obtained by reading first n letters

By reversibility, this state transformation belongs to the group G of state permutations.

The red word can be produced using a reversible machine with states G.

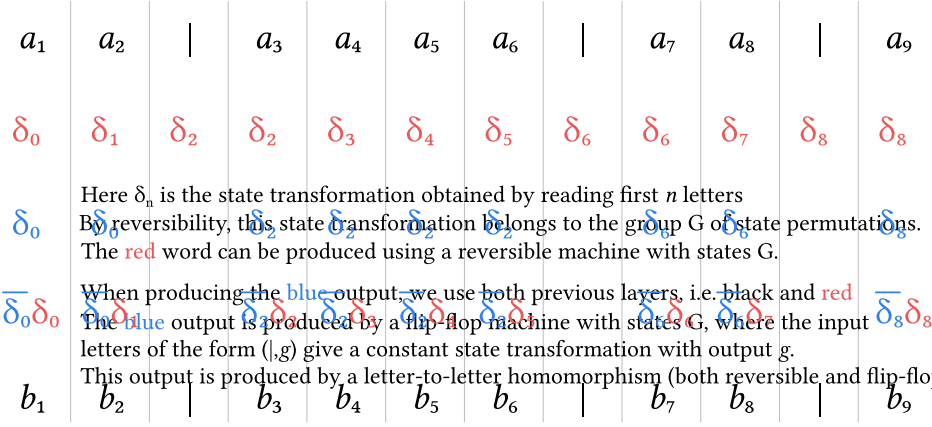
When producing the blue output, we use both previous layers, i.e. black and red

The blue output is produced by a flip-flop machine with states G, where the input letters of the form (|,g) give a constant state transformation with output g.

This output is produced by a letter-to-letter homomorphism (both reversible and flip-flop)

The interesting case is when f is reversible.

Suppose that f is reversible. We want to show that its map is in primes*.



Here δ_n is the state transformation obtained by reading first n letters

By reversibility, this state transformation belongs to the group G of state permutations.

The red word can be produced using a reversible machine with states G.

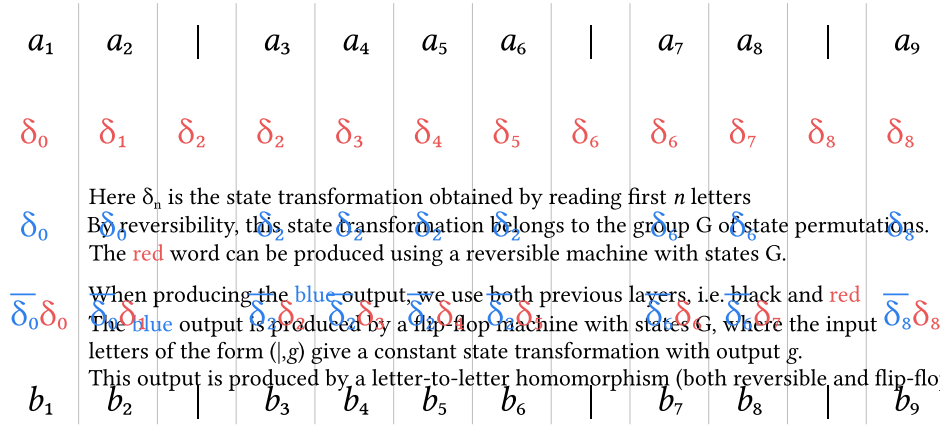
When producing the blue output, we use both previous layers, i.e. black and red

The blue output is produced by a flip-flop machine with states G, where the input letters of the form (|,g) give a constant state transformation with output g.

This output is produced by a letter-to-letter homomorphism (both reversible and flip-flop)

The interesting case is when f is reversible.

Suppose that f is reversible. We want to show that its map is in primes*.



Here δ_n is the state transformation obtained by reading first n letters

By reversibility, this state transformation belongs to the group G of state permutations.

The red word can be produced using a reversible machine with states G.

When producing the blue output, we use both previous layers, i.e. black and red

The blue output is produced by a flip-flop machine with states G, where the input letters of the form (|,g) give a constant state transformation with output g.

This output is produced by a letter-to-letter homomorphism (both reversible and flip-flop)

The final output can be produced using a letter-to-letter homomorphism from the red line and blue-red line. (Or directly from red red and blue lines.)

For a Mealy machine $f: \Sigma^* \longrightarrow \Gamma^*$, define its map to be:

$$w_1 \mid w_2 \mid \ldots \mid w_n \quad \mapsto \quad f(w_1) \mid f(w_2) \mid \ldots \mid f(w_n)$$

the input and output alphabets are extended with a separator |

This is clearly a Mealy machine. But also:

Lemma. If a function is in primes*, then so is its map.

Proof. It is enough to prove that if a function is in primes, then its map is in primes*. because the map of f;g is (map f);(map g)

The map of a flip-flop is also a flip-flop.

The separator has a constant state transformation.

The interesting case is when f is reversible.

Suppose that f is reversible. We want to show that its map is in primes*.

Mealy = (reversible + flip-flop)*

we use the name *primes* for these two kinds

Clearly primes* ⊆ Mealy.

Because primes ⊆ Mealy and Mealy* = Mealy.

The interesting part is Mealy ⊆ primes*.

Step 1. primes* is closed under map.

For a Mealy machine $f:\Sigma^*\longrightarrow\Gamma^*$, define its map to be:

$$w_1 \mid w_2 \mid \ldots \mid w_n \quad\mapsto\quad f(w_1)\mid f(w_2)\mid \ldots \mid f(w_n)$$

the input and output alphabets are extended with a separator |

This is clearly a Mealy machine. But also:

Lemma. If a function is in primes*, then so is its map.

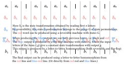
Proof. It is enough to prove that if a function is in primes, then its map is in primes*. because the map of f;g is (map f);(map g)

The map of a flip-flop is also a flip-flop.

The separator has a constant state transformation.

The interesting case is when f is reversible.

Suppose that f is reversible. We want to show that its map is in primes*.



Step 2. finish the job

We decompose every Mealy machine into primes, using induction on:

1. the number of states
2. the size of the input alphabet

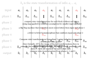
The parameters are ordered lexicographically, which means that the induction advances if we decrease the state space but increase the input alphabet

In the induction basis of one state, the machine is flip-flop. also, reversible.

Consider the induction step.

Either the machine is already reversible, or there is some input letter $a\in\Sigma$ such that the corresponding state transformation $Q\longrightarrow Q$ has an image that is a proper subset of Q.

Intuition: words that end with a fall under item 1 of the induction assumption, and words that do not contain a fall under item 2.



Bonus. We have also proved the Schützenberger Theorem.

A Mealy machine is called *counter-free* if it does not have a counter:

a counter is an input letter that creates a non-trivial cycle on states

non-trivial means that there are at least two different states in the cycle

Counter-free machines are connected to first-order logic.

Theorem. counter-free Mealy = (flip-flop)*.

Proof. If we apply the proof of the Krohn-Rhodes Theorem to a counter-free machine, then counters will not be created, and only flip-flop machines will be needed.

We decompose every Mealy machine into primes, using induction on:

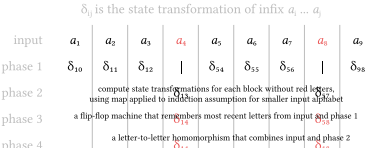
1. the number of states
2. the size of the input alphabet

The parameters are ordered lexicographically, which means that the induction advances if we decrease the state space but increase the input alphabet

In the induction basis of one state, the machine is flip-flop. also, reversible.
Consider the induction step.

Either the machine is already reversible, or there is some input letter $a \in \Sigma$ such that the corresponding state transformation $Q \longrightarrow Q$ has an image that is a proper subset of Q.

Intuition: words that end with a fall under item 1 of the induction assumption, and words that do not contain a fall under item 2.



We decompose every Mealy machine into primes, using induction on:

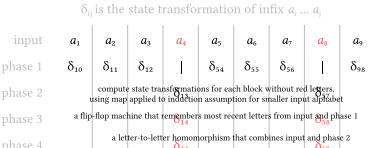
1. the number of states
2. the size of the input alphabet

The parameters are ordered lexicographically, which means that the induction advances if we decrease the state space but increase the input alphabet

In the induction basis of one state, the machine is flip-flop. also, reversible.
Consider the induction step.

Either the machine is already reversible, or there is some input letter $a \in \Sigma$ such that the corresponding state transformation $Q \longrightarrow Q$ has an image that is a proper subset of Q.

Intuition: words that end with a fall under item 1 of the induction assumption, and words that do not contain a fall under item 2.



We decompose every Mealy machine into primes, using induction on:

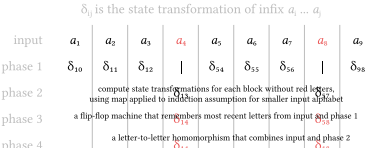
1. the number of states
2. the size of the input alphabet

The parameters are ordered lexicographically, which means that the induction advances if we decrease the state space but increase the input alphabet

In the induction basis of one state, the machine is flip-flop. also, reversible.
Consider the induction step.

Either the machine is already reversible, or there is some input letter $a \in \Sigma$ such that the corresponding state transformation $Q \longrightarrow Q$ has an image that is a proper subset of Q.

Intuition: words that end with a fall under item 1 of the induction assumption, and words that do not contain a fall under item 2.



We decompose every Mealy machine into primes, using induction on:

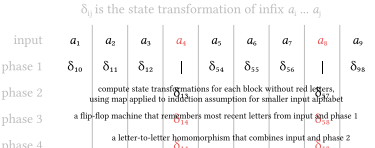
1. the number of states
2. the size of the input alphabet

The parameters are ordered lexicographically, which means that the induction advances if we decrease the state space but increase the input alphabet

In the induction basis of one state, the machine is flip-flop. also, reversible.
Consider the induction step.

Either the machine is already reversible, or there is some input letter $a \in \Sigma$ such that the corresponding state transformation $Q \longrightarrow Q$ has an image that is a proper subset of Q.

Intuition: words that end with a fall under item 1 of the induction assumption, and words that do not contain a fall under item 2.



We decompose every Mealy machine into primes, using induction on:

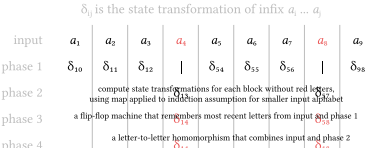
1. the number of states
2. the size of the input alphabet

The parameters are ordered lexicographically, which means that the induction advances if we decrease the state space but increase the input alphabet

In the induction basis of one state, the machine is flip-flop. also, reversible.
Consider the induction step.

Either the machine is already reversible, or there is some input letter $a \in \Sigma$ such that the corresponding state transformation $Q \longrightarrow Q$ has an image that is a proper subset of Q.

Intuition: words that end with a fall under item 1 of the induction assumption, and words that do not contain a fall under item 2.



We decompose every Mealy machine into primes, using induction on:

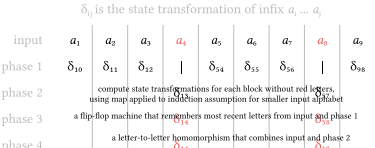
1. the number of states
2. the size of the input alphabet

The parameters are ordered lexicographically, which means that the induction advances if we decrease the state space but increase the input alphabet

In the induction basis of one state, the machine is flip-flop. also, reversible.
Consider the induction step.

Either the machine is already reversible, or there is some input letter $a \in \Sigma$ such that the corresponding state transformation $Q \longrightarrow Q$ has an image that is a proper subset of Q.

Intuition: words that end with a fall under item 1 of the induction assumption, and words that do not contain a fall under item 2.



We decompose every Mealy machine into primes, using induction on:

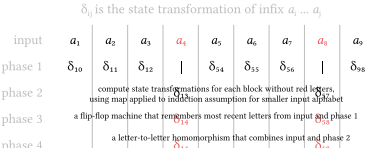
1. the number of states
2. the size of the input alphabet

The parameters are ordered lexicographically, which means that the induction advances if we decrease the state space but increase the input alphabet

In the induction basis of one state, the machine is flip-flop. also, reversible.
Consider the induction step.

Either the machine is already reversible, or there is some input letter $a \in \Sigma$ such that the corresponding state transformation $Q \longrightarrow Q$ has an image that is a proper subset of Q.

Intuition: words that end with a fall under item 1 of the induction assumption, and words that do not contain a fall under item 2.



We decompose every Mealy machine into primes, using induction on:

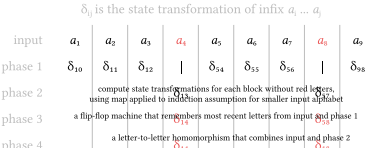
1. the number of states
2. the size of the input alphabet

The parameters are ordered lexicographically, which means that the induction advances if we decrease the state space but increase the input alphabet

In the induction basis of one state, the machine is flip-flop. also, reversible.
Consider the induction step.

Either the machine is already reversible, or there is some input letter $a \in \Sigma$ such that the corresponding state transformation $Q \longrightarrow Q$ has an image that is a proper subset of Q.

Intuition: words that end with a fall under item 1 of the induction assumption, and words that do not contain a fall under item 2.



We decompose every Mealy machine into primes, using induction on:

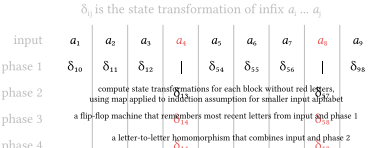
1. the number of states
2. the size of the input alphabet

The parameters are ordered lexicographically, which means that the induction advances if we decrease the state space but increase the input alphabet

In the induction basis of one state, the machine is flip-flop. also, reversible.
Consider the induction step.

Either the machine is already reversible, or there is some input letter $a \in \Sigma$ such that the corresponding state transformation $Q \longrightarrow Q$ has an image that is a proper subset of Q.

Intuition: words that end with a fall under item 1 of the induction assumption, and words that do not contain a fall under item 2.



We decompose every Mealy machine into primes, using induction on:

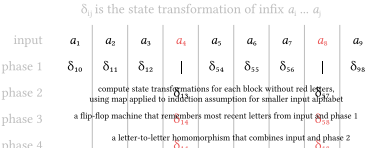
1. the number of states
2. the size of the input alphabet

The parameters are ordered lexicographically, which means that the induction advances if we decrease the state space but increase the input alphabet

In the induction basis of one state, the machine is flip-flop. also, reversible.
Consider the induction step.

Either the machine is already reversible, or there is some input letter $a \in \Sigma$ such that the corresponding state transformation $Q \longrightarrow Q$ has an image that is a proper subset of Q.

Intuition: words that end with a fall under item 1 of the induction assumption, and words that do not contain a fall under item 2.



δ_{ij} is the state transformation of infix $a_i \dots a_j$

input	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9
phase 1	δ_{10}	δ_{11}	δ_{12}	|	δ_{54}	δ_{55}	δ_{56}	|	δ_{98}
phase 2				δ_{13}				δ_{57}	
phase 3				δ_{14}				δ_{58}	
phase 4				δ_{14}				δ_{18}	
phase 5				δ_{14}	δ_{14}	δ_{14}	δ_{14}	δ_{18}	δ_{18}
phase 6	δ_{10}	δ_{11}	δ_{12}	δ_{13}	δ_{14}	δ_{15}	δ_{16}	δ_{17}	δ_{18}
output	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9

compute state transformations for each block without red letters, using map applied to induction assumption for smaller input alphabet

a flip-flop machine that remembers most recent letters from input and phase 1

a letter-to-letter homomorphism that combines input and phase 2

Compose state transformations from phase 3, using induction assumption on a smaller state space.

More formally, if P is the image of the red transformations, then to compose the red state transformations, we use a Mealy machine with states P, with one machine for each initial state.

copy most recent letter from phase 4

compose state transformations from phases 1, 2 and 5

δ_{ij} is the state transformation of infix $a_i \dots a_j$

input	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9
phase 1	δ_{10}	δ_{11}	δ_{12}	|	δ_{54}	δ_{55}	δ_{56}	|	δ_{98}
phase 2				δ_{13}				δ_{57}	
phase 3				δ_{14}				δ_{58}	
phase 4				δ_{14}				δ_{18}	
phase 5				δ_{14}	δ_{14}	δ_{14}	δ_{14}	δ_{18}	δ_{18}
phase 6	δ_{10}	δ_{11}	δ_{12}	δ_{13}	δ_{14}	δ_{15}	δ_{16}	δ_{17}	δ_{18}
output	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9

compute state transformations for each block without red letters, using map applied to induction assumption for smaller input alphabet

a flip-flop machine that remembers most recent letters from input and phase 1

a letter-to-letter homomorphism that combines input and phase 2

Compose state transformations from phase 3, using induction assumption on a smaller state space.

More formally, if P is the image of the red transformations, then to compose the red state transformations, we use a Mealy machine with states P, with one machine for each initial state.

copy most recent letter from phase 4

compose state transformations from phases 1, 2 and 5

δ_{ij} is the state transformation of infix $a_i \dots a_j$

input	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9
phase 1	δ_{10}	δ_{11}	δ_{12}	|	δ_{54}	δ_{55}	δ_{56}	|	δ_{98}
phase 2				δ_{13}				δ_{57}	
phase 3				δ_{14}				δ_{58}	
phase 4				δ_{14}				δ_{18}	
phase 5				δ_{14}	δ_{14}	δ_{14}	δ_{14}	δ_{18}	δ_{18}
phase 6	δ_{10}	δ_{11}	δ_{12}	δ_{13}	δ_{14}	δ_{15}	δ_{16}	δ_{17}	δ_{18}
output	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9

compute state transformations for each block without red letters, using map applied to induction assumption for smaller input alphabet

a flip-flop machine that remembers most recent letters from input and phase 1

a letter-to-letter homomorphism that combines input and phase 2

Compose state transformations from phase 3, using induction assumption on a smaller state space.

More formally, if P is the image of the red transformations, then to compose the red state transformations, we use a Mealy machine with states P, with one machine for each initial state.

copy most recent letter from phase 4

compose state transformations from phases 1, 2 and 5

δ_{ij} is the state transformation of infix $a_i \dots a_j$

input	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9
phase 1	δ_{10}	δ_{11}	δ_{12}	|	δ_{54}	δ_{55}	δ_{56}	|	δ_{98}
phase 2				δ_{13}				δ_{57}	
phase 3				δ_{14}				δ_{58}	
phase 4				δ_{14}				δ_{18}	
phase 5				δ_{14}	δ_{14}	δ_{14}	δ_{14}	δ_{18}	δ_{18}
phase 6	δ_{10}	δ_{11}	δ_{12}	δ_{13}	δ_{14}	δ_{15}	δ_{16}	δ_{17}	δ_{18}
output	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9

compute state transformations for each block without red letters, using map applied to induction assumption for smaller input alphabet

a flip-flop machine that remembers most recent letters from input and phase 1

a letter-to-letter homomorphism that combines input and phase 2

Compose state transformations from phase 3, using induction assumption on a smaller state space.

More formally, if P is the image of the red transformations, then to compose the red state transformations, we use a Mealy machine with states P, with one machine for each initial state.

copy most recent letter from phase 4

compose state transformations from phases 1, 2 and 5

δ_{ij} is the state transformation of infix $a_i \dots a_j$

input	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9
phase 1	δ_{10}	δ_{11}	δ_{12}	|	δ_{54}	δ_{55}	δ_{56}	|	δ_{98}
phase 2				δ_{13}				δ_{57}	
phase 3				δ_{14}				δ_{58}	
phase 4				δ_{14}				δ_{18}	
phase 5				δ_{14}	δ_{14}	δ_{14}	δ_{14}	δ_{18}	δ_{18}
phase 6	δ_{10}	δ_{11}	δ_{12}	δ_{13}	δ_{14}	δ_{15}	δ_{16}	δ_{17}	δ_{18}
output	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9

compute state transformations for each block without red letters, using map applied to induction assumption for smaller input alphabet

a flip-flop machine that remembers most recent letters from input and phase 1

a letter-to-letter homomorphism that combines input and phase 2

Compose state transformations from phase 3, using induction assumption on a smaller state space.

More formally, if P is the image of the red transformations, then to compose the red state transformations, we use a Mealy machine with states P, with one machine for each initial state.

copy most recent letter from phase 4

compose state transformations from phases 1, 2 and 5

δ_{ij} is the state transformation of infix $a_i \dots a_j$

input	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9
phase 1	δ_{10}	δ_{11}	δ_{12}	|	δ_{54}	δ_{55}	δ_{56}	|	δ_{98}
phase 2				δ_{13}				δ_{57}	
phase 3				δ_{14}				δ_{58}	
phase 4				δ_{14}				δ_{18}	
phase 5				δ_{14}	δ_{14}	δ_{14}	δ_{14}	δ_{18}	δ_{18}
phase 6	δ_{10}	δ_{11}	δ_{12}	δ_{13}	δ_{14}	δ_{15}	δ_{16}	δ_{17}	δ_{18}
output	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9

compute state transformations for each block without red letters, using map applied to induction assumption for smaller input alphabet

a flip-flop machine that remembers most recent letters from input and phase 1

a letter-to-letter homomorphism that combines input and phase 2

Compose state transformations from phase 3, using induction assumption on a smaller state space.

More formally, if P is the image of the red transformations, then to compose the red state transformations, we use a Mealy machine with states P, with one machine for each initial state.

copy most recent letter from phase 4

compose state transformations from phases 1, 2 and 5

δ_{ij} is the state transformation of infix $a_i \dots a_j$

input	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9
phase 1	δ_{10}	δ_{11}	δ_{12}	|	δ_{54}	δ_{55}	δ_{56}	|	δ_{98}
phase 2				δ_{13}				δ_{57}	
phase 3				δ_{14}				δ_{58}	
phase 4				δ_{14}				δ_{18}	
phase 5				δ_{14}	δ_{14}	δ_{14}	δ_{14}	δ_{18}	δ_{18}
phase 6	δ_{10}	δ_{11}	δ_{12}	δ_{13}	δ_{14}	δ_{15}	δ_{16}	δ_{17}	δ_{18}
output	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9

compute state transformations for each block without red letters, using map applied to induction assumption for smaller input alphabet

a flip-flop machine that remembers most recent letters from input and phase 1

a letter-to-letter homomorphism that combines input and phase 2

Compose state transformations from phase 3, using induction assumption on a smaller state space.

More formally, if P is the image of the red transformations, then to compose the red state transformations, we use a Mealy machine with states P, with one machine for each initial state.

copy most recent letter from phase 4

compose state transformations from phases 1, 2 and 5

δ_{ij} is the state transformation of infix $a_i \dots a_j$

input	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9
phase 1	δ_{10}	δ_{11}	δ_{12}	|	δ_{54}	δ_{55}	δ_{56}	|	δ_{98}
phase 2				δ_{13}				δ_{57}	
phase 3				δ_{14}				δ_{58}	
phase 4				δ_{14}				δ_{18}	
phase 5				δ_{14}	δ_{14}	δ_{14}	δ_{14}	δ_{18}	δ_{18}
phase 6	δ_{10}	δ_{11}	δ_{12}	δ_{13}	δ_{14}	δ_{15}	δ_{16}	δ_{17}	δ_{18}
output	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9

compute state transformations for each block without red letters, using map applied to induction assumption for smaller input alphabet

a flip-flop machine that remembers most recent letters from input and phase 1

a letter-to-letter homomorphism that combines input and phase 2

Compose state transformations from phase 3, using induction assumption on a smaller state space.

More formally, if P is the image of the red transformations, then to compose the red state transformations, we use a Mealy machine with states P, with one machine for each initial state.

copy most recent letter from phase 4

compose state transformations from phases 1, 2 and 5

δ_{ij} is the state transformation of infix $a_i \dots a_j$

input	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9
phase 1	δ_{10}	δ_{11}	δ_{12}	|	δ_{54}	δ_{55}	δ_{56}	|	δ_{98}
phase 2				δ_{13}				δ_{57}	
phase 3				δ_{14}				δ_{58}	
phase 4				δ_{14}				δ_{18}	
phase 5				δ_{14}	δ_{14}	δ_{14}	δ_{14}	δ_{18}	δ_{18}
phase 6	δ_{10}	δ_{11}	δ_{12}	δ_{13}	δ_{14}	δ_{15}	δ_{16}	δ_{17}	δ_{18}
output	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9

compute state transformations for each block without red letters, using map applied to induction assumption for smaller input alphabet

a flip-flop machine that remembers most recent letters from input and phase 1

a letter-to-letter homomorphism that combines input and phase 2

Compose state transformations from phase 3, using induction assumption on a smaller state space.

More formally, if P is the image of the red transformations, then to compose the red state transformations, we use a Mealy machine with states P, with one machine for each initial state.

copy most recent letter from phase 4

compose state transformations from phases 1, 2 and 5

δ_{ij} is the state transformation of infix $a_i \dots a_j$

input	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9
phase 1	δ_{10}	δ_{11}	δ_{12}	|	δ_{54}	δ_{55}	δ_{56}	|	δ_{98}
phase 2				δ_{13}				δ_{57}	
phase 3				δ_{14}				δ_{58}	
phase 4				δ_{14}				δ_{18}	
phase 5				δ_{14}	δ_{14}	δ_{14}	δ_{14}	δ_{18}	δ_{18}
phase 6	δ_{10}	δ_{11}	δ_{12}	δ_{13}	δ_{14}	δ_{15}	δ_{16}	δ_{17}	δ_{18}
output	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9

compute state transformations for each block without red letters, using map applied to induction assumption for smaller input alphabet

a flip-flop machine that remembers most recent letters from input and phase 1

a letter-to-letter homomorphism that combines input and phase 2

Compose state transformations from phase 3, using induction assumption on a smaller state space.

More formally, if P is the image of the red transformations, then to compose the red state transformations, we use a Mealy machine with states P, with one machine for each initial state.

copy most recent letter from phase 4

compose state transformations from phases 1, 2 and 5

δ_{ij} is the state transformation of infix $a_i \dots a_j$

input	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9
phase 1	δ_{10}	δ_{11}	δ_{12}	|	δ_{54}	δ_{55}	δ_{56}	|	δ_{98}
phase 2				δ_{13}				δ_{57}	
phase 3				δ_{14}				δ_{58}	
phase 4				δ_{14}				δ_{18}	
phase 5				δ_{14}	δ_{14}	δ_{14}	δ_{14}	δ_{18}	δ_{18}
phase 6	δ_{10}	δ_{11}	δ_{12}	δ_{13}	δ_{14}	δ_{15}	δ_{16}	δ_{17}	δ_{18}
output	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9

compute state transformations for each block without red letters, using map applied to induction assumption for smaller input alphabet

a flip-flop machine that remembers most recent letters from input and phase 1

a letter-to-letter homomorphism that combines input and phase 2

Compose state transformations from phase 3, using induction assumption on a smaller state space.

More formally, if P is the image of the red transformations, then to compose the red state transformations, we use a Mealy machine with states P, with one machine for each initial state.

copy most recent letter from phase 4

compose state transformations from phases 1, 2 and 5

δ_{ij} is the state transformation of infix $a_i \dots a_j$

input	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9
phase 1	δ_{10}	δ_{11}	δ_{12}	|	δ_{54}	δ_{55}	δ_{56}	|	δ_{98}
phase 2				δ_{13}				δ_{57}	
phase 3				δ_{14}				δ_{58}	
phase 4				δ_{14}				δ_{18}	
phase 5				δ_{14}	δ_{14}	δ_{14}	δ_{14}	δ_{18}	δ_{18}
phase 6	δ_{10}	δ_{11}	δ_{12}	δ_{13}	δ_{14}	δ_{15}	δ_{16}	δ_{17}	δ_{18}
output	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9

compute state transformations for each block without red letters, using map applied to induction assumption for smaller input alphabet

a flip-flop machine that remembers most recent letters from input and phase 1

a letter-to-letter homomorphism that combines input and phase 2

Compose state transformations from phase 3, using induction assumption on a smaller state space.

More formally, if P is the image of the red transformations, then to compose the red state transformations, we use a Mealy machine with states P, with one machine for each initial state.

copy most recent letter from phase 4

compose state transformations from phases 1, 2 and 5

δ_{ij} is the state transformation of infix $a_i \dots a_j$

input	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9
phase 1	δ_{10}	δ_{11}	δ_{12}	|	δ_{54}	δ_{55}	δ_{56}	|	δ_{98}
phase 2				δ_{13}				δ_{57}	
phase 3				δ_{14}				δ_{58}	
phase 4				δ_{14}				δ_{18}	
phase 5				δ_{14}	δ_{14}	δ_{14}	δ_{14}	δ_{18}	δ_{18}
phase 6	δ_{10}	δ_{11}	δ_{12}	δ_{13}	δ_{14}	δ_{15}	δ_{16}	δ_{17}	δ_{18}
output	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9

compute state transformations for each block without red letters, using map applied to induction assumption for smaller input alphabet

a flip-flop machine that remembers most recent letters from input and phase 1

a letter-to-letter homomorphism that combines input and phase 2

Compose state transformations from phase 3, using induction assumption on a smaller state space.

More formally, if P is the image of the red transformations, then to compose the red state transformations, we use a Mealy machine with states P, with one machine for each initial state.

copy most recent letter from phase 4

compose state transformations from phases 1, 2 and 5

δ_{ij} is the state transformation of infix $a_i \dots a_j$

input	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9
phase 1	δ_{10}	δ_{11}	δ_{12}	|	δ_{54}	δ_{55}	δ_{56}	|	δ_{98}
phase 2				δ_{13}				δ_{57}	
phase 3				δ_{14}				δ_{58}	
phase 4				δ_{14}				δ_{18}	
phase 5				δ_{14}	δ_{14}	δ_{14}	δ_{14}	δ_{18}	δ_{18}
phase 6	δ_{10}	δ_{11}	δ_{12}	δ_{13}	δ_{14}	δ_{15}	δ_{16}	δ_{17}	δ_{18}
output	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9

compute state transformations for each block without red letters, using map applied to induction assumption for smaller input alphabet

a flip-flop machine that remembers most recent letters from input and phase 1

a letter-to-letter homomorphism that combines input and phase 2

Compose state transformations from phase 3, using induction assumption on a smaller state space.

More formally, if P is the image of the red transformations, then to compose the red state transformations, we use a Mealy machine with states P, with one machine for each initial state.

copy most recent letter from phase 4

compose state transformations from phases 1, 2 and 5

δ_{ij} is the state transformation of infix $a_i \dots a_j$

input	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9
phase 1	δ_{10}	δ_{11}	δ_{12}	|	δ_{54}	δ_{55}	δ_{56}	|	δ_{98}
phase 2				δ_{13}				δ_{57}	
phase 3				δ_{14}				δ_{58}	
phase 4				δ_{14}				δ_{18}	
phase 5				δ_{14}	δ_{14}	δ_{14}	δ_{14}	δ_{18}	δ_{18}
phase 6	δ_{10}	δ_{11}	δ_{12}	δ_{13}	δ_{14}	δ_{15}	δ_{16}	δ_{17}	δ_{18}
output	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9

compute state transformations for each block without red letters, using map applied to induction assumption for smaller input alphabet

a flip-flop machine that remembers most recent letters from input and phase 1

a letter-to-letter homomorphism that combines input and phase 2

Compose state transformations from phase 3, using induction assumption on a smaller state space.

More formally, if P is the image of the red transformations, then to compose the red state transformations, we use a Mealy machine with states P, with one machine for each initial state.

copy most recent letter from phase 4

compose state transformations from phases 1, 2 and 5

δ_{ij} is the state transformation of infix $a_i \dots a_j$

input	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9
phase 1	δ_{10}	δ_{11}	δ_{12}	|	δ_{54}	δ_{55}	δ_{56}	|	δ_{98}
phase 2				δ_{13}				δ_{57}	
phase 3				δ_{14}				δ_{58}	
phase 4				δ_{14}				δ_{18}	
phase 5				δ_{14}	δ_{14}	δ_{14}	δ_{14}	δ_{18}	δ_{18}
phase 6	δ_{10}	δ_{11}	δ_{12}	δ_{13}	δ_{14}	δ_{15}	δ_{16}	δ_{17}	δ_{18}
output	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9

compute state transformations for each block without red letters, using map applied to induction assumption for smaller input alphabet

a flip-flop machine that remembers most recent letters from input and phase 1

a letter-to-letter homomorphism that combines input and phase 2

Compose state transformations from phase 3, using induction assumption on a smaller state space.

More formally, if P is the image of the red transformations, then to compose the red state transformations, we use a Mealy machine with states P, with one machine for each initial state.

copy most recent letter from phase 4

compose state transformations from phases 1, 2 and 5

δ_{ij} is the state transformation of infix $a_i \dots a_j$

input	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9
phase 1	δ_{10}	δ_{11}	δ_{12}	|	δ_{54}	δ_{55}	δ_{56}	|	δ_{98}
phase 2				δ_{13}				δ_{57}	
phase 3				δ_{14}				δ_{58}	
phase 4				δ_{14}				δ_{18}	
phase 5				δ_{14}	δ_{14}	δ_{14}	δ_{14}	δ_{18}	δ_{18}
phase 6	δ_{10}	δ_{11}	δ_{12}	δ_{13}	δ_{14}	δ_{15}	δ_{16}	δ_{17}	δ_{18}
output	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9

compute state transformations for each block without red letters, using map applied to induction assumption for smaller input alphabet

a flip-flop machine that remembers most recent letters from input and phase 1

a letter-to-letter homomorphism that combines input and phase 2

Compose state transformations from phase 3, using induction assumption on a smaller state space.

More formally, if P is the image of the red transformations, then to compose the red state transformations, we use a Mealy machine with states P, with one machine for each initial state.

copy most recent letter from phase 4

compose state transformations from phases 1, 2 and 5

δ_{ij} is the state transformation of infix $a_i \dots a_j$

input	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9
phase 1	δ_{10}	δ_{11}	δ_{12}	|	δ_{54}	δ_{55}	δ_{56}	|	δ_{98}
phase 2				δ_{13}				δ_{57}	
phase 3				δ_{14}				δ_{58}	
phase 4				δ_{14}				δ_{18}	
phase 5				δ_{14}	δ_{14}	δ_{14}	δ_{14}	δ_{18}	δ_{18}
phase 6	δ_{10}	δ_{11}	δ_{12}	δ_{13}	δ_{14}	δ_{15}	δ_{16}	δ_{17}	δ_{18}
output	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9

compute state transformations for each block without red letters, using map applied to induction assumption for smaller input alphabet

a flip-flop machine that remembers most recent letters from input and phase 1

a letter-to-letter homomorphism that combines input and phase 2

Compose state transformations from phase 3, using induction assumption on a smaller state space.

More formally, if P is the image of the red transformations, then to compose the red state transformations, we use a Mealy machine with states P, with one machine for each initial state.

copy most recent letter from phase 4

compose state transformations from phases 1, 2 and 5

δ_{ij} is the state transformation of infix $a_i \dots a_j$

input	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9
phase 1	δ_{10}	δ_{11}	δ_{12}	|	δ_{54}	δ_{55}	δ_{56}	|	δ_{98}
phase 2				δ_{13}				δ_{57}	
phase 3				δ_{14}				δ_{58}	
phase 4				δ_{14}				δ_{18}	
phase 5				δ_{14}	δ_{14}	δ_{14}	δ_{14}	δ_{18}	δ_{18}
phase 6	δ_{10}	δ_{11}	δ_{12}	δ_{13}	δ_{14}	δ_{15}	δ_{16}	δ_{17}	δ_{18}
output	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9

compute state transformations for each block without red letters, using map applied to induction assumption for smaller input alphabet

a flip-flop machine that remembers most recent letters from input and phase 1

a letter-to-letter homomorphism that combines input and phase 2

Compose state transformations from phase 3, using induction assumption on a smaller state space.

More formally, if P is the image of the red transformations, then to compose the red state transformations, we use a Mealy machine with states P, with one machine for each initial state.

copy most recent letter from phase 4

compose state transformations from phases 1, 2 and 5

δ_{ij} is the state transformation of infix $a_i \dots a_j$

input	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9
phase 1	δ_{10}	δ_{11}	δ_{12}	|	δ_{54}	δ_{55}	δ_{56}	|	δ_{98}
phase 2				δ_{13}				δ_{57}	
phase 3				δ_{14}				δ_{58}	
phase 4				δ_{14}				δ_{18}	
phase 5				δ_{14}	δ_{14}	δ_{14}	δ_{14}	δ_{18}	δ_{18}
phase 6	δ_{10}	δ_{11}	δ_{12}	δ_{13}	δ_{14}	δ_{15}	δ_{16}	δ_{17}	δ_{18}
output	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9

compute state transformations for each block without red letters, using map applied to induction assumption for smaller input alphabet

a flip-flop machine that remembers most recent letters from input and phase 1

a letter-to-letter homomorphism that combines input and phase 2

Compose state transformations from phase 3, using induction assumption on a smaller state space.

More formally, if P is the image of the red transformations, then to compose the red state transformations, we use a Mealy machine with states P, with one machine for each initial state.

copy most recent letter from phase 4

compose state transformations from phases 1, 2 and 5

δ_{ij} is the state transformation of infix $a_i \dots a_j$

input	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9
phase 1	δ_{10}	δ_{11}	δ_{12}	|	δ_{54}	δ_{55}	δ_{56}	|	δ_{98}
phase 2				δ_{13}				δ_{57}	
phase 3				δ_{14}				δ_{58}	
phase 4				δ_{14}				δ_{18}	
phase 5				δ_{14}	δ_{14}	δ_{14}	δ_{14}	δ_{18}	δ_{18}
phase 6	δ_{10}	δ_{11}	δ_{12}	δ_{13}	δ_{14}	δ_{15}	δ_{16}	δ_{17}	δ_{18}
output	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9

compute state transformations for each block without red letters, using map applied to induction assumption for smaller input alphabet

a flip-flop machine that remembers most recent letters from input and phase 1

a letter-to-letter homomorphism that combines input and phase 2

Compose state transformations from phase 3, using induction assumption on a smaller state space.

More formally, if P is the image of the red transformations, then to compose the red state transformations, we use a Mealy machine with states P, with one machine for each initial state.

copy most recent letter from phase 4

compose state transformations from phases 1, 2 and 5

δ_{ij} is the state transformation of infix $a_i \dots a_j$

input	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9
phase 1	δ_{10}	δ_{11}	δ_{12}	|	δ_{54}	δ_{55}	δ_{56}	|	δ_{98}
phase 2				δ_{13}				δ_{57}	
phase 3				δ_{14}				δ_{58}	
phase 4				δ_{14}				δ_{18}	
phase 5				δ_{14}	δ_{14}	δ_{14}	δ_{14}	δ_{18}	δ_{18}
phase 6	δ_{10}	δ_{11}	δ_{12}	δ_{13}	δ_{14}	δ_{15}	δ_{16}	δ_{17}	δ_{18}
output	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9

compute state transformations for each block without red letters, using map applied to induction assumption for smaller input alphabet

a flip-flop machine that remembers most recent letters from input and phase 1

a letter-to-letter homomorphism that combines input and phase 2

Compose state transformations from phase 3, using induction assumption on a smaller state space.

More formally, if P is the image of the red transformations, then to compose the red state transformations, we use a Mealy machine with states P, with one machine for each initial state.

copy most recent letter from phase 4

compose state transformations from phases 1, 2 and 5

δ_{ij} is the state transformation of infix $a_i \dots a_j$

input	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9
phase 1	δ_{10}	δ_{11}	δ_{12}	|	δ_{54}	δ_{55}	δ_{56}	|	δ_{98}
phase 2				δ_{13}				δ_{57}	
phase 3				δ_{14}				δ_{58}	
phase 4				δ_{14}				δ_{18}	
phase 5				δ_{14}	δ_{14}	δ_{14}	δ_{14}	δ_{18}	δ_{18}
phase 6	δ_{10}	δ_{11}	δ_{12}	δ_{13}	δ_{14}	δ_{15}	δ_{16}	δ_{17}	δ_{18}
output	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9

compute state transformations for each block without red letters, using map applied to induction assumption for smaller input alphabet

a flip-flop machine that remembers most recent letters from input and phase 1

a letter-to-letter homomorphism that combines input and phase 2

Compose state transformations from phase 3, using induction assumption on a smaller state space.

More formally, if P is the image of the red transformations, then to compose the red state transformations, we use a Mealy machine with states P, with one machine for each initial state.

copy most recent letter from phase 4

compose state transformations from phases 1, 2 and 5

We decompose every Mealy machine into primes, using induction on:

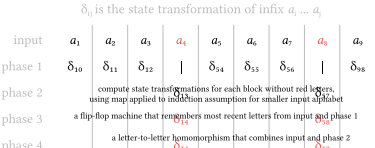
1. the number of states
2. the size of the input alphabet

The parameters are ordered lexicographically, which means that the induction advances if we decrease the state space but increase the input alphabet

In the induction basis of one state, the machine is flip-flop. also, reversible.
Consider the induction step.

Either the machine is already reversible, or there is some input letter $a \in \Sigma$ such that the corresponding state transformation $Q \longrightarrow Q$ has an image that is a proper subset of Q.

Intuition: words that end with a fall under item 1 of the induction assumption, and words that do not contain a fall under item 2.



Mealy = (reversible + flip-flop)*

we use the name *primes* for these two kinds

Clearly primes* ⊆ Mealy.

Because primes ⊆ Mealy and Mealy* = Mealy.

The interesting part is Mealy ⊆ primes*.

Step 1. primes* is closed under map.

For a Mealy machine $f : \Sigma^* \longrightarrow \Gamma^*$, define its map to be:

$$w_1 \mid w_2 \mid \ldots \mid w_n \quad \mapsto \quad f(w_1) \mid f(w_2) \mid \ldots \mid f(w_n)$$

the input and output alphabets are extended with a separator |

This is clearly a Mealy machine. But also:

Lemma. If a function is in primes*, then so is its map.

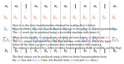
Proof. It is enough to prove that if a function is in primes, then its map is in primes*. because the map of f;g is (map f);(map g)

The map of a flip-flop is also a flip-flop.

The separator has a constant state transformation.

The interesting case is when f is reversible.

Suppose that f is reversible. We want to show that its map is in primes*.



Step 2. finish the job

We decompose every Mealy machine into primes, using induction on:

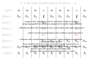
1. the number of states
2. the size of the input alphabet

The parameters are ordered lexicographically, which means that the induction advances if we decrease the state space but increase the input alphabet

In the induction basis of one state, the machine is flip-flop. also, reversible.
Consider the induction step.

Either the machine is already reversible, or there is some input letter $a \in \Sigma$ such that the corresponding state transformation $Q \longrightarrow Q$ has an image that is a proper subset of Q.

Intuition: words that end with a fall under item 1 of the induction assumption, and words that do not contain a fall under item 2.



Bonus. We have also proved the Schützenberger Theorem.

A Mealy machine is called *counter-free* if it does not have a counter:

a counter is an input letter that creates a non-trivial cycle on states

non-trivial means that there are at least two different states in the cycle

Counter-free machines are connected to first-order logic.

Theorem. counter-free Mealy = (flip-flop)*.

Proof. If we apply the proof of the Krohn-Rhodes Theorem to a counter-free machine, then counters will not be created, and only flip-flop machines will be needed.

Mealy = (reversible + flip-flop)*

we use the name *primes* for these two kinds

Clearly primes* ⊆ Mealy.

Because primes ⊆ Mealy and Mealy* = Mealy.

The interesting part is Mealy ⊆ primes*.

Step 1. primes* is closed under map.

For a Mealy machine $f: \Sigma^* \longrightarrow \Gamma^*$, define its map to be:

$$w_1 \mid w_2 \mid \ldots \mid w_n \quad \mapsto \quad f(w_1) \mid f(w_2) \mid \ldots \mid f(w_n)$$

the input and output alphabets are extended with a separator |

This is clearly a Mealy machine. But also:

Lemma. If a function is in primes*, then so is its map.

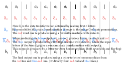
Proof. It is enough to prove that if a function is in primes, then its map is in primes*. because the map of f;g is (map f);(map g)

The map of a flip-flop is also a flip-flop.

The separator has a constant state transformation.

The interesting case is when f is reversible.

Suppose that f is reversible. We want to show that its map is in primes*.



Step 2. finish the job

We decompose every Mealy machine into primes, using induction on:

1. the number of states
2. the size of the input alphabet

The parameters are ordered lexicographically, which means that the induction advances if we decrease the state space but increase the input alphabet

In the induction basis of one state, the machine is flip-flop. also, reversible.
Consider the induction step.

Either the machine is already reversible, or there is some input letter $a \in \Sigma$ such that the corresponding state transformation $Q \longrightarrow Q$ has an image that is a proper subset of Q.

Intuition: words that end with a fall under item 1 of the induction assumption, and words that do not contain a fall under item 2.

Bonus. We have also proved the Schützenberger Theorem.

A Mealy machine is called *counter-free* if it does not have a counter:

a counter is an input letter that creates a non-trivial cycle on states

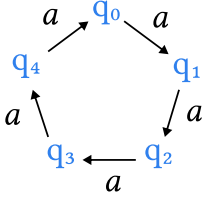
non-trivial means that there are at least two different states in the cycle

Counter-free machines are connected to first-order logic.

Theorem. counter-free Mealy = (flip-flop)*.

Proof. If we apply the proof of the Krohn-Rhodes Theorem to a counter-free machine, then counters will not be created, and only flip-flop machines will be needed.

A Mealy machine is called *counter-free* if it does not have a counter:



a counter is an input letter that creates a non-trivial cycle on states

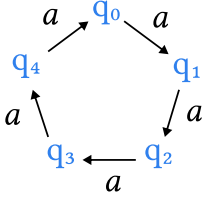
non-trivial means that there are at least two different states in the cycle

Counter-free machines are connected to first-order logic.

Theorem. counter-free Mealy = (flip-flop)*.

Proof. If we apply the proof of the Krohn-Rhodes Theorem to a counter-free machine, then counters will not be created, and only flip-flop machines will be needed.

A Mealy machine is called *counter-free* if it does not have a counter:



a counter is an input letter that creates a non-trivial cycle on states

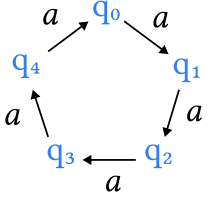
non-trivial means that there are at least two different states in the cycle

Counter-free machines are connected to first-order logic.

Theorem. counter-free Mealy = (flip-flop)*.

Proof. If we apply the proof of the Krohn-Rhodes Theorem to a counter-free machine, then counters will not be created, and only flip-flop machines will be needed.

A Mealy machine is called *counter-free* if it does not have a counter:



a counter is an input letter that creates a non-trivial cycle on states

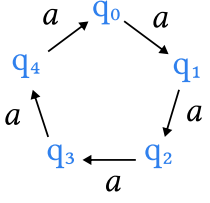
non-trivial means that there are at least two different states in the cycle

Counter-free machines are connected to first-order logic.

Theorem. counter-free Mealy = (flip-flop)*.

Proof. If we apply the proof of the Krohn-Rhodes Theorem to a counter-free machine, then counters will not be created, and only flip-flop machines will be needed.

A Mealy machine is called *counter-free* if it does not have a counter:



a counter is an input letter that creates a non-trivial cycle on states

non-trivial means that there are at least two different states in the cycle

Counter-free machines are connected to first-order logic.

Theorem. counter-free Mealy = (flip-flop)*.

Proof. If we apply the proof of the Krohn-Rhodes Theorem to a counter-free machine, then counters will not be created, and only flip-flop machines will be needed.

Mealy = (reversible + flip-flop)*

we use the name *primes* for these two kinds

Clearly primes* ⊆ Mealy.

Because primes ⊆ Mealy and Mealy* = Mealy.

The interesting part is Mealy ⊆ primes*.

Step 1. primes* is closed under map.

For a Mealy machine $f: \Sigma^* \longrightarrow \Gamma^*$, define its map to be:

$$w_1 \mid w_2 \mid \ldots \mid w_n \quad \mapsto \quad f(w_1) \mid f(w_2) \mid \ldots \mid f(w_n)$$

the input and output alphabets are extended with a separator |

This is clearly a Mealy machine. But also:

Lemma. If a function is in primes*, then so is its map.

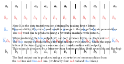
Proof. It is enough to prove that if a function is in primes, then its map is in primes*. because the map of f;g is (map f);(map g)

The map of a flip-flop is also a flip-flop.

The separator has a constant state transformation.

The interesting case is when f is reversible.

Suppose that f is reversible. We want to show that its map is in primes*.



Step 2. finish the job

We decompose every Mealy machine into primes, using induction on:

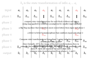
1. the number of states
2. the size of the input alphabet

The parameters are ordered lexicographically, which means that the induction advances if we decrease the state space but increase the input alphabet

In the induction basis of one state, the machine is flip-flop. also, reversible.
Consider the induction step.

Either the machine is already reversible, or there is some input letter $a \in \Sigma$ such that the corresponding state transformation $Q \longrightarrow Q$ has an image that is a proper subset of Q.

Intuition: words that end with a fall under item 1 of the induction assumption, and words that do not contain a fall under item 2.



Bonus. We have also proved the Schützenberger Theorem.

A Mealy machine is called *counter-free* if it does not have a counter:

a counter is an input letter that creates a non-trivial cycle on states

non-trivial means that there are at least two different states in the cycle

Counter-free machines are connected to first-order logic.

Theorem. counter-free Mealy = (flip-flop)*.

Proof. If we apply the proof of the Krohn-Rhodes Theorem to a counter-free machine, then counters will not be created, and only flip-flop machines will be needed.

Theorem. [Krohn-Rhodes 1965] Every Mealy machine f can be decomposed as $f = f_1; \ldots; f_n$ where each Mealy machine f_i is either:

Reversible: every state transformation is bijection; or

The state transformation of a letter is defined to be the corresponding function $Q \longrightarrow Q$. If a machine is reversible, and we know the last state, then we can reconstruct the run and output in a right-to-left pass.

Flip-flop: every state transformation is the identity or constant.

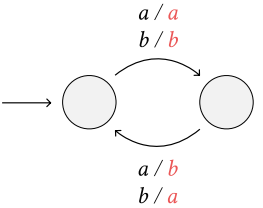
In a flip-flop machine, an input letter either keeps the state unchanged, or forgets it and sets it to some new state that depends on the letter

$$\text{Mealy} = (\underbrace{\text{reversible} + \text{flip-flop}})^*$$

we use the name *prime* for these two kinds

A Mealy machine is a DFA, but:

- each transition is labelled by one output letter
  and therefore it defines a letter-to-letter function (i.e. length preserving)
- we do not define accepting states
  it does not accept or reject, but only defines an output string



Formal definition.

Σ	Γ	Q	$q_0 \in Q$	$\delta : Q \times \Sigma \longrightarrow Q \times \Gamma$
input alphabet	output alphabet	states	initial state	transition function

Examples

Example. One-state Mealy machines are the same as letter-to-letter homomorphisms.
This is a function $\Sigma^* \longrightarrow \Gamma^*$ obtained by lifting some function $\Sigma \longrightarrow \Gamma$.
A (not necessarily letter-to-letter) homomorphism $lift_h$ a function $\Sigma \longrightarrow \Gamma^*$.

Example. The flip-flop machine.

Basic Properties

Theorem. (Functions computed by) Mealy machines are closed under composition.

Proof. A product construction.

Corollary. Mealy machines are continuous, i.e. inverse images of regular languages are regular.

Krohn-Rhodes Theorem

Theorem. [Krohn-Rhodes 1965] Every Mealy machine f can be decomposed as $f = f_1; ...; f_n$ where each Mealy machine f_i is either:

Reversible: every state transformation is bijection; or
The state transformation of a letter is defined to be the corresponding function $Q \longrightarrow Q$.
If a machine is reversible, and we know the last state, then we can reconstruct the run.

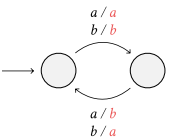
Transducers

Introduction and Mealy machines

We begin our course with Mealy machines, which are the simplest transducer model

A Mealy machine is a DFA, but:

- each transition is labelled by one output letter
  and therefore it defines a letter-to-letter function (i.e. length preserving)
- we do not define accepting states
  it does not accept or reject, but only defines an output string



Formal definition. Σ input alphabet; Γ output alphabet; Q states; $q_0 \in Q$ initial state; $\delta : Q \times \Sigma \longrightarrow Q \times \Gamma$ transition function

Examples

Basic Properties

Theorem. [Functions computed by] Mealy machines are closed under

Corollary. Mealy machines are continuous, i.e. (reverse

Krohn-Rhodes Theorem

Theorem. [Krohn-Rhodes 1965] Every Mealy machine f can be