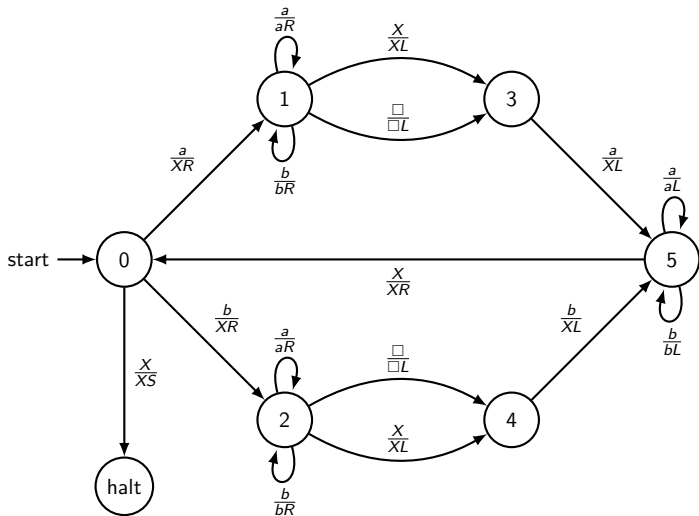


# Introduction to Theory of Computation

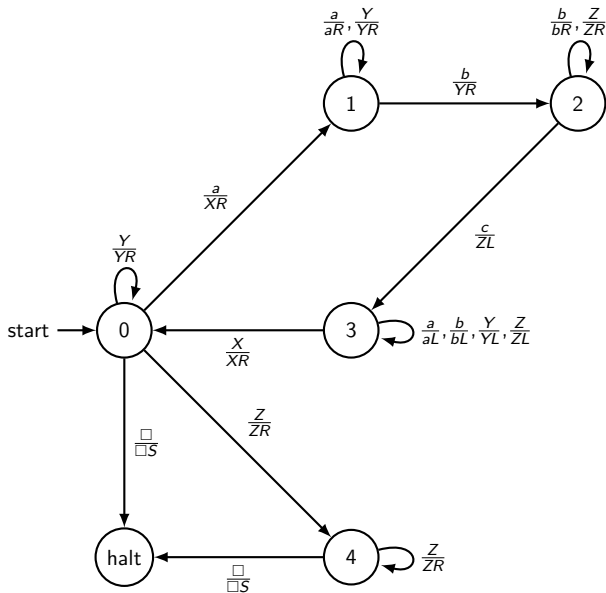
## Chapter 4, Turing Machines

November 30, 2015

# Even Palindromes



$a^n b^n c^n$



## Equivalent models:

1. One-tape Turing machines.
2.  $k$ -tape Turing machines.
3. Non-deterministic Turing machines.
4. Java programs.
5. Scheme programs.
6. C++ programs.
7. ...

# The Church-Turing Thesis

**Every computational process that is intuitively considered to be an algorithm can be converted to a Turing machine.**

# Decidability

A language  $A$  over  $\Sigma$  is *decidable* if there exists a Turing machine  $M$  such that for every string  $w \in \Sigma^*$ :

1. If  $w \in A$  then  $M$ , started on  $w$ , halts in the accept state.
2. If  $w \notin A$  then  $M$ , started on  $w$ , halts in the reject state.

# Describing machines and problems as strings

- ▶ We assume any machine (DFA, PDA, TM) can be described by a string  $M$  using some alphabet.
- ▶ The input to any machine is a string  $w$  using some alphabet.
- ▶ We can thus describe both a machine  $M$  and its input  $w$ , with a pair of strings:  $(M, w)$ .
- ▶ This pair can be converted to a single string  $\langle M, w \rangle$ .
- ▶ For convenience, we assume  $\langle M, w \rangle$  is encoded in binary.
- ▶ In general,  $\langle M \rangle$  means: encode  $M$  as a binary string.
- ▶ We can now define a language  $A$  as the set of all strings  $\langle M, w \rangle$  such that  $w$  is in the computation model of  $M$ .

The language  $A_{DFA}$  is decidable

$$A_{DFA} = \{\langle M, w \rangle : M \text{ is a DFA that accepts } w\}$$

Proof?



# The language $A_{DFA}$ is decidable

$$A_{DFA} = \{\langle M, w \rangle : M \text{ is a DFA that accepts } w\}$$

Proof?

- ▶ Given input  $\langle M, w \rangle$ :
  - ▶ Run  $M$  on  $w$ .
  - ▶ It must terminate.
  - ▶ If it accepts, accept, else reject.

The language  $A_{NFA}$  is decidable

$$A_{NFA} = \{\langle M, w \rangle : M \text{ is a NFA that accepts } w\}$$

Proof?

# The language $A_{NFA}$ is decidable

$$A_{NFA} = \{\langle M, w \rangle : M \text{ is a NFA that accepts } w\}$$

Proof?

- ▶ Given input  $\langle M, w \rangle$ :
  - ▶ Convert NFA  $M$  to DFA  $N$ .
  - ▶ This algorithm terminates.
  - ▶ Run  $N$  on  $w$ .
  - ▶ It must terminate.
  - ▶ If it accepts, accept, else reject.

The language  $A_{CFG}$  is decidable

$$A_{NFA} = \{\langle M, w \rangle : M \text{ is a CFG that accepts } w\}$$

Proof?

# The language $A_{CGF}$ is decidable

$$A_{NFA} = \{\langle M, w \rangle : M \text{ is a CFG that accepts } w\}$$

Proof?

- ▶ Given input  $\langle M, w \rangle$ :
  - ▶ Convert CFG  $M$  to Chomsky normal form CFG  $N$ .
  - ▶ This algorithm terminates.
  - ▶ Run  $N$  on  $w$  for all derivations up to length  $2|w|$ .
  - ▶ There are a finite number of these, so it must terminate.
  - ▶ If any derivation accepts, accept, else reject.

The language  $A_{TM}$  is not decidable.

$$A_{TM} = \{\langle M, w \rangle : M \text{ is a TM that accepts } w\}$$

Proof?

# The language $A_{TM}$ is not decidable.

$$A_{TM} = \{\langle M, w \rangle : M \text{ is a TM that accepts } w\}$$

Proof? By contradiction.

- ▶ Assume there is a TM  $H$  that decides this language.
- ▶ Construct the following TM,  $D$ :

$D$ : On input  $\langle M \rangle$ :

Step 1: Run  $H$  on  $\langle M, \langle M \rangle \rangle$ .

Step 2: If  $H$  accepts, reject, else accept.

- ▶ If  $H$  accepts  $\langle D, \langle D \rangle \rangle$ , then  $D$  rejects  $\langle D \rangle$ .
- ▶ If  $H$  rejects  $\langle D, \langle D \rangle \rangle$ , then  $D$  accepts  $\langle D \rangle$ .
- ▶  $H$  does not recognize  $A_{TM}$ .

# Diagonal argument

- ▶ Machine  $H$  that decides  $A_{TM}$  can fill in this table:

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	...
$M_0$	<b>accept</b>	accept	accept	reject	accept	reject	...
$M_1$	accept	<b>reject</b>	accept	accept	accept	reject	...
$M_2$	accept	reject	<b>accept</b>	accept	accept	reject	...
$M_3$	accept	accept	reject	<b>reject</b>	accept	accept	...
$M_4$	reject	accept	accept	reject	<b>accept</b>	accept	...
$M_5$	reject	reject	accept	accept	accept	<b>reject</b>	...
...	...	...	...	...	...	...	...

- ▶ Machine  $D$  uses  $H$  to give the opposite answer on the diagonal.
- ▶ Machine  $H$  must give the wrong answer somewhere on machine  $D$ .



The language *Halt* is not decidable.

$$\textit{Halt} = \{ \langle M, w \rangle : M \text{ is a TM that terminates on } w \}$$

Proof?

# The language *Halt* is not decidable.

$$\text{Halt} = \{ \langle M, w \rangle : M \text{ is a TM that terminates on } w \}$$

Proof?

By contradiction. Assume there is a TM  $H$  that decides this language. Construct the following TM,  $Q$ :

$Q$ : On input  $\langle M \rangle$ :

While  $H(\langle M, \langle M \rangle \rangle)$  do endwhile

- ▶ What happens if we run  $Q$  on itself?
- ▶  $Q(\langle Q \rangle)$  terminates iff  $Q(\langle Q \rangle)$  does not terminate.

# Rice's Theorem

Let  $\mathcal{T}$  be the set of all binary encoded TMs.

Let  $\mathcal{P}$  be a subset of  $\mathcal{T}$  such that

1.  $\mathcal{P} \neq \emptyset$
2.  $\mathcal{P} \neq \mathcal{T}$
3. If  $L(M_1) = L(M_2)$ , then either both or neither is in  $\mathcal{P}$ .

Then  $\mathcal{P}$  is undecidable.

# Enumerability

A language  $A$  over  $\Sigma$  is *enumerable* if there exists a Turing machine  $M$  such that for every string  $w \in \Sigma^*$ :

1. If  $w \in A$  then  $M$ , started on  $w$ , halts in the accept state.
2. If  $w \notin A$  then  $M$ , started on  $w$ , either halts in the reject state or loops forever.

# Hilbert's problem is enumerable but not decidable

$Hilbert = \{ \langle p \rangle : p \text{ is a polynomial with integer coefficients} \\ \text{that has an integral root} \}$

The language  $A_{TM}$  is enumerable but not decidable.

$$A_{TM} = \{\langle M, w \rangle : M \text{ is a TM that accepts } w\}$$

Proof?

# Why do we call it enumerable?

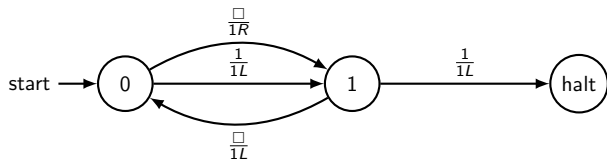
- ▶ If we can enumerate the elements with an algorithm, then we can create an algorithm to correctly identify elements of the set.
- ▶ If we can correctly identify elements of the set, then we can build an algorithm to enumerate them.

# Busy beavers are not enumerable

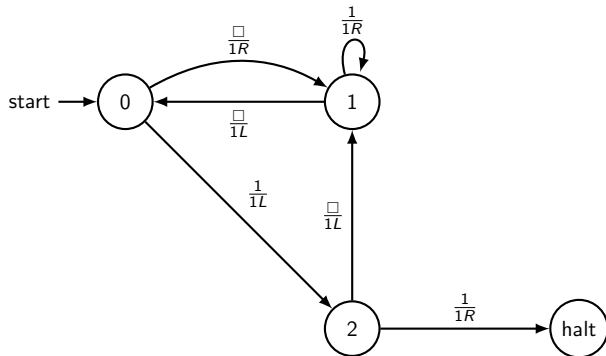
The  $n$ th busy beaver number is the largest (finite) number of 1s that can be output by a Turing machine with  $n$  states.



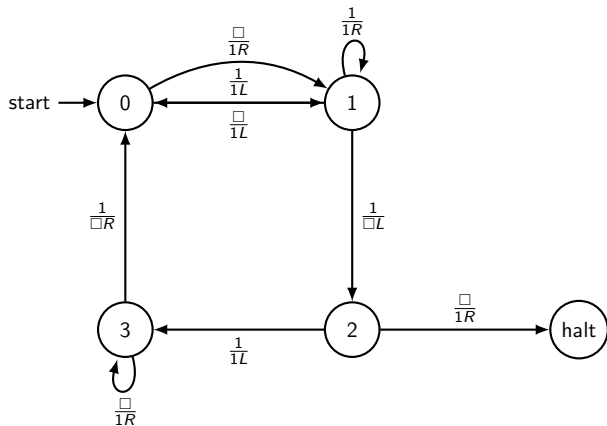
## 2 State Busy Beaver: four 1s



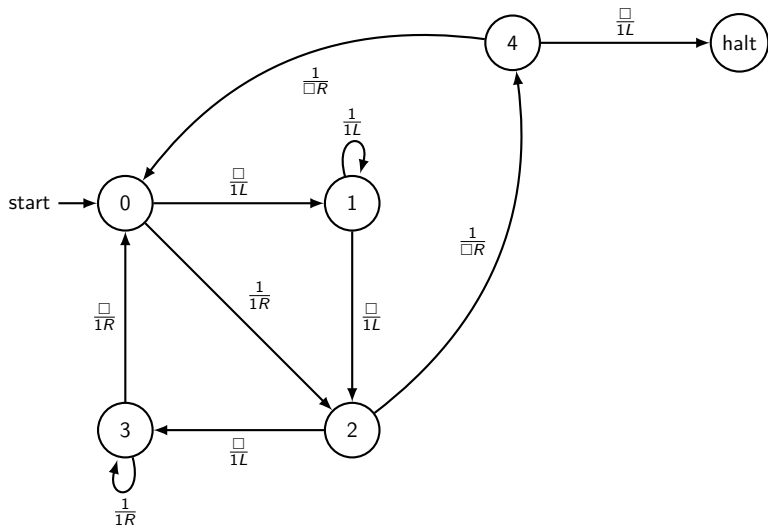
### 3 State Busy Beaver: six 1s



## 4 State Busy Beaver: thirteen 1s



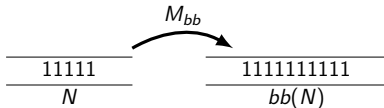
## 5 State Busy Beaver (?): 4098 1s



# Proof Busy Beaver function is uncomputable

Proof by contradiction.

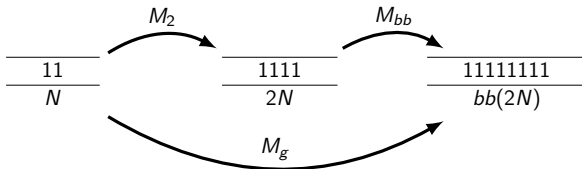
- ▶ Let  $bb(n)$  be the largest (finite) number of 1's output by a Turing Machine with  $n$  states.
- ▶ Suppose there is a Turing Machine  $M_{bb}$  that computes  $bb(n)$ , that is, starting with  $n$  on the tape, the machine halts with  $bb(n)$  on the tape.



- ▶ Note: this is a new use of TMs, computing a function from input to output, not recognizing a language.

# Busy Beaver proof

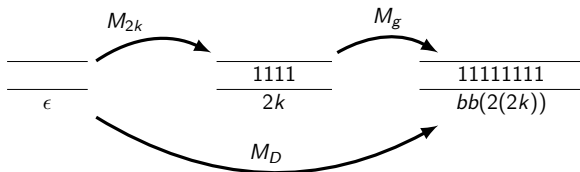
- ▶ Let  $g(n) = bb(2n)$ . We can build a TM for  $g$  by starting with a machine that doubles the input, and then runs the machine  $M_{bb}$ .



- ▶ Suppose the machine for  $g$ ,  $M_g$  has  $k$  states.

## Busy Beaver proof

- ▶ Build a machine  $M_{2k}$  with  $2k$  states that does nothing but put  $2k$  1s on a blank tape.
- ▶ Now build a machine  $M_D$  that starts by putting  $2k$  1's on the tape, and then runs the  $M_g$  machine.



- ▶  $M_D$  can be built with  $3k$  states.
- ▶ The output of  $M_D$  is  $g(2k) = bb(2(2k)) = bb(4k)$  1s.
- ▶ Do you see the problem?