Monash University
Faculty of Information Technology

# Lecture 14
# Turing Machines and Computability

Slides by David Albrecht (2011),
some additions and modifications by Graham Farr (2013, 2016).

FIT2014 Theory of Computation

# Overview

- Turing Machines
- Converting Finite Automaton to a Turing Machine
- Building Turing Machines
- Turing machines for computing functions
- Church's thesis

# Effective Process

- Can be done with pencil and paper.
- Is a finite set of instructions.
- Demands neither insight or ingenuity.
- Will definitely work without error.
- Produces in a finite number of steps either:
  - A final result, or
  - If the result is a sequence, each symbol in the sequence.

Alan Turing (1912-1954)
http://www.npg.org.uk/collections/search/portrait/mw165875

# How to model computation?

Consider a person doing a computation (pencil & paper).

At any given time, the person is …
- focused on some particular **position** on the paper;
- reading the **symbol** at the current position;
- in some particular mental **state**, i.e., is doing some particular part of the computation.

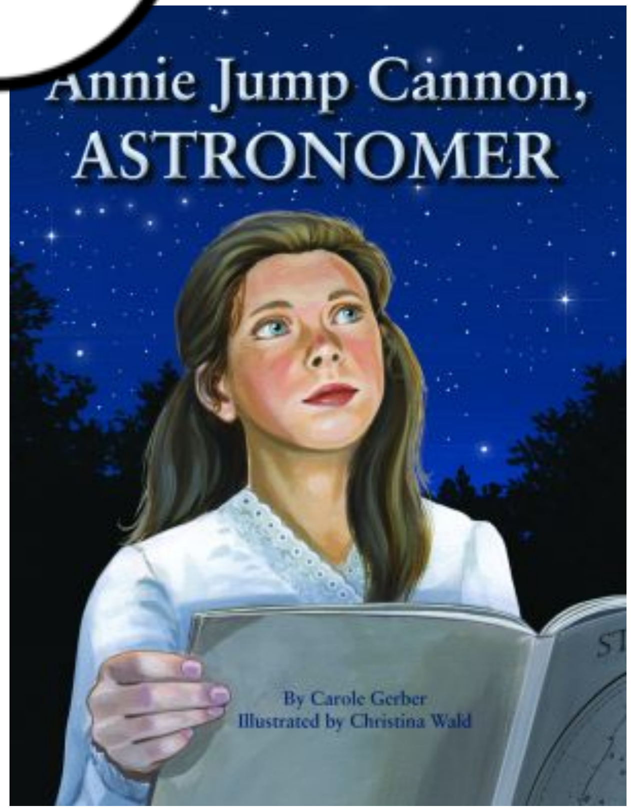
Depending on the state and symbol, the person then …
- writes a **symbol** there
  - (possibly overwriting what is already there);
- may change their **state**;
- **moves** their attention nearby.

11 Dec 2014

www.google.com

carry . . .
go to top of column ...
add column ...

Annie Jump Cannon (1863–1941)
Photo: Harvard College Observatory
http://www.skyandtelescope.com/astronomy-news/digitizing-harvards-century-of-sky/




http://www.pelicanpub.com/proddetail.php?prod=9781589809116

# Turing machine

Set-up:

- infinitely long **tape** divided into cells
- each cell may contain a **symbol** from a finite alphabet
- **head** scans one tape cell at a time
- at any time, the machine is in some **state**

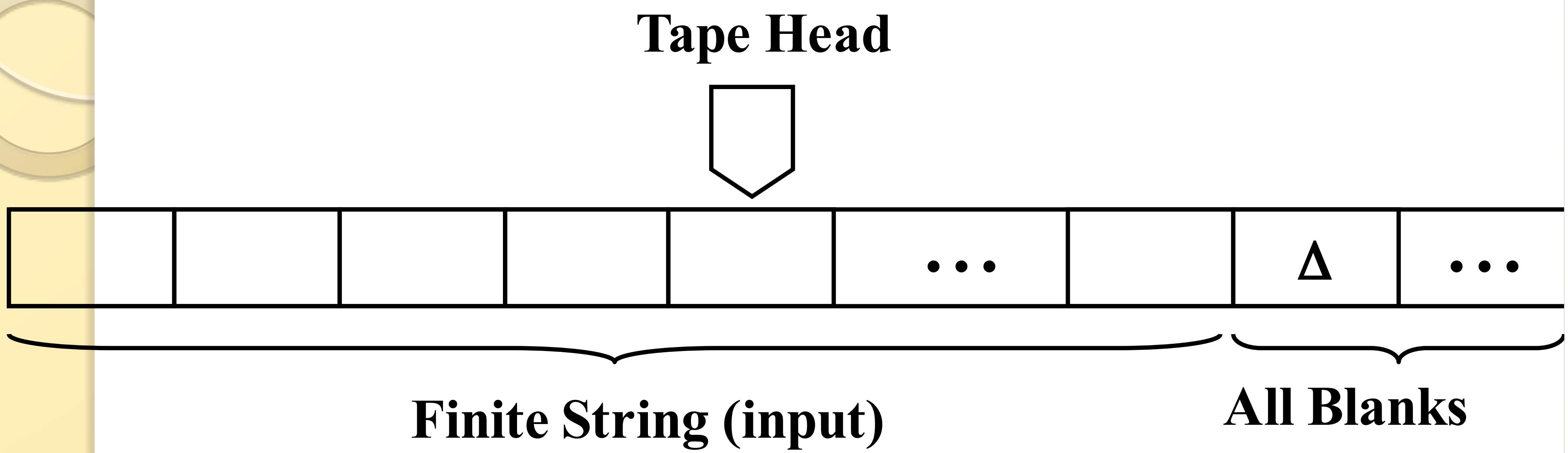
Program:

- For each *state* and *symbol*, specify the *next state*, *next symbol*, and *direction* (one step left, or one step right).

Computation step:

- At each step, apply the appropriate instruction.

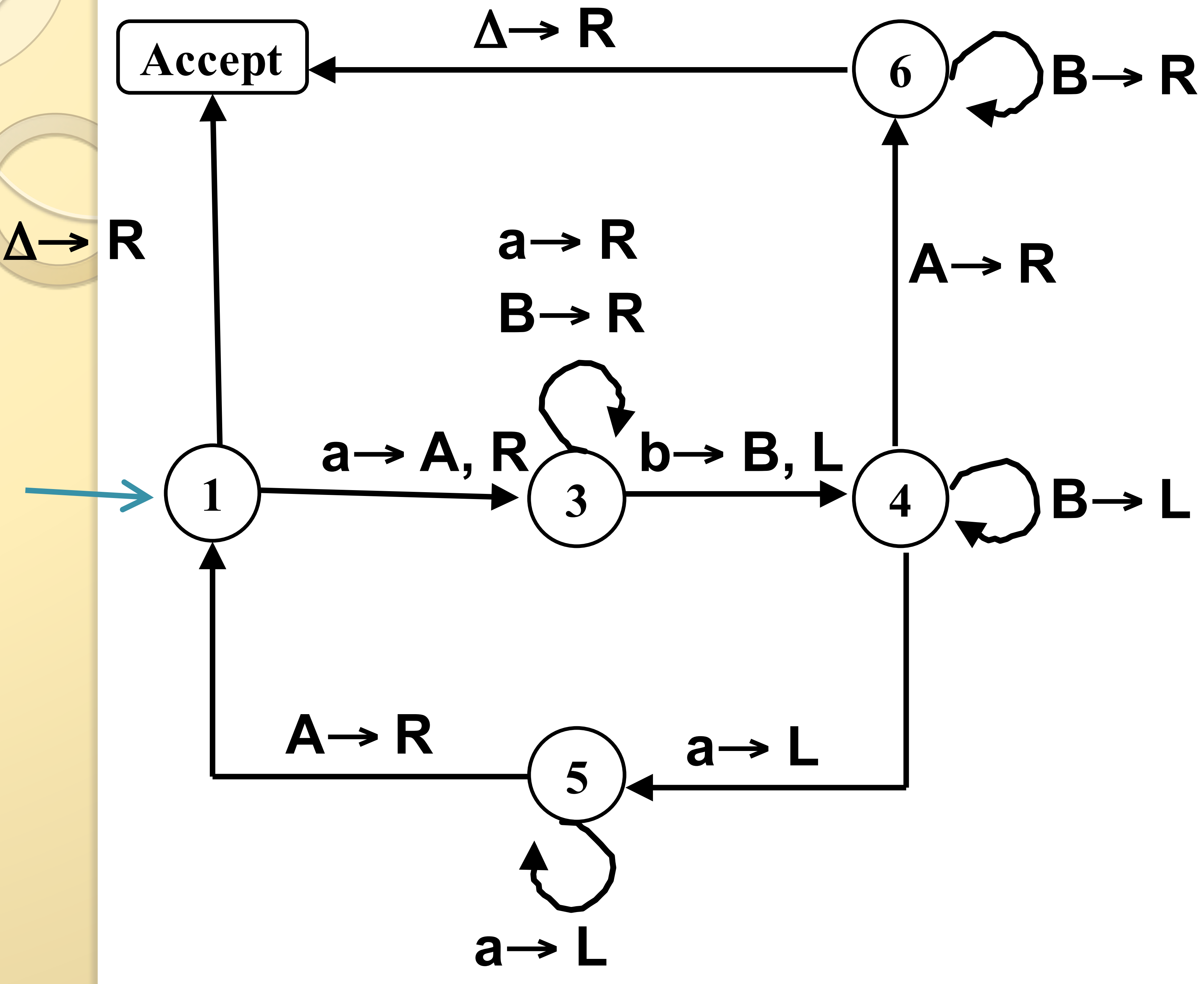
# Turing machine




Tape Head can:

- Move **left** and **right**.
- **Read letters** from the tape.
- **Write letters** onto the tape.

Computation is ***deterministic***.



Accept
Δ→ R
6
B→ R
Δ→ R
a→ R
B→ R
A→ R
a→ A, R
b→ B, L
1
3
4
B→ L
A→ R
a→ L
5
a→ L

# TM Components

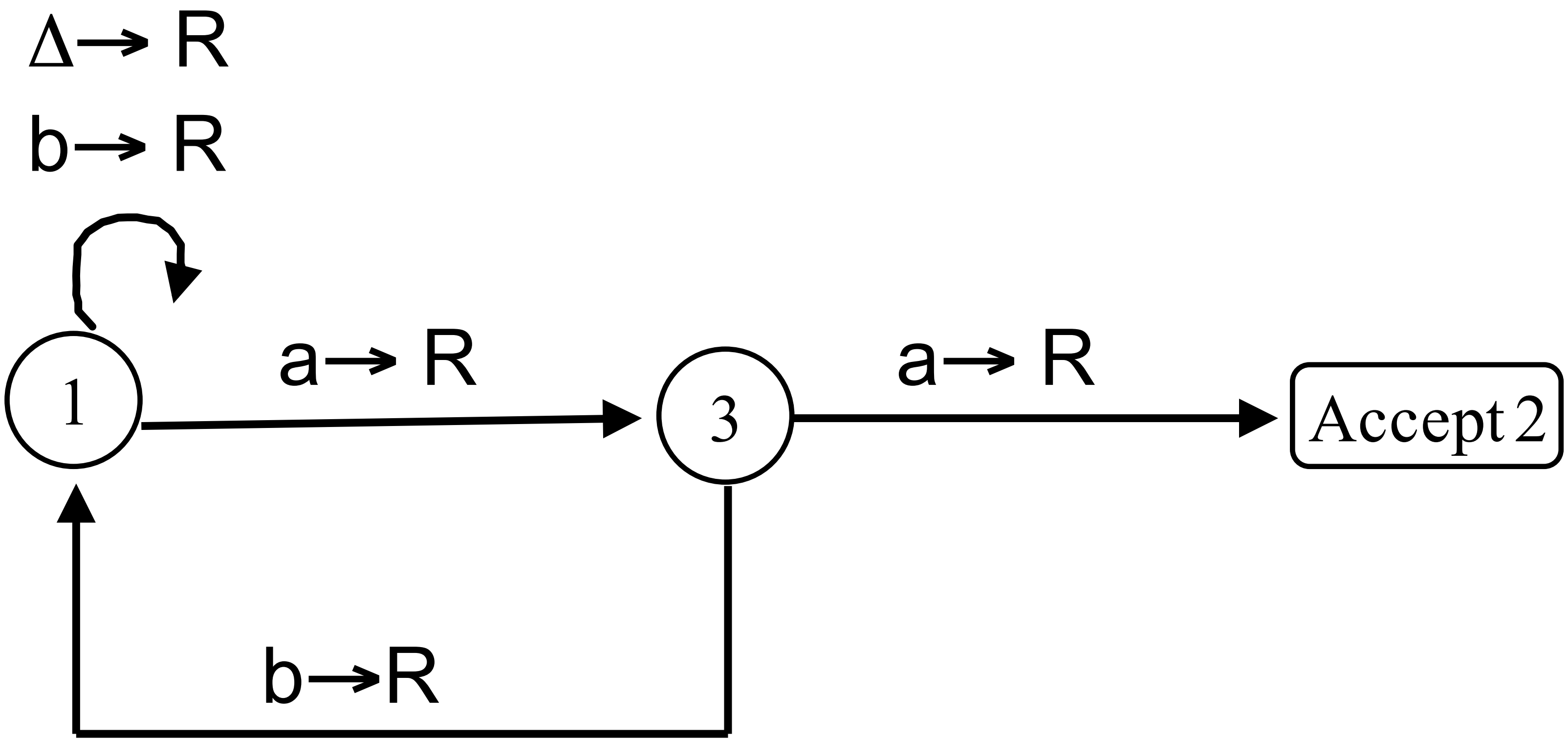
- A Tape and Tape Head
- An Input Alphabet
- A Tape Alphabet
- A finite set of states
  - Each state is numbered by an integer $\geq 1$.
  - **Start State** (1)
  - **Accept State** (2)
  - Note: a *Reject state* is optional (if crash = reject).
- A finite set of rules **letter → letter, direction** between states.

# Definitions

For a Turing Machine T

- **Accept(T)**
  - The set of strings leading to the **Accept** state.
  - Called the **language accepted by T.**
- **Reject(T)**
  - The set of strings that **crash**, or lead to a **Reject** state (if there is one), during execution.
- **Loop(T)**
  - The set of strings that cause T to loop forever.

# Example




- **Accept(T)** = strings with a double **aa**
- **Reject(T)** = strings without a double **aa** that end in **a**
- **Loop(T)** = **ε** or strings without a double **aa** that end in **b**

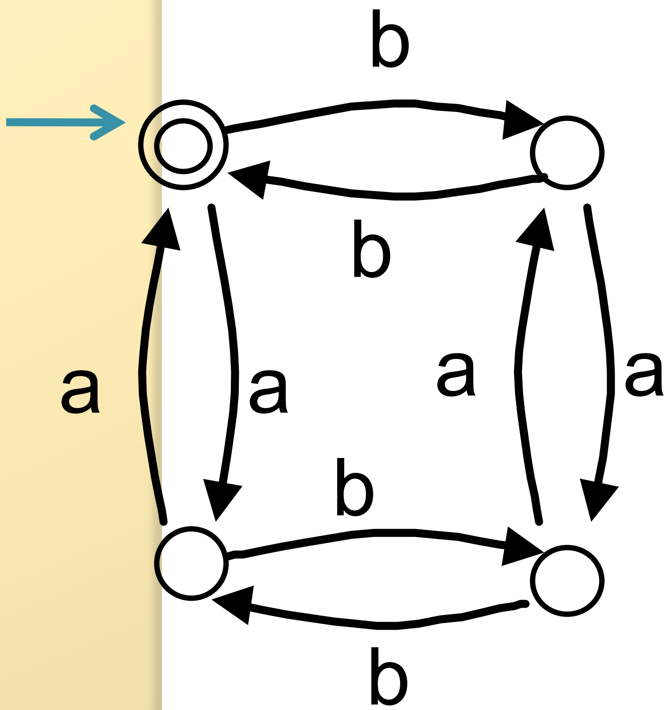
# Regular Languages

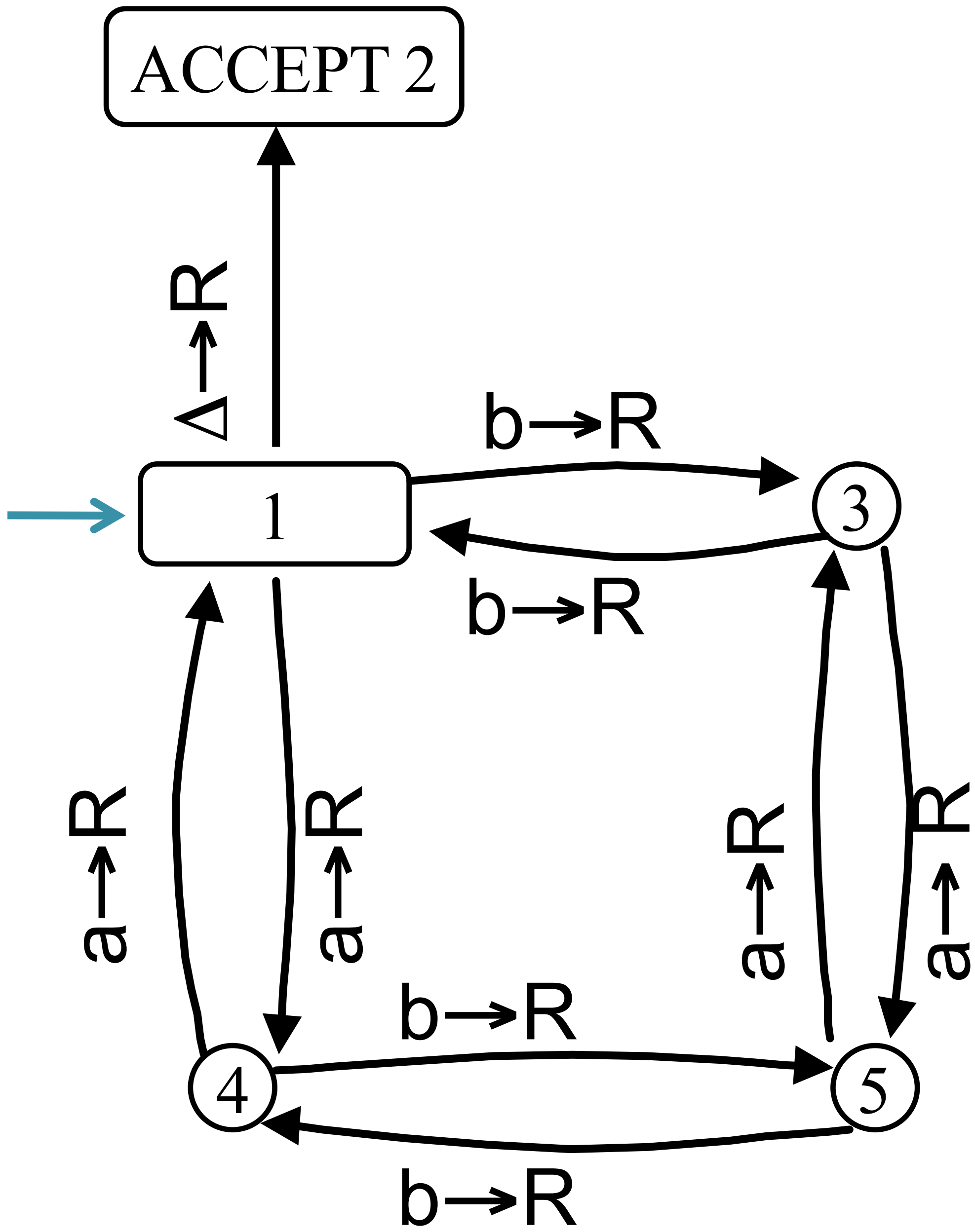
Every **Regular Language** can be accepted by a **Turing Machine**.

# Convert Finite Automaton into a Turing Machine

- Label start state with 1.
- Label all other states with a integer ≥ 3.
- Change the edge labels.
  - **a** to **a→R**
  - **b** to **b→R**
- Delete the second circle from all the Final states, and add an edge from each Final state to **Accept 2**, labelled with Δ →**R**.

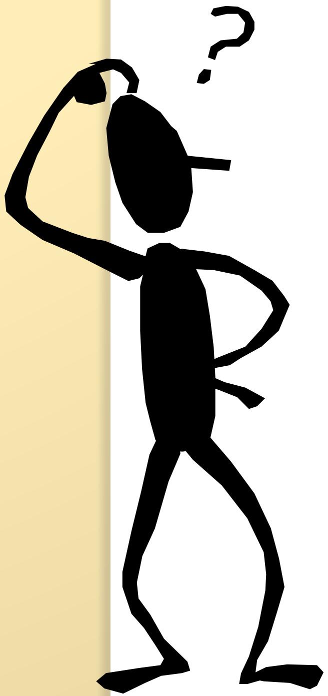
# EVEN-EVEN



# Problem

Build a Turing Machine that accepts the language $\{a^n b^n : n \geq 0\}$.



If the current letter is blank, then **Accept** string.

Loop {

    If current letter is **a**, then change **a** to **A** & move **right**.

    Move **right** over any **a**'s and **B**'s.

    If current letter is **b**, then change **b** to **B** & move **left.**

    Move **left** over any **B**'s.

    If current letter is **A**, then move **right** & **exit** the loop.

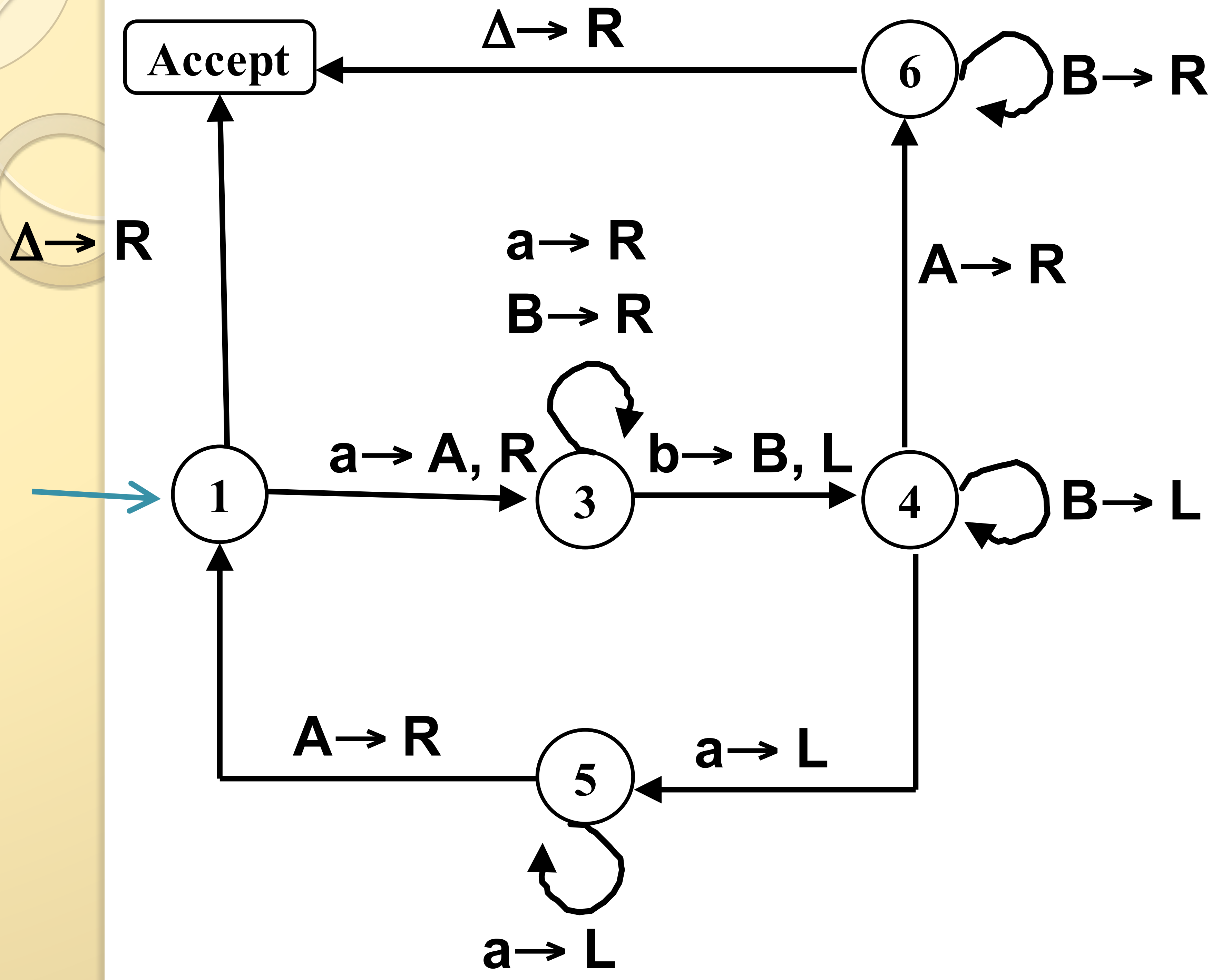
    Else if current letter is **a** move **left** over any **a**'s.

    If current letter is **A**, then move **right**.

}

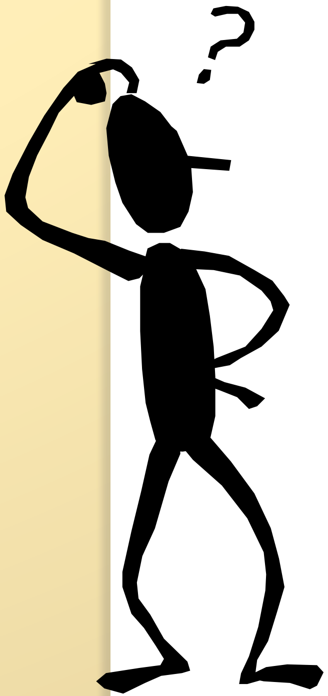
Move **right** over any **B's**.

If current letter is **blank**, then **Accept** string.


Accept
Δ→ R
6
B→ R
Δ→ R
a→ R
B→ R
A→ R
1
a→ A, R
3
b→ B, L
4
B→ L
A→ R
5
a→ L
a→ L

# Problem

Build a Turing Machine that accepts the language $\{a^n b^n a^n : n \geq 0\}$.



Loop {

If current letter is **blank**, then **Accept** string.

If current letter is **a**, then change **a** to **A** & move **right**.

Move **right** over **a*bb***.

If current letter is **a**, then move **left**.

If current letter is **b**, then change **b** to **a** & move **right**.

Move **right** over any **a**'s.

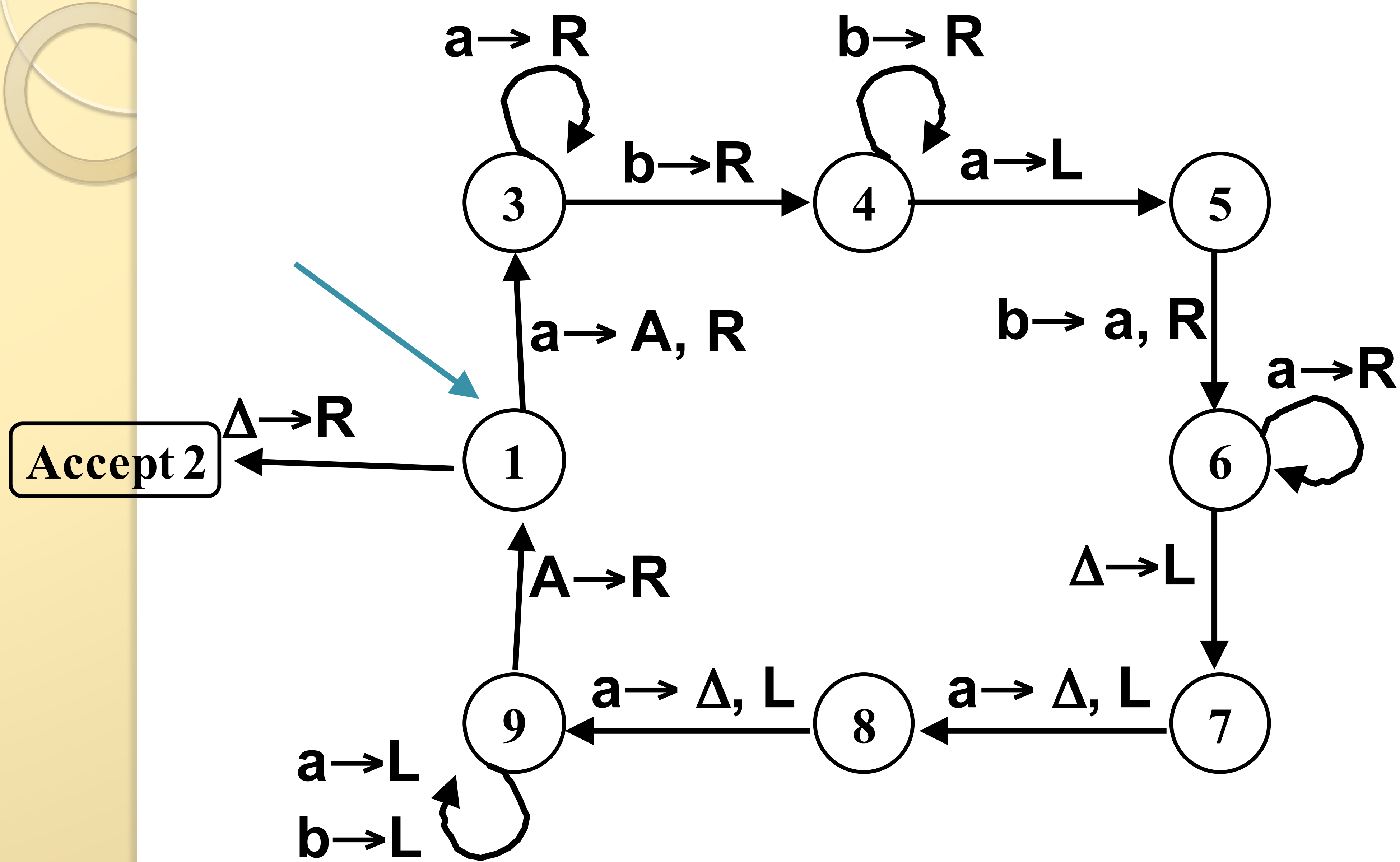
If current letter is **blank**, then **delete** 2 **a**'s on the left.

Move **left** over any **a**'s and **b**'s.

If current letter is **A**, then move **right**.

}

# TM for $a^nb^na^n$



# Other Machines

- **Queue automaton**
  - Like a deterministic PDA, but uses a Queue.
- **2PDA**
  - Like a deterministic PDA, but with 2 Stacks.
- **NTM**
  - A Nondeterministic Turing Machine.
- **kTM**
  - A Turing Machine with k Tapes.

# Theorems

- Any language which a Turing machine can accept can also be defined by any of these machines, and visa-versa.
- There are algorithms to convert all these machines (including Turing Machines) into each other.

# Turing machines for computing functions

So far, our Turing machines just accept/reject.

TMs can also compute functions.

What kinds of objects can Turing machines work with?

- any objects that can be encoded as strings …

# ASCII Code

| | | | |
|---|---|---|---|
| 0 | bbaaaa | 5 | bbabab |
| 1 | bbaaab | 6 | bbabba |
| 2 | bbaaba | 7 | bbabbb |
| 3 | bbaabb | 8 | bbbaaa |
| 4 | bbabaa | 9 | bbbaab |

# Binary Code

| | |
|---|---|
| 0 | a |
| 1 | b |
| 2 | ba |
| 3 | bb |
| 4 | baa |
| 5 | bab |
| 6 | bba |
| 7 | bbb |
| : | : |

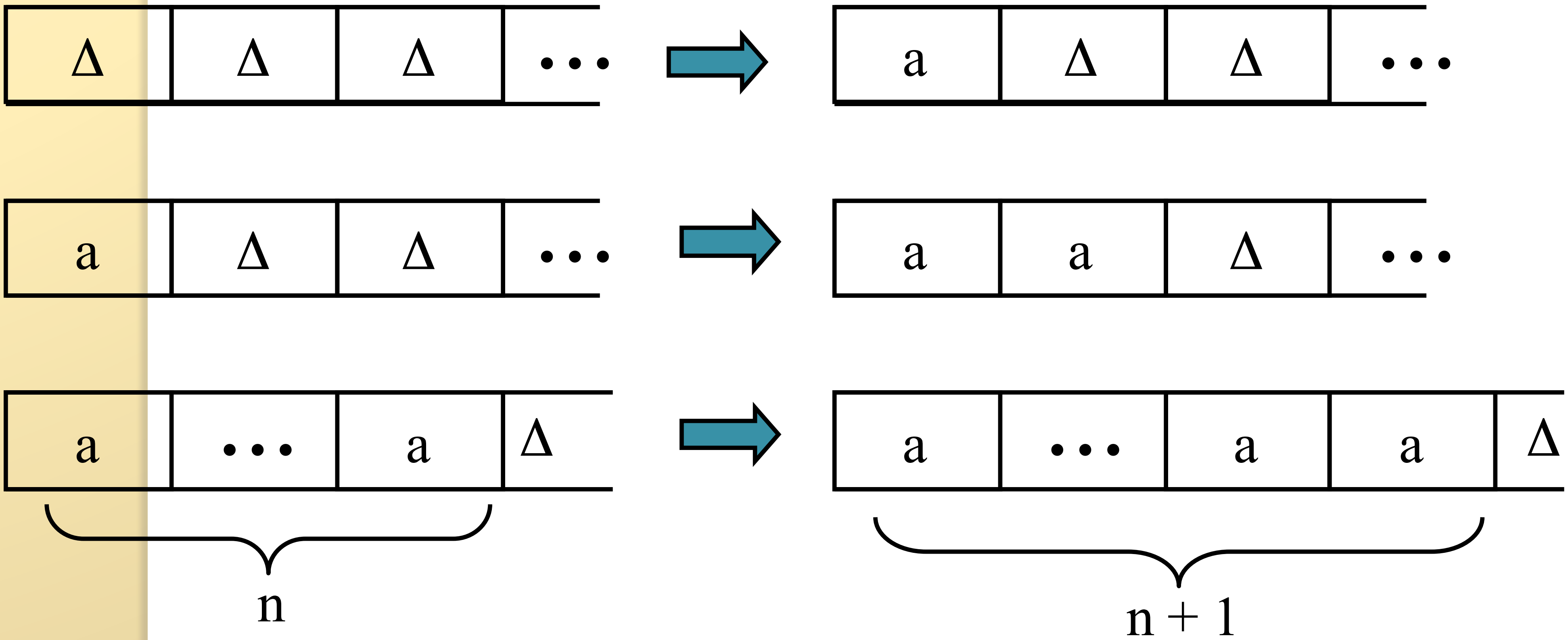
# Unary Code

| | | |
|---|---|---|
| 0 | Δ | Δ |
| 1 | a | a |
| 2 | aa | $a^2$ |
| 3 | aaa | $a^3$ |
| 4 | aaaa | $a^4$ |
| 5 | aaaaa | $a^5$ |
| 6 | aaaaaa | $a^6$ |
| 7 | aaaaaaa | $a^7$ |
| : | : | : |

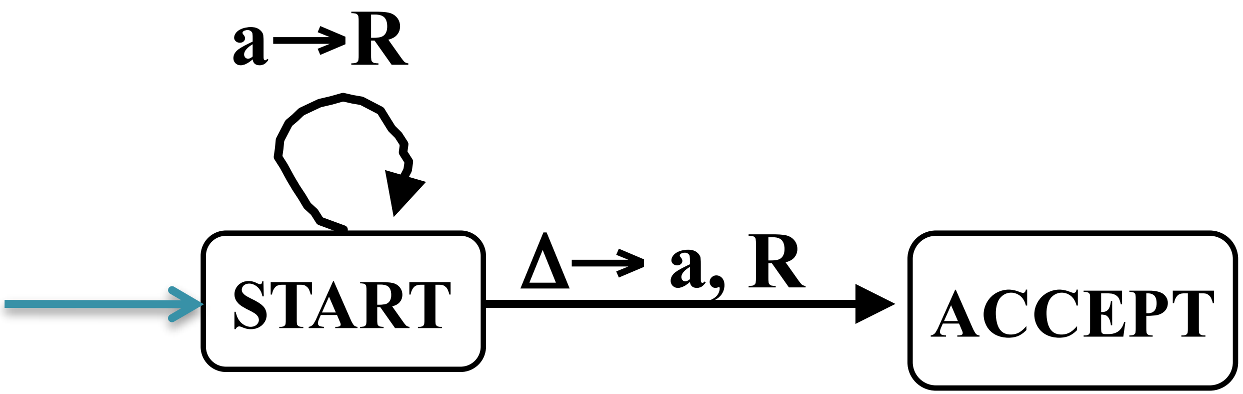
# Successor

Using the unary code for natural numbers build a Turing Machine that represents the function

**f(n) = n + 1**.



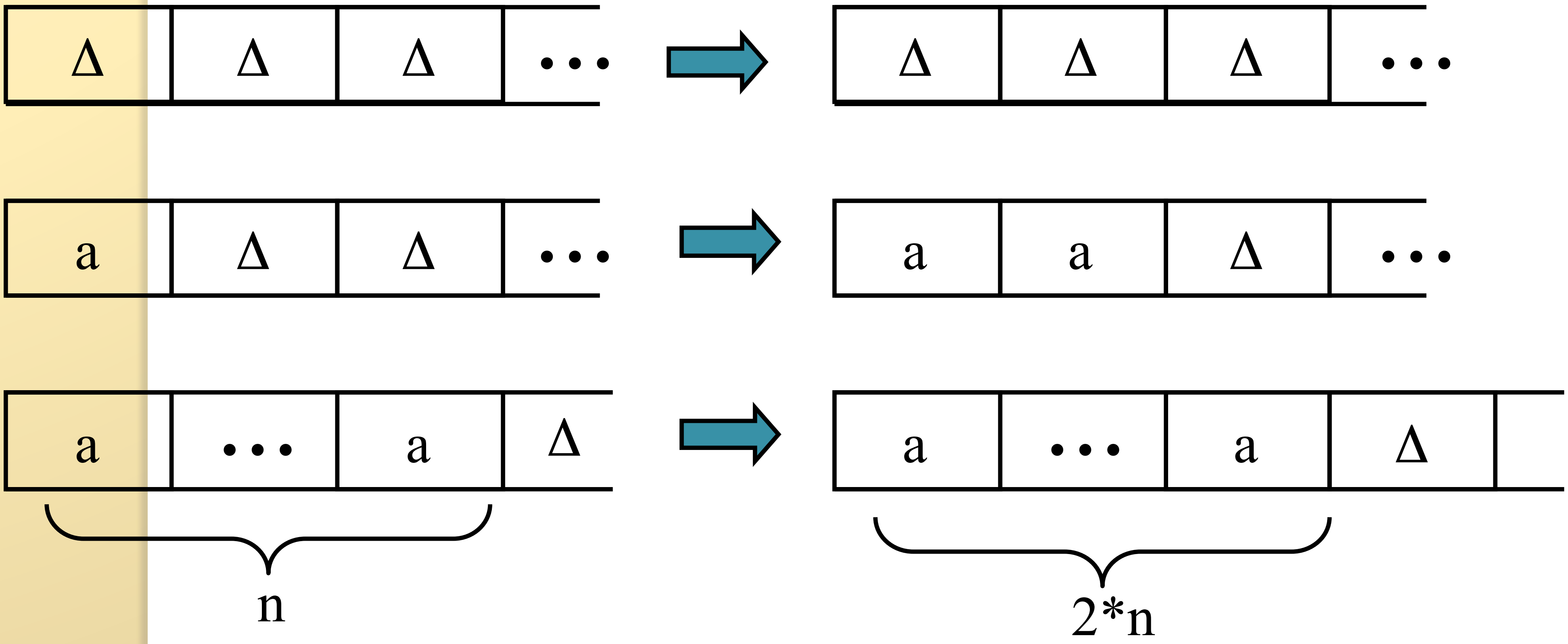
# Successor



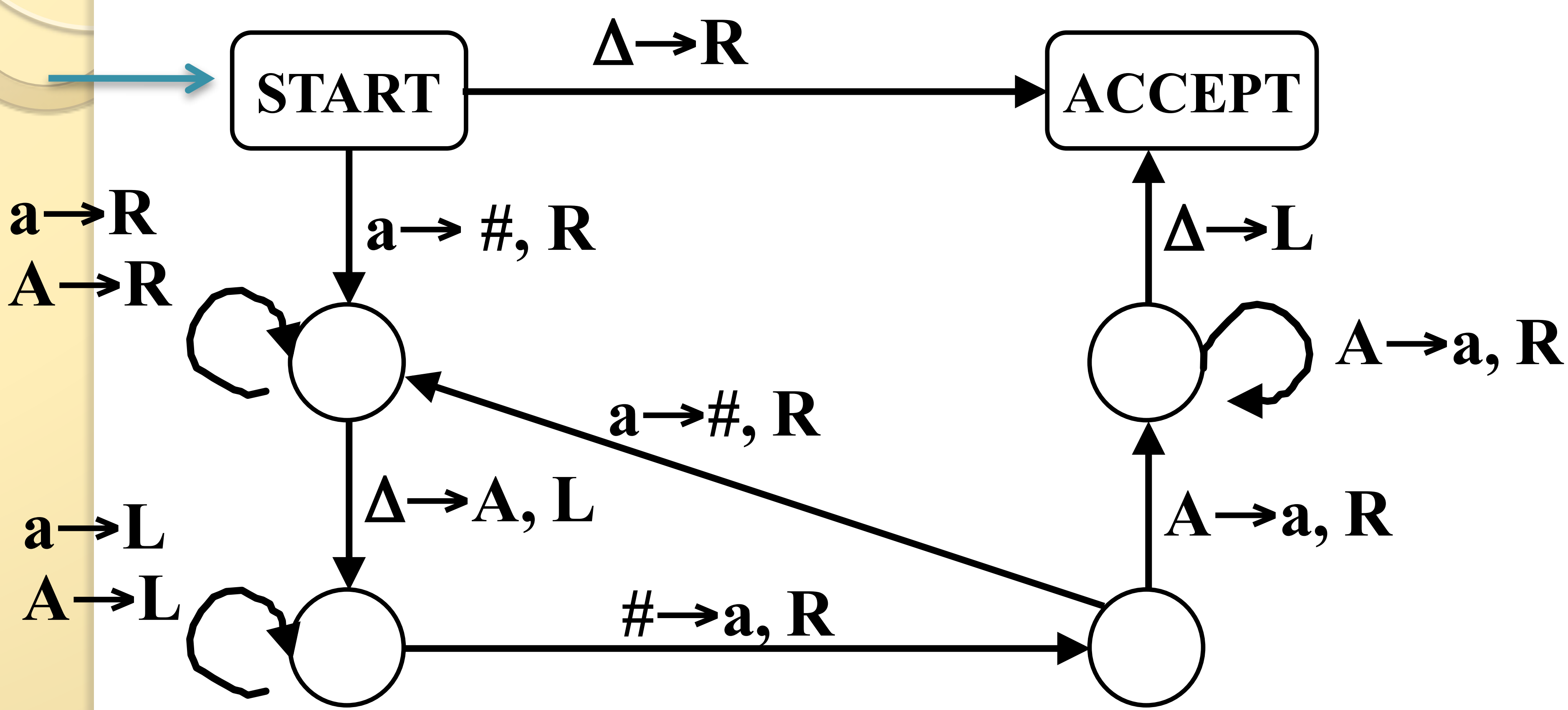
# Double

Using the unary code for natural numbers build a Turing Machine that represents the function

**f(n) = 2*n.**



# Double



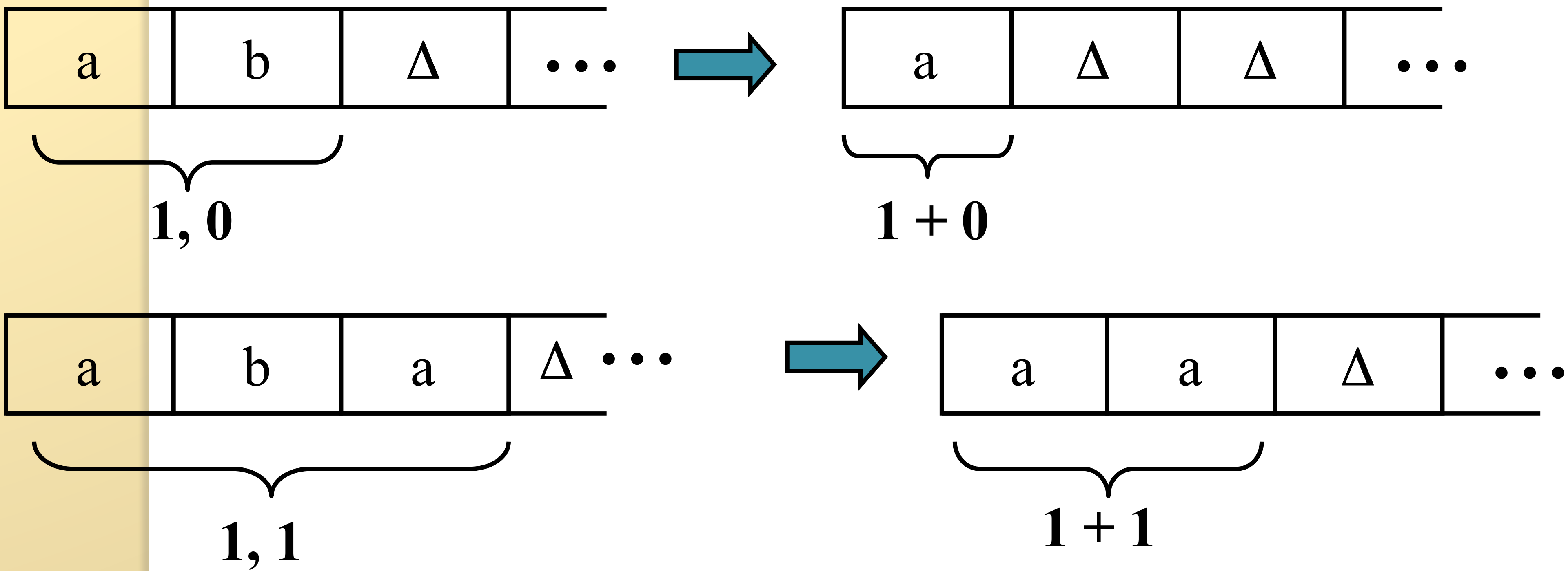
# Unary Code for Tuples of Integers

- Tuples of natural numbers
- Example: **1, 0, 2, 3**
- Encoding:
  - Each integer is coded using the unary code as a string of **a**’s
  - Integers are separated by a **b**.
- Example: **abbaabaaa**

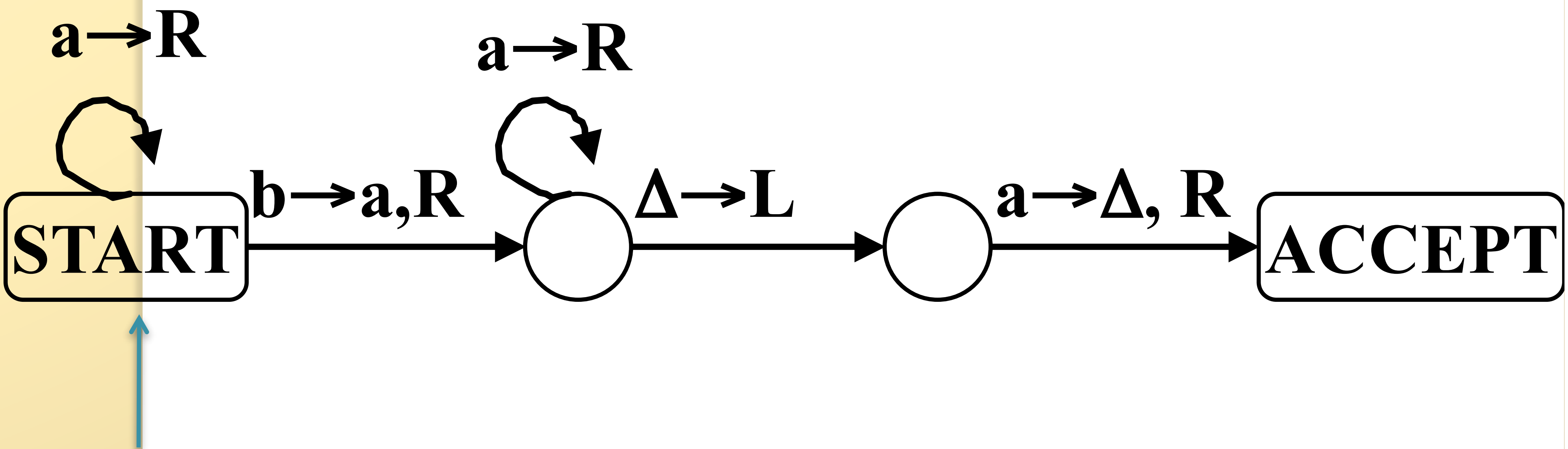
# Addition

Using the unary code of natural numbers build a Turing Machine that represents the function

**f(n, m) = n+m.**



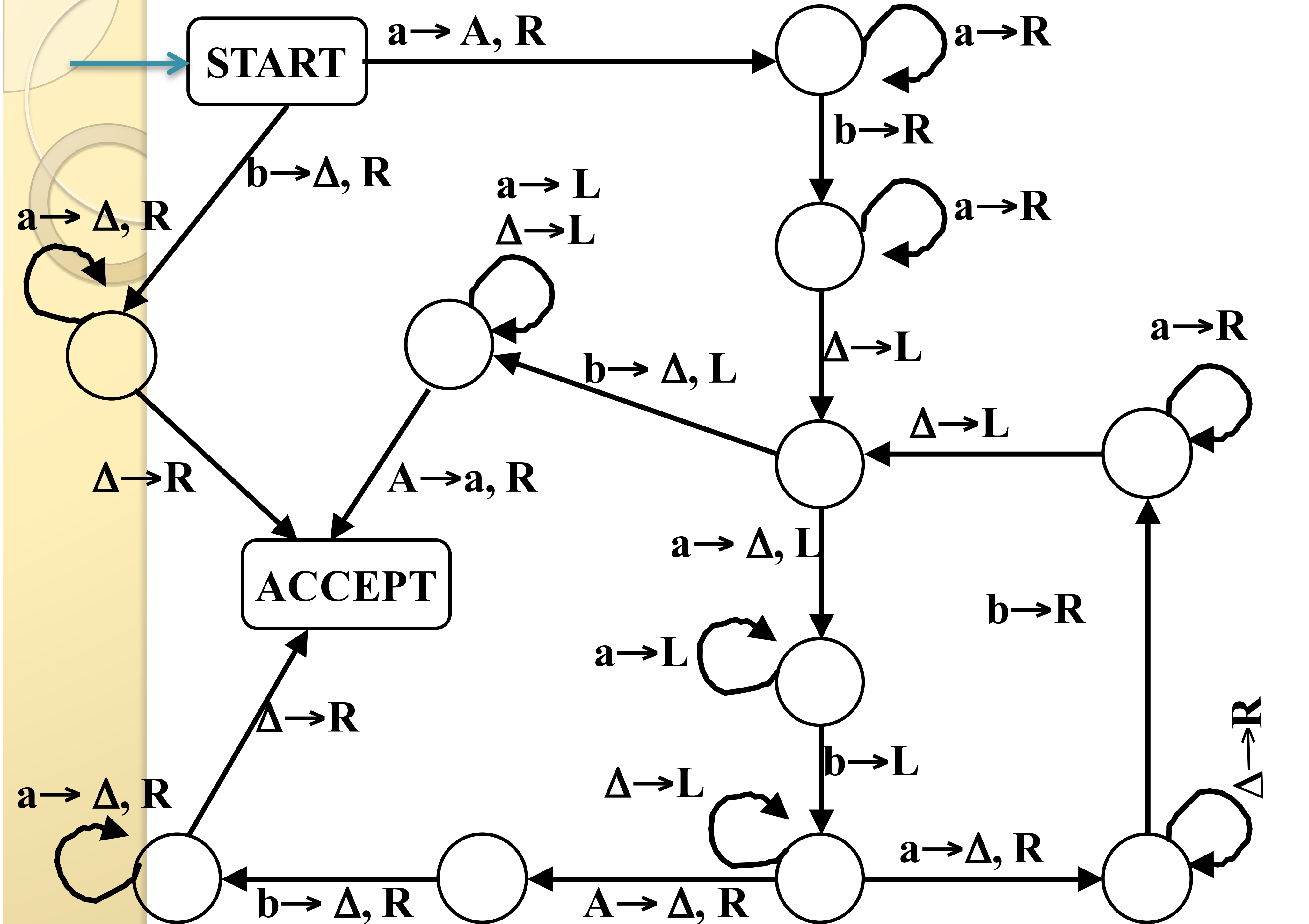
# Addition



# Monus

Using the unary code of tuples of natural numbers build a Turing Machine that represents the function:

$$f(n,m) = \begin{cases} n - m, & n \geq m \\ 0, & \text{Otherwise} \end{cases}$$



START
a→ A, R
a→R
b→R
a→R
b→Δ, R
a→ Δ, R
a→ L
Δ→L
Δ→L
a→R
b→ Δ, L
Δ→L
Δ→R
A→a, R
a→ Δ, L
ACCEPT
b→R
a→L
Δ→R
b→L
Δ→L
Δ→R
a→ Δ, R
a→Δ, R
b→ Δ, R
A→ Δ, R

# Definition

A function is **computable** if it can be represented as:

- A Turing Machine
- Input
  - sequences of natural numbers
- Output
  - one natural number

# Variations on Turing machines

- Direction:  stay still, as well as Left/Right
- Tapes:
  - two-way infinite
  - multiple tapes
  - separate input, output, work tapes
  - "tapes" of 2 or more dimensions
- ……

*Same class of computable functions*

# Other approaches to computability

- recursive function theory
  - starting with Kurt Gödel, 1931
- lambda calculus
  - Alonzo Church, 1936
- Turing machines
  - Alan Turing, 1936-37

*Same class of computable functions*

Kurt Gödel (1906-1978)

http://www-history.mcs.st-andrews.ac.uk/Biographies/Godel.html

Alonzo Church (1903-1995)

http://www-history.mcs.st-andrews.ac.uk/Biographies/Church.html

# Church-Turing Thesis

Any function which can defined by an algorithm can be represented by a Turing Machine.

Note: not a Theorem! But widely accepted.

Evidence:

- different approaches to computability end up in agreement
- long experience, that algorithms can be implemented as programs, and therefore on Turing machines
- no known counterexamples, i.e., no algorithms which seem to be unimplementable

# Alan Turing


Alan Turing Centenary Year (2012) website:
http://www.turingcentenary.eu/

B. Jack Copeland, *Turing: Pioneer of the Information Age*, OUP, 2013.
Andrew Hodges, *Alan Turing: The Enigma*, Vintage, London, 1983.
Andrew Hodges, *Turing*, Phoenix, London, 1997.

Turing bibliography: http://www.turing.org.uk/sources/biblio.html

G. Farr, Calls for a posthumous pardon … but who was Alan Turing?, *The Conversation*, 22 Dec 2011,
https://theconversation.com/calls-for-a-posthumous-pardon-but-who-was-alan-turing-4773

G. Farr, The Imitation Game: is it history, drama or myth?, *The Conversation*, 9 Jan 2015,
https://theconversation.com/the-imitation-game-is-it-history-drama-or-myth-35849

# Revision

- Know what a Turing Machine is, and how to use one.
- Be able to convert a Finite Automaton into a TM.
- Be able to build a Turing Machine to define a language.
- Know the unary code for natural numbers, and tuples
- Know what a computable function is, and how to define one using a TM.
- Know and understand the Church-Turing Thesis.

## Turing machine software (see Moodle)

- Tuatara (graphical environment)

## Reading:

- Sipser, Ch 3: Section 3.1, pp. 165-176, 181-190.

## Preparation:

- Sipser, Ch 3, start & end of Section 3.2; Section 3.3