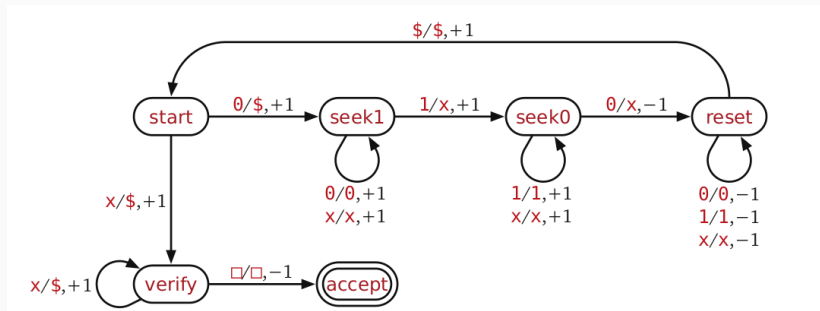




# Pre-lecture brain teaser

You have the following Turing machine diagram that accepts a particular language whose alphabet  $\Sigma = \{0, 1\}$ . Please describe the language.



# ECE-374-B: Lecture 9 - Universal Turing Machines

---

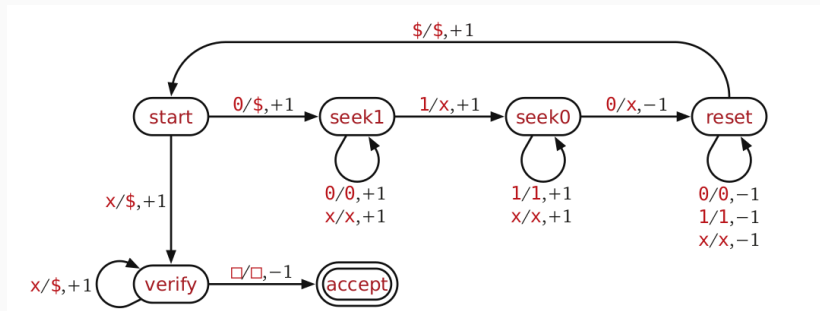
Instructor: Nickvash Kani

February 14, 2023

University of Illinois at Urbana-Champaign

## Pre-lecture brain teaser

You have the following Turing machine diagram that accepts a particular language whose alphabet  $\Sigma = \{0, 1\}$ . Please describe the language.



## Pre-lecture brain teaser - code

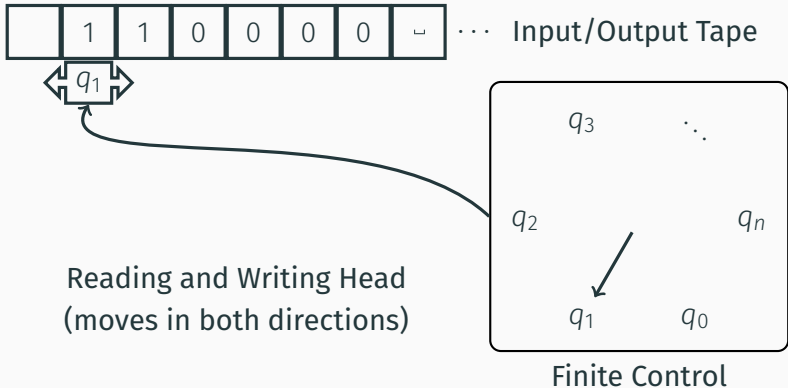
Can simulate TM on `turingmachine.io` using the following code:

```
start state: start
table:
start:
    # Inductive case: start with the same symbol.
    0: {write: '$', R: seek1}
    # Base case: empty string.
    'x': {write: '$', R: verify}
seek1:
    [0,'x']: R
    1: {write: 'x', R: seek0}
seek0:
    [1,'x']: R
    0: {write: 'x', L: reset}
reset:
    [0,1,'x']: L
    '$': {R: start}
verify:
    x: {write: '$', R}
    ' ': {L: accept}
accept:
```

## Turing machine recap

---

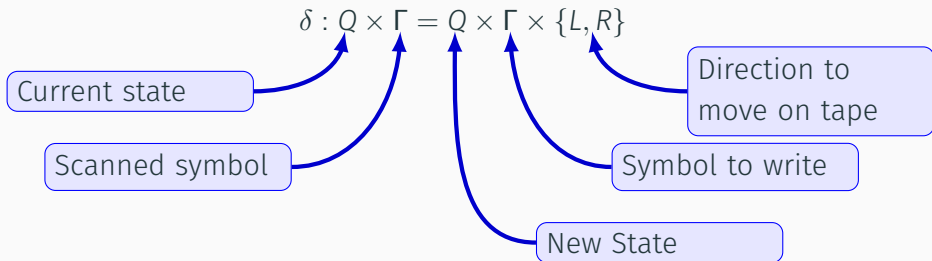
# Turing machine



- Input written on (infinite) one sided tape.
- Special blank characters.
- Finite state control (similar to DFA).
- Every step: Read character under head, write character out, move the head right or left (or stay).

# Transition function

## Transition Function



$\delta(q, a) = (p, b, L)$  means  
from state  $q$ , on reading  $a$ :

- go to state  $p$
- write  $b$
- move head **Left**



# Turing machine variants

---

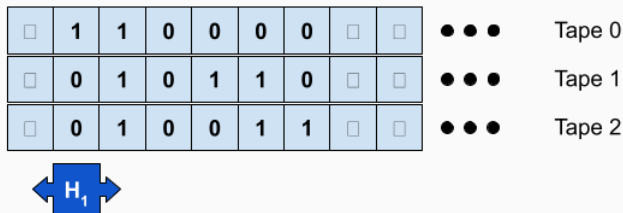
# Equivalent Turing Machines

Several variations of a Turing machine:

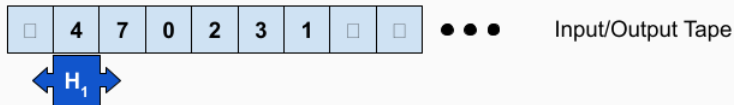
- Standard Turing machine (single infinite tape)
- Multi-track tapes
- Doubly-Infinite Tape
- Multiple heads
- Multiple heads and tapes

# Multi-track Tapes

Suppose we have a TM with multiple tracks:



Is there an equivalent single-track TM?

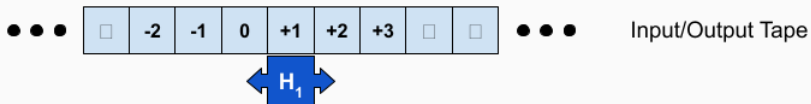


New transition function:

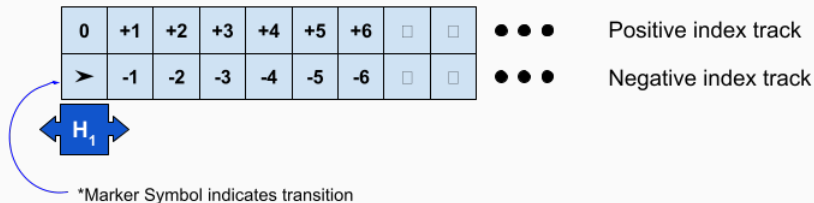
$$\delta : Q \times \Gamma_1 \times \Gamma_2 \times \Gamma_3 \rightarrow Q \times \Gamma_1 \times \Gamma_2 \times \Gamma_3 \times \{-1, +1\}$$

# Infinite Bi-directional Tape

Suppose we have a TM with multiple tracks:



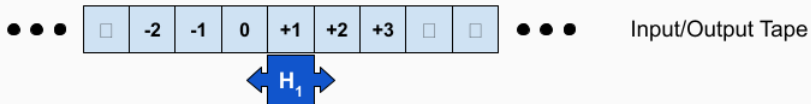
Is there an equivalent single-track TM?



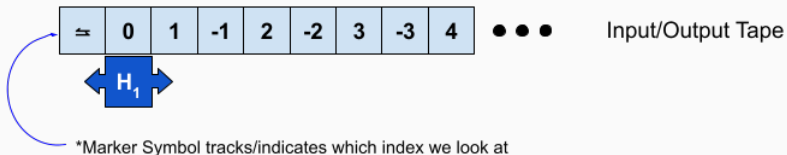
Can model as multiple tapes.

# Infinite Bi-directional Tape

Suppose we have a TM with a bidirectional tape:



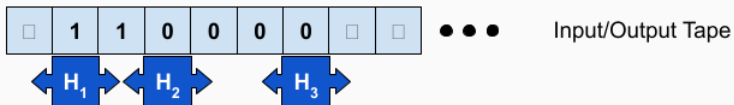
Is there an equivalent single-track TM?



Or as single tape interleaved with positive and negative indexes.

# Multiple Read/Write Heads

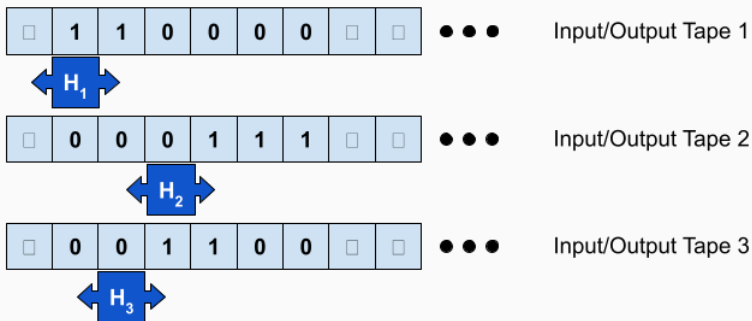
Suppose we have a TM with multiple heads:



What does the transition function for the equivalent nominal TM look like?

# Multiple Read/Write Heads

Suppose we have a TM with multiple heads and tracks:



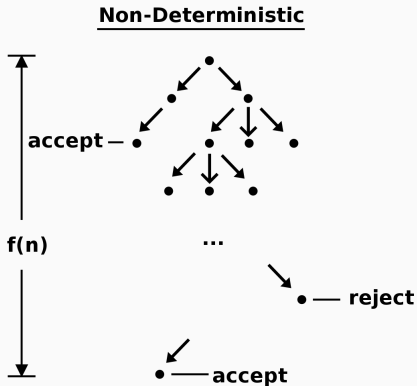
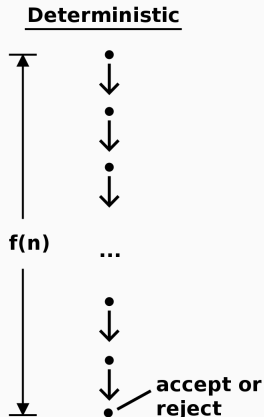
What does the transition function for the equivalent nominal TM look like?

# Determinism in Turing machines

---



# Remember Non-determinism?



# Non-deterministic Turing machine?

What does a non-deterministic Turing machine look like?

# Non-deterministic Turing machine?

What does a non-deterministic Turing machine look like?

Is a **NTM** more powerful than a **DTM**?

No. A DTM can simulate a NTM in the following ways:

- **Multiplicity of configuration of states**
  1. Have the store multiple configurations of the NTM.
  2. At every timestep, process each configuration. Add configurations to the set if multiple paths exist.
- **Multiple Tapes** - Can simulate NTM with 3-tape DTM:
  1. First tape holds original input
  2. Second used to simulate a particular computation of NTM
  3. Third tape encodes path in NTM computation tree.

Effectively this is a breadth-first search of non-deterministic computation tree.

# Savitch's Theorem

Proved by Walter Savitch in 1970, states that for any function  $f \in \Omega(\log(n))$ :

$$\text{NSPACE}(f(n)) \subseteq \text{DSPACE}(f(n)^2)$$

## Lemma

*If a **NTM** can solve a problem using  $f(n)$  space, a **DTM** can solve the same problem in the square of that space bound.*

$\implies$  Even though non-determinism significantly reduces time to solve problem, it reduces space requirements far less!

# Universal Turing Machine

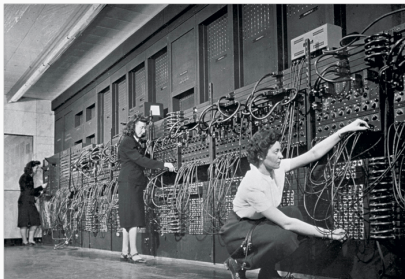
---

# Special Purpose Machines?

We've seen that you need different DFAs for different languages.

We've seen that you need different TMs for different languages.

Early computers were no different.



# Universal Turing Machine

A single TM  $M_u$  that can compute anything computable!

Takes as input:

- the description of some other TM  $M$
- data  $w$  for  $M$  to run on

Outputs:

- results of running  $M(w)$



# Coding of TMs

Show how to represent every *TM* as a natural number

## Lemma

*If  $L$  over alphabet  $\{0, 1\}$  is accepted by some *TM*  $M$ , then there is a one-tape *TM*  $M$  that accepts  $L$ , such that*

- $\Gamma = \{0, 1, B\}$
- *states numbered  $1, \dots, k$*
- *$q_1$  is a unique start state*
- *$q_2$  is a unique halt/accept state*
- *$q_3$  is a unique halt/reject state*

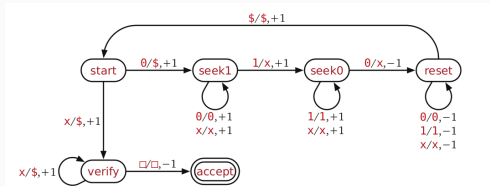
So to represent a *TM*, we need only list its set of transitions - everything else is implicit by the above.

# Encoding Alphabet

Consider the TM that recognizes the language  
 $L = \{0^n 1^n 0^n | n \geq 0\}$  with the state diagram shown below:

Input encoding:

- $\langle 0 \rangle = 001$
- $\langle 1 \rangle = 010$
- $\langle \$ \rangle = 011$
- $\langle x \rangle = 100$
- $\langle \_ \rangle = 000$



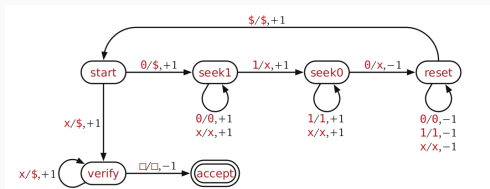
Example:  $\langle 001100 \rangle = [001 \cdot 001 \cdot 010 \cdot 010 \cdot 001 \cdot 001]$   
(Putting  $\cdot$  separators for the sake of legibility)

# Encoding states

Consider the TM that recognizes the language  
 $L = \{0^n 1^n 0^n | n \geq 0\}$  with the state diagram shown below:

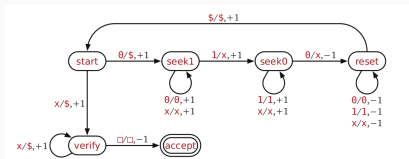
State encoding:

- $\langle \text{start} \rangle = 001$
- $\langle \text{seek1} \rangle = 010$
- $\langle \text{seek0} \rangle = 011$
- $\langle \text{reset} \rangle = 100$
- $\langle \text{verify} \rangle = 101$
- $\langle \text{accept} \rangle = 110$
- $\langle \text{reject} \rangle = 000$



# Encoding States and Alphabet

Consider the TM that recognizes the language  $L = \{0^n 1^n 0^n | n \geq 0\}$  with the state diagram shown below:

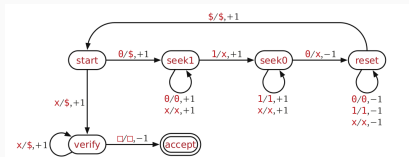


Now we need to encode a transition. Last thing we'll need is to encode the movement of the head which we'll describe as:  $[\text{left}, \text{right}] = [0, 1]$ .

Example: How do we encode:  $\delta(\text{reset}, \$) = (\text{start}, \$, \text{right})$

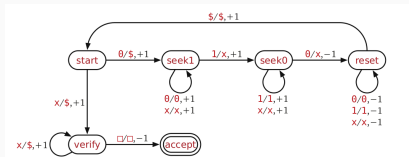
Answer:  $[100 \cdot 011 | 001 \cdot 011 \cdot 1]$

# Encoding machine through transitions



$$\delta^M = \begin{aligned} & [[001 \cdot 001 | 010 \cdot 011 \cdot 1] [001 \cdot 100 | 101 \cdot 011 \cdot 1] \\ & [010 \cdot 001 | 010 \cdot 001 \cdot 1] [010 \cdot 100 | 010 \cdot 100 \cdot 1] \\ & [010 \cdot 010 | 011 \cdot 100 \cdot 1] [011 \cdot 010 | 011 \cdot 010 \cdot 1] \\ & [011 \cdot 100 | 011 \cdot 100 \cdot 1] [011 \cdot 001 | 100 \cdot 100 \cdot 1] \\ & [100 \cdot 001 | 100 \cdot 001 \cdot 0] [100 \cdot 010 | 100 \cdot 010 \cdot 0] \\ & [100 \cdot 100 | 100 \cdot 100 \cdot 0] [100 \cdot 011 | 001 \cdot 011 \cdot 1] \\ & [101 \cdot 100 | 101 \cdot 011 \cdot 1] [101 \cdot 000 | 110 \cdot 000 \cdot 0]] \end{aligned}$$

# Encoding machine through transitions



$$\delta^M = \begin{bmatrix} [001 \cdot 001 | 010 \cdot 011 \cdot 1] & [001 \cdot 100 | 101 \cdot 011 \cdot 1] \\ [010 \cdot 001 | 010 \cdot 001 \cdot 1] & [010 \cdot 100 | 010 \cdot 100 \cdot 1] \\ [010 \cdot 010 | 011 \cdot 100 \cdot 1] & [011 \cdot 010 | 011 \cdot 010 \cdot 1] \\ [011 \cdot 100 | 011 \cdot 100 \cdot 1] & [011 \cdot 001 | 100 \cdot 100 \cdot 1] \\ [100 \cdot 001 | 100 \cdot 001 \cdot 0] & [100 \cdot 010 | 100 \cdot 010 \cdot 0] \\ [100 \cdot 100 | 100 \cdot 100 \cdot 0] & [100 \cdot 011 | 001 \cdot 011 \cdot 1] \\ [101 \cdot 100 | 101 \cdot 011 \cdot 1] & [101 \cdot 000 | 110 \cdot 000 \cdot 0] \end{bmatrix}$$

$$\delta(\text{seek0}, x) = (\text{seek0}, x, \text{right})$$

## Encoding initial state

Ok so now we've encoded the Turing machine ( $M$ ) into a string, how do we make a machine  $M_u(M, w)$  which accepts if  $M(w)$  accepts, and rejects if  $M(w)$  rejects?

## Encoding initial state

Ok so now we've encoded the Turing machine ( $M$ ) into a string, how do we make a machine  $M_u(M, w)$  which accepts if  $M(w)$  accepts, and rejects if  $M(w)$  rejects?

Let's start with the encoding of  $w$  (let's say  $w = 001100$ ):

$$\langle 001100 \rangle = [001 \cdot 001 \cdot 010 \cdot 010 \cdot 001 \cdot 001]$$



## Encoding initial state

Ok so now we've encoded the Turing machine ( $M$ ) into a string, how do we make a machine  $M_u(M, w)$  which accepts if  $M(w)$  accepts, and rejects if  $M(w)$  rejects?

Let's start with the encoding of  $w$  (let's say  $w = 001100$ ):

$$\langle 001100 \rangle = [001 \cdot 001 \cdot 010 \cdot 010 \cdot 001 \cdot 001]$$

Now let's add spaces next to each character so we can mark where  $M$ 's head is:

$$[[000 \cdot 001][000 \cdot 001][000 \cdot 010][000 \cdot 010][000 \cdot 001][000 \cdot 001]]$$

# Encoding states

Padding used to mark state.

In the beginning,  $q = \langle \text{start} \rangle = 001$  so our machine tapes initial string is:

$[[\underline{001} \cdot 001][000 \cdot 001][000 \cdot 010][000 \cdot 010][000 \cdot 001][000 \cdot 001]]$

Similarly intermediate configuration

$M = \langle \text{state}, \text{tape string}, \text{head position} \rangle = (\text{seek1}, \$0x1x0, 3)$

would be marked as:

$[[000 \cdot 011][000 \cdot 001][000 \cdot 100][010 \cdot 010][000 \cdot 100][000 \cdot 001]]$   
 $\underbrace{\hspace{1.5cm}}_{\text{reject \$}} \quad \underbrace{\hspace{1.5cm}}_{\text{reject 0}} \quad \underbrace{\hspace{1.5cm}}_{\text{reject x}} \quad \underbrace{\hspace{1.5cm}}_{\text{seek1 1}} \quad \underbrace{\hspace{1.5cm}}_{\text{reject x}} \quad \underbrace{\hspace{1.5cm}}_{\text{reject 0}}$

# The universal Turing machine

---

# UTM introduction

Now that we are able to encode Turing machines, we want to construct a Turing machine such that:

$$L(M_u) = \{\langle M \rangle \# w \mid M \text{ accepts } w\}$$

$M_u$  is a stored-program computer. It reads  $\langle M \rangle$  and executes it on data  $w$ .

$M_u$  simulates the run of  $M$  on  $w$ .

# Encodings

$M$ : Turing machine

$\langle M \rangle$ : a string uniquely describing  $M$  (i.e., it is a number).

$w$ : An input string.

$\langle M, w \rangle$ : A unique string encoding both  $M$  and input  $w$ .

$$L(M_u) = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ accepts } w \}.$$

## $M_u$ Operational concept

We assume without a loss of generality that our universal turing machine ( $M_u$ ) has two tapes and two heads:

- **Input tape:** which stores the encoding of  $\langle M \rangle = \langle \text{state, tape input, head position} \rangle$
- **Machine tape:** Encoding tape which stores  $M$ 's encoding

**General Idea:** For any given configuration of  $M$ , our  $M_u$  will.

- Starting from leftmost of input tape, scan tape for first state which is not  $\langle \text{reject} \rangle$
- $M_u$  scans machine tape for the transition function that matches the substring found in the input tape.
- Based on transition function,  $M_u$  writes the right half of this transition function into the current input tape cell.
- Based on head direction of the transition function,  $M_u$  moves the current state left or right

# Simulation example I

Let's start with the configuration:  $M = (\text{seek1}, \$\$x1x0, 3)$ :

- Input-Tape =

$\triangle$   $[[000 \cdot 011][000 \cdot 011][000 \cdot 100][010 \cdot 010][000 \cdot 100][000 \cdot 001]]$

- Machine-Tape =  $\delta^M =$

$\triangle$   $[[001 \cdot 001|010 \cdot 011 \cdot 1][001 \cdot 100|101 \cdot 011 \cdot 1][010 \cdot 001| \dots]$

First  $M_u$  searchers for none reject state:

- Input-Tape =

$[[000 \cdot 011][000 \cdot 011][000 \cdot 100][010 \cdot 010]_{\triangle}[000 \cdot 100][000 \cdot 001]]$

- Machine-Tape =  $\delta^M =$

$\triangle$   $[[001 \cdot 001|010 \cdot 011 \cdot 1][001 \cdot 100|101 \cdot 011 \cdot 1][010 \cdot 001| \dots]$

## Simulation example II

- Input-Tape =  

$$[[000 \cdot 011][000 \cdot 011][000 \cdot 100][010 \cdot 010][000 \cdot 100][000 \cdot 001]]$$

$$\triangle$$
- Machine-Tape =  $\delta^M =$   

$$\triangle [ [001 \cdot 001 | 010 \cdot 011 \cdot 1][001 \cdot 100 | 101 \cdot 011 \cdot 1][010 \cdot 001 | \dots$$

Then  $M_u$  searches for transition whose left side matches the input cell:

- Input-Tape =  

$$[[000 \cdot 011][000 \cdot 011][000 \cdot 100][010 \cdot 010][000 \cdot 100][000 \cdot 001]]$$

$$\triangle$$
- Machine-Tape =  $\delta^M =$   

$$\dots 100 \cdot 1][010 \cdot 010 | 011 \cdot 100 \cdot 1][011 \cdot 010 | 011 \cdot 010 \cdot 1] \dots$$

$$\triangle$$



## Simulation example III

- Input-Tape =  

$$[[000 \cdot 011][000 \cdot 011][000 \cdot 100][010 \cdot 010]_{\triangle}[000 \cdot 100][000 \cdot 001]]$$
- Machine-Tape =  $\delta^M =$   

$$\dots 100 \cdot 1][010 \cdot 010|011 \cdot 100 \cdot 1][011 \cdot 010|011 \cdot 010 \cdot 1] \dots$$

Then  $M_u$  copies the right side of the transition function into the input tape:

- Input-Tape =  

$$[[000 \cdot 011][000 \cdot 011][000 \cdot 100][011 \cdot 100]_{\triangle}[000 \cdot 100][000 \cdot 001]]$$
- Machine-Tape =  $\delta^M =$   

$$\dots 100 \cdot 1][010 \cdot 010|011 \cdot 100 \cdot 1][011 \cdot 010|011 \cdot 010 \cdot 1] \dots$$

## Simulation example IV

- Input-Tape =  

$$[[000 \cdot 011][000 \cdot 011][000 \cdot 100][011 \cdot 100] \underset{\triangle}{[000 \cdot 100]}[000 \cdot 001]]$$
- Machine-Tape =  $\delta^M =$   

$$\dots 100 \cdot 1][010 \cdot 010|011 \cdot 100 \cdot 1] \underset{\triangle}{[011 \cdot 010|011 \cdot 010 \cdot 1]} \dots$$

Then  $M_u$  move the state of the configuration according to the transition function:

- Input-Tape =  

$$[[000 \cdot 011][000 \cdot 011][000 \cdot 100][000 \cdot 100][011 \underset{\triangle}{\cdot} 100][000 \cdot 001]]$$
- Machine-Tape =  $\delta^M =$   

$$\dots 100 \cdot 1][010 \cdot 010|011 \cdot 100 \cdot 1] \underset{\triangle}{[011 \cdot 010|011 \cdot 010 \cdot 1]} \dots$$

## Simulation example V

- Input-Tape =  

$$[[000 \cdot 011][000 \cdot 011][000 \cdot 100][000 \cdot 100][011 \cdot 100][000 \cdot 001]]$$
- Machine-Tape =  $\delta^M =$   

$$\dots 100 \cdot 1][010 \cdot 010|011 \cdot 100 \cdot 1][011 \cdot 010|011 \cdot 010 \cdot 1] \dots$$

Then we reset:

- Input-Tape =  

$$\underset{\Delta}{[ [000 \cdot 011][000 \cdot 011][000 \cdot 100][000 \cdot 100][011 \cdot 100][000 \cdot 001] ]}$$
- Machine-Tape =  $\delta^M =$   

$$\underset{\Delta}{[ [001 \cdot 001|010 \cdot 011 \cdot 1][001 \cdot 100|101 \cdot 011 \cdot 1][010 \cdot 001|\dots] ]}$$

## What does this show?

- Every TM is encoded by a unique element of  $N$  (where  $N$  is a natural number)
- **Convention:** elements of  $N$  that do not correspond to any TM encoding represent the “null TM” that accepts nothing.
- Thus, every TM is a number, and vice versa
- Let  $\langle M \rangle$  mean the number that encodes  $M$ . Conversely, let  $M_n$  be the TM with encoding  $n$ .

**Big Idea:** Every TM can be represent by a number (strings of 0's and 1's) and there exists a universal TM,  $M_u$ , that can simulate any other TM.