Chapter 3

3.2 VARIANTS OF TURING MACHINES

VARIANTS OF TURING MACHINES

- Alternative definitions of Turing machines abound,
  - including versions with **multiple tapes** or with **nondeterminism**.
  - Called variants of the Turing machine model.

VARIANTS OF TURING MACHINES

- Alternative definitions of Turing machines abound,
  - including versions with **multiple tapes** or with **nondeterminism**.
  - Called variants of the Turing machine model.

- The **original** model and its reasonable **variants**
  - have the same power
  - recognize the same class of languages.

VARIANTS OF TURING MACHINES

- Example: Suppose that we had allowed the Turing machine the ability to stay put.
  - The **transition function** would then have the form

$$\delta: Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R, \boldsymbol{S}\}.$$

- Any additional power?
- Any additional languages can be recognized with this feature?

VARIANTS OF TURING MACHINES

- Suppose that we had allowed the Turing machine the ability to stay put.
  - The transition function would then have the form

$$\delta: Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R, S\}.$$

- Any additional power?
- Any additional languages can be recognized with this feature?

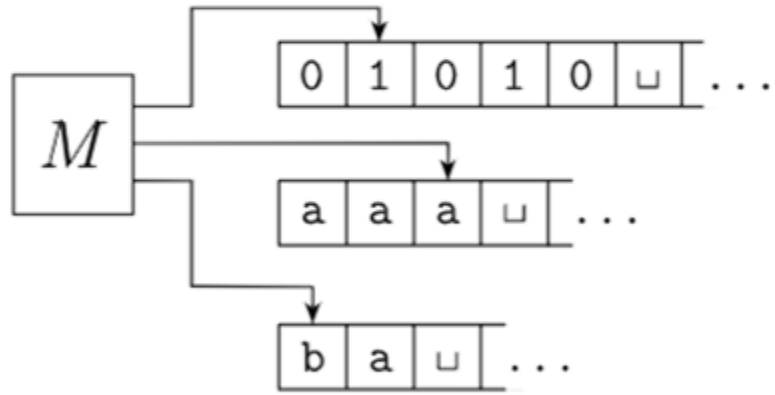
- **Of course not**.
  - because we can convert any **TM with the "stay put"** feature to **one that does not have it**.
    - by replacing each stay put transition **with two transitions**:
      - move to the right and
      - back to the left.

VARIANTS OF TURING MACHINES

- To show that two **models are equivalent**, we simply need to show that
  - one can simulate the other.

MULTITAPE TURING MACHINES

- A **multi tape Turing machine** is like an ordinary Turing machine with **several tapes.**
- **Each tape** has its **own head** for reading and writing.
- **Initially** the input appears on **tape 1**, and the **others** will be blank



An example of a TM with 3 tapes

MULTITAPE TURING MACHINES

- The **transition function** is changed to allow for reading, writing, and moving the heads on some or all of the tapes simultaneously.

$$\delta: Q \times \Gamma^k \longrightarrow Q \times \Gamma^k \times \{\mathrm{L}, \mathrm{R}, \mathrm{S}\}^k,$$

Where k is the **number of tapes**.

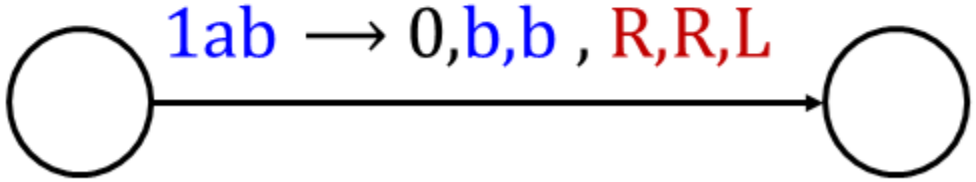
MULTITAPE TURING MACHINES

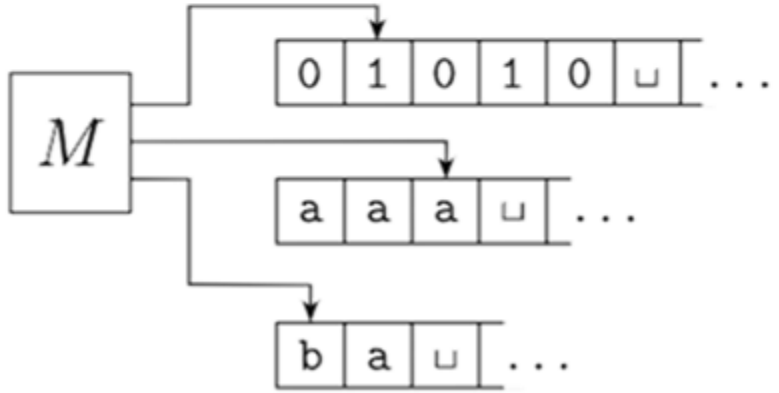
- The **transition function** is changed to allow for reading, writing, and moving the heads on some or all of the tapes simultaneously.

$$\delta: Q \times \Gamma^k \longrightarrow Q \times \Gamma^k \times \{\mathrm{L}, \mathrm{R}, \mathrm{S}\}^k,$$

Where k is the **number of tapes**.

$$\delta(q_i, a_1, \ldots, a_k) = (q_j, b_1, \ldots, b_k, \mathrm{L}, \mathrm{R}, \ldots, \mathrm{L})$$





MULTITAPE TURING MACHINES

- Theorem : Every multitape Turing machine has an equivalent single-tape Turing

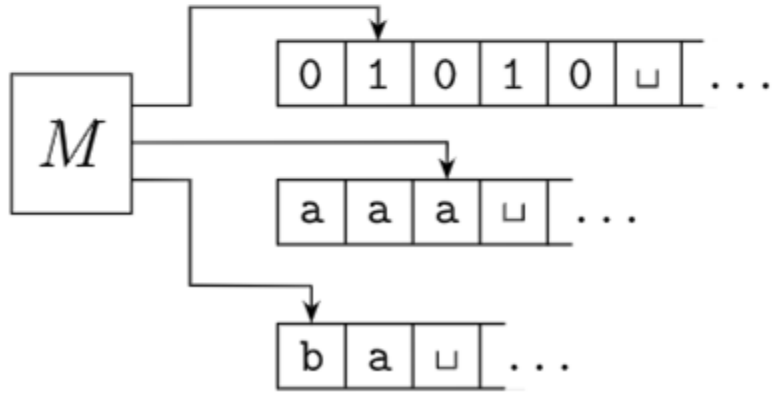
MULTITAPE TURING MACHINES

- Theorem : Every **multitape Turing machine** has an equivalent **single-tape Turing**

MULTITAPE TURING MACHINES

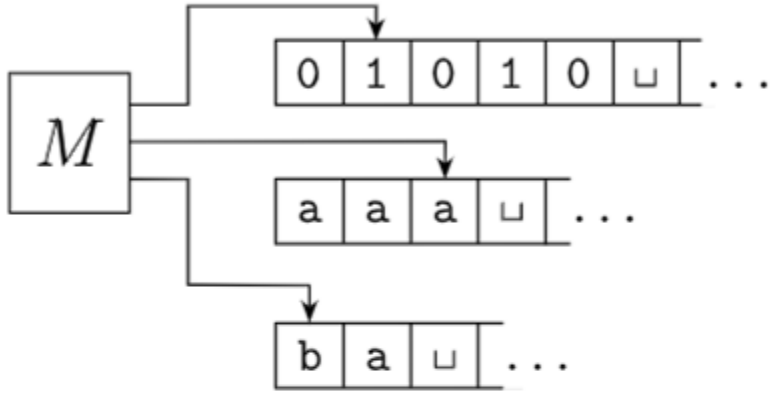
- Theorem : Every **multitape Turing machine** has an equivalent **single-tape Turing**

- Proof Idea:
  - We show how to convert a **multitape** $TM\ M$ to an equivalent **single-tape** $TM\ S$.



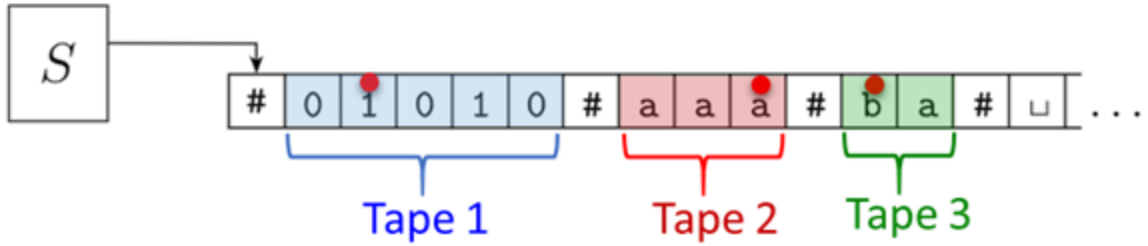
MULTITAPE TURING MACHINES

- Proof Idea(cont.):
  - Say that M has **k tapes**.
  - Then S simulates the effect of k tapes by
    - storing their information on its single tape.



  - It uses the **new symbol** # as a delimiter
    - to **separate** the contents of the different tapes.

  - In addition to the contents of these tapes,
    - S must keep track of the locations of the heads.
      - by writing a tape symbol with a dot above it
        - to mark the place where the head on that tape would be.



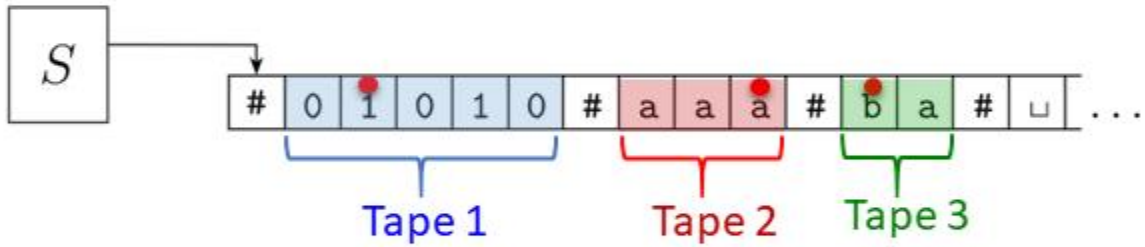
MULTITAPE TURING MACHINES

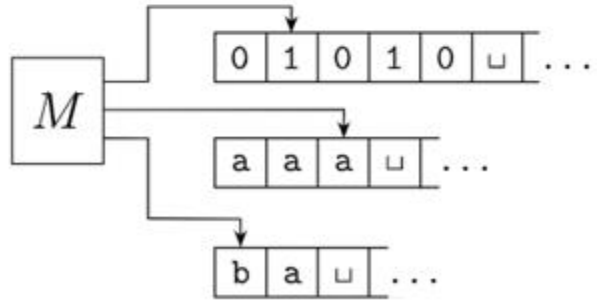
- **Proof :**

$S =$ "On input $w = w_1 \cdots w_n$:

1. First S puts its tape into the format that represents all k tapes of M. The formatted tape contains

$$\#\dot{w}_1 w_2 \cdots w_n \#\dot{\sqcup}\#\dot{\sqcup}\# \cdots \#.$$





MULTITAPE TURING MACHINES

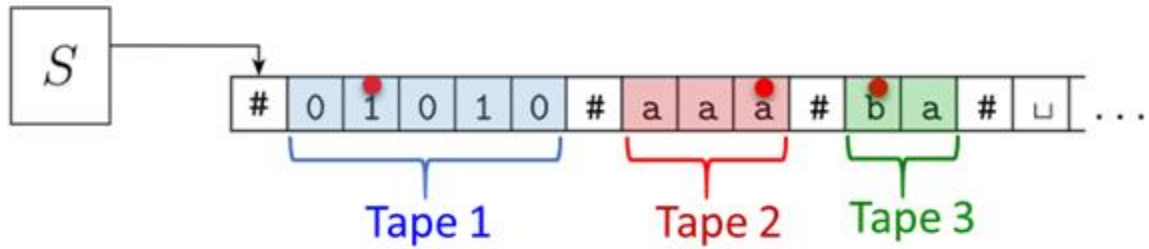
- Proof (cont.):

S = "On input $w = w_1 \cdots w_n$:

1. First S puts its tape into the format that represents all k tapes of M. The formatted tape contains

$$\#\dot{w}_1 w_2 \cdots w_n \#\dot{\sqcup}\#\dot{\sqcup}\# \cdots \#.$$

2. To simulate a single move, S scans its tape from the first #, which marks the left-hand end, to the $(k+1)$st #, which marks the right-hand end, in order to determine the symbols under the virtual heads. Then S makes a second pass to update the tapes according to the way that M's transition function dictates.

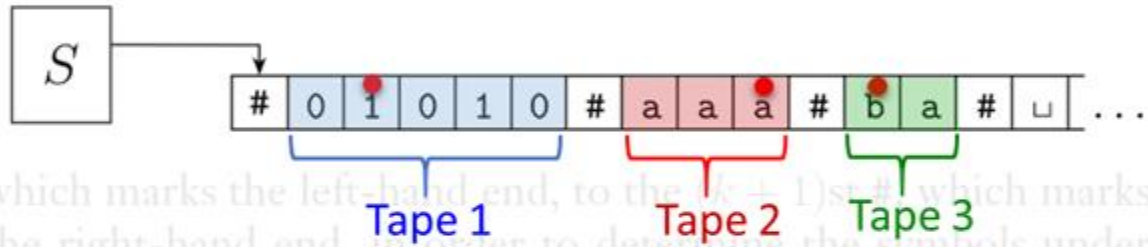


MULTITAPE TURING MACHINES

- Proof (cont.):

S = "On input $w = w_1 \cdots w_n$:

1. First S puts its tape into the format that represents all k tapes of M. The formatted tape contains



2. [illegible] which marks the left-hand end, to the $(k+1)$st # which marks the right-hand end, in order to determine the symbols under the virtual heads. Then S makes a second pass to update the tapes according to the way that M's transition function dictates.

3. If at any point S moves one of the virtual heads to the right onto a #, this action signifies that M has moved the corresponding head onto the previously unread blank portion of that tape. So S writes a blank symbol on this tape cell and shifts the tape contents, from this cell until the rightmost #, one unit to the right. Then it continues the simulation as before."

MULTITAPE TURING MACHINES

- Corollary : A language is Turing-recognizable if and only if some multitape Turing machine recognizes it.

  A language is Turing-recognizable $\leftrightarrow$ multitape Turing machine recognizes it

MULTITAPE TURING MACHINES

- Corollary : A language is Turing-recognizable if and only if some multitape Turing machine recognizes it.

  A language is Turing-recognizable ↔ multitape Turing machine recognizes it

- Proof:
  - **Part a (→):**
    - A **Turing-recognizable** language is recognized by a single-tape Turing machine, which is a special case of a multitape Turing machine.

MULTITAPE TURING MACHINES

- Corollary : A language is Turing-recognizable if and only if some multitape Turing machine recognizes it.

  A language is Turing-recognizable ↔ multitape Turing machine recognizes it

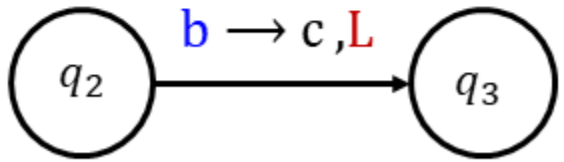
- Proof:
  - Part a (→):
    - A Turing-recognizable language is recognized by an ordinary(single-tape) Turing machine, which is a special case of a multitape Turing machine.
  - **Part b (←):**
    - A multitape Turing machine **recognizes the language**.
      - Its equivalent single-tape TM **recognizes** it.
      - Then, it is Turing-recognizable.

NONDETERMINISTIC TURING MACHINES

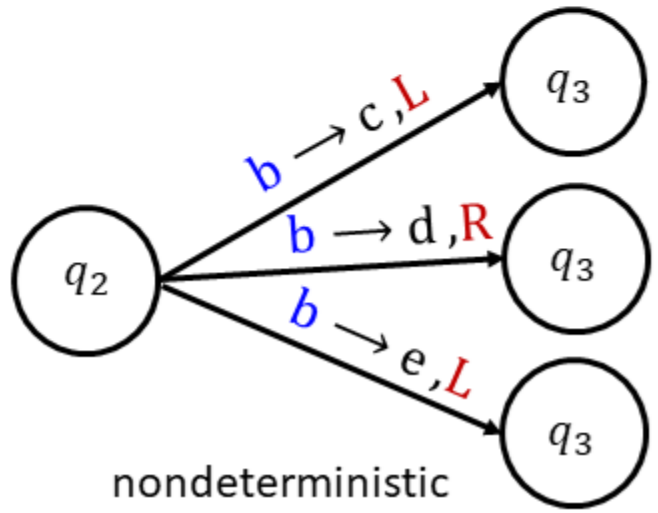
- In a nondeterministic Turing machine :
  - **At any point in a computation**, the machine may proceed according to several possibilities.

- The transition function for a nondeterministic Turing machine has the form

$$\delta : Q \times \Gamma \longrightarrow \mathcal{P}(Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}).$$



deterministic



nondeterministic

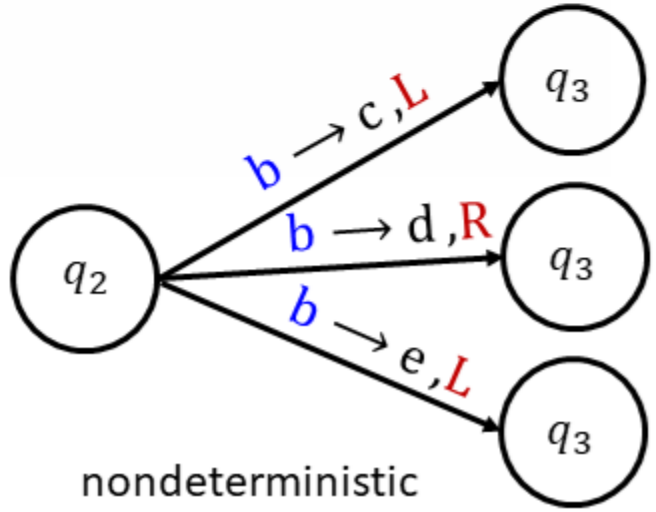
NONDETERMINISTIC TURING MACHINES

- In a nondeterministic Turing machine :
  - **At any point in a computation**, the machine may proceed according to several possibilities.

- The transition function for a nondeterministic Turing machine has the form

$$\delta : Q \times \Gamma \longrightarrow \mathcal{P}(Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}).$$

- **At each moment** in the computation there can be more than one successor configuration.

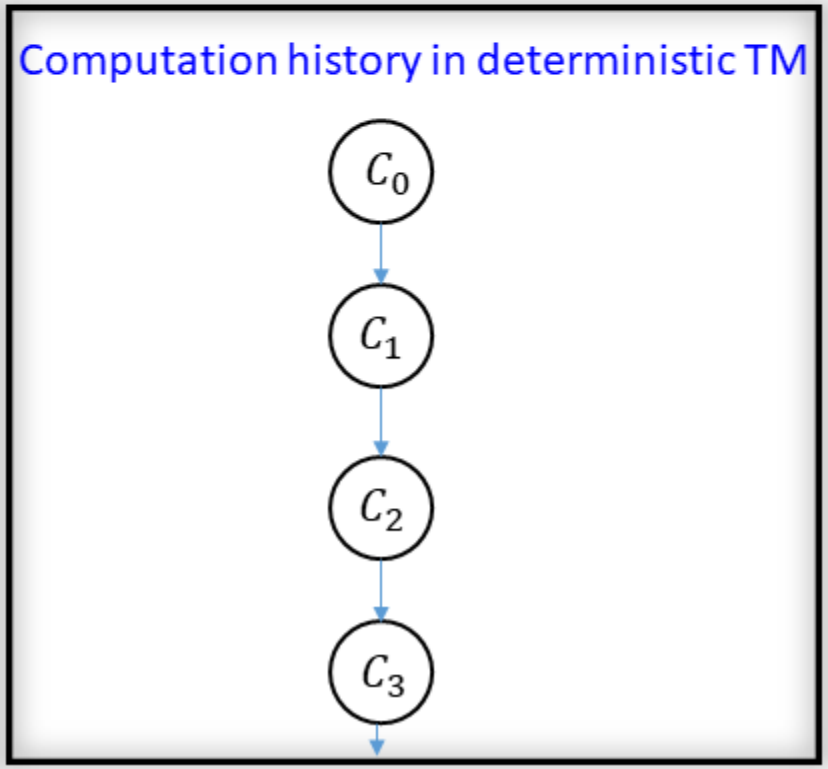


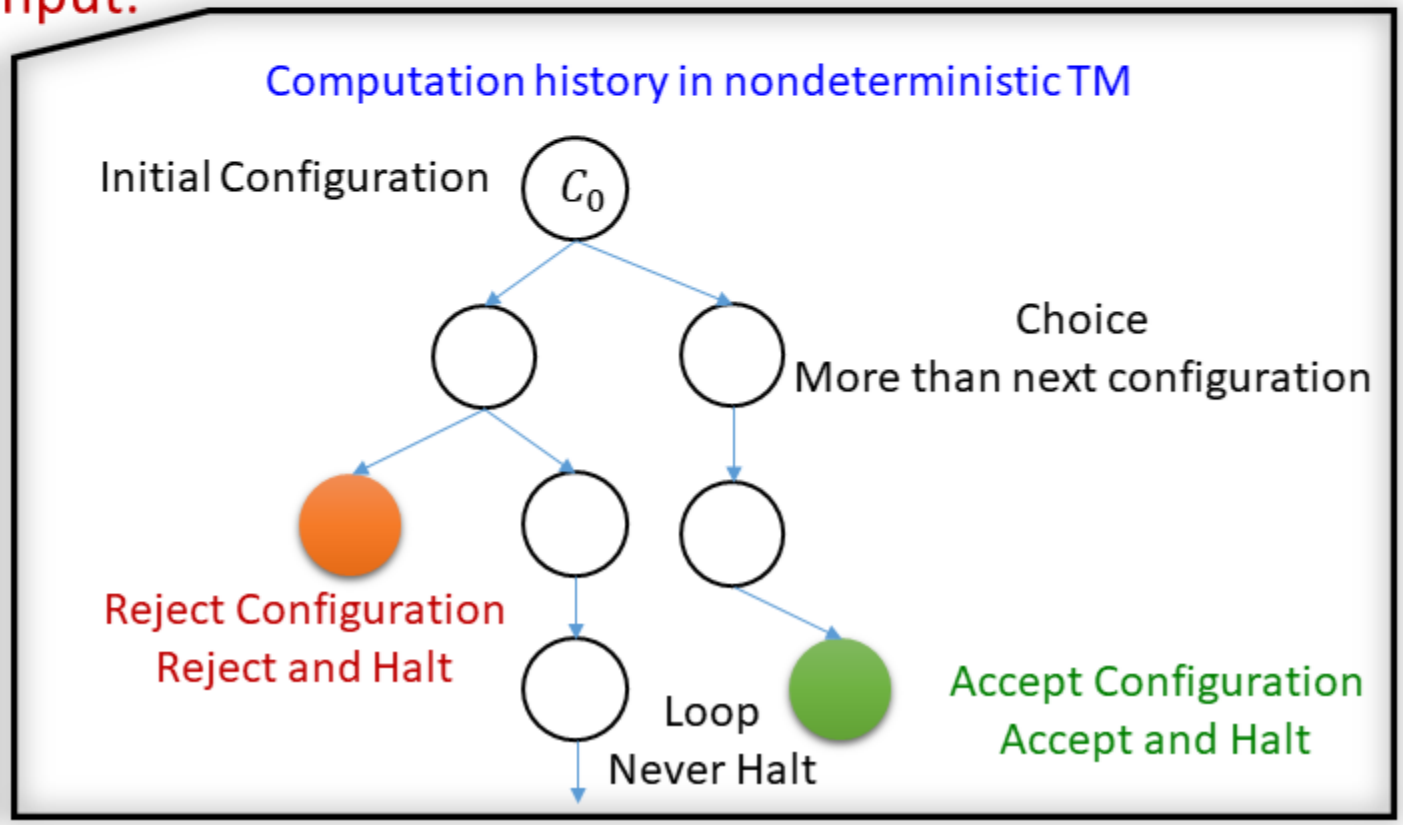
nondeterministic

NONDETERMINISTIC TURING MACHINES

- The computation of a nondeterministic Turing machine is a tree
  - whose branches correspond to **different possibilities for the machine**.

- If **some branch** of the computation leads to the accept state,
  - the machine accepts its input.





OUTCOMES OF A NONDETERMINISTIC TURING MACHINES

- **ACCEPT:**
  - If **any branch** of the computation accepts, then the NDTM will accept.
- **REJECT:**
  - If **all branches** of the computation halt and reject, then the NDTM rejects.
- **LOOP:**
  - Computation continues, but **accept** is **never** encountered.
  - **Some branches** in the computation history are **infinite**.

Note: NDTM = NONDETERMINISTIC TURING MACHINES

NONDETERMINISTIC TURING MACHINES

- Theorem : Every nondeterministic Turing machine has an equivalent deterministic Turing machine.

NONDETERMINISTIC TURING MACHINES

- Theorem : Every nondeterministic Turing machine has an equivalent deterministic Turing machine.

- Proof Idea:
  - We have to show how we can construct an equivalent of DTM (D) for a NDTM (N)
    - If **N accepts** (on any branch), then **D will accept**.
    - If N **halts on every branch without any accepts**, then D will **Halt and reject**.

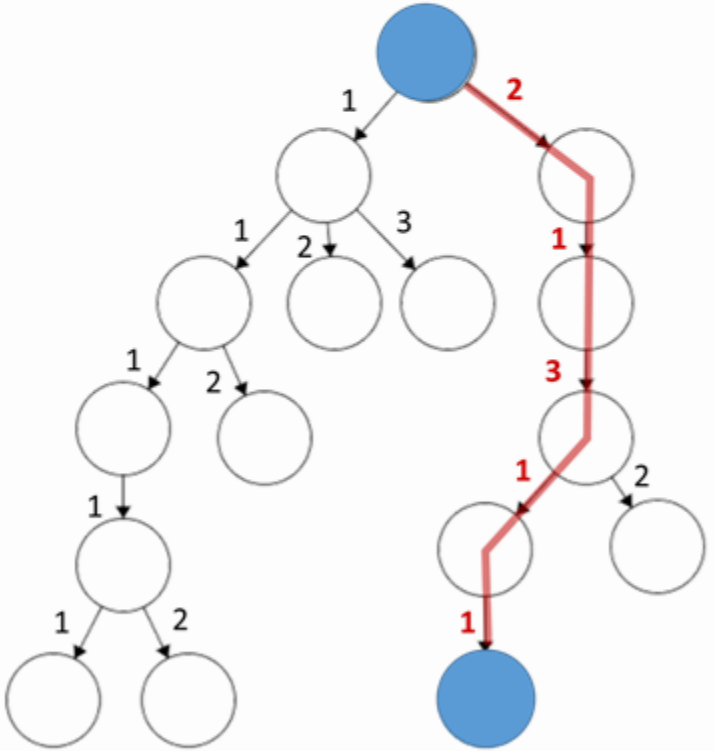
NONDETERMINISTIC TURING MACHINES

- Theorem : Every nondeterministic Turing machine has an equivalent deterministic Turing machine.

- Proof Idea:
  - We have to show how we can construct an equivalent of DTM (D) for a NDTM (N)
    - If **N accepts** (on any branch), then **D will accept**.
    - If N **halts on every branch without any accepts**, then D will **Halt and reject**.

NONDETERMINISTIC TURING MACHINES

- Proof Idea (cont.):
  - How to search the tree, looking for **"ACCEPT"** ?
  - How to show any node in a **computation history** by a **number** ?



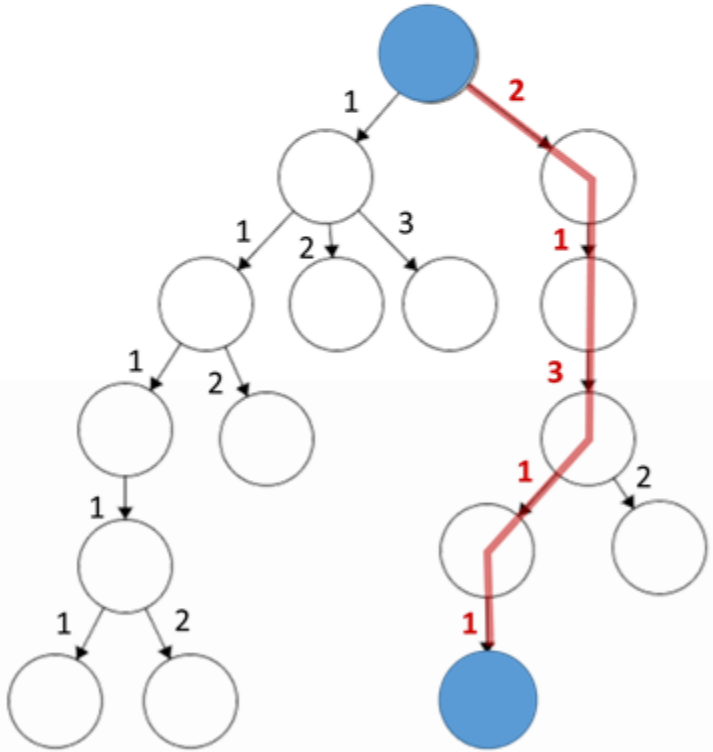
Node No.: 2 1 3 1 1

NONDETERMINISTIC TURING MACHINES

- Proof Idea (cont.):
  - How to search the tree, looking for **"ACCEPT"** ?
  - How to show any node in a **computation history** by a **number** ?

  - Which one of the following Search Method ?
    - Depth-First Search
    - Breadth-First Search

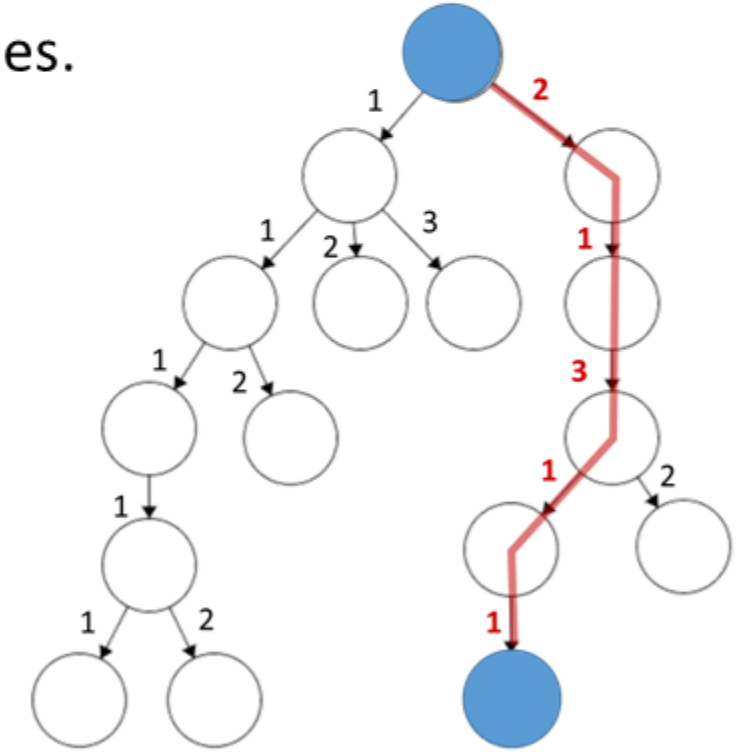
  - The depth-first search strategy goes all



Node No.: 2 1 3 1 1

NONDETERMINISTIC TURING MACHINES

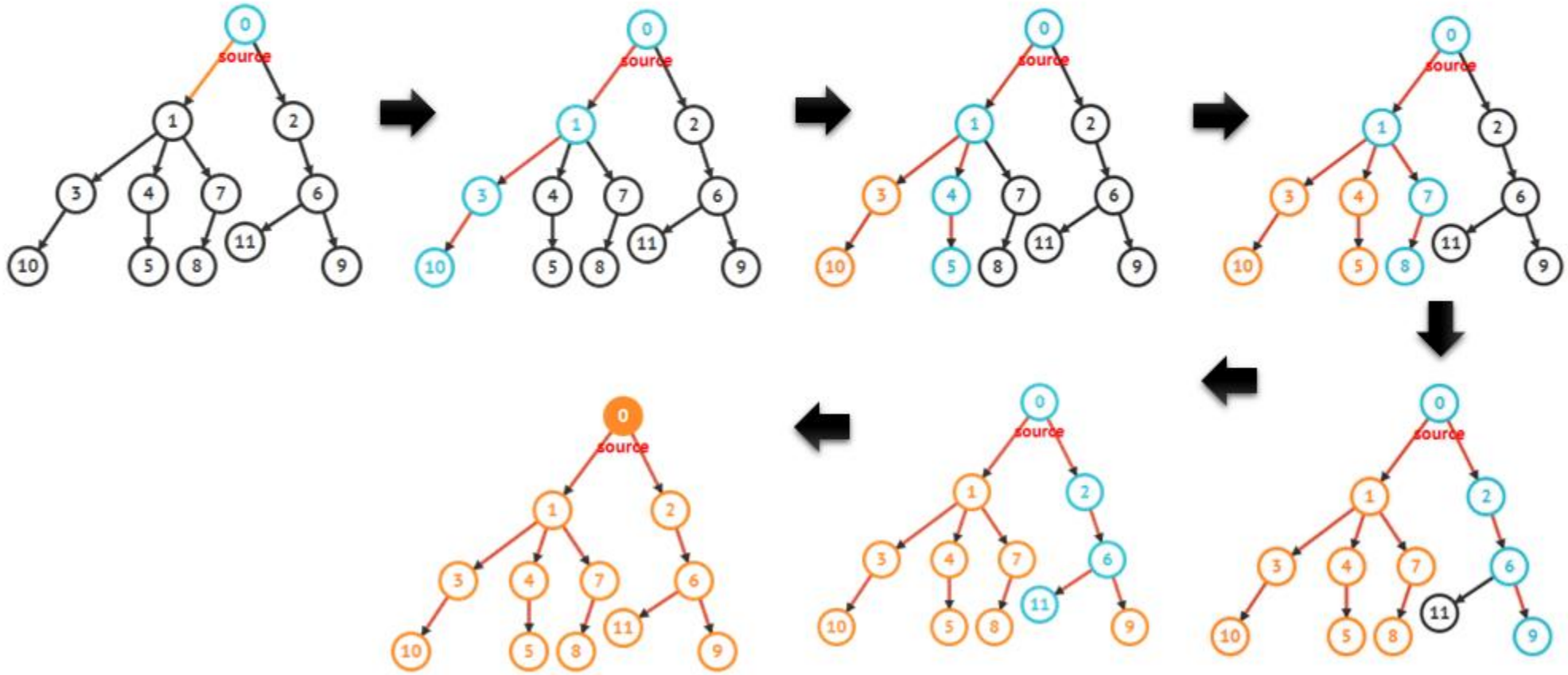
- Proof Idea (cont.):
  - Depth-First Search
    - This strategy goes all the way down one branch
      - before backing up to explore other branches.
    - could go forever down one **infinite branch**
      - and miss an accepting configuration on some other branch.



Node No.: 2 1 3 1 1

NONDETERMINISTIC TURING MACHINES

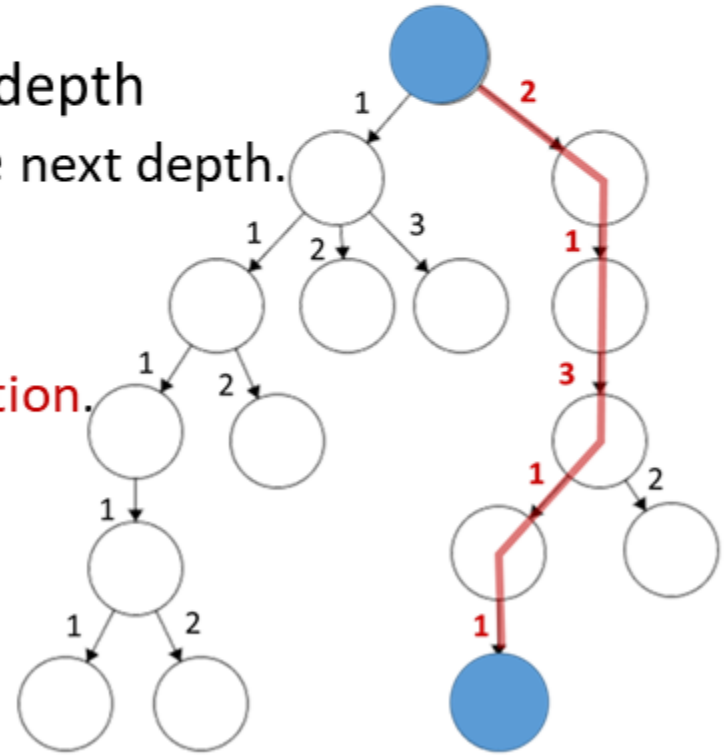
- Proof Idea (cont.):
  - Depth-First Search



NONDETERMINISTIC TURING MACHINES

- Proof Idea (cont.):
  - Hence we design D to **explore the tree** by using breadth-first search instead.

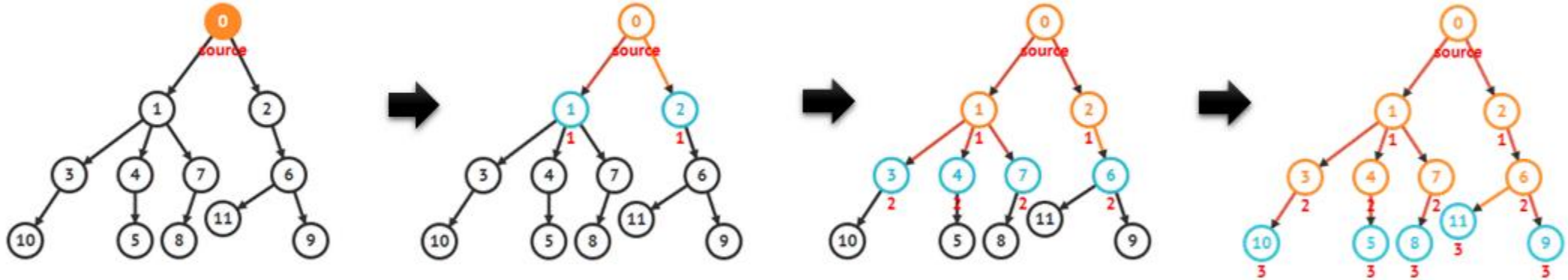
  - Breadth-First Search
    - This strategy explores all branches to the same depth
      - before going onto explore any branch to the next depth.
    - This method guarantees that
      - D will visit every node in the tree
        - until it encounters an accepting configuration.



Node No.: 2 1 3 1 1

NONDETERMINISTIC TURING MACHINES

- Proof Idea (cont.):
  - Breadth-First Search

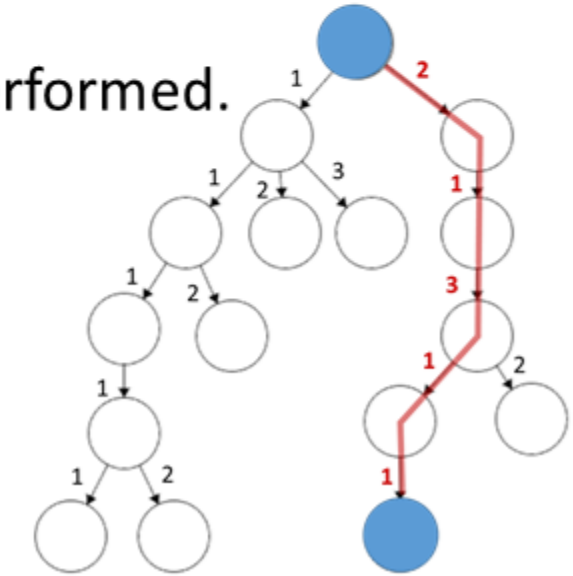


NONDETERMINISTIC TURING MACHINES

- Proof Idea (cont.):
  - **Searching Strategy: Breadth-First Search**

  - We need a numbering system for the nodes (node no.) to choose the next node for searching among of the many Nondeterministic choices.

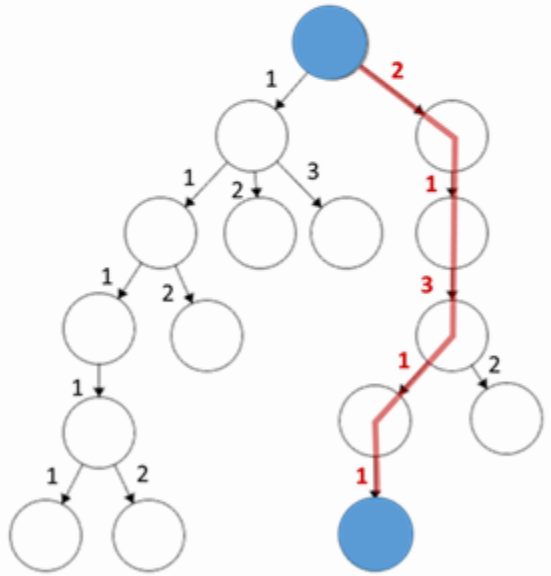
  - Our numbering system should support in idea that
    - to examine a node (configuration)
      - the entire computation from scratch should be performed.



NONDETERMINISTIC TURING MACHINES

- Proof Idea (cont.):
  - **Searching Strategy: Breadth-Firth Search**

  - How many choices are there at each step in the computation ?



NONDETERMINISTIC TURING MACHINES

- Proof Idea (cont.):
  - **Searching Strategy: Breadth-Firth Search**
  - How many choices are there at each step in the computation ?
  - Every **node** in the tree can have at most **b** children,
    - where b is the size of the largest set of possible choices given by N's transition function.

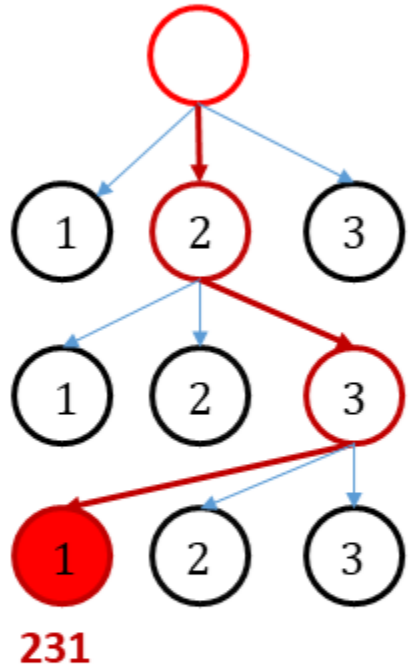
NONDETERMINISTIC TURING MACHINES

- Proof Idea (cont.):
  - Searching Strategy: Breadth-Firth Search
  - How many choices are there at each step in the computation ?
  - Every **node** in the tree can have at most **b** children,
    - where b is the size of the largest set of possible choices given by N's transition function.

- Then, to every node in the tree we assign an address (node. no)
  - that is a string over the alphabet $\Gamma_b = \{1,2,\dots,b\}$.
  - and shows the Path of computation from initial configuration to that node.

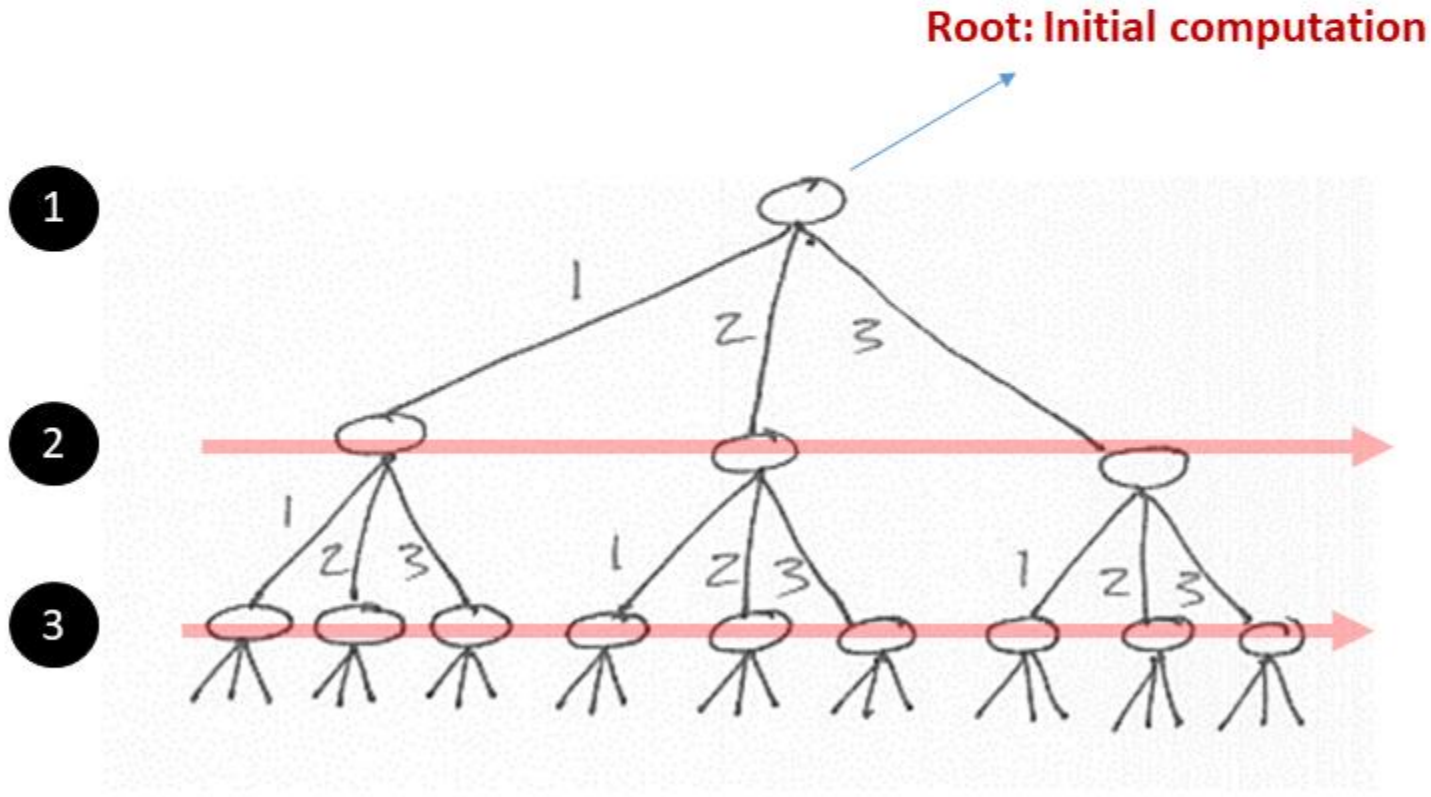
NONDETERMINISTIC TURING MACHINES

- Proof Idea (cont.):
  - Suppose : $\Gamma_b = \{1,2,3\}$ in the computation tree.
  - **Address 231 :**
    - starting at the root,
    - going to its 2nd child,
    - going to that node's 3rd child,
    - and finally going to that node's 1st child.



NONDETERMINISTIC TURING MACHINES

- Proof Idea (cont.):
  - **Breadth-First Search Order**

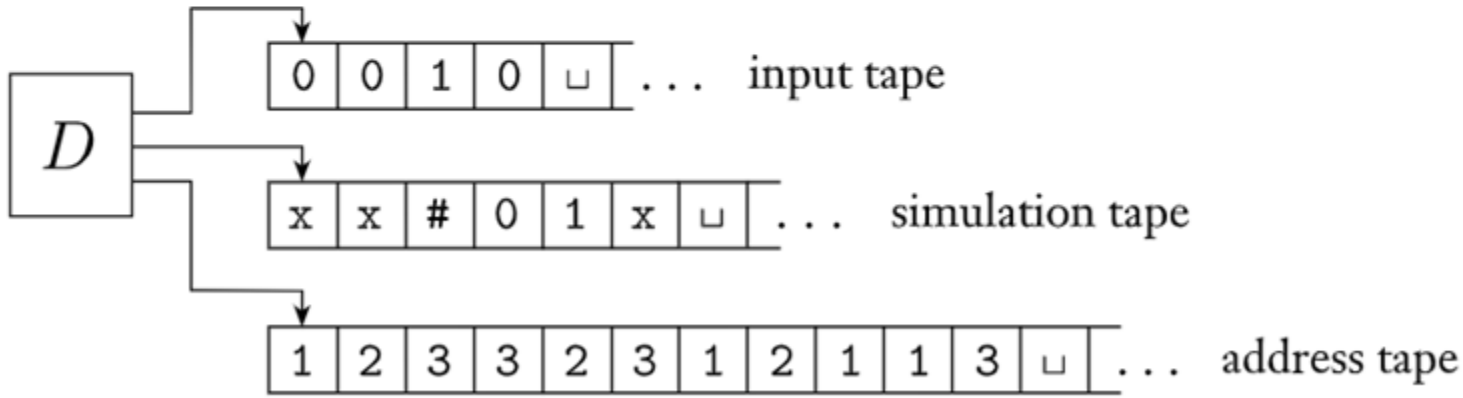


ε
1
2
3
11
12
13
21
22
23
31
32
33
111
112
113
121
122
123

Counting in base 3

NONDETERMINISTIC TURING MACHINES

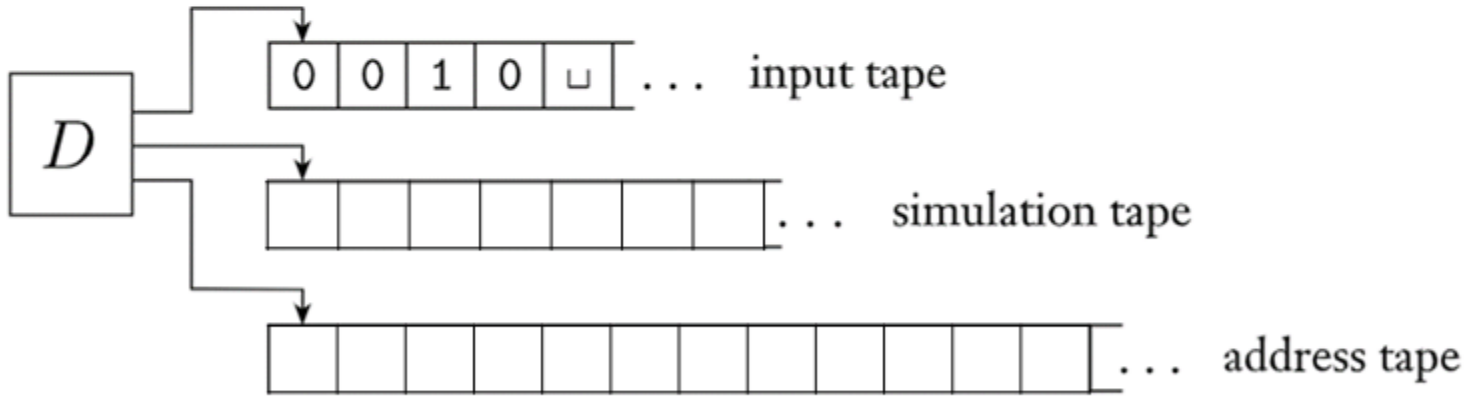
- Proof :
  - The simulating deterministic $TM\ D$ has three tapes.
    - This arrangement is **equivalent** to having a single tape.
  - **Tap e 1:** always contains the input string and is never altered .
  - **Tap e 2:** maintains a copy of **N's tape** on some branch of its nondeterministic computation.
  - **Tape 3:** keeps track of D's location in N's nondeterministic computation tree.



NONDETERMINISTIC TURING MACHINES

- Theorem : Every nondeterministic Turing machine has an equivalent deterministic Turing machine.

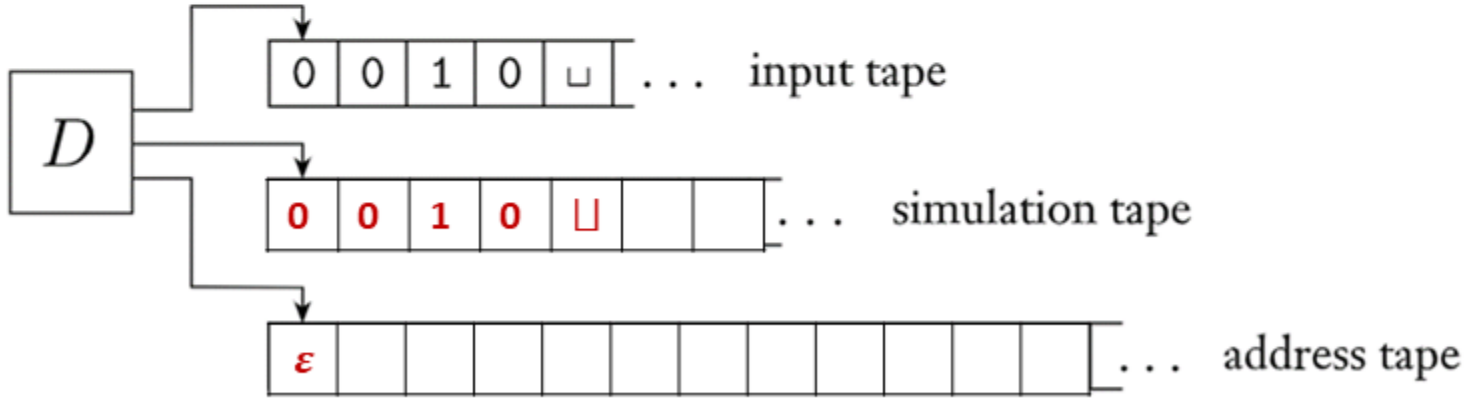
- Proof (cont.) :
  - **1.** Initially, tape 1 contains the input w, and tapes 2 and 3 are empty.



NONDETERMINISTIC TURING MACHINES

- Theorem : Every nondeterministic Turing machine has an equivalent deterministic Turing machine.

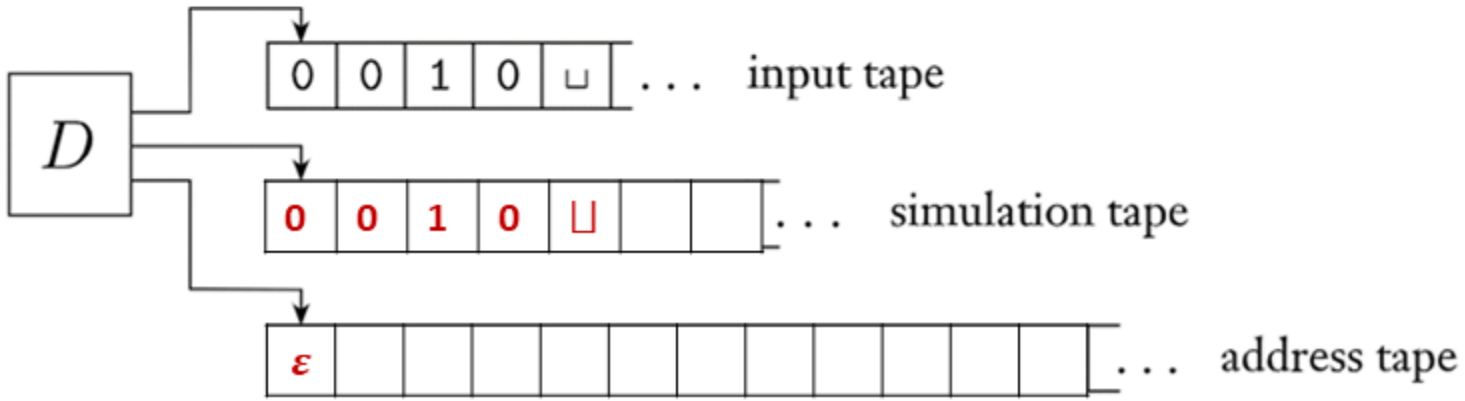
- Proof :
  - **1.** Initially, tape 1 contains the input w, and tapes 2 and 3 are empty.
    2. Copy tape 1 to tape 2 and initialize the string on tape 3 to be ε.



NONDETERMINISTIC TURING MACHINES

- Proof :

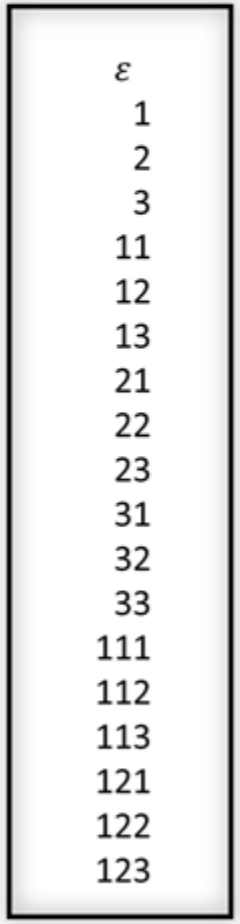
1. Initially, tape 1 contains the input w, and tapes 2 and 3 are empty.
2. Copy tape 1 to tape 2 and initialize the string on tape 3 to be ε.
3. Use tape 2 to simulate N with input w on one branch of its nondeterministic computation. Before each step of N, consult the next symbol on tape 3 to determine which choice to make among those allowed by N's transition function. If no more symbols remain on tape 3 or if this nondeterministic choice is invalid, abort this branch by going to stage 4. Also go to stage 4 if a rejecting configuration is encountered. If an accepting configuration is encountered, *accept* the input.



NONDETERMINISTIC TURING MACHINES

- Proof :

1. Initially, tape 1 contains the input w, and tapes 2 and 3 are empty.
2. Copy tape 1 to tape 2 and initialize the string on tape 3 to be ε.
3. Use tape 2 to simulate N with input w on one branch of its nondeterministic computation. Before each step of N, consult the next symbol on tape 3 to determine which choice to make among those allowed by N's transition function. If no more symbols remain on tape 3 or if this nondeterministic choice is invalid, abort this branch by going to stage 4. Also go to stage 4 if a rejecting configuration is encountered. If an accepting configuration is encountered, *accept* the input.
4. Replace the string on tape 3 with the next string in the string ordering. Simulate the next branch of N's computation by going to stage 2.



String ordering

NONDETERMINISTIC TURING MACHINES

- Corollary : A language is **Turing-recognizable** if and only if **some nondeterministic Turing machine recognizes it**.

- Proof :
  - Any DTM is automatically a NDTM, and so one direction of this corollary follows immediately.
  - The other direction follows from the previous Theorem.

NONDETERMINISTIC TURING MACHINES

- We call a nondeterministic Turing machine a decider
  - if all branches halt on all inputs.

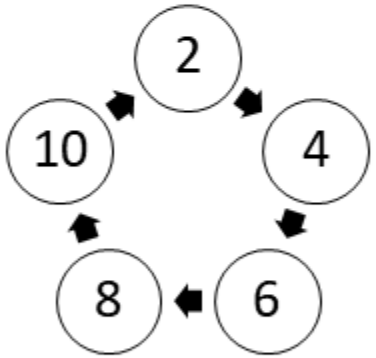
- Corollary : A language is decidable **if and only if** some nondeterministic Turing machine decides it.

VARIATIONS OF TMs

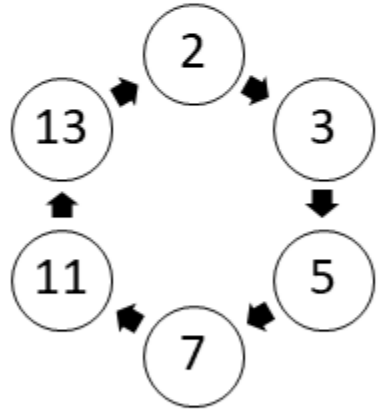
- Many variations of TM have been proposed:
  - **Ordinary Turing Machine**
  - **Multiple tape Turing machine**
  - Multiple track Turing machine (unified tape head)
  - **Non-deterministic Turing machine**
  - Two dimensional Turing machine
  - Multidimensional Turing machine
  - Two-way infinite tape
  - Recursively enumerable Turing Machine
  - Combinations of the above

SEQUENCE GENERATOR

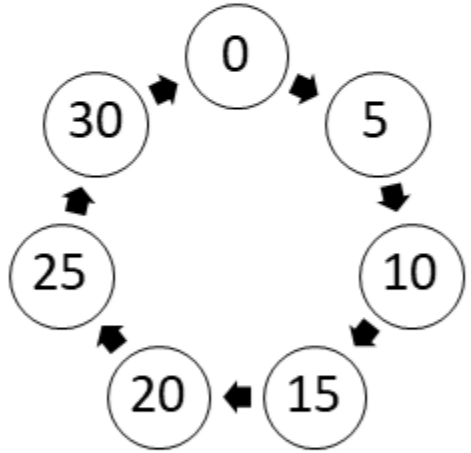
- Sequence generator (counter)
  - is a TM,
  - with no input,
  - but output.



Even integers from 2 to 10



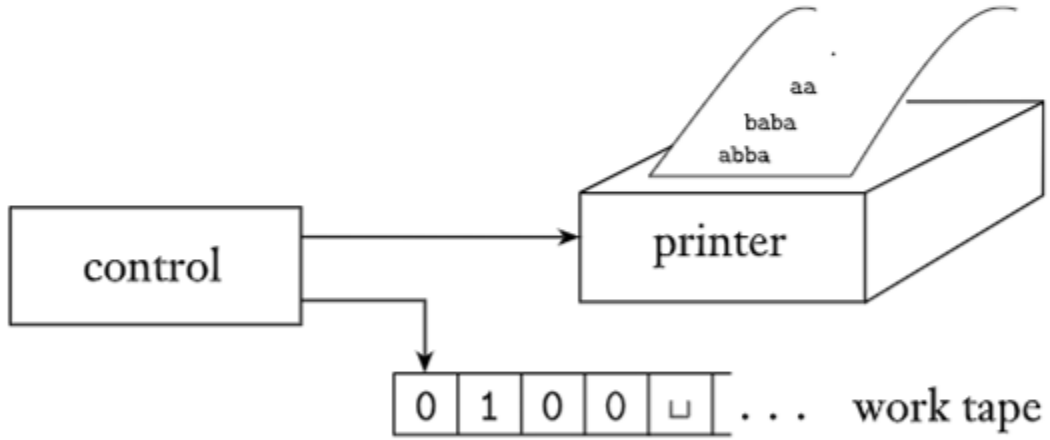
Prime integers from 2 to 15



Multiple of 5 integers from 0 to 30

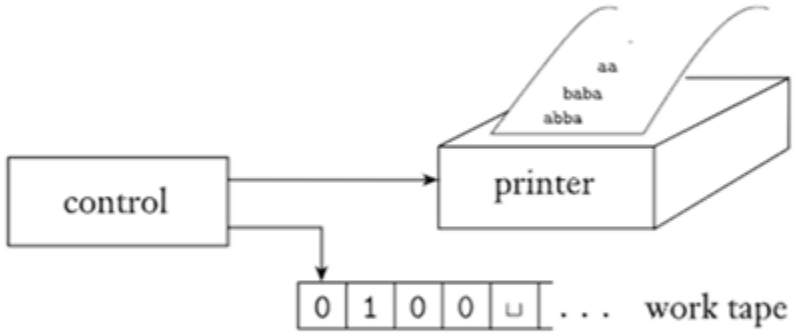
ENUMERATORS

- This **TM-like machine** prints out all the strings in a given language.
-
- **Loosely defined**: is a Turing machine with an attached printer.



ENUMERATORS

- TM can also be **designed to enumerate** a language.
  - Called Enumerator.
    - Produces (prints) the exhaustive list of string of the language, progressively longer and longer.
    - has no input (starts with a blank input),
    - its **computation** continues forever if
      - the language is infinite,
      - as well as when it is finite.
      - (if the enumerator doesn't halt)



- **Loosely defined**: an enumerator is a Turing machine with an attached printer.

ENUMERATORS

- **For enumeration**, a TM of tape k ≥ 2 is used,
  - tape 1 is output tape,
    - which would hold all the generated strings, separated by #, and
  - other tapes are working tapes.

  - **Output tape 1 has:** $\mathrm{B}\#u_1\#u_2\#\ldots\#u_i\#\ldots$, where $u_i \in \mathrm{L}$.
  - Tape head 1 always moves R, S, while others may move R, L, S.

- Enumerator may generate the strings of the language in any order,
  - possibly with repetitions.

ENUMERATORS

- Theorem: A language is **Turing-recognizable** if and only if some **enumerator** enumerates it.

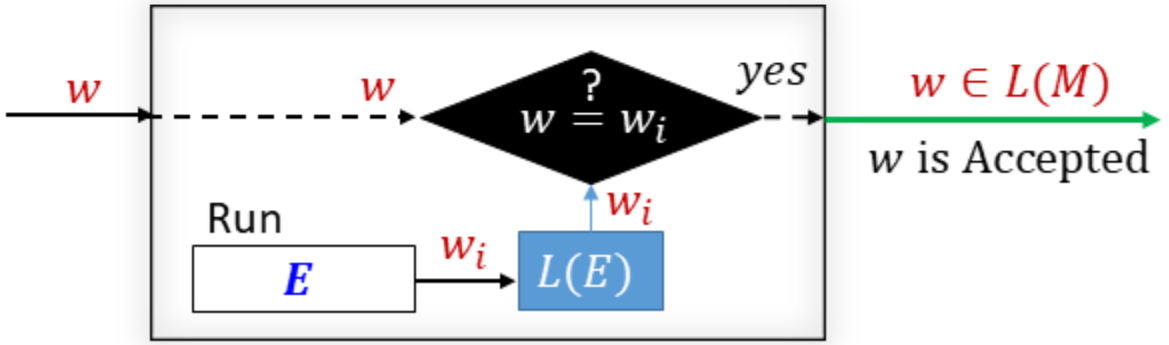
ENUMERATORS

- **Proof (cont.):**
  - **The If-part:** If an enumerator E enumerates the language A then a TM M recognizes A.

M="On input w

1. **Run E.** Every time E outputs a string, compare it with w.
2. If w ever appears in the output of E, **accept**.

- M accepts only those strings that appear on E's list.



ENUMERATORS

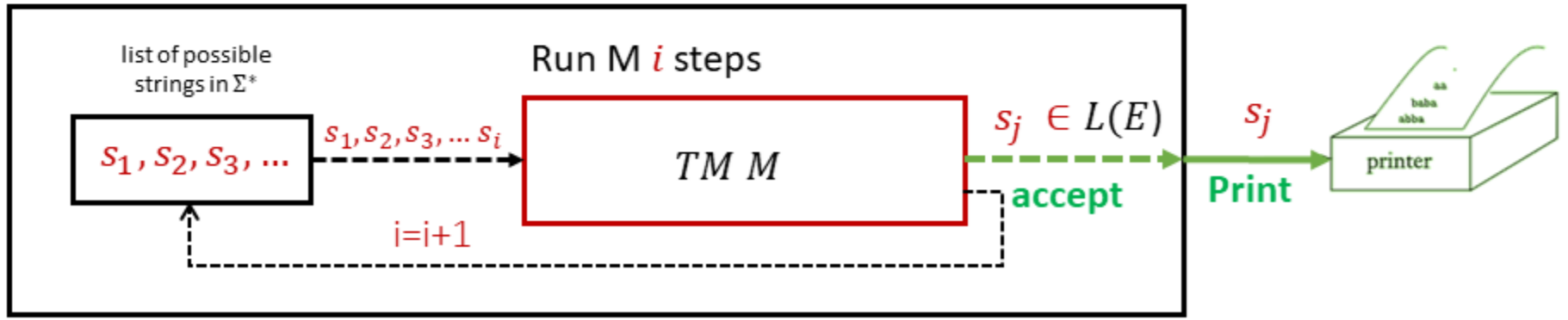
- **Proof (cont.):**
  - **The only If-part:** If a TM M recognizes a language A, we can construct a enumerator for A.

ENUMERATORS

- **Proof (cont.):** (See the next slide about the countable sets.)

  - Assume $s_1, s_2, s_3, \ldots$ is a list of possible strings in Σ^*. (Lexicographical order)

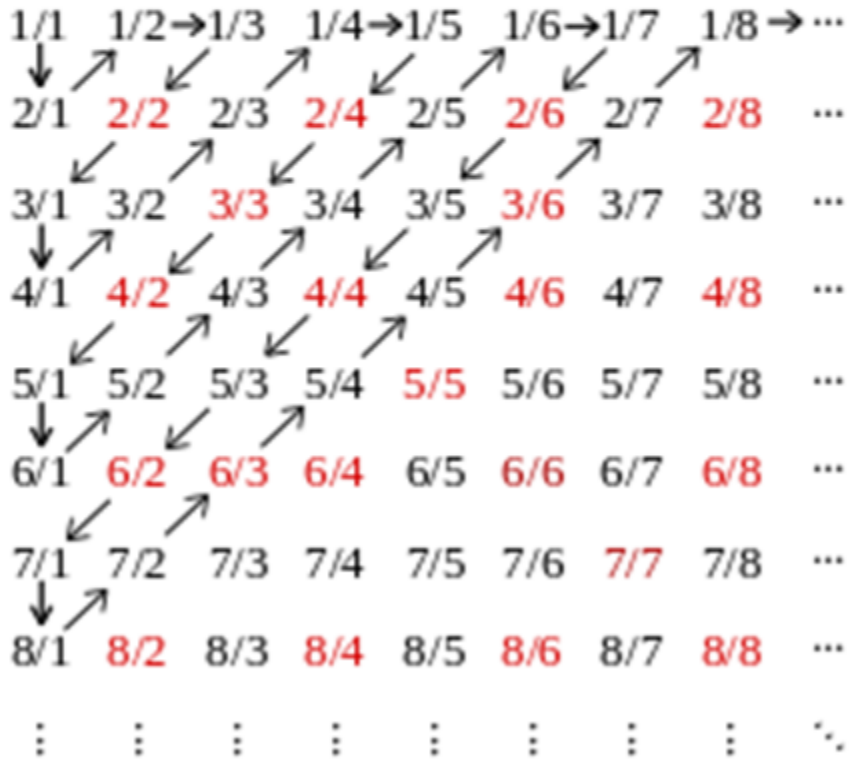
  - E="Ignore the input
    1. Repeat the following for $i = 1,2,3, \ldots$
       2. Run M for i steps on each inputs $s_1, s_2, s_3, \ldots s_i$.
       3. If any computations **accept**, print out the corresponding s_j."
       4. If M accepts a particular string, it will appear on the list generated by E (in fact infinitely many times).



E

COUNTABLE SETS

- In mathematics, a countable set is a set with the same cardinality (number of elements) as some subset of the set of natural numbers.
- A countable set is either a finite set or a countably infinite set.



Wikipedia

Chapter 3

3.3 THE DEFINITION OF ALGORITHM

THE DEFINITION OF ALGORITHM

- **Informally** speaking, an algorithm
  - a **set of rules** (instructions)that precisely defines a sequence of operations for carrying out some task.
  - also called procedure or recipe.

- Even though **algorithms** have had a long history in mathematics
  - the notion of algorithm itself was not defined precisely until the twentieth century.

- The word “algorithm” is derived, however, from the name of the Persian scholar Mathematician Al-Kwarizmi (ca. 780-850).

HILBERT'S PROBLEMS

- In 1900, mathematician David Hilbert presented a list of **mathematical open problems** and posed them as a challenge for the coming century.

- The tenth problem on his list concerned algorithms.

HILBERT'S PROBLEMS

- In 1900, mathematician David Hilbert presented a list of **mathematical open problems** and posed them as a challenge for the coming century.
- The tenth problem on his list concerned algorithms.

Given a polynomial of several variables with integer coefficients, does it have an integer root – an assignment of integers to variables, that make the polynomial evaluate to 0.

For example, $6x^3yz^2 + 3xy^2 - x^3 - 10$ has a root at $x = 5, y = 3, z = 0$.

Solution to Hilbert's 10th problem by Matiasevic, 1970

- Hilbert explicitly asked that an algorithm/procedure to be "devised".
- He assumed it existed; somebody **needed to find it**!

Solution to Hilbert's 10th problem by Matiasevic, 1970

- Hilbert explicitly asked that an algorithm/procedure to be "devised".
- He assumed it existed; somebody **needed to find it**!
- In 1970, the 23 year-old Russian mathematician Yuri Matiasevic showed that no algorithm exists for the task given by Hilbert.
- For such a "negative" result, first a precise formal definition of "algorithm" was needed.
  - **Hilbert** didn't have it yet, and probably didn't even imagine that this could be the solution to his problem.

CHURCH-TURING THESIS

- In early 20th century, there was no formal definition of an algorithm.

- In 1936, **Alonzo Church and Alan Turing** came up with formalisms to define algorithms.
  - **Church with** $\lambda - calculus$ and **Turing** with his machine.
  - leading to the CHURCH-TURING THESIS.

  - These two definitions were shown to be equivalent.

CHURCH-TURING THESIS

- The **connection** between the informal notion of algorithm and the precise definition has come to be called CHURCH-TURING THESIS.

Intuitive notion of algorithms	equals	*Turing machine algorithms*

The Church–Turing thesis

REPHRASING MATIASEVIC'S RESULT IN OUR TERMINOLOGY

- Let $D = \{p \mid p \text{ is a polynomial with integral roots}\}$.
- Hilbert's 10th problem **in TM terminology** is "Is D decidable?"
- **No!.** by the Church-Turing thesis, there is no algorithm to test membership in D.

CHURCH-TURING THESIS

- Let $D = \{p \mid p \text{ is a polynomial with integral roots}\}$.
- Hilbert's 10th problem **in TM terminology** is "Is D decidable?" **(No!)**
- However D is Turing-recognizable!

CHURCH-TURING THESIS

- Let $D = \{p \mid p \text{ is a polynomial with integral roots}\}$.
- Hilbert's 10th problem **in TM terminology** is "Is D decidable?" **(No!)**
- However D is Turing-recognizable!

- Consider a simpler version
  - $D_1 = \{p \mid p$ is a polynomial over x with integral roots$\}$.
  - such as $4x^3 - 2x^2 + x - 3$. (root x= 1)

CHURCH-TURINGTHESIS

- $D_1 = \{p \mid p \text{ is a polynomial over } x \text{ with integral roots}\}$.

Here is a TM M_1 that recognizes D_1:

M_1 = "On input $\langle p \rangle$: where p is a polynomial over the variable x.

1. Evaluate p with x set successively to the values $0, 1, -1, 2, -2, 3, -3, \ldots$. If at any point the polynomial evaluates to 0, *accept*."

- If p **has an integral root**, M_1 eventually will find it and accept.
- If p **does not** have an integral root, M_1 will run forever.

- For the multivariable case, we can present a similar TM M that recognizes D.
- Here, M goes through all possible settings of its variables to integral values.

CHURCH-TURINGTHESIS

- Both M_1 and M are recognizers but not deciders.

CHURCH-TURINGTHESIS

- Both M_1 and M are recognizers but not deciders.
- We can convert M_1 to be a decider for D_1.
  - It has been proved that the roots of such a polynomial must lie between the values

$$\pm k \frac{c_{\max}}{c_1},$$

- Where
  - k is the number of terms in the polynomial,
  - $cmax$ is the coefficient with the largest absolute value,
  - c_1 is the coefficient of the highest order term.

- If a root is not found within these bounds, the machine rejects.

CHURCH-TURINGTHESIS

- Both M_1 and M are recognizers but not deciders.
- We can convert M_1 to be a decider for D_1.
  - It has been proved that the roots of such a polynomial must lie between the values

$$\pm k \frac{c_{\max}}{c_1},$$

- Where
  - k is the number of terms in the polynomial,
  - $cmax$ is the coefficient with the largest absolute value,
  - c_1 is the coefficient of the highest order term.

- If a root is not found within these bounds, the machine rejects.

- Matijasevic's theorem shows that calculating such bounds for multivariable polynomials is impossible.

TERMINOLOGY FOR TURING MACHINE DESCRIPTIONS

- The Turing machine merely serves as a precise model for the definition of algorithm.
- We will mostly give high-level descriptions of TMs and do not spend much time on the low-level programming of TMs.
- We need only to be comfortable enough with TM to believe that they capture all algorithms.

THREE LEVELS OF TURING MACHINE DESCRIPTIONS

- High level description: use English to describe the algorithm; **ignore** implementation details.
- Implementation level description: use English to describe how the TM moves its heads, stores data, ...
- Formal description (often via a diagram)specify set of states, transition function δ, ...

DESCRIBING TURING MACHINES AND THEIR INPUTS

Standard way of describing TMs and their inputs.

- The input to TMs have to be strings.

DESCRIBING TURING MACHINES AND THEIR INPUTS

Standard way of describing TMs and their inputs.

- The input to TMs have to be strings.
- Every object O that enters a computation (such as a matrix or a graph) will be encoded by a string and represented with $< \mathbf{O} >$.

DESCRIBING TURING MACHINES AND THEIR INPUTS

Standard way of describing TMs and their inputs.

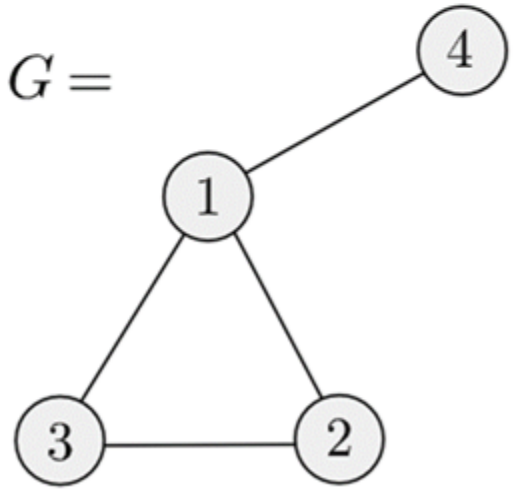
- The first line of the algorithm describes the input to the machine.
- The input to TMs have to be strings.
- Every object O that enters a computation (such as a matrix or a graph) will be encoded by a string and represented with $< \mathbf{O} >$.

- For example if G is a 4 node undirected graph with 4 edges

- $< \mathbf{G} >$ = (1,2,3,4)((1,2),(2,3),(3,1),(1,4))

nodes edges

$G =$



DESCRIBING TURING MACHINES AND THEIR INPUTS

Standard way of describing TMs and their inputs.

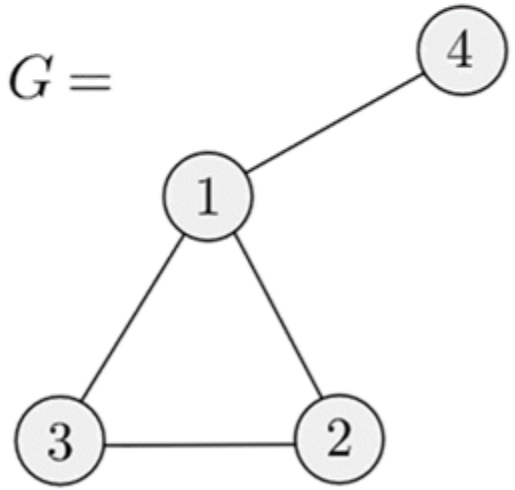
- The first line of the algorithm describes the input to the machine.
- The input to TMs have to be strings.
- Every object O that enters a computation (such as a matrix or a graph) will be encoded by a string and represented with $< \mathbf{O} >$.

- For example if G is a 4 node undirected graph with 4 edges

- $< \mathbf{G} >$ = (1,2,3,4)((1,2),(2,3),(3,1),(1,4))

nodes edges

$G =$



- Then we can define problems over graphs.
- **Example:** A={ ⟨G⟩ | G is a connected undirected graph }

DESCRIBING TURING MACHINES AND THEIR INPUTS

- If the **input** description is the encoding of an object as in ⟨O⟩,
  - the **Turing machine** first implicitly tests whether the input properly encodes an object of the desired form
  - and rejects it if it doesn't.

- Several objects $O_1, O_2, \ldots, O_k \rightarrow$ a single string $< O_1, O_2, \ldots, O_k >$.

EXAMPLE OF A HIGH-LEVEL DESCRIPTION OF A TM

- Let A be the language consisting of all strings representing undirected graph s that are connected.

- $A = \{ \langle G \rangle \mid G \text{ is a connected undirected graph} \}$

The following is a high-level description of a TM M that decides A.

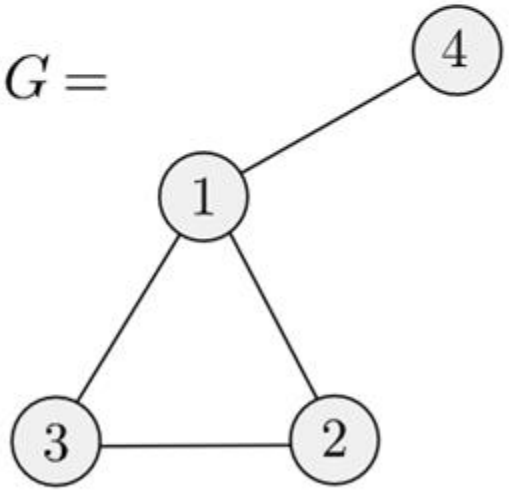
M = "On input $\langle G \rangle$, the encoding of a graph G:

1. Select the first node of G and mark it.
2. Repeat the following stage until no new nodes are marked:
3. For each node in G, mark it if it is attached by an edge to a node that is already marked.
4. Scan all the nodes of G to determine whether they all are marked. If they are, *accept*; otherwise, *reject*."

EXAMPLE OF A HIGH-LEVEL DESCRIPTION OF A TM

- First, we must understand how ⟨G⟩ encodes the graph G as a string.
- For example: Consider an encoding that is a list of the nodes of G followed by a list of the edges of G.

$G =$



$\langle G \rangle =$

(1,2,3,4)((1,2),(2,3),(3,1),(1,4))