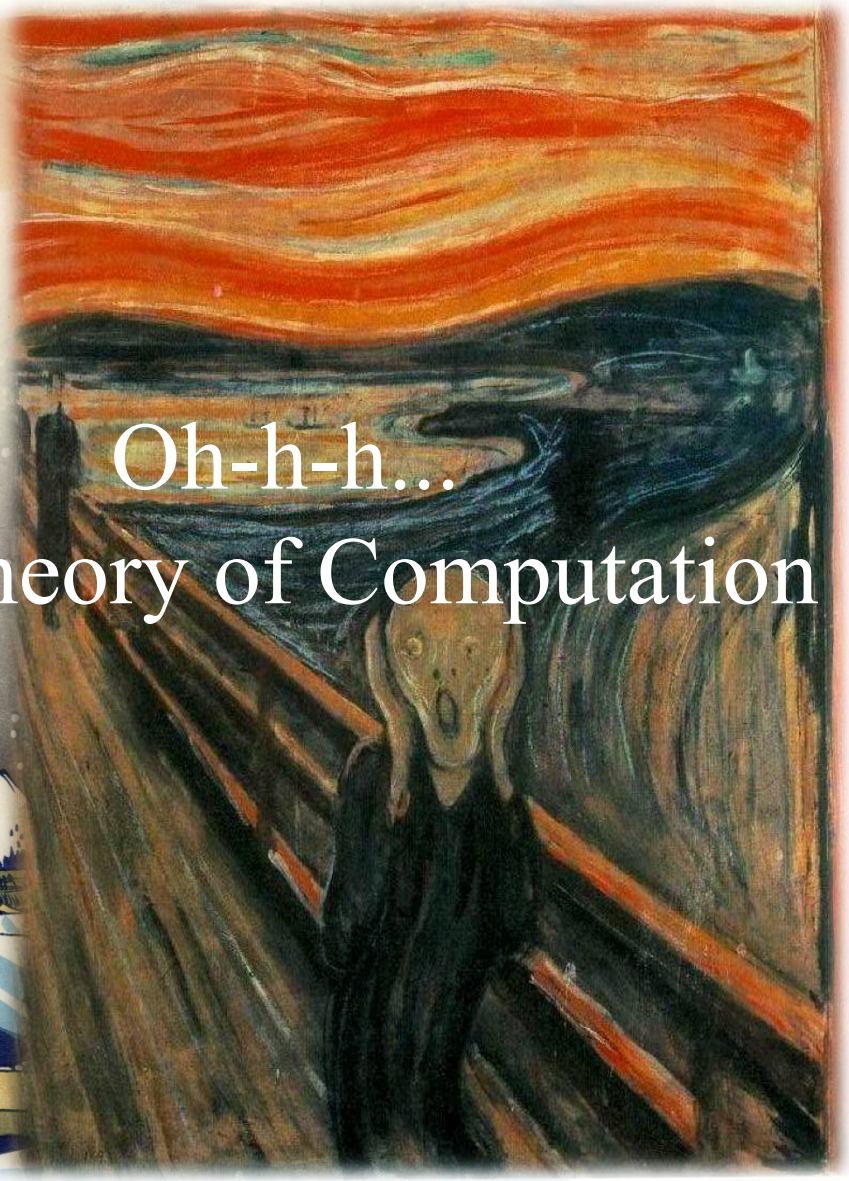
Università
della
Svizzera
italiana

**Facoltà
di scienze
informatiche**

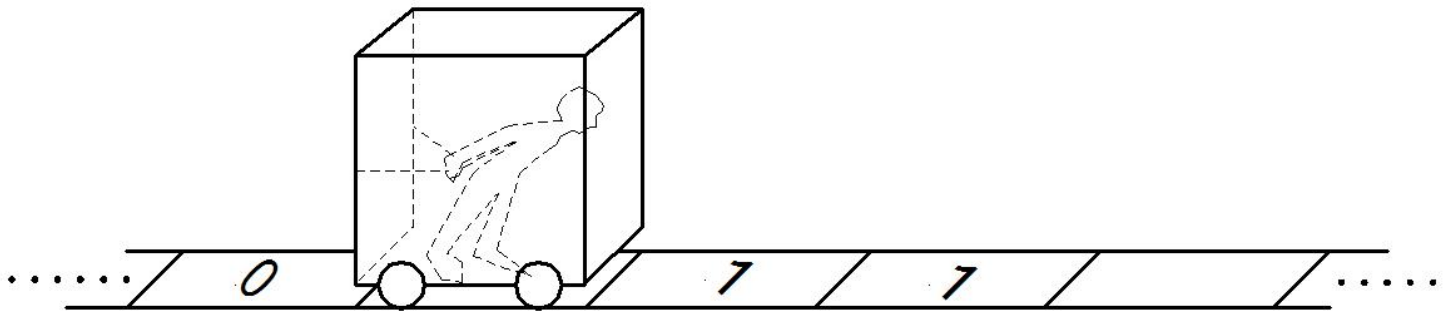


Oh-h-h...
it's Theory of Computation

Today's lecture

- We continue with Turing machines
  - Demo
  - Design ideas
  - Variants of Turing machines

Understanding Turing Machine



– Input tape is infinite
– Input head can both READ and WRITE
– Input head can move LEFT and RIGHT
– Has both ACCEPT and REJECT states ***and accepts/rejects right away*** (does not need to reach end of input)

Recall Formal Definition

Turing Machine is a 7- tuple $(Q, S, G, d, q_0, q_{accept}, q_{reject})$

Main points:

– Q is a set of states

– $\Sigma \subset \Gamma$, Σ is the input alphabet

– Blank symbol is in Γ, but not in Σ

– $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is the transition function

– Special case: when rd/wr head is at left end of the input tape

Summary on Turing Machines

- Unrestricted access to unlimited memory
- A Turing machine can do everything that a real computer can do
- Even a Turing machine can not solve certain problems: in a very real sense, these problems are beyond the theoretical concepts of computation

Demo of Lego Turing Machine

- Here is the link to the Lego Turing Machine

  https://www.youtube.com/watch?v=cYw2ewoO6c4

Enjoy watching this student project!

Ideas for design of TMs

Run various tests (sample inputs to your machine) and see if its transition function performs as expected

Important: make sure (if the problem is decidable) that your TM terminates for all instances not just for some

Find a solution which would avoid counting an infinite number (which is impossible)!

Variants of TMs

Versions of Turing Machines

- can you make suggestions on how a singletape TM could be modified?

Why one would want to have versions of TMs?

Would additional features add the power to the model?

Variants of TMs

Can you make suggestions on how a singletape TM could be modified?

Here is one example: TM with several tapes to work on

Why one would want to have versions of TMs?

For example, to improve efficiency or simplify the decision making

Would additional features add the power to the model?

No, it would not

Variants of TMs

Original transition of a TM machine:

– $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is the transition function

Consider changing the transition of a TM

– $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, S\}$ is the transition function

where S indicates that the controller head stays put

Does this feature allow a TM recognize additional languages?

NO

Variants of TMs

Original transition of a TM machine:

– $\delta : Q \times \Gamma \lozenge Q \times \Gamma \times \{L, R\}$ is the transition function

Consider changing the transition of TM

– $\delta : Q \times \Gamma \lozenge Q \times \Gamma \times \{L, R, S\}$ is the transition function

Where S indicates that the controller head stays put

Does this feature allow a TM recognize additional languages?

One can convert any TM with the "stay put" feature to one that does not have it (each stay can be replaced with 2 transitions: to the right and then to the left)

Variants of TMs

Invariance of a model to changes is called *robustness*

Turing machines have an astonishing degree of robustness

The original TM machines and its variants all have the same power – they recognize the same class of languages

Multitape TM

– $\delta : Q \times \Gamma^{\kappa} \rightarrow Q \times \Gamma^{\kappa} \times \{L, R, S\}^{\kappa}$

k – number of tapes

The transition function allows for reading, writing and moving the heads on some or all of tapes simultaneously

Multitape TM

Theorem: Every multitape Turing machine (MTM) has an equivalent single-tape Turing machine (STM)

Proof: We show how to convert MTM to STM.

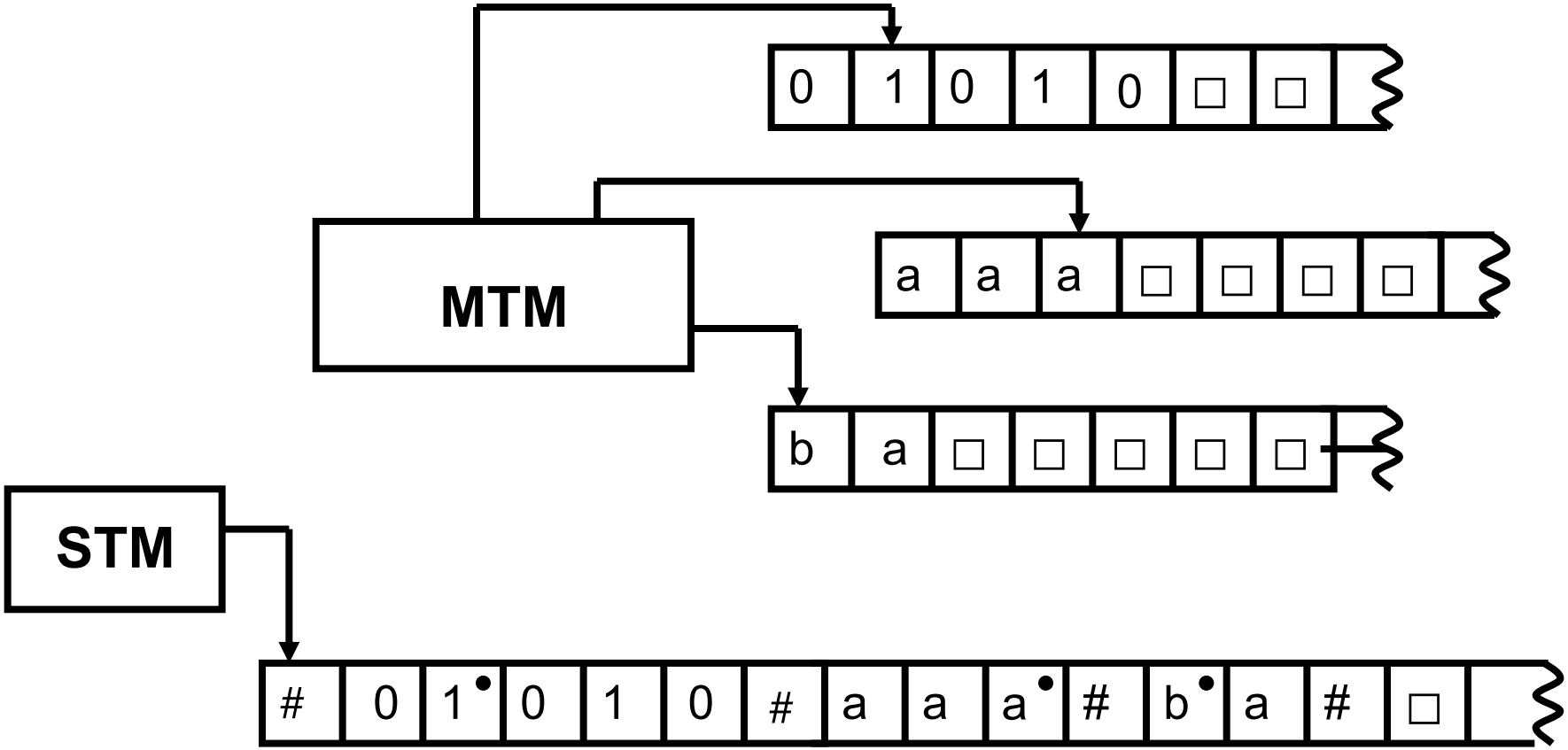
Idea: to show how to simulate MTM with STM.

- delimiter to separate the contents of the different tapes

• - mark to remember the location of the heads

Refer to Theorem 3.13 in Section 3.2 of M. Sipser's book, second edition for illustration and explanation

Multitape TM



Nondeterministic TM

At any point in computation the machine may proceed according to several choices

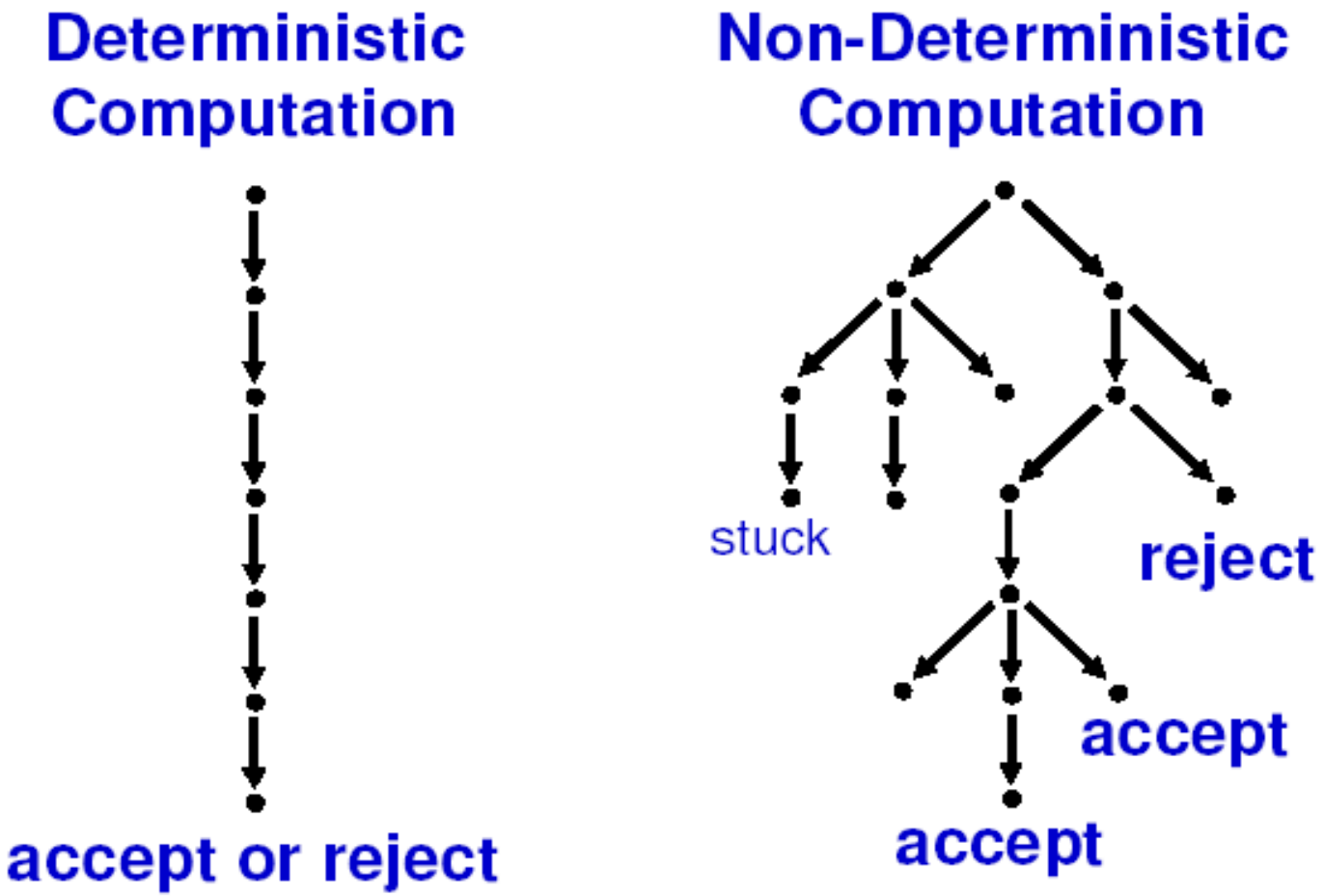
Transition function is

– $\delta : Q \times \Gamma \rightarrow 2^{(Q \times \Gamma \times \{L, R\})}$

The computation is a tree whose branches correspond to different possibilities for the machine

When does nondeterministic TM accept?

Tree of Computations



Nondeterministic TM

The computation is a tree whose branches correspond to different possibilities for the machine

When does nondeterministic TM accept?

Answer: if some branch of the computation leads to the accept state, the machine accepts

Nondeterministic TM

Theorem: Every nondeterministic TM (NTM) has an equivalent deterministic TM (DTM)

Proof: We show how to simulate any NTM with a DTM.

Idea: to have DTM try all possible branches of NTM's nondeterministic computation; if DTM finds an *accept* state on one of these branches, DTM accepts. Otherwise, DTM's simulation does not terminate

Nondeterministic TM

Ideas on how to traverse the tree?

Would Depth First Search (DFS) be a good idea?

NO

DFS goes all the way down one branch before backing up to explore other branches and might not terminate

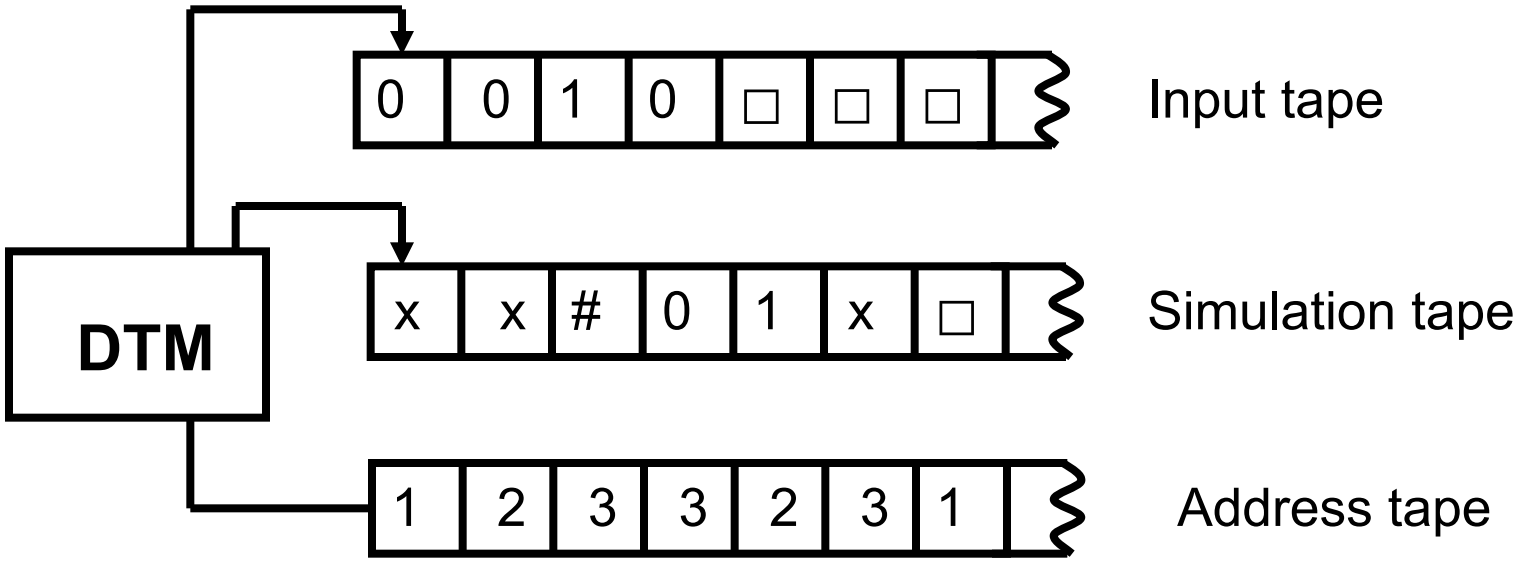
Nondeterministic TM

Ideas on how to traverse the tree?

Breadth First Search (BFS) is a good idea to use

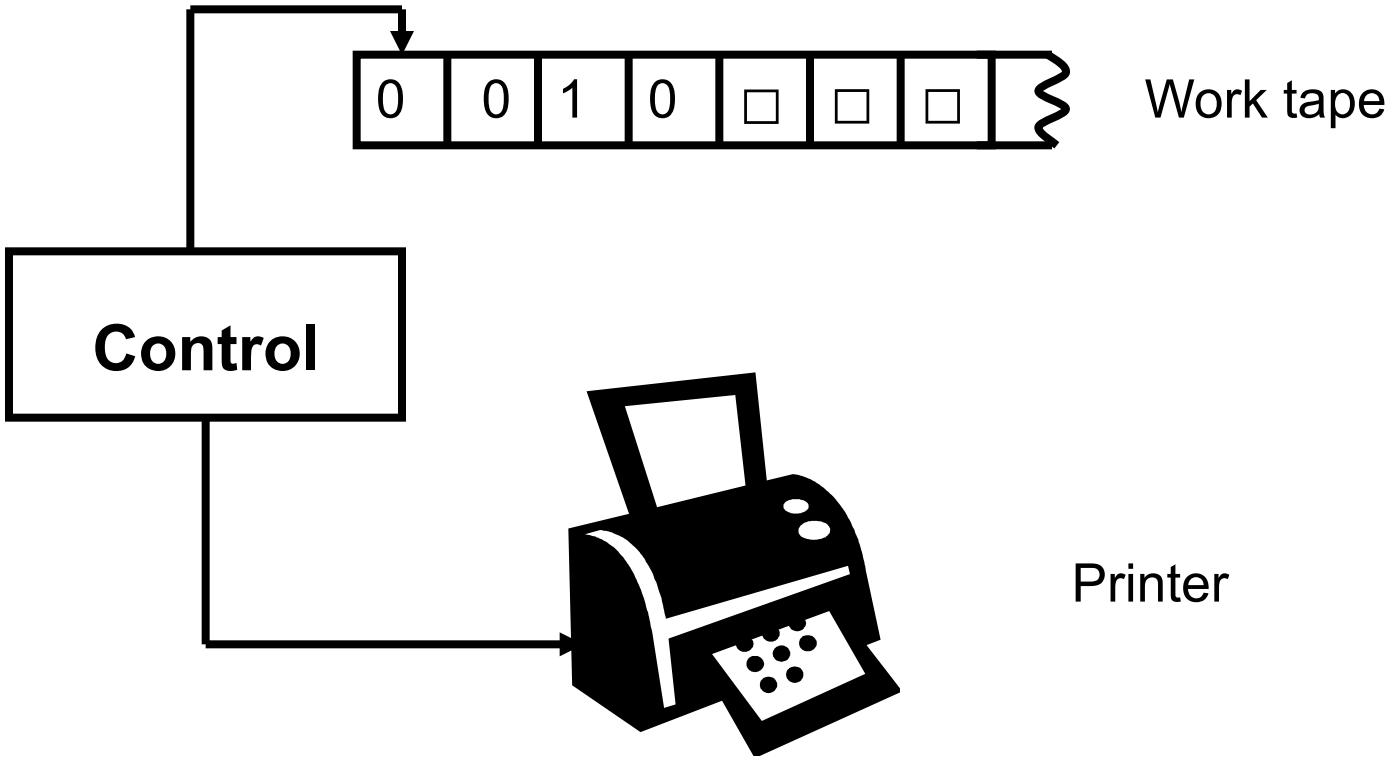
BFS explores all branches to the same depth before going to the next depth. It exhaustively checks all nodes until it finds an accept state

Nondeterministic TM



Refer to Theorem 3.16 in Section 3.2 of M. Sipser's book, second edition for illustration and explanation

Enumerator



Refer to Theorem 3.19 in Section 3.2 of M. Sipser's book, second edition for illustration and explanation

Enumerator

The language enumerated is the collection of all the strings that enumerator eventually prints out

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

Theorem: A language is Turing-recognizable iff some enumerator enumerates it

Recall Hilbert and his 10th problem

- In 1900: Posed 23 "challenge problems" in Mathematics
- The 10th problem:
  Devise an algorithm to decide if a given polynomial has an integral root.
- We now know: This is undecidable!
  - Needed a definition of "algorithm", which was given by Church and Turing (independently)

Church-Turing Hypothesis

Church-Turing Hypothesis

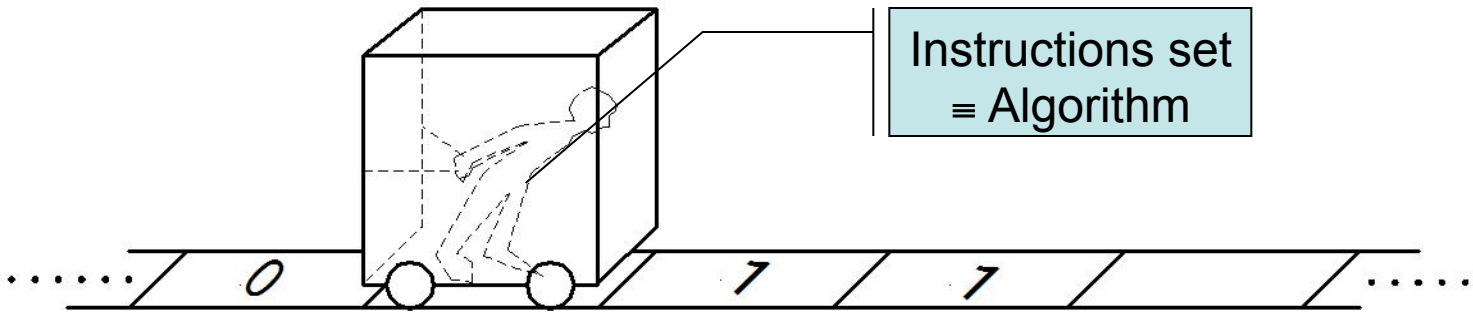
- Intuitive notion of algorithms
= Turing machine algorithms

- "Any process which could be naturally called *an effective procedure* can be realized by a Turing machine"

TM and Algorithms

Turing model serves as a precise model for the definition of algorithm

Understanding Turing Machine and Algorithms



We (and ToC) study power of algorithms to solve problems

Target: to explore the limits of algorithmic solvability (computability)

Standardized way to describe algorithms

The input to Turing machine: string

If an input object is not a string, the object MUST be represented as a string

The first step of Turing Machine is to check if the input is correct – that is that it is a string

Example

A = { <G> | G is a connected undirected graph}

Here A is a language of all strings representing graphs that are connected

<G> - encoding of a graph into it's string representation

How G can be encoded?

Consider Example 3.23 of Section 3.3 of M. Sipser's book for illustration

Example

A = { <G> | G is a connected undirected graph}

Here is the *high-level description* of TM that decides A

TM = “on input <G>:

1. Select the 1st node and label it;
2. Repeat the following until no new vertices are labeled:

   For each node, label it if it is attached by an edge to a node that is already labeled;
3. Scan all nodes. If they are all labeled, accept.

   Otherwise, reject.”

For *implementation-level* details see example 3.23 in the book