Variants of Turing Machines

Introduction and Motivation

There are many alternative definitions of Turing machines. Those are called ***variants*** of the original Turing machine. Among the variants are machines with many tapes and non deterministic machines.

Introduction and Motivation

A computational model is ***robust*** if the class of languages it accepts does not change under variants.

We have seen that DFA-s are ***robust for non determinism***.

The robustness of Turing Machines is by far greater than the robustness **of DFA-s** and **PDA-s**.

Introduction and Motivation

In this lecture we introduce several variants on Turing machines and show that all these variants have equal computational power.

In fact: All the "***reasonable***" variants have the same computational power.

Introduction and Motivation

When we prove that a TM exists with some properties, we do not deal with questions like:

How large is the TM?

Or

How complex is it to “program” that TM?

At this point we only seek existential proofs.

Example: TM with stay option (S)

Consider a Turing Machine whose head can stay on the same tape cell in a transition. The definition of the transition function for such a machine looks like this: $\delta: Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\}$, where s stands for staying.

TM with S Power Equal to Ordinary TM

We say that the computational power of two models is equal if they recognize the same class of languages.

Since each normal TM can be easily simulated by a "**TM with stay option which chooses not to stay"**, the computational power of a TM with stay option is at least as strong as the power of a normal machine.

TM with S Power Equal to Ordinary TM

In order to prove that the computational power of a TM with S is not greater than the power of a normal machine we show that for any TM with S there exists a normal TM recognizing the same language. The easiest way to prove this is to assume that M is a TM with S and to present an ordinary TM M' that ***simulates*** M.

Simulation of TM with S

The machine M' is defined just like M, except that each staying transition of M is replaced by a double transition in which M' first goes to a special additional state while moving its head right and then returns to the original state while moving its head left.

Simulation of TM with S

Now, It is very easy to prove that M and M' recognize **exactly the same language**. In fact, computations of M' are very similar to computations of M, and M' is said to ***simulate*** M.

In general, when we want to prove that two variants have the same power we show that they can ***simulate*** each other.

Implementing Memory

In many occasions a TM is required to ***store*** information that it reads from its tape. Recall that we encountered similar situations when designed DFA-s or NFA-s. The way we tackled these problems was to use states in order to store the information. Here we adopt the same technique:

Implementing Memory (cont.)

Assume for instance that TM M needs to read the first 2 input bits from its tape and write these bits beyond the input's end, where the two leftmost blanks are. One way to do this is to start by devising a TM M' that reads the first 2 input bits, searches for the input's right end and write 00 there.

Implementing Memory (cont.)

Once M' is devised we can proceed as follows:

1. Copy M' 4 times: Once for each possible combination of the two input bits.
2. At the point where M' writes 00, make each replica write its corresponding 2 bits.
3. After the 2 bits are read, move to the replica of M' representing the read input.

Implementing Memory (cont.)

Note: This technique can be used to store any **finite** amount of data.

For Example: If we want M to store a sequence of k symbols of Γ, we should copy M $|\Gamma|^k$ times. A copy for each possible sequence, and move to the replica corresponding the sequence read while scanning it.

Multitape Turing Machines

A ***multitape Turing machine*** is an ordinary Turing machine with several tapes. Initially, the input appears on tape 1 and the other tapes are blank. The transition function allows each head to behave independently:

$$\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{L, R\}^k$$

where k is the number of tapes.

Multitape Turing Machines

The expression:

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

means that if the k – tape machine M is at state $q_i \in Q$, and head i, $1 \le i \le k$, reads symbol $a_i \in \Gamma$, then the new state of M is q_j, the new symbol written by head i is b_i and head i, moves in the designated direction.

Multitape Turing Machines

Multitape TM-s appear to be stronger than ordinary TM-s. The following theorem shows that these two variants are equivalent.

Theorem

Every multitape TM has an equivalent single-tape TM.

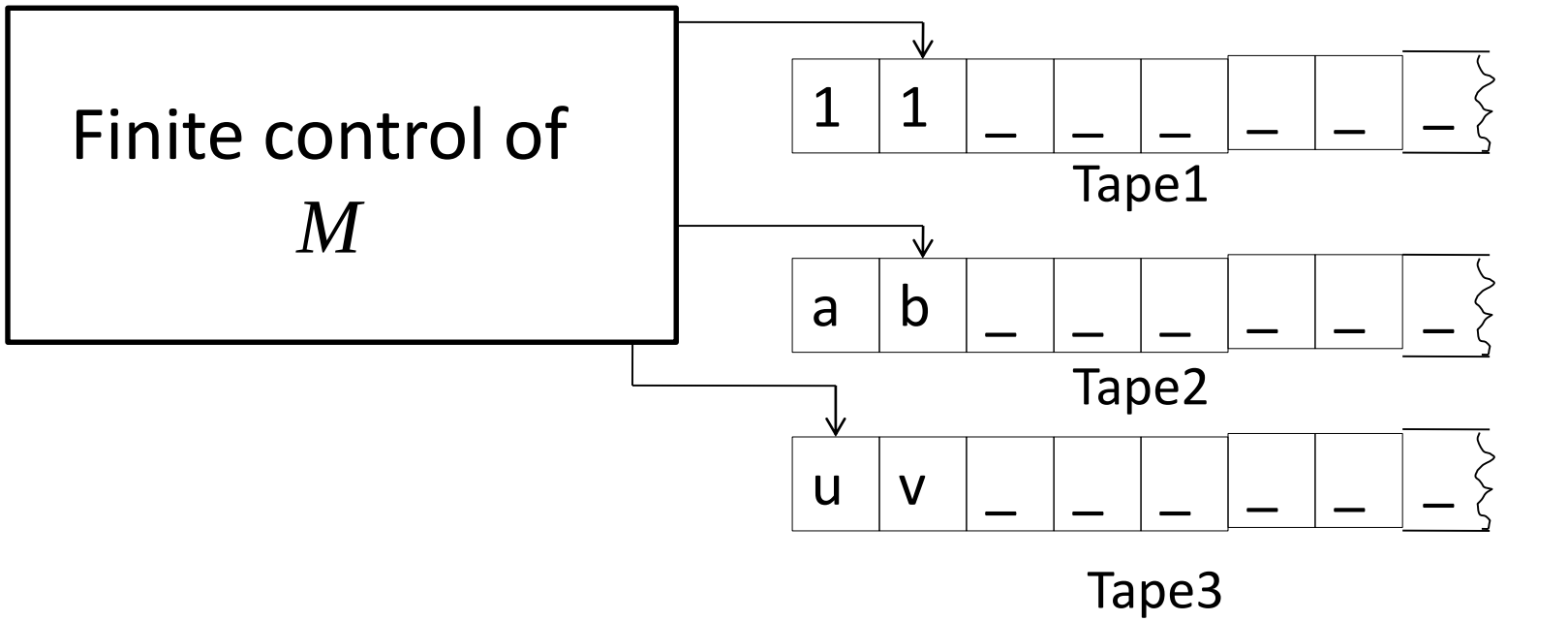
Proof

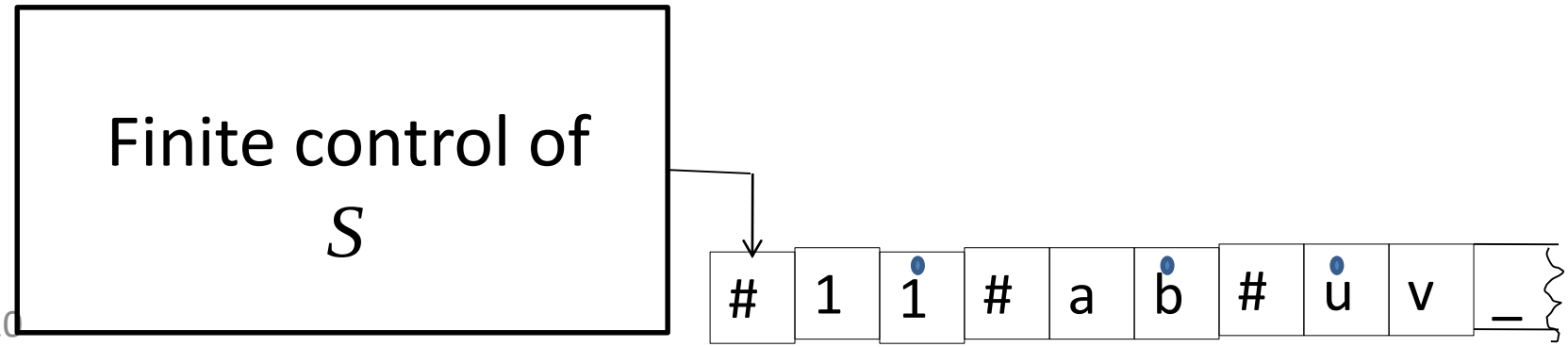
Say that M is a k- tape machine. Now we present an ordinary TM S that simulates M: The TM S stores the content of M's k tapes on its single tape, one after the other. Every pair of consecutive tape contents are separated by a special tape symbol of S, say #, which does not belong to M's tape alphabet.

Proof (cont.)

In order to keep the location of head $i, 1 \leq i \leq k$ we do the following: Let $\gamma \in \Gamma$ be a tape alphabet symbol of M. The TM S has **two symbols** corresponding to γ, denoted by γ and $\dot{\gamma}$. The "." signals that the head of the tape on which γ resides is above the symbol γ. A pictorial description on the next slide.

Simulating Three Tapes by One





Description of S

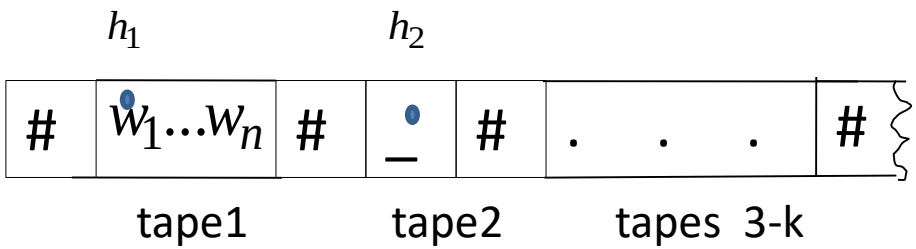
The observers of this proof should verify to themselves that all steps can be carried out by an ordinary TM.

Recall that M gets its input on its first tape and the other tapes are blank.

Here we assume that S gets M's input on its single tape.

Description of S

TM S starts its simulation of M, by preparing its tape to the described format. The tape should look like this:



Note that the leftmost blank on S's tape appears right after the $k+1$ instance of #.

Description of S (cont.)

After preparing the its tape S proceeds to scan its tape from the beginning to the (k+1)st #. During this scan S "stores" (in the way described previously) all k symbols on which its k heads reside.

Following that S makes a second pass to update its tapes according to M's transition function.

Description of S (cont.)

In its second pass S writes over all doted symbols and "moves the dots" to the new locations of the respective k heads.

In case a one of M's heads moves to the rightmost blank on its tape, the virtual head (dot) on S's corresponding tape segment would end up on the delimiting # .

Description of S (cont.)

In this case, S should shift the entire suffix of its tape, starting from the dotted delimiting # and ending on the $k+1$ #, one step right.

After this shift is completed, S writes a "dotted blank" symbol where the # symbol previously resided.

Description of S (cont.)

Following the update of its tape, including the necessary shifts, S returns to its first tape cell and assumes a state corresponding to M's next state.

Corollary

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

Proof: A Turing-recognizable language is recognized by an ordinary (single tape) TM which is a special case of a multitape TM. This proves one direction.

The other direction follows from the Theorem.

Nondeterministic Turing Machines

The transition function of a Turing machine:

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$$

The transition function of a ***Nondeterministic Turing machine***:

$$\delta : Q \times \Gamma \to P(Q \times \Gamma \times \{L, R\})$$

This definition is analogous to NFA-s and PDA-s.

Computations of Nondet. TM-s

Each computation of a Nondeterministic Turing machine is a tree, where each branch of the tree looks like a computation of an ordinary TM.

If a single branch reaches the accepting state – the Nondeterministic machine accepts, **even if other branches reach the rejecting state**.

Power of Nondeterministic TM-s

Nondeterministic TM-s appear to be stronger than ordinary TM-s. The following theorem shows that these two variants are equivalent.

Theorem

Every Nondeterministic TM has an equivalent deterministic (ordinary) TM.

Proof

Let N be a nondeterministic TM.

We look at N's computation as a (possibly infinite) tree whose nodes are configurations of N. Each branch of the tree represents a possible computation of N with input w. We will show that there exists an ordinary Turing machine D that simulates N.

Proof (cont.)

The idea of the proof is that for each input w, D should go through all possible computations of N with w and accept only if N accepts on any of its computations. Otherwise D will not terminate (will loop).

Proof (cont.)

Some of N's computations may be infinite, hence its computation tree has some infinite branches.

If D starts its simulation by following an infinite branch D may **loop forever** even though N's computation may have a different branch on which it **accepts**.

Proof (cont.)

In order to avoid this unwanted situation, we want D to execute all of N's computations **simultaneously**.

To do that, D goes on N's computation tree in a **BFS ordering**, as detailed in the next slide:

Proof (cont.)

1. Execute the first step of all computations. If any of them accepts – **accept**.
2. Execute the first 2 steps of all computations. If any of them accepts – **accept**.

…

i. Execute the first i steps of all computations. If any of them accepts – **accept**.

Proof (cont.)

The actual simulation is carried out by a 3-tape TM D.

Recall that we just showed that any 3-tape machine can be simulated by an ordinary TM.

The 1st tape of D holds the input for N.

The 2nd and third tapes are blank.

Proof (cont.)

The simulation of N by D proceeds as follows:

1. Copy the input to the 2nd tape.
2. Run a prefix of N's "next in line" computation, using the content of its 3rd tape.
   If this computation accepts – ***accept***.
3. Update the third tape to get the next in line computation.
4. Go to step 1.

Proof (cont.)

In order to complete the simulation's description we have to explain how the "computation line" is kept:

Since N is nondeterministic, it has some configurations in which it has several possible transitions:

Proof (cont.)

For every configuration of N, TM D encodes all possible N's transitions and all these transitions are enumerated.

Let b be the largest number of transitions out of any of N's configurations.

Proof (cont.)

Let $COMP_i$ be an i prefix of some computation of N on some input I. $COMP_i$ can be encoded by a string $B = b_1, b_2, \ldots, b_i$ of i b-ary digits, as follows:

TM N starts is in its initial configuration. Digit b_1 is the number of the actual transition made by N on its 1st step of $COMP_i$.

Proof (cont.)

For $2 \leq j \leq i$, b_j is the number of the actual transition made by N on its j^{th} step of $COMP_t$.

Proof (cont.)

For example: The string 421 encodes a prefix of length 3 in which on the 1st step N takes the 4th enumerated choice, on its second step it takes the 2nd enumerated choice and on its third step it takes the 1st enumerated choice.

Note: Some configurations may have less than b choices, hence not every b-ary number represents a computation prefix.

Proof (cont.)

The simulation of N by D proceeds as follows:

1. Write 1 on the third tape.
2. Copy the input from tape 1 to tape 2.
3. Execute N using Tape 2 as input and following the computation choices specified in Tape 3
   - If sequence of choices represents an accepting computation for N (i.e. N enters an Accept state), then D halts and **accepts**
   - Else (i.e. a number represents an infeasible or rejecting computation) continue with the next step
4. Update the third tape to the next b-ary number
5. Go to step 2.

The Church-Turing thesis

In this lecture we showed several variants of TM-s are equivalent. Over the years it has been proved that many similar and even not so similar computational models are equivalent, namely they have the same computational power. One particular model is called λ-calculus of **Church**.

The Church-Turing thesis

The λ-calculus of **Church** and Turing machines

are the first 2 **formal models** for a mathematical notion that was informal, intuitive and vague for centuries:

The notion of an

A L G O R I T H M

The Church-Turing thesis

Furthermore: It has been shown that these two definitions are equivalent:

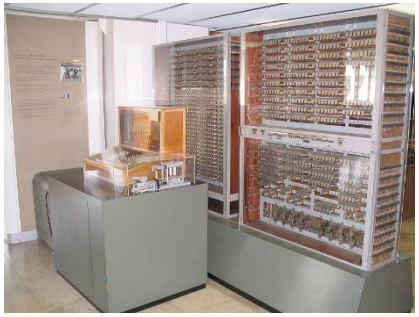
The Church Turing Thesis says that

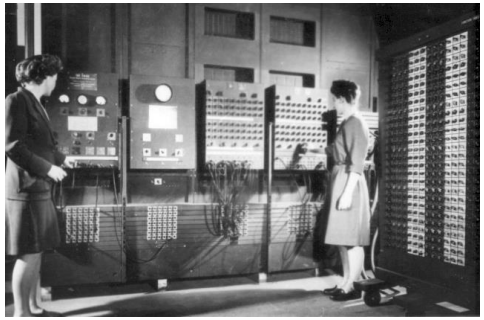
Intuitive Notion of Algorithm

equals

Turing Machines or λ-Calculus

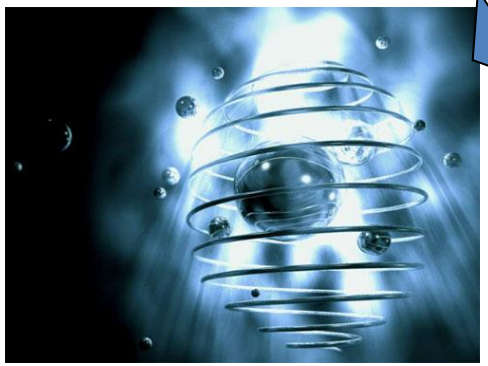
The Church-Turing Thesis





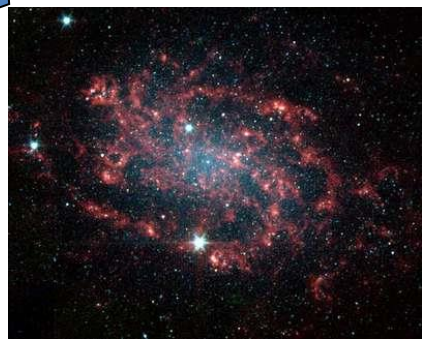
Turing Machine

quantum computing



DNA computing

cosmic computing

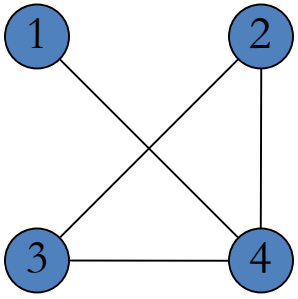


Programming Turing Machines

$L_4 = \{\langle G \rangle : G \text{ is a connected undirected graph}\}$

Q: How do we feed a graph into a Turing Machine?

A: We represent it by a string, e.g.



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

Convention for describing graphs:

(nodes) (edges)

no node must repeat

edges are pairs $(\text{node}_1, \text{node}_2)$

Programming Turing Machines

$L_4 = \{\langle G \rangle : G \text{ is a connected undirected graph}\}$

To check if $\langle G \rangle$ is in L_4:

0. Verify that $\langle G \rangle$ is the description of a graph
   (no vertex repeats; edges only go between nodes)

1. Mark the first node of G

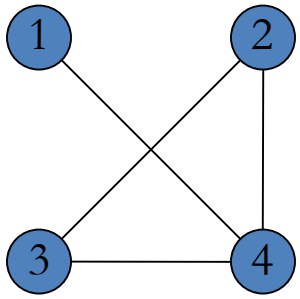
2. Repeat until no new nodes are marked:

   For each node, mark it if it is attached to an already marked node

3. If all nodes are marked accept, otherwise reject.

Programming Turing Machines

$L_4 = \{\langle G \rangle : G \text{ is a connected undirected graph}\}$



(i̲,2̲,3,4)((1,4)(2,3)(3,4)(4,2))

(i̲,2,3̲,4)((1,4)(2,3)(3,4)(4,2))

(i̲,2,3,4̲)((1,4)(2,3)(3,4)(4,2))

(i̇,2,3,4̇)((1,4)(2,3)(3,4)(4,2))

(i̲,2̲,3,4̇)((1,4)(2,3)(3,4)(4,2))

(i̇,2̲,3,4̲)((1,4)(2,3)(3,4)(4,2))

(i̇,2̇,3,4̇)((1,4)(2,3)(3,4)(4,2))

(i̲,2̇,3̲,4̇)((1,4)(2,3)(3,4)(4,2))

(i̇,2̲,3̲,4̇)((1,4)(2,3)(3,4)(4,2))

(i̇,2̇,3̇,4̇)((1,4)(2,3)(3,4)(4,2))