CS5371
Theory of Computation

Lecture 11: Computability Theory II
(TM Variants, Church-Turing Thesis)

Objectives

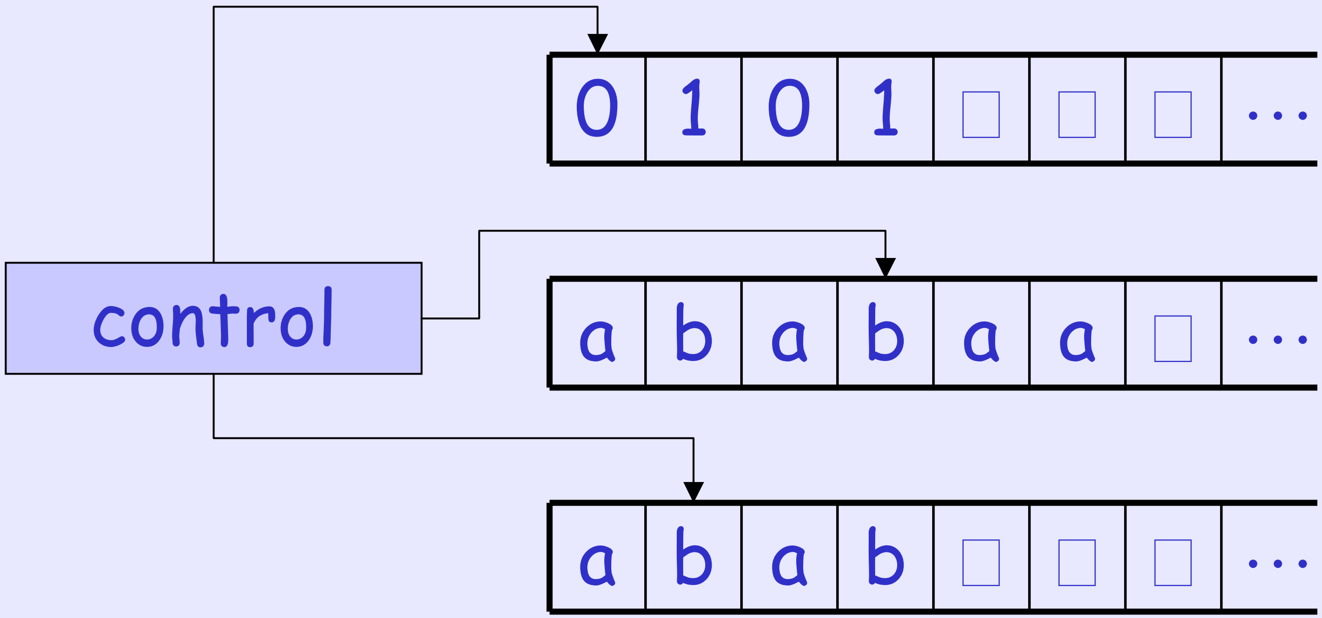
- Variants of Turing Machine
  - With Multiple Tapes
  - With Non-deterministic Choice
  - With a Printer
- Introduce Church-Turing Thesis
  - Definition of Algorithm

Variants of TM

- Different definition from the original TM
- However, they recognize the same set of languages as TM (Just like NFA vs DFA)
- One example is: TM such that the tape head can move left, right, or it can stay
  - This TM with stay put, recognize the same set of languages as TM (Why?)
  - Because by replacing each stay transition with two transitions (one right and one left), we can convert this TM into an equivalent TM without stay put.
- There are more variants…

Variant 1: Multi-Tape TM

□ = blank symbols



It is like a TM, but with several tapes

Multi-tape TM (2)

- Initially, the input is written on the first tape, and all other tapes blank
- The transition function of a k-tape TM has the form

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

- Obviously, given a TM, we can find a k-tape TM that recognizes the same language
- How about the converse?

Multi-tape TM (3)

Theorem: Given a k-tape TM, we can find an equivalent TM (that is, a TM that recognizes the same language).

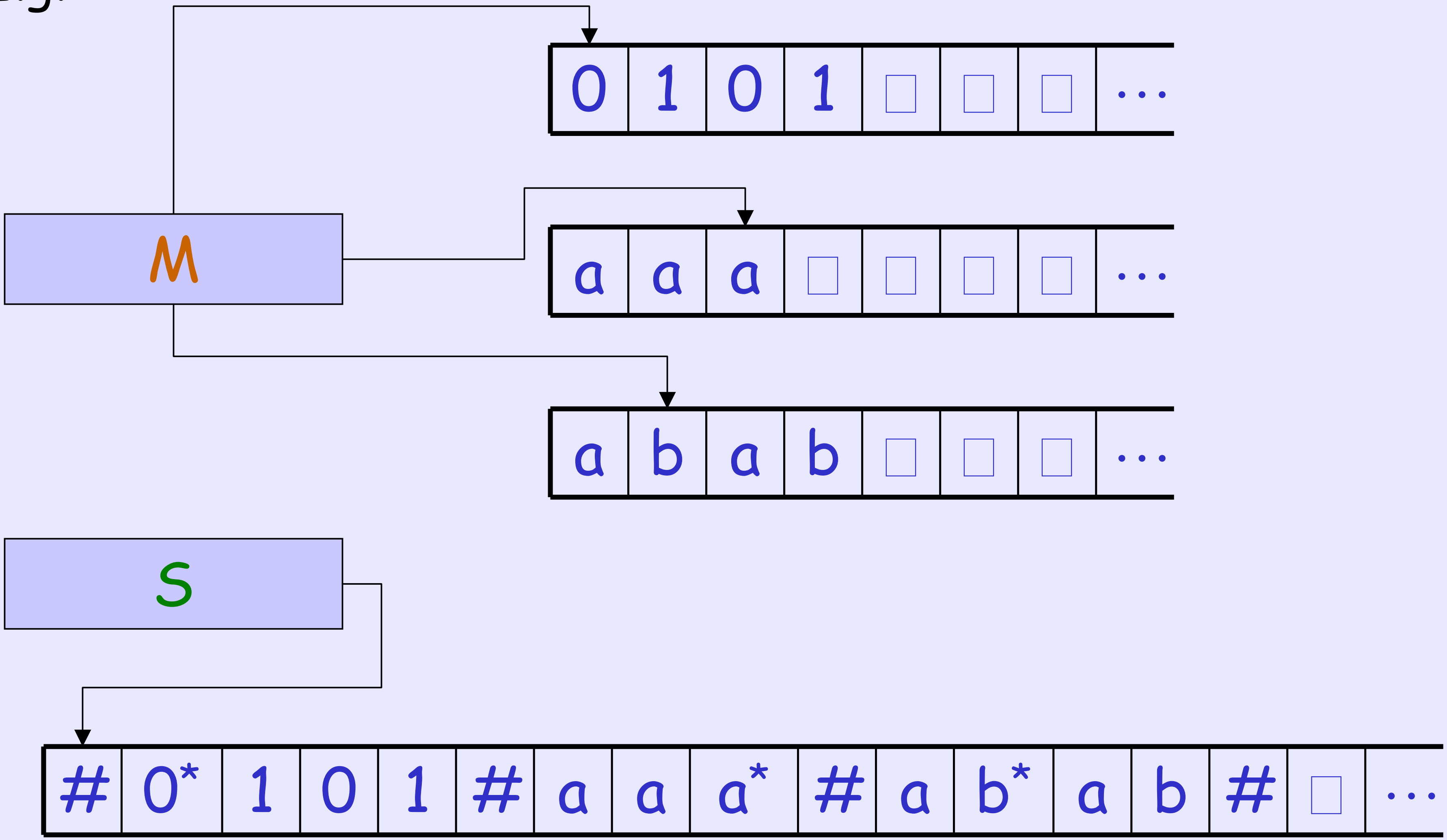
Proof: Let M be the k-tape TM (with multiple tape). We show how to convert M into some TM S (with single tape).

Multi-tape TM (4)

1. To simulate k tapes, S separates the contents of different tapes by #
2. To simulate the tape heads, S marks the symbol under each tape head with a star (The starred symbols are just new symbols in the tape alphabet of S)

We can now think of the tape of S containing k "virtual" tapes and tape heads

E.g.



Note: $\Gamma_M = \{0,1,a,b,\square\}$ and $\Gamma_S = \{0,1,a,b,\square,\#,0^*,1^*,a^*,b^*,\square^*,\#^*\}$

Multi-tape TM (5)

On input $w = w_1 w_2 \ldots w_n$

Step 1. S stores in the tape

$$\# \, w_1^* w_2 \ldots w_n \, \# \, \square^* \# \, \square^* \# \, \ldots \, \#$$

Step 2. To simulate a single move, S scans from the first # to the $(k+1)^{st}$ #, to find out what symbols are under each virtual tape head. Then, S goes back to the first # and updates the virtual tapes according to the way that M's transition function will do

Multi-tape TM (5)

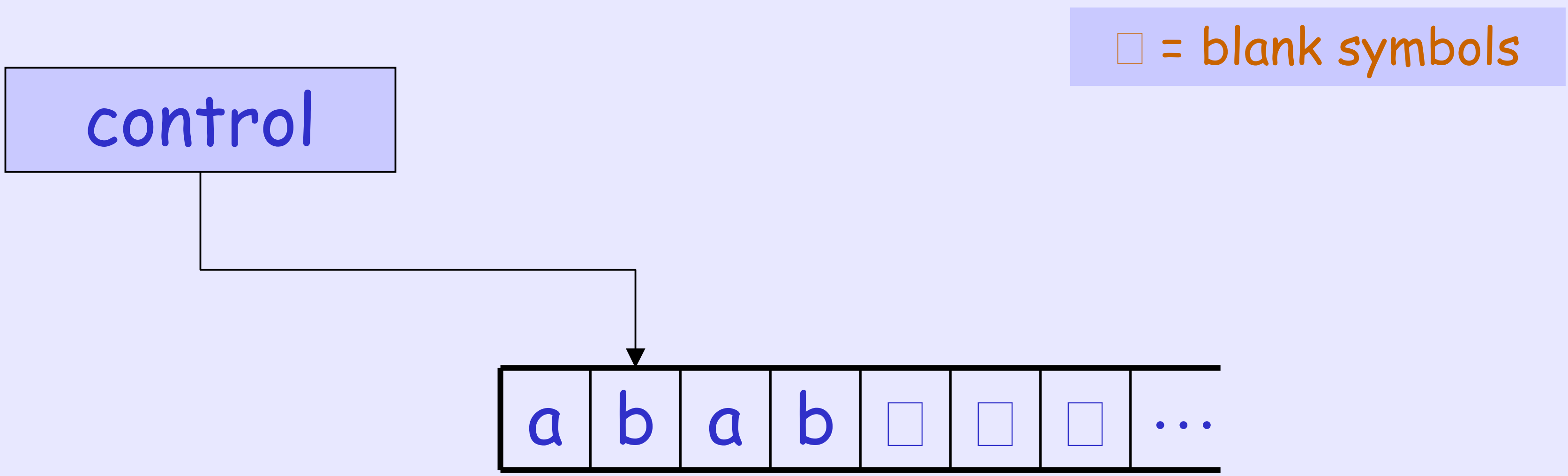
Questions: (1) What does it mean if the virtual tape head, after the transition, has moved to # ? (2) Then, what should we do?

Answer:

(1) This means that we have moved to the unread blank portion of the virtual tape.

(2) In this case, we overwrite # by □*, shifts the tape contents of S from this cell (i.e., #) to the rightmost #, one unit to the right. After that, comes back and continues the simulation

Variant 2: NTM



It is like a TM, but with non-deterministic control

NTM (2)

- The transition function of NTM has the form

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

- For a given input w, we can describe the computation of NTM as a tree, where the root represents the start configuration, and the children of a node C are the possible configurations that can be yielded by C
- The NTM accepts the input w if some branch of computation (i.e., a path from root to some node) leads to the accept state

NTM (3)

Theorem: Given an NTM, we can find a TM that recognizes the same language.

Proof: Let N be the NTM. We show how to convert N into some TM D. The idea is to simulate N by trying all possible branches of N's computation. If one branch leads to an accept state, D accepts. Otherwise, D's simulation will not terminate.

NTM (4)

- To simulate the search, we use a 3-tape TM for D
  - first tape stores the input string
  - second tape is a working memory, and
  - third tape "encodes" which branch to search
- What is the meaning of "encode"?

NTM (5)

- Let $b = |Q \times \Gamma \times \{ L, R \}|$, which is the maximum number of children of a node in N's computation tree.
- We encode a branch in the tree by a string over the alphabet {1,2,…,b}.
  - E.g., 231 means starting from the root r, goes to the r's 2nd child c, then goes to c's 3rd child d, then goes to d's 1st child

NTM (6)

On input string w,

Step 1. D stores w in Tape 1 and □ in Tape 3

Step 2. Repeat

2a. Copy Tape 1 to Tape 2

2b. Simulate N using Tape 2, with the branch of computation specified in Tape 3. Precisely, in each step, D checks the next symbol in Tape 3 to decide which choice to make. (Special case …)

NTM (7)

2b [Special Case].

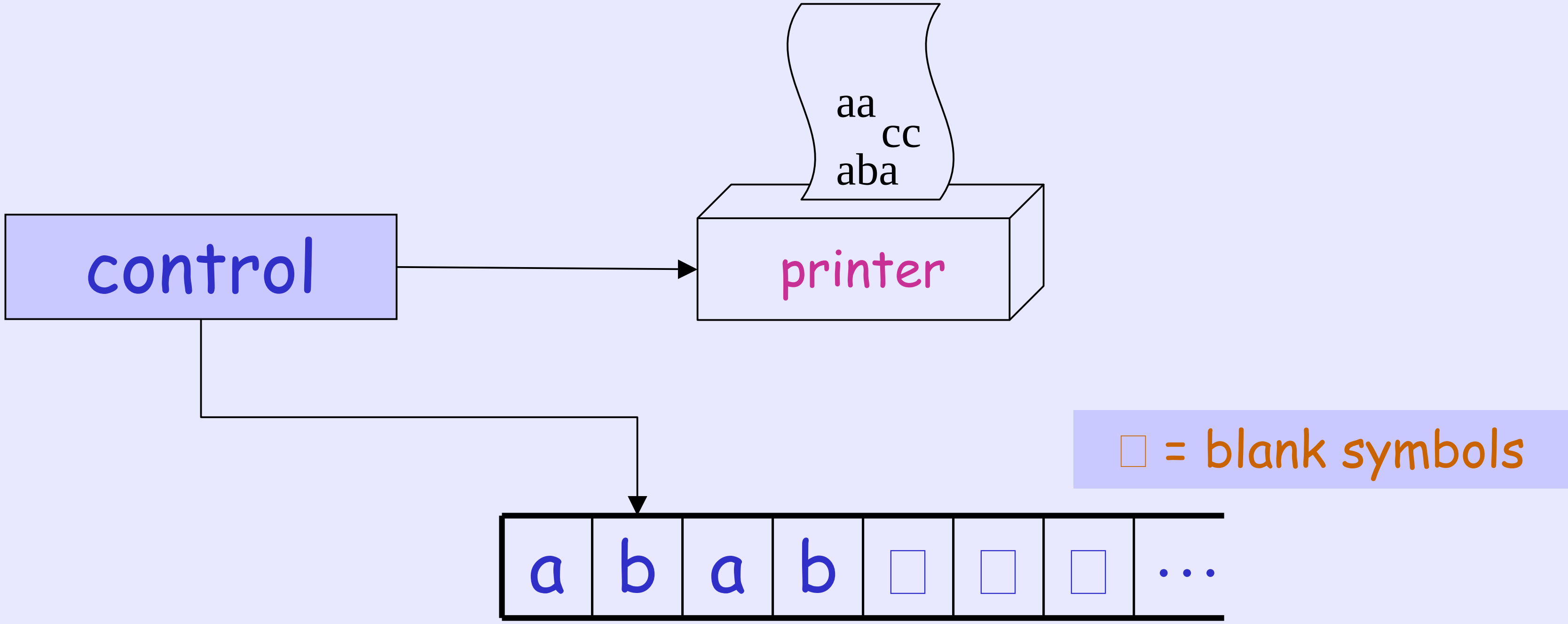
1. If this branch of N enters accept state, accepts w
2. If no more chars in Tape 3, or a choice is invalid, or if this branch of N enters reject state, D aborts this branch

2c. Copy Tape 1 to Tape 2, and update Tape 3 to store the next branch (in Breadth-First Search order)

NTM (8)

- In the simulation, D will first examine the branch ε (i.e., root only), then the branch 1 (i.e., root and 1st child only), then the branch 2, and then 3, 4, …, b, then the branches 11, 12, 13, …, 1b, then 21, 22, 23, …, 2b, and so on, until the examined branch of N enters an accept state
- If N does not accept w, the simulation of D will run forever
- Note that we cannot use DFS (depth-first search) instead of BFS (why?)

Variant 3: Enumerator



It is like a TM, but with a printer

Enumerator (2)

- An enumerator E starts with a blank input tape
- Whenever the TM wants to print something, it sends the string to the printer
- If the enumerator does not halt, it may print an infinite list of strings
- The language of E = the set of strings that are (eventually) printed by E
  - Note: E may generate strings in any order, and with repetitions

Enumerator (3)

Theorem: Let L be a language. (1) If L is enumerated by some enumerator, there is a TM that recognizes L. (2) If L is recognized by some TM, there is an enumerator that enumerates L.

Enumerator (4)

Proof of (1): Let E be the enumerator that enumerates L. We convert E into a TM M:

On input w:

Step 1. Run E. Whenever E wants to print, compare the string with w. If they are the same, accept w. Otherwise, continue to run E.

Thus, M accepts exactly strings that is on E's list.

Enumerator (5)

Proof of (2): Let M be the TM that recognizes L . We use M to construct an enumerator E that enumerates L:

Ignore the input (as E does not need an input):

Step 1. Repeat for i = 1,2,3,... (forever)

1a. Run M for i steps on the first i strings in Σ^* (sorted by length, then lex order) E.g., when $\Sigma = \{0,1\}$, the order of strings is: ε, 0, 1, 00, 01, 10, ...

1b. If M accepts a string w, print w

Enumerator (6)

- In the Proof of (2), we see that if a string is accepted by M, it will be printed by E eventually (why?), though with infinitely many times (why?)

- Note: Turing-recognizable language is also called recursively enumerable language. The latter term actually originates from enumerator

Hilbert's 10^{th} Problem

- In 1900, David Hilbert delivered a famous talk in the Internal Congress of Mathematicians (ICM) in Paris
- He identified 23 math problems which he thinks is important in the coming century
- The 10^{th} Problem asks: Given a multi-variable polynomial with integral coefficients (such as $P(x,y,z) = 6x^3yz^2 + 3xy^2 - 27$). Is there an algorithm that tells if there are any integral root for $P(x,y,z) = 0$? [E.g., in this case, $x=y=1$, $z=2$ is a possible integral root for $P(x,y,z)=0$]

Hilbert's 10^{th} Problem (2)

- However, what is meant by an algorithm?
- Roughly speaking, one meaning of algorithm is: a set of steps for solving a problem, such that when a human provided with unlimited supply of pencils and papers, he can blindly follow these steps and solve the problem
- There is no precise definition, until in 1936, two separate papers, one from Alonzo Church and one from Alan Turing, try to define it

Church-Turing Thesis

- Turing requires that for each step in the algorithm, we can implement it by a TM
- Church uses another definition of algorithm based on a notational system called λ-calculus
- Surprisingly, these two definitions are shown to be equivalent!! (That is, a problem P can be solved by some algorithm with Turing's definition if and only if P can be solved by some algorithm with Church's definition)
  - Later (in 1970), Yuri Matijasevič proves that, under their definition, no algorithm can test whether a multi-variable polynomial has integral root

Church-Turing Thesis (2)

- Also, it seems that all problems that we can think of solvable by an "algorithm" (with our "intuitive" and "non-precise" definition) are exactly the problems solvable by TM
- Therefore, Steven Kleene (1943) proposes this thesis, or conjecture in his paper, which is now known as the Church-Turing Thesis:

  "If a problem is intuitively solvable, it can be solved by TM"

Solving Problem by TM (example)

- Let A be the language

  $\{ \langle G \rangle \mid G = \text{undirected connected graph}\}$

  where $\langle G \rangle$ the encoding of G
- That is, given an undirected graph G, we want to determine if G is connected
- How to solve it by TM?

Solving Problem by TM (example)

M = "On input $\langle G \rangle$

Step 1. Select first node of G and mark it

Step 2. Repeat the following stage until no new nodes are marked:

2a. For each node in G, mark it if it is attached to a marked node

Step 3. Scan all nodes. If all are marked, accept. Otherwise, reject.

Next Time

- Decidable Language
  - Can be decided by some algorithm
- Undecidable Language
  - No algorithm can decide it