Theoretical Foundations of Computer Science

Lecture 5

Turing Machines

Outline

- Turing machines
  - Formal definition
  - Examples
- Variants of TM
  - Multi-tape
  - NTM
  - Equivalence of models
- Church-Turing Thesis

Unit Learning Outcomes

- Synthesize FA, PDA, CFG, and TMs with specific properties, and convert from one form to another.
- Understand recognisability and decidability, use the construction & mapping reducibility techniques to prove a problem decidable or undecidable.

Assessment Criteria

- **Model** a specification expressed in English or Mathematics as a TM.
- **Explain** the operation of a machine on an input string.
- **Show** that a string belongs to a language.
- **Classify** a language as Turing Decidable and/or Turing Recognizable

TURING MACHINES

Prior Models of Computation

Turing machine Concept & Example

Formal Definition

Models of computation

- Finite automata
  - good model for devices with a small memory
  - useful for pattern matching

- Pushdown automata
  - good model for devices with unlimited memory that is usable only as a stack or is otherwise access limited
  - useful for considering compilers

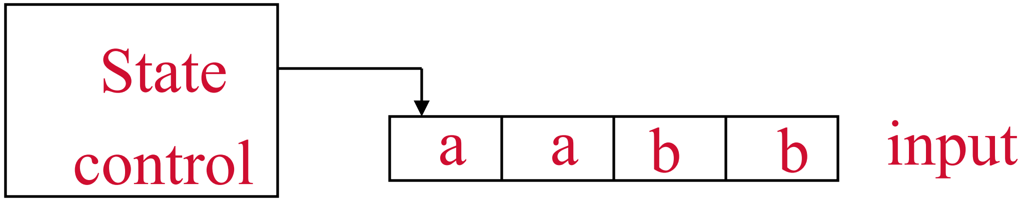
- FA and PDA are not suitable for general purpose computation
  - some simple tasks can’t be done using these

Turing machines

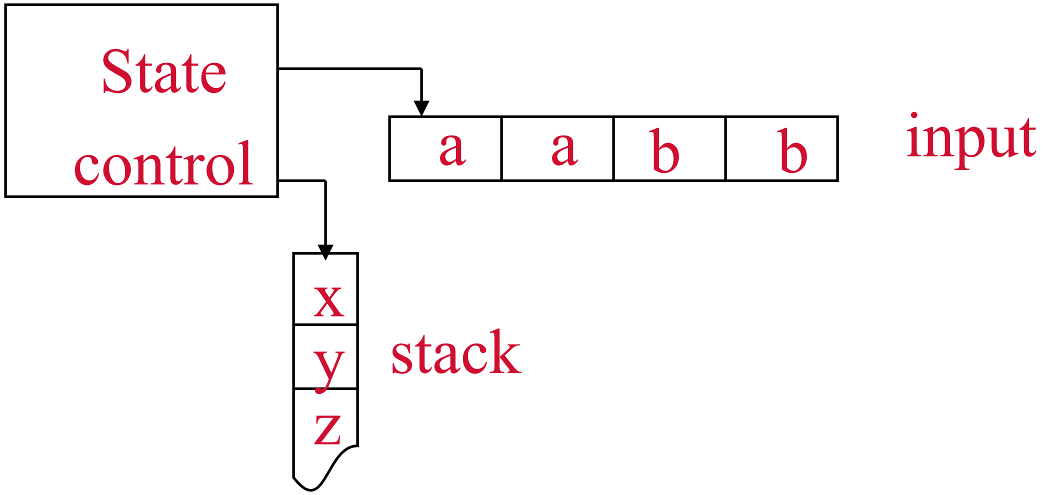
- Proposed by Alan Turing in 1936
- Similar to a finite automaton but with an unlimited and unrestricted memory
- A more accurate model of a general purpose computer
  - it can do everything that a real computer can do
  - even a Turing machine cannot solve certain problems
    - these problems are beyond the theoretical limits of computation

Turing Machines

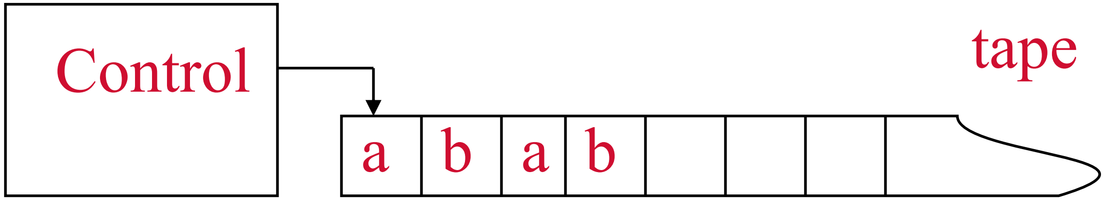
- Finite automaton



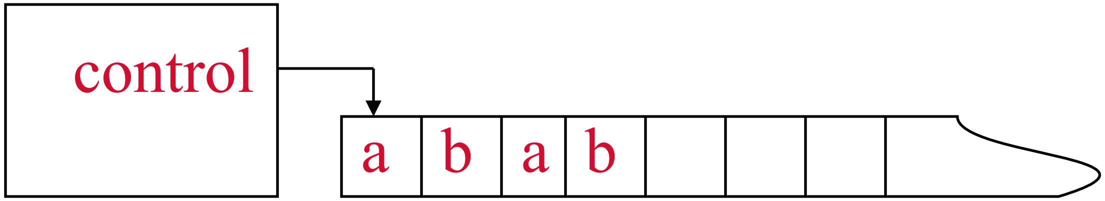
- Push-Down automaton



- Turing Machine



Schematic of a Turing machine



- Infinite tape as the unlimited memory
- Tape head can read and write symbols and move around the tape, both left and right
- Initially tape contains input string on the leftmost positions followed by blanks
- Writes information to be stored on the tape
- Computes until it decides to produce output
  - Outputs accept and reject obtained by entering designated accepting and rejecting states
  - If it doesn't enter an accepting or rejecting state it will go on forever

Machine Comparison

	FA	PDA	TM
Memory	States are memory (finite).	States and stack (push/pop).	States and tape (unrestricted access).
Movement	Read input from left to right only		Move freely left and right.
Input	Read only.		Read / write.
Initial	Head starts on leftmost square of input.		
Termination	When input has been processed.		When accept or reject state reached.

Curtin University of Technology

Differences from finite automata

- Turing machine can write on tape and read from it
- Read-write head can move both left and right
- Tape is infinite
- Special states for accepting and rejecting take immediate effect

- The Turing machine we are looking at (initially) is **deterministic.**

Example

- A Turing machine M_1 for testing membership in the language $B = \{w\#w \mid w \in \{0,1\}^*\}$
  - imagine a mile long input of millions of characters
  - goal is to determine if the input is a member of B
  - allowed to move back and forth over the input and put marks on it
  - obvious strategy:
    - zigzag to the corresponding places on the two sides of # and determine if they match
    - use marks to keep track of which places correspond

Curtin University of Technology

Algorithm for M_1

1. [Initialize] Scan the input to ensure it contains a single #.
   - If not reject.
2. [Traverse] Zigzag across the tape to corresponding positions on either side of # to check if they contain the same symbol.
   - If not reject.
   - Cross off symbols as they are checked to keep track of symbols that correspond.
3. [Termination]When all symbols to the left of # have been crossed off, check for any remaining symbols to the right of #.
   - If any symbols remain, reject.
   - Otherwise accept.

Curtin University of Technology

M_1 computing on an input

- Snapshots of M_1's tape while computing in steps 2 and 3
- Makes multiple passes over the input
- In each pass it matches one character on each side of #

0 1 1 0 0 0 # 0 1 1 0 0 0 ⊔ …

x 1 1 0 0 0 # 0 1 1 0 0 0 ⊔ …

x 1 1 0 0 0 # x 1 1 0 0 0 ⊔ …

x 1 1 0 0 0 # x 1 1 0 0 0 ⊔ …

x x 1 0 0 0 # x 1 1 0 0 0 ⊔ …

x x x x x x # x x x x x x ⊔ …

accept

M_1 computing on an input

- To keep track of the matched symbols, it crosses them off
- When all symbols on the left of # are crossed off, if no symbols remain to the right of #, accept state; else reject

0 1 1 0 0 0 # 0 1 1 0 0 0 ␣ …

x 1 1 0 0 0 # 0 1 1 0 0 0 ␣ …

x 1 1 0 0 0 # x 1 1 0 0 0 ␣ …

x 1 1 0 0 0 # x 1 1 0 0 0 ␣ …

x x 1 0 0 0 # x 1 1 0 0 0 ␣ …

x x x x x x # x x x x x x ␣ …

accept

Formal definition of a TM

- Analogous to those of FA and PDA
- Formal definition is seldom used in practice to describe a particular TM
  - such descriptions tend to be too large
- Transition function δ
  - tells us how a TM gets from one step to the next

- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L,R\}$
  - When the TM is in a state q, and the head is over a tape square containing symbol a,
  - if $\delta(q,a) = (r,b,L)$, the machine writes b replacing the a and goes to state r.
  - L indicates a move of the head to the left on the tape

Formal definition

- A TM is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, where:
  - Q is a set of states
  - Σ is the input alphabet not containing the special blank symbol
  - Γ is the tape alphabet, that includes Σ and the blank symbol
  - $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L,R\}$ is the transition function,
  - $q_0 \in Q$ is the start state,
  - $q_{accept} \in Q$ is the accept state
  - $q_{reject} \in Q$ is the reject state, where $q_{accept} \neq q_{reject}$

- Note that there is only a single accept and reject state. Why?

Curtin University of Technology

TURING MACHINE COMPUTATION

Computation

Turing Recognizable Language

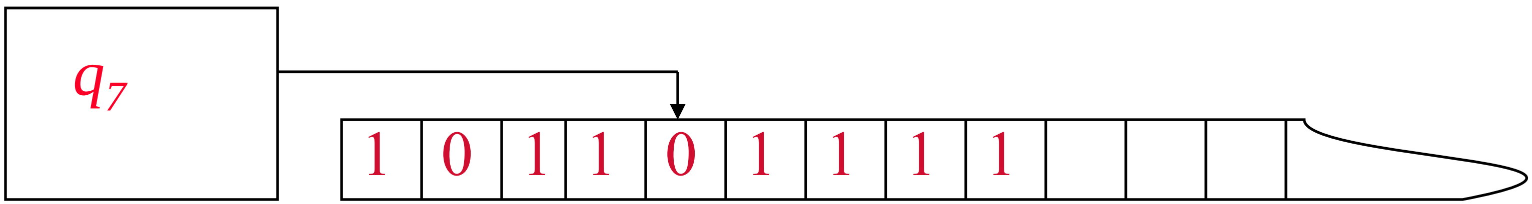
Turing Decidable Language

Example Computations

Computation of a TM

- Initially, *M* receives input $w=w_1w_2\ldots w_n \in S^*$ on the leftmost n squares of the tape
  - the rest of the tape is blank (filled with blank symbols)
  - the head starts on the leftmost square
  - first blank on the tape marks end of input
- Once *M* starts, computation proceeds according to the transition function
  - *M* cannot move to the left of the left hand end of the tape even if a transition function indicates L
  - How to deal with this? Can make the start with a special symbol (*e.g.* $)
- Computation continues until it reaches either the accept or reject state
  - if neither occurs, M does not halt

Configuration of TM



- A setting of current state, tape contents and head location is called a configuration
- As a TM computes its configuration changes
- Example: $1011q_7 01111$
  - current state is q_7
  - tape contents are 101101111
  - head is on the second 0 from the left

Curtin University of Technology

Computation of TM

- For a state q, and two strings *u* and *v* over the tape alphabet Γ, we write the configuration as *u*q*v* when,
  - q is current state
  - *uv* is current tape contents
  - current head location is first symbol of *v*
- Configuration C_1 yields configuration C_2
  - if the TM can go from C_1 to C_2 in a single step
- Let $a,b \in \Gamma$ and $u,v \in \Gamma^*$
  - $ua\ q_i\ bv$ yields $u\ q_j\ acv$ if $\delta(q_i,b) = (q_j,c,\mathrm{L})$
  - $ua\ q_i\ bv$ yields $uac\ q_j\ v$ if $\delta(q_i,b) = (q_j,c,\mathrm{R})$
- Special cases at the ends of the configuration
  - For the left-hand end, $q_i\ bv$ yields $q_j\ cv$ if transition is left moving
  - For the right-hand end, $ua\ q_i$ is equivalent to $ua\ q_i␣$

Curtin University of Technology

Computation of TM

- Start configuration of *M* on input *w* is $q_0\, w$
  - q_0 is start state
  - head in leftmost position on tape
- Accepting configuration has state q_{accept}
  - similarly rejecting configuration has state as q_{reject}
- Accepting and rejecting configurations are halting configurations
- A TM *M* accepts input *w* if a sequence of configurations $C_1, C_2, \ldots, C_k$ exists where
  - C_1 is the start configuration of *M* on *w*,
  - each C_i yields C_{i+1}, and
  - C_k is an accepting configuration.
- The collection of strings that *M* accepts is the language of *M*, denoted by L(*M*).

Curtin University of Technology

Turing-recognizable languages

- A language that some TM recognizes is called <u>Turing-recognizable</u> (or recursively enumerable)
  - When we start a TM on an input it may accept, reject or loop
  - Looping may involve simple or complex behaviour that never leads to a halting state
- TMs that halt on all inputs are called <u>deciders</u>
- A decider that recognizes some language is said to decide that language
- A language is <u>decidable</u> if a TM decides it
  - also called a recursive language
- **Every decidable language is Turing-recognizable**
  - But, some Turing-recognizable (also known as semi-decidable) languages are not decidable

Curtin University of Technology

TM examples

- Formal descriptions of TM can be cumbersome, except for very small machines
- Higher level descriptions are precise enough and easier to understand
- Every higher level description is just a short hand for its formal equivalent
- Next example illustrates the connection between higher level and formal descriptions

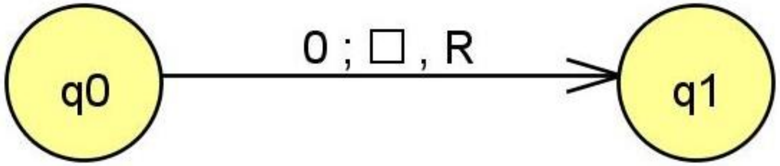
Example

- A TM to recognize the language of all strings of 0s whose length is a power of 2
- M_2 = "On input string w:
  1. Sweep left to right across the tape, crossing off every second '0'.
  2. When a space is detected:
     a) If the tape contained a single 0, *accept*.
     b) Otherwise, if the number of 0s was odd, *reject*.
  3. Return the head to the left hand end of the tape.
  4. Go to 1."

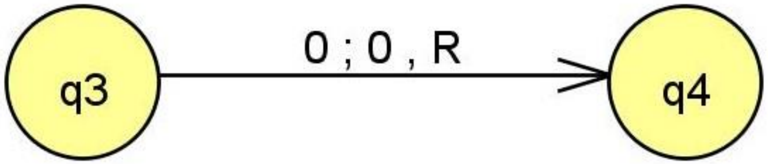
Curtin University of Technology

Formal description

- $M_2 = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$.
- $Q = (q_0, q_1, q_2, q_3, q_4, q_{accept}, q_{reject})$.
- $\Sigma = \{0\}$
- $\Gamma = \{0, x, \sqcup\}$
- δ described by a state diagram (next slide).



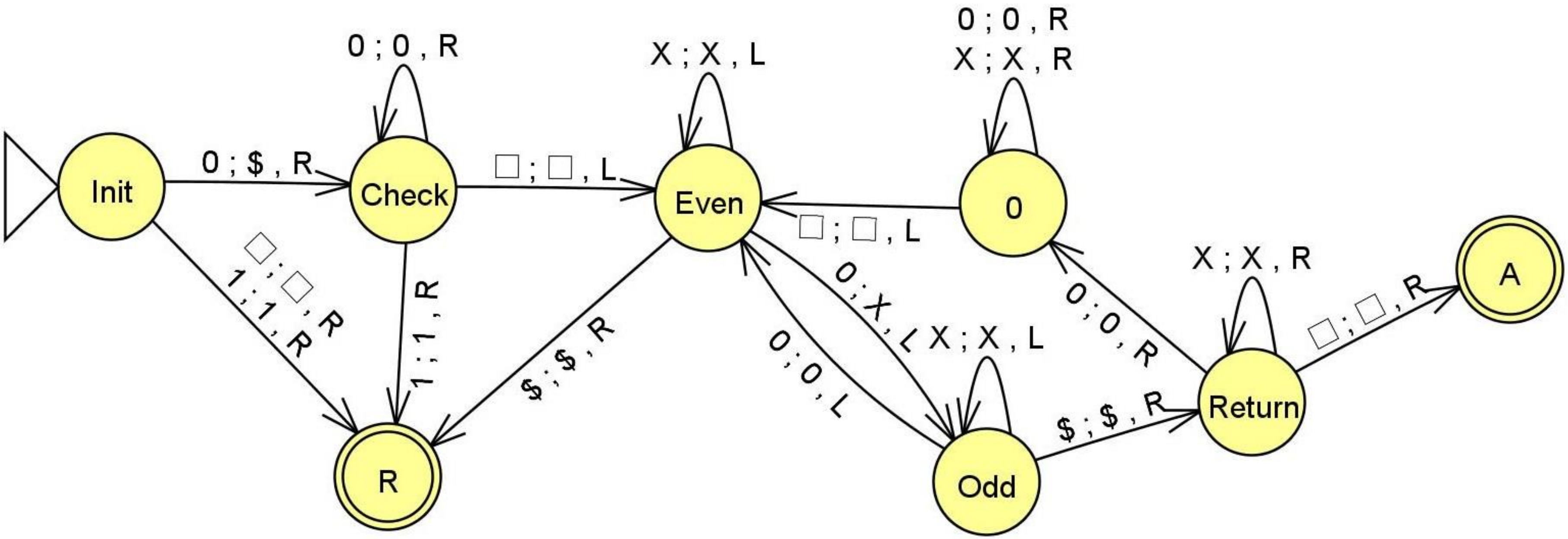
$\delta(q_0,0) = (q_1, \sqcup, R)$



can be shortened to 0; R

Meanings of transition labels

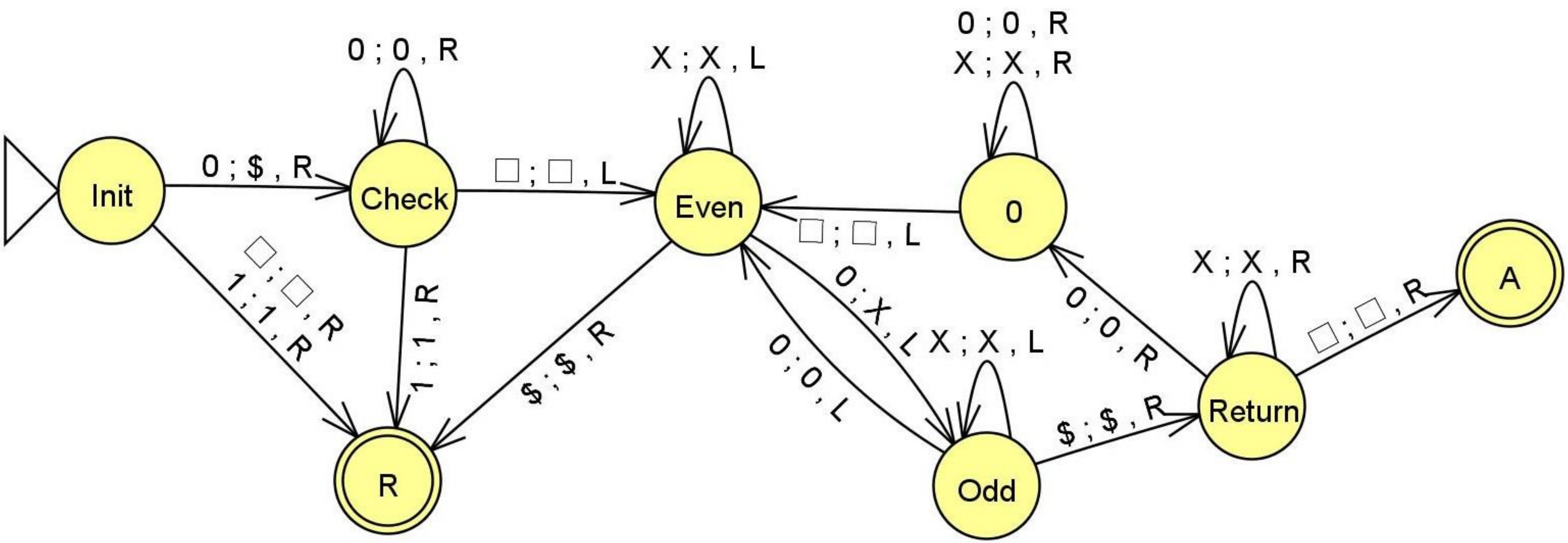
TM Example



- TM M_2 to recognize a language consisting of all strings of 0s whose length is a power of 2

$$A = \{0^{2^n} \mid n \geq 0\}$$

Sample run of M_2



Sample run on input 0000

- *Init*0000 \$*Check*000 \$0*Check*00 \$00*Check*0
- \$000*Check* \$00*Even*0 \$0*Odd*0X \$*Even*00X
- *Odd*\$X0X \$*Return*X0X \$X*Return*0X \$X0*0*X
- \$X0X*0* \$X0*Even*X \$X*Even*0X\$ *Odd*XXX
- *Odd*\$XXX \$*Return*XXX \$X*Return*XX \$XX*Return*X
- \$XXX*Return* \$XXX□*A*

Another example

- A TM to decide the language $C = \{a^i b^j c^k | i \times j = k \text{ and } i,j,k \geq 1\}$.
- M_3 = "On input string w:
  1. Scan the input from left to right to be sure that it is a member of a*b*c* and reject if it isn't.
  2. Return the head to the left-hand end of the tape.
  3. Cross off an 'a' and scan to the right until a 'b' occurs.
  4. Shuttle between the 'b's and 'c's, crossing off each until all 'b's are gone.
  5. Restore the crossed off b's and go to 3 if there is another 'a' to cross off.
  6. If all 'a's are crossed off, check whether all 'c's are also crossed off.
     – If yes, accept; otherwise reject.

Curtin University of Technology

TURING MACHINE VARIANTS

Robustness

Multi-Tape

Non-deterministic

Variants of TM

- Alternative definitions with
  - multiple tapes
  - non-determinism
- All have the same power as the original
  - they recognize the same class of languages
- Robustness :
  - variants have the same power as the original TM
  - FA are somewhat robust, PDAs less so
  - TMs have astonishing degree of robustness

Curtin University of Technology

Robustness

- To show that a variant has the same power as the original, simulate one by the other
- Two machines are equivalent if they recognize the same language
- Example:
  - A TM with transition function that allows the head to stay put instead of moving left or right

Example

- A TM with transition function that allows the head to stay put instead of moving left or right
  - $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L,R,S\}$
  - Each stay put transition can be replaced by two transitions, one moves right and second back to the left
  - So the variant has the same power as the original

Multi-tape Turing machine

- Like an ordinary TM but with several tapes
  - Each tape has its own head for read/write
- Initially input is on one tape, with others blank
- Transition functions changed to allow simultaneous read, write and moving of heads on all tapes
  - $\delta : Q\times \Gamma k \rightarrow Q \times \Gamma_k \times \{L,R\}_k$, where k is the number of tapes

Curtin University of Technology

Multi-tape Turing machine

- If the machine is in state q_i and heads 1 through k are reading symbols a_1 through a_k, the machine goes to state q_j , writes symbols b_1 through b_k, and moves each head to the left or right as specified:
  - $\delta(q_i, a_1, \ldots, a_k) = (q_j, b_1, \ldots, b_k, L, R, \ldots, L)$

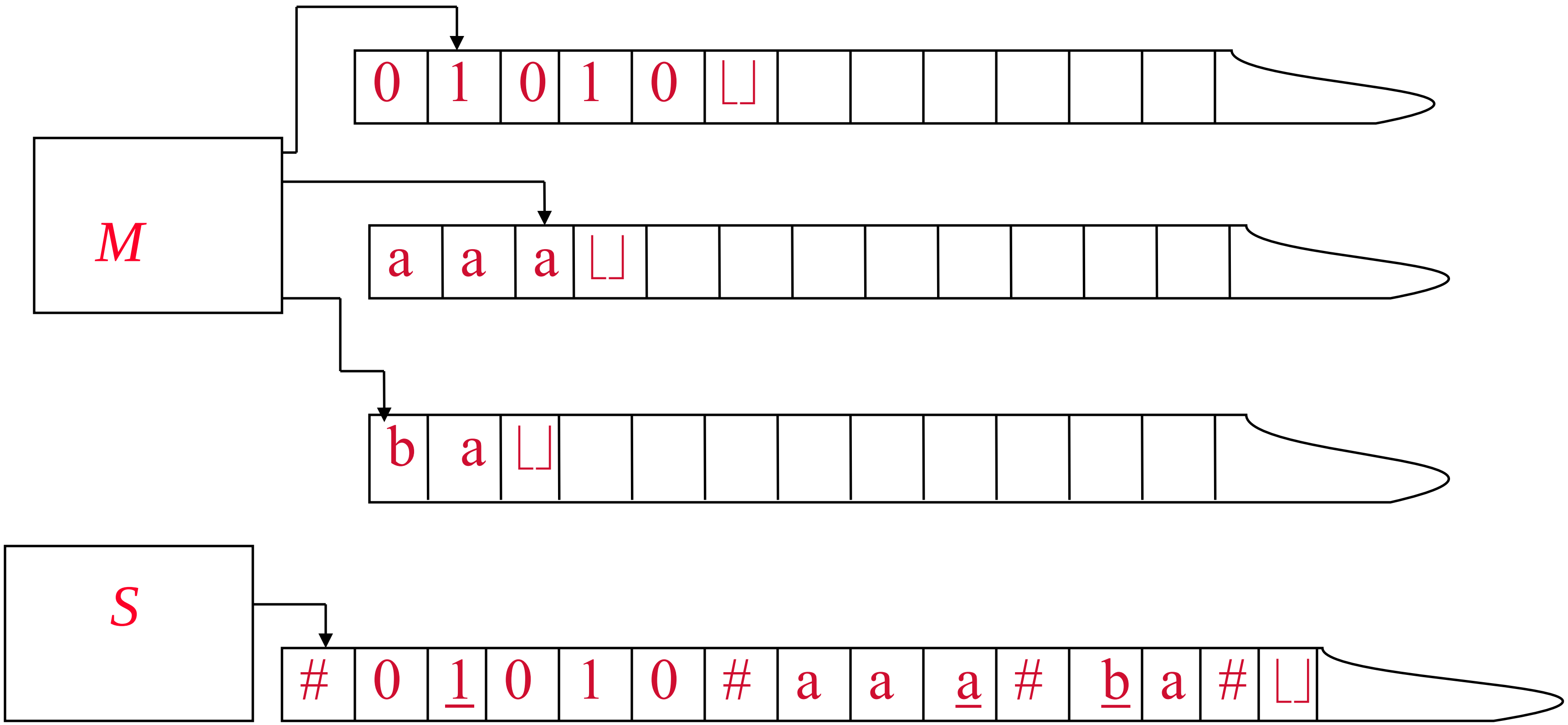
Curtin University of Technology

Multi-tape Turing machine

- **Theorem**: Every multi-tape TM has an equivalent single tape TM
- **Proof idea**: Simulate a multi-tape TM *M* on a single tape TM *S*.
  - The contents of all the tapes of *M* are put on the tape of *S*. A new symbol '#' is used to delimit the contents of different tapes.
  - The head position of each of the tapes of *M* is indicated on the corresponding symbol on the tape of *S* by an underline.
  - If a symbol is written to the end of a tape of *M*, a corresponding new cell is made available on the tape of *S* by shifting the strings to the right of it by one position.

Curtin University of Technology

Multi-tape TM



Non-deterministic Turing Machines

- At any point in a computation, the machine may proceed along several possibilities.
- The transition function has the form:
  - $\delta : Q \times \Gamma \rightarrow \boldsymbol{P}(Q \times \Gamma \times \{L,R\})$.
- The computation is a tree whose branches correspond to different possibilities.
  - If some branch of the computation leads to an accept state, the machine accepts its input.

Curtin University of Technology

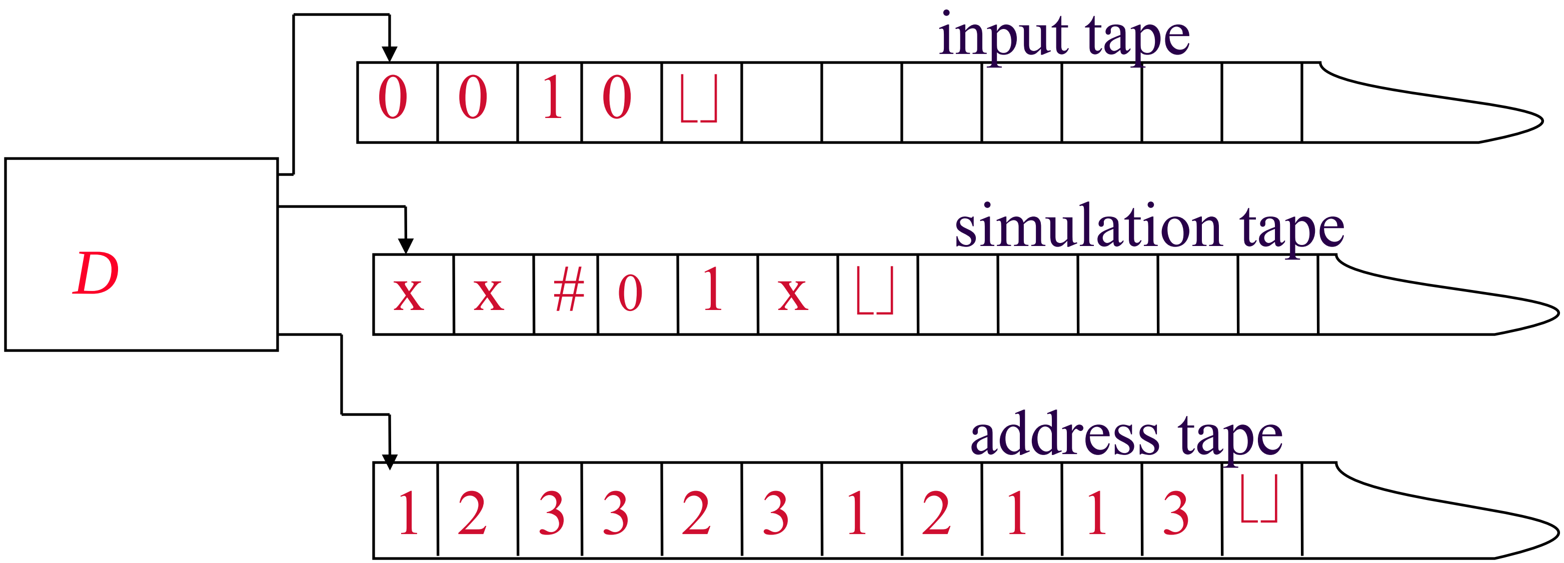
Non-deterministic Turing Machines

- Theorem: Every NTM has an equivalent DTM
- Proof idea: Simulate the NTM *N* on a 3-tape DTM *D*
  - *D* tries all possible branches of *N*.
  - If *D* finds the accept state on one of the branches, it accepts.
  - Otherwise *D*'s simulation will not terminate.

Simulation of NTM

- *N*'s computation on an input *w* is viewed as a tree.
  - Each branch of the tree represents one branch of non-determinism.
  - Each node of the tree is a configuration of *N*.
  - Root of the tree is the start configuration.
  - Breadth first search of the tree is used (rather than DFS).
  - All branches are explored to the same depth before exploring any branch to the next depth.
  - *D* is guaranteed to visit every node until it gets to an accepting configuration.
  - If DFS was followed, *D* could go deeper and deeper along an unproductive branch.

DTM Simulating NTM



- Input tape: Always contains input string, never altered.
- Simulation tape: Maintains a copy of *N*'s tape on some branch of non-determinism.
- Address tape: Keeps track of *D*'s location in *N*'s non-deterministic computation tree.

CHURCH-TURING THESIS

Equivalence of Models

Church-Turing Thesis

Equivalence of Models

- Variants of TM model have equivalent power.
- Other models of general purpose computation
  - Similar to TMs in unrestricted access to unlimited memory.
  - Shown to have power equivalent to that of TMs
  - When only a finite amount of work is allowed in a single step.

Curtin University of Technology

Implication of Equivalence

- Many different computational models, but the class of algorithms they describe is unique.
- Individual models may have some arbitrariness in their definition, but the class algorithms they describe is common with others.
- Programming language analogy:
  - Can some algorithm be programmed in C but not in Java?

Curtin University of Technology

Church-Turing Thesis

- ***Intuitive notion of algorithms equals Turing machine algorithms***
- Church used the notation of λ–calculus to define algorithms
- Turing did it with his “machines”
- The two were shown to be equivalent
- The connection between the informal notion of algorithms and precise definition is called the Church-Turing thesis

ALGORITHMS

Hilbert's Tenth Problem

Description

Graph Example

Hilbert and Mathematics

- Hilbert's Second Problem
  - Axioms of arithmetic are consistent
    - Later added: Independence & Complete
  - Optimist
    - Given a problem believed it could be solved
  - Existence Theorems
    - His first publications concerned that something existed
    - Later gave a constructive proof (Algorithm)
    - Some Controversy over Existence vs Construction

Curtin University of Technology

Hilbert’s Tenth Problem

- To devise an algorithm that tests whether a polynomial has an integral root
  - assumption that there is such an algorithm and it is a matter of finding it
  - what if an algorithm does not exist ?
  - need a precise definition of what is an algorithm
  - progress on the tenth problem had to wait for such a definition

Hilbert's tenth problem

- D= {p|p is a polynomial with an integral root}
  - "$x^2 - 2x + 1$" is in D
  - "$x^2 - 2$" is not in D
- Is the set D decidable ?
  - The answer is negative.
  - D is Turing-recognizable

Curtin University of Technology

Hilbert's tenth problem Algorithm

- Consider the one variable problem
  - D_1= {p|p is a polynomial over x with an integral root}
  - M_1= "The input is a polynomial p over the variable x. Evaluate p with x set successively to the values 0, 1, -1, 2, -2, 3, -3, ... If at any point the polynomial evaluates to 0, accept"
- Recognizer if p has integral root and string accepted
- Decider if recognizes in finite number of steps

Curtin University of Technology

Why Hilbert's tenth problem can't be solved

- If p does not have an integral root, M_1 will run forever
  - So M_1 is a recognizer, not a decider
- M_1 can be converted to a decider because
  - we can calculate bounds within which roots of a single variable polynomial must lie
- Impossible to calculate bounds for the roots of multivariable polynomials
  - So M_1 equivalent recognizer but not a decider

Describing algorithms

- Three possible types of descriptions

1. Formal description of the TM
   - states, transition function, etc.
   - Lowest, most detailed level
2. Implementation description
   - next higher level
   - use English prose to describe the way the TM moves its head and stores data on its tape
   - no details of states or transitions
3. High-level description
   - use English prose to describe an algorithm
   - ignores implementation model
   - *e.g* M_1 for D

Curtin University of Technology

High-level Description Example

- Let *A* be a language consisting of all strings representing undirected graphs.
- A graph is connected if every node can be reached from every other node along the edges.
- $A = \{<G> \mid G$ is a connected undirected graph$\}$
- Describe a TM to decide *A*.

TM Description Example

- M = "On input <G>, 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."

Curtin University of Technology

Input Format for Describing TMs

- The input to a TM is always a string.
  - If an object other than a string is to be provided as input, that object must be first represented as a string.
  - Strings can represent polynomials, graphs, grammars, automata, *etc*.
  - If the input description is simply *w*, it is taken as a string.
  - If it is of the form, <*A*>, the TM first implicitly tests whether the input properly encodes the object and rejects if it doesn't.

Summary

- Turing machines
  - Formal definition
  - Examples
- Variants of TM
  - Multi-tape
  - NTM
  - Equivalence of models
- Church-Turing Thesis