Module 04
Models of Computation
Turing Machines

CS 332 Organization of Programming Languages
Embry-Riddle Aeronautical University
Daytona Beach, FL

M04 Outcomes

At the end of this module you should be able to ...

1. State the definition of the Turing Machine (TM) and its component parts.
2. Given a TM and some input, perform the TM operations to obtain a result.
3. Given a problem, create a notional TM that would solve the problem.
4. State the definition of an decidable problem.
5. State the definition of a computable problem.
6. State the definition of an intractable problem.
7. Describe how the TM, in conjunction with the Halting Problem, define computability.

Turing Machine (TM): Introduction (1/2)

- TM's have theoretical value.
- We're still asking "What can a computer compute?"
- The TM model of computation finally answers the question in a general way.
- TMs are a stronger model than the PDA – TMs have random access memory.
- The ability of the TM to perform calculations allows exploration of what can be computed.
- If a TM can (or can't) compute something, a real world computer also can (or can't).

Turing Machine (TM): Introduction (2/2)

- TMs have no direct practical value
- No real world TMs exist, except those created for fun
- TMs define computable problems – this result has practical value
- TMs can act as language recognizers
- TMs can perform computations (this is far more important)
- TMs can simulate programmable computers – TMs are programmable

The Chomsky Hierarchy (Review)

- For now, define the Chomsky Hierarchy in terms of grammars
- Will describe restrictions on the Left Hand Side (LHS) and Right Hand Side (RHS)
- “Unrestricted” is sometimes called “recursively enumerable” but not covering that in this course – it would take several weeks.

Type	Name	Characteristics of Grammar
Type 3	Regular	LHS must contain exactly one non-terminal. Number of terminals in RHS cannot decrease. Strings derived from right to left, or left to right.
Type 2	Context Free	LHS must contain exactly one non-terminal. Number of terminals in RHS cannot decrease.
Type 1	Context Sensitive	LHS may have terminals and non-terminals. Number of terminals in RHS cannot decrease.
Type 0	Unrestricted	No restrictions

The Chomsky Hierarchy (Review)

- Previously defined the Chomsky Hierarchy in terms of grammars
- Can also define in terms of model of computation
- Will discuss why these models and language match as we go through

Type	Name	Equivalent Model of Computation
Type 3	Regular	Finite State Machine, FSM, also known as Discrete Finite Automata, DFA
Type 2	Context Free	Stack Machine (Push Down Automata, PDA)
Type 1	Context Sensitive	Turing Machine, TM, with finite tape
Type 0	Unrestricted	Turing Machine, TM, with infinite tape

TM: Informal Definition

- The TM machine is effectively a ...
  - finite state machine, with
  - a potentially infinite tape containing cells to contain data
  - a read/write head to read and write to the tape
  - The ability to traverse the tape in either direction, one cell at a time
- The TM has a set of finite states and a start state
  - Additional special final states: halt, yes, and no.
- The TM has a tape alphabet instead of an input alphabet
  - Additional symbol, Δ, indicates the leftmost portion of the tape to be used
  - Additional symbol, ␣, indicates the rightmost portion of the tape to be used
  - Δ may not be moved, ␣ may be overwritten/moved
- TM transitions include ←, →, and – indicating tape head movement

TM: Formal Definition

- Formally, a TM $M = \{Q \cup \{halt, yes, no\}, \Sigma \cup \{\Delta, \sqcup\}, q_o, \delta\}$, where,
  - Q = a finite set of states plus three special states: halt, yes, and no
    - Cornell notes uses K
  - Σ = a finite tape alphabet plus two special symbols:
    - Δ marks the leftmost portion of the tape – it cannot be moved or erased
    - ⊔ marks the rightmost portion of the tape – it can be moved and erased
  - q_o = a single *start state*, where processing begins ($q_o \in Q$)
    - Cornell notes uses s
  - δ = a transition function which includes three actions:
    - ← move one step to the left on the tape
    - → move one step to the right on the tape
    - -- Do not move

TM Example #1 (from Cornell Notes)

Example 1. As our first example, let's construct a Turing machine that takes a binary string and appends 0 to the left side of the string. The machine has four states: s, r_0, r_1, ℓ. State s is the starting state, in state r_0 and r_1 it is moving right and preparing to write a 0 or 1, respectively, and in state ℓ it is moving left.

The state s will be used only for getting started: thus, we only need to define how the Turing machine behaves when reading $\triangleright$ in state s. The states r_0 and r_1 will be used, respectively, for writing 0 and writing 1 while remembering the overwritten symbol and moving to the right. Finally, state ℓ is used for returning to the left side of the tape without changing its contents. This plain-English description of the Turing machine implies the following transition function. For brevity, we have omitted from the table the lines corresponding to pairs (q, σ) such that the Turing machine can't possibly be reading σ when it is in state q.

		$\delta(q,\sigma)$		
q	σ	state	symbol	direction
s	$\triangleright$	r_0	$\triangleright$	$\rightarrow$
r_0	0	r_0	0	$\rightarrow$
r_0	1	r_1	0	$\rightarrow$
r_0	$\sqcup$	ℓ	0	$\leftarrow$
r_1	0	r_0	1	$\rightarrow$
r_1	1	r_1	1	$\rightarrow$
r_1	$\sqcup$	ℓ	1	$\leftarrow$
ℓ	0	ℓ	0	$\leftarrow$
ℓ	1	ℓ	1	$\leftarrow$
ℓ	$\triangleright$	halt	$\triangleright$	–

TM Example #2 (from Cornell Notes)

Example 2. Using similar ideas, we can design a Turing machine that takes a binary integer n (with the digits written in order, from the most significant digits on the left to the least significant digits on the right) and outputs the binary representation of its successor, i.e. the number $n+1$.

It is easy to see that the following rule takes the binary representation of n and outputs the binary representation of $n+1$. Find the rightmost occurrence of the digit 0 in the binary representation of n, change this digit to 1, and change every digit to the right of it from 1 to 0. The only exception is if the binary representation of n does not contain the digit 0; in that case, one should change every digit from 1 to 0 and prepend the digit 1.

		$\delta(q,\sigma)$		
q	σ	state	symbol	direction
s	$\triangleright$	r	$\triangleright$	$\rightarrow$
r	0	r	0	$\rightarrow$
r	1	r	1	$\rightarrow$
r	$\sqcup$	ℓ	$\sqcup$	$\leftarrow$
ℓ	0	t	1	$\leftarrow$
ℓ	1	ℓ	0	$\leftarrow$
ℓ	$\triangleright$	prepend::s	$\triangleright$	–
t	0	t	0	$\leftarrow$
t	1	t	1	$\leftarrow$
t	$\triangleright$	halt	$\triangleright$	–
prepend::s	$\triangleright$	prepend::s	$\triangleright$	$\rightarrow$
prepend::s	0	prepend::r	1	$\rightarrow$
prepend::r	0	prepend::r	0	$\rightarrow$
prepend::r	$\sqcup$	prepend::ℓ	0	$\leftarrow$
prepend::ℓ	0	prepend::ℓ	0	$\leftarrow$
prepend::ℓ	1	prepend::ℓ	1	$\leftarrow$
prepend::ℓ	$\triangleright$	halt	$\triangleright$	–

TM Example #3

- Let L be the set of strings that encode a correct addition equation.
- Let TM M be the machine that recognizes L.
- Let $\Sigma = \{0, 1, \Delta, \sqcup\}$
- Sample strings in L:
  - Δ 000100100000 $\sqcup$ (3 + 2 = 5)
  - Δ 00011000 $\sqcup$ (3 + 0 = 3)
  - Δ 0000100010000000 $\sqcup$ (4 + 3 = 7)
- Operation:
  - Start at LHS and start state, traverse right until the first 1 is encountered
  - Remove the 1 and return left to Δ
  - Begin removing 0's from each side of the remaining 1
  - If final string is $\Delta 1 \sqcup$ then "yes", otherwise "no"

Note: This is the level of detail needed for the HW problems.

TM Example #4

- TM M can calculate addition, not just verify it. (This is new!)
- Instead of M recognizing strings in L, let M solve an addition problem
- Represent numbers by the number of 0's present: Δ0000␣ = 4
- Represent an addition problem by two numbers separated by a 1
  - Δ010000␣ = 1 +4
  - Δ0001␣ = 3 + 0
  - Δ00100␣ = 2 + 2
- M's operation: simply remove the 1, copy all symbols on the right one space to the left, and halt.
  - Δ00000␣ = 5
  - Δ000␣ = 3
  - Δ0000␣ = 4

Proof Sketch: TMs Cannot Compute All Problems

- Contradiction proof: Assume all problems are computable by a TM.
- Let S = {M0, M1, M2, M3, ….} be the set of all TM's.
- Each TM solves a specific problem.
- Since all problems are computable, there are no TMs not in S.
- Create a TM, M', not in S: diagonalization proof:
  - Each TM in S is represented by a string.
  - Let M' differ from M1 in the first symbol
  - Let M' differ from M2 in the second symbol
  - Let M' differ from M3 in the third symbol (and so on)
  - M' differs from every M in S
  - M' is not in S
  - Contradiction
- Not all problems are computable by a TM. (TM = computer)

Termination

- Not all computations are guaranteed to terminate.
- Robot Problem 1: consider an infinitely long wall with a door in it. Tell the robot to find the door. This is guaranteed to terminate.
- Robot Problem 2: Consider an infinitely long wall that may have a door in it. Tell the robot to determine if the wall has a door. This will not terminate if there is no door in the wall.
- The MU Puzzle – If MU cannot be derived from MI, at what point do you terminate?
  - Unrestricted grammars can reduce strings – can't brute force a result by creating all possible strings of length n.

The Halting Problem

- TMs take in strings and perform operations based on the strings.
- A TM is represented by a string (a sequence of symbols).
- A string representing a TM can serve as input to another TM.
- Universal Turing Machine (UTM) –
  - Takes in a string representing some TM, M
  - Takes in a string as input to a problem, p
  - Simulates the first TM acting on the problem
- Halting Problem: Is M guaranteed to terminate (halt) on problem p?
- This problem is asked of the UTM.
- This problem is a decision problem – decide yes or no.
- This problem is provably undecidable (diagonalization proof).

Decidability and Turing Computability

- Decidable problems require two things:
  - The ability to write an algorithm for the problem
  - Guarantee of termination with a clear, correct result.

- Have proven that not all problems are decidable/computable.
  - Specific non-computable problems not identified
  - Characteristics identified: algorithm must be written, termination guaranteed
  - Halting problem says that termination cannot be determined by a computer

- Bottom line: Software that determines the intent/actions of other software is impossible to create – security concerns.

Intractability

- Some problems are computable but not tractable.
- Algorithms can be written (TM's made)
- Termination is guaranteed.
- It takes too long to compute – $O(n^k)$, $k > 1$, or $O(n!)$
  - This is not about getting a faster computer
  - "too long" means centuries
- Classic intractable problems:
  - Traveling Salesman – most efficient way to traverse a graph
  - Bin Packing – most efficient way to pack a bin (or knapsack)
  - Graph Coloring – Fewest resources needed to avoid conflicting requirements