Machines with Memory

Chapter 3 (Part B)

Turing Machines

- Introduced by Alan Turing in 1936 in his famous paper “On Computable Numbers with an Application to the Entscheidungsproblem”
- For its simplicity, no other computational model has been invented/devised that surpasses the computational capability of the Turing Machine
- Used as the canonical model of computation by most theoreticians in studying computability problems
- Used to categorize languages and problems based on the time and space requirements of TMs needed to recognize or solve them
Turing Machines (cont.)

DEFINITION 3.7.1 A standard Turing machine (TM) is a six-tuple \( M = (\Gamma, \beta, Q, \delta, s, h) \), where \( \Gamma \) is the tape alphabet not containing the blank symbol \( \beta \), \( Q \) is the set of states, \( \delta : Q \times (\Gamma \cup \{\beta\}) \rightarrow (Q \cup \{L\} \times (\Gamma \cup \{\beta\}) \times \{L, N, R\} \) is the next-state function, \( s \) is the initial state, and \( h \notin Q \) is the accepting halt state. A TM cannot exit from \( h \). If \( M \) is in state \( q \) with letter \( \alpha \) under the tape head and \( \delta(q, \alpha) = (q', \beta, A) \), its control unit enters state \( q' \) and writes \( \beta \) in the cell under the head, and moves the head left (if possible), right, or not at all if \( A \) is \( L \), \( R \), or \( N \), respectively.

![Turing Machine Diagram]

Turing Machines (cont.)

- There are many other “flavors” of Turing Machines (TMs)
  - Multi-tape TMs, multi-head TMs
  - Deterministic and non-deterministic TMs
  - Infinitely long tape in both directions, in only one direction (i.e., leftmost tape cell exists)

- Can be shown that most are equivalent in capability
- Can show that TMs can simulate RAMs and vice-versa
- Universality results
  - TM can simulate any RAM computation
  - RAM programs can simulate TMs
  - RAMs are also universal (in the TM sense)
**Turing Machines (cont.)**

- Important class of languages that TMs recognize: \( P \) polynomial-time languages

**DEFINITION 3.7.2.** A language \( L \subseteq \Gamma^* \) is in \( P \) if there is a Turing machine \( M \) with tape alphabet \( \Gamma \) and a polynomial \( p(n) \) such that, for every \( w \in \Gamma^* \), \( M \) halts in \( p(|w|) \) steps and accepts \( w \) if it is in \( L \) and rejects it otherwise.

- Equally significant class of languages that NTMs recognize: \( NP \) non-deterministic polynomial-time languages

**DEFINITION 3.7.4.** A language \( L \subseteq \Gamma^* \) is in \( NP \) if there is a nondeterministic Turing machine \( M \) and a polynomial \( p(n) \) such that \( M \) accepts \( L \) and for each \( w \in L \), there is a choice input \( c \) such that \( M \) on input \( w \) with this choice input halts in \( p(|w|) \) steps.

**Turing Machines (cont.)**

- Languages in \( NP \) can be verified in a polynomial number of steps: a candidate string can be validated as member of the language in polynomial time

- Some problems that are in \( NP \):
  - *Traveling Salesperson Problem (TSP)*
  - *Graph 3-Colorability (3COL)*
  - *Multiprocessor Scheduling (MSKED)*
  - *Bin Packing (BNPAK)*
  - *Crossword Puzzle Construction (CRSCON)*

- It is not known whether \( P \) and \( NP \) include exactly the same set of problems, or one (\( NP \)) is larger than the other (\( P \))
Universality of the Turing Machine

- Show existence of a *universal* Turing machine in two senses:
  - TM exists that can simulate any RAM computation
  - TM that simulates RAM can simulate any other TM

**Theorem 3.8.1** Let \( S = r \cdot b \) and \( m \geq b \). Then for every \( m \)-word, \( b \)-bit Turing machine \( M_{\text{TM}} \) (with storage capacity \( S \)) there is an \( O(m) \)-word, \( b \)-bit RAM that simulates a \( T \) computation of \( M_{\text{TM}} \) in time \( O(T) \) and storage \( O(S) \). Similarly, for every \( m \)-word, \( b \)-bit RAM \( M_{\text{RAM}} \) there is an \( O((m/b) \log m) \)-word, \( O(b) \)-bit Turing machine that simulates a \( T \)-time, \( S \)-storage computation of \( M_{\text{RAM}} \) in time \( O(ST \log^2 S) \) and storage \( O(S \log S) \).

- Proof sketch
  - Describe RAM that simulates a TM
  - Describe TM that simulates a RAM

Universality of the Turing Machine (cont.)

- RAM that simulates a TM:
  - TM’s Control Unit is a FSM \( \Rightarrow \) there is a RAM program that can simulate this (cf. Theorem 3.4.1) in linear time

![Figure 3-5. Doubling the number of strokes.](from Boolos et al., p. 28)

- Need to show that TM’s Tape Unit can be simulated by a FSM \( \Rightarrow \) there is RAM program that can simulate this (cf. Theorem 3.4.1) in linear time
Universality of the Turing Machine (cont.)

- TM’s Tape Unit assumed to be a $b$-bit, $m$-word device

- Tape Unit FSM simulation encodes the configuration of the Tape Unit with its states:

  \[ q_{i,j} \quad i: \text{ith Tape Unit cell} \]
  \[ j \in \Gamma \cup \{\beta\} \]

- FSM simulation will thus have no more than $m \times (|\Gamma|+1)$ states = $O(m)$ states

Universality of the Turing Machine (cont.)

- FSM simulation takes “input” from the TM’s Control Unit in the form of an element of $\Gamma \cup \{\beta\}$ or $L, R, \text{ or } N$

- This “input” determines which edge transition will be taken in the transition diagram that defines the FSM “logic”
Universality of the Turing Machine (cont.)

- If “input” is $\sigma \in \Gamma \cup \{\beta\}$, then we take the edge

\[
q_{i,j} \xrightarrow{\sigma} q_{i',j'}
\]

where $i' = i$ and $j' = \sigma$

- If “input” is $L$, then we take the edge

\[
q_{i,j} \xrightarrow{L} q_{i',j'}
\]

where $i' = i-1$ and $j' = \text{current content of the } i-1\text{st cell}$

Universality of the Turing Machine (cont.)

- If “input” is $L$, then we take the edge

\[
q_{i,j} \xrightarrow{L} q_{i',j'}
\]

where $i' = i-1$ and $j' = \text{current content of the } i-1\text{st cell}$

- If “input” is $N$, then we take the edge

\[
q_{i,j} \xrightarrow{N}
\]

- This FSM can be simulated by a RAM program in linear time (cf. Theorem 3.4.1)
Universality of the Turing Machine (cont.)

- TM that simulates a RAM:
  - RAM’s memory unit can be simulated by the TM’s Tape Unit

<table>
<thead>
<tr>
<th>Leftmost tape cell</th>
<th>n</th>
<th>w_n</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

*Figure 3.24* Organization of a tape unit to simulate a RAM. Each RAM memory word w_j is accompanied by its address j in binary.

- Leftmost tape cell holds the address of the RAM CPU program counter
- TM simulates the RAM by emulating the fetch-and-execute cycle

Universality of the Turing Machine (cont.)

- Fetch step: move to cell whose address is stored in leftmost tape cell

- Execute step: w_n is “decoded” by the TM Control Unit and the corresponding action undertaken; can be any of five types of RAM instruction
Universality of the Turing Machine (cont.)

- Sample instruction: multiply two numbers
- Though meant for unary-represented numbers, shows it can be done
- Number of steps is a function of the operands and not any other parameter

Turing Machine Circuit Simulation

- The following result follows from the fact that TMs can be realized by composing FSMs:

**Theorem 3.9.1** Any computation performed by a one-tape Turing machine $M$, deterministic or nondeterministic, on an input string $w$ in $T$ steps using $m$ b-bit memory cells can be simulated by a circuit $C_{M,T}$ over the standard complete basis $Ω$ of size and depth $O(ST)$ and $O(T \log S)$, respectively, where $S = m b$ is the storage capacity in bits of $M$’s tape. For the deterministic TM the inputs to this circuit consist of the values of $w$. For the nondeterministic TM the inputs consist of $w$ and the Boolean choice input variables whose values are not set in advance.

- This allows us to extend to Turing Machine computable functions the result on size and depth of equivalent circuits that compute their solutions.
Computational Inequalities for Turing Machines

**THEOREM 3.9.2** The function $f$ computed by an $m$-word, $b$-bit one-tape Turing machine in $T$ steps can also be computed by a circuit whose size and depth satisfy the following bounds over any complete basis $\Omega$, where $S = \min$ is the storage capacity used by this machine:

$$C_\Omega(f) = O(ST)$$
$$D_\Omega(f) = O(T \log S)$$

- Since TMs can only scan $O(T)$ cells in $T$ steps, the result can be expressed as follows:

**COROLLARY 3.9.1** Let the function $f$ be computed by an $m$-word, $b$-bit one-tape Turing machine in $T$ steps, $b$ fixed. Then, over any complete basis $\Omega$ the following inequality must hold:

$$C_\Omega(f) = O(T^2)$$

---

Definition of $\mathbf{P}$-Complete and $\mathbf{NP}$-Complete Languages

**DEFINITION 3.9.1** A language $L \subseteq \Sigma^*$ is **$\mathbf{P}$-complete** if it is in $\mathbf{P}$ and if for every language $L_0 \subseteq \Sigma^*$ in $\mathbf{P}$, there is a log-space deterministic program that translates each $w \in \Sigma^*$ into a string $w' \in \Sigma^*$ such that $w \in L_0$ if and only if $w' \in L$.

- $w$ is a problem instance of $L_0$ (solvable in log-space by a DTM) which is “transformed” by the converter program in log-space to an instance $w'$ of $L$ which is then solvable in log-space by a DTM $\Rightarrow w$ is solvable in log-space using $L$ as a “subprogram”
Definition of P-Complete and NP-Complete Languages (cont.)

**Definition 3.9.2** A language $L \subseteq \Sigma^*$ is NP-complete if it is in NP and if for every language $L_0 \subseteq \Sigma^*$ in NP, there is a polynomial-time deterministic program that translates each $w \in \Sigma^*$ into a string $w' \in \Sigma^*$ such that $w \in L_0$ if and only if $w' \in L$.

- $w$ is a problem instance of $L_0$ (solvable in polynomial-time by a NDTM) which is “transformed” by the converter program in polynomial-time to an instance $w'$ of $L$ which is then solvable in polynomial-time by a NDTM $\Rightarrow w$ is solvable in polynomial-time using $L$ as a “subprogram.”

Definition of P-Complete and NP-Complete Languages (cont.)

- Key to fundamental open problem in theoretical computer science is this theorem

**Theorem 3.9.4** If an NP-complete language is in P, then $P = NP$.

By 1994, well over 10,000 basic NP-complete decision problems were known.
Reductions to P-Complete Languages

- Our first P-complete language:
  - **CIRCUIT VALUE**
    - **Instance:** A circuit description with fixed values for its input variables and designated output gate.
    - **Answer:** “Yes” if the output of the circuit has value 1.

  ![Circuit Diagram]

  **THEOREM 3.9.5** *The language CIRCUIT VALUE is P-complete.*

  - Proof sketch:
    - “P-hardness” cf. Theorem 3.9.1
    - “P-ness” ⇔ circuits and straight-line programs are equivalent

Reductions to P-Complete Languages (cont.)

- Typical proof technique:

  ![Reduction Diagram]

  - $L$ is a language in P;
  - $L_0$ is a P-complete language;
  - $L_1$ is a candidate P-complete language
Reductions to NP-Complete Languages

- Our first NP-complete language:
  - CIRCUIT SAT
    - *Instance*: A circuit description with input variables \(x_1, x_2, \ldots, x_n\) and designated output gate.
    - *Answer*: “Yes” if there is an assignment of values to the variables such that the output of the circuit has value 1.

*THEOREM 3.9.6* The language CIRCUIT SAT is NP-complete.

- Proof sketch:
  - “NP-hardness” *cf.* Theorem 3.9.1
  - “NP-ness” *⇒* use Theorem 3.9.5 to verify that input produces a 1 output

Reductions to NP-Complete Languages (cont.)

- Typical proof technique:

  - \(L\) is a language in NP;
  - \(L_0\) is a NP-complete language;
  - \(L_1\) is a candidate NP-complete language
Reductions to NP-Complete Languages (cont.)

- Our second NP-complete language:
  - SATISFIABILITY (SAT)
    - **Instance**: A set of literals $X = \{x_1, \neg x_1, x_2, \neg x_2, \ldots, x_n, \neg x_n\}$ and a sequence of clauses $C = \{c_1, c_2, \ldots, c_m\}$ where each clause $c_i$ is a subset of $X$.
    - **Answer**: “Yes” if there is a (satisfying) assignment of values to the variables $\{x_1, x_2, \ldots, x_n\}$ over the set $B$ such that each clause has at least one literal whose value is 1.

**THEOREM 3.9.7** SATISFIABILITY is NP-complete.

- Proof sketch:
  - “NP-ness” $\Leftrightarrow$ can verify a “guess” in polynomial-time
  - “NP-hardness” $\Leftrightarrow$ reduce CIRCUIT SAT to SAT

Reductions to NP-Complete Languages (cont.)

- CIRCUIT SAT reduction to SAT key:
  - Any circuit can be described by a straight-line program
  - It suffices to show that straight-line program “statements” can be transformed to equivalent clauses such that the resulting SAT instance produces a “Yes” answer exactly when the straight-line program produces a “Yes” answer for the CIRCUIT SAT instance

<table>
<thead>
<tr>
<th>Step Type</th>
<th>Corresponding Clauses</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i$ READ $x$</td>
<td>$(\overline{g}_i \lor x) \quad (g_i \lor \overline{x})$</td>
</tr>
<tr>
<td>$i$ NOT $j$</td>
<td>$(\overline{g}_i \lor \overline{g}_j) \quad (g_i \lor \overline{g}_j)$</td>
</tr>
<tr>
<td>$i$ OR $j$ $k$</td>
<td>$(g_i \lor \overline{g}_j) \quad (g_i \lor \overline{g}_k) \quad (\overline{g}_i \lor g_j \lor g_k)$</td>
</tr>
<tr>
<td>$i$ AND $j$ $k$</td>
<td>$(\overline{g}_i \lor g_j) \quad (\overline{g}_i \lor g_k) \quad (g_i \lor \overline{g}_j \lor \overline{g}_k)$</td>
</tr>
<tr>
<td>$i$ OUTPUT $j$</td>
<td>$(g_j)$</td>
</tr>
</tbody>
</table>