Introduction: Overview

• Automata theory deals with the theory of computation

• Theory of computation
  – Provides set of abstract structures that can be used for solving certain classes of problems
    * These problems are independent of any platform (software or hardware)
    * Based on mathematical properties of problems and algorithms
  – Defines what is computable
    * I.e., Identifies the limitations of computability

• Theory can be used to answer questions like
  1. Can a given problem be solved computationally?
     – If not, is there a restricted solution that is useful?
  2. Can a solution be implemented in a fixed amount of time?
  3. How efficient is a solution?
  4. Can problems be classified wrt their solutions?

• Theory is the basis on which computer science exists

• Theory has many practical applications beyond the purely theoretical issues

• General topics to be discussed:
  1. Languages
  2. Finite state machines
  3. Regular expressions, grammars, and languages
  4. Pushdown automata
  5. Context free grammars and languages
  6. Turing machines
  7. Context sensitive and unrestricted grammars and languages
  8. Decidability
  9. Complexity and complexity classes
Introduction: Strings - Basic Definitions

- **Alphabet**: Finite set of symbols or characters
  - Denoted $\Sigma$

- **String**: Finite sequence - possibly empty - of symbols from some alphabet $\Sigma$
  - Empty string denoted $\epsilon$
  - Set of all possible strings over alphabet $\Sigma$ denoted $\Sigma^*$
Introduction: Strings - Functions

• The following notations are used below
  1. Symbols/characters represented by $c$
  2. Strings represented as $s, t, w$
  3. Variables represented as $x, y$
  4. Numbers represented as $i$

1. Length
   • Denoted $|s|$
   • Returns number of symbols in $s$

2. Occurrence frequency
   • Denoted $\#_c(s)$
   • Returns number of times $c$ occurs in $s$

3. Concatenation
   • Denoted $s||t$, or simply $st$
   • Returns single string whose prefix is $s$ and whose suffix is $t$
   • $\epsilon$ is identity for concatenation
     \[ \forall x (x\epsilon = \epsilon x = x) \]
   • Concatenation is associative
     \[ \forall s, t, w ((st)w = s(tw)) \]

4. Replication
   • Denoted $w^i$
     (a) $w^0 = \epsilon$
     (b) $w^{i+1} = w^i w$

5. Reversal
   • Denoted $s^R$
   • If $|s| = 0$, $w^R = w = \epsilon$
   • If $|w| \geq 1$, and $\exists a \in \Sigma (\exists u \in \Sigma^* (w = ua))$
     Then $w^R = au^R$
   • Theorem 2.1
     – Statement: Given strings $w$ and $s$, $(ws)^R = s^Rw^R$
     – Proof: By induction. See p9
Introduction: Strings - Relations

1. Substring
   • Substring is a sequence of consecutive symbols of $s$
   • $t$ is a proper substring of $s$ iff $s \neq t$
   • $\epsilon$ is a substring of every string

2. Prefix
   • $s$ is a prefix of $t$ iff $\exists w \in \Sigma^* (t = sw)$
   • $s$ is a proper prefix of $t$ iff $s$ is a prefix of $t$ and $s \neq t$
   • $\epsilon$ is a prefix of every string

3. Suffix
   • $s$ is a suffix of $t$ iff $\exists w \in \Sigma^* (t = ws)$
   • $s$ is a proper suffix of $t$ iff $s$ is a suffix of $t$ and $s \neq t$
   • $\epsilon$ is a suffix of every string
Introduction: Languages - Basic Definitions

- **Language**: Set of strings over a finite alphabet
  - Language $L$ defined wrt alphabet $\Sigma$
    - Denoted $\Sigma_L$
- **Empty language**: Language that contains no strings
  - Denoted $\{\}$ or $\emptyset$
  - $\epsilon \not\equiv \emptyset$
- **Natural languages** will not be dealt with
  - No formal grammars exist due to complexity
- Languages can be represented using several approaches:
  - In the following, $\Sigma = \{a, b\}$
  1. Standard set representation formalism
     (a) Enumeration-based, e.g.,
        \[ L = \{b, baa, bab, bba, bbb, bbbaa, bbaba, bbbba, bbaab, bbabb, bbbab, bbbbb\} \]
     (b) Rule-based (using characteristic function), e.g.
        i. $L = \{w : \exists x \in \{a, b\}^* (w = bx \land |w| \leq 5 \land |w| \text{ is odd})\}$
        ii. Using prefix relation
            $L = \{w \in \{a, b\}^* : \text{Every prefix of } w \text{ starts with } b\} \cup \emptyset$
        iii. Using replication relation
            $L = \{a^n : |w| \leq 5 \land |w| \text{ is odd}\}$
  2. Computationally, using
     (a) Language generator
        - Produces all strings in the language
        - **Lexicographic order**:
          - Denoted $<_L$
          - Shorter strings precede longer ones:
            \[ \forall w (\forall s (|w| < |s|) \rightarrow (w <_L s))) \]
          - Let $D$ be a total order on $\Sigma_L$.
            Then, strings of same length ordered wrt $D$
     (b) Language recognizer
        - Input is strings
        - Output is $T$ or $F$ (accept or reject)
Languages - Cardinality

- **Cardinality** of a language: Number of strings the language contains
- Cardinality of smallest language is 0 ($L = \emptyset$)
- Cardinality of largest language wrt alphabet $\Sigma = \Sigma^*$

- **Theorem 2.2**
  - **Statement:** If $\Sigma \neq \emptyset$, $\Sigma^*$ is countably infinite
  - **Proof:** See p14

- Therefore, cardinality of every language is between 0 and $\aleph_0$

- **Given alphabet $\Sigma$, let $P(\Sigma^*)$ represent the power set of $\Sigma^*$**
  - The number of languages derivable from $P$ is
    * If $\Sigma = \emptyset$, then $\Sigma^* = \{\epsilon\}$, and $P(\Sigma^*) = \{\emptyset, \{\epsilon\}\}$
    * **Theorem 2.3**
      - **Statement:** If $\Sigma \neq \emptyset$, set of languages over $\Sigma$ is uncountably infinite
      - **Proof:** See p15
Introduction: Languages - Functions

1. All standard set functions apply to languages

2. Concatenation of $L_1$ and $L_2$
   - Denoted $L_1L_2$
     \[ L_1L_2 = \{w \in \Sigma^* : \exists s \in L_1(\exists t \in L_2(w = st))\} \]
   - $\{\epsilon\}$ is identity for concatenation:
     \[ L\{\epsilon\} = \{\epsilon\}L = L \]
   - $\emptyset$ is zero for concatenation:
     \[ L\emptyset = \emptyset L = \emptyset \]
   - Associative

3. Kleene star
   - Denoted $L^*$
   - Let $L$ be defined over alphabet $\Sigma$
     \[ L^* = \{\epsilon\} \cup \{w \in \Sigma^* : \exists k \geq 1(\exists w_1, w_2, \ldots, w_k \in L(w = w_1w_2\ldots w_k))\} \]
   - $L^*$ contains an infinite number of strings unless $L = \emptyset$ or $L = \{\epsilon\}$

4. $+$
   - $L^+ = LL^*$
   - $L^+$ is closure of $L$ under concatenation
   - $L^+ = L^* - \epsilon$ iff $\epsilon \notin L$

5. Reverse
   - Denoted $L^R$
     \[ L^R = \{w \in \Sigma^* : w = s^R \text{ for some } s \in L\} \]
   - Theorem 2.4
     - Statement: If $L_1$ and $L_2$ are languages, then $(L_1L_2)^R = L_2^R L_1^R$
     - Proof: See p17

6. Chop
   - $\text{chop}(L) = \{w : \exists x \in L(x = x_1cx_2 \land x_1 \in \Sigma^*_L \land x_2 \in \Sigma^*_L \land c \in \Sigma_L \land |x_1| = |x_2| \land w = x_1x_2)\}$
   - Semantics: $\text{chop}$ removes middle symbol for odd-length strings
7. firstChars

- \( \text{firstChars}(L) = \{ w : \exists y \in L (y = cx \land c \in \Sigma_L \land x \in \Sigma_L^* \land w \in c^*) \} \)
- Semantics: \( \text{firstChars} \) is the union of strings of any length that consist of a single symbol, where that symbol is the first symbol of some string in \( L \)
Introduction: Languages - Semantics

• **Semantics** of a language: Meaning mapped to strings of the language
• Meaning assigned by a *semantic interpretation function*
• Since cardinality of many languages is infinite, want to assign meaning using rules
  – Allows assignment of meaning to strings of arbitrary length
  – Rule-based functions called *compositional* semantic interpretation functions
  – Usually not 1:1
Introduction: Languages - Recognition

- Theory of computation based on strings and languages
- Unifying framework is the language recognition problem (LRP):
  - Given language $L$ and string $w$, is $w \in L$?
  - This is a decision problem (DP)
- Dealing with LRPs is not restrictive:
  - Many other types of problems can be recast as LRPs
- Types of problems fall into 2 categories:
  1. Those that are already DPs
     (a) Need to encode inputs as strings
        - Given some object $x$, denote string encoding of $x$ as $<x>$
        - Given objects $x$ and $y$, denote string encoding of $x$ and $y$ as $<x,y>$ (a single string)
     (b) Define a language that contains exactly those strings for which the answer is true
  2. Those that are not DPs
     (a) Reformulate as a DP
        - Encode input and corresponding output as a single string
        - Language accepts those strings for which the output component is the result when the program processes the input components
     (b) Encode as for DPs
        - The reformulated problem is equivalent to the original (and vice-versa)
          * The original problem is said to be reduced to the reformulated problem
          * If have solution for one, have solution for the other
- Many problems can be encoded in more than one way
  - Choice of encoding may affect
    1. Difficulty of problem
    2. Efficiency of solution
Introduction: LRP Examples (Already DPs)

(See Rich pp 22 - 25)

1. Halting problem
   • Given: Program $p$ in a standard programming language
   • Problem: Does $p$ halt on all inputs?
   • Language: $HP_{all} = \{ p : p \text{ halts on all inputs} \}$

2. Verifying addition
   • Given: $int_1$, $int_2$, $int_3$
   • Problem: Does $int_1 + int_2 = int_3$?
   • Language: $IntegerSum = \{ w \text{ of form } < int_1 > + < int_2 > = < int_3 > :$
                 each substring $int_1, int_2, int_3 \in \{0, ..., 9\}^*$ and $int_1 + int_2 = int_3 \}$
(See Rich pp 26 - 27)

1. Casting addition as DP
   - Given: 2 non-negative integers \( int_1 \) and \( int_2 \)
   - Problem: Compute their sum
   - Transformation: Reformulate as problem of deciding whether \( int_1 + int_2 = \) some third integer \( int_3 \)
   - Encoding: As \textit{IntegerSum}
   - Language: See \textit{IntegerSum}
• **Decision procedure**: Algorithm to solve a decision problem
  1. Returns Boolean value appropriate to input
  2. Halts on all inputs

• Important questions for which decision procedures constructed:
  1. Is a given string in a given language?
  2. Does a machine $M$ accept any strings?
  3. Given machines $M_1$ and $M_2$, are the languages they accept the same?
  4. Is a given machine minimal?

• For a given decision problem, may want to know:
  1. Does a decision procedure exist for the problem?
  2. If a decision procedure exists, what is its efficiency?

• For a given decision problem, may want to:
  1. Construct a decision procedure if one exists
Introduction: Languages - Hierarchy

- Languages can be placed in a hierarchy
  - Based on language properties
  - Properties relate to the types of programs that can be used as language recognition devices
  - Hierarchy:

- Most expressive/powerful model is outermost

- Language classes:
  1. Regular
     - Accepted by a *Finite Automaton* (FA)
     (also called a *Finite State Machine* (FSM))
  2. Context free
     - Accepted by a *Pushdown Automaton* (PDA)
  3. Decidable
     - Accepted by a *Turing Machine* (TM)
  4. Semidecidable
     - Accepted by a *Turing Machine* (TM)
Introduction: Languages - Hierarchy (2)

- Each level of hierarchy includes those languages lower in the hierarchy
- The higher the level, the more expressive/powerful the language
- Greater expressiveness comes at a cost:
  1. Efficiency
     - FAs: Linear wrt $|s|$
     - PDAs: Cubic wrt $|s|$
     - TMs: Exponential wrt $|s|$
  2. Decidability
     - Can we determine definitively whether a sentence is a member of a language or not?
  3. Clarity
     - Analysis tools exist only for regular and context free languages
Introduction: Languages - Tractability

• **Tractability**: Whether a problem can be solved in a reasonable amount of time

• Decidable languages can be classified wrt efficiency (tractability):

  1. Class P: Languages that can be decided in time that grows proportionally as some polynomial function of the input length
  2. Class NP: Languages that can be decided *non-deterministically* in time that grows proportionally as some polynomial function of the input length
  3. Class PSPACE: Languages that can be decided by a machine whose space requirement grows proportionally as some polynomial function of the input length

• $P \subseteq NP \subseteq PSPACE$