Why Study Automata Theory?

- **Introduction to Finite Automata**
  - In the late 1950’s, the linguist Noam Chomsky begun the study of formal “grammars.”
  - In 1969, Stephen Cook defined “intractable” or “NP-hard” problems — problems that can in principle be solved, but in practice take so much time that computers are useless for all but very small instances of the problem.

- **Automata Theory** – study of abstract computing devices, or “machines.”
  - In the 1930’s, Alan Turing studied and described precisely the boundary between what a(n) (abstract) computing machine could do and what it could not do.

**Structural Representations**

- **Grammars**
  - Useful models when designing software that processes data with a recursive structure.
  - Example: parser for a compiler.

- **Regular Expressions**
  - Denote the structure of data; in particular, text strings.
  - Example: Unix-style regular expressions

' ([A-Z][a-z]* [ ])* [A-Z][A-Z]'
### Automata and Complexity

- **Decidability (see Chapter 9)**
  - What can a computer do at all?
  - Problems that can be solved by computer are called “decidable.”

- **Intractability (see Chapter 10)**
  - What can a computer do efficiently?
  - Problems that can be solved by a computer using no more time than some slowly growing (polynomial) function of the size of the input are called “tractable.”

### Introduction to Formal Proof

- **Deductive Proofs**
  - **Form**: From some initial statement $H$, called the *hypothesis* or the *given statement(s)*, provide a sequence of statements whose truth leads to a *conclusion* statement $C$; $C$ is *deduced* from $H$.
  - Typically given as a theorem of the form, “If $H$ then $C$,,” as in
    - Theorem: If $x \geq 4$, then $2^x \geq x^2$.
  - Perhaps, the most common type of proof...

- **Reduction to Definitions**
  - It is sometimes helpful to convert all terms in the hypothesis to their definitions ...
  - **Note**: Given the statement “If $H$ then $C,$”
    - contrapositive: “If not $C$ then not $H$”
    - converse: “If $C$ then $H$”
    - contradiction: “$H$ and not $C$ implies false”
  - **Note**: To prove that a statement $S$ is not a theorem, it suffices to show a *counterexample*.

- **Proof by Mathematical Induction**
  - **Form**: Prove a statement $S(X)$ about a family of objects $X$ (e.g. integers, trees) in three parts:
    - **Basis**: Show $S(X)$ holds for one or several small values of $X$ directly.
    - **Inductive Hypothesis**: Assume $S(Y)$ holds for values $Y \leq n$.
    - **Inductive Step**: Show that $S(n+1)$ holds using the inductive hypothesis.
Proof by Mathematical Induction

Example:

Prove that a binary tree with \( n \) leaves has \( 2^n - 1 \) nodes.

Formally, \( S(T) \): If \( T \) is a binary tree with \( n \) leaves, then \( T \) has \( 2^n - 1 \) nodes.

Induction is on the number of nodes in \( T \).

Basis: If \( T \) is a one-node tree, then has only one leaf; \( 1 = 2 \times 1 - 1 \), so OK.

Inductive Hypothesis: Assume \( S(U) \) holds for all binary trees \( U \) with at most \( k \) leaves

Hence, if \( U \) is a binary tree with \( k \) leaves, \( U \) has \( 2^k - 1 \) nodes.

Inductive Step: Consider a binary tree \( T \) with \( k+1 \) leaves ...

\( T \) must have two subtrees \( U \) and \( V \).

If \( U \) and \( V \) have \( u \) and \( v \) leaves, respectively, then \( T \) has \( u + v = k+1 \) leaves.

Proof by Mathematical Induction

Example (continued):

Prove that a binary tree with \( n \) leaves has \( 2^n - 1 \) nodes.

Inductive Step: Consider a binary tree \( T \) with \( k+1 \) leaves ...

\( T \) must have two subtrees \( U \) and \( V \).

If \( U \) and \( V \) have \( u \) and \( v \) leaves, respectively, then \( T \) has \( u + v = k+1 \) leaves.

Proof by Equivalence

Example (continued):

Form: Prove “\( X \) if and only if \( Y \)”

The proof has to be done in two steps:

Prove the if-part: Assume \( Y \) and prove \( X \).

Prove the only-if-part: Assume \( X \) and prove \( Y \).

Example: Equivalence of sets \( S \) and \( T \) can be shown when \( x \) is in \( S \) if and only if \( x \) is in \( T \).

Assume \( x \) is in \( S \); prove \( x \) is in \( T \).

Assume \( x \) is in \( T \); prove \( x \) is in \( S \).
Proof by Equivalence

Form: Prove “X if and only if Y.”

Remember:
- The if-part and only-if-part are converses of each other.
- One part, say “if X then Y,” says nothing about whether Y is true when X is false.
- An alternate, equivalent form of “if X then Y” is “if not Y then not X” – the latter is a contrapositive of the former.

Example: Balanced Parentheses
Two ways to define “balanced parentheses”:

1. Grammatically (GB):
   a) The empty string, $\varepsilon$, is balanced.
   b) If $w$ is balanced, then $(w)$ is balanced.
   c) If $w$ and $x$ are balanced, then so is $wx$.

2. By Scanning (SB): $w$ is balanced if and only if
   a) $w$ has an equal number of left and right parentheses
   b) Every prefix of $w$ has at least as many left as right parentheses

Theorem: A string of parentheses $w$ is GB if and only if $w$ is SB.

Proof (if-part):
- Assume $w$ is SB; prove it is GB by induction on $|w|$, the length of string $w$.

Basis: If $|w| = 0$ (i.e. $w$ is $\varepsilon$), then $w$ is GB by GB rule a.
Inductive Hypothesis: Suppose the statement “if SB then GB” is true for all $w$ where $|w| < n$.

Inductive Step: Show that the statement “if SB then GB” is true for all $w$ where $|w| \geq n$.

Case 1: $w$ is not $\varepsilon$, but has no nonempty prefix that has an equal number of ( and ). Then, $w$ must begin with ( and end with ); i.e. $w = (x)$.
- $x$ must be SB (why?).
- $x$ is GB by the Inductive Hypothesis.
- $(x)$ is GB by GB rule b; but $(x) = w$, so $w$ is GB.
Proof by Equivalence

**Inductive Step:** Show that the statement “if $SB$ then $GB$” is true for all $w$ where $|w| \geq n$.

- Case 2: $w=xy$, where $x$ is the shortest, nonempty prefix of $w$ with an equal number of ( and ), and $y \neq \varepsilon$.
  - $x$ and $y$ are both $SB$ (why?).
  - $x$ and $y$ are both $GB$ by the Inductive Hypothesis.
  - $w$ is $GB$ by $GB$ rule c.

Proof by Equivalence

**Theorem:** A string of parentheses $w$ is $GB$ if and only if $w$ is $SB$.

**Proof (only-if-part):**

- Assume $w$ is $GB$; prove it is $SB$ by induction on $|w|$, the length of string $w$.

**Basis:** If $|w|=0$ (i.e. $w$ is $\varepsilon$), then clearly $w$ obeys the conditions for being $SB$.

**Inductive Hypothesis:** Suppose the statement “$SB$ only if $GB$” is true for all $w \neq \varepsilon$ where $|w|<n$.

**Inductive Step:** Show that the statement “$SB$ only if $GB$” is true for all $w$ where $|w| \geq n$.

- **Case 1:** $w$ is $GB$ by $GB$ rule b; i.e. $w=(x)$ and $x$ is $GB$.
  - $x$ is $SB$ by Inductive Hypothesis.
  - Since $x$ has equal number of ( and ), so does $(x)$.
  - Since $x$ has no prefix with more ( than ), then so does $(x)$.

- **Case 2:** $w \neq \varepsilon$ is $GB$ by $GB$ rule c; i.e. $w=xy$ and both $x$ and $y$ are $GB$.
  - $x$ and $y$ are $SB$ by Inductive Hypothesis.
  - (Aside) Trickier than it looks: we have to argue that neither $x \neq \varepsilon$ nor $y \neq \varepsilon$, because if one were, the other would be $w$, and this rule application could not be the one that first shows $w$ to be $GB$.
  - Since each of $x$ and $y$ have equal number of ( and ), so does $xy$.
Proof by Equivalence

**Inductive Step:** Show that the statement “SB only if GB” is true for all \( w \) where \( |w| \geq n \).

*Case 2: continued ...*

If \( w \) had a prefix with more ) than (, that prefix would either be a prefix of \( x \) (contradicting the fact that \( x \) has no such prefix) or it would be \( x \) followed by a prefix of \( y \) (contradicting the fact that \( y \) also has no such prefix).

(Aside) Above is an example of **proof by contradiction** — we assumed our conclusion about \( w \) was false and showed it would imply something that we know is false.

---

**Languages**

- **Alphabet** = finite set of symbols
  - Examples: \( \{0,1\} \) (the binary alphabet) \( \{a,b,c,...,z\} \)
- **String** = finite sequence of symbols chosen from some alphabet
  - Examples: \( 01101 \)
  - **abracadabra**
- **Language**
  - = set of strings chosen from some alphabet

**Powers of an alphabet**

- If \( \Sigma \) is an alphabet, define \( \Sigma^k \) to be the set of strings of length \( k \), consisting of symbols in \( \Sigma \).
- The set of **all** strings over \( \Sigma \) is denoted \( \Sigma^* \); and,
  \[
  \Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \cdots \cup \omega
  \]
- The set of **nonempty** strings over \( \Sigma \) is denoted \( \Sigma^+ \); further,
  \[
  \Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \cdots \cup \omega
  \]
  \[\Sigma^* = \Sigma^+ \cup \{\varepsilon\}\]

**Note:**
- A language may be infinite, but there is some finite set of symbols of which all its strings are composed from.
- **Examples:**
  - The set of all **binary** strings consisting of some number of \( 0 \)'s followed by an equal number of \( 1 \)'s; that is, \( \{ \varepsilon, 01, 0011, 000111, \ldots \} \).
  - C (the set of compilable C programs)
  - **English**
**Languages**

- More Abstract Examples:
  - The set of binary numbers whose value is prime; that is, \{10, 11, 101, 111, 1011, \ldots\}
  - \(\Sigma^*\) is a language for any alphabet \(\Sigma\)
  - \(\emptyset\), the empty language, is a language over any alphabet.
  - \(\{\varepsilon\}\), the language consisting of only the empty string, is also a language over any alphabet.

**Problems**

- In automata theory, a *problem* is the question of deciding whether a given string is a member of some particular language.
- Hence, if \(\Sigma\) is an alphabet, and \(L\) is a language over \(\Sigma\), then the problem \(L\) is:
  
  Given a string \(w \in \Sigma^*\), decide if \(w \in L\).