Modeling Computation

Introduction to Formal Languages and Automata
Turing Machines and
Complexity, Computability, and Decidability
Different Types of Turing Machines

• Allow TM to stay still after reading input.
  – How can this be done with the standard machine?
• Use multiple tapes.
  – What do transitions look like?
• Use 2-dimensional tape.
• Use multiple read/write heads.
• Allow non-determinism.
• Restrict tape to be infinite in a single direction.
• Restrict alphabet to 2 symbols.
What effect do these modifications have on the power of a Turing machine?
Decision Problems

• Entscheidungsproblem
  – Is a set of first-order logical propositions universally valid (can be deduced from the axioms)?
• The easiest kind of problem to study by using Turing machines.
  – Also called: yes-or-no problems
  – Is the input string a member of the language?
  – Is the input string a prime number?
    • Same as recognizing the language of strings which lead to “yes”.
Decidability

• **Solvable** or **Decidable**:  
  – There exists an effective algorithm (Turing machine) that *solves* the problem (*decides* the language).

• **Unsolvable** or **Undecidable**:  
  – No effective algorithm that solves the problem (*decides* the language).
Halting Problem in Terms of TMs

- Does a TM exist which can determine whether a given TM will halt on some specified input?
Other Undecidable Problems

• Given two CFGs, do they generate the same language?

• Can the plane be tiled by a given set of tiles?

• Are there integer solutions to a given polynomial with integer coefficients?
Computability

- A function that can be computed by a Turing machine is *computable*.
- A function that cannot be computed by a Turing machine is *uncomputable*.
  - Example: *Busy Beaver Function*
The Complexity Class $\mathcal{P}$

Given a Turing machine $M$ and an input $w$, the **running time** $t_M(w)$ is the number of steps $M$ carries out on $w$ from the initial configuration to a halting configuration.

A deterministic Turing machine $M$ is **polynomially bounded** if there exists a polynomial $p(x)$ such that, for any positive integer $n$, $t_M(n) \leq p(n)$.

A language $L$ is **polynomially decidable** if there exists a polynomially bounded deterministic Turing machine that decides it.

*The set of all polynomially decidable languages is denoted by $\mathcal{P}$.*
The Complexity Class $\mathcal{NP}$

A nondeterministic Turing machine $M$ is **polynomially bounded** if there exists a polynomial $p(x)$ such that, for any input string $w$, at least one computation of $M$ on input $w$ halts in at most $p(|w|)$ steps.

A **verification algorithm** $A$ takes two arguments, $w$ and $u$. The string $u$ is called a **certificate**. A language $L$ is verified by a deterministic verification algorithm $A$ if, for every $w \in L$, there exists a certificate $u$ such that $A(w,u) = 1$.

A language $L$ is **polynomially verifiable** if there exists a deterministic polynomial-time algorithm (e.g. a polynomially bounded Turing machine) $A$ and a polynomial $q$ such that, for every $w \in L$, there exists a certificate $u$ such that $|u| \leq q(|w|)$ and $A(w,u) = 1$. That is, $A$ verifies $L$ in polynomial time.

The set of all polynomially verifiable languages is denoted by $\mathcal{NP}$. 
The Complexity Class \( \mathcal{NP} \)-Complete

A function \( f : \Sigma^* \rightarrow \Sigma^* \) is \textbf{polynomial-time computable} if there exists a polynomially bounded Turing machine which computes it.

Language \( L \subseteq \Sigma^* \) is \textbf{polynomial-time reducible} to language \( R \subseteq \Sigma^* \) if there exists a polynomial-time computable function \( r \) such that, for every \( w \in \Sigma^* \), \( w \in L \) iff \( r(w) \in R \). The function \( r \) is called a \textbf{polynomial-time reduction}.

\( L \) is \textbf{\( \mathcal{NP} \)-complete} if \( L \in \mathcal{NP} \) and every language \( L' \in \mathcal{NP} \) is polynomial-time reducible to \( L \).
Review Days

• 5/4 and 5/5
• Come bearing bribes questions