10 Decision Problems

A decision problem is a general question to be answered, usually possessing several parameters, or free variables, whose values are left unspecified. An instance of a problem is obtained by specifying particular values for all the problem parameters.

Example.

Hamiltonian Circuit

Instance A Graph $G = (V,E)$.

Question Does $G$ contain a Hamiltonian Circuit?

A "problem" is not the same as a "question".

"Does the following graph have a Hamiltonian Circuit?" is a question. It refers to a specific instance.

\[ 1 \quad 2 \quad 4 \quad 3 \quad 1 \]

is a cycle that contains every vertex.

A solution to a problem is an algorithm that answers the question that results from each instance. A decision problem is decidable if a solution exist and it is undecidable otherwise.

In general we are concerned with the questions of whether certain decision problems are decidable and if so, whether they can be decided in polynomial time.

According to Church's thesis, every computational device can be simulated by some Turing machine. Thus, a decision problem is decidable if and only if there is a Turing machine that solves the problem. To show that a decision problem is undecidable, we take the point of view that it suffices to show that no Turing machine solves the problem. Conversely, to show that a Turing machine
exists to solve a decision problem, it suffices to present an informal algorithm. If a decision problem is decidable, we want to know whether it is feasible. That is we want to know whether there is a polynomial time-bounded turing machine (This notion will be formally defined during this course.) that decides the problem.

Recalling that input to a Turing machine must be presented as a word over a finite alphabet, the first technical issue we must address is one of reasonable encodings. In order for a Turing machine to solve a decision problem about graphs, for example, graphs must be encoded as words over some finite alphabet. Furthermore, the encoding must be reasonable in the sense that the length of the word that represents the graph must be no more than a polynomial in the length of whatever is considered to be a natural presentation of a graph.

Reasonable encoding should be concise and not “padded” with unnecessary information or symbols. Numbers should be represented in binary, or any other concise representation, and should not be represented in unary. If we restrict ourselves to encoding schemes with these properties, then the particular choice of representation will not affect whether a given problem is feasible. For example, one might present a graph \( G = (V,E) \) either by a Vertex List and an Edge List or with an Adjacency matrix. For our purposes, either is acceptable. Once such a decision is made, it is straightforward to encode the presentation as a word over a finite alphabet. To wit, suppose that \( V = \{1, \ldots, k\} \) and \( E \) consists of pairs of the form \((e_1, e_2)\). Then,

1. Denote each integer by its 2-adic representation. Let \( c(i) \) be the 2-adic representation of the number \( i \).
2. For each \( I, I = 1, \ldots k \), let \( [c(i)] \) represent the vertex \( I \).
3. In general, if \( x_1, \ldots, x_n \) represent the objects \( X_1, \ldots, X_n \), let \((x_1, \ldots, x_n)\) represent the object \((X_1, \ldots, X_n)\). Thus, in particular, an edge \((e_1, e_2)\) is represented by the string \([c(e_1)], [c(e_2)]\).

In this manner every graph is representable as a string over the
finite alphabet \{ 0, 1, [\, ], (\, ) \}. Other data structures are encoded in a similar manner.
\[ A_{DFA} = \{ \langle B, w \rangle \mid B \text{ is a DFA that accepts } w \} \]

\[ E_{DFA} = \{ \langle A \rangle \mid A \text{ is a DFA and } L(A) = \emptyset \} \]

\[ EQ_{DFA} = \{ \langle A, B \rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B) \} \]
Theorem 4.4
The following is decidable:

Given DFA \( M \), whether \( L(M) = \emptyset \).

\( M \) accepts some word if and only if there is a path from start state to an accept state.

Algorithm:
mark start state
repeat
mark every state that has a transition coming into it from a state that is already marked
until no new states get marked
if no accept state is marked
then accept
else reject

Theorem 4.5
There is an algorithm to determine whether two finite automata are equivalent.

Let \( L_1 = L(M_1) \) and \( L_2 = L(M_2) \).

\[ (L_1 \cap \overline{L_2}) \cup (\overline{L_1} \cap L_2) \]

is symmetric difference accepted by some DFA \( M_3 \).

\( L(M_3) = \emptyset \iff L_1 = L_2. \)
$A_{DFA} = \{ \langle B, w \rangle \mid B \text{ is a DFA that accepts } w \}$

Th. 4.1 Proof: Design $M$ as follows:

*Input* $\langle B, w \rangle$.

- Simulate $B$ on $w$.
- If $B$ accepts $w$, then $M$ accepts $\langle B, w \rangle$, else $M$ rejects $\langle B, w \rangle$.

Some details:

Let $B = (Q, \Sigma, \delta, q_0, F)$ be input to $M$.

First, $M$ verifies that $B$ is a FA and $w$ is a string in $\Sigma^*$.

Then, $M$ simulates $B$ on $w$.

$M$ writes $B$'s current state and position of $B$'s head on $w$ on a tape. $M$ updates this info by reading $B$'s transition fn. $w$.

When $B$ reads the last symbol, then $M$ accepts or rejects accordingly.
Undecidable problems

Using the technique you learned on Tuesday, for any Turing machine $M$, let $<M>$ denote the code of $M$ as a string over a finite alphabet $\Sigma$.

A simple syntactic test can examine a string and determine whether it is the code of a T.M.

Consider the following Turing machine $U$ whose high-level description is:

- **input** $<M>$, $x$
- *if* $<M>$ is a code
  - *then* simulate $M$ on $x$
  - else reject

$U$ is called a universal T.M. It is the first example of a general-purpose stored-program computer.

Early computers had their programs hard-wired into them. Several years after Turing's 1936 paper, von Neumann and coworkers built the first computer that stored instructions internally in the same manner as data. Von Neumann knew and was influenced by Turing's universal machine.
Theorem. $L$ is decidable iff $\overline{L}$ and $\overline{L}$ are c.e.

Define $K = \{ \langle M \rangle \mid TM M accepts \langle M \rangle \}$.

1. $K$ is c.e.

Theorem. $K$ is not decidable.

Proof. We prove that $\overline{K}$ is not c.e. Suppose otherwise. Let $M$ be a TM that accepts $\overline{K}$; i.e., $L(M) = \overline{K}$.

Then, $\langle M \rangle \in \overline{K} \implies M$ accepts $\langle M \rangle$ 

\[ \implies \langle M \rangle \in K \]

By definition, for all TM $N$, $\langle N \rangle \in L(M) \iff \langle N \rangle \in K \iff N$ does not accept $\langle N \rangle$.

Thus, in particular, $\langle M \rangle \in L(M) \iff M$ does not accept $\langle M \rangle$. 
Acceptance Problem

Instance \( M \) and \( x \)

Question: Does \( M \) accept \( x \)?

Theorem. The accept problem is undecidable.

Proof. Let

\[
A_{TM} = \{(\langle M \rangle, x) \mid M \text{ accepts } x\}
\]

\( \langle M \rangle \in \mathcal{E} \iff (\langle M \rangle, \langle M \rangle) \in A_{TM} \)

Use \( \text{RHS} \) as a subroutine:

If \( A_{TM} \) is decidable, then so is \( \mathcal{E} \). Hence

\( A_{TM} \) is undecidable.
Consider the Halting Problem
\[ H_{TM} = \{ \langle M, w \rangle \mid M \text{ on input } w \text{ eventually halts} \} \]

Suppose \( \mathcal{A} \) is an algorithm that decides the Halting Problem. Then decide \( H_{TM} \) as follows:

Given \( \langle M, w \rangle \) and \( w \). Input \( \langle M, w \rangle \) to \( \mathcal{A} \).
If \( \mathcal{A} \) returns that \( M \) on \( w \) does not eventually halt, then \( M \) does not accept \( w \).
If \( \mathcal{A} \) returns that \( M \) on \( w \) does eventually halt, then run \( M \) on \( w \) (i.e., simulate \( U \) on \( \langle M \rangle \) and \( w \)).
If \( M \) accepts \( w \), then accept. Else do not accept.