Graph Connectedness (Decision) Problem

The decision problem: *Give a finite undirected graph G, is it connected?*

The corresponding language:

$$C = \{\langle G \rangle \mid G \text{ is a connected finite undirected graph}\}.$$

Consider this TM:

$M_C =$ "On input $\langle G \rangle$:

0. If the input string is not a valid encoding of a finite undirected graph, *reject.*
1. Mark the first node of G.
2. Repeat until no new nodes get marked:
3. Mark each node in G that is attached by an edge to an already marked node.
4. If all nodes are marked, *accept*; otherwise, *reject.*"

Assuming the encoding is as described earlier, here are some examples of strings that should get rejected in stage 0:

(1,(2

(1,3,4)((1,2),(1,3),(1,4),(3,4))

(1,2)((1,2),(1,1))

Consider the input string $(1, 2, 3, 4)((1, 2), (2, 3))$.
It's rejected in stage 4 because node 4 will not be marked.

Consider the input string $(1, 2, 3, 4, 5)((1, 2), (2, 3), (2, 4)(4, 5))$.
It's accepted in stage 4 because all nodes will be marked.

Observations about the general behavior of M_C:

- At least one node gets marked each time through the loop except the last.
- There are only finitely many nodes.
- Therefore M_C terminates on all inputs.
- Clearly, M_C accepts a string iff the graph it encodes is connected.
- Therefore M_C is a decider for the language C.

Overall conclusion:

- Stated formally: *C is a decidable language.*
- Stated informally: *Graph connectedness is a decidable problem.*

DFA Simulator - Acceptance Problem For DFAs

The decision problem: *Give a DFA D and a string w, does D accept w?*

The corresponding language:

$$A_{\mathrm{DFA}} = \{\langle D, w\rangle \mid D \text{ is a DFA that accepts input string } w\}.$$

Consider this TM:

$Sim_{\mathrm{DFA}} =$ "On input $\langle D, w\rangle$, where D is a DFA and w is a string:

0. Check that this is a valid encoding of a DFA together with a string in the corresponding input alphabet. If not, *reject.*
1. Simulate D on input w.
2. If the simulation ends in an accept state of D, *accept*; if not, *reject.*"

Remarks:

- Stage 0, the validity check, is usually not shown explicitly as it is here. Henceforth it will be omitted, but it is always implicitly assumed to be present.
- Stage 1 is itself a loop that iterates once for each symbol in w, consulting the transition function each time to determine the next state.

Observations about the general behavior of Sim_{DFA}:

- The loop implicitly present in stage 1 iterates $|w|$ times.
- Since $|w|$ is finite, stage 1 always halts.
- Therefore Sim_{DFA} terminates on all inputs.
- Clearly, Sim_{DFA} accepts a string $\langle D, w\rangle$ iff the DFA D accepts the string w.
- Therefore Sim_{DFA} is a decider for the language A_{DFA}.

Overall conclusion:

- Stated formally: A_{DFA} *is a decidable language.*
- Stated informally: *The acceptance problem for DFAs is decidable.*

TM Simulator - Acceptance Problem For TMs

The decision problem: *Given a TM T and a string w, does T accept w?*

The corresponding language:

$$A_{\text{TM}} = \{\langle T, w\rangle \mid T \text{ is a TM that accepts input string } w\}.$$

Consider this TM:

$Sim_{\text{TM}} =$ "On input $\langle T, w\rangle$:

1. Simulate T on input w.
2. If the simulation ends in T's accept state, *accept*.
   If it ends in a T's reject state, *reject*."

Remarks:

- Stage 1 is carried out iteratively by consulting the transition function to determine the next configuration at each iteration.
- This TM has been called a *universal Turing machine* because it is able to simulate the behavior of any other TM given an encoding of that TM.

Observations about the general behavior of Sim_{TM}:

- If the simulated TM T halts and accepts w, then Sim_{TM} halts and accepts $\langle T, w\rangle$.
- If the simulated TM T halts and rejects w, then Sim_{TM} halts and rejects $\langle T, w\rangle$.
- If the simulated TM T fails to halt on input w, then Sim_{TM} also fails to halt on input $\langle T, w\rangle$.
- Therefore Sim_{TM} is a recognizer, but not a decider, for A_{TM}.

Does there exist a decider for A_{TM}?

No! We'll soon see a proof that this language is undecidable.

Acceptance Problem For NFAs and Regular Expressions

Two decision problems:

1. *Given NFA N and string w, does N accept w?*
2. *Given regular expression R and string w, does R generate w?*

The corresponding languages:

1. $A_{\text{NFA}} = \{\langle N, w\rangle \mid N \text{ is an NFA that accepts input string } w\}$
2. $A_{\text{REX}} = \{\langle R, w\rangle \mid R \text{ is an regular expression that generates string } w\}$

Deciders for these languages:

$M_{A_{\text{NFA}}} =$ "On input $\langle N, w\rangle$, where N is an NFA and w is a string:
1. Convert N to an equivalent DFA D using the procedure we learned in class (and described on pp. 55-56 of Sipser).
2. Run Sim_{DFA} on $\langle D, w\rangle$.
3. If it accepts, *accept*; if it rejects, *reject*."

$M_{A_{\text{REX}}} =$ "On input $\langle R, w\rangle$, where R is a regular expression and w is a string:
1. Convert R to an equivalent NFA N using the procedure we learned in class (and described on pp. 67-69 of Sipser).
2. Run $M_{A_{\text{NFA}}}$ on $\langle N, w\rangle$.
3. If it accepts, *accept*; if it rejects, *reject*."

Overall conclusion:

1. A_{NFA} is a decidable language.
2. A_{REX} is a decidable language.

Exhaustive Testing Strategy

Decision problem: *Given DFA D, is there some string that D accepts?*

Corresponding language:

$$SOME_{\text{DFA}} = \{\langle D\rangle \mid D \text{ is a DFA and } L(D) \neq \Phi\}$$

Consider this TM:

$M =$ "On input $\langle D\rangle$, where D is a DFA:
1. For each possible string w (enumerated, say, in lexicographic order):
2. Run Sim_{DFA} on $\langle D, w\rangle$.
3. If it accepts, *accept*
4. If no $\langle D, w\rangle$ is accepted, *reject*."

Observations:

- There are infinitely many possible strings w to try.
- Therefore the loop will never terminate if the DFA accepts no strings.
- Stage 4 will never run.
- This TM never enters its reject state.
- It either accepts or runs forever.
- This TM is a recognizer, but not a decider, for $SOME_{\text{DFA}}$.

Exhaustive Testing Strategy (Continued)

Another decision problem: *Is there some string of length no more than k that the DFA D accepts?*

Corresponding language:

$$\{\langle D, k\rangle \mid D \text{ is a DFA and } D \text{ accepts some string of length} \leq k\}$$

Consider this TM:

$M' =$ "On input $\langle D, k\rangle$, where D is a DFA and k is a number:

1. For each possible string w of length $\leq k$ (enumerated, say, in lexicographic order):
2. Run Sim_{DFA} on $\langle D, w\rangle$.
3. If it accepts, *accept*
4. If no $\langle D, w\rangle$ is accepted, *reject*."

Observations:

- There are only finitely many strings of length $\leq k$.
- Therefore this TM halts on all inputs.
- Therefore this TM is a decider for this language.

Moral:

- Exhaustive testing will generally yield only a recognizer if there are infinitely many instances to test.
- Exhaustive testing may yield a decider if there are finitely many instances to test.

Acceptance Problem For CFGs

The decision problem: *Given CFG G and string w, does G generate w?*

Corresponding language:

$$A_{\text{CFG}} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates string } w\}$$

One possible approach: Try all derivations to see if any of them generate the given string.
Since there could be infinitely many derivations to try, the best this could yield is a recognizer for A_{CFG}.

Some facts about CFGs in Chomsky normal form (see pp. 106-109 and Problem 2.26 in Sipser):

- If G is a CFG in Chomsky normal form, then any nonempty string w in its language can be derived in exactly $2|w| - 1$ steps.
- There is a procedure for converting any CFG to an equivalent CFG in Chomsky normal form.

Consider this TM:

$M_{A_{\text{CFG}}} =$ "On input $\langle G\rangle$, where G is a CFG:

1. Convert G to an equivalent CFG G' in Chomsky normal form.
2. If $w = \varepsilon$:
3.   If G' contains the rule $S \to \varepsilon$, *accept*; else *reject*.
4. For each possible derivation consisting of $2|w| - 1$ steps in G':
5.   If the derivation generates w, *accept*.
6. If none of these derivations generate w, *reject*."

Observations:

- There are only finitely many possible $(2|w| - 1)$-step derivations in any CFG.
- Therefore stage 5 runs only finitely many times.
- Therefore this TM always halts.
- Therefore this TM is a decider for A_{CFG}.
- Therefore A_{CFG} is a decidable language.

Decidability of A_{CFL} Implies Decidability of any CFL

Theorem. Every CFL is decidable.

Proof. Let L be a CFL, and let G be a CFG that generates L.
Define a TM as follows:

$M_G =$ "On input string w:
1. Run $M_{A_{\text{CFG}}}$ on $\langle G, w \rangle$.
2. If it accepts, *accept*; if it rejects, *reject*."

Then:

- Since $M_{A_{\text{CFG}}}$ is a decider, stage 1 halts.
- Thus M_G is a decider.
- M_G accepts exactly those strings that G generates, so $\text{ACCEPT}(M_G) = L(G) = L$.
- Therefore M_G is a decider for L.
- Therefore the CFL L is decidable.

Emptiness Problem For DFAs

Decision problem: *Given DFA D, does D accept no strings at all?*

Corresponding language:

$$E_{\mathrm{DFA}} = \{\langle D\rangle \mid D \text{ is a DFA and } L(D) = \Phi\}$$

Consider this TM:

$M_{E_{\mathrm{DFA}}} =$ "On input $\langle D\rangle$, where D is a DFA:

1. Mark the start state of D.
2. Repeat until no more states get marked:
3.   Mark any state having a transition into it from any state already marked.
4. If no accept state is marked, *accept*; otherwise *reject*."

Observations:

- There are only finitely many states.
- Thus stage 3 runs only finitely many times.
- Therefore this TM always halts.
- Therefore it's a decider for E_{DFA}.
- Therefore E_{DFA} is a decidable language.

Emptiness Problem For CFGs

Decision problem: *Given CFG G, does G generate no strings at all?*

Corresponding language:

$$E_{\mathrm{CFG}} = \{\langle G \rangle \mid G \text{ is a CFG and } L(G) = \Phi\}$$

Consider this TM:

$M_{E_{\mathrm{CFG}}} =$ "On input $\langle G \rangle$, where G is a CFG:

1. Mark all terminal symbols in G.
2. Repeat until no new variables get marked:
3. Mark any variable A for which there is a rule $A \to U_1 U_2 \ldots U_k$ with all symbols $U_1, U_2, \ldots, U_k$ marked.
4. If the start variable is not marked, *accept*; otherwise *reject*."

Observations:

- There are only finitely many variables.
- Thus stage 3 runs only finitely many times.
- Therefore this TM always halts.
- Therefore it's a decider for E_{CFG}.
- Therefore E_{CFG} is a decidable language.

Subset and Equivalence Problems For DFAs

Two decision problems:

1. *Given two DFAs D_1 and D_2, is the language recognized by D_1 a subset of the language recognized by D_2?*
2. *Give two DFAs D_1 and D_2, are they equivalent?*

Corresponding languages:

1. $SUB_{\text{DFA}} = \{\langle D_1, D_2\rangle \mid D_1 \text{ and } D_2 \text{ are DFAs and } L(D_1) \subseteq L(D_2)\}$
2. $EQ_{\text{DFA}} = \{\langle D_1, D_2\rangle \mid D_1 \text{ and } D_2 \text{ are DFAs and } L(D_1) = L(D_2)\}$

Consider this TM for SUB_{DFA}:

$M_{SUB_{\text{DFA}}} =$ "On input $\langle D_1, D_2\rangle$, where D_1 and D_2 are DFAs:
1. Construct a DFA C such that $L(C) = L(D_1) - L(D_2)$.
2. Run $M_{E_{\text{DFA}}}$ on $\langle C\rangle$.
3. If it accepts, *accept*; if it rejects, *reject*."

Observations on $M_{SUB_{\text{DFA}}}$:

- $L(D_1) - L(D_2) = L(D_1) \cap \overline{L(D_2)}$, so stage 1 involves combining the intersection and complement constructions for DFAs from p. 46 and Exercise 1.14, respectively, of Sipser.
- Thus stage 1 always terminates since it requires finitely many steps.
- Stage 2 always terminates since $M_{E_{\text{DFA}}}$ is a decider.
- For any sets A and B,
  - $A - B$ consists of all elements of A that do not belong to B; so
  - $A - B$ is empty iff every element of A belongs to B; so
  - $A - B$ is empty iff $A \subseteq B$.
- Therefore this TM accepts $\langle D_1, D_2\rangle$ iff $L(D_1) \subseteq L(D_2)$.
- Therefore this TM is a decider for SUB_{DFA}.

Since $L(D_1) = L(D_2)$ if and only if $L(D_1) \subseteq L(D_2)$ and $L(D_2) \subseteq L(D_1)$, we can use $M_{SUB_{\text{DFA}}}$ to construct the following decider for EQ_{DFA}:

$M_{EQ_{\text{DFA}}} =$ "On input $\langle D_1, D_2\rangle$, where D_1 and D_2 are DFAs:
1. Run $M_{SUB_{DFA}}$ on $\langle D_1, D_2\rangle$. If it rejects, *reject*.
2. Run $M_{SUB_{DFA}}$ on $\langle D_2, D_1\rangle$. If it accepts, *accept*; otherwise *reject*."

Therefore:

1. SUB_{DFA} is a decidable language.
2. EQ_{DFA} is a decidable language.