CSCI 341–Fall 2024: Lecture Notes
Set 14: Church-Turing Thesis, Encodings, and Decidability

Edward Talmage

November 14, 2024

Recall from CSCI 311 that an algorithm is a finite, deterministic, sequence of steps to solve a problem. The Church-Turing Thesis says that the set of all algorithms is equivalent to the set of all Turing Machines. That is, there is an algorithm to solve a problem if and only if there is a Turing Machine that solves it.

Of course, that begs the question: How does a TM solve a problem? All they do is recognize (or decide) languages. It turns out, we can represent computational problems as languages, and the question of whether those languages are decidable.. For example,

Problem 1. Is there an algorithm for determining whether an arbitrary polynomial in two variables (e.g. $w^2 - 61y^2 - 1 = 0$) has integer solutions?

We can formulate this equivalently as a language question: Is $L = \{$2-variable polynomials with integer roots$\}$ a decidable language? If L is decidable, then there is a decider which can take a given 2-variable polynomial and determine whether it is in L, which is the same as determining whether it has integer roots.

Of course, not all problems are quite so obvious. We will not attempt to give a general proof, but will see a number of examples of how different types of problems, including those with non-Boolean return values, can be represented as languages we wish to decide.

1 DFA Decision Problems

Consider the problem of determining whether a DFA's language is empty. That is, given DFA D, return True if $L(D) = \emptyset$, False otherwise. To make this a decision problem, we need to encode a DFA so that it can be a string input to a TM. We know how to represent a DFA as a string: $D = (Q, \Sigma, \delta, q_0, F)$. We can convert that string to binary (using ASCII, Unicode, or some other encoding) to reduce the size of the tape language.

Observation: All kinds of objects (automata, grammars, TMs, mathematical expressions, problems, etc.) can be expressed as binary strings. Really, anything you can write can then be encoded. Given an object X, we use $\langle x \rangle$ to represent its encoding in some reasonable format.

Consider the set $E_{DFA} = \{w \mid w$ is the encoding of a DFA D with $L(D) = \emptyset\}$. E_{DFA} is decidable iff the problem of determining whether a DFA recognizes the empty language is solvable. In general, deciding a language is equivalent to solving a problem. (Turing-)Recognizing a language is equivalent to "semi-solving" a problem: if the answer is True, we are guaranteed to return it. If the answer is False, we may or may not return it.

Claim 1. *$E_{DFA} = \{\langle D \rangle \mid$ D is a DFA with $L(D) = \emptyset\}$ is decidable.*

Proof. We construct a decider for E_{DFA}:

On input $\langle D \rangle$, where D is a DFA:

1. If there is a path in D's state transition diagram from the start state of D to any final state of D, reject.
2. Otherwise, accept.

We need to describe how we determine whether there is a path from D's start state to any accept state. We can generally consider how to implement Breadth-First Search to check whether a graph G is connected:

1. Mark a vertex of G

2. While there are unmarked neighbors of marked vertices:
   (a) Mark all neighbors of marked vertices.
3. If all nodes are marked, the graph is connected, so accept. Reject otherwise.

For this specific problem, we do not need to know if the entire graph is connected, merely if the start state is connected to any accept state, so we can just check whether any accept state is marked at the end. If so, there is a path from the start state to an accept state, and if not, there is no such path. □

Next, consider the acceptance problem for DFAs:

Problem 2. Given a DFA M and a string w, is $w \in L(M)$?

Exercise: Reformulate this problem to express it as a language we would like to decide. How would you solve this problem? Can you give an algorithm to solve it for any M and w, that is guaranteed to halt in finite time?

$$A_{DFA} = \{\langle D, w\rangle \mid D \text{ is a DFA, } w \text{ is a string, } w \in L(D)\}$$

Claim 2. *A_{DFA} is decidable.*

Idea: A TM can simulate M on w, accept if M ends in an accept state, reject otherwise.

Proof. We describe a TM deciding A_{DFA}:

On input $\langle D, w\rangle$:

1. Verify that D is a DFA, w a string:
2. Find q_0 in $\langle D\rangle$ and write it on the tape, past the input.
3. To simulate D running on input w, find the first character of w, the transition on q_0, w in the description of δ, and update the q_0 written in the previous step to the new current state.
4. Repeat this simulation process until we have read all of w. If the current state at this point is in the list of final states in $\langle D\rangle$, accept. Else, reject.

□

Corollary 1. *A_{NFA} and A_{RE} are decidable.*

Proof. Convert the NFA or RE to a DFA using the algorithms we discussed earlier in the semester, then run the TM that decides A_{DFA}. □

This ability to use previous TMs as subroutines will make our lives a lot easier.

Problem 3. Can we tell if two DFAs recognize the same language? That is, given M_1 and M_2, is $L(M_1) = L(M_2)$?

First, we need to express this problem as a language:

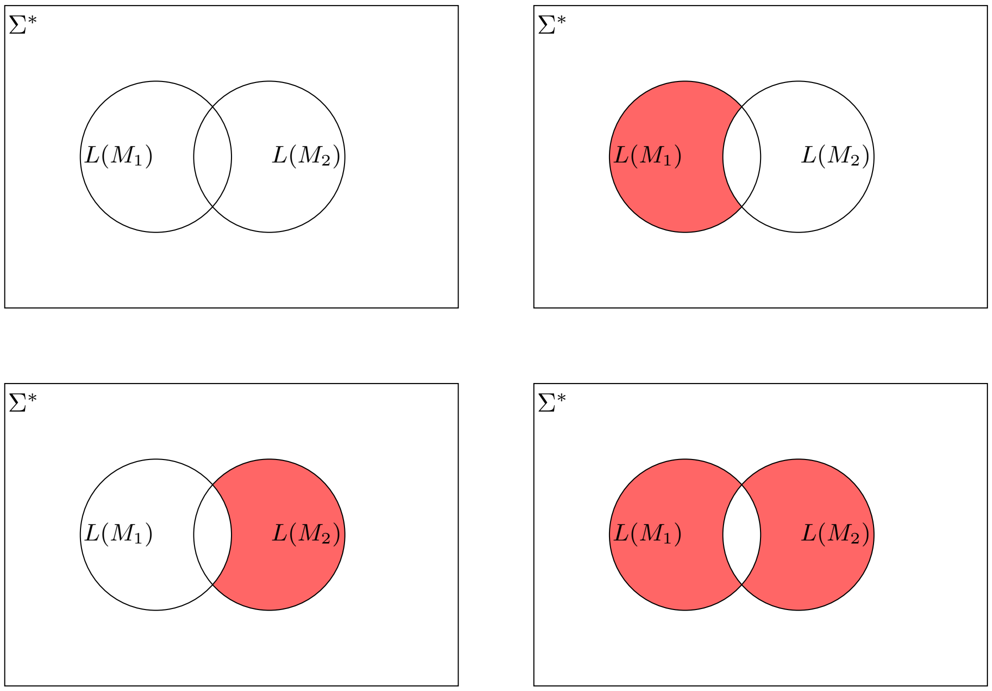
Exercise: Express this language, EQ_{DFA}, as a language in the style we used above for E_{DFA} and A_{DFA}.

$$EQ_{DFA} = \{\langle M_1, M_2\rangle \mid M_1, M_2 \text{ are DFAs, } L(M_1) = L(M_2)\}$$

Next, we want to reduce this to one of our previous DFA algorithms. Specifically, we will reduce to E_{DFA}, the emptiness checker.

Exercise: How can we use existing constructions to reduce this problem to checking whether a DFA's language is empty?

Idea: Consider the general Venn Diagram for two languages $L(M_1)$ and $L(M_2)$, as in the top left of the figure below. What we want to show is that there is nothing in $L(M_1)$ that is not also in $L(M_2)$ and also that there is nothing in $L(M_2)$ that is not in $L(M_1)$. That is, we want the sets $L(M_1) \cap \overline{L(M_2)}$ (top right) and $\overline{L(M_1)} \cap L(M_2)$ (bottom left) to be empty. To reduce this to a single question, union the two sets we want to be empty. If the resulting set, $(L(M_1) \cup \overline{L(M_2)}) \cup (\overline{L(M_1)} \cap L(M_2))$ shown in the bottom right, is empty, then the two languages of interest are equal. If the union is not empty, then one language contains a string the other does not, and they are not empty.



Proof.

On input $\langle M_1, M_2 \rangle$:

1. Construct a DFA D_L for $L = \left(L(M_1) \cap \overline{L(M_2)}\right) \cup \left(\overline{L(M_1)} \cap L(M_2)\right)$, using the union, intersection, and complement constructions for DFAs.
2. Run the TM for E_{DFA} on D_L.
3. If it accepts, $L = \emptyset$, and we accept $\langle M_1, M_2 \rangle$. Else, we reject.

$\square$

2 CFL Problems

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

Theorem 1. *A_{CFG} is decidable.*

Idea: We can recognize this language by doing a Breadth-First Search of possible parse trees starting from G's start variable. However, we may not be able to reject, since there may be infinitely-looping derivations, and we will never know whether we are on an infinite path, or if the derivation will end soon. To decide, we can use Chomsky Normal Form and the fact from homework about the length of derivations in CNF grammars.

Proof.

On input $\langle G, w \rangle$:

1. Check that G is a CFG and $w \in \Sigma^*$.

2. Convert G to equivalent grammar G' in CNF by our previous algorithm.
3. Generate all strings in $L(G') = L(G)$ which use $1, 2, \ldots, 2|w| - 1$ rule applications.
   - We can do this by trying all derivations of length 1, then all of length 2, etc.
   - If $|w| = 0$, check all derivations of length 1.
4. If we generated w at any point, accept. Else, reject.

We know that every string in $L(G)$ of length $|w|$ has a derivation in no more than $2|w| - 1$ rule applications in the CNF grammar G', so if we have not generated w by the end of step 3, we know we will never generate w. □

Problem 4. Can we determine whether a CFG generates the empty language?

This is also somewhat tricky, as any infinite recursion in a CFG leads to generating no strings on derivations including that recursion. But there may be other derivations that avoid the infinite recursion and generate a string. These infinite recursions are not even unusual. Consider the grammar

$$S \to aS \mid \varepsilon$$

The path that always uses the rule $S \to aS$ will loop infinitely, so we do not want to explore that path.

We express the problem as a language:

$$E_{CFG} = \{\langle G \rangle \mid G \text{ is a CFG with } L(G) = \emptyset\}$$

Theorem 2. *E_{CFG} is decidable.*

Idea: We can work backwards from variables which generate only terminals. First, consider those variables which generate terminals directly, then those which generate a combination of terminals and variables which can themselves generate terminals. By doing this repeatedly, we are collecting the set of variables whose individual languages are non-empty. Repeat this until we are not finding new variables for our set, then check to see if the start variable is one of those which has a non-empty language.

Proof.

On input $\langle G \rangle$:

1. Check that G is a CFG.
2. Mark all terminals appearing in rules in G.
3. Repeat:
   (a) Mark any variable V which has a rule $V \to U_1 U_2 \ldots U_k$, where every $U_i, 1 \leq i \leq k$ is marked.
   (b) If there is no such V, break.
4. If G's start variable is marked, reject. Else, accept.

We know this process will halt in finite time because it will halt after any iteration in which it does not mark a variable, so it can run for at most $|V|$ variables, marking at least one each time. □

Problem 5. Are two context-free grammars equivalent?

Expressed as a language:

$$EQ_{CFG} = \{\langle G_1, G_2 \rangle \mid G_1, G_2 \text{ are CFGs}, L(G_1) = L(G_2)\}$$

Exercise: Is EQ_{CFG} decidable?

We cannot use intersection and complement like we did for EQ_{DFA}, since the class of CFLs is not closed under either of those operations. We will actually show later that EQ_{CFG} is undecidable.

Theorem 3. *Every context-free language L is decidable.*

Exercise: How would you prove this?

Ideas:

- Simulate PDA with TM. Easy to simulate stack with tape. Problem: PDA is non-deterministic, and some branches may not halt, so if you first simulate a non-terminating branch before an accept branch, you will never decide. This approach could still work with a Non-deterministic TM.
- Use a CFG for the CFL, instead of a PDA, and simulate the CFG. Note that we can do this indirectly, since we have already considered the question of whether a CFG accepts a given string.

Proof. Let S be the TM we designed that decides A_{CFG}. Let G be a CFG for L, the given CFL we want to decide.

On input w:

1. Run TM S on $\langle G, w\rangle$.
2. If S accepts, accept. If S rejects, reject.

□