1 Decidability (4.1)

In this section we give some examples of languages that are decidable by algorithms.

A computational problem is **decidable** if and only if the language encoding the problem instances is decidable.

1.1 Encoding Inputs

First, we should mention that not all inputs are strings. In past examples and problems, we've generally worked with strings in a set alphabet. But, what if our input is an array of integers?

By definition, all Turing machine inputs are *strings*. If we're given an input that isn't a string, we need to **encode** the input (represent it as a string) first.

Before we discuss further, here is some notation:

- $\langle O \rangle$ is the string that represents, or encodes, the object O (O could be, for example, an integer, an array, etc.)
- $\langle O_1, O_2, \ldots, O_n \rangle$ is the single string that represents the tuple of objects $O_1, O_2, \ldots, O_n$.

In this way, problems that we care about can be reframed as languages of strings.

1.1.1 Example: Encoding Inputs

- Recognizes whether a string is a palindrome.

  $$\{w \mid w \in \{0,1\}^* \text{ and } w = w^R\}$$

  No encoding is necessary since w is a bitstring.

- Check whether a string is accepted by a DFA.

  $$\{\langle B, w \rangle \mid B \text{ is a DFA over } \Sigma, w \in \Sigma^*, w \in L(B)\}$$

  Here, we encode the pair B and w, where

  - B is a DFA.
  - w is a string.

  Essentially, for this language, saying that it's decidable is equivalent to checking if the string is accepted by the DFA.

  As a more general example, suppose we have a JSON file describing a DFA and a string. Then, we want to create an algorithm that checks whether this string is accepted by this DFA.

- Check whether the language of a PDA is infinite.

  $$\{\langle A \rangle \mid A \text{ is a PDA and } L(A) \text{ is infinite}\}$$

  Here, A is an encoding of a PDA.

1.2 Computational Problems

We now go over some example computational problems that are decidable.

1.2.1 Example: Recognizing Input Strings

Consider, again, the following language

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

This language contains the encodings of all DFAs together with strings that the DFAs accept. Then, the problem of testing whether the DFA B accepts an input w is the same as the problem of testing whether $\langle B, w\rangle \in A_{\text{DFA}}$.

Theorem 1.1

A_{DFA} is a decidable language.

Proof. (Idea.) We simply need to present a Turing machine M that decides A_{DFA}. Let M be the Turing machine which takes in an input $\langle B, w\rangle$, where B is a DFA and w is a string, and does the following:

1. Simulate B on input w (by keeping track of states in B, the transition function of B, etc.)[a]
2. If the simulation ends in an accept state, then accept. If it ends in a nonaccepting state, then reject.

A reasonable implementation of B is simply a list of its five components, Q, Σ, δ, q_0, and F. Then, when M receives its input, M first determines whether it properly represents a DFA B and a string w. If not, M rejects. Otherwise, M carries out the simulation directly; it keeps track of B's current state and B's current position in the input w by writing this information to its tape. Initially, B's current state is q_0, and B's current input position is the leftmost symbol of w. The states and position are updated according to the specified transition function δ. When M finishes processing the last symbol of w, it checks whether B is in an accept state. If so, M accepts. Otherwise, M rejects. □

[a] It's like writing a program that simulates the DFA.

1.2.2 Example: Emptiness Test

Consider the following language

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

Here, we want to determine whether a finite automaton accepts any strings at all.

Theorem 1.2

E_{DFA} is a decidable language.

Proof. A DFA accepts some string if and only if reaching an accept state from the start state by traveling along the arrows of the DFA is possible. To test this condition, we can design a TM T that uses a marking algorithm similar to what we used in the previous sections. For an input A, where A is a DFA, we describe the TM T as follows:

1. Check whether A is a valid encoding of a DFA; if not, reject.
2. Mark the start state of A.
3. Repeat until no new states get marked:
   - Loop over states of A and mark any unmarked state that has an **incoming** edge from a marked state.

4. If no accept state is marked, *accept*. Otherwise, *reject*.

This concludes the proof. □

1.2.3 Example: Equivalent DFAs

Consider the following language

$$EQ_{\text{DFA}} = \{\langle A, B\rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$$

Here, this language contains the encodings of all equivalent DFAs.

Theorem 1.3

EQ_{DFA} is a decidable language.

Proof. To prove this theorem, we make use of the previous theorem. We construct a DFA C from A and B, where C accepts only those strings that are accepted by either A and B but *not* by both. Thus, if A and B recognize the same language, then C will accept nothing. Recall that, for two sets X and Y:

$$X = Y \iff \left(X \cap \overline{Y}\right) \cup \left(Y \cap \overline{X}\right) = \emptyset$$

Using this as inspiration, the language of C is then given by:

$$L(C) = \left(L(A) \cap \overline{L(B)}\right) \cup \left(\overline{L(A)} \cap L(B)\right)$$

This expression is sometimes called the **symmetric difference** of $L(A)$ and $L(B)$. Then, we can build a new DFA that recognizes the symmetric difference of A and B. Then, we can check if this set is empty by using the example from the previous section. If so, then we *accept*. Otherwise, we *reject*.

The construction of this Turing machine is as follows; define a Turing machine M_3 that takes in the input $\langle A, B\rangle$ and does the following:

1. Check whether A and B are valid encodings of DFAs; if not, reject.
2. Construct a new DFA, D, from A and B using the algorithms for complementing and taking union of regular languages such that

$$L(D) = \text{Symmetric difference of } A \text{ and } B$$

3. Run the machine from the previous example on D.
4. If the machine from the previous example accepts, then accept. If it rejects, then reject.

□

1.3 Techniques

The techniques we have discussed are as follows:

- **Subroutines:** We can use decision procedures of decidable problems as subroutines in other algorithms.
- **Constructions:** We can use algorithms for constructions as subroutines in other algorithms.
  - Converting DFA to DFA recognizing complement or Kleene star.
  - Converting two DFAs/NFAs to one recognizing union, intersection, concatenation.

- Converting NFA to equivalent DFA.
- Converting regular expression to equivalent NFA.
- Converting DFA to equivalent regular expression.