5CCS2FC2: Foundations of Computing II

Tutorial Sheet 2

Solutions

2.1 Describe an algorithm, that can be implemented on a Turing Machine, that accepts the following language

$$\mathsf{E}_{DFA} = \{\langle A \rangle \ : \ A \text{ is a DFA such that } L(A) = \emptyset\}$$

In other words, E_{DFA} is the language of all DFAs that do not accept any words (including the empty string ε).

<u>SOLUTION:</u> When given the a description of a DFA $\langle A \rangle$, we want to check whether there is *any* word that reaches an accept state. If it does then we can return `false`, otherwise, return `true`.

This is the same as asking whether there is any *path* through the automaton that can reach an accept state.

We can do this can keeping a log of all the states that are reachable within k transitions, for $k = 0, 1, 2, \ldots$.

Let $R_0 = \{q_0\}$ be the set containing only the initial state and define

$$R_{k+1} \ = \ \{q' \in Q \ : \ q \text{ is reachable from some states in } R_k \text{ in one transition}\}$$

for each $k > 0$.

If ever there is some final state $q_F \in F$ that appears in R_k, we know that there is some path from q_0 to q_F, so the language must not be empty.

The algorithm can safely terminate when $R_n = R_{n-1}$, which must occur at some point since there are only finitely many states. In which case there must be no path from q_0 to a final accepting state, so the language must be empty.

2.2 Show that the following language is decidable by reducing it to the language E_{DFA},

$$\mathsf{E}_{NFA} = \{\langle A \rangle \ : \ A \text{ is an NFA such that } L(A) = \emptyset\}$$

<u>SOLUTION:</u> The same algorithm as above will also work for NFAs, since any paths through an NFA that reaches an accepting state corresponds to a word that is accepted by the automaton.

However, we can also show that the language is decidable by reducing it to the language E_{DFA} which we already know to be decidable. To do this, we need convert any NFA A into an equivalent DFA B such that

$$L(A) = \emptyset \qquad \Longleftrightarrow \qquad L(B) = \emptyset$$

since this means that

$$\langle A \rangle \in \mathsf{E}_{NFA} \qquad \Longleftrightarrow \qquad \langle B \rangle \in \mathsf{E}_{DFA}.$$

In FC1 we saw that we can always construct a DFA that is equivalent to an NFA using the *subset construction*. Hence it is sufficient to let B be the automaton given by applying the subset construction to A.

2.3 *(Tricky!)* Show that the following language is decidable, by reducing it to the language E_{DFA}:

$$\mathsf{EQ}_{DFA} = \{\langle A, B \rangle \ : \ A \text{ and } B \text{ are DFAs such that } L(A) = L(B)\}$$

In other words, EQ_{DFA} is the language of all pairs of *'equivalent'* DFAs that accept precisely the same words.

<u>SOLUTION:</u>

Two language are equivalent if they accept the same words. We want to convert the two automata A and B into a single automata C such that

$$L(A) = L(B) \qquad \Longleftrightarrow \qquad L(C) = \emptyset$$

since this means that

$$\langle A, B \rangle \in \mathsf{EQ}_{DFA} \qquad \Longleftrightarrow \qquad \langle C \rangle \in \mathsf{E}_{DFA}.$$

Once approach, would be to construct an automata C that accepted the language $L = \big(L(A) \cap \overline{L(B)}\big) \cup \big(L(B) \cap \overline{L(A)}\big)$ of all words that are either accepted by A but not B, or accepted by B but not A. This will be empty if A and B accept the same words.

The question is whether such a finite automata that accepts L is even possible? Is the language regular?

We note the following two facts:

- The complement of a regular language is regular: Take an automaton that recognises a language L and swap the accepting states with the non-accepting states. The new automaton now recognises the complement $\overline{L}$.
- The union of two regular languages is regular: We saw how to construct an NFA that accepted the union of two regular expressions in FC1, using epsilon jumps.

Using these two constructions we can always build an automata C that accepts the language

$$\begin{aligned} L &= \big(L(A) \cap \overline{L(B)}\big) \cup \big(L(B) \cap \overline{L(A)}\big) \\ &= \overline{\big(\overline{L(A)} \cup L(B)\big)} \cup \overline{\big(\overline{L(B)} \cup L(A)\big)} \end{aligned}$$

Alternatively, see Theorem 4.5 of Sipser.

2.4 Show that the following language is undecidable

$$\mathsf{EQ}_{TM} = \{\langle M_1, M_2\rangle : M_1, M_2 \text{ are TMs such that } L(M_1) = L(M_2)\}.$$

by a reduction from the language E_{TM}.

<u>SOLUTION:</u> We want to show that EQ_{TM} is at least as hard as E_{TM}, so that any algorithm for EQ_{TM} can be used as a subroutine to solve E_{TM}. So suppose that there is some algorithm `EQTM` that takes an input a pair of encodings $\langle M_1, M_2\rangle$ and returns `true` if and only if $L(M_1) = L(M_2)$.

That is to say, we have an algorithm that checks if two machines share the same language. So if we want to check whether a machine accepts the empty language, we can use this algorithm to compare it to a machine that we *know* accepts the empty language.

```
public static boolean ETM(String M) {

        String M1 = M;
        String M2 = "[code for M_empty]";

        return EQTM(M1,M2);
}
```

The encoding of the machine `M2` is a in-built constant that does not depend on `M`. We can choose any machine for $\mathcal{M}_2$ that accepts the empty language, such as a TM that immediately enters the reject state.

```
public static boolean M_empty(String w) {
        return false;
}
```

2.5 (i) Show that the language A_{TM} is recursively enumerable by constructing a sound and complete algorithm that recognises all words $\langle M, w\rangle$, where M encodes a TM that accepts w.

(ii) Hence, or otherwise, show that its complement $\overline{\mathsf{A}_{TM}}$ is *not* recursively enumerable.

2.6 *(Tricky!)*

(i) Show that the language $\overline{\mathsf{EQ}_{TM}}$ is not recursively enumerable by reducing A_{TM} to its complement EQ_{TM}. (In other words, that EQ_{TM} is not co-recursively enumerable.)

(ii) Show that the language $\overline{\mathsf{EQ}_{TM}}$ is also not co-recursively enumerable by reducing A_{TM} to $\overline{\mathsf{EQ}_{TM}}$. (In other words, that EQ_{TM} is not recursively enumerable.)

(It follows that $\overline{\mathsf{EQ}_{TM}}$ and EQ_{TM} are 'harder' than any recursively enumerable or co-recursively enumerable problem. There are not even any sound-and-complete algorithms for either problem)