Decidable languages

We can express different computational problems as languages. For example, testing whether a particular DFA accepts the given string:

\[ L_{\text{DFA}} = \{ \langle A, w \rangle \mid A \text{ is a DFA,} \]
\[ \text{that accepts the input string } w \}. \]

Here, \( \langle A, w \rangle \) represents a pair:

- encoding of the DFA \( A \) (list of five ingredients: \( Q, \Sigma, \delta, q_0, F \));
- input string \( w \).

The task of deciding whether DFA \( A \) accepts a string \( w \) is equivalent to checking if the pair \( \langle A, w \rangle \) is in the language \( L_{\text{DFA}} \).

**Theorem.** \( L_{\text{DFA}} \) is a decidable language.

**Proof.** We design a TM \( M \) that decides the language \( L_{\text{DFA}} \).

On the input \( \langle A, w \rangle \), the machine \( M \) will simulate the automaton \( A \) on \( w \), and accept/reject according to the automaton’s decision.

First, \( M \) scans the input and determines if the input properly represents a DFA (which we denote as \( A \)) and a string (which we denote as \( w \)). If not, \( M \) rejects.
Second, \( M \) simulates \( A \). It keeps track of \( A \)'s current state and \( A \)'s current position in the input \( w \) by writing the information directly on the tape.

In the beginning, the input of \( M \) is \( w \), and the head position is the leftmost symbol of \( w \). The states and the positions are updated according to the transition function \( \delta \). When \( M \) is finishing processing the last symbol of \( w \), it goes to accept/reject state depending on whether \( A \) is in the accept/reject state.

Similarly define
\[
L_{\text{NFA}} = \{ \langle A, w \rangle \mid A \text{ is an NFA that accepts the input string } w \}\.
\]

**Theorem.** \( L_{\text{NFA}} \) is a decidable language.

**Proof.** We present a TM \( M' \) that decides \( L_{\text{NFA}} \): on the input \( \langle A, w \rangle \), \( M' \) does the following:

1. Converts \( A \) into equivalent DFA \( A' \), by using the procedure that was studied in the course.
2. Run the machine \( M \) from the previous theorem on the input \( \langle A', w \rangle \).
3. If \( M \) accepts – accepts, otherwise – rejects.

One more example. Let
\[
L_{\emptyset} = \{ \langle A \rangle \mid A \text{ is a DFA and } L(A) = \emptyset \}\.
\]
I.e. all DFAs that do not accept anything.

**Theorem.** \( L_{\emptyset} \) is a decidable language.

**Proof.** A DFA \( A \) accepts some string if and only if reaching one of the accept states by travelling along the arrows of the DFA is possible. Therefore, a TM \( \hat{M} \) will test if there exists such a path.

For example, in the automaton

![Automaton Diagram](image-url)
there is a path \( q_0 \to q_1 \to q_3 \to q_4 \). This correspond to the input 010. Therefore \( L(A) \neq \emptyset \) as \( 010 \in L(A) \).

TM \( \hat{M} \) works as follows.

1. Mark the start state of \( A \).
2. Repeat until no new states are marked:
   - Mark any unmarked state that has an incoming arrow from any state that was marked already.
3. If no accept state is marked – accept, otherwise – reject.

For the example above, \( \hat{M} \) will mark the states in the following order: \( q_0 \to q_1 \to q_2 \to q_3 \to q_4 \).

\( q_4 \) is marked, so \( \hat{M} \) rejects (\( A \) accepts at least one string).

Undecidable languages

Define:

\[
L_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ accepts the input } w \}.
\]

Theorem. \( L_{TM} \) is undecidable.

Note. We present a proof based on a technique called “diagonalisation”.

Proof. We prove by contradiction. Assume, that there exists a Turing machine \( H \), where

\[
H(\langle M, w \rangle) = \begin{cases} 
\text{accepts,} & \text{if } M \text{ accepts } w, \\
\text{rejects,} & \text{if } M \text{ does not accept } w \text{ (either rejects or loops).}
\end{cases}
\]
Now we construct a new machine \(D\), which uses \(H\) as a subroutine. On input \(\langle M \rangle\), \(D\) does the following:

1. Runs \(H\) on input \(\langle M, \langle M \rangle \rangle\).

2. Outputs the opposite of what \(H\) outputs. That is, if \(H\) accepts – \(D\) rejects; if \(H\) rejects – \(D\) accepts.

In summary,

\[
D(\langle M \rangle) = \begin{cases} 
\text{accepts}, & \text{if } M \text{ does not accept } \langle M \rangle, \\
\text{rejects}, & \text{if } M \text{ accepts } \langle M \rangle.
\end{cases}
\]

Question: what happens when we run \(D\) with its own encoding \(\langle D \rangle\) as an input? In this case

\[
D(\langle D \rangle) = \begin{cases} 
\text{accepts}, & \text{if } D \text{ does not accept } \langle D \rangle, \\
\text{rejects}, & \text{if } D \text{ accepts } \langle D \rangle.
\end{cases}
\]

No matter what \(D\) is supposed to do, it does the opposite. Contradiction. Therefore such \(H\) does not exist.

**Practise session**

1. Define the language

\[L_{\text{REX}} = \{ \langle R, w \rangle \mid R \text{ is a regular expression that generates the string } w \} \]

Show that \(L_{\text{REX}}\) is a decidable language.

*Solution.* We construct a TM \(M\) that on the input \(\langle R, w \rangle\) does the following:

1. Converts \(R\) into an equivalent NFA \(A\) by using the procedure for conversion that we studied.

2. Gives an input \(\langle A, w \rangle\) to the TM that decides \(L_{\text{NFA}}\).

3. If \(\langle A, w \rangle \in L_{\text{NFA}}\) – accepts, otherwise – rejects.

2. Define the language:

\[L_{\text{DFAEQ}} = \{ \langle A, B \rangle \mid A \text{ and } B \text{ are DFA and } L(A) = L(B) \}\]

Prove that \(L_{\text{DFAEQ}}\) is a decidable language.
Solution. We construct a new DFA $C$, which accepts strings that are accepted by either $A$ or $B$, but not by both\footnote{Such an automaton is easy to build: it runs $A$ and $B$ in parallel and accepts if and only if exactly one of $A$ and $B$ accepts}. Then

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

\begin{itemize}
  \item If $L(A) = L(B)$, then $L(A) \cap \overline{L(B)} = \emptyset$ and $\overline{L(A)} \cap L(B) = \emptyset$, hence $L(C) = \emptyset$.
  \item If $L(A) \neq L(B)$, then there exists $w \in L(A)$, $w \notin L(B)$ (or vice versa). Then $w \in L(A) \cap \overline{L(B)}$ (or, respectively, $w \in \overline{L(A)} \cap L(B)$) and therefore $w \in L(C)$ and $L(C) \neq \emptyset$.
\end{itemize}

So $L(A) = L(B)$ if and only if $L(C) = \emptyset$.

We construct a TM $M$ as follows. On the input $\langle A, B \rangle$ it does the following:

1. Constructs $C$ as described.
2. Runs TM that decides the language $L_{\emptyset}$ on $\langle C \rangle$.
3. If $\langle C \rangle \in L_{\emptyset}$ – accepts. If $\langle C \rangle \notin L_{\emptyset}$ – rejects.

3. Define the language

$$L_1 = \{ \langle A \rangle \mid A \text{ is a DFA that accepts at least one string of the form } 1^* \}.$$  

Prove that $L_1$ is decidable.

Solution. We construct TM $M$ that decides $L_1$. On the input $\langle A \rangle$, $M$ does the following:

1. Constructs a DFA $B$ that accepts exactly language described by $1^*$. 

2. Constructs a DFA $C$, such that

$$L(C) = L(A) \cap L(B).$$

3. Checks if $\langle C \rangle \in L_\emptyset$. If no – accepts, if yes – rejects.

Let us justify the construction.

- If $\langle C \rangle \in L_\emptyset$ then $L(C) = \emptyset$ and so $L(A) \cap L(B) = \emptyset$. This means that for each $w \in L(A)$, it holds that $w \notin L(B)$ and therefore $w$ does not have the form $1^*$.

- If $\langle C \rangle \notin L_\emptyset$, then $L(C) \neq \emptyset$ and $L(A) \cap L(B) \neq \emptyset$. Thus there exists $w$, such that $w \in L(A)$ and $w \in L(B)$. This means that $w$ has the form $1^*$ and $w \in L(A)$. Correct.

4. Define the language

$$L_{k-\text{STR}} = \{ \langle A, k \rangle \mid A \text{ is a DFA and } L(A) \text{ consists of exactly } k \text{ strings, } k \in \mathbb{N} \}.$$  

Prove that $L_{k-\text{STR}}$ is decidable.

Proof. We construct a TM $M$, which decides $L_{k-\text{STR}}$. On the input $\langle A, k \rangle$, $M$ does the following.

1. Checks the number of states of $A$. Denote this number by $p$.

2. Constructs a DFA $B$, that accepts all strings of length $p$ or longer. Also constructs a DFA $C$, such that $L(C) = L(A) \cap L(B)$.

3. Generates all strings of length $\leq p - 1$ and tests whether each string is accepted by $A$. Counts the number of such strings, denote this number by $c_A$.

4. Tests whether $L(C) = \emptyset$.

5. If $L(C) = \emptyset$ and $c_A = k$ – accepts, otherwise – rejects.

Let us show that $M$ does what we want.

- First, note that due to the pumping lemma, if $A$ accepts any string of length $\geq p$, then it accepts infinitely many strings. This condition is tested by testing if $L(C) = \emptyset$.

- Provided $A$ does not accept any strings of length $\geq p$, $c_A$ is exactly the cardinality of $L(A)$. Thus $M$ accepts if and only if $|L(A)| = k$.

\[ \square \]