Léonard Brice, Mathieu Sassolas
leonard.brice@ulb.be, mathieu.sassolas@ulb.be

Session 3:
Undecidability and reductions

Solutions

1 Decidable problems

1. *Prove that the following problem is decidable (an intuitive algorithm is sufficient, as long as it is rigorously explained):*

$$\textsc{Accessibility} : \begin{cases} \textit{Input:} & \textit{A directed graph } G \textit{ (encoded by its adjacency matrix) and two sets of vertices } X \textit{ and } Y \\ \textit{Question:} & \textit{Does there exist a path in } G \textit{ from a vertex of } X \textit{ to a vertex of } Y \textit{?} \end{cases}$$

☛ The following algorithm solves that problem: mark every vertex $u \in X$ blue, and every other vertex red.

Then, repeat the following operations as long as there are blue vertices: for each blue vertex w, mark all of its red successors blue. Then, mark w green.

In the end, there is a path from X to Y if and only if at least one vertex $v \in Y$ is marked green.

2. *Prove that the following problem:*

$$\textsc{RegNonEmptiness} : \begin{cases} \textit{Input:} & \textit{A non-deterministic finite automaton } A \\ \textit{Question:} & \textit{Do we have } \mathcal{L}(A) \neq \emptyset \textit{?} \end{cases}$$

reduces to Accessibility.

☛ We define the mapping f as follows: for each non-deterministic finite automaton A, we set $f(A) = (G, X, Y)$, where:

- G is the underlying graph of A (the vertices of G are the states of A, and there is an edge from vertex p to vertex q in G if and only if there is a transition, with any label, from state p to state q in A);
- X is the set of initial states in A;
- Y is the set of final states in A.

Let us now show that f is a reduction. First, the mapping f is computable:

If the language $\mathcal{L}(A)$ contains at least one word $w = w_1 \dots w_n$, then there exists a run $q_0 \xrightarrow{w_1} q_1 \xrightarrow{w_2} \dots \xrightarrow{w_n} q_n$ of A, where q_0 is initial and q_n is final, that accepts w. Then, the path $q_0 \dots q_n$ is a path from X to Y in G.

Conversely, if $q_0 \dots q_n$ is a path from X to Y in G, then for each $i \in \{1, \dots, n\}$, there exists a letter w_i such that $q_{i-1} \xrightarrow{w_i} q_i$ is a transition in A, and the word $w_1 \dots w_n$ is then accepted by A.

Therefore, the automaton A is a positive instance of RegNonEmptiness if and only if the tuple (G, X, Y) is a positive instance of Accessibility, hence we have reduced the former to the latter.

3. *What can be deduced about the problem* RegNonEmptiness*?*

☛ That problem reduces to a decidable problem, and is therefore decidable.

2 Acceptance and halting problems

In the coming questions, we aim at showing that the acceptance problem of Turing machines:

$$\text{A}_{\text{TM}} : \begin{cases} \textit{Input:} & \textit{A Turing machine } M \textit{ and a word } w \\ \textit{Question:} & \textit{Does } M \textit{ accept } w\textit{?} \end{cases}$$

and the halting problem of Turing machines:

$$\textsc{Halt} : \begin{cases} \textit{Input:} & \textit{A Turing machine } M \textit{ and a word } w \\ \textit{Question:} & \textit{Does } M \textit{ halt on } w\textit{?} \end{cases}$$

are undecidable.

1. *Assume that* A_{TM} *is decidable. Explain why the following problem is decidable:*

$$\textsc{Diagonal} : \begin{cases} \textit{Input:} & \textit{A Turing machine } M \\ \textit{Question:} & \textit{Does } M \textit{ reject the encoding of } M\textit{?} \end{cases}$$

   ☛ If A_{TM} is decidable, then there exists a Turing machine A that takes as input the encoding of a machine M and a word w, and that accepts if M accepts w, and rejects otherwise. We can then define the machine D, that:

   - takes as input the encoding of a machine M;
   - simulates the machine A to decide whether M accepts the encoding of M;
   - rejects if M accepts, and accepts if M rejects.

   That machine decides the problem Diagonal.

2. *Deduce undecidability for* A_{TM}.

   ☛ Since the problem Diagonal is decided by the machine D, then either D accepts D, or D rejects D. But then, by definition of Diagonal, if D accepts D, then D rejects D; and if D rejects D, then D accepts D. Both cases are absurd, hence the starting hypothesis, the decidability of A_{TM}, is false.

3. *Let* M *be a Turing machine, and* w *be a word. Define a Turing machine* M' *and a word* w' *such that* M *accepts* w *if and only if* M' *halts on* w'.

   ☛ We define M' as equal to the machine M, but where the state q_{rej} is replaced by a fresh state q_{sink} in which the machine M' loops forever whatever it reads. We define $w' = w$.

   Then, if M accepts w, then M' accepts w', and therefore terminates on it. Conversely, if M' terminates on w', then it accepts it (since M' has no rejecting state), which means that M accepts w.

4. *Prove that* Halt *is undecidable.*

   ☛ Let us assume that Halt is decidable. Then, given a machine M and a word w, the following algorithm:

   - construct M' and w' as previously;
   - decide whether M' halts on w';

   decides the problem A_{TM}. As shown in Question 1, that is impossible.

   More shortly: the following justification is sufficient: since M' and w', as defined previously, can be effectively constructed from M and w, the problem A_{TM} reduces to Halt. Since A_{TM} is undecidable, so is Halt.

3 Reductions

Check whether the following problems are decidable. If they are, describe a Turing machine or an algorithm in pseudo-code that decides them. If not, prove it via reduction from an undecidable problem.

1. $$\textsc{Riemann} : \begin{cases} \textit{Input:} & \textit{A word } w \\ \textit{Question:} & \textit{Is Riemann's hypothesis}^1 \textit{ true?} \end{cases}$$

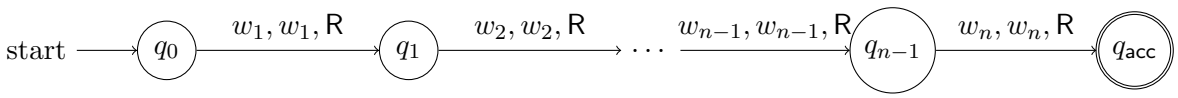
   ☛ Decidable.

   Indeed, let us consider the two following cases: if Riemann's hypothesis is true, then a Turing machine deciding that problem is the machine that immediately accepts. If it is false, then a Turing machine deciding that problem is the machine that immediately rejects.

2. $$\textsc{Inclusion}_{\text{TM}} : \begin{cases} \textit{Input:} & \textit{Two Turing machines } M_1 \textit{ and } M_2 \\ \textit{Question:} & \textit{Does } \mathcal{L}(M_1) \subseteq \mathcal{L}(M_2) \textit{ hold?} \end{cases}$$

   ☛ Undecidable.

   Let us prove it via reduction from the problem A_{TM}: let M be a Turing machine and w be a word. Let us define M_1 as a Turing machine that accepts only the word w:

   Let us now defined $M_2 = M$. Then, we have $\mathcal{L}(M_1) \subseteq \mathcal{L}(M_2)$ if and only if M accepts the word w. Therefore, the problem A_{TM} reduces to this problem. Since the former is undecidable, so is the latter.

3. $$\textsc{Inclusion}_{\text{aut}} : \begin{cases} \textit{Input:} & \textit{Two deterministic finite automata } A_1 \textit{ and } A_2 \\ \textit{Question:} & \textit{Does } \mathcal{L}(A_1) \subseteq \mathcal{L}(A_2) \textit{ hold?} \end{cases}$$

   ☛ Decidable.

   Since A_2 is deterministic, we can define the automaton $\bar{A}_2$, that accepts the language $\Sigma^* \setminus \mathcal{L}(A_2)$, simply by exchanging accepting and non-accepting states. Then, we can define the automaton A as the product automaton of A_1 and A_2, that accepts the language $\mathcal{L}(A_1) \cap \mathcal{L}(\bar{A}_2)$. Then, we have $\mathcal{L}(A_1) \subseteq \mathcal{L}(A_2)$ if and only if the language $\mathcal{L}(A)$ is empty, which is decidable by Question 1.3.

4. $$\textsc{EmptyWord} : \begin{cases} \textit{Input:} & \textit{A Turing machine } M \\ \textit{Question:} & \textit{Question: Does } \mathcal{L}(M) \textit{ contain the empty word?} \end{cases}$$

   ☛ Undecidable.

[1] Riemann's hypothesis states that *Riemann's zeta function*:

$$\zeta : \begin{cases} \mathbb{C} & \to \mathbb{C} \\ s & \mapsto \sum_{n \geq 1} \frac{1}{n^s} \end{cases}$$

is such that every non-trivial zero s of ζ has real part $\Re(s) = \frac{1}{2}$. The study of the zeta function is strongly linked to the study of prime numbers and, therefore, to cryptography. Proposed by Bernhard Riemann in 1859, that conjecture is still open today. The Clay Mathematics Institute offers US$1 million to anyone who would solve it.

Let us prove it by reduction from $\mathrm{A_{TM}}$: let M be a machine and w be a word. Let us define a machine M' as follows: first, it writes the word w on the tape. Then, it simulates M. Thus, the machine M' accepts the empty word if and only if M accepts w. Therefore, the problem $\mathrm{A_{TM}}$ reduces to this problem. Since the former is undecidable, so is the latter.

5.

$$\textsc{BoundedSpace} : \begin{cases} \textit{Input:} & \textit{A Turing machine } M \textit{ and a word } w \\ \textit{Question:} & \textit{Is there an integer } B \textit{ such that running } M \textit{ on input } w \textit{ does not use more than } B \textit{ cells of the tape?} \end{cases}$$

☛ Undecidable.

Let us prove it by reduction from the halting problem. Let M be a Turing machine, and w a word. Let us define the machine M' that simulates the machine M, but that, after each transition, goes to the end of the tape and writes a fresh symbol \$ instead of the first blank, and then moves back the reading head to where it was and continues the simulation. Then, the machine M' uses a bounded number of cells to run of w if and only if the machine M halts on w. Therefore, the halting problem reduces to this problem. Since the former is undecidable, so is the latter.

6.

$$\textsc{PythonZeroDiv} : \begin{cases} \textit{Input:} & \textit{A Python function } \texttt{f} \\ \textit{Question:} & \textit{Are there arguments such that } \texttt{f} \textit{ throw a } \texttt{ZeroDivisionError} \textit{ exception?} \end{cases}$$

(We assume here that the code can be ran on a computer with arbitrary memory; in particular, there is no limit on the size of the arguments.)

☛ Undecidable, by reduction from Halt: let M be a Turing machine and let w be a word. We can then define the following Python function `f`:

```
def f(x):
    (Simulate M on w)
    return(1 / 0)
```

Clearly, the function `f` can send a `ZeroDivisionError` exception if and only if M does halt on w, hence the problem PythonZeroDiv is undecidable.