

Solution Sheet n°5

Solution of exercise 1: We suppose we are given a reasonable coding in the alphabet $\{0, 1\}$ of the Turing machines with $\Sigma = \{0, 1\}$ and $\Gamma = \{0, 1, \sqcup\}$ (cf. Lecture). We consider the following languages

$$\begin{aligned} \text{Acc} &= \{ \langle \mathcal{M}, w \rangle \mid \mathcal{M} \text{ is a Turing machine which accepts } w. \} \\ \text{Halt} &= \{ \langle \mathcal{M}, w \rangle \mid \mathcal{M} \text{ is a Turing machine which halts on } w. \} \\ \text{Halt}_\emptyset &= \{ \mathcal{M} \mid \mathcal{M} \text{ is a Turing machine which halts on the empty word.} \} \\ \text{MaxStep} &= \left\{ \langle n, S(n) \rangle \left| \begin{array}{l} S(n) \text{ is the maximal number of steps that} \\ \text{an } n\text{-state Turing machine can do before halting} \\ \text{when starting on the empty word.} \end{array} \right. \right\}. \end{aligned}$$

The language Acc was proved to be undecidable during the lecture. We first show that the (un)decidability of Acc is equivalent to the (un)decidability of Halt .

First suppose that we can decide Halt . To decide Acc do as follows.

On input $(\mathcal{M}, w)$:

1. Decide whether $\mathcal{M}$ halts on w . If not, reject. Otherwise go to step 2.
2. Run $\mathcal{M}$ on w . If $\mathcal{M}$ rejects w then reject. If $\mathcal{M}$ accepts w , then accept.

Conversely suppose that we can decide Acc . We can decide Halt as follows:

On input $(\mathcal{M}, w)$:

1. Modify $\mathcal{M}$ into a machine $\widetilde{\mathcal{M}}$ by replacing each transition to the rejecting state for a transition to the accepting state.
2. Decide whether $\widetilde{\mathcal{M}}$ accepts w . If $\mathcal{M}$ accepts w , then accept. Otherwise reject.

Now for the questions of the exercise sheet, we have:

1. Suppose towards a contradiction that $\text{Halt}_\emptyset$ is Turing decidable. Then we can decide Halt as follows.

On input $(\mathcal{M}, w)$:

- (a) Write on a second tape the code of a Turing machine $\mathcal{M}_w$ which, on the empty input, acts as follows:
 - i) writes w and then goes back to the left hand end of the tape in the initial state;
 - ii) then works just as $\mathcal{M}$.
- (b) Decide whether $\mathcal{M}_w$ halts or not on the empty word. If $\mathcal{M}_w$ halts on the empty word, then accept. Otherwise reject.

2. Suppose towards a contradiction that MaxStep is decidable. Then we can decide $\text{Halt}_\emptyset$ as follows:

On input $\mathcal{M}$:

- (a) Modify $\mathcal{M}$ into a machine $\widetilde{\mathcal{M}}$ by replacing each transition to the rejecting state for a transition to the accepting state.
- (b) For n the number of states of $\widetilde{\mathcal{M}}$, compute $S(n)$. This can be achieved by testing for $i = 0$ to ∞ whether $(n, i) \in \text{MaxStep}$; since MaxStep is decidable, the computation will eventually halt.
- (c) Simulate the first steps of $\widetilde{\mathcal{M}}$ on the empty word. If a halting state is reached, accept. If no halting state is reached after $S(n)$ steps, reject.

Solution of exercise 2: This solution is based on Sipser, M. (2006). *Introduction to the Theory of Computation*, pp. 203–209.

1. **Reduction of PCP to MPCP:** Notice that there exists a match for an instance $I = \langle p_1, \dots, p_m \rangle$ of PCP if and only if there exists a match for one of the instances $I_1, I_2, \dots, I_n$ of MPCP where $I_i = \langle p_i, p_{i+1}, \dots, p_n, p_0, \dots, p_{i-1} \rangle$ for $i = 1, \dots, m$.

Reduction of MPCP to PCP Let $\star$ and $\diamond$ be symbols which do not belong to A and let us denote by $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$ the set of non empty words on a finite alphabet Σ . We define two Turing computable functions $p, s : A^+ \rightarrow (A \cup \{\star\})^+$ by

$$\begin{aligned} p(a_1 a_2 \cdots a_n) &= \star a_1 \star a_2 \cdots \star a_n \\ s(a_1 a_2 \cdots a_n) &= a_1 \star a_2 \star \cdots a_n \star. \end{aligned}$$

Notice that for all $v, w \in A^+$, $p(vw) = p(v)p(w)$ and $s(vw) = s(v)s(w)$. Moreover for each $v \in A^+$ we have $p(v)\star = \star s(v)$. Now consider the instance of MPCP on the alphabet A

$$I = \left\langle \frac{x_1}{y_1}, \dots, \frac{x_n}{y_n} \right\rangle$$

where $x_1, \dots, x_n, y_1, \dots, y_n \in A^+$ and a pair (x, y) is depicted as $\frac{x}{y}$. We define a corresponding instance of PCP with $m + 2$ pairs on the alphabet $A \cup \{\star, \diamond\}$:

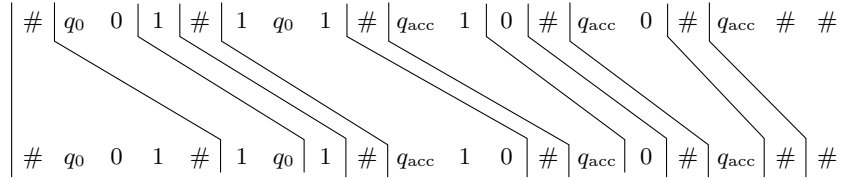
$$I' = \left\langle \frac{p(x_1)}{\star s(y_1)}, \frac{p(x_1)}{s(y_1)}, \frac{p(x_2)}{s(y_2)}, \dots, \frac{p(x_n)}{s(y_n)}, \frac{\star \diamond}{\diamond} \right\rangle,$$

Notice that the only pair both whose strings start with the same symbol is $\frac{p(x_1)}{\star s(y_1)}$, while the only pair for which both strings end with the same symbol is $\frac{\star \diamond}{\diamond}$. Hence any match for I' is necessarily of the following form: it starts with the pair $\frac{p(x_1)}{\star s(y_1)}$, followed by some pairs among the $\frac{p(x_i)}{s(y_i)}$ for $1 \leq i \leq n$ and then ends with the pair $\frac{\star \diamond}{\diamond}$. We see that any match for I' therefore corresponds to a match for I which starts with the pair (x_1, y_1) .

Conversely if $(i_1, \dots, i_k)$ is a match for I , then we have $i_1 = 1$ and $x_1 x_{i_2} \dots x_{i_k} = y_1 y_{i_2} \dots y_{i_k}$. It follows that the following sequence constitutes a match for I' :

$$\frac{p(x_1)}{\star s(y_1)}, \frac{p(x_{i_2})}{s(y_{i_2})}, \dots, \frac{p(x_{i_k})}{s(y_{i_k})}, \frac{\star \diamond}{\diamond}.$$

2. Let $M = (Q, \Sigma = \{0, 1\}, \Gamma = \{0, 1, \sqcup\}, \delta, q_0, q_{\text{acc}}, q_{\text{rej}})$ be (the code of) a Turing machine and $w \in \Sigma^*$ be an input¹. Assume that Q and Γ are disjoint and let $\#$ be a new symbol which does not belong to Γ . We define an instance $I_{M,w}$ of MPCP on the alphabet $A = Q \cup \Gamma \cup \{\#\}$ which is Turing computable from M, w . The idea is to define $I_{M,w}$ so that a match encodes the computation history of an accepting run of M on w . Let us first give an example. Suppose $M = (\{q_0, q_{\text{acc}}\}, \{0, 1\}, \{0, 1, \sqcup\}, \delta, q_0, q_{\text{acc}})$ with notably $\delta(q_0, 0) = (q_0, 1, R)$, $\delta(q_0, 1) = (q_{\text{acc}}, 0, L)$, and $w = 01$ then $I_{M,w}$ is designed so that the following is a match:



In general, we define the first pair of $I_{M,w}$ (the starting pair for a match) to be:

- r0) $\frac{\#}{\#q_0w\#}$ if w is non empty and $\frac{\#}{\#q_0\sqcup\#}$ otherwise.

Other pairs are added to $I_{M,w}$ using the following rules:

- r1) For each transition $\delta(q, a) = (r, b, R)$ with $q \neq q_{\text{rej}}$, we add $\frac{qa}{br}$.
- r2) For each transition $\delta(q, a) = (r, b, L)$ with $q \neq q_{\text{rej}}$ and each tape symbol $c \in \Gamma$ we add $\frac{cqa}{rcb}$.
- r3) For each $a \in \Gamma$ we add $\frac{a}{a}$ (this allows to copy unchanged symbols from one configuration to the next).
- r4) We add $\frac{\#}{\#}$ and $\frac{\#}{\#\sqcup}$ (this allows to go from one configuration to the next by possibly adding a blank $\sqcup$, in case the head has reached the right end of the configuration).
- r5) Finally for every $a \in \Gamma$ we add the pairs $\frac{aq_{\text{acc}}}{q_{\text{acc}}}$, $\frac{q_{\text{acc}}a}{q_{\text{acc}}}$ and $\frac{q_{\text{acc}}\#\#}{\#}$ (this allows to conclude a match when the accepting state q_{acc} is reached).

Since applying rule r0) gives us a strictly longer string on the “bottom”, and rules r1) to r4) add the same amount of symbols to both strings, a *match* must contain q_{acc} from rule r5). So it is the encoding of an accepting run of M on w .

3. Assume PCP is decidable. We construct a decider for the acceptance problem as follows.

On input M, w :

- (a) compute the instance $I_{M,w}$ of MPCP corresponding to M, w as described in the previous item.
- (b) compute the corresponding instance $I'_{M,w}$ of PCP as in the first item.
- (c) Decide the existence of a match for $I'_{M,w}$.

¹We assume for convenience that M never attempts to run off the left-hand of the tape while computing on w . This requires first modifying M to prevent this behaviour. Can you see how to perform this in a Turing computable way?