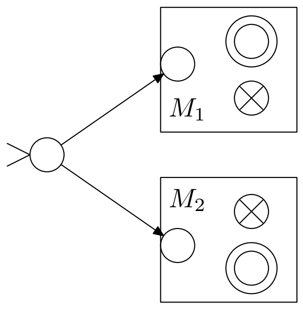
T-79.1001 **Syksy 2005**
Introduction to Theoretical Computer Science (T)
Session 10
Answers to demonstration exercises

4. **Problem**: Prove that the class of recursively enumerable languages is closed under unions and intersections. Why cannot one prove that the class is closed under complements in a similar way as in the case of recursive languages, i.e. simply by interchanging the accepting and rejecting states of the respective Turing machines?

   Solutions: Let L_1 and L_2 be recursively enumerable languages ($L_1, L_2 \in RE$). Then, there exists Turing machines M_1 and M_2 such that $L(M_1) = L_1$ and $L(M_2) = L_2$. We can now form machines $M_\cup$ and $M_\cap$ that recognize the languages $L_1 \cup L_2$ and $L_1 \cap L_2$.

   Union: The Turing machine $M_\cup$ is formed by composing the machines M_1 and M_2 as follows[1]:

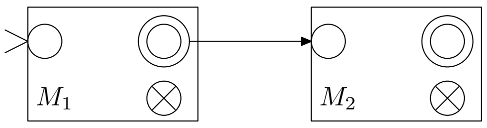
   $M_\cup$:



   The machine is nondeterministic and it initially chooses whether it simulates M_1 or M_2. Since nondeterministic and deterministic Turing machines have the same expressive power and $M_\cup$ recognizes $L_1 \cup L_2$, $L_1 \cup L_2 \in RE$.

   Intersection: The machine $M_\cap$ is formed as follows:

   $M_\cap$:



   Given input x, machine $M_\cap$ first simulates the computation $M_1(x)$. If it halts (in the accepting state q_{acc}), $M_\cap$ continues with computation $M_2(x)$. If both computations accept, $x \in L_1 \cap L_2$ and $M_\cap$ accepts. Thus, $L_1 \cap L_2 \in RE$. (To be precise, $M_\cap$ has to store the input x somewhere so that it can be given to M_2 after M_1 has finished.)

   Complementation: When a language L belongs to class $RE - R$ (= recursively enumerable but not recursive), then a Turing machine M that recognizes it can reject a word x in two different ways:

   (a) M halts in state q_{rej}; or

   (b) M does not halt at all.

   We now construct a machine $\overline{M}$ where the accepting and rejecting states are switched. Now also $\overline{M}$ rejects x if the computation does not halt, so $x \notin L(M) \cup L(\overline{M})$, so $L(\overline{M}) \neq \overline{L}$.

   It can be proved (exercises 1d and 2b) that if $L \in RE - R$, then $\overline{L} \notin RE$.

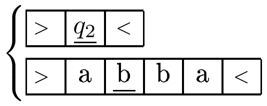
[1]Note that we again suppose that the machines have disjoint sets of states.

5. **Problem**: Show that Turing machines that have only *one* internal state in addition to their accepting and rejecting states are capable of recognising exactly the same languages as the standard machines with arbitrarily many states. For simplicity, you may assume that the simulating machines have multiple tapes and may also keep their tape heads stationary (direction code 'S') in a transition. How many internal states would be needed to effect the simulation on single-tape machines?

   Solution:

   Let M be a Turing machine with n states. It is possible to simulate M with a single-state two-tape machine M' if we add a symbol q_i to the tape alphabet for each state q_i of M. The first tape of M' is used for the actual computation while the second tape only holds the current state of the machine.

   For example, the configuration $(q_2, a\underline{b}ba)$ of M would be represented as:



   If M has a transition $\delta(q_i, a) = (q_j, b, \Delta)$, then M' has a transition $\delta(q'_0, (a, q_i)) = (q'_0, (b, q_j), (\Delta, S))$, where q'_0 is the sole state of M'. The read/write head of the second tape stays in place since otherwise the state information of the machine would be lost.

   Formally the construction is as follows: Let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}})$ be a standard-issue Turing machine where $Q = \{q_0, \ldots, q_n\}$. We construct a 2-tape machine $M' = (\{q'_0\}, \Sigma, \Gamma \cup \{q_0, \ldots, q_n\}, \delta', q'_0, q'_{\text{acc}}, q'_{\text{rej}})$, where

$$\begin{aligned}\delta' =& \{(q'_0, (a, q_i), q'_0, (b, q_j), (\Delta, S)) \mid \delta(q_i, a) = (b, q_j, \Delta), q_j \notin \{q_{\text{acc}}, q_{\text{rej}}\}\} \\ &\cup \{(q'_0, (a, q_i), q', (b, q)) \mid \delta(q_i, a) = (b, q, (\Delta, S)), q \in \{q_{\text{acc}}, q_{\text{rej}}\}\} \ .\end{aligned}$$

   It might seem that the same construction could be done also using a two-track machine. However, this works only when M does not move its read/write head at all; as soon as M moves its head, the state information is lost. If M has a convenient form, some of its states may be removed by writing the state information in the end of the tape, but in the worst case this also demands n states so M' is not necessarily smaller than M.

6. **Problem**:

   (a) Prove that any decision problem that has only finitely many possible inputs is decidable.

   (b) Prove that the problem "Does the decimal expansion of π contain 100 consecutive zeros?" is decidable. What does this result tell you about (i) the decimal expansion of π, (ii) the notions of decidability and undecidability?

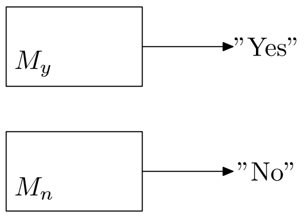
   Solution

   (a) If a decision problem has a finite number of possible inputs, it is possible to construct a Turing machine that encodes all inputs and the correct answers into its states. Thus, a problem is always decidable if there are only a finite number of inputs.

   (b) The decision problem "Does the decimal expansion of π contain 100 consecutive zeros?" has only one input, namely, π so by the preceding point it is decidable.

   However, it is not easy to come up with the actual algorithm to compute the answer. The natural way would be to generate the digits of π one by one and check whether there are 100 consequtive zeroes. Unfortunately this only semidecides the problem; if the answer is "no", the machine never halts.

The problem has only one answer: either π has 100 consequtive zeroes or it does not have. Thus, one of the following Turing machines decides it:



The machine M_y accepts the input and M_n rejects it. Unfortunately, we cannot know which one of the machines is the correct one (though M_y seems to be more probable).

As we see, the concept of decidability is a very weak one. A problem may be decidable even if we cannot solve it in practice, for example, if we do not have enough computational resources.