T-79.148 **Autumn 2003**
Introduction to Theoretical Computer Science
Tutorial 12
Solutions to the demonstration problems

4 **Problem**:

Prove, without appealing to Rice's theorem, that the following problem is undecidable:

Given a Turing machine M; does M accept the empty string?

Solution:

First we define a language $L = \{M \mid M \text{ halts with the input } \varepsilon\}$. Now, L is recursive if and only if the decision problem in the exercise statement is decisive. Next we show that the language $H = \{Mw \mid M \text{ halts with input } w\}$ can be recursively reduced to L (denoted $H \leq_m L$) so L is at least as difficult as H. Since H is not recursive, L may not be recursive, either.

The concept of a recursive reduction is defined as follows: Let $A \subseteq \Sigma^*$ and $B \subseteq \Gamma^*$ be languages. Now $A \leq_m B$ if and only if there exists a recursive function$f : \Sigma^* \to \Gamma^*$ such that

$$\forall w \in \Sigma^* : w \in A \Leftrightarrow f(w) \in B \ .$$

In this case we want to find a function f such that $f(Mw) \in L$ if and only if $Mw \in H$. In practice this means that we want to find a systematic way to construct a Turing machine M' that halts with an empty input exactly when M halts with $w = w_1 w_2 \cdots w_n$.

Fortunately, this is an easy thing to do: M' starts by writing w to its tape and after that it simulates M. Now M' stops only if M stops.

Formally, f can be defined as:

$$f(\langle Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}}\rangle, w_1 w_2 \cdots w_n) = \langle Q', \Sigma, \Gamma, \delta', q_0', q_{\text{acc}}, q_{\text{rej}}\rangle,$$

where

$$\begin{aligned} Q' &= Q \cup \{q_i' \mid 0 \leq i \leq n\} \\ \delta' &= \delta \cup \{\langle q_i', \varepsilon, q_{i+1}', w_{i+1}, R\rangle \mid 0 \leq i < n\} \\ &\quad \cup \{\langle q_n', x, q_n', x, L\rangle \mid x \in \Gamma \cup \{<\}\} \\ &\quad \cup \{\langle q_n', >, q_0, >, R\rangle\} \end{aligned}$$

Since we add only a finite number of states and transitions to M (n has to be finite), f is trivially recursive.

5. **Problem**: Prove the following connections between recursive functions and languages:

(i) A language $A \subseteq \Sigma^*$ is recursive ("Turing-decidable"), if and only its characteristic function

$$\chi_A : \Sigma^* \to \{0, 1\}, \qquad \chi_A(x) = \begin{cases} 1, & \text{if } x \in A; \\ 0, & \text{if } x \notin A \end{cases}$$

is a recursive ("Turing-computable") function.

(ii) A language $A \subseteq \Sigma^*$ is recursively enumerable ("semidecidable", "Turing-recognisable"), if and only if either $A = \emptyset$ or there exists a recursive function $g : \{0, 1\}^* \to \Sigma^*$ such that

$$A = \{g(x) \mid x \in \{0, 1\}^*\}.$$

Solution: We start by defining five simple helper machines:

- **1** writes '1' to the input tape, moves the read/write head to right and stops.
- **0** writes '0' to the tape and stops.
- C empties the input tape, moves the head to the beginning of the tape and stops.
- NEXT reads the input $x \in \Sigma^*$ and replaces it with the lexicographic successor of x.
- $Cmp^{i,j}$ compares the contents of the input tapes i and j of a multi-tape Turing machine and accepts if they are identical.

Since the machines are simple, they are not presented here.

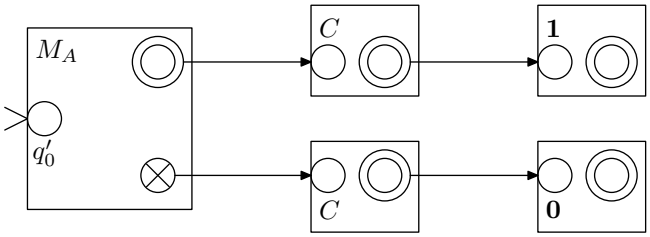
(i) [$\Rightarrow$] Let $A \subseteq \Sigma^*$ be a recursive language. Then there exists a Turing machine M_A:

$$M_A = \langle Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}} \rangle$$

such that

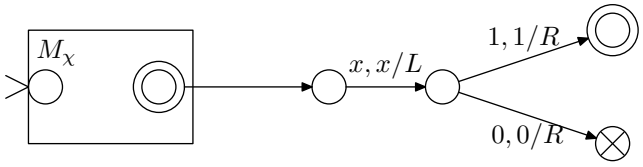
$$\forall w \in \Sigma^* : w \in L \Leftrightarrow (q_0, w) \vdash^*_{M_A} (q_{\text{acc}}, \alpha) \quad \text{ja}$$
$$w \notin L \Leftrightarrow (q_0, w) \vdash^*_{M_A} (q_{\text{rej}}, \alpha)$$

We construct a machine M by combining M_A with machines **1**, **0**, C as follows:



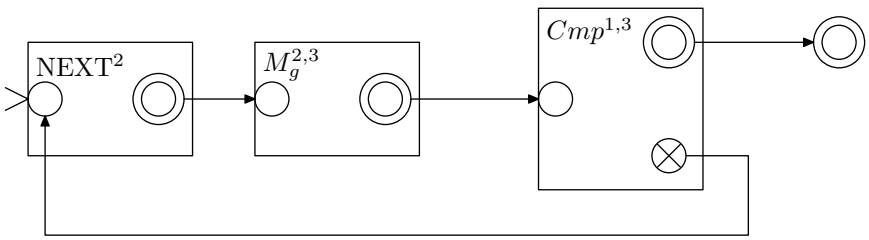
If $w \in L$, then M_A accepts w. After that M clears the tape and writes 1 to the tape. Otherwise 0 is written. Since A is recursive, M_A halts always so also M halts and it computes the function $\chi(w) = \begin{cases} 1, w \in A \\ 0, w \notin A \end{cases}$ that is the characteristic function of A.

[$\Leftarrow$] Suppose that the function $\chi(w)$ is recursive. Then there exists a Turing machine M_χ that computes it. We can now construct a machine M as follows:



Now M accepts w whenever $\chi(w) = 1$ and rejects it when $\chi(w) = 0$, so M decides the language A and A is recursive.

(ii) If $A = \emptyset$, then trivially $A \in RE$ and $g(x) = 0$ is its characteristic function.
If there exists a function g that fulfills the conditions, then there exists a Turing machine M_g that computes g. We can trivially modify it so that it becomes a 2-tape machine $M_g^{1,2}$ that computes g but stores the result in the second tape instead of the first. We now construct a 3-tape machine as follows:



The machine gets its input from its first tape and it stays untouched for the whole computation. In each iteration M_A replaces the bit string x on the second tape by its lexicographic successor y, computes $g(y)$ and writes the output on the third tape. Finally, the contents of tapes 1 and 3 are compared and if they match, the word is accepted, otherwise the iteration proceeds into the next round.

[$\Leftarrow$] Consider the word $w \in A$. Suppose that a recursive function g that fulfills the conditions exists. Then $w = g(x)$ for some $x = x_1 x_2 \cdots x_n$ where n is finite. Since each finite string has a finite number of predecessors in the lexicographic order, NEXT eventually generates x, $M_g^{2,3}$ generates w on the third tape and M_A accepts the word. Thus, M_A recognizes the language A so $A \in RE$.

[$\Rightarrow$] Next, suppose that $A \in RE - \{\emptyset\}$. Then there exists a Turing machine M_A that recognizes it. We now define a helper machine $M_{A,i}$ that simulates M_A for i steps. The machine $M_{A,i}$ accepts x if M_A accepts it using at most i steps, and rejects it otherwise. We note that $M_{A,i}$ always halts.

We construct the function g with the help of $M_{A,i}$. Every input x and bound i is encoded into bit strings using the function $c(x, y) = 0^x 1 0^y$. We define that $g(c(x, y)) = x$, if $M_{A,y}$ accepts x. We define that $g' : \{0,1\}^* \to \{0,1\}^*$ is the function:

$$g'(w) = \begin{cases} x, & w = 0^x 1 0^y \text{ and } M_{A,y}(x) \text{ accepts} \\ x_0, & \text{otherwise} , \end{cases}$$

where $x_0 \in A$. Finally, $g(x) = d(g'(x))$ where d is a function that maps a bit string 0^x into the xth element of n Σ^* in the lexicographic order. The value of g' may be computed in a finite time since $M_{A,y}(x)$ always halts. Thus, g' is recursive and so also g is.

Note that while g always exists, it is not always possible to find it since in the general case it is an undecidable problem to find an element $x_0 \in A$ that is needed for the definition.

6. **Problem**: Show that all context-sensitive languages can be recognised by linear-bounded automata. (Make use of the fact that in applying the grammar's production rules, the length of the sentential form under consideration can never decrease, except in the special case of the empty string.) Deduce from this result the fact that all context-sensitive languages are recursive.

   Solution:

   By definition, all rules of a context-sensitive languages are of the form where the right side is at least as long as the left side (apart from the possible rule $S \to \varepsilon$).

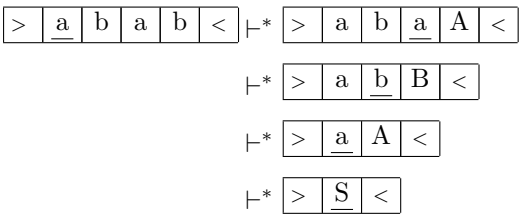
   A context-sensitive language can be recognized by a linear-bounded automaton that non-deterministically moves into some place of the input and applies one of the rules of the grammar from right to left. Since the string may only become shorter, no new space is needed. Also, if symbols are removed, then it is trivial to construct a machine that removes all extra space. If we have only the initial symbol S on the tape at the end, the word is accepted.

   Consider the following context-sensitive grammar:

$$\begin{aligned} S &\to aA \mid bB \\ aA &\to abB \mid ab \\ bB &\to baA \mid ba \end{aligned}$$

   A linear-bounded automaton recognizing this language would work as follows for the input

abab:



The machine described above is not total, since it is possible that it ends in an infinite loop. For example, if the grammar in question is:

$$S \to \varepsilon$$
$$ab \to ba$$
$$ba \to ab,$$

then all computations with a non-empty input fail to terminate.

We can fix the problem by noting that since the length of the tape may not increase, the number of possible configurations is finite. This number is of the magnitude $q \times n \times |\Gamma|^n$, where q is the number of states, n the length of input, and $|\Gamma|$ the size of the tape alphabet.

We can totalize the machine by adding a counter for it that counts the number of steps taken. The easiest way to do this is to have a two-track machine that keeps the counter on the second track encoded as a binary number. This number is then increased with each step of the original machine. When the counter reaches the limit, we can reject the word as the machine is in a loop.

Finally, we have to check that we can implement the counter without breaking the linear space bound. To encode a number $q \times n \times |\Gamma|^n$ we need $k = \log_2(q) + \log_2(n) + n\log(|\Gamma|)$ bits that is linear with respect to n. Even though $k > n$, we can squeeze it into the available space by encoding it in a suitable base.

7. **Problem**:

   Show that every language generated by an unrestricted grammar can also be generated by a grammar where no terminal symbols occur on the left hand side of any production.

   Solution: We can systematically construct a grammar G' that fulfills the conditions and generates the same language as a given grammar G by adding a new nonterminal A_a for each symbol $a \in \Sigma$, replacing a by A_a in each rule of the grammar, and finally adding a rule $A_a \to a$.

   Formally: Let G be an unrestricter grammar $G = (V, \Sigma, P, S)$. We construct a grammar $G' = (V', \Sigma, P', S')$, where

   $$V' = V \cup \{A_a \mid a \in \Sigma\}$$

   Each rule $r = x_1 \cdots x_n \to x_{n+1} \cdots x_{n+m}$ of G where $x_i \in V$ is transformed into:

   $$c(r) = y_1 \cdots y_n \to y_{n+1} \cdots y_{n+m}$$

   where

   $$y_i = \begin{cases} x_i, & x_i \in V - \Sigma \\ A_{x_i}, & x_i \in \Sigma \end{cases} .$$

   Now the set of rules P' can be defined as follows:

   $$P' = \{c(r) \mid r \in P\} \cup \{A_a \to a \mid a \in \Sigma\} .$$

Consider the grammar from the Exercise 4:

$$\begin{aligned} S &\to aA \mid bB \\ aA &\to abB \mid ab \\ bB &\to baA \mid ba \end{aligned}$$

By using the construction, we get a grammar:

$$\begin{aligned} S &\to A_aA \mid A_bB \\ A_aA &\to A_aA_bB \mid A_aA_b \\ A_bB &\to A_bA_aA \mid A_bA_a \\ A_a &\to a \\ A_b &\to b \end{aligned}$$

8. **Problem**:

   Show that every context-sensitive grammar can be put in a normal form where the productions are of the form $S \to \varepsilon$ or $\alpha A\beta \to \alpha\omega\beta$, where A is a nonterminal symbol and $\omega \neq \varepsilon$. (S denotes here the start symbol of the grammar.)

   Solution:

   In normalizing the grammar we have three steps:

   i) Remove the initial symbol S from the right side of rules.

   ii) Remove all terminal symbols from the left sides of the rules.

   iii) Fix all rules that are of a wrong form.

   The three steps are defined as follows:

   i) If S occurs in the right side of a rule, we add a new initial symbol S' and a rule $S' \to S$ to the grammar.

   ii) The terminal symbols are removed using the method presented in solution of Exercise 5.

   iii) Each incorrect rule
   $$X_1 \cdots X_n \to Y_1 \cdots Y_m \ ,$$
   where $m \geq n$ we add $n-1$ new nonterminals $(Z_1, \ldots, Z_{n-1})$ and the rule is replaced by the set of rules:

$$\begin{aligned} X_{n-1}X_n &\to Z_1X_n \\ Z_1X_n &\to Z_1Y_n \cdots Y_m \\ X_{n-2}Z_1 &\to Z_2Z_1 \\ Z_2Z_1 &\to Z_2Y_{n-1} \\ &\vdots \\ Z_{n-1}Z_{n-2} &\to Z_{n-1}Y_2 \\ Z_{n-1}Y_2 &\to Y_1Y_2 \end{aligned}$$

   For example, let us consider the rule

$$ABBA \to BAABA$$

As $n = 4$ we need three new nonterminals: Z_1, Z_2, and Z_3. The corresponding set of rules is:

$$\begin{aligned} BA &\to Z_1 A \\ Z_1 A &\to Z_1 BA \\ BZ_1 &\to Z_2 Z_1 \\ Z_2 Z_1 &\to Z_2 A \\ AZ_2 &\to Z_3 Z_2 \\ Z_3 Z_2 &\to Z_3 A \\ Z_3 A &\to BA \ . \end{aligned}$$

Now the derivation of the original rule becomes the following derivation:

$$\begin{aligned} AB\underline{BA} \Rightarrow AB\underline{Z_1 A} \Rightarrow A\underline{BZ_1}BA \Rightarrow A\underline{Z_2 Z_1}BA \Rightarrow \underline{AZ_2}ABA \\ \Rightarrow \underline{Z_3 Z_2}ABA \Rightarrow \underline{Z_3 A}ABA \Rightarrow BAABA \ . \end{aligned}$$