3.4.1

Note: in this exercise we use reductions from problems to problems. The only computational requirement for these reductions is that they have to be computable, i.e, recursive. This is in contrast to reductions we use later which have stricter computational requirements. The point of using recursive reductions is of course that the composition of two recursive computations is recursive. Thus, we are not allowed to "hide" non-recursive computation power into reductions.

As a preliminary result we have that the problem HALTING "Given a Turing machine M and its input x, does M halt on x?" is not recursive. That is, the language $H = \{M; x \mid M(x) \neq \nearrow\}$ is not recursive, i.e. there is no Turing machine that decides it.

(a)

The problem EMPTYHALTING is: Given a Turing machine M, does it halt on empty string ϵ?

The problem is not recursive since we can reduce the problem HALTING to it as follows (cf proof of (a) of Proposition 3.2 in the book). Given a Turing machine M and an input x for it, we construct the following machine M':

$$M'(y) := \textbf{if } y = \epsilon \textbf{ then } M(x) \textbf{ else halt.}$$

That is, M' first checks whether its input was empty and if this was the case, executes the machine M with input x; otherwise M' just halts. Clearly M' halts on empty string if and only if M halts on x. Thus, because HALTING is not recursive, neither is EMPTYHALTING (if EMPTYHALTING were recursive, we could use the (recursive) construction above to solve HALTING).

(d)

The problem is: Given a Turing machine M, does it ever write a symbol different from the symbol currently scanned?

The problem is recursive. First of all, note that the problem is about all inputs, i.e., is there an input on which M writes a symbol different from the one currently scanned? The recursive algorithm goes as follows. First, find all "bad" transitions of M of form $\delta(p, \sigma) = (r, \rho, d)$, where $\sigma \neq \rho$. Add a new state t in M and replace each "bad" transition $\delta(q, \sigma) = (r, \rho, d)$ with $\delta(q, \sigma) = (t, \sigma, -)$. The result is a variant of two-way finite automaton. Convert the states "yes", "no" and h into "normal" states in which the automaton, once entered, stays keeping the tape intact and cursor not moving. Mark t as accepting. Now the problem of whether there is an input for which M executes a "bad" transition is equal to finding out whether the constructed two-way finite automaton accepts any word (whether its language is empty). This problem is recursive (two-way finite automata can be converted into one-way finite automata and for one-way finite automata it is easy to check whether the language accepted is empty; see e.g. Hopcroft & Ullman: Introduction to Automata Theory, Languages and Computation) and therefore so is the original one.

(f)

The problem FINITE LANGUAGE is: Given a Turing machine M, is $L(M)$ finite?

The problem is not recursive (proof by reduction from the complement of HALTING). Given a Turing machine M and input x, construct the machine M' which discards its own input and then executes M on x. Clearly M has finite (empty) accepted language if and only if the original machine M does *not* halt on x. Therefore, if FINITE LANGUAGE were recursive, then the complement of HALTING would also be recursive implying that HALTING is recursive, too. But this is impossible.

3.4.6

The problem is: Show that there is an algorithm which, given a string x and the description of a Turing machine M accepting the language $\{x; y \mid (x, y) \in R\}$, where R is a relation, constructs the description of a Turing machine M_x accepting the language $\{y \mid (x, y) \in R\}$.

The algorithm goes as follows. First, M_x starts with $|x| + 1$ sequential copies of the unit shifter (Example 2.1 in the book). These transform the input $\underline{\triangleright} y$ to $\triangleright\underbrace{\underline{\sqcup}\ldots\sqcup}_{|x|\text{ times}}\underline{\sqcup} y$. The second part of M_x moves the cursor to the beginning of the tape and writes x; (which, for M_x, is a fixed constant) on the tape and returns to the beginning of the tape, resulting in $\underline{\triangleright} x; y$. The last part of M_x is the copy of M. Clearly M_x accepts the language $\{y \mid (x, y) \in R\}$ because (i) the first and second parts of M_x always terminate and (ii) the third part of M_x (the machine M) accepts the language $\{x; y \mid (x, y) \in R\}$. Furthermore, the construction of M_x, given x and M, can be clearly done algorithmically i.e. by a Turing machine.