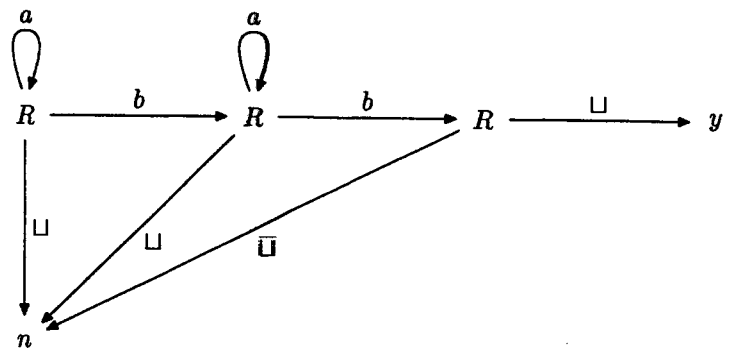
CS4124

Answers to Selected Problems of HW #7

Problem 4.2.3



Problem 4.3.5

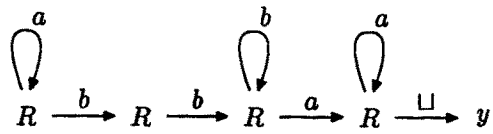
The abbreviated notation for Turing machines becomes unpleasantly convoluted with more than two tapes, so we describe the construction, in sufficiently precise terms that it should be clear how to write down the actual tuples for such a machine. The machine we give will destroy its inputs; a machine that wished to preserve them could first call the copying Turing machine, so that the multiplication TM works with copies.

The machine first writes down the number 0 on its third tape. This number will be the running sum of the multiplication. It then executes the following loop so long as the first number remains nonzero:

1. If the rightmost digit of the first number is 1, write the second number on the second tape. Otherwise, write 0 on the second tape.
2. Follow the number just written with a ";".
3. Copy the number from the third tape to immediately after the ";".
4. Erase the number from the third tape.
5. Call the addition TM to add the two numbers on the second tape and store the result on the third tape.
6. Erase the (garbage) contents of the second tape.
7. Erase the last digit of the first number on the first tape, overwriting it with a blank.
8. Append a 0 to the right end of second number on the first tape.
9. If the first number is zero, terminate. Otherwise, go back to the top of the loop.

This algorithm emulates the traditional pencil-and-paper method of multiplication, in which a partial product is computed for each digit of one multiplicand, and these partial products, appropriate left-shifted with zeroes, are added together.

Problem 4.5.1

(a) 

(b) This algorithm is sufficiently complicated that it makes more sense in words than in the abbreviated notation.

Begin by moving right repeatedly; at any point in the string, stop and invoke the right-shifting machine to insert a blank in the middle of the string. Now return to the starting blank at the left end of the tape. Repeat the following loop:

1. Move right one square.
2. If the square is blank, break from the loop. If the square is not blank, erase it, and remember the symbol σ in the state.
3. Move right to the next blank.
4. Move left one square.
5. If the symbol in this square is σ, erase it, and move left to the next blank. If not, go into an infinite loop.
6. Go back to the top of the loop.

If this loop exits, move right to the next non-blank square, move left one square from there, and repeat the loop. If the loop exits a second time, halt and accept.

In essence, the loop looks for a string followed by its reversal; the overall algorithm first splits the string, then checks that each half of the split consists of a string followed by its reversal.

Problem 4.5.2

There are 17 possible computations, of which 2 halt and 4 take exactly 5 steps.

When started in this configuration, M moves to the right, writing some number of as, moving right after each one. It then either loops forever or else writes another a, moves right, and halts.

For this machine, $r = 2$, since there are two different circumstances (state q_0, scanning a $\sqcup$, or state q_1, scanning an a) under which 2 quadruples are applicable, and there are no circumstances under which three or more quadruples are applicable.

Problem 4.5.3

- union

  Suuppose we have Turing machines M_1 and M_2. Let M' be the nondeterministic Turing machine which, on input w, non-deterministically chooses to simulate the action of either M_1 or M_2 on w, accepting if the chosen machine accepts. Then $L(M') = L(M_1) \cup L(M_2)$. For if $w \in L(M_1) \cup L(M_2)$ then one of M_1 or M_2 must accept w and M', by choosing that machine and seeing that accepting computation, also accepts, so that $w \in L(M')$. On the other hand, if $w \in L(M')$, then there must be an accepting computation of M' on w, a computation which consists of a choice of M_1 or M_2 and then an accepting computation of that machine. In either case, one of M_1, M_2 accepts w, so $w \in L(M_1) \cup L(M_2)$.

- concatenation

  Let M'' be the nondeterministic two-tape Turing machine which, on input w, nondetermnistically splits w into two strings x and y with $xy = w$, then copies y to its second tape (leaving x on the first tape). M'' then runs M_1 on its first tape; if M_1 accepts, M'' runs M_2 on its second tape. Then $L(M'') = L(M_1)L(M_2)$. For suppose $w \in L(M_1)L(M_2)$. Then $w = xy$, where $x \in L(M_1)$ and $y \in L(M_2)$. In this case, M'' can choose the split $w = xy$, which will allow it to see M_1 accept x and M_2 accept y, so that M'' will accept w in this computation, and thus $w \in L(M'')$. On the other hand, if $w \in L(M'')$, then there is some computation of M'' that accepts. In this computation, M'' splits w into x and y, then watches M_1 accept x and M_2 accept y. From these facts, though, we know that $w = xy \in L(M_1)L(M_2)$.

- Kleene star

  Let M^* be the nondeterministic two-tape Turing machine, which, on input w, nondeterministically splits w into strings $w_1, w_2, \ldots, w_k$. M^* then copies each string to its second tape and runs M_1 on just that string. If M_1 accepts each and every w_i, then M^* accepts w. Then $L(M^*) = L(M_1)^*$. For suppose $w \in L(M)^*$. Then $w = w_1 w_2 \ldots w_k$ for some $k \geq 0$, where each $w_i \in L(M)$. If M^* chooses the decomposition $w = w_1 w_2 \ldots w_k$, then it will see M accept each of them, so that it itself will accept w, making $w \in L(M^*)$. On the other hand, if $w \in L(M^*)$, then M^* must have some accepting computation on w. In such a computation, M^* splits w into $w_1 w_2 \ldots w_k$, and sees M accept each, so that $w = w_1 w_2 \ldots w_k$, where each $w_i \in L(M)$. By definition, then, $w \in L(M)^*$.

The proofs for the closure properties of the recursively enumerable languages are identical, except that we write "halt" instead of "accept."