# Selected solutions week 8 - IN2080

The following are solutions to some of the weekly tasks which we discuss in the group sessions, which are not present in the book.

## 3.6

The reason is that $M$ might run forever on $s_i$ if it is not in $L(M)$. If it does, our enumerator will never print all strings in $L(M)$. By limiting the number of steps, we ensure the machine never loops forever. However, since we retry running the machine for on $s_j$ in step $i$, for some $j < i$, we also ensure that if the machine does accept $s_j$, it gets printed at some point, as any accepting computation is done in a finite number of steps.

## 3.7

Step 1. requires the machine to collect all possible settings over the variables. Collecting all possible settings for variables over an infinite set yields a set of infinitely many settings. It is therefore not possible to store the settings, as the machine suggests. Step 2. evaluates the polynomial on all the settings. Doing this is impossible, as there are infinitely many settings. A proper machine that follows the same idea, i.e. using brute-force to try to find a setting evaluating the polynomial to 0, would look something more like the following.

First we fix some ordering of the possible combinations of $k$ integers. One way might be to consider the string $x_1 \dots x_k$ a $k$-digit number and do all such numbers in order. The first number would be all 0s. Then:

$M_{good}$ = "On input $<p>$, a polynomial over variables $x_1, ..., x_k$ :

1. Initialize all the variables to 0.
2. Evaluate $p$ with all possible combinations of positive and negative signs for the digits.
3. If any of the results were 0, *accept*. Otherwise, set the current setting to be the next in the ordering, and go to step 2.

It is important to note that the new machine only recognizes the language, not decides it.

## 3.8

### b.

$M_b$ = "On input $w$, where $w$ is a string over $\{0, 1\}$:

1. If $w = \epsilon$, *accept*.
2. Scan the input, and cross off the first two 0s and the first 1, skipping any crossed off symbols. If this was not possible, *reject*.
3. If all symbols are crossed off, *accept*. Otherwise, repeat from step 2."

### c.

$M_c$ = "On input $w$, where $w$ is a string over $\{0, 1\}$:

1. If $w = \epsilon$, *accept*.
2. Scan the input, and cross off the first two 0s and the first 1, skipping any crossed off symbols. If this was not possible, *accept*.
3. If all symbols are crossed off, *reject*. Otherwise, repeat from step 2."

This is the same machine as from the previous exercise, only that the accept and reject states are flipped. We can write this machine as:

$M_c$ = "On input $w$, where $w$ is a string over $\{0, 1\}$:

1. Run $M_b$ on $w$. *Accept* if $M_b$ *rejects*, and vice versa."

## 3.15

Let $M_A$ and $M_B$ recognize the languages $A$ and $B$ respectively.

### b.

$M_{AB}$ = "On input $w$:

1. Non-deterministically guess a split of $w$ into $w_{first}$ and $w_{second}$.
2. Run $M_A$ on $w_{first}$ and $M_B$ on $w_{second}$.
3. *Accept* if both accepts, and *reject* if either rejects."

By theorem 3.16, this machine is a recognizer for the concatenation of $A$ and $B$.

### c.

$M_{A^*}$ = "On input $w$:

1. Non-deterministically guess a split of $w$ into $w_0$ ... $w_n$, where $n \leq |w|$.
2. Run $M_A$ on each $w_i$, and accept if they all accept."

By theorem 3.16, this machine is a recognizer for $A^*$.

### d.

$M_{A \cap B}$ = "On input $w$:

1. Run $M_A$ and $M_B$ on $w$, and accept if both machines does."

## 3.16

Let $M_A$ and $M_B$ decide the languages $A$ and $B$ respectively. As these machines are deciders, they either accept or reject any input. As such, since we essentially just simulate machines in the previous exercise, we can just reuse the previous machines, and they will be deciders, given that we're simulating deciders.

For the sake of thoroughness though, deterministic TMs are given below for these languages:

### b.

$M_{AB}$ = "On input $w$:

1. Let $w_{first} = w$ and $w_{second} = \epsilon$.
2. Run $M_A$ on $w_{first}$, and $M_B$ on $w_{second}$. If both machines accept, *accept*. If $w_{first} = \epsilon$, *reject*.
3. Given that $w_{first} = st$, and $w_{second} = u$, let $w_{first} = s$ and $w_{second} = tu$, where $t \neq \epsilon$ and $t \in \Sigma$ (the input alphabet), and repeat from step 2.

As both $M_A$ and $M_B$ are deciders, $M_{AB}$ is also obviously one too, as it simply tries to run the two former machines on some inputs.

### c.

$M_{A^*}$ = "On input $w$:

1. Let $i = 1$.
2. For each set of strings $\{s_1, s_2...s_i\}$, such that $s_1 s_2...s_i = w$, run $M_A$ on each $s_1, s_2...s_i$. If $M_A$ accepts all $s_1, s_2...s_i$, *accept*.
3. Let $i = i + 1$. If $i > |w|$ *reject*. Otherwise, repeat from step 2."

As $M_A$ is a decider, and for a given $i$, there are finitely many sets of $i$ strings which can be concatenated into $w$, $M_{A^*}$ is also a decider.

## d.

$M_{\overline{A}}$ = "On input $w$:

1. Run $M_A$ on w, and *accept* if it rejects, and vice versa."

As $M_A$ is a decider, $M_{\overline{A}}$ is also clearly one, as it simply simulates the former, and negates the answer.

It is important to note that the class of Turing-recognizable languages are **not** closed under complement, only the class of decidable ones.

## e.

This will be exactly the same as 3.15d. Since both machines are deciders, the new one will also be one.