Selected solutions week 9 - 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.

4.2

We need to create a machine for the language

$EQ_{DFA-RX} = \{< M, R > \mid M$ is a DFA, R is a regular expression, and $L(M) = L(R)\}$

There are several ways of doing this. By far the easiest, is to simply use the machine C deciding EQ_{DFA} after transforming the regular expression to a DFA:

$T =$ "On input $< M, R >$, where M is a DFA, and R a regular expression:

1. Create R', an NFA equivalent with R, as described in theorem 1.54.
2. Create R'', a DFA equivalent with R', as described in theorem 1.39.
3. Simulate C on M and R and output it's result."

This machine is clearly a decider for EQ_{DFA}, as both the transformation algorithms and C are decidable.

4.3

Let the following TM decide the language ALL_{DFA}:

$S =$ "On input $< A >$, where A is a DFA:

1. Construct A', a DFA recognizing $\overline{L(A)}$ (we have previously solved an exercise where we show that this is possible).
2. Run T, the machine deciding E_{DFA}, on A'. If it accepts, *accept*; otherwise *reject*."

E_{DFA} is the language of DFAs whose languages are empty, which is shown to be decidable in chapter 4.

4.4

A tempting idea is to make a PDA equivalent to the input CFG, and run it on ϵ. The problem is that PDAs are non-deterministic, so we'd need to do more. We'll rather take some inspiration from theorem 4.8.

$K =$ "On input $< G >$, where G is a CFG:

1. Mark all occurences of ϵ in the rules.
2. Repeat until no new variables get marked:
3. Mark all variables A, such that the rule $A \rightarrow U_1U_2...U_k$ exists, where each U_i are marked.
4. If the start variable is marked *accept*; otherwise, *reject*."

We begin by marking all ϵ occurences. Afterwards, we mark any variable that can yield a sequence of marked symbols. By limiting the marking of variables to where all the symbols on the right-hand side of a rule are marked, we ensure that ϵ can be produced by the right-hand side. If the start variable is marked, we know the grammar derives ϵ.

4.13

The idea of the following TM is to first check if the CFG contains any loops containing terminals other than ϵ, which also can terminate. If it does, we know it can produce an infinite number of words, so we simply check if $k = \infty$. If not, the language of the grammar must be finite. Then we try all derivations, skipping loops (as they do not produce any terminals), and check if we have derived k unique words.

Let $LOOP$ be a TM which decides if a CFG has a loop containing some non-empty terminals (try making this decider for yourself, think graph algorithms).

Let the following TM decide the language C_{CFG}:

$TM_C =$ "On input $< G, k >$, where G is a CFG and $k \geq 0$ or $k = \infty$:

1. Run $LOOP$ with G as input. If it accepts, and $k = \infty$, *accept*. If $k = n$, for some $n \in \mathcal{N}$, *reject*.
2. Let E = $\{e \mid S \overset{*}{\Rightarrow} e$, such that $U_1 \dots U_i \dots U_m \overset{*}{\Rightarrow} X_1 \dots U_i \dots X_n$ does not occur in $S \overset{*}{\Rightarrow} e\}$, where each U, X is a terminal or a variable except U_i, which is a variable, and S is the start variable of G.
3. If $|E| = k$, *accept*. Otherwise, *reject*."

4.29

We go ahead with a brain-dead solution; a set is a subset of another, iff the intersection between the two is equal to the first. The machine will first convert the regular expressions into DFAs, then construct the DFA computing the intersection of the two other DFAs, and finally check if the language of the intersection is equivalent to the first language using the decider for EQ_{DFA}.

$TM_A =$ "On input $< R, S >$, where R, S are regular expressions:

1. Let R', S' be DFAs equivalent to R, S respectively (theorem 1.39 and 1.54).
2. Let Q be the DFA computing $L(R) \cap L(S)$.
3. Run F, the machine deciding EQ_{DFA}, on Q and R', and *accept/reject* accordingly.