School of Computer Science, University of Nottingham

G52MAL Machines and their Languages, Spring 2012

Thorsten Altenkirch

Exercises, Set 2

Friday 17th February 2012

Deadline: Wednesday 29nd February 2012 in your tutorial

1. Let $\Sigma = \{0, 1\}$. Consider the language $L \subseteq \Sigma^*$ of binary numbers which are divisble by 5. We allow leading 0s and interpret the empty word as 0. For example:

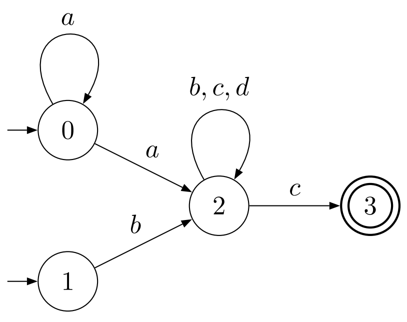
$\epsilon \in L$	0 is divisible by 5
$01 \notin L$	1 is not divisible by 5
$101 \in L$	$101 = 5$ is divisible by 5
$111 \notin L$	$111 = 7$ is not divisible by 5
$11001 \in L$	$11001 = 15$ is divisible by 5

   (a) Construct a DFA D which accepts L by drawing a transition diagram.

   (b) Explain how you have derived your construction, i.e. explain the meaning of your states.

   (c) (*) Construct another DFA D^R which accepts the L^R, the reverse of L. That is the set of all words whose reverse is in L. E.g. $10011 \in L^R$ but $11001 \notin L^R$.

2. Consider the following NFA N over $\Sigma = \{a, b, c, d\}$:



   (a) Give the transition table for N.

   (b) Explicitly calculate $\hat{\delta}\,(\{0, 1\}, acb)$. Clearly show each step of the calculation.

   (c) State which of the following words are accepted by N:

      i. ε
      ii. a
      iii. bc
      iv. dac
      v. $bbcc$
      vi. $acdac$
      vii. $aadbdc$

   (d) Describe the language accepted by N in plain English.

3. Construct a DFA $D\ (N)$ from the NFA N from the previous question by applying the *lazy subset construction.*

   That is only construct the reachable states of the subset construction applied to N. Present the resulting DFA by the transition diagram. Label the states with the subsets of the states of N they correspond to.

4. **Bonus Exercise**

   Consider the following Haskell encodings of DFAs and NFAs:

   ```
   data DFA q σ = DFA (q → σ → q)   q   [q]
   data NFA q σ = NFA (q → σ → [q]) [q] [q]
   ```

   The two type parameters are the set of states and the set of symbols. The three fields are the transition function, the start state(s), and the final states.

   For example, the NFA N_1 from Lecture 4 could be defined as follows:

   ```
   data Q = Q0 | Q1 | Q2 deriving (Eq, Enum, Bounded)
   data Σ = A | B

   n1 :: NFA Q Σ
   n1 = NFA δ [Q0] [Q2]
     where
       δ Q0 A = [Q0]
       δ Q0 B = [Q0, Q1]
       δ Q1 _ = [Q2]
       δ Q2 _ = []
   ```

   (a) Define a function that determines if a word is accepted by a DFA. It should have the following type:

   ```
   runDFA :: Eq q ⇒ DFA q σ → [σ] → Bool
   ```

   (b) Define a function that determines if a word is accepted by an NFA. It should have the following type:

   ```
   runNFA :: Eq q ⇒ NFA q σ → [σ] → Bool
   ```

   (c) Define a function that converts a DFA to an NFA. It should have the following type:

   ```
   dfa2nfa :: DFA q σ → NFA q σ
   ```

   (d) Define a function that converts an NFA to a DFA. It should have the following type:

   ```
   nfa2dfa :: (Eq q, Enum q, Bounded q) ⇒ NFA q σ → DFA [q] σ
   ```

   Hint: You may find the following functions helpful:

   ```
   import Data.List (intersect, union)

   powerset :: [a] → [[a]]
   powerset []       = [[]]
   powerset (a : as) = powerset as ++ map (a :) (powerset as)

   enumerate :: (Enum a, Bounded a) ⇒ [a]
   enumerate = [minBound .. maxBound]

   unions :: Eq a ⇒ [[a]] → [a]
   unions = foldl union []
   ```