The University of Melbourne
Department of Computing and Information Systems
COMP30026 Models of Computation

Assignment 2, 2016

Released: 25 September. Deadline: 16 October at 23:00

Purpose

To improve and consolidate your understanding of finite-state automata and Turing machines.

DFAs and two-way Turing machines

This assignment involves DFAs and Turing machines. The kind of Turing machine we consider is the *two-way Turing machine* whose tape extends indefinitely in both directions. Input to such a machine is placed somewhere on the tape, and the tape head is positioned over the blank cell *immediately to the left* of the input. Then the machine is ready to perform its computation. When it terminates (if it terminates), the machine's output is sitting on the tape. We will use the convention that upon termination, the tape head should be over the blank cell *immediately to the left* of the output.

We assume that, for both DFAs and Turing machines, input can be any combination of digits and lowercase letters. The Turing machine's tape alphabet can include other symbols as needed.

Nine challenges

There are 9 small design/programming tasks. Make sure you carefully read the instructions (at the end of this document and on the LMS) on submission formats and how to submit.

1. Let alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ and let L_1 be the set of strings that start with `ab` and end in `ba`, that is, $L_1 = \mathtt{ab}\Sigma^* \cap \Sigma^*\mathtt{ba}$. Note that $\mathtt{aba} \in L_1$. Construct a DFA d that recognises L_1.

2. Again let $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ and let L_2 be the set of strings in which every substring of length four has exactly one `a`. That is,

$$L_2 = \{xyz \mid x, y, z \in \Sigma^* \text{ and } (|y| = 4 \Rightarrow y \text{ contains exactly one } \mathtt{a})\}$$

   For example, `abbb`, `cbabc`, and `babbbabb` are all in L_2, as is every string of length 3 or less. However, `abbbb` and `cbababc` are not in L_2. Construct a DFA e that recognises L_2.

3. Construct a two-way Turing machine t (with input alphabet $\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$) which decides the language $L_3 = \{\mathtt{a}^i\mathtt{bc}^i \mid i \geq 0\}$. The machine should communicate its decision by

leaving a '`Y`' on its tape, if input was indeed in L_3. Otherwise it should leave an '`N`'. That is, t should implement the function f defined by

$$f(w) = \begin{cases} \texttt{Y} & \text{if } w \text{ is of the form } \texttt{a}^i\texttt{bc}^i \text{ for non-negative integer } i \\ \texttt{N} & \text{otherwise} \end{cases}$$

Your Turing machine should always halt, and upon termination, the tape head should be immediately to the left of output.

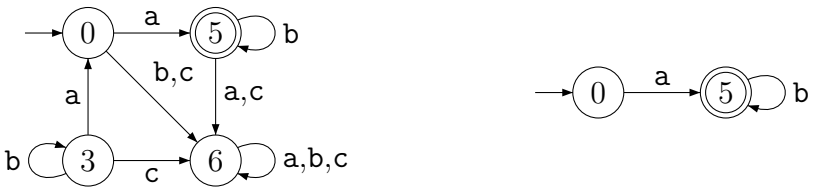
4. Construct a two-way Turing machine u (with input alphabet $\{\texttt{a}, \texttt{b}, \texttt{c}\}$) which decides the non-context-free language $L_4 = \{\texttt{a}^i\texttt{b}^j\texttt{c}^k \mid i, j, k > 0 \wedge i + j = k\}$. The machine should communicate its decision by leaving a '`Y`' on its tape, if input was indeed in L_4. Otherwise it should leave an '`N`'. That is, u should implement the function g defined by

$$g(w) = \begin{cases} \texttt{Y} & \text{if } w \text{ is of the form } \texttt{a}^i\texttt{b}^j\texttt{c}^{i+j} \text{ for positive integers } i \text{ and } j \\ \texttt{N} & \text{otherwise} \end{cases}$$

Again, the machine should always halt, and upon termination, the tape head should be immediately to the left of output.

5. A state q in a DFA is useful only if q is both *reachable* and *generating*. By "reachable" we mean that some input will take the DFA to state q, or in other words, if there is some path from the start state to q. By "generating" we mean that some (remaining) input takes the DFA from q to an accept state, or in other words, there is some path from q to some accept state. A state is *useless* unless it is both reachable and generating. By a *trimmed* DFA we mean a DFA that has had all useless states removed, together with all transitions involving useless states (except we never remove a start state).

   Many of the DFA diagrams on our lecture slides have been trimmed. Below is a DFA for $\texttt{ab}^*$ (on the left) and its trimmed version (on the right).



   An attached file `DFA.hs` (on the LMS) defines a Haskell type for DFAs. The file `RunDFA.hs` contains an emulator that can run DFAs provided as Haskell expressions of type `DFA`.

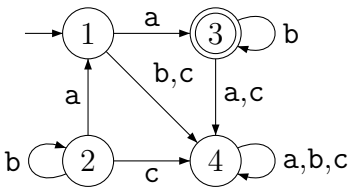
   The file `ManipulateDFA.hs` contains a number of stubs for Haskell functions, and also a working function `reachable` that can find a DFA's set of reachable states. Your job is to complete `ManipulateDFA.hs` as specified in the four remaining tasks.

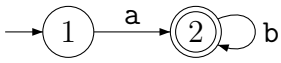
   First, write a Haskell function `trim :: DFA -> DFA` which takes a DFA and returns its trimmed version.

6. Most DFA-manipulating algorithms assume input DFAs to be *complete*. That is, for each state, and each symbol in the DFA's alphabet, there is an explicit transition. The DFA on the left, above, is complete (assuming its alphabet is $\{\texttt{a}, \texttt{b}, \texttt{c}\}$).

Write a Haskell function `complete :: DFA -> DFA` which takes a DFA and returns its completed version. It should work correctly for any input; in particular it should not assume that the input DFA is trimmed or normalised (see the next task).

7. Some DFA-manipulating algorithms are easier to implement if their input DFAs are *normalised*. By normalised we mean that an n-state DFA has its states numbered from 1 through n. If we normalise the two DFAs from above, we may get





The chosen renamings are natural, but any permutation of the states' names would be acceptable, and in particular, the start state does not have to become state 1.

Write a Haskell function `normalise :: DFA -> DFA` which takes a DFA and returns a normalised version. It should work correctly for any input; in particular it should not assume that the input DFA is trimmed, nor that it is complete.

8. Write a Haskell function `complement :: DFA -> DFA` which takes a DFA d and returns a complete DFA d' such that d and d' recognise complementary languages. The function should work correctly for any input; in particular it should not assume that the input DFA is trimmed, or normalised, or complete (but it may of course utilise functions that achieve one or more of these properties—indeed it may be an advantage to do so).

9. Write a Haskell function `prod :: DFA -> DFA -> DFA` which takes two DFAs d_1 and d_2 and returns a complete DFA d such that d accepts a string w iff both d_1 and d_2 accept w. The function should work correctly for any input; in particular it should not assume that the input DFAs are trimmed, or normalised, or complete.

Emulators and starter kit

You will find a DFA emulator (`RunDFA.hs`) attached. It is a slight extension of the one we have used in a lab. This one assumes that a DFA comes with its alphabet provided explicitly (the one we used before assumed you could just get that from the DFA's transition function). I have also added a `trace` utility that may be useful for debugging DFAs.

The file `DFA.hs` defines the Haskell type `DFA` and has some utility functions that can help check well-formedness of DFAs.

The file `RunTM.hs` has a Turing machine emulator for two-way Turing machines. It also has a `trace` function that may be useful for debugging Turing machines. Read the scripts for more detail on how to use the emulators. The scripts also contain example machines, in the required formats.

As already mentioned, `ManipulateDFA.hs` contains a number of stubs for the Haskell functions you are asked to complete, and a working function `reachable`.

Submission formats

For the first four questions, please use the required formats, and use the two emulators to test your solutions. The files you submit should be named '`D.hs`' (for Question 1), '`E.hs`' (for Question 2), '`T.hs`' (for Question 3), '`U.hs`' (for Question 4), and '`ManipulateDFA.hs`' (for Questions 5–9). There is no need to submit `DFA.hs`, although you can if you want. For the first four, the first few lines of Haskell code should be:

```
module D
where

import RunDFA

d :: DFA
```

```
module E
where

import RunDFA

e :: DFA
```

```
module T
where

import RunTM

t :: TM
```

```
module U
where

import RunTM

u :: TM
```

Marking will be semi-automated, so following these instructions is critical. Note that case matters: Use upper case `D`, `E`, `T`, and `U` for both file and module names, and use lower case letters `d`, `e`, `t`, and `u` for the DFAs and Turing machines.

Submission and assessment

Your solution will count for 10 out of 100 for the subject. Each question is worth one mark. The machines d, e, t and u are basically marked on correctness, that is, you will receive full marks for a working machine unless it uses more than, say, three times the number of states that a "reasonable" correct machine has. Similarly the Haskell functions are primarily marked on correctness. The remaining mark is allocated for tidiness and readability of comments, that is, explanations in English. We don't expect virtuoso Haskell code, but a bonus mark is available for nice Haskell code (so 11 marks out of 10 could be possible).

The deadline is 16 October at 23:00. Late submission will be possible, but a late submission penalty will apply: a flagfall of 2 marks, and then 1 mark per 24 hours late.

The five files mentioned above should be submitted to one of the Engineering student servers; instructions on using `submit` for this will be posted on the LMS. (You are encouraged to test submit your solutions well before the deadline, so that you don't run into machine-dependent surprises in the last hour.) The submit command is '`submit COMP30026 2`'. You can submit as often as you want; each new submission overwrites your previous one. At the Unix prompt, you can simply type '`submit COMP30026 2`', and '`submit`' will then ask you for file names. Alternatively you can include the files in the command:

```
submit COMP30026 2 D.hs E.hs T.hs U.hs ManipulateDFA.hs
```

Do not use commas between the file names, just a space. Use '`verify COMP30026 2`' to check that your submission was received. More detail on how to access the Engineering machines will be made available on the LMS before Week 10.

Individual work is expected, but if you get stuck, email Carter or Harald a precise description of the problem, bring up the problem at a lecture, or (our preferred option) use the LMS discussion board. The COMP30026 LMS discussion forum is both useful and appropriate for this; soliciting help from sources other than the above will be considered cheating and will lead to disciplinary action.

Harald Søndergaard
24 September 2016