School of Computing and Information Systems
COMP90045 Programming Language Implementation Tutorial Week 2

11 August 2020

Plan

Some of the following exercises are about writing input to `alex`. They can be solved and discussed in our virtual class, but you really should test your solutions. We have repeated Exercise 4 from last week, as you probably skipped that.

We have also included, at the end, some hints for how you log in to MSE student servers to run Haskell and allied tools under unix. If you have the Haskell platform installed on your machine then, right now, you won't need those student machines at all. However, it is likely that we will use them as submission machines for the project. So you may still want to try out running Haskell there.

The exercises

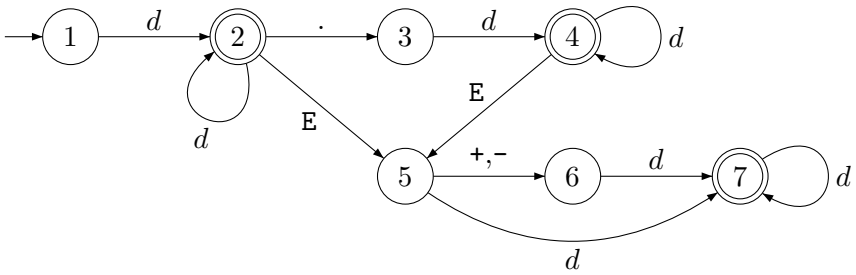
4. Write an alex script `A.x` to count the number of times each of the articles "a", "an", or "the" appear in given input. Only count the longest matching character sequences (so that "lantana" is considered to have two occurrences of "an", and only one occurrence of "a").

   Some hints: Include the "`%wrapper "basic"`' directive. If you make the actions have type `String -> String` then you can make the list of "tokens" generated be the list of articles found in the text. The rest is then a matter of counting these. If you include a `main` function then you can compile the generated scanner and test it more easily. Test the result on a file `my.txt`, for example as follows:

   ```
   alex A.x
   ghc A.hs
   A < my.txt
   ```

   (If the last command does not work, try `./A my.txt`.)

6. Consider this recogniser for unsigned number literals:



   where d is `[0-9]` (that is, any digit). Write regular definitions in `alex` notation to capture the set of strings that are recognised.

7. Question 3(d) asked for a regular expression that can capture the set of valid strings-with-comments. Check out the following `alex` solution (the source is on the LMS with this sheet):

```
{
module Main (main) where
}
%wrapper "basic"     -- Actions will be of type String -> String

$normalc = [^\" \/ \*]
$ndbquot = [^\"]
@string  = \" $ndbquot* \"
@middle  = \/ | (\*)* ($normalc | @string)
@comment = \/ \* @middle* \* \/

rules :-

  @comment      { \s -> "<<<" ++ s ++ ">>>" }
  .             { id }
  \n            ;

{
main
  = do
      s <- getContents
      putStrLn (concat (alexScanTokens s))
}
```

The first rule says to print anything recognised as a comment in three pairs of angle brackets; the second says to just echo everything else (except a newline character—that is just ignored). Run the resulting scanner on this input: `A "comment /* buried */ inside" a string`. Discuss the result; is it surprising?

8. Write an alex script to generate a scanner that can turn a sequence of simple C assignments into a sequence of tokens, as outlined in a lecture. The scanner should be able to handle assignments like

```
degc = (degf - 32) * (5.0/9);
```

(To test the scanner, you may want to use actions that print the token name, rather than return it.)

9. Construct DFAs for the following regular expressions (you don't have to follow any particular algorithm). Show the sequences of moves made by each DFA in processing the input string `ababbab`.

(a)	$(\mathtt{a} \mid \mathtt{b})^*$	(b)	$(\mathtt{a}^*\mathtt{b}^*)^*$	(c)	$((\epsilon \mid \mathtt{a})\mathtt{b}^*)^*$
(d)	$(\mathtt{a} \mid \mathtt{b})^*\mathtt{abb}(\mathtt{a} \mid \mathtt{b})^*$	(e)	$(\mathtt{ab} \mid \mathtt{b})^+(\epsilon \mid \mathtt{a})$	(f)	$(\mathtt{ab})^*(\epsilon \mid \mathtt{b})$

10. There are some important tricks for the construction of certain DFAs. This exercise is about one such trick. Next week we will use another useful trick.

    (a) Draw a DFA A for the set of number-strings (subsets of `[0-9]+`) that have value 42 (leading zeros allowed). We designed a regular expression for this set in Exercise 3(a).

    (b) Assuming alphabet $\Sigma = \{0, 1, \ldots, 9\}$, draw a DFA B which recognises the complement of A's language. That is, B should recognise $\Sigma^* \setminus L(A)$.

    (c) Given a DFA $(Q, \Sigma, \delta, q_0, F)$ for some language L, what is the general method for obtaining a DFA for $\Sigma^* \setminus L$, that is, a recogniser for L's complement?

    (d) Show, with an example, that the same trick does not necessarily work when we deal with NFAs.

Running alex and happy

The Haskell platform comes with `alex` and `happy` (later exercises will use the parser generator `happy`). If you have a Mac and run Unix via 'terminal', you should make sure that you have downloaded and installed Apple's command-line tools for Xcode.

The following assumes a basic familiarity with Unix.

Ultimately, you will need to submit assignment solutions on the MSE student servers, dimefox2.eng.unimelb.edu.au or nutmeg2.eng.unimelb.edu.au. You are encouraged to use any early exercises that use Haskell, `alex` or `happy` as an opportunity to test that you are able to get things working on the MSE student servers. If you have problems, you will want to have those sorted out well before the assignment submissions.

Remote login to MSE student machines

The Virtual Private Network (VPN) service allows you to access the campus network from outside the university. The AnyConnect student VPN is apparently (comparatively) simple to install, see `http://studentit.unimelb.edu.au/findconnect/vpn`.

Remote login from a PC. You can install puTTY ('putty') on your own machine, and 'scp'. Both are available at `http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html`.

Now use puTTY to remote login to one of the student servers. Use "All programs > Network Apps > PuTTY > PuTTY" on the lab machines, or the location that you placed it on you home computer (perhaps the Desktop), to start PuTTY running. On your home computer, it may be necessary for you to accept to execute a program that has been downloaded from the internet. When a dialog box opens, type nutmeg2.eng.unimelb.edu.au as the "destination you want to connect to" and check that 'Port' is set to 22. Then click "Open" and wait for a connection to be established; click "Yes" to accept the server's host key if you get asked. If nutmeg2 is unavailable, try dimefox2—the two are equivalent and use a shared file system. On the login prompt, use your University credentials.

Remote login from a Mac or Linux machine. From a terminal window, use ssh to connect to the server:

```
ssh myusername@nutmeg2.eng.unimelb.edu.au
```

and then once you have typed your University password (and typed "yes" if it asks you to accept the encryption key) everything you type in that window is being executed on the server. To copy files across securely, use `scp`. To exit the server, type "exit".