Homework 4: Lectures 7 & 8

CS 440: Programming Languages and Translators, Spring 2020

Due Fri Feb 14, 11:59 pm

What to submit

Submit the Haskell programs in a *.hs file; the solutions to the written problems can go in a separate file (pdf from word processor is popular, as is pdf or jpg of scan of handwritten file). Please name the files something like `Doe_John_440_h4.hs` or `pdf`, put them into a folder named something like `Doe_John_440_h4`, zip the folder, and submit the zip file.

Remember the new requirements: If you work alone, please say so in your submission. If you work in a group but aren't the person submitting the solution, then create a short file with the names of everyone in your group (including yourself), and submit that to Blackboard (in the homework 4 folder). These new requirements will make it easier for us to detect if someone forgot to put names down on the submission or didn't do the homework.

Problems [50 points]

Lecture 7: Regular Expressions, part 2

1. [3 points] Write a regular expression for the language over `a`'s and `b`'s where the strings have an even number of `a`'s or an even number of `b`'s.
2. [6 points] Write a regular expression for comments of the form `/*` .... `*/`.
   (1) The comment begins with `/*` and end with the first `*/` encountered.
   (2) Optional whitespace can precede the comment or follow the comment, but nothing else. Whitespace consists of spaces and or tabs (`\t`). Don't include newline (`\n`).
   (3) The text of the comments can include letters or digits, space, tab, underscore, `*`, and `/`.
   (4) The comment can include `/*`, but we don't nest comments.
   (5) The star of the beginning `/*` cannot be the star of the ending `*/`.
   (6) The symbol before the ending */ doesn't have to be whitespace.
   (7) Inside a regular expression, if you want a star symbol, escape it as `\*`, since just `*` means Kleene star. (So within the expression, write `/\*` to mean a slash followed by a star. As an expression, just `/*` means zero or more slashes

   Some strings that should fail to match:

`/* xyz */ */`	By rules 1 and 2
`/* /* xyz */ */`	By rules 1, 2, and 4
`/*/`	By rule 5

Some strings that should match:

`/**/`	By rules 3, 5, and 6
`/*///***/`	By rules 3, 5, and 6
`    /* * * Typical * * */`	By rules 2 and 3

Programming Assignment (Lecture 7)

3. [8 points] Take the regular expression structure and match routine from Lecture 7 and give definitions for `match (RE_star …)` … and `match (RE_in_set …)` …. You may treat *rexp** as a derived expression (built up from already-existing parts): *rexp** is equivalent to (*rexpr rexpr** | *empty*). For `[`*symbols*`]`, a match occurs if the head of the input is in the list of symbols.. Don't treat $[x_1 x_2 \ldots x_n]$ as equivalent to $(x_1 \mid x_2 \mid \ldots \mid x_n)$.

   For this problem, you can submit just the code for the two cases (you don't have to include the `RegExpr` data structure or the other clauses of `match`). You don't have to put them in a `*.hs` file, but you can if you want to.

4. [16 points] The attached file `HW_04_440_skeleton.hs` contains part of a definition of a function `capture`. The `capture` function should behave like `match` except that on success, it returns the string token that matched along with the leftover input. E.g., `match` with the expression for `abc` and the string `"abcd"` returned `Just("d")`. `capture` on the same arguments should return `Just("abc", "d")`. The attached file `HW_04_440_cap_tests.hs` that does the same tests as the test file for Lecture 7 but checks for `capture` output instead of `match` output.

   Note that `capture` uses a helper function `capture'` that returns the reverse of the token (plus the leftover input), so you'll need to reverse it. E.g., `capture'` might return `Just("cba", "d")` but `capture` should return `Just("abc, "d")`.

   You're only required to implement code for `RE_const`, `RE_and`, and `RE_or`. [Point breakdown: `capture`: 3 pts, `capture'` for (`RE_const`: 3 pts; `RE_and`: 5 pts; and `RE_or`: 5 pts). Your submission should include the entire definition of `capture` in your `*.hs` file (including the skeleton part). Leave out the `RegExpr data` declaration.

Lecture 8: Finite State Automata

5. [3 points] Take the regular expression of Problem 1 (`a`'s and `b`'s with an even number of `a`'s or an even number of `b`'s), and design two DFAs that each solve half the problem (one DFA should check for even number of `a`'s and don't care parity of `b`'s, and the other should check for an even number of `b`'s and don't care parity of `a`'s). Then build an NFA that at start, simply does ε-transitions to the start states of the two DFAs. Write out a description of the NFA using either a state transition table or diagram (your choice).

6. [14 points] Study the NFA described by the following transition table. The start state is *A*; the accept state is *G*. An empty entry indicates that there is no transition. (E.g., there's no transition from *B* labeled ε.) The entry *D*, *E* for state *D* on input `x` means that there are two arrows labeled `x`, one from *D* to *D* and one from *D* to *E*.

State	ε	`x`	`y`	`z`
A	*B*	*C*	*F*	
B	*D*		*H*	*G*
C	*H*	*C*	*A*	
D		*D, E*	*E*	
E	*D*	*D*		*E*
F		*G*		*G*
G		*C*	*E*	*F*
H	*C*	*H*		*H*

(The start state is A; the single accept state is G.)

a. [3 points] Check the table for error states (states that can't possibly lead to an accepting state) and drop them from the NFA. (This will create more "stuck" configurations.) Show the new transition table.

b. [5 points] Take the ε-closure of the result of part (a) to get an equivalent ε-free NFA. Show the new transition table.

c. [6 points] Take the result of part (b) and use the set-of-states algorithm to calculate an equivalent DFA. (You can leave empty table entries empty; we'll treat them as implicitly going to an error state.)