Coursework 1

6CCS3CFL - Compilers & Formal Languages

Finley Warman

October 15, 2020

N.B. to run all test cases and questions, run: ‘`amm 01_coursework.sc all`’

Contents

Question 1 **2**

Question 2 **2**

Question 3 **2**

Question 4 **3**

Question 5 **4**

Question 6 **4**

Question 7 **4**

Question 1

Q: What is your King's Email Address? (and where will you be studying?)

A: `finley.warman@kcl.ac.uk`
I will be studying (at least for now) from my family home in Bath, England.

Question 2

Q: In which programming languages have you already written programs?

A:

- Perl (text processing @ Netcraft)
- PHP (unfortunately)
- JavaScript (and its various magical frameworks)
- Python (e.g. https://github.com/finwarman/chordy)
- C++ (Console-Based Raytracer)
- C# .NET (e.g. https://github.com/finwarman/careful-renamer)
- Scala (BF Interpreter last year)
- TeX (this document!)
- Shell script
- HTML / CSS (does this count?)
- Swift
- Java
- possibly more, e.g. some small amounts of Go, Haskell.

Question 3

Q: Definitions for nullable:

```
nullable([c1,...,cn]) == false
nullable(r+)          == nullable(r)
nullable(r?)          == true
nullable(r{n})        == if (n=0) true else nullable(r)
nullable(r{..m})      == true
nullable(r{n..})      == if (n=0) true else nullable(r)
nullable(r{n..m})     == if (n=0) true else nullable(r)
nullable(~r)          == not nullable(r)
```

Q: Definitions for der:

```
der c ([c1,...,cn]) == if c ∈ [c1,...,cn] then 1 else 0
der c (r+)          == (der c r) . r*
der c (r?)          == (der c r)
der c (r{n})        == if (n=0) then 0 else ((der c r) . r{n-1})
der c (r{..m})      == if (m=0) then 0 else ((der c r) . r{..m-1})
der c (r{n..})      == if (n=0) then ((der c r) . r{0..}) else ((der c r) . r{n-1..})
der c (r{n..m})     == if (n=0 and m=0) then 0
                         elif (n=0) then ((der c r) . r{0..m-1})
                         else ((der c r) . r{n-1..m-1})
der c (~r)          == ~(der c r)
```

(Further transformations are possible, such as `r{0..} -> r*` and `r{0..m} -> r{..m}`, however these are not implemented in order to keep the recursive definitions simple.)

Q: Test Table Results:
A: (This can be generated by running '`amm 01_coursework.sc question3`')

```
 string | a?  | ~a  | a{3}  | (a?){3}  | a{..3}  | (a?){..3}  | a{3..5}  | (a?){3..5}  | a{0}  |
--------+-----+-----+-------+----------+---------+------------+----------+------------+-------+
[]      | YES | YES | -     | YES      | YES     | YES        | -        | YES        | YES   |
a       | YES | -   | -     | YES      | YES     | YES        | -        | YES        | -     |
aa      | -   | YES | -     | YES      | YES     | YES        | -        | YES        | -     |
aaa     | -   | YES | YES   | YES      | YES     | YES        | YES      | YES        | -     |
aaaa    | -   | YES | -     | -        | -       | -          | YES      | YES        | -     | <-(extra)
aaaaa   | -   | YES | -     | -        | -       | -          | YES      | YES        | -     |
aaaaaa  | -   | YES | -     | -        | -       | -          | -        | -          | -     |
```

Additional test cases for each rexp type can be checked by running:

```
amm 01_coursework.sc unitTests
```

These tests pass, so the results produced are as I expected!

Question 4

Q: Definitions for nullable, der, and cfun-related functions.

A: I implemented CFUN after the initial CHAR implementation, and used `CFUN(_CHAR(c))`, `CHAR2(c)`, etc. only after implementing them.

To run CFUN tests: '`amm 01_coursework.sc question4`'
This adds `CFUN`:

```
case class CFUN(f: Char => Boolean) extends Rexp
def nullable ... case CFUN(f) => false
der der      ... case CFUN(f) => if (f(c)) ONE else ZERO
```

alongside the following functions for `char, range, all`:

```
def _char(ch: Char): Char => Boolean           = { (c: Char) => {(ch == c)} }
def _range(chars: Set[Char]) : Char => Boolean = { (c: Char) => {chars.contains(c)} }
def _all() : Char => Boolean                   = { (c: Char) => true }
```

and these specific instances of CFUN to replace the existing CHAR, RANGE, etc.:

```
def CHAR2(c: Char)           = CFUN(_char(c))
def RANGE2(chars: Set[Char]) = CFUN(_range(chars))
val ALL                      = CFUN(_all())
```

Example: `SEQ(CFUN(_CHAR('a')), SEQ(CFUN(_RANGE(Set('b', 'B'))), STAR(CFUN(_ALL))))` matches: $a[bB].*$, as does `CHAR2('a') o RANGE2(Set('b', 'B')) o STAR(ALL)`
(using custom 'o' infix notation for SEQ)

Question 5

Q: Email Address Regular Expressions and Derivative w.r.t. my email.

A: (To run: '`amm 01_coursework.sc question5`')
Ders `"finley.warman@kcl.ac.uk"` $([-.\_0\text{-}9a\text{-}z]^{+} \cdot (@ \cdot ([-.0\text{-}9a\text{-}z]^{+} \cdot (. \cdot [.a\text{-}z]^{\{2..6\}}))))$:

$$((([-.0\text{-}9a\text{-}z]^{*} \cdot (. \cdot [.a\text{-}z]^{\{2..6\}})) + [.a\text{-}z]^{\{0..4\}}) + [.a\text{-}z]^{\{0..1\}})$$

This final derivative matches the empty string ε, therefore the Email Rexp matches the input string of my email address.

Question 6

Q: Determine whether the following match the expression $/ \cdot * \cdot (\tilde{}(ALL^{*} \cdot * \cdot / \cdot ALL^{*})) \cdot * \cdot /$

A: (To run: '`amm 01_coursework.sc question6`')

- matches `/**/`? - *YES*
- matches `/*foobar*/`? - *YES*
- matches `/*test*/test*/`? - *NO*
- matches `/*test/*test*/`? - *YES*

Question 7

Q: Determine whether the following match the expressions $r_1 = a \cdot a \cdot a$ and $r_2 = (a^{\{19,19\}}) \cdot (a^{?})$ when in the form $({r_1}^{+})^{+}$ and $({r_2}^{+})^{+}$.

A: (To run: '`amm 01_coursework.sc question7`')

- $({r_1}^{+})^{+}$ matches 5.? - *YES*
- $({r_1}^{+})^{+}$ matches 6.? - *NO*
- $({r_1}^{+})^{+}$ matches 7.? - *NO*
- $({r_2}^{+})^{+}$ matches 5.? - *YES*
- $({r_2}^{+})^{+}$ matches 6.? - *NO*
- $({r_2}^{+})^{+}$ matches 7.? - *YES*