Homework 2

6CCS3CFL - Compilers & Formal Languages

Finley Warman

October 10, 2020

Contents

Question 1

Q: What is the difference between basic and extended regular expressions?

A: 'Basic' Regular Expressions are those defined by the grammar of `r: 0, 1, c (char), ALT, SEQ, STAR` - they represent the formal concept of regular expressions.

'Extended' Regular Expressions are those found in most modern programming languages, with extended features for matching patterns in strings, such as `a{n}, a+, etc.` Some of these features are simply syntactic sugar over features found in basic regular expressions.

Question 2

Q: What is the language recognised by regular expression `(0*)*`?

Through the following simplifications:

`(0*)* -> (0)* -> 0* -> 0` we can show that `(0*)* == 0`.

Therefore this expression only matches the empty language, `L(0) = {}`.

Question 3

Q: Decide which of the following are true in general for arbitrary languages A, B, and C. (Assuming alphabet a, b)

A: $(A \cup B)@C =?A@C \cup B@C$

YES - Since in both cases, the resulting set contains only those strings from A concatenated with C, and B concatenated with C. (in general, $(A \cup B) \times C = (A \times B) \cup (B \times C)$)

A: $A* \cup B* =?(A \cup B)*$

NO -

```
A=a, B=b
A*    = [], a, aa, aaa, ...
B*    = [], b, bb, bbb, ...
A*UB* = [], a, b, aa, bb, aaa, bbb

AUB    = a, b
(AUB)* = [], a, b, ab, ..
```

"ab" not in A* ∪ B* and is in (A ∪ B)*, therefore A* ∪ B* !== (A ∪ B)*

A: $A* @ A* =?A*$

YES - Since the Kleene star of a language is the union of that language raised to every power $>= 0$, squaring the Kleene star itself will contain no additional strings.

A: $(A \cap B)@C =?(A@C) \cap (B@C)$

YES (in general, $(A \cap B) \times C = (A \times B) \cap (B \times C)$)

Question 4

Q: Given expressions `r1 = 1` and `r2 = 0` and `r2 = a`, how many strings can the regular expressions r_1^*, r_2^*, and r_3^* each match?

A: r1* can match 1 string, ϵ, r2* can match 0 strings, and r3 can match arbitrarily many (>= 0).

Question 5

Q: Give regular expressions for a) decimal numbers, and b) binary numbers NB: empty string is not a number, leading 0s are not normally written (e.g. forbidden is JSON).

A: Decimal: `0 + ((1+2+3+4+5+6+7+8+9).(0+1+2+3+4+5+6+7+8+9)*)`
A: Binary: `0 + (1.(0+1)*)`

Question 6

Q: Decide whether the following two regular expressions are equivalent:
`(1+a)* ==? a* AND (a.b)*.a ==? a.(b.a)*`
A: YES; `(1+a)*` - 'the empty string or a, any number of times', this is equivalent to 'accept 0 or more a, or the empty string', which is `a*`.

A: YES; `(a.b)*.a == a.(b.a)*`

Question 7

Q: Given the regular expressions r = (a.b+b), compute what the derivate of r is w.r.t. a, b, and c. Is r nullable?
A: i) der a r: `b.r`
ii) der b r: `r`
iii) der c r: `0`
iv) nullable(r): `true` (since nullable(r*) == true, r* is 0 or more so can match empty string)

Question 8

[Moved to Homework 3]

Question 9

Q: Define what is meant by the derivative of a regular expression w.r.t a characters. (Hint: it is defined recursively)

A: The derivate of a regular expression r w.r.t. to a character c is the expression matching only s if r matches c::s, i.e. the resulting expression matches only the tail of every string starting with character c.

```
der c (0)     == 0
der c (1)     == 0
der c (d)     == if c=d then 1 else 0
der c (r1+r2) == der c (r1) + der c (r2)
der c (r1.r2) == if nullable(r1):
```

```
                    then: (der c r1) . r2 + der c r2
                    else: (der c r1) . r2
  der c (r*)    == (der c r) . (r*)
```

Question 10

Q: Assume the set `Der` is defined as:
`Der c A == {s | c::s ∈ A}`
What is the relation between Der, and the notion of derivatives of regular expressions.
A: The set `Der c A` contains all strings s where c::s is in A, that is to say it contains the tail of all strings in the language A starting with character c. If language A were the language of some expression r, that is `L(r) = A`, then `Der c A` would be the language accepted by the derivative of expression r w.r.t c, `der c r` .
`L(der c r) == Der c L(r)`

Question 11

Q: Give a rexp over a, b recognising all strings that do not contain any substring bb and end in a.
A: `((b.(bb)*) + a)* . a`

Question 12

A: NO, the former accepts "bab" whereas the latter does not.

Question 13

A: zeroable(r) iff L(r) = {}

```
  zeroable(0)        = true
  zeroable(1)        = false
  zeroable(c)        = false
  zeroable(r1 + r2)  = zeroable(r1) && zeroable(r2)
  zeroable(r1 . r2)  = zeroable(r1) && zeroable(r2)
  zeroable(r*)       = zeroable(r)
```

A: Where does `zeroable(~r) != ~(zeroable(r))` ?
e.g. `r = 1` , since `zeroable(~r) = false, and ~(zeroable(r)) = true`

Question 14

A: Interpreting this as the set of all strings `a{n}` where n is a multiple of 3, plus one.
`(a.a.a)*.a`

Question 15

A: `(a.(a.a)*) + (b.b)*`

Question 16 (Optional)

This tool created for Stanford's CS103 class is built for working with DFAs and NFAs, including parsing regular expressions to these automata.
https://github.com/bakkot/dfa-lib
It also contains functionality for comparing these automata for equivalence, and finding a counterexample where they are not.
This library is used at https://bakkot.github.io/dfa-lib/regeq.html which is a regex equivalence checker.

A cursory look at the source shows that this finds these counterexamples by finding the shortest string accepted by the resulting intersection (a new DFA) of machine A, and the complement of machine B (if such a string exists), and vice-versa.