Homework 4

6CCS3CFL - Compilers & Formal Languages

Finley Warman

November 1, 2020

Contents

Question 1

Q: Give *all* the values and indicate which one is POSIX for how these expressions can recognise strings.

A: `(ab + a) . (1 + b)` matching `ab`.

- `Sequ( Left(Sequ(Chr(a),Chr(b))), Left(Empty) )` (*POSIX*)
- `Sequ( Right(Chr(a)), Right(Chr(b)) )`

A: `(aa + a)*` matching `aaa`.

- `Stars([ Left(Sequ(Chr(a), Chr(a))), Right(Chr(a)) ])` (*POSIX*)
- `Stars([ Right(Chr(a)) ])`
- `Stars([ Left(Sequ(Chr(a), Chr(a))) ])`

Question 2

Q: If a regular expression r does not contain any occurence of **0**, is it possible for L(r) to be empty?

Assuming that an expression can only be defined in terms of a valid alphabet (i.e. there is no 'unmatchable character'), and negation is not allowed, then L(r) cannot be empty. This is because excluding **0**, all expressions are defined recursively in terms of either **1** or the empty string, and if either of these is accepted then the language cannot be empty. If negation is allowed, then an example of an empty language would be `r` . r ., or the union of the language of r, and its complement. (Which is always empty)

Question 3

Q: Define tokens for a language with numbers, parenthesis, and operations. Can the given strings be lexed?

A: Token / Expression Defs:

```
DIGIT       = RANGE("0123456789")
START_DIGIT = RANGE("123456789")
NUMBER      = DIGIT + (START_DIGIT . DIGIT*)

LPAREN      = CHAR('(')
RPAREN      = CHAR(')')

OPERATOR    = CHAR('+') + CHAR('-') + CHAR('*')

LOWERCASE   = RANGE("abcdefghijklmnopqrstuvwxyz")
ID          = LOWERCASE . LOWERCASE*

LANG        = ("num":NUMBER) + ("op":OPERATOR) + ("lp":LPAREN) + ("rp":RPAREN) + ("id":I
```

- `(a+3)*b` = YES, `lp:(, id:a, op:+, num:3, rparen:), op:*, id:b`
- `)()++-33` = YES, `rp:), lp:(, rp:), op:+, op:+, op:-, num:33`
- `(a/3)*3` = NO, `/` is not an accepted token.

Question 4

Q: Assuming r is nullable, show that `1+r+r.r == r.r` holds.

```
1+r+r.r as a proper tree:
  = (((1+r)+r).r)
since r is nullable, (1+r) == r
  = (((r)+r).r) = ((r+r).r)
since (r+r) == r:
  = (r.r) = r.r
therefore
  1+r+r.r == r.r
```

Question 5 (Deleted)

Question 6

Q: Give a regular expression to match comments of the form `/* ... */`

A:

```
SEQ(
  SEQ(CHAR('/'), CHAR('*')),
  SEQ(
    STAR(NOT(SEQ(CHAR('*'), CHAR('/')))),
    SEQ(CHAR('*'), CHAR('/'))
  )
)
```

Question 7

Q: Simplify the given expression. Does simplification always preserve the meaning of a regular expression?

A:

```
Simplifying (0.(b.c))+((0.c)+1):
Using 0.r = 0:
    (0) + ((0)+1)
  = 0 + (0 + 1)
Using 0+r = r:
    0 + (1)
  = 0 + 1
  = 1
```

The regex produced by simplification will be equivalent to its pre-simplified form, in that they will accept the same language.
However, the resulting expression may vary in *how* it matches a string, and thus (unless steps are taken to rectify this), the returned matching value may be different.

Question 8

Q: What is *mkeps* for the following expressions? A:

- `(0.(b.c))+((0.c)+1)` $= Right(Right(Empty))$
- `(a+1).(1+1)` $= Sequ(Right(Empty), Left(Empty))$
- `a*` $= Stars(Nil)$

Question 9

Q: What is the purpose of the record regular expression in the Sulzmann & Lu Algorithm?

A: When tokenizing an expression (e.g. splitting into its component words), the record expression is used for classifying these tokens.
e.g. when lexing a block of code, we can produce a resulting expression of records which label each (notable) sub-expression with the token they matched.

Question 9

Q: Define recursive functions *atmostempty*, *somechars*, $infinitestrings$. (Recalling *nullable* and *zeroable*).

```
atmostempty -
  atmostempty(0):     true
  atmostempty(1):     true
  atmostempty(c):     false
  atmostempty(r1+r2): atmostempty(r1) && atmostempty(r2)
  atmostempty(r1.r2): atmostempty(r1) || atmostempty(r2)
  atmostempty(r*):    atmostempty(r)
```

```
somechars -
  somechars(0):     false
  somechars(1):     false
  somechars(c):     true
  somechars(r1+r2): somechars(r1) || somechars(r2)
  somechars(r1.r2): somechars(r1) || somechars(r2)
  somechars(r*):    somechars(r)

infinitestrings -
  infinitestrings(0):     false
  infinitestrings(1):     false
  infinitestrings(c):     false
  infinitestrings(r1+r2): infinitestrings(r1) || infinitestrings(r2)
  infinitestrings(r1.r2): infinitestrings(r1) || infinitestrings(r2)
  infinitestrings(r*):    ~atmostempty(r)
```