6CCS3CFL Homework 4

1.If a regular expression r does not contain any occurrence of 0, is it possible for L(r) to be empty? Explain why, or give a proof.

No, it is not possible for L(r) to be empty if there is no occurrence of 0 in r.

Let P(r) be the property that L(r) is not empty if and only if r does not contain any occurrence of 0.

P(0): in this case, r contains 0, and L(0) = {} so our property holds.

P(1): r is 1, and does not contain any 0. L(1) = {[]} so is not empty. Property holds.

P(c): r is c, and does not contain any 0. L(c) = {[c]} so is not empty. Property holds.

P(r1 + r2): assume that P(r1) holds, and P(r2) holds, meaning that r1 and r2 do not contain any occurrence of 0, and that L(r1) is not empty and L(r2) is not empty.

> L(r1 + r2) = L(r1) ∪ L(r2). If both L(r1) and L(r2) are not empty, then the union of two non-empty sets cannot be the empty set. So P(r1 + r2) holds.

P(r1 • r2): assume that P(r1) holds, and P(r2) holds, meaning that r1 and r2 do not contain any occurrence of 0, and that L(r1) is not empty and L(r2) is not empty.

> L(r1 • r2) = {s1@s2 | s1 ∈ L(r1) ∧ s2 ∈ L(r2) }. Neither L(r1) or L(r2) are empty so at least one concatenation operation occurs, meaning L(r1 • r2) is not empty. Property holds.

P(r*): assume that P(r) holds meaning that r do not contain any occurrence of 0 and that L(r) is not empty.

> L(r*) = $\cup_{0 \leq n} L(r)^n$ = $L(r)^0 \cup L(r)^1 \cup L(r)^2 \cup L(r)^3$.............. = {[]} ∪ L(r) ∪ L(r) @ L(r)......
> {} is not contained in the language. So Property holds.

2. Define the tokens and regular expressions for a language consisting of numbers, left-parenthesis (, right parenthesis), identifiers and the operations +, − and ∗. Can the following strings in this language be lexed?

• (a+3)∗b
•)()++−33 • (a/3)∗3

In case they can, can you give the corresponding token sequences.

Tokens:

LEFT_PARENTHESES
RIGHT_PARENTHESES
NUMBER
IDENTIFIER
OPERATOR

Regular Expressions:

Assume regular expression **NONZERODIGIT, DIGIT** and **LETTER** has already been defined.

LEFT_PARENTHESES = (
RIGHT_PARENTHESES =)
NUMBER = (NONZERODIGIT • DIGIT*) + 0
IDENTIFIER = LETTER • (LETTER + DIGIT + _)*
OPERATOR = +, -, *

- (a+3)∗b This can be matched
-)()++−33 • (a/3)∗3 This cannot be matched as it contains / which is not a valid operator

MORE ON NEXT PAGE

3. Assume that s^{-1} stands for the operation of reversing a string s. Given the following *reversing* function on regular expressions

$$\begin{aligned}
rev(\mathbf{0}) &\stackrel{\text{def}}{=} \mathbf{0} \\
rev(\mathbf{1}) &\stackrel{\text{def}}{=} \mathbf{1} \\
rev(c) &\stackrel{\text{def}}{=} c \\
rev(r_1 + r_2) &\stackrel{\text{def}}{=} rev(r_1) + rev(r_2) \\
rev(r_1 \cdot r_2) &\stackrel{\text{def}}{=} rev(r_2) \cdot rev(r_1) \\
rev(r^*) &\stackrel{\text{def}}{=} rev(r)^*
\end{aligned}$$

and the set

$$Rev\, A \stackrel{\text{def}}{=} \{s^{-1} \mid s \in A\}$$

prove whether

$$L(rev(r)) = Rev(L(r))$$

Let P(r) be property that L(rev(r)) = Rev(L(r))

If r = 0:

- rev(0) = 0
- L(rev(0)) = L(0) = {}
- L(0) = {}
- Rev(L(0)) = Rev({}) = {}
- This property holds

If r = 1:

- rev(1) = 1
- L(rev(1)) = L(1) = {[]}
- L(1) = {[]}
- Rev(L(1)) = Rev({[]}) = {[]}
- This property holds

If r = c:

- rev(c) = c
- L(rev(c)) = L(c) = {[c]}
- L(c) = {[c]}
- Rev(L(c)) = Rev({c}) = {[c]}
- This property holds

If r = r1 + r2:

- Assume property holds for r1, and r2.
- rev(r1 + r2) = rev(r1) + rev(r2)
- L(rev(r1) + rev(r2)) = L(rev(r1)) ∪ L(rev(r2))
- L(r1 + r2) = L(r1) ∪ L(r2)
- Rev(L(r1 + r2)) = Rev(L(r1) ∪ L(r2))
- Property holds

If r = r1 • r2:

- Assume property holds for r1, and r2.
- rev(r1 • r2) = rev(r2) • rev(r1)
- L(rev(r2) • rev(r1)) = {s2 @ s1 | s2 ∈ rev(r2) ∧ s1 ∈ rev(r1) }
- L(r1 • r2) = {s1 @ s2 | s1 ∈ L(r1) ∧ s2 ∈ L(r2) }
- Rev(L(r1 • r2)) = Rev({s1 @ s2 | s1 ∈ L(r1) ∧ s2 ∈ L(r2) })
- Property holds

4. Assume the delimiters for comments are /* and */. Give a regular ex- pression that can recognise comments of the form
/* ... */
where the three dots stand for arbitrary characters, but not comment de- limiters. (Hint: You can assume you are already given a regular expres- sion wri�en ALL, that can recognise any character, and a regular expres- sion NOT that recognises the complement of a regular expression.)

/ • * • (NOT((ALL* • ((/ • *) + (* • /)) • ALL*) • * • /

5. Simplify the regular expression
(0 · (b · c)) + ((0 · c) + 1)
Does simplification always preserve the meaning of a regular expression?

(0 · (b · c)) + ((0 · c) + 1)
(0 + ((0 · c) + 1)
(0 + (0 + 1))
(0 + 1)
1

Yes, simplication must always preserve the meaning of a regular expression. Meaning that both non-simplified and simplified regular expressions are equivalent and have the same language.

6. The Sulzmann & Lu algorithm contains the function mkeps which answers how a regular expression can match the empty string. What is the answer of mkeps for the regular expressions:

$(0 \cdot (b \cdot c)) + ((0 \cdot c) + 1)$
$(a + 1) \cdot (1 + 1)$
$a*$

$(0 \cdot (b \cdot c)) + ((0 \cdot c) + 1)$ -> Right(Right(Empty))

$(a + 1) \cdot (1 + 1)$ -> Left(Right(Empty))

$a*$ -> Stars[]

7. What is the purpose of the record regular expression in the Sulzmann & Lu algorithm?

When we tokenise an input string, it is easier to be able to identify each token with some readable word, the record regular expression simply annotates a regular expression with some identifier.

8. Recall the functions nullable and zeroable. Define recursive functions at- mostempty (for regular expressions that match no string or only the empty string), somechars (for regular expressions that match some non-empty string), infinitestrings (for regular expressions that can match infinitely many strings).

Atmostempty(0) = true
Atmostempty(1) = true
Atmostempty(c) = false
Atmostempty(r1 + r2) = (nullable(r1) or zeroable(r1)) or (nullable(r2) or zeroable(r2))
Atmostempty(r1 • r2) = (nullable(r1) and nullable(r2)) or (zeorable(r1) or zeroable(r2))
Atmostempty(r*) = true

Somechars(0) = false
Somechars(1) = false
Somechars(c) = true
Somechars(r1 + r2) = somechars(r1) or somechars(r2)
Somechars(r1 • r2) = if (somechars(r1)) then true
Else if (somechars(r2)) then true
Else false
Somechars(r*) = if (somechars(r)) then true else false

Infinite(0) = false, infinite(1) = false, infinite(c) = false
Infinite(r1 + r2) = infinite(r1) or infinite(r1)
Infinite(r1 • r2) = infinite(r1) or infinite(r2)
Infinite(r*) = if (nullable(r) or zeroable(r)) then false else true