B.

i. Exercise

regular expressions

re ::= !	no string (R NoString)
| #	empty string (R Empty String)
| .	any character (R Any Char)
| c	the character c (R Single(c))
| re1re2	concatenation (R Concat(re1,re2))
| re1‘|’re2	union, “or” (RUnion(re1,re2))
| re1*	Kleene star, “0-or-more” (RStar(re1))
| re1+	“1-or-more” RPlus(re1))
| re1?	“0-or-1” ((R Option(re1))
| re1&re2	intersection, “and” (R Intersect(re1,re2))
| ~re1	complement, “not” (RNeg(re1))

i. Regular:

re ::= union
union ::= union ‘|’ intersect | intersect
intersect ::= intersect ‘&’ concat | concat
concat ::= concat not | not
not ::= ‘~’not | star
star ::= star ‘*’ |star ‘+’ | star ‘?’ | atom
atom ::= ‘c’ | ’#’ | ’!’ | ’.’ | re

ii.Exercise

When we implement this grammar:
Def union(next: Input): ParseResult[RegExpr] = union(next) match {… union will always match on union(next), and call the union() function again. Because nothing in 'next' is consumed, the function will lead to a infinite loop.

iii. Exercise

EBNF

re ::= union
union ::= intersect {‘|’ intersect}
intersect ::= concat {‘&’ concat}
concat ::= not {not}

```
not ::= ‘~’not | star
star ::=  atom  {‘*’}|  atom  {‘+’}| atom {‘?’}
atom ::= ‘c’|’#’|’!’|’.’| {}| re
```

iv. Exercise

EBNF to BNF

```
re ::= union

union ::= intersect unions
unions ::= E | ‘|’ intersect unions

intersect ::= concat intersects
intersects ::= E | ‘&’ concat intersects

concat ::=  not concats
concats ::=  E | not concats

not ::= nots star
nots ::= E | ‘~’ notsa

star ::=  atom stars
stars ::=  E | ‘*’ atom | ‘+’ atom| ‘?’ atom

atom ::= ‘c’|’#’|’!’|’.’| {}|re
```

C.

TypeRegExp

$$\frac{\Gamma \vdash e1 : \text{RegExp} \quad \Gamma \vdash e2 : \text{String}}{\Gamma \vdash e1.\text{test}(e2) : \text{bool}}$$

SearchCallRegExp

$$\frac{e1 = \text{GetField}(e1, \text{“test”}) \quad e2 \rightarrow e2'}{e1.\text{test}(e2) \rightarrow e1.\text{test}(e2')}$$

DoCallRegExp

$$\frac{e = \text{GetField}(e1, \text{“test”}) \quad v1 = \text{List}(e2)}{e1.\text{test}(e2) \rightarrow \text{retest}(e1,e2)}$$