Lab 6 Write-up

b)

I)

re ::= union

union ::= union ‘|’ intersect | intersect

intersect ::= intersect '&' concat | concat

concat ::= concat not | not

not ::= ‘~’ not | star

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

atom ::= ‘!’ | ‘#’ | ‘.’ | ‘c’

ii) Recursive descent parse: top to bottom, left to right.

Going from top to bottom and left to right will force the recursion to always match on the non-terminal on the left hand side and continue infinitely rather than reaching a terminal in the grammar. For the parsing to work the production must be consuming the characters from the string to match a terminal, but in this case the left recursion will continue to call the function corresponding to a non-terminal to try to match that non-terminal resulting in the infinite loop.

iii)

re ::= union

union ::= intersects unions

unions ::= empty | ‘|’ intersects unions

intersect ::= concat intersects

concat ::= not concats

concats ::= empty | not concats

not ::= ‘~’ not | star

star ::= atom stars

stars ::= empty | ‘*’ atom stars | ‘+’ atom stars | ‘?’ atom stars

atom ::= ‘!’ | ‘#’ | ‘.’ | ‘c’

c)

i) Typing Rules:

$$\frac{}{\Gamma \vdash /\hat{}re\ \$/ : RegExp}\ \text{TypeRegExLiteral}$$

$$\frac{\Gamma \vdash e_1 : RegExp \quad \Gamma \vdash e_2 : string}{\Gamma \vdash e_1.test(e_2) : bool}\ \text{TypeRegExTest}$$

Small-step operational semantics:

$$\text{Search 1}\quad \frac{e_1 \rightarrow e_1'}{e_1.test(e_2) \rightarrow e_1'.test(e_2)}$$

$$\text{Search 2}\quad \frac{v_1 = re \quad e_2 \rightarrow e_2'}{v_1.test(e_2) \rightarrow v_1.test(e_2')}$$

$$\text{Do}\quad \frac{b = retest(re, s) \quad v_1 = re \quad v_2 = s}{v_1.test(v_2) \Downarrow b}$$