Problem Sheet 1

Henri Sota
h.sota@jacobs-university.de
Computer Science 2022

September 20, 2019

1 Problem 1.1

a) **Naive String Search**

```
A B B A B B C A C C A B A B A C B C C A B A B
A B A b
  A b a b
    A b a b
      A B A b
        A b a b
          A b a b
            A b a b
              A B a b
                A b a b
                  A b a b
                    A B A B
```

Using the Naive String Search, we went through 11 alignments and 19 comparisons in total to find first "ABAB" inside "ABBABBCACCABABACBCCABAB"

b) **Bad Character Rule**

```
A B B A B B C A C C A B A B A C B C C A B A B   // Skip Calculation
a b a B                                         // 0
  A B A B                                       // 3 (shift whole)
          a b a B                               // 3 (shift whole)
                  a b a B                       // 0
                    A B A B                     // Found
```

Using the Bad Character Rule, we went through 5 alignments and 11 comparisons in total to find first "ABAB" inside "ABBABBCACCABABACBCCABAB"

c) **Good Suffix Rule**

```
A B B A B B C A C C A B A B A C B C C A B A B   // Skip Calculation
a b a B                                         // Skip 0
  A B A B                                       // Skip 1 (ii) *
      a b a B                                   // Skip 0
        a b a B                                 // Skip 0
          a b a B                               // Skip 0
            a b a B                             // Skip 0
              a b a B                           // Skip 0
                a B A B                         // Skip 1 (i) **
                    A B A B                     // Found
```

* Found prefix "AB" matching with suffix "AB" of suffix "BAB"

** Found suffix "AB" before position 2 of the pattern

Using the Good Suffix Rule, we went through 9 alignments and 17 comparisons in total to find first "ABAB" inside "ABBABBCACCABABACBCCABAB"

d) **Bad Character & Good Suffix**

```
A B B A B B C A C C A B A B A C B C C A B A B   // Skip Calculation
a b a B                                         // bc: 0 gs: 0  *
  A B A B                                       // bc: 3 gs: 1  **
          a b a B                               // bc: 3 gs: 0  ***
                  a b a B                       // bc: 0 gs: 0  *
                    A B A B                     // Found
```

* We choose either one

** We choose Bad Character since we have a mismatch of the first character of our pattern

*** We choose Bad Character since we can't find 'C' anywhere in the leftside of our suffix

Using both the Bad Character Rule and Good Suffix Rule, we went through 5 alignments and 11 comparisons in total to find first "ABAB" inside "ABBABBCACCABABACBCCABAB"

2 Problem 1.2

a) Infix: 5 * (5 + 2 * (4 + 3)) - (5 * 10 + 3)
Prefix: (-) ((*) 5 ((+) 5 ((*) 2 ((+) 4 3)))) ((+) ((*) 5 10) 3)

b) Prefix: gcd (div 42 2) (mod 30 16)
Infix: (42 ‘div‘ 2) ‘gcd‘ (30 ‘mod‘ 16)

3 Problem 1.3

Operator	Precedence Level	Associativity
+	6	infixl (left)
-	6	infixl (left)
*	7	infixl (left)
/	7	infixl (left)
ˆ	8	infixr (right)
$	0	infixr (right)
&&	3	infixr (right)
||	2	infixr (right)

b) Some of the operators that are non-associative in Haskell are:
==, /=, <, <=, >, >=, ‘elem‘, ‘notElem‘

Following the guidelines of Haskell:

Consecutive unparenthesized operators with the same precedence must both be either left or right associative to avoid a syntax error. [Haskell 2010 Report, Ch. 3]

```
Example: 1 < 2 > 3

           Trying a combination of 2 comparison operators
           (same precedence level) which have no associativity
           produces a syntax error and at the same time
           parsing error:

           Precedence parsing error
                   cannot mix '<' [infix 4] and '>'
                   [infix 4] in the same infix expression
```