Introduction to Computer Science
Jacobs University Bremen
Dr. Jürgen Schönwälder

Course: CH-232-A
Date: 2019-09-13
Due: 2019-09-20

ICS 2019 Problem Sheet #1

Problem 1.1: *boyer moore bad character rule* (2+2+2+2 = 8 points)

Let $\Sigma = \{A, B, C, D\}$ be an alphabet and $t \in \Sigma^*$ be a text of length n and $p \in \Sigma^*$ be a pattern of length m. We are looking for the first occurrence of p in t.

Consider the text $t = ABBABBCACCABABACBCCABAB$ and the pattern $p = ABAB$.

You are asked to carry out string search. Write down how the search proceeds using the notation used on the slides. Show each alignment (via indentation) and indicate comparisons performed with a capital letter and comparisons not performed with lowercase letters. Indicate in each step which rule you apply (default shift, bad-character, good-suffix(i), good-suffix(ii), or good-suffix(iii)).

a) Execute the naive string search algorithm. Show all alignments and indicate comparisons performed by writing uppercase characters and comparisons skipped by writing lowercase characters. How many alignments are used? How many character comparisons are done?

b) Execute the Boyer-Moore string search algorithm with the bad character rule only. How many alignments are used? How many character comparisons are done?

c) Execute the Boyer-Moore string search algorithm with the good suffix rule only. How many alignments are used? How many character comparisons are done?

d) Execute the Boyer-Moore string search algorithm applying both the bad character rule and and good suffix rule.

Problem 1.2: *infix and prefix expressions (haskell)* (1 point)

a) Convert the following infix expression into the corresponding prefix expression.

```
5 * (5 + 2 * (4 + 3)) - (5 * 10 + 3)
```

b) Convert the following prefix expression into the corresponding infix expression.

```
gcd (div 42 2) (mod 30 16)
```

Problem 1.3: *operator precedence and associativity (haskell)* (1 point)

Haskell operators have associativity and precedence. The associativity defines in which order operators with the same precedence are evaluated. For example, the operators + and * both have left associativity while the operator ^ has right associativity. This leads to the following evaluation orders:

```
2+2+3  -- (2+2)+4  (left associativity)
2*2*3  -- (2*2)*4  (left associativity)
2^2^3  -- 2^(2^3)  (right associativity)
```

The precedence defines in which order operators with different precedence levels are evaluated (higher precedence level first). This is in particular important in expressions where several different operators appear:

```
2+2*3  -- 2+(2*3)  (* has higher precedence than +)
2^2*3  -- (2^2)*3  (^ has higher precedence than *)
```

A very special operator is the `$` operator, which is right associative and has precedence level 0 (the lowest precedence level possible). The `$` operator calls the function, which is its left-hand argument, on the value, which is its right-hand argument. This does not seem terribly useful but it can be used to reduce number of parenthesis needed. For example, the following two prefix expressions have the same evaluation order:

```
(*) 2 ((+) 2 3)
(*) 2 $ (+) 2 3
```

a) Lookup the precedence levels and the associativity of the following Haskell operators: `+`, `-`, `*`, `/`, `^`, `$`, `&&`, `||`

b) Some operators are neither left nor right associative. What happens if such operators appear multiple times in an expression (without additional parenthesis defining the evaluation order)? Provide an example and an explanation.