Introduction to Computer Science
Jacobs University Bremen
Dr. Jürgen Schönwälder

Course: CH-232-A
Date: 2019-12-20
Due: 2020-01-15

ICS 2019 Problem Sheet #13

This sheet is only for students who failed to reach the 50% module achievements.

Problem 13.1: *sum formula* (2 points)

Prove that $1 + 4 + \ldots + (3n - 2) = \frac{1}{2}n(3n - 1)$ for $n \in \mathbb{N}$ and $n > 0$.

Problem 13.2: *equivalence relation* (2 points)

Let $A = \mathbb{N}_+ \times \mathbb{N}_+$ be the set of pairs of positive natural numbers. Let $\sim \subseteq A \times A$ be a binary relation on A. The tuple $((a, b), (c, d))$ is an element of $\sim$ if and only if $ad = bc$ (the product of a and d is equal to the product of b and c).

Show that $\sim$ is an equivalence relation (i.e., $\sim$ is reflexive, symmetric and transitive). For each property, first state what you are trying to show before you provide the argument.

Problem 13.3: *not-or is a universal boolean function* (2 points)

Prove that not-or ($\downarrow$) is a universal boolean function by showing and proving how $\downarrow$ can be used to implement the Boolean functions $\wedge, \vee, \neg$.

Problem 13.4: *operating system processes* (1+1 = 2 points)

The following C program creates multiple processes. Assume that all system calls succeed at runtime (error handling code has been left out for brevity).

```
#include <unistd.h>

int main()
{
    pid_t pid;

    pid = fork();
    if (fork() == 0) {
        fork();
        if (pid != 0) {
            execlp("date", "date", NULL);
        }
    }

    return 0;
}
```

a) Draw a tree diagram showing the processes created during the execution of the program (include the initial process created when the program is started in your diagram). For each process in the diagram, indicate what the variable `pid` contains.

b) How many times will the program print the current date and time? Explain. How many processes return from the `main()` function of the program? Explain.

Problem 13.5: *functional programming in haskell* (1+1 = 2 points)

a) Write a function `divisors :: Int -> [Int]` that returns the list of proper divisors of a given number `x`. The result of `divisors x` includes 1, but not the number `x` itself. For example:

```
Prelude> divisors 6
[1,2,3]
Prelude> divisors 12
[1,2,3,4,6]
Prelude> divisors 15
[1,3,5]
Prelude> divisors 1
[]
Prelude> divisors 2
[1]
```

Recall that the Haskell function `div` gives you the result of an integer division (truncated toward negative infinity) and the function `mod` gives you the integer modulus (remainder of an integer division).

b) Consider the following function definition:

```
m f xs = foldr g [] xs
  where g y ys = (f y) : ys
```

How is the expression `m (*2) [1..3]` evaluated? Explain step-by-step how the expression is expanded and how the result is produced. Describe what the function `m` is doing, i.e., to which standard Haskell function it relates.