Introduction to Computer Science
Jacobs University Bremen
Dr. Jürgen Schönwälder

Course: CH-232-A
Date: 2020-10-02
Due: 2020-10-09

ICS 2020 Problem Sheet #4

Problem 4.1: *prefix order relations* (2+2+1 = 5 points)

Let Σ be a finite set (called an alphabet) and let Σ^* be the set of all words that can be created out the symbols in the alphabet Σ. (Σ^* is the Kleene closure of Σ, which includes the empty word ϵ.) A word $p \in \Sigma^*$ is called a prefix of a word $w \in \Sigma^*$ if there is a word $q \in \Sigma^*$ such that $w = pq$. A prefix p is called a proper prefix if $p \neq w$.

a) Let $\preceq \subseteq \Sigma^* \times \Sigma^*$ be a relation such that $p \preceq w$ for $p, w \in \Sigma^*$ if p is a prefix of w. Show that $\preceq$ is a partial order.

b) Let $\prec \subset \Sigma^* \times \Sigma^*$ be a relation such that for $p \prec w$ for $p, w \in \Sigma^*$ if p is a proper prefix of w. Show that $\prec$ is a strict partial order.

c) Are the two order relations $\preceq$ and $\prec$ total?

Make sure you write complete proofs for the properties of the order relations. Do not assume something is 'obvious' or 'trivial' — always reason with the definition of the order relation.

Problem 4.2: *function composition* (2+1+1 = 4 points)

Let A, B and C be sets and let $f : A \to B$ and $g : B \to C$ be two functions.

a) Prove the following statement: If $g \circ f$ is bijective, then f is injective and g is surjective.

b) Find an example demonstrating that $g \circ f$ is not bijective even though f is injective and g is surjective.

c) Find an example demonstrating that $g \circ f$ is bijective even though f is not surjective and g is not injective.

Problem 4.3: *prime numbers with a fixed prime gap (haskell)* (1 point)

We call the difference between two successive prime numbers their *prime gap*. Prime numbers with a prime gap of 2 are called *twin primes* while prime numbers with a prime gap of 4 are called *cousing primes*. Prime numbers with a prime gap of 6 are called *sexy primes*.

The predicate `isPrime` shown below determines whether a number is a prime number of not.

```haskell
isPrime :: Integer -> Bool
isPrime n = null [ x | x <- [2..n `div` 2], n `mod` x == 0]
```

Implement a function `primes` that takes two arguments `a` and `b` and returns the list of all prime numbers in the interval `[a,b]`. With that, implement a function `gappies` that receives three arguments, a prime gap `g`, a lower interval bound `a`, and an upper interval bound `b`. The function returns all prime number pairs with the prime gap `g` in the interval `[a,b]`. By "currying" the first argument of `gappies`, we easily obtain the functions `twins`, `cousins`, and `sexies`.

Some sample results so that you can test your implementation:

```
> twins 1 100
[(3,5),(5,7),(11,13),(17,19),(29,31),(41,43),(59,61),(71,73)]
> cousins 1 100
```

```
[(3,7),(7,11),(13,17),(19,23),(37,41),(43,47),(67,71),(79,83)]
> sexies 100 150
[(101,107),(103,109),(107,113),(131,137)]
```

Below is a template that may serve as a starting point.

```haskell
isPrime :: Integer -> Bool
isPrime n = null [ x | x <- [2..n `div` 2], n `mod` x == 0]

primes :: Integer -> Integer -> [Integer]
primes a b = undefined

gappies :: Integer -> Integer -> Integer -> [(Integer,Integer)]
gappies g a b = undefined

twins   = gappies 2
cousins = gappies 4
sexies  = gappies 6
```