Introduction to Dependently Typed Programming SS 2022

Assignment 5

Deadline for solutions: 25.07.2022

Exercise 1 GCD and Termination (8 Points)

Greatest common divisor $gcd(a, b)$ of two natural positive (!) numbers is inductively defined as $gcd(a - b, b)$ if $a > b$, as $gcd(a, b - a)$ if $b > a$ and as a if $a = b$.

a) Implement *gcd* in Agda using the modules of Iowa Agda library. To that end you will need to design a corresponding termination proof.

Hint: A concise and elegant solution can be obtained by using the *lexicographic order* and a corresponding termination proof `↓-lex` from `termination.agda`. Note, however, that it is only one possible approach.

It is tactically advantageous to define $gcd(n, m)$ for all natural n and m. If n and m are not both 0 the result is well defined, otherwise it is sensible to put $gcd(0, 0) = 0$.

b) Formalize and prove that any divisor of a and b is a divisor of $gcd(a, b)$.

Hint: It is convenient to couch the developments in terms of the relation

```
_divides_ : ∀ (n m : ℕ) → Set
```

expressing the fact that n divides m, and to prove the lemma

```
divides∸ : ∀ (n m k : ℕ) → k divides n → k divides m → k divides (m ∸ n)
```

Exercise 2 Binary Logarithms, Termination and Proof (Ir-)Relevance (9 Points)

a) Implement the following functions that calculate *binary* logarithms of natural numbers and round the results down and up respectively:

```
⌊log₂_⌋ : ℕ → ℕ
⌈log₂_⌉ : ℕ → ℕ
```

So, e.g. $\lfloor log_2 2\rfloor = \lceil log_2 2\rceil = 1$ and $\lfloor log_2 3\rfloor = 1$, $\lceil log_2 3\rceil = 2$. You can assume that $\lfloor log_2 0\rfloor = \lceil log_2 0\rceil = 0$.

Hint: You should implement a helper `⌊log₂_⌋-wf : (n : ℕ) → ↓𝔹 _>_ n → ℕ` from which `⌊log₂_⌋` would be definable by feeding `↓->` `n` as the second argument. Some preparation would be needed, in particular, you will need a function `div2 : (n : ℕ) → ℕ` for integer division by 2 and rudimental properties of it.

Attention: The remaining overhead will critically depend on the definition of `⌊log₂_⌋-wf`. It is thus strongly suggested to implement the following clause

```
⌊log₂ (suc (suc n)) ⌋-wf (pf↓ sn) = suc (⌊log₂ suc (div2 n) ⌋-wf (sn (div2n<sn n)))
```

where `div2<sn n` must be a proof that `div2 n < suc n`.

b) Prove the following monotonicity property:

```
⌊log₂⌋-mono : (n m : ℕ) → (n ≤ m ≡ tt) → ⌊log₂ n ⌋ ≤ ⌊log₂ m ⌋ ≡ tt
```

(to that end you will need to formulate and prove the corresponding property of the underlying helper).

c) Implement the following characteristic properties:

```
⌊log₂⌋-ind₁ : (n : ℕ) → is-even (suc n) ≡ tt
                      → ⌊log₂ (suc n) ⌋ ≡ suc (⌊log₂ (div2 (suc n)) ⌋)
⌊log₂⌋-ind₂ : (n : ℕ) → is-even (suc n) ≡ ff
                      → ⌊log₂ (suc n) ⌋ ≡ ⌊log₂ n ⌋
```

and use them to prove the following:

```
⌊log₂⌋-n+n : ∀ (n : ℕ) → n > 0 ≡ tt → ⌊log₂ (n + n) ⌋ ≡ suc ⌊log₂ n ⌋
⌊log₂⌋-n+n+1 : ∀ (n : ℕ) → n > 0 ≡ tt → ⌊log₂ (suc (n + n)) ⌋ ≡ suc ⌊log₂ n ⌋
```

Hint: It is advisable to prove the folloinwing property of `⌊log₂_⌋-wf` first:

```
⌊log₂⌋-wf-irrel : (n m : ℕ) → {p : ↓𝔹 _>_ n} → {q : ↓𝔹 _>_ m} → n ≡ m
  → ⌊log₂ n ⌋-wf p ≡ ⌊log₂ m ⌋-wf q
```

in order to be able to enforce *proof irrelevance*. Indeed, the result of `⌊log₂ n ⌋-wf p` does not depend on the particular choice of the proof object `p`, however, in proof-relevant environments, this is generally not a property, available by default. You can use the above part **b)** where the analogous inequational property must have been proven.

Exercise 3 Logarithms and Tree Heights (7 Points)

The following function

```
bt-height : ∀ {n : ℕ} → braun-tree n → ℕ
bt-height bt-empty = 0
bt-height (bt-node _ l r _) = suc (max (bt-height l) (bt-height r))
```

calculates the hight of a brown tree. Using the properties of binary logarithms from the previous excercise, prove the following properties in Agda:

```
bt-height-gt : ∀ {n : ℕ} → (n > 0 ≡ tt)
  → (t : braun-tree n) → suc ⌊log₂ n ⌋ ≤ bt-height t ≡ tt
bt-height-lt : ∀ {n : ℕ}
  → (t : braun-tree n) → bt-height t ≤ suc ⌈log₂ n ⌉ ≡ tt
```

Exercise 4 Kripke Semantics (6 Points)

a) Design a proof of the intuitionistic tautology $\phi = p \to ((p \to q) \to q)$ and verify the proof in Agda by showing that the type $[] \vdash \phi$ is inhabited. Normalize your proof by calling the `nbe` function. Does it result in a different proof?

b) *(Peirce's law)* The formula $((p \to q) \to p) \to p$ is famously known for being a classical tautology, which does not hold intuitionistically. Construct a Kripke model, falsifying it and implement in Agda. That is, you will need to deliver a proof for

```
no-peirce's-law : (k , w0 ⊨
    Implies (Implies (Implies ($ "p") ($ "q")) ($ "p")) ($ "p")) → ⊥
```

for suitable `k` and `w0`.