Types for Programs and Proofs 2018/2019
Exercise Session 2

September 17, 2018

The deadline for registering your talk proposals in the wiki is on *Thursday 20 September*.

Homework 2 is now available. It is due on *Monday 1 October at 13.15*. Please submit it in Fire. You should *be prepared* to present your solutions in the exercise session the same day at 15.15 – 17.00.

1 Exercises

1. The type of the identity function I with implicit polymorphism is

   $\mathsf{I} : \forall \{X : \mathsf{Set}\} \to X \to X$

   Write I using λ-notation.

   $\mathsf{I} = \{!!\}$

2. Define the K- and S-combinators.

   $\mathsf{K} : \forall \{X\ Y : \mathsf{Set}\} \to X \to Y \to X$
   $\mathsf{K} = \{!!\}$

   $\mathsf{S} : \forall \{X\ Y\ Z : \mathsf{Set}\} \to (X \to Y) \to (X \to Y \to Z) \to X \to Z$
   $\mathsf{S} = \{!!\}$

3. Define the I-combinator using only application, K and S.

   $\mathsf{I'} : \forall \{X : \mathsf{Set}\} \to X \to X$
   $\mathsf{I'} = \{!!\}$

4. Prove symmetry and transitivity of propositional identity _≡_ by pattern matching.

   $\mathsf{sym} : \forall \{A\}\ \{a\ b : A\} \to a \equiv b \to b \equiv a$
   $\mathsf{sym} = \{!!\}$

   $\mathsf{trans} : \forall \{A\}\ \{a\ b\ c : A\} \to a \equiv b \to b \equiv c \to a \equiv c$
   $\mathsf{trans} = \{!!\}$

5. Prove symmetry and transitivity without pattern matching by using subst.

   $\mathsf{subst} : \forall \{A\}\ \{P : A \to \mathsf{Set}\}\ \{a\ b\} \to a \equiv b \to P\ a \to P\ b$
   $\mathsf{subst}\ \mathsf{refl}\ p = p$

```
sym' : ∀ {A} {a b : A} → a ≡ b → b ≡ a
sym' = {!!}

trans' : ∀ {A} {a b c : A} → a ≡ b → b ≡ c → a ≡ c
trans' = {!!}
```

6. Prove commutativity of addition _+_.

```
0-neutral-right : ∀ {m} → m + 0 ≡ m
0-neutral-right = {!!}

0-neutral-left : ∀ {n} → 0 + n ≡ n
0-neutral-left = {!!}

succ-+-right : ∀ {m n} → m + succ n ≡ succ (m + n)
succ-+-right = {!!}

succ-+-left : ∀ {m n} → succ m + n ≡ succ (m + n)
succ-+-left = {!!}

comm-+ : ∀ {m n} → m + n ≡ n + m
comm-+ = {!!}
```

7. Prove the commutativity, idempotency and neutrality laws for conjunction _∧_.

```
comm-∧ : ∀ {A B} → A ∧ B ≈ B ∧ A
comm-∧ = {!!}

idem-∧ : ∀ {A} → A ∧ A ≈ A
idem-∧ = {!!}

neutral-∧ : ∀ {A} → A ∧ ⊤ ≈ A
neutral-∧ = {!!}
```

8. Prove the involution law for negation, assuming you can decide whether a type is empty (the false proposition) or inhabited (the true proposition).

```
-- Homework 2, Problem 2 b)
postulate L : ∀ {A} → A ⊔ (A → Empty)

invol-¬ : ∀ {A} → ¬ ¬ A ≈ A
invol-¬ = {!!}
```

9. Find a Kripke structure[?, Chapter 10] k where negation is *not* involutive, i. e. there is a formula f and a world w such that the instance of the law fails there.

```
record KripkeStructure : Set₁ where
  field
    -- a set of worlds
    W      : Set
    -- a reflexive and transitive relation on W
```

```
    R      : W → W → Set
    reflR  : ∀ (w : W) → R w w
    transR : ∀ {w v u : W} → R w v → R v u → R w u
    -- a valuation telling whether atomic formula i is true or false in a given
    -- world
    V      : W → Nat → Set
    monoV  : ∀ {w v : W} → R w v → ∀ (i : Nat) → V w i → V v i
open KripkeStructure

_,_⊨k_ : ∀ (k : KripkeStructure) → W k → Formula → Set
k , w ⊨k $ x          = V k w x
k , w ⊨k True         = Unit
k , w ⊨k False        = Empty
k , w ⊨k Implies f1 f2 = ∀ {v : W k} → R k w v → k , v ⊨k f1 → k , v ⊨k f2
k , w ⊨k And f1 f2    = k , w ⊨k f1 × k , w ⊨k f2
k , w ⊨k Or f1 f2     = k , w ⊨k f1 ⊔ k , w ⊨k f2


not-invol-Not : (k , w ⊨k Implies (Not (Not f)) f) → Empty
not-invol-Not = {!!}
```

References

[1] Aaron Stump, *Verified Functional Programming in Agda*, Association for Computing Machinery and Morgan & Claypool Publishers, 2016.

[2] Peter Dybjer, *An Introduction to Programming and Proving in Agda*, 2017.

[3] Agda contributors, *The Agda User Manual*, `https://agda.readthedocs.io/en/latest/`, 2017.