

Exam Logical Verification

June 25, 2010

There are six (6) exercises.

Answers may be given in Dutch or English. Good luck!

Exercise 1. This exercise is concerned with first-order propositional logic (prop1) and simply typed λ -calculus ($\lambda \rightarrow$).

- a. Show that the formula $((A \rightarrow B) \rightarrow C) \rightarrow B \rightarrow C$ is a tautology of minimal prop1.

(5 points)

- b. Give the type derivation in simply typed λ -calculus corresponding to the proof of 1a.

(5 points)

- c. Give three different closed inhabitants in $\lambda \rightarrow$ of the following type:

$$B \rightarrow (B \rightarrow A) \rightarrow (B \rightarrow B) \rightarrow A$$

(5 points)

- d. Replace in the following three terms the ?'s by simple types, such that we obtain typable λ -terms. (NB: it is not asked to give the type derivations.)

$$\lambda x : ?. \lambda y : ?. \lambda z : ?. (x z) (y z)$$

$$\lambda x : ?. \lambda y : ?. x (x y)$$

$$\lambda x : ?. \lambda y : ?. \lambda z : ?. (y x) (x z)$$

(5 points)

Exercise 2. This exercise is concerned with first-order predicate logic (`pred1`) and λ -calculus with dependent types (λP).

- a. What is the type checking problem? Is it decidable for λP ?
What is the type inhabitation problem? Is it decidable for λP ?
(5 points)
- b. Give the λP -term corresponding to the formula

$$\forall x. (P(x) \rightarrow (\forall y. P(y) \rightarrow A) \rightarrow A)$$

(5 points)

- c. Give a closed inhabitant in λP of the answer to 2b.
(5 points)

Exercise 3. This exercise is concerned with second-order propositional logic (`prop2`) and polymorphic λ -calculus ($\lambda 2$).

- a. Show that the following formula is a tautology of `prop2`:

$$\forall a. (\forall c. (a \rightarrow c) \rightarrow c) \rightarrow a$$

(5 points)

- b. Give the $\lambda 2$ type corresponding to the formula of 3a.
(5 points)
- c. Give a closed inhabitant of the type found in (3b.).
(5 points)

Exercise 4. This exercise is concerned with encodings.

- a. Give an impredicative definition of *false* in `prop2`, call it *false*.
(3 points)
- b. Show that $\forall c. \text{false} \rightarrow c$ is a tautology of `prop2`.
(with *false* your answer to 4a).
(3 points)
- c. Give the type in $\lambda 2$ corresponding to your definition of *false*.
(3 points)
- d. Give the type in $\lambda 2$ corresponding to the formula $\forall c. \text{false} \rightarrow c$.
(3 points)

- e. Give a closed inhabitant in $\lambda 2$ of the type given as answer to 4d.
(3 points)
- f. First-order propositional logic can be encoded in Coq using dependent types as follows:

```
(* prop representing the propositions is declared as a Set *)
Variable prop:Set.
(* T expresses if a proposition in prop is valid
   if (T p) is inhabited then p is valid
   if (T p) is not inhabited then p is not valid *)
Variable T: prop -> Prop.
(* conjunction is a binary operator represented by conj *)
Variable conj : prop -> prop -> prop .
```

Give the types of the variables

```
conj_intro
conj_elim_l
conj_elim_r
```

modeling introduction of conjunction, and left- and right elimination of conjunction.

(5 points)

Exercise 5. This definition is concerned with inductive data-types in Coq.

- a. Consider the following inductive of the data-type for binary trees:

```
Inductive tree : Set :=
| leaf : tree
| node : tree -> tree -> tree.
```

Give the induction principle that belongs to this type (in Coq notation).

(5 points)

- b. Give an inductive definition of the data-type binary trees with nodes labelled with natural numbers.

(5 points)

Exercise 6. This exercise is concerned with inductive predicates in Coq.

- a. Consider the inductive predicate for less-than-equal in Coq:

```
Inductive le (n:nat) : nat -> Prop :=
| le_n : le n n
| le_S : forall m:nat , le n m -> le n (S m) .
```

Give if possible an inhabitant of the following, if it is not possible explain shortly why not:

```
le 0 (S 0)
le (S 0) 0
```

(5 points)

- b. Give the definition in Coq of an inductive predicate `tre` on natural numbers that holds exactly if the number can be divided by 3.

(The constructors for the natural numbers are `0 : nat` and `S : nat -> nat`.)

(5 points)

The final note is (the total amount of points plus 10) divided by 10.