

Exam Logical Verification

August 22, 2011

There are six (6) exercises.

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

Exercise 1. (*5+5+4+4 points*)

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

- Show that the following formula is a tautology of minimal **prop1**:
 $((B \rightarrow A \rightarrow B) \rightarrow A) \rightarrow A$.
- Give the type derivation in $\lambda\rightarrow$ corresponding to the proof of 1a.
- Give closed inhabitants in normal form of the following two types
(NB: it is not asked to give the type derivations):

$$\begin{aligned} &A \rightarrow B \rightarrow A \\ &(A \rightarrow A \rightarrow B) \rightarrow (A \rightarrow B) \end{aligned}$$

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

$$\begin{aligned} &\lambda y:?. \lambda z:?. z (y z) \\ &\lambda x:?. \lambda y:?. \lambda z:?. x z (y z) \end{aligned}$$

Exercise 2. (*5+3+5 points*)

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

- Show that the following formula is a tautology of minimal **pred1**:
 $(A \rightarrow \forall x. P(x)) \rightarrow (\forall y. A \rightarrow P(y))$.
- Give the λP -term corresponding to the formula in 2a.
- Give a closed inhabitant in λP of the answer to 2b.

Exercise 3. (5+3+5 points)

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

- Show that the following formula is a tautology of minimal `prop2`:
 $a \rightarrow \forall c. ((a \rightarrow c) \rightarrow (b \rightarrow c) \rightarrow c)$.
- Give the $\lambda 2$ -term corresponding to the formula in 3a.
- Give a closed inhabitant in $\lambda 2$ of the answer to 3b.

Exercise 4. (5+5+5 points)

This exercise is concerned with different representations of conjunction.

- First-order propositional logic can be encoded in Coq using dependent types as follows:

```
(* prop representing the propositions is a Set *)
Variable prop:Set.
(* conjunction on prop is a binary operator *)
Parameter conjunction : prop -> prop -> prop.
(* 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.
```

Give the type of the variable `conjunction_introduction` modelling the introduction rule of conjunction.

- The inductive definition of conjunction in Coq is as follows:

```
Inductive and (A : Prop) (B : Prop) : Prop :=
  conj : A -> B -> A /\ B.
```

Explain how the constructor `conj` is used to model the introduction rule of conjunction.

- The impredicative definition of conjunction in `prop2` is as follows:

`new_and A B = $\forall c. ((A \rightarrow B \rightarrow c) \rightarrow c)$.`

Show `(new_and A B) \rightarrow A` in `prop2`.

(Unfold the definition of `new_and` in your proof.)

Exercise 5. (*4+5+5+4 points*)

This exercise is concerned with inductive data-types in Coq.

- a. Give the definition of an inductive data-type `natpair` of pairs of natural numbers (with `nat` the type of natural numbers).
- b. Give the induction principle for `natpair`.
- c. Give the definition of a function that takes as input a `natpair` and gives as output the sum of the two elements.
(You can use the built-in addition of natural numbers in Coq.)
- d. Consider the inductive data-type for dependent natlists:

```
Inductive natlist_dep : nat -> Set :=
  | nil_dep : natlist_dep 0
  | cons_dep : forall n : nat,
    nat -> natlist_dep n -> natlist_dep (S n).
```

Give the type (NB: the definition is not asked) of the function that appends two dependent natlists.

Exercise 6. (*4+4+5 points*)

This exercise is concerned with inductive predicates in Coq. We use 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).
```

- a. What are the types of the constructors `le_n` and `le_S`?
- b. 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
```

- c. Give a definition of an inductive predicate `sorted` on elements of `natlist`, using `le` for sorting. The inductive definition of `natlist` is as follows:

```
Inductive natlist : Set :=
  | nil : natlist
  | cons : nat -> natlist -> natlist.
```

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