

# Exam Logical Verification

February 8, 2006

**There are six (6) exercises.**

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

**Exercise 1.** This exercise is concerned with first-order minimal propositional logic and simply typed  $\lambda$ -calculus.

- a. Show that  $(B \rightarrow (A \rightarrow B) \rightarrow C) \rightarrow B \rightarrow C$  is a tautology.  
(5 points)
- b. Give the type derivation in simply typed  $\lambda$ -calculus corresponding to the proof of 1a.  
(5 points)
- c. Replace in the following three terms the ?'s by simple types, such that we obtain typable  $\lambda$ -terms.  
 $\lambda x:?. \lambda y:?. \lambda z:?. y (\lambda u:?. x)$   
 $\lambda x:?. \lambda y:?. x (x y)$   
 $\lambda x:?. \lambda y:?. \lambda z:?. z ((\lambda u:?. y) x)$   
(5 points)

**Exercise 2.** This exercise is concerned with inductive definitions in Coq.

- a. Consider the definition of `natlist` for lists of natural numbers:

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

Give the type of `natlist_ind`, which is used to give proofs by induction.  
(5 points)

- b. Give the definition of an inductive predicate `last_element` such that `(last n l)` means that `n` is the last element of `l`.  
(5 points)

c. What is expressed by the following predicate:

```
Inductive remove_last : nat -> natlist -> natlist -> Prop :=
| remove_last_h :
  forall n:nat, remove_last n (cons n nil) nil
| remove_last_t :
  forall (n m:nat) (k l:natlist),
    remove_last n k l -> remove_last n (cons m k) (cons m l).
```

(5 points)

(d) Give a definition of a predicate `palindrome` on natlists expressing that a natlist is the same as its reverse. You may use `remove_last`.

(5 points)

**Exercise 3.** This exercise is concerned with first-order predicate logic.

a. Give an example of a proof that is incorrect because the side-condition for the introduction rule for  $\forall$  is violated.

(5 points)

b. Show that  $(\forall x. \neg P(x)) \rightarrow \neg(\exists x. P(x))$  is a tautology.

(5 points)

**Exercise 4.** This exercise is concerned with dependent types. We use the following definition in Coq:

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

a. What is the type of `natlist_dep 2`? Describe the elements of `natlist_dep 2`.

(5 points)

b. Describe the inputs and the output of `append_dep`, the function that appends two dependent lists.

(5 points)

c. Suppose we want to define a function `nth` that takes as input a list and gives back the  $n$ th element of that list. How can dependent lists be used to avoid errors?

(5 points)

**Exercise 5.** This exercise is concerned with the Curry–Howard–de Bruijn isomorphism.

- a. What is the type checking problem and what is the corresponding problem in logic?  
(5 points)
- b. What is the type of the function that can be extracted from the proof of the following theorem:

```
forall l : natlist,  
{l' : natlist | Permutation l l' /\ Sorted l'}.
```

(5 points)

**Exercise 6.** This exercise is concerned with second-order propositional logic and polymorphic  $\lambda$ -calculus ( $\lambda 2$ ).

- a. Show that  $\forall a : *. ((\forall b : *. b) \rightarrow a)$  is a tautology.  
(5 points)
- b. Give the  $\lambda 2$ -term corresponding to the formula  $\forall a : *. ((\forall b : *. b) \rightarrow a)$ .  
(5 points)
- c. Give a  $\lambda 2$ -term that is an inhabitant of the answer to 6b.  
(5 points)
- d. Give a  $\lambda 2$ -derivation of  $\vdash \Pi b : *. b : *$ .  
(5 points)

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

appendix to the exam logical verification 2005 - 2006

**Typing rules of the simply typed lambda calculus.**

The environment is a finite set of declarations.

$$\begin{array}{l}
 \text{variable} \quad \overline{\Gamma, x : A \vdash x : A} \\
 \\
 \text{abstraction} \quad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash (\lambda x:A. M) : A \rightarrow B} \\
 \\
 \text{application} \quad \frac{\Gamma \vdash F : A \rightarrow B \quad \Gamma \vdash N : A}{\Gamma \vdash (F N) : B}
 \end{array}$$

**Typing rules in the style of pure type systems (PTSs).**

In these rules the variable  $s$  ranges over the set of sorts  $\{*, \square\}$ .

The environment is a finite list of declarations.

$$\begin{array}{l}
 \text{start} \quad \overline{\vdash * : \square} \\
 \\
 \text{variable} \quad \frac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A} \\
 \\
 \text{weakening} \quad \frac{\Gamma \vdash A : B \quad \Gamma \vdash C : s}{\Gamma, x : C \vdash A : B} \\
 \\
 \text{product } (\lambda \rightarrow) \quad \frac{\Gamma \vdash A : * \quad \Gamma, x : A \vdash B : *}{\Gamma \vdash \Pi x:A. B : *} \\
 \\
 \text{product } (\lambda P) \quad \frac{\Gamma \vdash A : * \quad \Gamma, x : A \vdash B : s}{\Gamma \vdash \Pi x:A. B : s} \\
 \\
 \text{product } (\lambda 2) \quad \frac{\Gamma \vdash A : s \quad \Gamma, x : A \vdash B : *}{\Gamma \vdash \Pi x:A. B : *} \\
 \\
 \text{abstraction} \quad \frac{\Gamma, x : A \vdash M : B \quad \Gamma \vdash \Pi x:A. B : s}{\Gamma \vdash \lambda x:A. M : \Pi x:A. B} \\
 \\
 \text{application} \quad \frac{\Gamma \vdash F : \Pi x:A. B \quad \Gamma \vdash M : A}{\Gamma \vdash (F M) : B[x := M]} \\
 \\
 \text{conversion} \quad \frac{\Gamma \vdash A : B \quad \Gamma \vdash B' : s}{\Gamma \vdash A : B'} \quad \text{with } B =_{\beta} B'
 \end{array}$$