# The University of Nottingham

SCHOOL OF COMPUTER SCIENCE

A LEVEL 2 MODULE, FULL YEAR, 2022-2023

**ALGORITHMS CORRECTNESS AND EFFICIENCY**

Time allowed TWO HOURS

---

*Candidates may complete the front cover of their answer book and sign their desk card but must NOT write anything else until the start of the examination period is announced*

***Answer ALL FOUR questions***

*No calculators are permitted in this examination.*

*Dictionaries are not allowed with one exception. Those whose first language is not English may use a dictionary to translate between that language and English provided that neither language is the subject of this examination.*

*No electronic devices capable of storing and retrieving text may be used.*

**DO NOT turn examination paper over until instructed to do so**

**INFORMATION FOR INVIGILATORS:**

*Students should write their answers to these questions in the answer book. The exam paper should be collected and placed inside the answer book.*

**Question 1:** This question is concerned with propositional logic in Lean. [25 marks total]

a. How are the propositional connectives $\leftrightarrow$ (if and only if) and $\neg$ (negation) defined in Lean? [4 marks]

b. What are the deMorgan laws? Which part is not provable in intiutionistic logic? [6 marks]

c. Which of the following are propositional tautologies in Lean? (without using classical logic) ?

(i) $P \to (Q \land R) \leftrightarrow (P \to Q) \land (P \to R)$

(ii) $(P \land Q) \to R \leftrightarrow (P \to R) \land (Q \to R)$

(iii) $P \to (Q \lor R) \leftrightarrow (P \to Q) \lor (P \to R)$

(iv) $(P \lor Q) \to R) \leftrightarrow (P \to R) \lor (Q \to R)$

(v) $(P \lor Q) \to R) \leftrightarrow (P \to R) \land (Q \to R)$

[10 marks]

d. In intuitionistic logic the principle "reductio ad absurdum" RAA, $\neg\neg P \to P$, is not provable in general but it is provable for negated propositions, that is we can show $\neg\neg\neg P \to \neg P$. How would you prove this in Lean?

[5 marks]

**Question 2:** This question is concerned with predicate logic in Lean. [25 marks total]

a. How do you prove an equality of the form $a = a$ in Lean? How do you use an assumption of the form $h : a = b$?

[5 marks]

b. Given a type of `People` and a predicate `Loves`, where `Loves x y` means x loves y.

```
variable People : Type
variable Loves : People → People → Prop
```

Translate the following English expressions into predicate logic using Lean syntax:

(i) Everybody loves somebody.
(ii) Somebody is loved by everybody.
(iii) Love isn't transitive.
(iv) There are people who don't love anybody.
(v) Love isn't symmetric.

[15 marks]

c. How do you specify that `A : Type` is non-empty? How do you specify that it is empty?

[5 marks]

**Question 3:** This question is concerned with reasoning about booleans and natural numbers in Lean.

[25 marks total]

a. Define a function

```
implb : bool → bool → bool
```

such that

```
∀ x y : bool,
(x = tt) → (y=tt) ↔ implb x y = tt
```

(you don't have to prove it).

[5 marks]

b. Which of the following propositions about booleans are provable in Lean? For which ones can we prove their negation? Can it happen that we cannot prove a proposition about booleans nor their negation?

(i) `∀ x : bool, ∃ y:bool, x≠y`
(ii) `∃ x : bool, ∀ y:bool, x≠y`
(iii) `∀ x : bool, ∃ y:bool, x=y`
(iv) `∃ x : bool, ∀ y:bool, x=y`

[10 marks]

c. We define a function by recursion over the natural numbers:

```
def foo : ℕ → ℕ
| zero := 1
| (succ zero) := 0
| (succ (succ n)) := succ (succ (foo n))
```

What are the values of `foo 4` and `foo 5`?
Which of the following properties hold?

(i) `foo` is injective.
`∀ x y : ℕ, foo x = foo y → x=y`
(ii) `foo` is surjective.
`∀ y : ℕ, ∃ x : ℕ , foo x = y`
(iii) `foo` has a fixpoint.
`∃ x:ℕ, foo x = x`

[10 marks]

**Question 4:** This question is concerned with reasoning about lists and trees in Lean. [25 marks total]

a. What is the definition of `list` as an inductive type in Lean?

[4 marks]

b. We define append as follows:

```
definition append : list A → list A → list A
| []       l := l
| (h :: s) t := h :: (append s t)

local notation l1 ++ l2 := append l1 l2
```

Which of the following propositions about ++ hold?

(i) `∀ l : list A, [] ++ l = l`
(ii) `∀ l m : list A, l++m = m++l`
(iii) `∃ l : list A, l++l = l`
(iv) `∀ l m1 m2 : list A,  l ++ m1 = l ++ m2 → m1 = m2`

[8 marks]

c. We define trees whose leaves are labelled with natural numbers:

```
inductive Tree : Type
| leaf : ℕ → Tree
| node : Tree → Tree → Tree
```

An example is

```
def t1 : Tree
:= node (node (leaf 1) (leaf 2)) (leaf 3)
```

Define a function `tree2list : Tree → list ℕ` which collects all the leaves in a list. E.g. `tree2list t1 = [1,2,3]`. [8 marks]

d. Given the following definition of permutation of lists in Lean:

```
inductive Insert : A → list A → list A → Prop
| ins_hd : ∀ a:A, ∀ as : list A,Insert a as (a :: as)
| ins_tl : ∀ a b:A, ∀ as as': list A, Insert a as as'
      → Insert a (b :: as) (b :: as')

inductive Perm : list A → list A → Prop
| perm_nil : Perm [] []
| perm_cons : ∀ a : A, ∀ as bs bs' : list A,
    Perm as bs → Insert a bs bs' → Perm (a :: as) bs'
```

How do you prove `Perm [1,2] [2,1]`?

[5 marks]