

Regular Examination in the Course Interactive Theorem Proving

You have **120 minutes** at your disposal. Written or electronic aids are not permitted except for normal watches. Carrying forbidden devices, even switched off, will be considered a cheating attempt.

Write your full name and matriculation number clearly legible on this cover sheet, as well as your name in the header on each sheet. Hand in all sheets. Leave them stapled together. Use only **pens** and **neither** the color **red nor green**.

Check that you have received all the sheets. Guidelines for writing pen-and-paper proofs are given on **page 2**. Questions can be found on **pages 3–17**. There are 7 questions for a total of 100 points. You may use the back of the sheets for secondary calculations. If you use the back of a sheet to answer, clearly mark what belongs to which question and indicate in the corresponding question where all parts of your answer can be found. Cross out everything that should not be graded.

With your signature, you confirm that you are sufficiently healthy at the beginning of the examination and that you accept the examination bindingly.

Last name (in CAPITAL LETTERS):

First name (in CAPITAL LETTERS):

Matriculation number:

Program of study:

Hierby I confirm the correctness of the above information:

Signature

Please leave the following table blank:

Question	1	2	3	4	5	6	7	Σ
Points	23	14	16	13	16	7	11	100
Score								

Guidelines for Paper Proofs

We expect detailed, rigorous, mathematical proofs, but we do not ask you to write Lean proofs. You are welcome to use standard mathematical notation or Lean structured commands (e.g., **assume**, **have**, **show**, **calc**). You can also use tactical proofs (e.g., **intro**, **apply**), but then please indicate some of the intermediate goals, so that we can follow the chain of reasoning.

Major proof steps, including applications of induction and invocation of the induction hypothesis, must be stated explicitly. For each case of a proof by induction, you must list the induction hypotheses assumed (if any) and the goal to be proved. Minor proof steps corresponding to **refl**, **simp**, or **linarith** need not be justified if you think they are obvious, but you should mention which key theorems they depend on. You should be explicit whenever you use a function definition or an introduction rule for an inductive predicate.

Question 1 (Types and Terms):**(23 points)**

a) [14 points] Recall the following simplified typing rules for Lean's dependent type theory:

$$\begin{array}{c}
 \frac{}{C \vdash c : \sigma} \text{CST} \quad \text{if } c \text{ is globally declared with type } \sigma \\
 \\
 \frac{}{C \vdash x : \sigma} \text{VAR} \quad \text{if } x : \sigma \text{ is the rightmost occurrence of } x \text{ in } C \\
 \\
 \frac{C \vdash t : (x : \sigma) \rightarrow \tau[x] \quad C \vdash u : \sigma}{C \vdash t u : \tau[u]} \text{APP}' \\
 \\
 \frac{C, x : \sigma \vdash t : \tau[x]}{C \vdash (\text{fun } x : \sigma \mapsto t) : (x : \sigma) \rightarrow \tau[x]} \text{FUN}'
 \end{array}$$

Let $a : \mathbb{N}$, $b : \mathbb{N}$, $c : \mathbb{N}$, $P : \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbf{Prop}$, and $\text{Nat.add} : \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N}$ be globally declared constants. What is the type of the following two Lean terms? Give in each case a typing derivation as justification for the type.

(i) $P (\text{Nat.add } a \ b) \ c$

(ii) $\text{fun } (n : \mathbb{N}) \mapsto \text{fun } (m : \mathbb{N}) \mapsto n$

b) [9 points] Let α , β , and γ be Lean types. Give an inhabitant for each of the following types:

(i) $(\alpha \rightarrow \beta \rightarrow \gamma) \rightarrow \beta \rightarrow \alpha \rightarrow \gamma$

(ii) $(\alpha \rightarrow \beta) \rightarrow \alpha \rightarrow (\beta \rightarrow \gamma) \rightarrow (\alpha \rightarrow \gamma)$

(iii) $\alpha \rightarrow \mathbb{N}$

Question 2 (Functional Programming):**(14 points)**

a) [8 points] Consider the following Lean function definition:

```
def replicate {α : Type} (a : α) : ℕ → List α
| 0 => []
| Nat.succ n => a :: replicate a n
```

Prove the following Lean theorem. Make sure to follow the proof guidelines given on page 2.

```
theorem replicate_length {α : Type} (a : α) (n : ℕ) :
  List.length (replicate a n) = n
```

- b) [6 points] Define a polymorphic Lean function `constantSquareMatrix` that replicates a value in two dimensions. For example, `constantSquareMatrix 1 5` should evaluate to `[[5]]` and `constantSquareMatrix 3 "a"` should evaluate to `[[ "a", "a", "a"], ["a", "a", "a"], ["a", "a", "a"]]`.

Question 3 (Inductive Predicates):**(16 points)**

- a) [6 points] Consider the following Lean inductive predicate, which determines whether a list is decreasing with a difference of 2 between consecutive items:

```
inductive IsDiffTwoSeq: List ℕ → Prop
| nil : IsDiffTwoSeq []
| sing (n : ℕ) : IsDiffTwoSeq [n]
| add_two (a : ℕ) (l : List ℕ) (h : IsDiffTwoSeq (a :: l)) :
  IsDiffTwoSeq ((a + 2) :: a :: l)
```

For example, `IsDiffTwoSeq [14, 12, 10]` should hold, whereas `IsDiffTwoSeq [3, 2, 1]` should not hold.

Prove the following Lean theorem about `IsDiffTwoSeq`. If you want to use the inductive rules for defining `IsDiffTwoSeq`, write down the concrete values `n`, `a`, and `l` to be instantiated to the variables. Make sure to follow the proof guidelines given on page 2.

```
theorem IsDiffTwoSequence_example : IsDiffTwoSeq [122, 120]
```

- b)** [10 points] State the following propositions in Lean. Indicate whether they are theorems or not.
- (i) If two lists both satisfy the predicate `IsDiffTwoSeq`, then concatenating them produces a list satisfying the predicate `IsDiffTwoSeq`.
 - (ii) Reversing a list satisfying the predicate `IsDiffTwoSeq` produces a list satisfying the predicate `IsDiffTwoSeq`. You can use the `List.reverse` function.

In each case, give a brief proof or counterexample.

Question 4 (Effectful Programming):**(13 points)**

During the execution of a program, we may want to gather all the error messages together to print them all at the very end. The *console* monad records all the error messages along the whole execution. In Lean, the console monad can be defined as follows:

```
def Console ( $\alpha$  : Type) : Type :=  $\alpha \times \text{String}$ 
def Console.value { $\alpha$  : Type} : Console  $\alpha \rightarrow \alpha$  := Prod.fst
def Console.message { $\alpha$  : Type} : Console  $\alpha \rightarrow \text{String}$  := Prod.snd

def Console.pure { $\alpha$  : Type} :  $\alpha \rightarrow \text{Console } \alpha$  := fun a  $\mapsto$  (a, "")
def Console.bind { $\alpha$   $\beta$  : Type} : Console  $\alpha \rightarrow (\alpha \rightarrow \text{Console } \beta) \rightarrow \text{Console } \beta$  :=
  fun ma f  $\mapsto$ 
    let mb := f (Console.value ma)
    (Console.value mb, Console.message ma ++ Console.message mb)

instance Console.Pure : Pure Console :=
  { pure := Console.pure }

instance Console.Bind : Bind Console :=
  { bind := Console.bind }
```

- a) [6 points] Prove the following law about the console monad. Make sure to follow the proof guidelines given on page 2. In addition, show all the steps when unfolding the definition of monad operators.

```
lemma Console.pure_bind { $\alpha$   $\beta$  : Type} (a :  $\alpha$ ) (f :  $\alpha \rightarrow \text{Console } \beta$ ) :
  Console.pure a >>= f = f a
```

b) [3 points] We can define a function in Lean

```
def Console.log {α : Type} : Console α → String → Console α :=  
  fun (a, m) m' ↦ (a, m ++ m')
```

that adds a message after a value in `Console α`. Evaluate the following expression:

```
Console.bind (Console.pure 3)  
  (fun a ↦ Console.log (Console.pure a) "multiplication by 7 detected")
```

- c) [4 points] Commutative monads are monads for which we can reorder actions that do not depend on each other. For the console monad, the property can be stated as follows:

```
theorem Console.bind_comm {α β γ δ : Type} (ma : Console α)
  (f : α → Console β) (g : α → Console γ) (h : α → β → γ → Console δ) :
  ma >>= (fun a ↦ f a >>= fun b ↦ g a >>= fun c ↦ h a b c) =
  ma >>= (fun a ↦ g a >>= fun c ↦ f a >>= fun b ↦ h a b c)
```

Is **Console** a commutative monad? If yes, provide a justification (as text or in the form of a brief proof sketch). If no, provide a counterexample (i.e., concrete values for **ma**, **f**, **g**, and **h** above and the result of evaluating both sides of the equality).

Question 5 (Operational Semantics):**(16 points)**

The INTER programming language is similar to the familiar WHILE language, with the **while-do** statement replaced by an **interleave** statement. The **interleave** statement takes a natural number **n** and two statements **S** and **T**, and performs **n** iterations, alternating **S** and **T**. For example, the INTER program

```
interleave 3 (j := j + 1) (i := i + 2)
```

adds 1 to **j**, adds 2 to **i**, and adds 1 to **j** again.

In Lean, the INTER syntax is modeled abstractly by the following datatype:

```
inductive Stmt : Type where
  | skip      : Stmt
  | assign    : String → (State → ℕ) → Stmt
  | seq       : Stmt → Stmt → Stmt
  | ifThen    : (State → Prop) → Stmt → Stmt
  | interleave : ℕ → Stmt → Stmt → Stmt
```

```
infixr:90 "; " => Stmt.seq
```

- a) [8 points] Complete the following specification of a small-step semantics for INTER by giving the missing derivation rules for **interleave**.

$$\frac{}{(x := a, s) \Rightarrow (\text{skip}, s[x \mapsto s(a)])} \text{ASSIGN}$$

$$\frac{(S, s) \Rightarrow (S', s')}{(S; T, s) \Rightarrow (S'; T, s')} \text{SEQ-STEP} \qquad \frac{}{(\text{skip}; T, s) \Rightarrow (T, s)} \text{SEQ-SKIP}$$

$$\frac{}{(\text{ifThen } B \text{ } S, s) \Rightarrow (S, s)} \text{IF-TRUE} \quad \text{if } s(B) \text{ is true}$$

$$\frac{}{(\text{ifThen } B \text{ } S, s) \Rightarrow (\text{skip}, s)} \text{IF-FALSE} \quad \text{if } s(B) \text{ is false}$$

b) [8 points] Recall that for the inductive predicate

inductive SmallStep : Stmt × State → Stmt × State → Prop where

the rule for sequential composition (;) is encoded in Lean as

```
| seq_step (S S' T s s') (hS : SmallStep (S, s) (S', s')) :  
    SmallStep (S; T, s) (S'; T, s')  
| seq_skip (T s) :  
    SmallStep (Stmt.skip; T, s) (T, s)
```

Encode the rules for **ifThen** in a Lean definition of an inductive predicate that encodes the small-step semantics based on subquestion a) above. You are not asked to provide any rules for any other constructor.

Question 6 (Logic):**(7 points)**

One of the following statements is a theorem; the other one is not. Prove the theorem.

$$(\forall y, \exists x, P(x, y)) \rightarrow \exists x, \forall y, P(x, y)$$

$$(\exists x, \forall y, P(x, y)) \rightarrow \forall y, \exists x, P(x, y)$$

In your proof, identify clearly the quantifier reasoning. Make sure to follow the proof guidelines given on page 2.

Question 7 (Foundations):**(11 points)**

a) [4 points] Let $\sigma : \text{Type } 3$ be a Lean type. Give the type of each of the following Lean terms.

$\mathbb{N} \rightarrow \text{Prop}$

$\sigma \rightarrow \mathbb{N}$

`Sort 3`

`fun ( $\alpha : \text{Type}$ )  $\mapsto$  fun ( $\beta : \text{Type}$ )  $\mapsto \beta$`

- b) [7 points] The set of natural numbers less than n can be defined in Lean as a subtype of $\mathbb{N}$:

```
def Finite (n : ℕ) : Type :=  
  { m : ℕ // m < n }
```

We can define an operation on the subtype as follows:

```
def Finite.succ {n : ℕ} (a : Finite n) :  
  Finite (n + 1) :=  
  Subtype.mk (Subtype.val a + 1)  
(by  
  apply Nat.succ_lt_succ  
  simp[Subtype.property a])
```

Inspired by the definition of `Finite.succ`, define the addition on elements of the type `Finite n` for a fixed natural number n . You can use basic facts about arithmetic without proving them. Make sure to follow the proof guidelines given on page 2.

