Institute for Informatics of the Ludwig-Maximilians-Universität München
Prof. Dr. Jasmin Blanchette
Xavier Généreux
Balazs Toth

Summer semester 2024
14.10.2024

Third Examination in the Course

Interactive Theorem Proving

You have **120 minutes** at your disposal. Written or electronic aids are not permitted. Carrying electronic devices, even turned off, will be considered cheating.

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 1**. Questions can be found on **pages 2–15**. There are 6 questions for a total of 100 points.

You may use the back of the sheets for auxiliary calculations. If you use the back for actual answers, 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 in sufficiently good health at the beginning of the examination and that you accept this examination bindingly.

Last name:

First name:

Matriculation number:

Program of study:

□ Please check with an X *only* if the exam should be voided and not graded.

Bitte *nur* ankreuzen, wenn die Klausur entwertet und nicht korrigiert werden soll.

Hierby I confirm the correctness of the above information: _____

Signature

Please leave the following table blank:

Question	1	2	3	4	5	6	$\sum$
Points	20	25	25	8	12	10	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 lemmas 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): **(20 points)**

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

$$\frac{}{\texttt{C} \vdash \texttt{c} : \sigma}\ \textsc{Cst} \quad \text{if } \texttt{c} \text{ is globally declared with type } \sigma$$

$$\frac{}{\texttt{C} \vdash \texttt{x} : \sigma}\ \textsc{Var} \quad \text{if } \texttt{x} : \sigma \text{ is the rightmost occurrence of } \texttt{x} \text{ in } \texttt{C}$$

$$\frac{\texttt{C} \vdash \texttt{t} : \texttt{(x} : \sigma\texttt{)} \to \tau\texttt{[x]} \quad \texttt{C} \vdash \texttt{u} : \sigma}{\texttt{C} \vdash \texttt{t u} : \tau\texttt{[u]}}\ \textsc{App}'$$

$$\frac{\texttt{C, x} : \sigma \vdash \texttt{t} : \tau\texttt{[x]}}{\texttt{C} \vdash \texttt{(fun x} : \sigma \mapsto \texttt{t)} : \texttt{(x} : \sigma\texttt{)} \to \tau\texttt{[x]}}\ \textsc{Fun}'$$

Let $\texttt{a} : \mathbb{N}$, $\texttt{f} : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N} \to \mathbb{N}$, $\texttt{g} : \mathbb{N} \to \mathbb{N}$, and $\texttt{h} : \texttt{(y} : \mathbb{N}\texttt{)} \to \texttt{\{x} : \mathbb{N} \texttt{ // x < 5\}}$ 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) `h a`

(ii) `fun x ↦ f g x`

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

- $\alpha \to \beta \to \beta$

- $(\alpha \to \beta) \to (\alpha \to \alpha \to \alpha) \to \alpha \to \beta$

- $((\alpha \to \alpha \to \beta) \to \alpha) \to \gamma \to \beta \to \alpha$

Question 2 (Functional Programming): (25 points)

a) Consider the following Lean function definition:

```
def stutter {α : Type} : List α → List α
  | []      => []
  | a :: as => a :: a :: stutter as
```

(i) Give the value of `stutter [4, 3, 2]`. (There is no need to provide intermediate steps.)

(ii) Prove the following Lean theorem. Make sure to follow the proof guidelines given on page 1.

```
theorem map_stutter {α β : Type} (f : α → β) (ys : List α) :
  List.map f (stutter ys) = stutter (List.map f ys) :=
```

(iii) Prove the following Lean theorem. Make sure to follow the proof guidelines given on page 1.

```
theorem stutter_snoc {α : Type} (xs : List α) (y : α) :
  stutter (xs ++ [y]) = stutter xs ++ [y, y] :=
```

b) Define a Lean function `singletonify` that takes a list `[`$x_1, \ldots, x_n$`]` and that returns a list of singletons `[[`x_1`], ..., [`x_n`]]`. For example, `singletonify [1, 2, 3, 5, 7]` should evaluate to `[[1], [2], [3], [5], [7]]`.

Question 3 (Inductive Predicates): **(25 points)**

a) Recall the `stutter` function from Question 2:

```
def stutter {α : Type} : List α → List α
  | []      => []
  | a :: as => a :: a :: stutter as
```

Now consider the following Lean inductive predicate, which holds when a list has even length:

```
inductive EvenLength {α : Type} : List α → Prop where
  | nil :
    EvenLength []
  | add_two (x y : α) {xs : List α} :
    EvenLength xs → EvenLength (x :: y :: xs)
```

For example, `EvenLength [1, 2]` holds, whereas `EvenLength [3, 4, 5]` does not hold.

Prove the following Lean theorem about `EvenLength`. Make sure to follow the proof guidelines given on page 1.

```
theorem EvenLength_stutter {α β : Type} (xs : List α)
    (hxs : EvenLength xs) :
  EvenLength (stutter xs)
```

b) Define an inductive predicate `Suffix` in Lean that takes two lists over a polymorphic type α as arguments and that holds when the first list is a suffix of the second. For exemple, `Suffix [1, 2] [1, 2]` and `Suffix [2, 4] [1, 2, 4]` should hold.

c) Define an inductive predicate `EvenPalindrome` in Lean that takes a list over a polymorphic type α as argument and that holds if the list is a palindrome (i.e., if it equals its reverse) and has even length. For example, `EvenPalindrome [1, 2, 2, 1]` should hold, whereas `EvenPalindrome [1, 3, 1]` and `EvenPalindrome [1, 3, 7]` should not hold.

Question 4 (Metaprogramming): **(8 points)**

Consider the following custom Lean tactic:

```
macro "enigma" : tactic =>
  `(tactic| repeat' first
              | assumption
              | intro _
              | apply True.intro
              | apply And.intro
              | apply Iff.intro)
```

a) Briefly explain what the `enigma` tactic does. You may assume that we already know what `assumption`, `intro`, and `apply` does.

b) In the following Lean code fragment, the `enigma` tactic is applied to transform the goal:

```
theorem abbatf (a b : Prop) :
  a → b → b ∧ a ∧ True ∧ False :=
  by
    enigma
```

The proof state before invoking `enigma` is

```
a b : Prop
⊢ a → b → b ∧ a ∧ True ∧ False
```

Give the proof state *after* invoking `enigma`. Make sure to include all subgoals.

Question 5 (Operational Semantics): (12 points)

The IF0 programming language is similar to WHILE, with two differences. First, the `while`–`do` statement is omitted. Second, the `if`–`then`–`else` statement is replaced by `if_zero`–`then`–`else`. For example, the program

```
if_zero m * n then
  x := 0
else
  y := 1
```

executes `x := 0` if the condition `m * n = 0` holds when entering the construct; otherwise, it executes `y := 1`.

In Lean, IF0 is modeled by the following inductive type:

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

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

a) Complete the following specification of a small-step semantics for IF0 in Lean by giving the missing derivation rules for `seq` and `if_zero`.

$$\frac{}{\texttt{(x := a, s)} \Rightarrow \texttt{(skip, s[x} \mapsto \texttt{s(a)])}}\ \textsc{Assign}$$

$$\frac{}{\texttt{(Stmt.skip; T, s)} \Rightarrow \texttt{(T, s)}}\ \textsc{SeqSkip}$$

b) Complete the following Lean definition of an inductive predicate that encodes the small-step semantics you specified in subquestion a) above.

```
inductive SmallStep : Stmt × State → Stmt × State → Prop where
  | assign (x a s) :
    SmallStep (Stmt.assign x a, s) (Stmt.skip, s[x ↦ a s])
  | seq_skip (T s) :
    SmallStep (Stmt.skip; T, s) (T, s)
```

Question 6 (Mathematics): **(10 points)**

a) Let σ `: Type 4` and τ `: Type 2` be Lean types. Give the type of each of the following Lean terms.

`fun (x :` σ`) (y :` σ`) ↦ x`

`Sort 5`

`fun` α `: Type ↦ List (ℕ ×` α`)`

$\sigma \to \tau$

$\tau \to \sigma$

b) We call a `NonUnitalSemiring` a type with addition, multiplication, and a 0 element and where addition is commutative and associative, multiplication is associative and left and right distributive over addition, and 0 is the additive identity.

Complete the following `class` declaration of `NonUnitalSemiring`:

```
universe u

class NonUnitalSemiring (α : Type u) : Type u where
  add           : α → α → α
  mul           :
  zero          : α
  mul_assoc     :
  add_assoc     : ∀a b c, add (add a b) c = add a (add b c)
  left_distrib  :
  right_distrib : ∀a b c, mul (add a b) c = add (mul a c) (mul b c)
  zero_add      : ∀a, add zero a = a
  add_zero      :
  zero_mul      :
  mul_zero      : ∀a, mul a zero = zero
  add_comm      : ∀a b, add a b = add b a
```