Test 2020 resit

Test instruction

Welcome to the resit exam for the course **Logical Verification** (X_400115).

The only permitted tools are scratch paper, pens, pencils.

Books and other reference materials are **not** permitted.

Proof Guidelines

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 even 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 inductive hypotheses assumed (if any) and the goal to be proved. Unless specified otherwise, minor proof steps corresponding to `refl`, `simp`, or `linarith` need not be justified if you think they are obvious, but you should say which key lemmas they depend on.

You should be explicit whenever you use a function definition or an introduction rule for an inductive predicate, especially for functions and predicates that are specific to an exam question.

In Case of Ambiguities or Errors in an Exam Question

The lecturers cannot answer questions during the exam. We strongly recommend that you work things out by yourselves, stating explicitly any ambiguity or error and explaining how you interpret or repair the question. The more explicit you are, the easier it will be for the lecturers to give you points.

In Case of Late Signup

If you have **not** signed up for this exam, you will not receive a result. Through VUnet you can object to the fact that you can no longer sign up after the expiry of the registration deadline (and the fact that you will not receive a result for this exam). Submit your appeal online within one week after the exam. More information can be found at www.vu.nl/intekenen.

Term inhabitation

Question order: Fixed
The following questions ask you to show that a type is inhabited or not inhabited.

Question 1 – 2020 resit 1a – 225452.4.0

Complete the following Lean definitions by supplying arbitrary terms of the expected type, thereby showing that the types are inhabited.

```
constants α β γ : Type

def weidenbach : α → β → β

def sturm : (α → α → γ) → β → α → γ

def waldmann : (α → γ → β) → (α → β) → α → β

def mueller : (γ → γ) → (β → γ) → (α → β) → α → γ
```

Grading instruction

Definitions (Number of points: 6)

1.5 points for each type-correct definition.

Question 2 – 2020 resit 1b – 225461.2.0

Explain briefly why the following definitions cannot be completed. You can for example refer to the typing rules of the simply typed calculus in your justification.

```
def perfect_sturm : (α → β → γ) → α → α → γ

def angry_mueller : (γ → γ) → (β → β) → (α → β) → α → γ
```

Grading instruction

Explanations (Number of points: 4)

2 points for each explanation.

Connective and quantifiers

Question order: Fixed

The following two questions are about basic mastery of logic. As an exception to the proof guidelines given at the beginning of the exam, please provide highly detailed proofs, including steps we would normally regard as obvious.

Question 3 – 2020 resit 2a – 225464.3.2

Give a detailed proof of the following lemma about conjunction and disjunction. Make sure to emphasize and clearly label every step corresponding to the introduction or elimination of a connective.

```
lemma about_conjunction_and_disjunction {p q r : Prop} :
  (p ∨ q → r) → (p ∧ q → r)
```

Grading instruction

Implication introduction (Number of points: 1)

Assume the two hypotheses.

And elimination (Number of points: 2)

Derive p from p ∧ q.

Or introduction (Number of points: 2)

Derive p ∨ q from p.

Implication elimination (Number of points: 1)

Derive r from p ∨ q → r.

Question 4 – 2020 resit 2b – 225466.3.2

Consider the following proposition:

```
∀p q, p ∨ q → p
```

Is this true? If so, prove it. If not, prove its negation.

In either case, give a detailed proof, emphasizing and clearly labeling every step corresponding to the introduction or elimination of a connective or quantifier.

Grading instruction

Falsity of the statement (Number of points: 1)

Note that the statement is false.

Negated statement (Number of points: 2)

Give the negated statement.

Forall elimination (Number of points: 3)

Instantiate the hypothesis with p = false and q = true (or similar).

Remainder of the proof (Number of points: 2)

Use or introduction and implication elimination to finish the proof.

Transitive closure

Question order: Fixed

The following questions are about specifying and reasoning about inductive predicates representing the transitive closure of a relation.

Question 5 – 2020 resit 3a – 225468.3.1

The transitive closure `r^+` of a binary relation `r` over a set `A` can be defined as the smallest relation satisfying these two rules:

(base) for all `a, b ∈ A`, if `(a, b) ∈ r`, then `(a, b) ∈ r^+`;
(step) for all `a, b, c ∈ A`, if `(a, b) ∈ r` and `(b, c) ∈ r^+`, then `(a, c) ∈ r^+`.

Complete the following Lean definition of the transitive closure, in which relations are represented by binary predicates. Your definition should follow the structure of the above mathematical definition.

```
inductive tc {α : Type} (r : α → α → Prop) : α → α → Prop
```

Grading instruction

base constructor (Number of points: 3)

The constructor corresponding to the 'base' rule.

step constructor (Number of points: 3)

The constructor corresponding to the 'step' rule.

Question 6 – 2020 resit 3b – 225469.3.0

The transitive closure can also be defined like this:

```
inductive tc_alt {α : Type} (r : α → α → Prop) : α → α → Prop
| base (a b : α) : r a b → tc_alt a b
| trans (a b c : α) : tc_alt a b → tc_alt b c → tc_alt a c
```

Prove that for any relation `r`, its closure `tc r` is contained in `tc_alt r`:

```
lemma tc_tc_alt {α : Type} {r : α → α → Prop} {a b} :
  tc r a b → tc_alt r a b
```

Grading instruction

Induction (Number of points: 1)

Rule induction on the derivation of `tc r a b`.

Base case (Number of points: 2)

We have `r a b` (1 point) and derive `tc_alt r a b` from `tc_alt.base` (1 point).

Step case (Number of points: 5)

We have `r a x` and `tc r x b` for some `x;` the induction hypothesis is `tc_alt r x b` (3 points). We conclude `tc_alt r a x` (1 point) and `tc_alt r a b` (1 point).

Logical foundations and mathematics

Question order: Fixed
The following questions concern Lean's logical foundations and its applications to mathematics.

Question 7 – 2020 resit 4a – 225471.2.0

What are the types of the following Lean expressions?

```
ℕ
option (list ℕ)
option
(0 : nat)
Prop
```

Grading instruction

Types (Number of points: 5)

1 point for each type.

Question 8 – 2020 resit 4b – 225476.2.1

Consider the equivalence relation `neg_rel` that relates `x : ℤ` with `x` itself and with `-x`.

```
inductive neg_rel : ℤ → ℤ → Prop
| pos (x y : ℤ) : x = y → neg_rel x y
| neg (x y : ℤ) : x = - y → neg_rel x y
```

You may assume the following lemma about negation:

```
lemma int.neg_neg (x : ℤ) :
  - - x = x
```

Prove that `neg_rel` is reflexive and symmetric.

```
lemma neg_rel.refl (x : ℤ) :
  neg_rel x x

lemma neg_rel.symm (x y : ℤ) :
  neg_rel x y → neg_rel y x
```

Grading instruction

Reflexivity (Number of points: 2)

Apply `neg_rel.pos` (1 point), noting that `x = x` by reflexivity (1 point).

Symmetry (Number of points: 5)

Case distinction on the assumption `neg_rel x y` (1 point). Apply `neg_rel.pos` in the first case and `neg_rel.neg` in the second (1 point). Prove `y = x` in the first case (1 point) and `y = - x` in the second (2 points).

Question 9 – 2020 resit 4c – 225480.2.0

Now, we will define our own copy of the natural numbers ℕ as a quotient of the integers ℤ. First, observe that the `neg_rel` relation is transitive (in addition to reflexive and symmetric):

```
lemma neg_rel.trans (x y z : ℤ) : neg_rel x y → neg_rel y z → neg_rel x z
```

We now have all the ingredients to define our new copy of the natural numbers as a quotient:

```
def new_nat.setoid : setoid ℤ :=
{ r     := neg_rel,
  iseqv := and.intro neg_rel.refl (and.intro neg_rel.symm neg_rel.trans) }

def new_nat : Type := quotient new_nat.setoid
```

Dana Hacker claims that `– x` and `x` correspond to the same element in the type `new_nat`—i.e., `– x = x` for all `x : ℤ`. Do you agree? If so, sketch a proof. Otherwise, give a counterexample.

Grading instruction

Quotient soundness (Number of points: 2)

The lemma holds if `– x` and `x` are related by `neg_rel`.

Establish neg_rel (Number of points: 2)

We have `neg_rel (– x) x` by `neg_rel.neg` (1 point) and `– x = – x` (1 point).

Question 10 – 2020 resit 4d – 225485.2.0

Consider the sequence 3, 3.1, 3.14, 3.141, 3.1415, 3.14159, …, where elements contain more and more digits of π. According to the representation of real numbers as Cauchy sequences, which number does the sequence represent?

Grading instruction

Correct number (Number of points: 2)

Monads

Question order: Fixed

The following questions concern monads.

Question 11 – 2020 resit 5a – 225498.2.0

For an arbitrary type σ, we define the reader monad `reader σ : Type → Type`. We may think of the values of `reader σ α` as programs that return an α value but also have access to some input value of type σ. This monad is similar to the state monad, but `reader` programs cannot change the input σ. The type `reader σ α` is defined as follows:

```
def reader (σ α : Type) := σ → α
```

Complete the following Lean definitions of the monad operations `pure` and `bind` for `reader`.

Hint: You can use the type inhabitation procedure to find the answers.

```
def reader.pure {σ α} : α → reader σ α

def reader.bind {σ α β} (ma : reader σ α) (f : α → reader σ β) : reader σ β
```

Grading instruction

reader.pure (Number of points: 2)

reader.bind (Number of points: 3)

Introduce `s : σ` (1 point). Obtain `ma s : α` (1 point). Obtain `f (ma s) s` (1 point).

Question 12 – 2020 resit 5b – 225499.4.0

Assume that `ma >>= f` is notation for `reader.bind ma f`. Prove that your `reader.pure` and `reader.bind` definitions satisfy the following monad laws. Your proofs should be step-by-step calculational, with at most one rewrite rule or definition expansion per step, so that we can clearly see what happens.

```
lemma reader.pure_bind {σ α β} (a : α) (f : α → reader σ β) :
  (reader.pure a >>= f) = f a

lemma reader.bind_pure {σ α} (ma : reader σ α) :
  (ma >>= reader.pure) = ma
```

Grading instruction

reader.pure_bind (Number of points: 4)

Expand the definitions of `reader.pure` and `reader.bind` (2 points). Perform a beta reduction and an eta reduction (2 points).

reader.bind_pure (Number of points: 4)

Expand the definitions of `reader.pure` and `reader.bind` (2 points). Perform a beta reduction and an eta reduction (2 points).

Question 13 – 2020 resit 5c – 226041.1.0

Prove the following equation for any lawful monad `m`. Your proofs should be step-by-step calculational, with at most one rewrite rule or definition expansion per step, so that we can clearly see what happens.

```
lemma pure_bind_pure {m} [monad m] [is_lawful_monad m] {α β} {a : α} {mb : m β} :
  pure a >>= (λ_, mb >>= pure) = mb
```

Grading instruction

pure_bind (Number of points: 2)

Application of the `pure_bind` law.

bind_pure (Number of points: 2)

Application of the `bind_pure` law.

Glue (Number of points: 1)

Combine the previous two proof steps.

Run-length encoding

Question order: Fixed

In computer science, the run-length encoding is used to compress lists with many repetitions. For example, "AABBB" is coded as "2A3B". The following Lean type can be used to store run-length encoded lists:

```
inductive rle (α : Type)
| empty : rle
| ncons : ℕ → α → rle → rle
```

This type is similar to the standard `list` type, except that `ncons` also includes a repetition count. The `empty` constructor represents a list with no elements. The `ncons` constructor stores a repetition count, an element, and the remainder of the list. Thus, we can encode "AABBB" as `ncons 2 'A' (ncons 3 'B' empty)`.

Question 14 – 2020 resit 6a – 225500.3.0

Define a Lean function called `is_ncons` that tests whether `rle` is of the form `ncons …`.

```
def is_ncons {α : Type} : rle α → bool
```

(To see the definition of `rle` again, use the "Show block intro" button on the right above.)

Grading instruction

Case distinction (Number of points: 2)

Distinguish the cases for `rle.empty` and `rle.ncons`.

empty case (Number of points: 1)

ncons case (Number of points: 1)

Question 15 – 2020 resit 6b – 225501.3.0

Define a Lean function `rle_length` that takes an `rle` and returns the sum of all repetition counts in the list. For example, it would return 5 for the encoding of "AABBB".

```
def rle_length {α : Type} : rle α → ℕ
```

Grading instruction

Case distinction (Number of points: 1)

Distinguish the cases for `rle.empty` and `rle.ncons`.

empty case (Number of points: 1)

ncons case (Number of points: 4)

Recurse into the tail of the `rle` (2 points) and add the repetition count of the current element (2 points).

Question 16 – 2020 resit 6c – 225504.2.1

Consider the following `insert` function, which inserts an element `a` at the front of an `rle`:

```
def insert {α : Type} (a : α) : rle α → rle α
| rle.empty         := rle.ncons 1 a rle.empty
| (rle.ncons n b t) :=
  if a = b then rle.ncons (n + 1) a t else rle.ncons 1 a (rle.ncons n b t)
```

Prove that `insert`ing an element into an `rle` cannot return `rle.empty`:

```
lemma insert_ne_empty {α : Type} (a : α) (l : rle α) :
  insert a l ≠ rle.empty
```

Grading instruction

Case distinction (Number of points: 2)

Distinguish the case for `l = rle.empty` and the case for `l = rle.ncons`.

empty case (Number of points: 1)

ncons case (Number of points: 3)

Distinguish the case where the inserted element is equal to the first element of the `rle` and the case where the inserted element is not equal to the first element (2 points). Observe that in each case, the `rle` after insertion is an `ncons` (1 point).