Unofficial mock exam solutions

Disclaimers

These are unofficial solutions written by the tutors (Pablo Cageao and Hannah Scholz). We make no guarantees for correctness. Feel free to send us an email (pcageao@uni-bonn.de or scholz.hannah@uni-bonn.de) if you have any questions or think that you have found a mistake.

Instructions

- In many answers, you are asked to write a Lean proof. Write this Lean proof on paper
- You do not need to write a starting `by`. If you are asked to write a Lean proof of

  ```
  example {p : Prop} : p → p
  ```

  then a valid answer would be, e.g.

  ```
  intro h
  exact h
  ```

- If you are not asked to prove something a particular way, any valid Lean proof is accepted.
- If you are asked to write something using **basic tactics**, you should not use tactics that can perform many steps at once, such as `grind`, `simp`, `ring`, `tauto` or `linarith`. Tactics like `exact?` are also not allowed in such proofs.

Mathlib lemmas Here is a list of some definitions and lemmas from mathlib which you may or may not need for the exam.

- `Prod.mk.{u, v} {X : Type u} {Y : Type v} (fst : X) (snd : Y) : X × Y`
- `add_assoc.{u_1} {G : Type u_1} [AddSemigroup G] (a b c : G) : a + b + c = a + (b + c)`
- `add_comm.{u_1} {G : Type u_1} [AddCommMagma G] (a b : G) : a + b = b + a`
- `add_zero.{u} {M : Type u} [AddZeroClass M] (a : M) : a + 0 = a`
- `zero_add.{u} {M : Type u} [AddZeroClass M] (a : M) : 0 + a = a`
- `one_mul.{u} {M : Type u} [MulOneClass M] (a : M) : 1 * a = a`
- `mul_one.{u} {M : Type u} [MulOneClass M] (a : M) : a * 1 = a`

Exercises:

1. Using only **basic tactics**, give a Lean proof for

   (a) `example {p q : Prop} : p → q → p`

   Solution. It can be done using the following code

   ```
   intro hp hq
   exact hp
   ```

   (b) `example {p q r s : Prop} (hp : p → r) (hq : q → s) : p ∧ q → r ∧ s`

   Solution. It can be done using the following code

   ```
   intro ⟨hp', hq'⟩
   exact ⟨hp hp', hq hq'⟩
   ```

   (c) `example {p q : Prop} : p ∨ q → q ∨ p`

   Solution. It can be done using the following code

   ```
   rintro (hp | hq)
   . right
     exact hp
   . left
     exact hq
   ```

2. The following are all proofs that can be done with a single tactic. Give a Lean proof for these examples (you are allowed to give a longer proof).

   ! To prepare for this type of exercise, you should look at the tactic cheat sheet.

   (a) `example {x y : ℝ} (hx : 0 ≤ x) (h : 5 * x ≤ 2 * y - 1) : x ≤ y / 2`

   Solution. This can be done by `linarith`. It makes sense to use this tactic here because these are all linear inequalities. See Lecture 4.

   (b) `example : Differentiable ℝ (fun x : ℝ ↦ exp (x ^ 2) + sin (π * x))`

   Solution. This can be done with `fun_prop`. This makes sense because this is a property of a function. See Lecture 15.

   (c) `example {p q : ℝ} (h : p = q) : p = q + 0`

Solution. This can be done by `simp [h]`. This is because this is a simple application of a simp-lemma (`add_zero`) and then simply a rewrite with the hypothesis `h`. See Lecture 4.

(d) `example {x y z : ℝ} : x * y * z * (x + y) * (x - y) = x * y * z * (x ^ 2 - y ^ 2)`

Solution. This can be done using `ring`. This makes sense because this is rewriting with simple rules that hold in every ring. See Lecture 2.

(e) `example {x y : ℝ} (hx : y ≠ 0) : x * y / y = x`

Solution. This can be done by `field_simp`. This is because `field_simp` simplifies fractions in fields. Not See Lecture 6.

(f) `example {a b c d e : ℝ} (hab : a ≤ b) (hc : 0 ≤ c) (hde : d ≤ e) : exp a * c + d ≤ exp b * c + e`

Solution. This can be done using `gcongr` because it applies congruence lemmas for inequalities. Note that `gcongr` will try to close side goals using hypotheses. See Lecture 4.

(g) `example {p : Prop} : p ↔ p ∧ True ∨ False`

Solution. This can be done with `simp` because it just applies simp-lemmas (`or_false`, `and_true`, `iff_self`). See Lecture 4. It could also be done using `tauto` since this is using simple logical rules.

3. Using the lemmas

```
exp_add (x y : ℝ) : rexp (x + y) = rexp x * rexp y
exp_mul (x y : ℝ) : rexp (x * y) = rexp x ^ y
```

give a `calc`-proof of
`example {x y z : ℝ} : exp (z * x + z * y) = (exp x * exp y) ^ z.`

Solution. The following calc solves the exercise

```
example {x y z : ℝ} : exp (z * x + z * y) = (exp x * exp y) ^ z := by
    calc
        exp (z * x + z * y) = exp ((x + y) * z) := by ring
        _ = exp (x + y) ^ z := by rw [exp_mul]
        _ = (exp x * exp y) ^ z := by rw [exp_add]
```

4. State the following things in Lean.

(a) (i) The real numbers are an abelian group (under addition).

Solution. You could write `AddCommGroup ℝ`.

(ii) The non-zero real numbers $\mathbb{R}^* = \{x \in \mathbb{R} \mid x \neq 0\}$ are an abelian group (under multiplication).

Solution. You could write `CommGroup` $\mathbb{R}^x$.

(iii) For all n, we have $\sum_{i=0}^{n} i = \frac{n(n+1)}{2}$.

Solution. You could write this as

```
example (n : ℕ) : ∑ i ∈ Finset.range (n + 1), i = n * (n + 1) / 2
```

(iv) The real number 2 is the least upper bound of the interval $(-\infty, 2)$.

Solution. You could write this as

```
example : sSup (Iio (2 : ℝ)) = 2
```

(b) Which of the statements in part (a) could be made an instance?

Solution. You could make (i) and (ii) into instances because both `AddCommGroup` and `CommGroup` are classes. Intuitively, this is because being an (abelian) group is a property of an object.

5. Give a definition of a function $f : \mathbb{N} \to \mathbb{N}$ such that for $n \geq 3$ we have

$$f(n) = f(n-1) + f(n-2) + f(n-3)$$

and $f(1) = f(2) = 1$ and $f(4) = 4$. (You can define your function differently than this characterization and do *not* have to prove that your function satisfies these properties)

Solution. You could define it as follows:

```
def f : ℕ → ℕ
    | 0 => 0
    | 1 => 1
    | 2 => 1
    | n + 3 => f (n + 2) + f (n + 1) + f n
```

6. In the following exercise, assume your local context includes the hypotheses `{X Y Z : Type} {f : X × Y → Z} {a : X} {b : Y} {c : Z}`. For each of the following expressions, determine if it is well-typed. If not, explain why. If it is well-typed, give its type.

(a) `f a`

Solution. This is not well-typed because the input of the function needs to be a pair of type `X × Y` and `a` has type `X`

(b) `(c, f (b, b))`

Solution. No, this is not well-typed because the type of `(b, b)` is `Y × Y` not `X × Y`.

(c) `(c, f (a, b))`

Solution. Yes, this is well-typed with type `Z × Z`.

(d) `Prod.mk X Y a b`

Solution. No, this is not well-typed because (as you can see in the Mathlib lemma section) `Prod.mk` only takes two explicit arguments and we are providing four here.

(e) `Prod.mk (Y := X) (X := Y)`

Solution. Yes, this is well-typed with type `Y → X → Y × X`.

(f) `Prod.mk (X := Y) (Y := Y) a`

Solution. No, this is not well-typed because `Prod.mk (X := Y) (Y := Y)` has type `Y → Y → Y × Y` so we need to apply it to elements of type `Y` but `a` has type `X`.

(g) `Prod.mk (Y := Y) (fst := a)`

Solution. Yes, this is well-typed with type `Y → X × Y`.

(h) `Prod.mk b a`

Solution. Yes, this has type `Y × X`.

(i) `Prod.mk (a, b) (b, a)`

Solution. Yes, this has type `(X × Y) × (Y × X)`

(j) `Prod.mk (snd := Y) (fst := X)`

Solution. Yes, this has type `Type × Type`.

7. (a) Explain the difference between a `structure` and a `class` in Lean.

Solution. The difference is that for classes you can register instances which can then be searched for automatically using typeclass inference. See Lecture 7.

(b) Declarations can have arguments enclosed by parentheses `( )`, braces `{ }` or square brackets `[ ]`. What is the difference between their behavior?

Solution. Explicit arguments ("()") need to provided by you. Implicit arguments ("{}") are figured out by Lean from the context. Instance implicit arguments ("[]") are automatically figured out through typeclass inference. See Lectures 4 and 7.

8. In the following questions, your local context is `example {a b c : ℤ} : (a + c) + b = a + (b + c)`. You may find the reference of mathlib lemmas at the beginning helpful.

   (a) Will the tactic `rw [add_comm]` succeed in this state? If so, what does the goal look like afterwards? If not, why not?

   Solution. Yes, this succeeds and the goal will be `b + (a + c) = a + (b + c)` afterwards. This is because it rewrites not the first but the outermost addition.

   (b) Answer the same question about `rw [add_comm a _]`.

   Solution. Yes, this goal will be `(c + a) + b = a + (b + c)`

   (c) Answer the same question about `rw [add_assoc a c b]`.

   Solution. Yes, the goal will be `a + (c + b) = a + (b + c)`.

   (d) Answer the same question about `rw [← add_assoc a c]`.

   Solution. No, because there is no term of the form `a + (c + ?_)` in the goal.

9. In the following questions, assume your local context includes `{k E F : Type*} [NormedField k] [NormedAddCommGroup E] [NormedSpace k E] [NormedAddCommGroup F] [NormedSpace k F]`.

   (a) Explain the error in the following example.

```
example : F → E × F := by
  have := Prod.mk 0 (Y := F)
  exact this
```

   Lean's error is

```
Type mismatch
  this
has type
  F → Prod.{0, u_3} ℕ F
of sort `Type (max u_3 0)` but is expected to have type
  F → Prod.{u_2, u_3} E F
of sort `Type (max u_2 u_3)`
```

   Solution. `0` is automatically inferred as a natural number but the function was expecting an element of the normed commutative group `E`. (To get the 0 of `E` one needs to write `(0 : E)`.)

   (b) In
   `example {s : Set E} : F → E × F := by apply Prod.mk s`,
   explain Lean's error message.

```
Tactic `apply` failed: could not unify the type of `Prod.mk s`
  ?m.3 → Set E × ?m.3
with the goal
  F → E × F
```

Solution. The function they are trying to define has in its image elements of `E`, that is, when applying the `Prod.mk` they need to use an element of `E`, but instead they used a subset of `E`.

(c) In
```
example {a : X} (b : Y) (f : X → Y → X) : Y → X × Y := f a b a,
```
explain Lean's error message
```
Function expected at f a b but this term has type X.
```

Solution. The error comes from trying to apply the element `f a b` of `X` to another element `a` of `X` which is not possible since elements of `X` are not functions.

(d) In the following example, explain Lean's error message and discuss how to fix it.

```
example {p q : ℕ → Prop} (h : exists x, p x) (h' : ∀ x, p x ↔ q x) :
    ∃ x, q x := by rw [h'] at h
```

The error message is
```
Tactic `rewrite` failed: Did not find an occurrence of the pattern p ?x in
the target expression exists x, p x
```

Solution. `rw` does not rewrite under binders like $\forall$, $\exists$, $\int$ or $\sum$; `simp_rw` does.

10. For each pair of a true informal statement and a Lean statement below, comment on whether the Lean statement matches the informal statement. If there is a mismatch; explain what causes the mismatch and how you can fix it.

(a) "$5 - 7 + 3 = 1$" with Lean code
```
example : 5 - 7 + 3 = 1
```

Solution. This would be false as numbers are interpreted as naturals, this subtraction is the natural subtraction. That is, `5 - 7 + 3 = 0 + 3 = 3`. A correct translation would be `(5 : ℤ) - 7 + 3 = 3`.

(b) "For every integer n we have $(\frac{n}{2})^2 = \frac{n^2}{4}$" with Lean code
```
example (n : ℤ) : (n / 2) ^ 2 = n ^ 2 / 4
```

Solution. This would be false, too. As before, `n` is being defined as an integer and as such `n/2` is using integer division. A correct translation would be `example (n : ℤ) : ((n : ℚ) / 2) ^ 2 = n ^ 2 / 4`.

(c) "If $f\colon \mathbb{R} \to \mathbb{R}$ has derivative 0 everywhere and $f(0) = 1$ then $f(x) = 1$ for all x." with Lean code
```
example {f : ℝ → ℝ} (h1 : ∀ x, deriv f x = 0) (h2 : f 0 = 1) : f = 1
```

Solution. This statement doesn't match the original one. From matlib: "`deriv f x` is a derivative of `f` at `x`. If the derivative does not exist, then `deriv f x` equals zero.". So we are not expressing that `f` is differentiable. You could fix this by adding a hypothesis `(h3 : Differentiable ℝ f)` or by replacing `h1` with `(h1 : ∀ x, HasDerivAt f 0 x)`.