Final Project

Logical Foundations using the Coq Proof Assistant 2023/B

Lecturer: Liron Cohen

Submission instructions:

- **Due date**: 19.7.23 at 23:59. **There will be no extensions.**
- The completed project is to be submitted via Moodle.
- Specific instructions for each part of the project are given below.

General instructions:

- The final project must be done **individually**, and you may **NOT** discuss the project's problems with other students.
- You can, and should, use the course material. It is recommended to draw inspiration from similar definitions and proofs in the course book.
- It is also highly recommended to use automation in the proofs (again, in a similar manner to the way they are used in the course book). Without automation the proofs can be very long and tedious.
- **Note that you may be asked to explain your solution in a personal meeting after submitting your work.**

Academic integrity:

I'd like to restate the academic integrity policy:

The utmost level of academic integrity is expected of all students. Violations of the code of academic integrity are treated very seriously by BGU and can have long-term repercussions. Academic dishonesty includes any act of obtaining, soliciting or making available to others, material related to homework assignments. If you commit any of the above, you are guilty of academic dishonesty. Note that BGU holds responsible for the code violation both the recipient and the donor of improper information.

Therefore, together with your solution to the project, **you are requested to upload a scan of this page after you have filled in the following statement.**

My signature below constitutes my pledge that I have followed BGU's policy on academic integrity as well as the specific instructions for this project. I affirm that this project represents my own work, without the use of any unpermitted aids or resources. I understand that there will no tolerance towards academic dishonesty, and that cheating can and will lead to automatic failure from the class as well as a report to the Academic Integrity Committee.

Full name:	Daniel kheyfets
ID number:	207993833
Signature:	[illegible]

Part 1 – Regular Expressions (30%)

In the chapter INDPROP, there is a section called “Case Study: Regular Expressions”. Unsurprisingly, it represents and studies regular expressions as inductive propositions.

Complete ALL the exercises of this case study in the attached IndProp.v file. It is of course recommended to read through the whole section.

- exp_match_ex1
- reg_exp_of_list_spec
- re_not_empty
- exp_match_ex2
- weak_pumping
- pumping

Part 2 – Imperative programming (30%)

Fill in the following exercises in the attached Imp.v file:

- loop_never_stops
- ~~no_whiles_eqv~~
- no_whiles_terminating
- break_imp
- while_break_true
- ceval_deterministic
- add_for_loop

Instructions: As usual, in order for the auto grader to work correctly (and give you full credit for your work!), please be careful to follow these rules:

- Please use only the file attached to this assignment. Do not use any other version of this file.
- Do not alter the "markup" that delimits exercises: The Exercise header, the name of the exercise, the "empty square bracket" marker at the end, etc. Please leave this markup exactly as you find it.
- Do not delete or comment exercises. If you skip an exercise (e.g., because it is marked "optional," or because you can't solve it), it is OK to leave a partial proof in your [.v] file; in this case, please make sure it ends with [Admitted] (not, for example [Abort]).
- It is fine to use additional definitions (of helper functions, useful lemmas, etc.) in your solutions. You can put these between the exercise header and the theorem you are asked to prove.

Part 3 – Constructive mathematics (40%):

This question concerns the constructive universe, and in particular, it explores some of its benefits in the context of synthetic computability. Specifically, the goal of this exercise is to show that there can't be a constructive proof that every enumerable predicate is decidable, as this would induce a decider of the halting problem.

For a subset of the natural numbers (i.e., a predicate over the natural numbers) the properties of being enumerable and being decidable can be formally stated in Coq as follows:

```
Set Implicit Arguments.
Unset Strict Implicit.
Require Import List Lia.

Definition enum_mono (P : nat -> Prop) :=
  exists e : nat -> nat, (forall k k', k <= k' -> e k <= e k') /\ forall
n, P n <-> exists k, e k = n.

Definition dec (P : nat -> Prop) :=
  exists d : nat -> bool, forall n, P n <-> d n = true.
```

The **limited principle of omniscience** (LPO) states that the existential quantification of any decidable proposition is again decidable. For the natural numbers, a functional variant this principle can be formalized in Coq as follows:

```
Definition LPO :=
  forall f : nat -> bool, (exists n, f n = true) \/ (forall n, f n =
false).
```

This statement is a common divider between constructive and classical mathematics.

Copy the above lines into your solution file.

1. Implement a function `print f` that induces a predicate `P_print f` such that `enum_mono (P_print f)` holds and `dec (P_print f)` implies `LPO`.
   Copy the code below to your solution file and complete the missing bits. To help you on your quest, we supply one lemma you might want to use (but need to prove in order to do so). You might need to come up with other helper lemmas yourself.

```
Definition print (f : nat -> bool) (k : nat) : nat :=
  (* FILL IN HERE *) .

Lemma print_mono f k k' :
  k <= k' -> print f k <= print f k'.
Proof.
  (* FILL IN HERE *)
Admitted.

Definition P_print (f : nat -> bool) :=
  fun n => exists k, print f k = n.
```

2. Copy the code below to your solution file and complete the missing bits.

```
Theorem enum_mono_dec_LPO :
  (forall P, enum_mono P -> dec P) -> LPO.
Proof.
  intros H f. destruct (H (P_print f)) as [d Hd].
  - exists (print f). split; try apply print_mono. reflexivity.
  -   (* FILL IN HERE *)
Admitted.
```

Instructions:

- For this part you will need to create your own Coq file from scratch. Name the file "LPO.v".
- To get partial grading, it is possible to answer only some of the questions, leaving the rest of the proofs as "Admitted".

Good Luck!
