Advanced Topics in Formal Methods and Prog. Languages
– Software Verification with Isabelle/HOL –

Take-Home Exam

ver 1.0

Submission Guidelines

- Start time: Nov 1, 2024, 9am (Canberra Time)
- Due time: Nov 2, 2024, 9am (Canberra Time)
- Submit via Wattle.
- Accepted formats are plain text (.txt) files, PDF (.pdf) files, and Isabelle theory (.thy) files.
- Isabelle files should be executable (a template is provided on the course webpage).
- Please read and sign the declaration on the last page and attach a copy to your submission.
- **This is an exam, no late submission, deadline is strict.**

Remarks

1. Unless stated otherwise, all proof methods and proof automation available in the standard Isabelle distribution are allowed. This includes, but is not limited to `simp`, `auto`, `blast`, `force`, and `fastforce`. However, if you are going for full marks, you should not use "proof"-methods that bypass the inference kernel, such as `sorry`. We may award partial marks for plausible proof sketches where some subgoals or lemmas are sorried.
2. For all questions, you may prove your own helper lemmas, and you may use lemmas proved earlier in other questions. You can also use automated tools like sledghammer. If you can't finish an earlier proof, use sorry to assume that the result holds so that you can use it if you wish in a later proof. You won't be penalised in the later proof for using an earlier true result you were unable to prove, and you'll be awarded partial marks for the earlier question in accordance with the progress you made on it.
3. If you use `sledgehammer`, it is important to understand that the proofs suggested by sledgehammer are just suggestions, and aren't guaranteed to work. Make sure that the proof suggested by sledgehammer actually terminates on your machine (assuming an average spec machine). If not, you can try to reconstruct the proof yourself based on the output, or apply a few manual steps to make the subgoal smaller before using sledgehammer. Please **avoid proofs based on** `smt`. It may useful (and necessary) to fine control the simplifier by using `simp only:`, `simp (no-asm-simp)`, etc.
4. All work must be your own, the exam policy is more restrictive than for assignments: You must not discuss the exam with anyone except the lecturers of this course before the exam is due. Do not give or receive assistance. You are allowed to use all lecture material, slides, and assignment solutions from the web. You are also allowed to use other passive internet resources such as Google, the Isabelle tutorial or Isabelle documentation. You are not allowed to ask for assistance on mailing lists, forums, generative AI (e.g. chatGPT or Grammerly), or anywhere else. You are allowed to clarify questions with the lecturers via EdStem.
5. The exam consists of **4 exercises for COMP4011**, totalling 100 marks, and of **5 exercises for COMP8011**, totalling 110marks. The difference is due to separate AQF levels of the two courses.

© P. Höfner, R. Barry

Exercise 1 (Higher-Order Logic) (10 Marks)

Prove each of the following statements, using only the proof methods: `rule`, `erule`, `assumption`, `cases`, `frule`, `drule`, `rule_tac`, `erule_tac`, `frule_tac`, `drule_tac`, `rename_tac`, and `case_tac`; and using only the proof rules: `impI`, `impE`, `conjI`, `conjE`, `disjI1`, `disjI2`, `disjE`, `notI`, `notE`, `iffI`, `iffE`, `iffD1`, `iffD2`, `ccontr`, `classical`, `FalseE`, `TrueI`, `conjunct1`, `conjunct2`, `mp`, `allI`, `allE`, `exI`, and `exE`. You do not need to use all of these methods and rules.

As usual, you can introduce your own helper lemmas.

(a) $(\forall x.\ \neg (P\ x)) \implies (\nexists x.\ P\ x)$

(b) $(\forall x.\ F\ x \longrightarrow \neg G\ x) = (\nexists x.\ F\ x \wedge G\ x)$

Exercise 2 (Termination) (22 Marks)

In this exercise we consider the function `func :: nat ⇒ nat list ⇒ nat` defined as follows:

```
func x xs = (if x∈set xs then func (x+1) xs else x)
```

Remember that `set` returns the set of all elements of a given list.

(a) What does this function calculate? Write a short paragraph.

(b) Write a recursive function `maxlist :: nat list ⇒ nat` that returns the maximum of a given list.

(c) Prove that the function behaves as expected. That means:

- element is in list: `xs ≠ [] ⟹ maxlist xs ∈ set xs`
- all elements are smaller or equal: `x ∈ set xs ⟹ x ≤ maxlist xs`

(d) Prove termination of function `func`. Remember that you should provide a measurement that strictly decreases in each iteration of `func`.

Exercise 3 (Induction and Inductive Sets) (33 Marks)

In this exercise we are looking at palindromes, which can be defined inductively.
Hint: When using induction, consider which induction `rule` you want to apply.

(a) Using the **inductive_set** command, define the set `par_lists` that contains:

- the empty list,
- all lists with a single element, and
- all lists formed by adding the *same* elements to the front and back of an existing list.

A palindrome is a word, phrase, number, or sequence that reads the same forward and backward, ignoring spaces, punctuation, and capitalisation. When the sequence is encoded as a list, that means `palindrome xs ≡ (rev xs = xs)`.

(b) Prove that `xs` is an element of `par_lists` iff xs is a palindrome.
`palindrome xs = (xs ∈ par_lists)`

We are now constructing palindromes. The simplest way is to append a reversed list.

(c) Show that `xs@(rev xs) ∈ par_lists` and `xs@tl(rev xs) ∈ par_lists`, where `tl xs` indicates the tail of a list.

A more interesting construction is to 'wrap' a palindrome around another. For that purpose we define a function `split` that divides the palindrome in the middle.

function `split :: 'a list ⇒ 'a list × 'a list` **where**

```
split [] = ([],[])  |
split [x] = ([x],[x])  |
split ([x]@xs@[y]) = ([x]@fst(split xs), snd(split xs)@[y])
```

Based on this definition we define a wrapping function:

definition `pappend (infixr "⋄"65)` **where**

```
pappend xs ys = fst(split xs)@ys@snd(split xs)
```

(d) Prove that $\diamond$ maintains the palindrome property:

$[\![$`palindrome xs; palindrome ys`$]\!] \Longrightarrow$ `palindrome (xs⋄ys)`

(e) Last, prove that the operator $\diamond$ is associative.

Exercise 4 (Verification and Hoare Logic) (35 Marks)

This exercise captures Hoare logic. We aim at the verification of an algorithm that factorises a natural number `n0`. The prime factors will be collected in a (sorted) list. The definition of prime number is standard.

(a) Using the **primrec** construct, provide a function `mult_list` that multiplies all elements in a list. Some examples are given in the Isabelle file.

These definitions give raise to the following pre- and postconditions:

`PRE` $\equiv$ `n0 > 0`,
`POST1` $\equiv$ `mult_list ps = n0`, and
`POST2` $\equiv$ `(∀p ∈ set ps. prime p)`,

where `ps` is the calculated list of prime factors.

(b) Describe the following algorithm in your own words.

```
{ n = n0 ∧ PRE }
  d := 2;
  ps := [];
  WHILE n > 1
  INV {INV1-todo}
  DO
    WHILE d dvd n
    INV {INV2-todo}
    DO
      ps := d#ps;
      n := n div d
    OD;
    d := d + 1
  OD
{ POST1 ∧ POST2 }
```

The algorithm features two while loops, which means we have to derive two invariants.

(c) Find an invariant that relates `ps` and `n0`. Briefly argue why your formula is reasonable (valid). Show that this invariant is strong enough to verify the algorithm for postcondition `mult_list ps = n0`.

(d) The invariant must also consider the fact that `ps` consists of primes only, i.e. `(∀p ∈ set ps. prime p)`, and a formula containing `d`. Try to create 'complete' invariants, and briefly argue (informally) why they should hold.

(e) Finally prove the algorithm to be (partially) correct. Of course you can refine your invariants.

Exercise 5 (Mutual Recursion) (10 Marks)

!! Answer this question only if you are enrolled in COMP8011 !!

When asked, ChatGPT produces the following example for two mutually recursive functions, which work on binary trees.

```
datatype tree = Leaf | Node tree tree

function even_tree :: tree ⇒ bool and odd_tree :: tree ⇒ bool where
    even_tree Leaf = True
  | even_tree (Node l r) = (odd_tree l ∧ odd_tree r)
  | odd_tree Leaf = False
  | odd_tree (Node l r) = (even_tree l ∧ even_tree r)
```

ChatGPT states that "one function (`even_tree`) checks whether the number of nodes in a tree is even, while the other function (`odd_tree`) checks if the number of nodes is odd. This is an example of a correctly structured mutual recursive definition, where both functions are properly defined, terminate, and handle all cases for the tree datatype. The mutual recursion works logically, ensuring soundness and completeness."

Critically analyse this statement, and discuss pros and cons of the definition. You should use Isabelle/HOL to underpin your statements with lemmas and formal definitions.

Academic Integrity

I declare that this work upholds the principles of academic integrity, as defined in the University Academic Misconduct Rule; is entirely my own work, with only the exceptions listed; is produced for the purposes of this assessment task and has not been submitted for assessment in any other context, except where authorised in writing by the course convener; gives appropriate acknowledgement of the ideas, scholarship and intellectual property of others insofar as these have been used; in no part involves copying, cheating, collusion, fabrication, plagiarism or recycling.

_____ _____

Date Signature