

ITP Exercise 3

due Friday 12th May

1 Self-Study

1.1 Tactics and Tacticals

You should read background information on the tactics mentioned in the lecture. For each of the tactics and tacticals mentioned on the slides, read the entry in the description manual.

1.2 hol-mode

Carefully study the `Goalstack` submenu of hol-mode. Learn the keycodes to set goals, expand tactics, undo the last tactic expansion, restart the current proof and drop the current goal.

1.3 num and list Theories

We will use the natural number theory `numTheory` as well as the list theories `listTheory` and `rich_listTheory` a lot. Please familiarise yourself with these HOL theories (e.g. by looking at their signature in the HTML version of the HOL Reference). I also recommend reading up on other common theories like `optionTheory`, `oneTheory` and `pairTheory`. These won't be needed for this weeks exercises, though.

2 Backward Proofs

Part of this exercise is searching for useful existing lemmata (e.g. useful rewrite rules). Another part is understanding the effect of different rewrite rules and their combination. Even if you have experience with using HOL, please refrain from using automated rewrite tools that spoil these purposes and have not been covered in the lecture yet. Please don't use HOL's simplifier and especially don't use stateful simp-sets. Similarly, please don't use the compute lib via e.g. `EVAL_TAC`.

For these exercise, it might be beneficial to open the modules `listTheory` and `rich_listTheory` via `open listTheory rich_listTheory`. You will notice that when opening them a lot of definitions are printed. This can consume quite some time when opening many large theories. Play around with the hol-mode commands `Send region to HOL - hide non-errors` and `Quite - hide output except errors` to avoid this printout and the associated waiting time.

2.1 Replay Proofs from Lecture

If you never did a tactical proof in HOL before, I recommend following the example interactive proofs from Part VIII. Type them in your own HOL session, make the same mistakes shown in the lecture. Use hol-mode to control the goalStack via commands like `expand`, `back-up`, `set goal`, `drop goal`. Get a feeling for how to interactively develop a tactical proof.

This exercise is optional. If you are confident enough, feel free to skip it.

2.2 Formalise Induction Proofs from Exercise 1

For exercise sheet 1, some simple properties of appending lists were proved with pen and paper via induction. Let's now try to prove them formally using HOL. Prove $\vdash 1 \mathrel{++} [] = 1$ via induction using the definition of APPEND ($++$). Similarly, prove the associativity of APPEND, i.e. prove $\vdash 11 \ 12 \ 13. \ 11 \mathrel{++} (12 \mathrel{++} 13) = (11 \mathrel{++} 12) \mathrel{++} 13$.

2.3 Reverse

In HOL, `revAppend` is called `REV`. Using any useful lemmata you can find, prove the lemma $\vdash 11 \ 12. \text{LENGTH} (\text{REV } 11 \ 12) = (\text{LENGTH } 11 + \text{LENGTH } 12)$. Now, let us as an exercise reprove the existing lemmata `REVERSE_REV` and `REV_REVERSE_LEM`. This means, prove, first prove $\vdash 11 \ 12. \text{REV } 11 \ 12 = \text{REVERSE } 11 \mathrel{++} 12$. Then prove $\vdash 1. \text{REVERSE } 1 = \text{REV } 1 []$ using this theorem. You should not use the theorems `REVERSE_REV` or `REV_REVERSE_LEM` in these proofs.

2.4 Length of Drop

Prove $\vdash 11 \ 12. \text{LENGTH} (\text{DROP} (\text{LENGTH } 12) (11 \mathrel{++} 12)) = \text{LENGTH } 11$. This is a bit tricky. Please play around with the proof for some time. If you can't figure it out, look at the hints at the end of this exercise sheet.

2.5 Making Change

On exercise sheet 1, you were asked to implement a function `make_change` in SML. Let's now define it in HOL and prove some properties. Define the function `MAKE_CHANGE` in HOL via

```
val MAKE_CHANGE_def = Define '  
  (MAKE_CHANGE [] a = if (a = 0) then [[]] else []) /\  
  (MAKE_CHANGE (c::cs) a = (  
    (if (c <= a /\ 0 < a) then  
      (MAP (\1. c::1) (MAKE_CHANGE cs (a - c)))  
    else []) ++ (MAKE_CHANGE cs a)))';
```

Prove that $\vdash cs. \text{MAKE_CHANGE } cs \ 0 = [[]]$ and $\vdash cs \ a \ 1. \text{MEM } 1 (\text{MAKE_CHANGE } cs \ a) \implies (\text{SUM } 1 = a)$ hold.

3 Hints

3.1 Length of Drop

For proving $\vdash 11 \ 12. \text{LENGTH} (\text{DROP} (\text{LENGTH } 12) (11 \mathrel{++} 12)) = \text{LENGTH } 11$ induction on the structure of 12 is a good strategy. However, one needs to be careful that 11 stays universally quantified. Expanding naively `Induct_on '12'` or `GEN_TAC >> Induct` will remove the needed universal quantification of 11.

To solve this, you can get rid of both universal quantifiers and then introduce the one for 11 again. This is achieved by `REPEAT GEN_TAC >> SPEC_TAC (''11:''a list'', ''11:''a list'')`.