# ITP Exercise 5

**due Friday 26th May**

## 1 Multiple Definitions / Formal Sanity

`rich_listTheory` provides a predicate `IS_SUBLIST`. It checks whether a list appears somewhere as part of another list:

```
|- !l1 l2. IS_SUBLIST l1 l2 <=> ?l l'. l1 = l ++ (l2 ++ l')
```

Define a weaker version of such a predicate called `IS_WEAK_SUBLIST` that allows additional elements between the elements of `l2`. So, for example `IS_WEAK_SUBLIST [1;2;3;4;5;6;7] [2;5;6]` should hold. In contrast the statements `IS_WEAK_SUBLIST [1;2;3;4;5;6;7] [2;6;5]` or `IS_WEAK_SUBLIST [1;2;3;4;5;6;7] [2;5;6;8]` do not hold. Another way of describing the semantics of `IS_WEAK_SUBLIST l1 l2` is saying that one can get `l2` by removing elements from `l1` while keeping the order.

### 1.1 Recursive Definition

Define `IS_WEAK_SUBLIST` recursively using `Define`. Name your function `IS_WEAK_SUBLIST_REC`. Test this definition via `EVAL` and prove at least 2 sanity check lemmata, which do not coincide with the lemmata you are asked to show below.

### 1.2 Filter Definition

Define a version of `IS_WEAK_SUBLIST` called `IS_WEAK_SUBLIST_FILTER` using the existing list function `FILTER`. You might want to use `ZIP`, `MAP`, `FST` and `SND` as well. The idea is to check for the existence of a list of booleans of the same length as `l1`, zip this list with `l1` and filter. You probably want to introduce auxiliary definitions before defining `IS_WEAK_SUBLIST_FILTER`.

The resulting definition is not executable via `EVAL`. Anyhow, show at least 2 sanity check lemmata, which do not coincide with the lemmata you are asked to show below.

### 1.3 Equivalence Proof

Show `IS_WEAK_SUBLIST_REC = IS_WEAK_SUBLIST_FILTER`. You might want to prove various auxiliary lemmata first. You might want to use among other things `FUN_EQ_THM` and the list function `REPLICATE`.

### 1.4 Properties

Show the following properties of `IS_WEAK_SUBLIST_REC` and `IS_WEAK_SUBLIST_FILTER`. This means that for each property stated below in terms of `IS_WEAK_SUBLIST` you should prove one lemma using `IS_WEAK_SUBLIST_REC` and another lemma using `IS_WEAK_SUBLIST_FILTER`. Don't use the fact that both functions are equal. The point of this exercise is partly to demonstrate the impact of different definitions on proofs. You might of course use previously proved lemmata to prove additional ones.

1. `!l1 l2 l2a l2b. IS_WEAK_SUBLIST l1 l2 ==> IS_WEAK_SUBLIST l1 (l2a ++ l2 ++ l2b)`
2. `!l1a l1b l2a l2b. IS_WEAK_SUBLIST l1a l2a ==> IS_WEAK_SUBLIST l1b l2b ==> IS_WEAK_SUBLIST (l1a ++ l1b) (l2a ++ l2b)`
3. `!l. IS_WEAK_SUBLIST l l`
4. `!l1 l2 l3. IS_WEAK_SUBLIST l1 l2 ==> IS_WEAK_SUBLIST l2 l3 ==> IS_WEAK_SUBLIST l1 l3`
5. `!l1 l2. IS_WEAK_SUBLIST l1 l2 ==> IS_WEAK_SUBLIST l2 l1 ==> (l1 = l2)`

# 2 Deep and Shallow Embeddings

As seen in the lecture let's define a deep and a shallow embedding of propositional logic. Use the names and definitions from the lecture notes. Add a definition stating that two propositional formulas are equivalent, iff their semantics coincides for all variable assignments, i. e.

```
PROP_IS_EQUIV p1 p2 <=> (!a. PROP_SEM a p1 = PROP_SEM a p2)
```

## 2.1 Syntax for propositional formulas

Define in SML syntax functions for all shallowly embedded propositional formulas. Define for each constructor a make - function, a destructor and a check. For `sh_and` I would like to have for example

- `mk_sh_and : term -> term -> term`,
- `dest_sh_and : term -> (term * term)` and
- `is_sh_and : term -> bool`.

Define a check `is_sh_prop : term -> bool` that checks whether a term is a shallowly embedded propositional formula.

## 2.2 Getting Rid of Conjunction and Implication

Define a function in HOL `PROP_CONTAINS_NO_AND_IMPL : prop -> bool` that checks whether a propositional formula contains no conjunction and implication operators. Define a similar function `sh_prop_contains_no_and_impl` in SML that checks the same property for shallowly embedded formulas.

Define a function `PROP_REMOVE_AND_IMPL` in HOL that removes all conjunctions and implications from a propositional formula and returns an equivalent one. Prove these properties, i. e. prove

- `!p. PROP_IS_EQUIV (PROP_REMOVE_AND_IMPL p) p`
- `!p. PROP_CONTAINS_NO_AND_IMPL (PROP_REMOVE_AND_IMPL p)`

Implement a similar function `sh_prop_remove_and_impl : term -> thm` in SML that performs the same operation on the shallow embedding and returns a theorem stating that the input term is equal to a version without conjunctions and implications. The SML version is allowed to fail, if the input term does not satisfy `is_sh_prop`.

Notice, that `PROP_REMOVE_AND_IMPL` is a verified function, whereas `sh_prop_remove_and_impl` is a verifying one.

## 3 Fancy Function Definitions

In the lecture the termination proof for quicksort was briefly discussed. As an exercise, let's define `minsort`. This function `minsort` sorts a list of natural numbers, by always searching a minimal element of the list, put it in front of the list a recursively sort the rest of this list. In HOL, it can be defined as

```
val expunge_def =
 Define
    `(expunge x []     = [])
 /\  (expunge x (h::t) = if x=h then expunge x t else h::expunge x t)`;

val min_def =
 Define
    `(min [] m = m)
 /\  (min (h::t) m = if m <= h then min t m else min t h)`;

val minsort_defn =
 Hol_defn "minsort"
    `(minsort [] = [])
 /\  (minsort (h::t) =
       let m = min t h
       in
         m::minsort (expunge m (h::t)))`;
```

Notice, that TFL (i. e. `Define`) is not able to show automatically that `minsort` is terminating. You need to do this manually. Show auxiliary lemmata about `min` and `expunge` and use them with `Defn.tprove` (and `Defn.tgoal`) to show that `minsort` terminates.

# 4 Hints

## 4.1 Definition of IS_WEAK_SUBLIST

IS_WEAK_SUBLIST_REC and IS_WEAK_SUBLIST_FILTER can be defined by

```
val IS_WEAK_SUBLIST_REC_def = Define `
  (IS_WEAK_SUBLIST_REC (l1 : 'a list) ([]:'a list) = T) /\
  (IS_WEAK_SUBLIST_REC [] (_::_) = F) /\
  (IS_WEAK_SUBLIST_REC (y::ys) (x::xs) = (
     (x = y) /\ IS_WEAK_SUBLIST_REC ys xs) \/ (IS_WEAK_SUBLIST_REC ys (x::xs)))`;

val FILTER_BY_BOOLS_def = Define `
  FILTER_BY_BOOLS bl l = MAP SND (FILTER FST (ZIP (bl, l)))`

val IS_WEAK_SUBLIST_FILTER_def = Define `IS_WEAK_SUBLIST_FILTER l1 l2 =
  ?(bl : bool list). (LENGTH bl = LENGTH l1) /\ (l2 = FILTER_BY_BOOLS bl l1)`
```

## 4.2 Termination of minsort

minsort is an example of the TFL library. You can find a termination proof in the HOL sources. However, really try to prove termination yourself first. Before you start looking up the proof, here a few hints:

- The main idea is that the length of `expunge m (h::t)` is shorter than the length of `h::t`, i. e. start your termination proof with `WF_REL_TAC LENGTH`.
- show the lemma `!x xs. MEM x xs ==> LENGTH (expunge x xs) < LENGTH xs`
- show the lemma `!x xs. MEM (min xs x) (x::xs)`