Isabelle/HOL Exercises
Lists

Sets as Lists

Finite sets can obviously be implemented by lists. In the following, you will be asked to implement the set operations union, intersection and difference and to show that these implementations are correct. Thus, for a function

```
  list_union :: "['a list, 'a list] ⇒ 'a list"

primrec
  "list_union xs []     = xs"
  "list_union xs (y#ys) = (let result = list_union xs ys in if y mem result then
result else y#result)"
```

to be defined by you it has to be shown that

```
lemma "set (list_union xs ys) = set xs ∪ set ys"
  apply (induct "ys")
    apply simp
  apply (simp add: Let_def mem_iff)
  apply auto
done
```

In addition, the functions should be space efficient in the sense that one obtains lists without duplicates (`distinct`) whenever the parameters of the functions are duplicate-free. Thus, for example,

```
lemma [rule_format]:
  "distinct xs ⟶ distinct ys ⟶ (distinct (list_union xs ys))"
  apply (induct "ys")
  apply (auto simp add: Let_def mem_iff)
done
```

Hint: `distinct` is defined in `List.thy`. Also the function `mem` and the lemma `set_mem_eq` may be useful.

We omit the definitions and correctness proofs for set intersection and set difference.

Quantification over Sets

Define a (non-trivial) set `S` such that the following proposition holds:

lemma `"((∀ x ∈ A. P x) ∧ (∀ x ∈ B. P x)) ⟶ (∀ x ∈ S. P x)"`
lemma `"((∀ x ∈ A. P x) ∧ (∀ x ∈ B. P x)) ⟶ (∀ x ∈ A∪B. P x)"`
  by `auto`

Define a (non-trivial) predicate `P` such that

lemma `"∀ x ∈ A. P (f x) ⟹ ∀ y ∈ f ‘ A. Q y"`
lemma `"∀ x ∈ A. Q (f x) ⟹ ∀ y ∈ f ‘ A. Q y"`
  by `auto`