Isabelle/HOL Exercises
Lists

Recursive Functions and Induction: Zip

Read the chapter about total recursive functions in the "Tutorial on Isabelle/HOL" (*fun*, Chapter 3.5).

In this exercise you will define a function *Zip* that merges two lists by interleaving. Examples: *Zip [a1, a2, a3] [b1, b2, b3] = [a1, b1, a2, b2, a3, b3]* and *Zip [a1] [b1, b2, b3] = [a1, b1, b2, b3]*.

Use three different approaches to define *Zip*:

1. by primitive recursion on the first list,
2. by primitive recursion on the second list,
3. by total recursion (using *fun*).

```
primrec zip1 :: "'a list ⇒ 'a list ⇒ 'a list" where
  "zip1 []     ys = ys"
| "zip1 (x#xs) ys = (case ys of [] ⇒ x#xs | z#zs ⇒ x # z # zip1 xs zs)"
```

```
primrec zip2 :: "'a list ⇒ 'a list ⇒ 'a list" where
  "zip2 xs []     = xs"
| "zip2 xs (y#ys) = (case xs of [] => y#ys | z#zs => z # y # zip2 zs ys)"
```

```
fun zipr :: "'a list ⇒ 'a list ⇒ 'a list" where
  "zipr [] ys = ys"
| "zipr xs [] = xs"
| "zipr (x#xs) (y#ys) = x # y # zipr xs ys"
```

Show that all three versions of *Zip* are equivalent.

```
lemma zip1_zip2: "zip1 xs ys = zip2 xs ys"
  apply (induct xs arbitrary: ys)
    apply (case_tac ys)
    apply auto
  apply (case_tac ys)
  apply auto
```

```
done

lemma zip2_zipr: "zip2 xs ys = zipr xs ys"
  apply (induct ys arbitrary: xs)
    apply (case_tac xs)
    apply auto
  apply (case_tac xs)
  apply auto
done

lemma "zipr xs ys = zip1 xs ys"
by (simp add: zip1_zip2 zip2_zipr)
```

Show that *zipr* distributes over *append*.

```
lemma "⟦length p = length u; length q = length v⟧ ⟹
  zipr (p@q) (u@v) = zipr p u @ zipr q v"
  apply (induct p arbitrary: q u v)
    apply auto
  apply (case_tac u)
    apply auto
done
```

Note: For *fun*, the order of your equations is relevant. If equations overlap, they will be disambiguated before they are added to the logic. You can have a look at these equations using *thm zipr.simps*.