Exercise 5.1  Program Equivalence

Prove or disprove (by giving counterexamples) the following program equivalences.

1. $\text{IF And } b_1 b_2 \text{ THEN } c_1 \text{ ELSE } c_2 \sim \text{IF } b_1 \text{ THEN } \text{IF } b_2 \text{ THEN } c_1 \text{ ELSE } c_2 \text{ ELSE } c_2$

2. $\text{WHILE And } b_1 b_2 \text{ DO } c \sim \text{WHILE } b_1 \text{ DO WHILE } b_2 \text{ DO } c$

3. $\text{WHILE And } b_1 b_2 \text{ DO } c \sim \text{WHILE } b_1 \text{ DO } ; ; \text{WHILE And } b_1 b_2 \text{ DO } c$

4. $\text{WHILE Or } b_1 b_2 \text{ DO } c \sim \text{WHILE Or } b_1 b_2 \text{ DO } c ; ; \text{WHILE } b_1 \text{ DO } c$

Hint: Use the following definition for $\text{Or}$:

$\text{definition Or :: } \text{"bexp } \Rightarrow \text{bexp } \Rightarrow \text{bexp" where}$

$\text{"Or } b_1 b_2 = \text{Not } (\text{And } (\text{Not } b_1) (\text{Not } b_2))$"

Exercise 5.2  Nondeterminism

In this exercise we extend our language with nondeterminism. We will define $\text{nondeterministic choice } (c_1 \text{ OR } c_2)$, that decides nondeterministically to execute $c_1$ or $c_2$; and $\text{assumption } (\text{ASSUME } b)$, that behaves like $\text{SKIP}$ if $b$ evaluates to true, and returns no result otherwise.

1. Modify the datatype $\text{com}$ to include the new commands $\text{OR}$ and $\text{ASSUME}$.

2. Adapt the big step semantics to include rules for the new commands.

3. Prove that $c_1 \text{ OR } c_2 \sim c_2 \text{ OR } c_1$

4. Prove: $(\text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2) \sim ((\text{ASSUME } b ; c_1) \text{ OR } (\text{ASSUME } (\text{Not } b); c_2))$

Note: It is easiest if you take the existing theories and modify them.

Homework 5.1  Break

Submission until Tuesday, November 17, 10:00am.

Note: This homework comes with a template file. You are strongly encouraged to use it!
Your task is to add a break command to the IMP language. The break command should immediately exit the innermost while loop.

The new command datatype is:

```plaintext
datatype com = SKIP
  | Assign vname aexp ("_ ::= _" [1000, 61] 61)
  | Seq com com ("_/" [60, 61] 60)
  | If bexp com com ("(IF / THEN / ELSE )" [0, 0, 61] 61)
  | While bexp com ("(WHILE / DO )" [0, 61] 61)
  | BREAK
```

The idea of the big-step semantics is to return not only a state, but also a break flag, which indicates a pending break. Modify/augment the big-step rules accordingly:

```plaintext
inductive big_step :: "com × state ⇒ bool × state ⇒ bool" (infix "⇒" 55)
```

Now, write a function that checks that breaks only occur in while-loops

```plaintext
fun break_ok :: "com ⇒ bool"
```

Show that the pending break-flag is not set after executing a well-formed command

```plaintext
lemma
  assumes "break_ok c"
  assumes "(c, s) ⇒ (brk, t)"
  shows "¬brk"
```

**Homework 5.2 Variables not occurring in command**

*Submission until Tuesday, November 17, 10:00am.*

Write a function which checks whether a variable occurs in a command. (Hint: You need to write such functions also for Boolean and arithmetic expressions)

```plaintext
fun occ :: "vname ⇒ com ⇒ bool" where
```

Show the following two lemmas, which state that a program does not modify, nor depends on variables that do not occur in it.

Hint: For induction, use the customized `big_step_induct` rule!

```plaintext
lemma no_touch:
  assumes "¬occ x c"
  assumes "(c, s) ⇒ (brk, t)"
  shows "t x = s x"

lemma no_dep:
  assumes "¬occ x c"
  assumes "(c, s) ⇒ (brk, t)"
  shows "(c, s(x:=v)) ⇒ (brk, t(x:=v))"
```
Homework 5.3  Eliminating Breaks

Submission until Tuesday, November 17, 10:00am.

In this homework, you shall prove correct an elimination procedure for breaks, which we have already specified for you.

The procedure works by using an auxiliary variable. We will assume that it does not occur in the original program.

definition “breakvar ≡ ”_break_“”

fun ebrk :: “com ⇒ com” — Rules given in homework template!

The following lemma states one direction of the correctness of our construction: If we execute the original program, the modified program has the same execution, and, if and only if the original program has a pending break, breakvar is set. (Note that, as breakvar is initially zero and does not occur in c, it is also zero in t (cf lemma no_touch)!)

We give you a proof template here, you have to prove the interesting cases for loops, the other cases go through automatically!

lemma
assumes “¬occ breakvar c”
assumes “s breakvar = 0”
assumes “(c,s) ⇒ (brk,t)”
shows “case brk of
  False ⇒ (ebrk c, s) ⇒ (False,t)
| True ⇒ (ebrk c, s) ⇒ (False,t(breakvar := 1)) ”
using assms(3,1,2)
proof (induction rule: big_step_induct)
case (WhileFalse b s c)
next
case (WhileTrue b s1 c s2 brk t)
next
case (WhileBreak b s1 c s2)
qed (auto split: bool.splits elim!: Seq1 simp: assign.simps)