Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
Dr. Peter Lammich

WS 2015/16
10. 11. 2015

Semantics of Programming Languages

Exercise Sheet 5

Exercise 5.1 Program Equivalence

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

1. *IF And b1 b2 THEN c1 ELSE c2* $\sim$ *IF b1 THEN IF b2 THEN c1 ELSE c2 ELSE c2*
2. *WHILE And b1 b2 DO c* $\sim$ *WHILE b1 DO WHILE b2 DO c*
3. *WHILE And b1 b2 DO c* $\sim$ *WHILE b1 DO c*;; *WHILE And b1 b2 DO c*
4. *WHILE Or b1 b2 DO c* $\sim$ *WHILE Or b1 b2 DO c*;; *WHILE b1 DO c*

Hint: Use the following definition for *Or*:

definition *Or* :: *"bexp ⇒ bexp ⇒ bexp"* **where**
"Or b1 b2 = Not (And (Not b1) (Not b2))"

Exercise 5.2 Nondeterminism

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

1. Modify the datatype *com* to include the new commands *OR* and *ASSUME*.
2. Adapt the big step semantics to include rules for the new commands.
3. Prove that c_1 *OR* $c_2 \sim c_2$ *OR* c_1.
4. Prove: (*IF b THEN c1 ELSE c2*) $\sim$ ((*ASSUME b*; *c1*) *OR* (*ASSUME* (*Not b*); *c2*))

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:

```
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:

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

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

```
fun break_ok :: "com ⇒ bool"
```

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

```
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)

```
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!

```
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_simp)
```