Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
Simon Wimmer

WS 2016/17
29. 11. 2016

Semantics of Programming Languages

Exercise Sheet 6

Exercise 6.1 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 6.2 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 6.1 Continue

Submission until Tuesday, December 6, 10:00am.

Note: This homework comes with a template file. You are strongly encouraged to use it!

Your task is to add a continue command to the IMP language. The continue command should skip all remaining commands in 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*)
    | *CONTINUE*

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

inductive
  big_step :: "$com \times state \Rightarrow bool \times state \Rightarrow bool$" (**infix** "$\Rightarrow$" *55*)

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

fun *continue_ok* :: "$com \Rightarrow bool$"

Show that the continue triggered-flag is not set after executing a well-formed command

lemma
  "$[\![(c,s) \Rightarrow (continue,t);\ continue\_ok\ c]\!] \Longrightarrow \neg continue$"

In the presence of *CONTINUE*, some additional sources of dead code arise. We want to eliminate those which can be identified syntactically (that is we do not want to analyze boolean expressions). For instance, the following holds:

lemma
  "$(IF\ b\ THEN\ CONTINUE\ ELSE\ CONTINUE;;\ c) \sim (CONTINUE)$"

Write a function *elim* that eliminates dead code caused by use of *CONTINUE*. You only need to contract commands because of *CONTINUE*, you do not need to eliminate *SKIPs*.

The following should hold for *elim*:

lemma
  "$elim\ c \sim c$"

Prove this direction:

lemma *elim_complete*:
  "$(c,\ s) \Rightarrow (b,\ s') \Longrightarrow (elim\ c,\ s) \Rightarrow (b,\ s')$"

BONUS: Also prove the converse direction:

```
lemma elim_sound:
  "(elim c, s) ⇒ (b, s') ⟹ (c, s) ⇒ (b, s')"

lemma
  "elim c ∼ c"
using elim_sound elim_complete by fast
```

Homework 6.2 Fuel your executions

Submission until Tuesday, December 6, 10:00am. Note: We provide a template for this homework on the lecture's homepage.

If you try to define a function to execute a program, you will run into trouble with the termination proof (The program might not terminate).

In this exercise, you will define an execution function that tries to execute the program for a bounded number of loop iterations. It gets an additional *nat* argument, called fuel, which decreases for every loop iteration. If the execution runs out of fuel, it stops returning *None*. We will work on the variant of IMP from the first exercise.

```
fun exec :: "com ⇒ state ⇒ nat ⇒ (bool × state) option" where
  "exec _ s 0 = None"
| "exec SKIP s f = Some (False, s)"
| "exec (x::=v) s f = Some (False, s(x:=aval v s))"
| "exec (c1;;c2) s f = (
    case exec c1 s f of
      None ⇒ None
    | Some (True, s') ⇒ Some (True, s')
    | Some (False, s') ⇒ exec c2 s' f)"
| "exec (IF b THEN c1 ELSE c2) s f =
    (if bval b s then exec c1 s f else exec c2 s f)"
| "exec (WHILE b DO c) s (Suc f) = (
    if bval b s then
      (case (exec c s f) of
        None ⇒ None |
        Some (cont, s') ⇒ exec (WHILE b DO c) s' f)
    else Some (False, s))"
| "exec CONTINUE s f = Some (True, s)"
```

Prove that the execution function is correct wrt. the big-step semantics:

```
theorem exec_equiv_bigstep: "(∃ i. exec c s f = Some s') ⟷ (c,s) ⇒ s'"
```

In the following, we give you some guidance for this proof. The two directions are proved separately. The proof of the first direction should be rather straightforward, and is left to you. Recall that is usually best to prove a statement for a (complex) recursive function using its specific induction rule (c.f. sect. 2.3.4 in the book), and that auto can automatically split "case"-expressions using the *split* attribute (c.f. sect. 2.5.6).

lemma *exec_imp_bigstep*: "$\mathit{exec}\ c\ s\ f\ =\ \mathit{Some}\ s' \Longrightarrow (c,s) \Rightarrow s'$"

For the other direction, prove a monotonicity lemma first: If the execution terminates with fuel *f*, it terminates with the same result using a larger amount of fuel $f' \geq f$. For this, first prove the following lemma:

lemma *exec_add*: "$\mathit{exec}\ c\ s\ f\ =\ \mathit{Some}\ s' \Longrightarrow \mathit{exec}\ c\ s\ (f\ +\ k) = \mathit{Some}\ s'$"

Only the WHILE-case requires some effort. Hint: Make a case distinction on the value of the condition *b*. You can find the proof for the easy cases in the template.

Now the monotonicity lemma that we want follows easily:

lemma *exec_mono*: "$\mathit{exec}\ c\ s\ f\ =\ \mathit{Some}\ (\mathit{brk},\ s') \Longrightarrow f' \geq f \Longrightarrow \mathit{exec}\ c\ s\ f' = \mathit{Some}\ (\mathit{brk},\ s')$"
by (*auto simp*: *exec_add dest*: *le_Suc_ex*)

The main lemma is proved by induction over the big-step semantics. Recall the adapted induction rule *big_step_induct* that nicely handles the pattern *big_step* (c,s) $(\mathit{brk},\ s')$. You can find the skip, while-true and if-true cases in the template. The other cases are left to you.

lemma *bigstep_imp_si*:
"$(c,s) \Rightarrow (\mathit{brk},\ s') \Longrightarrow \exists k.\ \mathit{exec}\ c\ s\ k\ =\ \mathit{Some}\ (\mathit{brk},\ s')$"

Finally, prove the main theorem of the homework:

theorem *exec_equiv_bigstep*: "$(\exists k.\ \mathit{exec}\ c\ s\ k\ =\ \mathit{Some}\ (\mathit{brk},\ s')) \longleftrightarrow (c,s) \Rightarrow (\mathit{brk},\ s')$"