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

WS 2011/12
18. 11. 2011

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* $\Rightarrow$ *bexp* $\Rightarrow$ *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 want to include a command c_1 *OR* c_2, which expresses the nondeterministic choice between two commands. That is, when executing c_1 *OR* c_2 either c_1 or c_2 may be executed, and it is not specified which one.

1. Modify the datatype *com* to include a new constructor *OR*.
2. Adapt the big step semantics to include rules for the new construct.
3. Prove that c_1 *OR* $c_2 \sim c_2$ *OR* c_1.
4. Adapt the small step semantics, and the equivalence proof of big and small step semantics.

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

Homework 5 Step-Index Semantics

Submission until Wednesday, November 30, 2011, 12:00 (noon).

Note: In order to save you some typing, we provide a template for this homework on the lecture's homepage.

In this homework, a denotational semantics for while-programs will be defined, i.e., a function that takes a command and a state, and returns the result state.

In order to make this function well-defined even for non-terminating programs, it is parameterized with an additional number, that indicates the maximum number of steps to make. If the program has not yet terminated after this many steps, *None* is returned.

```
fun si :: "com ⇒ state ⇒ nat ⇒ state option" where
  si_None: "si _ s 0 = None" |
  si_SKIP: "si SKIP s (Suc i) = Some s" |
  si_ASS: "si (x::=v) s (Suc i) = Some (s(x:=aval v s))" |
  si_SEMI: "si (c1;c2) s (Suc i) = (
    case (si c1 s i) of None ⇒ None | Some s' ⇒ si c2 s' i)" |
  si_IF: "si (IF b THEN c1 ELSE c2) s (Suc i) =
    (if bval b s then si c1 s i else si c2 s i)" |
  si_WHILE: "si (WHILE b DO c) s (Suc i) = (
    if bval b s then
      (case (si c s i) of
        None ⇒ None |
        Some s' ⇒ si (WHILE b DO c) s' i)
    else Some s)"
```

Prove the equivalence of the big-step and the step-index semantics, i.e., show that

$(\exists\, i.\ si\ c\ s\ i = Some\ s') \longleftrightarrow big\_step\ (c,s)\ s'$

As this proof is more complicated than any proof in homeworks so far, we will give a bit of guidance:

The two directions are proved separately. The proof of the first direction should be quite straightforward, and is left to you.

```
lemma si_imp_bigstep: "si c s i = Some s' ⟹ big_step (c,s) s'"
```

For the other direction, it is useful to prove a monotonicity lemma first. If the step-index semantics yields a result for index i, it yields the same result for any $i' \geq i$.

```
lemma si_mono: "si c s i = Some s' ⟹ si c s (i+k) = Some s'"
proof (induction c s i arbitrary: s'
    rule: si.induct[case_names None SKIP ASS SEMI IF WHILE])
  case (WHILE b c s i s') thus ?case
```

Only the WHILE-case requires some effort. Hint: Make a case distinction on the value of the condition b.

```
qed (auto split: option.split option.split_asm)
```

The main lemma is proved by induction over the big-step semantics. Remember the adapted induction rule *big_step_induct* that nicely handles the pattern *big_step* (c,s) s'.

```
lemma bigstep_imp_si:
  "big_step (c,s) s' ⟹ ∃i. si c s i = Some s'"
proof (induct rule: big_step_induct)
```

We demonstrate the skip, while-true and sequential composition case here. The other cases are left to you!

```
  case (Skip s) have "si SKIP s 1 = Some s" by auto
  thus ?case by blast
next
  case (WhileTrue b s1 c s2 s3)
  then obtain i1 i2 where "si c s1 i1 = Some s2"
    and "si (WHILE b DO c) s2 i2 = Some s3" by auto
  with si_mono[of c s1 i1 s2 i2]
    si_mono[of "WHILE b DO c" s2 i2 s3 i1] have
    "si c s1 (i1+i2) = Some s2"
    and "si (WHILE b DO c) s2 (i2+i1) = Some s3"
    by auto
  hence "si (WHILE b DO c) s1 (Suc (i1+i2)) = Some s3"
    using ‹bval b s1› by (auto simp add: add_ac)
  thus ?case by blast
next
  case (Semi c1 s1 s2 c2 s3)
  then obtain i1 i2 where "si c1 s1 i1 = Some s2" and "si c2 s2 i2 = Some s3"
    by auto
  with si_mono[of c1 s1 i1 s2 i2]
    si_mono[of c2 s2 i2 s3 i1]
  have
    "si c1 s1 (i1+i2) = Some s2" and "si c2 s2 (i2+i1) = Some s3"
    by auto
  hence "si (c1;c2) s1 (Suc (i1+i2)) = Some s3" by (auto simp add: add_ac)
  thus ?case by blast
```

Finally, prove the main theorem of the homework:

```
theorem si_equiv_bigstep: "(∃i. si c s i = Some s') ⟷ big_step (c,s) s'"
end
```