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

WS 2019/20
18.11.2019

Semantics of Programming Languages

Exercise Sheet 5

Exercise 5.1 Program Equivalence

Let *Or* be the disjunction of two *bexp*s:

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

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

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

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*) ∼ ((*ASSUME b*; *c1*) *OR* (*ASSUME* (*Not b*); *c2*))

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

Exercise 5.3 Deskip

Define a recursive function

fun *deskip* :: *"com ⇒ com"*

that eliminates as many *SKIP*s as possible from a command. For example:

deskip (*SKIP*;; *WHILE b DO* (*x* ::= *a*;; *SKIP*)) = *WHILE b DO x* ::= *a*

Prove its correctness by induction on c:

lemma
  assumes "(*WHILE b DO c*, *s*) ⇒ *t*" **and** "∀ *s t*. (*c*, *s*) ⇒ *t* ⟶ (*c'*, *s*) ⇒ *t*"
    shows "(*WHILE b DO c'*, *s*) ⇒ *t*"
lemma "*deskip c* ∼ *c*"

Homework 5.1 Functional Small-Step

Submission until Monday, Nov 25, 10:00am.

Specify a functional version of the small-step semantics as function *small* with the following signature:

fun *small* :: *"com ∗ state ⇒ (com ∗ state) option"* **where**

Prove that it is indeed equivalent to the small-step semantics:

theorem "(*c*,*s*) → (*c'*,*s'*) ⟷ *small* (*c*,*s*) = *Some* (*c'*,*s'*)"

Now define a version of *small* that corresponds to →∗. That is, define a function *smalls* with the following signature where the first argument gives an upper bound on the number of execution steps:

fun *smalls* :: *"nat ⇒ com ∗ state ⇒ (com ∗ state) option"* **where**

Again prove that the two semantics are equivalent:

```
theorem smalls_small_steps_equiv:
  "(∃ s'. (c,s) →∗ (c',s')) ⟷ (
    if c' = SKIP then
      (∃ n. smalls n (c, s) = None)
    else
      (∃ n s'. smalls n (c, s) = Some (c', s'))
  )"
```

Homework 5.2 Nondeterminism

Submission until Monday, Nov 25, 10:00am.

We again consider the extension of IMP with nondeterminism from the tutorial. This time, first extend the small-step semantics with the new constructs:

```
inductive
  small_step :: "com * state ⇒ com * state ⇒ bool" (infix "→" 55)
where
Assign:  "(x ::= a, s) → (SKIP, s(x := aval a s))" |
Seq1:    "(SKIP;;c₂,s) → (c₂,s)" |
Seq2:    "(c₁,s) → (c₁',s') ⟹ (c₁;;c₂,s) → (c₁';;c₂,s')" |
IfTrue:  "bval b s ⟹ (IF b THEN c₁ ELSE c₂,s) → (c₁,s)" |
IfFalse: "¬bval b s ⟹ (IF b THEN c₁ ELSE c₂,s) → (c₂,s)" |
While:   "(WHILE b DO c,s) → (IF b THEN c;; WHILE b DO c ELSE SKIP,s)" |
— Your cases here:
```

Then correct the proof of the equivalence theorem between big-step and small-step semantics:

```
theorem big_iff_small:
  "cs ⇒ t = cs →* (SKIP,t)"
```

Does the following theorem still hold? Prove or disprove! (Will not be checked by the submission system):

```
definition final where "final cs ⟷ ¬(EX cs'. cs → cs')"

lemma big_iff_small_termination:
  "(∃t. cs ⇒ t) ⟷ (∃cs'. cs →* cs' ∧ final cs')"
```