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

WS 2016/17
17. 1. 2017

Semantics of Programming Languages

Exercise Sheet 11

Exercise 11.1 Hoare Logic

In this exercise, you shall prove correct some Hoare triples.

First, write a program that stores the maximum of the values of variables a and b in variable c.

definition *MAX* :: *com* **where**

For the next task, you will need the following lemmas. Hint: Sledgehammering may be a good idea.

lemma [*simp*]: "$(a{::}int) < b \Longrightarrow max\ a\ b = b$"

lemma [*simp*]: "$\neg(a{::}int) < b \Longrightarrow max\ a\ b = a$"
by *auto*

Show that *MAX* satisfies the following Hoare-triple:

lemma "$\vdash \{\lambda s.\ True\}\ MAX\ \{\lambda s.\ s\ ''c'' = max\ (s\ ''a'')\ (s\ ''b'')\}$"

Now define a program *MUL* that returns the product of x and y in variable z. You may assume that y is not negative.

definition *MUL* :: *com* **where**

Prove that *MUL* does the right thing.

lemma "$\vdash \{\lambda s.\ 0 \leq s\ ''y''\}\ MUL\ \{\lambda s.\ s\ ''z'' = s\ ''x'' * s\ ''y''\}$"

Hints You may want to use the lemma *algebra_simps*, that contains some useful lemmas like distributivity.

Note that we use a backward assignment rule. This implies that the best way to do proofs is also backwards, i.e., on a semicolon $c_1;;\ c_2$, you first continue the proof for c_2, thus instantiating the intermediate assertion, and then do the proof for c_1. However, the first premise of the *Seq*-rule is about c_1. Hence, you may want to use the *rotated*-attribute, that rotates the premises of a lemma:

lemmas *Seq_bwd* = *Seq*[*rotated*]

lemmas *hoare_rule*[*intro?*] = *Seq_bwd Assign Assign′ If*

Note that our specifications still have a problem, as programs are allowed to overwrite arbitrary variables.

For example, regard the following (wrong) implementation of *MAX*:

definition *"MAX_wrong = (″a″::=N 0;;″b″::=N 0;;″c″::= N 0)"*

Prove that *MAX_wrong* also satisfies the specification for *MAX*:

What we really want to specify is, that *MAX* computes the maximum of the values of a and b in the initial state. Moreover, we may require that a and b are not changed.

For this, we can use logical variables in the specification. Prove the following more accurate specification for *MAX*:

lemma *"⊢ {λs. a=s ″a″ ∧ b=s ″b″}*
MAX
{λs. s ″c″ = max a b ∧ a = s ″a″ ∧ b = s ″b″}"

The specification for *MUL* has the same problem. Fix it!

Exercise 11.2 Forward Assignment Rule

Think up and prove a forward assignment rule, i.e., a rule of the form $\vdash \{P\}\ x{::=}a\ \{\ldots\}$, where $\ldots$ is some suitable postcondition. Hint: To prove this rule, use the completeness property, and prove the rule semantically.

lemmas *fwd_Assign′* = *weaken_post*[*OF fwd_Assign*]

Redo the proofs for *MAX* and *MUL* from the previous exercise, this time using your forward assignment rule.

lemma *"⊢ {λs. True} MAX {λs. s ″c″ = max (s ″a″) (s ″b″)}"*
lemma *"⊢ {λs. 0 ≤ s ″y″} MUL {λs. s ″z″ = s ″x″ ∗ s ″y″}"*

Homework 11.1 Hoare Logic OR

Submission until Tuesday, January 24, 2017, 10:00am.

Extend IMP with a new command c_1 *OR* c_2 that is a nondeterministic choice: it may execute either c_1 or c_2. Add the constructor

```
Or com com   ("_ OR/ _" [60, 61] 60)
```

to datatype *com* in theory *Com*, adjust the definition of the big-step semantics in theory *Big_Step*, add a rule for *OR* to the Hoare logic in theory *Hoare*, and adjust the soundness and completeness proofs in theory *Hoare_Sound_Complete*.

All these changes should be quite minimal and very local if you have got the definitions right.

Homework 11.2 Fixed point reasoning

Submission until Tuesday, January 24, 2017, 10:00am.

In the lecture, you have seen the Knaster-Tarski least fixed point theorem. The relevant constant is $lfp :: ('a \Rightarrow 'a) \Rightarrow 'a$, which assumes a complete lattice order $\leq$ on $'a$ and returns, for each monotonic operator $f :: 'a \Rightarrow 'a$, its least fixed point *lfp f*.

In the lectures as well as in this exercise, one only deals with the case where $'a$ is $'b$ *set* (the type of sets over an arbitrary type $'b$) and $\leq$ is $\subseteq$ (set inclusion). In this exercise, you will prove a different kind of fixed point theorem. It says that if there are two injective functions, one from $'a$ to $'b$, and one the other way round, then there also exists an bijection between $'a$ and $'b$:

```
theorem
  assumes "inj (f :: 'a ⇒ 'b)" and "inj (g :: 'b ⇒ 'a)"
  shows "∃ h :: 'a ⇒ 'b. inj h ∧ surj h"
```

This is a fixed point theorem because we will use a least fixed point for the construction of *h*. Use the provided template and follow the proof outline below to finish the proof.

```
theorem
  assumes "inj (f :: 'a ⇒ 'b)" and "inj (g :: 'b ⇒ 'a)"
  shows "∃ h :: 'a ⇒ 'b. inj h ∧ surj h"
proof
  def S ≡ "lfp (λX. − (g ‘ (− (f ‘ X))))"
  let ?g' = "inv g"
  def h ≡ "λz. if z ∈ S then f z else ?g' z"

  have "S = − (g ‘ (− (f ‘ S)))"

  have ∗: "?g' ‘ (− S) = − (f ‘ S)"

  show "inj h ∧ surj h"
  proof
    from ∗ show "surj h"
      have "inj_on f S"
      moreover have "inj_on ?g' (− S)"
    moreover
    { fix a b
      assume "a ∈ S" "b ∈ − S" and eq: "f a = ?g' b"
      have False    }
```

ultimately show *"inj h"*
qed
qed