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

WS 2019/20
16.12.2017

Semantics of Programming Languages

Exercise Sheet 09

Exercise 9.1 Hoare Logic

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

Step 1 Write a program that stores the maximum of the values of variables a and b in variable c.

definition *Max* :: *com* **where**

Step 2 Prove these lemmas about *max*:

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

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

Show that *ex09.Max* satisfies the following Hoare triple:

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

Step 3 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**

Step 4 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*, containing 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. In an Isar proof, this is no problem. In an **apply**-style proof, the ordering matters. Hence, you may want to use the [*rotated*] attribute:

lemmas *Seq_bwd = Seq[rotated]*

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

Step 5 Note that our specifications still have a problem, as programs are allowed to overwrite arbitrary variables.

For example, regard the following (wrong) implementation of *ex09.Max*:

definition "$MAX\_wrong = (''a''::=N\ 0;;''b''::=N\ 0;;''c''::=\ N\ 0)$"

Prove that *MAX_wrong* also satisfies the specification for *ex09.Max*:

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

What we really want to specify is, that *ex09.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 *ex09.Max*:

lemma "$\vdash \{\lambda s.\ a{=}s\ ''a'' \wedge b{=}s\ ''b''\}$
  Max
  $\{\lambda s.\ s\ ''c'' = max\ a\ b \wedge a = s\ ''a'' \wedge b = s\ ''b''\}$"

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

Exercise 9.2 Forward Assignment Rule

Think up and prove correct a forward assignment rule, i.e., a rule of the form $\vdash \{P\}\ x ::= a\ \{Q\}$, where Q 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 *ex09.Max* and *MUL* from the previous exercise, this time using your forward assignment rule.

lemma "$\vdash \{\lambda s.\ True\}\ Max\ \{\lambda s.\ s\ ''c'' = max\ (s\ ''a'')\ (s\ ''b'')\}$"
lemma "$\vdash \{\lambda s.\ 0 \leq s\ ''y''\}\ MUL\ \{\lambda s.\ s\ ''z'' = s\ ''x'' * s\ ''y''\}$"

Homework 9.1 Hoare Logic with Continue

Submission until Monday, January 13, 10:00am. (5 Points)

We again consider the extension of IMP with *CONTINUE*.

Your task is to adopt the rules of the Hoare calculus for partial correctness to this language and to prove the calculus sound and complete.

In addition to the previous predicate for validity $\models \{P\}c\{Q\}$, we will use a notion of validity $\models\{I\}\ \{P\}c\{Q\}$ that tracks the invariant of the surrounding While-loop, and a corresponding Hoare calculus $\vdash\{I\}\ \{P\}\ c\ \{Q\}$:

definition
hoare_valid :: *"assn ⇒ com ⇒ assn ⇒ bool"* (*"⊨ {(1_)}/ (_)/ {(1_)}" 50*) **where**
$``\models \{P\}c\{Q\} = (\forall s\ f\ t.\ P\ s \wedge (c,s) \Rightarrow (f,\ t) \ \longrightarrow\ \ Q\ t)"$

definition
$hoare\_valid_c$ :: *"assn ⇒ assn ⇒ com ⇒ assn ⇒ bool"* (*"$\models_c${(1_)}/ {(1_)}/ (_)/ {(1_)}" 50*)
where
$``\models_c\{I\}\ \{P\}c\{Q\} = (\forall s\ f\ t.\ P\ s \wedge (c,s) \Rightarrow (f,\ t) \ \longrightarrow (\textit{if } f \textit{ then } I\ t \textit{ else } Q\ t))"$

inductive
hoare :: *"assn ⇒ assn ⇒ com ⇒ assn ⇒ bool"* (*"⊢{(1_)}/ ({(1_)}/ (_)/ {(1_)})" 50*)
where
Skip: $``\vdash\{I\}\ \{P\}\ SKIP\ \{P\}"$ |
Assign: $``\vdash\{I\}\ \{\lambda s.\ P(s[a/x])\}\ x{::=}a\ \{P\}"$ |
Seq: $``[\![\vdash\{I\}\ \{P\}\ c_1\ \{Q\};\ \vdash\{I\}\ \{Q\}\ c_2\ \{R\}]\!]$
$\Longrightarrow \vdash\{I\}\ \{P\}\ c_1;;c_2\ \{R\}"$ |
If: $``[\![\vdash\{I\}\ \{\lambda s.\ P\ s \wedge bval\ b\ s\}\ c_1\ \{Q\};\ \vdash\{I\}\ \{\lambda s.\ P\ s \wedge \neg\ bval\ b\ s\}\ c_2\ \{Q\}]\!]$
$\Longrightarrow \vdash\{I\}\ \{P\}\ IF\ b\ THEN\ c_1\ ELSE\ c_2\ \{Q\}"$ |
conseq: $``[\![\forall s.\ P'\ s \longrightarrow P\ s;\ \vdash\{I\}\ \{P\}\ c\ \{Q\};\ \forall s.\ Q\ s \longrightarrow Q'\ s]\!]$
$\Longrightarrow \vdash\{I\}\ \{P'\}\ c\ \{Q'\}"$ |
— Your cases here:

Complete the definition of the Hoare calculus!

Prove soundness of the calculus:

theorem *hoare_sound*: $``\vdash\{I\}\ \{P\}c\{Q\} \Longrightarrow \models_c\{I\}\ \{P\}c\{Q\}"$

In analogy to $\models_c\{I\}\ \{P\}c\{Q\}$, define the the weakest precondition *wp c I Q* of program *c*:

definition *wp* :: *"com ⇒ assn ⇒ assn ⇒ assn"* **where**

Prove the following theorem, which establishes completeness of the calculus:

theorem *wp_is_pre*: $``\vdash\{I\}\ \{wp\ c\ I\ Q\}\ c\ \{Q\}"$

Finally show that the calculus is sound and complete:

theorem *hoare_sound_complete*: $``\vdash\{Q\}\ \{P\}c\{Q\} \longleftrightarrow \models \{P\}c\{Q\}"$

Hints: Use the theory *HOL−IMP.Hoare_Sound_Complete* as a template for your proofs. For soundness, you will need a lemma about the state of the flag after executing a while-loop. For completeness, it may be easier to not attempt to prove a variant of *wp_While_If* directly, but rather to figure out what variants of *wp_While_True* and *wp_While_False* are needed, and then to prove them directly.

Homework 9.2 Be Original!

Submission until Monday, January 13, 2019, 10:00am. (15 regular points, plus bonus points for nice submissions)

Think up a nice formalization yourself, for example

- Prove some interesting result about algorithms/graphs/automata/formal language theory
- Formalize some results from mathematics
- Find interesting modifications of IMP material and prove interesting properties about them
- ...

You should set yourself a time limit before starting your project. Also incomplete/unfinished formalizations are welcome and will be graded!

Please comment your formalization well, such that we can see what it does/is intended to do.

You are welcome to discuss your plans with the tutor (via e-mail) before starting your project. This is, however, not a necessity by any means.