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

WS 2011/12
11. 12. 2012

Semantics of Programming Languages

Exercise Sheet 9

Exercise 9.1 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*]: "$\neg(a{::}int) < b \Longrightarrow max\ a\ b = a$"

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 $S_1;\ S_2$, you first continue the proof for S_2, thus instantiating the intermediate assertion, and then do the proof for S_1. However, the first premise of the *Seq*-rule is about S_1. Hence, you may want to use the *rotated*-attribute, that rotates the premises of a lemma:

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

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 "$MAX\_wrong \equiv ''a''::=N\ 0;\ ''b''::=N\ 0;\ ''c''::=N\ 0$"

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 "$\vdash \{\lambda s.\ a{=}s\ ''a'' \wedge b{=}s\ ''b''\}$
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!

Homework 9.1 Making programs more public

Submission until Wednesday, December 18, 2012, 12:00 (noon).

In this homework, you need to define a function

fun *public* :: *"level ⇒ com ⇒ com"*

that removes all assignments to confidential variables. That is, *public l c* should replace all assignments $x ::= a$ by *SKIP* if $l < sec\ x$. In fact, you can also remove certain *IF*s and *WHILE*s (but please, not all of them!), which simplifies the proof below. Now show that c and *public l c* behave the same on the variables up to l:

theorem *noninterference*:
"$\llbracket\ (c,s) \Rightarrow s'$; $(public\ l\ c,t) \Rightarrow t'$; $0 \vdash c$; $s = t\ (< l)\ \rrbracket$
$\Longrightarrow s' = t'\ (< l)$"

Hint: The name of the lemma indicates that it is very similar to the noninterference lemma in *Sec_Typing*. (Note however that, unlike in that lemma, here we use strict inequality.) You may want to start with that proof and modify it where needed. A lot of local modifications will be necessary, but the structure should remain the same. You may also need one or two simple additional lemmas (for example $\ldots \Longrightarrow aval\ a\ s_1 = aval\ a\ s_2$), but nothing major.

EXTRA CREDIT TASK: For 4 additional points, prove the following confinement lemma, which states that the execution of an l-modified program does not affect the variables of security level above l:

lemma *confinement*: "$(public\ l\ c,\ s) \Rightarrow t \Longrightarrow sec\ x \geq l \Longrightarrow t\ x = s\ x$"
proof (*induction "public l c" s t arbitrary*: *l c rule*: *big_step_induct*)