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 $MAX :: c = \lambda s. \max (a \colon \mathbb{int}) (b \colon \mathbb{int})$

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

lemma [simp]: "(a::int)<b \Rightarrow \max a b = b"
lemma [simp]: "\neg (a::int)<b \Rightarrow \max a b = a"

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

lemma "\vdash \{ \lambda s. True \} MAX \{ \lambda s. c = \max (a :: \mathbb{int}) (b :: \mathbb{int}) \}"

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 $MUL :: z = \lambda s. x \cdot y$

Prove that $MUL$ does the right thing.

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

Hints You may want to use the lemma $\text{algebra\_sims}$, 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 $\text{rotated}$-attribute, that rotates the premises of a lemma:

lemmas $Seq\_bwd = Seq[\text{rotated}]$
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 ≡ "'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. "'c" = max a b ∧ a = s "'a" ∧ 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 `IFS` 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:**
"\[
[(c,s) \Rightarrow s'; (public l c,t) \Rightarrow t'; \ 0 \vdash c; \ 0 \vdash s = t (< l) ]
\]
\[\implies s' = t' (< l)\]"

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 ... \[\implies aval a s_1 = aval a s_2\]), 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 \implies sec x \geq l \implies t x = s x"

**proof** (induction "public l c" s t arbitrary; l c rule: big_step_induct)