Exercise 11.1 Using the VCG

Use the VCG to prove correct a multiplication and a square root program:

```plaintext
definition MUL :: com
where
  "MUL =
    "z" := N 0 ;;
    "c" := N 0 ;;
  WHILE (Less (V "c") (V "y")) DO ( 
    "z" := Plus (V "z") (V "x");
    "c" := Plus (V "c") (N 1) )
)

lemma "¬-
  { As. 0 ≤ s "y" ∧ s = sorig }
  MUL
  { As. s "z" = s "x" * s "y" ∧ (∀ v. v[f \{"z","c"\} → s v = sorig v}) }

definition SQRT :: com
where
  "SQRT =
    "r" := N 0 ;;
    "s" := N 1 ;;
  WHILE (Not (Less (V "z") (V "s"))) DO ( 
    "r" := Plus (V "r") (N 1) ;;
    "s" := Plus (V "s") (V "r");;
    "s" := Plus (V "s") (V "r") ;;
    "s" := Plus (V "s") (N 1) 
  )

lemma "¬-
  { As. s = sorig ∧ s "x" ≥ 0 }
  SQRT
  { As. (s "r")^2 ≤ s "z" ∧ s "s" < (s "r" + 1)^2 ∧ (∀ v. v[f \{"s","r"\} → s v = sorig v}) }
```

Exercise 11.2 Hoare Logic \textit{OR}

Extend IMP with a new command $c_1 \text{ OR } c_2$ that is a nondeterministic choice: it may execute either $c_1$ or $c_2$. Add the constructor

```plaintext"
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 got the definitions right.

Homework 11.1 A Hoare Calculus with Execution Times

Submission until Tuesday, January 16, 2018, 10:00am.

In this homework, we will consider a Hoare calculus with execution times.

Hint: Use the template provided on the website.

Step 1 We first give a modified big-step semantics to account for execution times. A judgement of the form (c, s) ⇒ n ↓ t has the intended meaning that we can get from state s to state t by an terminating execution of program c that takes exactly n time steps.

inductive
big_step :: "com × state ⇒ nat ⇒ state ⇒ bool" ("_⇒_" 55)
where
Skip: "(SKIP, s) ⇒ Suc 0 ↓ s" |
Assign: "(x ::= a, s) ⇒ Suc 0 ↓ s(x := aval a s)" |
Seq: "[(c1, s1) ⇒ x ↓ s2; (c2, s2) ⇒ y ↓ s3 ; z=x+y ] ⇒ (c1;c2, s1) ⇒ z ↓ s3" |
IfTrue: "[ bval b s; (c1, s) ⇒ x ↓ t; y=x+1 ] ⇒ (IF b THEN c1 ELSE c2, s) ⇒ y ↓ t" |
IfFalse: "[ ¬bval b s; (c2, s) ⇒ x ↓ t; y=x+1 ] ⇒ (IF b THEN c1 ELSE c2, s) ⇒ y ↓ t" |
WhileFalse: "[ ¬bval b s ] ⇒ (WHILE b DO c, s) ⇒ Suc 0 ↓ s" |
WhileTrue:
"[ bval b s1; (c,s) ⇒ x ↓ s2; (WHILE b DO c, s2) ⇒ y ↓ s3; 1+x+y=z ]
⇒ (WHILE b DO c, s1) ⇒ z ↓ s3"

Step 2 Some theoretical background: We need extended natural numbers. These are provided by the Extended_Nat theory. We can imagine extended natural numbers as the union of all natural numbers N and ∞. Here are some examples to illustrate their arithmetic behaviour:

value "3::enat" — 3
value "∞::enat" — ∞
value "(3::enat) + 4" — 7
value "(3::enat) + ∞" — ∞
value "eSuc 3" — 4
value "eSuc ∞" — ∞
**Step 3** Next, we define a Hoare calculus that also accounts for execution times. Assertions are still the same (of type \textit{state} \Rightarrow \textit{bool}), but we introduce new quantitative assertions of type \textit{state} \Rightarrow \textit{enat}.

\begin{verbatim}
type synonym assn = "state \Rightarrow bool"

It is thought that the result of a \textit{gassn} represents a potential, where \(\infty\) corresponds to a \textit{False} assertion in classical Hoare calculus. We can hence embed assertions into quantitative assertions:

\begin{verbatim}
fun emb :: "bool \Rightarrow enat" ("\downarrow") where
  "emb False = \infty"
| "emb True = 0"
\end{verbatim}

We can define what it means for a quantitative Hoare triple to be valid:

\begin{verbatim}
definition hoareQvalid :: "gassn \Rightarrow com \Rightarrow gassn \Rightarrow bool"
  ("\models\{\}\) ((\(\infty\))/ (\(\infty\))/ ((\(\infty\))/ 50)) where
  "\models\{\}\ {P} {Q} \iff (\forall s. \ P s < \infty \implies (\exists t p. ((c,s) \Rightarrow p \downarrow t) \land \ P s \geq p + Q t))"
\end{verbatim}

Finally, we define quantitative Hoare judgements. The idea is that both pre- and post-condition assign an \textit{enat} to a state that is then decreased as the execution progresses. We will see an example in the next step.

\begin{verbatim}
inductive hoareQ :: "gassn \Rightarrow com \Rightarrow gassn \Rightarrow bool" ("\models\{\}\) ((\(\infty\))/ (\(\infty\))/ ((\(\infty\))/ 50)) where

  Skip: "\models\{\}\ {\lambda s. cSuc (P s)} {SKIP} {P}" \\
  Assign: "\models\{\}\ {\lambda s. cSuc (P (s[a/x]))}) {x::=a} {P}" \\

  IF \_ THEN \_ ELSE \_ is a bit tricky: We decrease the potential by one before executing either branch. Then we add 0 to the branch that gets executed and \(\infty\) to the branch that does not get executed. This is similar to how in classical Hoare calculus, the branch that does not get executed gets \textit{False} as precondition.

  If: "[\models\{\}\ \{\lambda s. P s + \downarrow(\text{bval b s})\} \ c \ {Q}];
  \models\{\}\ {\lambda s. P s + \downarrow(\neg \text{bval b s})\} \ c \ {Q}];
  \implies \models\{\}\ {\lambda s. cSuc (P s)} IF b THEN c1 ELSE c2 \ {Q}" \\

  Sequence works about as expected.

  Seq: "[\models\{\}\ {P1}; \models\{\}\ {P2}; \models\{\}\ {P3}] \implies \models\{\}\ {P1};c1;c2 {P3}"

  WHILE \_ DO \_ is a combination of conditional and sequence. The invariant is also a function to \textit{enat}.

  While:
  "\models\{\}\ \{\lambda s. I s + \downarrow(\text{bval b s})\} c \ {\lambda t. I t + 1};
  \models\{\}\ {\lambda s. I s + 1} WHILE b DO c \ {\lambda s. I s + \downarrow(\neg \text{bval b s})}"

  The consequence rule also works like in the classic Hoare calculus.

  conseq: "[\models\{\}\ {P} {Q}; \\lambda s. P s \leq P' s; \\lambda s. Q' s \leq Q s] \implies"
\end{verbatim}
⊢ \{P\} \ c \ {Q\}"

**Step 4**  To exercise our newly-introduce Hoare calculus with timing, we will prove a Hoare triple for an example program that computes the sum of numbers from 1 to \(n\). However, we are only interested in computing the total runtime and disregard correctness properties.

**fun** `sum :: “int ⇒ int” where
“sum i = (if i ≤ 0 then 0 else sum (i - 1) + i)”

**definition** `wsum :: com where
“wsum = "y" := N 0;;
WHILE Less (N 0) (V "x")
DO ("y" := Plus (V "y") (V "x");;
"x" := Plus (V "x") (N (- 1)))”

The following lemma states the the `wsum` program will take at most \(2 + 3 \cdot n\) steps to complete. Prove it!

**lemma** `wsum: “\+\_Q \ {λs. enat (2 + 3*n) + \downarrow (s "x" = int n)} {wsum \ {λs. 0}}”

**unfolding** `wsum_def`
**apply**(rule `Seq[rotated]`)
**apply**(rule `conseq`)
**apply**(rule `While[where I="λs. enat (3 * nat (s "x"))"]`)

**Step 5**  Your task is to prove a fragment of the soundness theorem, namely for sequences.

**theorem** `hoareQ_sound: “\+\_Q \ {P\} \ c \ {Q\} \implies \+\_Q \ {P\} \ c \ {Q\}”

**proof**(induction rule: `hoareQ.induct`)
  case (Skip P)
  — Proven already.
  **show** ?case next
  case (Seq P₁ c₁ P₂ c₂ P₃)
  — Prove this as a lemma: \([\+\_Q \ {P₁} \ c₁ \ {P₂}; \+\_Q \ {P₂} \ c₂ \ {P₃}] \implies \+\_Q \ {P₁} \ c₁;; \ c₂ \ {P₃}\)
  then **show** ?case
  using `Seq_sound` by auto
next
— For bonus points, prove the remaining cases.

**qed**