Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
Lars Hupel

WS 2017/18
09.01.2018

Semantics of Programming Languages

Exercise Sheet 11

Exercise 11.1 Using the VCG

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

```
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 "⊢
  {λs. 0 ≤ s ''y'' ∧ s=sorig}
  MUL
  {λs. s ''z'' = s ''x'' * s ''y'' ∧ (∀ v. v∉{''z'',''c''} ⟶ s v = sorig v)}"

definition SQRT :: com where
"SQRT =
  ''r'' ::= N 0;;
  ''s'' ::= N 1;;
  WHILE (Not (Less (V ''x'') (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 "⊢
  {λs. s=sorig ∧ s ''x'' ≥ 0}
  SQRT
  {λs. (s ''r'')^2 ≤ s ''x'' ∧ s ''x'' < (s ''r''+1)^2 ∧ (∀ v. v∉{''s'',''r''} ⟶ s v = sorig v)}"
```

Exercise 11.2 Hoare Logic *OR*

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 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 $(c,\ s) \Rightarrow n \Downarrow t$ 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_t :: "$com \times state \Rightarrow nat \Rightarrow state \Rightarrow bool$" ("$\_ \Rightarrow \_ \Downarrow \_$" 55)
where
Skip: "$(SKIP,s) \Rightarrow Suc\ 0 \Downarrow s$" |
Assign: "$(x ::= a,s) \Rightarrow Suc\ 0 \Downarrow s(x := aval\ a\ s)$" |
Seq: "$\llbracket\ (c1,s1) \Rightarrow x \Downarrow s2;\ \ (c2,s2) \Rightarrow y \Downarrow s3\ ;\ z=x+y\ \rrbracket \Longrightarrow (c1;;c2,\ s1) \Rightarrow z \Downarrow s3$" |
IfTrue: "$\llbracket\ bval\ b\ s;\ \ (c1,s) \Rightarrow x \Downarrow t;\ y=x+1\ \rrbracket \Longrightarrow (IF\ b\ THEN\ c1\ ELSE\ c2,\ s) \Rightarrow y \Downarrow t$" |
IfFalse: "$\llbracket\ \neg bval\ b\ s;\ \ (c2,s) \Rightarrow x \Downarrow t;\ y=x+1\ \rrbracket \Longrightarrow (IF\ b\ THEN\ c1\ ELSE\ c2,\ s) \Rightarrow y \Downarrow t$" |
WhileFalse: "$\llbracket\ \neg bval\ b\ s\ \rrbracket \Longrightarrow (WHILE\ b\ DO\ c,s) \Rightarrow Suc\ 0 \Downarrow s$" |
WhileTrue:
"$\llbracket\ bval\ b\ s1;\ \ (c,s1) \Rightarrow x \Downarrow s2;\ \ (WHILE\ b\ DO\ c,\ s2) \Rightarrow y \Downarrow s3;\ 1+x+y=z\ \rrbracket$
$\Longrightarrow (WHILE\ b\ DO\ c,\ s1) \Rightarrow z \Downarrow s3$"

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 $\mathbb{N}$ and ∞. Here are some examples to illustrate their arithmetic behaviour:

value "$3::enat$" — 3
value "$\infty::enat$" — ∞
value "$(3::enat) + 4$" — 7
value "$(3::enat) + \infty$" — ∞
value "$eSuc\ 3$" — 4
value "$eSuc\ \infty$" — ∞

Step 3 Next, we define a Hoare calculus that also accounts for execution times. Assertions are still the same (of type *state* ⇒ *bool*), but we introduce new *quantitative assertions* of type *state* ⇒ *enat*.

type_synonym *assn* = *"state ⇒ bool"*
type_synonym *qassn* = *"state ⇒ enat"*

It is thought that the result of a *qassn* represents a *potential*, where ∞ corresponds to a *False* assertion in classical Hoare calculus. We can hence embed assertions into quantitative assertions:

fun *emb* :: *"bool ⇒ enat"* ("↓") **where**
  "emb False = ∞"
| *"emb True = 0"*

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

definition *hoare_Qvalid* :: *"qassn ⇒ com ⇒ qassn ⇒ bool"*
  ("$\models_Q$ {(1_)}/ (_)/ {(1_)}" 50) **where**
"$\models_Q \{P\}\ c\ \{Q\} \longleftrightarrow (\forall s.\ \ P\ s < \infty \longrightarrow (\exists\, t\ p.\ ((c,s) \Rightarrow p \Downarrow t) \land P\ s \geq p + Q\ t))$"

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

inductive *hoareQ* :: *"qassn ⇒ com ⇒ qassn ⇒ bool"* ("$\vdash_Q$ ({(1_)}/ (_)/ {(1_)})" 50) **where**

— Skipping and assignment both decrease the potential.
Skip: "$\vdash_Q \{\lambda s.\ eSuc\ (P\ s)\}\ SKIP\ \{P\}$" |
Assign: "$\vdash_Q \{\lambda s.\ eSuc\ (P\ (s[a/x]))\}\ x{::=}a\ \{P\}$" |

— *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 ∞ 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 *False* as precondition.
If: "$[\![\vdash_Q \{\lambda s.\ P\ s + {\downarrow}(\ bval\ b\ s)\}\ c_1\ \{Q\};$
$\vdash_Q \{\lambda s.\ P\ s + {\downarrow}(\neg\ bval\ b\ s)\}\ c_2\ \{Q\}\ ]\!]$
$\Longrightarrow \vdash_Q \{\lambda s.\ eSuc\ (P\ s)\}\ IF\ b\ THEN\ c_1\ ELSE\ c_2\ \{Q\}$" |

— Sequence works about as expected.
Seq: "$[\![\vdash_Q \{P_1\}\ c_1\ \{P_2\}; \vdash_Q \{P_2\}\ c_2\ \{P_3\}]\!] \Longrightarrow \vdash_Q \{P_1\}\ c_1;;c_2\ \{P_3\}$" |

— *WHILE _ DO _* is a combination of conditional and sequence. The invariant is also a function to *enat*.
While:
"$\vdash_Q \{\lambda s.\ I\ s + {\downarrow}(bval\ b\ s)\}\ c\ \{\lambda t.\ I\ t + 1\}$
$\Longrightarrow \vdash_Q \{\lambda s.\ I\ s + 1\}\ WHILE\ b\ DO\ c\ \{\lambda s.\ I\ s + {\downarrow}(\neg\ bval\ b\ s)\}$" |

— The consequence rule also works like in the classic Hoare calculus.
conseq: "$[\![\vdash_Q \{P\}\ c\ \{Q\};\ \bigwedge s.\ P\ s \leq P'\ s;\ \bigwedge s.\ Q'\ s \leq Q\ s\ ]\!] \Longrightarrow$

$\vdash_Q \{P'\}\ c\ \{Q'\}$"

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 $2 + 3 * n$ steps to complete. Prove it!

lemma *wsum*: "$\vdash_Q \{\lambda s.\ enat\ (2 + 3{*}n) + \downarrow (s\ ''x'' = int\ n)\}\ wsum\ \{\lambda s.\ 0\}$"
unfolding *wsum_def*
apply(*rule Seq*[*rotated*])
 apply(*rule conseq*)
  apply(*rule While*[**where** *I*="λ*s*. *enat* (*3* ∗ *nat* (*s* ''*x*''))"])

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

theorem *hoareQ_sound*: "$\vdash_Q \{P\}\ c\ \{Q\} \Longrightarrow\ \models_Q \{P\}\ c\ \{Q\}$"
proof(*induction rule*: *hoareQ.induct*)
  case (*Skip P*)
  — Proven already.
  show *?case* **next**
  case (*Seq* P_1 c_1 P_2 c_2 P_3)
  — Prove this as a lemma: $[\![\models_Q \{P_1\}\ c_1\ \{P_2\};\ \models_Q \{P_2\}\ c_2\ \{P_3\}]\!] \Longrightarrow\ \models_Q \{P_1\}\ c_1;;\ c_2\ \{P_3\}$
  then show *?case*
    using *Seq_sound* **by** *auto*
next
  — For bonus points, prove the remaining cases.
  qed