Final Exam

Semantics

14.02.2020

First name: ______________________

Last name: ______________________

Student-Id (Matrikelnummer): ______________________

Signature: ______________________

1. You may only use a pen/pencil, eraser, and one A4 sheet of notes to solve the exam. Switch off your mobile phones!
2. Please use the empty pages to answer the questions, preferably. If this space is not sufficient, you may use the extra sheets that have been handed out. Please state it clearly, below the question, when doing so.
3. You have 120 minutes to solve the exam.
4. Please put your student ID and ID-card or driver's license on the table until we have checked it.
5. Please do not leave the room in the last 15 minutes of the exam.

Proof Guidelines: We expect detailed, rigorous, mathematical proofs — but we do not ask you to write Isabelle proof scripts! You are welcome to use standard mathematical notation; you do not need to follow Isabelle syntax. Proof steps should be explained in ordinary language like a typical mathematical proof.

Major proof steps, especially inductions, need to be stated explicitly. For each case of a proof by induction, you must list the **inductive hypotheses** assumed (if any), and the **goal** to be proved.

Minor proof steps (corresponding to by simp, by blast etc) need not be justified if you think they are obvious, but you should say which facts they follow from. You should be explicit whenever you use a function definition or an introduction rule for an inductive predicate — especially for functions and predicates that are specific to an exam question. (You need not reference individual lemmas for standard concepts like integer arithmetic, however, and in any case we do not ask you to recall lemma names from any Isabelle theories.)

1 Induction

Question 1 Consider the following inductive definition:

```
inductive M :: "nat => bool" where
"M (n+2) ==> M n" |
"M n ==> M (n+5) ==> M (n+1)"
```

a) Write down the Isabelle proof state after the following two commands:

```
lemma "M a ==> a = 7"
apply(induction a rule: M.induct)
```

b) Is the formula `M a ==> False` provable? Give a short justification of your answer.

Question 2 Consider the following function definitions:

```
fun f :: "nat => nat" where
"f 0 = 0" |
"f (Suc 0) = 1" |
"f (Suc(Suc n)) = f n + f (Suc n)"
```

```
fun g :: "nat => nat => nat => nat" where
"g a b 0 = b" |
"g a b (Suc n) = g b (a+b) n"
```

Find terms `?1` and `?2` sucht that the equation

```
g a b n = f n * ?1 + f (Suc n) * ?2
```

becomes true. Give a proof of the resulting equation.

1.1 Solution

Question 1
lemma $M\ a \Longrightarrow a = 7$
apply (*induction a rule*: *M.induct*)

$1.\ \bigwedge n.\ M\ (n + 2) \Longrightarrow n + 2 = 7 \Longrightarrow n = 7$
$2.\ \bigwedge n.\ M\ n \Longrightarrow n = 7 \Longrightarrow M\ (n + 5) \Longrightarrow n + 5 = 7 \Longrightarrow n + 1 = 7$

oops

We define an empty predicate, thus the induction rule for M allows us to prove the theorem.

lemma $M\ a \Longrightarrow False$
by (*erule M.induct*)

Question 2 Instantiation: ?1 = a and ?2 = b

lemma
$g\ a\ b\ n = f\ n * a + f\ (Suc\ n) * b$
proof (*induction n arbitrary*: *a b*)
— Base case
fix $a\ b$ **show** $g\ a\ b\ 0 = f\ 0 * a + f\ (Suc\ 0) * b$
by *auto*
next
— Induction step
fix $n\ a\ b$
assume IH: $\bigwedge a\ b.\ g\ a\ b\ n = f\ n * a + f\ (Suc\ n) * b$
have $g\ a\ b\ (Suc\ n) = g\ b\ (a + b)\ n$
by (*rule g.simps*)
also have $\ldots = f\ n * b + f\ (Suc\ n) * (a + b)$
by (*rule IH*)
also have $\ldots = f\ (Suc\ n) * a + (f\ n + f\ (Suc\ n)) * b$
by *algebra*
also have $\ldots = f\ (Suc\ n) * a + f\ (Suc\ (Suc\ n)) * b$
unfolding *f.simps* **..**
finally show $g\ a\ b\ (Suc\ n) = f\ (Suc\ n) * a + f\ (Suc\ (Suc\ n)) * b$ **.**
qed

2 While-Loops

Consider the the big-step semantics of Imp. We will denote the set of variables of a command c by $vars\ c$.

Question 1 Show:

$$(WHILE\ b\ DO\ c,\ s) \Rightarrow t \Longrightarrow vars\ c\ \cap\ vars\ b = \{\} \Longrightarrow \neg\ bval\ b\ s$$

Hint: You may use the following fact:

$$vars\ c\ \cap\ vars\ b = \{\} \Longrightarrow (c, s) \Rightarrow t \Longrightarrow bval\ b\ s \longleftrightarrow bval\ b\ t \tag{1}$$

Question 2 Define a function $no\ ::\ com\ \Rightarrow\ bool$ such that $no\ c$ holds if and only if c contains no while loops. Show:

$$no\ c \Longrightarrow \forall s.\ \exists t.\ (c, s) \Rightarrow t$$

Hint: You may skip the cases for $SKIP$ and assignment when performing an induction.

2.1 Solution

lemma *aux*:
assumes *1*: *vars c* ∩ *vars b* = {} **and** *2*: (*c*, *s*) ⇒ *t*
shows *bval b s* ⟷ *bval b t*

lemma *ex1*: (*WHILE b DO c*, *s*) ⇒ *t* ⟹ *vars c* ∩ *vars b* = {} ⟹ ¬ *bval b s*
proof (*induction WHILE b DO c s t rule*: *big_step_induct*)
  case *WhileFalse*
  thus *?case* **by** *simp*
next
  case (*WhileTrue s1 s2 s3*)
  from ‹*bval b s1*› ‹*vars c* ∩ *vars b* = {}› ‹(*c*, *s1*) ⇒ *s2*› *aux* **have** *bval b s2* **by** *auto*
  with *WhileTrue*(*5*) ‹*vars c* ∩ *vars b* = {}› **show** *?case* **by** *auto*
qed

fun *no* :: *com* ⇒ *bool* **where**
 no SKIP ⟷ *True*
|*no* (*x* ::= *a*) ⟷ *True*
|*no* (*c1* ;; *c2*) ⟷ *no c1* ∧ *no c2*
|*no* (*IF b THEN c1 ELSE c2*) ⟷ *no c1* ∧ *no c2*
|*no* (*WHILE b DO c*) ⟷ *False*

lemma *ex2*: *no c* ⟹ ∀ *s*. ∃ *t*. (*c*, *s*) ⇒ *t*
proof(*induction c*)
  case (*Seq c1 c2*)
  show *?case* **proof** *safe*
    fix *s*
    have *c1*: *no c1* **and** *c2*: *no c2* **using** *Seq.prems* **by** *simp_all*
    from *c1* **obtain** *t1* **where** (*c1*,*s*) ⇒ *t1* **using** *Seq.IH*(*1*) **by** *auto*
    moreover from *c2* **obtain** *t* **where** (*c2*,*t1*) ⇒ *t* **using** *Seq.IH*(*2*) **by** *auto*
    ultimately have (*c1*;; *c2*, *s*) ⇒ *t* **by** (*auto intro*: *big_step.intros*)
    thus ∃ *t*. (*c1*;; *c2*, *s*) ⇒ *t* **by** *blast*
  qed
next
  case (*If b c1 c2*)
  show *?case* **proof** *safe*
    fix *s*
    have *c1*: *no c1* **and** *c2*: *no c2* **using** *If.prems* **by** *simp_all*
    show ∃ *t*. (*IF b THEN c1 ELSE c2*, *s*) ⇒ *t*
    proof(*cases bval b s*)
      case *True* **thus** *?thesis* **using** *If.IH*(*1*)[*OF c1*] **by** *auto*
    next
      case *False* **thus** *?thesis* **using** *If.IH*(*2*)[*OF c2*] **by** *auto*
    qed
  qed
qed *auto*

3 Hoare-Logic

We replace the assignment in IMP by a command *REL R* that performs an arbitrary state transition according to relation $R :: (state \times\ state)\ set$.
In the big-step semantics, we remove the *assign*-rule, and add the following rule:

Rel: $(s,s') \in R \implies (REL\ R,s) \Rightarrow s'$

Questions

1. Is the semantics deterministic, i.e., does the following hold (proof or counterexample):

   $(c,s) \Rightarrow t \implies (c,s) \Rightarrow t' \implies t{=}t'$

2. What does the weakest precondition $wp\ (REL\ R)\ Q$ look like?
3. Specify a Hoare-rule for *REL*.
4. Prove: $\vdash \{wp\ (REL\ R)\ Q\}\ REL\ R\ \{Q\}$.

Hints

- Question 2: Recall the definition of the weakest precondition:

  $wp\ c\ Q\ =\ (\lambda s.\ \forall t.\ (c,s) \Rightarrow t \longrightarrow Q\ t)$

  Now we want an equation that shows how to expand *wp* syntactically, i.e., the right hand side should not contain the Big/Small-step semantics. You need **not** prove your equation here.

- Question 4: The main lemma in the completeness proof of Hoare logic is $\vdash \{wp\ c\ Q\}\ c\ \{Q\}$. Here you have to prove the case for the *REL*-command. Use your equation for *wp* from Question 2 here!

3.1 Solution

1. No! We have, e.g., $\forall s'.\ (REL\ UNIV, s) \Rightarrow s'$, and there exists more than one state s'.

2. $wp\ (REL\ R)\ Q = \lambda s.\ \forall s'.\ (s,s') \in R \longrightarrow Q\ s'$

3. $\vdash \{\lambda s.\ \forall s'.\ (s,s') \in R \longrightarrow Q\ s'\}\ REL\ R\ \{Q\}$

4. Using 2), we have to prove: $\vdash \{\lambda s.\ \forall s'.\ (s,s') \in R \longrightarrow Q\ s'\}\ REL\ R\ \{Q\}$ Which matches exactly the Hoare-rule for REL.

4 Abstract Interpretation

IMP is extended by adding a multiplication operator, and restricted to only compute with natural numbers:

datatype *aexp* = *N nat* | *V vname* | *Plus aexp aexp* | *Mul aexp aexp*

Design a static analysis that tries to determine a lower bound on the number of 2s in the prime factorization of a value.
Abstract values are just natural numbers:

datatype *twos* = *N2s nat*

The concretization function is: $\gamma\ (N2s\ k) = \{2^k * x \mid x \in \mathbb{N}\}$

1. Define the ordering $\leq$ on the abstract domain.
2. Define the join-operator $\sqcup$ on the abstract domain.
3. Define the functions num', $plus'$ and mul' on the abstract domain.
4. Run the analysis on the following program:

```
x := 4; {A1}
x := x*x + 8; {A2}
IF b THEN
    {A3} x=x+2 {A4}
ELSE
    {A5} x=x*10 {A6}
{A7}
```

We have already added the annotations for you. Iterate the step function on this program until a fixed point is reached, and document the result of each iteration in the following table. Note: You only need to specify the abstract value of variable x in each cell.

	0	1	2	3	4	5	6	7	8	9	...
$A1$	$\perp$										
$A2$	$\perp$										
$A3$	$\perp$										
$A4$	$\perp$										
$A5$	$\perp$										
$A6$	$\perp$										
$A7$	$\perp$										

4.1 Solution

1. $N2s\ j\ \leq\ N2s\ k$ iff $k{\leq}j$
2. $N2s\ i \sqcup N2s\ j = min\ i\ j$
3. $num'\ n = Max\ \{k \mid \exists\, m.\ n = (2{::}'a)^k * m\}$

   $plus'\ (N2s\ i)\ (N2s\ j) = min\ i\ j$

   $mul'\ (N2s\ i)\ (N2s\ j) = i + j$

	0	1	2	3	4	5	6	7	8	9	...
$A1$	$\perp$	N2s 2									
$A2$	$\perp$		N2s 3								
$A3$	$\perp$			N2s 3							
$A4$	$\perp$				N2s 1						
$A5$	$\perp$			N2s 3							
$A6$	$\perp$				N2s 4						
$A7$	$\perp$					N2s 1					

5 Post-fixed Points

Recall that a complete lattice is a type $'a$ with a partial order $\leq$ such that every set $X :: {'a}\ set$ has a greatest lower bound, denoted $\bigsqcap X$. This means that $\forall\, x \in X.\ \bigsqcap X \leq$ and $\forall\, y.\ ((\forall\, x \in X.\ y \leq x) \longrightarrow y \leq \bigsqcap X)$.

Prove that for a complete lattice and a monotone function $f :: {'a} \Rightarrow {'a}$ on it, the set of post-fixed points of f is closed under $\bigsqcap$:

4. $\forall\, X :: {'a}\ set.\ ((\forall\, x \in X.\ f\, x \leq x) \longrightarrow f\ (\bigsqcap X) \leq \bigsqcap X)$

5.1 Solution

```
lemma
  assumes 1: mono f
    and 2: ∀ X :: ('a::complete_lattice) set. (∀ x ∈ X. f x ≤ x)
  shows f (Inf X) ≤ Inf X
proof (rule Inf_greatest)
  fix x assume x: x ∈ X
  have Inf X ≤ x using Inf_lower[OF x] .
  hence f (Inf X) ≤ f x using 1 unfolding mono_def by auto
  also have ... ≤ x using 2 x by auto
  finally show f (Inf X) ≤ x .
qed
```