Final Exam

Semantics

11. 2. 2014

First name: _____

Last name: _____

Student-Id (Matrikelnummer): _____

Signature: _____

1. You may only use a pen/pencil, eraser, and two A4 sheets of notes to solve the exam. Switch off your mobile phones!
2. Please write on the sheets of this exam. At the end of the exam, there are two extra sheets. If you need more sheets, ask the supervisors during the exam.
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 20 minutes of the exam — you may disturb other students who need this time.
6. All questions of this exam are worth the same number of points.

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 **variables** fixed, 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 relation — especially for functions and relations 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 Command Equivalence

We call two commands c and c' equivalent wrt. the big-step semantics when *c started in s terminates in s′ iff c′ started in the same s also terminates in the same s′*. Formally:

$c_1 \sim c_2 \equiv (\forall\, s\ t.\ (c_1,\ s) \Rightarrow t \ \longleftrightarrow\ (c_2,\ s) \Rightarrow t)$

1. Define a function *is_SKIP* :: *com* $\Rightarrow$ *bool* which holds on commands equivalent to *SKIP*. The function *is_SKIP* should be as precise as possible, but it should not analyse arithmetic or boolean expressions.
   Prove: *is_SKIP* $c \Longrightarrow c \sim$ *SKIP*

2. The following command equivalence is wrong. Give a counterexample in the form of concrete instances for b_1, b_2, c_1, c_2, and a state s.

$$\begin{array}{l} \mathit{WHILE}\ b_1\ \mathit{DO}\ \mathit{IF}\ b_2\ \mathit{THEN}\ c_1\ \mathit{ELSE}\ c_2 \\ \sim \mathit{IF}\ b_2\ \mathit{THEN}\ (\mathit{WHILE}\ b_1\ \mathit{DO}\ c_1)\ \mathit{ELSE}\ (\mathit{WHILE}\ b_1\ \mathit{DO}\ c_2) \end{array} \quad (*)$$

3. Define a condition P on b_1, b_2, c_1, and c_2 such that the previous statement $(*)$ holds, i.e. $P\ b_1\ b_2\ c_1\ c_2 \Longrightarrow (*)$
   Your condition should be as precise as possible, but only using:
   - *lvars* :: *com* $\Rightarrow$ *vname set* (all left variables, i.e. written variables),
   - *rvars* :: *com* $\Rightarrow$ *vname set* (all right variables, i.e. all read variables),
   - *vars* :: *bexp* $\Rightarrow$ *vname set* (all variables in a condition), and
   - boolean connectives and set operations

   No proof required.

1.1 Solution

Question 1 Definition of *is_SKIP*:

is_SKIP SKIP = *True*
is_SKIP $(x ::= a)$ = *False*
is_SKIP $(c_1;;\ c_2)$ = (*is_SKIP* $c_1 \wedge$ *is_SKIP* c_2)
is_SKIP (*IF b THEN* c_1 *ELSE* c_2) = (*is_SKIP* $c_1 \wedge$ *is_SKIP* c_2)
is_SKIP (*WHILE b DO c*) = *False*

Question 2 Note that we have $(c \sim \mathit{SKIP}) = (\forall\, s\ t.\ (c,\ s) \Rightarrow t = (s = t))$.
Prove *is_SKIP* $c \Longrightarrow c \sim$ *SKIP* by structural induction on c.
For assignment and the while-statement *is_SKIP* is False.
For SKIP we apply reflexivity of *op* $\sim$.

In the Seq-case we know that *is_SKIP* $(c_1;;\ c_2)$ and the IHs are *is_SKIP* $c_i \longrightarrow c_i \sim$ *SKIP* for $i = 1, 2$. We have to prove $c_1;;\ c_2 \sim$ *SKIP*, i.e., $(c_1;;\ c_2,\ s) \Rightarrow t$ iff $s = t$. By definition of *is_SKIP* we get $\forall\, i.$ *is_SKIP* c_i, and with IH we get $\forall\, i.\ c_i \sim$ *SKIP*.

Now we have $(c_1;;\ c_2,\ s) \Rightarrow t$
iff $\exists\, r.\ (c_1,\ s) \Rightarrow r \wedge (c_2,\ r) \Rightarrow t$ (rule inversion and Seq-rule)
iff $s = t$ (due to $\forall\, i.\ c_i \sim$ *SKIP*). This proves the Seq-case.

In the If-case we know that *is_SKIP* (*IF b THEN* c_1 *ELSE* c_2), and the IHs are *is_SKIP* $c_i \longrightarrow c_i \sim$ *SKIP* for $i = 1, 2$. We have to prove *IF b THEN* c_1 *ELSE* $c_2 \sim$ *SKIP*, i.e., (*IF b THEN* c_1 *ELSE* c_2, s) $\Rightarrow t$ iff $s = t$. Again, by def. of *is_SKIP* and IH we get $\forall\, i.$ $c_i \sim$ *SKIP*.

We have (*IF b THEN* c_1 *ELSE* c_2, s) $\Rightarrow t$
iff *if bval b s then* $(c_1,\ s) \Rightarrow t$ *else* $(c_2,\ s) \Rightarrow t$ (cases on bval, rule inversion, If-rules)
iff $s = t$ (due to $\forall\, i.\ c_i \sim$ *SKIP*). This proves the Seq-case.

We have proved all cases. QED

$b_1 \equiv$ *"Less (V ''x'') (N 2)"*
$b_2 \equiv$ *"Less (V ''x'') (N 1)"*
$c_1 \equiv$ *"''x'' ::= Plus (V ''x'') (N 1) ;; ''y'' ::= N (−1::int)"*
$c_2 \equiv$ *"''x'' ::= Plus (V ''x'') (N 1) ;; ''y'' ::= N 1"*

Question 4 The condition is $(lvars\ c_1 \cup lvars\ c_2) \cap vars\ b_2 = \{\}$.

2 Palindrome – Induction

A *palindrome* is a word which reads the same in forward and backward direction. We introduce an inductive predicate *palindrome* :: *'a list* ⇒ *bool*:

inductive *palindrome* **where**
  "palindrome []"
| *"palindrome [x]"*
| *"palindrome xs ⟹ palindrome ([x] @ xs @ [x])"*

xs @ ys is the concatenation of the lists *xs* and *ys*. *rev* is list reversal:

"rev [] = []"
"rev (x # xs) = rev xs @ [x]"

1. Show *palindrome xs ⟹ rev xs = xs*.
2. Show *rev xs = xs ⟹ palindrome xs*.

You are allowed to use rule induction, structural induction, and the following induction rule:

$$\frac{P\ []\qquad \forall x.\ P\ [x]\qquad \forall x\ y\ xs.\ P\ xs \longrightarrow P\ ([x]\ @\ xs\ @\ [y])}{\forall xs.\ P\ xs}\ \text{IND}$$

2.1 Solution

First, we prove the auxiliary lemma *rev (xs @ [x]) = [x] @ rev xs*, by induction on *xs*. The case *xs = []* is obvious, in the case *x' # xs*, we have the IH *rev (xs @ [x]) = [x] @ rev xs* and have to show *rev (x' # xs @ [x]) = [x] @ rev (x' # xs)*. We have *rev (x'#xs@[x]) = rev (xs@[x])@[x'] = [x]@(rev xs@[x']) = [x]@(rev (x'#xs))*. The first and last equality is due to def. of rev, and associativity of list concatenation, the second one due to IH.

For 1, we use rule induction. The cases for empty and singleton list are trivial. In the last case, we have the IH *rev xs = xs*, and have to show *rev ([x] @ xs @ [x]) = [x] @ xs @ [x]*. We have *rev ([x]@xs@[x]) = rev (xs@[x])@[x] = [x]@rev xs@[x] = [x]@xs@[x]*. The first equality is due to def. of rev, the second one due to the aux-lemma, and the third one due to IH.

For 2, we use the given induction principle IND. The first two cases are straightforward due to the intro-rules of palindrome. In the third case, we have the IH *rev xs = xs ⟶ palindrome xs*. Moreover, we may assume (*) *rev ([x] @ xs @ [y]) = [x] @ xs @ [y]*. We have to show *palindrome ([x] @ xs @ [y])*. Using the intro-rule for palindrome, this follows from *x = y* and *palindrome xs*. Using IH, *palindrome xs* follows from *rev xs = xs*. Thus, it remains to show: *x = y ∧ rev xs = xs*.

We have *rev ([x]@xs@[y]) = rev (xs@[y])@[x] = [y]@rev xs@[x]* (analogously to the proof of 1). With (*), we get *x = y ∧ rev xs = xs*. QED.

3 Hoare-Logic

We extend IMP by an assertion command *ASSERT bexp*. Intuitively, the execution gets stuck if the asserted expression evaluates to false, otherwise *ASSERT bexp* behaves like *SKIP*. This is expressed by adding the following rule to the big-step semantics:

assert: $bval\ b\ s \Longrightarrow (ASSERT\ b,s) \Rightarrow s$

Moreover, we add the following rule to the Hoare-Logic for total correctness:

$(\forall\, s.\ P\ s \longrightarrow Q\ s \wedge bval\ b\ s) \Longrightarrow \vdash_t \{P\}\ ASSERT\ b\ \{Q\}$

Questions

1. What does the weakest precondition $wp_t\ (ASSERT\ b)\ Q$ look like?
2. Prove: $\vdash_t \{wp_t\ (ASSERT\ b)\ Q\}\ ASSERT\ b\ \{Q\}$.
3. Prove: $\vdash_t \{P\}\ ASSERT\ b\ \{Q\} \Longrightarrow \models_t \{P\}\ ASSERT\ b\ \{Q\}$.

Hints

1. We have the definition
   $wp_t\ c\ Q\ =\ (\lambda s.\ \exists\, t.\ (c,s) \Rightarrow t \wedge Q\ t)$
   However, for Question 1, we want an equation that shows how to expand wp_t syntactically, i.e., the right hand side should not contain the Big/Small-step semantics. You need **not** prove your equation here.
2. The main idea of the completeness proof is to show $\vdash_t \{wp_t\ c\ Q\}\ c\ \{Q\}$. What you have to prove here is the case for the *ASSERT*-command. Your characterization of wp_t from Question 1 may be useful here!
3. For the correctness proof, one shows, by induction over c:
   $\vdash_t \{P\}\ c\ \{Q\} \Longrightarrow \models_t \{P\}\ c\ \{Q\}$
   What you have to prove here is the (base) case for the *ASSERT*-command.

Extra space for solving Question 3.

3.1 Solution

1. $wp_t\ (ASSERT\ b)\ Q = \lambda s.\ Q\ s \wedge bval\ b\ s$
2. Using 1), we have to prove: $\vdash_t \{\lambda s.\ Q\ s \wedge bval\ b\ s\}\ ASSERT\ b\ \{Q\}$ With the assert-rule, this follows from the trivial proposition $(\forall s.\ Q\ s \wedge bval\ b\ s \longrightarrow Q\ s \wedge bval\ b\ s)$
3. We assume $\vdash_t \{P\}\ ASSERT\ b\ \{Q\}$ and show $\models_t \{P\}\ ASSERT\ b\ \{Q\}$. Unfolding the definition of $\models_t$, we fix an s and assume $P\ s$. We have to show $(\exists t.\ (ASSERT\ b,s) \Rightarrow t \wedge Q\ t)$ (*).

   From the assumption $\vdash_t \{P\}\ ASSERT\ b\ \{Q\}$, rule inversion yields $\forall s.\ P\ s \longrightarrow Q\ s \wedge bval\ b\ s$. With the assumption $P\ s$, we get $Q\ s \wedge bval\ b\ s$, and the assert-rule of the big-step semantics yields $(ASSERT\ b,s) \Rightarrow s$. This concludes the proof of (*).

4 Abstract Interpretation

IMP is extended by adding a multiplication operator to arithmetic expressions:

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

Design a static analysis that tries to determine whether a variable is −1, 0, 1, or any other value. The abstract domain consists of the values $-1 \mid 0 \mid 1 \mid Any$, and γ is defined as:

$\gamma(-1) = \{-1\}$
$\gamma(0) = \{0\}$
$\gamma(1) = \{1\}$
$\gamma(Any) =$ All integers

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

```
x := -1; {A1}
x := x*x + (-1); {A2}
IF b THEN
    {A3} x=x+1 {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.

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

Extra space for solving Question 4.

4.1 Solution

1. $x \leq y$ iff $x{=}y$ or $y = Any$
2. $x \sqcup y = (\mathit{if}\ x{=}y\ \mathit{then}\ x\ \mathit{else}\ Any)$
3.
```
plus' 0 x = x
plus' x 0 = x
plus' -1 -1  = Any
plus' -1 1  = 0
plus' 1 -1  = 0
plus' 1 1  = Any
plus' _ _ = Any

mul' 0 x = 0
mul' x 0 = 0
mul' 1 x = x
mul' x 1 = x
mul' -1 -1 = 1
mul' _ _ = Any
```
4.

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

5 Fixed Point Theory

Let 'a be a complete lattice with ordering $\leq$ and $f::'a \Rightarrow 'a$ be a monotonic function. Moreover, let x_0 be a post-fixpoint of f, i.e., $x_0 \leq f\, x_0$. Prove:

$\bigsqcup \{f^i(x_0) \mid i \in \mathbb{N}\} \leq \bigsqcup \{f^{i+1}\ (x_0) \mid i \in \mathbb{N}\}$

Hint The least upper bound satisfies the following properties

$$x \in A \Longrightarrow x \leq \bigsqcup A \qquad \text{(upper)}$$

$$(\forall\, x \in A.\ x \leq u) \Longrightarrow \bigsqcup A \leq u \qquad \text{(least)}$$

5.1 Solution

Due to (least), it is enough to show that for all i, we have $f^i(x_0) \leq \bigsqcup \{f^{i+1}\ (x_0) \mid i \in \mathbb{N}\}$. We proceed by cases on $i = 0$. If $i > 0$, we have $f^i(x_0) \in \{f^{i+1}\ (x_0) \mid i \in \mathbb{N}\}$, and the proposition follows with (upper). If $i = 0$, we have $f^i(x_0) = x_0 \leq f\, x_0$ due to the post-fixpoint property, and the proposition follows analogously to the previous case.

Extra Sheet 1

Extra Sheet 2