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:

\[
c_1 \sim c_2 \equiv (\forall s \ t. (c_1, s) \Rightarrow t \leftrightarrow (c_2, s) \Rightarrow t)
\]

1. Define a function \( \text{is}\_\text{SKIP} :: \text{com} \Rightarrow \text{bool} \) which holds on commands equivalent to \( \text{SKIP} \). The function \( \text{is}\_\text{SKIP} \) should be as precise as possible, but it should not analyse arithmetic or boolean expressions.

Prove: \( \text{is}\_\text{SKIP} \ c \implies c \sim \text{SKIP} \)

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 \( s \).

\[
\text{WHILE} \ b_1 \ \text{DO} \ IF \ b_2 \ THEN \ c_1 \ ELSE \ c_2 \sim \ IF \ b_2 \ THEN \ (\text{WHILE} \ b_1 \ \text{DO} \ c_1) \ ELSE \ (\text{WHILE} \ b_1 \ \text{DO} \ c_2)
\]

\((*)\)

3. Define a condition \( P \) on \( b_1, b_2, c_1, \) and \( c_2 \) such that the previous statement \((*)\) holds, i.e. \( P \ b_1 \ b_2 \ c_1 \ c_2 \implies (*) \)

Your condition should be as precise as possible, but only using:

- \( \text{lvars} :: \text{com} \Rightarrow \text{vname set} \) (all left variables, i.e. written variables),
- \( \text{rvars} :: \text{com} \Rightarrow \text{vname set} \) (all right variables, i.e. all read variables),
- \( \text{vars} :: \text{bexp} \Rightarrow \text{vname set} \) (all variables in a condition), and
- boolean connectives and set operations

No proof required.

1.1 Solution

**Question 1** Definition of \( \text{is}\_\text{SKIP} \):

\[
\text{is}\_\text{SKIP} \ \text{SKIP} = \ True
\]
\[
\text{is}\_\text{SKIP} \ (x := a) = \ False
\]
\[
\text{is}\_\text{SKIP} \ (c_1 ; c_2) = (\text{is}\_\text{SKIP} \ c_1 \land \text{is}\_\text{SKIP} \ c_2)
\]
\[
\text{is}\_\text{SKIP} \ (IF \ b \ THEN \ c_1 \ ELSE \ c_2) = (\text{is}\_\text{SKIP} \ c_1 \land \text{is}\_\text{SKIP} \ c_2)
\]
\[
\text{is}\_\text{SKIP} \ (\text{WHILE} \ b \ \text{DO} \ c) = \ False
\]

**Question 2** Note that we have \((c \sim \text{SKIP}) = (\forall s \ t. (c, s) \Rightarrow t = (s = t))\).

Prove \( \text{is}\_\text{SKIP} \ c \implies c \sim \text{SKIP} \) by structural induction on \( c \).

For assignment and the while-statement \( \text{is}\_\text{SKIP} \) is False.

For \( \text{SKIP} \) we apply reflexivity of \( \text{op} \sim \).
In the Seq-case we know that \textit{is SKIP} \((c_1; c_2)\) and the IHs are \textit{is SKIP} \(c_i \rightarrow c_i \sim SKIP\) 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 \textit{is SKIP} we get \(\forall i. \textit{is SKIP} c_i\), and with IH we get \(\forall i. c_i \sim SKIP\).

Now we have \((c_1; c_2, s) \Rightarrow t\)

iff \(\exists r. (c_1, s) \Rightarrow r \land (c_2, r) \Rightarrow t\) (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 \textit{is SKIP} \((\text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2)\), and the IHs are \textit{is SKIP} \(c_i \rightarrow c_i \sim SKIP\) for \(i = 1, 2\). We have to prove \(\text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2 \sim SKIP\), i.e., \((\text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2, s) \Rightarrow t\) iff \(s = t\). Again, by def. of \textit{is SKIP} and IH we get \(\forall i. c_i \sim SKIP\).

We have \((\text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2, s) \Rightarrow t\)

iff \(\text{if } \text{bval } b \text{ s then } (c_1, s) \Rightarrow t \text{ else } (c_2, s) \Rightarrow t\) (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 \text{"Less} (V "x") (N 2) \"
\[ b_2 \equiv \text{"Less} (V "x") (N 1) \"
\[ c_1 \equiv "x" ::= \text{Plus} (V "x") (N 1) ;; "y" ::= N (\text{int}) \"
\[ c_2 \equiv "x" ::= \text{Plus} (V "x") (N 1) ;; "y" ::= N 1 \"

\textbf{Question 4} \hspace{1cm} \text{The condition is } (\text{lvars } c_1 \cup \text{lvars } c_2) \cap \text{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:

\[ \begin{array}{c}
  P [] \quad \forall x. \ P [x] \\
  \forall x \ y \ xs. \ P xs \rightarrow P ([x] @ xs @ [y]) \\
  \forall xs. \ P xs
\end{array} \]

\[ \text{IND} \]

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']) = [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] @ rev 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] @ rev 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 $\text{ASSERT } bexp$. Intuitively, the execution gets stuck if the asserted expression evaluates to false, otherwise $\text{ASSERT } bexp$ behaves like $\text{SKIP}$. This is expressed by adding the following rule to the big-step semantics:

assert: $\text{bval } b \ s \implies (\text{ASSERT } b, s) \Rightarrow s$

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

$(\forall s. \ P \ s \implies Q \ s \land \text{bval } b \ s) \implies \vdash t \ {P} \ \text{ASSERT } b \ {Q}$

Questions

1. What does the weakest precondition $\text{wp}_t (\text{ASSERT } b) \ Q$ look like?
2. Prove: $\vdash t \ {\text{wp}_t (\text{ASSERT } b) \ Q} \ \text{ASSERT } b \ {Q}$.
3. Prove: $\vdash t \ {P} \ \text{ASSERT } b \ {Q} \implies \vdash t \ {P} \ \text{ASSERT } b \ {Q}$.

Hints

1. We have the definition

$\text{wp}_t \ c \ Q = (\lambda s. \exists t. \ (c,s) \Rightarrow t \land Q \ t)$

However, for Question 1, we want an equation that shows how to expand $\text{wp}_t$ 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 $\vdash t \ {\text{wp}_t c \ Q} \ c \ {Q}$. What you have to prove here is the case for the $\text{ASSERT}$-command. Your characterization of $\text{wp}_t$ from Question 1 may be useful here!

3. For the correctness proof, one shows, by induction over $c$:

$\vdash t \ {P} \ c \ {Q} \implies \vdash t \ {P} \ c \ {Q}$

What you have to prove here is the (base) case for the $\text{ASSERT}$-command.
Extra space for solving Question 3.

3.1 Solution

1. \( wp_t (\text{ASSERT } b) \ Q = \lambda s. \ Q \ s \land \ bval \ b \ s \)

2. Using 1), we have to prove: \( \vdash_t \{ \lambda s. \ Q \ s \land \ bval \ b \ s \} \ \text{ASSERT } b \ \{ Q \} \) With the assert-rule, this follows from the trivial proposition \( (\forall s. \ Q \ s \land \ bval \ b \ s \rightarrow Q \ s \land \ bval \ b \ s) \)

3. We assume \( \vdash_t \{ P \} \ \text{ASSERT } b \ \{ Q \} \) and show \( \models_t \{ P \} \ \text{ASSERT } b \ \{ Q \} \). Unfolding the definition of \( \models_t \), we fix an \( s \) and assume \( P \ s \). We have to show \( (\exists t. (\text{ASSERT } b, s) \Rightarrow t \land Q \ t) \) (*).

From the assumption \( \vdash_t \{ P \} \ \text{ASSERT } b \ \{ Q \} \), rule inversion yields \( \forall s. \ P \ s \rightarrow Q \ s \land bval \ b \ s \). With the assumption \( P \ s \), we get \( Q \ s \land bval \ b \ s \), and the assert-rule of the big-step semantics yields \( (\text{ASSERT } b, s) \Rightarrow s \). 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 $-1$ | $0$ | $1$ | Any, and $\gamma$ is defined as:

$\gamma(-1) = \{-1\}$

$\gamma(0) = \{0\}$

$\gamma(1) = \{1\}$

$\gamma(\text{Any}) = \text{All integers}$

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

```plaintext
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.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>⊥</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A2</td>
<td>⊥</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A3</td>
<td>⊥</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A4</td>
<td>⊥</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A5</td>
<td>⊥</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A6</td>
<td>⊥</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A7</td>
<td>⊥</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Extra space for solving Question 4.

4.1 Solution

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

   \( \text{mul'} \ 0 \ x = 0 \)
   \( \text{mul'} \ x \ 0 = 0 \)
   \( \text{mul'} \ 1 \ x = x \)
   \( \text{mul'} \ x \ 1 = x \)
   \( \text{mul'} \ -1 \ -1 = 1 \)
   \( \text{mul'} \ _ \ _ = \text{Any} \)

|    | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |...
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>⊥</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A2</td>
<td>⊥</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A3</td>
<td>⊥</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A4</td>
<td>⊥</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A5</td>
<td>⊥</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A6</td>
<td>⊥</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A7</td>
<td>⊥</td>
<td>Any</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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:

\[
\bigcup \{ f^i(x_0) \mid i \in \mathbb{N} \} \leq \bigcup \{ f^{i+1}(x_0) \mid i \in \mathbb{N} \}
\]

**Hint** The least upper bound satisfies the following properties

\[
x \in A \implies x \leq \bigcup A \quad \text{(upper)}
\]

\[
(\forall x \in A. \; x \leq u) \implies \bigcup A \leq u \quad \text{(least)}
\]

5.1 Solution

Due to (least), it is enough to show that for all \( i \), we have \( f^i(x_0) \leq \bigcup \{ f^{i+1}(x_0) \mid i \in \mathbb{N} \} \).

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