Software Verification – Exam

ETH Zürich

15 December 2014

Surname, first name: .................................................................................................
Student number: ....................................................................................................

I confirm with my signature that I was able to take this exam under regular circumstances and that I have read and understood the directions below.

Signature: .............................................................................................................

Directions:

• Exam duration: 1 hour 45 minutes.

• Except for a dictionary you are not allowed to use any supplementary material.

• All solutions can be written directly on the exam sheets. If you need more space for your solution ask the supervisors for a sheet of official paper. You are not allowed to use other paper. Please write your student number on each additional sheet.

• Only one solution can be handed in per question. Invalid solutions need to be crossed out clearly.

• Please write legibly! We will only correct solutions that we can read.

• Manage your time carefully (take into account the number of points for each question).

• Please immediately tell the exam supervisors if you feel disturbed during the exam.

Good luck!
<table>
<thead>
<tr>
<th>Question</th>
<th>Available points</th>
<th>Your points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) Axiomatic semantics</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>2) Separation logic</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>3) Data Flow Analysis</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>4) Model checking</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>5) Software model checking</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>49</strong></td>
<td></td>
</tr>
</tbody>
</table>
1 Axiomatic semantics (8 points)

Consider the following annotated program, where $a$ and $b$ are distinct arrays (indexed from 0) both of length $n$, $k$ is an integer variable, and $x \mod y$ denotes the remainder of the integer division of $x$ by $y$. Note that the else branch is empty.

$$\{ n > 0 \land \forall i : 0 \leq i < n \implies a[i] = b[i] = 0 \}$$

1 \textbf{from}
2 \quad k := 0
3 \textbf{until} k = n \textbf{loop}
4 \quad \textbf{if} k \mod 3 = 0 \textbf{then}
5 \quad \quad b[k] := a[k] + 1
6 \quad \textbf{else}
7 \quad \end
8 \quad k := k + 1
9 \end

$$\{ \forall i : 0 \leq i < n \land i \mod 3 = 0 \implies b[i] = 1 \}$$

Prove that the above triple (precondition, program, postcondition) is a theorem of Hoare’s axiomatic system for partial correctness. In other words, prove the program correct with respect to the given specification.
2 Separation Logic (12 points)

2.1 Linked List Proof (7 points)

We can assert that a heap contains a linked list by using the following inductively defined predicate:

\[
\begin{align*}
\text{list}([], i) & \iff \text{emp} \land i = \text{nil} \\
\text{list}(a :: as, i) & \iff \exists j. i \mapsto a, j \mapsto \text{list}(as, j)
\end{align*}
\]

where \(\text{nil}\) is a constant used to terminate the list.

Give a proof outline for the following triple using the list predicate and the proof rules of separation logic:

\[
\{ \text{list}(a :: as, x) \}
\]

\[
\begin{align*}
& y := \text{cons}(0,0); \\
& \text{temp1} := [x]; \\
& \text{temp2} := [x+1]; \\
& [y] := \text{temp1}; \\
& [y+1] := \text{nil}; \\
& \text{dispose}(x); \\
& \text{dispose}(x+1); \\
& x := \text{temp2}; \\
\{ \text{list}(as, x) \ast \text{list}(a :: [], y) \}
\end{align*}
\]
2.2 Satisfaction of Assertions (5 points)

For each separation logic assertion below, draw a program state (i.e. store and heap) that satisfies it.

(a). \((x \mapsto 6,7) \ast \neg (x \mapsto 6,7)\)

(b). \((x \mapsto 6,7) \land (y \mapsto 6,7)\)
(c). \( \text{emp} \Rightarrow x = y \)

(d). \( \exists i. x \mapsto i * i \mapsto i \)
3 Data Flow Analysis (9 points)

Consider the following program fragment, which computes the value of the $m$th Fibonacci number for $m \geq 1$.

\[
\begin{align*}
  f_0 & := 0 \\
  f_1 & := 1 \\
  \text{if } m \leq 1 \text{ then} & \quad f_2 := m \\
  \text{else} & \\
    \text{from} & \\
    \quad i := 2 \\
    \text{until } i > m \text{ loop} & \\
    \quad i := i + 1 \\
    f_2 & := f_0 + f_1 \\
    f_0 & := f_1 \\
    f_1 & := f_2 \\
  \text{end} & \\
\end{align*}
\]

print $(f_2)$

(a). Draw the control flow graph of the program fragment and label each elementary block. (2 points)

(b). Annotate your control flow graph with the analysis result of a Reaching Definitions analysis of the program fragment, considering the variables $f_0, f_1, f_2, i$. (5 points)
(e). Copy Propagation, an application of the Reaching Definitions analysis, is defined as follows:

A use of a variable $x$ at a program point $\ell'$ can be replaced by $y$ if $[x := y]^{\ell'}$ is the only definition of $x$ that reaches $\ell'$ and $y$ is not modified between $\ell$ and $\ell'$.

Is there a possibility for copy propagation in the program fragment? In justifying your answer, use your analysis result. (2 points)
4 Model Checking (9 points)

Recall the semantics of LTL over finite words with alphabet $\mathcal{P}$. For a word $w = w(1)w(2)\cdots w(n) \in \mathcal{P}^*$ with $n \geq 0$ and a position $1 \leq i \leq n$ the satisfaction relation $\models$ is defined recursively as follows (where $p, q \in \mathcal{P}$).

$$
\begin{align*}
  w, i \models p & \iff p = w(i) \\
  w, i \models \neg \phi & \iff w, i \not\models \phi \\
  w, i \models \phi_1 \land \phi_2 & \iff w, i \models \phi_1 \text{ and } w, i \models \phi_2 \\
  w, i \models X\phi & \iff i < n \text{ and } w, i + 1 \models \phi \\
  w, i \models \phi_1 \lor \phi_2 & \iff \exists i \leq j \leq n \text{ such that: } w, j \models \phi_2 \\
  & \text{ and for all } i \leq k < j \text{ it is the case that } w, k \models \phi_1 \\
  w, i \models \bigcirc \phi & \iff \exists i \leq j \leq n \text{ such that: } w, j \models \phi \\
  w, i \models \square \phi & \iff \text{ for all } i \leq j \leq n \text{ it is the case that: } w, j \models \phi \\
  w \models \phi & \iff w, 1 \models \phi
\end{align*}
$$

4.1 Automata and LTL formulas (5 points)

Consider the automaton $A$ (with states $A, B, C, D$) in Figure 1, over the alphabet $\{p, q\}$. Notice that $A$ is the initial state, $A$ and $D$ are final states, and the automaton is nondeterministic.

![Figure 1: Automaton A over alphabet \{p, q\}](image)

For each of the following LTL formulas say whether every accepting run of $A$ satisfies the formula. If it does, argue informally (but precisely) why this is the case; if it does not, provide a counterexample.

(1) $A \models \square (\bigcirc q)$
(2) \( A \models \Box (p \implies Xp) \)

(3) \( A \models p \implies Xp \)

(4) \( A \models \Diamond (\Box q) \)
4.2 Automata-based model checking (4 points)

Consider the LTL formula:

\[ \phi \equiv \Box (r \implies (q \lor (\neg r \land Xp))) \]

**Property automaton.** Construct an automaton \( \mathcal{F} \) that accepts precisely the words that satisfy \( \phi \).
5 Software model checking (11 points)

Consider the following code snippet $C$, where $x$ and $y$ are integer variables, and $z$ is a Boolean variable.

1. assume $z = \text{True}$ end
2. if $x > y$ and $x \leq 0$ then
3. \hspace{1em} $z := \text{True}$
4. else
5. \hspace{1em} $z := \text{False}$
6. end
7. assert $y \leq 0$ or not $z$ end

Recall that:

- $\text{Pred}(\text{exp})$ denotes the weakest under-approximation of the expression $\text{exp}$ that is expressible as a Boolean combination of the given predicates.

- The predicate abstraction of an assume instruction $\text{assume \ \exp \ \text{end}}$ is $\text{assume \ not \ \text{Pred}(not \ \exp) \ \text{end}}$ followed by a parallel conditional assignment updating the predicates with respect to the original assume.

- The predicate abstraction of an assert instruction $\text{assert \ \exp \ \text{end}}$ simply is $\text{assert \ \text{Pred}(\exp) \ \text{end}}$, provided that $\exp$ can be approximated exactly by means of the available predicates (which is the case in this exercise).

5.1 Boolean abstractions (8 points)

Build the Boolean abstraction $A$ of the code snippet $C$ above with respect to the predicates:

$p \equiv x > y$
$q \equiv y > 0$
$r \equiv z = \text{True}$
5.2 Traces (3 points)

Trace classification: Using the table below, classify every possible initial abstract state \( I \) into the following categories (note that, due to nondeterminism, a state may belong to more than one category):

- **invalid**: \( I \) may lead to a trace that is infeasible in \( A \);
- **spurious counterexample**: \( I \) may lead to an error trace in \( A \) that is infeasible in \( C \) (that is, it is invalid in \( C \));
- **error**: \( I \) may lead to an error trace in \( A \) that is feasible in \( C \) (that is, it is also an error in \( C \));
- **valid**: \( I \) may lead to a valid (without errors) trace in \( A \).

<table>
<thead>
<tr>
<th>INITIAL STATE ( I )</th>
<th>CLASSIFICATION (invalid, spurious, error, valid)</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ \textit{p}, \textit{q}, \textit{r} }</td>
<td>..................................................</td>
</tr>
<tr>
<td>{ \textit{not} \textit{p}, \textit{q}, \textit{r} }</td>
<td>..................................................</td>
</tr>
<tr>
<td>{ \textit{p}, \textit{not} \textit{q}, \textit{r} }</td>
<td>..................................................</td>
</tr>
<tr>
<td>{ \textit{not} \textit{p}, \textit{not} \textit{q}, \textit{r} }</td>
<td>..................................................</td>
</tr>
<tr>
<td>{ \textit{p}, \textit{q}, \textit{not} \textit{r} }</td>
<td>..................................................</td>
</tr>
<tr>
<td>{ \textit{not} \textit{p}, \textit{q}, \textit{not} \textit{r} }</td>
<td>..................................................</td>
</tr>
<tr>
<td>{ \textit{p}, \textit{not} \textit{q}, \textit{not} \textit{r} }</td>
<td>..................................................</td>
</tr>
<tr>
<td>{ \textit{not} \textit{p}, \textit{not} \textit{q}, \textit{not} \textit{r} }</td>
<td>..................................................</td>
</tr>
</tbody>
</table>

Implications for the concrete program: Based on the classification, what do you conclude about the correctness of the concrete program \( C \)? Please justify your answer.

.................................................................
.................................................................
.................................................................
.................................................................
.................................................................
.................................................................
.................................................................
.................................................................
.................................................................
.................................................................
.................................................................
.................................................................
.................................................................
.................................................................
.................................................................
.................................................................
.................................................................
.................................................................
.................................................................
.................................................................

22