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.

\[
\begin{align*}
\{ & n > 0 \land \forall i : 0 \leq i < n \implies a[i] = b[i] = 0 \} \\
1 \textbf{from} \\
2 & k := 0 \\
3 \textbf{until} \ k = n \textbf{ loop} \\
4 & \textbf{if} \ k \mod 3 = 0 \textbf{ then} \\
5 & \quad b[k] := a[k] + 1 \\
6 & \textbf{else} \\
7 & \quad k := k + 1 \\
8 \textbf{end} \\
9 \{ & \forall i : 0 \leq i < n \land i \mod 3 = 0 \implies b[i] = 1 \}
\end{align*}
\]

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.
Solution:

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

to

12 { $0 \leq k < n \land k \mod 3 = 0 \implies b[k] = 1 \land$
13 $a[i] = 0$ }

15 if $k \mod 3 = 0$ then

16 { $-1 \leq k < n \land k \mod 3 = 0 \land$
17 $a[i] = 0$ }

18 $k \mod 3 = 0 \implies b[k] = 1 \land$
19 $a[i] = 0$ }

20 $b[k] = a[k] + 1$

21 { $-1 \leq k < n \land$
22 $k \mod 3 = 0 \implies b[i] = 1 \land$
23 $a[i] = 0$ }

25 else

26 { $-1 \leq k < n \land k \mod 3 = 0 \land$
27 $a[i] = 0$ }

28 $k \mod 3 = 0 \implies b[k] = 1 \land$
29 $a[i] = 0$ }

30 end

31 { $-1 \leq k < n \land$
32 $a[i] = 0$ }

33 $k \mod 3 = 0 \implies b[k] = 1 \land$
34 $a[i] = 0$ }

35 { $0 \leq k + 1 \leq n \land$
36 $a[i] = 0$ }

37 $k \mod 3 = 0 \implies b[i] = 1 \land$
38 $a[i] = 0$ }

39 $k := k + 1$

40 { $0 \leq k < n \land$
41 $a[i] = 0$ }

42 end

43 { $k = n \land$
44 $a[i] = 0$ }

45 $k := k + 1$

46 { $0 \leq k < n \land$
47 $a[i] = 0$ }

41 $a[i] = 0$ }

42 end

43 { $k = n \land$
44 $a[i] = 0$ }

45 $a[i] = 0$ }

46 { $0 \leq i < n \land$
47 $a[i] = 0$ }

41 $a[i] = 0$ }

42 end

43 { $k = n \land$
44 $a[i] = 0$ }

45 $a[i] = 0$ }

46 { $0 \leq i < n \land$
47 $a[i] = 0$ }

41 $a[i] = 0$ }

42 end
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:

\[
\text{list}(\text{[]}, i) \iff \text{emp} \land i = \text{nil} \\
\text{list}(a :: as, i) \iff \exists j. i \equiv a, j \ast \text{list} (as, j)
\]

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:

\[
\{ \text{list}(a :: as, x) \} \\
y := \text{cons}(0, 0); \\
temp1 := [x]; \\
temp2 := [x+1]; \\
[y] := \text{temp1}; \\
[y+1] := \text{nil}; \\
dispose(x); \\
dispose(x+1); \\
x := \text{temp2}; \\
\{ \text{list}(as, x) \ast \text{list}(a :: [\text{[]}, y]) \}
\]
Sample solution:

\{\text{list}(a::as,x)\}
\quad y := \text{cons}(0,0);
\{\text{list}(a::as,x) \cdot y|->0,0\}
\{\text{EX} \ j. \ x|->a,j \cdot \text{list}(as,j) \cdot y|->0,0\}
\quad \{x|->a,j \cdot \text{list}(as,j) \cdot y|->0,0\}
\quad \text{temp1} := [x];
\quad \{x|->a,j \cdot \text{list}(as,j) \cdot y|->0,0 \land \text{temp1} = a\}
\quad \text{temp2} := [x+1];
\quad \{x|->a,j \cdot \text{list}(as,j) \cdot y|->0,0 \land \text{temp1} = a \land \text{temp2} = j\}
\quad \{y] := \text{temp1};
\quad \{x|->a,j \cdot \text{list}(as,j) \cdot y|->\text{temp1},0 \land \text{temp1} = a \land \text{temp2} = j\}
\quad \{y+1] := \text{nil};
\quad \{x|->a,j \cdot \text{list}(as,j) \cdot y|->\text{temp1},\text{nil} \land \text{temp1} = a \land \text{temp2} = j\}
\quad \text{dispose}(x);
\quad \{x+1|->j \cdot \text{list}(as,j) \cdot y|->\text{temp1},\text{nil} \land \text{temp1} = a \land \text{temp2} = j\}
\quad \text{dispose}(x+1);
\quad \{\text{list}(as,j) \cdot y|->\text{temp1},\text{nil} \land \text{temp1} = a \land \text{temp2} = j\}
\{\text{EX} \ j. \ \text{list}(as,j) \cdot y|->\text{temp1},\text{nil} \land \text{temp1} = a \land \text{temp2} = j\}
\{\text{list}(as,temp2) \cdot y|->\text{temp1},\text{nil} \land \text{temp1} = a\}
\quad x := \text{temp2};
\{\text{list}(as,temp2) \cdot y|->\text{temp1},\text{nil} \land \text{temp1} = a \land x = \text{temp2}\}
\{\text{list}(as,x) \cdot y|->\text{temp1},\text{nil} \land \text{temp1} = a\}
\{\text{list}(as,x) \cdot y|->a,\text{nil}\}
\{\text{list}(as,x) \cdot y|->a,\text{nil} \cdot \text{list}([],\text{nil})\}
\{\text{list}(as,x) \cdot \text{list}(a::[],y)\}
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). \((x \leftrightarrow 6,7) \ast \neg(x \leftrightarrow 6,7)\)

Sample solution:

\[
\begin{array}{c}
\text{Store} \\
x \\
y
\end{array}
\quad
\begin{array}{c}
\text{Heap} \\
6 & 7 \\
8
\end{array}
\]

(b). \((x \leftrightarrow 6,7) \land (y \leftrightarrow 6,7)\)

Sample solution:

\[
\begin{array}{c}
\text{Store} \\
x \\
y
\end{array}
\quad
\begin{array}{c}
\text{Heap} \\
6 & 7
\end{array}
\]
(c). \( \text{emp} \Rightarrow x = y \)

Sample solution:

![Sample solution diagram](store_heap_diagram)

(d). \( \exists i. x \leftrightarrow i \text{ and } i \leftrightarrow i \)

Sample solution:

![Sample solution diagram](store_heap_diagram)
3 Data Flow Analysis (9 points)

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

\[
f_0 := 0 \\
f_1 := 1 \\
\text{if } m \leq 1 \text{ then} \\
f_2 := m \\
\text{else} \\
\quad \text{from} \\
\quad \quad i := 2 \\
\quad \quad \text{until } i > m \text{ loop} \\
\quad \quad \quad i := i + 1 \\
\quad \quad \quad f_2 := f_0 + f_1 \\
\quad \quad \quad f_0 := f_1 \\
\quad \quad \quad f_1 := f_2 \\
\quad \text{end} \\
\text{end} \\
\text{print } (f_2)
\]

(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)
(c). 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)

There is no possibility for copy propagation in the program fragment. The variables $f_0$, $f_1$, $f_2$, $i$ are used only in blocks 6-11, and in none of these blocks copy propagation can be applied: for blocks 6-9 and 11, the analysis result shows that each of the used variables in the blocks could be reached by more than one definition; and for block 10, while it is reached by only one definition, $[f_2 := f_0 + f_1]^{10}$, that definition is not a copy statement.
4 Model Checking (9 points)

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

\[
\begin{align*}
  \text{if } p = w(i) & \quad w, i \models p \\
  \text{if } w, i \not\models \phi & \quad w, i \models \neg \phi \\
  \text{if } w, i \models \phi_1 \land \phi_2 & \quad w, i \models \phi_1 \land \phi_2 \\
  \text{if } i < n \text{ and } w, i + 1 \models \phi & \quad w, i \models X \phi \\
  \text{if there exists } i \leq j \leq n \text{ such that: } w, j \models \phi_1 \text{ and for all } i \leq k < j \text{ it is the case that } w, k \models \phi_1 & \quad w, i \models \Diamond \phi \ \\
  \text{if there exists } i \leq j \leq n \text{ such that: } w, j \models \phi & \quad w, i \models \Box \phi \\
  \text{if for all } i \leq j \leq n \text{ it is the case that: } w, j \models \phi & \quad w \models \phi
\end{align*}
\]

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.

![Automaton A](image)

Figure 1: Automaton \( A \) over alphabet \( \{p, q\} \).

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. \( A \models \Box (\Diamond q) \)

   No: the word \( w = pp \) is accepted by \( A \), but no letter \( q \) appears in it; hence \( \Diamond q \) does not hold.

2. \( A \models \Box (p \Longrightarrow X p) \)

   No: the word \( w = qp \) is accepted by \( A \), but \( w, 2 \not\models X p \) because there are no letters after the unique \( p \) in \( w \).

3. \( A \models p \Longrightarrow X p \)

   Yes: every word \( w \) accepted by \( A \) such that \( p \) holds in the first position drives \( A \) into state \( B \), from where a \( p \) must follow for \( w \) to be accepted.

4. \( A \models (\Diamond q) \)

   No: the empty word \( \epsilon \) is accepted by \( A \), but \( \epsilon \) does not satisfy formula \( (\Diamond q) \), which requires a valid position in the word \( \epsilon \) has none, that is
Another counterexample is the word \( w = pp \), where \( q \) does not appear.

(5) \( \mathcal{A} \models \Box (p \implies (p \lor q)) \)

No: the word \( w = pp \) is accepted by \( \mathcal{A} \), but \( w, 2 \neq p \lor q \) because no \( q \) occurs eventually.

4.2 Automata-based model checking (4 points)

Consider the LTL formula:

\[
\phi \triangleq \Box (r \implies X (q \lor (\neg r \land X p)))
\]

Property automaton. Construct an automaton \( \mathcal{F} \) that accepts precisely the words that satisfy \( \phi \).

Solution:
An edge with two letters denote a double transition (one for each of the letters).
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.   $z := \text{True}$
4. else
5.   $z := \text{False}$
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 $\text{assume exp end}$ is $\text{assume not Pred(not exp) end}$ followed by a parallel conditional assignment updating the predicates with respect to the original $\text{assume}$.

- The predicate abstraction of an assert instruction $\text{assert exp end}$ simply is $\text{assert Pred(exp) end}$, provided that $\text{exp}$ 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:

- $p \equiv x > y$
- $q \equiv y > 0$
- $r \equiv z = \text{True}$

Solution:

After the usual simplifications, the predicate abstraction $A$ is:

1. assume $r$ end
2. if ? then
3.   assume $p$ and not $q$ end
4.   $r := \text{True}$
5. else
6.   $r := \text{False}$
7. end
8. assert not $q$ or not $r$ end

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.

Solution:

<table>
<thead>
<tr>
<th>Initial state I</th>
<th>Classification (invalid, spurious, error, valid)</th>
</tr>
</thead>
<tbody>
<tr>
<td>{p, q, r}</td>
<td>valid or invalid</td>
</tr>
<tr>
<td>{not p, q, r}</td>
<td>valid or invalid</td>
</tr>
<tr>
<td>{p, not q, r}</td>
<td>valid</td>
</tr>
<tr>
<td>{not p, not q, r}</td>
<td>valid or invalid</td>
</tr>
<tr>
<td>{p, q, not r}</td>
<td>invalid</td>
</tr>
<tr>
<td>{not p, q, not r}</td>
<td>invalid</td>
</tr>
<tr>
<td>{p, not q, not r}</td>
<td>invalid</td>
</tr>
<tr>
<td>{not p, not q, not r}</td>
<td>invalid</td>
</tr>
</tbody>
</table>

Every initial state I with the component “not r” is invalid because of the initial assume in A. The remaining states are valid since they can lead to the error-free abstract traces listed below. Furthermore, the initial states {p, q, r}, {not p, q, r}, and {not p, not q, r} are also invalid because the nondeterministic conditional if ? may lead to a violation assume p and not q end, that is to an infeasible trace.

```plaintext
{p, q, r}
  [not (p and not q)]
{p, q, r}
r := False
{p, q, not r}
  assert not q or not r end

{not p, q, r}
  [not (p and not q)]
{not p, q, r}
r := False
{not p, q, not r}
  assert not q or not r end
```
\{p, \text{not } q, r\} \\
[p \text{ and not } q] \\
\{p, \text{not } q, r\} \\
r := \text{True} \\
\{p, \text{not } q, r\} \\
\text{assert not } q \text{ or not } r \text{ end} \\
\{\text{not } p, \text{not } q, r\} \\
[\text{not } (p \text{ and not } q)] \\
\{\text{not } p, \text{not } q, r\} \\
r := \text{False} \\
\{\text{not } p, \text{not } q, \text{not } r\} \\
\text{assert not } q \text{ or not } r \text{ end} \\

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

\text{Solution:} \\
\text{Since A determines a set of abstract traces that are an over-approximation of C’s set of concrete traces, we can conclude that C is correct because A is.}