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!

Question	Available points	Your points
1) Axiomatic semantics	8	
2) Separation logic	12	
3) Data Flow Analysis	9	
4) Model checking	9	
5) Software model checking	11	
Total	**49**	

[This page is intentionally left blank.]

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.

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

```
from
   k := 0
until k = n loop
   if k mod 3 = 0 then
      b[k] := a[k] + 1
   else
   end
   k := k + 1
end
```

$$\{\ \forall\ i : 0 \leq i < n \ \wedge\ i \ \mathbf{mod}\ 3 = 0 \Longrightarrow b[i] = 1\ \}$$

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.

[This page is intentionally left blank.]

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:

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

where 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 := cons(0,0);
temp1 := [x];
temp2 := [x+1];
[y] := temp1;
[y+1] := nil;
dispose(x);
dispose(x+1);
x := temp2;
```

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

. . . .

. . . .

. . . .

. . . .

. . . .

. . . .

. . . .

. . . .

. . . .

. . . .

. . . .

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). $(x \mapsto 6, 7) * \neg(x \mapsto 6, 7)$

(b). $(x \mapsto 6, 7) \wedge (y \mapsto 6, 7)$

(c). $\mathrm{emp} \Rightarrow x = y$

(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 mth Fibonacci number for $m \geq 1$.

```
f0 := 0
f1 := 1
if m ≤ 1 then
  f2 := m
else
  from
    i := 2
  until i > m loop
    i := i + 1
    f2 := f0 + f1
    f0 := f1
    f1 := f2
  end
end
print (f2)
```

(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 *f0*, *f1*, *f2*, *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 ℓ' can be replaced by y if $[x := y]^{\ell}$ is the only definition of x that reaches ℓ' and y is not modified between ℓ and ℓ'.

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

[This page is intentionally left blank.]

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 $p, q \in \mathcal{P}$).

$w, i \models p$	iff	$p = w(i)$
$w, i \models \neg\phi$	iff	$w, i \not\models \phi$
$w, i \models \phi_1 \wedge \phi_2$	iff	$w, i \models \phi_1$ and $w, i \models \phi_2$
$w, i \models \mathsf{X}\phi$	iff	$i < n$ and $w, i+1 \models \phi$
$w, i \models \phi_1 \,\mathsf{U}\, \phi_2$	iff	there exists $i \leq j \leq n$ such that: $w, j \models \phi_2$ and for all $i \leq k < j$ it is the case that $w, k \models \phi_1$
$w, i \models \Diamond\, \phi$	iff	there exists $i \leq j \leq n$ such that: $w, j \models \phi$
$w, i \models \Box\, \phi$	iff	for all $i \leq j \leq n$ it is the case that: $w, j \models \phi$
$w \models \phi$	iff	$w, 1 \models \phi$

4.1 Automata and LTL formulas (5 points)

Consider the automaton $\mathcal{A}$ (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.

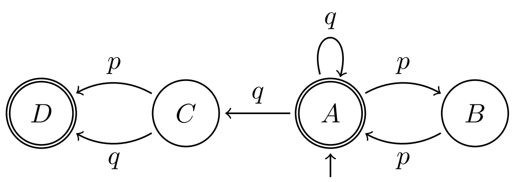

Figure 1: Automaton $\mathcal{A}$ over alphabet $\{p, q\}$.

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

(1) $\mathcal{A} \models \Box(\Diamond\, q)$

. . . .

. . . .

. . . .

. . . .

. . . .

(2) $\mathcal{A} \models \Box(p \implies \mathsf{X}\, p)$

. . . .

. . . .

. . . .

. . . .

. . . .

. . . .

(3) $\mathcal{A} \models p \implies \mathsf{X}\, p$

. . . .

. . . .

. . . .

. . . .

. . . .

. . . .

(4) $\mathcal{A} \models \Diamond(\Box\, q)$

. . . .

. . . .

. . . .

. . . .

. . . .

(5) $\mathcal{A} \models \square(p \implies (p \mathsf{U} q))$

. . . .

. . . .

. . . .

. . . .

. . . .

4.2 Automata-based model checking (4 points)

Consider the LTL formula:

$$\phi \quad \triangleq \quad \square(r \implies \mathsf{X}\,(q \,\mathsf{U}\, (\neg r \wedge \mathsf{X}\, p)))$$

Property automaton. Construct an automaton $\mathcal{F}$ that accepts *precisely* the words that satisfy ϕ.

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.

```
assume z = True end
if x > y and x ≤ 0 then
    z := True
else
    z := False
end
assert y ≤ 0 or not z end
```

Recall that:

- *Pred* (*exp*) denotes the weakest under-approximation of the expression *exp* that is expressible as a Boolean combination of the given predicates.
- The predicate abstraction of an assume instruction **assume** *exp* **end** is **assume not** *Pred* (**not** *exp*) **end** followed by a parallel conditional assignment updating the predicates with respect to the original **assume**.
- The predicate abstraction of an assert instruction **assert** *exp* **end** simply is **assert** *Pred* (*exp*) **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:

$$\begin{aligned} p &\equiv x > y \\ q &\equiv y > 0 \\ r &\equiv z = \textbf{True} \end{aligned}$$

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.

INITIAL STATE I	CLASSIFICATION (invalid, spurious, error, valid)
$\{p,\ q,\ r\}$	....
$\{$**not** $p,\ q,\ r\}$	....
$\{p,$ **not** $q,\ r\}$	....
$\{$**not** $p,$ **not** $q,\ r\}$	....
$\{p,\ q,$ **not** $r\}$	....
$\{$**not** $p,\ q,$ **not** $r\}$	....
$\{p,$ **not** $q,$ **not** $r\}$	....
$\{$**not** $p,$ **not** $q,$ **not** $r\}$	....

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.

....

....

....

....

....

....

....