Software Verification – Exam

ETH Zürich

19 December 2011

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	18	
2) Separation logic	15	
3) Data flow analysis	10	
4) Model checking	15	
5) Software model checking	12	
Total	**70**	

[This page is intentionally left blank.]

1 Axiomatic semantics (18 points)

Consider the following annotated program, where A is an array (indexed from 1), n is an integer variable storing A's size, i, j, and **Result** are other integer variables, and $[1..n]$ denotes an interval of the integers from 1 to n included.

```
{ n ≥ 1 }
from
    i := 1
    j := n
until i = j loop
   if A[i] > A[j] then
      j := j − 1
   else
      i := i + 1
   end
end
Result := A[i]
{ ∀k ∈ [1..n]: Result ≥ A[k]  ∧  Result = A[i] }
```

1.1 Program semantics (2 points)

Characterize, in plain English, which value of **Result** the program computes from the inputs A and n. In other words: what does the program do?

....

....

....

....

....

1.2 Partial correctness (14 points)

Prove that the triple (precondition, program, postcondition) is a theorem of Hoare's axiomatic system for partial correctness.

....

....

....

1.3 Termination (2 points)

Find a suitable *variant* function V to prove termination. V must be such that it decreases along all branches of the loop body, and it is nonnegative after every iteration of the loop. You do *not* have to prove termination, just write a suitable variant and informally argue why it is a suitable variant.

. . . .

. . . .

. . . .

. . . .

. . . .

. . . .

2 Separation Logic (15 points)

2.1 Predicates and satisfaction

1. Define a recursive predicate *tree* t i which asserts that i is a pointer to a well-formed binary tree t. Here

   $t \stackrel{\text{def}}{=} n \mid (t_1, t_2)$

   so a tree value t can be either a leaf, which is a single number n, or an internal node with a left subtree t_1 and a right subtree t_2.

   ....

   ....

   ....

   ....

   ....

   ....

2. Draw the diagram of a state satisfying *tree* $(1, ((2, 3), 4))$ i.

2.2 Code verification

Give a proof of the following triple:

$\{x \mapsto a * y \mapsto b\}\ t := [x]; \quad [y] := t; \quad \textbf{dispose}\ x\ \{y \mapsto a\}$

3 Program slicing (10 points)

Consider the following program fragment (all variables are of type *INTEGER*):

```
a := 42
from
    i := 0
until i ≥ 10 loop
    if c < 0 then
        b := a
        x := 1
    else
        c := b
        y := 2
    end
    i := i + 1
end
print (c)
print (x + y)
```

(1) (5 points) Draw the Program Dependency Graph (PDG) of the program fragment.

(2) (3 points) Using the PDG, compute the static backward slice of the program fragment for both following slicing criteria: **(a)** line 14; **(b)** line 15. In both cases clearly mark the nodes you have visited with the slicing algorithm in the PDG, and provide the line number(s) of those lines which are deleted from the original program fragment (i.e. are *not* part of the slice).

. . . .

. . . .

. . . .

. . . .

(3) (2 points) For the same original program P, a program slice S_1 is said to be *more precise* than a slice S_2 if S_1 is a slice of S_2 as well as P and contains fewer statements than S_2.

For the program slice computed in **(2) (b)**, argue whether or not there exists a more precise slice; provide this slice in case it exists.

. . . .

. . . .

. . . .

. . . .

. . . .

. . . .

. . . .

. . . .

. . . .

. . . .

. . . .

4 Model Checking (15 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 (7 points)

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

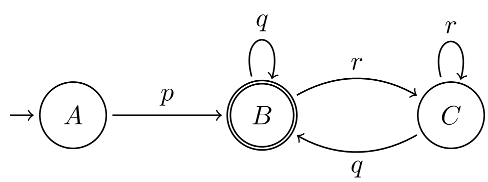

Figure 1: Automata A over alphabet $\{p, q, r\}$.

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) $\mathcal{A} \models \Diamond\, p$

. . . .

. . . .

. . . .

. . . .

. . . .

(2) $\mathcal{A} \models (\Diamond \neg p) \Longrightarrow (\Box\, q)$

. . . .

. . . .

. . . .

. . . .

. . . .

(3) $\mathcal{A} \models \mathsf{X}(\Box \neg p)$

. . . .

. . . .

. . . .

. . . .

. . . .

. . . .

(4) $\mathcal{A} \models (\mathsf{X}\mathsf{X}\mathsf{X}\, q) \Longrightarrow (\mathsf{X}\mathsf{X}\, q)$

. . . .

. . . .

. . . .

. . . .

. . . .

. . . .

(5) $\mathcal{A} \models p \implies \mathsf{X}(r \mathbin{\mathsf{U}} q)$

. . . .

. . . .

. . . .

. . . .

. . . .

4.2 Automata-based model checking (8 points)

Show that $\mathcal{A} \not\models p \implies \mathsf{X}\, q$ using automata-based model checking as follows.

Property automaton (4 points). Construct an automaton $\mathcal{P}$ that accepts *precisely* the words that satisfy $\neg(p \implies \mathsf{X}\, q)$, that is, the *complement* of the property we want to falsify.

Intersection automaton (4 points). Construct the intersection automaton $\mathcal{A} \times \mathcal{P}$ that accepts precisely the words accepted by both $\mathcal{A}$ and $\mathcal{P}$. Show that $\mathcal{A} \times \mathcal{P}$ accepts some word.

5 Software model checking (12 points)

Consider the following code snippet C, where x, y, z are integer variables.

```
assume x >0 end
z := (x * y) + 1
assert z ≥ 1 end
```

Recall that:

- The Boolean abstraction of an **assume** c **end** statement is **assume not** *Pred* (**not** c) **end** followed by a parallel conditional assignment updating the predicates with respect to the original **assume** statement.
- Similarly, the Boolean abstraction of an **assert** c **end** statement is **assert not** *Pred* (**not** c) **end** followed by a parallel conditional assignment updating the predicates with respect to the original **assert** statement.
- *Pred* (f) denotes the weakest under-approximation of the expression f expressible as a Boolean combination of the given predicates.

5.1 Boolean abstractions (10 points)

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

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

5.2 Error traces (2 points)

Provide an annotated trace for the Boolean abstraction A, and a corresponding annotated trace for the concrete program C that is feasible and such that **assert** $z \geq 1$ **end** evaluates to **False** when reached. Note that in general there are multiple traces of C corresponding to the same trace of A: you must select one which is feasible and violates the assertion.

The trace of A should be in the form of a valid sequence of statements and branch conditions in A which reaches the bottom of A. Each statement in the sequence must be preceded by a complete description of the abstract program state in terms of values of the Boolean predicates p, q, r. Similarly, the trace of C should be in the form of a valid sequence of statements and branch conditions in C which reaches the bottom of C. Each statement in the sequence must be preceded by a concrete value for the variables x, y, z which satisfies the corresponding state in the abstract trace of A.

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

. . . .