Software Verification – Exam

ETH Zürich

16 December 2013

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	12	
2) Separation logic	10	
3) Abstract interpretation	10	
4) Model checking	10	
5) Real time verification	10	
Total	**52**	

[This page is intentionally left blank.]

1 Axiomatic semantics (12 points)

Consider the following annotated program, where A is an array (indexed from 1) of generic element type G, n is an integer variable storing A's size, i and j are two other integer variables, and *swap* (A, i, j) is a call to routine *swap* whose effect is to swap the elements at indexes i and j in array A (provided i and j are valid indexes), and **old** $A[n - k + 1]$ in the postcondition refers to the value of the expression $A[n - k + 1]$ in the precondition.

```
{ $n \geq 0$ }
from
    $i := 1$
    $j := n$
until $j \leq i$ loop
    swap $(A, i, j)$
    $i := i + 1$
    $j := j - 1$
end
{ $\forall\, k\, (1 \leq k \leq n \Longrightarrow A[k] = \textbf{old}\ A[n - k + 1])$ }
```

1.1 Program semantics (2 points)

Formalize the effect of a generic call *swap* (A, i, j) by providing assertion formulae P and Q such that both

- $\{P\}$ *swap* (A, i, j) $\{\ x \neq i\ \wedge\ x \neq j\ \wedge\ A[x] = 8\ \}$ and
- $\{Q\}$ *swap* (A, i, j) $\{\ x = j\ \wedge\ i \neq j\ \wedge\ A[x] = 7\ \}$

are valid Hoare triples. You can assume a variable n denotes A's size.

1.2 Partial correctness (10 points)

Using the following (partial) proof rule for *swap* (where x is an integer variable, B an integer array, and m denotes B's size):

$$\begin{array}{l} \{\ 1 \leq\ x \leq\ m\ \wedge \forall\ k\ (1 \leq k < x\ \Longrightarrow\ B[k] = \textbf{old}\ B[m-k+1])\ \} \\ \quad swap\ (B,\ x,\ m-x+1) \\ \{\ 1 \leq\ x \leq\ m\ \wedge \forall\ k\ (1 \leq k \leq x\ \Longrightarrow\ B[k] = \textbf{old}\ B[m-k+1])\ \} \end{array}$$

prove that the triple (precondition, program, postcondition) introduced in the previous page 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 (10 points)

2.1 Predicates and Constructing States (4 points)

A well-formed binary tree t is defined by the grammar:

$$t \triangleq n \mid (t_1, t_2)$$

i.e. 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. Consider the following definition of the inductive predicate $tree(t, i)$ which asserts that i is a pointer to a well-formed binary tree t:

$$\begin{aligned} tree(n, i) &\triangleq i \mapsto n \\ tree((t_1, t_2), i) &\triangleq \exists l, r.\ i \mapsto l, r * tree(t_1, l) * tree(t_2, r) \end{aligned}$$

Using these definitions, *draw state diagrams* (i.e. stores and heaps) satisfying each of the following formulae:

(a). $tree((8, 8), x)$

(b). $tree(5, x) \wedge tree(5, y)$

(c). $tree(((3, 2), 1), x)$

2.2 Semantics of Triples (2 points)

Why is the following triple *not* valid?

$$\{\text{true}\}\ [x] := 7\ \{\text{true}\}$$

. . . .

. . . .

. . . .

. . . .

. . . .

. . . .

. . . .

. . . .

. . . .

. . . .

2.3 Pointer Proof (4 points)

Give a proof outline for the following triple, using the axioms and inference rules of separation logic:

$\{\text{emp}\}$

```
    x := cons(3,4);
    z := [x+1];
    y := cons(z,3);
    [x+1] := [y+1];
    dispose(y+1);
```

$\{y \mapsto 4 * x \mapsto 3, 3\}$

. . . .

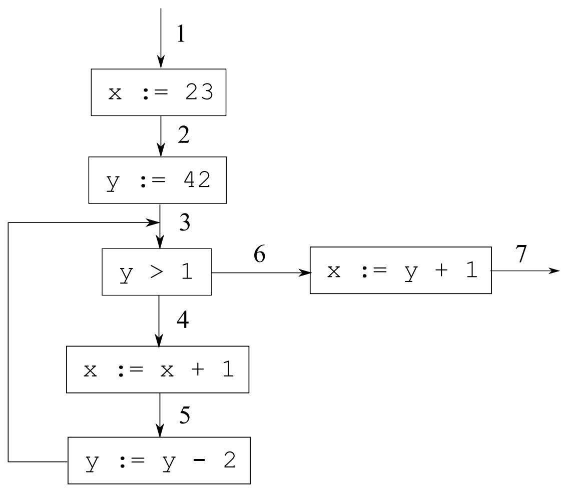
. . . .

. . . .

. . . .

3 Abstract interpretation (10 points)

Consider the following control flow graph of a program fragment, annotated with labels $1, 2, \ldots, 7$. All variables are of type *INTEGER*.



The goal of this exercise is to use abstract interpretation to perform *parity analysis*, i.e. to determine which variables are *even* or *odd*. The computation in the abstract domain is done in the complete lattice **Parity**

$$\begin{tikzcd}
 & \top \arrow[dl, dash] \arrow[dr, dash] & \\
E & & O \\
 & \bot \arrow[ul, dash] \arrow[ur, dash] &
\end{tikzcd}$$

where $\top$ represents all integers, E even integers, O odd integers, and $\bot$ the empty set.

(1) Specify the abstract operations $\oplus$ and $\ominus$ on **Parity** that correspond to integer addition and subtraction, respectively.

(2) Define an equation system to compute the values of the abstract states $A_i : \{\mathtt{x}, \mathtt{y}\} \to \textbf{Parity}$ at each program point $i \in \{1, 2, \ldots, 7\}$.

....

....

....

....

....

....

....

....

....

....

....

....

(3) Compute the fixed point of your equation system by iteration and display the effect of each iteration step in the table below.

In the table, it suffices to enter the pair of the abstract values for `x` and `y` when they are initialized or recomputed. For example, if in step 3 of the iteration, at program point i the variable `x` gets the value $\top$ and `y` has the value O, enter $\top O$ in row A_i column 3; if in the following iteration, A_i does not change, leave row A_i column 4 empty.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A_1																
A_2																
A_3																
A_4																
A_5																
A_6																
A_7																

4 Model Checking (10 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 $\vDash$ is defined recursively as follows (where $p, q \in \mathcal{P}$).

$w, i \vDash p$	iff	$p = w(i)$
$w, i \vDash \neg\phi$	iff	$w, i \nvDash \phi$
$w, i \vDash \phi_1 \wedge \phi_2$	iff	$w, i \vDash \phi_1$ and $w, i \vDash \phi_2$
$w, i \vDash \mathsf{X}\phi$	iff	$i < n$ and $w, i+1 \vDash \phi$
$w, i \vDash \phi_1 \mathsf{U} \phi_2$	iff	there exists $i \leq j \leq n$ such that: $w, j \vDash \phi_2$ and for all $i \leq k < j$ it is the case that $w, k \vDash \phi_1$
$w, i \vDash \Diamond \phi$	iff	there exists $i \leq j \leq n$ such that: $w, j \vDash \phi$
$w, i \vDash \Box \phi$	iff	for all $i \leq j \leq n$ it is the case that: $w, j \vDash \phi$
$w \vDash \phi$	iff	$w, 1 \vDash \phi$

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.

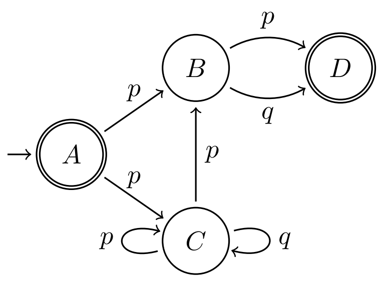


Figure 1: Automaton $\mathcal{A}$ over alphabet $\{p, q\}$.

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) $\mathcal{A} \vDash \Box(\Diamond p)$

....

....

....

....

(2) $\mathcal{A} \models \Diamond(\text{True}) \implies \Diamond(p)$

(3) $\mathcal{A} \models p \implies (p \,\mathsf{U}\, (p \vee q))$

(4) $\mathcal{A} \models (p \wedge \mathsf{X}\, q) \implies \mathsf{X}\mathsf{X}\Diamond(q)$

(5) $\mathcal{A} \vDash \Diamond(p) \implies p$

. . . .

. . . .

. . . .

. . . .

. . . .

4.2 Automata-based model checking (5 points)

Consider the LTL formula:

$$\phi \quad \triangleq \quad \Diamond(p) \;\wedge\; (p \implies \mathsf{XX}(\Box(\text{False})))$$

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

5 Real Time Verification (10 points)

Recall the semantics of MTL over finite timed words with alphabet $\mathcal{P}$ and time domain $\mathbb{R}_{\geq 0}$ (the nonnegative real numbers). For a timed word

$$w = [\sigma(1), t(1)][\sigma(2), t(2)] \cdots [\sigma(n), t(n)] \in (\mathcal{P} \times \mathbb{R}_{\geq 0})^*$$

with $n \geq 0$ and a position $1 \leq i \leq n$ the satisfaction relation $\models$ is defined recursively as follows for $p \in \mathcal{P}$ and J an interval of $\mathbb{R}_{\geq 0}$ with integer endpoints.

$w, i \models p$	iff	$p = \sigma(i)$
$w, i \models \neg\phi$	iff	$w, i \not\models \phi$
$w, i \models \phi_1 \wedge \phi_2$	iff	$w, i \models \phi_1$ and $w, i \models \phi_2$
$w, i \models \Diamond_J \phi$	iff	there exists $i \leq j \leq n$ such that: $t(j) - t(i) \in J$ and $w, j \models \phi$
$w, i \models \Box_J \phi$	iff	for all $i \leq j \leq n$: if $t(j) - t(i) \in J$ then $w, j \models \phi$
$w, i \models \phi_1 \mathsf{U}_J \phi_2$	iff	there exists $i \leq j \leq n$ such that: $t(j) - t(i) \in J$, $w, j \models \phi_2$, and, for all $i \leq k < j$, $w, k \models \phi_1$
$w \models \phi$	iff	$w, 1 \models \phi$

5.1 Timed automata and MTL formulae (5 points)

Consider the timed automaton $\mathcal{T}$ (with locations A, B, C) in Figure 2, over the alphabet $\{p, q\}$ and with clocks x and y. Notice that A is the initial location and B is final.

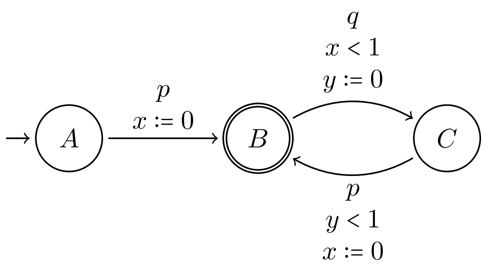


Figure 2: Timed automaton $\mathcal{T}$ over alphabet $\{p, q\}$ with clocks x and y.

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

(1) $\mathcal{T} \models \Diamond_{(0,1)} p$

. . . .

. . . .

. . . .

(2) $\mathcal{T} \vDash \Box_{[0,1)}\Big(p \Longrightarrow \Diamond_{(0,1)}\, q\Big)$

(3) $\mathcal{T} \vDash \Big(p \,\mathsf{U}_{(0,1)}\, q\Big) \Longrightarrow \Diamond_{(0,2)}\, p$

(4) $\mathcal{T} \vDash \Box_{[0,1]}(\neg q)$

5.2 Emptiness check of timed automata (5 points)

Construct the *region automaton* $\mathsf{reg}(\mathcal{T})$ for the timed automaton $\mathcal{T}$ in Figure 2. Recall that $\mathsf{reg}(\mathcal{T})$ is a finite-state automaton whose states are labeled with a pair (ℓ, R), where ℓ is a location of $\mathcal{T}$ (A, B, or C), and R is a *region*, that is an equivalence class of clock valuations specified by a system of equalities and inequalities involving the clocks x and y. Make sure to also mark the initial and final states of $\mathsf{reg}(\mathcal{T})$.