Formal Methods and Tools
Faculty of EEMCS
University of Twente

EXAMINATION
SYSTEM VALIDATION

code: **214012**
date: **28 August 2008**
time: **9.00–12.30**

- When taking this exam, you are allowed to have a copy of the *lecture notes*, a copy of the *slides* and one *book* of your own choice.
- The final grade for System Validation is built up from the grade for the SPIN assignment (S), the grade for the SUMO project (P) and the grade for this written examination (T):

  $$\textit{final grade} = (S + 2 \times P + 2 \times T)/5$$

  However, if T is less than 5 then the final grade for System Validation will at most be 4.
- You can earn 100 points with the following 7 questions. The score for the examination is $T = \textit{your score}/10$, rounded to one digit after the decimal point.

VEEL SUCCES!

Question 1 (15 points)

Suppose we have two users, *Koot* and *Bie*, and a single printer device *Printer*. Both users perform several tasks, and every now and then they want to print their results on the *Printer*. Since there is only a single printer, only one user can print a job at a time. Suppose we have the following atomic propositions for *Koot* at our disposal:

- *Koot.request* – indicates that *Koot* requests usage of the printer;
- *Koot.use* – indicates that *Koot* uses the printer;
- *Koot.release* – indicates that *Koot* releases the printer.

For *Bie* similar predicates are defined.

Specify in Linear Temporal Logic (LTL) the following properties:

a. *(3p) Mutual exclusion:* only one user at a time can use the printer.

b. *(4p) Finite time of usage:* a user can print only for a finite amount of time.

c. *(4p) Absence of starvation:* if a user wants to print something, he eventually is able to do so.

d. *(4p) Alternating access:* users must strictly alternate in printing.

Question 2 (15 points)

For each property below, give – if possible – an LTL formula expressing the property. If it is not expressible, explain why. We assume p, q, and r to be atomic propositions.

a. After p has happened, q will never be true.

b. The events p and q come in pairs: after each p there will be q before a new p appears. Furthermore between each pair of p and q, r is never true.

PQ PQ PQPQ

(PQ)Q(PQ)

(PQ) Q Q R Q Q (PQ) QQQR

$p \leq 1 \rightarrow GF(p \leq 2)$

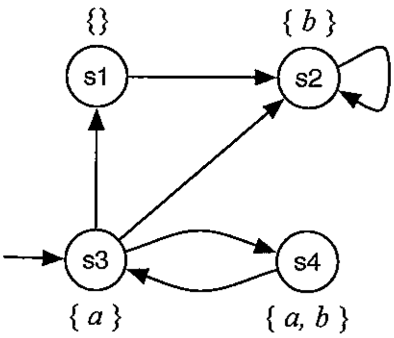
c. Transitions to states satisfying p occur at most twice, i.e. there are most two states in the path where p is true.

d. Event p always ~~precedes~~ directly q. $\neq$ $p \rightarrow X Q$

e. Property p is true in each 'odd' state but false in each 'even' state, i.e. p is true in the 1st, 3rd, 5th, etc. state, but false in the 2nd, 4th, 6th, etc. state.

Question 3 (10 points)

Consider the following Kripke structure M that consists of four states.



For each of the following formulae ϕ below,

(i) Find an infinite path from the initial state s3 which satisfies ϕ, *and*

(ii) Determine whether $M \models \phi$. If not, provide a counterexample.

The formulae ϕ are the following:

a. $\phi \equiv \mathsf{G}\, a$

b. $\phi \equiv a \,\mathsf{U}\, b$

c. $\phi \equiv a \,\mathsf{U}\, \mathsf{X}(a \wedge \neg b)$

d. $\phi \equiv \mathsf{X} \neg b \;\wedge\; \mathsf{G}(\neg a \vee \neg b)$

e. $\phi \equiv \mathsf{X}(a \wedge b) \;\wedge\; \mathsf{F}(\neg a \wedge \neg b)$

P Q Q Q P R P R Q Q

R RQRQRQ

PR PQ

P Q R

P P P Q Q

$P \rightarrow XQ \vee XP$

Question 4 (15 points)

The partial order reduction algorithm of Doron Peled (see Chapter 10 of [Clarke et.al. 1999], i.e. [Peled 1999]) is centralised around four constraints on the set $ample(s)$. The most complicated among the constraints is condition **C1**. Suppose that we replace condition **C1** by the following condition **C1′**, which is easier to understand and use:

C1′ The transitions in $ample(s)$ are all independent of those in $enabled(s) \setminus ample(s)$.

Is the partial order reduction algorithm still correct when using **C1′**?
If yes, give a proof (sketch). If not, explain why it is not correct.

ample(S1) = C

C DEP ON AMPLE(S0)

S7-S10 CANNOT OCCUR W/O S2-S5 FIRST

AMPLE(S0) = A, B

P Q P Q P

P P Q P Q P

Question 5 (20 points)

In this exercise we consider the organisation of the states of the state space of a small system. Each state s consists of three components: $s.x$, $s.y$ and $s.z$. During exploration, the following states are being generated (in the order $s_0, s_1, \ldots s_9$).

s	$s_i.x$	$s_i.y$	$s_i.z$
s_0	3	5	2
s_1	1	2	3
s_2	2	4	4
s_3	2	1	2
s_4	1	4	6
s_5	2	1	4
s_6	1	2	2
s_7	2	3	2
s_8	1	5	4
s_9	2	2	4

You are asked to give the organisation of the resulting state space – i.e. the hash table and its buckets containing the states themselves – for the following situations.

a. *(6p)* The state space is organised using a hash table that uses *direct chaining*. The states themselves are stored without any form of compression. The hash table has 7 buckets and the following hash function is being used:

$$h_a(s) = (s.x + s.y + s.z) \,\%\, 7$$

b. *(7p)* The state space is organised using a hash table that uses *separate chaining* and where the states are stored in a compressed way using the recursive indexing method. Use 2 bits for the index of the table for $s.x$, 3 bits for the index of the table for $s.y$ and 3 bits for the index of the table for $s.z$. The hash table has 11 buckets and the following hash function is being used:

$$h_b(s) = (s.x + 2 * s.y + 2 * s.z) \,\%\, 11$$

where s is the *original*, non-compressed state.

c. *(7p)* The states are stored using 2-fold bit-state hashing, using the following two hash-functions

$$\begin{aligned} h_{c1}(s) &= h_b(s) \\ h_{c2}(s) &= (2 * s.x + s.y + s.z) \,\%\, 11 \end{aligned}$$

The bucket size is 11.
During exploration, which states are *wrongly* considered to be visited already?

Question 6 (10 points)

Answer the following questions (using at most five sentences for each question):

a. *(3p)* Suppose the approach of [Kattenbelt et.al. 2007] is used to design and implement a generic model checker. On what layer should partial order reduction be implemented? Explain your answer.

b. *(3p)* Virtual machine based model checkers like JPF or MMC exploit the fact that a transition is typically local: only a small part of the (current) state is changed by a transition. The model checker SPIN does not exploit the locality of transitions. Why is that?

c. *(4p)* Consider a state space explorer, which is used to check for deadlocks. The state space explorer uses a conventional hash table to store the states. However, each time when the hash table gets full, the explorer *randomly* removes half of the states from the hash table. Does this approach always terminate? If not, how can we ensure that the approach does terminate?

Question 7 (15 points)

Apply *runtime analysis* (i.e. the *Eraser* algorithm as described by Visser et.al. 2002) to the Java program below and explain the potential of a *data race*.

```
public class MyThreads {
    public static void main(String[] args) {
        Value v1 = new Value();
        Value v2 = new Value();
        Task t1  = new Task(v1, v2); t1.start();
        Task t2  = new Task(v2, v1); t2.start();
    }
}

class Value {
    private int x = 1;
    public synchronized void add(Value v) {
        x = x + v.get();
    }
    public int get() {
        return x;
    }
}

class Task extends Thread {
    Value v1, v2;

    public Task(Value v1, Value v2) {
        this.v1 = v1;
        this.v2 = v2;
    }

    public void run() {
        v1.add(v2);
    }
}
```