HAND IN EXAM
Answers recorded on examination paper

Queen's University
Faculty of Arts and Science
School of Computing
CISC/CMPE 422 and CISC 835
Final Examination

Friday, Dec 18, 2015

Instructor: J. Dingel

Student id (clearly printed please): _____

Instructions: This examination is **three hours** in length. You are permitted to have one 8.5"x11" data sheet with information on both sides. No other references, calculators, or aids are allowed.

Read all questions carefully. There are 5 questions in this examination, some with several parts. You must answer all questions. Write all your answers in this exam. An extra page is provided at the end for rough work. If you answer a question on a different page, you must indicate where the answer is to be found.

Important notes: Make sure your student id is clearly printed on this page. If the instructor is unavailable in the examination room and doubt exists as to the interpretation of any question, you are urged to submit your answer with a clear statement of any assumptions made.

Good luck!

For marking use only:

Question	Points
1	/15
2	/28
3	/18
4	/21
5	/28
Total	/110

Id: _____

Question 1: Propositional & Predicate Logic, 15 points

Each of the three parts below contains a predicate logic formula. For each formula, say whether it is satisfiable by writing "yes" or "no". If you answered "yes", write down a model in which the formula holds; if you answered "no", write down a model in which the negation of the formula holds. Remember to fully define your models by providing the domain and the interpretation of any function and predicate symbols used.

1. (5 points) $\big(\forall x.P(x) \rightarrow Q(x)\big) \ \wedge \ \big(\exists x.P(x)\big) \ \wedge \ \big(\neg\forall x.Q(x)\big)$

2. (5 points) $\big(\exists x.P(x, f(x))\big) \ \wedge \ \big(\forall x.\exists y.\neg P(x, y)\big)$

3. (5 points) $\big(\forall x.\exists y.P(x, y)\big) \ \wedge \ \big(\exists y.\forall x.P(x, y)\big)$

Id: _____

Question 2: (Alloy, 28 points)

A program is needed to assign inmates to cells in a prison. To support the design of the program, the following Alloy specification M is constructed:

```
sig Cell {}
sig Inmate {
   room : Cell
}
sig Gang {
   members : set Inmate
}
```

Note how every instance of M represents an assignment of inmates to prison cells via the attribute `room`.

1. (3 points) In the space below, draw the graphical representation (i.e., meta model) of M. Make sure you include multiplicity constraints.

2. (4 points) Consider the following predicate

   ```
   pred show() {
      some Inmate
      some Gang
   }
   ```

   Draw an instance of M that also satisfies all the constraints in `show`, that is, draw an instance that could be created by Alloy in response to executing the command `run show for 3` on M.

Id: _____

Question 2 (Alloy, continued)

For your convenience, the Alloy model M from the previous page is repeated here.

```
sig Cell {}
sig Inmate {
   room : Cell
}
sig Gang {
   members : set Inmate
}
```

3. (4 points) The assignment of inmates to prison cells must be *safe*, that is, it must avoid placing two inmates in the same cell if they are members of different gangs. Complete the predicate `safe()` below such that it characterizes safe assignments, i.e., `safe()` must hold in an instance if and only if that instance represents a safe assignment.

   ```
   pred safe() {
   ```

4. (2 points) Write down the Alloy command that can be used to generate safe instances of M.

5. (2 points) Write down the Alloy command that can be used to generate unsafe instances of M.

6. (4 points) An assignment of inmates to prison cells is called *happy*, if and only if gang members only share cells with members of the same gang. Complete the predicate `happy()` below such that it holds in an instance if and only if that instance is happy.

   ```
   pred happy() {
   ```

Id: _____

Question 2 (Alloy, continued)

For your convenience, the Alloy model M from the previous page is repeated here.

```
sig Cell {}
sig Inmate {
   room : Cell
}
sig Gang {
   members : set Inmate
}
```

7. (5 points) A safe assignment is not necessarily happy. Write down an instance of M that is safe, but not happy.

8. (4 points) Complete the fact `SafetyImpliesHappiness` below such that, when added to M, it ensures that safety will imply happiness, that is, that every safe instance of M also is happy.

```
fact SafetyImpliesHappiness {
```

Id: _____

Question 3: CTL, 18 points

For each of the following three pairs of formulas φ, φ', decide if they are equivalent. If they are equivalent, write **"Yes"**; if they are not equivalent, write **"No"**, draw a Kripke structure (i.e., finite state machine) M such that one formula holds in M, but not the other, and clearly indicate which formula holds in M.

When drawing M, clearly show the initial state and which atomic propositions hold in which states; also, remember that the transition relation of a Kripke structure is total.

1. (6 points) $\varphi = \mathbf{A}[p\ \mathbf{U}\ q]$ and $\varphi' = q \vee \mathbf{AX}\ \mathbf{A}[p\mathbf{U}q]$

2. (6 points) $\varphi = (\mathbf{EF}\ p \wedge q)$ and $\varphi' = (\mathbf{EF}\ p) \wedge (\mathbf{EF}\ q)$

3. (6 points) $\varphi = \mathbf{AX}p \rightarrow \mathbf{AF}p$ and $\varphi' = \neg\mathbf{AF}p \rightarrow \mathbf{EX}\neg p$

Id: _____

Question 4: (Mutual Exclusion Protocols, 21 points)

1) (18 points) In the concurrent pseudo code below, $nc0$ and $nc1$ stand for possibly non-terminating non-critical regions, and $cr0$ and $cr1$ stand for always terminating critical regions. The variables $req0$, $req1$, and $last$ do not get modified in $nc0$, $nc1$, $cr0$, or $cr1$. $B0$ and $B1$ are placeholders for boolean expressions.

$req0 := false\,;\, req1 := false;$

$$
\left[\begin{array}{lcl}
nc0; & & nc1; \\
req0 := true; & & req1 := true; \\
last := 0; & \| & last := 1; \\
\textbf{await } B0; & & \textbf{await } B1; \\
cr0; & & cr1; \\
req0 := false; & & req1 := false;
\end{array}\right]
$$

In Parts 1) to 3) below, three replacements for $B0$ and $B1$ are suggested. For each suggestion, evaluate the behaviour of the resulting program with respect to three properties: deadlock freedom, mutual exclusion, and eventual entry. For each property P, determine how frequently P is satisfied, that is, determine whether

- every execution of the resulting program satisfies P (frequency = "always"),
- the resulting program has at least one execution satisfying P (frequency = "at least once"), or
- every execution of the resulting program violates P (frequency = "never")

and circle the appropriate frequency. You may assume that the scheduler executing the program is fair.

1. If *B0* and *B1* are both replaced by the expression *true*, the resulting program would satisfy:

Deadlock freedom:	always	at least once	never
Mutual exclusion:	always	at least once	never
Eventual entry:	always	at least once	never

2. If *B0* is replaced by the expression $\neg req1$ and *B1* is replaced by the expression $\neg req0$ the resulting program would satisfy:

Deadlock freedom:	always	at least once	never
Mutual exclusion:	always	at least once	never
Eventual entry:	always	at least once	never

3. If *B0* is replaced by $\neg req1 \vee (last \neq 0)$ and *B1* is replaced by $\neg req0 \vee (last \neq 1)$ the resulting program would satisfy:

Deadlock freedom:	always	at least once	never
Mutual exclusion:	always	at least once	never
Eventual entry:	always	at least once	never

Question 4 (continued)

2) (3 points) Very, very briefly (points will not be awarded based solely on the length of your answer) explain why testing typically is insufficient to determine the correctness of concurrent code such as the protocol on the previous page. Also, remember to write legibly. Thanks!

Id: _____

Question 5: SMV, 28 points

Consider the following code defining an NuSMV program `main`:

```
MODULE P1
VAR
  x : {a,b};
ASSIGN
  init(x) := a;
  next(x) := {a,b};
```

```
MODULE P2(x)
VAR
  y : {0,1,2,3,4};
ASSIGN
  init(y) := 0;
  next(y) := case
               next(x)=a & y<3 : y+1;
               y=3 : {1,2};
               TRUE : y;
             esac;
```

```
MODULE main
VAR
  p1 : P1;
  p2 : P2(p1.x);
```

1. (9 points) Draw the finite state machine M that is defined by the code above. Represent a single state s of M by a pair (x, y) where x is the value of `x` in process `p1` and y is the value of `y` in process `p2`. For instance, the initial state of M is represented as $(a, 0)$. Your drawing should clearly indicate the initial states of M, the reachable states of M, the transition relation of M, and the labelling function.

Id: _____

Question 5 (continued)

```
MODULE P1
VAR
  x :  {a,b};
ASSIGN
  init(x) := a;
  next(x) := {a,b};
```

```
MODULE P2(x)
VAR
  y :  {0,1,2,3,4};
ASSIGN
  init(y) := 0;
  next(y) := case
               next(x)=a & y<3 :  y+1;
               y=3 :  {1,2};
               TRUE : y;
             esac;
```

```
MODULE main
VAR
  p1 :  P1;
  p2 :  P2(p1.x);
```

2. (9 points) Let M be the Kripke structure (i.e., Finite State Machine) defined by program `main` (repeated above for your convenience).

   1. (2 points) If you ignore reachability, how many different states could M be in at most?

   2. (2 points) How many of these theoretically possible states are unreachable?

   3. (5 points) Pick one of these unreachable states s and write down a CTL formula φ_s that allows you to use NuSMV to check with that s is indeed unreachable. Explain how you would use NuSMV very briefly.

Id: _____

Question 5 (continued)

3. (20 points) For each of the following properties φ_1 through φ_5, express it formally in CTL and decide whether or not it is satisfied by program `main` given in Part 1) of this question by writing **"Yes"** or **"No"**. No justification necessary. A *successor* of a state s is a state that can be reached in one step from s.

1. (4 points) φ_1: *"Along all paths, if* `p2.y` *is* `2` *in some state s, then* `p2.y` *will be greater than 1 in all states reachable from s."*
   - φ_1 in CTL:
   - φ_1 true in `main`?:
2. (4 points) φ_2: *"There exists a path in M along which we eventually get to a state s from which there exists a path along which* `p2.y` *will always be* `2`*, and* `p1.x` *will be* `a` *until s."*
   - φ_2 in CTL:
   - φ_2 true in `main`?:
3. (4 points) φ_3: *"Along every path it is always the case that when* `p1.x` *is* `a` *and* `p2.y` *is* `1` *in some state s, then* `p2.y` *is greater* `1` *in all successors of all successors of s."*
   - φ_3 in CTL:
   - φ_3 true in `main`?:
4. (4 points) φ_4: *"Along all paths,* `p1.x` *will be equal to* `b` *eventually."*
   - φ_4 in CTL:
   - φ_4 true in `main`?:
5. (4 points) φ_5: *"Along all paths,* `p2.y` *will be* `3` *eventually and every state until then will have at least one successor in which* `p1.x` *is* `b`*."*
   - φ_5 in CTL:
   - φ_5 true in `main`?:

Id: _____

Scratch sheet: