Student id: ____________________

HAND IN
Answers recorded on examination paper

QUEEN'S UNIVERSITY FINAL EXAMINATION

FACULTY OF ARTS AND SCIENCE
SCHOOL OF COMPUTING

CISC/CMPE 422 and CISC 835

Instructor: J. Dingel

Sunday, Dec 15, 2019

INSTRUCTIONS TO STUDENTS:

This examination is 3 HOURS in length.
Please answer all questions in the exam.

The following aids are allowed:
One 8.5"x 11" data sheet

Put your student number on all pages including this one (see upper right corner).
GOOD LUCK!

PLEASE NOTE:
Proctors are unable to respond to queries about the interpretation of exam questions.
Do your best to answer exam questions as written.

This material is copyrighted and is for the sole use of students registered in CISC/CMPE 422, CISC 835 and writing this exam. This material shall not be distributed or disseminated. Failure to abide by these conditions is a breach of copyright and may also constitute a breach of academic integrity under the University Senate's Academic Integrity Policy Statement.

For marking use only:

Q1	/15
Q2	/38
Q3	/18
Q4	/18
Q5	/40
Total	/129

Id: ____________________

Question 1: Predicate Logic (15 points total)

Consider the following two predicate logic formulas

$$\varphi_1 = \forall x. \neg\big[P(f(x)) \rightarrow \forall y. Q(y)\big]$$

$$\varphi_2 = \forall x. \big[(\neg P(f(x))) \rightarrow \forall y. Q(y)\big]$$

where x and y are variables, P and Q are predicate symbols of arity 1, and f is a function symbol of arity 1.

a) φ_1 and φ_2 are not equivalent. Find a model $\mathcal{M}$ such that one of the two formulas holds in $\mathcal{M}$ and the other one does not. Provide a *complete* definition of your model $\mathcal{M}$ in the space below.

b) Which of the two formulas holds in your model $\mathcal{M}$ as defined above and which one does not? Of the two answers below, circle the correct one.

Answer 1:	Answer 2:
φ_1 holds in $\mathcal{M}$, and φ_2 does not hold in $\mathcal{M}$	φ_1 does not hold in $\mathcal{M}$, and φ_2 holds in $\mathcal{M}$

Student id: ______________

Question 2: Alloy (38 points total)

Consider the following partial Alloy specification *TA* of a task assignment system. *TA* uses the `ordering` library module to impose a total order on the elements of the signature `Position`. It then uses these positions to order tasks. Recall that the `ordering` module has several predicates and functions built in, including `lt`, `lte`, `first`, and `min`.

```
module TA
open util/ordering[Position]

fact {
   all p:Position | one pos.p    // fact F1
   all t1,t2:Task | t1->t2 in dependsOn => lt[t1.pos,t2.pos]    // fact F2
}
```

```
sig Position {}
sig Worker {}
sig Task {
   pos : Position,
   dependsOn : set Task,
   assignedTo : Worker
}
```

a) (4 points) In the space below, draw an instance of the Alloy specification above. Your instance should satisfy all the constraints expressed in the specification and contain at least two tasks. In your instance, clearly label every object with the signature (i.e., type) and every link (i.e., edge) with the relation (i.e., attribute) they belong to.

b) (2 points) What is the smallest scope in which the Alloy analyzer would be able to produce your instance?

Student id: ____________________

Question 2: Alloy (continued)

For your convenience, the Alloy specification *TA* from the previous page is repeated here.

```
module TA
open util/ordering[Position]

fact {
   all p:Position | one pos.p    // fact F1
   all t1,t2:Task | t1->t2 in dependsOn => lt[t1.pos,t2.pos]    // fact F2
}
```

```
sig Position {}
sig Worker {}
sig Task {
   pos : Position,
   dependsOn : set Task,
   assignedTo : Worker
}
```

c) (4 points) Complete the function definition below such that the function returns exactly the set of all tasks that are assigned to worker `w`.

```
fun getTasks[w:Worker] : set Task {







}
```

d) (4 points) We say that a task `t` is the *first task* if and only if `t` has the smallest position. We say that worker `w` is the *first worker* if and only if `w` is assigned the first task. Complete the function definition below such that it returns the first worker.

```
fun getFirstWorker[] : Worker {







}
```

e) (4 points) We say that task `t2` is *between* tasks `t1` and `t3` if and only if the position of `t1` is less than that of `t2`, and the position of `t2` is less than that of `t3`. Complete the predicate definition below such that it returns true exactly when `t2` is between `t1` and `t3`.

```
pred between[t1,t2,t3 : Task] {
```

Student id: ____________________

Question 2: Alloy (continued)

For your convenience, the Alloy specification *TA* from the previous page is repeated here.

```
module TA
open util/ordering[Position]

fact {
   all p:Position | one pos.p    // fact F1
   all t1,t2:Task | t1->t2 in dependsOn => lt[t1.pos,t2.pos]    // fact F2
}
```

```
sig Position {}
sig Worker {}
sig Task {
   pos : Position,
   dependsOn : set Task,
   assignedTo : Worker
}
```

Express each of the following 3 properties formally as formulas in Alloy. You may use previously defined functions or predicates.

f) (4 points) *"Tasks are assigned evenly to workers"*, i.e., *"The number of tasks assigned to any two workers does not differ by more than 1"*

g) (4 points) *"For any worker* `w` *and any two tasks* `t1` *and* `t2` *assigned to* `w`*, all tasks between* `t1` *and* `t2` *(in the sense of Question 2.e) are also assigned to* `w`*"*

h) (4 points) *"All tasks that do not depend on any other task are assigned to the first worker (in the sense of Question 2.d)"*

Student id: ____________________

Question 2: Alloy (continued)

For your convenience, the Alloy specification *TA* from the previous page is repeated here.

```
module TA
open util/ordering[Position]

fact {
   all p:Position | one pos.p   // fact F1
   all t1,t2:Task | t1->t2 in dependsOn => lt[t1.pos,t2.pos]   // fact F2
}
```

```
sig Position {}
sig Worker {}
sig Task {
   pos : Position,
   dependsOn : set Task,
   assignedTo : Worker
}
```

For each of the following 2 formulas, decide whether or not it is implied by the specification *TA*, i.e., whether or not every instance that satisfies all constraints in *TA*, also satisfies the formula. In the space below the formula, write "Yes", if you think the formula is implied. Write "No", if you think it is not implied. If you write "Yes", also very briefly indicate which part of *TA* cause the formula to be implied. If you write "No", draw a counter example, i.e., an instance that satisfies all constraints in *TA*, but not the formula.

i) (4 pts) `no t:Task | t in t.^dependsOn`

j) (4 pts) `all w:Worker | all t1,t2:Task | (t1 in w.assignedTo && t2 in w.assignedTo) => t1=t2`

Student id: ____________________

Question 3: CTL (18 points total)

Consider the three pairs of non-equivalent formulas φ_i, φ_i' below. For each pair, find a Kripke structure that distinguishes them. More precisely, for each pair, draw a Kripke structure M_i such that one formula holds in M_i, but not the other. **Important:** When drawing M_i, make sure that you clearly indicate (1) the initial state of M_i, (2) which atomic propositions occuring in φ_i and φ_i' hold in which states of M_i, and (3) which of the two formulas holds in M_i. Also, remember that the transition relation of a Kripke structure is total.

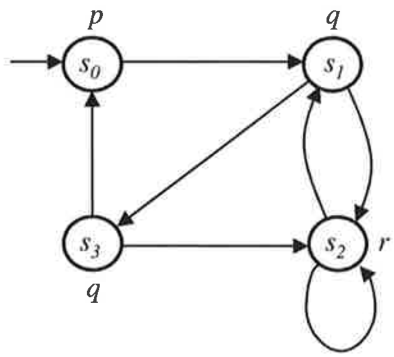
a) (6 points) $\varphi_1 = \mathbf{AG}\ (p \rightarrow \mathbf{AX}\ \neg p)$ and $\varphi_1' = \mathbf{AG}\ \neg p$

b) (6 points) $\varphi_2 = \mathbf{EG}\ p$ and $\varphi_2' = \mathbf{EX}\ \mathbf{EG}\ p$

c) (6 points) $\varphi_3 = \mathbf{EG}\ \mathbf{EF}\ p$ and $\varphi_3' = \mathbf{EG}\ \mathbf{AF}\ p$

Question 4: Model checking (18 points total)

Consider the following graphical representation of a Kripke structure M.



For each of the following six CTL formulas φ decide whether the formula holds in M. If your answer is *"No"*, that is, φ does not hold in M, then give a counter example, that is, a sequence of states corresponding to an execution path in M illustrating the *violation* of φ. Remember that some counter examples are infinite. To show infinite counter examples enclose the sequence of states that are repeated in parentheses. E.g., the sequence $s_0(s_1 s_3 s_2)$ represents an execution that starts with s_0 after which states s_1, s_3 and s_2 are repeated forever in this order.

a) $\mathbf{AX}\ \mathbf{AX}\ \mathbf{AX}\ \mathbf{AX}\ (q \vee r)$

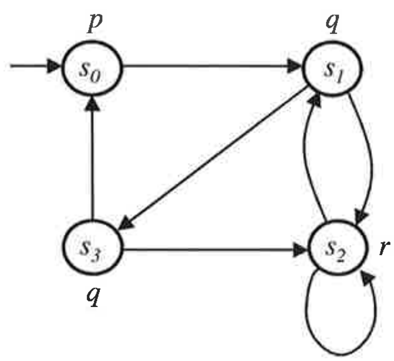
b) $\mathbf{EF}\ \mathbf{EG}\ \neg p$

c) $\mathbf{AG}\ \big(p \vee (\mathbf{EX}\ p) \vee (\mathbf{EX}\ \mathbf{EX}\ p)\big)$

Student id: ______________________

Question 4: Model checking (continued)

For your convenience, the Kripke structure M from the previous page is repeated here.



d) $\mathbf{AG}\ \mathbf{AF}\ (p \vee q)$

e) $\mathbf{A}\big[(q \vee \mathbf{EX}\ q)\ \mathbf{U}\ \mathbf{EG}\ r\big]$

f) $\mathbf{AG}\ \big((\mathbf{EX}\ p) \ \rightarrow\ \mathbf{EX}\ \mathbf{AX}\ q\big)$

Student id: ____________________

Question 5: SMV (40 points total)

Consider the following SMV program.

```
MODULE P1
VAR sig :  {yes,no};
ASSIGN init(sig) := yes;
       next(sig) := case
                      sig=yes :  no;
                      sig=no :  yes;
                    esac;
MODULE P3(s,c)
VAR cnt :  {0,1,2};
ASSIGN init(cnt) := 0;
       next(cnt) := case
                      s=no & c=z & cnt<1 :  cnt+1;
                      s=no & c=z & cnt=2 :  0;
                      TRUE : cnt;
                    esac;
```

```
MODULE P2(s)
VAR choice :  {x,y,z};
ASSIGN init(choice) := x;
       next(choice) := case
                         s=yes :  {y,z};
                         s=no :  x;
                       esac;
MODULE main
VAR p1 :  P1;
    p2 :  P2(p1.sig);
    p3 :  P3(p1.sig,p2.choice);
```

a) (10 points) Draw the Kripke structure M that is defined by the SMV program above. Represent a single state of M by a triple $(v_{sig}, v_{choice}, v_{cnt})$ where v_{sig} is the value of sig in process p1, v_{choice} is the value of choice in process p2, and v_{cnt} is the value of cnt in process p3. For instance, the initial state of M is represented as $(yes, x, 0)$. Your drawing should clearly indicate the initial states of M, the reachable states of M, and the transition relation of M. Hints: You don't need to show the labelling function. M should not have more than 10 states.

Student id: ____________________

Question 5: SMV (continued)

For your convenience, the SMV program from the previous page is repeated here.

```
MODULE P1
VAR sig :  {yes,no};
ASSIGN init(sig) := yes;
       next(sig) := case
                      sig=yes :  no;
                      sig=no :  yes;
                    esac;

MODULE P3(s,c)
VAR cnt :  {0,1,2};
ASSIGN init(cnt) := 0;
       next(cnt) := case
                      s=no & c=z & cnt<1 :  cnt+1;
                      s=no & c=z & cnt=2 :  0;
                      TRUE : cnt;
                    esac;
```

```
MODULE P2(s)
VAR choice :  {x,y,z};
ASSIGN init(choice) := x;
       next(choice) := case
                         s=yes :  {y,z};
                         s=no :  x;
                       esac;

MODULE main
VAR p1 :  P1;
    p2 :  P2(p1.sig);
    p3 :  P3(p1.sig,p2.choice);
```

b) (2 points) In the SMV program above, do the processes execute synchronously or asynchronously? Circle the correct answer.

Synchronously   Asynchronously

c) (2 points) Is the SMV program above deterministic? Circle the correct answer.

Yes   No

d) (6 points) Suppose you want to check that in the SMV program above state $(yes, x, 1)$ is reachable in exactly 3 steps (transitions).

- Write down a CTL formula φ that you could use for this purpose.

  $\varphi =$

- What could you conclude if the SMV program satisfies φ, i.e., if NuSMV does not find a counter example to the formula φ? Circle the correct answer.

State $(yes, x, 1)$ is reachable in exactly 3 steps   State $(yes, x, 1)$ is not reachable in exactly 3 steps

Student id: ____________________

Question 5: SMV (continued)

e) (20 points) Each of the five informally given properties φ_1 through φ_5 refers to the SMV program defined on the previous page. Express each of them formally in CTL. We use `p1.sig` to refer to the value of variable `sig` in process `p1`. Similarly for `p2.choice` and `p3.cnt`. Given a state s, a successor of s is a state that can be reached from s in one transition.

φ_1: *"It is not the case that a state s can be reached such that* `p1.sig` *is equal to* `Yes` *in s and also in at least one successor of s"*

φ_1 in CTL:

φ_2: *"Along all paths, it is always possible to reach a state in which* `p3.cnt` *is equal to 0"*

φ_2 in CTL:

φ_3: *"There exists a path with a state s such that* `p2.choice` *is not equal to* `z` *until s, and s is the beginning of a path along which* `p3.cnt` *is always 2"*

φ_3 in CTL:

φ_4: *"All states that are exactly three transitions away from the initial state are such that* `p3.cnt` *is less than 2 in them"*

φ_4 in CTL:

φ_5: *"Along all paths, it is always the case that if* `p1.sig` *is equal to* `Yes` *in a state, then* `p1.sig` *is equal to* `No` *in all successors of that state"*

φ_5 in CTL:

Student id: ____________________

Scratch sheet: