TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI
VICTORIA
UNIVERSITY OF WELLINGTON

EXAMINATIONS — 2010

END-OF-YEAR

SWEN 102
Introduction to Software Modelling

Time Allowed: 3 Hours

Instructions: There are 180 possible marks on the exam.
Answer all questions in the boxes provided.
Every box requires an answer.
If additional space is required you may use a separate answer booklet.
Some example Alloy code is provided on the last page.
Non-electronic foreign language dictionaries are allowed.
Calculators ARE NOT ALLOWED.
No other reference material is allowed.

Question	Topic	Marks
1.	Use Case Diagrams	30
2.	Object and Class Diagrams I	30
3.	Object and Class Diagrams II	30
4.	Alloy I	30
5.	Alloy II	30
6.	State Machines	30
	Total	180

Student ID: .....

Question 1. Use Case Diagrams [30 marks]

(a) [3 marks] Perform a *textual analysis* on the following description, to find candidate use cases. You should carefully and neatly underline key verb phrases in the text.

The Open University uses an online system to manage and run its courses. Lecturers create courses and assignments, whilst students upload solutions. Tutors are employed to mark assignments, and can submit marks and comments using the system.

To create a new course, the course co-ordinator selects the course code from a list of valid courses. He/she then provides a description of the course, and the login of each tutor. The logins are checked to ensure they exist, and are not enrolled in the course.

To create an assignment for a course, the co-ordinator provides a course name, the start and end dates, and any restrictions on file types that may be submitted. The end date cannot be before the start date, whilst the name cannot already have been used.

Students log in and submit solutions to assignments. If theend date has past, or incorrect file types are used, the submission is rejected.

Tutors log in and mark assignments. Each tutor can only see the courses they are involved in. A tutor does not have to submit marks for every question at once. Instead, they may return at a later time and complete their marks. When all marks are entered, the lecturer is notified by email.

(Question 1 continued on next page)

(Question 1 continued)

(b) [12 marks] Provide **use case descriptions** for the main success sequence of the following **three** use cases.

Create Course

Create Course (Course Co-ordinator) Requires: Logged In	
Select Course Enter Description and Logins	Display Course List Check Logins Create Course Record

Create Assignment

Create Assignment (Course Co-ordinator) Requires: Logged In	
Select Course Enter Name Enter start/end dates Enter Restrictions	Display Course List Check name not already used Check start date before end date Create Assignment Record

Submit Solutions

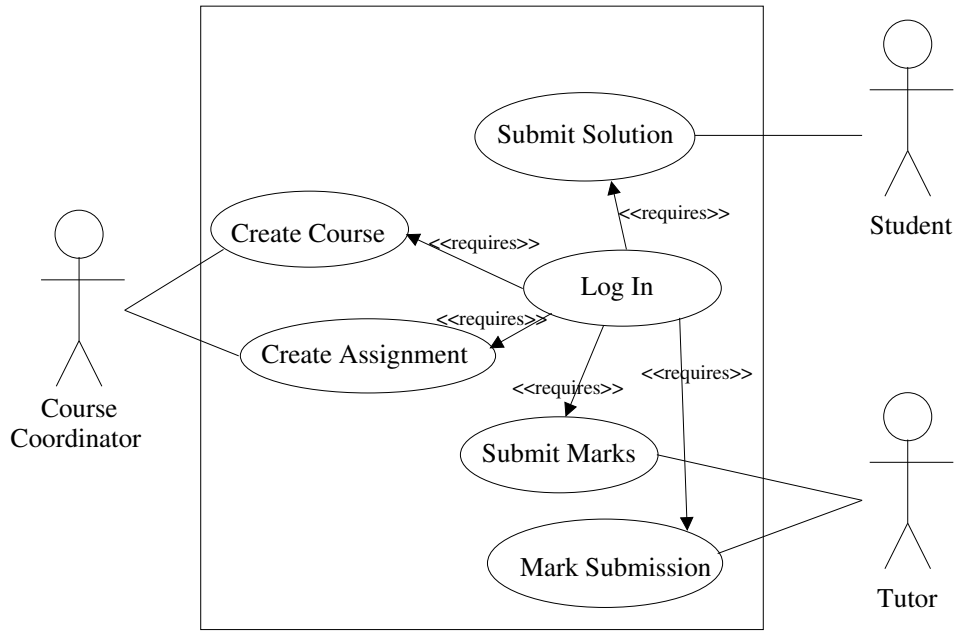
Submit Solutions (Student) Requires: Logged In	
Select Course Submit File	Display Course List Check end date Check file type Record Submission

(Question 1 continued on next page)

Student ID: . . . . . .

(Question 1 continued)

(c) [9 marks] Draw a **use case diagram** showing at least 3 actors and at least 6 use cases for the University system. You may include the use cases given on Page 3.



(Question 1 continued on next page)

(Question 1 continued)

(d) [6 marks] The description text is incomplete. Give **two** questions you would ask users or clients to clarify these requirements.

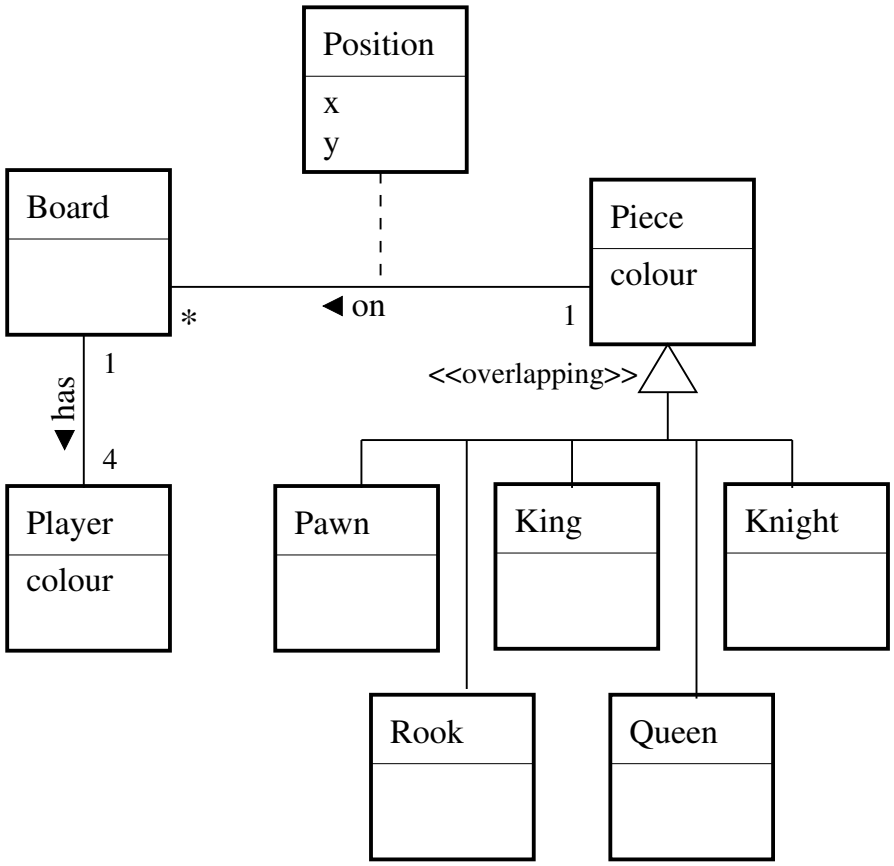
How are course coordinators distinguished from lecturers?

How are login accounts created in the system?

Student ID: .....

Question 2. Object and Class Diagrams II [30 marks]

(a) Consider the following (incomplete) class diagram which models the game of chess, and answer the following questions.



(i) [14 marks] For each question, state whether or not it is correct with respect to the above diagram. If it is not correct, briefly state why.

1. Each board may have one or more pieces on it

   Incorrect. Each board must have exactly one piece on it.

2. Every board has two players

   Incorrect. Every board has four players.

(Question 2 continued on next page)

(Question 2 continued)

3

Pawns and Kings are pieces

Correct.

4

A piece cannot be both a Pawn and a King

Incorrect. Subclasses of piece are listed as *overlapping* and, hence, a piece can be both a Pawn and a King.

5

Every piece is on a board

Incorrect. A piece is on zero or more boards.

6

Every piece on a board is at a given x and y position

Correct.

7

No two pieces on the same board can be at the same position

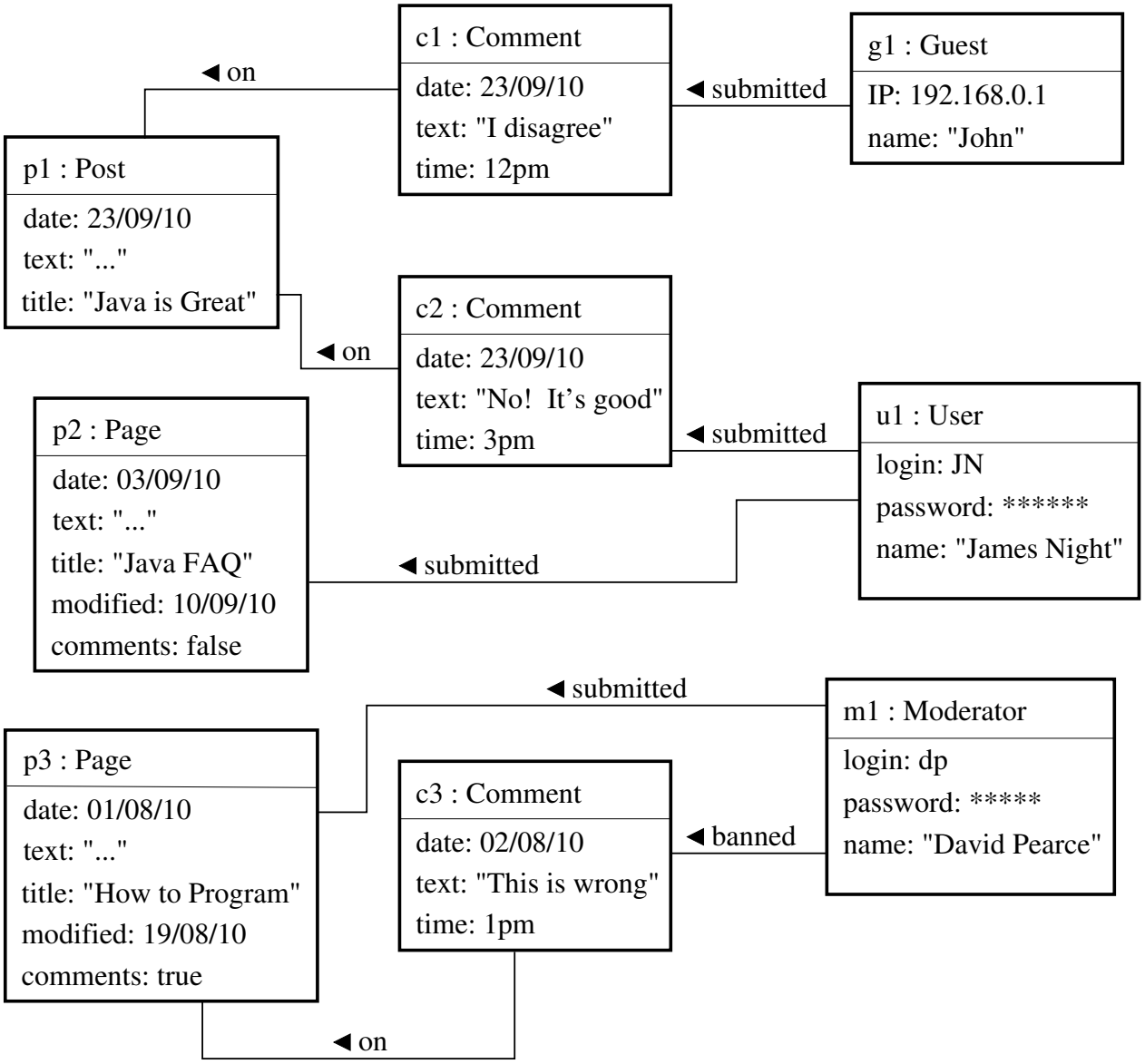
Incorrect. There is no contraint which expresses this.

(Question 2 continued on next page)

Student ID: . . . . . .

(Question 2 continued)

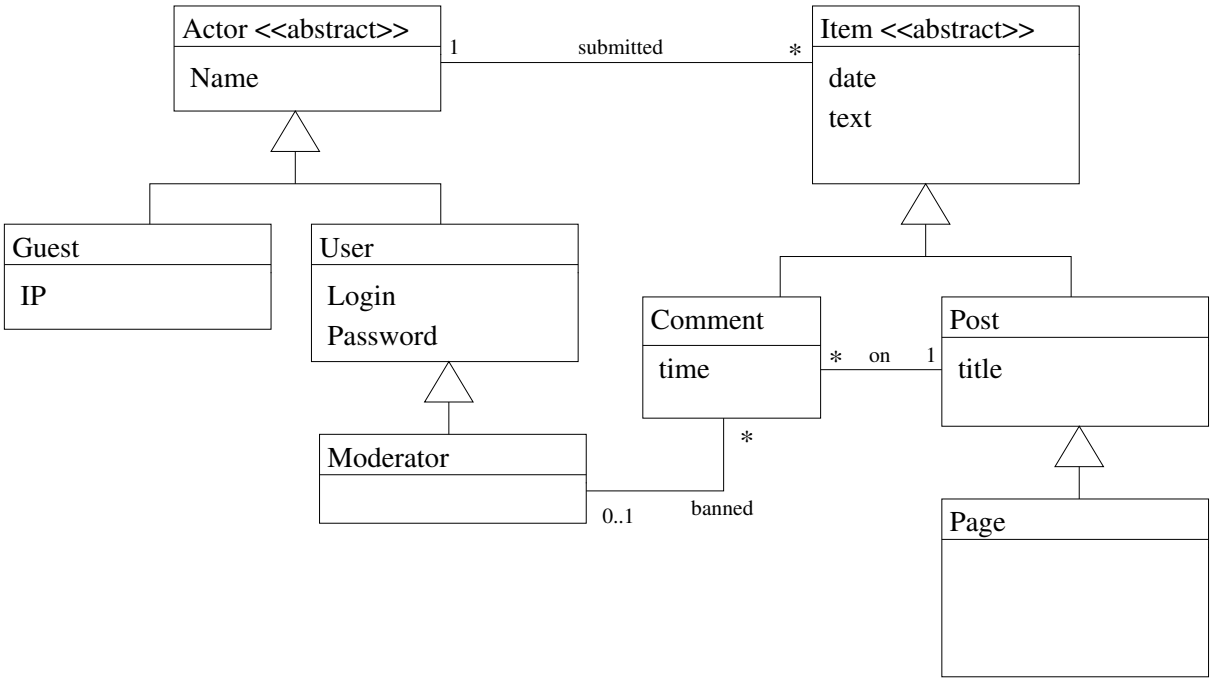
(b) [16 marks] Consider the object diagram below and draw a corresponding class diagram on the facing page.



(Question 2 continued on next page)

Student ID: . . . . . .

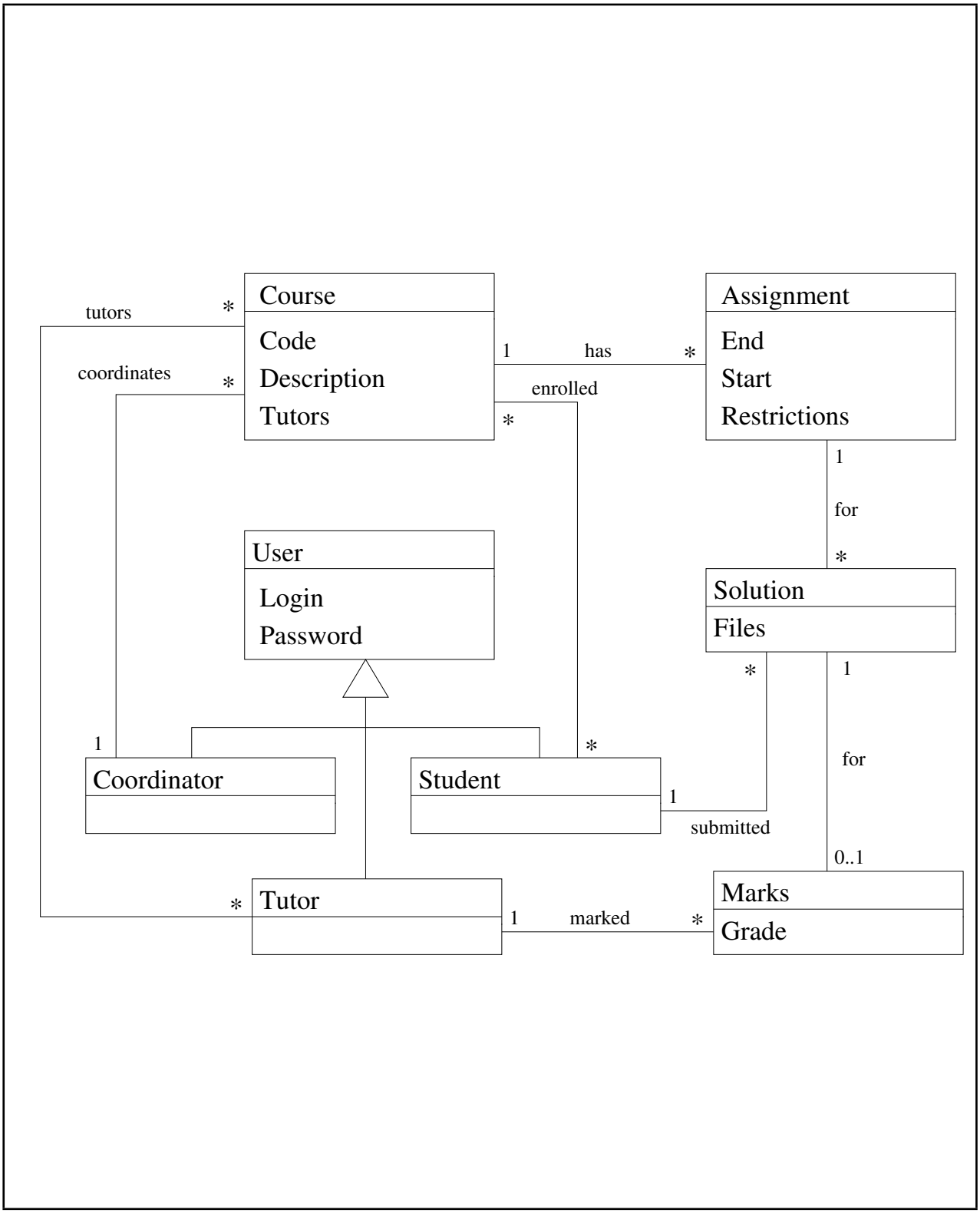
(Question 2 continued)



Student ID: .....

Question 3. Object and Class Diagrams I [30 marks]

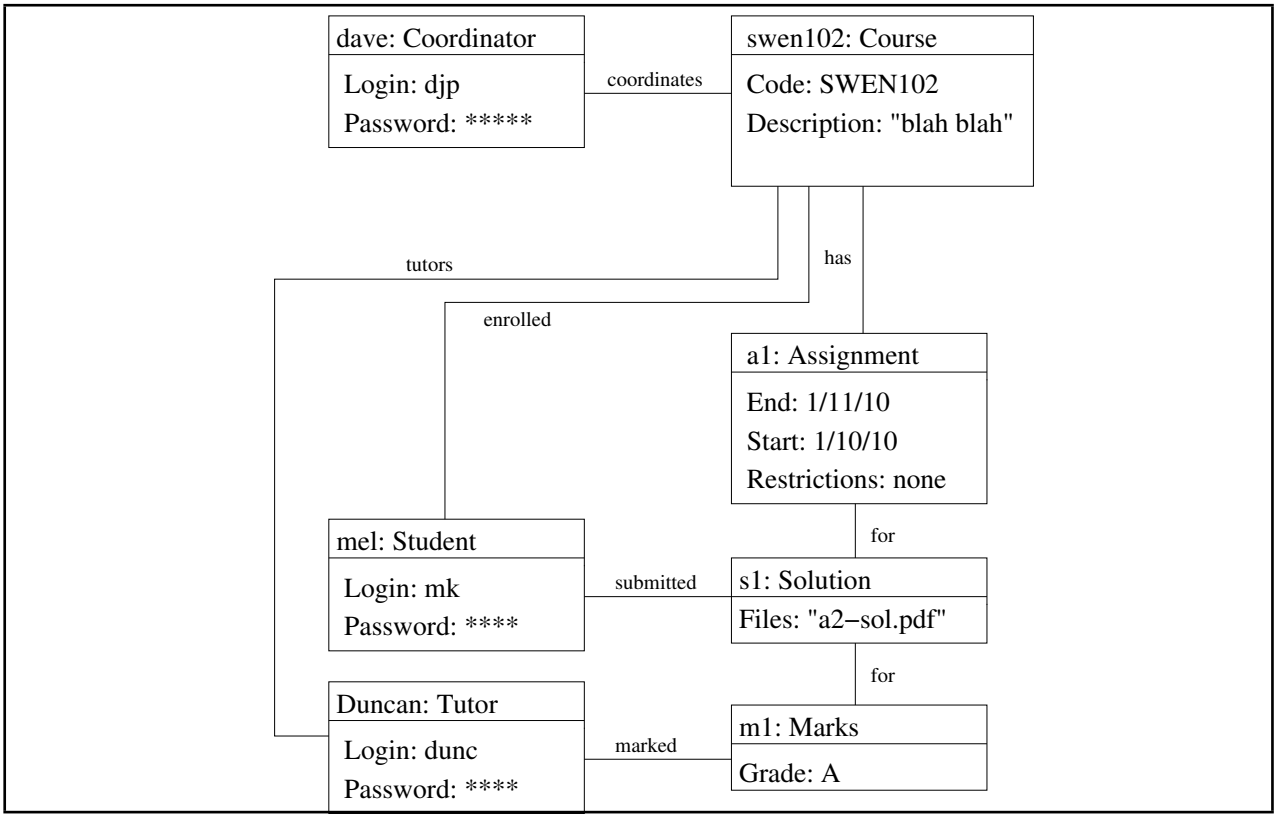
(a) [15 marks] Based on the description from Page 2, draw a *well-designed* **class diagram** to model the University system. This should contain at most 6 classes and 10 attributes and use inheritance and associations where appropriate.



Student ID: .....

(b) [6 marks] In the box below, draw an *object diagram* consistent with your class diagram which captures the following scenario:

> "Dave created Assignment 1 for course SWEN102. Mel submitted her solution, and it was marked by Duncan. Mel got an A for that assignment."



(c) [5 marks] An important requirement of the University system is that it is *secure*. Briefly, discuss why it is difficult to capture this requirement in a class diagram.

Security is not a concern that can be expressed easily in a class diagram. This is because security primarily revolves around restricting what users are allowed to do. A class diagram does not capture what users can do in a system — use cases capture this. Class diagrams capture only information about the structure of data, not how it is accessed or manipulated.

(d) [4 marks] Another important requirement of the University system is that it is *reliable*. Briefly, discuss what this means in the context of the university system.

The university system is an online system. Therefore, one would expect to be able to access the system at any time or day. For example, lectures and/or students may wish to access the system in another time zone. Therefore, it is important that the system should have high availability — that is, it should not suffer from significant downtime (i.e. time when the system is unavailble for use).

Student ID: .....

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

Student ID: .....

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

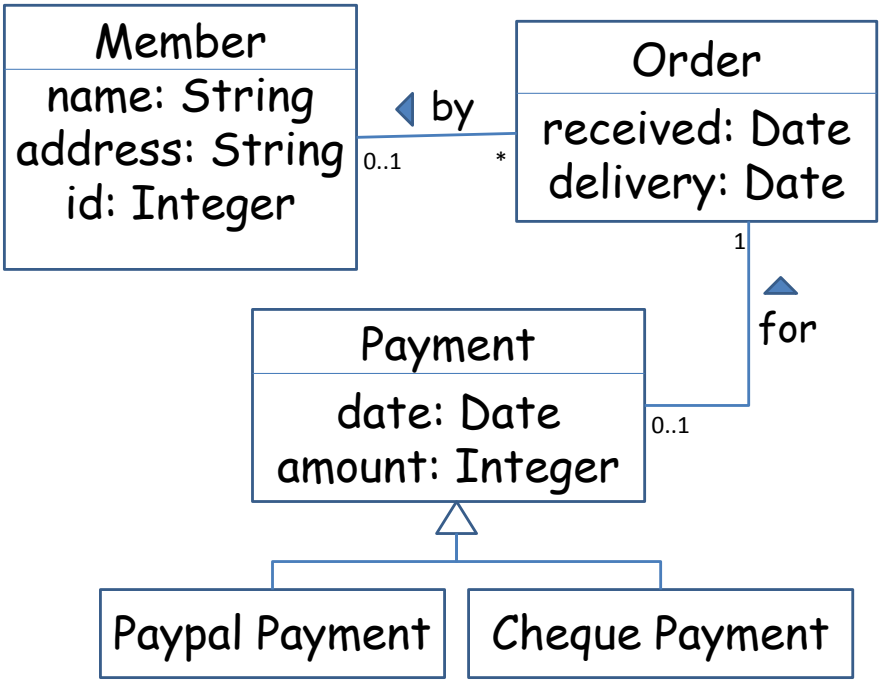
Student ID: .....

Question 4. Alloy I

[30 marks]

Consider the following description of an *on-line flower ordering system*, which is made up of some *text* and a *class diagram*:

> The on-line flower ordering system records orders for flowers and other small presents. When an order is made, the date the order is received is recorded. The customer has to specify delivery date and payment method. However, payment date must be before the delivery date. Supported payment options are: Paypal and Cheque.
>
> Customers can register as a member, so that the system can store their name, address, and order history. All members are associated with a positive number that uniquely identifies them. Members who have ordered at least 10 times might get the option to place an order for free. For such an order, no payment is required.



(Question 4 continued on next page)

Student ID: . . . . . .

(Question 4 continued)

(a) [10 marks] By considering the given text and class diagram, identify (in English) five *candidate invariants*:

1) Payment date must be before delivery date.

2) No two customers may have the same id.

3) For every order with no payment, there exists a member who made the payment and has made at least 10 other payments.

4) Received date must be before delivery date.

5) Payment amount must be positive.

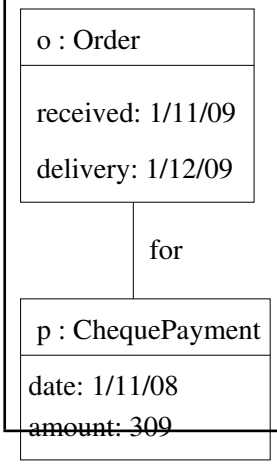
(Question 4 continued on next page)

Student ID: . . . . . .

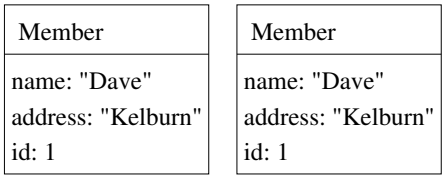
(Question 4 continued)

(b) [10 marks] For each candidate invariant, provide a *counter-example*—an object diagram that is consistent with the class diagram but inconsistent with your invariant.

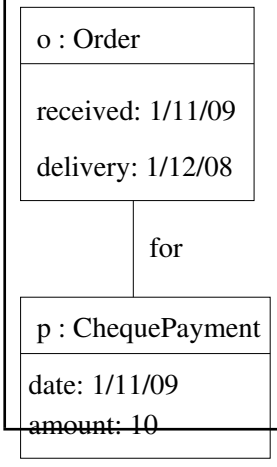
1)



2)



3)

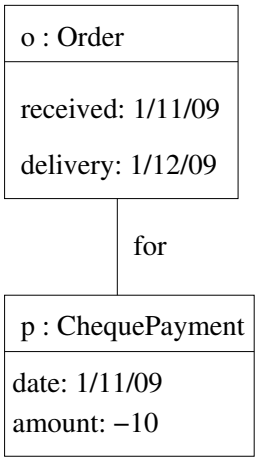


(Question 4 continued on next page)

(Question 4 continued)

4)

5)



(Question 4 continued on next page)

Student ID: .....

(Question 4 continued)

(c) [10 marks] Translate the class diagram on Page 15 into Alloy code. You do *not* need to translate your candidate invariants. Example Alloy code is provided on page 32.

```
sig string {}

sig Member {
name : string,
address : string,
id: Int
}

sig Order {
received: Int,
delibery: Int,
by : lone Member
}{
lone this.~@forOrder
}

abstract sig Payment {
date: Int,
amount: Int,
forOrder : one Order
}

sig PaypalPayment extends Payment {}
sig ChequePayment extends Payment {}
```

Student ID: .....

Question 5. Alloy II

[30 marks]

Consider the following description of a *registry office* system:

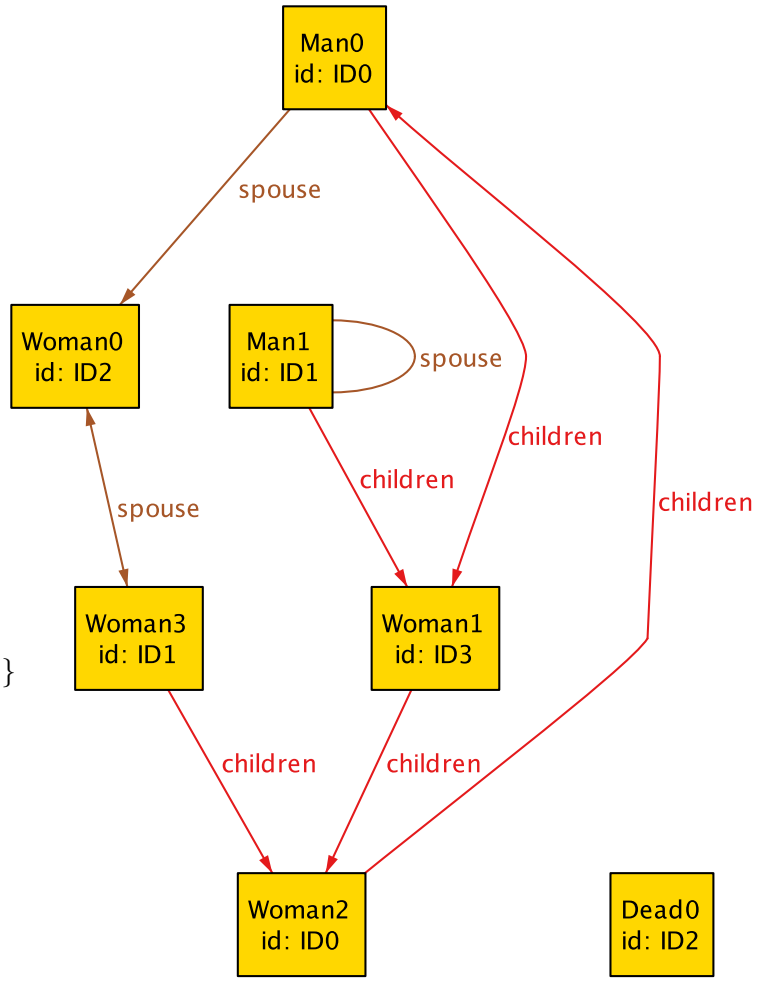
> "A Registry Office records births, deaths, and marriages. For each registered person, the system stores a unique ID, gender, (biological) parents, the current spouse, and whether the person has died. The system enforces some consistency rules to ensure the data is correct: A person cannot be the child of several mothers or several fathers, but a person's parents might be unknown. Furthermore, a person cannot be a descendant (child or child of a child or child of a child of a child, etc.) or spouse of itself. Same sex marriages are allowed but a person cannot have several spouses at once. The spouse of a married person must be married (i.e. must have a spouse) but not to somebody else."

An *incorrect* model of the registry office in Alloy and an *object diagram* of that model is given below:

```
sig ID {}

abstract sig Person {
  id: ID,
  children: set Person,
  spouse: lone Person
}

sig Dead extends Person {}
sig Man, Woman extends Person {}
```



The above object diagram was generated using the command "`run {} for 6`" from the Alloy model.

(Question 5 continued on next page)

Student ID: .....

(Question 5 continued)

(a) [5 marks] Evaluate each of the following Alloy expressions on the object diagram given on the previous page:

```
Man$0.spouse

{Woman$0}
```

```
Woman$1.~children

{Man$0, Man$1}
```

```
spouse.spouse

{Man$0→ Woman$3, Woman$0→ Woman$0, Woman$3→ Woman$3, Man$1→ Man$1}
```

(Question 5 continued on next page)

Student ID: .....

(Question 5 continued)

(b) [10 marks] Circle and number five distinct ways in which the object diagram given on the previous page is *inconsistent* with the description of the registry office. For each, write a brief (i.e. one line) **description of the problem** in the corresponding box below.

1)
Two people have the same ID.

2)
Woman$1 has two fathers.

3)
Man$1 is a spouse of himself.

4)
Man$0 is a spouse of Woman$0 but not vice-versa.

5)
Man$0 is a descendant of himself.

(Question 5 continued on next page)

Student ID: .....

(Question 5 continued)

(c) [15 marks] For each problem identified in **(b)**, indicate what changes you would make to the Alloy model to fix it and **give Alloy code to illustrate**.

1)

Add: `fact { all disj p1, p2 :  Person | p1.id != p2.id}`

2) Add `Person` constraints:

```
no disj p1,p2 :  spouse | p1 in Man and p2 in Man
no disj p1,p2 :  spouse | p1 in Woman and p2 in Woman
```

3) Add `Person` constraint:

```
this not in spouse
```

4) Add `Person` constraint:

```
all p :  spouse | this in p.@spouse
```

5) Add `Person` constraint:

```
this not in ^children
```

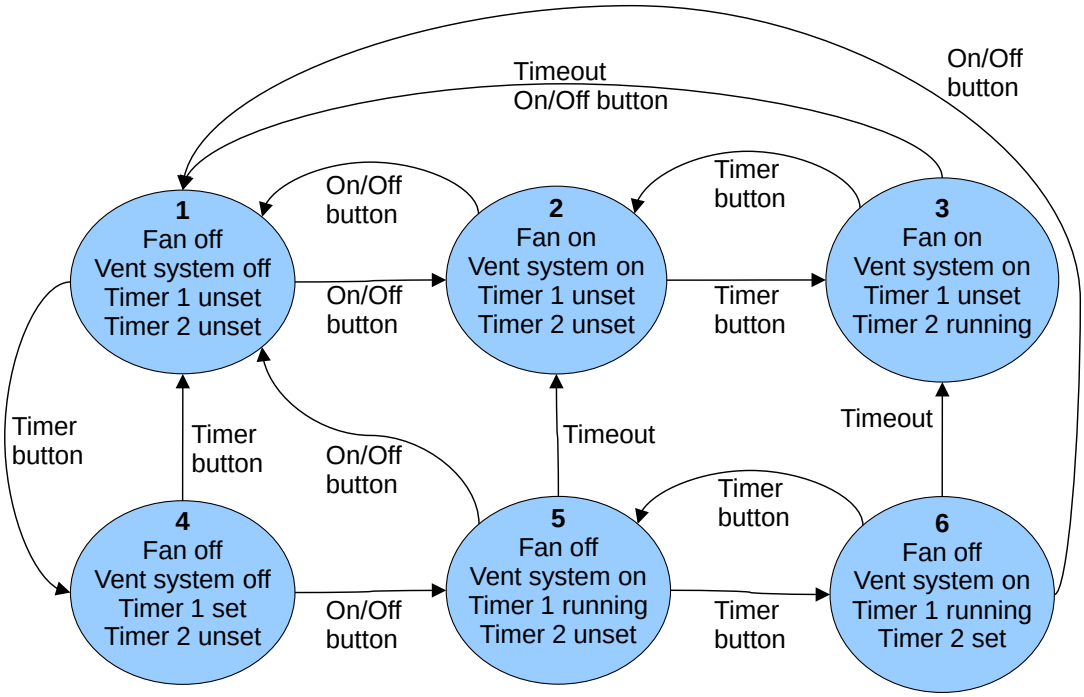
Student ID: .....

Question 6. State Machines [30 marks]

Consider the following description of a simple *ventilation system* with timers:

> "The ventilation system can be set to start and/or stop after predefined times. Our simple model does not support changing the time lengths, and has only two buttons: On/Off and Timer. The ventilation system can be switched on and off using the *On/Off* button. If the ventilation system is switched off, pressing the Timer button selects when the fan should switch on. If the ventilation system is switched on, pressing the Timer button selects when the fan should switch off."

A *state machine diagram* for the ventilation system has been provided:



(Question 6 continued on next page)

(Question 6 continued)

(a) For each of the following statements, indicate whether it is a true or false statement based on the state machine diagram.

(i) [2 marks] The fan is on whenever the ventilation system is switched on.

False

(ii) [2 marks] When the fan is on, the ventilation system is switched on.

True

(iii) [2 marks] If the fan is running, it can always be switched off by pressing the *On/Off* button once.

True

(iv) [2 marks] A *Timeout* either switches the fan on or off.

True

(v) [2 marks] Once the ventilation system is switched on, the *Timer* button only influences timer 2.

True

(Question 6 continued on next page)

(Question 6 continued)

(b) Provide a suitable *execution trace* for the following scenarios. Your execution trace may start from whichever state you chose.

(i) [2 marks]

"John notices the fan working. He presses the Timer *button once. Later on, the ventilation system switches the fan off automatically."*

$2 \rightarrow 3 \rightarrow 1$

(ii) [2 marks]

"Jane programs the system in the morning so that it switches automatically on and off during the day."

$1 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 3 \rightarrow 1$

(iii) [2 marks]

"Sue notices that the fan is off. She presses the On/Off *button once but the fan doesn't switch on. Later on, she notices that the fan is on."*

$4 \rightarrow 5 \rightarrow 2$

(Question 6 continued on next page)

Student ID: .....

(Question 6 continued)

(c) Consider the following *incomplete* Alloy model of the ventilation system:

```
enum OnOff { On, Off }

sig MachineState {
  fan: OnOff,
  ventSystem: OnOff,
  onTimer: Int,
  offTimer: Int
}{
  fan = On iff (ventSystem = On and onTimer = 0)
  0 <= onTimer && onTimer <= 7
  0 <= offTimer && offTimer <= 7
}

pred init(s: MachineState) {
  s.ventSystem = Off
  s.onTimer = 0
  s.offTimer = 0
}

pred onOffButton(s1, s2: MachineState) {
  s1.ventSystem = On iff s2.ventSystem = Off
  s2.ventSystem = Off implies (s2.onTimer = 0 and s2.offTimer = 0)
  s2.ventSystem = On implies (s1.onTimer = s2.onTimer and s1.offTimer = s2.offTimer)
}

pred nextTimerState(s1, s2: Int) {
  (s1 < 7 and s2 = s1 + 1) or (s1 = 7 and s2 = 0)
}

pred timerButton(s1, s2: MachineState) {
  s1.ventSystem = s2.ventSystem
  s1.fan = s2.fan
  nextTimerState[s1.onTimer, s2.onTimer] or nextTimerState[s1.offTimer, s2.offTimer]
  s1.onTimer = s2.onTimer or s1.offTimer = s2.offTimer
  s1.ventSystem = Off iff s1.offTimer = s2.offTimer
}

pred transition(s1, s2: MachineState) {
  onOffButton[s1,s2] or timerButton[s1,s2]
}

sig ExecutionTrace { states: seq MachineState }{
  init[states[0]] and all i: states.inds | i > 0 implies transition[states[i−1], states[i]]
}
```

(Question 6 continued on next page)

Student ID: .....

(Question 6 continued)

(i) [2 marks] Give an instance of MachineState which corresponds to state two (**2**) from the diagram on page 24.

fan: On
ventSystem: On
onTimer: 0
offTimer: 0

(ii) [2 marks] What do you know about the fan in the first state of an `ExecutionTrace`?.

It is Off.

(iii) [2 marks] Give an instance of ExecutionTrace which corresponds to the execution trace "$1 \rightarrow 4 \rightarrow 5$" from the diagram on page 24.

fan: Off		fan: Off		fan: Off
ventSystem: Off	$\rightarrow$	ventSystem: Off	$\rightarrow$	ventSystem: On
onTimer: 0		onTimer: 1		onTimer: 1
offTimer: 0		offTimer: 0		offTimer: 0

(iv) [2 marks] What functionality is provided by the Alloy model but was not in the state machine diagram?

The Alloy model indicates that the timer runs for a specific number of steps before reseting

(Question 6 continued on next page)

Student ID: .....

(Question 6 continued)

(v) [4 marks] Complete the following predicate to model the "Timeout" transition from the diagram on page 24.

```
// s1 is before state, s2 is after state
pred Timeout(s1, s2 : MachineState) {
  (s1.onTimer > 0 && s2.onTimer = 0 && s1.ventSystem = s2.ventSystem && s2.fan = On)
  or (s1.onTimer = 0 && s1.offTimer > 0 && s2.offTimer = 0 && s2.fan = Off && s2.ventSystem = Off)



}
```

(vi) [2 marks] Give an Alloy constraint that ensures every MachineState is always part of an ExecutionTrace.

```
fact { all s : MachineState | one e : ExecutionTrace | s in e.states.elems}
```

****

Student ID: .....

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

Student ID: .....

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

Student ID: .....

(This page may be detached)

Appendix

Example Alloy code for the Monopoly board is given here for reference.

```
sig Name, Colour {}

abstract sig Building {}{
  some buildings.this
}
sig House extends Building {}
sig Hotel extends Building {}

fact {
  // There can be no more than 32 houses and 12 hotels at one time
  // For simplicity, we lower these values since they're too big for Alloy!
  #House <= 7
  #Hotel <= 2
}

sig Player {}

sig Property {
  name: Name,
  owner: lone Player,
  buildings: set Building,
  colourGroup : one ColourGroup,
}{
 // Up to four houses or one hotel in buildings
 #buildings <= 4
 some h : Hotel | h in buildings
 all h : Hotel | h in buildings implies #buildings = 1

 // built properties must have an owner
 some buildings implies one owner
}

sig Mortgaged in Property {}{ no buildings }

sig ColourGroup {
 colour: Colour,
}{
 // Every ColourGroup has a unique colour
 all cg : ColourGroup | cg.@colour = colour implies cg = this
```

Student ID: .....

```
 // ColourGroup Arity
 #~colourGroup >= 2 && #~colourGroup <= 3

 // You must own a whole ColourGroup in order to build
 all p : this.~colourGroup | (some p.buildings) implies
   (all p' : this.~colourGroup | p.owner = p'.owner)

 // buildings must be equally distributed
 all disj p, p' : this.~colourGroup, h: Hotel |
   (h in p.buildings) implies ((some h' : Hotel | h' in p'.buildings) ||
                               #p'.buildings = 4)

 all disj p, p' : this.~colourGroup |
   (no h : Hotel | h in p.buildings) implies (
   #p.buildings = #p'.buildings ||
   #p.buildings = #p'.buildings + 1 ||
   #p.buildings = #p'.buildings - 1 ||
   (#p.buildings = 4 && some h : Hotel | h in p'.buildings))
}

run {} for 4 but 2 ColourGroup, 8 Building, 6 int, exactly 1 Hotel
```