Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano

Prof. Elisabetta Di Nitto, Matteo Rossi

20133 Milano (Italia)
Piazza Leonardo da Vinci, 32
Tel. (39) 02-2399.3400
Fax (39) 02-2399.3411

Software Engineering II

January 24th, 2018

Last Name
First Name
Id number (Matricola)

Note

1. The exam is not valid if you don't fill in the above data.
2. Write your answers on these pages. Extra sheets will be ignored. You may use a pencil.
3. Incomprehensible hand-writing is equivalent to not providing an answer.
4. The use of any electronic apparatus (computer, cell phone, camera, etc.) is strictly forbidden.
5. You cannot keep a copy of the exam when you leave the room.
6. The exam is composed of three exercises. Read carefully all points in the text!

Question 1 Alloy (7 points)
NB: Please read the whole exercise text (including the remarks) before starting to work at your solution.

Describe through an Alloy model the setup and main operations of the game of bingo (the British version of "tombola").
When a game of bingo is played, a coordinator draws numbers from a bag and places them on a board, to keep track of the fact that they have been drawn (so they cannot be drawn again). Each participant is given one or more tickets consisting of 15 different numbers, arranged along 3 rows (5 numbers per row), as in the following example:

5				49		63	75	80
		28	34		52	66	77	
6	11				59	69		82

The game stops when all 15 numbers appearing on a ticket have been drawn. This is called "bingo" and the owner of the ticket wins a prize. During the game, other prizes are assigned when various combinations of the numbers in a ticket are drawn. For simplicity, we consider only the case "5-in-a-row" that occurs when a row in a ticket is completed (i.e., all its 5 numbers have been drawn).

You are required to describe:

- A game setup (tickets, numbers, numbers drawn, and game coordinator), with all relevant constraints.
- Two predicates that, given a game and a ticket in that game, return true if the ticket wins 5-in-a-row or bingo, respectively.
- The operation of drawing a number, whose effect is, given the drawn number, to update the status of the game to keep track of the fact that the number has been drawn; as already mentioned, it is not possible to draw an already drawn number.

Remarks:

- The set of numbers that can be extracted is typically in the 1-90 range; however, this is of no particular interest for this exercise, you can consider the set to be any.
- The ordering of the numbers in rows and tickets is not important in this exercise.
- We are not interested in modeling participants (e.g., who is the person who wins the game), only the tickets and numbers extracted.

Solution 1

```
sig Number{}

sig RowId{}{ #RowId = 3 }


sig Ticket{
  numbers : RowId -> some Number
}{
  //There are exactly 3 different rows
  #(numbers.Number) = 3

  //There are exactly 5 different numbers per row
```

```
  all r : RowId | #(numbers[r]) = 5

  //All numbers in a ticket are different
  #(RowId.numbers) = 15
}

sig Coordinator{
  tickets : some Ticket,
  drawn : set Number
}


pred win5InARow[c: Coordinator, t: Ticket] {
  some r: RowId | t in c.tickets and r in t.(numbers.Number) and
                  t.numbers[r] in c.drawn
}

pred bingo[c: Coordinator, t: Ticket] {
  t in c.tickets and
  all r: RowId | r in t.(numbers.Number) and t.numbers[r] in c.drawn
}

pred draw [g, g' : Coordinator, num : Number]{
  //precondition
  not num in g.drawn
  //postcondition
  g'.tickets = g.tickets
  g'.drawn = g.drawn + num
}

pred show{}
run show for 20 but 1 Coordinator, 2 Ticket
run draw for 20 but 2 Coordinator, 3 Ticket
run win5InARow for 20 but 2 Coordinator, 3 Ticket
run bingo for 20 but 2 Coordinator, 3 Ticket
```

Solution 2

```
sig Number {}

sig Row {
  numbers: set Number
}{#numbers = 5}


sig Ticket {
  rows: set Row
}{#rows = 3}


fact noNumberDuplicatedInTicket {
  no t: Ticket | some disj r1, r2: Row |
    r1 in t.rows and r2 in t.rows and
    ((r1.numbers) & (r2.numbers) != none)
}
```

```
sig Coordinator {
  drawnNumbers: set Number,
  tickets: set Ticket
}

pred win5InARow[c:Coordinator, t: Ticket] {
  some r: Row | t in c.tickets and r in t.rows and
                r.numbers in c.drawnNumbers
}

pred bingo[c: Coordinator, t: Ticket] {
  t in c.tickets and
  all r: Row | r in t.rows implies r.numbers in c.drawnNumbers
}

pred draw[c, c': Coordinator, num: Number] {
  //precondition
  not num in c.drawnNumbers

  //postcondition
  c'.tickets = c.tickets
  c'.drawnNumbers = c.drawnNumbers+num
}

pred show{}
run show for 20 but 1 Coordinator, 2 Ticket
run draw for 20 but 2 Coordinator, 3 Ticket
run win5InARow for 20 but 2 Coordinator, 3 Ticket
run bingo for 20 but 2 Coordinator, 3 Ticket
```

Question 2 Requirements and Function Points (7 points)

Consider the following scenario:
The company *TravelSpaces* decides to help tourists visiting a city in finding places that can keep their luggage for some time. The company establishes agreements with small shops in various areas of the city and acts as a mediator between these shops and the tourists that need to leave their luggage in a safe place. To this end, the company wants to build a system, called *LuggageKeeper*, that offers tourists the possibility to: look for luggage keepers in a certain area; reserve a place for the luggage in the selected place; pay for the service when they are at the luggage keeper; and, optionally, rate the luggage keeper at the end of the service. Payment occurs by credit card. *LuggageKeeper* relies on the service offered by an external company that, upon request by *LuggageKeeper*, takes care of all payment transactions.
Also, the company allows small shops to: i) register themselves as luggage keepers to the platform, by entering information about their location, the number of suitcases they can handle, and their opening time; ii) visualize the reservations made by tourists for their luggage space; and iii) confirm the arrival of a customer and the number of suitcases he/she leaves.

Q1: Given the system described above and referring to the Jackson-Zave distinction between the world and the machine, identify:

- At least two world phenomena that are not shared with the machine.
- At least six shared phenomena controlled by the world.
- At least three shared phenomena controlled by the machine.
- At least one machine phenomenon not shared with the world.

Please use the table below to answer this question (the number of rows does not necessarily correspond to the number of phenomena you can identify for this case).

Phenomena	**Shared nor not**	**Who controls them**	**Short explanation (if the name you assign to the phenomenon is not self-explanatory or you need to specify some conditions on the phenomenon)**

Q2. Referring to the Function Points approach, Identify the function types for the *LuggageKeeper* system, focusing exclusively on **Internal Logic Files** and **External Inputs**.
Explain how you would classify each of the identified function types in terms of their complexity (i.e., simple, medium, or complex). Provide an explanation for your classification.

Solution

Q1

Phenomena	**Shared nor not**	**Who controls them**	**Short explanation (if the name you assign to the phenomenon is not self-explanatory or you need to specify some conditions on the phenomenon)**
Tourist visits city C	N	World	
TravelSpace establishes an agreement with a shop	N	World	
Tourist has luggage	N	World	
Tourist looks for a luggage keeper through *LuggageKeeper*	Y	World	

Tourist reserves a place for his/her luggage	Y	World	
LuggageKeeper confirms/refuse a reservation	Y	Machine	Reservation can be refused if there is not enough space in the selected shop for the amount of luggage to be stored, or the length of the keeping service would not be compatible with the opening time of the shop
LuggageKeeper provides the shop address	Y	Machine	
Tourist asks the owner/clerk to leave the luggage	N	World	
The shop owner/clerk confirms the arrival of a customer and the number of suitcases he/she leaves	Y	World	
LuggageKeeper stores the information about the deposited luggage including the service starting time	N	Machine	This starts from the time the shop confirms the arrival of the suitcases
Tourist confirms checkout of luggage	Y	World	
LuggageKeeper computes the price of the service	N	Machine	
LuggageKeeper provides information about the price of the service	Y	Machine	
Tourist provides payment information	Y	World	
Payment transaction is executed	N	World	This happens outside the control of *LuggageKeeper* as it involves the payment service and the tourist
LuggageKeeper contacts the credit card service for managing payment	Y	Machine	This is shared with the world because the credit card service is pre-existing and it is part of the world
Tourist rates the service	Y	World	
Shop owner registers to the platform	Y	World	We assume that only those that have established an agreement with the company (outside the control of the system) can register
Shop owner/clerk views the list of reservations for the shop	Y	World	
LuggageKeeper alerts shop owner/clerk that a new reservation has been issued	Y	Machine	

Q2

Internal Logic Files

```
Shop (ID, name, location, max number of suitcases, opening hours)
Reservation (id of tourist, number of suitcases, estimated arrival, shopID)
OngoingKeeping(id of tourist, number of suitcases, starting time, shopID)
Invoice(Id of tourist, shopID, date, duration of keeping,
        number of suitcases, final price)
```

All identified ILFs except `Invoice` appear to be related at least to another ILF. The amount of data per ILF depends on the size of *TravelSpace* business, i.e., the number of cities and shops connected to *TravelSpaces*. Given the specific case under consideration, it is reasonable to assume that each city can have its own installation of the *LuggageKeeper* platform, in fact, tourists will look for luggage keepers in a specific city. This way, the dataset per database can be kept small. Based on the above line of reasoning, we may assign a medium or simple complexity to all ILF except invoice, to which we assign a medium complexity.

External inputs
The following ones are those associated with the tourist:

- Make reservation
- Checkout
- Pay
- Rate the service

Note that “Look for a luggage keeper” can be classified as an external inquiry.

The following ones are those associated with the shop:

- Register to the platform
- Confirm deposit

Note that “Check the pending reservations” can be classified as an external inquiry.

All external inputs are simple ones as they require the usage of at most two ILF.

Question 3 Testing (5 points)
Consider the following piece of C code:

```
typedef int Table[3][5];
typedef enum {false, true} bool;

1  bool check3 (Table t, int nums[], int nnums){
2    int n_found;
3    int i = 0;
4    int j;
5    int k = 0;
6    bool found;
7    while (i < 3){
8      j = 0;
9      while (k < 5){
10       k = 0;
11       found = false;
12       while (k < nnums){
13         if (t[i][j] == nums[k])
14           found = true;
15         k++;
16       }
17       if (found == true)
18         n_found = n_found + 1;
19       j++;
20     }
21     i++;
22   }
23   return (n_found == 3);
}
```

1. Define the def-use pairs for `check3` focusing on the variables `n_found`, `i`, `j`, `k`, `found` by filling out the following table.

Variable	**Def-use pairs**
`n_found`	
`i`	
`j`	
`k`	
`found`	

2. Provide one or more potential problems a typical verification effort using data flow analysis may highlight in `check3`.

Solution

1.

Variable	Def-use pairs
`n_found`	**<_, 18>** <18, 18> <18, 23>
`i`	<3, 7> <3, 13> <3, 21> <21, 7> <21, 13> <21, 21>
`j`	<8, 13> <8, 19> <19, 13> <19, 19>
`k`	<5, 9> <10,9> <10, 12> <10, 13> <10, 15> <15, 9> <15, 12> <15, 13> <15, 15>
`found`	<11, 17> <14, 17>

2.
Data flow analysis highlights that `n_found` is used before being defined.

Notice that the function has another problem (on line 9 `k` is used instead of `j`), but this is not necessarily detected by data-flow analysis, as the variable is still used after being defined. At best, we can detect that there is an unusual pattern of usage for variable `k` (`k` is an index, but the definition at line 5 is used only once).