Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano

Prof. Matteo Camilli, Elisabetta Di Nitto, and Matteo Rossi

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

Software Engineering 2 – Written Exam 1 (WE1)

June 26, 2023

Last name, first name and Id number (Matricola)

Number of paper sheets you are submitting as part of the exam

Notes

A. Remember to write your name and Id number (matricola) on each piece of paper that you hand in.

B. You may use a pencil.

C. Unreadable handwriting is equivalent to not providing an answer.

D. The use of any electronic apparatus (computer, cell phone, camera, etc.) is strictly forbidden, except for an ebook reader or a plain calculator.

E. The exam is composed of three exercises. Read carefully all points in the text.

F. Total available time for WE1: 1h and 30 mins

Question 1 – Alloy (8 points)
Consider an application that manages recipes that are of interest to users and provides suggestions when requested.

Alloy_1 (4 points) Define suitable signatures, constraints and facts to describe the following phenomena:

- Recipes are characterized by a set of ingredients, and by the type of cuisine (Italian, Indian, etc.).
- Users have ingredients at their disposal.
- Users have favorite types of cuisine.
- Users maintain a list of favorite recipes.
- Users are provided with suggested recipes (which cannot be recipes that the user already favors).
- Users can be suggested only recipes that include at least 1 ingredient that is already at the user's disposal.

Alloy_2 (2 points) Define a suitable predicate specifying the behavior of a procedure *missingIngredients* that, given a user and a recipe that has been suggested to the user, produces the list of ingredients that the user is missing.

Alloy_3 (2 points) Define a suitable predicate specifying the behavior of a procedure *suggestRecipe* that, given a user and a set of possible recipes, produces a subset of the input recipes that can be suggested to the user. The produced subset should be non-empty if in the input set there is at least one recipe that can be suggested to the user.

Solution

Alloy_1

```
sig Ingredient{}

abstract sig Cuisine {}
one sig Italian extends Cuisine {}
one sig Indian extends Cuisine {}
one sig French extends Cuisine {}
one sig Japanese extends Cuisine {}
// the list of types of cuisine is typically finite (it is an enumeration)
// this list should be completed with the various possibilities

sig Recipe {
  ingredients : some Ingredient,
  cuisine: Cuisine
}

sig User {
  favoriteCuisine : set Cuisine,
  favoriteRecipes : set Recipe,
  availableIngredients : set Ingredient,
  suggestedRecipes : set Recipe
}{
  all sr : suggestedRecipes | sr.ingredients & availableIngredients != none
  suggestedRecipes & favoriteRecipes = none
}
```

Alloy_2

```
pred missingIngredients[ u : User, r : Recipe, res : set Ingredient] {
  // pre-condition
  r in u.suggestedRecipes
  // post-condition
  res = r.ingredients - u.availableIngredients
}
```

Alloy_3

```
pred suggestRecipe [u : User, possibleRecipes : some Recipe, res : set Recipe] {
  //postcondition
  u.availableIngredients & possibleRecipes.ingredients != none
  implies
    ( some r : possibleRecipes | not r in u.favoriteRecipes and
                                 r.ingredients & u.availableIngredients != none and
                                 r in res )
    and
    ( all r : res | r in possibleRecipes and
                    not r in u.favoriteRecipes and
```

```
                    r.ingredients & u.availableIngredients != none )

  u.availableIngredients & possibleRecipes.ingredients = none implies res = none

}
```

Question 2 – JEE (4 points)
A new gym called **TechGym** is set to open soon. In addition to a well-equipped weight room, **TechGym** offers to its members access to various courses such as spinning, functional training, and pilates. As a result, they have two types of membership contract types available:

- Basic plan: Provides access to the weight room only.
- Full plan: Provides access to both the weight room and the entire range of courses.

Both the basic and full plans can be subscribed to for a minimum duration of one month.

TechGym has contacted your software house to develop a JEE web-based software system that allows the gym staff to manage the membership of the customers. More specifically, the following features are requested:

(F.1) Login with username and password to access the application.
(F.2) Add new membership upon subscription by entering relevant information such as name, surname, address, contract type, contract ending date. Customers whose contract is active are given a "Member" status. Customers whose contract expires are given a "Subscription Expired" status.
(F.3) Upon contract expiration, the status switches from "Member" to "Subscription Expired". This routine runs each day at midnight.
(F.4) Provides a utility dashboard that:
  (F.4.1) Displays a list of customers with status "Member" whose membership will expire within a given date.
  (F.4.2) View the list of customers with status "Subscription Expired".
  (F.4.3) View the list of customers with status "Member".

Additional notes:

- For the sake of simplicity, suppose a unique staff account is available to the entire staff. The account needs to be used in parallel by different members of the gym staff.
- There is no need to manage payments through the software system since members interact directly with the gym staff for payments.

JEE_1: Identify at least two components of the web-tier and their main responsibility in terms of the list of requirements (F1-F4.3) they satisfy.

JEE_2: Is there any web-tier component whose responsibility is directly related to F.3? Justify your answer.

JEE_3: On the business tier, what is the most appropriate JEE component to implement F.3? Justify your answer.

JEE_4: Identify the type of business tier bean required to implement F.4.1 and specify the most important operation that it implements.

JEE_5: Can F.4.3, in principle, be implemented using the operation developed for F.4.1? Justify your answer.

Solution

Q1:
JSP: login page for gym's staff. (F.1)
JSP: front end for adding a new client. (F.2)

Q2: there are no web-tier components implementing the feature F.3. It is an internal routine that runs periodically into the business tier.

Q3: A simple solution is to use a singleton bean that implements the operation in charge of identifying all newly expired contracts and of updating their status.

Detailed note, not necessarily part of the expected answers: The bean may enter a method that loops the identification of expired contract until the end of its lifecycle. The method may execute proper checks every 24 hours using method `Thread.sleep()`, as in the following code

```
@Singleton
public class CheckBean {
    …
    public void runCheck(final long timer) {
        while(true) {
            Thread.sleep(timer); // sleep, according to timer
                                 // (e.g., 24 hrs -- 8.64e+7 millisec)
            checkExpiredContracts();
        }
    }
...
```

Alternatively, the singleton bean may exploit the JEE timer service to trigger automatic timers as follows:

```
@Singleton
public class CheckBeanWithTimer {
    …
    @Schedule(dayOfMonth ="*", hour="23") // schedule every day at midnight
    public void runCheck() {
            checkExpiredContracts();
    }
}
```

Q4: A stateless bean is enough. It has to implement an operation that, given a date, queries the database and returns a list of customer contracts that are currently in the "Member" state but will expire by the given date.

Q5: If the operation offered by the stateless bean is able to return all active members when the date parameter has a special value, for instance, "null", then we can just use it to implement F.4.3.

Question 3 – Testing (4 points)
Consider the following C function *foo*:

```
0 int foo(int a[], int b, int c) {
1     int r, x = 0;
```

```
2     int y = 0;
3     if (a[b] >= 0) {
4         y = 7;
5     }
6     else {
7         y = 5;
8     }
9     a[y] = c;
10    if (a[b] < 3) {
11        x = 2;
12        if (a[b] <= 0 || c != 0) {
13            y = y - 3;
14        }
15    }
16    r = a[x + y - 2];
17    return r;
18 }
```

Note: you can assume that the array *a*[] has 10 elements and $0 \leq b \leq 9$ holds.

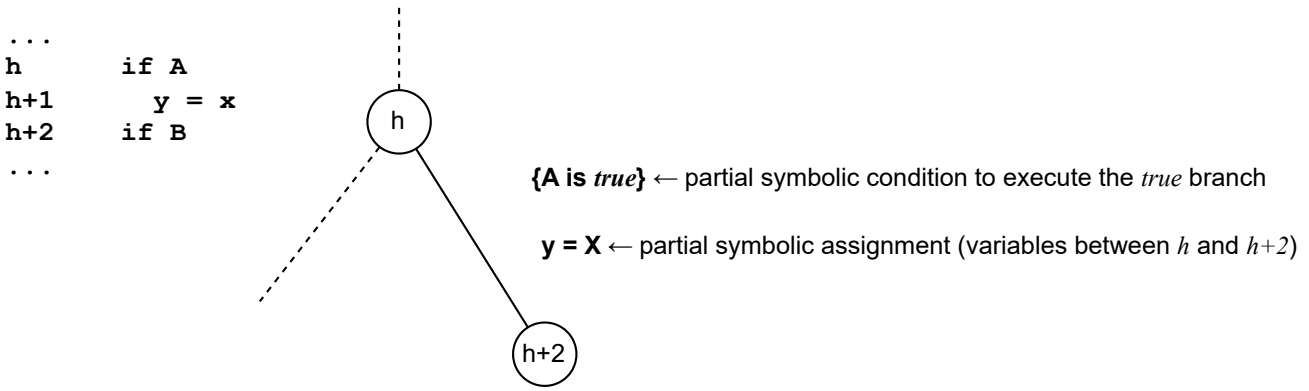
Testing_1

Consider the function *foo* and carry out a symbolic execution for all paths (note that, in this case, the number of paths is finite).

Given that this code fragment does not contain loops, you may be facilitated in your work if you create a binary tree to represent these paths. The nodes of the tree shall include at least the initial symbolic assignment in lines 0-2 (root), the final assignments (leaves), and all the conditional statements (internal nodes). You can annotate the arcs with the conditions and assignments that are relevant for the corresponding part of code.

For each path (i.e., each leaf of the previous tree structure), determine the symbolic condition that ensures the execution of the corresponding path. Then, identify the unfeasible path(s), if any.

Example (tree structure): the following piece of tree represents a portion of the symbolic paths extracted from a program where lines *h* and *h*+2 are conditions, between *h* and *h*+2 there are no conditions, and there is a new assignment for variable *y* (where the symbolic value of variable *x* before line *h*+1 is *X*).

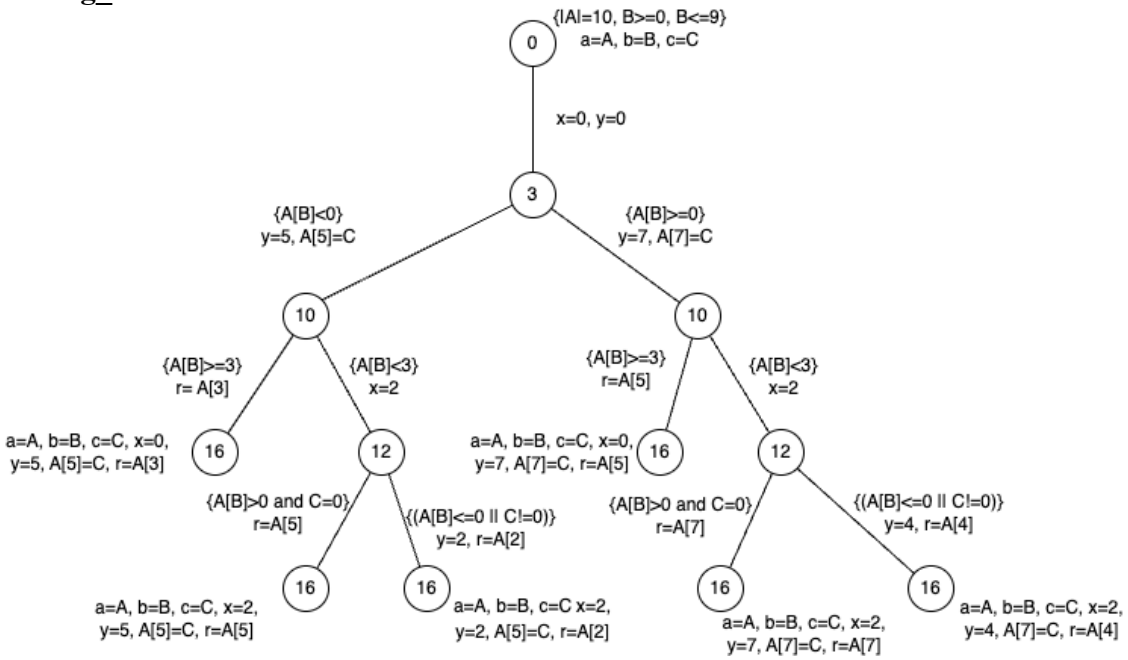


Testing_2

Suppose that we wanted to guarantee that the result of the *foo* function *is not* zero. According to the previous analysis, determine a precondition among the possible ones that ensures the satisfaction of this property.

Solution

Testing_1



There are 6 leaves (from left to right) with the following path conditions:

1. {A[B]<0 and A[B]>=3} path 0, 1, 2, 3, 6, 7, 8, 9, 10, 15, 16, 17 **unfeasible**
2. {A[B]<0 and ~~A[B]<3~~ and A[B]>0 and C=0)} path 0, 1, 2, 3, 6, 7, 8, 9, 10, 11, 12, 14, 15, 16, 17 **unfeasible**
3. {A[B]<0 and ~~A[B]<3~~ and ~~(A[B]<=0 or C!=0)~~} path 0, 1, 2, 3, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17
4. {~~A[B]>=0~~ and A[B]>=3} path 0, 1, 2, 3, 4, 5, 9, 10, 15, 16, 17
5. {~~A[B]>=0~~ and A[B]<3 and A[B]>0 and C=0} path 0, 1, 2, 3, 4, 5, 9, 10, 11, 12, 14, 15, 16, 17
6. {A[B]>=0 and A[B]<3 and (A[B]<=0 or C!=0)}, that is {A[B] = 0 or (0 <= A[B] <3 and C != 0} path 0, 1, 2, 3, 4, 5, 9, 10, 11, 12, 13, 14, 15, 16, 17

Testing_2

Possible preconditions are the following (you were asked to provide only one of them):

$|a|=10$ and $0 \le b \le 9$ and $a[b]<0$ and $a[2] \neq 0$

$|a|=10$ and $0 \le b \le 9$ and $a[b]>=3$ and $a[5] \neq 0$

$|a|=10$ and $0 \le b \le 9$ and $0<a[b]<3$ and c=0 and $a[7] \neq 0$.

$|a|=10$ and $0 \le b \le 9$ and (a[b]=0 or (0<=$a[b]$<3 and c!=0)) and $a[4] \neq 0$

Considering the tree, the conditions above ensure the execution of feasible paths to the leaves and further constrain the return value *r*.

Note that path 0, 1, 2, 3, 4, 5, 9, 10, 11, 12, 14, 15, 16, 17 (path 5) cannot lead to the desired condition; indeed, it would give the following condition

$|a|=10$ and $0 \leq b \leq 9$ and $0<a[b]<3$ and c=0 and $a[7] \neq 0$

Which cannot hold because a[7] is equal to *c*, which, in turn, should be 0. Therefore, the corresponding path must return 0 to the caller.