CS5232 Project

NUS Graduate Course Registration System

YANG MO
A0091836X

CHAPTER 1

Introduction

NUS Graduate Course Registration System is an online web application which allows graduate students to register their courses. Unlike a traditional course-bidding system, this system only allows student to rank their choices based on their preference of each candidate module. The final module allocation results will base on each student's choices and rank of preferences under the circumstances that the course quota meet the needs.

In this project, I model the NUS graduate student course registration and seek the best strategy of ranking preferences. In the experiment, some necessary assumptions are made to make the modelling possible to be evaluated using PAT.

CHAPTER 2

Problem Description

NUS Graduate Course Registration System is a relatively big system and it needs to be simplified with necessary assumption, while at the same time it should be realistic enough to provide us with a potential winning strategy for real course registration purpose. We need to decide the number of courses available for registration, number of students, ranking of preferences of each of other students, my own workload upper limit, and what should the course registration system do if there exist same ranks of preferences and there is not enough quota.

Therefore, following main issues need to be settled when the system is modeled:

- Details of students, courses, other student's ranking of preferences, my desirability of each course, my workload limit…
- How the course registration result is handled
- How do model "my ranking of preferences" to allow PAT computation and analysis in order to find an optimal ranking strategy
- How to deal with same my same ranking of preference with other student's ranking
- How does the system run to perform the registration process

CHAPTER 3

System Modeling

3.1 Necessary Assumptions

As mentioned in last chapter, before modelling the system we need to make some necessary assumptions:

1. Course allocation result only come out after the system has done the ranking checking of all students, which will only be triggered if all students including me has already decided on the final preference ranking of the courses.
2. For simplicity, it is assumed that the number of total students is 6 including “myself”. The number of courses is assumed to be 5.
3. It is assumed that all the other students will have already decided their ranking of preferences of each courses. “I” will be the last one to rank my preferences of these courses.
4. Based on 2 and 3, it is assumed that valid ranking range is [1, 5] and everyone will need to put all ranks onto all course.
5. It is assumed that all students will rank their preferences for all the 5 courses. And there should be no duplicated ranking of preference from each of the students.
6. It is assumed that “I” could be either a part-time or full-time master student when registering the courses.
    a. If “I” am a part-time student, my upper workload limit will be 3 courses
    b. If “I” am a full-time student, my upper workload limit will be 5 courses
7. It is assumed that there should be two different situations when my rank of preference of one course is the same with other students and there is not enough quota for all the students given the condition that the rank of preference fall within the winning range:
    a. If it is allowed to have extra quota, “I” will get the course using the extra quota
    b. If it is not allowed to have extra quota, since “I” am assumed to be that last one to rank my preferences I will not get the course due to the policy of “first come first served”.
8. It is assumed that there are four system states: {PRE, OPEN, SUSPEND, SHUTDOWN}. And “I” can only start ranking my preferences when the system state is OPEN. System will change to SUSPEND if “I” finished ranking (based on 3 that all other students have already determined their ranking of preferences) and start allocating the course.
9. All analysis will focus on the strategy of how “I” rank in order to get the optimal results. For all other students, their course allocation details will not be recorded and only the details of total number of students for each course will be tracked.

3.2 Global Constants

Constant	Value	Description
numOfCourses	5	Number of courses for registration
numOfOtherStudents	5	Number of other students except “myself”
PartTimeStudentWorkLoadLimit	3	Upper limit of part-time student’s workload per semester
FullTimeStudentWorkLoadLimit	5	Upper limit of full-time student’s workload per semester
PRE	0	System status – Course registration has not started yet
OPEN	1	System status – Course registration has started, “I” am able to rank preferences of each courses
SUSPEND	2	System status – Course registration has been suspended after all students including “I” have already finished ranking. In this state, course system will check and compare each student’s ranking of preferences to determine whether “I” should be allocated the course as well as the total number of students enrolled in the certain course.
SHUTDOWN	3	System status – Course allocation has ended and the system is closed.

enum {PRE, OPEN, SUSPEND, SHUTDOWN}

3.3 Global Variable Definitions – Conditional Variables

These variables are pre-configured before the simulation is carried out. By assigning different values to these variables, we can simulate various situations in which we can verify and check different properties as well as finding optimal strategy for “my” course ranking to obtain the max number of courses or the max value of my total desirability of obtained courses.

Variable	Definition
myDesire[numOfCourses]	Stores my desirability of each course
courseQuota[numOfCourses]	Stores the quota of each course available for student course registration
otherStudentRank [numOfCourses] [numOfOtherStudents]	Stores the system-observed course preference ranking from all the other students except “me”.
myWorkloadLimit	Stores the upper limit of my workload. • 3 if I am a part-time student • 5 if I am a full-time student
allowExtraQuotaIfHaveSame Rank	Determines whether extra quota is allowed to allocate the course to “me” if my rank of preference is that same with anyone of the other students and the rank is exactly the lowest rank which is eligible for course enrollment based on the quota. Eg. For course 1, other students’ ranking are 1,2,3,4,5 and the quota is 3. My rank for this course is 3. Then : • If allowExtraQuotaIfHaveSameRank is true, I will get the course with the extra quota • If allowExtraQuotaIfHaveSameRank is false, I will not get the course since there is no quota for me.

3.3 Global Variable Definitions – Unconditional Variables

These variables are not conditional variables and they are simply used for recording and system state tracking purpose. Some of them need to be initialized with default value for later usage.

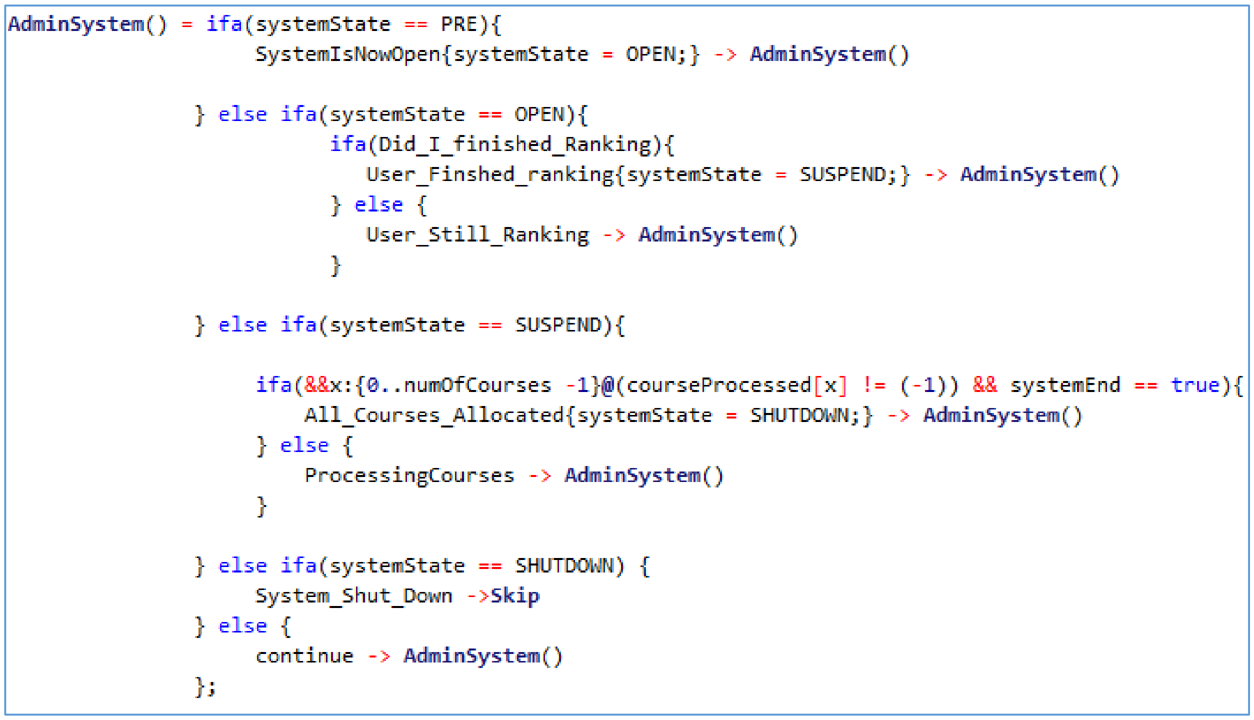
Variable	Definition	Initial value
myTotalDesire	Stores and tracks my total desirability of all the courses allocated to me.	0
myTotalCourses	Stores and tracks the total number of all the courses allocated to me	0
maxIntegerValueOfRank	Stores the maximum possible value of the rank	5
myRank[numOfCourses]	Stores and tracks my rank preferences of all the courses. -1 means not ranked yet.	[-1,-1,-1,-1,-1]
courseProcessed[numOfCourses]	Stores and tracks the processing status of each course. -1 means not processed and allocated to students yet. 1 means the allocation for this course is done	[-1,-1,-1,-1,-1]
courseAllocatedToMe[numOfCourses]	Stores and tracks whether each of the courses are allocated to me. -1 means not allocated to me. 1 means this course is allocated to me.	[-1,-1,-1,-1,-1]
numOfStudentPerCourse[numOfCourses]	Stores and tracks the total number of students who are	[0,0,0,0,0]

	successfully allocated with this course.	
totalNumOfStudentForCurrentCourse	Temp variable used to compute and store the total number of students in a course for each course's rank checking and comparison	0
courseRankingIndex	Stores and tracks the index of course which is being ranked by me	0
numOfPeopleHigherThanMine	Temp variable used to compute and store the total number of other students who has ranked the course higher than me.	0
systemEnd	Stores and tracks whether the system should be shutdown	false
systemState	Stores and tracks the status of the whole system. This is the control variable of the whole simulation.	PRE
courseCheckingIndex	Stores and tracks the index of course which is being checked	0
Did_I_finished_Ranking	Used to track my ranking process status. Indicates whether I am still ranking the course preferences.	false

3.4 Process Definitions

3.4.1 AdminSystem()

This is the process which manages the system control flow according to the system status. It also simulates the action of an administrator monitoring the user ranking (according to our assumptions, this particular refers to the status of my ranking action).



```
AdminSystem() = ifa(systemState == PRE){
                    SystemIsNowOpen{systemState = OPEN;} -> AdminSystem()

               } else ifa(systemState == OPEN){
                          ifa(Did_I_finished_Ranking){
                             User_Finshed_ranking{systemState = SUSPEND;} -> AdminSystem()
                          } else {
                             User_Still_Ranking -> AdminSystem()
                          }

               } else ifa(systemState == SUSPEND){

                    ifa(&&x:{0..numOfCourses -1}@(courseProcessed[x] != (-1)) && systemEnd == true){
                        All_Courses_Allocated{systemState = SHUTDOWN;} -> AdminSystem()
                    } else {
                        ProcessingCourses -> AdminSystem()
                    }

               } else ifa(systemState == SHUTDOWN) {
                    System_Shut_Down ->Skip
               } else {
                    continue -> AdminSystem()
               };
```

It will perform different actions corresponding to the different system status:

- **PRE**
  Course registration system is in PRE state, which means students have not ranked any courses. This will lead to the action of opening the system and change the system state to OPEN

- **OPEN**
  System is already opened for students ranking. Admin needs to check whether students have already finished ranking. In our simulation, it is assumed that all other students will have already finished their ranking. As a result, here it will keep checking whether the last user ("I") have finished ranking all courses.
  - If yes, suspend the system for later course allocation checking
  - If no, indicates that user is still ranking, wait for user to finish ranking.

- **SUSPEND**

System is now in SUSPEND state which means it is checking all students' ranking of preferences for each course. Admin will check if all courses have been processed and allocated:

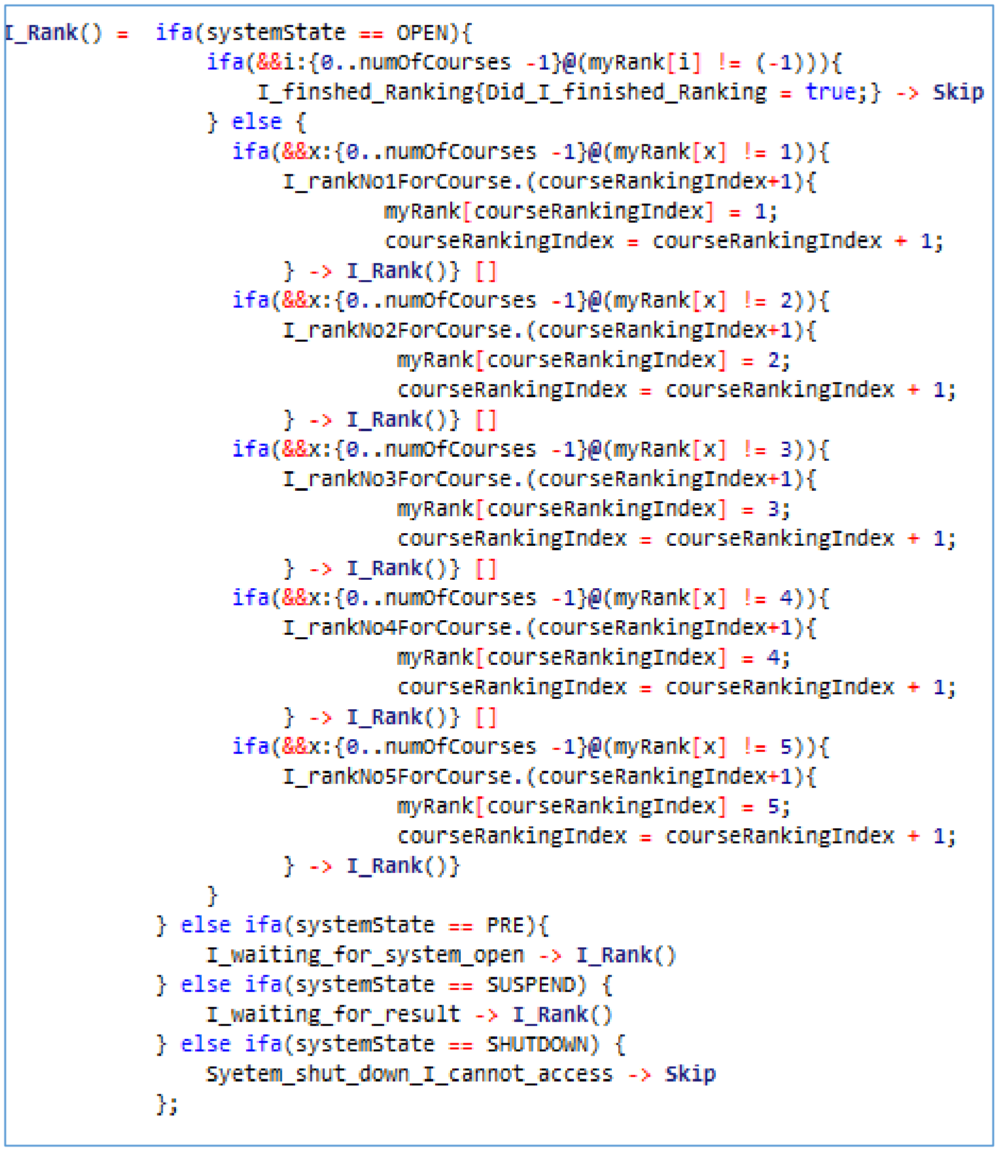
- If yes, bring the system state to SHUTDOWN for future complete shutdown
- If no, indicates that the system is still checking courses and wait for it.

- **SHUTDOWN**

  System is about to be shut down. Simply indicate that the system is now shut down and proceed to "Skip"

3.4.2 I_Rank()

This is the process which simulates "my ranking action". Based on assumption, there are five courses which I need rank.



```
I_Rank() =  ifa(systemState == OPEN){
                ifa(&&i:{0..numOfCourses -1}@(myRank[i] != (-1))){
                    I_finshed_Ranking{Did_I_finished_Ranking = true;} -> Skip
                } else {
                  ifa(&&x:{0..numOfCourses -1}@(myRank[x] != 1)){
                      I_rankNo1ForCourse.(courseRankingIndex+1){
                              myRank[courseRankingIndex] = 1;
                              courseRankingIndex = courseRankingIndex + 1;
                      } -> I_Rank()} []
                  ifa(&&x:{0..numOfCourses -1}@(myRank[x] != 2)){
                      I_rankNo2ForCourse.(courseRankingIndex+1){
                               myRank[courseRankingIndex] = 2;
                               courseRankingIndex = courseRankingIndex + 1;
                      } -> I_Rank()} []
                  ifa(&&x:{0..numOfCourses -1}@(myRank[x] != 3)){
                      I_rankNo3ForCourse.(courseRankingIndex+1){
                               myRank[courseRankingIndex] = 3;
                               courseRankingIndex = courseRankingIndex + 1;
                      } -> I_Rank()} []
                  ifa(&&x:{0..numOfCourses -1}@(myRank[x] != 4)){
                      I_rankNo4ForCourse.(courseRankingIndex+1){
                               myRank[courseRankingIndex] = 4;
                               courseRankingIndex = courseRankingIndex + 1;
                      } -> I_Rank()} []
                  ifa(&&x:{0..numOfCourses -1}@(myRank[x] != 5)){
                      I_rankNo5ForCourse.(courseRankingIndex+1){
                               myRank[courseRankingIndex] = 5;
                               courseRankingIndex = courseRankingIndex + 1;
                      } -> I_Rank()}
                }
            } else ifa(systemState == PRE){
                I_waiting_for_system_open -> I_Rank()
            } else ifa(systemState == SUSPEND) {
                I_waiting_for_result -> I_Rank()
            } else ifa(systemState == SHUTDOWN) {
                Syetem_shut_down_I_cannot_access -> Skip
            };
```

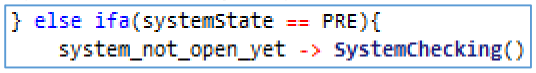
This process also perform differently according to the global system state:

- **PRE**
  System is not open yet. “I” need to wait for the system to be open.
- **OPEN**
  System is now open for course registration. “I” can proceed to ranking for the courses.
    - If I finished ranking for all the course, the variable “`Did_I_finished_Ranking`” will be updated to true. This will be seen by the AdminSystem process for later system state change.
- **SUSPEND**
  System is suspended for course allocation. “I” need to wait for the course registration result.
- **SHUTDOWN**
  System is now shutdown and hence “I” cannot access the system anymore.

3.4.3 SystemChecking()

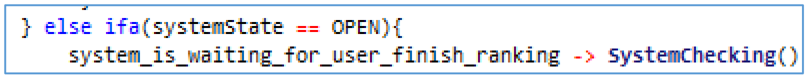
This is the process which dose the actual job to checking the students’ ranking of preferences for all the courses. It also perform different actions according to the system state:

- **PRE**
  System is not open yet, need to wait



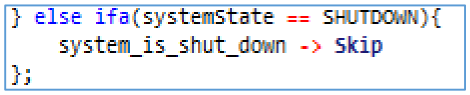
```
} else ifa(systemState == PRE){
    system_not_open_yet -> SystemChecking()
```

- **OPEN**
  System is open for registration but not all students have finished their ranking, so needs to wait for them to finish ranking.



```
} else ifa(systemState == OPEN){
    system_is_waiting_for_user_finish_ranking -> SystemChecking()
```

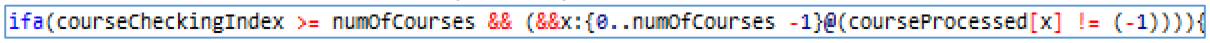
- **SHUTDOWN**
  System is already shut down, no need to do checking anymore. Simply proceed to “Skip”.



```
} else ifa(systemState == SHUTDOWN){
    system_is_shut_down -> Skip
};
```

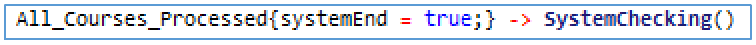
- **SUSPEND**
  This is the major functional part of this SystemChecking() process. It will

  1. Check if all courses have already been processed



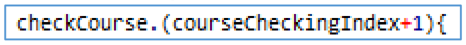
```
ifa(courseCheckingIndex >= numOfCourses && (&&x:{0..numOfCourses -1}@(courseProcessed[x] != (-1)))){
```

1.1 If yes, it will update the variable systemEnd to be true to notify the AdminSystem() process



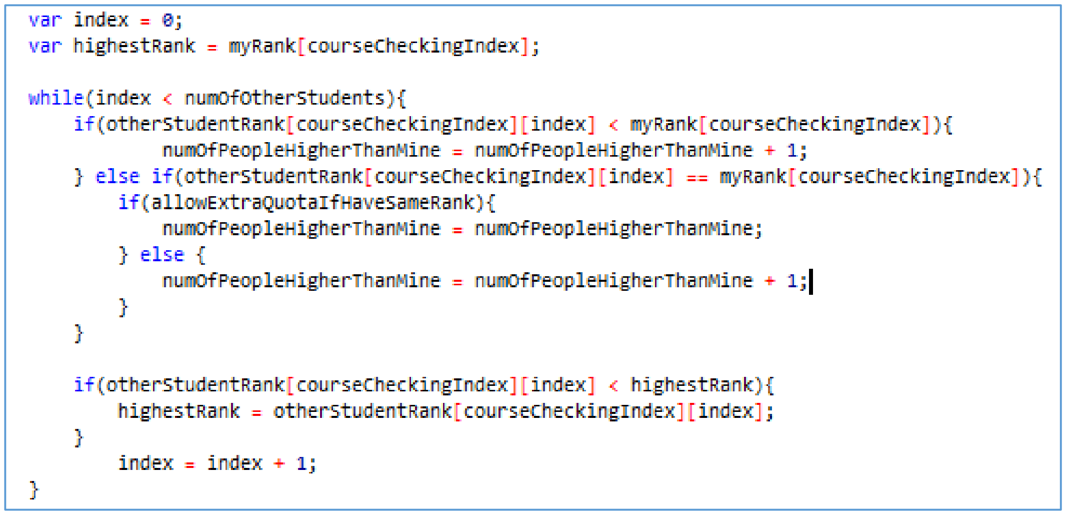
```
All_Courses_Processed{systemEnd = true;} -> SystemChecking()
```

1.2 If no, it will proceed to check the next course for allocation



```
checkCourse.(courseCheckingIndex+1){
```

1.2.1 First find the highest rank for this course as well as the number of students who have ranked higher than me for this course



```
var index = 0;
var highestRank = myRank[courseCheckingIndex];

while(index < numOfOtherStudents){
    if(otherStudentRank[courseCheckingIndex][index] < myRank[courseCheckingIndex]){
            numOfPeopleHigherThanMine = numOfPeopleHigherThanMine + 1;
    } else if(otherStudentRank[courseCheckingIndex][index] == myRank[courseCheckingIndex]){
        if(allowExtraQuotaIfHaveSameRank){
            numOfPeopleHigherThanMine = numOfPeopleHigherThanMine;
        } else {
            numOfPeopleHigherThanMine = numOfPeopleHigherThanMine + 1;
        }
    }

    if(otherStudentRank[courseCheckingIndex][index] < highestRank){
        highestRank = otherStudentRank[courseCheckingIndex][index];
    }
        index = index + 1;
}
```

1.2.2 Compute the total number of students who have successfully secured the course. If there exist same ranks from other students as my rank and the remaining quota for this course is less than the number of students(including me) who has this particular rank

1.2.2.1 If extra quota is allowed for same rank

1.2.2.1.1 If my total courses does not exceed the workload limit, I will get the course

1.2.2.1.2 If my total courses has already exceed the workload limit, I will not get the course

1.2.2.2 If extra quota is not allowed for same rank, I will not get the course

```
        while(highestRank <= maxIntegerValueOfRank && numOfStudentPerCourse[courseCheckingIndex] < courseQuota[courseCheckingIndex]){
            var loop_index = 0;

            if(myRank[courseCheckingIndex] == highestRank && myTotalCourses < myWorkloadLimit ){
                    numOfStudentPerCourse[courseCheckingIndex] = numOfStudentPerCourse[courseCheckingIndex] + 1;
            }

            while(loop_index < numOfOtherStudents){

                if(otherStudentRank[courseCheckingIndex][loop_index] == highestRank){
                        numOfStudentPerCourse[courseCheckingIndex] = numOfStudentPerCourse[courseCheckingIndex] + 1;
                }

                if(numOfStudentPerCourse[courseCheckingIndex] >= courseQuota[courseCheckingIndex] && !allowExtraQuotaIfHaveSameRank){
                    numOfStudentPerCourse[courseCheckingIndex] = courseQuota[courseCheckingIndex];
                }


                loop_index = loop_index + 1;
            }

            highestRank = highestRank + 1;
        }
} -> totalNumberofStudentsIs.(numOfStudentPerCourse[courseCheckingIndex]){courseProcessed[courseCheckingIndex] = 1; }->
```

1.2.3 Indicate whether "I" have been allocated with the course successfully

1.2.3.1 If I get the course within my workload limit, indicate success

```
ifa(courseQuota[courseCheckingIndex] > numOfPeopleHigherThanMine && myTotalCourses < myWorkloadLimit){
    I_Get_Course_.(courseCheckingIndex+1){
            courseAllocatedToMe[courseCheckingIndex] = 1;
            myTotalDesire = myTotalDesire + myDesire[courseCheckingIndex];
            courseCheckingIndex = courseCheckingIndex + 1;
            myTotalCourses = myTotalCourses + 1;
            numOfPeopleHigherThanMine = 0;
    } -> SystemChecking()
```

1.2.3.2 If I did not get the course

```
} else {
    I_Did_Not_Get_Course_.(courseCheckingIndex+1){
            courseCheckingIndex = courseCheckingIndex + 1;
            numOfPeopleHigherThanMine = 0;
```

1.2.3.2.1 If course quota is allowed but I have already hit the workload limit, indicate failure due to "cannot overload"

```
} -> ifa(myTotalCourses >= myWorkloadLimit){
        Reason_Cannot_Overload -> SystemChecking()
```

1.2.3.2.2 If course quota is not allowed and I have not already hit the workload limit, indicate failure due to "others have higher rank than me"

```
} else {
    Reason_OthersGotHigherRank -> SystemChecking()
}
```

3.4.3 CourseResgistration()

```
CourseResgistration() = AdminSystem()|| I_Rank()|| SystemChecking();
```

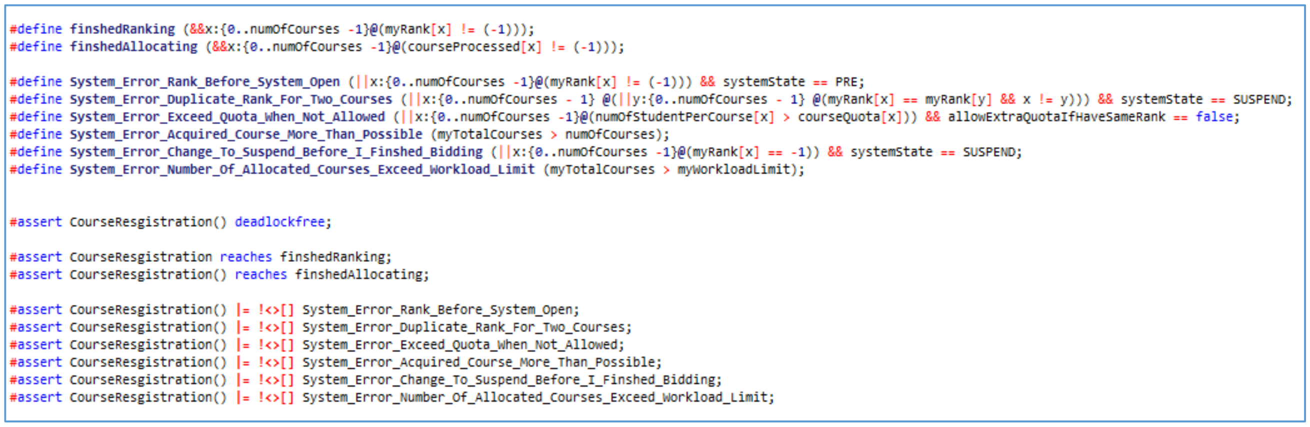
This is the process which correlate all the three processes above to represent the whole course registration process. It simply put all three processes in parallel and they will coordinate with each other based on the commonly used global variables like "systemState".

CHAPTER 4

Investigated Properties

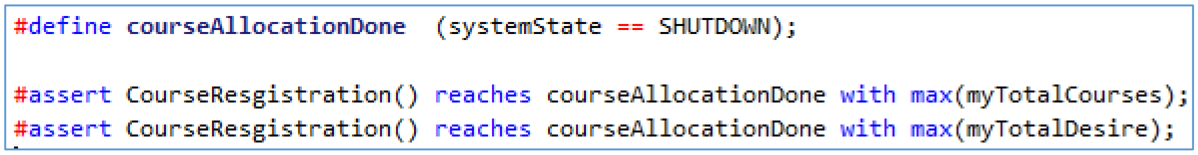
There are two set of main properties I checked for this project:

- **Set 1:** Properties that should be applicable to all test cases and can be checked once
  (Not related to finding optimal course registration strategy)



```
#define finshedRanking (&&x:{0..numOfCourses -1}@(myRank[x] != (-1)));
#define finshedAllocating (&&x:{0..numOfCourses -1}@(courseProcessed[x] != (-1)));

#define System_Error_Rank_Before_System_Open (||x:{0..numOfCourses -1}@(myRank[x] != (-1))) && systemState == PRE;
#define System_Error_Duplicate_Rank_For_Two_Courses (||x:{0..numOfCourses - 1} @(||y:{0..numOfCourses - 1} @(myRank[x] == myRank[y] && x != y))) && systemState == SUSPEND;
#define System_Error_Exceed_Quota_When_Not_Allowed (||x:{0..numOfCourses -1}@(numOfStudentPerCourse[x] > courseQuota[x])) && allowExtraQuotaIfHaveSameRank == false;
#define System_Error_Acquired_Course_More_Than_Possible (myTotalCourses > numOfCourses);
#define System_Error_Change_To_Suspend_Before_I_Finshed_Bidding (||x:{0..numOfCourses -1}@(myRank[x] == -1)) && systemState == SUSPEND;
#define System_Error_Number_Of_Allocated_Courses_Exceed_Workload_Limit (myTotalCourses > myWorkloadLimit);


#assert CourseResgistration() deadlockfree;

#assert CourseResgistration reaches finshedRanking;
#assert CourseResgistration() reaches finshedAllocating;

#assert CourseResgistration() |= !<>[] System_Error_Rank_Before_System_Open;
#assert CourseResgistration() |= !<>[] System_Error_Duplicate_Rank_For_Two_Courses;
#assert CourseResgistration() |= !<>[] System_Error_Exceed_Quota_When_Not_Allowed;
#assert CourseResgistration() |= !<>[] System_Error_Acquired_Course_More_Than_Possible;
#assert CourseResgistration() |= !<>[] System_Error_Change_To_Suspend_Before_I_Finshed_Bidding;
#assert CourseResgistration() |= !<>[] System_Error_Number_Of_Allocated_Courses_Exceed_Workload_Limit;
```

- **Set 2:** Properties that are specific to each test case and should be checked separately in each test case



```
#define courseAllocationDone  (systemState == SHUTDOWN);

#assert CourseResgistration() reaches courseAllocationDone with max(myTotalCourses);
#assert CourseResgistration() reaches courseAllocationDone with max(myTotalDesire);
```

CHAPTER 5

Experiments

The variables that affect the PAT analysis are:

1. My desirability of each course
2. Quota for each course
3. The preference ranking for each course from other students
4. My upper workload limit
5. Whether extra quota is allowed if I have the same rank with other students

Based on these and the properties above, following section will be divided into two parts:

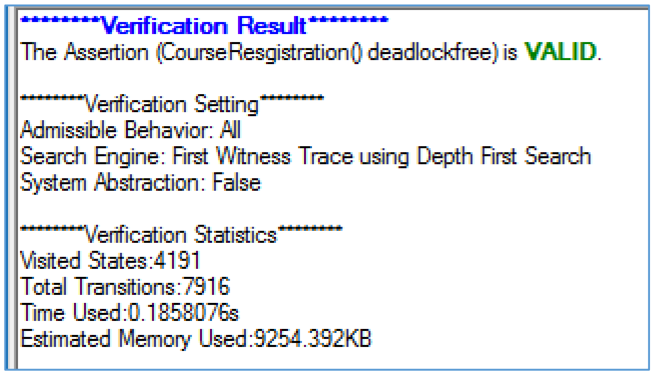
1. Typical case for which both properties set 1 and set 2 from Chapter 4 will be tested
2. Further experiments on investigating my “optimal” course ranking strategy in different variable settings

5.1 Typical Case for Set 1 and 2 properties

numOfCourses	5
numOfOtherStudents	5
myDesire[numOfCourses]	[1,1,1,1,2]
courseQuota[numOfCourses]	[3,3,3,3,3]
otherStudentRank [numOfCourses] [numOfOtherStudents]	[1,2,3,4,5, 2,3,4,5,1, 3,4,5,1,2, 4,5,1,2,3, 5,1,2,3,4]
myWorkloadLimit	PartTimeStudentWorkLoadLimit
allowExtraQuotaIfHaveSameRank	false

5.1.1 Set One Properties

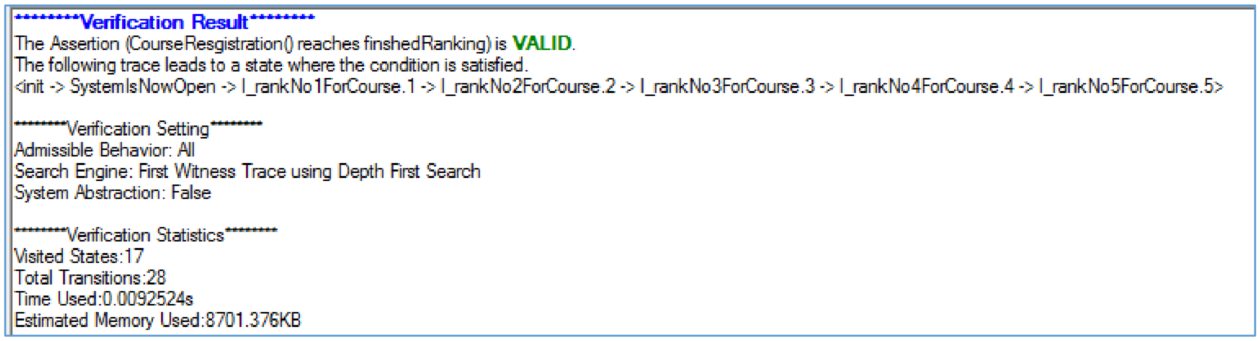
- `#assert CourseResgistration() deadlockfree;`



```
*********Verification Result*********
The Assertion (CourseResgistration() deadlockfree) is VALID.

*********Verification Setting*********
Admissible Behavior: All
Search Engine: First Witness Trace using Depth First Search
System Abstraction: False

*********Verification Statistics*********
Visited States:4191
Total Transitions:7916
Time Used:0.1858076s
Estimated Memory Used:9254.392KB
```

This verifies that my simulated system is deadlock free

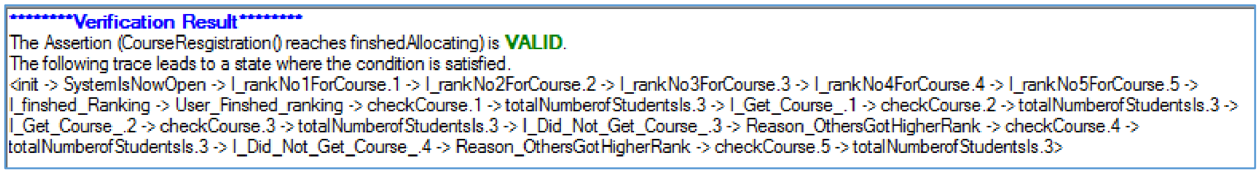
- `#assert CourseResgistration reaches finshedRanking;`



```
*********Verification Result*********
The Assertion (CourseResgistration() reaches finshedRanking) is VALID.
The following trace leads to a state where the condition is satisfied.
<init -> SystemIsNowOpen -> l_rankNo1ForCourse.1 -> l_rankNo2ForCourse.2 -> l_rankNo3ForCourse.3 -> l_rankNo4ForCourse.4 -> l_rankNo5ForCourse.5>

*********Verification Setting*********
Admissible Behavior: All
Search Engine: First Witness Trace using Depth First Search
System Abstraction: False

*********Verification Statistics*********
Visited States:17
Total Transitions:28
Time Used:0.0092524s
Estimated Memory Used:8701.376KB
```

This verifies that I can proceed to finish ranking all courses without a problem

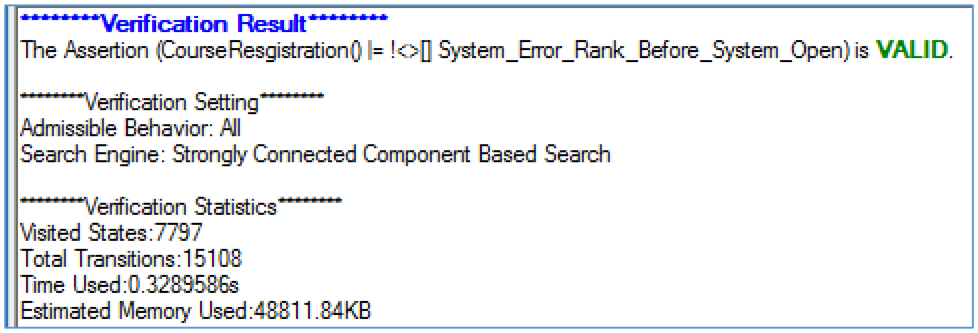
- `#assert CourseResgistration() reaches finshedAllocating;`



```
*********Verification Result*********
The Assertion (CourseResgistration() reaches finshedAllocating) is VALID.
The following trace leads to a state where the condition is satisfied.
<init -> SystemIsNowOpen -> l_rankNo1ForCourse.1 -> l_rankNo2ForCourse.2 -> l_rankNo3ForCourse.3 -> l_rankNo4ForCourse.4 -> l_rankNo5ForCourse.5 ->
l_finshed_Ranking -> User_Finshed_ranking -> checkCourse.1 -> totalNumberofStudentsIs.3 -> l_Get_Course_.1 -> checkCourse.2 -> totalNumberofStudentsIs.3 ->
l_Get_Course_.2 -> checkCourse.3 -> totalNumberofStudentsIs.3 -> l_Did_Not_Get_Course_.3 -> Reason_OthersGotHigherRank -> checkCourse.4 ->
totalNumberofStudentsIs.3 -> l_Did_Not_Get_Course_.4 -> Reason_OthersGotHigherRank -> checkCourse.5 -> totalNumberofStudentsIs.3>
```

This provides a typical path to the system status where all courses will have been allocated to students without any problem

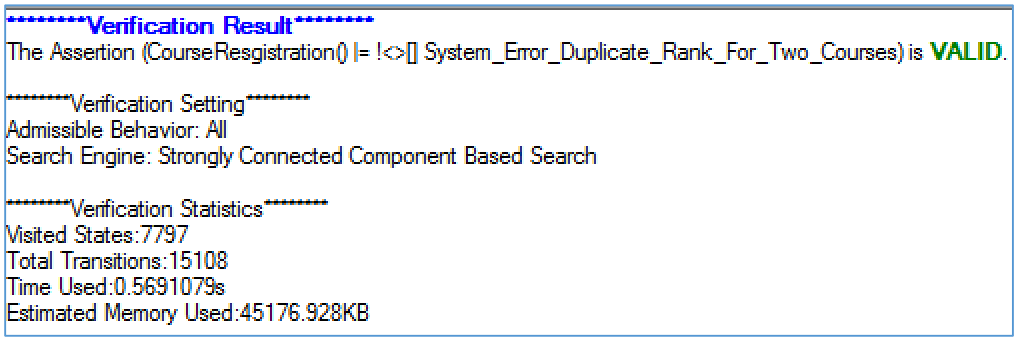
- `#assert CourseResgistration() |= !<>[] System_Error_Rank_Before_System_Open;`



```
*********Verification Result*********
The Assertion (CourseResgistration() |= !<>[] System_Error_Rank_Before_System_Open) is VALID.

*********Verification Setting*********
Admissible Behavior: All
Search Engine: Strongly Connected Component Based Search

*********Verification Statistics*********
Visited States:7797
Total Transitions:15108
Time Used:0.3289586s
Estimated Memory Used:48811.84KB
```

This verifies that I cannot start ranking before the system is open for student course registration

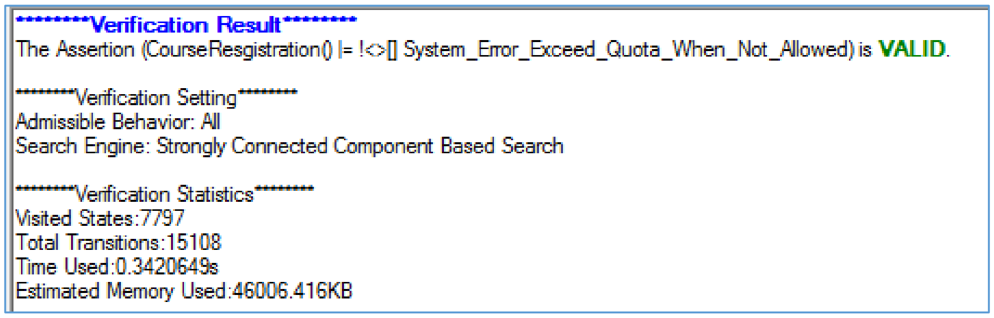
- `#assert CourseResgistration() |= !<>[] System_Error_Duplicate_Rank_For_Two_Courses;`



```
*********Verification Result*********
The Assertion (CourseResgistration() |= !<>[] System_Error_Duplicate_Rank_For_Two_Courses) is VALID.

*********Verification Setting*********
Admissible Behavior: All
Search Engine: Strongly Connected Component Based Search

*********Verification Statistics*********
Visited States:7797
Total Transitions:15108
Time Used:0.5691079s
Estimated Memory Used:45176.928KB
```

This verifies that I cannot have any duplicated ranks for any two courses. This is important because having same ranking for two courses will violate the system policy

- `#assert CourseResgistration() |= !<>[] System_Error_Exceed_Quota_When_Not_Allowed;`

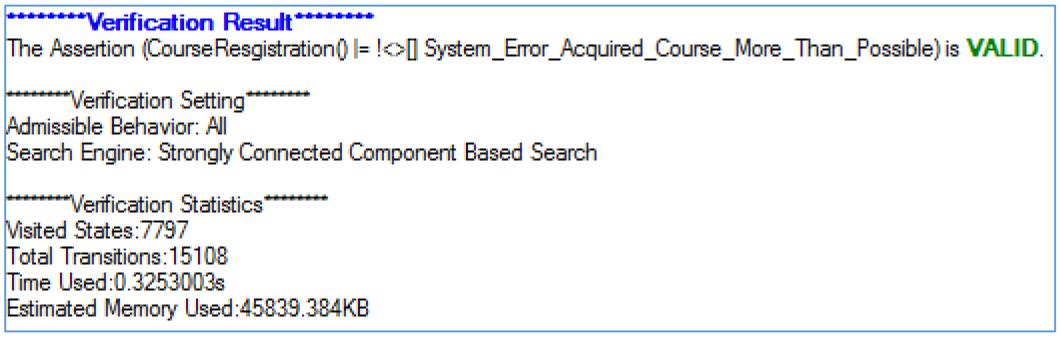


```
*********Verification Result*********
The Assertion (CourseResgistration() |= !<>[] System_Error_Exceed_Quota_When_Not_Allowed) is VALID.

*********Verification Setting*********
Admissible Behavior: All
Search Engine: Strongly Connected Component Based Search

*********Verification Statistics*********
Visited States:7797
Total Transitions:15108
Time Used:0.3420649s
Estimated Memory Used:46006.416KB
```

This verifies that given that `allowExtraQuotaIfHaveSameRank` is false, no extra quota should be given to me even if

- I have the same rank with any other students for any course and
- The remaining quota for this course is less than the number of students having the same rank for this courses

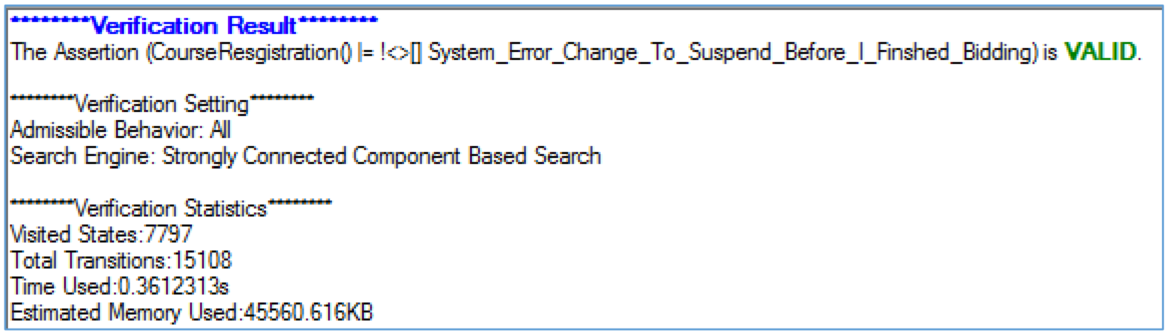
- `#assert CourseResgistration() |= !<>[] System_Error_Acquired_Course_More_Than_Possible;`



```
*********Verification Result*********
The Assertion (CourseResgistration() |= !<>[] System_Error_Acquired_Course_More_Than_Possible) is VALID.

*********Verification Setting*********
Admissible Behavior: All
Search Engine: Strongly Connected Component Based Search

*********Verification Statistics*********
Visited States:7797
Total Transitions:15108
Time Used:0.3253003s
Estimated Memory Used:45839.384KB
```

This verifies that the number of my total allocated courses could not exceed the number of total courses available

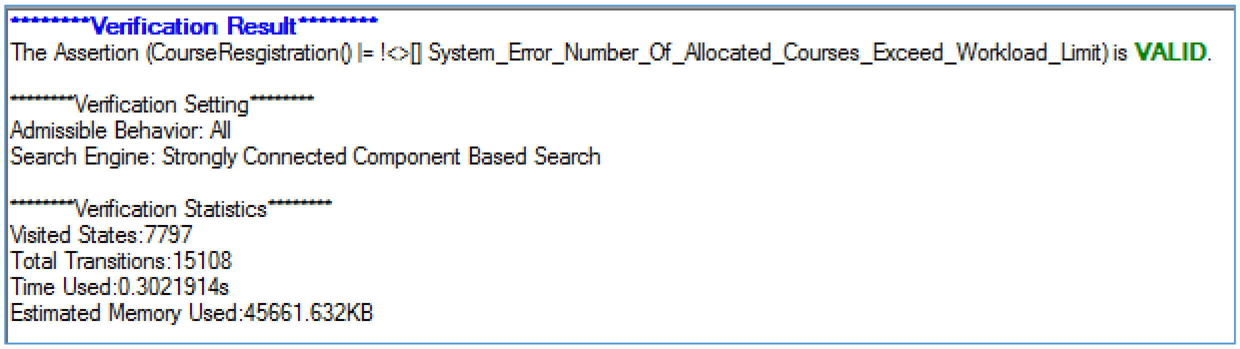
- `#assert CourseResgistration() |= !<>[] System_Error_Change_To_Suspend_Before_I_Finshed_Bidding;`



```
*********Verification Result*********
The Assertion (CourseResgistration() |= !<>[] System_Error_Change_To_Suspend_Before_I_Finshed_Bidding) is VALID.

*********Verification Setting*********
Admissible Behavior: All
Search Engine: Strongly Connected Component Based Search

*********Verification Statistics*********
Visited States:7797
Total Transitions:15108
Time Used:0.3612313s
Estimated Memory Used:45560.616KB
```

This verifies that admin will not change system status from OPEN to SUSPEND if “I” have not finished ranking the courses

- `#assert CourseResgistration() |= !<>[] System_Error_Number_Of_Allocated_Courses_Exceed_Workload_Limit;`

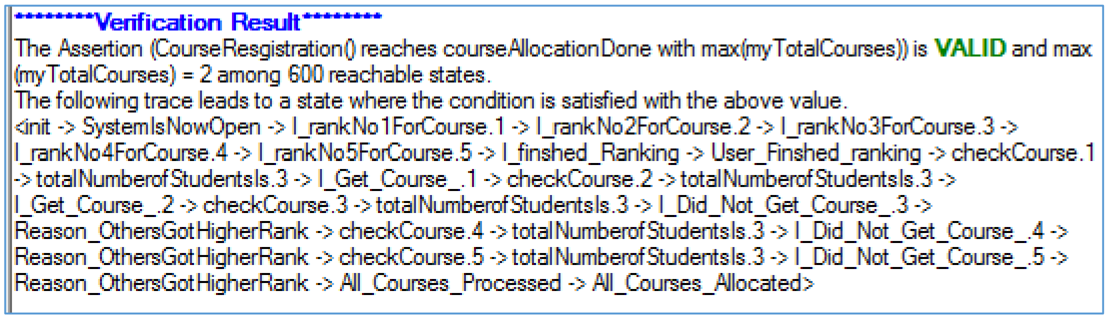


```
*********Verification Result*********
The Assertion (CourseResgistration() |= !<>[] System_Error_Number_Of_Allocated_Courses_Exceed_Workload_Limit) is VALID.

*********Verification Setting*********
Admissible Behavior: All
Search Engine: Strongly Connected Component Based Search

*********Verification Statistics*********
Visited States:7797
Total Transitions:15108
Time Used:0.3021914s
Estimated Memory Used:45661.632KB
```

  This verifies that the number of courses allocated to me cannot exceed my workload upper limit

5.1.2 Set Two Properties

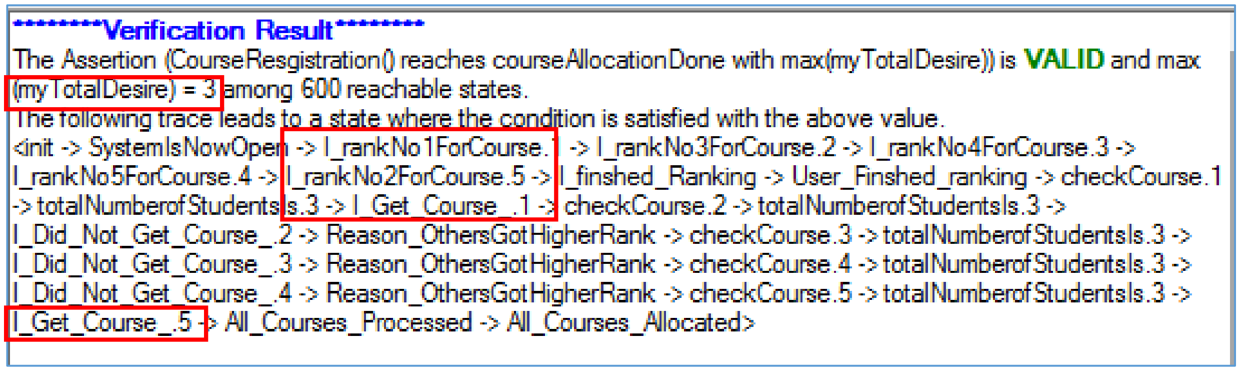
- `#assert CourseResgistration() reaches courseAllocationDone with max(myTotalCourses);`



```
*********Verification Result*********
The Assertion (CourseResgistration() reaches courseAllocationDone with max(myTotalCourses)) is VALID and max
(myTotalCourses) = 2 among 600 reachable states.
The following trace leads to a state where the condition is satisfied with the above value.
<init -> SystemIsNowOpen -> I_rankNo1ForCourse.1 -> I_rankNo2ForCourse.2 -> I_rankNo3ForCourse.3 ->
I_rankNo4ForCourse.4 -> I_rankNo5ForCourse.5 -> I_finshed_Ranking -> User_Finshed_ranking -> checkCourse.1
-> totalNumberofStudentsIs.3 -> I_Get_Course_.1 -> checkCourse.2 -> totalNumberofStudentsIs.3 ->
I_Get_Course_.2 -> checkCourse.3 -> totalNumberofStudentsIs.3 -> I_Did_Not_Get_Course_.3 ->
Reason_OthersGotHigherRank -> checkCourse.4 -> totalNumberofStudentsIs.3 -> I_Did_Not_Get_Course_.4 ->
Reason_OthersGotHigherRank -> checkCourse.5 -> totalNumberofStudentsIs.3 -> I_Did_Not_Get_Course_.5 ->
Reason_OthersGotHigherRank -> All_Courses_Processed -> All_Courses_Allocated>
```

PAT searched through all possible states and assert that the max number of total courses I can possibly get is 2. This is reasonable since no extra quota is allowed for me even if I have the same rank with other students for a certain course.

- `#assert CourseResgistration() reaches courseAllocationDone with max(myTotalDesire);`



```
*********Verification Result*********
The Assertion (CourseResgistration() reaches courseAllocationDone with max(myTotalDesire)) is VALID and max
(myTotalDesire) = 3 among 600 reachable states.
The following trace leads to a state where the condition is satisfied with the above value.
<init -> SystemIsNowOpen -> I_rankNo1ForCourse.1 -> I_rankNo3ForCourse.2 -> I_rankNo4ForCourse.3 ->
I_rankNo5ForCourse.4 -> I_rankNo2ForCourse.5 -> I_finshed_Ranking -> User_Finshed_ranking -> checkCourse.1
-> totalNumberofStudentsIs.3 -> I_Get_Course_.1 -> checkCourse.2 -> totalNumberofStudentsIs.3 ->
I_Did_Not_Get_Course_.2 -> Reason_OthersGotHigherRank -> checkCourse.3 -> totalNumberofStudentsIs.3 ->
I_Did_Not_Get_Course_.3 -> Reason_OthersGotHigherRank -> checkCourse.4 -> totalNumberofStudentsIs.3 ->
I_Did_Not_Get_Course_.4 -> Reason_OthersGotHigherRank -> checkCourse.5 -> totalNumberofStudentsIs.3 ->
I_Get_Course_.5 -> All_Courses_Processed -> All_Courses_Allocated>
```

This is to use PAT to find an optimal strategy of ranking courses to maximize my total desirability of obtained courses.

In this optimal strategy found by PAT, we can see that the maximum total desirability I can have is 3 which is obtained by:

- Rank 1 for course 1 -> get course 1 (desire for course 1 is 1)
- Rank 2 for course 5 -> get course 5 (desire for course 5 is 2)

This strategy is very clever as it successfully managed to obtain course 5 which has the highest desirability since the max number of course I can get in this situation is only 2, which will be verified by the previous property.

5.2 Further experiments on investigating my "optimal" course ranking strategy in different variable settings

In this section more scenarios will be tested with different variable settings

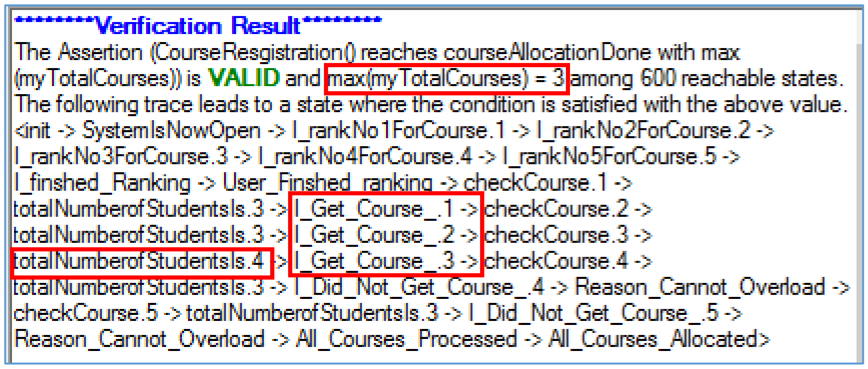
5.2.1 *Scenario 1*

This setting is similar to the above **typical case** except that

- `allowExtraQuotaIfHaveSameRank` is set to true.

numOfCourses	5
numOfOtherStudents	5
myDesire[numOfCourses]	[1,1,1,1,2]
courseQuota[numOfCourses]	[3,3,3,3,3]
otherStudentRank [numOfCourses] [numOfOtherStudents]	[1,2,3,4,5, 2,3,4,5,1, 3,4,5,1,2, 4,5,1,2,3, 5,1,2,3,4]
myWorkloadLimit	PartTimeStudentWorkLoadLimit
allowExtraQuotaIfHaveSameRank	true

- `#assert CourseResgistration() reaches courseAllocationDone with max(myTotalCourses);`

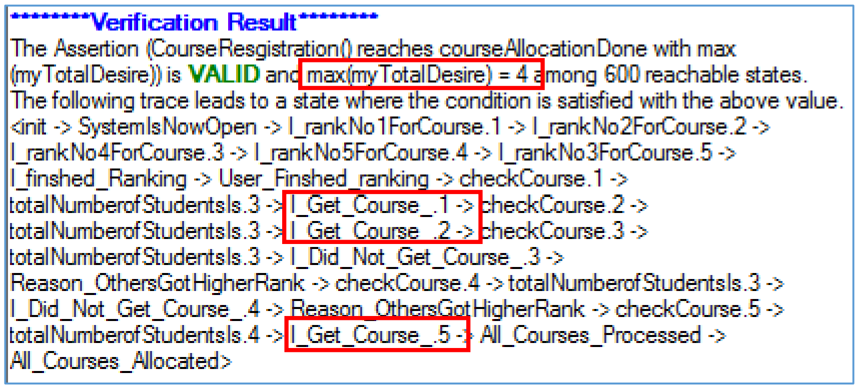


```
********Verification Result********
The Assertion (CourseResgistration() reaches courseAllocationDone with max
(myTotalCourses)) is VALID and max(myTotalCourses) = 3 among 600 reachable states.
The following trace leads to a state where the condition is satisfied with the above value.
<init -> SystemIsNowOpen -> I_rankNo1ForCourse.1 -> I_rankNo2ForCourse.2 ->
I_rankNo3ForCourse.3 -> I_rankNo4ForCourse.4 -> I_rankNo5ForCourse.5 ->
I_finshed_Ranking -> User_Finshed_ranking -> checkCourse.1 ->
totalNumberofStudentsIs.3 -> I_Get_Course_.1 -> checkCourse.2 ->
totalNumberofStudentsIs.3 -> I_Get_Course_.2 -> checkCourse.3 ->
totalNumberofStudentsIs.4 -> I_Get_Course_.3 -> checkCourse.4 ->
totalNumberofStudentsIs.3 -> I_Did_Not_Get_Course_.4 -> Reason_Cannot_Overload ->
checkCourse.5 -> totalNumberofStudentsIs.3 -> I_Did_Not_Get_Course_.5 ->
Reason_Cannot_Overload -> All_Courses_Processed -> All_Courses_Allocated>
```

It can be seen that instead of getting 2 courses in the typical case, now if extra quota is allowed for me having same rank with other students, the max number of courses I can possibly get is now 3.

  - Although quota for course 3 is only 3, my rank for this course(which is 3) is the same with one of another students.
  - Since it is allowed for extra quota, I can get course 3, which is not possible in the typical case.
  - The total number of students for course 3 has also changed from 3(in typical case) to 4 here.

- `#assert CourseResgistration() reaches courseAllocationDone with max(myTotalDesire);`



```
********Verification Result********
The Assertion (CourseResgistration() reaches courseAllocationDone with max
(myTotalDesire)) is VALID and max(myTotalDesire) = 4 among 600 reachable states.
The following trace leads to a state where the condition is satisfied with the above value.
<init -> SystemIsNowOpen -> I_rankNo1ForCourse.1 -> I_rankNo2ForCourse.2 ->
I_rankNo4ForCourse.3 -> I_rankNo5ForCourse.4 -> I_rankNo3ForCourse.5 ->
I_finshed_Ranking -> User_Finshed_ranking -> checkCourse.1 ->
totalNumberofStudentsIs.3 -> I_Get_Course_.1 -> checkCourse.2 ->
totalNumberofStudentsIs.3 -> I_Get_Course_.2 -> checkCourse.3 ->
totalNumberofStudentsIs.3 -> I_Did_Not_Get_Course_.3 ->
Reason_OthersGotHigherRank -> checkCourse.4 -> totalNumberofStudentsIs.3 ->
I_Did_Not_Get_Course_.4 -> Reason_OthersGotHigherRank -> checkCourse.5 ->
totalNumberofStudentsIs.4 -> I_Get_Course_.5 -> All_Courses_Processed ->
All_Courses_Allocated>
```

Similarly, this new optimal plan to maximize my total desirability also increased it from 3(in typical case) to 4. It is so clever that it rank 1 and 2 for course 1 and 2 to successfully get these two courses, after which it rank 3 for course 5 to just exactly obtain course 5. This also lead to an extra quota brought to course 5.

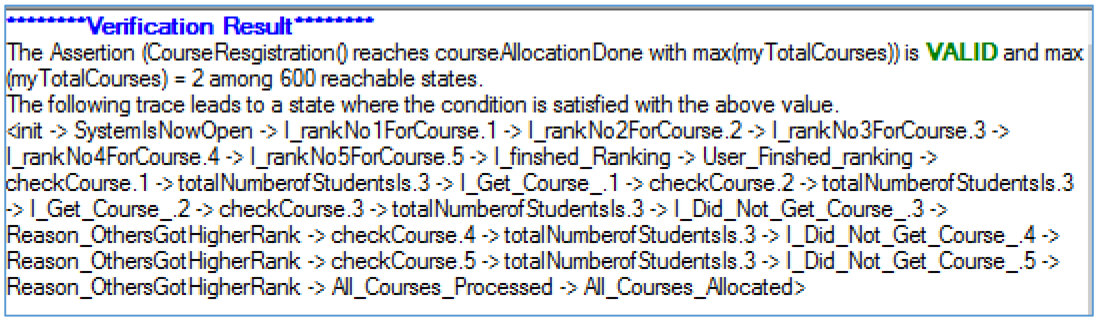
5.2.2 *Scenario 2*

This setting is similar to the above **typical case** except that

- `myWorkloadLimit` is set to `FullTimeStudentWorkLoadLimi`.

numOfCourses	5
numOfOtherStudents	5
myDesire[numOfCourses]	[1,1,1,1,2]
courseQuota[numOfCourses]	[3,3,3,3,3]
otherStudentRank [numOfCourses] [numOfOtherStudents]	[1,2,3,4,5, 2,3,4,5,1, 3,4,5,1,2, 4,5,1,2,3, 5,1,2,3,4]
myWorkloadLimit	FullTimeStudentWorkLoadLimit
allowExtraQuotaIfHaveSameRank	true

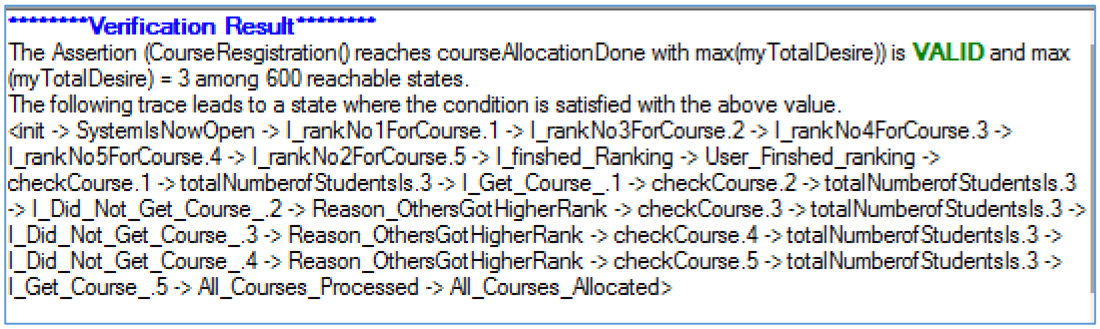
-
  ```
  #assert CourseResgistration() reaches courseAllocationDone with
  max(myTotalCourses);
  ```



```
*********Verification Result*********
The Assertion (CourseResgistration() reaches courseAllocationDone with max(myTotalCourses)) is VALID and max
(myTotalCourses) = 2 among 600 reachable states.
The following trace leads to a state where the condition is satisfied with the above value.
<init -> SystemIsNowOpen -> I_rankNo1ForCourse.1 -> I_rankNo2ForCourse.2 -> I_rankNo3ForCourse.3 ->
I_rankNo4ForCourse.4 -> I_rankNo5ForCourse.5 -> I_finshed_Ranking -> User_Finshed_ranking ->
checkCourse.1 -> totalNumberofStudentsIs.3 -> I_Get_Course_.1 -> checkCourse.2 -> totalNumberofStudentsIs.3
-> I_Get_Course_.2 -> checkCourse.3 -> totalNumberofStudentsIs.3 -> I_Did_Not_Get_Course_.3 ->
Reason_OthersGotHigherRank -> checkCourse.4 -> totalNumberofStudentsIs.3 -> I_Did_Not_Get_Course_.4 ->
Reason_OthersGotHigherRank -> checkCourse.5 -> totalNumberofStudentsIs.3 -> I_Did_Not_Get_Course_.5 ->
Reason_OthersGotHigherRank -> All_Courses_Processed -> All_Courses_Allocated>
```

This verifies that although my workload limit is raised from 3 to 5, in this case with the ranking from other students and quota being 3 for each course, the max number of courses I can get is still just 2. This is correct and reasonable.

-
  ```
  #assert CourseResgistration() reaches courseAllocationDone with
  max(myTotalDesire);
  ```



```
*********Verification Result*********
The Assertion (CourseResgistration() reaches courseAllocationDone with max(myTotalDesire)) is VALID and max
(myTotalDesire) = 3 among 600 reachable states.
The following trace leads to a state where the condition is satisfied with the above value.
<init -> SystemIsNowOpen -> I_rankNo1ForCourse.1 -> I_rankNo3ForCourse.2 -> I_rankNo4ForCourse.3 ->
I_rankNo5ForCourse.4 -> I_rankNo2ForCourse.5 -> I_finshed_Ranking -> User_Finshed_ranking ->
checkCourse.1 -> totalNumberofStudentsIs.3 -> I_Get_Course_.1 -> checkCourse.2 -> totalNumberofStudentsIs.3
-> I_Did_Not_Get_Course_.2 -> Reason_OthersGotHigherRank -> checkCourse.3 -> totalNumberofStudentsIs.3 ->
I_Did_Not_Get_Course_.3 -> Reason_OthersGotHigherRank -> checkCourse.4 -> totalNumberofStudentsIs.3 ->
I_Did_Not_Get_Course_.4 -> Reason_OthersGotHigherRank -> checkCourse.5 -> totalNumberofStudentsIs.3 ->
I_Get_Course_.5 -> All_Courses_Processed -> All_Courses_Allocated>
```

Similarly, raising the workload limit here will not change the max total desirability I can get from the courses allocated to me.

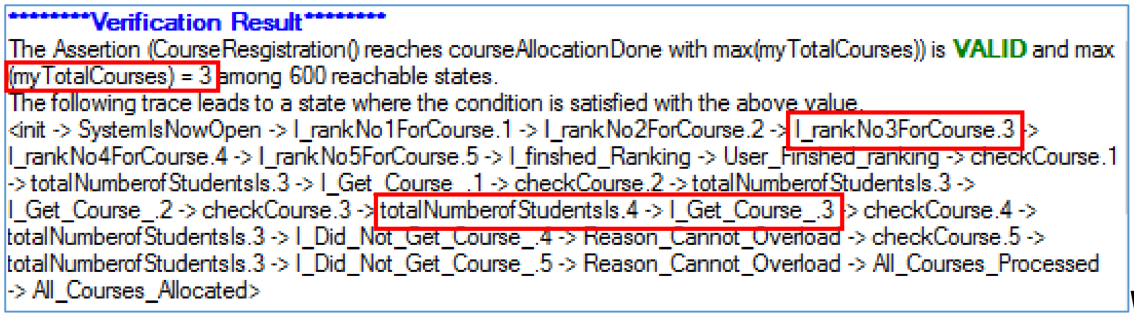
5.2.3 Scenario 3

This setting is similar to the above **typical case** except that

- Quota for course 3 is increased to 4.

numOfCourses	5
numOfOtherStudents	5
myDesire[numOfCourses]	[1,1,1,1,2]
courseQuota[numOfCourses]	[3,3,4,3,3]
otherStudentRank [numOfCourses] [numOfOtherStudents]	[1,2,3,4,5, 2,3,4,5,1, 3,4,5,1,2, 4,5,1,2,3, 5,1,2,3,4]
myWorkloadLimit	PartTimeStudentWorkLoadLimit
allowExtraQuotaIfHaveSameRank	true

-
  ```
  #assert CourseResgistration() reaches courseAllocationDone with
  max(myTotalCourses);
  ```

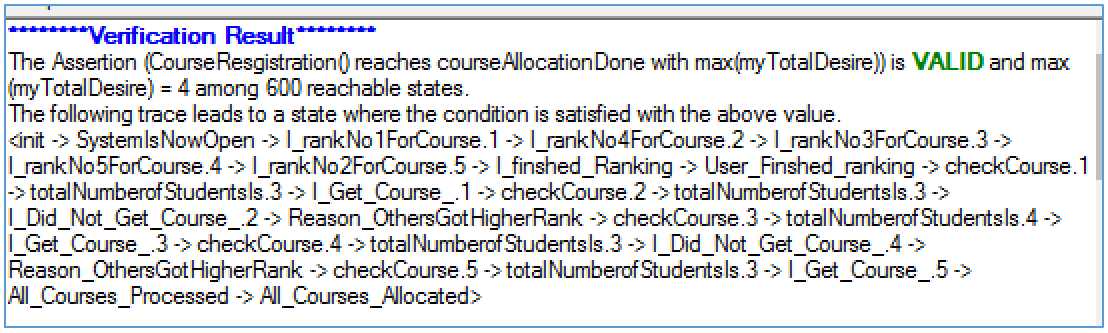


```
********Verification Result********
The Assertion (CourseResgistration() reaches courseAllocationDone with max(myTotalCourses)) is VALID and max
(myTotalCourses) = 3 among 600 reachable states.
The following trace leads to a state where the condition is satisfied with the above value.
<init -> SystemIsNowOpen -> I_rankNo1ForCourse.1 -> I_rankNo2ForCourse.2 -> I_rankNo3ForCourse.3 ->
I_rankNo4ForCourse.4 -> I_rankNo5ForCourse.5 -> I_finshed_Ranking -> User_Finshed_ranking -> checkCourse.1
-> totalNumberofStudentsIs.3 -> I_Get_Course_.1 -> checkCourse.2 -> totalNumberofStudentsIs.3 ->
I_Get_Course_.2 -> checkCourse.3 -> totalNumberofStudentsIs.4 -> I_Get_Course_.3 -> checkCourse.4 ->
totalNumberofStudentsIs.3 -> I_Did_Not_Get_Course_.4 -> Reason_Cannot_Overload -> checkCourse.5 ->
totalNumberofStudentsIs.3 -> I_Did_Not_Get_Course_.5 -> Reason_Cannot_Overload -> All_Courses_Processed
-> All_Courses_Allocated>
```

\

Different from typical case, now the max number of courses I can get is increased to 3 by ranking 3 for course 3 to obtain the additional quota added in the case. It can be seen that the optimal plan successfully observed this and obtained this course 3.

-
  ```
  #assert CourseResgistration() reaches courseAllocationDone with
  max(myTotalDesire);
  ```



```
********Verification Result********
The Assertion (CourseResgistration() reaches courseAllocationDone with max(myTotalDesire)) is VALID and max
(myTotalDesire) = 4 among 600 reachable states.
The following trace leads to a state where the condition is satisfied with the above value.
<init -> SystemIsNowOpen -> I_rankNo1ForCourse.1 -> I_rankNo4ForCourse.2 -> I_rankNo3ForCourse.3 ->
I_rankNo5ForCourse.4 -> I_rankNo2ForCourse.5 -> I_finshed_Ranking -> User_Finshed_ranking -> checkCourse.1
-> totalNumberofStudentsIs.3 -> I_Get_Course_.1 -> checkCourse.2 -> totalNumberofStudentsIs.3 ->
I_Did_Not_Get_Course_.2 -> Reason_OthersGotHigherRank -> checkCourse.3 -> totalNumberofStudentsIs.4 ->
I_Get_Course_.3 -> checkCourse.4 -> totalNumberofStudentsIs.3 -> I_Did_Not_Get_Course_.4 ->
Reason_OthersGotHigherRank -> checkCourse.5 -> totalNumberofStudentsIs.3 -> I_Get_Course_.5 ->
All_Courses_Processed -> All_Courses_Allocated>
```

Similarly, by ranking course 3 with rank 3 this optimal strategy now can bring the total desirability from 3 (in the typical case) to 4 now.

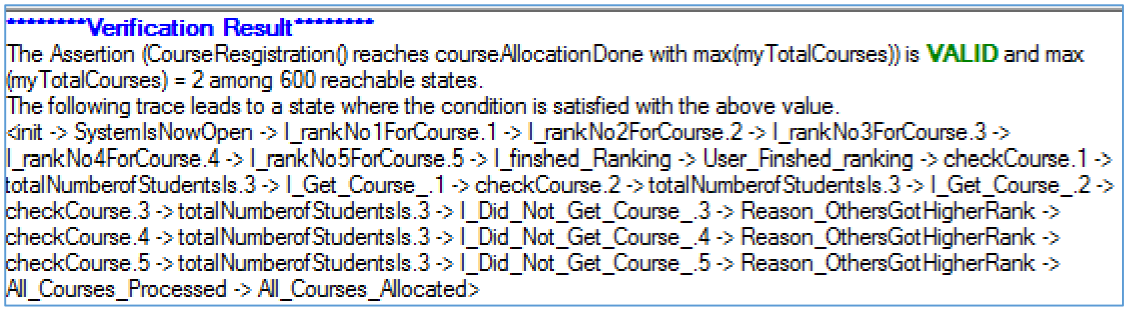
5.2.4 Scenario 4

This setting is similar to the above **typical case** except that

- My desirability for course 5 is increased to 4.

numOfCourses	5
numOfOtherStudents	5
myDesire[numOfCourses]	[1,1,1,1,4]
courseQuota[numOfCourses]	[3,3,3,3,3]
otherStudentRank [numOfCourses] [numOfOtherStudents]	[1,2,3,4,5, 2,3,4,5,1, 3,4,5,1,2, 4,5,1,2,3, 5,1,2,3,4]
myWorkloadLimit	PartTimeStudentWorkLoadLimit
allowExtraQuotaIfHaveSameRank	true

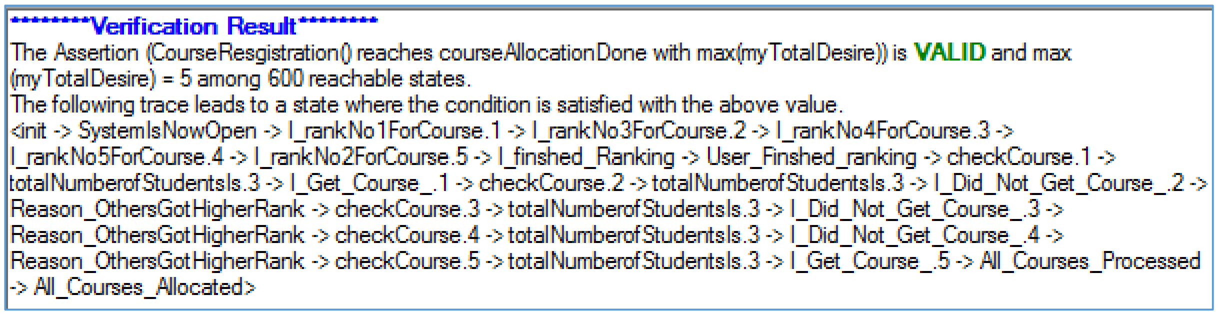
-
  ```
  #assert CourseResgistration() reaches courseAllocationDone with
  max(myTotalCourses);
  ```

  ```
  *********Verification Result*********
  The Assertion (CourseResgistration() reaches courseAllocationDone with max(myTotalCourses)) is VALID and max
  (myTotalCourses) = 2 among 600 reachable states.
  The following trace leads to a state where the condition is satisfied with the above value.
  <init -> SystemIsNowOpen -> I_rankNo1ForCourse.1 -> I_rankNo2ForCourse.2 -> I_rankNo3ForCourse.3 ->
  I_rankNo4ForCourse.4 -> I_rankNo5ForCourse.5 -> I_finshed_Ranking -> User_Finshed_ranking -> checkCourse.1 ->
  totalNumberofStudentsIs.3 -> I_Get_Course_.1 -> checkCourse.2 -> totalNumberofStudentsIs.3 -> I_Get_Course_.2 ->
  checkCourse.3 -> totalNumberofStudentsIs.3 -> I_Did_Not_Get_Course_.3 -> Reason_OthersGotHigherRank ->
  checkCourse.4 -> totalNumberofStudentsIs.3 -> I_Did_Not_Get_Course_.4 -> Reason_OthersGotHigherRank ->
  checkCourse.5 -> totalNumberofStudentsIs.3 -> I_Did_Not_Get_Course_.5 -> Reason_OthersGotHigherRank ->
  All_Courses_Processed -> All_Courses_Allocated>
  ```

  This verifies that the change in my desirability of courses will not affect the possible max number of courses I can get.

-
  ```
  #assert CourseResgistration() reaches courseAllocationDone with
  max(myTotalDesire);
  ```

  ```
  *********Verification Result*********
  The Assertion (CourseResgistration() reaches courseAllocationDone with max(myTotalDesire)) is VALID and max
  (myTotalDesire) = 5 among 600 reachable states.
  The following trace leads to a state where the condition is satisfied with the above value.
  <init -> SystemIsNowOpen -> I_rankNo1ForCourse.1 -> I_rankNo3ForCourse.2 -> I_rankNo4ForCourse.3 ->
  I_rankNo5ForCourse.4 -> I_rankNo2ForCourse.5 -> I_finshed_Ranking -> User_Finshed_ranking -> checkCourse.1 ->
  totalNumberofStudentsIs.3 -> I_Get_Course_.1 -> checkCourse.2 -> totalNumberofStudentsIs.3 -> I_Did_Not_Get_Course_.2 ->
  Reason_OthersGotHigherRank -> checkCourse.3 -> totalNumberofStudentsIs.3 -> I_Did_Not_Get_Course_.3 ->
  Reason_OthersGotHigherRank -> checkCourse.4 -> totalNumberofStudentsIs.3 -> I_Did_Not_Get_Course_.4 ->
  Reason_OthersGotHigherRank -> checkCourse.5 -> totalNumberofStudentsIs.3 -> I_Get_Course_.5 -> All_Courses_Processed
  -> All_Courses_Allocated>
  ```

  This verifies that the new possible total desirability will always tend to obtain the highest desired course. In this case, the increased desirability was successfully captured.

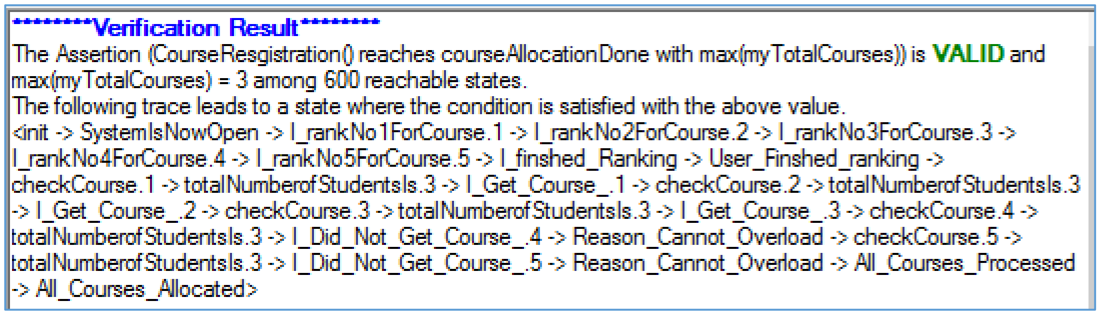
5.2.5 Scenario 5

This setting is similar to the scenario 4 except that

- otherStudentRank matrix is changed for student 1

numOfCourses	5
numOfOtherStudents	5
myDesire[numOfCourses]	[1,1,1,1,4]
courseQuota[numOfCourses]	[3,3,3,3,3]
otherStudentRank [numOfCourses] [numOfOtherStudents]	[**3**,2,3,4,5, **4**,3,4,5,1, **5**,4,5,1,2, **1**,5,1,2,3, **2**,1,2,3,4]
myWorkloadLimit	PartTimeStudentWorkLoadLimit
allowExtraQuotaIfHaveSameRank	true

-
  ```
  #assert CourseResgistration() reaches courseAllocationDone with
  max(myTotalCourses);
  ```



********Verification Result********

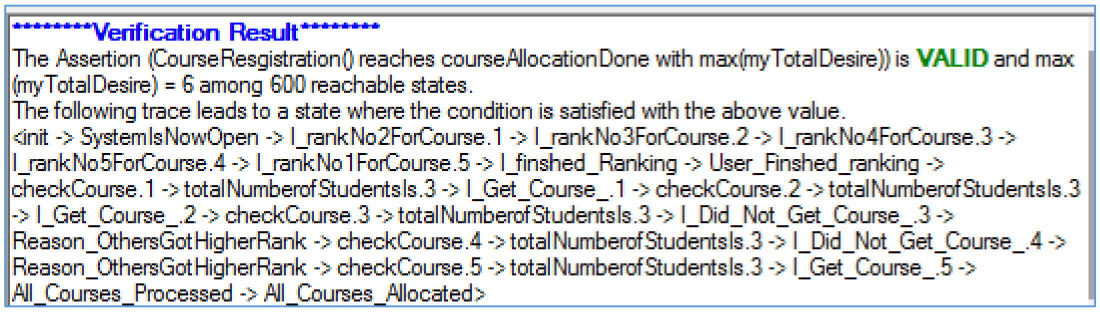
The Assertion (CourseResgistration() reaches courseAllocationDone with max(myTotalCourses)) is VALID and max(myTotalCourses) = 3 among 600 reachable states.

The following trace leads to a state where the condition is satisfied with the above value.

<init -> SystemIsNowOpen -> I_rankNo1ForCourse.1 -> I_rankNo2ForCourse.2 -> I_rankNo3ForCourse.3 -> I_rankNo4ForCourse.4 -> I_rankNo5ForCourse.5 -> I_finshed_Ranking -> User_Finshed_ranking -> checkCourse.1 -> totalNumberofStudentsIs.3 -> I_Get_Course_.1 -> checkCourse.2 -> totalNumberofStudentsIs.3 -> I_Get_Course_.2 -> checkCourse.3 -> totalNumberofStudentsIs.3 -> I_Get_Course_.3 -> checkCourse.4 -> totalNumberofStudentsIs.3 -> I_Did_Not_Get_Course_.4 -> Reason_Cannot_Overload -> checkCourse.5 -> totalNumberofStudentsIs.3 -> I_Did_Not_Get_Course_.5 -> Reason_Cannot_Overload -> All_Courses_Processed -> All_Courses_Allocated>

This optimal strategy to maximize the total number of courses is reasonable since it successfully make use of the ranking gaps in courses 1,2 and 3 to get these 3 courses(scenario 4 is 2 courses)

-
  ```
  #assert CourseResgistration() reaches courseAllocationDone with
  max(myTotalDesire);
  ```



********Verification Result********

The Assertion (CourseResgistration() reaches courseAllocationDone with max(myTotalDesire)) is VALID and max (myTotalDesire) = 6 among 600 reachable states.

The following trace leads to a state where the condition is satisfied with the above value.

<init -> SystemIsNowOpen -> I_rankNo2ForCourse.1 -> I_rankNo3ForCourse.2 -> I_rankNo4ForCourse.3 -> I_rankNo5ForCourse.4 -> I_rankNo1ForCourse.5 -> I_finshed_Ranking -> User_Finshed_ranking -> checkCourse.1 -> totalNumberofStudentsIs.3 -> I_Get_Course_.1 -> checkCourse.2 -> totalNumberofStudentsIs.3 -> I_Get_Course_.2 -> checkCourse.3 -> totalNumberofStudentsIs.3 -> I_Did_Not_Get_Course_.3 -> Reason_OthersGotHigherRank -> checkCourse.4 -> totalNumberofStudentsIs.3 -> I_Did_Not_Get_Course_.4 -> Reason_OthersGotHigherRank -> checkCourse.5 -> totalNumberofStudentsIs.3 -> I_Get_Course_.5 -> All_Courses_Processed -> All_Courses_Allocated>

This is similar but the most obvious change is for course 5. In this case quota for course 5 is 3 and two students have already ranked 2 for course 5. To obtain this highly desired course, the rank for it needs to be changed to 1 (rank was 2 in scenario 4). This was successfully achieved by this new optimal strategy.

CHAPTER 6

Discussions

Limitations:

- My model is simplified to a small number of students and courses for experiment purpose. And in the real NUS graduate registration system, the number of students and courses is much larger than this.
- It is assumed that I will always come as the last one to place my ranking of preferences. It would be more realistic if the order of me finalizing the ranking could be handled in more complex cases.

Possible extensions:

- This simulation can be potentially extended to a more large-scaled model to simulate the real NUS graduate course registration system. In the open ranking stage, course quota and current ranking from all other students are available to all students. Maybe this tool can be used to help me obtain my desired courses in a much strategic way.

CHAPTER 7

Feedback and Suggestions for PAT

First, please let me express my appreciation to PAT team implementing such a powerful software. After using the PAT for my project, it is the first time that I realize some difficult problems which I though hard to solve can actually be easily solved using PAT. However, after this wonderful learning experience with PAT, next time when I encounter a similar issue to work on I will not hesitate to use PAT as the only tool. Moreover, I will recommend PAT to others with the same need.

Second, I have one suggestion on the display of multi-dimension arrays. Currently I use a 2D array to store the ranking of preferences from each student for each course. When I use the simulation tool to check the process flow and simulate a possible auction, the way it display the value of 2D array is displaying it by converting it to a single long array. This is fine my array is small, but it may not be very easy to check the value when the array becomes very big. Hence it is my suggestion that if possible, the display of multi-dimension array in simulation tool could have a better look.