Universitetet i Stavanger

FACULTY OF SCIENCE AND TECHNOLOGY

MASTER'S THESIS

Study programme/specialisation: Master of Science in Computer Science	Spring / ~~Autumn~~ semester, 2019 Open/~~Confidential~~
Author: Bilal Amjad	(signature of author)
Supervisor(s): Reggie Davidrajuh	
Title of master's thesis: Modelling, Simulation and Performance Evaluation of Modern Elevators	
Credits: 30	
Keywords: Elevators, Elevator Algorithms, Classical Algorithm, Modern Algorithms, Shabbat Algorithm, First Come First Serve Elevator Algorithm	Number of pages: 169 + supplemental material/other: Code included in PDF Stavanger, 15/06/2019 date/year

Table of Contents

Abstract: 1
1. Introduction: 1
1.1 Problem Definition: 1
1.2 Classical & modern elevator algorithms: 2
1.2.1 Classical algorithm: 2
1.2.1.1 Shabbat elevator algorithm: 2
1.2.2 Modern Algorithms: 2
1.2.2.1 First come first serve elevator algorithm: 2
1.2.2.2 Fixed sectoring elevator algorithm: 2
1.2.2.3 Dynamic sectoring elevator algorithm: 2
1.2.2.4 First come first serve elevator with VIP passcode: 2
1.3 Adaptations: 3
2. Method and Design: 3
2.1 Overall Design: 3
2.2 Modular Approach: 5
2.2.1 Arrival Module: 5
2.2.2 Initialization Module: 5
2.2.3 Car At a Level Module: 5
2.2.4 Departure Module: 6
2.2.5 Car Moving Module: 6
2.2.6 Inter module connections: 7
2.3 Technique: 8
3. Implementation: 8
3.1 Shabbat Algorithm: 8
3.1.1 Coding of modules: 8
3.1.1.1 Arrival module: 8
3.1.1.2 Initialization module: 11
3.1.1.3 Car At a Level Module: 11
3.1.1.4 Departure module: 17

3.1.1.5 Car Moving module: 17
3.1.1.6 Inter modular connections: 20
3.2 First come first serve elevator algorithm: 21
3.2.1 Coding of modules: 21
3.2.1.1 Arrival module: 21
3.2.1.2 Initialization module: 25
3.2.1.3 Car At a Level Module: 25
3.2.1.4 Departure module: 30
3.2.1.5 Car Moving module: 31
3.2.1.6 Inter modular connections: 34
3.3 First come first serve elevator with VIP passcode: 35
3.3.1 Coding of modules: 35
3.3.1.1 Arrival module: 35
3.3.1.2 Initialization module: 39
3.3.1.3 Car At a Level Module: 39
3.3.1.4 Departure module: 44
3.3.1.5 Car Moving module: 45
3.3.1.6 Inter modular connections: 48
3.4 Fixed sectoring elevator algorithm: 49
3.4.1 Coding of modules: 49
3.4.1.1 Arrival module: 49
3.4.1.2 Initialization module: 52
3.4.1.3 Car At a Level Module: 52
3.4.1.4 Departure module: 61
3.4.1.5 Car Moving module: 62
3.4.1.6 Inter modular connections: 65
3.5 Dynamic sectoring elevator algorithm: 66
3.5.1 Coding of modules: 68
3.5.1.1 Arrival module: 68
3.5.1.2 Initialization module: 71
3.5.1.3 Car At a Level Module: 71
3.5.1.4 Departure module: 83

3.5.1.5 Car Moving module: 84
3.5.1.6 Inter modular connections: 87
4. Testing, Analysis and Results: 88
4.1 Simple User Manual: 88
4.1.1 For Shabbat, FCFS and fixed sectoring : 88
4.1.2 For FCFS with VIP passcode: 89
4.1.3 For Dynamic sectoring: 89
4.2 Sample Run: 90
4.2.1 Shabbat Algorithm: 90
4.2.2 First Come First Serve elevator algorithm: 92
4.2.3 Fixed sectoring elevator algorithm: 93
4.2.4 Dynamic sectoring elevator algorithm: 95
4.2.4.1 Night-time sample run: 95
4.2.4.2 Morning time sample run: 96
4.2.5 First Come First Serve elevator algorithm with VIP passcode: 98
4.3 Data Description: 99
4.4 Statistical Analysis: 99
4.4.1 Shabbat Algorithm: 99
4.4.2 First Come First Serve elevator algorithm: 100
4.4.3 Fixed sectoring elevator algorithm: 101
4.4.4 Dynamic sectoring elevator algorithm: 102
4.4.4.1 Night-time: 102
4.4.4.2 Morning-time: 103
4.4.5 First Come First Serve elevator algorithm with VIP passcode: 104
4.5 Originality of this work: 105
4.6 Petri Net properties analysis: 105
4.6.1 Safeness: 105
4.6.2 Liveness: 105
4.6.2.1 In case of FCFS movement of car: 105
4.6.2.2 In case of Shabbat movement of car: 106
4.6.3 Deadlock: 106
5. Discussion: 106

5.1 Originality of this work: 106
5.2 Relevant Works: 106
5.2.1 First Relevance: 106
5.2.2 Second Relevance: 107
5.2.3 Third relevance: 107
5.3 Limitations of my work: 107
5.4 Further work: 107
5.5 Learning Experience: 108
References: 108
Appendix: 109
A1: Installation Guide: 109
A2: User Manual: 109
A2.1: For Shabbat, FCFS and fixed sectoring: 109
A2.2: For FCFS with VIP passcode: 110
A2.3: Dynamic sectoring: 110
A3: Complete Code: 111
A3.1: Shabbat elevator algorithm: 111
A3.1.1 Main Simulation File (MSF) 111
A3.1.2 PDF Files: 112
A3.1.2.1 Init_pdf() 112
A3.1.2.2 Arrival_pdf() 112
A3.1.2.3 Car_ata_Level_pdf() 112
A3.1.2.4 Car_Moving_pdf() 113
A3.1.2.5 Departure_pdf() 113
A3.1.2.6 IMC_pdf() 113
A3.1.3 tArrival_pre(trans) 114
A3.1.4 loadData() 114
A3.1.5 fillMissedCarTable() 116
A3.1.6 separateColors(the2colors) 116
A3.1.7 COMMON_PRE(trans) 116
A3.1.8 COMMON_POST(trans) 119
A3.2 First come first serve elevator algorithm: 119

A3.2.1 Main Simulation File (MSF): 119
A3.2.2 PDF Files: 120
A3.2.2.1 Init_pdf() 120
A3.2.2.2 Arrival_pdf() 120
A3.2.2.3 Car_ata_Level_pdf() 120
A3.2.2.4 Car_Moving_pdf() 121
A3.2.2.5 Departure_pdf() 121
A3.2.2.6 IMC_pdf() 121
A3.2.3 tArrival_pre(trans) 122
A3.2.4 loadData() 123
A3.2.5 fillMissedCarTable() 124
A3.2.6 separateColors(the2colors) 125
A3.2.7 COMMON_PRE(trans) 125
A3.2.8 COMMON_POST(trans) 129
A3.3 Fixed sectoring elevator algorithm: 130
A3.3.1 Main Simulation File (MSF): 130
A3.3.2 PDF Files: 131
A3.3.2.1 Init_pdf() 131
A3.3.2.2 Arrival_pdf() 131
A3.3.2.3 Car_ata_Level_pdf() 131
A3.3.2.4 Car_Moving_pdf() 132
A3.3.2.5 Departure_pdf() 132
A3.3.2.6 IMC_pdf() 132
A3.3.3 tArrival_pre(trans) 133
A3.3.4 loadData() 134
A3.3.5 fillMissedCarTable() 135
A3.3.6 separateColors(the2colors) 136
A3.3.7 COMMON_PRE(trans) 136
A3.3.8 COMMON_POST(trans) 143
A3.4 Dynamic sectoring elevator algorithm: 144
A3.4.1 Main Simulation File (MSF): 144
A3.4.2 PDF Files: 145

A3.4.2.1 Init_pdf() 145
A3.4.2.2 Arrival_pdf() 145
A3.4.2.3 Car_ata_Level_pdf() 145
A3.4.2.4 Car_Moving_pdf() 146
A3.4.2.5 Departure_pdf() 146
A3.4.2.6 IMC_pdf() 146
A3.4.3 tArrival_pre(trans) 147
A3.4.4 loadData() 148
A3.4.5 fillMissedCarTable() 149
A3.4.6 separateColors(the2colors) 149
A3.4.7 COMMON_PRE(trans) 150
A3.4.8 COMMON_POST(trans) 157
A3.5 First come first serve elevator with VIP pass: 158
A3.5.1 Main Simulation File (MSF): 158
A3.5.2 PDF Files: 158
A3.5.2.1 Init_pdf() 158
A3.5.2.2 Arrival_pdf() 159
A3.5.2.3 Car_ata_Level_pdf() 159
A3.5.2.4 Car_Moving_pdf() 159
A3.5.2.5 Departure_pdf() 160
A3.5.2.6 IMC_pdf() 160
A3.5.3 tArrival_pre(trans) 160
A3.5.4 loadData() 162
A3.5.5 fillMissedCarTable() 163
A3.5.6 separateColors(the2colors) 164
A3.5.7 COMMON_PRE(trans) 164
A3.5.8 COMMON_POST(trans) 168

Abstract:

Elevators are an important source of transportation in high rise building to take people across different floors in short amount of time. It helps to reduce time and energy of people. My thesis work is all about elevators. As we know, behind the working of any elevator there is an algorithm which is responsible for the movement of car and handling requests of the passengers, who wants to use it. That's why I have done a comparison between a classical algorithm called Shabbat algorithm and four of my own proposed algorithm, which I have categorized them as modern algorithm to enhance various short coming of classical algorithm like: turnaround time of passengers, capacity of handling more requests in less time etc. For that purpose, I have used Petri Net theory and GPenSIM. I have used them to model and simulate the classical and modern algorithms. For testing and analysis part, I have done sample run of my modern algorithms and made plots out of it, to have a graphical output of them. I have done statistical analysis also which shows turnaround time, ideal travel time, travel time inside car and efficiency of the proposed modern algorithms. According to statistical analysis, First Come First Serve elevator algorithm has less turn around time, less travel time inside car and increased efficiency as compared to Shabbat algorithm.

1. Introduction:

This work is done for master's thesis whose topic is "Modelling, Simulation & Performance Evaluation of Modern Elevators".

An elevator is a machine which moves people across different floors of a building ranging from small to large skyscrapers. It saves our time and energy. Its working mechanism includes a car, an electric motor, a pulley system and a counterweight. As the elevator moves up by the help of an electric motor, the counterweight moves down and vice versa. An elevator is called /requested at a respective floor by pushing a button by a person from outside of the car waiting to go to his destination floor. As it arrives, door opens, and the person will enter the car and touch his desired floor button to reach. The movement of the car and the call requests of the passengers depends on the algorithm which is being used for that elevator.

1.1 Problem Definition:

I have used GPenSIM and Petri NET theory to simulate the process of an elevator which includes five different types of algorithms. I have categorized them in two types i.e. classical and modern algorithms. The classical one is called *Shabbat Algorithm* while Modern algorithms include the following:

- First come first serve elevator algorithm
- Fixed sectoring elevator algorithm
- Dynamic sectoring elevator algorithm
- First come first serve elevator algorithm with VIP passcode

Basically, I am comparing modern algorithms working in elevators with a conventional or classical one, that includes various aspects like turnaround time of passengers, movement of the elevator, number of passenger requests or calls handled in a specific time etc.

Above mentioned parameters play a vital role to gauge the performance of an elevator. Hence, the possible outcomes and finding would be both interesting and useful for real world applications as well.

1.2 Classical & modern elevator algorithms:

1.2.1 Classical algorithm:

1.2.1.1 Shabbat elevator algorithm:

An elevator continues to move in the same direction i.e. upwards or downwards until there are passenger requests in that direction. Once it has exhausted all the request including passengers inside and outside of the car moving in the same direction, then it changes its direction. Otherwise, it just stops at a floor and wait for the next call.

1.2.2 Modern Algorithms:

1.2.2.1 First come first serve elevator algorithm:

In this algorithm, passengers are picked according to the order of their call requests and then dropped to their destination floors in the same order as they were picked from their departure floors.

1.2.2.2 Fixed sectoring elevator algorithm:

In this algorithm, elevator is fixed to serve a specific number of floors and the elevator car moves across different floors using Shabbat movement. We can take a tall building as an example, which has commercial offices at bottom and residential apartments at the top. So, this algorithm makes sure that the elevator is fixed to residential apartment floors and could only be used by them.

1.2.2.3 Dynamic sectoring elevator algorithm:

This algorithm makes sure that the elevator is being used efficiently. Here also elevator is fixed to specific number of floors but based on time of the day. Consider we have a building which has offices and commercial hubs at the bottom floors and residential apartments at the top. Accordingly, elevator would be fixed to the bottom floors containing offices during official working hours and after that it would be at the disposal of persons living in the apartments at the top floors of the building. That's how it makes sure to get the max out of the least.

1.2.2.4 First come first serve elevator with VIP passcode:

This algorithm can be used to restrict the use elevators to some of the authorized personnel only. It can be used by multinational companies to serve their higher hierarchy members to save their time to get to their destination floor on time without any hassle. It reduces the passenger traffic by segregating the ordinary and special persons.

This type of elevator can only be used by an authorized person who has the passcode to operate it. The passenger calls the elevator by pushing the button while standing outside of the car. Once, it arrives, he enters and type the passcode after pushing his destination floor button to reach there. It also works in the first come first serve order and passengers leaves the car in the same order as they had entered it.

1.3 Adaptations:

I am using Shabbat algorithm code, originally developed by Professor Reggie Davidrajuh, in my thesis as classical algorithm to compare it with my proposed modern algorithms.

In my modern algorithms, I have used the model which is being used in the Shabbat algorithm because the basic working of the elevator is same in all the algorithms. That's why I have adopted the pdf files of Shabbat algorithm in my modern algorithms. Also, I have used some of it functions like: *loadData(), separateColors(the2colors), fillMissedCarTable()* and the pre-processor of transition *tArrival* .

2. Method and Design:

2.1 Overall Design:

My model consists of five major modules that includes: Arrival module, Car At a level module, Departure module, Initialization module and Car Moving module. For connecting these modules, we also have inter-module connections which constitutes a working model of an elevator. The overall design of an elevator is shown as follows:

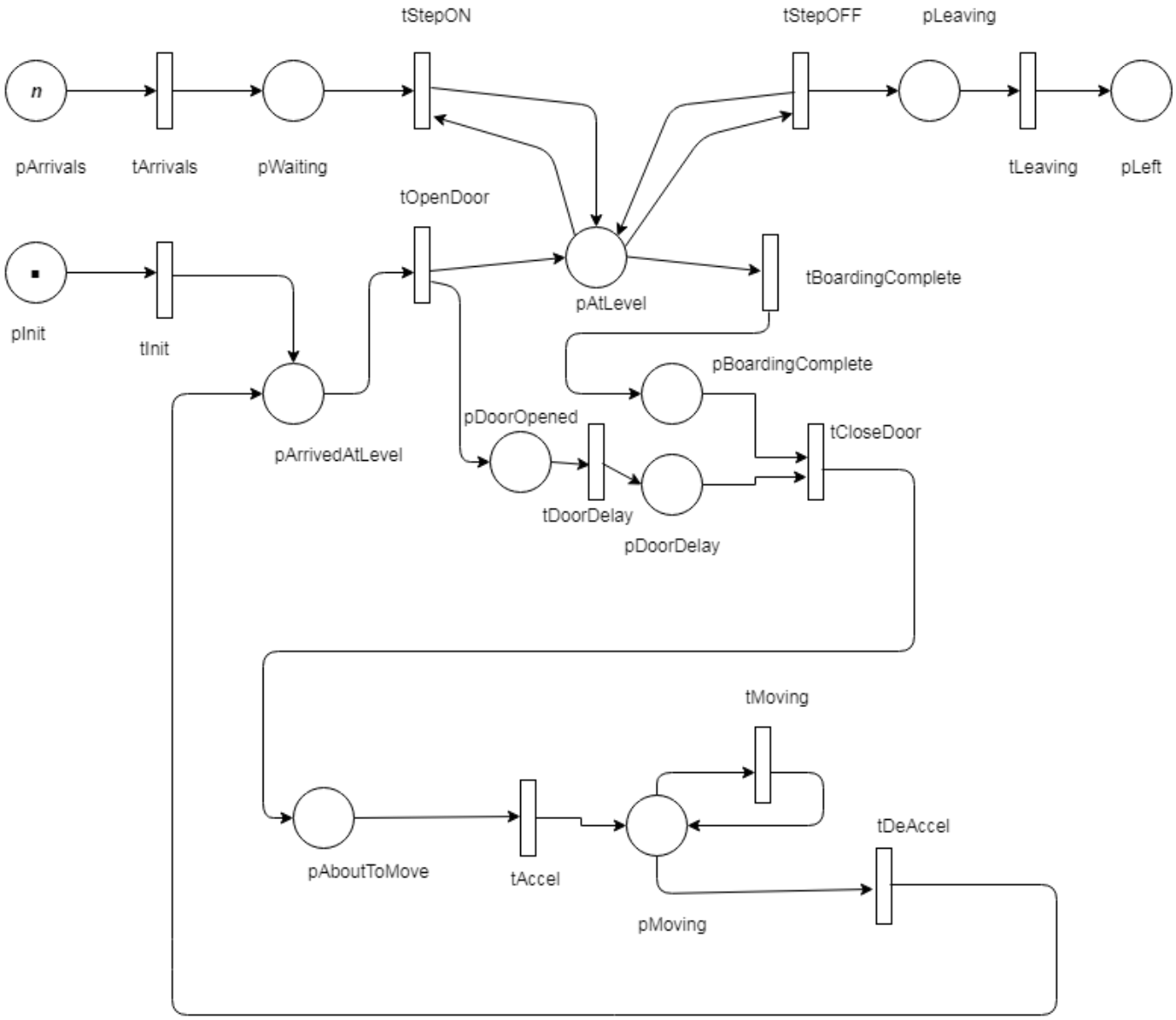

Figure 1: Full Scale Model

2.2 Modular Approach:

Here I will describe the functionality of each above stated module one by one which is as follows:

2.2.1 Arrival Module:

It deals with the arrival of passengers at the given times, described in the *"loadData()"* function in the code. It represents arrival of passengers at different levels waiting outside the elevator to step on to the car and to go to their desired floor accordingly.

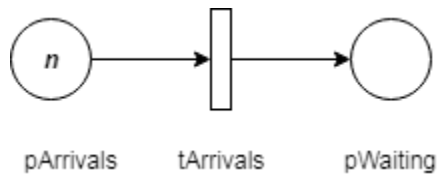

Figure 2: Arrival Module

2.2.2 Initialization Module:

This module initiates the workflow of the elevator. Token inside place *"pInit"* depicts the car of an elevator and make the transition *"tInit"* enabled. After that token reaches to place *"pArrivedAtLevel"*

which is a part of inter-module connections which shows that car has arrived at a specific level, allowing passengers to step on the car after opening the door. That's how it works along with the Arrival module to dispatch passengers to their desired destination floor.

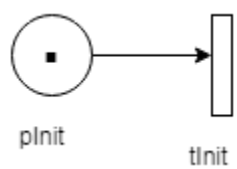

Figure 3: Initialization Module

2.2.3 Car At a Level Module:

This module gets operational when two of its transitions gets enabled which are *"tStepON"* and *"tOpenDoor".* After the elevator arrives at a level, it opens its doors to allow passengers to step on and leave the elevator by using transitions *"tStepON"* and *"tStepOFF".* Once boarding is completed, it closes its door using transitions *"tBoardingComplete"* and *"tCloseDoor".* The elevator, after closing its door, starts to move to next requested floor using "Car Moving " module.

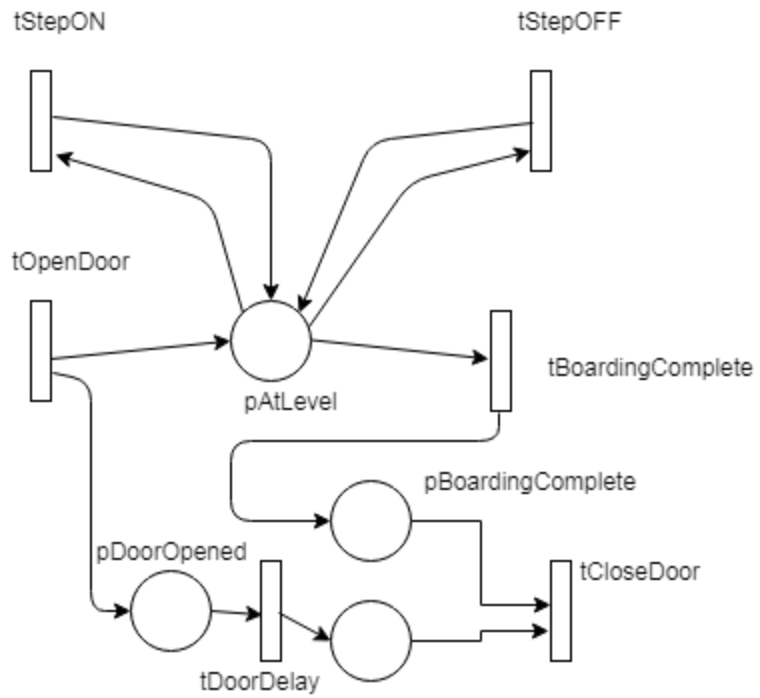

Figure 4: Car At a Level Module

2.2.4 Departure Module:

This module represents passengers who are leaving the elevator using transition *"tStepOff"* from module "Car At a Level" and transition *"tLeaving"* from this very module itself i.e. "Departure module".

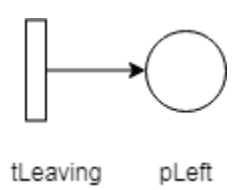

Figure 5: Departure Module

2.2.5 Car Moving Module:

After door is closed, this module is responsible to move the car to its requested floor. Transition "tAccel" is used to accelerate the car and transition "tMoving" shows that the elevator is moving or going towards its requested floor. Transition "tMoving" can be used repeatedly, if the elevator has to move through multiple floors before coming to a halt by using transition "DeAccel".

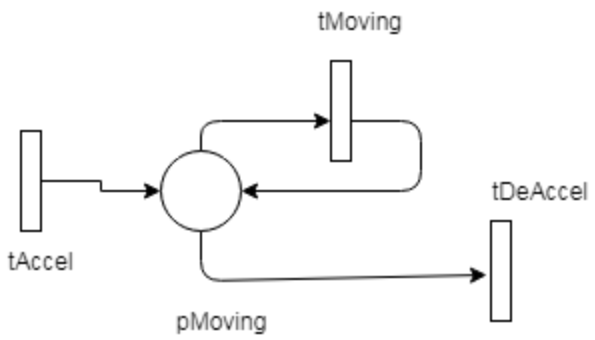

Figure 6: Car Moving Module

2.2.6 Inter module connections:

There are four vital places in the inter module connections which integrates all the modules together and are essential for the workflow of the model namely: "*pwaiting, pLeaving, pArrivedAtLevel and pAboutToMove".*

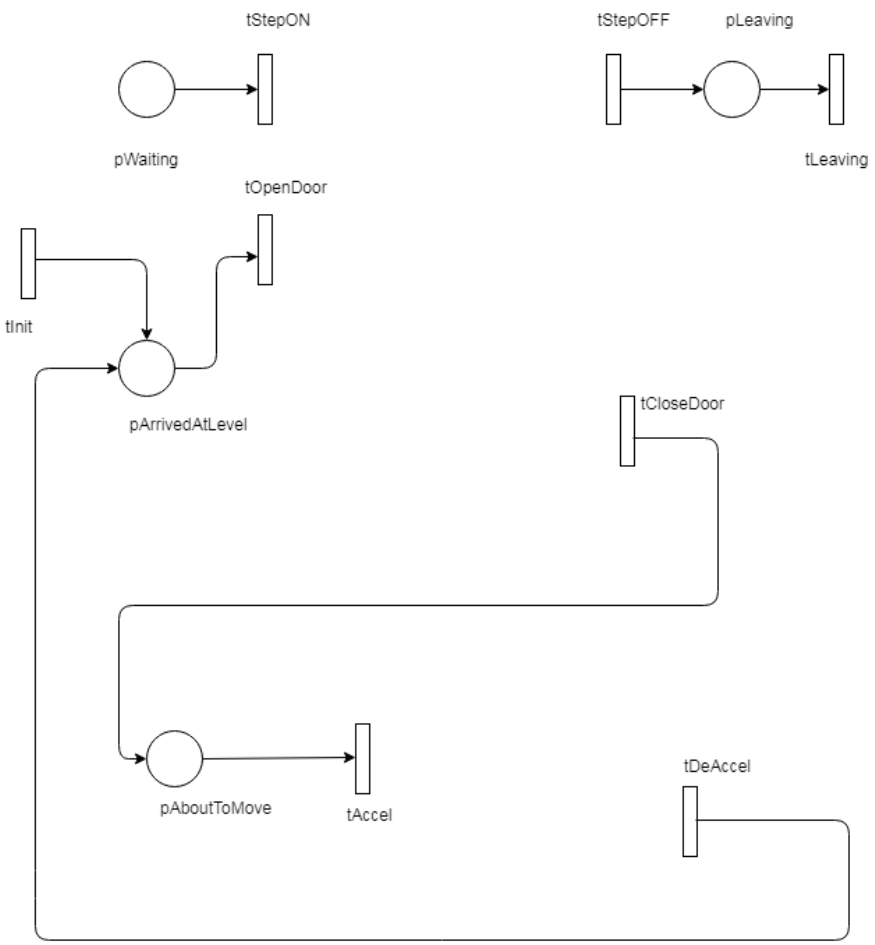

Figure 7: Inter Module Connections

2.3 Technique:

I have used colored petri net technique for simulating the elevator model. Specifically, it is used when passenger arrives at a floor, the passenger/token is assigned two colors that are “Lx” and “Px”. Whereas, Lx represents the level at which the passenger arrives and Px signifies person number that is associated to that passenger/token. In “Lx” and “Px”, x is replaced by corresponding number/digit based upon counter value. For example, a passenger has arrived at level 1, so accordingly a token would get colors as L1 and P1 respectively.

Colored Petri Net is also used when a passenger needs to step on the elevator based on the token color which specifies that which level he arrived and contain the person number also. Level number is used to step on the elevator while person number is used to fill the Passenger Data table columns which stores the time he stepped on and his travelling status that is true or false.

3. Implementation:

Here I will describe the strategies and techniques involved in developing classical and modern algorithms for elevators which were mentioned earlier.

3.1 Shabbat Algorithm:

An elevator continues to move in the same direction i.e. upwards or downwards until there are passenger requests in that direction. Once it has exhausted all the request including passengers inside and outside of the car moving in the same direction, then it changes its direction. Otherwise, it just stops at a floor and wait for the next call.

3.1.1 Coding of modules:

3.1.1.1 Arrival module:

This module consists of two places and a transition namely: *pArrivals,pWaiting and tArrivals.*
Passengers arrive at the given time in loadData() function, present in the code and wait for the elevator to arrive and step on to it. After the arrival/generation of tokens, they wait or reside in place *pWaiting*.

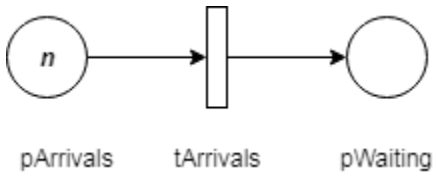

Figure 8: Arrival Module

Following code snippet shows pre-processor of transition tArrival. As the passengers arrive it updates the person counter which saves the number of persons that has arrived yet using variable named as *"global_info.Person_No"*.It also removes the passenger from the arrival list in the loadData() function which contains its arrival time and destination floor.For example if the loadData() function contains a vector where it has two passengers detail like: *global_info.ATL1=[15 1, 20 2],* then it will remove "15 1" from the vector after its arrival. Also, three fields of global_info.PASSENGER_DATA, which is a vector to store various parameters about passengers are filled like arrival time, at which level he arrived and the destination level. Each passenger/token gets two colors as it arrives that is Lx and Px where x corresponds to a digit i.e. L1 and P1. So, its colors are L1 and P1 which means level 1 and person number 1.

```
function [fire, trans] = tArrival_pre(trans)
global global_info

for Level = global_info.MIN_LEVEL:global_info.MAX_LEVEL
    Waiting = eval(['global_info.ATL',int2str(Level+3),';']);
    if not(isempty(Waiting))
        timeOfArrival = Waiting(1);
        if ge(current_time, timeOfArrival)
            % this passenger has come
            global_info.Person_No = global_info.Person_No + 1;
            PNo = global_info.Person_No;
            whereTo = Waiting(2);

            % reduce global_info.ATLx
            eval(['global_info.ATL',int2str(Level+3),'=Waiting(3:end);']);
            % passenger generated time
            global_info.PASSENGER_DATA(PNo, 1) = current_time;
            global_info.PASSENGER_DATA(PNo, 4) = Level; % from which level
            global_info.PASSENGER_DATA(PNo, 5) = whereTo; % to which level

            color1 = ['L', int2str(Level)];
            color2 = ['P', int2str(PNo)];
            trans.new_color = {color1, color2};
            fire = 1;
            disp(['Arrival: Time:', num2str(current_time),...
                '  at level:', color1, '   PNo:', color2]);
            return
        end
    end
end
fire = 0;
```

Code Snippet 1: For Pre-Processor of transition tArrival

loadData() function is related to arrival module because it contains the list of passengers that would arrive at each level which consist of arrival time and destination floor of each passenger that is going to use the elevator. Using this list it calculates the total number of passengers beforehand. It also forms a vector /matrix named as *global_info.PASSENGER_DATA* to keep record of passengers regarding different parameters like person number, arrival time, elevator leaving time, elevator entering time, arrival floor, destination floor etc.
global_info.Travellers_at_Present is also a vector which is used to save number of passengers at any given time to make sure that elevator does not allow to carry passengers more than it capacity.

```
function [] = loadData()
global global_info

global_info.People_in_Car = 0; % to start with, 0 passengers inside car

% Historial data: Arrival of people to the lift, at different levels
% Test-1:
global_info.ATL1 = [20  0]; % level -2
global_info.ATL2 = [10 2]; % level -1
global_info.ATL3 = [30 2];  % level 0
global_info.ATL4 = [50 -2];  % level 1
global_info.ATL5 = [40 -1];  % level 2

passengers = 0;
for Level = global_info.MIN_LEVEL:global_info.MAX_LEVEL
    ATLx = eval(['global_info.ATL',int2str(Level+3),';']);   %%%%%  !!!!!!
    passengers = passengers + length(ATLx)/2;
end
disp(['Total number of passengers: ', int2str(passengers)]);
global_info.TOTAL_USERS = passengers;
global_info.PASSENGER_DATA = zeros(passengers,7);
global_info.Travellers_at_Present = zeros(1, global_info.MAX_CAPACITY);
global_info.Person_No = 0; % Person counter (or person ID number)

for i = 1:global_info.TOTAL_USERS
    eval(['global_info.P2DL_', int2str(i), ' = [];']);
end
```

Code Snippet 2: For loadData() function

3.1.1.2 Initialization module:

Initialization module contain one place and a transition namely *pInit* and *tInit* . Transition *tInit* is related to one of the transitions of *Car At a Level* module in a way that it enables the transition *tOpenDoor* of *Car At a Level* module. In other words, it helps in initiating the workflow of the elevator. The token residing in place *pInit* symbolizes the elevator car and it gets initiated by firing transition *tInit* to reach the place named as *pArrivedAtLevel* which enables the transition *tOpenDoor* of *Car At a Level* module to open the elevator's door to allow passengers to step on it.

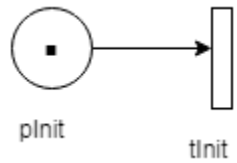

Figure 9: Initialization Module

The following code snippet shows the pre-processor of the transition *tInit* which is fired as it is or without any further condition to initiate the elevator workflow

```
function [fire, trans] = COMMON_PRE(trans)
global global_info

switch trans.name
    case 'tInit', fire = 1;
```

Code Snippet 3: Pre-Processor for transition tInit

3.1.1.3 Car At a Level Module:

It consists of six transitions and four places namely: *tStepON, tStepOFF, tOpenDoor, tBoardingComplete, tDoorDelay, tCloseDoor and pAtLevel, pBoardingComplete, pDoorOpened, pDoorDelay.*

After the arrival of passengers in the form of tokens in the place *pWaiting* and door of the elevator is opened by transition tOpenDoor, the passenger step on it. The passenger completes the boarding procedure by pressing the button of their desired floor to reach, placed inside the elevator car. The elevator door remains open by the set duration of door delay. This happens by executing transition *tDoorDelay*. Door closes after the completion of boarding process and the set time of door delay.

Another case is that a passenger wants to step off the elevator, then after reaching its desired level/floor, door opens, and he leaves the car by using transition *tStepOFF*.

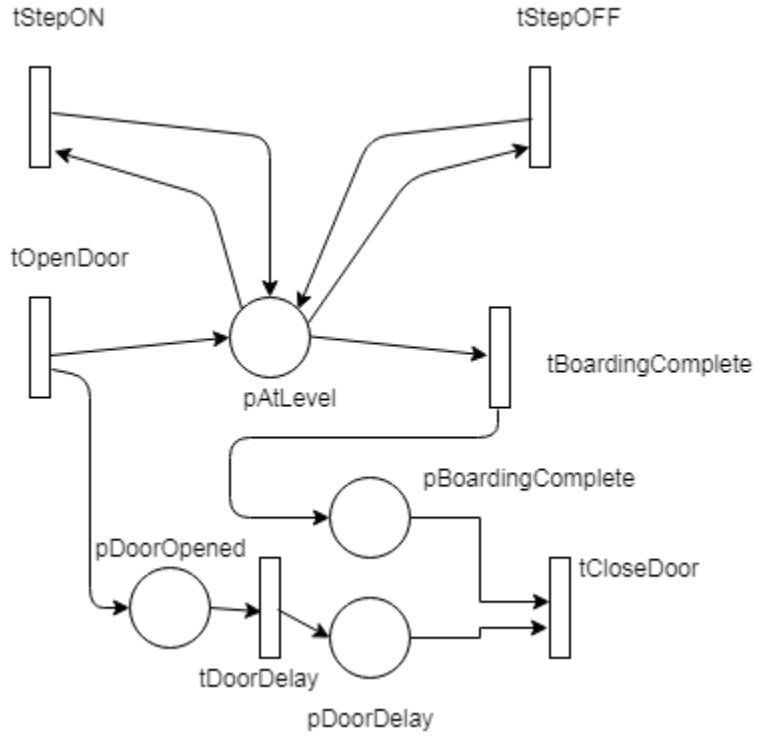

Figure 10: Car At a Level Module

The following code snippet shows pre-processor for transaction *tStepON.* It selects a token from *pWaiting* whose color is equal to current floor level. As we have set maximum passenger capacity of elevator equal to five that's why we check for passenger vacancy in the car. If there is a vacancy then we select the token and allow the transition to fire.Then to get the person number of the selected token, we call the function named as *separateColors(the2colors),* since a token has two colors which shows the floor number at which a passenger arrived and the person number. Also we save the time when passenger enters the car and put its travelling status to *true* in the *global_info.PASSENGER_DATA* matrix.

Otherwise, we change priority of transition *tStepON* from 1 to 0 to prevent it from firing again and again. After that function filledMissedCarTable() is called to execute. This function first find all the passengers whose step on floor number is equal to current floor number.Then check all of the passengers that whether they have already arrived and waiting or not. If a passenger has arrived and waiting then add 1 to the value of 7th column of that passenger's row in *global_info.PASSENGER_DATA* matrix, which signifies that that passenger could not step on the car because it was full to its maximum capacity.

```
case 'tStepON'
    currentLevel = global_info.Current_Level;
    tokID = tokenAnyColor('pWaiting', 1, ...
                {['L',int2str(currentLevel)]});
    if not(tokID) % no one is waiting at this level!!!
        fire = 0; return
    end

    % some people are waiting at this level
    % then check for vacancy
    peopleInCar = sum(global_info.PASSENGER_DATA(:, 6));
    vacant = global_info.MAX_CAPACITY  - peopleInCar;
    if not(vacant)
        disp('** Car is full, cannot take more passengers!');

        % make "tStepON" weaker so that it will not try again & again
        priorset('tStepON', 0);
        fillMissedCarTable();
        fire = 0; return
    end
    trans.selected_tokens = tokID;
    trans.override = 1;
    fire = 1;
    the2colors = get_color('pWaiting', tokID);
    PNo = separateColors(the2colors);
    % the time passenger ENTERS car
    global_info.PASSENGER_DATA(PNo, 2) = current_time;
    % the passenger is inside the car: travelling
    global_info.PASSENGER_DATA(PNo, 6) = true;
    disp(['Entering Car: Time:', num2str(current_time),...
          '  at level:', int2str(currentLevel),...
          '   PNo:', int2str(PNo)]);
```

Code Snippet 4: Pre-Processor for transition tStepON

```
function [PNo] = separateColors(the2colors)

color1 = the2colors{1};
color2 = the2colors{2};

% finding from which Level
if strcmp(color1(1), 'L')
    fromWhichLevel = color1;
    PNo = color2;
else
    fromWhichLevel = color2;
    PNo = color1;
end

PNo = PNo(2:end);
PNo = str2double(PNo);
```

Code Snippet 5: For separateColors(the2colors) function

```
function [] = fillMissedCarTable()
global global_info

[indices_for_this_level, ~] = ...
    find(global_info.PASSENGER_DATA(:,4)== global_info.Current_Level);

for i = 1:numel(indices_for_this_level)
    row_index = indices_for_this_level(i);

    if not(global_info.PASSENGER_DATA(row_index, 2))
        % arrived and waiting !!!
        global_info.PASSENGER_DATA(row_index,7) = ...
            global_info.PASSENGER_DATA(row_index,7) + 1;
    end
end
```

Code Snippet 6: For fillMissedCarTable() function

The following code snippet shows pre-processor for transition *tStepOFF.* First it gets sum of all the passengers travelling inside the car. If the car is empty it will not fire. Otherwise, it will compare the destination level of all the passengers present in the car one by one with the current floor level. If there is a match, then that passenger leaves the car. Also, the value of sixth column of

PASSENGER_DATA matrix which shows travelling status of a passenger inside the car is set to zero that is false, of the passenger who leaves.

```
    case 'tStepOFF'
        % people currently inside car (travelling)
        current_passengers = global_info.PASSENGER_DATA(:, 6);
        peopleInCar = sum(current_passengers);
        if not(peopleInCar), fire = 0; return, end % car is empty
        [~, PNoInCar] = find(transpose(current_passengers));
        for i = 1:numel(PNoInCar)
            PNo = PNoInCar(i);
            whereTo = global_info.PASSENGER_DATA(PNo, 5);
            if eq(global_info.Current_Level, whereTo)

                % this guy is leaving
                global_info.PASSENGER_DATA(PNo, 6) = 0;
                % leaving time from CAR
                global_info.PASSENGER_DATA(PNo, 3) = current_time;
                disp(['Leaving Car: Time:', num2str(current_time),...
                    '  at level:', int2str(whereTo),...
                    '   PNo:', int2str(PNo)]);
                fire = 1; return
            end
        end
        fire = 0;
```

Code Snippet 7: Pre-Processor for transition tStepOFF

The following code snippet shows pre-processor for transition *tOpenDoor* which is fired without any further condition.

```
    case 'tOpenDoor', fire = 1;
```

Code Snippet 8: Pre-Processor for transition tOpenDoor

The following code snippet shows post-processor for transition *tOpenDoor* which displays a message on console after reaching at a floor and opening the elevator's door.

```
    case 'tOpenDoor'
        disp(' ');
        disp(['**** Arrived at Level ',...
              int2str(global_info.Current_Level), ...
              '     at time: ', num2str(current_time), ' *******']);
```

Code Snippet 9: Post-Processor for transition tOpenDoor

The following code snippet shows pre-processor for transition *tBoardingComplete* in which we are setting transition priority of *tStepON* back to 1 from 0 using *priorset*() function and allowing the transition to fire.

```
case 'tBoardingComplete'
    % reset priority of "tStepON," if it was set to 0
    priorset('tStepON', 1);
    fire = 1;
```

Code Snippet 10: Pre-Processor for transition tBoardingComplete

The following code snippet shows pre-processor for transition *tDoorDelay* in which we are overriding the transition and allowing it to fire.

```
case 'tDoorDelay', trans.override = 1; fire = 1;
```

Code Snippet 11: Pre-Processor for transition tDoorDelay

The following code snippet shows pre-processor for transition *tCloseDoor* which displays a console message that the door is getting closed at current level and current time and allowing the transition to fire.

```
case 'tCloseDoor'
    disp(['**** Closing the door at Level ',...
        int2str(global_info.Current_Level), ...
        '     at time: ', num2str(current_time), ' *******']);
    disp(' ');
    fire = 1;
```

Code Snippet 12: Pre-Processor for transition tCloseDoor

3.1.1.4 Departure module:

This module consists of a transition and a place namely: *tLeaving* and *pLeft.* After transition *tStepOFF* at *Car At a Level* module to exit the elevator, a passenger leaves using transition *tLeaving.*

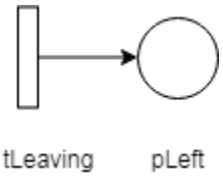

Figure 11: Departure Module

The following code snippet shows pre-processor for transition *tLeaving* which is getting fired without any further condition.

```
case 'tLeaving', fire = 1;
```

Code Snippet 13: Pre-Processor for transition tLeaving

The following code snippet shows post-processor for transition *tLeaving* which makes sure that simulation does not stop until all passengers have reached their destination floors by comparing number of tokens at place "*pLeft*" with total number of users/passengers that is *global_info.TOTAL_USERS.*

```
case 'tLeaving'
    n = ntokens('pLeft');
    if eq(n, global_info.TOTAL_USERS)
        global_info.STOP_SIMULATION = true;
    end
```

Code Snippet 14: Post-Processor for transition tLeaving

3.1.1.5 Car Moving module:

This module contains three transitions and a place namely: *tAccel, tMoving, tDeAccel* and *pMoving*. After closing door at Car At a Level module, elevator accelerates using transition *tAccel* to take passengers to their destination floors. Transition *tMoving* fire only once and moves the car by one level only because in Shabbat algorithm, the elevator stops at every floor.

Finally, after reaching the destination floor it comes to a halt by using transition *tDeAccel.* Otherwise, if the next floor is the destination floor then it uses *tMoving* once and stops using *tDeAccel.*

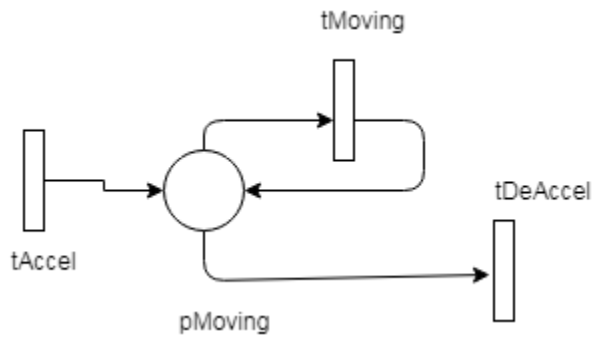

Figure 12: Car Moving Module

The following code snippet shows code of pre-processor of transition *tAccel*. It implements the actual shabbat algorithm regarding the movement of elevator's car. According to which, if current direction of the car is upwards then it adds 1 to the value of current level and further it checks whether current level value is greater than maximum level value, if yes then it subtracts 2 from the current level value and change the elevator's direction to downwards.
Otherwise, if the elevator's current direction is towards downwards then it subtracts 1 from the value of current level and further it checks whether current level value is smaller than minimum level value, if yes then it adds 2 to the current level value and change the elevator's direction to upwards.

```
case 'tAccel' % the main algorithm for moving car to destination !!!
    % simple Sabbat Movement
    if strcmp(global_info.Current_Direction, 'UP')
        global_info.Current_Level = global_info.Current_Level + 1;
        if gt(global_info.Current_Level, global_info.MAX_LEVEL)
            global_info.Current_Level = global_info.Current_Level - 2;
            global_info.Current_Direction = 'DOWN';
        end
    else % global_info.Current_Direction == 'DOWN')
        global_info.Current_Level = global_info.Current_Level - 1;
        if lt(global_info.Current_Level, global_info.MIN_LEVEL)
            global_info.Current_Level = ...
                    global_info.Current_Level + 2;
            global_info.Current_Direction = 'UP';
        end
    end

    global_info.Starting_to_move = true; % used by "tMoving"
    fire = 1;
```

Code Snippet 15: Pre-Processor for transition tAccel

The following code snippet shows code of pre-processor of transition *tDeAccel* which is fired without any further conditions.

```
case 'tDeAccel'
    fire = 1;
```

Code Snippet 16: Pre-Processor for transition tDeAccel

The following code snippet shows code of pre-processor of transition *tMoving* where it is allowed to fire when the bool value of *global_info.Starting_to_move* is set to *true, previously in tAccel. After that we set it to false* because elevator is allowed to move only once in Shabbat algorithm i.e. movement by one floor only. Otherwise, it's not allowed to fire if previously in *tAccel* the value of *global_info.Starting_to_move* is not set to true.

```
% moves car by one level; tMoving fires only once for shabbat elevator
case 'tMoving'
    if global_info.Starting_to_move
        global_info.Starting_to_move = false;
        fire = 1;
    else
        fire = 0;
    end
```

Code Snippet 17: Pre-Processor for transition tMoving

3.1.1.6 Inter modular connections:

Following diagram shows the inter module connections which integrates all modules to make a single model of an elevator. There are three prominent places in it which helps to connects all the modules together i.e. *pArrivedAtLevel*, *pAboutToMove* and *pLeaving*.

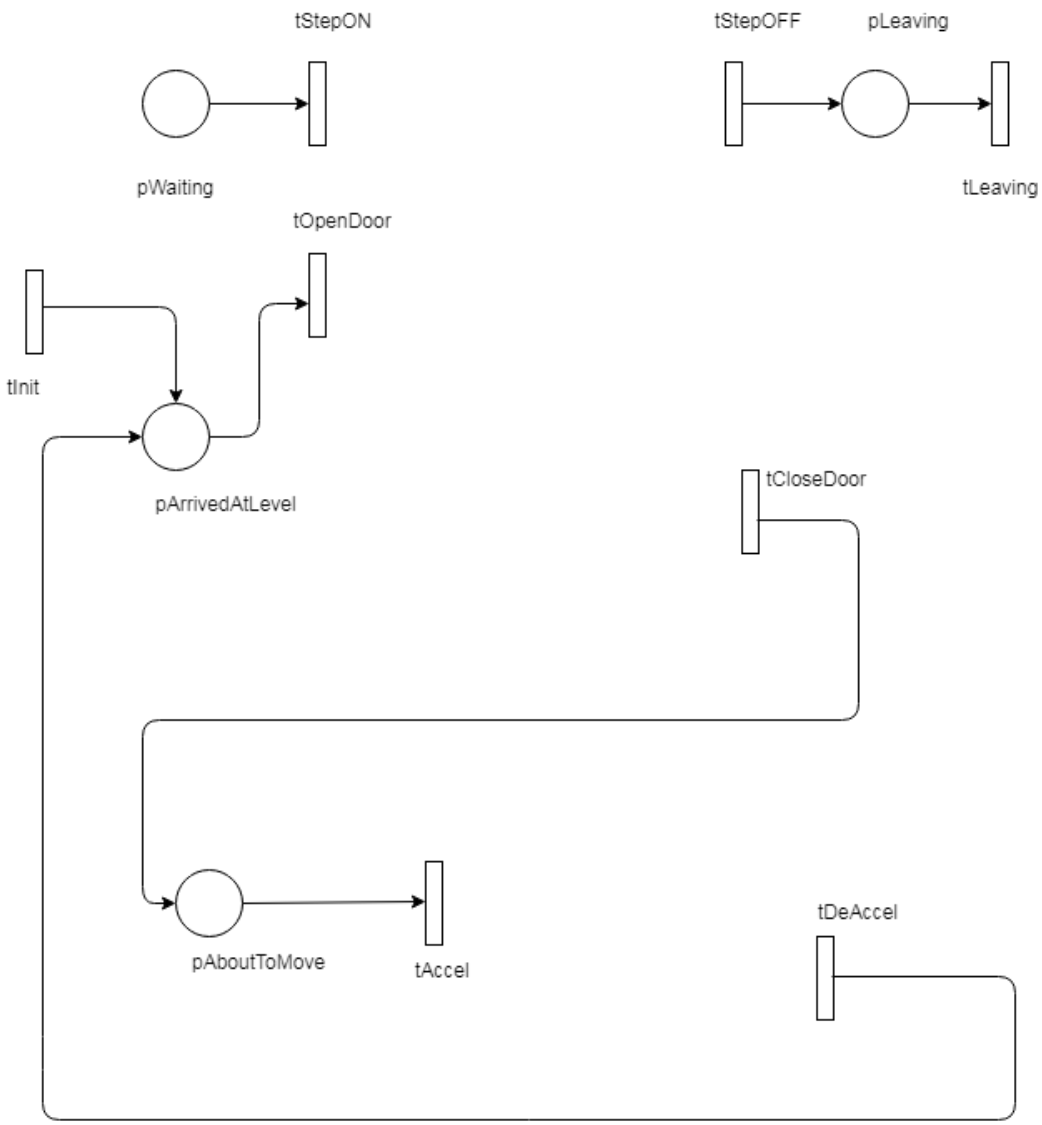

Figure 13: Inter Module Connections

3.2 First come first serve elevator algorithm:

In this algorithm, passengers are picked up according to the order of their call requests or according to the order in which they have arrived at their departure floors and then dropped to their destination floors in the same order as they were picked up from their departure floors.

3.2.1 Coding of modules:

3.2.1.1 Arrival module:

This module consists of two places and a transition namely: *pArrivals,pWaiting and tArrivals.*
Passengers arrive at the given time in *loadData()* function, present in the code and wait for the elevator to arrive and step on to it. After the arrival/generation of tokens, they wait or reside in place *pWaiting*.

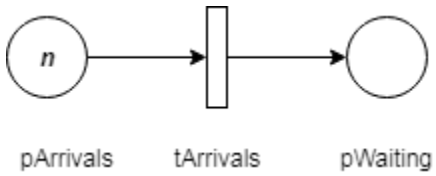

Figure 14: Arrival Module

Following code snippet shows pre-processor of transition *tArrival*. As the passengers arrive it updates the person counter which saves the number of persons that has arrived yet using variable named as *"global_info.Person_No"*.It also removes the passenger from the arrival list in the *loadData()* function which contains its arrival time and destination floor.For example if the *loadData()* function contains a vector where it has two passengers detail like: *global_info.ATL1=[15 1, 20 2],* then it will remove "15 1" from the vector after its arrival. Also, three fields of *global_info.PASSENGER_DATA*, which is a vector to store various parameters about passengers are filled like arrival time, at which level he arrived and the destination level. Each passenger/token gets two colors as it arrives that is Lx and Px where x corresponds to a digit i.e. L1 and P1. So, its colors are L1 and P1 which means level 1 and person number 1.

```
%Pre-Processor code for tArrival
function [fire, trans] = tArrival_pre(trans)
global global_info
%will run from minimum to maximum level
for Level = global_info.MIN_LEVEL:global_info.MAX_LEVEL
    %get the value of 'global_info.ATL' e.g global_info.ATL1 frm loadData()
    %function
    Waiting = eval(['global_info.ATL',int2str(Level+3),';']);
    if not(isempty(Waiting))
        %if not empty then get the value
        %present at first index which is passenger's arrival time
        timeOfArrival = Waiting(1);
         % if current time is greater than passenger's arrival time than
         %this is the time for passenger to arrive
        if ge(current_time, timeOfArrival)

            %update the person counter
            global_info.Person_No = global_info.Person_No + 1;
            PNo = global_info.Person_No;
            %second index value shows the destination floor of passenger
            whereTo = Waiting(2);

            % reduce the 1st and 2nd index value from global_info.ATLx
            eval(['global_info.ATL',int2str(Level+3),'=Waiting(3:end);']);
            % time at which passenger arrived
            global_info.PASSENGER_DATA(PNo, 1) = current_time;
            % At which level passenger arrived
            global_info.PASSENGER_DATA(PNo, 4) = Level;
            % To which level passenger wants to go
            global_info.PASSENGER_DATA(PNo, 5) = whereTo;
            %two colors are assigned to each token/passenger
            color1 = ['L', int2str(Level)];
            color2 = ['P', int2str(PNo)];
            trans.new_color = {color1, color2};
            fire = 1;
            disp(['Passenger Arrival: Time:', num2str(current_time),...
                '  at level:', color1, '   Person No:', color2]);
            return
        end
    end
end
fire = 0;
```

Code Snippet 18: For Pre-Processor of transition tArrival

The following code snippet shows Post-Processor for transition "*tArrival*" in which we are getting the list of arrival floors at which all passengers arrived as *global_info.PickLevel* and the list of floors at which passengers want to reach as *global_info.DropLevel.* After that we append both of these lists to make an aggregate list of floors which would be the iterinary of floors for the movement of the car, later on it would be used by the pre-processor of transition "tAccel" i.e. *global_info.Iterinary* to reflect the true nature of first come first serve approach.

```
case 'tArrival'
    %Contains arrival floors of all the passengers
     global_info.PickLevel = global_info.PASSENGER_DATA(:, 4) ; % from which level
      %Contains departure floors of all the passengers
     global_info.DropLevel = global_info.PASSENGER_DATA(:, 5)  ; % to which level
     %makes up a final iterinary of elevator by appending both
     %global_info.PickLevel & global_info.DropLevel
     global_info.Iterinary = [global_info.PickLevel; global_info.DropLevel ] ;
```

Code Snippet 19: For Post-Processor of transition tArrival

loadData() function is related to arrival module because it contains the list of passengers that would arrive at each level which consist of arrival time and destination floor of each passenger that is going to use the elevator. Using this list, it calculates the total number of passengers beforehand. It also forms a vector /matrix named as *global_info.PASSENGER_DATA* to keep record of passengers regarding different parameters like person number, arrival time, elevator leaving time, elevator entering time, arrival floor, destination floor etc.

global_info.Travellers_at_Present is also a vector which is used to save number of passengers at any given time to make sure that elevator does not allow to carry passengers more than it capacity.

```
function [] = loadData()
global global_info

global_info.People_in_Car = 0; % initially we have 0 passengers inside car

% Passenger's Historial data: Arrival time of passengers to the lift at
% different levels. Also contain destination floor of passengers
% Test-1:
global_info.ATL1 = [20  0]; % level -2
global_info.ATL2 = [10 2]; % level -1
global_info.ATL3 = [28 2];  % level 0
global_info.ATL4 = [15 -2];  % level 1
global_info.ATL5 = [22 -1];  % level 2

passengers = 0;
%for calculating number of passengers according to historical data
for Level = global_info.MIN_LEVEL:global_info.MAX_LEVEL
    ATLx = eval(['global_info.ATL',int2str(Level+3),';']);
    passengers = passengers + length(ATLx)/2;
end
disp(['Total number of passengers: ', int2str(passengers)]);
global_info.TOTAL_USERS = passengers;
%a matrix/ vector to hold information about passenger like person no,
%arrival and departure time, arrival and departure floor, car travelling
%status,missed car status
global_info.PASSENGER_DATA = zeros(passengers,7);
global_info.Travellers_at_Present = zeros(1, global_info.MAX_CAPACITY);
global_info.Person_No = 0; % Person counter (or person ID number)

for i = 1:global_info.TOTAL_USERS
    eval(['global_info.P2DL_', int2str(i), ' = [];']);
end
```

Code Snippet 20: For loadData() function

3.2.1.2 Initialization module:

Initialization module contain one place and a transition namely *pInit* and *tInit* . Transition *tInit* is related to one of the transitions of *Car At a Level* module in a way that it enables the transition *tOpenDoor* of *Car At a Level* module. In other words, it helps in initiating the workflow of the elevator. The token residing in place *pInit* symbolizes the elevator car and it gets initiated by firing transition *tInit* to reach the place named as *pArrivedAtLevel* which enables the transition *tOpenDoor* of *Car At a Level* module to open the elevator's door to allow passengers to step on it.

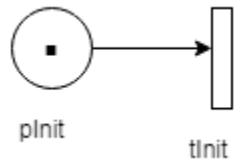

Figure 15: Initialization Module

The following code snippet shows the pre-processor of the transition *tInit* which is fired as it is or without any further condition to initiate the elevator workflow

```
function [fire, trans] = COMMON_PRE(trans)
global global_info

switch trans.name
    case 'tInit', fire = 1;
```

Code Snippet 21: Pre-Processor for transition tInit

3.2.1.3 Car At a Level Module:

It consists of six transitions and four places namely: *tStepON, tStepOFF, tOpenDoor, tBoardingComplete, tDoorDelay, tCloseDoor and pAtLevel, pBoardingComplete, pDoorOpened, pDoorDelay.*

After the arrival of passengers in the form of tokens in the place *pWaiting* and door of the elevator is opened by transition *tOpenDoor*, the passenger step on it. The passenger completes the boarding procedure by pressing the button of their desired floor to reach, placed inside the elevator car. The elevator door remains open by the set duration of door delay. This happens by executing transition *tDoorDelay*. Door closes after the completion of boarding process and the set time of door delay.

Another case is that a passenger wants to step off the elevator, then after reaching its desired level/floor, door opens, and he leaves the car by using transition *tStepOFF*.

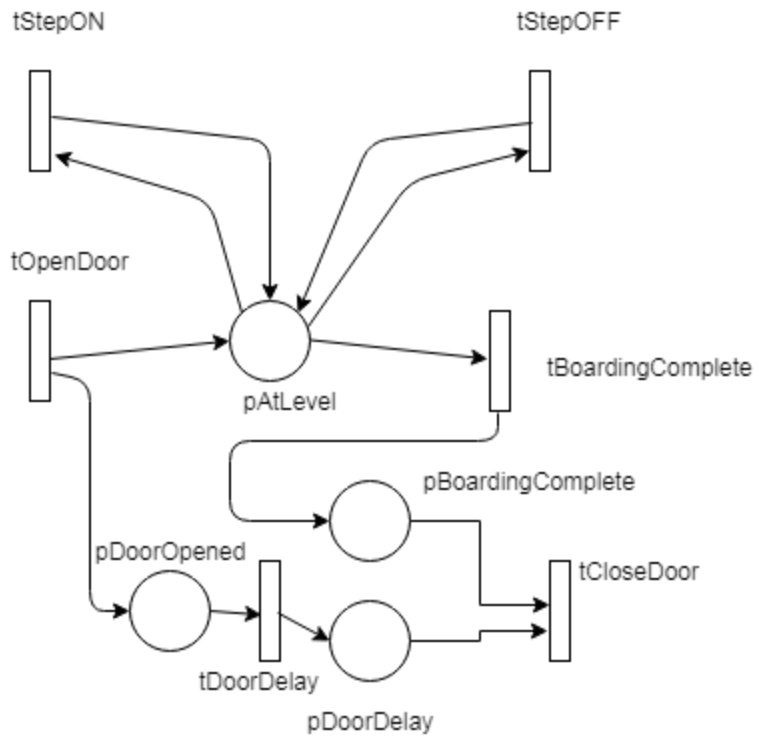

Figure 16: Car At a Level Module

The following code snippet shows pre-processor for transaction *tStepON.* It selects a token from *pWaiting* whose color which contain the arrival level is equal to current floor level. As we have set maximum passenger capacity of elevator equal to five that's why we check for passenger vacancy in the car. If there is a vacancy then we select the token and allow the transition to fire.Then to get the person number of the selected token, we call the function named as *separateColors(the2colors),* since a token has two colors which shows the floor number at which a passenger arrived and the person number. Also we save the time when passenger enters the car and put its travelling status to *true* in the *global_info.PASSENGER_DATA* matrix.

Otherwise, we change priority of transition tStepON from 1 to 0 to prevent it from firing again and again. After that function filledMissedCarTable() is called to execute. This function first find all the passengers whose step on floor number is equal to current floor number.Then check all of the passengers that whether they have already arrived and waiting or not. If a passenger has arrived and waiting then add 1 to the value of 7th column of that passenger's row in *global_info.PASSENGER_DATA* matrix, which signifies that that passenger could not step on the car because it was full to its maximum capacity.

```
case 'tStepON'
     currentLevel = global_info.Current_Level;
     %selecting the token whose arrival level is equal to current level
    tokID = tokenAnyColor('pWaiting', 1, ...
                {['L',int2str(currentLevel)]});
    if not(tokID) % if there is no one waiting at this level, then do not fire
        fire = 0; return
    end

    % Otherwise,check for vacancy in the car, if some people are waiting at this level

    peopleInCar = sum(global_info.PASSENGER_DATA(:, 6));
    vacant = global_info.MAX_CAPACITY  - peopleInCar;
    if not(vacant)
        disp('** Caution! cannot take more passengers! Car is full');

        % reducing priority of "tStepON" to make it weak so that it will not try again & again
        priorset('tStepON', 0);
        fillMissedCarTable(); %holds the persons who could not enter the car because of no vacancy
        fire = 0; return
    end
    trans.selected_tokens = tokID;
    trans.override = 1;
    fire = 1;
    %using separateColors(the2colors) to get the person number to record
    %car entering time and to put travelling stauts=true, of that
    %particular person
    the2colors = get_color('pWaiting', tokID);
    PNo = separateColors(the2colors);

    % the time passenger ENTERS car
    global_info.PASSENGER_DATA(PNo, 2) = current_time;
    % the passenger is inside the car: travelling
    global_info.PASSENGER_DATA(PNo, 6) = true;
    disp(['Passenger is Entering Car: Time:', num2str(current_time),...
          '  at level:', int2str(currentLevel),...
          '   Person No:', int2str(PNo)]);
```

Code Snippet 22: Pre-Processor for transition tStepON

```
function [PNo] = separateColors(the2colors)

color1 = the2colors{1};
color2 = the2colors{2};

% finding from which Level
if strcmp(color1(1), 'L')
    fromWhichLevel = color1;
    PNo = color2;
else
    fromWhichLevel = color2;
    PNo = color1;
end

PNo = PNo(2:end);
PNo = str2double(PNo);
```

Code Snippet 23: For separateColors(the2colors) function

```
function [] = fillMissedCarTable()
global global_info

[indices_for_this_level, ~] = ...
    find(global_info.PASSENGER_DATA(:,4)== global_info.Current_Level);

for i = 1:numel(indices_for_this_level)
    row_index = indices_for_this_level(i);

    if not(global_info.PASSENGER_DATA(row_index, 2))
        % arrived and waiting !!!
        global_info.PASSENGER_DATA(row_index,7) = ...
            global_info.PASSENGER_DATA(row_index,7) + 1;
    end
end
```

Code Snippet 24: For fillMissedCarTable() function

The following code snippet shows pre-processor for transition *tStepOFF.* First it gets sum of all the passengers travelling inside the car. If the car is empty it will not fire. Otherwise, it will compare the destination level of all the passengers present in the car one by one with the current floor level. If there is a match, then that passenger leaves the car. Also, the value of sixth column of PASSENGER_DATA matrix which shows travelling status of a passenger inside the car is set to zero that is false, of the passenger who leaves.

```
case 'tStepOFF'
    % it shows people who are currently inside car, means travelling
    current_passengers = global_info.PASSENGER_DATA(:, 6);
    peopleInCar = sum(current_passengers);
    if not(peopleInCar), fire = 0; return, end % not allowed to fire if the car is empty
    [~, PNoInCar] = find(transpose(current_passengers));
    for i = 1:numel(PNoInCar)
        PNo = PNoInCar(i);
        whereTo = global_info.PASSENGER_DATA(PNo, 5);
        %if current level is equal to destination level of specified person,then that person leaves
        if eq(global_info.Current_Level, whereTo)

            % set travelling in the car status of that person equal to
            % 0, as he is leaving
            global_info.PASSENGER_DATA(PNo, 6) = 0;
            % saving the leaving time from CAR
            global_info.PASSENGER_DATA(PNo, 3) = current_time;
            disp(['Passenger is Leaving Car: Time:', num2str(current_time),...
                '  at level:', int2str(whereTo),...
                '   Person No:', int2str(PNo)]);

            fire = 1; return
        end
    end
    fire = 0;
```

Code Snippet 25: Pre-Processor for transition tStepOFF

The following code snippet shows pre-processor for transition *tOpenDoor* which is fired without any further condition.

```
case 'tOpenDoor', fire = 1;
```

Code Snippet 26: Pre-Processor for transition tOpenDoor

The following code snippet shows post-processor for transition *tOpenDoor* which displays a message on console after reaching at a floor and opening the elevator's door.

```
case 'tOpenDoor'
    disp(' ');
    disp(['**** Elevator Arrived at Level ',...
          int2str(global_info.Current_Level), ...
          '      at time: ', num2str(current_time), ' *******']);
```

Code Snippet 27: Post-Processor for transition tOpenDoor

The following code snippet shows pre-processor for transition *tBoardingComplete* in which we are setting transition priority of *tStepON* back to 1 from 0 using *priorset*() function and allowing the transition to fire.

```
case 'tBoardingComplete'
    % increasing  priority of "tStepON" back to 1 as it was set to
    %0 previously in "tStepON"
    priorset('tStepON', 1);
    fire = 1;
```

Code Snippet 28: Pre-Processor for transition tBoardingComplete

The following code snippet shows pre-processor for transition *tDoorDelay* in which we are overriding the transition and allowing it to fire.

```
case 'tDoorDelay', trans.override = 1; fire = 1;
```

Code Snippet 29: Pre-Processor for transition tDoorDelay

The following code snippet shows pre-processor for transition *tCloseDoor* which displays a console message that the door is getting closed at current level and current time and allowing the transition to fire.

```
case 'tCloseDoor'
    disp(['**** Closing the door now at Level ',...
        int2str(global_info.Current_Level), ...
        '     at time: ', num2str(current_time), ' *******']);
    disp(' ');
    fire = 1;
```

Code Snippet 30: Pre-Processor for transition tCloseDoor

3.2.1.4 Departure module:

This module consists of a transition and a place namely: *tLeaving* and *pLeft.* After transition *tStepOFF* at *Car At a Level* module to exit the elevator, a passenger leaves using transition *tLeaving.*

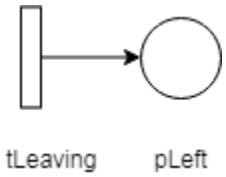

Figure 17: Departure Module

The following code snippet shows pre-processor for transition *tLeaving* which is getting fired without any further condition.

```
case 'tLeaving', fire = 1;
```

Code Snippet 31: Pre-Processor for transition tLeaving

The following code snippet shows post-processor for transition *tLeaving* which makes sure that simulation does not stop until all passengers have reached their destination floors by comparing number of tokens at place "*pLeft*" with total number of users/passengers that is *global_info.TOTAL_USERS.*

```
case 'tLeaving'
    n = ntokens('pLeft');
    %don't stop the simulation untill all the passengers have
    %reached their destination floors
    if eq(n, global_info.TOTAL_USERS)
        global_info.STOP_SIMULATION = true;
    end
```

Code Snippet 32: Post-Processor for transition tLeaving

3.2.1.5 Car Moving module:

This module contains three transitions and a place namely: *tAccel, tMoving, tDeAccel* and *pMoving*. After closing door at Car At a Level module, elevator accelerates using transition *tAccel* to take passengers to their destination floors. Transition *tMoving* fires until the car reaches to its required floor because in first come first serve elevator algorithm car moves according to the fixed itinerary which is formed in the post-processor of transition *tArrival* after the arrival of passengers.
Finally, after reaching the destination floor it comes to a halt by using transition *tDeAccel.*

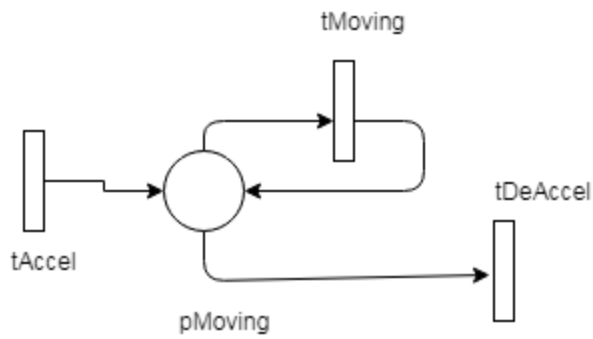

Figure 18: Car Moving Module

The following code snippet shows code of pre-processor of transition *tAccel*. It implements the actual first come first serve algorithm regarding the movement of elevator's car. According to which, the car follows the fixed itinerary to move, which is created in the post processor of *tArrival* after the arrival of passengers. In other words, car will pick up all the passengers in the order in which they have requested for the elevator and deliver the passengers to their required floor in the same order as they requested the elevator. It is done by selecting the first index value of *global_info.Itinerary(1)* as the current level and move the car to that level by setting the value of *global_info.Starting_to_move =true.*

```
case 'tAccel' % It is the main strategy for moving car to destination floors!!!

   %the car moves according to the set Iterinary in the Post-Processor
   %of "tArrival"
   %it selects first index value of global_info.Iterinary(1) as the next floor to reach there
     global_info.Current_Level =  global_info.Iterinary(1)  ;


    global_info.Starting_to_move = true; % It is used by "tMoving"
    fire = 1;
```

Code Snippet 33: Pre-Processor for transition tAccel

The following code snippet shows code of post-processor of transition *tAccel*. According to which after accelerating the car towards the floor mentioned as the first index value of *global_info.Itinerary*, we are removing that value and assigning rest of the values to it as shown below.

```
case 'tAccel'
    %getting the values of global_info.Iterinary
    Move= eval('global_info.Iterinary');
    %removing the 1st index value and assigning the rest of the values to global_info.Iterinary
    eval(['global_info.Iterinary','=Move(2:end);']);
```

Code Snippet 34: Post-Processor for transition tAccel

The following code snippet shows code of pre-processor of transition *tDeAccel* which is fired without any further conditions.

```
    case 'tDeAccel'
        fire = 1;
```

Code Snippet 35: Pre-Processor for transition tDeAccel

The following code snippet shows code of pre-processor of transition *tMoving* which moves the car to the floor as per defined by the 1st index value of *global_info.Itinerary(1).* Also it fires, if the value of *global_info.Starting_to_move* is set to true previously in *tAccel*. After that we set *global_info.Starting_to_move* to false.

```
%it moves the car to the floor as per defined by the value of global_info.Iterinary(1)
   case 'tMoving'
       if global_info.Starting_to_move
           global_info.Starting_to_move = false;
           fire = 1;
       else
           fire = 0;
       end
```

Code Snippet 36: Pre-Processor for transition tMoving

3.2.1.6 Inter modular connections:

Following diagram shows the inter module connections which integrates all modules to make a single model of an elevator. There are three prominent places in it which helps to connects all the modules together i.e. *pArrivedAtLevel*, *pAboutToMove* and *pLeaving*.

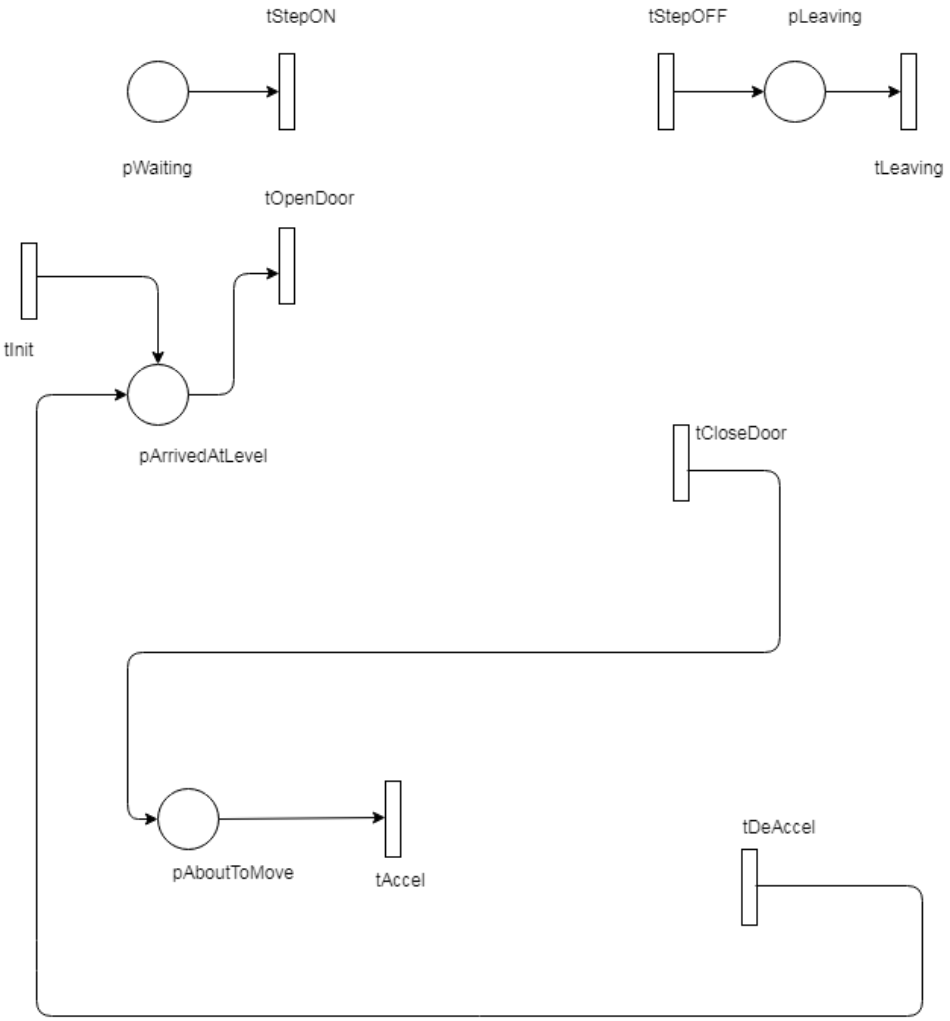

Figure 19: Inter Module Connections

3.3 First come first serve elevator with VIP passcode:

This algorithm can be used to restrict the use elevators to some of the authorized personnel only. It can be used by multinational companies to serve their higher hierarchy members to save their time to get to their destination floor on time without ant hassle. It reduces the passenger traffic by segregating the ordinary and special persons.

This type of elevator can only be used by an authorized person who has the passcode to operate it. The passenger calls the elevator by pushing the button while standing outside of the car. Once, it arrives, he enters and type the passcode after pushing his destination floor button to reach there. It also works in the first come first serve order and passengers leaves the car in the same order as they had entered it.

3.3.1 Coding of modules:

3.3.1.1 Arrival module:

This module consists of two places and a transition namely: *pArrivals,pWaiting and tArrivals.*
Passengers arrive at the given time in *loadData()* function, present in the code and wait for the elevator to arrive but only those can step on the elevator who have the passcode to travel to their desired floor. After the arrival/generation of tokens, they wait or reside in place *pWaiting*.

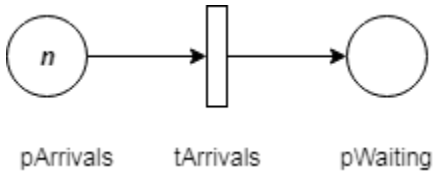

Figure 20: Arrival Module

Following code snippet shows pre-processor of transition *tArrival*. As the passengers arrive it updates the person counter which saves the number of persons that has arrived yet using variable named as *"global_info.Person_No"*.It also removes the passenger from the arrival list in the *loadData()* function which contains its arrival time, destination floor and the passcode holding status .i.e. 0 or1 .For example if the *loadData()* function contains a vector where it has two passengers detail like: *global_info.ATL1=[15 1 1, 20 2 0],* then it will remove "15 1 1" from the vector after its arrival. Also, four fields of *global_info.PASSENGER_DATA*, which is a vector to store various parameters about passengers are filled like arrival time, at which level he arrived, the destination level and passcode holding status. Each passenger/token gets two colors as it arrives that is Lx and Px where x corresponds to a digit i.e. L1 and P1. So, its colors are L1 and P1 which means level 1 and person number 1.

```
%Pre-Processor code for tArrival
function [fire, trans] = tArrival_pre(trans)
global global_info
%will run from minimum to maximum level
for Level = global_info.MIN_LEVEL:global_info.MAX_LEVEL
    %get the value of 'global_info.ATL' e.g global_info.ATL1 frm loadData()
    %function
    Waiting = eval(['global_info.ATL',int2str(Level+3),';']);
    if not(isempty(Waiting))
        %if not empty then get the value
        %present at first index which is passenger's arrival time
        timeOfArrival = Waiting(1);
         % if current time is greater than passenger's arrival time than
         %this is the time for passenger to arrive
        if ge(current_time, timeOfArrival)
            %update the person counter
            global_info.Person_No = global_info.Person_No + 1;
            PNo = global_info.Person_No;
             %second index value shows the destination floor of passenger
            whereTo = Waiting(2);
             %third index value will tell that whether a passenger has
             %passcode or not i.e. bool value: 0 or 1
              vipPass=Waiting(3);
          % reduce the 1st,2nd and 3rd index value from global info.ATLx
            eval(['global_info.ATL',int2str(Level+3),'=Waiting(4:end);']);
             % time at which passenger arrived
            global_info.PASSENGER_DATA(PNo, 1) = current_time;
             % At which level passenger arrived
            global_info.PASSENGER_DATA(PNo, 4) = Level; % from which level
             % To which level passenger wants to go
            global_info.PASSENGER_DATA(PNo, 5) = whereTo; % to which level
            %hold value which shows that passenger has passcode or not i.e
            %0 or 1
            global_info.PASSENGER_DATA(PNo, 8) = vipPass;
            %two colors are assigned to each token/passenger
            color1 = ['L', int2str(Level)];
            color2 = ['P', int2str(PNo)];
            trans.new_color = {color1, color2};
            fire = 1;
            disp(['Passenger Arrival: Time:', num2str(current_time),...
                '  at level:', color1, '   Person No:', color2]);
            return
        end
    end
end
fire = 0;
```

Code Snippet 37: For Pre-Processor of transition tArrival

The following code snippet shows Post-Processor for transition "*tArrival*" in which we are getting the list of arrival floors at which all passengers arrived as *global_info.PickLevel*.After that we get all the passengers who holds vip passcode from *global_info.PASSENGER_DATA* matrix. Then, we run a *for* loop for the number of passengers who holds a passcode that we got in the previous step, to get the drop/destination level of all those passengers i.e *global_info.b* . Ultimately,we take transpose of *global_info.b* and make a final drop level list as *global_info.DropLevel*. After that we append both of these lists i.e. *global_info.PickLevel and global_info.DropLevel* to make an aggregate list of floors which would be the itenirary of floors for the movement of the car, later on it would be used by the pre-processor of transition "tAccel" i.e. *global_info.Iterinary* to reflect the true nature of first come first serve approach.

```
    case 'tArrival'
        %Contains arrival floors of all the passengers
         global_info.PickLevel = global_info.PASSENGER_DATA(:, 4) ; % from which level

%        get all the passengers from global_info.PASSENGER_DATA who have
%        VIP passcode
         global_info.vPassenger = find(global_info.PASSENGER_DATA(:, 8)==1);

        % to get drop level of all the passengers who have VIP passcode
         for i = 1:length(global_info.vPassenger)
         DropLevelList = global_info.PASSENGER_DATA(global_info.vPassenger(i),5);

             global_info.b(i)=DropLevelList;
         end
          global_info.DropLevel= transpose(global_info.b);
          %makes up a final iterinary of elevator by appending both
         %global_info.PickLevel & global_info.DropLevel
         global_info.Iterinary = [global_info.PickLevel; global_info.DropLevel ] ;
```

Code Snippet 38: For Post-Processor of transition tArrival

loadData() function is related to arrival module because it contains the list of passengers that would arrive at each level which consist of arrival time, destination floor and VIP passcode holding status of each passenger that is going to use the elevator. Using this list, it calculates the total number of passengers beforehand. In this elevator algorithm, we are dividing the length of ATLx by 3, for calculating the number of passengers. Because, now we have the third parameter in historical data also which contains VIP passcode holding status. It also forms a vector /matrix named as *global_info.PASSENGER_DATA* to keep record of passengers regarding different parameters like person number, arrival time, elevator leaving time, elevator entering time, arrival floor, destination floor, VIP passcode holding status etc.

global_info.Travellers_at_Present is also a vector which is used to save number of passengers at any given time to make sure that elevator does not allow to carry passengers more than it capacity.

```
function [] = loadData()
global global_info

global_info.People_in_Car = 0; % initially we have 0 passengers inside car

% Passenger's Historial data: Arrival time of passengers to the lift at
% different levels. Also contain destination floor of passengers and
% passcode holding status value i.e. 0 or 1
% Test-1:
global_info.ATL1 = [20  0 0]; % level -2
global_info.ATL2 = [10 2 1]; % level -1
global_info.ATL3 = [30 2 0];  % level 0
global_info.ATL4 = [50 -2 1];  % level 1
global_info.ATL5 = [40 -1 0];  % level 2
passengers = 0;
%for calculating number of passengers according to historical data
for Level = global_info.MIN_LEVEL:global_info.MAX_LEVEL
    ATLx = eval(['global_info.ATL',int2str(Level+3),';']);
    passengers = passengers + length(ATLx)/3;
end
disp(['Total number of passengers: ', int2str(passengers)]);
global_info.TOTAL_USERS = passengers;
%a matrix/ vector to hold information about passenger like person no,
%arrival and departure time, arrival and departure floor, car travelling
%status,missed car status, passcode holding status
global_info.PASSENGER_DATA = zeros(passengers,8);
global_info.Travellers_at_Present = zeros(1, global_info.MAX_CAPACITY);
global_info.Person_No = 0; % Person counter (or person ID number)

for i = 1:global_info.TOTAL_USERS
    eval(['global_info.P2DL_', int2str(i), ' = [];']);
end
```

Code Snippet 39: For loadData() function

3.3.1.2 Initialization module:

Initialization module contain one place and a transition namely *pInit* and *tInit* . Transition *tInit* is related to one of the transitions of *Car At a Level* module in a way that it enables the transition *tOpenDoor* of *Car At a Level* module. In other words, it helps in initiating the workflow of the elevator. The token residing in place *pInit* symbolizes the elevator car and it gets initiated by firing transition *tInit* to reach the place named as *pArrivedAtLevel* which enables the transition *tOpenDoor* of *Car At a Level* module to open the elevator's door to allow passengers to step on it.

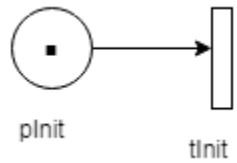

Figure 21: Initialization Module

The following code snippet shows the pre-processor of the transition *tInit* which is fired as it is or without any further condition to initiate the elevator workflow

```
function [fire, trans] = COMMON_PRE(trans)
global global_info

switch trans.name
    case 'tInit', fire = 1;
```

Code Snippet 40: Pre-Processor for transition tInit

3.3.1.3 Car At a Level Module:

It consists of six transitions and four places namely: *tStepON, tStepOFF, tOpenDoor, tBoardingComplete, tDoorDelay, tCloseDoor and pAtLevel, pBoardingComplete, pDoorOpened, pDoorDelay.*

After the arrival of passengers in the form of tokens in the place *pWaiting* and door of the elevator is opened by transition *tOpenDoor*, only those passengers can step on it who have VIP passcode. The passenger completes the boarding procedure by pressing the button of their desired floor to reach, placed inside the elevator car. The elevator door remains open by the set duration of door delay. This happens by executing transition *tDoorDelay*. Door closes after the completion of boarding process and the set time of door delay.

Another case is that a passenger wants to step off the elevator, then after reaching its desired level/floor, door opens, and he leaves the car by using transition *tStepOFF*.

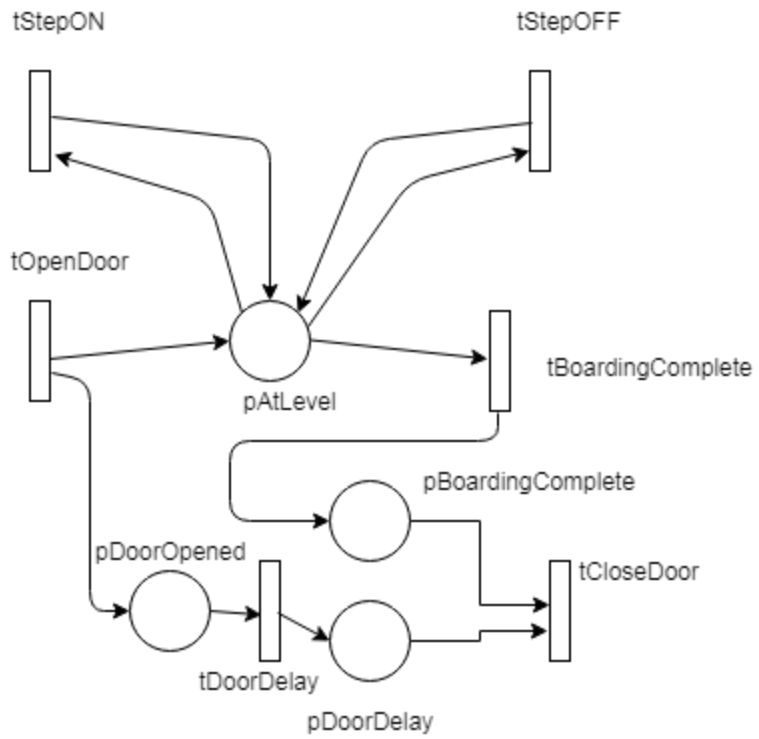

Figure 22: Car At a Level Module

The following code snippet shows pre-processor for transaction *tStepON.* It selects a token from *pWaiting* whose color which contain the arrival level is equal to current floor level. As we have set maximum passenger capacity of elevator equal to five that's why we check for passenger vacancy in the car. If there is a vacancy then we select the token.Then to get the person number of the selected token, we call the function named as *separateColors(the2colors),* since a token has two colors which shows the floor number at which a passenger arrived and the person number. After that we check the VIP passcode holding status of that passenger.If he holds a VIP passcode, then we allow the transition to fire and save the time when passenger enters the car and put its travelling status to *true* in the *global_info.PASSENGER_DATA* matrix. Otherwise, it is not fired.

In case , we don't have vacancy in the car then, we change priority of transition tStepON from 1 to 0 to prevent it from firing again and again. After that function filledMissedCarTable() is called to execute. This function first find all the passengers whose step on floor number is equal to current floor number.Then check all of the passengers that whether they have already arrived and waiting or not. If a passenger has arrived and waiting then add 1 to the value of 7th column of that passenger's row in *global_info.PASSENGER_DATA* matrix, which signifies that that passenger could not step on the car because it was full to its maximum capacity.

```
    case 'tStepON'
%         tokID = tokenArrivedEarly('pWaiting', 1);
         currentLevel = global_info.Current_Level;
         %selecting the token whose arrival level is equal to current level
        tokID = tokenAnyColor('pWaiting', 1, ...
                      {['L',int2str(currentLevel)]});

        if not(tokID) % if there is no one waiting at this level, then do not fire
            fire = 0; return
        end

       % Otherwise,check for vacancy in the car, if some people are waiting at this level
        peopleInCar = sum(global_info.PASSENGER_DATA(:, 6));
        vacant = global_info.MAX_CAPACITY  - peopleInCar;
        if not(vacant)
            disp('** Caution! cannot take more passengers! Car is full');

            % reducing priority of "tStepON" to make it weak so that it will not try again & again
            priorset('tStepON', 0);
            fillMissedCarTable();%holds the persons who could not enter the car because of no vacancy
            fire = 0; return
        end
        trans.selected_tokens = tokID;
        trans.override = 1;
         %using separateColors(the2colors) to get the person number to record
        %car entering time and to put travelling stauts=true, of that
        %particular person. Also to verify passcode holding status.
        the2colors = get_color('pWaiting', tokID);
        PNo = separateColors(the2colors);
        %gets the passcode holding status of the passenger
        vPass = global_info.PASSENGER_DATA(PNo, 8) ;
        %if it's equal to 1 then transition fires
        if eq(vPass, 1)

        fire = 1;
                % the time passenger ENTERS car
        global_info.PASSENGER_DATA(PNo, 2) = current_time;
        % the passenger is inside the car: travelling
        global_info.PASSENGER_DATA(PNo, 6) = true;
        disp(['Passenger Entering Car: Time:', num2str(current_time),...
              '  at level:', int2str(currentLevel),...
              '   Person No:', int2str(PNo)]);

        else
         fire = 0;
        end
```

Code Snippet 41: Pre-Processor for transition tStepON

```
% function for separating two colors of the token to get the person number
% of a passenger
function [PNo] = separateColors(the2colors)

color1 = the2colors{1};
color2 = the2colors{2};

% To find at which Level he arrived and also his passenger number
if strcmp(color1(1), 'L')
    fromWhichLevel = color1;
    PNo = color2;
else
    fromWhichLevel = color2;
    PNo = color1;
end

PNo = PNo(2:end);
PNo = str2double(PNo);
```

Code Snippet 42: For separateColors(the2colors) function

```
%function for saving information about passengers who could not enter
%elevator because it was already full to its max capacity
function [] = fillMissedCarTable()
global global_info

[indices_for_this_level, ~] = ...
    find(global_info.PASSENGER_DATA(:,4)== global_info.Current_Level);

for i = 1:numel(indices_for_this_level)
    row_index = indices_for_this_level(i);

    if not(global_info.PASSENGER_DATA(row_index, 2))
       % shows that passenger has arrived and waiting to step on!!!
        global_info.PASSENGER_DATA(row_index,7) = ...
            global_info.PASSENGER_DATA(row_index,7) + 1;
    end
end
```

Code Snippet 43: For fillMissedCarTable() function

The following code snippet shows pre-processor for transition *tStepOFF.* First it gets sum of all the passengers travelling inside the car. If the car is empty it will not fire. Otherwise, it will compare the destination level of all the passengers present in the car one by one with the current floor level. If there is a match, then that passenger leaves the car. Also, the value of sixth column of

PASSENGER_DATA matrix which shows travelling status of a passenger inside the car is set to zero that is false, of the passenger who leaves.

```
case 'tStepOFF'
    % it shows people who are currently inside car, means travelling
    current_passengers = global_info.PASSENGER_DATA(:, 6);
    peopleInCar = sum(current_passengers);
    if not(peopleInCar), fire = 0; return, end % not allowed to fire if the car is empty
    [~, PNoInCar] = find(transpose(current_passengers));
    for i = 1:numel(PNoInCar)
        PNo = PNoInCar(i);
        whereTo = global_info.PASSENGER_DATA(PNo, 5);
        %if current level is equal to destination level of specified person,then that person leaves
        if eq(global_info.Current_Level, whereTo)

            % set travelling in the car status of that person equal to
            % 0, as he is leaving
            global_info.PASSENGER_DATA(PNo, 6) = 0;
            % saving the leaving time from CAR
            global_info.PASSENGER_DATA(PNo, 3) = current_time;
            disp(['Passenger is Leaving Car: Time:', num2str(current_time),...
                '  at level:', int2str(whereTo),...
                '   Person No:', int2str(PNo)]);

            fire = 1; return
        end
    end
    fire = 0;
```

Code Snippet 44: Pre-Processor for transition tStepOFF

The following code snippet shows pre-processor for transition *tOpenDoor* which is fired without any further condition.

```
case 'tOpenDoor', fire = 1;
```

Code Snippet 45: Pre-Processor for transition tOpenDoor

The following code snippet shows post-processor for transition *tOpenDoor* which displays a message on console after reaching at a floor and opening the elevator's door.

```
case 'tOpenDoor'
    disp(' ');
    disp(['**** Elevator Arrived at Level ',...
          int2str(global_info.Current_Level), ...
          '     at time: ', num2str(current_time), ' *******']);
```

Code Snippet 46: Post-Processor for transition tOpenDoor

The following code snippet shows pre-processor for transition *tBoardingComplete* in which we are setting transition priority of *tStepON* back to 1 from 0 using *priorset*() function and allowing the transition to fire.

```
case 'tBoardingComplete'
    % increasing  priority of "tStepON" back to 1 as it was set to
    %0 previously in "tStepON"
    priorset('tStepON', 1);
    fire = 1;
```

Code Snippet 47: Pre-Processor for transition tBoardingComplete

The following code snippet shows pre-processor for transition *tDoorDelay* in which we are overriding the transition and allowing it to fire.

```
case 'tDoorDelay', trans.override = 1; fire = 1;
```

Code Snippet 48: Pre-Processor for transition tDoorDelay

The following code snippet shows pre-processor for transition *tCloseDoor* which displays a console message that the door is getting closed at current level and current time and allowing the transition to fire.

```
case 'tCloseDoor'
    disp(['**** Closing the door now at Level ',...
        int2str(global_info.Current_Level), ...
        '     at time: ', num2str(current_time), ' *******']);
    disp(' ');
    fire = 1;
```

Code Snippet 49: Pre-Processor for transition tCloseDoor

3.3.1.4 Departure module:

This module consists of a transition and a place namely: *tLeaving* and *pLeft.* After transition *tStepOFF* at *Car At a Level* module to exit the elevator, a passenger leaves using transition *tLeaving.*

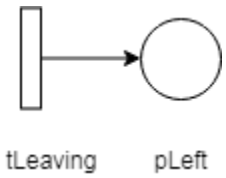

Figure 23: Departure Module

The following code snippet shows pre-processor for transition *tLeaving* which is getting fired without any further condition.

```
    case 'tLeaving', fire = 1;
```

Code Snippet 50: Pre-Processor for transition tLeaving

The following code snippet shows post-processor for transition *tLeaving* which gets all the passengers from *global_info.PASSENGER_DATA* who have VIP passcode. Then we get the number of passengers who have VIP passcode. After that, we compare the number of passengers who have VIP passcode with the total number of passengers who had VIP passcode and already left, to continue simulation until all the passengers with the VIP passcode do not leave.

```
    case 'tLeaving'
        n = ntokens('pLeft');
%        get all the passengers from global_info.PASSENGER_DATA who have
%        VIP passcode
        global_info.vPassenger = find(global_info.PASSENGER_DATA(:, 8)==1);
%         gets the number of passengers who have VIP passcode
        L=length( global_info.vPassenger);
         %don't stop the simulation untill all the passengers with VIP
        %passcode have reached their destination floors
        if eq(n, L)
            global_info.STOP_SIMULATION = true;
        end
```

Code Snippet 51: Post-Processor for transition tLeaving

3.3.1.5 Car Moving module:

This module contains three transitions and a place namely: *tAccel, tMoving, tDeAccel* and *pMoving*. After closing door at Car At a Level module, elevator accelerates using transition *tAccel* to take passengers to their destination floors. Transition *tMoving* fires until the car reaches to its required floor because in first come first serve elevator algorithm car moves according to the fixed itinerary which is formed in the post-processor of transition *tArrival* after the arrival of passengers.
Finally, after reaching the destination floor it comes to a halt by using transition *tDeAccel.*

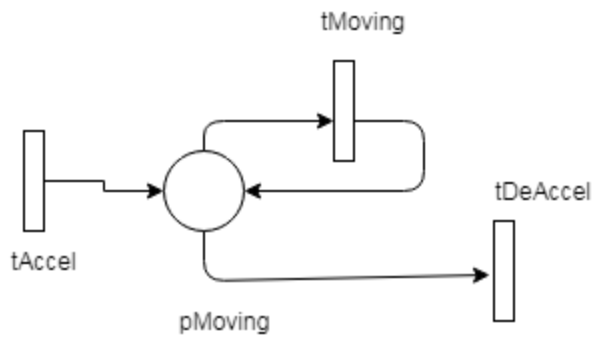

Figure 24: Car Moving Module

The following code snippet shows code of pre-processor of transition *tAccel*. It implements the actual first come first serve algorithm regarding the movement of elevator's car. According to which, the car follows the fixed itinerary to move, which is created in the post processor of *tArrival* after the arrival of passengers. In other words, car will pick up all the passengers in the order in which they have requested for the elevator and deliver the passengers to their required floor in the same order as they requested the elevator. It is done by selecting the first index value of *global_info.Itinerary(1)* as the current level and move the car to that level by setting the value of *global_info.Starting_to_move =true.*

```
case 'tAccel' % It is the main strategy for moving car to destination floors!!!

    n = ntokens('pLeft');
   global_info.vPassenger = find(global_info.PASSENGER_DATA(:, 8)==1);
   L=length( global_info.vPassenger);
    if eq(n, L)
        global_info.STOP_SIMULATION = true;

    else
   %the car moves according to the set Itinerary in the Post-Processor
   %of "tArrival"
   %it selects first index value of global_info.Iterinary(1) as the next floor to reach there
     global_info.Current_Level =  global_info.Iterinary(1)  ;


    global_info.Starting_to_move = true; % It is used by "tMoving"
    fire = 1;
    end
```

Code Snippet 52: Pre-Processor for transition tAccel

The following code snippet shows code of post-processor of transition *tAccel*. According to which after accelerating the car towards the floor mentioned as the first index value of *global_info.Itinerary*, we are removing that value and assigning rest of the values to it as shown below.

```
case 'tAccel'
    %getting the values of global_info.Iterinary
    Move= eval('global_info.Iterinary');
    %removing the 1st index value and assigning the rest of the values to global_info.Iterinary
    eval(['global_info.Iterinary','=Move(2:end);']);
```

Code Snippet 53: Post-Processor for transition tAccel

The following code snippet shows code of pre-processor of transition *tDeAccel* which is fired without any further conditions.

```
case 'tDeAccel'
    fire = 1;
```

Code Snippet 54: Pre-Processor for transition tDeAccel

The following code snippet shows code of pre-processor of transition *tMoving* which moves the car to the floor as per defined by the 1st index value of *global_info.Itinerary(1).* Also it fires, if the value of *global_info.Starting_to_move* is set to true previously in *tAccel*. After that we set *global_info.Starting_to_move* to false.

```
%it moves the car to the floor as per defined by the value of global_info.Iterinary(1)
   case 'tMoving'
       if global_info.Starting_to_move
           global_info.Starting_to_move = false;
           fire = 1;
       else
           fire = 0;
       end
```

Code Snippet 55: Pre-Processor for transition tMoving

3.3.1.6 Inter modular connections:

Following diagram shows the inter module connections which integrates all modules to make a single model of an elevator. There are three prominent places in it which helps to connects all the modules together i.e. *pArrivedAtLevel*, *pAboutToMove* and *pLeaving*.

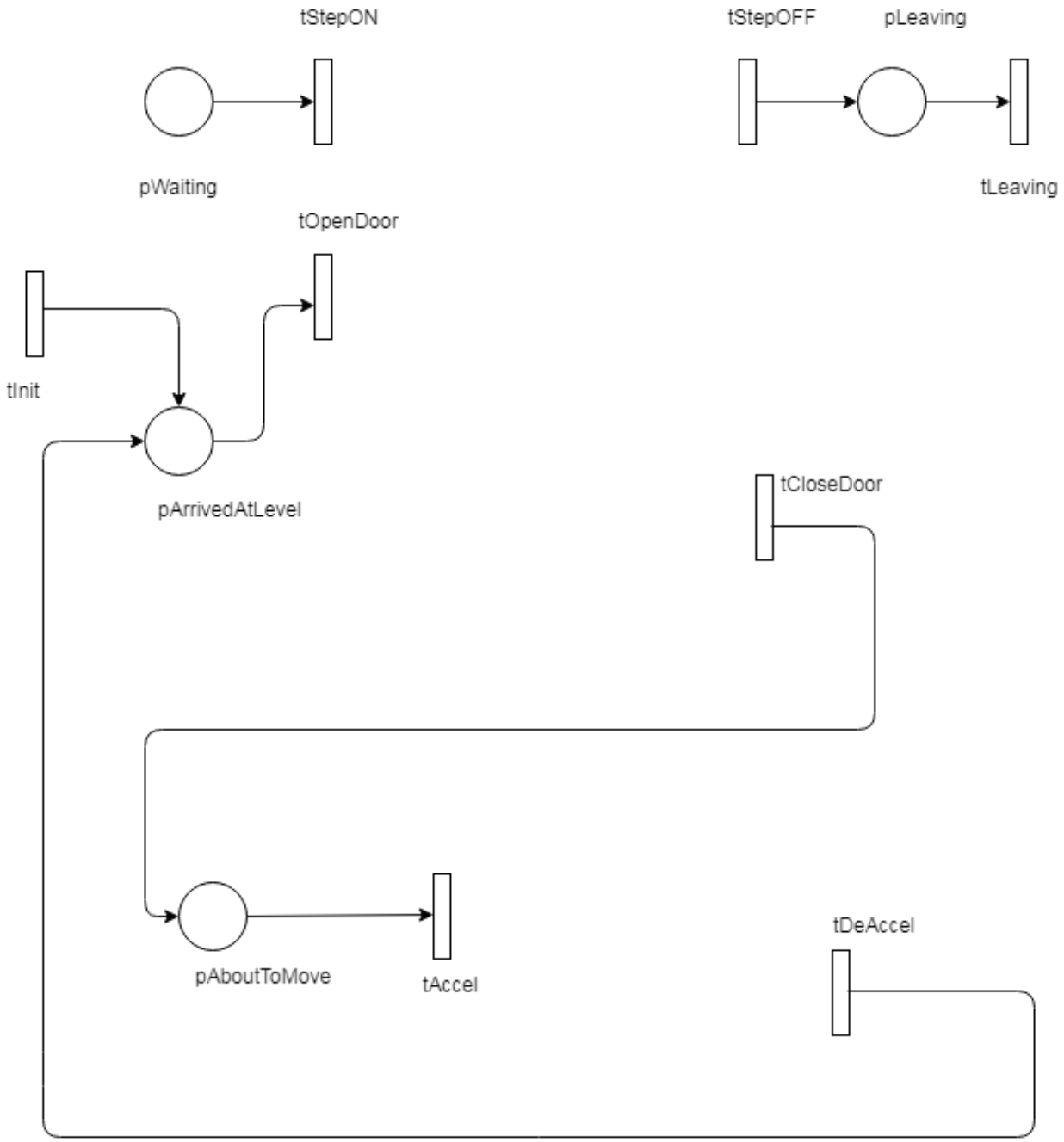

Figure 25: Inter Module Connections

3.4 Fixed sectoring elevator algorithm:

In this algorithm, elevator is fixed to serve a specific number of floors and the elevator car moves across different floors using Shabbat movement. We can take a tall building as an example, which has commercial offices at bottom and residential apartments at the top. So, this algorithm makes sure that the elevator is fixed to residential apartment floors and could only be used by them.

3.4.1 Coding of modules:

3.4.1.1 Arrival module:

This module consists of two places and a transition namely: *pArrivals, pWaiting and tArrivals.* Passengers arrive at the given time in loadData() function, present in the code and wait for the elevator to arrive and step on to it. After the arrival/generation of tokens, they wait or reside in place *pWaiting*.

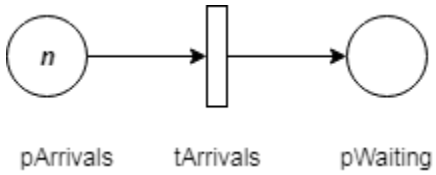

Figure 26: Arrival Module

Following code snippet shows pre-processor of transition tArrival. A for loop will run for each floor that is from minimum to maximum floor. Then it will get the value of *"global_info.ATL"* e.g *"global_info.ATL1"* from loadData() function in a variable named as "*Waiting".* If that variable is not empty then get the first index value which is the arrival time of a passenger. Then we compare current time with the arrival time of a passenger and if the current time is greater than the arrival time, then it is the time for a passenger to arrive.

As the passengers arrive it updates the person counter which saves the number of persons that has arrived yet using variable named as *"global_info.Person_No"*. The second index value of "*Waiting"* variable represents the destination floor of a passenger. It also removes the passenger from the arrival list in the loadData() function which contains its arrival time and destination floor.For example if the loadData() function contains a vector where it has two passengers detail like: *global_info.ATL1=[15 1, 20 2],* then it will remove "15 1" from the vector after its arrival. Also, three fields of global_info.PASSENGER_DATA, which is a vector to store various parameters about passengers are filled like arrival time, at which level he arrived and the destination level. Each passenger/token gets two colors as it arrives that is Lx and Px where x corresponds to a digit i.e. L1 and P1. So, its colors are L1 and P1 which means level 1 and person number 1.

```
function [fire, trans] = tArrival_pre(trans)
global global_info
%will run from minimum to maximum level
for Level = global_info.MIN_LEVEL:global_info.MAX_LEVEL
    %get the value of 'global_info.ATL' e.g global_info.ATL1 frm loadData()
    %function
    Waiting = eval(['global_info.ATL',int2str(Level+3),';']);
    if not(isempty(Waiting))
         %if not empty then get the value
        %present at first index which is passenger's arrival time
        timeOfArrival = Waiting(1);
         % if current time is greater than passenger's arrival time than
         %this is the time for passenger to arrive
        if ge(current_time, timeOfArrival)
            %update the person counter
            global_info.Person_No = global_info.Person_No + 1;
            PNo = global_info.Person_No;
             %second index value shows the destination floor of passenger
            whereTo = Waiting(2);

           % reduce the 1st and 2nd index value from global_info.ATLx
            eval(['global_info.ATL',int2str(Level+3),'=Waiting(3:end);']);
            % time at which passenger arrived
            global_info.PASSENGER_DATA(PNo, 1) = current_time;
            % At which level passenger arrived
            global_info.PASSENGER_DATA(PNo, 4) = Level;
             % To which level passenger wants to go
            global_info.PASSENGER_DATA(PNo, 5) = whereTo;
             %two colors are assigned to each token/passenger
            color1 = ['L', int2str(Level)];
            color2 = ['P', int2str(PNo)];
            trans.new_color = {color1, color2};
            fire = 1;
            disp(['Passenger Arrival: Time:', num2str(current_time),...
                '  at level:', color1, '   Person No:', color2]);
            return
        end
    end
end
fire = 0;
```

Code Snippet 56: For Pre-Processor of transition tArrival

loadData() function is related to arrival module because it contains the list of passengers that would arrive at each level which consist of arrival time and destination floor of each passenger that is going to use the elevator. Using this list, it calculates the total number of passengers beforehand. It also forms a vector /matrix named as *global_info.PASSENGER_DATA* to keep record of passengers regarding different parameters like person number, arrival time, elevator leaving time, elevator entering time, arrival floor, destination floor etc.
global_info.Travellers_at_Present is also a vector which is used to save number of passengers at any given time to make sure that elevator does not allow to carry passengers more than it capacity.

```
function [] = loadData()
global global_info

global_info.People_in_Car = 0; % initially we have 0 passengers inside car

% Passenger's Historial data: Arrival time of passengers to the lift at
% different levels. Also contain destination floor of passengers
% Test-1:
global_info.ATL1 = [20  0]; % level -2
global_info.ATL2 = [10 2]; % level -1
global_info.ATL3 = [30 2];  % level 0
global_info.ATL4 = [50 2];  % level 1
global_info.ATL5 = [40 1];  % level 2

passengers = 0;
%for calculating number of passengers according to historical data
for Level = global_info.MIN_LEVEL:global_info.MAX_LEVEL
    ATLx = eval(['global_info.ATL',int2str(Level+3),';']);
    length(ATLx);
    passengers = passengers + length(ATLx)/2;
end
disp(['Total number of passengers: ', int2str(passengers)]);
global_info.TOTAL_USERS = passengers;
%a matrix/ vector to hold information about passenger like person no,
%arrival and departure time, arrival and departure floor, car travelling
%status,missed car status
global_info.PASSENGER_DATA = zeros(passengers,7);
global_info.Travellers_at_Present = zeros(1, global_info.MAX_CAPACITY);
global_info.Person_No = 0; % Person counter (or person ID number)

for i = 1:global_info.TOTAL_USERS
    eval(['global_info.P2DL_', int2str(i), ' = [];']);
end
```

Code Snippet 57: For loadData() function

3.4.1.2 Initialization module:

Initialization module contain one place and a transition namely *pInit* and *tInit* . Transition *tInit* is related to one of the transitions of *Car At a Level* module in a way that it enables the transition *tOpenDoor* of *Car At a Level* module. In other words, it helps in initiating the workflow of the elevator. The token residing in place *pInit* symbolizes the elevator car and it gets initiated by firing transition *tInit* to reach the place named as *pArrivedAtLevel* which enables the transition *tOpenDoor* of *Car At a Level* module to open the elevator's door to allow passengers to step on it.

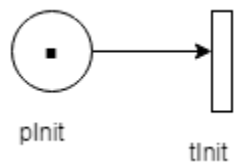

Figure 27: Initialization Module

The following code snippet shows the pre-processor of the transition *tInit* which is fired as it is or without any further condition to initiate the elevator workflow

```
function [fire, trans] = COMMON_PRE(trans)
global global_info

switch trans.name
    case 'tInit', fire = 1;
```

Code Snippet 58: Pre-Processor for transition tInit

3.4.1.3 Car At a Level Module:

It consists of six transitions and four places namely: *tStepON, tStepOFF, tOpenDoor, tBoardingComplete, tDoorDelay, tCloseDoor and pAtLevel, pBoardingComplete, pDoorOpened, pDoorDelay.*

After the arrival of passengers in the form of tokens in the place *pWaiting* and door of the elevator is opened by transition tOpenDoor, the passenger step on it. The passenger completes the boarding procedure by pressing the button of their desired floor to reach, placed inside the elevator car. The elevator door remains open by the set duration of door delay. This happens by executing transition *tDoorDelay*. Door closes after the completion of boarding process and the set time of door delay.

Another case is that a passenger wants to step off the elevator, then after reaching its desired level/floor, door opens and he leaves the car by using transition tStepOFF.

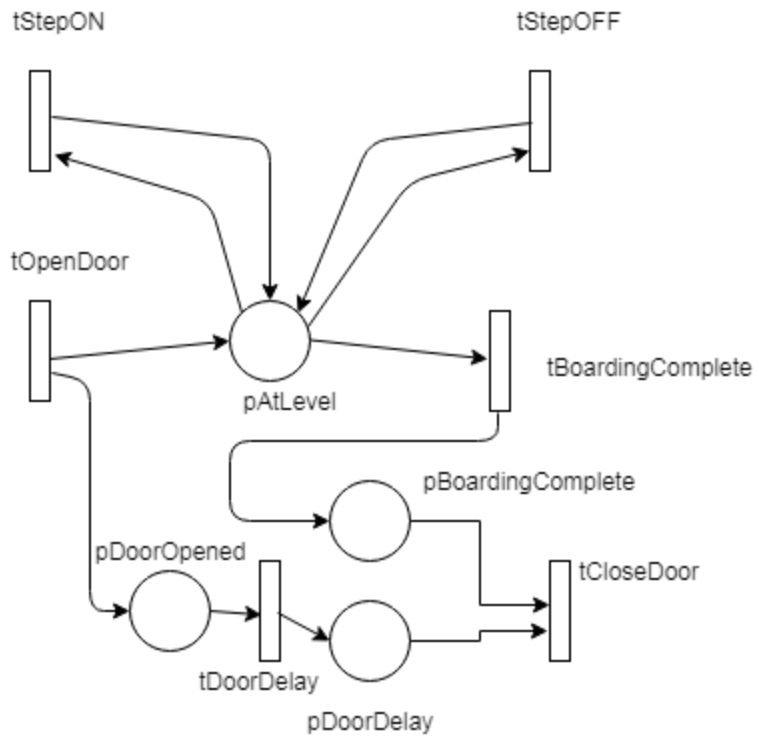

Figure 28: Car At a Level Module

The following code snippet shows pre-processor for transaction *tStepON.* As we have fixed our elevator to only three floors out of five that is from 0 to 2, that's why we will first compare the current floor number with floor number 0,1 and 2. Also, there are two passengers in the loadData() function who will arrive at level -1 and -2, they would not be able to step on the elevator because elevator is restricted to only three floors that is from 0 to 2. For example,as shown below:

```
global_info.ATL1 = [20  0]; % level -2
global_info.ATL2 = [10 2]; % level -1
```

Code Snippet 59: From loadData() function

If current floor is equal to 1 then, it selects a token from *pWaiting* whose color is equal to *L1* which represents floor number 1. If there is no token with the required color then transition will not fire. Else, as we have set maximum passenger capacity of elevator equal to five that's why we check for passenger vacancy in the car. If there is a vacancy then we select the token and allow the transition to fire.Then to get the person number of the selected token, we call the function named as *separateColors(the2colors),* since a token has two colors which shows the floor number at which a passenger arrived and the person number. Also we save the time when passenger enters the car and put its travelling status to *true* in the *global_info.PASSENGER_DATA* matrix.

Otherwise, if there is no vacancy then we change priority of transition tStepON from 1 to 0 to prevent it from firing again and again. After that function filledMissedCarTable() is called to execute. This function first find all the passengers whose step on floor number is equal to current floor number.Then check all of the passengers that whether they have already arrived and waiting or not. If a passenger has arrived and waiting then add 1 to the value of 7th column of that passenger's row in *global_info.PASSENGER_DATA* matrix, which signifies that that passenger could not step on the car because it was full to its maximum capacity.

Else if current floor is equal to 2 then, it selects a token from *pWaiting* whose color is equal to *L2* which represents floor number 2. If there is no token with the required color then transition will not fire. Else, as we have set maximum passenger capacity of elevator equal to five that's why we check for passenger vacancy in the car. If there is a vacancy then we select the token and allow the transition to fire.Then to get the person number of the selected token, we call the function named as *separateColors(the2colors),* since a token has two colors which shows the floor number at which a passenger arrived and the person number. Also we save the time when passenger enters the car and put its travelling status to *true* in the *global_info.PASSENGER_DATA* matrix.

Otherwise, if there is no vacancy then we change priority of transition tStepON from 1 to 0 to prevent it from firing again and again. After that function filledMissedCarTable() is called to execute. This function first find all the passengers whose step on floor number is equal to current floor number.Then check all of the passengers that whether they have already arrived and waiting or not. If a passenger has arrived and waiting then add 1 to the value of 7th column of that passenger's row in *global_info.PASSENGER_DATA* matrix, which signifies that that passenger could not step on the car because it was full to its maximum capacity.

Else if current floor is equal to 0 then, it selects a token from *pWaiting* whose color is equal to *L0* which represents floor number 0. If there is no token with the required color then transition will not fire. Else, as we have set maximum passenger capacity of elevator equal to five that's why we check for passenger vacancy in the car. If there is a vacancy then we select the token and allow the transition to fire.Then to get the person number of the selected token, we call the function named as *separateColors(the2colors),* since a token has two colors which shows the floor number at which a passenger arrived and the person number. Also we save the time when passenger enters the car and put its travelling status to *true* in the *global_info.PASSENGER_DATA* matrix.

Otherwise, if there is no vacancy then we change priority of transition tStepON from 1 to 0 to prevent it from firing again and again. After that function filledMissedCarTable() is called to execute. This function first find all the passengers whose step on floor number is equal to current floor number.Then check all of the passengers that whether they have already arrived and waiting or not. If a passenger has arrived and waiting then add 1 to the value of 7th column of that passenger's row in *global_info.PASSENGER_DATA* matrix, which signifies that that passenger could not step on the car because it was full to its maximum capacity.

```
case 'tStepON'

     currentLevel = global_info.Current_Level;
    % Modifications has been done here
   if eq(currentLevel,1)
      %selecting the token whose arrival level is equal to one
      tokID = tokenAnyColor('pWaiting', 1, ...
                   {'L1'});
    if not(tokID) % if there is no one waiting at this level, then do not fire
        fire = 0; return
    end

     % Otherwise,check for vacancy in the car, if some people are waiting at this level
    peopleInCar = sum(global_info.PASSENGER_DATA(:, 6));
    vacant = global_info.MAX_CAPACITY  - peopleInCar;
    if not(vacant)
        disp('**Caution! cannot take more passengers! Car is full ');

        % reducing priority of "tStepON" to make it weak so that it will not try again & again
        priorset('tStepON', 0);
        fillMissedCarTable();%holds the persons who could not enter the car because of no vacancy
        fire = 0; return
    end
    trans.selected_tokens = tokID;
    trans.override = 1;
    fire = 1;
    %using separateColors(the2colors) to get the person number to record
    %car entering time and to put travelling stauts=true, of that
    %particular person
    the2colors = get_color('pWaiting', tokID);
    PNo = separateColors(the2colors);

     % the time passenger ENTERS car
    global_info.PASSENGER_DATA(PNo, 2) = current_time;
   % the passenger is inside the car: travelling
    global_info.PASSENGER_DATA(PNo, 6) = true;
    disp(['Passenger is Entering Car: Time:', num2str(current_time),...
          '  at level:', int2str(currentLevel),...
          '   Person No:', int2str(PNo)])
```

```
 % Modifications has been done here
elseif eq(currentLevel,2)
     %selecting the token whose arrival level is equal to two
    tokID = tokenAnyColor('pWaiting', 1, ...
               {'L2'});
 if not(tokID) % if there is no one waiting at this level, then do not fire
     fire = 0; return
 end
  % Otherwise,check for vacancy in the car, if some people are waiting at this level
 peopleInCar = sum(global_info.PASSENGER_DATA(:, 6));
 vacant = global_info.MAX_CAPACITY  - peopleInCar;
 if not(vacant)
     disp('** Caution! cannot take more passengers! Car is full');

      % reducing priority of "tStepON" to make it weak so that it will not try again & again
     priorset('tStepON', 0);
     %holds the persons who could not enter the car because of no vacancy
     fillMissedCarTable();
     fire = 0; return
 end
 trans.selected_tokens = tokID;
 trans.override = 1;
 fire = 1;
 %using separateColors(the2colors) to get the person number to record
 %car entering time and to put travelling stauts=true, of that
 %particular person
 the2colors = get_color('pWaiting', tokID);
 PNo = separateColors(the2colors);

 % the time passenger ENTERS car
 global_info.PASSENGER_DATA(PNo, 2) = current_time;
 % the passenger is inside the car: travelling
 global_info.PASSENGER_DATA(PNo, 6) = true;
 disp(['Passenger is Entering Car: Time:', num2str(current_time),...
       '  at level:', int2str(currentLevel),...
       '   Person No:', int2str(PNo)])
```

```
% Modifications has been done here
elseif eq(currentLevel,0)
    %selecting the token whose arrival level is equal to zero
   tokID = tokenAnyColor('pWaiting', 1, ...
                {'L0'});
 if not(tokID) % if there is no one waiting at this level, then do not fire
     fire = 0; return
 end

 % Otherwise,check for vacancy in the car, if some people are waiting at this level
 peopleInCar = sum(global_info.PASSENGER_DATA(:, 6));
 vacant = global_info.MAX_CAPACITY  - peopleInCar;
 if not(vacant)
     disp('**Caution! cannot take more passengers! Car is full');

     % reducing priority of "tStepON" to make it weak so that it will not try again & again
     priorset('tStepON', 0);
     %holds the persons who could not enter the car because of no vacancy
     fillMissedCarTable();
     fire = 0; return
 end
 trans.selected_tokens = tokID;
 trans.override = 1;
 fire = 1;
 %using separateColors(the2colors) to get the person number to record
 %car entering time and to put travelling stauts=true, of that
 %particular person
 the2colors = get_color('pWaiting', tokID);
 PNo = separateColors(the2colors);

 % the time passenger ENTERS car
 global_info.PASSENGER_DATA(PNo, 2) = current_time;
 % the passenger is inside the car: travelling
 global_info.PASSENGER_DATA(PNo, 6) = true;
 disp(['Passenger is Entering Car: Time:', num2str(current_time),...
       '  at level:', int2str(currentLevel),...
       '   Person No:', int2str(PNo)])
```

Code Snippet 60: Pre-Processor for transition tStepON

```
% function for separating two colors of the token to get the person number
% of a passenger

function [PNo] = separateColors(the2colors)

color1 = the2colors{1};
color2 = the2colors{2};

% To find at which Level he arrived and also his passenger number
if strcmp(color1(1), 'L')
    fromWhichLevel = color1;
    PNo = color2;
else
    fromWhichLevel = color2;
    PNo = color1;
end

PNo = PNo(2:end);
PNo = str2double(PNo);
```

Code Snippet 61: For separateColors(the2colors) function

```
%function for saving information about passengers who could not enter
%elevator because it was already full to its max capacity
function [] = fillMissedCarTable()
global global_info

[indices_for_this_level, ~] = ...
    find(global_info.PASSENGER_DATA(:,4)== global_info.Current_Level);

for i = 1:numel(indices_for_this_level)
    row_index = indices_for_this_level(i);

    if not(global_info.PASSENGER_DATA(row_index, 2))
        % shows that passenger has arrived and waiting to step on!!!
        global_info.PASSENGER_DATA(row_index,7) = ...
            global_info.PASSENGER_DATA(row_index,7) + 1;
    end
end
```

Code Snippet 62: For fillMissedCarTable() function

The following code snippet shows pre-processor for transition *tStepOFF.* First it gets sum of all the passengers travelling inside the car. If the car is empty it will not fire. Otherwise, it will compare the destination level of all the passengers present in the car one by one with the current floor level. If

there is a match, then that passenger leaves the car. Also, the value of sixth column of PASSENGER_DATA matrix which shows travelling status of a passenger inside the car is set to zero that is false, of the passenger who leaves.

```
case 'tStepOFF'

     % it shows people who are currently inside car, means travelling
    current_passengers = global_info.PASSENGER_DATA(:, 6);
    peopleInCar = sum(current_passengers);
    if not(peopleInCar), fire = 0; return, end % not allowed to fire if the car is empty
    [~, PNoInCar] = find(transpose(current_passengers));
    for i = 1:numel(PNoInCar)
        PNo = PNoInCar(i);
        whereTo = global_info.PASSENGER_DATA(PNo, 5);
        %if current level is equal to destination level of specified person,then that person leaves
        if eq(global_info.Current_Level, whereTo)

            % set travelling in the car status of that person equal to
            % 0, as he is leaving
            global_info.PASSENGER_DATA(PNo, 6) = 0;
             % saving the leaving time from CAR
            global_info.PASSENGER_DATA(PNo, 3) = current_time;
            disp(['Passenger is Leaving Car: Time:', num2str(current_time),...
                '  at level:', int2str(whereTo),...
                '   Person No:', int2str(PNo)]);

            fire = 1; return
        end
    end
    fire = 0;
```

Code Snippet 63: Pre-Processor for transition tStepOFF

The following code snippet shows pre-processor for transition *tOpenDoor* which is fired without any further condition.

```
case 'tOpenDoor', fire = 1;
```

Code Snippet 64: Pre-Processor for transition tOpenDoor

The following code snippet shows post-processor for transition *tOpenDoor* in which a message is being displayed on console which informs that the elevator has arrived on a specific floor at a specific time.

```
case 'tOpenDoor'
    disp(' ');
    disp(['**** Elevator Arrived at Level ',...
          int2str(global_info.Current_Level), ...
          '     at time: ', num2str(current_time), ' *******']);
```

Code Snippet 65: Post-Processor for transition tOpenDoor

The following code snippet shows pre-processor for transition *tBoardingComplete* in which we are setting transition priority of *tStepON* back to 1 from 0 using *priorset*() function and allowing the transition to fire.

```
case 'tBoardingComplete'
    % increasing  priority of "tStepON" back to 1 as it was set to
    %0 previously in "tStepON"
    priorset('tStepON', 1);
    fire = 1;
```

Code Snippet 66: Pre-Processor for transition tBoardingComplete

The following code snippet shows pre-processor for transition *tDoorDelay* in which we are overriding the transition and allowing it to fire.

```
case 'tDoorDelay', trans.override = 1; fire = 1;
```

Code Snippet 67: Pre-Processor for transition tDoorDelay

The following code snippet shows pre-processor for transition *tCloseDoor* which displays a console message that the door is getting closed at current level and current time and allowing the transition to fire.

```
case 'tCloseDoor'
    disp(['**** Closing the door now at Level ',...
        int2str(global_info.Current_Level), ...
        '     at time: ', num2str(current_time), ' *******']);
    disp(' ');
    fire = 1;
```

Code Snippet 68: Pre-Processor for transition tCloseDoor

3.4.1.4 Departure module:

This module consists of a transition and a place namely: *tLeaving* and *pLeft.* After transition *tStepOFF* at *Car At a Level* module to exit the elevator, a passenger leaves using transition *tLeaving.*

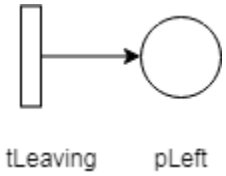

Figure 29: Departure Module

The following code snippet shows pre-processor for transition *tLeaving* which is getting fired without any further condition.

```
    case 'tLeaving', fire = 1;
```

Code Snippet 69: Pre-Processor for transition tLeaving

The following code snippet shows post-processor for transition *tLeaving* in which we get the number of tokens in "*pLeft*". Then we compare number of tokens in "pLeft" with the total number of passengers and don't stop the simulation until all the passengers have left the elevator or have reached their respective destination floors.

```
    case 'tLeaving'
        %gets the number of token in 'pLeft'
        n = ntokens('pLeft');
%comparing number of tokens in 'pLeft' with total number of passengers
        if eq(n, global_info.TOTAL_USERS)
        %don't stop the simulation untill all the passengers have
        %reached their destination floors
            global_info.STOP_SIMULATION = true;
        end
```

Code Snippet 70: Post-Processor for transition tLeaving

3.4.1.5 Car Moving module:

This module contains three transitions and a place namely: *tAccel, tMoving, tDeAccel* and *pMoving*.

After closing door at Car At a Level module, elevator accelerates using transition *tAccel* to take passengers to their destination floors. Transition *tMoving* fire only once and moves the car by one level only because this algorithm is using Shabbat movement that is, the elevator stops at every floor.

Finally, after reaching the destination floor it comes to a halt by using transition *tDeAccel.* Otherwise, if the next floor is the destination floor then it uses *tMoving* once and stops using *tDeAccel.*

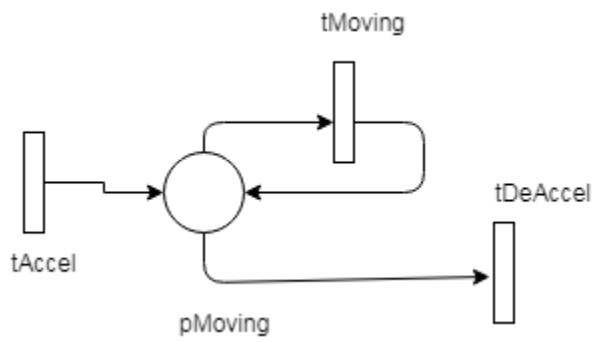

Figure 30: Car Moving Module

The following code snippet shows code of pre-processor of transition *tAccel*. It implements the actual shabbat movement regarding the movement of elevator's car. According to which, if current direction of the car is upwards then it adds 1 to the value of current level and further it checks whether current level value is greater than maximum level value, if yes then it subtracts 2 from the current level value and change the elevator's direction to downwards.

Otherwise, if the elevator's current direction is towards downwards then it subtracts 1 from the value of current level and further it checks whether current level value is smaller than minimum level value, if yes then it adds 2 to the current level value and change the elevator's direction to upwards.

```
case 'tAccel' % the main algorithm for moving car to destination !!!
    % It follows simple Shabbat Movement
    if strcmp(global_info.Current_Direction, 'UP')
        global_info.Current_Level = global_info.Current_Level + 1;
        if gt(global_info.Current_Level, global_info.MAX_LEVEL)
            global_info.Current_Level = global_info.Current_Level - 2;
            global_info.Current_Direction = 'DOWN';
        end
    else % global_info.Current_Direction == 'DOWN')
        global_info.Current_Level = global_info.Current_Level - 1;
        if lt(global_info.Current_Level, global_info.MIN_LEVEL)
            global_info.Current_Level = ...
                   global_info.Current_Level + 2;
            global_info.Current_Direction = 'UP';
        end
    end


    global_info.Starting_to_move = true; % will be used by "tMoving"
    fire = 1;
```

Code Snippet 71: Pre-Processor for transition tAccel

The following code snippet shows code of pre-processor of transition *tDeAccel* which is fired without any further conditions.

```
case 'tDeAccel'
    fire = 1;
```

Code Snippet 72: Pre-Processor for transition tDeAccel

The following code snippet shows code of pre-processor of transition *tMoving* where it is allowed to fire when the bool value of *global_info.Starting_to_move* is set to *true, previously in tAccel. After that we set it to false* because elevator is allowed to move only once in Shabbat movement i.e. movement by one floor only. Otherwise, it's not allowed to fire if previously in *tAccel* the value of *global_info.Starting_to_move* is not set to true.

```
% moves car by one level; tMoving fires only once for shabbat movement
case 'tMoving'
    if global_info.Starting_to_move
        global_info.Starting_to_move = false;
        fire = 1;
    else
        fire = 0;
    end
```

Code Snippet 73: Pre-Processor for transition tMoving

3.4.1.6 Inter modular connections:

Following diagram shows the inter module connections which integrates all modules to make a single model of an elevator. There are three prominent places in it which helps to connects all the modules together i.e. *pArrivedAtLevel*, *pAboutToMove* and *pLeaving*.

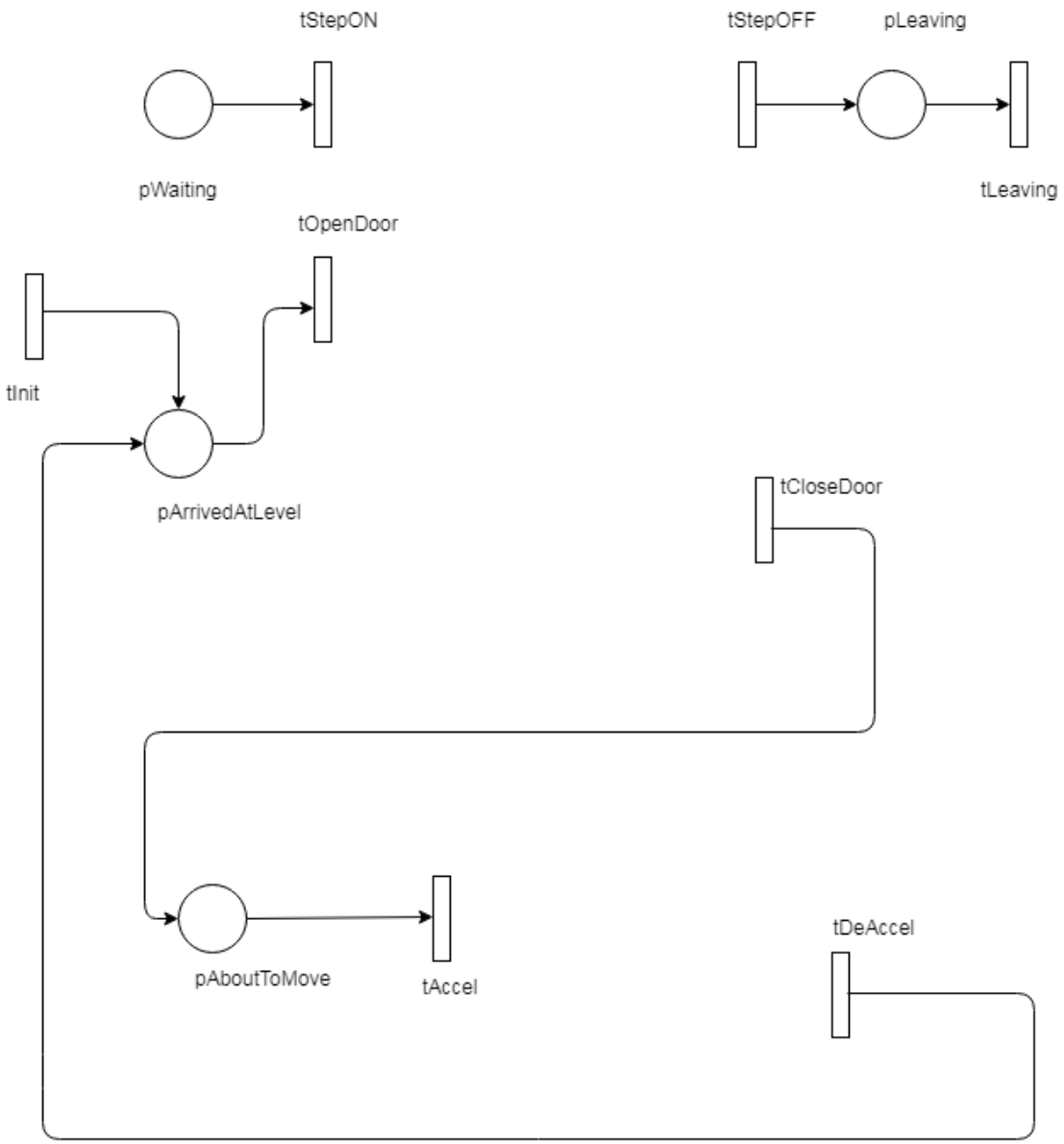

Figure 31: Inter Module Connections

3.5 Dynamic sectoring elevator algorithm:

This algorithm makes sure that the elevator is being used efficiently. Here also elevator is fixed to specific number of floors based on time of the day. Consider we have a building which has offices and commercial hubs at the bottom floors and residential apartments on the top. Accordingly, elevator would be fixed to the bottom floors containing offices during official working hours and after that it would be at the disposal of persons living in the apartments at the top floors of the building. That's how it makes sure to get the max out of the least.

The following code snippet shows the *msf* file in which we are getting the time now value and comparing it with the official working hours to set the sector of floors that the elevator will serve to accordingly. If the time of the day is between 8 AM and 5 PM then the elevator would be set to bottom floors that is: -1 and -2. Otherwise, it would be fixed to top floors which have residential apartments i.e. 0, 1 and 2.

```
clear all; clc;

global global_info
global_info.STOP_AT = 1000;

global_info.MAX_CAPACITY = 5;   % Maximum capacity to hold passengers in car
format shortg
%variable to hold the timenow value
global_info.c = clock;

%comparing timenow value with office timings to set elevator to specific
%floors accordingly
if(global_info.c(4)>7 && global_info.c(4)<18 )
disp("Its Morning time");disp('');
disp("Elevator is fixed to bottom floors: -1 & -2");disp('');

global_info.MIN_LEVEL = -2;     % Bottom floor
global_info.MAX_LEVEL = -1;      % top floor

global_info.Current_Level = -1;
global_info.Current_Direction = 'DOWN';

else
disp("Its Night time");disp('');
disp("Elevator is fixed to top floors: 0,1 & 2");disp('');
global_info.MIN_LEVEL = 0;   % Bottom floor
global_info.MAX_LEVEL = 2;   % top floor

global_info.Current_Level = 0;
global_info.Current_Direction = 'UP';
end
loadData();

% we have five modules and one inter modular connector
pns = pnstruct({'Init_pdf', 'Arrival_pdf', 'Car_ata_Level_pdf',...
                'Car_Moving_pdf', 'Departure_pdf', 'IMC_pdf'});

dyn.m0 = {'pInit',1, 'pArrival',global_info.TOTAL_USERS};
dyn.ft = {'tMoving',7, 'tOpenDoor',5,'tCloseDoor',5, ...
          'tAccel',3, 'tDeAccel',3,...
          'tDoorDelay',15, 'tStepON',1, 'tStepOFF',1,'allothers',0.1}; %
dyn.ip = {'tStepOFF',2,'tStepON',1,  'tMoving',1};
pni = initialdynamics(pns, dyn);
sim = gpensim(pni);
 prnss(sim);
disp(' ');prnstate('Final markings: ');
```

Code Snippet 74: From msf

3.5.1 Coding of modules:

3.5.1.1 Arrival module:

This module consists of two places and a transition namely: *pArrivals, pWaiting and tArrivals.* Passengers arrive at the given time in loadData() function, present in the code and wait for the elevator to arrive and step on to it. After the arrival/generation of tokens, they wait or reside in place *pWaiting*.

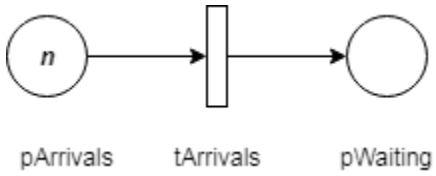

Figure 32: Arrival Module

Following code snippet shows pre-processor of transition tArrival. A for loop will run for each floor that is from minimum to maximum floor. Then it will get the value of *"global_info.ATL"* e.g *"global_info.ATL1"* from loadData() function in a variable named as "*Waiting".* If that variable is not empty then get the first index value which is the arrival time of a passenger. Then we compare current time with the arrival time of a passenger and if the current time is greater than the arrival time, then it is the time for a passenger to arrive.

As the passengers arrive it updates the person counter which saves the number of persons that has arrived yet using variable named as *"global_info.Person_No"*. The second index value of "*Waiting"* variable represents the destination floor of a passenger. It also removes the passenger from the arrival list in the loadData() function which contains its arrival time and destination floor.For example if the loadData() function contains a vector where it has two passengers detail like: *global_info.ATL1=[15 1, 20 2],* then it will remove "15 1" from the vector after its arrival. Also, three fields of global_info.PASSENGER_DATA, which is a vector to store various parameters about passengers are filled like arrival time, at which level he arrived and the destination level. Each passenger/token gets two colors as it arrives that is Lx and Px where x corresponds to a digit i.e. L1 and P1. So, its colors are L1 and P1 which means level 1 and person number 1.

```
function [fire, trans] = tArrival_pre(trans)
global global_info
%will run from minimum to maximum level
for Level = global_info.MIN_LEVEL:global_info.MAX_LEVEL
    %get the value of 'global_info.ATL' e.g global_info.ATL1 frm loadData()
    %function
    Waiting = eval(['global_info.ATL',int2str(Level+3),';']);
    if not(isempty(Waiting))
        %if not empty then get the value
        %present at first index which is passenger's arrival time
        timeOfArrival = Waiting(1);
        % if current time is greater than passenger's arrival time than
         %this is the time for passenger to arrive
        if ge(current_time, timeOfArrival)
         %update the person counter
            global_info.Person_No = global_info.Person_No + 1;
            PNo = global_info.Person_No;
            %second index value shows the destination floor of passenger
            whereTo = Waiting(2);

            % reduce the 1st and 2nd index value from global_info.ATLx
            eval(['global_info.ATL',int2str(Level+3),'=Waiting(3:end);']);
            % time at which passenger arrived
            global_info.PASSENGER_DATA(PNo, 1) = current_time;
            % At which level passenger arrived
            global_info.PASSENGER_DATA(PNo, 4) = Level;
             % To which level passenger wants to go
            global_info.PASSENGER_DATA(PNo, 5) = whereTo;
            %two colors are assigned to each token/passenger
            color1 = ['L', int2str(Level)];
            color2 = ['P', int2str(PNo)];
            trans.new_color = {color1, color2};
            fire = 1;
            disp(['Passenger Arrival: Time:', num2str(current_time),...
                '  at level:', color1, '   Person No:', color2]);
            return
        end
    end
end
fire = 0;
```

Code Snippet 75: For Pre-Processor of transition tArrival

loadData() function is related to arrival module because it contains the list of passengers that would arrive at each level which consist of arrival time and destination floor of each passenger that is going to use the elevator. Using this list, it calculates the total number of passengers beforehand. It also forms a vector /matrix named as *global_info.PASSENGER_DATA* to keep record of passengers regarding different parameters like person number, arrival time, elevator leaving time, elevator entering time, arrival floor, destination floor etc.
global_info.Travellers_at_Present is also a vector which is used to save number of passengers at any given time to make sure that elevator does not allow to carry passengers more than it capacity.

```
function [] = loadData()
global global_info

global_info.People_in_Car = 0; % initially we have 0 passengers inside car

% Passenger's Historial data: Arrival time of passengers to the lift at
% different levels. Also contain destination floor of passengers
% Test-1:
global_info.ATL1 = [20  -1]; % level -2
global_info.ATL2 = [10 -2]; % level -1
global_info.ATL3 = [30 2];  % level 0
global_info.ATL4 = [50 2];  % level 1
global_info.ATL5 = [40 1];  % level 2
passengers = 0;
%for calculating number of passengers according to historical data
for Level = global_info.MIN_LEVEL:global_info.MAX_LEVEL
    ATLx = eval(['global_info.ATL',int2str(Level+3),';']);
    length(ATLx);
    passengers = passengers + length(ATLx)/2;
end
disp(['Total number of passengers: ', int2str(passengers)]);
global_info.TOTAL_USERS = passengers;
%a matrix/ vector to hold information about passenger like person no,
%arrival and departure time, arrival and departure floor, car travelling
%status,missed car status
global_info.PASSENGER_DATA = zeros(passengers,7);
global_info.Travellers_at_Present = zeros(1, global_info.MAX_CAPACITY);
global_info.Person_No = 0; % Person counter (or person ID number)

for i = 1:global_info.TOTAL_USERS
    eval(['global_info.P2DL_', int2str(i), ' = [];']);
end
```

Code Snippet 76: For loadData() function

3.5.1.2 Initialization module:

Initialization module contain one place and a transition namely *pInit* and *tInit* . Transition *tInit* is related to one of the transitions of *Car At a Level* module in a way that it enables the transition *tOpenDoor* of *Car At a Level* module. In other words, it helps in initiating the workflow of the elevator. The token residing in place *pInit* symbolizes the elevator car and it gets initiated by firing transition *tInit* to reach the place named as *pArrivedAtLevel* which enables the transition *tOpenDoor* of *Car At a Level* module to open the elevator's door to allow passengers to step on it.

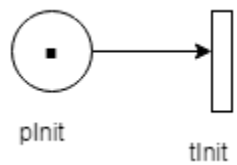

Figure 33: Initialization Module

The following code snippet shows the pre-processor of the transition *tInit* which is fired as it is or without any further condition to initiate the elevator workflow.

```
function [fire, trans] = COMMON_PRE(trans)
global global_info

switch trans.name
    case 'tInit', fire = 1;
```

Code Snippet 77: Pre-Processor for transition tInit

3.5.1.3 Car At a Level Module:

It consists of six transitions and four places namely: *tStepON, tStepOFF, tOpenDoor, tBoardingComplete, tDoorDelay, tCloseDoor and pAtLevel, pBoardingComplete, pDoorOpened, pDoorDelay.*

After the arrival of passengers in the form of tokens in the place *pWaiting* and door of the elevator is opened by transition tOpenDoor, the passenger step on it. The passenger completes the boarding procedure by pressing the button of their desired floor to reach, placed inside the elevator car. The elevator door remains open by the set duration of door delay. This happens by executing transition *tDoorDelay*. Door closes after the completion of boarding process and the set time of door delay.

Another case is that a passenger wants to step off the elevator, then after reaching its desired level/floor, door opens and he leaves the car by using transition tStepOFF.

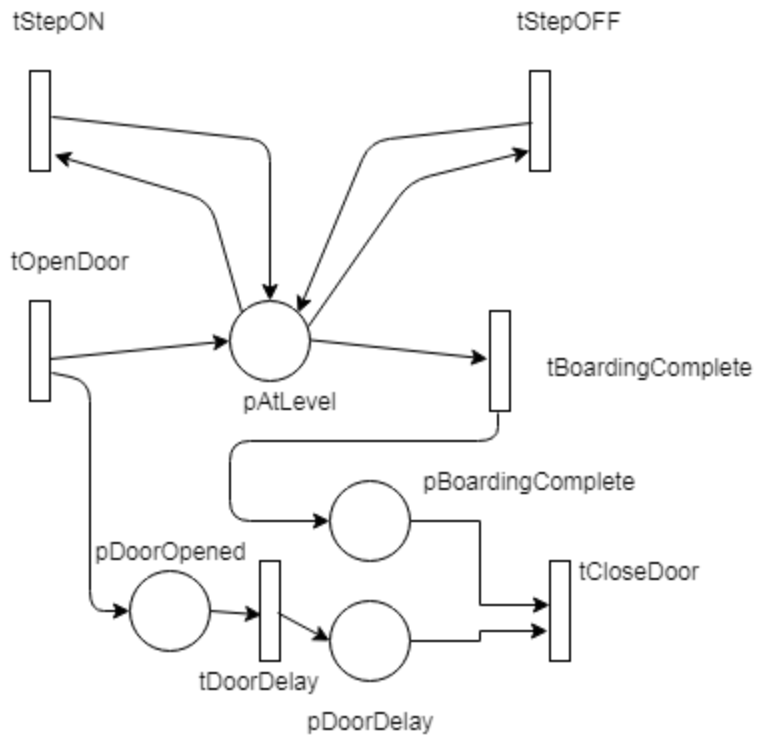

Figure 34: Car At a Level Module

The following code snippet shows pre-processor for transaction *tStepON.* As the elevator is fixed to set of floors based on time of the day that's why we first check that the value of timenow variable i.e. global_info.c is between 8 am and 5 pm. If it is then, we set the elevator to two floors: -1 and -2, which comprises of offices and commercial hubs situated on bottom floors of the building. Otherwise, the elevator is set to floors: 0, 1 & 2, which consists of residential properties and apartments in the top floors of the building.
If current time of the day is between 8 am and 5 pm then we set the elevator to two floors: -1 and -2 and further we check that if the current floor is equal to -1 or -2. In this case we proceed as follows:

If current floor is equal to -1 then, it selects a token from *pWaiting* whose color is equal to *L-1* which represents floor number -1. If there is no token with the required color then transition will not fire. Else, as we have set maximum passenger capacity of elevator equal to five that's why we check for passenger vacancy in the car. If there is a vacancy then we select the token and allow the transition to fire.Then to get the person number of the selected token, we call the function named as *separateColors(the2colors),* since a token has two colors which shows the floor number at which a passenger arrived and the person number. Also we save the time when passenger enters the car and put its travelling status to *true* in the *global_info.PASSENGER_DATA* matrix.
Otherwise, if there is no vacancy then we change priority of transition tStepON from 1 to 0 to prevent it from firing again and again. After that function filledMissedCarTable() is called to execute. This function first find all the passengers whose step on floor number is equal to current floor

number.Then check all of the passengers that whether they have already arrived and waiting or not. If a passenger has arrived and waiting then add 1 to the value of 7th column of that passenger's row in *global_info.PASSENGER_DATA* matrix, which signifies that that passenger could not step on the car because it was full to its maximum capacity.

Else If current floor is equal to -2 then, it selects a token from *pWaiting* whose color is equal to *L-2* which represents floor number -2. If there is no token with the required color then transition will not fire. Else, as we have set maximum passenger capacity of elevator equal to five that's why we check for passenger vacancy in the car. If there is a vacancy then we select the token and allow the transition to fire.Then to get the person number of the selected token, we call the function named as *separateColors(the2colors),* since a token has two colors which shows the floor number at which a passenger arrived and the person number. Also we save the time when passenger enters the car and put its travelling status to *true* in the *global_info.PASSENGER_DATA* matrix.
Otherwise, if there is no vacancy then we change priority of transition tStepON from 1 to 0 to prevent it from firing again and again. After that function filledMissedCarTable() is called to execute. This function first find all the passengers whose step on floor number is equal to current floor number.Then check all of the passengers that whether they have already arrived and waiting or not. If a passenger has arrived and waiting then add 1 to the value of 7th column of that passenger's row in *global_info.PASSENGER_DATA* matrix, which signifies that that passenger could not step on the car because it was full to its maximum capacity.

Otherwise, if current time of the day is not between 8 am and 5 pm then we set the elevator to three floors: 0, 1 and 2 and further we check that if the current floor is equal to 0, 1 or 2. In this case we proceed as follows:

If current floor is equal to 1 then, it selects a token from *pWaiting* whose color is equal to *L1* which represents floor number 1. If there is no token with the required color then transition will not fire. Else, as we have set maximum passenger capacity of elevator equal to five that's why we check for passenger vacancy in the car. If there is a vacancy then we select the token and allow the transition to fire.Then to get the person number of the selected token, we call the function named as *separateColors(the2colors),* since a token has two colors which shows the floor number at which a passenger arrived and the person number. Also we save the time when passenger enters the car and put its travelling status to *true* in the *global_info.PASSENGER_DATA* matrix.

Otherwise, if there is no vacancy then we change priority of transition tStepON from 1 to 0 to prevent it from firing again and again. After that function filledMissedCarTable() is called to execute. This function first find all the passengers whose step on floor number is equal to current floor number.Then check all of the passengers that whether they have already arrived and waiting or not. If a passenger has arrived and waiting then add 1 to the value of 7th column of that passenger's row in *global_info.PASSENGER_DATA* matrix, which signifies that that passenger could not step on the car because it was full to its maximum capacity.

Else if current floor is equal to 2 then, it selects a token from *pWaiting* whose color is equal to *L2* which represents floor number 2. If there is no token with the required color then transition will not fire. Else, as we have set maximum passenger capacity of elevator equal to five that's why we check for passenger vacancy in the car. If there is a vacancy then we select the token and allow the transition to fire.Then to get the person number of the selected token, we call the function named as *separateColors(the2colors),* since a token has two colors which shows the floor number at which a passenger arrived and the person number. Also we save the time when passenger enters the car and put its travelling status to *true* in the *global_info.PASSENGER_DATA* matrix.
Otherwise, if there is no vacancy then we change priority of transition tStepON from 1 to 0 to prevent it from firing again and again. After that function filledMissedCarTable() is called to execute. This function first find all the passengers whose step on floor number is equal to current floor number.Then check all of the passengers that whether they have already arrived and waiting or not. If a passenger has arrived and waiting then add 1 to the value of 7th column of that passenger's row in *global_info.PASSENGER_DATA* matrix, which signifies that that passenger could not step on the car because it was full to its maximum capacity.

Else if current floor is equal to 0 then, it selects a token from *pWaiting* whose color is equal to *L0* which represents floor number 0. If there is no token with the required color then transition will not fire. Else, as we have set maximum passenger capacity of elevator equal to five that's why we check for passenger vacancy in the car. If there is a vacancy then we select the token and allow the transition to fire.Then to get the person number of the selected token, we call the function named as *separateColors(the2colors),* since a token has two colors which shows the floor number at which a passenger arrived and the person number. Also we save the time when passenger enters the car and put its travelling status to *true* in the *global_info.PASSENGER_DATA* matrix.
Otherwise, if there is no vacancy then we change priority of transition tStepON from 1 to 0 to prevent it from firing again and again. After that function filledMissedCarTable() is called to execute. This function first find all the passengers whose step on floor number is equal to current floor number.Then check all of the passengers that whether they have already arrived and waiting or not. If a passenger has arrived and waiting then add 1 to the value of 7th column of that passenger's row in *global_info.PASSENGER_DATA* matrix, which signifies that that passenger could not step on the car because it was full to its maximum capacity.

```
    case 'tStepON'

         currentLevel = global_info.Current_Level;
% Modifications has been done here
%if timenow(global_info.c(4)) value is between 8 am and 5 pm, then elevator
% is fixed to floors: -1 % -2
    if (global_info.c(4)>7 && global_info.c(4)<18 )
       if eq(currentLevel,-1)
           %selecting the token whose arrival level is equal to -1
          tokID = tokenAnyColor('pWaiting', 1, ...
                      {'L-1'});
        % if there is no one waiting at this level, then do not fire
        if not(tokID)
            fire = 0; return
        end

         % Otherwise,check for vacancy in the car, if some people
         %are waiting at this level
        peopleInCar = sum(global_info.PASSENGER_DATA(:, 6));
        vacant = global_info.MAX_CAPACITY  - peopleInCar;
        if not(vacant)
            disp('** Caution! cannot take more passengers! Car is full');

             % reducing priority of "tStepON" to make it weak so that it
             %will not try again & again
            priorset('tStepON', 0);
      %holds the persons who could not enter the car because of no vacancy
            fillMissedCarTable();
            fire = 0; return
        end
        trans.selected_tokens = tokID;
        trans.override = 1;
        fire = 1;
       %using separateColors(the2colors) to get the person number to record
        %car entering time and to put travelling stauts=true, of that
        %particular person
        the2colors = get_color('pWaiting', tokID);
        PNo = separateColors(the2colors);

        % the time passenger ENTERS car
        global_info.PASSENGER_DATA(PNo, 2) = current_time;
        % the passenger is inside the car: travelling
        global_info.PASSENGER_DATA(PNo, 6) = true;
        disp(['Passenger is Entering Car: Time:', num2str(current_time),...
              '  at level:', int2str(currentLevel),...
              '   Person No:', int2str(PNo)])
```

```
% Modifications has been done here
elseif eq(currentLevel,-2)
     %selecting the token whose arrival level is equal to -2
    tokID = tokenAnyColor('pWaiting', 1, ...
                {'L-2'});
  % if there is no one waiting at this level, then do not fire
 if not(tokID)
     fire = 0; return
 end

 % Otherwise,check for vacancy in the car, if some people
  %are waiting at this level
 peopleInCar = sum(global_info.PASSENGER_DATA(:, 6));
 vacant = global_info.MAX_CAPACITY  - peopleInCar;
 if not(vacant)
     disp('** Caution! cannot take more passengers! Car is full');

    % reducing priority of "tStepON" to make it weak so that it
      %will not try again & again
     priorset('tStepON', 0);
%holds the persons who could not enter the car because of no vacancy
     fillMissedCarTable();
     fire = 0; return
 end
 trans.selected_tokens = tokID;
 trans.override = 1;
 fire = 1;
 %using separateColors(the2colors) to get the person number to record
 %car entering time and to put travelling stauts=true, of that
 %particular person
 the2colors = get_color('pWaiting', tokID);
 PNo = separateColors(the2colors);

 % the time passenger ENTERS car
 global_info.PASSENGER_DATA(PNo, 2) = current_time;
 % the passenger is inside the car: travelling
 global_info.PASSENGER_DATA(PNo, 6) = true;
 disp(['Passenger is Entering Car: Time:', num2str(current_time),...
       '  at level:', int2str(currentLevel),...
       '   Person No:', int2str(PNo)])

else
    fire=0;
end
```

```
  % Modifications has been done here
% the elevator is fixed to floors: 0, 1 & 2, because it is night time
else
    if eq(currentLevel,0)
     %selecting the token whose arrival level is equal to 0
      tokID = tokenAnyColor('pWaiting', 1, ...
                  {'L0'});
      % if there is no one waiting at this level, then do not fire
    if not(tokID)
        fire = 0; return
    end

    % Otherwise,check for vacancy in the car, if some people
     %are waiting at this level
    peopleInCar = sum(global_info.PASSENGER_DATA(:, 6));
    vacant = global_info.MAX_CAPACITY  - peopleInCar;
    if not(vacant)
        disp('** Caution! cannot take more passengers! Car is full');

         % reducing priority of "tStepON" to make it weak so that it
         %will not try again & again
        priorset('tStepON', 0);
   %holds the persons who could not enter the car because of no vacancy
        fillMissedCarTable();
        fire = 0: return
    end
    trans.selected_tokens = tokID;
    trans.override = 1;
    fire = 1;
  %using separateColors(the2colors) to get the person number to record
    %car entering time and to put travelling stauts=true, of that
    %particular person
    the2colors = get_color('pWaiting', tokID);
    PNo = separateColors(the2colors);

    % the time passenger ENTERS car
    global_info.PASSENGER_DATA(PNo, 2) = current_time;
    % the passenger is inside the car: travelling
    global_info.PASSENGER_DATA(PNo, 6) = true;
    disp(['Passenger is Entering Car: Time:', num2str(current_time),...
          '  at level:', int2str(currentLevel),...
          '   Person No:', int2str(PNo)])
```

```
 % Modifications has been done here
elseif eq(currentLevel,1)
    %selecting the token whose arrival level is equal to 1
    tokID = tokenAnyColor('pWaiting', 1, ...
                {'L1'});
 % if there is no one waiting at this level, then do not fire
 if not(tokID)
     fire = 0; return
 end

 % Otherwise,check for vacancy in the car, if some people
  %are waiting at this level
 peopleInCar = sum(global_info.PASSENGER_DATA(:, 6));
 vacant = global_info.MAX_CAPACITY  - peopleInCar;
 if not(vacant)
     disp('**Caution! cannot take more passengers! Car is full');

    % reducing priority of "tStepON" to make it weak so that it
      %will not try again & again
     priorset('tStepON', 0);
%holds the persons who could not enter the car because of no vacancy
     fillMissedCarTable();
     fire = 0; return
 end
 trans.selected_tokens = tokID;
 trans.override = 1;
 fire = 1;
%using separateColors(the2colors) to get the person number to record
 %car entering time and to put travelling stauts=true, of that
 %particular person
 the2colors = get_color('pWaiting', tokID);
 PNo = separateColors(the2colors);

 % the time passenger ENTERS car
 global_info.PASSENGER_DATA(PNo, 2) = current_time;
 % the passenger is inside the car: travelling
 global_info.PASSENGER_DATA(PNo, 6) = true;
 disp(['Passenger is Entering Car: Time:', num2str(current_time),...
       '  at level:', int2str(currentLevel),...
       '   Person No:', int2str(PNo)])
```

```
    % Modifications has been done here
    elseif eq(currentLevel,2)
        %selecting the token whose arrival level is equal to 2
       tokID = tokenAnyColor('pWaiting', 1, ...
                    {'L2'});
    % if there is no one waiting at this level, then do not fire
     if not(tokID)
         fire = 0; return
     end

      % Otherwise,check for vacancy in the car, if some people
      %are waiting at this level
     peopleInCar = sum(global_info.PASSENGER_DATA(:, 6));
     vacant = global_info.MAX_CAPACITY  - peopleInCar;
     if not(vacant)
         disp('** Caution! cannot take more passengers! Car is full');

        % reducing priority of "tStepON" to make it weak so that it
          %will not try again & again
         priorset('tStepON', 0);
   %holds the persons who could not enter the car because of no vacancy
         fillMissedCarTable();
         fire = 0; return
     end
     trans.selected tokens = tokID;
     trans.override = 1;
     fire = 1;
    %using separateColors(the2colors) to get the person number to record
     %car entering time and to put travelling stauts=true, of that
     %particular person
     the2colors = get_color('pWaiting', tokID);
     PNo = separateColors(the2colors);

     % the time passenger ENTERS car
     global_info.PASSENGER_DATA(PNo, 2) = current_time;
     % the passenger is inside the car: travelling
     global_info.PASSENGER_DATA(PNo, 6) = true;
     disp(['Passenger is Entering Car: Time:', num2str(current_time),...
           '  at level:', int2str(currentLevel),...
           '   Person No:', int2str(PNo)])
    else
        fire=0;
     end
end
```

Code Snippet 78: Pre-Processor for transition tStepON

```
% function for separating two colors of the token to get the person number
% of a passenger

function [PNo] = separateColors(the2colors)

color1 = the2colors{1};
color2 = the2colors{2};

% To find at which Level he arrived and also his passenger number
if strcmp(color1(1), 'L')
    fromWhichLevel = color1;
    PNo = color2;
else
    fromWhichLevel = color2;
    PNo = color1;
end

PNo = PNo(2:end);
PNo = str2double(PNo);
```

Code Snippet 79: For separateColors(the2colors) function

```
%function for saving information about passengers who could not enter
%elevator because it was already full to its max capacity
function [] = fillMissedCarTable()
global global_info

[indices_for_this_level, ~] = ...
    find(global_info.PASSENGER_DATA(:,4)== global_info.Current_Level);

for i = 1:numel(indices_for_this_level)
    row_index = indices_for_this_level(i);

    if not(global_info.PASSENGER_DATA(row_index, 2))
        % shows that passenger has arrived and waiting to step on!!!
        global_info.PASSENGER_DATA(row_index,7) = ...
            global_info.PASSENGER_DATA(row_index,7) + 1;
    end
end
```

Code Snippet 80: For fillMissedCarTable() function

The following code snippet shows pre-processor for transition *tStepOFF.* First it gets sum of all the passengers travelling inside the car. If the car is empty it will not fire. Otherwise, it will compare the destination level of all the passengers present in the car one by one with the current floor level. If there is a match, then that passenger leaves the car. Also, the value of sixth column of PASSENGER_DATA matrix which shows travelling status of a passenger inside the car is set to zero that is false, of the passenger who leaves.

```
case 'tStepOFF'

     % it shows people who are currently inside car, means travelling
    current_passengers = global_info.PASSENGER_DATA(:, 6);
    peopleInCar = sum(current_passengers);
    if not(peopleInCar), fire = 0; return, end % not allowed to fire if the car is empty
    [~, PNoInCar] = find(transpose(current_passengers));
    for i = 1:numel(PNoInCar)
        PNo = PNoInCar(i);
        whereTo = global_info.PASSENGER_DATA(PNo, 5);
        %if current level is equal to destination level of specified person,then that person leaves
        if eq(global_info.Current_Level, whereTo)

           % set travelling in the car status of that person equal to
            % 0, as he is leaving
            global_info.PASSENGER_DATA(PNo, 6) = 0;
            % saving the leaving time from CAR
            global_info.PASSENGER_DATA(PNo, 3) = current_time;
            disp(['Passenger is Leaving Car: Time:', num2str(current_time),...
                '  at level:', int2str(whereTo),...
                '   Person No:', int2str(PNo)]);
            fire = 1; return
        end
    end
    fire = 0;
```

Code Snippet 81: Pre-Processor for transition tStepOFF

The following code snippet shows pre-processor for transition *tOpenDoor* which is fired without any further condition.

```
case 'tOpenDoor', fire = 1;
```

Code Snippet 82: Pre-Processor for transition tOpenDoor

The following code snippet shows post-processor for transition *tOpenDoor* in which a message is being displayed on console which informs that the elevator has arrived on a specific floor at a specific time.

```
case 'tOpenDoor'
    disp(' ');
    disp(['****Elevator Arrived at Level ',...
          int2str(global_info.Current_Level), ...
          '     at time: ', num2str(current_time), ' *******']);
```

Code Snippet 83: Post-Processor for transition tOpenDoor

The following code snippet shows pre-processor for transition *tBoardingComplete* in which we are setting transition priority of *tStepON* back to 1 from 0 using *priorset*() function and allowing the transition to fire.

```
case 'tBoardingComplete'
    % increasing  priority of "tStepON" back to 1 as it was set to
    %0 previously in "tStepON"
    priorset('tStepON', 1);
    fire = 1;
```

Code Snippet 84: Pre-Processor for transition tBoardingComplete

The following code snippet shows pre-processor for transition *tDoorDelay* in which we are overriding the transition and allowing it to fire.

```
case 'tDoorDelay', trans.override = 1; fire = 1;
```

Code Snippet 85: Pre-Processor for transition tDoorDelay

The following code snippet shows pre-processor for transition *tCloseDoor* which displays a console message that the door is getting closed at current level and current time and allowing the transition to fire.

```
case 'tCloseDoor'
    disp(['**** Closing the door now at Level ',...
        int2str(global_info.Current_Level), ...
        '     at time: ', num2str(current_time), ' *******']);
    disp(' ');
    fire = 1;
```

Code Snippet 86: Pre-Processor for transition tCloseDoor

3.5.1.4 Departure module:

This module consists of a transition and a place namely: *tLeaving* and *pLeft.* After transition *tStepOFF* at *Car At a Level* module to exit the elevator, a passenger leaves using transition *tLeaving.*

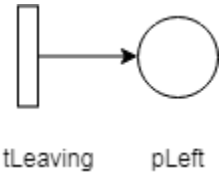

Figure 35: Departure Module

The following code snippet shows pre-processor for transition *tLeaving* which is getting fired without any further condition.

```
    case 'tLeaving', fire = 1;
```

Code Snippet 87: Pre-Processor for transition tLeaving

The following code snippet shows post-processor for transition *tLeaving* in which we get the number of tokens in "*pLeft*". Then we compare number of tokens in "pLeft" with the total number of passengers and don't stop the simulation until all the passengers have left the elevator or have reached their respective destination floors.

```
    case 'tLeaving'
        %gets the number of token in 'pLeft'
        n = ntokens('pLeft');
%comparing number of tokens in 'pLeft' with total number of passengers
        if eq(n, global_info.TOTAL_USERS)
        %don't stop the simulation untill all the passengers have
        %reached their destination floors
            global_info.STOP_SIMULATION = true;
        end
```

Code Snippet 88: Post-Processor for transition tLeaving

3.5.1.5 Car Moving module:

This module contains three transitions and a place namely: *tAccel, tMoving, tDeAccel* and *pMoving*.

After closing door at Car At a Level module, elevator accelerates using transition *tAccel* to take passengers to their destination floors. Transition *tMoving* fire only once and moves the car by one level only because this algorithm is using Shabbat movement that is, the elevator stops at every floor.

Finally, after reaching the destination floor it comes to a halt by using transition *tDeAccel.* Otherwise, if the next floor is the destination floor then it uses *tMoving* once and stops using *tDeAccel.*

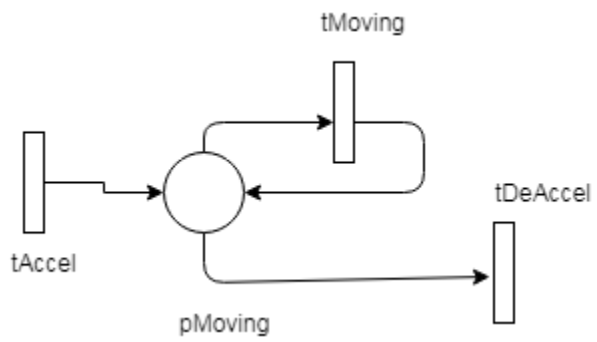

Figure 36: Car Moving Module

The following code snippet shows code of pre-processor of transition *tAccel*. It implements the actual shabbat movement regarding the movement of elevator's car. According to which, if current direction of the car is upwards then it adds 1 to the value of current level and further it checks whether current level value is greater than maximum level value, if yes then it subtracts 2 from the current level value and change the elevator's direction to downwards.

Otherwise, if the elevator's current direction is towards downwards then it subtracts 1 from the value of current level and further it checks whether current level value is smaller than minimum level value, if yes then it adds 2 to the current level value and change the elevator's direction to upwards.

```
case 'tAccel' % the main algorithm for moving car to destination !!!
    % It follows simple Shabbat Movement
    if strcmp(global_info.Current_Direction, 'UP')
        global_info.Current_Level = global_info.Current_Level + 1;
        if gt(global_info.Current_Level, global_info.MAX_LEVEL)
            global_info.Current_Level = global_info.Current_Level - 2;
            global_info.Current_Direction = 'DOWN';
        end
    else % global_info.Current_Direction == 'DOWN')
        global_info.Current_Level = global_info.Current_Level - 1;
        if lt(global_info.Current_Level, global_info.MIN_LEVEL)
            global_info.Current_Level = ...
                   global_info.Current_Level + 2;
            global_info.Current_Direction = 'UP';
        end
    end


    global_info.Starting_to_move = true; % will be used by "tMoving"
    fire = 1;
```

Code Snippet 89: Pre-Processor for transition tAccel

The following code snippet shows code of pre-processor of transition *tDeAccel* which is fired without any further conditions.

```
case 'tDeAccel'
    fire = 1;
```

Code Snippet 90: Pre-Processor for transition tDeAccel

The following code snippet shows code of pre-processor of transition *tMoving* where it is allowed to fire when the bool value of *global_info.Starting_to_move* is set to *true, previously in tAccel. After that we set it to false* because elevator is allowed to move only once in Shabbat movement i.e. movement by one floor only. Otherwise, it's not allowed to fire if previously in *tAccel* the value of *global_info.Starting_to_move* is not set to true.

```
% moves car by one level; tMoving fires only once for shabbat movement
case 'tMoving'
    if global_info.Starting_to_move
        global_info.Starting_to_move = false;
        fire = 1;
    else
        fire = 0;
    end
```

Code Snippet 91: Pre-Processor for transition tMoving

3.5.1.6 Inter modular connections:

Following diagram shows the inter module connections which integrates all modules to make a single model of an elevator. There are three prominent places in it which helps to connects all the modules together i.e. *pArrivedAtLevel*, *pAboutToMove* and *pLeaving*.

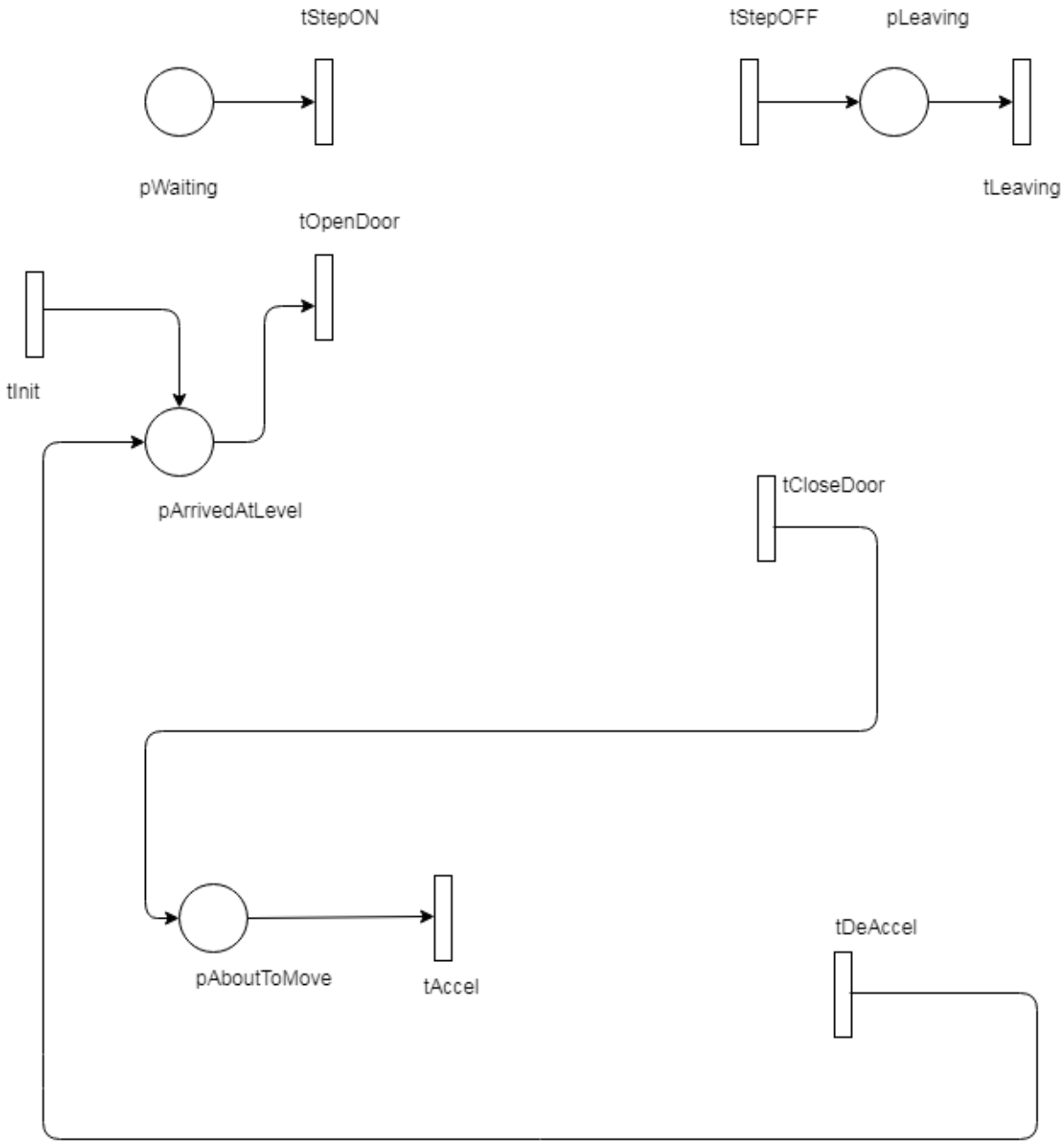

Figure 37: Inter Module Connections

4. Testing, Analysis and Results:

4.1 Simple User Manual:

4.1.1 For Shabbat, FCFS and fixed sectoring :

1. First install MATLAB & GpenSIM
2. Unzip the file named as “Code Files.7z”
3. It will have a folder named as “Code Files”
4. Open this folder to run simulations
5. Open the main simulation file named as “msf.m”
6. You can make changes to arrival times and destination floors of the passengers in the loadData() function as shown in the following Code Snippet 92. You can also add/remove the passengers by adding/removing the arrival time and destination floor of a passenger on different levels in loadData() function.
7. You can change the minimum and maximum floors of the elevator in the “msf.m” file as shown in the Code Snippet 93 below. Also, passenger holding capacity of the elevator can be modified.
8. Run it after making desired changes

```
global_info.ATL1 = [20  0]; % level -2
global_info.ATL2 = [10 2]; % level -1
global_info.ATL3 = [28 2];  % level 0
global_info.ATL4 = [15 -2];  % level 1
global_info.ATL5 = [22 -1];  % level 2
```

Code Snippet 92: For changing arrival time and destination floor

```
global_info.MAX_CAPACITY = 5;    % Maximum capacity to hold passengers in car
global_info.MIN_LEVEL = -2;      % Bottom level floor
global_info.MAX_LEVEL = 2;       % Highest/top level floor
```

Code Snippet 93: For changing passenger holding capacity and (min,max) floors

4.1.2 For FCFS with VIP passcode:

1. First install MATLAB & GpenSIM
2. Unzip the file named as “Code Files.7z”
3. It will have a folder named as “Code Files”
4. Open this folder to run simulations
5. Open the main simulation file named as “msf.m”
6. You can make changes to arrival times, destination floors and passcode holding status of the passengers in the loadData() function as shown in the following Code Snippet 94. You can also add/remove the passengers by adding/removing the arrival time, destination floor and passcode holding status of a passenger on different levels in loadData() function.
7. You can change the minimum and maximum floors of the elevator in the “msf.m” file as shown in the Code Snippet 95 below. Also, passenger holding capacity of the elevator can be modified.
8. Run it after making desired changes

```
global_info.ATL1 = [20  0 0]; % level -2
global_info.ATL2 = [10 2 1,20 1 1,24 -2 1]; % level -1
global_info.ATL3 = [26 2 0];  % level 0
global_info.ATL4 = [15 -2 1, 18 -1 1];  % level 1
global_info.ATL5 = [22 -1 0];  % level 2
```

Code Snippet 94: For changing arrival time, destination floor and passcode holding status

```
global_info.MAX_CAPACITY = 5;    % Maximum capacity to hold passengers in car
global_info.MIN_LEVEL = -2;      % Bottom level floor
global_info.MAX_LEVEL = 2;       % Highest/top level floor
```

Code Snippet 95: For changing passenger holding capacity and (min,max) floors

4.1.3 For Dynamic sectoring:

1. First install MATLAB & GpenSIM
2. Unzip the file named as “Code Files.7z”
3. It will have a folder named as “Code Files”
4. Open this folder to run simulations
5. Open the main simulation file named as “*msf.m*”
6. You can make changes to arrival times and destination floors of the passengers in the loadData() function as shown in the following Code Snippet 96. You can also add/remove

the passengers by adding/removing the arrival time and destination floor of a passenger on different levels in loadData() function.

7. You can change the minimum and maximum floors for the nighttime section of floors in the "msf.m" file as shown in the Code Snippet 97 below. Also, passenger holding capacity of the elevator can be modified in the msf.m file as shown in Code Snippet 98.
8. Run it after making desired changes

```
global_info.ATL1 = [20 -1, 25 -1, 30 -1]; % level -2
global_info.ATL2 = [10 -2, 15 -2, 20 -2]; % level -1
global_info.ATL3 = [30 2, 35 1];  % level 0
global_info.ATL4 = [50 2, 55 0];  % level 1
global_info.ATL5 = [40 1, 45 0];  % level 2
```

Code Snippet 96: For changing arrival time, and destination floor

```
disp("Its Night time");disp('');
disp("Elevator is fixed to top floors: 0,1 & 2");disp('');
global_info.MIN_LEVEL = 0;   % Bottom floor
global_info.MAX_LEVEL = 2;   % top floor
```

Code Snippet 97: For changing min & max floors

```
global_info.MAX_CAPACITY = 5;   % Maximum capacity to hold passengers in car
```

Code Snippet 98: For changing passenger holding capacity

4.2 Sample Run:

4.2.1 Shabbat Algorithm:

Following plot shows five passengers, who travelled in an elevator in the form of tokens. It shows the time at which each passenger left or reached his desired floor in an elevator which is using shabbat algorithm for the movement of car. Blue line of tokens represents arrival time of passengers at various floors, as stated in the loadData() function of the code which ranges from 10 to 28 TU as shown in plot. After arrival of passengers in the form of tokens at different floors, they reside in a place called *pWaiting* to step on the elevator as it arrives. It is represented by orange line. It continues to rise until 28.15 TU but at 43.25 TU it starts to decrease/fall, which means passenger tokens are waiting to step on until 28.15 TU but at 43.25 TU they start to enter the elevator to reach to their destination floors. Yellow line represents a place named as *pATLevel*, which shows the time at which elevator has stopped at various number of floors. Finally, purple line represents the time at which each passenger has reached to its desired destination floor. The number of tokens in *pLeft* increases as the time increases. According to plot, leaving time of passengers are as follows: 195.3 TU, 233.3 TU, 309.525 TU, 385.825 TU, 386.85 TU

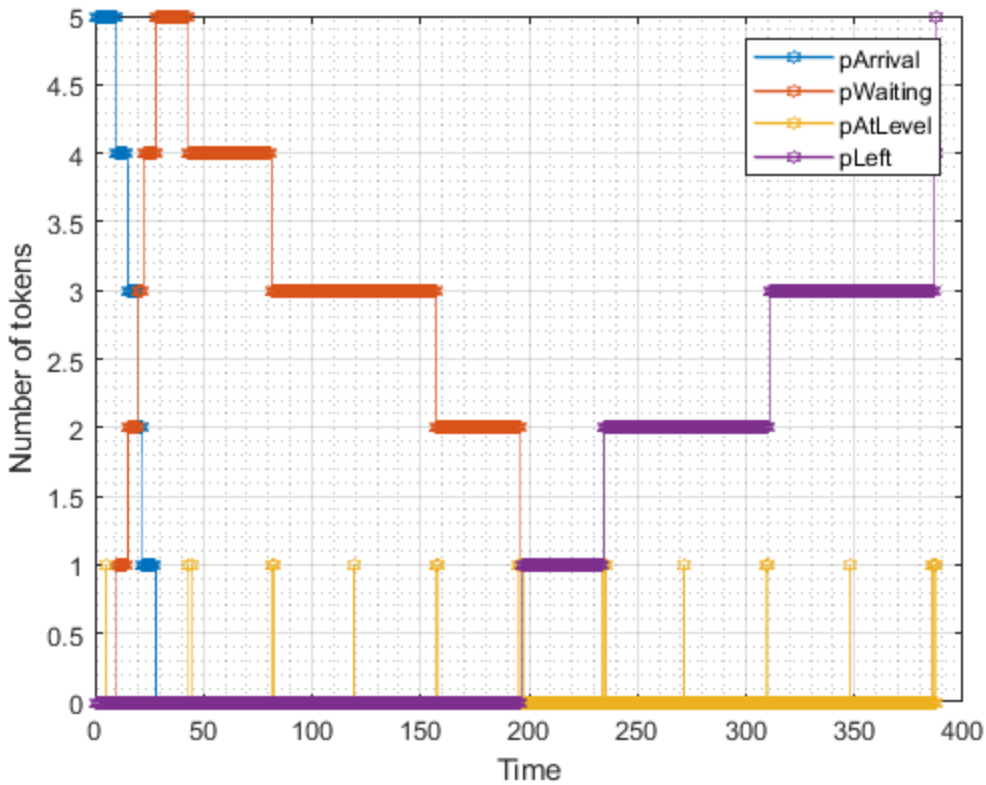

Figure 38: Sample Run for Shabbat Algorithm

Following historical data of passengers is used for the generation of above plot, who arrived at different levels.

```
global_info.ATL1 = [20  0]; % level -2
global_info.ATL2 = [10 2]; % level -1
global_info.ATL3 = [28 2];  % level 0
global_info.ATL4 = [15 -2];  % level 1
global_info.ATL5 = [22 -1];  % level 2
```

Code Snippet 99: From loadData() function

4.2.2 First Come First Serve elevator algorithm:

Following plot shows five passengers, who travelled in an elevator in the form of tokens. It shows the time at which each passenger left or reached his desired floor in an elevator which is using First Come First Serve elevator algorithm for the movement of car. Blue line of tokens represents arrival time of passengers at various floors, as stated in the loadData() function of the code which ranges from 10 to 28 TU as shown in plot. After arrival of passengers in the form of tokens at different floors, they reside in a place called *pWaiting* to step on the elevator as it arrives. It is represented by orange line. It continues to rise until 28.15 TU but at 43.25 TU it starts to decrease/fall, which means passenger tokens are waiting to step on until 28.15 TU but at 43.25 TU they start to enter the elevator to reach to their destination floors. Yellow line represents a place named as *pATLevel*, which shows the time at which elevator has stopped at various number of floors. Finally, purple line represents the time at which each passenger has reached to its desired destination floor. The number of tokens in *pLeft* increases as the time increases. According to plot, leaving time of passengers are as follows: 119.3 TU, 157.3 TU, 195.3 TU, 233.3 TU, 347.675 TU.

Now, if we compare the leaving times of passengers presented by both Shabbat and FCFS algorithm then we get to know that in FCFS passengers leave the car considerably earlier as compare to Shabbat algorithm which saves the passenger's time and that's how the elevator can serve more people in less time.

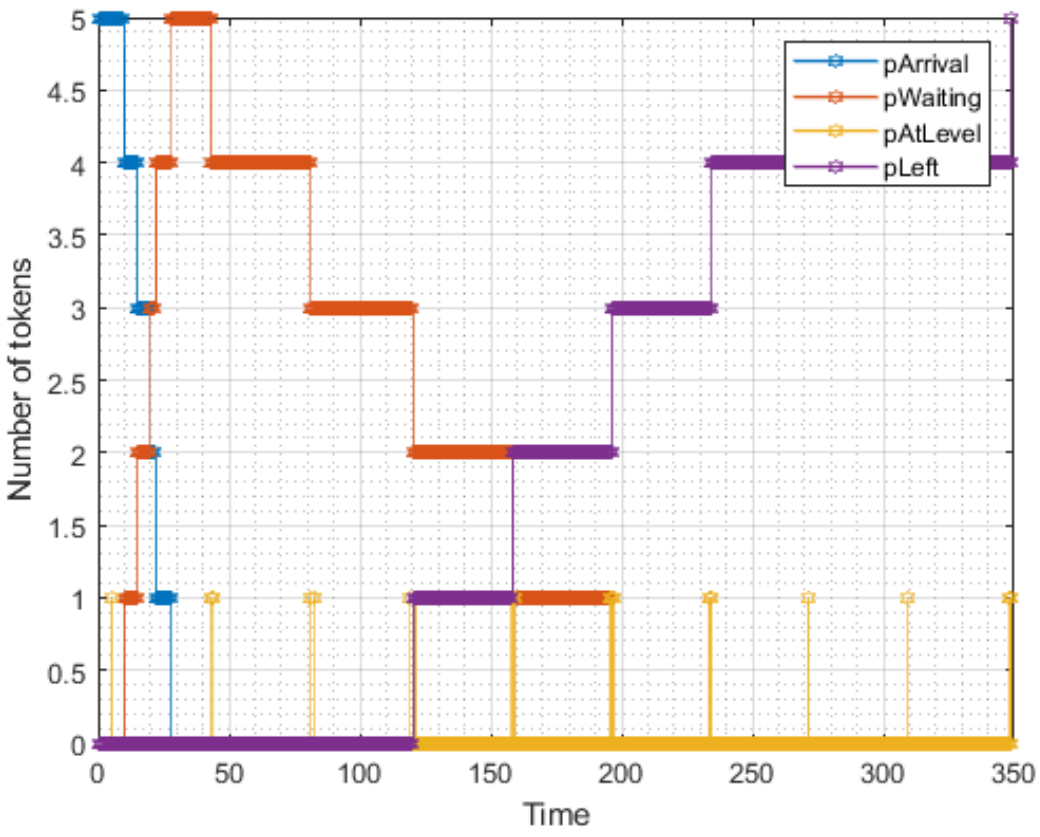

Figure 39: Sample Run for First Come First Serve elevator algorithm

Following historical data of passengers is used for the generation of above plot, who arrived at different levels.

```
global_info.ATL1 = [20  0]; % level -2
global_info.ATL2 = [10 2]; % level -1
global_info.ATL3 = [28 2];  % level 0
global_info.ATL4 = [15 -2];  % level 1
global_info.ATL5 = [22 -1];  % level 2
```

Code Snippet 100: From loadData() function

4.2.3 Fixed sectoring elevator algorithm:

Following plot shows six passengers, who travelled in an elevator in the form of tokens. It shows the time at which each passenger left or reached his desired floor in an elevator which is using Shabbat algorithm for the movement of car. Blue line of tokens represents arrival time of passengers at various floors, as stated in the loadData() function of the code which ranges from 30 to 55 TU as shown in plot. After arrival of passengers in the form of tokens at different floors, they reside in a place called *pWaiting* to step on the elevator as it arrives. It is represented by orange line. It continues to rise until 55.15 TU but at 81.3 TU it starts to decrease/fall, which means passenger tokens are waiting to step on until 55.15 TU but at 81.3 TU they start to enter the elevator to reach to their destination floors. Yellow line represents a place named as *pATLevel*, which shows the time at which elevator has stopped at various number of floors. Finally, purple line represents the time at which each passenger has reached to its desired destination floor. The number of tokens in *pLeft* increases as the time increases. According to plot, leaving time of passengers are as follows: 119.3 TU, 157.3 TU, 158.3 TU, 195.3 TU, 233.3 TU, 234.3 TU.

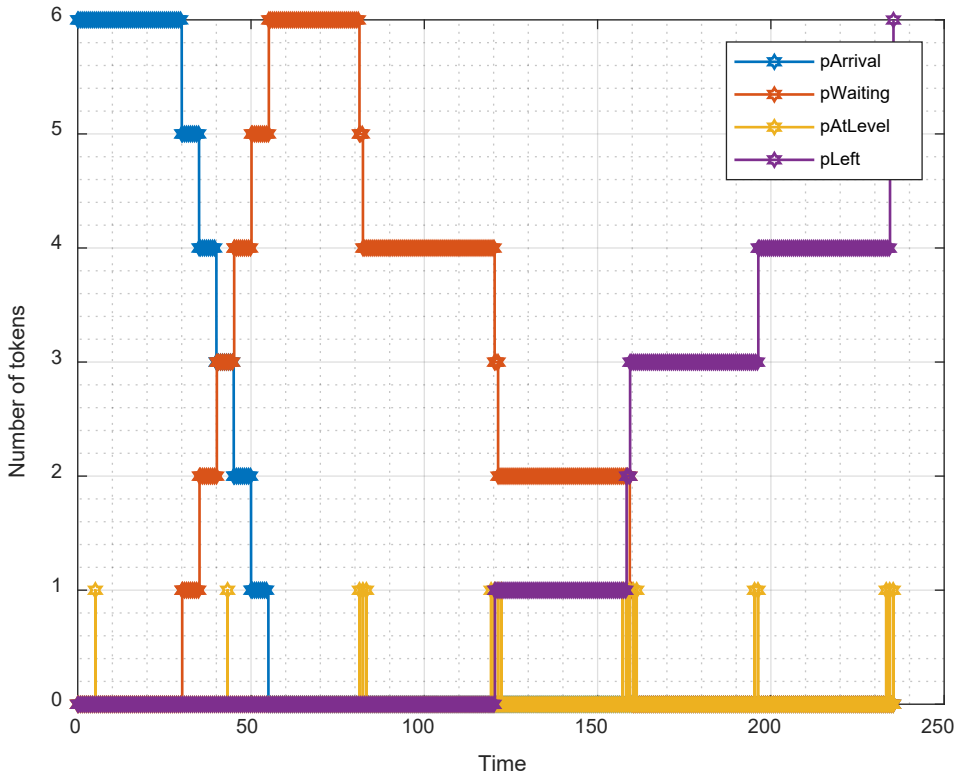

Figure 40: Sample Run for Fixed sectoring elevator algorithm

Following historical data of passengers is used for the generation of above plot, who arrived at different levels.

```
global_info.ATL1 = [20  0]; % level -2
global_info.ATL2 = [10 2]; % level -1
global_info.ATL3 = [30 2, 35 1];  % level 0
global_info.ATL4 = [50 2, 55 0];  % level 1
global_info.ATL5 = [40 1, 45 0];  % level 2
```

Code Snippet 101: From loadData() function

Since, this algorithm is all about fixed sectoring so, two passengers who arrived at level -1 and -2 respectively, could not enter the elevator because we have fixed the elevator to three floors i.e. from 0 to 2, as shown in the above historical data.

4.2.4 Dynamic sectoring elevator algorithm:

4.2.4.1 Night-time sample run:

Following plot shows six passengers, who travelled in an elevator in the form of tokens. It shows the time at which each passenger left or reached his desired floor in an elevator which is using Shabbat algorithm for the movement of car. Blue line of tokens represents arrival time of passengers at various floors, as stated in the loadData() function of the code which ranges from 10 to 55 TU as shown in plot. After arrival of passengers in the form of tokens at different floors, they reside in a place called *pWaiting* to step on the elevator as it arrives. It is represented by orange line. It continues to rise until 55.15 TU but at 81.3 TU it starts to decrease/fall, which means passenger tokens are waiting to step on until 55.15 TU but at 81.3 TU they start to enter the elevator to reach to their destination floors. Yellow line represents a place named as *pATLevel*, which shows the time at which elevator has stopped at various number of floors. Finally, purple line represents the time at which each passenger has reached to its desired destination floor. The number of tokens in *pLeft* increases as the time increases. According to plot, leaving time of passengers are as follows: 119.3 TU, 157.3 TU, 158.3 TU, 195.3 TU, 233.3 TU, 234.3 TU.

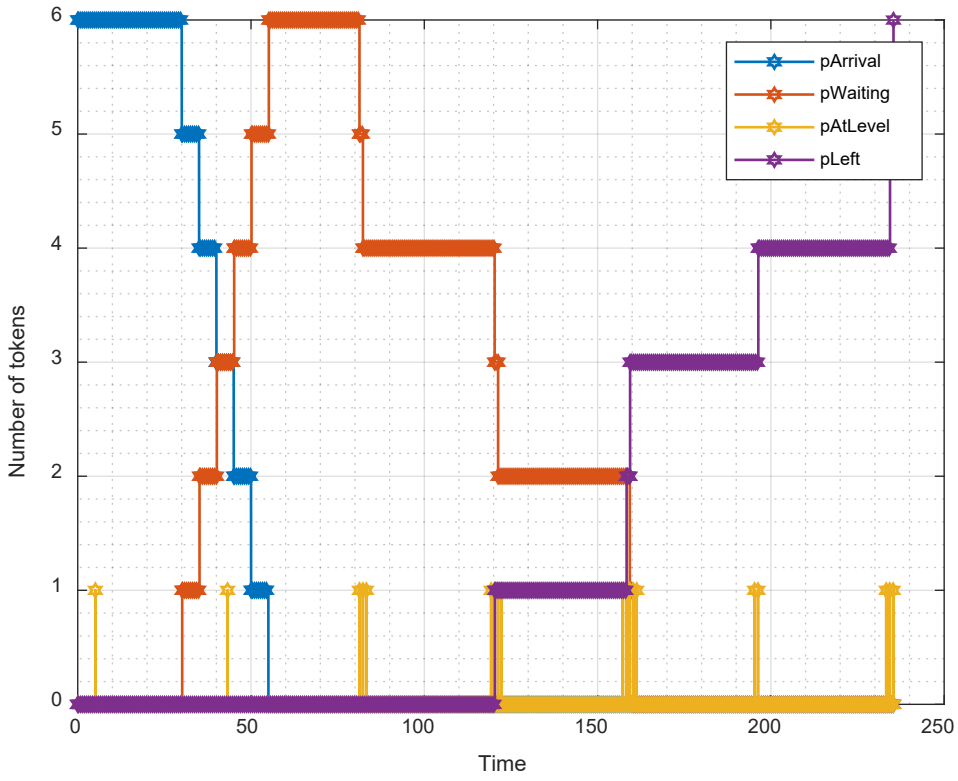

Figure 41: Sample Run for Dynamic sectoring elevator algorithm (Night-time sample run)

Following historical data of passengers is used for the generation of above plot, who arrived at different levels.

```
global_info.ATL1 = [20 -1, 25 -1, 30 -1]; % level -2
global_info.ATL2 = [10 -2, 15 -2, 20 -2]; % level -1
global_info.ATL3 = [30 2, 35 1];  % level 0
global_info.ATL4 = [50 2, 55 0];  % level 1
global_info.ATL5 = [40 1, 45 0];  % level 2
```

Code Snippet 102: From loadData() function

Since, this algorithm is all about dynamic sectoring so, six passengers who arrived at level -1 and -2 respectively, could not enter the elevator because the elevator has dynamically fixed itself according to the time of the day, to three floors i.e. from 0 to 2, as shown in the above historical data. This particular sample run represents the night-time when elevator is fixed for residential use i.e. from 5 PM to 8 AM in the morning.

4.2.4.2 Morning time sample run:

Following plot shows six passengers, who travelled in an elevator in the form of tokens. It shows the time at which each passenger left or reached his desired floor in an elevator which is using Shabbat algorithm for the movement of car. Blue line of tokens represents arrival time of passengers at various floors, as stated in the loadData() function of the code which ranges from 10 to 30 TU as shown in plot. After arrival of passengers in the form of tokens at different floors, they reside in a place called *pWaiting* to step on the elevator as it arrives. It is represented by orange line. It continues to rise until 30.15 TU but at 43.25 TU it starts to decrease/fall, which means passenger tokens are waiting to step on until 30.15 TU but at 43.25 TU they start to enter the elevator to reach to their destination floors. Yellow line represents a place named as *pATLevel*, which shows the time at which elevator has stopped at various number of floors. Finally, purple line represents the time at which each passenger has reached to its desired destination floor. The number of tokens in *pLeft* increases as the time increases. According to plot, leaving time of passengers are as follows: 81.3 TU, 82.3 TU, 83.3 TU, 119.3 TU, 120.3 TU, 121.3 TU.

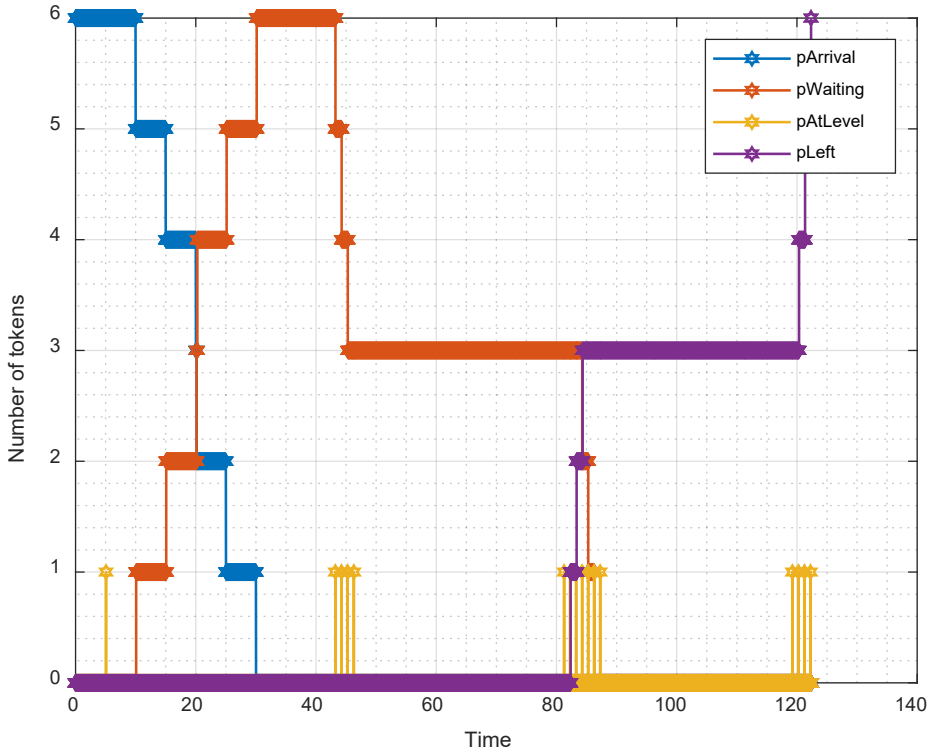

Figure 42: Sample Run for Dynamic sectoring elevator algorithm (Morning time sample run)

Following historical data of passengers is used for the generation of above plot, who arrived at different levels.

```
global_info.ATL1 = [20 -1, 25 -1, 30 -1]; % level -2
global_info.ATL2 = [10 -2, 15 -2, 20 -2]; % level -1
global_info.ATL3 = [30 2, 35 1];  % level 0
global_info.ATL4 = [50 2, 55 0];  % level 1
global_info.ATL5 = [40 1, 45 0];  % level 2
```

Code Snippet 103: From loadData() function

Since, this algorithm is all about dynamic sectoring so, six passengers who arrived at level 0, 1 and 2 respectively, could not enter the elevator because the elevator has dynamically fixed itself according to the time of the day, to two floors i.e. from -1 to -2, as shown in the above historical data. This particular sample run represents the morning-time when elevator is fixed for commercial use i.e. from 8 AM to 5 PM in the evening.

4.2.5 First Come First Serve elevator algorithm with VIP passcode:

Following plot shows five passengers, who travelled in an elevator in the form of tokens. It shows the time at which each passenger left or reached his desired floor in an elevator which is using First Come First Serve elevator algorithm for the movement of car. Blue line of tokens represents arrival time of passengers at various floors, as stated in the loadData() function of the code which ranges from 10 to 26 TU as shown in plot. After arrival of passengers in the form of tokens at different floors, they reside in a place called *pWaiting* to step on the elevator as it arrives. It is represented by orange line. It continues to rise until 26.15 TU but at 43.25 TU it starts to decrease/fall, which means passenger tokens are waiting to step on until 26.15 TU but at 43.25 TU they start to enter the elevator to reach to their destination floors. Yellow line represents a place named as *pATLevel*, which shows the time at which elevator has stopped at various number of floors. Finally, purple line represents the time at which each passenger has reached to its desired destination floor. The number of tokens in *pLeft* increases as the time increases. According to plot, leaving time of passengers are as follows: 81.3 TU, 157.3 TU, 158.3 TU, 195.3 TU, 233.3 TU.

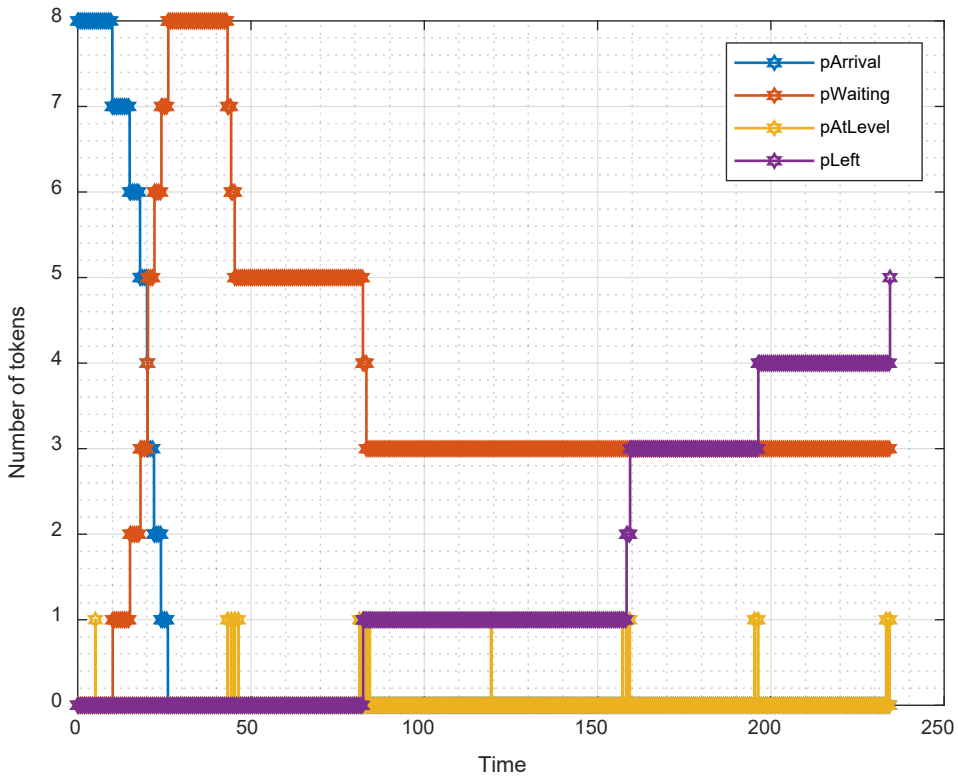

Figure 43: Sample Run for First Come First Serve elevator algorithm with VIP passcode

Following historical data of passengers is used for the generation of above plot, who arrived at different levels.

```
global_info.ATL1 = [20  0 0]; % level -2
global_info.ATL2 = [10 2 1,20 1 1,24 -2 1]; % level -1
global_info.ATL3 = [26 2 0];  % level 0
global_info.ATL4 = [15 -2 1, 18 -1 1];  % level 1
global_info.ATL5 = [22 -1 0];  % level 2
```

Code Snippet 104: From loadData() function

Since, according to this algorithm only the VIPs or the authorized persons can use and enter the elevator that's why three of the arrived passenger could not board the elevator because they were not permitted to do that as they did not possess the required passcode. Among these three passengers include, who arrived at level -2, 0 and 2 as shown in the above historical data.

4.3 Data Description:

I have used cooked up data in the form of passenger's arrival time, arrival floor and destination floor because of the absence of real and authentic data of an elevator. But my proposed algorithms can still work, if they are provided with real data to gauge various aspects related to the performance of an elevator.

4.4 Statistical Analysis:

4.4.1 Shabbat Algorithm:

Arrival Time	Arrival Floor	Destination Floor	Pickup Time	Leaving Time
10	-1	2	196.3	385.8
15	1	-2	43.2	233.3
20	-2	0	234.3	309.5
22	2	-1	81.3	195.3
28	0	2	157.3	386.8

Table 1: Simulation Results for Shabbat Algorithm

Arrival Time	Δ Floor	Turnaround Time (ToT)	Travel Time inside car (CarT)	Ideal Travel Time (ITT)	Efficiency (ITT/ToT)
10	3	375.8	189	31	8%
15	-3	218.3	190.1	31	14%
20	2	289.3	75.2	24	8%
22	-3	173.3	114	31	18%
28	2	358.3	229.5	31	9%

Table 2: Statistical Analysis for Shabbat Algorithm

4.4.2 First Come First Serve elevator algorithm:

Arrival Time	Arrival Floor	Destination Floor	Pickup Time	Leaving Time
10	-1	2	43.2	157.3
15	1	-2	81.3	119.3
20	-2	0	120.3	195.3
22	2	-1	158.3	347.6
28	0	2	196.3	233.3

Table 3: Simulation Results for First Come First Serve elevator algorithm

Arrival Time	Δ Floor	Turnaround Time (ToT)	Travel Time inside car (CarT)	Ideal Travel Time (ITT)	Efficiency (ITT/ToT)
10	3	147.3	114.1	31	27%
15	-3	104.3	38	31	30%
20	2	175.3	75	24	14%
22	-3	325.6	189.3	31	10%
28	2	205.3	37	24	12%

Table 4: Statistical Analysis for First Come First Serve elevator algorithm

4.4.3 Fixed sectoring elevator algorithm:

Arrival Time	Arrival Floor	Destination Floor	Pickup Time	Leaving Time
30	0	2	159.3	233.3
35	0	1	160.3	195.3
40	2	1	81.3	119.3
45	2	0	82.3	157.3
50	1	2	120.3	234.3
55	1	0	121.3	158.3

Table 5: Simulation Results for Fixed sectoring elevator algorithm

Arrival Time	Δ Floor	Turnaround Time (ToT)	Travel Time inside car (CarT)	Ideal Travel Time (ITT)	Efficiency (ITT/ToT)
30	2	203.3	74	24	12
35	1	160.3	35	17	11
40	-1	79.3	38	17	21
45	-2	112.3	75	24	21
50	1	184.3	114	17	9
55	-1	103.3	37	17	16

Table 6: Statistical Analysis for Fixed sectoring elevator algorithm

4.4.4 Dynamic sectoring elevator algorithm:

4.4.4.1 Night-time:

Arrival Time	Arrival Floor	Destination Floor	Pickup Time	Leaving Time
30	0	2	159.3	233.3
35	0	1	160.3	195.3
40	2	1	81.3	119.3
45	2	0	82.3	157.3
50	1	2	120.3	234.3
55	1	0	121.3	158.3

Table 7: Simulation Results for Dynamic sectoring elevator algorithm (Night-time)

Arrival Time	Δ Floor	Turnaround Time (ToT)	Travel Time inside car (CarT)	Ideal Travel Time (ITT)	Efficiency (ITT/ToT)
30	2	203.3	74	24	12
35	1	160.3	35	17	11
40	-1	79.3	38	17	21
45	2	112.3	75	24	21
50	1	184.3	114	17	9
55	-1	103.3	37	17	16

Table 8: Statistical Analysis for Dynamic sectoring elevator algorithm (Night-time)

4.4.4.2 Morning-time:

Arrival Time	Arrival Floor	Destination Floor	Pickup Time	Leaving Time
10	-1	-2	84.3	119.3
15	-1	-2	85.3	120.3
20	-2	-1	43.2	81.3
20.1	-1	-2	86.3	121.3
25	-2	-1	44.2	82.3
30	-2	-1	45.3	83.3

Table 9: Simulation Results for Dynamic sectoring elevator algorithm (*Morning *-time)

Arrival Time	Δ Floor	Turnaround Time (ToT)	Travel Time inside car (CarT)	Ideal Travel Time (ITT)	Efficiency (ITT/ToT)
10	-1	109.3	35	17	16
15	-1	105.3	35	17	16
20	1	61.3	38.1	17	28
20.1	-1	101.2	35	17	17
25	1	57.3	38.1	17	30
30	1	53.3	38	17	32

Table 10: Statistical Analysis for Dynamic sectoring elevator algorithm (*Morning *-time)

4.4.5 First Come First Serve elevator algorithm with VIP passcode:

Arrival Time	Arrival Floor	Destination Floor	Pickup Time	Leaving Time
10	-1	2	43.2	233.3
15	1	-2	82.3	157.3
18	1	-1	83.3	195.3
20	-2	0	0	0
20.1	-1	1	44.2	81.3
22	2	-1	0	0
24	-1	-2	45.3	158.3
26	0	2	0	0

Table 11: Simulation Results for First Come First Serve elevator algorithm with VIP passcode

Arrival Time	Δ Floor	Turnaround Time (ToT)	Travel Time inside car (CarT)	Ideal Travel Time (ITT)	Efficiency (ITT/ToT)
10	3	233.3	190.1	31	13%
15	-3	142.3	75	31	22%
18	-2	177.3	112	24	14%
20	0	0	0	0	0%
20.1	2	61.2	37.1	24	39%
22	0	0	0	0	0%
24	-1	134.3	113	17	13%
26	0	0	0	0	0%

Table 12: Statistical Analysis for First Come First Serve elevator algorithm with VIP passcode

Passenger number 4, 6 and 8 could not board the elevator because they did not have passcode to use the elevator, since they were not authorized to do so.
Hence, they get a value of zero in the respective parameters like: Turnaround Time, Travel Time inside car, Ideal Travel Time and Efficiency.

4.5 Originality of this work:

I have adapted the idea of statistical analysis for the classic and modern elevator algorithms from module 2, lecture # 5 of one of my courses named as "DAT530 Discrete Simulation and Performance Analysis".

4.6 Petri Net properties analysis:

4.6.1 Safeness:

All the places in the model are safe in the model except the following;

1. *pArrivals*
2. *pWaiting*
3. *pLeft*

Because they contain more than one token in them that's why they are not safe. *pArrival* contain all the passengers who wants to travel in the elevator, *pWaiting* has all the passengers who wants to step on the elevator after their arrival and *pLeft* stores all the passengers who leave the elevator after getting to their destination floor.

That's why I can say that my petri net model is not safe as a whole.

4.6.2 Liveness:

4.6.2.1 In case of FCFS movement of car:

Level 0: There is no transition in my model which is at level 0 or there is no dead transition in my model.

Level 1: tArrival, tStepON, tStepOFF, tLeaving, tInit, tOpenDoor, tBoardingComplete, tDoorDelay, tCloseDoor, tAccel, tDeAccel

Level 2: tMoving. For example, if the car has to move two floors upwards or downwards then tMoving will fire twice in a loop, based on a condition of number of floor leaps that it has to take.

Level 3: There is no transition that is on level 3 that is, there is no transition that once entered a loop and can fire unconditionally infinite times.

Level 4: There is no transition that is on level 4 that is, there is no transition that has a path that leads to firing of that transition from anywhere in the reachability tree.

4.6.2.2 In case of Shabbat movement of car:

Level 0: There is no transition in my model which is at level 0 or there is no dead transition in my model.

Level 1: tArrival, tStepON, tStepOFF, tLeaving, tInit, tOpenDoor, tBoardingComplete, tDoorDelay, tCloseDoor, tAccel, tDeAccel and tMoving. Because in Shabbat movement car only moves by one floor, that's why tMoving is also at level 1, since it also fires once.

Level 2: there is no transition at level 2 that is, there is no transition that once entered in a loop has possibility to fire infinite times but not unconditionally.

Level 3: There is no transition that is on level 3 that is, there is no transition that once entered a loop and can fire unconditionally infinite times.

Level 4: There is no transition that is on level 4 that is, there is no transition that has a path that leads to firing of that transition from anywhere in the reachability tree.

4.6.3 Deadlock:

All the transitions in my model is live and there is no deadlock between them, which means all of them can be enabled.

5. Discussion:

5.1 Originality of this work:

I have adapted the idea of statistical analysis for the classic and modern elevator algorithms from module 2, lecture # 5 of one of my courses named as "DAT530 Discrete Simulation and Performance Analysis".

5.2 Relevant Works:

I am using Shabbat algorithm code originally developed by Professor Reggie Davidrajuh in my thesis as classical algorithm to compare it with my proposed modern algorithms.

Regarding my modern algorithms, I have found some algorithms which relates to my modern algorithm to some extent but not completely.

5.2.1 First Relevance[1]:

For example, in my fixed sectoring algorithm the elevator is fixed to some specific floors to serve only with a single elevator. But the relevant work which I found on internet says that it will divide the building into the same number of sectors as it has number of elevators and each elevator is fixed to a specific sector to serve. So, the difference between mine and the relevant work is that I am working with a single elevator and the relevant algorithm contains multiple elevators at its disposal to work with.

[1] http://vedantmisra.com/elevator-algorithms/

5.2.2 Second Relevance[1]:

Second relevance which I found is about my dynamic sectoring algorithm. In my dynamic sectoring algorithm, elevator is being fixed to bottom floors in the official hours and the rest of the day it is fixed for the residential floors. So, the sector is changed based on day of the time.

While in the relevant work to this, sector's location and size can vary, in case of elevators which are busy serving other passengers or in the case of idle elevators, to make use of them efficiently. So, the difference between mine and the relevant work is that in my dynamic sectoring algorithm, there is a single elevator working and once it is fixed to a sector it cannot change the location or size of its sector based on the elevator's unavailability or the elevator which is not being used in a sector.

5.2.3 Third relevance[2]:

Third relevance is with regards to my FCFS with VIP passcode elevator algorithm. I have found this relevance on internet which says that only those passengers have access to travel in the elevators who have PIN/passcode at their disposal.

5.3 Limitations of my work:

Due to unavailability of real or official data about the elevators, I have used cooked up or supposed data in terms of arrival floor, arrival time and destination floor of the passengers. That's why statistical analysis could be compromised. But the algorithms will perform well, if provided real data.

5.4 Further work:

Further work can be done on following aspects:

1. All the modern algorithms can be implemented in a multiple elevator system.
2. Specifically, dynamic sectoring algorithm can perform well in multiple elevator system which can change its sector's size and location as required.
3. In multiple elevator system, nearest car technique can be implemented in which passengers are picked up by the elevator which is nearest to the called floor.
4. For having minimized waiting times for the passengers, we can make use of fuzzy algorithms[3].
5. We can use evacuation algorithm [4] for the safety of passengers in case of an emergency.

---

[2] https://www.schindler.com/content/dam/web/uk/Brochures/Schidnler-7000-traffic-managemet-system.pdf
[3] https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6108794

[4] https://link.springer.com/chapter/10.1007/978-1-4419-9725-8_57

6. By using Power Factor Correction[5](PFC) and super-capacitor energy recover, we can make an elevator energy efficient and less power consuming.
7. Elevator's performance can be improved by using Reinforcement Learning[6].
8. In a scenario of multiple elevators system, we can incorporate genetic algorithms[7] for the allocation of elevator in real time with decreased passenger waiting time.
9. Knowledge based elevator controller [8] can be useful to choose an optimum route of floors to serve maximum number of passenger calls and minimize their waiting time

5.5 Learning Experience:

I have used GPenSIM and MATLAB earlier also for the term project of one of my courses named as "DAT530 Discrete Simulation and Performance Analysis". But that was a small project as compared to my thesis project. It is big in terms of scale and complexity. I learned a lot of new things and I think I have overcome my shortcomings which were present during the term project regarding the knowledge of Petri Net theory. It was both challenging and exciting for me to work on my thesis which kept me interested in it.

References:

1. http://vedantmisra.com/elevator-algorithms/
2. https://www.schindler.com/content/dam/web/uk/Brochures/Schidnler-7000-traffic-managemet-system.pdf
3. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6108794
4. https://link.springer.com/chapter/10.1007/978-1-4419-9725-8_57
5. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4272388
6. http://papers.nips.cc/paper/1073-improving-elevator-performance-using-reinforcement-learning.pdf
7. https://link.springer.com/chapter/10.1007/978-3-7091-6492-1_56
8. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=327171
9. http://www.davidrajuh.net/gpensim/

[5] https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4272388

[6] http://papers.nips.cc/paper/1073-improving-elevator-performance-using-reinforcement-learning.pdf

[7] https://link.springer.com/chapter/10.1007/978-3-7091-6492-1_56

[8] https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=327171

10. http://www.davidrajuh.net/gpensim/v9/GPenSIM_v9_User_Manual.pdf

Appendix:

A1: Installation Guide:

1. First install MATLAB & GpenSIM
2. Unzip the file named as “Code Files.7z”

A2: User Manual:

A2.1: For Shabbat, FCFS and fixed sectoring:

1. It will have a folder named as “Code Files”
2. Open this folder to run simulations
3. Open the main simulation file named as “msf.m”
4. You can make changes to arrival times and destination floors of the passengers in the loadData() function as shown in the following Code Snippet 105. You can also add/remove the passengers by adding/removing the arrival time and destination floor of a passenger on different levels in loadData() function.
5. You can change the minimum and maximum floors of the elevator in the “msf.m” file as shown in the Code Snippet 106 below. Also, passenger holding capacity of the elevator can be modified.
6. Run it after making desired changes

```
global_info.ATL1 = [20  0]; % level -2
global_info.ATL2 = [10 2]; % level -1
global_info.ATL3 = [28 2];  % level 0
global_info.ATL4 = [15 -2];  % level 1
global_info.ATL5 = [22 -1];  % level 2
```

Code Snippet 105: For changing arrival time and destination floor

```
global_info.MAX_CAPACITY = 5;    % Maximum capacity to hold passengers in car
global_info.MIN_LEVEL = -2;      % Bottom level floor
global_info.MAX_LEVEL = 2;       % Highest/top level floor
```

Code Snippet 106: For changing passenger holding capacity and (min,max) floors

A2.2: For FCFS with VIP passcode:

1. It will have a folder named as “Code Files”
2. Open this folder to run simulations
3. Open the main simulation file named as “msf.m”
4. You can make changes to arrival times , destination floors and passcode holding status of the passengers in the loadData() function as shown in the following Code Snippet 107. You can also add/remove the passengers by adding/removing the arrival time, destination floor and passcode holding status of a passenger on different levels in loadData() function.
5. You can change the minimum and maximum floors of the elevator in the “msf.m” file as shown in the Code Snippet 108 below. Also, passenger holding capacity of the elevator can be modified.
6. Run it after making desired changes

```
global_info.ATL1 = [20  0 0]; % level -2
global_info.ATL2 = [10 2 1,20 1 1,24 -2 1]; % level -1
global_info.ATL3 = [26 2 0];  % level 0
global_info.ATL4 = [15 -2 1, 18 -1 1];  % level 1
global_info.ATL5 = [22 -1 0];  % level 2
```

Code Snippet 107: For changing arrival time, destination floor and passcode holding status

```
global_info.MAX_CAPACITY = 5;    % Maximum capacity to hold passengers in car
global_info.MIN_LEVEL = -2;      % Bottom level floor
global_info.MAX_LEVEL = 2;       % Highest/top level floor
```

Code Snippet 108: For changing passenger holding capacity and (min,max) floors

A2.3: Dynamic sectoring:

1. It will have a folder named as “Code Files”
2. Open this folder to run simulations
3. Open the main simulation file named as “msf.m”
4. You can make changes to arrival times and destination floors of the passengers in the loadData() function as shown in the following Code Snippet 109. You can also add/remove the passengers by adding/removing the arrival time and destination floor of a passenger on different levels in loadData() function.
5. You can change the minimum and maximum floors for the night time section of floors in the “msf.m” file as shown in the Code Snippet 110 below. Also, passenger holding capacity of the elevator can be modified in the msf.m file as shown in Code Snippet 111.

6. Run it after making desired changes

```
global_info.ATL1 = [20 -1, 25 -1, 30 -1]; % level -2
global_info.ATL2 = [10 -2, 15 -2, 20 -2]; % level -1
global_info.ATL3 = [30 2, 35 1];  % level 0
global_info.ATL4 = [50 2, 55 0];  % level 1
global_info.ATL5 = [40 1, 45 0];  % level 2
```

Code Snippet 109: For changing arrival time, and destination floor

```
disp("Its Night time");disp('');
disp("Elevator is fixed to top floors: 0,1 & 2");disp('');
global_info.MIN_LEVEL = 0;   % Bottom floor
global_info.MAX_LEVEL = 2;   % top floor
```

Code Snippet 110: For changing min & max floors

```
global_info.MAX_CAPACITY = 5;   % Maximum capacity to hold passengers in car
```

Code Snippet 111: For changing passenger holding capacity

A3: Complete Code:

A3.1: Shabbat elevator algorithm:

A3.1.1 Main Simulation File (MSF)

```
% MSF for the Lift Simulator
% Reggie.davidrajuh@uis.no  (c) 20 March 2018

clear all; clc;

global global_info
global_info.STOP_AT = 1000;

global_info.MAX_CAPACITY = 5;   % Max passengers in car
global_info.MIN_LEVEL = -2;     % Basement level
global_info.MAX_LEVEL = 2;      % top floor

global_info.Current_Level = 0;          % level = MIN_LEVEL .. MAX_LEVEL
global_info.Current_Direction = 'UP';   % direction: 'UP' or 'DOWN'

loadData();

% we have five modules and one connector
pns = pnstruct({'Init_pdf', 'Arrival_pdf', 'Car_ata_Level_pdf',...
                'Car_Moving_pdf', 'Departure_pdf', 'IMC_pdf'});

dyn.m0 = {'pInit',1, 'pArrival',global_info.TOTAL_USERS};
dyn.ft = {'tMoving',7, 'tOpenDoor',5,'tCloseDoor',5, ...
          'tAccel',3, 'tDeAccel',3,...
          'tDoorDelay',15, 'tStepON',1, 'tStepOFF',1,'allothers',0.1}; %
```

```
dyn.ip = {'tStepOFF',2,'tStepON',1,  'tMoving',1};
pni = initialdynamics(pns, dyn);
sim = gpensim(pni);

disp(' ');prnstate('Final markings: ');
plotp(sim,{'pArrival','pWaiting','pAtLevel','pLeft'});
% disp(' ');prnss(sim);
% disp(' ');prnfinalcolors(sim);
```

A3.1.2 PDF Files:

A3.1.2.1 Init_pdf()

```
% SMC-2018 Miyazaki: The Lift
% Reggie.davidrajuh@uis.no  (c) 20 March 2018
% PDF for the module Initialization (Init_pdf)

function [png] = Init_pdf()
png.PN_name = 'Init';
png.set_of_Ps = {'pInit'};
png.set_of_Ts = {'tInit'};
png.set_of_As = {'pInit','tInit',1};
```

A3.1.2.2 Arrival_pdf()

```
% SMC-2018 Miyazaki: The Lift
% Reggie.davidrajuh@uis.no  (c) 20 March 2018
% PDF for Arriving passengers(Arrival_pdf)

function [png] = Arrival_pdf()
png.PN_name = 'Arrival';
png.set_of_Ps = {'pArrival', 'pWaiting', 'pWaitingDUMMY'};
png.set_of_Ts = {'tArrival'};
png.set_of_As = {'pArrival','tArrival',1, 'tArrival','pWaiting',1, ...
                 'tArrival','pWaitingDUMMY',1 ... % for testing only
                };
```

A3.1.2.3 Car_ata_Level_pdf()

```
% SMC-2018 Miyazaki: The Lift
% Reggie.davidrajuh@uis.no  (c) 20 March 2018
% PDF for the module "Car at a Level" (CAR_ata_Level_pdf)

function [png] = Car_ata_Level_pdf()
png.PN_name = 'Car_ata_Level';
png.set_of_Ps = {'pAtLevel', 'pCloseDoor', 'pDoorOpened',...
                 'pDoorDelay', 'pBoardingComplete'};
png.set_of_Ts = {'tOpenDoor', 'tStepOFF', 'tStepON', ...
                 'tCloseDoor','tDoorDelay', 'tBoardingComplete'};
png.set_of_As = {'tOpenDoor','pAtLevel',1, ... % tOpenDoor
      'pAtLevel','tStepOFF',1, 'tStepOFF','pAtLevel',1, ... % tStepOFF
```

```
        'pAtLevel','tStepON',1, 'tStepON','pAtLevel',1, ...   % tStepON
        'pAtLevel','tBoardingComplete',1, ... % tBoardingComplete
        'tBoardingComplete', 'pBoardingComplete',1, ... % tBoardingComplete
        'pBoardingComplete','tCloseDoor',1,... % tCloseDoor
        'pDoorDelay','tCloseDoor',1,...      % tCloseDoor
        'tOpenDoor','pDoorOpened',1,...
        'pDoorOpened','tDoorDelay',1,... % tDoorDelay
        'tDoorDelay','pDoorDelay',1 ... % tDoorDelay
```

A3.1.2.4 Car_Moving_pdf()

```
% SMC-2018 Miyazaki: The Lift
% Reggie.davidrajuh@uis.no  (c) 20 March 2018
% PDF for the Moving Car(Car_Moving_pdf)

function [png] = Car_Moving_pdf()
png.PN_name = 'Car_Moving';
png.set_of_Ps = {'pMoving', 'pArrivedAtLevelDUMMY'};
png.set_of_Ts = {'tAccel','tMoving','tDeAccel'};
png.set_of_As = {'tAccel','pMoving',1, ...
        'pMoving','tMoving',1, 'tMoving','pMoving',1, ...
        'tDeAccel', 'pArrivedAtLevelDUMMY', 1, ... % testing ONLY
        'pMoving','tDeAccel',1};
```

A3.1.2.5 Departure_pdf()

```
% SMC-2018 Miyazaki: The Lift
% Reggie.davidrajuh@uis.no  (c) 20 March 2018
% PDF for Leaving passengers(Departure_pdf)

function [png] = Departure_pdf()
png.PN_name = 'Departure';
png.set_of_Ps = {'pLeft'};
png.set_of_Ts = {'tLeaving'};
png.set_of_As = {'tLeaving','pLeft',1};
```

A3.1.2.6 IMC_pdf()

```
% SMC-2018 Miyazaki: The Lift
% Reggie.davidrajuh@uis.no  (c) 20 March 2018
% PDF for the inter-modular connections (IMC_pdf)

function [png] = IMC_pdf()
png.PN_name = 'Inter-Modular';
png.set_of_Ps = {'pArrivedAtLevel', 'pAboutToMove', 'pLeaving'};
png.set_of_Ts = {};
png.set_of_As = {... % Init
        'tInit','pArrivedAtLevel',1, ...
        ... % Car_ata_Level
        'pArrivedAtLevel','tOpenDoor',1, ...
        'tCloseDoor','pAboutToMove',1, ...
        'pWaiting','tStepON',1, ...
        'tStepOFF','pLeaving',1, ...
        ... % Car_Moving
        'pAboutToMove','tAccel',1,...
        'tDeAccel','pArrivedAtLevel',1, ...
```

```
        ... % Departure
        'pLeaving','tLeaving',1 ...
            };
```

A3.1.3 tArrival_pre(trans)

```
function [fire, trans] = tArrival_pre(trans)
global global_info

for Level = global_info.MIN_LEVEL:global_info.MAX_LEVEL
    Waiting = eval(['global_info.ATL',int2str(Level+3),';']);
    if not(isempty(Waiting))
        timeOfArrival = Waiting(1);
        if ge(current_time, timeOfArrival)
            % this passenger has come
            global_info.Person_No = global_info.Person_No + 1;
            PNo = global_info.Person_No;
            whereTo = Waiting(2);

            % reduce global_info.ATLx
            eval(['global_info.ATL',int2str(Level+3),'=Waiting(3:end);']);
            % passenger generated time
            global_info.PASSENGER_DATA(PNo, 1) = current_time;
            global_info.PASSENGER_DATA(PNo, 4) = Level; % from which level
            global_info.PASSENGER_DATA(PNo, 5) = whereTo; % to which level

            color1 = ['L', int2str(Level)];
            color2 = ['P', int2str(PNo)];
            trans.new_color = {color1, color2};
            fire = 1;
            disp(['Arrival: Time:', num2str(current_time),...
                '  at level:', color1, '   PNo:', color2]);
            return
        end
    end
end
fire = 0;
```

A3.1.4 loadData()

```
function [] = loadData()
global global_info

global_info.People_in_Car = 0; % to start with, 0 passengers inside car

% Historial data: Arrival of people to the lift, at different levels
% Test-1:
% global_info.ATL1 = [20  0]; % level -2
% global_info.ATL2 = [10 2]; % level -1
% global_info.ATL3 = [30 2];  % level 0
% global_info.ATL4 = [50 -2];  % level 1
% global_info.ATL5 = [40 -1];  % level 2

global_info.ATL1 = [20  0]; % level -2
global_info.ATL2 = [10 2]; % level -1
global_info.ATL3 = [28 2];  % level 0
```

```
global_info.ATL4 = [15 -2];  % level 1
global_info.ATL5 = [22 -1];  % level 2



% % Test-2:
% global_info.ATL1 = [15 2, 27 -2, 3 1, 4 0, 9 2];                 % level -2
% global_info.ATL2 = [10 1, 3 0,            7 2, 11 1];    % level -1
% global_info.ATL3 = [20 1,                     26 1, 14 1, 18 1]; % level0
% global_info.ATL4 = [2 -2, 13 -2,   5 -1, 6 0, 8 2];% level 1
% global_info.ATL5 = [10  0,          23 0];                        % level 2

% global_info.ATL1 = [1 2, 2 -2, 3 1, 4 0, 9 2];                  % level -2
% global_info.ATL2 = [10 1, 3 0,            7 2, 10 1];    % level -1
% global_info.ATL3 = [20 1,                     10 1, 10 1, 10 1]; % level 0
% global_info.ATL4 = [0 -2, 5 -2,   3 -1, 6 0, 8 2];% level 1
% global_info.ATL5 = [0  0,          8 0];                         % level 2

% % Test-3:
% global_info.ATL1 = [16 2, 27 9, 30 10, 110 4, 150 7, 170 3, 190 8, 210 7,
252 4, 310 6, 313 10, 313 10, 332 3, 332 3, 403 3, 455 5, 520 5];
% global_info.ATL2 = [30 1, 40 3, 220 4, 239 10, 270 8, 360 4, 480 10, 580
3];
% global_info.ATL3 = [10 1, 30 1, 75 4,  550 1, 580 10, 587 8];
% global_info.ATL4 = [20 2, 100 1, 370 1,728 1, 880 10];
% global_info.ATL5 = [];
% global_info.ATL6 = [0 1,  55 1, 230 10, 550 1, 780 7];
% global_info.ATL7 = [75 3, 90 1, 100 4, 150 3, 200 3, 700 1, 840 10];
% global_info.ATL8 = [25 1, 50 3, 90 2, 120 6, 120 3, 700 1];
% global_info.ATL9 = [35 1, 70 3, 100 1, 203 3, 506 7, 710 10, 823 6];
% global_info.ATL10 =[8 1, 34 1, 204 3, 208 3, 390 5, 390 5, 399 1, 510 5,
512 3, 630 3, 656 8, 700 4, 710 2, 845 6];

passengers = 0;
for Level = global_info.MIN_LEVEL:global_info.MAX_LEVEL
    ATLx = eval(['global_info.ATL',int2str(Level+3),';']);    %%%%%  !!!!!! 3
!!!!!!!!!
    passengers = passengers + length(ATLx)/2;
end
disp(['Total number of passengers: ', int2str(passengers)]);
global_info.TOTAL_USERS = passengers;
global_info.PASSENGER_DATA = zeros(passengers,7);
global_info.Travellers_at_Present = zeros(1, global_info.MAX_CAPACITY);
global_info.Person_No = 0; % Person counter (or person ID number)

for i = 1:global_info.TOTAL_USERS
    eval(['global_info.P2DL_', int2str(i), ' = [];']);
end
```

A3.1.5 fillMissedCarTable()

```
function [] = fillMissedCarTable()
global global_info

[indices_for_this_level, ~] = ...
    find(global_info.PASSENGER_DATA(:,4)== global_info.Current_Level);

for i = 1:numel(indices_for_this_level)
    row_index = indices_for_this_level(i);

    if not(global_info.PASSENGER_DATA(row_index, 2))
        % arrived and waiting !!!
        global_info.PASSENGER_DATA(row_index,7) = ...
            global_info.PASSENGER_DATA(row_index,7) + 1;
    end
end
```

A3.1.6 separateColors(the2colors)

```
function [PNo] = separateColors(the2colors)

color1 = the2colors{1};
color2 = the2colors{2};

% finding from which Level
if strcmp(color1(1), 'L')
    fromWhichLevel = color1;
    PNo = color2;
else
    fromWhichLevel = color2;
    PNo = color1;
end

PNo = PNo(2:end);
PNo = str2double(PNo);
```

A3.1.7 COMMON_PRE(trans)

```
function [fire, trans] = COMMON_PRE(trans)
global global_info

switch trans.name
    case 'tInit', fire = 1;
    case 'tArrival', fire = 1;
    case 'tOpenDoor', fire = 1;
    case 'tDoorDelay', trans.override = 1; fire = 1;

    case 'tLeaving', fire = 1;
```

```
case 'tCloseDoor'
        disp(['**** Closing the door at Level ',...
            int2str(global_info.Current_Level), ...
            '     at time: ', num2str(current_time), ' *******']);
        disp(' ');
        fire = 1;

    case 'tStepOFF'
        % people currently inside car (travelling)
        current_passengers = global_info.PASSENGER_DATA(:, 6);
        peopleInCar = sum(current_passengers);
        if not(peopleInCar), fire = 0; return, end % car is empty
        [~, PNoInCar] = find(transpose(current_passengers));
        for i = 1:numel(PNoInCar)
            PNo = PNoInCar(i);
            whereTo = global_info.PASSENGER_DATA(PNo, 5);
            if eq(global_info.Current_Level, whereTo)

                % this guy is leaving
                global_info.PASSENGER_DATA(PNo, 6) = 0;
                % leaving time from CAR
                global_info.PASSENGER_DATA(PNo, 3) = current_time;
                disp(['Leaving Car: Time:', num2str(current_time),...
                    '  at level:', int2str(whereTo),...
                    '   PNo:', int2str(PNo)]);
                fire = 1; return
            end
        end
        fire = 0;

    case 'tStepON'
        currentLevel = global_info.Current_Level;
        tokID = tokenAnyColor('pWaiting', 1, ...
                    {['L',int2str(currentLevel)]});
        if not(tokID) % no one is waiting at this level!!!
            fire = 0; return
        end

        % some people are waiting at this level
        % then check for vacancy
        peopleInCar = sum(global_info.PASSENGER_DATA(:, 6));
        vacant = global_info.MAX_CAPACITY  - peopleInCar;
        if not(vacant)
            disp('** Car is full, cannot take more passengers!');

            % make "tStepON" weaker so that it will not try again & again
            priorset('tStepON', 0);
            fillMissedCarTable();
            fire = 0; return
        end
        trans.selected_tokens = tokID;
        trans.override = 1;
        fire = 1;
        the2colors = get_color('pWaiting', tokID);
        PNo = separateColors(the2colors);
```

```
        % the time passenger ENTERS car
        global_info.PASSENGER_DATA(PNo, 2) = current_time;
        % the passenger is inside the car: travelling
        global_info.PASSENGER_DATA(PNo, 6) = true;
        disp(['Entering Car: Time:', num2str(current_time),...
              '  at level:', int2str(currentLevel),...
              '   PNo:', int2str(PNo)]);

    case 'tBoardingComplete'
        % reset priority of "tStepON," if it was set to 0
        priorset('tStepON', 1);
        fire = 1;

    %%%%
```

A3.1.8 COMMON_POST(trans)

```
function [] = COMMON_POST(trans)
global global_info


switch trans.name
    case 'tLeaving'
        n = ntokens('pLeft');
        if eq(n, global_info.TOTAL_USERS)
            global_info.STOP_SIMULATION = true;
        end

    case 'tOpenDoor'
        disp(' ');
        disp(['**** Arrived at Level ',...
              int2str(global_info.Current_Level), ...
              '     at time: ', num2str(current_time), ' *******']);
end % switch
```

A3.2 First come first serve elevator algorithm:

A3.2.1 Main Simulation File (MSF):

```
% MSF for the Lift Simulator
% Author:Reggie Davidrajuh
% Date: 20 March 2018
% Adapted By: Bilal Amjad
% Date: 1 Feb 2019

clear all; clc;

global global_info
global_info.STOP_AT = 1000;

global_info.MAX_CAPACITY = 5;   % Maximum capacity to hold passengers in car
global_info.MIN_LEVEL = -2;     % Bottom level floor
global_info.MAX_LEVEL = 2;      % Highest/top level floor

global_info.Current_Level = 0;         % level = MIN_LEVEL .. MAX_LEVEL
% global_info.Current_Direction = 'UP';  % direction: 'UP' or 'DOWN'

loadData();

% we have five modules and one inter modular connector
pns = pnstruct({'Init_pdf', 'Arrival_pdf', 'Car_ata_Level_pdf',...
                'Car_Moving_pdf', 'Departure_pdf', 'IMC_pdf'});

dyn.m0 = {'pInit',1, 'pArrival',global_info.TOTAL_USERS}; %initial markings
%firing times of transitions
dyn.ft = {'tMoving',7, 'tOpenDoor',5,'tCloseDoor',5, ...
```

```
          'tAccel',3, 'tDeAccel',3,...
          'tDoorDelay',15, 'tStepON',1, 'tStepOFF',1,'allothers',0.1}; %
dyn.ip = {'tStepOFF',2,'tStepON',1,  'tMoving',1};      %initial priorities
pni = initialdynamics(pns, dyn);
sim = gpensim(pni);
%  prnss(sim);
disp(' ');prnstate('Final markings: ');
% plotp(sim,{'pArrival', 'pWaiting','pAtLevel','pLeft'});
```

A3.2.2 PDF Files:

A3.2.2.1 Init_pdf()

```
% Author:Reggie Davidrajuh
% Adapted By: Bilal Amjad
% PDF for the module Initialization (Init_pdf)

function [png] = Init_pdf()
png.PN_name = 'Init';
png.set_of_Ps = {'pInit'};
png.set_of_Ts = {'tInit'};
png.set_of_As = {'pInit','tInit',1};
```

A3.2.2.2 Arrival_pdf()

```
% Author:Reggie Davidrajuh
% Adapted By: Bilal Amjad
% PDF for Arriving passengers(Arrival_pdf)

function [png] = Arrival_pdf()
png.PN_name = 'Arrival';
png.set_of_Ps = {'pArrival', 'pWaiting', 'pWaitingDUMMY'};
png.set_of_Ts = {'tArrival'};
png.set_of_As = {'pArrival','tArrival',1, 'tArrival','pWaiting',1, ...
                 'tArrival','pWaitingDUMMY',1 ... % for testing only
                };
```

A3.2.2.3 Car_ata_Level_pdf()

```
% Author:Reggie Davidrajuh
% Adapted By: Bilal Amjad
% PDF for the module "Car at a Level" (CAR_ata_Level_pdf)

function [png] = Car_ata_Level_pdf()
png.PN_name = 'Car_ata_Level';
png.set_of_Ps = {'pAtLevel', 'pCloseDoor', 'pDoorOpened',...
                 'pDoorDelay', 'pBoardingComplete'};
png.set_of_Ts = {'tOpenDoor', 'tStepOFF', 'tStepON', ...
                 'tCloseDoor','tDoorDelay', 'tBoardingComplete'};
png.set_of_As = {'tOpenDoor','pAtLevel',1, ... % tOpenDoor
      'pAtLevel','tStepOFF',1, 'tStepOFF','pAtLevel',1, ... % tStepOFF
      'pAtLevel','tStepON',1, 'tStepON','pAtLevel',1, ...   % tStepON
      'pAtLevel','tBoardingComplete',1, ... % tBoardingComplete
      'tBoardingComplete', 'pBoardingComplete',1, ... % tBoardingComplete
```

```
        'pBoardingComplete','tCloseDoor',1,... % tCloseDoor
        'pDoorDelay','tCloseDoor',1,...      % tCloseDoor
        'tOpenDoor','pDoorOpened',1,...
        'pDoorOpened','tDoorDelay',1,... % tDoorDelay
        'tDoorDelay','pDoorDelay',1 ... % tDoorDelay
                    };
```

A3.2.2.4 Car_Moving_pdf()

```
% Author:Reggie Davidrajuh
% Adapted By: Bilal Amjad
% PDF for the Moving Car(Car_Moving_pdf)

function [png] = Car_Moving_pdf()
png.PN_name = 'Car_Moving';
png.set_of_Ps = {'pMoving', 'pArrivedAtLevelDUMMY'};
png.set_of_Ts = {'tAccel','tMoving','tDeAccel'};
png.set_of_As = {'tAccel','pMoving',1, ...
        'pMoving','tMoving',1, 'tMoving','pMoving',1, ...
        'tDeAccel', 'pArrivedAtLevelDUMMY', 1, ... % testing ONLY
        'pMoving','tDeAccel',1};
```

A3.2.2.5 Departure_pdf()

```
% Author:Reggie Davidrajuh
% Adapted By: Bilal Amjad
% PDF for Leaving passengers(Departure_pdf)

function [png] = Departure_pdf()
png.PN_name = 'Departure';
png.set_of_Ps = {'pLeft'};
png.set_of_Ts = {'tLeaving'};
png.set_of_As = {'tLeaving','pLeft',1};
```

A3.2.2.6 IMC_pdf()

```
% Author:Reggie Davidrajuh
% Adapted By: Bilal Amjad
% PDF for the inter-modular connections (IMC_pdf)

function [png] = IMC_pdf()
png.PN_name = 'Inter-Modular';
png.set_of_Ps = {'pArrivedAtLevel', 'pAboutToMove', 'pLeaving'};
png.set_of_Ts = {};
png.set_of_As = {... % Init
        'tInit','pArrivedAtLevel',1, ...
        ... % Car_ata_Level
        'pArrivedAtLevel','tOpenDoor',1, ...
        'tCloseDoor','pAboutToMove',1, ...
        'pWaiting','tStepON',1, ...
        'tStepOFF','pLeaving',1, ...
        ... % Car_Moving
        'pAboutToMove','tAccel',1,...
        'tDeAccel','pArrivedAtLevel',1, ...
```

```
        ... % Departure
        'pLeaving','tLeaving',1 ...
            };
```

A3.2.3 tArrival_pre(trans)

```
% Author:Reggie Davidrajuh
% Adapted By: Bilal Amjad
% Function name:tArrival_pre
% Purpose: It handles the arrival of each passenger at a floor by comparing
% current time with the arrival time and assigns two colors "L" & "P" which
% represents level and person number of a passenger
function [fire, trans] = tArrival_pre(trans)
global global_info
%will run from minimum to maximum level
for Level = global_info.MIN_LEVEL:global_info.MAX_LEVEL
    %get the value of 'global_info.ATL' e.g global_info.ATL1 frm loadData()
    %function
    Waiting = eval(['global_info.ATL',int2str(Level+3),';']);
    if not(isempty(Waiting))
        %if not empty then get the value
        %present at first index which is passenger's arrival time
        timeOfArrival = Waiting(1);
         % if current time is greater than passenger's arrival time than
         %this is the time for passenger to arrive
        if ge(current_time, timeOfArrival)

            %update the person counter
            global_info.Person_No = global_info.Person_No + 1;
            PNo = global_info.Person_No;
            %second index value shows the destination floor of passenger
            whereTo = Waiting(2);

            % reduce the 1st and 2nd index value from global_info.ATLx
            eval(['global_info.ATL',int2str(Level+3),'=Waiting(3:end);']);
            % time at which passenger arrived
            global_info.PASSENGER_DATA(PNo, 1) = current_time;
            % At which level passenger arrived
            global_info.PASSENGER_DATA(PNo, 4) = Level;
            % To which level passenger wants to go
            global_info.PASSENGER_DATA(PNo, 5) = whereTo;
            %two colors are assigned to each token/passenger
            color1 = ['L', int2str(Level)];
            color2 = ['P', int2str(PNo)];
            trans.new_color = {color1, color2};
            fire = 1;
            disp(['Passenger Arrival: Time:', num2str(current_time),...
                '  at level:', color1, '   Person No:', color2]);
            return
        end
    end
end
fire = 0;
```

A3.2.4 loadData()

```
% Author:Reggie Davidrajuh
% Adapted By: Bilal Amjad
% Function name:loadData
% Purpose: This function contain arrival time & destination floors of
% passengers. It calculates the number of passenger according to historical
% data and creates a matrix/ vector to hold information about passenger like
%person no, arrival and departure time, arrival and departure floor, car
%travelling status, missed car status.


function [] = loadData()
global global_info

global_info.People_in_Car = 0; % initially we have 0 passengers inside car

% Passenger's Historial data: Arrival time of passengers to the lift at
% different levels. Also contain destination floor of passengers
% Test-1:
% global_info.ATL1 = [20  0]; % level -2
% global_info.ATL2 = [10 2]; % level -1
% global_info.ATL3 = [28 2];  % level 0
% global_info.ATL4 = [15 -2];  % level 1
% global_info.ATL5 = [22 -1];  % level 2

global_info.ATL1 = [20  0]; % level -2
global_info.ATL2 = [10 2]; % level -1
global_info.ATL3 = [28 2];  % level 0
global_info.ATL4 = [15 -2];  % level 1
global_info.ATL5 = [22 -1];  % level 2


% Test-2:
% global_info.ATL1 = [15 2, 27 -2, 30 1, 40 0, 90 2];                     %
level -2
% global_info.ATL2 = [10 1, 30 0,            75 2, 110 1];    % level -1
% global_info.ATL3 = [20 1,                         100 1, 100 1, 100 1];  %
level 0
% global_info.ATL4 = [0 -2, 25 -2,   35 -1, 60 0, 80 2];% level 1
%   global_info.ATL5   =   [0       0,                                 38   0];
% level 2

% global_info.ATL1 = [15 2, 27 -2, 3 1, 4 0, 9 2];                   % level -
2
% global_info.ATL2 = [10 1, 3 0,            7 2, 11 1];    % level -1
% global_info.ATL3 = [20 1,                        26 1, 14 1, 18 1];  % level
0
% global_info.ATL4 = [2 -2, 13 -2,   5 -1, 6 0, 8 2];% level 1
%   global_info.ATL5   =   [10      0,                                 23   0];
% level 2
```

```
% % Test-3:
% global_info.ATL1 = [16 2, 27 9, 30 10, 110 4, 150 7, 170 3, 190 8, 210 7,
252 4, 310 6, 313 10, 313 10, 332 3, 332 3, 403 3, 455 5, 520 5];
% global_info.ATL2 = [30 1, 40 3, 220 4, 239 10, 270 8, 360 4, 480 10, 580
3];
% global_info.ATL3 = [10 1, 30 1, 75 4,  550 1, 580 10, 587 8];
% global_info.ATL4 = [20 2, 100 1, 370 1,728 1, 880 10];
% global_info.ATL5 = [];
% global_info.ATL6 = [0 1,  55 1, 230 10, 550 1, 780 7];
% global_info.ATL7 = [75 3, 90 1, 100 4, 150 3, 200 3, 700 1, 840 10];
% global_info.ATL8 = [25 1, 50 3, 90 2, 120 6, 120 3, 700 1];
% global_info.ATL9 = [35 1, 70 3, 100 1, 203 3, 506 7, 710 10, 823 6];
% global_info.ATL10 =[8 1, 34 1, 204 3, 208 3, 390 5, 390 5, 399 1, 510 5,
512 3, 630 3, 656 8, 700 4, 710 2, 845 6];

passengers = 0;
%for calculating number of passengers according to historical data
for Level = global_info.MIN_LEVEL:global_info.MAX_LEVEL
    ATLx = eval(['global_info.ATL',int2str(Level+3),';']);
    passengers = passengers + length(ATLx)/2;
end
disp(['Total number of passengers: ', int2str(passengers)]);
global_info.TOTAL_USERS = passengers;
%a matrix/ vector to hold information about passenger like person no,
%arrival and departure time, arrival and departure floor, car travelling
%status,missed car status
global_info.PASSENGER_DATA = zeros(passengers,7);
global_info.Travellers_at_Present = zeros(1, global_info.MAX_CAPACITY);
global_info.Person_No = 0; % Person counter (or person ID number)

for i = 1:global_info.TOTAL_USERS
    eval(['global_info.P2DL_', int2str(i), ' = [];']);
end
```

A3.2.5 fillMissedCarTable()

```
% Author:Reggie Davidrajuh
% Adapted By: Bilal Amjad
% Function name: fillMissedCarTable
%  Purpose:Function  for  saving  information  about  passengers  who  could  not
%enter elevator because it was already full to its max capacity

function [] = fillMissedCarTable()
global global_info

[indices_for_this_level, ~] = ...
    find(global_info.PASSENGER_DATA(:,4)== global_info.Current_Level);

for i = 1:numel(indices_for_this_level)
    row_index = indices_for_this_level(i);

    if not(global_info.PASSENGER_DATA(row_index, 2))
        % shows that passenger has arrived and waiting to step on!!!
        global_info.PASSENGER_DATA(row_index,7) = ...
```

```
            global_info.PASSENGER_DATA(row_index,7) + 1;
    end
end
```

A3.2.6 separateColors(the2colors)

```
% Author:Reggie Davidrajuh
% Adapted By: Bilal Amjad
% Function name: separateColors(the2colors)
% Purpose: Function for separating two colors of the token to get the person
%number of a passenger
% Input: the2colors
% Output: PNo
function [PNo] = separateColors(the2colors)

color1 = the2colors{1};
color2 = the2colors{2};

% To find at which Level he arrived and also his passenger number
if strcmp(color1(1), 'L')
    fromWhichLevel = color1;
    PNo = color2;
else
    fromWhichLevel = color2;
    PNo = color1;
end

PNo = PNo(2:end);
PNo = str2double(PNo);
```

A3.2.7 COMMON_PRE(trans)

```
% Function name:COMMON_PRE(trans)
% Purpose: To fire pre-processor code of a transition
function [fire, trans] = COMMON_PRE(trans)
global global_info

switch trans.name
    %Author: Reggie Davidrajuh
    %Adapted By: Bilal Amjad
    %Purpose:To fire transition tInit
    case 'tInit', fire = 1;

    %Author: Reggie Davidrajuh
    %Adapted By: Bilal Amjad
    %Purpose:To fire transition tArrival
    case 'tArrival', fire = 1;
```

```
    %Author: Reggie Davidrajuh
    %Adapted By: Bilal Amjad
    %Purpose:To fire transition tOpenDoor
    case 'tOpenDoor', fire = 1;



     %Author: Reggie Davidrajuh
     %Adapted By: Bilal Amjad
     %Purpose:To fire transition tDoorDelay
     case 'tDoorDelay', trans.override = 1; fire = 1;

     %Author: Reggie Davidrajuh
     %Adapted By: Bilal Amjad
     %Purpose:To fire transition tLeaving
    case 'tLeaving', fire = 1;

     %Author: Reggie Davidrajuh
     %Adapted By: Bilal Amjad
     %Purpose: To display a message on console before closing the elevator
door
        case 'tCloseDoor'
        disp(['**** Closing the door now at Level ',...
            int2str(global_info.Current_Level), ...
            '     at time: ', num2str(current_time), ' *******']);
        disp(' ');
        fire = 1;

    %Author: Reggie Davidrajuh
    %Adapted By: Bilal Amjad
    %Purpose: This transition decides by comparing current level to the
    %destination level of a passenger to step off
    case 'tStepOFF'
        % it shows people who are currently inside car, means travelling
        current_passengers = global_info.PASSENGER_DATA(:, 6);
        peopleInCar = sum(current_passengers);
        if not(peopleInCar), fire = 0; return, end % not allowed to fire if
the car is empty
        [~, PNoInCar] = find(transpose(current_passengers));
        for i = 1:numel(PNoInCar)
            PNo = PNoInCar(i);
            whereTo = global_info.PASSENGER_DATA(PNo, 5);
            %if current level is equal to destination level of specified
person,then that person leaves
            if eq(global_info.Current_Level, whereTo)

                % set travelling in the car status of that person equal to
                % 0, as he is leaving
                global_info.PASSENGER_DATA(PNo, 6) = 0;
                % saving the leaving time from CAR
                global_info.PASSENGER_DATA(PNo, 3) = current_time;
                disp(['Passenger      is      Leaving      Car:      Time:',
num2str(current_time),...
                    '  at level:', int2str(whereTo),...
                    '   Person No:', int2str(PNo)]);
```

```
                fire = 1; return
            end
        end
        fire = 0;



%Author: Reggie Davidrajuh
%Adapted By: Bilal Amjad
%Purpose : This transition selects a token to step on, from pWaiting who has
%same color 'L' i.e arrival floor as the current floor


    case 'tStepON'
         currentLevel = global_info.Current_Level;
         %selecting the token whose arrival level is equal to current level
        tokID = tokenAnyColor('pWaiting', 1, ...
                    {['L',int2str(currentLevel)]});
        if not(tokID) % if there is no one waiting at this level, then do not
fire
            fire = 0; return
        end

        % Otherwise,check for vacancy in the car, if some people are waiting
at this level

        peopleInCar = sum(global_info.PASSENGER_DATA(:, 6));
        vacant = global_info.MAX_CAPACITY  - peopleInCar;
        if not(vacant)
            disp('** Caution! cannot take more passengers! Car is full');

            % reducing priority of "tStepON" to make it weak so that it will
not try again & again
            priorset('tStepON', 0);
            fillMissedCarTable(); %holds the persons who could not enter the
car because of no vacancy
            fire = 0; return
        end
        trans.selected_tokens = tokID;
        trans.override = 1;
        fire = 1;
        %using separateColors(the2colors) to get the person number to record
        %car entering time and to put travelling stauts=true, of that
        %particular person
        the2colors = get_color('pWaiting', tokID);
        PNo = separateColors(the2colors);

        % the time passenger ENTERS car
        global_info.PASSENGER_DATA(PNo, 2) = current_time;
        % the passenger is inside the car: travelling
        global_info.PASSENGER_DATA(PNo, 6) = true;
        disp(['Passenger is Entering Car: Time:', num2str(current_time),...
              '  at level:', int2str(currentLevel),...
              '   Person No:', int2str(PNo)]);
```

```
%Author: Reggie Davidrajuh
%Adapted By: Bilal Amjad
%Purpose: This transition completes the boarding procedure and also
%  increases  priority of "tStepON" back to 1 as it was set to
%0 previously in "tStepON
    case 'tBoardingComplete'
        % increasing  priority of "tStepON" back to 1 as it was set to
        %0 previously in "tStepON"
        priorset('tStepON', 1);
        fire = 1;

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%             For First Come First Serve Elevator Algorithm
    %%%%%%%%%%                 FCFS Movement
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Author: Reggie Davidrajuh
%Adapted By: Bilal Amjad
%Purpose: To deaccelerate the car after movement
    case 'tDeAccel'
        fire = 1;

 %Author: Bilal Amjad
 %Purpose: This transition describes the movement of the car that is
 %according to the list of floors in global_info.Iterinary(1)
       case 'tAccel' % It is the main strategy for moving car to destination
floors!!!

       %the car moves according to the set Iterinary in the Post-Processor
       %of "tArrival"
       %it selects first index value of global_info.Iterinary(1) as the next
floor to reach there
         global_info.Current_Level =  global_info.Iterinary(1)  ;


        global_info.Starting_to_move = true; % It is used by "tMoving"
        fire = 1;

%Author: Reggie Davidrajuh
%Adapted By: Bilal Amjad
%Purpose:It moves the car to the floor as per defined by the value of
global_info.Iterinary(1)
    case 'tMoving'
        if global_info.Starting_to_move
            global_info.Starting_to_move = false;
            fire = 1;
        else
            fire = 0;
        end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

end
```

A3.2.8 COMMON_POST(trans)

```
% Function name:COMMON_POST(trans)
% Purpose: To fire post processor code of a transition after
% pre-processor code of the same transition has been fired

function [] = COMMON_POST(trans)
global global_info


switch trans.name
%     Author:Reggie Davidrajuh
%     Adapted By: Bilal Amjad
%     Purpose: To stop the simulations after all passengers have left
    case 'tLeaving'
        n = ntokens('pLeft');
        %don't stop the simulation untill all the passengers have
        %reached their destination floors
        if eq(n, global_info.TOTAL_USERS)
            global_info.STOP_SIMULATION = true;
        end

%     Author:Reggie Davidrajuh
%     Adapted By: Bilal Amjad
%       Purpose: To display a message on console after opening the elevator
door
    case 'tOpenDoor'
        disp(' ');
        disp(['**** Elevator Arrived at Level ',...
              int2str(global_info.Current_Level), ...
              '     at time: ', num2str(current_time), ' *******']);

   %Author: Bilal Amjad
   %Purpose: To make an itenirary for the car to follow by combining the
   % arrival and destination floors of the passengers
    case 'tArrival'
        %Contains arrival floors of all the passengers
         global_info.PickLevel  =  global_info.PASSENGER_DATA(:,  4)  ;  %  from
which level
          %Contains departure floors of all the passengers
         global_info.DropLevel  =  global_info.PASSENGER_DATA(:,  5)    ;  %  to
which level
         %makes up a final iterinary of elevator by appending both
         %global_info.PickLevel & global_info.DropLevel
         global_info.Iterinary            =            [global_info.PickLevel;
global_info.DropLevel ] ;


   %Author: Bilal Amjad
   % Purpose: For updating the itenirary list of the car by removing the 1st
    %      index   value   and   assigning   the   rest   of   the   values   to
global_info.Iterinary
   case 'tAccel'
       %getting the values of global_info.Iterinary
```

```
        Move= eval('global_info.Iterinary');
        %removing the 1st index value and assigning the rest of the values to
global_info.Iterinary
        eval(['global_info.Iterinary','=Move(2:end);']);

end % switch
```

A3.3 Fixed sectoring elevator algorithm:

A3.3.1 Main Simulation File (MSF):

```
% MSF for the Lift Simulator
% Author:Reggie Davidrajuh
% Date: 20 March 2018
% Adapted By: Bilal Amjad
% Date: 1 April 2019

clear all; clc;

global global_info
global_info.STOP_AT = 1000;

global_info.MAX_CAPACITY = 5;   % Maximum capacity to hold passengers in car
global_info.MIN_LEVEL = -2;     % Bottom level floor
global_info.MAX_LEVEL = 2;      % Highest/top level floor

global_info.Current_Level = 0;         % level = MIN_LEVEL .. MAX_LEVEL
global_info.Current_Direction = 'UP';  % direction: 'UP' or 'DOWN'

loadData();

% we have five modules and one inter modular connector
pns = pnstruct({'Init_pdf', 'Arrival_pdf', 'Car_ata_Level_pdf',...
                'Car_Moving_pdf', 'Departure_pdf', 'IMC_pdf'});
 %initial markings
dyn.m0 = {'pInit',1, 'pArrival',global_info.TOTAL_USERS};
%firing times of transitions
dyn.ft = {'tMoving',7, 'tOpenDoor',5,'tCloseDoor',5, ...
          'tAccel',3, 'tDeAccel',3,...
          'tDoorDelay',15, 'tStepON',1, 'tStepOFF',1,'allothers',0.1}; %
dyn.ip = {'tStepOFF',2,'tStepON',1,  'tMoving',1};     %initial priorities
pni = initialdynamics(pns, dyn);
sim = gpensim(pni);
%  prnss(sim);
disp(' ');prnstate('Final markings: ');
% plotp(sim,{'pArrival', 'pWaiting','pAtLevel','pLeft'});
```

A3.3.2 PDF Files:

A3.3.2.1 Init_pdf()

```
% Author:Reggie Davidrajuh
% Adapted By: Bilal Amjad
% PDF for the module Initialization (Init_pdf)

function [png] = Init_pdf()
png.PN_name = 'Init';
png.set_of_Ps = {'pInit'};
png.set_of_Ts = {'tInit'};
png.set_of_As = {'pInit','tInit',1};
```

A3.3.2.2 Arrival_pdf()

```
% Author:Reggie Davidrajuh
% Adapted By: Bilal Amjad
% PDF for Arriving passengers(Arrival_pdf)

function [png] = Arrival_pdf()
png.PN_name = 'Arrival';
png.set_of_Ps = {'pArrival', 'pWaiting', 'pWaitingDUMMY'};
png.set_of_Ts = {'tArrival'};
png.set_of_As = {'pArrival','tArrival',1, 'tArrival','pWaiting',1, ...
                 'tArrival','pWaitingDUMMY',1 ... % for testing only
                };
```

A3.3.2.3 Car_ata_Level_pdf()

```
% Author:Reggie Davidrajuh
% Adapted By: Bilal Amjad
% PDF for the module "Car at a Level" (CAR_ata_Level_pdf)

function [png] = Car_ata_Level_pdf()
png.PN_name = 'Car_ata_Level';
png.set_of_Ps = {'pAtLevel', 'pCloseDoor', 'pDoorOpened',...
                 'pDoorDelay', 'pBoardingComplete'};
png.set_of_Ts = {'tOpenDoor', 'tStepOFF', 'tStepON', ...
                 'tCloseDoor','tDoorDelay', 'tBoardingComplete'};
png.set_of_As = {'tOpenDoor','pAtLevel',1, ... % tOpenDoor
      'pAtLevel','tStepOFF',1, 'tStepOFF','pAtLevel',1, ... % tStepOFF
      'pAtLevel','tStepON',1, 'tStepON','pAtLevel',1, ...   % tStepON
      'pAtLevel','tBoardingComplete',1, ... % tBoardingComplete
      'tBoardingComplete', 'pBoardingComplete',1, ... % tBoardingComplete
      'pBoardingComplete','tCloseDoor',1,... % tCloseDoor
      'pDoorDelay','tCloseDoor',1,...      % tCloseDoor
      'tOpenDoor','pDoorOpened',1,...
      'pDoorOpened','tDoorDelay',1,... % tDoorDelay
      'tDoorDelay','pDoorDelay',1 ... % tDoorDelay
                 };
```

A3.3.2.4 Car_Moving_pdf()

```
% Author:Reggie Davidrajuh
% Adapted By: Bilal Amjad
% PDF for the Moving Car(Car_Moving_pdf)

function [png] = Car_Moving_pdf()
png.PN_name = 'Car_Moving';
png.set_of_Ps = {'pMoving', 'pArrivedAtLevelDUMMY'};
png.set_of_Ts = {'tAccel','tMoving','tDeAccel'};
png.set_of_As = {'tAccel','pMoving',1, ...
        'pMoving','tMoving',1, 'tMoving','pMoving',1, ...
        'tDeAccel', 'pArrivedAtLevelDUMMY', 1, ... % testing ONLY
        'pMoving','tDeAccel',1};
```

A3.3.2.5 Departure_pdf()

```
% Author:Reggie Davidrajuh
% Adapted By: Bilal Amjad
% PDF for Leaving passengers(Departure_pdf)

function [png] = Departure_pdf()
png.PN_name = 'Departure';
png.set_of_Ps = {'pLeft'};
png.set_of_Ts = {'tLeaving'};
png.set_of_As = {'tLeaving','pLeft',1};
```

A3.3.2.6 IMC_pdf()

```
% Author:Reggie Davidrajuh
% Adapted By: Bilal Amjad
% PDF for the inter-modular connections (IMC_pdf)

function [png] = IMC_pdf()
png.PN_name = 'Inter-Modular';
png.set_of_Ps = {'pArrivedAtLevel', 'pAboutToMove', 'pLeaving'};
png.set_of_Ts = {};
png.set_of_As = {... % Init
        'tInit','pArrivedAtLevel',1, ...
        ... % Car_ata_Level
        'pArrivedAtLevel','tOpenDoor',1, ...
        'tCloseDoor','pAboutToMove',1, ...
        'pWaiting','tStepON',1, ...
        'tStepOFF','pLeaving',1, ...
        ... % Car_Moving
        'pAboutToMove','tAccel',1,...
        'tDeAccel','pArrivedAtLevel',1, ...
        ... % Departure
        'pLeaving','tLeaving',1 ...
            };
```

A3.3.3 tArrival_pre(trans)

```
% Author:Reggie Davidrajuh
% Adapted By: Bilal Amjad
% Function name:tArrival_pre
% Purpose: It handles the arrival of each passenger at a floor by comparing
% current time with the arrival time and assigns two colors "L" & "P" which
% represents level and person number of a passenger
function [fire, trans] = tArrival_pre(trans)
global global_info
%will run from minimum to maximum level
for Level = global_info.MIN_LEVEL:global_info.MAX_LEVEL
    %get the value of 'global_info.ATL' e.g global_info.ATL1 frm loadData()
    %function
    Waiting = eval(['global_info.ATL',int2str(Level+3),';']);
    if not(isempty(Waiting))
         %if not empty then get the value
        %present at first index which is passenger's arrival time
        timeOfArrival = Waiting(1);
         % if current time is greater than passenger's arrival time than
         %this is the time for passenger to arrive
        if ge(current_time, timeOfArrival)
            %update the person counter
            global_info.Person_No = global_info.Person_No + 1;
            PNo = global_info.Person_No;
             %second index value shows the destination floor of passenger
            whereTo = Waiting(2);

           % reduce the 1st and 2nd index value from global_info.ATLx
            eval(['global_info.ATL',int2str(Level+3),'=Waiting(3:end);']);
            % time at which passenger arrived
            global_info.PASSENGER_DATA(PNo, 1) = current_time;
            % At which level passenger arrived
            global_info.PASSENGER_DATA(PNo, 4) = Level;
             % To which level passenger wants to go
            global_info.PASSENGER_DATA(PNo, 5) = whereTo;
             %two colors are assigned to each token/passenger
            color1 = ['L', int2str(Level)];
            color2 = ['P', int2str(PNo)];
            trans.new_color = {color1, color2};
            fire = 1;
            disp(['Passenger Arrival: Time:', num2str(current_time),...
                '  at level:', color1, '   Person No:', color2]);
            return
        end
    end
end
fire = 0;
```

A3.3.4 loadData()

```
% Author:Reggie Davidrajuh
% Adapted By: Bilal Amjad
% Function name:loadData
% Purpose: This function contain arrival time & destination floors of
% passengers. It calculates the number of passenger according to historical
% data and creates a matrix/ vector to hold information about passenger like
%person no, arrival and departure time, arrival and departure floor, car
%travelling status, missed car status.


function [] = loadData()
global global_info

global_info.People_in_Car = 0; % initially we have 0 passengers inside car

% Passenger's Historial data: Arrival time of passengers to the lift at
% different levels. Also contain destination floor of passengers
% Test-1:
global_info.ATL1 = [20  -1]; % level -2
global_info.ATL2 = [10 -2]; % level -1
global_info.ATL3 = [30 2, 35 1];  % level 0
global_info.ATL4 = [50 2, 55 0];  % level 1
global_info.ATL5 = [40 1, 45 0];  % level 2



% Test-2:
% global_info.ATL1 = [15 2, 27 -2, 30 1, 40 0, 90 2];                      %
level -2
% global_info.ATL2 = [10 1, 30 0,            75 2, 110 1];    % level -1
% global_info.ATL3 = [20 1,                        100 1, 100 1, 100 1];   %
level 0
% global_info.ATL4 = [0 -2, 25 -2,   35 -1, 60 0, 80 2];% level 1
%   global_info.ATL5   =   [0       0,                                 38   0];
% level 2


% % Test-3:
% global_info.ATL1 = [16 2, 27 9, 30 10, 110 4, 150 7, 170 3, 190 8, 210 7,
252 4, 310 6, 313 10, 313 10, 332 3, 332 3, 403 3, 455 5, 520 5];
% global_info.ATL2 = [30 1, 40 3, 220 4, 239 10, 270 8, 360 4, 480 10, 580
3];
% global_info.ATL3 = [10 1, 30 1, 75 4,  550 1, 580 10, 587 8];
% global_info.ATL4 = [20 2, 100 1, 370 1,728 1, 880 10];
% global_info.ATL5 = [];
% global_info.ATL6 = [0 1,  55 1, 230 10, 550 1, 780 7];
% global_info.ATL7 = [75 3, 90 1, 100 4, 150 3, 200 3, 700 1, 840 10];
% global_info.ATL8 = [25 1, 50 3, 90 2, 120 6, 120 3, 700 1];
% global_info.ATL9 = [35 1, 70 3, 100 1, 203 3, 506 7, 710 10, 823 6];
% global_info.ATL10 =[8 1, 34 1, 204 3, 208 3, 390 5, 390 5, 399 1, 510 5,
512 3, 630 3, 656 8, 700 4, 710 2, 845 6];
```

```
passengers = 0;
%for calculating number of passengers according to historical data
for Level = global_info.MIN_LEVEL:global_info.MAX_LEVEL
    ATLx = eval(['global_info.ATL',int2str(Level+3),';']);
    length(ATLx);
    passengers = passengers + length(ATLx)/2;
end
disp(['Total number of passengers: ', int2str(passengers)]);
global_info.TOTAL_USERS = passengers;
%a matrix/ vector to hold information about passenger like person no,
%arrival and departure time, arrival and departure floor, car travelling
%status,missed car status
global_info.PASSENGER_DATA = zeros(passengers,7);
global_info.Travellers_at_Present = zeros(1, global_info.MAX_CAPACITY);
global_info.Person_No = 0; % Person counter (or person ID number)

for i = 1:global_info.TOTAL_USERS
    eval(['global_info.P2DL_', int2str(i), ' = [];']);
end
```

A3.3.5 fillMissedCarTable()

```
% Author:Reggie Davidrajuh
% Adapted By: Bilal Amjad
% Function name: fillMissedCarTable
%  Purpose:Function  for  saving  information  about  passengers  who  could  not
%enter elevator because it was already full to its max capacity

function [] = fillMissedCarTable()
global global_info

[indices_for_this_level, ~] = ...
    find(global_info.PASSENGER_DATA(:,4)== global_info.Current_Level);

for i = 1:numel(indices_for_this_level)
    row_index = indices_for_this_level(i);

    if not(global_info.PASSENGER_DATA(row_index, 2))
        % shows that passenger has arrived and waiting to step on!!!
        global_info.PASSENGER_DATA(row_index,7) = ...
            global_info.PASSENGER_DATA(row_index,7) + 1;
    end
end
```

A3.3.6 separateColors(the2colors)

```
% Author:Reggie Davidrajuh
% Adapted By: Bilal Amjad
% Function name: separateColors(the2colors)
% Purpose: Function for separating two colors of the token to get the person
%number of a passenger
% Input: the2colors
% Output: PNo

function [PNo] = separateColors(the2colors)

color1 = the2colors{1};
color2 = the2colors{2};

% To find at which Level he arrived and also his passenger number
if strcmp(color1(1), 'L')
    fromWhichLevel = color1;
    PNo = color2;
else
    fromWhichLevel = color2;
    PNo = color1;
end

PNo = PNo(2:end);
PNo = str2double(PNo);
```

A3.3.7 COMMON_PRE(trans)

```
% Function name:COMMON_PRE(trans)
% Purpose: To fire pre-processor code of a transition
function [fire, trans] = COMMON_PRE(trans)
global global_info

switch trans.name
    %Author: Reggie Davidrajuh
    %Adapted By: Bilal Amjad
    %Purpose:To fire transition tInit
    case 'tInit', fire = 1;

    %Author: Reggie Davidrajuh
    %Adapted By: Bilal Amjad
    %Purpose:To fire transition tArrival
    case 'tArrival', fire = 1;

    %Author: Reggie Davidrajuh
    %Adapted By: Bilal Amjad
    %Purpose:To fire transition tOpenDoor
    case 'tOpenDoor', fire = 1;
```

```
     %Author: Reggie Davidrajuh
     %Adapted By: Bilal Amjad
     %Purpose:To fire transition tDoorDelay
    case 'tDoorDelay', trans.override = 1; fire = 1;

     %Author: Reggie Davidrajuh
     %Adapted By: Bilal Amjad
     %Purpose:To fire transition tLeaving
    case 'tLeaving', fire = 1;

     %Author: Reggie Davidrajuh
     %Adapted By: Bilal Amjad
     %Purpose: To display a message on console before closing the elevator
door
    case 'tCloseDoor'
        disp(['**** Closing the door now at Level ',...
            int2str(global_info.Current_Level), ...
            '     at time: ', num2str(current_time), ' *******']);
        disp(' ');
        fire = 1;


    %Author: Reggie Davidrajuh
    %Adapted By: Bilal Amjad
    %Purpose: This transition decides by comparing current level to the
    %destination level of a passenger to step off
    case 'tStepOFF'

         % it shows people who are currently inside car, means travelling
        current_passengers = global_info.PASSENGER_DATA(:, 6);
        peopleInCar = sum(current_passengers);
        if not(peopleInCar), fire = 0; return, end % not allowed to fire if
the car is empty
        [~, PNoInCar] = find(transpose(current_passengers));
        for i = 1:numel(PNoInCar)
            PNo = PNoInCar(i);
            whereTo = global_info.PASSENGER_DATA(PNo, 5);
            %if current level is equal to destination level of specified
person,then that person leaves
            if eq(global_info.Current_Level, whereTo)

                % set travelling in the car status of that person equal to
                % 0, as he is leaving
                global_info.PASSENGER_DATA(PNo, 6) = 0;
                 % saving the leaving time from CAR
                global_info.PASSENGER_DATA(PNo, 3) = current_time;
                disp(['Passenger      is      Leaving      Car:      Time:',
num2str(current_time),...
                    '  at level:', int2str(whereTo),...
                    '   Person No:', int2str(PNo)]);

                fire = 1; return
            end
        end
```

```
        fire = 0;


%Author: Reggie Davidrajuh
%Adapted & Modified By: Bilal Amjad
%Purpose : This transition selects a token to step on, from pWaiting who has
% same color 'L' i.e. arrival floor as the current floor.


% Modification: It first compares the current level with the
% set of floors decided for the fixed sector in the msf
% file i.e. min to max floors. Modifications has been indicated in the code
below as well.

    case 'tStepON'

         currentLevel = global_info.Current_Level;
        % Modifications has been done here
       if eq(currentLevel,1)
          %selecting the token whose arrival level is equal to one
          tokID = tokenAnyColor('pWaiting', 1, ...
                      {'L1'});
        if not(tokID) % if there is no one waiting at this level, then do not
fire
            fire = 0; return
        end

         % Otherwise,check for vacancy in the car, if some people are waiting
at this level
        peopleInCar = sum(global_info.PASSENGER_DATA(:, 6));
        vacant = global_info.MAX_CAPACITY  - peopleInCar;
        if not(vacant)
            disp('**Caution! cannot take more passengers! Car is full ');

            % reducing priority of "tStepON" to make it weak so that it will
not try again & again
            priorset('tStepON', 0);
            fillMissedCarTable();%holds the persons who could not enter the
car because of no vacancy
            fire = 0; return
        end
        trans.selected_tokens = tokID;
        trans.override = 1;
        fire = 1;
        %using separateColors(the2colors) to get the person number to record
        %car entering time and to put travelling stauts=true, of that
        %particular person
        the2colors = get_color('pWaiting', tokID);
        PNo = separateColors(the2colors);

         % the time passenger ENTERS car
        global_info.PASSENGER_DATA(PNo, 2) = current_time;
       % the passenger is inside the car: travelling
        global_info.PASSENGER_DATA(PNo, 6) = true;
        disp(['Passenger is Entering Car: Time:', num2str(current_time),...
```

```
              '  at level:', int2str(currentLevel),...
              '   Person No:', int2str(PNo)])

        % Modifications has been done here
       elseif eq(currentLevel,2)
            %selecting the token whose arrival level is equal to two
           tokID = tokenAnyColor('pWaiting', 1, ...
                       {'L2'});
        if not(tokID) % if there is no one waiting at this level, then do not
fire
            fire = 0; return
        end
         % Otherwise,check for vacancy in the car, if some people are waiting
at this level
        peopleInCar = sum(global_info.PASSENGER_DATA(:, 6));
        vacant = global_info.MAX_CAPACITY  - peopleInCar;
        if not(vacant)
            disp('** Caution! cannot take more passengers! Car is full');

             % reducing priority of "tStepON" to make it weak so that it will
not try again & again
            priorset('tStepON', 0);
            %holds the persons who could not enter the car because of no
vacancy
            fillMissedCarTable();
            fire = 0; return
        end
        trans.selected_tokens = tokID;
        trans.override = 1;
        fire = 1;
        %using separateColors(the2colors) to get the person number to record
        %car entering time and to put travelling stauts=true, of that
        %particular person
        the2colors = get_color('pWaiting', tokID);
        PNo = separateColors(the2colors);

        % the time passenger ENTERS car
        global_info.PASSENGER_DATA(PNo, 2) = current_time;
        % the passenger is inside the car: travelling
        global_info.PASSENGER_DATA(PNo, 6) = true;
        disp(['Passenger is Entering Car: Time:', num2str(current_time),...
              '  at level:', int2str(currentLevel),...
              '   Person No:', int2str(PNo)])

       % Modifications has been done here
       elseif eq(currentLevel,0)
           %selecting the token whose arrival level is equal to zero
          tokID = tokenAnyColor('pWaiting', 1, ...
                       {'L0'});
        if not(tokID) % if there is no one waiting at this level, then do not
fire
            fire = 0; return
        end

        % Otherwise,check for vacancy in the car, if some people are waiting
at this level
```

```
        peopleInCar = sum(global_info.PASSENGER_DATA(:, 6));
        vacant = global_info.MAX_CAPACITY  - peopleInCar;
        if not(vacant)
            disp('**Caution! cannot take more passengers! Car is full');

            % reducing priority of "tStepON" to make it weak so that it will
not try again & again
            priorset('tStepON', 0);
            %holds the persons who could not enter the car because of no
vacancy
            fillMissedCarTable();
            fire = 0; return
        end
        trans.selected_tokens = tokID;
        trans.override = 1;
        fire = 1;
        %using separateColors(the2colors) to get the person number to record
        %car entering time and to put travelling stauts=true, of that
        %particular person
        the2colors = get_color('pWaiting', tokID);
        PNo = separateColors(the2colors);

        % the time passenger ENTERS car
        global_info.PASSENGER_DATA(PNo, 2) = current_time;
        % the passenger is inside the car: travelling
        global_info.PASSENGER_DATA(PNo, 6) = true;
        disp(['Passenger is Entering Car: Time:', num2str(current_time),...
              '  at level:', int2str(currentLevel),...
              '   Person No:', int2str(PNo)])
          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
          % Modifications has been done here
          elseif eq(currentLevel,-1)
           %selecting the token whose arrival level is equal to zero
          tokID = tokenAnyColor('pWaiting', 1, ...
                      {'L-1'});
        if not(tokID) % if there is no one waiting at this level, then do not
fire
            fire = 0; return
        end

        % Otherwise,check for vacancy in the car, if some people are waiting
at this level
        peopleInCar = sum(global_info.PASSENGER_DATA(:, 6));
        vacant = global_info.MAX_CAPACITY  - peopleInCar;
        if not(vacant)
            disp('** Car is full, cannot take more passengers!');

            % reducing priority of "tStepON" to make it weak so that it will
not try again & again
            priorset('tStepON', 0);
            %holds the persons who could not enter the car because of no
vacancy
            fillMissedCarTable();
            fire = 0; return
        end
```

```
        trans.selected_tokens = tokID;
        trans.override = 1;
        fire = 1;
        %using separateColors(the2colors) to get the person number to record
        %car entering time and to put travelling stauts=true, of that
        %particular person
        the2colors = get_color('pWaiting', tokID);
        PNo = separateColors(the2colors);

        % the time passenger ENTERS car
        global_info.PASSENGER_DATA(PNo, 2) = current_time;
        % the passenger is inside the car: travelling
        global_info.PASSENGER_DATA(PNo, 6) = true;
        disp(['Entering Car: Time:', num2str(current_time),...
              '  at level:', int2str(currentLevel),...
              '   PNo:', int2str(PNo)])

          % Modifications has been done here
          elseif eq(currentLevel,-2)
           %selecting the token whose arrival level is equal to zero
          tokID = tokenAnyColor('pWaiting', 1, ...
                      {'L-2'});
        if not(tokID) % if there is no one waiting at this level, then do not
fire
            fire = 0; return
        end

        % Otherwise,check for vacancy in the car, if some people are waiting
at this level
        peopleInCar = sum(global_info.PASSENGER_DATA(:, 6));
        vacant = global_info.MAX_CAPACITY  - peopleInCar;
        if not(vacant)
            disp('** Car is full, cannot take more passengers!');

            % reducing priority of "tStepON" to make it weak so that it will
not try again & again
            priorset('tStepON', 0);
            %holds the persons who could not enter the car because of no
vacancy
            fillMissedCarTable();
            fire = 0; return
        end
        trans.selected_tokens = tokID;
        trans.override = 1;
        fire = 1;
        %using separateColors(the2colors) to get the person number to record
        %car entering time and to put travelling stauts=true, of that
        %particular person
        the2colors = get_color('pWaiting', tokID);
        PNo = separateColors(the2colors);

        % the time passenger ENTERS car
        global_info.PASSENGER_DATA(PNo, 2) = current_time;
        % the passenger is inside the car: travelling
        global_info.PASSENGER_DATA(PNo, 6) = true;
        disp(['Entering Car: Time:', num2str(current_time),...
```

```
            '  at level:', int2str(currentLevel),...
            '   PNo:', int2str(PNo)])
          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
       else
           fire=0;
       end

%Author: Reggie Davidrajuh
%Adapted By: Bilal Amjad
%Purpose: This transition completes the boarding procedure and also
%  increases  priority of "tStepON" back to 1 as it was set to
%0 previously in "tStepON
    case 'tBoardingComplete'
        % increasing  priority of "tStepON" back to 1 as it was set to
        %0 previously in "tStepON"
        priorset('tStepON', 1);
        fire = 1;

    %%%%%%%%%%%%%%%%%%%%
```

```
%Author: Reggie Davidrajuh
%Adapted By: Bilal Amjad
%Purpose:It moves car by one level; tMoving fires only once for shabbat
%movement

    case 'tMoving'
        if global_info.Starting_to_move
            global_info.Starting_to_move = false;
            fire = 1;
        else
            fire = 0;
        end
    %%%%
```

A3.4 Dynamic sectoring elevator algorithm:

A3.4.1 Main Simulation File (MSF):

```
% MSF for the Lift Simulator
% Author:Reggie Davidrajuh
% Date: 20 March 2018
% Adapted & Modified By: Bilal Amjad
% Date: 20 March 2019
% Modification: To assign different set of floors based on time of the day, I
%am comparing the time of the day with the official working hours i.e between
% 7 AM and 6 PM. Modification has been pointed in the code below.


clear all; clc;

global global_info
global_info.STOP_AT = 1000;

global_info.MAX_CAPACITY = 5;   % Maximum capacity to hold passengers in car

% Modification has been done here
format shortg
%variable to hold the timenow value
global_info.c = clock;

%comparing timenow value with office timings to set elevator to specific
%floors accordingly
if(global_info.c(4)>7 && global_info.c(4)<18 )
disp("Its Morning time");disp('');
disp("Elevator is fixed to bottom floors: -1 & -2");disp('');

global_info.MIN_LEVEL = -2;     % Bottom floor
global_info.MAX_LEVEL = -1;      % top floor

global_info.Current_Level = -1;
global_info.Current_Direction = 'DOWN';

else
disp("Its Night time");disp('');
disp("Elevator is fixed to top floors: 0,1 & 2");disp('');
global_info.MIN_LEVEL = 0;   % Bottom floor
global_info.MAX_LEVEL = 2;   % top floor

global_info.Current_Level = 0;
global_info.Current_Direction = 'UP';
end
loadData();

% we have five modules and one inter modular connector
pns = pnstruct({'Init_pdf', 'Arrival_pdf', 'Car_ata_Level_pdf',...
                'Car_Moving_pdf', 'Departure_pdf', 'IMC_pdf'});
```

```
dyn.m0 = {'pInit',1, 'pArrival',global_info.TOTAL_USERS};
dyn.ft = {'tMoving',7, 'tOpenDoor',5,'tCloseDoor',5, ...
          'tAccel',3, 'tDeAccel',3,...
          'tDoorDelay',15, 'tStepON',1, 'tStepOFF',1,'allothers',0.1}; %
dyn.ip = {'tStepOFF',2,'tStepON',1,  'tMoving',1};
pni = initialdynamics(pns, dyn);
sim = gpensim(pni);
 prnss(sim);
disp(' ');prnstate('Final markings: ');
% plotp(sim,{'pArrival', 'pWaiting','pAtLevel','pLeft'});
```

A3.4.2 PDF Files:

A3.4.2.1 Init_pdf()

```
% Author:Reggie Davidrajuh
% Adapted By: Bilal Amjad
% PDF for the module Initialization (Init_pdf)

function [png] = Init_pdf()
png.PN_name = 'Init';
png.set_of_Ps = {'pInit'};
png.set_of_Ts = {'tInit'};
png.set_of_As = {'pInit','tInit',1};
```

A3.4.2.2 Arrival_pdf()

```
% Author:Reggie Davidrajuh
% Adapted By: Bilal Amjad
% PDF for Arriving passengers(Arrival_pdf)

function [png] = Arrival_pdf()
png.PN_name = 'Arrival';
png.set_of_Ps = {'pArrival', 'pWaiting', 'pWaitingDUMMY'};
png.set_of_Ts = {'tArrival'};
png.set_of_As = {'pArrival','tArrival',1, 'tArrival','pWaiting',1, ...
                 'tArrival','pWaitingDUMMY',1 ... % for testing only
                };
```

A3.4.2.3 Car_ata_Level_pdf()

```
% Author:Reggie Davidrajuh
% Adapted By: Bilal Amjad
% PDF for the module "Car at a Level" (CAR_ata_Level_pdf)

function [png] = Car_ata_Level_pdf()
png.PN_name = 'Car_ata_Level';
png.set_of_Ps = {'pAtLevel', 'pCloseDoor', 'pDoorOpened',...
                 'pDoorDelay', 'pBoardingComplete'};
png.set_of_Ts = {'tOpenDoor', 'tStepOFF', 'tStepON', ...
                 'tCloseDoor','tDoorDelay', 'tBoardingComplete'};
png.set_of_As = {'tOpenDoor','pAtLevel',1, ... % tOpenDoor
      'pAtLevel','tStepOFF',1, 'tStepOFF','pAtLevel',1, ... % tStepOFF
      'pAtLevel','tStepON',1, 'tStepON','pAtLevel',1, ...   % tStepON
```

```
        'pAtLevel','tBoardingComplete',1, ... % tBoardingComplete
        'tBoardingComplete', 'pBoardingComplete',1, ... % tBoardingComplete
        'pBoardingComplete','tCloseDoor',1,... % tCloseDoor
        'pDoorDelay','tCloseDoor',1,...      % tCloseDoor
        'tOpenDoor','pDoorOpened',1,...
        'pDoorOpened','tDoorDelay',1,... % tDoorDelay
        'tDoorDelay','pDoorDelay',1 ... % tDoorDelay
                  };
```

A3.4.2.4 Car_Moving_pdf()

```
% Author:Reggie Davidrajuh
% Adapted By: Bilal Amjad
% PDF for the Moving Car(Car_Moving_pdf)

function [png] = Car_Moving_pdf()
png.PN_name = 'Car_Moving';
png.set_of_Ps = {'pMoving', 'pArrivedAtLevelDUMMY'};
png.set_of_Ts = {'tAccel','tMoving','tDeAccel'};
png.set_of_As = {'tAccel','pMoving',1, ...
        'pMoving','tMoving',1, 'tMoving','pMoving',1, ...
        'tDeAccel', 'pArrivedAtLevelDUMMY', 1, ... % testing ONLY
        'pMoving','tDeAccel',1};
```

A3.4.2.5 Departure_pdf()

```
% Author:Reggie Davidrajuh
% Adapted By: Bilal Amjad
% PDF for Leaving passengers(Departure_pdf)

function [png] = Departure_pdf()
png.PN_name = 'Departure';
png.set_of_Ps = {'pLeft'};
png.set_of_Ts = {'tLeaving'};
png.set_of_As = {'tLeaving','pLeft',1};
```

A3.4.2.6 IMC_pdf()

```
% Author:Reggie Davidrajuh
% Adapted By: Bilal Amjad
% PDF for the inter-modular connections (IMC_pdf)

function [png] = IMC_pdf()
png.PN_name = 'Inter-Modular';
png.set_of_Ps = {'pArrivedAtLevel', 'pAboutToMove', 'pLeaving'};
png.set_of_Ts = {};
png.set_of_As = {... % Init
        'tInit','pArrivedAtLevel',1, ...
        ... % Car_ata_Level
        'pArrivedAtLevel','tOpenDoor',1, ...
        'tCloseDoor','pAboutToMove',1, ...
        'pWaiting','tStepON',1, ...
        'tStepOFF','pLeaving',1, ...
```

```
        ... % Car_Moving
        'pAboutToMove','tAccel',1,...
        'tDeAccel','pArrivedAtLevel',1, ...
        ... % Departure
        'pLeaving','tLeaving',1 ...
            };
```

A3.4.3 tArrival_pre(trans)

```
% Author:Reggie Davidrajuh
% Adapted By: Bilal Amjad
% Function name:tArrival_pre
% Purpose: It handles the arrival of each passenger at a floor by comparing
% current time with the arrival time and assigns two colors "L" & "P" which
% represents level and person number of a passenger
function [fire, trans] = tArrival_pre(trans)
global global_info
%will run from minimum to maximum level
for Level = global_info.MIN_LEVEL:global_info.MAX_LEVEL
    %get the value of 'global_info.ATL' e.g global_info.ATL1 frm loadData()
    %function
    Waiting = eval(['global_info.ATL',int2str(Level+3),';']);
    if not(isempty(Waiting))
        %if not empty then get the value
        %present at first index which is passenger's arrival time
        timeOfArrival = Waiting(1);
        % if current time is greater than passenger's arrival time than
         %this is the time for passenger to arrive
        if ge(current_time, timeOfArrival)
         %update the person counter
            global_info.Person_No = global_info.Person_No + 1;
            PNo = global_info.Person_No;
            %second index value shows the destination floor of passenger
            whereTo = Waiting(2);

            % reduce the 1st and 2nd index value from global_info.ATLx
            eval(['global_info.ATL',int2str(Level+3),'=Waiting(3:end);']);
            % time at which passenger arrived
            global_info.PASSENGER_DATA(PNo, 1) = current_time;
            % At which level passenger arrived
            global_info.PASSENGER_DATA(PNo, 4) = Level;
             % To which level passenger wants to go
            global_info.PASSENGER_DATA(PNo, 5) = whereTo;
            %two colors are assigned to each token/passenger
            color1 = ['L', int2str(Level)];
            color2 = ['P', int2str(PNo)];
            trans.new_color = {color1, color2};
            fire = 1;
            disp(['Passenger Arrival: Time:', num2str(current_time),...
                '  at level:', color1, '   Person No:', color2]);
            return
        end
    end
end
fire = 0;
```

A3.4.4 loadData()

```
% Author:Reggie Davidrajuh
% Adapted By: Bilal Amjad
% Function name:loadData
% Purpose: This function contain arrival time & destination floors of
% passengers. It calculates the number of passenger according to historical
% data and creates a matrix/ vector to hold information about passenger like
%person no, arrival and departure time, arrival and departure floor, car
%travelling status,missed car status.


function [] = loadData()
global global_info

global_info.People_in_Car = 0; % initially we have 0 passengers inside car

% Passenger's Historial data: Arrival time of passengers to the lift at
% different levels. Also contain destination floor of passengers
% Test-1:
global_info.ATL1 = [20 -1, 25 -1, 30 -1]; % level -2
global_info.ATL2 = [10 -2, 15 -2, 20 -2]; % level -1
global_info.ATL3 = [30 2, 35 1];  % level 0
global_info.ATL4 = [50 2, 55 0];  % level 1
global_info.ATL5 = [40 1, 45 0];  % level 2

% Test-2:
% global_info.ATL1 = [15 2, 27 -2, 30 1, 40 0, 90 2];                      %
level -2
% global_info.ATL2 = [10 1, 30 0,            75 2, 110 1];    % level -1
% global_info.ATL3 = [20 1,                          100 1, 100 1, 100 1];  %
level 0
% global_info.ATL4 = [0 -2, 25 -2,   35 -1, 60 0, 80 2];% level 1
%   global_info.ATL5   =   [0       0,                                38   0];
% level 2


% % Test-3:
% global_info.ATL1 = [16 2, 27 9, 30 10, 110 4, 150 7, 170 3, 190 8, 210 7,
252 4, 310 6, 313 10, 313 10, 332 3, 332 3, 403 3, 455 5, 520 5];
% global_info.ATL2 = [30 1, 40 3, 220 4, 239 10, 270 8, 360 4, 480 10, 580
3];
% global_info.ATL3 = [10 1, 30 1, 75 4,  550 1, 580 10, 587 8];
% global_info.ATL4 = [20 2, 100 1, 370 1,728 1, 880 10];
% global_info.ATL5 = [];
% global_info.ATL6 = [0 1,  55 1, 230 10, 550 1, 780 7];
% global_info.ATL7 = [75 3, 90 1, 100 4, 150 3, 200 3, 700 1, 840 10];
% global_info.ATL8 = [25 1, 50 3, 90 2, 120 6, 120 3, 700 1];
% global_info.ATL9 = [35 1, 70 3, 100 1, 203 3, 506 7, 710 10, 823 6];
% global_info.ATL10 =[8 1, 34 1, 204 3, 208 3, 390 5, 390 5, 399 1, 510 5,
512 3, 630 3, 656 8, 700 4, 710 2, 845 6];
```

```
passengers = 0;
%for calculating number of passengers according to historical data
for Level = global_info.MIN_LEVEL:global_info.MAX_LEVEL
    ATLx = eval(['global_info.ATL',int2str(Level+3),';']);
    length(ATLx);
    passengers = passengers + length(ATLx)/2;
end
disp(['Total number of passengers: ', int2str(passengers)]);
global_info.TOTAL_USERS = passengers;
%a matrix/ vector to hold information about passenger like person no,
%arrival and departure time, arrival and departure floor, car travelling
%status,missed car status
global_info.PASSENGER_DATA = zeros(passengers,7);
global_info.Travellers_at_Present = zeros(1, global_info.MAX_CAPACITY);
global_info.Person_No = 0; % Person counter (or person ID number)

for i = 1:global_info.TOTAL_USERS
    eval(['global_info.P2DL_', int2str(i), ' = [];']);
end
```

A3.4.5 fillMissedCarTable()

```
% Author:Reggie Davidrajuh
% Adapted By: Bilal Amjad
% Function name: fillMissedCarTable
%  Purpose:Function  for  saving  information  about  passengers  who  could  not
%enter elevator because it was already full to its max capacity

function [] = fillMissedCarTable()
global global_info

[indices_for_this_level, ~] = ...
    find(global_info.PASSENGER_DATA(:,4)== global_info.Current_Level);

for i = 1:numel(indices_for_this_level)
    row_index = indices_for_this_level(i);

    if not(global_info.PASSENGER_DATA(row_index, 2))
         % shows that passenger has arrived and waiting to step on!!!
        global_info.PASSENGER_DATA(row_index,7) = ...
            global_info.PASSENGER_DATA(row_index,7) + 1;
    end
end
```

A3.4.6 separateColors(the2colors)

```
% Author:Reggie Davidrajuh
% Adapted By: Bilal Amjad
% Function name: separateColors(the2colors)
% Purpose: Function for separating two colors of the token to get the person
%number of a passenger
% Input: the2colors
```

```
% Output: PNo
function [PNo] = separateColors(the2colors)

color1 = the2colors{1};
color2 = the2colors{2};

% To find at which Level he arrived and also his passenger number
if strcmp(color1(1), 'L')
    fromWhichLevel = color1;
    PNo = color2;
else
    fromWhichLevel = color2;
    PNo = color1;
end

PNo = PNo(2:end);
PNo = str2double(PNo);
```

A3.4.7 COMMON_PRE(trans)

```
% Function name:COMMON_PRE(trans)
% Purpose: To fire pre-processor code of a transition
function [fire, trans] = COMMON_PRE(trans)
global global_info

switch trans.name
    %Author: Reggie Davidrajuh
    %Adapted By: Bilal Amjad
    %Purpose:To fire transition tInit
    case 'tInit', fire = 1;

    %Author: Reggie Davidrajuh
    %Adapted By: Bilal Amjad
    %Purpose:To fire transition tArrival
    case 'tArrival', fire = 1;

    %Author: Reggie Davidrajuh
    %Adapted By: Bilal Amjad
    %Purpose:To fire transition tOpenDoor
    case 'tOpenDoor', fire = 1;

     %Author: Reggie Davidrajuh
     %Adapted By: Bilal Amjad
     %Purpose:To fire transition tDoorDelay
    case 'tDoorDelay', trans.override = 1; fire = 1;

     %Author: Reggie Davidrajuh
     %Adapted By: Bilal Amjad
     %Purpose:To fire transition tLeaving
    case 'tLeaving', fire = 1;
```

```
     %Author: Reggie Davidrajuh
     %Adapted By: Bilal Amjad
     %Purpose: To display a message on console before closing the elevator
%door
    case 'tCloseDoor'
        disp(['**** Closing the door now at Level ',...
            int2str(global_info.Current_Level), ...
            '     at time: ', num2str(current_time), ' *******']);
        disp(' ');
        fire = 1;

    %Author: Reggie Davidrajuh
    %Adapted By: Bilal Amjad
    %Purpose: This transition decides by comparing current level to the
    %destination level of a passenger to step off
    case 'tStepOFF'

         % it shows people who are currently inside car, means travelling
        current_passengers = global_info.PASSENGER_DATA(:, 6);
        peopleInCar = sum(current_passengers);
        if not(peopleInCar), fire = 0; return, end % not allowed to fire if
the car is empty
        [~, PNoInCar] = find(transpose(current_passengers));
        for i = 1:numel(PNoInCar)
            PNo = PNoInCar(i);
            whereTo = global_info.PASSENGER_DATA(PNo, 5);
            %if current level is equal to destination level of specified
person,then that person leaves
            if eq(global_info.Current_Level, whereTo)

               % set travelling in the car status of that person equal to
                % 0, as he is leaving
                global_info.PASSENGER_DATA(PNo, 6) = 0;
                % saving the leaving time from CAR
                global_info.PASSENGER_DATA(PNo, 3) = current_time;
                disp(['Passenger      is      Leaving      Car:      Time:',
num2str(current_time),...
                     '  at level:', int2str(whereTo),...
                     '   Person No:', int2str(PNo)]);
                fire = 1; return
            end
        end
        fire = 0;

%Author: Reggie Davidrajuh
%Adapted & Modified By: Bilal Amjad
%Purpose : This transition selects a token to step on, from pWaiting who has
%same color 'L' i.e arrival floor as the current floor.

% Modification: It first compare the time of the day that whether it's
% Morning time or night time. Accordingly, it then compares the current level
%with the set of floors decided for morning and night time in the msf
```

```
% file. Modifications has been indicated in the code below as well.


    case 'tStepON'

         currentLevel = global_info.Current_Level;
% Modifications has been done here
%if timenow(global_info.c(4)) value is between 8 am and 5 pm, then elevator
% is fixed to floors: -1 % -2
    if (global_info.c(4)>7 && global_info.c(4)<18 )
       if eq(currentLevel,-1)
           %selecting the token whose arrival level is equal to -1
          tokID = tokenAnyColor('pWaiting', 1, ...
                      {'L-1'});
        % if there is no one waiting at this level, then do not fire
        if not(tokID)
            fire = 0; return
        end

         % Otherwise,check for vacancy in the car, if some people
         %are waiting at this level
        peopleInCar = sum(global_info.PASSENGER_DATA(:, 6));
        vacant = global_info.MAX_CAPACITY  - peopleInCar;
        if not(vacant)
            disp('** Caution! cannot take more passengers! Car is full');

             % reducing priority of "tStepON" to make it weak so that it
             %will not try again & again
            priorset('tStepON', 0);
      %holds the persons who could not enter the car because of no vacancy
            fillMissedCarTable();
            fire = 0; return
        end
        trans.selected_tokens = tokID;
        trans.override = 1;
        fire = 1;
       %using separateColors(the2colors) to get the person number to record
        %car entering time and to put travelling stauts=true, of that
        %particular person
        the2colors = get_color('pWaiting', tokID);
        PNo = separateColors(the2colors);

        % the time passenger ENTERS car
        global_info.PASSENGER_DATA(PNo, 2) = current_time;
        % the passenger is inside the car: travelling
        global_info.PASSENGER_DATA(PNo, 6) = true;
        disp(['Passenger is Entering Car: Time:', num2str(current_time),...
              '  at level:', int2str(currentLevel),...
              '   Person No:', int2str(PNo)])

       % Modifications has been done here
       elseif eq(currentLevel,-2)
            %selecting the token whose arrival level is equal to -2
           tokID = tokenAnyColor('pWaiting', 1, ...
                      {'L-2'});
         % if there is no one waiting at this level, then do not fire
```

```
    if not(tokID)
        fire = 0; return
    end

    % Otherwise,check for vacancy in the car, if some people
     %are waiting at this level
    peopleInCar = sum(global_info.PASSENGER_DATA(:, 6));
    vacant = global_info.MAX_CAPACITY  - peopleInCar;
    if not(vacant)
        disp('** Caution! cannot take more passengers! Car is full');

       % reducing priority of "tStepON" to make it weak so that it
         %will not try again & again
        priorset('tStepON', 0);
   %holds the persons who could not enter the car because of no vacancy
        fillMissedCarTable();
        fire = 0; return
    end
    trans.selected_tokens = tokID;
    trans.override = 1;
    fire = 1;
    %using separateColors(the2colors) to get the person number to record
    %car entering time and to put travelling stauts=true, of that
    %particular person
    the2colors = get_color('pWaiting', tokID);
    PNo = separateColors(the2colors);

    % the time passenger ENTERS car
    global_info.PASSENGER_DATA(PNo, 2) = current_time;
    % the passenger is inside the car: travelling
    global_info.PASSENGER_DATA(PNo, 6) = true;
    disp(['Passenger is Entering Car: Time:', num2str(current_time),...
          '  at level:', int2str(currentLevel),...
          '   Person No:', int2str(PNo)])


   else
       fire=0;
   end

  % Modifications has been done here
% the elevator is fixed to floors: 0, 1 & 2, because it is night time
else
    if eq(currentLevel,0)
     %selecting the token whose arrival level is equal to 0
      tokID = tokenAnyColor('pWaiting', 1, ...
                  {'L0'});
      % if there is no one waiting at this level, then do not fire
    if not(tokID)
        fire = 0; return
    end

    % Otherwise,check for vacancy in the car, if some people
     %are waiting at this level
    peopleInCar = sum(global_info.PASSENGER_DATA(:, 6));
```

```
  vacant = global_info.MAX_CAPACITY  - peopleInCar;
  if not(vacant)
      disp('** Caution! cannot take more passengers! Car is full');

       % reducing priority of "tStepON" to make it weak so that it
       %will not try again & again
      priorset('tStepON', 0);
 %holds the persons who could not enter the car because of no vacancy
      fillMissedCarTable();
      fire = 0; return
  end
  trans.selected_tokens = tokID;
  trans.override = 1;
  fire = 1;
%using separateColors(the2colors) to get the person number to record
  %car entering time and to put travelling stauts=true, of that
  %particular person
  the2colors = get_color('pWaiting', tokID);
  PNo = separateColors(the2colors);

  % the time passenger ENTERS car
  global_info.PASSENGER_DATA(PNo, 2) = current_time;
  % the passenger is inside the car: travelling
  global_info.PASSENGER_DATA(PNo, 6) = true;
  disp(['Passenger is Entering Car: Time:', num2str(current_time),...
        '  at level:', int2str(currentLevel),...
        '   Person No:', int2str(PNo)])

  % Modifications has been done here
 elseif eq(currentLevel,1)
     %selecting the token whose arrival level is equal to 1
     tokID = tokenAnyColor('pWaiting', 1, ...
                {'L1'});
  % if there is no one waiting at this level, then do not fire
  if not(tokID)
      fire = 0; return
  end

  % Otherwise,check for vacancy in the car, if some people
   %are waiting at this level
  peopleInCar = sum(global_info.PASSENGER_DATA(:, 6));
  vacant = global_info.MAX_CAPACITY  - peopleInCar;
  if not(vacant)
      disp('**Caution! cannot take more passengers! Car is full');

     % reducing priority of "tStepON" to make it weak so that it
       %will not try again & again
      priorset('tStepON', 0);
 %holds the persons who could not enter the car because of no vacancy
      fillMissedCarTable();
      fire = 0; return
  end
  trans.selected_tokens = tokID;
  trans.override = 1;
  fire = 1;
 %using separateColors(the2colors) to get the person number to record
```

```
  %car entering time and to put travelling stauts=true, of that
  %particular person
  the2colors = get_color('pWaiting', tokID);
  PNo = separateColors(the2colors);

  % the time passenger ENTERS car
  global_info.PASSENGER_DATA(PNo, 2) = current_time;
  % the passenger is inside the car: travelling
  global_info.PASSENGER_DATA(PNo, 6) = true;
  disp(['Passenger is Entering Car: Time:', num2str(current_time),...
        '  at level:', int2str(currentLevel),...
        '   Person No:', int2str(PNo)])

 % Modifications has been done here
 elseif eq(currentLevel,2)
     %selecting the token whose arrival level is equal to 2
    tokID = tokenAnyColor('pWaiting', 1, ...
                {'L2'});
 % if there is no one waiting at this level, then do not fire
  if not(tokID)
      fire = 0; return
  end

   % Otherwise,check for vacancy in the car, if some people
   %are waiting at this level
  peopleInCar = sum(global_info.PASSENGER_DATA(:, 6));
  vacant = global_info.MAX_CAPACITY  - peopleInCar;
  if not(vacant)
      disp('** Caution! cannot take more passengers! Car is full');

     % reducing priority of "tStepON" to make it weak so that it
       %will not try again & again
      priorset('tStepON', 0);
%holds the persons who could not enter the car because of no vacancy
      fillMissedCarTable();
      fire = 0; return
  end
  trans.selected_tokens = tokID;
  trans.override = 1;
  fire = 1;
 %using separateColors(the2colors) to get the person number to record
  %car entering time and to put travelling stauts=true, of that
  %particular person
  the2colors = get_color('pWaiting', tokID);
  PNo = separateColors(the2colors);

  % the time passenger ENTERS car
  global_info.PASSENGER_DATA(PNo, 2) = current_time;
  % the passenger is inside the car: travelling
  global_info.PASSENGER_DATA(PNo, 6) = true;
  disp(['Passenger is Entering Car: Time:', num2str(current_time),...
        '  at level:', int2str(currentLevel),...
        '   Person No:', int2str(PNo)])
 else
     fire=0;
  end
```

```
    end


%Author: Reggie Davidrajuh
%Adapted By: Bilal Amjad
%Purpose: This transition completes the boarding procedure and also
%  increases  priority of "tStepON" back to 1 as it was set to
%0 previously in "tStepON
    case 'tBoardingComplete'
       % increasing  priority of "tStepON" back to 1 as it was set to
        %0 previously in "tStepON"0
        priorset('tStepON', 1);
        fire = 1;

    %%%%
```

```
%Author: Reggie Davidrajuh
%Adapted By: Bilal Amjad
%Purpose:It moves car by one level; tMoving fires only once for shabbat
%movement

     case 'tMoving'
        if global_info.Starting_to_move
            global_info.Starting_to_move = false;
            fire = 1;
        else
            fire = 0;
        end
    %%%%
```

```
end % switch
```

A3.5 First come first serve elevator with VIP pass:

A3.5.1 Main Simulation File (MSF):

```
% MSF for the Lift Simulator
% Author:Reggie Davidrajuh
% Date: 20 March 2018
% Adapted By: Bilal Amjad
% Date: 28 Feb 2019

clear all; clc;

global global_info
global_info.STOP_AT = 1000;

global_info.MAX_CAPACITY = 5;   % Maximum capacity to hold passengers in car
global_info.MIN_LEVEL = -2;     % Bottom level floor
global_info.MAX_LEVEL = 2;      % Highest/top level floor

global_info.Current_Level = 0;         % level = MIN_LEVEL .. MAX_LEVEL
% global_info.Current_Direction = 'UP';  % direction: 'UP' or 'DOWN'

loadData();

% we have five modules and one inter modular connector
pns = pnstruct({'Init_pdf', 'Arrival_pdf', 'Car_ata_Level_pdf',...
                'Car_Moving_pdf', 'Departure_pdf', 'IMC_pdf'});

dyn.m0 = {'pInit',1, 'pArrival',global_info.TOTAL_USERS};
dyn.ft = {'tMoving',7, 'tOpenDoor',5,'tCloseDoor',5, ...
          'tAccel',3, 'tDeAccel',3,...
          'tDoorDelay',15, 'tStepON',1, 'tStepOFF',1,'allothers',0.1}; %
dyn.ip = {'tStepOFF',2,'tStepON',1,  'tMoving',1};
pni = initialdynamics(pns, dyn);
sim = gpensim(pni);
 prnss(sim);
disp(' ');prnstate('Final markings: ');
% plotp(sim,{'pArrival', 'pWaiting','pAtLevel','pLeft'});
```

A3.5.2 PDF Files:

A3.5.2.1 Init_pdf()

```
% Author:Reggie Davidrajuh
% Adapted By: Bilal Amjad
% PDF for the module Initialization (Init_pdf)

function [png] = Init_pdf()
png.PN_name = 'Init';
png.set_of_Ps = {'pInit'};
```

```
png.set_of_Ts = {'tInit'};
png.set_of_As = {'pInit','tInit',1};
```

A3.5.2.2 Arrival_pdf()

```
% Author:Reggie Davidrajuh
% Adapted By: Bilal Amjad
% PDF for Arriving passengers(Arrival_pdf)

function [png] = Arrival_pdf()
png.PN_name = 'Arrival';
png.set_of_Ps = {'pArrival', 'pWaiting', 'pWaitingDUMMY'};
png.set_of_Ts = {'tArrival'};
png.set_of_As = {'pArrival','tArrival',1, 'tArrival','pWaiting',1, ...
                 'tArrival','pWaitingDUMMY',1 ... % for testing only
                };
```

A3.5.2.3 Car_ata_Level_pdf()

```
% Author:Reggie Davidrajuh
% Adapted By: Bilal Amjad
% PDF for the module "Car at a Level" (CAR_ata_Level_pdf)

function [png] = Car_ata_Level_pdf()
png.PN_name = 'Car_ata_Level';
png.set_of_Ps = {'pAtLevel', 'pCloseDoor', 'pDoorOpened',...
                 'pDoorDelay', 'pBoardingComplete'};
png.set_of_Ts = {'tOpenDoor', 'tStepOFF', 'tStepON', ...
                 'tCloseDoor','tDoorDelay', 'tBoardingComplete'};
png.set_of_As = {'tOpenDoor','pAtLevel',1, ... % tOpenDoor
      'pAtLevel','tStepOFF',1, 'tStepOFF','pAtLevel',1, ... % tStepOFF
      'pAtLevel','tStepON',1, 'tStepON','pAtLevel',1, ...   % tStepON
      'pAtLevel','tBoardingComplete',1, ... % tBoardingComplete
      'tBoardingComplete', 'pBoardingComplete',1, ... % tBoardingComplete
      'pBoardingComplete','tCloseDoor',1,... % tCloseDoor
      'pDoorDelay','tCloseDoor',1,...      % tCloseDoor
      'tOpenDoor','pDoorOpened',1,...
      'pDoorOpened','tDoorDelay',1,... % tDoorDelay
      'tDoorDelay','pDoorDelay',1 ... % tDoorDelay
                 };
```

A3.5.2.4 Car_Moving_pdf()

```
% Author:Reggie Davidrajuh
% Adapted By: Bilal Amjad
% PDF for the Moving Car(Car_Moving_pdf)

function [png] = Car_Moving_pdf()
png.PN_name = 'Car_Moving';
png.set_of_Ps = {'pMoving', 'pArrivedAtLevelDUMMY'};
png.set_of_Ts = {'tAccel','tMoving','tDeAccel'};
png.set_of_As = {'tAccel','pMoving',1, ...
        'pMoving','tMoving',1, 'tMoving','pMoving',1, ...
        'tDeAccel', 'pArrivedAtLevelDUMMY', 1, ... % testing ONLY
```

```
        'pMoving','tDeAccel',1};
```

A3.5.2.5 Departure_pdf()

```
% Author:Reggie Davidrajuh
% Adapted By: Bilal Amjad
% PDF for Leaving passengers(Departure_pdf)

function [png] = Departure_pdf()
png.PN_name = 'Departure';
png.set_of_Ps = {'pLeft'};
png.set_of_Ts = {'tLeaving'};
png.set_of_As = {'tLeaving','pLeft',1};
```

A3.5.2.6 IMC_pdf()

```
%% Author:Reggie Davidrajuh
% Adapted By: Bilal Amjad
% PDF for the inter-modular connections (IMC_pdf)

function [png] = IMC_pdf()
png.PN_name = 'Inter-Modular';
png.set_of_Ps = {'pArrivedAtLevel', 'pAboutToMove', 'pLeaving'};
png.set_of_Ts = {};
png.set_of_As = {... % Init
        'tInit','pArrivedAtLevel',1, ...
        ... % Car_ata_Level
        'pArrivedAtLevel','tOpenDoor',1, ...
        'tCloseDoor','pAboutToMove',1, ...
        'pWaiting','tStepON',1, ...
        'tStepOFF','pLeaving',1, ...
        ... % Car_Moving
        'pAboutToMove','tAccel',1,...
        'tDeAccel','pArrivedAtLevel',1, ...
        ... % Departure
        'pLeaving','tLeaving',1 ...
            };
```

A3.5.3 tArrival_pre(trans)

```
% Author:Reggie Davidrajuh
%Adapted & Modified By: Bilal Amjad
%Function name:tArrival_pre
% Purpose: It handles the arrival of each passenger at a floor by comparing
% current time with the arrival time and assigns two colors "L" & "P" which
% represents level and person number of a passenger
%Modification: In this algorithm we have on more parameter in historical
%data of a passenger in loadData() function which is passcode holding status
%i.e 0 or 1.
%For example: global_info.ATL1 = [20 0 0]. Third value which is
%0,represents passcode holding status. First and second represent arrival
time %& destination floor.
```

```
function [fire, trans] = tArrival_pre(trans)
global global_info
%will run from minimum to maximum level
for Level = global_info.MIN_LEVEL:global_info.MAX_LEVEL
    %get the value of 'global_info.ATL' e.g global_info.ATL1 frm loadData()
    %function
    Waiting = eval(['global_info.ATL',int2str(Level+3),';']);
    if not(isempty(Waiting))
        %if not empty then get the value
        %present at first index which is passenger's arrival time
        timeOfArrival = Waiting(1);
         % if current time is greater than passenger's arrival time than
         %this is the time for passenger to arrive
        if ge(current_time, timeOfArrival)
            %update the person counter
            global_info.Person_No = global_info.Person_No + 1;
            PNo = global_info.Person_No;
             %second index value shows the destination floor of passenger
            whereTo = Waiting(2);
             %third index value will tell that whether a passenger has
             %passcode or not i.e. bool value: 0 or 1
              vipPass=Waiting(3);
          %Modification has been done here. Now we are removing 3rd index
          %value from global_info.ATLx also.
          % reduce the 1st,2nd and 3rd index value from global_info.ATLx
            eval(['global_info.ATL',int2str(Level+3),'=Waiting(4:end);']);
             % time at which passenger arrived
            global_info.PASSENGER_DATA(PNo, 1) = current_time;
             % At which level passenger arrived
            global_info.PASSENGER_DATA(PNo, 4) = Level; % from which level
             % To which level passenger wants to go
            global_info.PASSENGER_DATA(PNo, 5) = whereTo; % to which level

            %Modification has been done here by saving passcode holding
            %status to  global_info.PASSENGER_DATA(PNo, 8)
            %hold value which shows that passenger has passcode or not i.e
            %0 or 1
            global_info.PASSENGER_DATA(PNo, 8) = vipPass;
            %two colors are assigned to each token/passenger
            color1 = ['L', int2str(Level)];
            color2 = ['P', int2str(PNo)];
            trans.new_color = {color1, color2};
            fire = 1;
            disp(['Passenger Arrival: Time:', num2str(current_time),...
                '  at level:', color1, '   Person No:', color2]);
            return
        end
    end
end
fire = 0;
```

A3.5.4 loadData()

```
% Author:Reggie Davidrajuh
% Adapted & Modified By: Bilal Amjad
% Function name:loadData
% Purpose: This function contain arrival time & destination floors of
% passengers. It calculates the number of passenger according to historical
% data and creates a matrix/ vector to hold information about passenger like
%person no, arrival and departure time, arrival and departure floor, car
%travelling status,missed car status.

 % Modifications have been discussed in the code below

function [] = loadData()
global global_info

global_info.People_in_Car = 0; % initially we have 0 passengers inside car

% Passenger's Historial data: Arrival time of passengers to the lift at
% different levels. Also contain destination floor of passengers and
% passcode holding status value i.e. 0 or 1
% Test-1:
global_info.ATL1 = [20  0 0]; % level -2
global_info.ATL2 = [10 2 1,20 1 1,24 -2 1]; % level -1
global_info.ATL3 = [26 2 0];  % level 0
global_info.ATL4 = [15 -2 1, 18 -1 1];  % level 1
global_info.ATL5 = [22 -1 0];  % level 2

% Test-2:
% global_info.ATL1 = [15 2, 27 -2, 30 1, 40 0, 90 2];                     %
level -2
% global_info.ATL2 = [10 1, 30 0,            75 2, 110 1];    % level -1
% global_info.ATL3 = [20 1,                         100 1, 100 1, 100 1];  %
level 0
% global_info.ATL4 = [0 -2, 25 -2,   35 -1, 60 0, 80 2];% level 1
%   global_info.ATL5   =   [0      0,                                38   0];
% level 2


% % Test-3:
% global_info.ATL1 = [16 2, 27 9, 30 10, 110 4, 150 7, 170 3, 190 8, 210 7,
252 4, 310 6, 313 10, 313 10, 332 3, 332 3, 403 3, 455 5, 520 5];
% global_info.ATL2 = [30 1, 40 3, 220 4, 239 10, 270 8, 360 4, 480 10, 580
3];
% global_info.ATL3 = [10 1, 30 1, 75 4,  550 1, 580 10, 587 8];
% global_info.ATL4 = [20 2, 100 1, 370 1,728 1, 880 10];
% global_info.ATL5 = [];
% global_info.ATL6 = [0 1,  55 1, 230 10, 550 1, 780 7];
% global_info.ATL7 = [75 3, 90 1, 100 4, 150 3, 200 3, 700 1, 840 10];
% global_info.ATL8 = [25 1, 50 3, 90 2, 120 6, 120 3, 700 1];
% global_info.ATL9 = [35 1, 70 3, 100 1, 203 3, 506 7, 710 10, 823 6];
% global_info.ATL10 =[8 1, 34 1, 204 3, 208 3, 390 5, 390 5, 399 1, 510 5,
512 3, 630 3, 656 8, 700 4, 710 2, 845 6];
```

```
passengers = 0;
%for calculating number of passengers according to historical data
for Level = global_info.MIN_LEVEL:global_info.MAX_LEVEL
    ATLx = eval(['global_info.ATL',int2str(Level+3),';']);
    %Modification has been doon here. Here we are dividing "length(ATLx)/3"
by three as compared to other algorithms
    %where it has been divided by 2 because here we have introduced third
    %parameter of a passenger in the historical data above i.e passcode
holding status
    passengers = passengers + length(ATLx)/3;
end
disp(['Total number of passengers: ', int2str(passengers)]);
global_info.TOTAL_USERS = passengers;
%Modification has been doon here. The size of global_info.PASSENGER_DATA
%has been increased to 8 instead of 7 to add a new column for saving passcode
holding status
%a matrix/ vector to hold information about passenger like person no,
%arrival and departure time, arrival and departure floor, car travelling
%status,missed car status, passcode holding status
global_info.PASSENGER_DATA = zeros(passengers,8);
global_info.Travellers_at_Present = zeros(1, global_info.MAX_CAPACITY);
global_info.Person_No = 0; % Person counter (or person ID number)

for i = 1:global_info.TOTAL_USERS
    eval(['global_info.P2DL_', int2str(i), ' = [];']);
end
```

A3.5.5 fillMissedCarTable()

```
% Author:Reggie Davidrajuh
% Adapted By: Bilal Amjad
% Function name: fillMissedCarTable
%  Purpose:Function for saving information about passengers who could not
enter
% elevator because it was already full to its max capacity
function [] = fillMissedCarTable()
global global_info

[indices_for_this_level, ~] = ...
    find(global_info.PASSENGER_DATA(:,4)== global_info.Current_Level);

for i = 1:numel(indices_for_this_level)
    row_index = indices_for_this_level(i);

    if not(global_info.PASSENGER_DATA(row_index, 2))
       % shows that passenger has arrived and waiting to step on!!!
        global_info.PASSENGER_DATA(row_index,7) = ...
            global_info.PASSENGER_DATA(row_index,7) + 1;
    end
end
```

A3.5.6 separateColors(the2colors)

```
% Author:Reggie Davidrajuh
% Adapted By: Bilal Amjad
% Function name: separateColors(the2colors)
% Purpose: Function for separating two colors of the token to get the person
%number of a passenger
% Input: the2colors
% Output: PNo
function [PNo] = separateColors(the2colors)

color1 = the2colors{1};
color2 = the2colors{2};

% To find at which Level he arrived and also his passenger number
if strcmp(color1(1), 'L')
    fromWhichLevel = color1;
    PNo = color2;
else
    fromWhichLevel = color2;
    PNo = color1;
end

PNo = PNo(2:end);
PNo = str2double(PNo);
```

A3.5.7 COMMON_PRE(trans)

```
% Function name:COMMON_PRE(trans)
% Purpose: To fire pre-processor code of a transition
function [fire, trans] = COMMON_PRE(trans)
global global_info

switch trans.name
    % Author: Reggie Davidrajuh
    % Adapted By: Bilal Amjad
    % Purpose:To fire transition tInit
    case 'tInit', fire = 1;

     % Author: Reggie Davidrajuh
     % Adapted By: Bilal Amjad
     % Purpose:To fire transition tArrival
    case 'tArrival', fire = 1;

     % Author: Reggie Davidrajuh
     % Adapted By: Bilal Amjad
     % Purpose:To fire transition tOpenDoor
    case 'tOpenDoor', fire = 1;

     %Author: Reggie Davidrajuh
     %Adapted By: Bilal Amjad
     %Purpose:To fire transition tDoorDelay
    case 'tDoorDelay', trans.override = 1; fire = 1;
```

```
     %Author: Reggie Davidrajuh
     %Adapted By: Bilal Amjad
     %Purpose:To fire transition tLeaving
    case 'tLeaving', fire = 1;


     %Author: Reggie Davidrajuh
     %Adapted By: Bilal Amjad
     %Purpose: To display a message on console before closing the elevator
%door
    case 'tCloseDoor'
        disp(['**** Closing the door now at Level ',...
            int2str(global_info.Current_Level), ...
            '     at time: ', num2str(current_time), ' *******']);
        disp(' ');
        fire = 1;



    %Author: Reggie Davidrajuh
    %Adapted By: Bilal Amjad
    %Purpose: This transition decides by comparing current level to the
    %destination level of a passenger to step off
    case 'tStepOFF'

         % it shows people who are currently inside car, means travelling
        current_passengers = global_info.PASSENGER_DATA(:, 6);
        peopleInCar = sum(current_passengers);
        if not(peopleInCar), fire = 0; return, end %not allowed to fire if
the car is empty
        [~, PNoInCar] = find(transpose(current_passengers));
        for i = 1:numel(PNoInCar)
            PNo = PNoInCar(i);
            whereTo = global_info.PASSENGER_DATA(PNo, 5);
             %if current level is equal to destination level of specified
person,then that person leaves
            if eq(global_info.Current_Level, whereTo)

                % set travelling in the car status of that person equal to
                % 0, as he is leaving
                global_info.PASSENGER_DATA(PNo, 6) = 0;
               % saving the leaving time from CAR
                global_info.PASSENGER_DATA(PNo, 3) = current_time;
                disp(['Passenger      is      Leaving      Car:      Time:',
num2str(current_time),...
                    '  at level:', int2str(whereTo),...
                    '   Person No:', int2str(PNo)]);

                fire = 1; return
            end
        end
        fire = 0;
```

```
        %Author: Reggie Davidrajuh
        %Adapted & Modified By: Bilal Amjad
        %Purpose : This transition selects a token to step on, from pWaiting
%who has same color 'L' i.e arrival floor as the current floor and it only
%fires if the passenger who has been selected to step on has passcode. This
%modification has been done by Bilal Amjad.




    case 'tStepON'
%         tokID = tokenArrivedEarly('pWaiting', 1);
         currentLevel = global_info.Current_Level;
         %selecting the token whose arrival level is equal to current level
        tokID = tokenAnyColor('pWaiting', 1, ...
                      {['L',int2str(currentLevel)]});

        if not(tokID) % if there is no one waiting at this level, then do not
fire
            fire = 0; return
        end

       % Otherwise,check for vacancy in the car, if some people are waiting
at this level
        peopleInCar = sum(global_info.PASSENGER_DATA(:, 6));
        vacant = global_info.MAX_CAPACITY  - peopleInCar;
        if not(vacant)
            disp('** Caution! cannot take more passengers! Car is full');

            % reducing priority of "tStepON" to make it weak so that it will
not try again & again
            priorset('tStepON', 0);
            fillMissedCarTable();%holds the persons who could not enter the
car because of no vacancy
            fire = 0; return
        end
        %%%%%%%%%%%%%
        trans.selected_tokens = tokID;
        trans.override = 1;
         %using separateColors(the2colors) to get the person number to record
        %car entering time and to put travelling stauts=true, of that
        %particular person. Also to verify passcode holding status.
        the2colors = get_color('pWaiting', tokID);
        PNo = separateColors(the2colors);

        %Modification has been done here to allow step on to only those
        %passengers who have passcode

        %gets the passcode holding status of the passenger
        vPass = global_info.PASSENGER_DATA(PNo, 8) ;
        %if it's equal to 1 then transition fires
        if eq(vPass, 1)


        fire = 1;
```

```
                % the time passenger ENTERS car
        global_info.PASSENGER_DATA(PNo, 2) = current_time;
        % the passenger is inside the car: travelling
        global_info.PASSENGER_DATA(PNo, 6) = true;
        disp(['Passenger is Entering Car: Time:', num2str(current_time),...
              '  at level:', int2str(currentLevel),...
              '   Person No:', int2str(PNo)]);

        else
         fire = 0;
        end
        %%%%
```

```
         global_info.Current_Level =  global_info.Iterinary(1)  ;


        global_info.Starting_to_move = true; % It is used by "tMoving"
        fire = 1;
        end

%Author: Reggie Davidrajuh
%Adapted By: Bilal Amjad
%Purpose:It moves the car to the floor as per defined by the value of
global_info.Iterinary(1)

        case 'tMoving'
        if global_info.Starting_to_move
            global_info.Starting_to_move = false;
            fire = 1;
        else
            fire = 0;
        end
    %%%%
```

```
%     Author:Reggie Davidrajuh
%     Adapted By: Bilal Amjad
%       Purpose: To display a message on console after opening the elevator
door
    case 'tOpenDoor'
        disp(' ');
        disp(['**** Elevator Arrived at Level ',...
              int2str(global_info.Current_Level), ...
              '     at time: ', num2str(current_time), ' *******']);

   %Author: Bilal Amjad
   %Purpose: To make an itenirary for the car to follow by combining the
   % arrival floors of all passengers and destination floors of only those
   % passengers who have passcode
    case 'tArrival'
        %Contains arrival floors of all the passengers
         global_info.PickLevel  =  global_info.PASSENGER_DATA(:,  4)  ;  %  from
which level

%        get all the passengers from global_info.PASSENGER_DATA who have
%        VIP passcode
         global_info.vPassenger = find(global_info.PASSENGER_DATA(:, 8)==1);


        % to get drop level of all the passengers who have VIP passcode
         for i = 1:length(global_info.vPassenger)
         DropLevelList                                                      =
global_info.PASSENGER_DATA(global_info.vPassenger(i),5);

             global_info.b(i)=DropLevelList;
         end
          global_info.DropLevel= transpose(global_info.b);
          %makes up a final iterinary of elevator by appending both
         %global_info.PickLevel & global_info.DropLevel
         global_info.Iterinary            =            [global_info.PickLevel;
global_info.DropLevel ] ;


   %  Author: Bilal Amjad
   %  Purpose: For updating the itenirary list of the car by removing the 1st
   %      index   value   and   assigning   the   rest   of   the   values   to
global_info.Iterinary
   case 'tAccel'
       %getting the values of global_info.Iterinary
           Move= eval('global_info.Iterinary');
           %removing the 1st index value and assigning the rest of the values
to global_info.Iterinary
       eval(['global_info.Iterinary','=Move(2:end);']);

end % switch
```