Imperial College London
Department of Electrical Engineering

EE4-57: Discrete Event Systems Coursework

March 2020

Name: Rebecca Hallam
Degree: 4th Year EEE
Email: rch16@ic.ac.uk
CID: 01190898

Contents

1 Modelling the Map

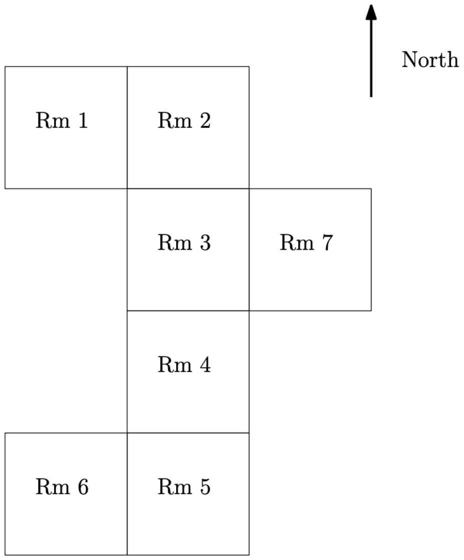

Figure 1: Map of Indoor Environment

Taking an indoor environment as seen in Figure 1, the spacial information it describes can be modelled using a finite determinstic automaton, G_M.

$$G_M = \{X, E, f, x_0, \Gamma, X_M\} \tag{1}$$

Where $X = \{r1, r2, r3, r4, r5, r6, r7\}$ is the state space, $E = \{n, s, e, w\}$ is the set of events, and the initial state is arbitrarily chosen as $x_0 = r1$. Furthermore, f represents the transition function, Γ the sent of enabled events and $X_M = \emptyset$ as no end states have been specified. The graphical representation of G_M can be seen in Figure 2.

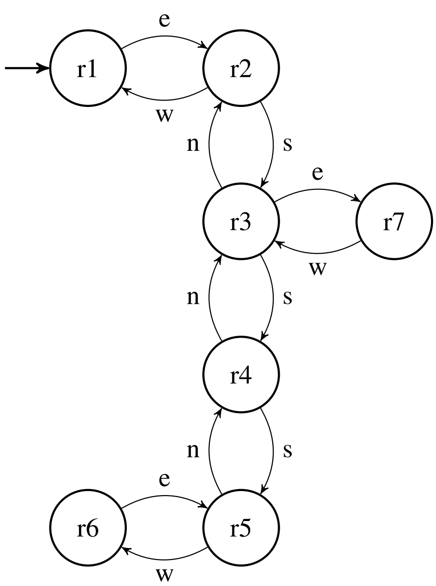

Figure 2: Graphical Representation of G_M

2 Modelling the Robot

If a robot is introduced that can traverse the environment displayed in Figure 1, a further finite deterministic automaton, G_R can be defined.

$$G_R = \{X, E, f, x_0, \Gamma, X_M\} \tag{2}$$

Here, the state space models the direction that the robot is facing, instead of the room, so $X = \{N, S, E, W\}$, and the event set $E = \{n, s, e, w, r\}$ now includes an event r that represents the robot rotating 90 degrees clockwise to face the next direction. As before, the initial state is arbitrarily chosen, this time to be $x_0 = N$. It is important to note that the robot can only move in the direction that it is facing. Therefore, the following transitions are defined:

Movement	**Rotation**
$f(N,n) = N$	$f(N,r) = S$
$f(S,s) = S$	$f(S,r) = E$
$f(E,e) = E$	$f(E,r) = W$
$f(W,w) = W$	$f(W,r) = N$

The graphical representation of G_R can be seen in Figure 3.

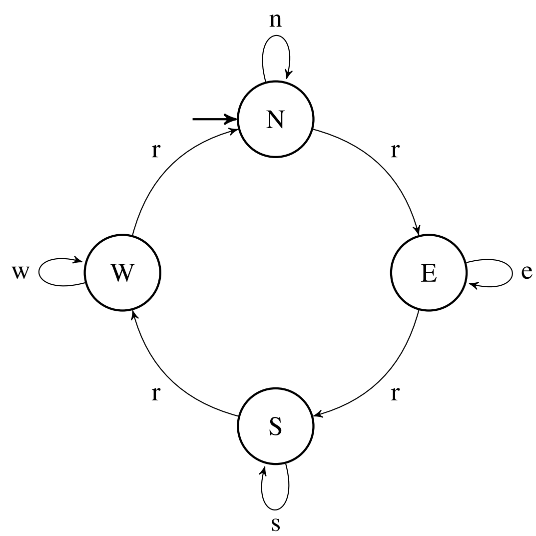

Figure 3: Graphical Representation of G_R

3 Modelling the Robot inside the Map

Computing the parallel composition of G_M and G_R creates a further finite determinstic automaton, $G_{M||R}$, that models the behaviour of the robot inside the map. This means that this automaton will keep track of the robot's location, as well as the direction it is facing.

This can be completed using Matlab by first defining the state space, initial state, event set and the transition map for each automaton G_M and G_R as covered in Questions 1 and 2. The transition map is a matrix where each row represents enabled transitions in the automaton - the first element is the initial state, the second element the new state, and the third element the event that causes this transition:

```
E_M = char('n','s','e','w'); % list of events
X_M = char('r1','r2','r3','r4','r5','r6','r7'); % list of states
T_M = [1,2,3; % list of transitions
       2,1,4;
       2,3,2;
       3,2,1;
       3,7,3;
       3,4,2;
       4,3,1;
       4,5,2;
       5,4,1;
       5,6,4;
       6,5,3;
       7,3,4];

x0_M = 'r1'; % initial state

G_M = {E_M,X_M,T_M,x0_M};
```

Listing 1: Initialising Automaton G_M

```
E_R = char('n','s','e','w','r'); % list of events
X_R = char('N','S','E','W'); % list of states
T_R = [1,1,1; % list of transitions (start, finish, event)
       1,3,5;
       2,2,2;
       2,4,5;
       3,3,3;
       3,2,5;
       4,4,4;
       4,1,5];
x0_R = 'N'; % initial state

G_R = {E_R,X_R,T_R,x0_R};
```

Listing 2: Initialising Automaton G_R

The parallel composition can then be computed in three stages;

1. **Calculate the new (combined) state space**
   This is equivalent to the number of states in G_M multiplied by the number of states in G_R, equal to $7 * 4 = 28$. Essentially, for every state in G_M, all 4 states of G_R occur. Therefore, the new state space becomes:

$$
\begin{aligned}
X_{M||R} = \{ & r1.N, r1.S, r1.E, r1.W, \\
& r2.N, r2.S, r2.E, r2.W, \\
& r3.N, r3.S, r3.E, r3.W, \\
& r4.N, r4.S, r4.E, r4.W, \\
& r5.N, r5.S, r5.E, r5.W, \\
& r6.N, r6.S, r6.E, r6.W, \\
& r7.N, r7.S, r7.E, r7.W \}
\end{aligned} \tag{3}
$$

2. **Compile the set of possible events**
   This is the union of the event sets E_M and E_R, which is, in this case, equivalent to E_R. Therefore:

$$E_{M||R} = \{n, s, e, w, r\} \tag{4}$$

3. **Form the transition map**
   To calculate this, for each state in $X_{M||R}$, one has to consider three types of events:

   (a) **Events private to $\mathbf{G_R}$**: Events enabled in G_R but not in G_M
   (b) **Events private to $\mathbf{G_M}$**: Events enabled in G_M but not in G_R
   (c) **Common events**: Events enabled in both G_R and G_M

   This is calculated by iterating over all states in $X_{M||R}$ and then, for each state, iterating over all events in $E_{M||R}$. For each event e, the code uses a lambda look up function to check whether e was enabled in the states from X_M and X_R that formed the new state in $X_{M||R}$ that is being considered. If enabled, the next states are found from the original transition maps T_M and T_R and combined to calculate the next state in $X_{M||R}$, with the current state, next state and event being added in numerical form as a row in the new transition map, $T_{M||R}$. If not enabled in G_M and/or G_R, the 'sub state' from X_M and/or X_R doesn't change. The new transition map formed in this way contains 40 transitions between the 28 states:

$$T_{R||M} = [\begin{array}{llll} 1,3,5 & 8,5,5 & 14,18,2 & 21,23,5 \\ 2,4,5 & 9,5,1 & 14,16,5 & 22,24,5 \\ 3,7,3 & 9,11,5 & 15,14,5 & 23,19,3 \\ 3,2,5 & 10,14,2 & 16,13,5 & 23,22,5 \\ 4,1,5 & 10,12,5 & 17,13,1 & 24,21,5 \\ 5,7,5 & 11,27,3 & 17,19,5 & 25,27,5 \\ 6,10,2 & 11,10,5 & 18,20,5 & 26,28,5 \\ 6,8,5 & 12,9,5 & 19,18,5 & 27,26,5 \\ 7,6,5 & 13,9,1 & 20,24,4 & 28,12,4 \\ 8,4,4 & 13,15,5 & 20,17,5 & 28,25,5] \end{array} \tag{5}$$

The code used to compute the parallel composition of G_M and G_R can be found in Appendix A. A graphical representation of the resulting automaton can be seen in Figure 4.

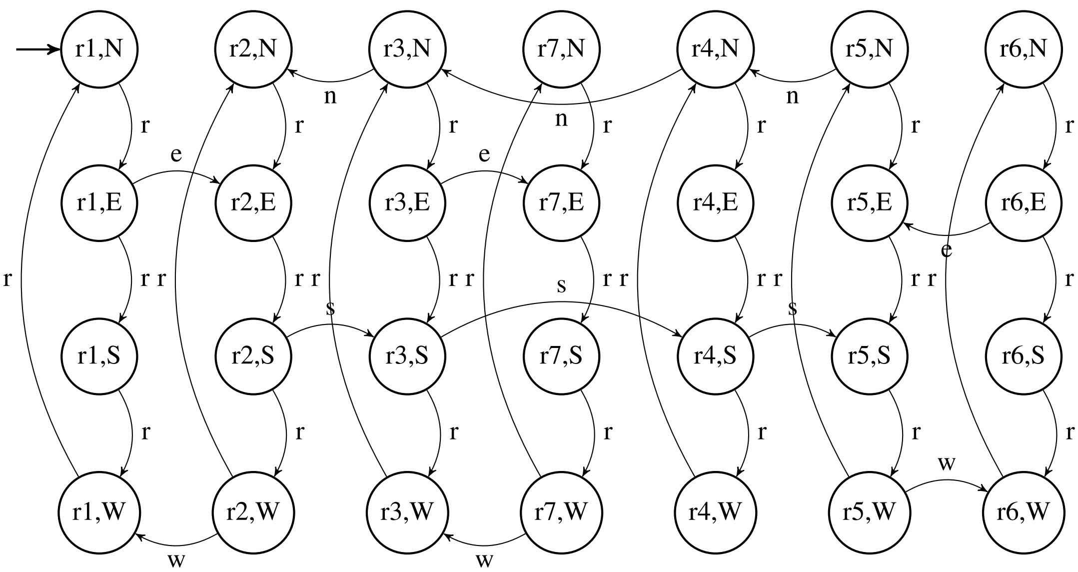

Figure 4: Graphical Representation of $G_{M||R}$

4 Modelling Partial Observability

By replacing events n, s, e, w in the automaton with a generic event, m, representing movement, the previously deterministic automaton $G_{M||R}$ can be converted into a potentially non-deterministic automaton G_N. This will, in turn, update the transition map to contain only 2 events instead of 5[1]. This new automaton can be defined as follows:

$$G_N = \{X, E, f, x_0, \Gamma, X_M\} \tag{6}$$

Where the state space, transition function, set of enabled events, set of marked states and the initial state remain unchanged from $G_{M||R}$. The new event set and transition map can be seen below.

$$E_N = \{m, r\} \tag{7}$$

$$T_{R||M} = \begin{array}{llll} [1,3,2 & 8,5,2 & 14,18,1 & 21,23,2 \\ 2,4,2 & 9,5,1 & 14,16,2 & 22,24,2 \\ 3,7,1 & 9,11,2 & 15,14,2 & 23,19,1 \\ 3,2,2 & 10,14,1 & 16,13,2 & 23,22,2 \\ 4,1,2 & 10,12,2 & 17,13,1 & 24,21,2 \\ 5,7,2 & 11,27,1 & 17,19,2 & 25,27,2 \\ 6,10,1 & 11,10,2 & 18,20,2 & 26,28,2 \\ 6,8,2 & 12,9,2 & 19,18,2 & 27,26,2 \\ 7,6,2 & 13,9,1 & 20,24,1 & 28,12,1 \\ 8,4,1 & 13,15,2 & 20,17,2 & 28,25,2] \end{array} \tag{8}$$

The graphical representation of G_N is intuitively not much different from that of $G_{M||R}$, apart from the n, s, e, w state labels. This new state diagram can be seen in Figure 5.

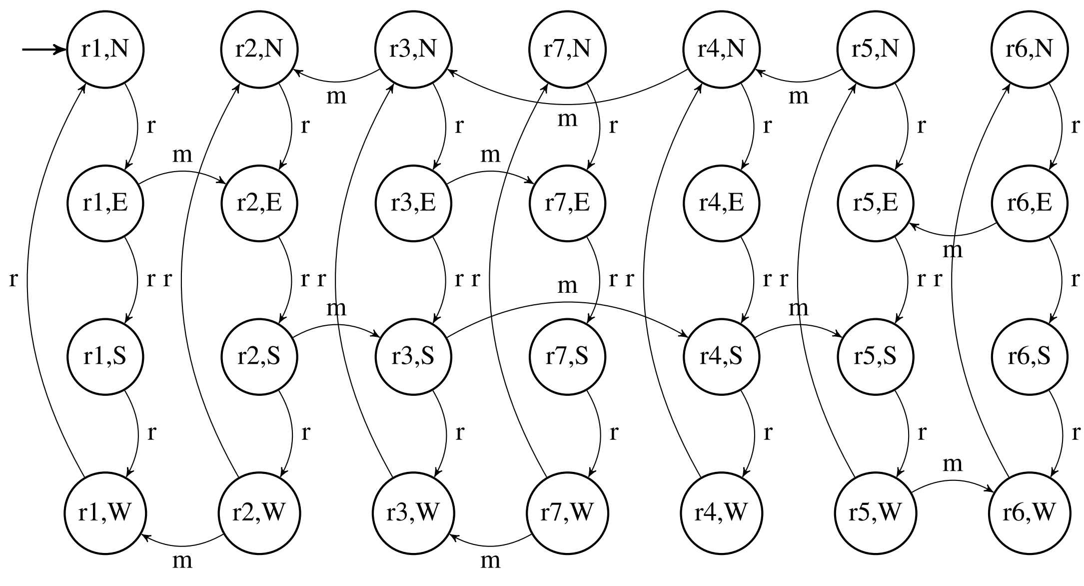

Figure 5: Graphical Representation of G_N

However, due to the fact that the robot can only move in the direction in which it is facing, if the current state is known and an m event occurs, the next state can be predicted. This means that G_N is still a deterministic automaton.

[1]i.e. events 1 2 3 and 4 become 'merged' into event 1, and hence event 5 becomes event 2.

5 Estimating Position and Heading

If the robot is unable to keep track/unaware of its current room location or direction it is facing, then it can try to predict this information by considering the sequence of events that has already occurred. This is achieved using an *Observer Automaton*, which can be constructed as per Algorithm 1:

initialization: let $X_{New} = \{\varepsilon R[x_0]\}$ be our first state. We also set $X_{Old} = \emptyset$;
while $X_{New}! = \emptyset$ **do**
  pick the first item of X_{New} as the current state, x_c;
  for $e \in E$ **do**
    $x' = \varepsilon R[f_{ND}(x_c, e)]$;
    if $x' \notin X_{New} \bigcup X_{Old}$ **then**
      add x' to X_{New};
      define $f_{Obs}(x_c, e) = x'$;
    else
  end
  move x_c from X_{New} to X_{Old}
end

Algorithm 1: Observer Automaton Algorithm

The idea of this algorithm is that it considers, for every possible state, what future states are possible following different sequences of events. It can be built in Matlab by using vectors of ones and zeros of length $N = 28$ (as there are 28 states in Automaton G_N). Here, position i in such a vector represents state i of G_N, and this position holding a value of 1 means that that state is possible. Intuitively, a value of 0 means that the state is not possible. The code works by starting with a vector of all 1s - this means that the robot could be in any state. Each event in E_N is then applied, and the possible transitions that could occur from the enabled states, as defined by T_N, are considered. For each event applied, a new vector is declared, where all elements are zero apart from at the locations of the next states that were enabled by the transitions considered. This new vector represents a new state in the observer automaton, that must then be explored in the same manner. Once no more new vectors are being created (only replicas of those already explored), these vectors form the Observer matrix, which represents the various states of the Observer Automaton. Algorithm 2 explains the implementation of Algorithm 1 in Matlab further. The code for Algorithm 2 can be found in Appendix B.

initialization: let X_{New} a vector of 1s of length $N = 28$, and $X_{Old} = \emptyset$;
while $X_{New}! = \emptyset$ **do**
  pick the first item of X_{New} as the current vector (state), x_c;
  remove x_c from X_{New};
  for $e \in E_N$ **do**
    declare a new vector of 0s of length N, x_{next};
    for $i \in x_c$ **do**
      if $x_c[i] = 1$ **then**
        that particular state in G_N is enabled;
        get the next state, j from T_N;
        assign $x_{next}[j] = 1$;
      end
    end
    if not already present, add x_c to X_{Old};
    if not already present, add x_{next} to X_{Next};
    add transition to observer transition map, T_obs;
  end
end

Algorithm 2: Method of Implementation of the Observer Automaton Algorithm in Matlab

Q1. How many states in the Observer Automaton?
The new automata generated from the observer automaton algorithm has 78 states.

Q2. List the states of the Observer Automaton
The full observer matrix and resulting transition matrix produced can be found in Appendices C and D respectively. However, for quick verification, the sums of the rows[2], the sums of the columns[3] and the singular values of the observer matrix

[2]Representing the number of times each state is possible.
[3]Representing the number of original states possible in each new state.

can be considered. These can be seen in Tables 1 to 3 respectively.

	28	12	4	12	2	4	4	12	0	2
↵	2	4	1	4	12	1	2	2	4	4
↵	1	1	4	4	1	2	2	2	4	1
↵	1	1	4	4	2	4	1	2	2	2
↵	2	1	1	1	1	1	4	2	1	2
↵	4	1	1	2	1	1	1	4	2	1
↵	2	4	1	1	1	4	2	1	2	1
↵	1	2	2	1	2	2	2	2		

Table 1: Sums of the Rows of the Observer Matrix

	5	5	5	5	9	9	9	9	13	13
↵	13	13	10	10	10	10	8	8	8	8
↵	5	5	5	5	7	7	7	7		

Table 2: Sums of the Columns of the Observer Matrix

	8.221943255	4.771344655	4.23097095	4.23097095	3.325961488
↵	3.022127163	3.022127163	2.926558881	2.899708741	2.567084977
↵	2.456193539	2.456193539	2.037546096	1.974355676	1.873572818
↵	1.735569528	1.735569528	1.630963915	1.630963915	1.60283163
↵	1.493025335	1.463230426	1.149402303	1.122720202	1.122720202
↵	1.090231244	1	1		

Table 3: Singular Values of the Observer Matrix, in Descending Order

Q3. Where is the robot located after the following sequence of events: m,r,r,m,r,m,r,r,m,r,m?
The sequence of events m,r,r,m,r,m,r,r,m,r,m when converted into 'event number form' is equivalent to 1,2,2,1,2,1,2,2,1,2,1. When applying this to the observer automaton just built, the resultant state is represented in the observer automatan as a vector of all zeros, apart from a 1 at index 5. This means that after this sequence of events, there is only one possible state that the robot could be in - state 5, or Room 2, heading North ('r2,N').

Q4. How can you find out using the Observer Automaton?
The result discussed in Q3 were calculated by considering the transition map that was generated when building the Observer Automaton. The states in this correspond to the indexes of the vectors found in the Observer Matrix, e.g. the vector of all 1s was the first vector considered, so is at index 1 of the observer matrix, and is therefore state 1 in the transition map.
As the initial state before the event sequence occurs is unknown, one must again start with the vector of all 1s that represents all states being possible. Looking this up in the transition matrix with the first event of the sequence can tell us the next state. Using this state as the current state, and looking for it in the transition matrix with event 2 results in the next state. This process of replacing the current state with the next state found is repeated until all the events in the sequence have been considered. The next state of this final transition map look up is the state that the robot ends up in after the sequence of events. The code used to compute this can be found in Appendix E.

6 Deterministic Interpretation

Looking at the Observer Matrix that represents the Observer Automaton built, it can be seen that multiple vectors exist that contain 27 zeros and only a single one. These represent states where only one of the states of the original automaton are possible. It can therefore be concluded that, providing the input event sequence is long enough, it *is* possible for the robot to exactly reconstruct its position and heading and therefore the system is observable.

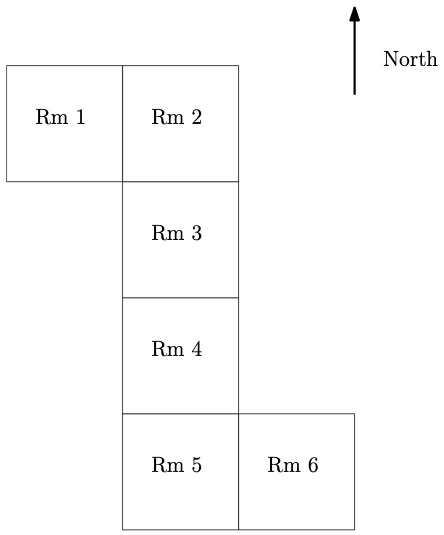

Figure 6: Modified Map of Indoor Environment

However, if the indoor environment was modified such that it looked like the map in Figure 6, the original automaton modelling the map, G_M, the parallel composition of G_M and G_R, and the observer automaton will change:

$$G_{M_2} = \{X, E, f, x_0, \Gamma, X_M\} \tag{9}$$

Where $X = \{r1, r2, r3, r4, r5, r6\}$ now has one less state. Hence, the graphical representation becomes:

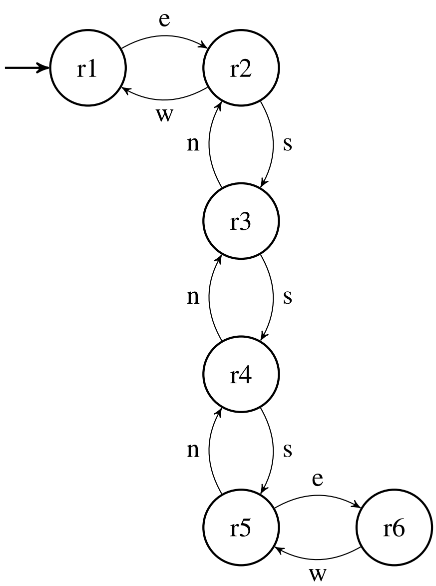

Figure 7: Graphical Representation of G_{M_2}

Furthermore, the states resulting from parallel composition become:

$$
\begin{aligned}
X_{M_2||R} = \{ & r1.N, r1.S, r1.E, r1.W, \\
& r2.N, r2.S, r2.E, r2.W, \\
& r3.N, r3.S, r3.E, r3.W, \\
& r4.N, r4.S, r4.E, r4.W, \\
& r5.N, r5.S, r5.E, r5.W, \\
& r6.N, r6.S, r6.E, r6.W \}
\end{aligned} \tag{10}
$$

With transition map as seen below, and graphical representation as seen in Figure 8.

$$
\begin{aligned}
T_{R||M_2} = [& 1,3,5 && 8,5,5 && 14,18,2 && 21,23,5 \\
& 2,4,5 && 9,5,1 && 14,16,5 && 22,24,5 \\
& 3,7,3 && 9,11,5 && 15,14,5 && 23,22,5 \\
& 3,2,5 && 10,14,2 && 16,13,5 && 24,20,4 \\
& 4,1,5 && 10,12,5 && 17,13,1 && 24,21,5 \\
& 5,7,5 && 11,27,3 && 17,19,5 && 25,27,5 \\
& 6,10,2 && 11,10,5 && 18,20,5 && 26,28,5 \\
& 6,8,5 && 12,9,5 && 19,23,3 && 27,26,5 \\
& 7,6,5 && 13,9,1 && 19,18,5 && 28,25,5] \\
& 8,4,4 && 13,15,5 && 20,17,5 &&
\end{aligned} \tag{11}
$$

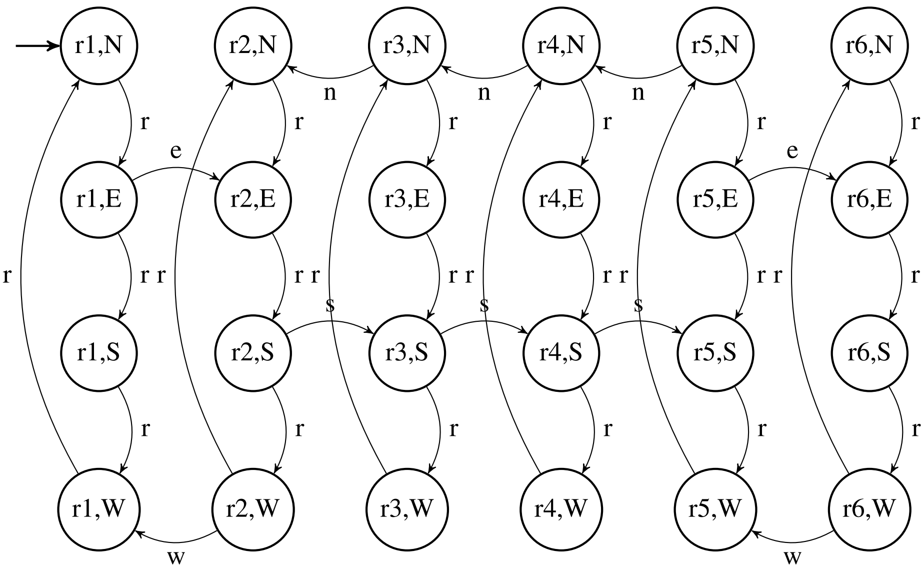

Figure 8: Graphical Representation of $G_{M_2||R}$

Converting this to a partially observable automaton, as per Question 4 produces a graphical representation similar to that seen in Figure 8, but with m arcs replacing the n, s, e, w ones. This is then used to compute the new observer automaton, whose resulting observer matrix can be seen in Appendix F. Inpsection of this when compared to the original observer matrix (as seen in Appendix C) highlights two key differences - first, that the new observer automaton has only 40 states, compared to 78 in the original one. Secondly, only rows, and hence states, 11, 18, 27 and 34 of this new automaton have a single 1. This means that there are only four states in this obsserver automaton where the current robot heading and direction is known for sure. Therefore, the robot will very rarely be able to exactly reconstruct its position and heading. This is intuitive when considering the rotational symmetry of both maps - the original map (Figure 1) has no rotational symmetry, meaning that when it is rotated, no position of rotation will look the same as any other. However, in contrast, the new map (Figure 6) has rotational symmetry of order 2. This means that rotating it by 180 degrees will cause the shape of the map to look the same as when it is in the original position. Therefore, it can be concluded that the system defined by the modified map is not observable.

Appendices

A Question 3: Parallel Composition

```
function [E_AB,X_AB,T_AB] = parallelComposition(E_A,X_A,T_A,E_B,X_B,T_B)
%PARALLELCOMPOSITION Compute the parallel composition of Automatons A and B

a = size(X_A,1);
b = size(X_B,1);

% ---------- Create a new set of events ----------
% generate set of events common to A and B
commonEvents = intersect(E_A,E_B,'stable');          % string form
numCommon = size(commonEvents,1);
common = [];                                          % number form
for n=1:numCommon
    common = [common,n];
end
% generate union of events from A and B
E_AB = union(E_A,E_B,'stable');

% -------------- New state space -------------
% generate arrays to iterate over
[gridA,gridB] = meshgrid(1:a,1:b);
X = [gridA(:) gridB(:)];                           % states in numbered form
X_AB = strcat(X_A(X(:,1),:),",",X_B(X(:,2),:)); % states in string form

% ------------- New Transition Matrix -----------
% T = (start, end, event) -> mapping event transitions
% initialise new transition matrix
T_AB = [];
% define lambda look up functions that find the next state given current state and event
fA = @(state,event) T_A((T_A(:,1)==state & T_A(:,3)==event),2);
fB = @(state,event) T_B((T_B(:,1)==state & T_B(:,3)==event),2);
% iterate over states in X_AB
for i = 1:size(X,1)
    currentState = i;
    state = X(i,:);    % current state of combined automata
    stateA = state(1); % current state from automata A
    stateB = state(2); % current state from automata B

    % for T_A and T_B, get indexes of transitions that occur in the current state
    idxA = find(T_A(:,1)==stateA);
    idxB = find(T_B(:,1)==stateB);

    % using these indices, get the enabled transitions
    enabledA = T_A(idxA,:);
    enabledB = T_B(idxB,:);

    % transition = (start, end, event) therefore get enabled events from 3rd element in array
    gammaA = enabledA(:,3);
    gammaB = enabledB(:,3);

    % ------ CASE 1: transitions enabled in A and B simultaneously -----
    commonEnabled = intersect(gammaA,gammaB);
    % check that this is not empty
```

```
    if(numel(commonEnabled)~=0)
        % iterate over commonEnabled
        for j = 1:numel(commonEnabled)
            event = commonEnabled(j);
            nextStateA = fA(stateA,event);
            nextStateB = fB(stateB,event);
            % form combined state
            nextState = find(X(:,1)==nextStateA & X(:,2)==nextStateB);
            % add to transition matrix
            transition = [currentState,nextState,event];
            T_AB = [T_AB;transition];
        end
    end

    % ----- CASE 2: transitions enabled in A but not B -> private to A -----
    privateA = setdiff(gammaA,common);
    % check that this is not empty
    if(numel(privateA)~=0)
        % iterate over privateA
        for j = 1:numel(privateA)
            event = privateA(j);
            nextStateA = fA(stateA,event);
            nextStateB = stateB; % doesn't change
            % form combined state
            nextState = find(X(:,1)==nextStateA & X(:,2)==nextStateB);
            % add to transition matrix
            transition = [currentState,nextState,event];
            T_AB = [T_AB;transition];
        end
    end


    % ----- CASE 3: transitions enabled in B but not A -> private to B -----
    privateB = setdiff(gammaB,common);
    % check that this is not empty
    if(numel(privateB)~=0)
        % iterate over privateB
        for j = 1:numel(privateB)
            event = privateB(j);
            nextStateA = stateA; % doesn't change
            nextStateB = fB(stateB,event);
            % form combined state
            nextState = find(X(:,1)==nextStateA & X(:,2)==nextStateB);
            % add to transition matrix
            transition = [currentState,nextState,event];
            T_AB = [T_AB;transition];
        end
    end
end
end
```

Listing 3: Function for Calculation of Parallel Composition of Two Automata

B Question 5: Observer Automaton

```
function [X_obs,T_obs] = observerAutomaton(X_N,T_N,E_N)
%OBSERVERAUTOMATON Construct an observer automaton for a non deterministic
%finite state automaton

% number of states
numStates = size(X_N,1);

% number of events
numEvents = size(E_N,1);

% initialise new transition map
T_obs = zeros(1,3);

% start with a vector of 1s of size N in X_new
% represents that the automata could be in any state
X_new = ones(1,numStates);

% initialise X_old -> list of lists that have already been considered
X_old = ones(1,numStates);

% define lambda look up function that finds the next state given current state
    and event
fN = @(state,event) T_N((T_N(:,1)==state & T_N(:,3)==event),2);

% while X_new isn't empty
while size(X_new,1)>0
    % take first item in X_new as current list
    currentList = X_new(1,:);
    % remove currentList (first item) from X_new
    X_new(1,:) = [];
    % apply all events to current list
    for event=1:numEvents
        % for each event, apply results to a new vector
        % -> initialise new list with results of applying events to current list
        % -> represents a new state to be explored
        newList = zeros(1,numStates);

        % iterate over array i.e. each state
        for state=1:numel(currentList)

            % find if transition is enabled for current state and event
            nextState = fN(state,event);

            % multiply by value of currentList(state) to ensure transition is
                enabled for currentList
            nextState = nextState*currentList(state);

            % if nextState is not empty, transition is enabled for current state
            if(numel(nextState)~=0 & nextState~=0)
                % add a 1 at that location in the new array
                newList(nextState) = 1;
            end

        end

        % check if current list is in X_old
        [~,currentPresent] = ismember(X_old,currentList,'rows');

```

```
        % index is current state for the transition map
        currentState = find(currentPresent,1,'first');

        % check if new list is in X_old
        [~,newPresent] = ismember(X_old,newList,'rows');

        % find the index of where the new list is present
        newIndex = find(newPresent,1,'first');

        if isempty(newIndex)
            % newList is not present in X_old
            % add to X_new to mark as needing exploring
            X_new = [X_new;newList];
            % add to X_old to reserve an index for reference in T_obs
            X_old = [X_old;newList];
            % get index as next state for transition map
            [~,newState]=ismember(newList,X_old,'rows');
        else
            % newList is already in X_old
            % get index as next state for transition map
            newState = newIndex;
        end

        % update new transition map with this transition
        transition = [currentState,newState,event];
        T_obs = [T_obs;transition];

    end

end

X_obs = X_old;

%remove zeros from first row of T_obs (used in Initialisation)
numTransitions = size(T_obs,1);
T_obs = T_obs(2:numTransitions,:);
end
```

Listing 4: Function for Calculation of an Observer Automaton

C Question 5: Observer Matrix

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	0	0	0	1	1	0	1	0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	0	0	1	0
3	0	0	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0
4	1	0	0	0	0	1	1	0	1	0	1	1	0	0	1	1	0	1	0	1	1	0	0	0	0	1	0	0
5	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0
7	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0
8	0	0	1	0	0	1	0	1	1	1	1	0	1	1	0	0	1	0	0	1	0	0	1	0	0	0	0	1
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0
12	0	0	0	0	0	1	0	0	0	1	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0
15	0	1	0	0	1	0	0	1	0	1	1	1	0	0	1	1	1	0	1	0	0	1	0	0	1	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
17	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0
19	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1
24	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1
29	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
30	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

32	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
33	0	0	0	0	1	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
34	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0
35	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
36	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0
37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
38	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
39	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
40	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
41	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0
42	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
43	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
44	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
45	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
46	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
47	0	0	0	0	0	0	1	0	1	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
48	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
49	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
50	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
51	0	0	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
52	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
53	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
54	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
55	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
56	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
57	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
58	0	0	0	0	0	1	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
59	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
60	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
61	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
62	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0
63	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
64	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

65	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
66	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
67	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
68	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
69	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
70	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
71	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
72	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
73	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
74	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
75	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
76	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
77	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
78	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

D Question 5: Observer Transition Map

1	1	2	1
2	1	1	2
3	2	3	1
4	2	4	2
5	3	5	1
6	3	6	2
7	4	7	1
8	4	8	2
9	5	9	1
10	5	10	2
11	6	11	1
12	6	12	2
13	7	13	1
14	7	14	2
15	8	2	1
16	8	15	2
17	9	9	1

18	9	9	2
19	10	16	1
20	10	17	2
21	11	9	1
22	11	18	2
23	12	19	1
24	12	20	2
25	13	21	1
26	13	22	2
27	14	9	1
28	14	23	2
29	15	24	1
30	15	2	2
31	16	9	1
32	16	25	2
33	17	26	1
34	17	27	2
35	18	9	1
36	18	28	2
37	19	3	1
38	19	29	2
39	20	30	1
40	20	3	2
41	21	9	1
42	21	31	2
43	22	9	1
44	22	32	2
45	23	33	1
46	23	34	2
47	24	35	1
48	24	36	2
49	25	9	1
50	25	37	2

51	26	38	1
52	26	39	2
53	27	30	1
54	27	5	2
55	28	40	1
56	28	41	2
57	29	42	1
58	29	19	2
59	30	9	1
60	30	43	2
61	31	16	1
62	31	44	2
63	32	45	1
64	32	46	2
65	33	13	1
66	33	47	2
67	34	48	1
68	34	7	2
69	35	49	1
70	35	50	2
71	36	9	1
72	36	51	2
73	37	52	1
74	37	53	2
75	38	5	1
76	38	54	2
77	39	9	1
78	39	38	2
79	40	9	1
80	40	35	2
81	41	9	1
82	41	11	2
83	42	9	1

84	42	55	2
85	43	9	1
86	43	56	2
87	44	32	1
88	44	52	2
89	45	49	1
90	45	57	2
91	46	9	1
92	46	13	2
93	47	49	1
94	47	58	2
95	48	9	1
96	48	59	2
97	49	9	1
98	49	60	2
99	50	11	1
100	50	61	2
101	51	47	1
102	51	62	2
103	52	9	1
104	52	21	2
105	53	9	1
106	53	16	2
107	54	42	1
108	54	26	2
109	55	9	1
110	55	63	2
111	56	60	1
112	56	64	2
113	57	42	1
114	57	65	2
115	58	7	1
116	58	66	2

117	59	9	1
118	59	67	2
119	60	9	1
120	60	68	2
121	61	69	1
122	61	40	2
123	62	9	1
124	62	24	2
125	63	70	1
126	63	71	2
127	64	9	1
128	64	30	2
129	65	13	1
130	65	70	2
131	66	24	1
132	66	33	2
133	67	72	1
134	67	73	2
135	68	65	1
136	68	74	2
137	69	35	1
138	69	75	2
139	70	9	1
140	70	45	2
141	71	9	1
142	71	42	2
143	72	9	1
144	72	76	2
145	73	9	1
146	73	48	2
147	74	30	1
148	74	49	2
149	75	9	1

150	75	69	2
151	76	77	1
152	76	78	2
153	77	13	1
154	77	72	2
155	78	48	1
156	78	77	2

E Question 5: Finding the State after a Sequence of Events

```
function [stateIndex] = calculateState(X_obs,T_obs,eventSequence)
%CALCULATESTATE calculate resultant state from observer automaton given sequence
    of events

% start with vector of all 1s as the robot could be in any state
initialVec = ones(1,28);

% find this vector in X_obs
[~,initialState] = ismember(initialVec,X_obs,'rows');

% start with this initial state as the current state
currentState = initialState;

% define a lambda look up function that finds the next state given current state
    and event
fN = @(state,event) T_obs((T_obs(:,1)==state & T_obs(:,3)==event),2);

% iterate through event sequence
for e = 1:numel(eventSequence)
    % each event causes a transition
    event = eventSequence(e);
    % for current state and event, find next state
    nextState = fN(currentState,event);
    % assign next state as current state and look for next event
    currentState = nextState;
end

stateIndex = currentState;

end
```

Listing 5: Function for Calculation of the State of the Automaton after a Sequence of Events

F Question 6: Observer Matrix of Modified Map

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	0	0	0	1	1	0	1	0	1	1	0	0	1	1	0	0	0	1	0	1	0	0	1	0	0	0	1	0
3	0	0	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0
4	1	0	0	0	0	1	1	0	0	0	1	1	0	0	1	1	1	0	0	1	0	1	0	0	0	1	0	0
5	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
8	0	0	1	0	0	1	0	1	1	1	0	0	1	1	0	0	1	0	1	0	0	0	0	1	0	0	0	1
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
12	0	0	0	0	0	1	0	0	0	1	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0
15	0	0	0	1	1	0	1	0	1	1	0	0	1	1	0	0	0	1	0	1	0	0	1	0	0	0	0	0
16	0	1	0	0	1	0	0	1	0	0	1	1	0	0	1	1	0	1	1	0	1	0	0	0	1	0	0	0
17	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
19	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
23	1	0	0	0	0	1	1	0	0	0	1	1	0	0	1	1	1	0	0	1	0	1	0	0	0	0	0	0
24	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0
25	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
28	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
29	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0
31	0	0	1	0	0	1	0	1	1	1	0	0	1	1	0	0	1	0	1	0	0	0	0	1	0	0	0	0

32	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0
33	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
35	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
36	0	1	0	0	1	0	0	1	0	0	1	1	0	0	1	1	0	1	1	0	1	0	0	0	0	0	0	0
37	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1
38	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
39	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0
40	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

G Complete Code

The complete code written to complete this coursework can be found at `https://github.com/rch16/DiscreteEventSystems/`.