The purpose of this first excercise is to apply the finite state modeling technique to a idealized but "reasonable" elevator.

Part 1:

The elevator simulation could have been written in any number of languages, Matlab was choosen. The simulation demo involves running the Matlab script "hw1_study.m". The state of the simulation is printed (display_state.m) at each step. It exercises the elevator in a few special cases which illustrate specific conditions; then in a high load condition for 20 steps followed by a low load condition for an additional 20 steps. The display of the elevator state is as follows:

```
<name of the test> : id(<step in the text>)
Heading     : <heading indicator>
Floor       : 0 1 2 3 4 5
Cage Loc    : <current floor indicator>
InCall      : <internal call indicator>
ExCall UP   : <external up call indicator>
ExCall DOWN : <external down call indicator>
```

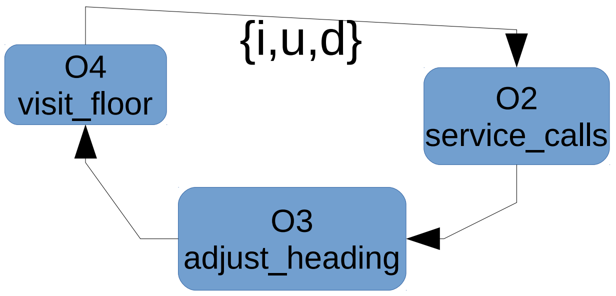
Example: Here the cage is at floor 3, an there are unserviced calls to floors 1 and 5 orginating inside the cage. There are externally originated up-calls at floors 0, 1 and 4 and down-calls at floors 2 and 4. The up-call on floor 2 was just serviced. This differs from that in the assignment inasmuch as the orientation is rotated and the immediately serviced requests are indicated with a 'O'.

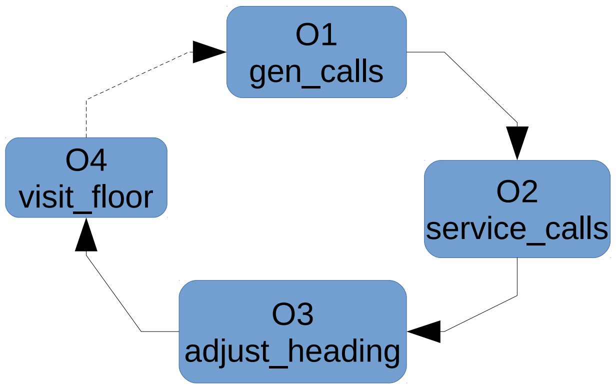
```
Trial A : id(5)
Heading     : > > > > > >
Floor       : 0 1 2 3 4 5
Cage Loc    : _._._.X._._
InCall      : _.X._._._.X
ExCall UP   : X.X.O._.X._
ExCall DOWN : _._.X._.X._
```

The approach manages the number of states by introducing state variables. An alternate approach would be to construct multiple state machines and consolidate. The driver script (hw1_study.m) trivially implements the demo automata. The demo automata differs from the target automata by introducing an additional state which replaces the input transition with a state setting the updated call state (gen_state.m) and an output indicating the current state (display_state.m) the remaining automata is implemented by a single function (handle_events.m). The servicing of calls involves no more than one floor change .

The is decomposed into a finite state machine where sets of states and their transitions are consolidated. In order to prevent confusion the input values for the state variables are not changed, and their output values are only dependent upon the input values.

Part 2:





The model is a Moore style model where the outputs are implemented as the nodes are entered and on therefore indicated on the nodes. Generally the transitions are true and instantaneous. The two versions indicate the elevator in operation with a single input gate and a demo where the gen_calls function imitates the input gate. In the code the gen_calls function is called for each of the call types, but it can be considered a single aggregate call...

```
gen_calls: (calls) -> calls_next
```

The "gen_calls" function indicates the new calls introduced via button pushes. Any of the calls state-variables can be "set" are allowed but none "reset".

```
States:
```

The state set used here takes the following form (similar to the output ordering above):

```
Q  =  H  x  F
   x  (  I0  x  I1  x  I2  x  I3  x  I4  x  I5  )
   x  (  U0  x  U1  x  U2  x  U3  x  U4  )
   x        (  D1  x  D2  x  D3  x  D4  x  D5  )
```

where H = {-1 0 1} = {down rest up} is the cage heading set, F = {0..5} is the cage floor location, and Ii, Ui, Di = {0 1} = {set reset} are button setting for cage-interior, cage-exterior-up and cage-exterior down, respectively. The current state variables corresponding to these state sets are h and f. The buttons are collected into current-call-vectors, i, u and d, where, for example, the current-up-call-vector is u = (u0..u4). This describes all the potential states.

```
Initial_state: (f,h,i,u,d) = (0, 0, (0 0 0 0 0 0), (0 0 0 0 0), (0 0 0 0 0))

Inputs: ((i,u,d) -> {set reset}
```

As indicated in the States, the input can be any combination of button pushes over the call variables where {set reset} are the allowed values.

`Outputs:`

visit_floor causes the elevator to move to the new floor and service_calls turns specific button off. The outputs are generated as the states are entered.

Part 3:

As the simulation files match the nodes in the automaton model, the details of the automaton model are placed in-line with the source code as comments. The automaton model is helpful in directing the specific functions which need to be constructed as part of the simulation.

Part 4:

T1 : Every call is eventually serviced.

Once the heading is set the cage advances along that heading until all calls on that heading are serviced. If there are any calls then the heading will be set toward at least one call. If there are no calls on the current heading a productive heading will be selected. Therefore progress in servicing headings will be made. Livelock is prevented by continuing on a heading until all calls on that heading have been serviced. If there were more than two productive headings were possible additional work would be necessary to ensure fairness but as there are but two, no additional mechanisms are necessary.

T2: Once an external call is serviced, the passenger will travel directly to their destination presuming they promptly request a floor in the direction which was indicated externally.

An external call is serviced only when the heading matches the external-call direction. The heading does not change until all calls on that heading are serviced. Therefore the passenger will be delivered directly to their interior-call.

T3: Every internal call will be serviced in less than 2* | |F_max| - |F_min| | steps

It is possible that a passenger may generate an internal call that does not lie in the direction that they indicated on their external call. In the worst case the heading may be wrong and the elevator will need to make a complete traversal of each floor on the wrong heading. After that the heading will be corrected and every floor visited again. Therefore every floor could be visited no more than twice.