Names:
Akshay Swaminathan
Sarah Person
Jane Lockshin
Stacia Near

Project 1: Regular Languages and Pursuit-Evasion
CSCI 561
September 23, 2018

Pursuit-evasion games are scenarios with multiple agents where one agent attempts to avoid capture by another. Consider a variation of pursuit-evasion games as follows:
Two agents share a grid environment: a human (evader) and wumpus (pursuer).
The human and wumpus alternate moves on the grid. The wumpus moves each turn up, down, left or right. The human can move up, down, left, right, or remain in place.
If the wumpus and human ever occupy the same grid cell, the wumpus eats the human.
If the human reaches a designed grid cell, they escape.
Answer the following questions using your implementation of nite automata operations for support.

1. For the map in Figure 1, construct a discrete event system model. Assume that the human's movements are controllable and that the wumpus's movements are not controllable.

I wrote a program to generate the below graph, which is a graph of all possible states in the map provided. Each node represents a location for both the human and the wumpus. This is because the only states that can change in the discrete event simulation is the coordinates of the human, and the coordinates of the wumpus.

Therefore, a node that is named H00-W02 means that node represents the state where the wumpus is in cell (0,0) and the human is in cell (0,2).

We assume for all of these graphs that the human always moves first, and then turns alternate for the rest of the game.

The edges of this DES are the different possible state transitions for the model. These involve human and wumpus movements, and the wumpus eating the human, or the human escaping. All possible edges are listed below:

All edges:
human_up

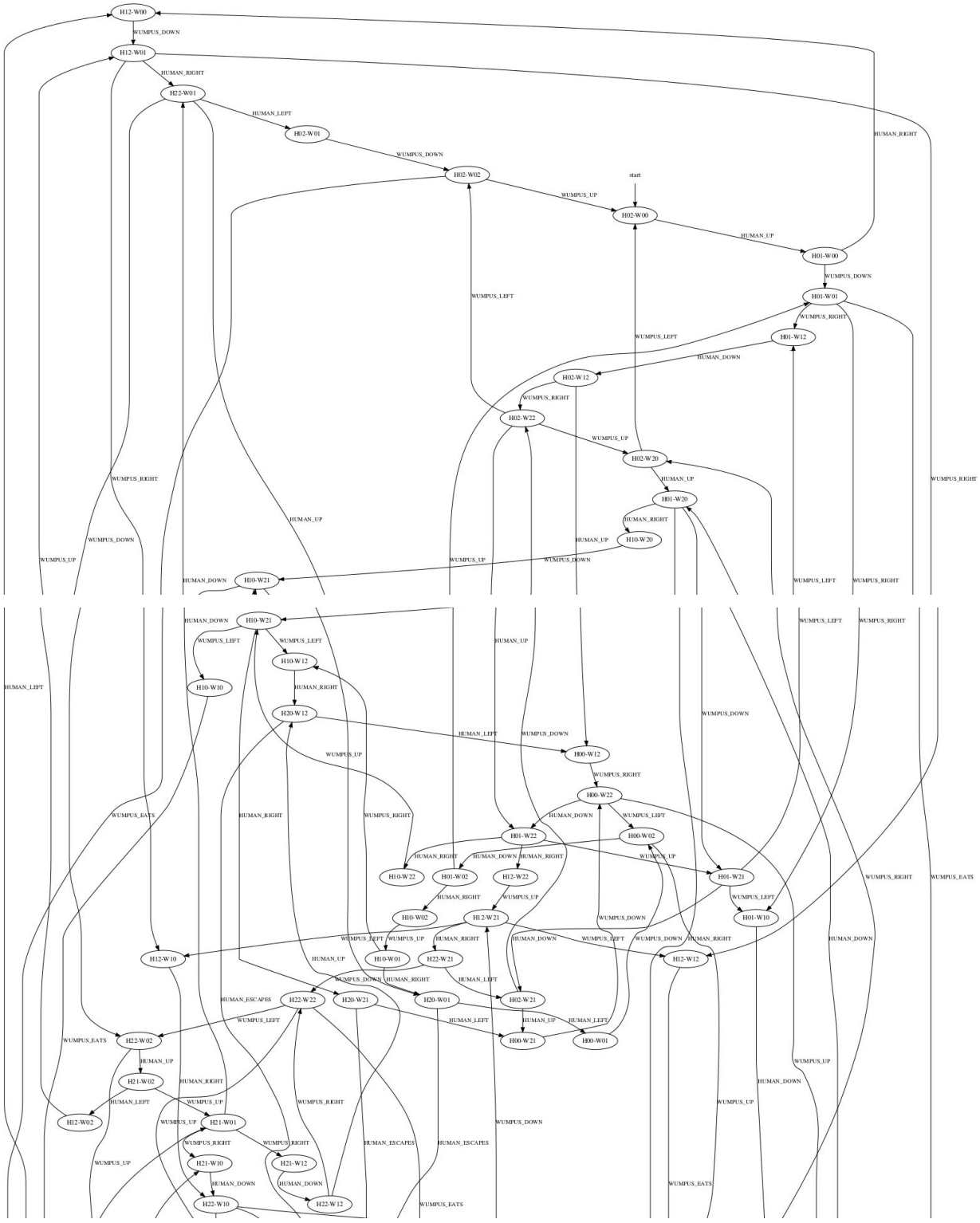
wumpus_down
wumpus_eats
wumpus_right
human_down
wumpus_up
wumpus_left
human_right
human_escapes
human_left

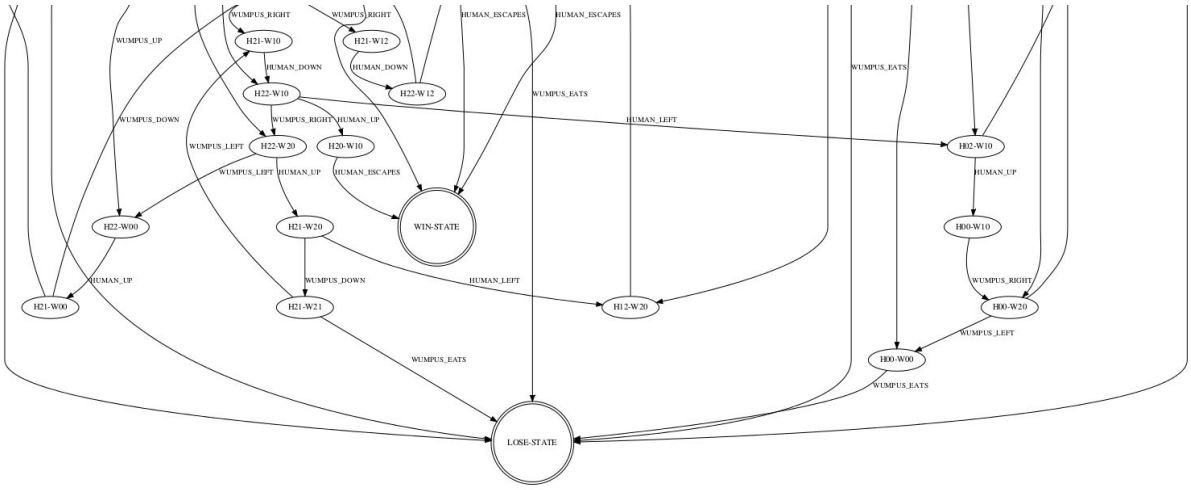
The start and accept states are listed below. The start state is where the wumpus and human are on the map when the game starts, in this case coordinates (0,2) for human and (0,0) for wumpus.

There are only two accept states, because there are only two ways the game can end. Either the wumpus eats the human and we go to the lose state, or the human escapes and we go to the win state. Both are accept states.

Start state: h02-w00
Accept states: lose-state win-state

The graph was too big to fit in one screenshot, so here is what our DES looks like in multiple screenshots:


H12-W00
WUMPUS_DOWN
H12-W01
HUMAN_RIGHT
H22-W01
HUMAN_LEFT
H02-W01
WUMPUS_DOWN
H02-W02
WUMPUS_UP
start
H02-W00
HUMAN_UP
H01-W00
WUMPUS_DOWN
H01-W01
WUMPUS_RIGHT
H01-W12
HUMAN_DOWN
H02-W12
WUMPUS_RIGHT
H02-W22
WUMPUS_UP
H02-W20
HUMAN_UP
H01-W20
HUMAN_RIGHT
H10-W20
HUMAN_UP
H10-W21
HUMAN_DOWN
H10-W10
H10-W12
H20-W12
H00-W12
H00-W22
H01-W22
H00-W02
H10-W22
H01-W02
H12-W22
H01-W21
H10-W02
H12-W21
H01-W10
H12-W10
H10-W01
H22-W21
H12-W12
H20-W22
H20-W21
H20-W01
H02-W21
H22-W02
H00-W21
H00-W01
H21-W02
H12-W02
H21-W01
H21-W10
H21-W12
H22-W10
H22-W12
WUMPUS_LEFT
WUMPUS_RIGHT
HUMAN_LEFT
HUMAN_RIGHT
HUMAN_UP
HUMAN_DOWN
WUMPUS_UP
WUMPUS_DOWN
WUMPUS_EATS
HUMAN_ESCAPES



2. For your DES model of Figure 1, construct a specification for the human to avoid the wumpus and escape.

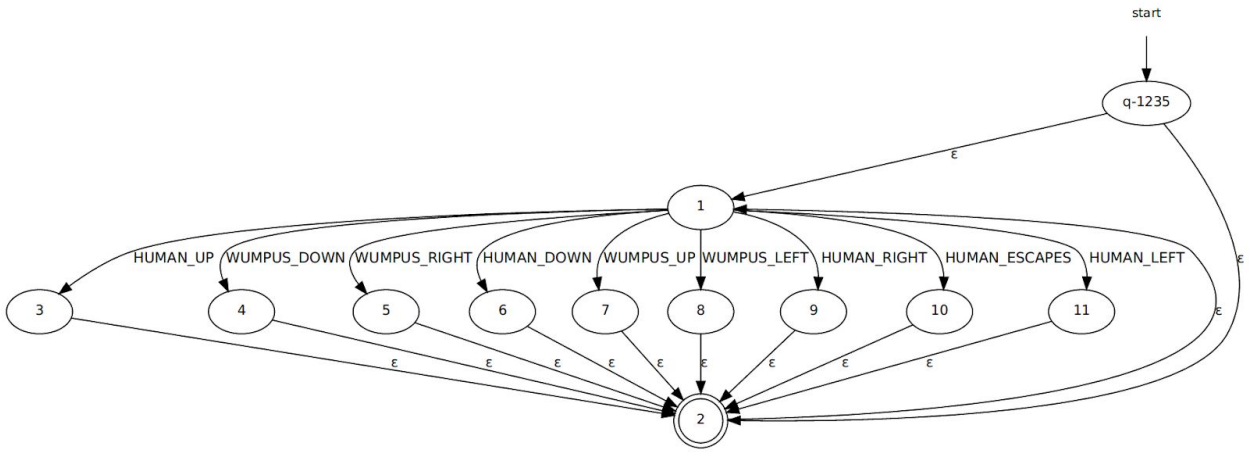
We will use a regular expression to specify how the human can avoid the wumpus and escape. To make this regular expression, we simply take the kleene closure of the union of all the possible edges, EXCEPT for the “wumpus_eats” edge. Because we do not want to take that route, so the human can avoid the wumpus and escape.

Here is the regular expression:

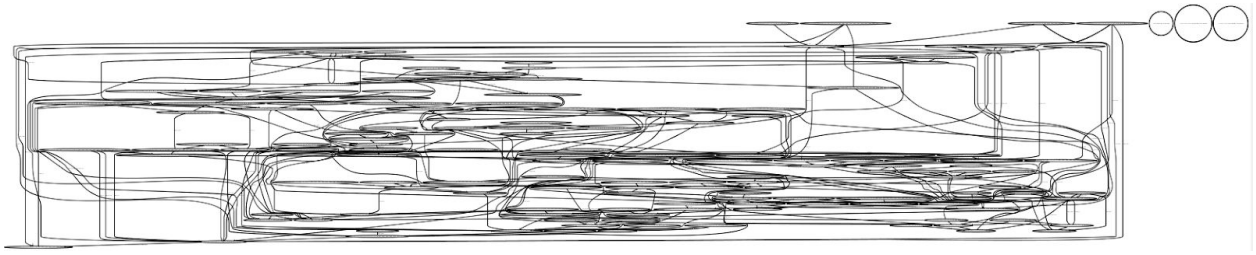
(human_up | wumpus_down | wumpus_right | human_down | wumpus_up | wumpus_left | human_right | human_escapes | human_left)*

(a) Can the human avoid being eaten? Prove yes or no via automata operations.
(b) Can the human escape in nite time (xed number of steps)? Prove yes or no.

Here is the DFA generated from the above Regex.



Here is the intersection of this DFA with the above DES:



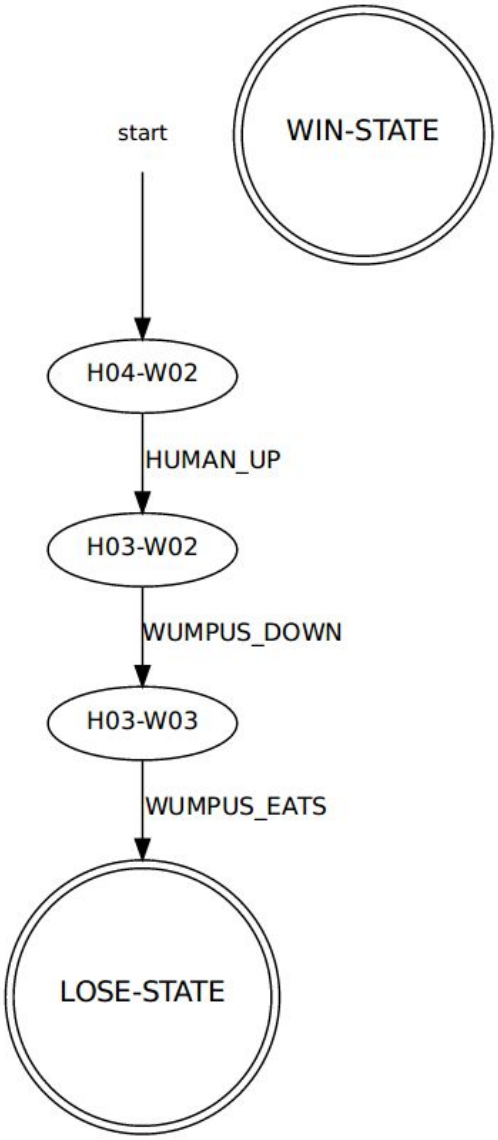
Upon intersecting this DFA with the above DFA derived from the discrete event simulation, there are routes to get to an accept state without becoming stuck in a non-accept-state. Therefore, the human can avoid being eaten, if they make the right choices. They also can escape in a finite number of steps, because there are routes that will solve the decision without getting stuck in an infinite loop.

3. Design a map where the wumpus can always eat the human and prove via a DES model that this is the case.

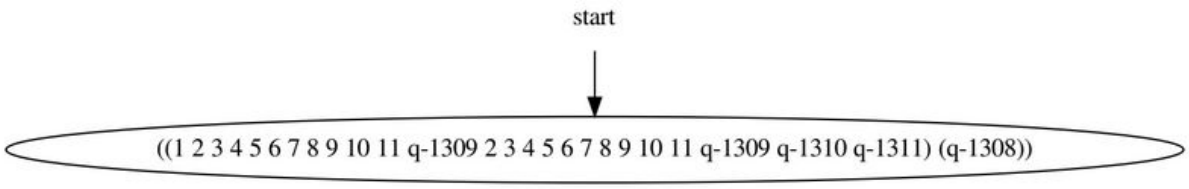
The map looks like this:

Escape	Obstacle	Wumpus		Human

The DES model:



Intersection of DES with Regex specifying the desired accept state:



As you can see, there is only one possible route, and that is directly to the lose state, which is why the intersection is so simple.

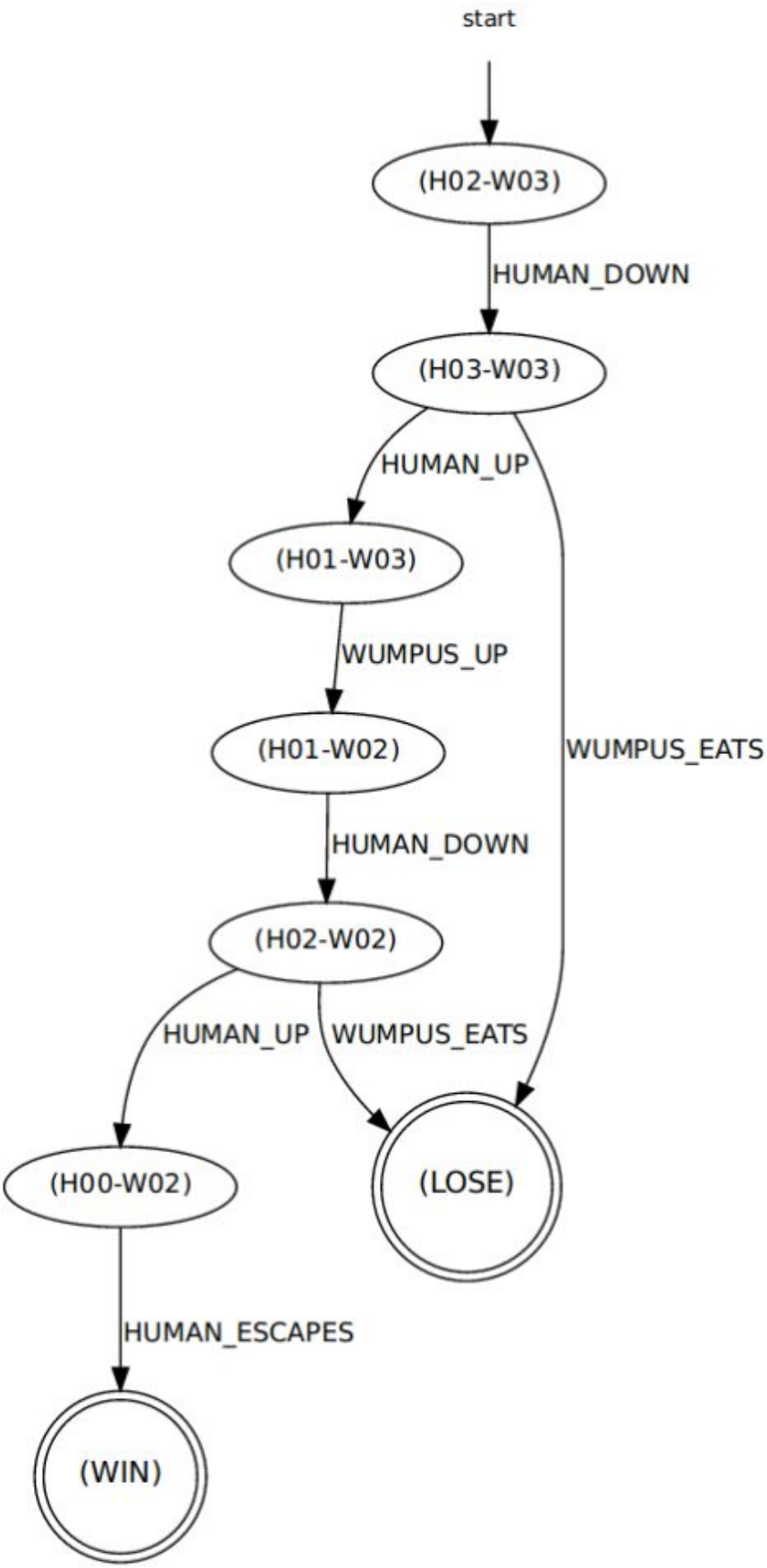
There are no other possibilities, so the wumpus will always eat the human.

4. Design a map where the human can always escape and prove via a DES model that this is the case.

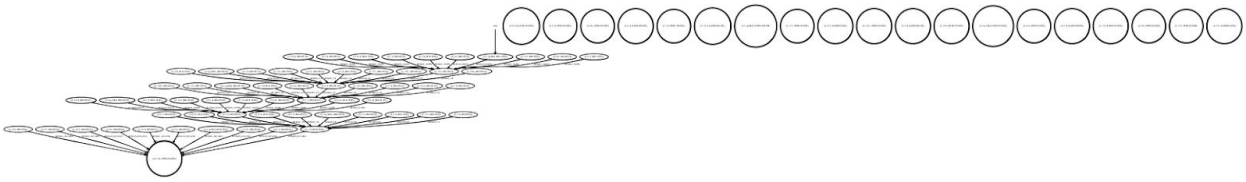
The map looks like this:

Escape		Human	Wumpus	Obstacle

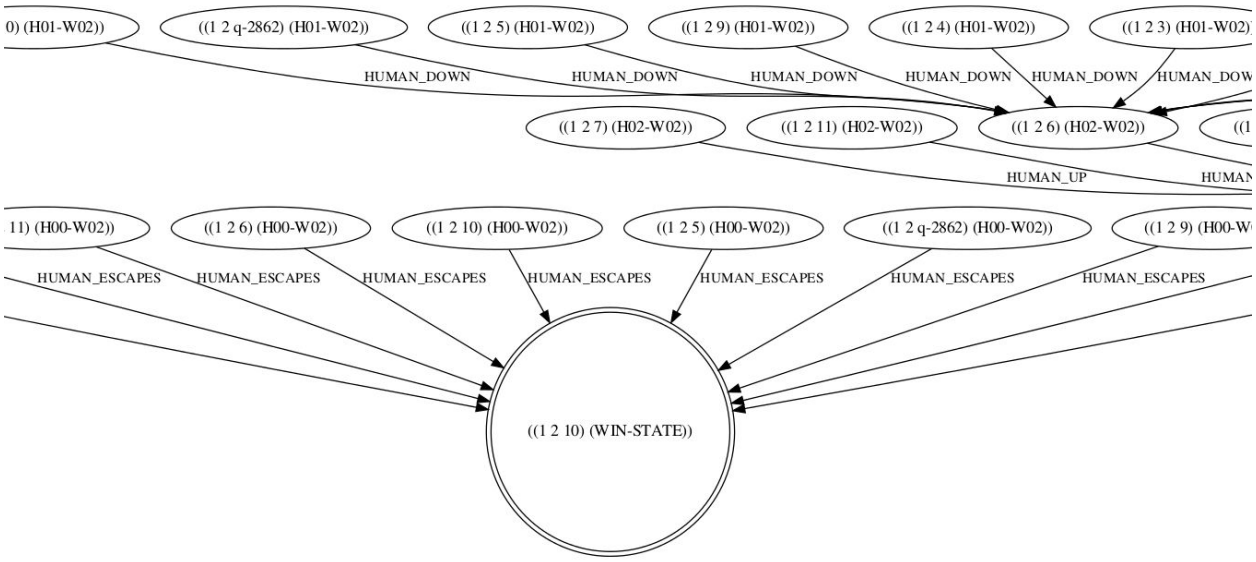
The DES:


start
(H02-W03)
HUMAN_DOWN
(H03-W03)
HUMAN_UP
WUMPUS_EATS
(H01-W03)
WUMPUS_UP
(H01-W02)
HUMAN_DOWN
(H02-W02)
HUMAN_UP
WUMPUS_EATS
(H00-W02)
(LOSE)
HUMAN_ESCAPES
(WIN)

The intersection:



Close up of same:



In the intersection of the DFA with the Regex, the win state is the only reachable accept state, and you will never become trapped in a non-accept state if you take the correct route.

Therefore, the human can always escape and win the game if they follow the correct choices.

5. **Extra Credit:** The previous questions asked you to apply regular langauges to the control of discrete event systems. There are many other applications of regular languages (text processing, DNA matching, software verication, etc.). Identify, implement, and discuss some other application of regular languages.

DNA can be thought of as a regular language. It has a finite number of letters, ACGT, and has commonly occuring word sequences that code for specific proteins.

Regular languages can be used to match a corresponding DNA strand to its partner while checking to see if the correct nucleotides match each other.

We looked into this a little bit in a previous assignment when we made a DFA to represent DNA.

This could be taken a step further to analyze commonly occurring patterns in DNA, such as the tendency for certain nucleotides to appear after a certain sequence of other ones. To make a fictional example, perhaps it is common for ATAT to be followed by CG, making ATATCG a common string.

Regular languages could be used to analyze these patterns, and study likely combinations of strings.