ECE 322
Assignment 2

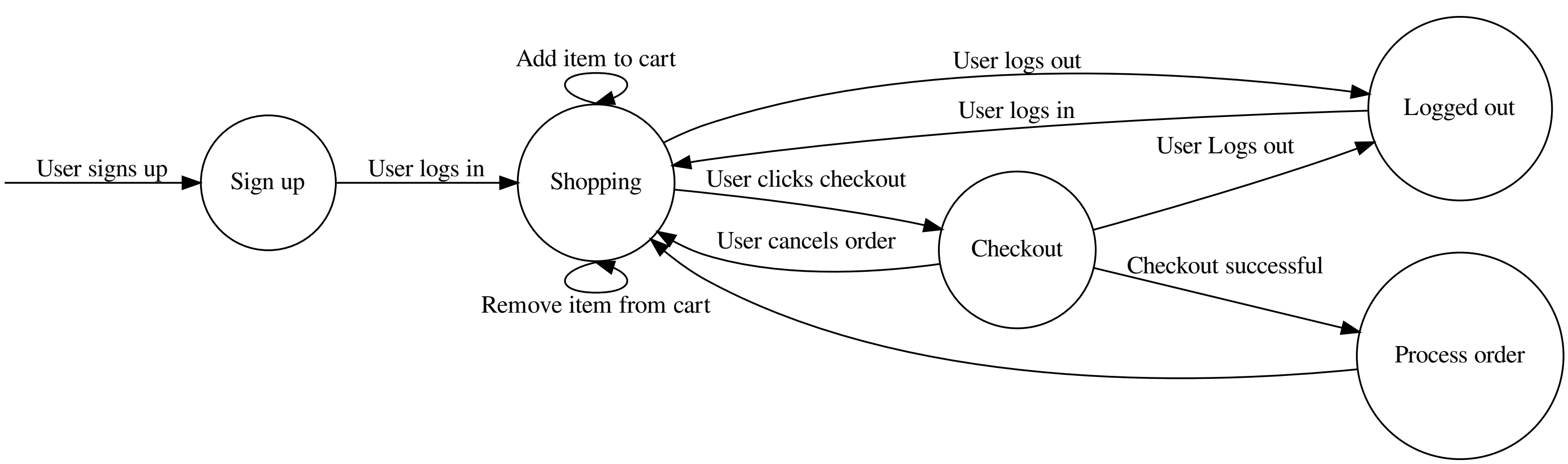
Arun Woosaree
XXXXXXX

October 7, 2019

1 e-Shopping System FSM

The following assumptions were made:

1. The items added to the user's online shopping cart are always in stock
2. A user must sign up before being able to purchase an item from this e-shopping system.
3. Once the user has signed up, their account cannot be deleted. They can however, remain logged out indefinitely.
4. Once an order is processed, the user cannot cancel their order



2 maxofThreeNumbers(int n1, int n2, int n3

2 i) Exhaustive Testing

By definition, with exhaustive testing, we would have to check for every possible combination of inputs to cover the input space. Assuming the program in question stores its **int** data type as a 64-bit signed integer, each parameter can have a minimum value of -9223372036854775808, and a maximum value of 9223372036854775807. Therefore, for each input argument, there are 18446744073709551615 possibilities. So, to account for each possible combination of inputs, there would be

$$18446744073709551615 \times 18446744073709551615 \times 18446744073709551615 = \\ 6277101735386680762814942322444851025767571854389858533375$$

test cases.

2 ii) Error Guessing

With error guessing, we can choose some inputs from the input space that from previous experience and from guessing we might think could break the program. A few test cases are listed below:

1. maxOfThreeNumbers(-1, 0 2) checks for negative and positive inputs
2. maxOfThreeNumbers(0, 0, 1) checks for when two inputs are the same
3. maxOfThreeNumbers(-9223372036854775808, 0 4) minimum value for one input
4. maxOfThreeNumbers(2, -2, 9223372036854775807) maximum value for one input
5. maxOfThreeNumbers(0, 0, 0) checks for when all arguments are zero, and also when all the arguments are the same
6. maxOfThreeNumbers(1, 2, 3) checks for all positive arguments
7. maxOfThreeNumbers(-5, -9, -2) checks for all negative arguments
8. maxofThreeNumbers('a', 'b', 'c') checks for non integer arguments

3 Equivalence Partitioning

3 i) equivalence classes

Given n input variables and m equivalence classes in each n^{th} input space, there would be $m \times n$ total equivalence classes. There would be at most one test case for each equivalence class using strong normal equivalence class testing. Using weak normal equivalence class testing, the number of test cases can be lowered by covering as many valid input equivalence classes with as few test cases as possible. Thus, we would have at most $m \times n$ test cases. As we've been implying, this is an upper bound, and the number of test cases can absolutely be reduced using the single fault assumption. We still need one test case for each invalid input equivalence class, but one test case can cover multiple valid inputs for different equivalence classes using weak normal equivalence class testing.

For example, for $n = 10$ and $m = 10$, the most number of test cases can be calculated as follows:

$$10 \times 10 = 100$$

3 ii) example with function S

given:

1. input range [-50, 50]
2. S is invoked if the reading of a sensor is within [a.b] or [c,d], $b < c$

Input Condition	Valid Input Classes	Invalid Input Classes
sensor reading	reading within [a,b] (1) reading within [c,d] (2)	reading $< a$ (3) reading $> d$ (4) $b <$ reading $< c$ (5)

Valid input test cases

- an input between a and b would cover (1). e.g. if a was -25 and b was 10, 0 would suffice
- and input between c and d would cover (2). e.g if c was 20 and d was 30, then 20 would suffice

Invalid input test cases

- an input less than a would cover (3). e.g. if a was -25 then -30 would suffice
- an input greater than d would cover (4). e.g. if d was 30 then 40 would suffice
- an input between b and c would cover (5). e.g. if b was 10 and c was 20 then 15 would suffice.

3 iii) Generalizing the problem in (ii)

With two sensors, function S can be invoked if either of the sensors are within the range $[a_i, b_i]$ or $[c_i, d_i]$, $i = 1, 2$. It does not seem like we can cover multiple valid input equivalence classes using weak normal equivalence class testing, so using strong normal equivalence class testing, we would take the Cartesian product of the equivalence classes. In (ii) we had 5 equivalence classes, so the Cartesian product is $5 \times 5 = 25$. So, there would be 25 equivalence classes for the valid inputs, since we could have any combination of either one or both sensors being within their ranges, and we would again have one test case for each equivalence class resulting in 25 total test cases.

4 3-D Input Domain

For W_1, W_2, and W_3 to form a partition, they must be mutually exclusive. Using a naive Python script, I determined with up to 0.0001 accuracy that e must be between $0 < e < 5$. If there was no restriction that e must be positive, then it seems that any negative number would suffice as well.

```
import numpy as np
X = [i for i in range(0, 11)]
Y = [i for i in range(-5, 21)]
Z = [i for i in range(0, 8)]

possible_values_of_e = np.arange(-30, 101, 0.0001).tolist()
w = set([ (x, y, z) for x in X for y in Y for z in Z])


for e in possible_values_of_e:
    w1 = set([ i for i in w if max(abs(i[0]-1), abs(i[1]-1), abs(i[2]-1)) <= e])
    w2 = set([ i for i in w if max(abs(i[0]-5), abs(i[1]-10), abs(i[2]-4)) <= e])
    w3 = w - w1 - w2

    # w3 is automatically disjoint from w1 and w2 bevause w3 = w - w1 - w2
    if len( w1 & w2  ) == 0:
        print(e)
```

Thus, it would appear that e can take on any numerical value between 0 and 5 exclusive at both ends.

$$0 < e < 5$$