# ECE 322

## Assignment 1

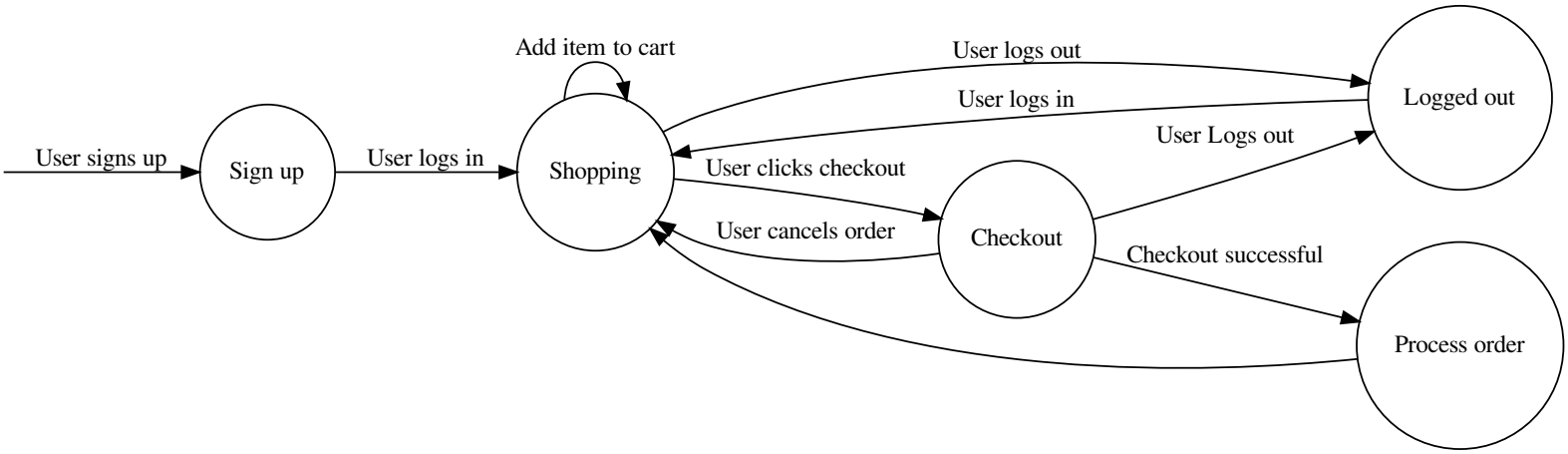
Arun Woosaree
October 6, 2019

Arun Woosaree XXXXXXX

# 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 a) 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

$$18446744073709551615 \times 18446744073709551615 \times 18446744073709551615 = 627710173538668076281494232244485102576757185438985853337 5$$

test cases.

### 2 b) 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

## 3 Equivalence Partitioning

### 3 a) 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 test case with a valid input for that equivalence class (because multiple valid equivalence classes can be covered with one test case), and at least one test case for each invalid input for that equivalence class. Thus, we would have at most $m \times n \times 2$ test cases. As we've been implying, this is an upper bound, and the number of test cases can absolutely be reduced. We still need one test case for each equivalence class's invalid input, but one test case can cover multiple valid inputs for different equivalence classes. Thus, the minimum number of test cases would be $m \times n + 1$, which would be the case when all valid equivalence classes can be covered by one test case.

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

$$10 \times 10 \times 2 = 200$$

And, if there is an input that covers all valid equivalence classes, the minimum number of test cases would be

$$10 \times 10 + 1 = 101$$

test cases.

## 3 b) 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)<br>reading within [c,d] (2) | reading $< a$ (3)<br>reading $> d$ (4)<br>$b <$ 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.