CS132: Software Engineering

Midterm Exam

15:00-16:40, May 8th, 2024

There are 5 problem sets and the total points are 20 points. Each problem set includes a few questions. For each question, the maximum possible points are stated.

Please write your answers **legibly on the answer booklet** so that we can read and understand your answers. If a problem seems ambiguous, please feel free to state your assumption explicitly and solve the problem. Obviously, your assumption should be reasonable and should not trivialize the problem.

Pledge. Copy the following pledge and sign your name in your answer booklet:

I neither cheated myself nor helped anyone cheat on this exam.

Problem 1. (4 points)

Imagine we're designing a library system to organize how we manage users and resources. In this system, everyone—librarians and library members alike—is classified as a 'user'. Each user is identified by a unique ID, a name, and an email that they use to log in and log out of the system.

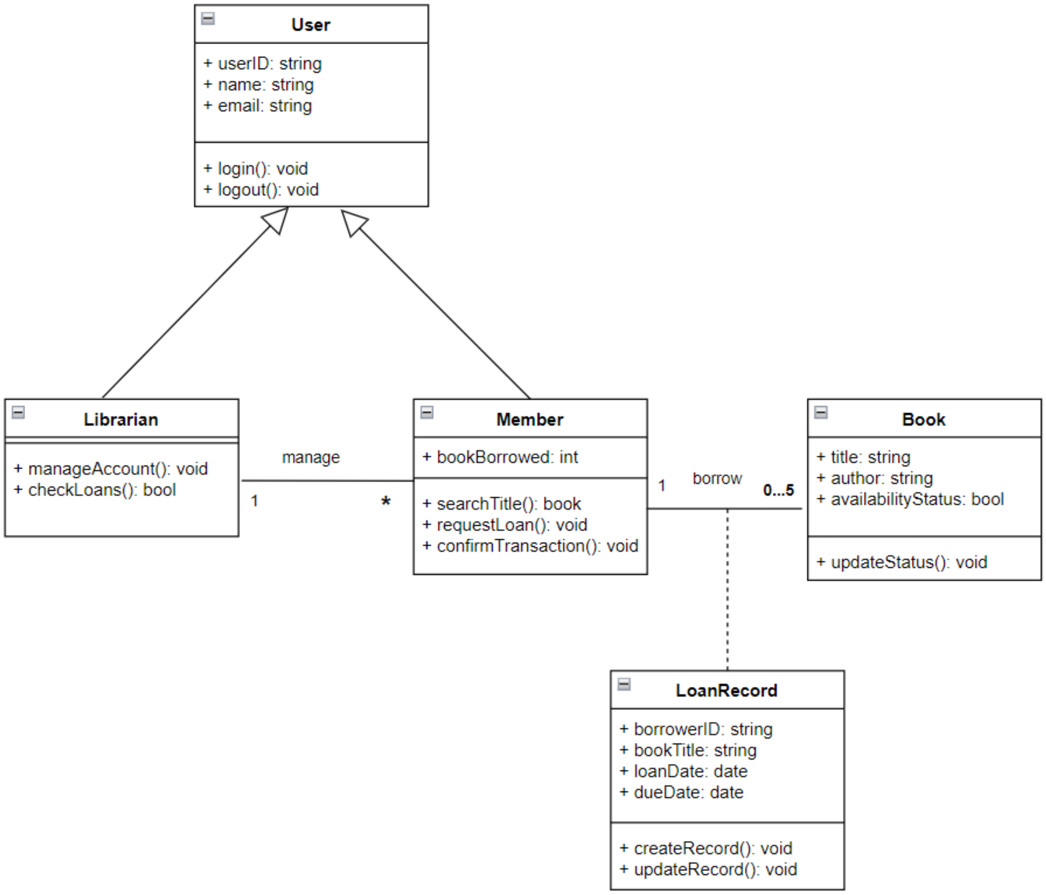
Librarians are responsible for managing member accounts and check book loans to ensure operational efficiency. Members are entitled to borrow up to five books simultaneously. They engage with the system by searching for titles, requesting loans, and confirming these transactions.

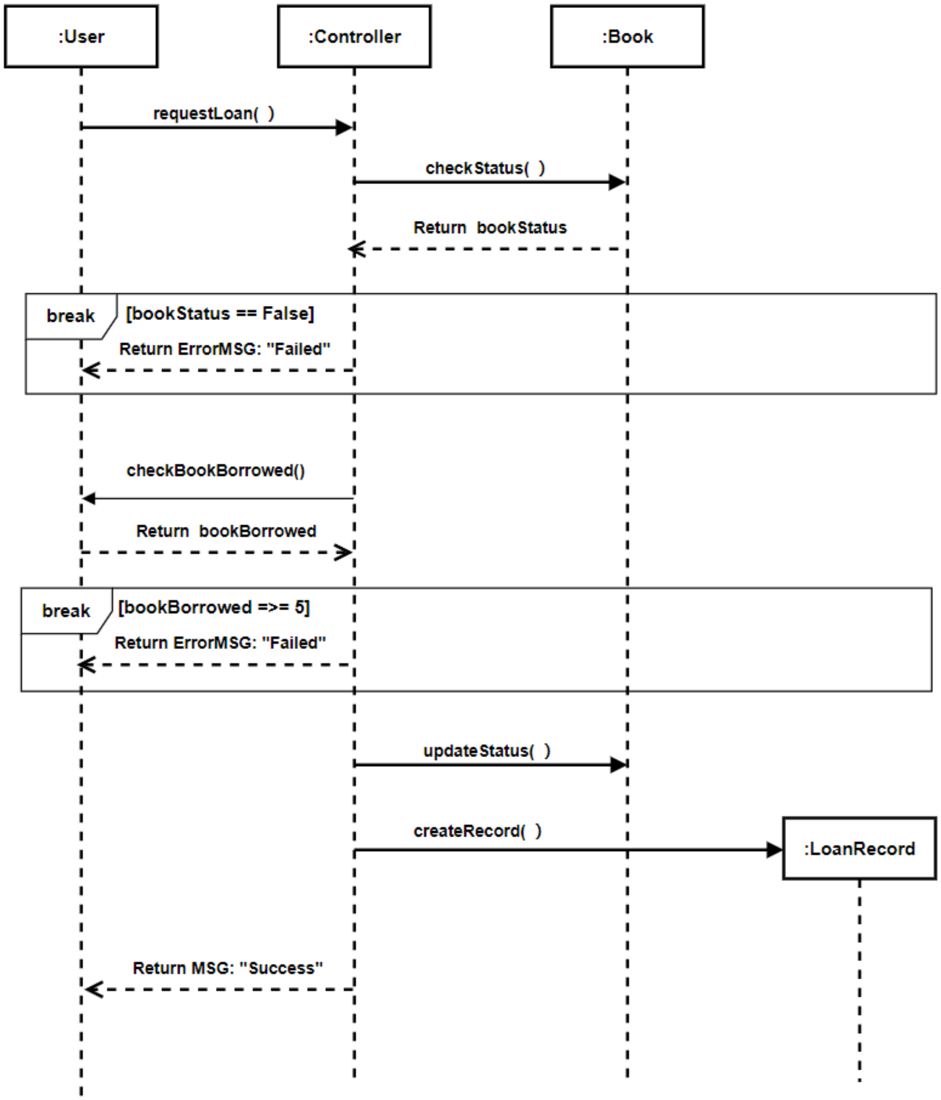
Regarding resources, each book is cataloged with a specific title, author, and its availability status. Loan records are crucial; they document the transaction details including the borrower's identity, book title, the loan date, and the due date, linking each book to its borrower.

1) (2 Points) Create a **class diagram** that shows the relationships between different classes. For each class, include a list of its attributes and methods. Additionally, clearly indicate the multiplicity of the associations among the classes. Keep in mind that loan records have a unique relationship with both the member and the book. Ensure that these relationships are accurately represented in your diagram.

2) (2 Points) Create a **sequence diagram** that details the interactions between various roles within the context of a member attempting to loan a book (Assuming the member has logged in). Include a system class that can check and update the attributes of other classes. In your diagram, account for the availability of the book and check if the member has reached their maximum allowed loans.

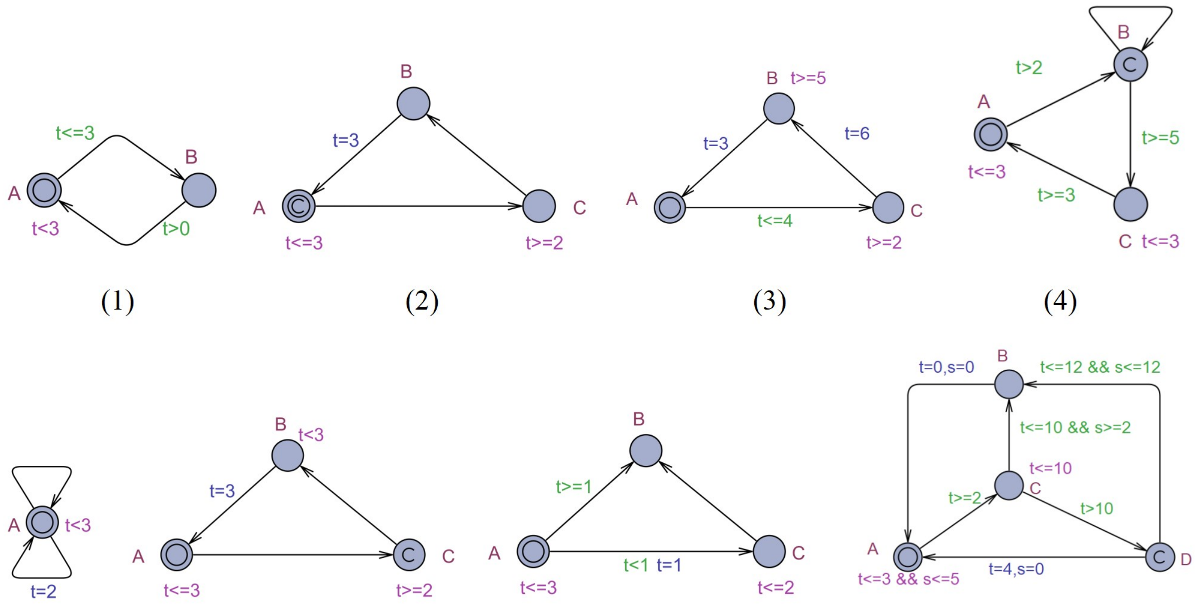
Answer:


User
+ userID: string
+ name: string
+ email: string
+ login(): void
+ logout(): void
Librarian
+ manageAccount(): void
+ checkLoans(): bool
manage
1
*
Member
+ bookBorrowed: int
+ searchTitle(): book
+ requestLoan(): void
+ confirmTransaction(): void
1
borrow
0...5
Book
+ title: string
+ author: string
+ availabilityStatus: bool
+ updateStatus(): void
LoanRecord
+ borrowerID: string
+ bookTitle: string
+ loanDate: date
+ dueDate: date
+ createRecord(): void
+ updateRecord(): void


:User
:Controller
:Book
requestLoan( )
checkStatus( )
Return bookStatus
break
[bookStatus == False]
Return ErrorMSG: "Failed"
checkBookBorrowed()
Return bookBorrowed
break
[bookBorrowed =>= 5]
Return ErrorMSG: "Failed"
updateStatus( )
createRecord( )
:LoanRecord
Return MSG: "Success"

Problem 2. (4 points)

Assume s, t are defined clocks, there are 8 different UPPAAL models shown in the figure below. For each model, write down whether the model has deadlock. For the model(s) you think have deadlock, please explain the reason for deadlock. Note that the definition of deadlock is when there are no enabled transitions at some locations at some time.



(1) (2) (3) (4)

(5) (6) (7) (8)

0.5 for each yes/no answer. -0.2 if correctly answered Deadlock but wrong explanation.

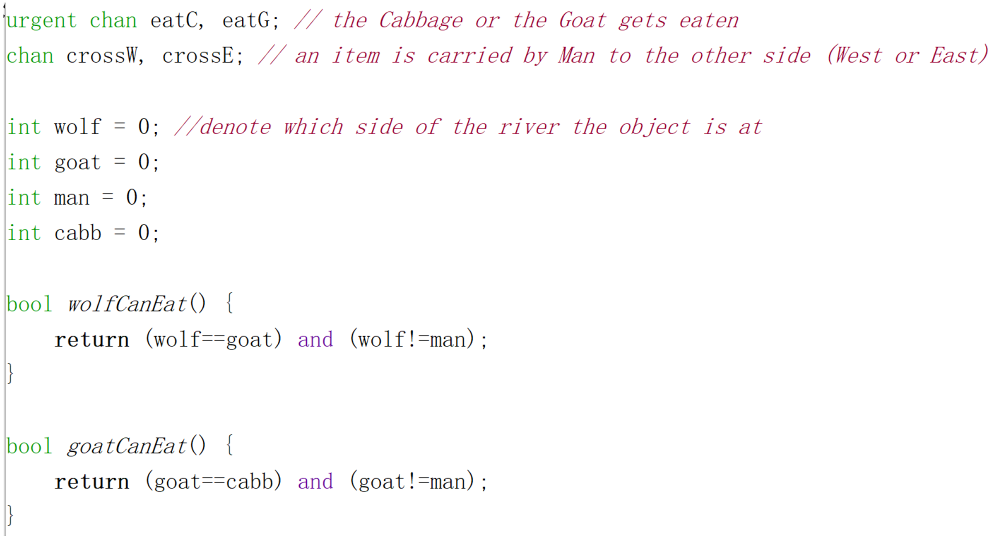
1. **Answer:** Deadlock. B to A may violate invariant of A.
2. **Answer:** Deadlock. A to C may violate invariant of C.
3. **Answer:** Deadlock. A to C may violate invariant of C
4. **Answer:** No deadlock.
5. **Answer:** No deadlock.
6. **Answer:** Deadlock. C to B may violate invariant of B.
7. **Answer:** Deadlock. No enabled transition in B.
8. **Answer:** No deadlock.

Problem 3. (4 points)

The Wolf, Goat, and Cabbage puzzle presents a classic river crossing challenge. In this scenario, a person must ferry a wolf, a goat, and some cabbage across a river using a boat that can only accommodate one of these items at a time, in addition to the person. The challenge arises from the need to ensure that neither the wolf is left alone with the goat, as the wolf will eat the goat, nor the goat with the cabbage, as the goat will consume the cabbage.

To solve this, we consider the setup with six key elements: the Man, Goat, Cabbage, Wolf, Boat, and River. The Man is the only one who can navigate the boat and hence is the only one who can cross the river unaccompanied. Each of the other items—Goat, Cabbage, and Wolf—requires the Man's presence to move across the river. The time required for crossing the river is irrelevant for solving this problem, so we can consider the crossing to be instantaneous.

In our representation, we will model the riverbanks numerically and nominatively, with '0' or 'West' representing the starting bank and '1' or 'East' representing the destination. The state of each element (Man, Goat, Cabbage, Wolf, and Boat) in relation to the riverbanks will be represented in this numeric system. This model will help us track the position of each element and strategize the sequence of crossings to ensure all are transported safely without any incidents.

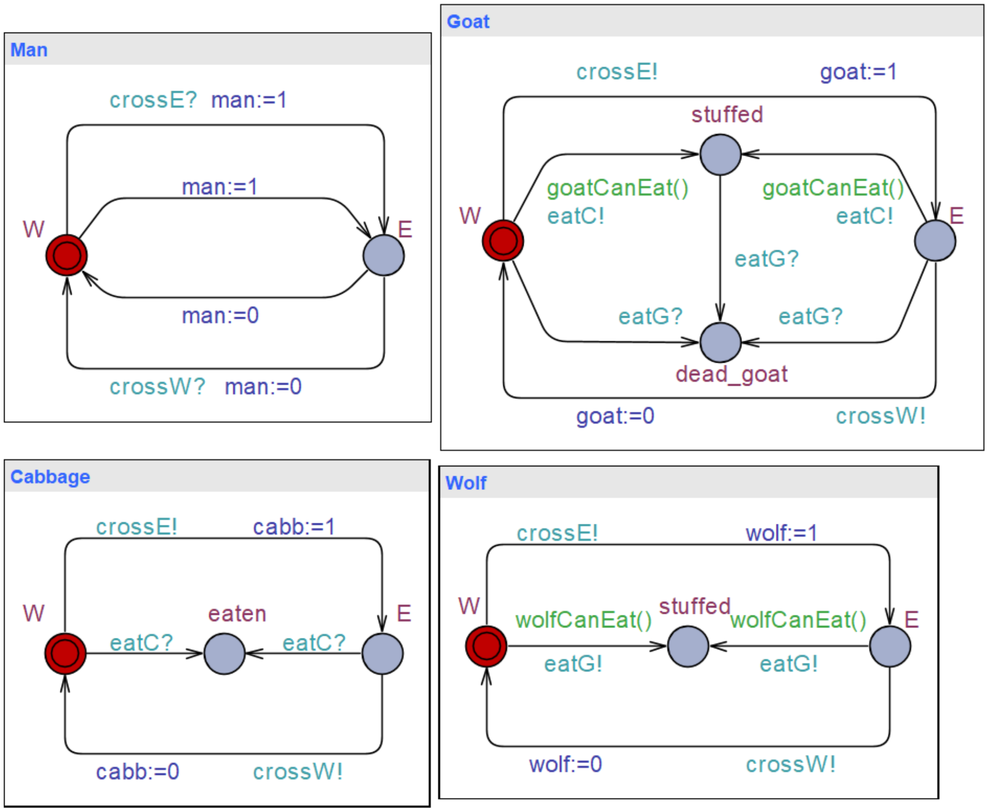


```
urgent chan eatC, eatG; // the Cabbage or the Goat gets eaten
chan crossW, crossE; // an item is carried by Man to the other side (West or East)

int wolf = 0; //denote which side of the river the object is at
int goat = 0;
int man = 0;
int cabb = 0;

bool wolfCanEat() {
    return (wolf==goat) and (wolf!=man);
}

bool goatCanEat() {
    return (goat==cabb) and (goat!=man);
}
```



For each of the following properties, state it as an UPPAAL query, and then, state if it is true or not and provide an explanation.

1) (1 Point) The system does not have deadlock.

2) (1 Point) In some execution path, wolf, goat and cabbage will reach location `stuffed`, `dead_goat` and `eaten` respectively.

3) (2 Points) The puzzle can be solved.

Answer: (1) (0.5 point) True

(0.5 point) A[] not deadlock

(2) (0.5 point) True

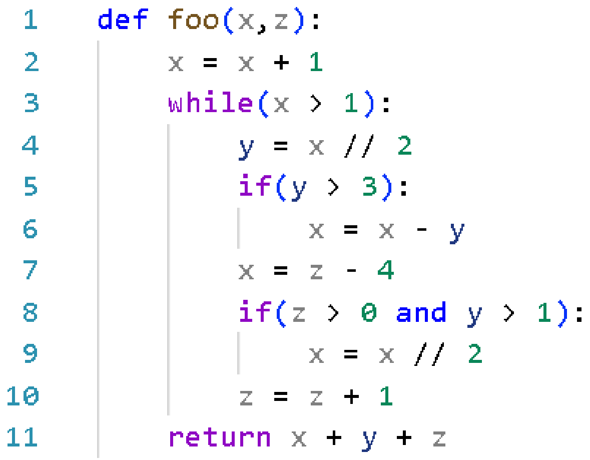
(0.5 point) E<> (Wolf.stuffed and Goat.dead_goat and Cabbage.eaten)

(3) (0.5 point) True

(0.5 point) E<> (Wolf.E and Goat.E and Cabbage.E and Man.E)

Problem 4. (4 points)

Consider the following Python program snippet. All variables are integers, and the range of **z** is **(-10, 7**].



```
1   def foo(x,z):
2       x = x + 1
3       while(x > 1):
4           y = x // 2
5           if(y > 3):
6               x = x - y
7           x = z - 4
8           if(z > 0 and y > 1):
9               x = x // 2
10          z = z + 1
11      return x + y + z
```

(a) **(2 points)** Please provide a test suite that achieves 100% branch coverage using the **fewest** possible test cases. Include the specific values of the inputs (x, y, z) for each test case and detail the paths that the program took during execution.

Answer: Branch coverage:
$x = 7, z = 0, 1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 10 - 3 - 11$
$x = 3, z = 5, 1 - 2 - 3 - 4 - 5 - 7 - 8 - 9 - 10 - 3 - 11$

(b) **(2 points)** Please specify a test suite that achieves 100% **condition** coverage but **not** 100% **branch** coverage.

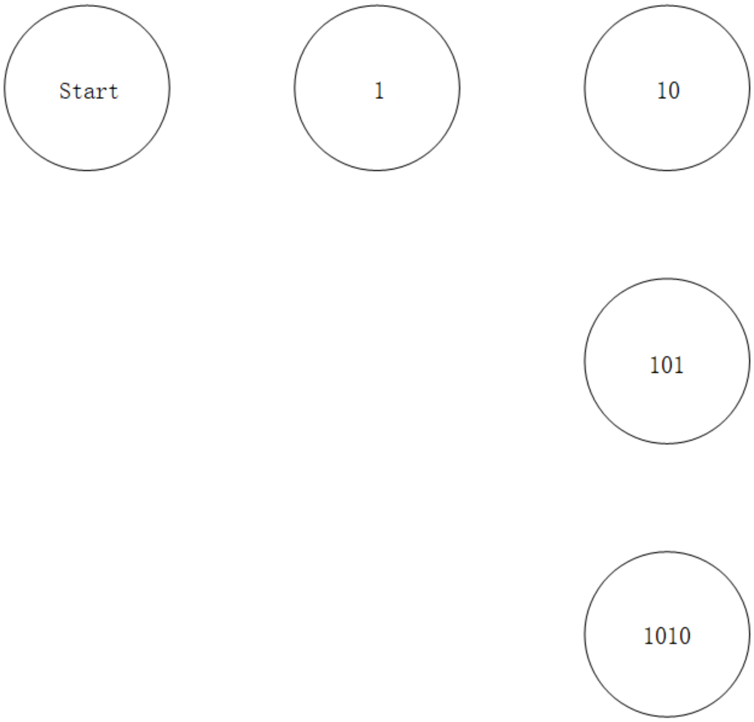
Answer: Condition coverage:
$x = 1, z = 1, 1 - 2 - 3 - 4 - 5 - 7 - 8 - 10 - 3 - 11$
$x = 7, z = 0, 1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 10 - 3 - 11$

Problem 5. (4 points)

Design an automaton that recognizes the code sequence '1010'. Your finite automaton should take inputs from single digit {0, 1} and produce outputs according to the following rules:

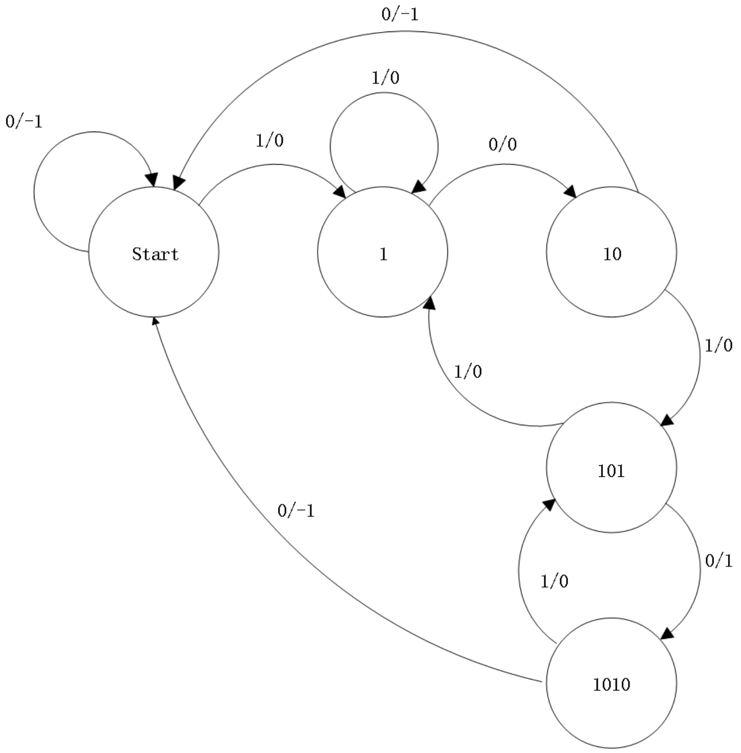
1. If the sequence '1010' is recognized, output 1.
2. If the Start state is revisited, output -1.
3. Otherwise, output 0.

The given diagram defines all states. Your automaton should initialize with the 'start' state. Ensure you account for every input possibility at each state when drawing the transitions.



Answer the following questions.

(a) **(1 point)** Complete the State Diagram for the state machine.



(b) **(1 point)** Complete the State Table for the state machine.(Fill in each cell with (Next State/Output))

Current State/Input	**0**	**1**
Start	(Start/-1)	(1/0)
1	(10/0)	(1/0)
10	(Start/-1)	(101/0)
101	(1010/1)	(1/0)
1010	(Start/-1)	(101/0)

(c) **(2 points)** Specify a test suite that achieves 100% 0-switch transition coverage.

Test Case	**TC1**	**TC2**	**TC3**	**TC4**	**TC5**	**TC6**	**TC7**	**TC8**	**TC9**	**TC10**
Start State	Start	Start	1	1	10	10	101	101	1010	1010
Input	0	1	0	1	0	1	0	1	0	1
Final State	Start	1	10	1	Start	101	1010	1	Start	101
Expected Output	-1	0	0	0	-1	0	1	0	-1	0
Test Coverage Item	Start/0	Start/1	1/0	1/1	10/0	10/1	101/0	101/1	1010/0	1010/1

Note: The X/Y on each transition refers to input/output of the transition.