CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG

Lecture 15: Exam Review

Gregory Gay
DIT635 - March 13, 2020

The Impending Exam

- Thursday, March 19, 8:30 - 12:30
  - The usual room (Alfons)
- Practice exam on Canvas.
  - Let's go over it.
  - Recommended Practice:
    - Try solving the exam without using the sample solutions. Compare your answers.

Topics

- Quality (Dependability, Performance, Scalability, Availability, Security)
- Unit Testing
- Exploratory Testing
- Functional Testing
- Structural Testing
- Data Flow Testing
- Acceptance Testing
- Regression Testing
- Integration Testing
- Testing OO Systems
- Mutation Testing
- Model-Based Testing
- Finite State Verification
- Testing in Industry

Question 1

1. A program may be correct, yet not reliable.
   a. True
   b. False
2. If a system is on an average down for a total 30 minutes during any 24-hour period:
   a. Its availability is about 98% (approximated to the nearest integer)
   b. Its reliability is about 98% (approximated to the nearest integer)
   c. Its mean time between failures is 23.5 hours
   d. Its maintenance window is 30 minutes

Question 1

1. A program may be correct, yet not reliable.
   a. **True**
   b. False
2. If a system is on an average down for a total 30 minutes during any 24-hour period:
   a. **Its availability is about 98% (approximated to the nearest integer)**
   b. Its reliability is about 98% (approximated to the nearest integer)
   c. Its mean time between failures is 23.5 hours
   d. Its maintenance window is 30 minutes

Question 1

3. In general, we need either mock objects or drivers but not both, when testing a module.
   a. True
   b. False
4. If a temporal property holds for a finite-state model of a system, it holds for any implementation that conforms to the model.
   a. True
   b. False

Question 1

3. In general, we need either mock objects or drivers but not both, when testing a module.
    a. True
    b. **False**
4. If a temporal property holds for a finite-state model of a system, it holds for any implementation that conforms to the model.
    a. **True**
    b. False

Question 1

5. Self-check oracles (assertions) do not require the expected output for judging whether a program passed or failed a test.
   a. True
   b. False
6. Object-oriented design and implementation typically have an impact on verification such that OO specific approaches are required for:
   a. Unit Testing
   b. Integration Testing
   c. System Testing
   d. Acceptance Testing

Question 1

5. Self-check oracles (assertions) do not require the expected output for judging whether a program passed or failed a test.
   a. **True**
   b. False

6. Object-oriented design and implementation typically have an impact on verification such that OO specific approaches are required for:
   a. **Unit Testing**
   b. **Integration Testing**
   c. System Testing
   d. Acceptance Testing

Question 1

7. A test suite that meets a stronger coverage criterion will find any defects that are detected by any test suite that meets only a weaker coverage criterion
    - True
    - False
8. A test suite that is known to achieve Modified Condition/Decision Coverage (MC/DC) for a given program, when executed, will exercise, at least once:
    - Every statement in the program.
    - Every branch in the program.
    - Every combination of condition values in every decision.
    - Every path in the program.

Question 1

7. A test suite that meets a stronger coverage criterion will find any defects that are detected by any test suite that meets only a weaker coverage criterion
   - True
   - **False**
8. A test suite that is known to achieve Modified Condition/Decision Coverage (MC/DC) for a given program, when executed, will exercise, at least once:
   - **Every statement in the program.**
   - **Every branch in the program.**
   - Every combination of condition values in every decision.
   - Every path in the program.

Question 1

9. Sources of information for functional testing include:
    - Requirements Specification
    - User Manuals
    - Program Source Code
    - Domain Experts
10. Category-Partition Testing technique requires identification of:
    - Parameter characteristics
    - Representative values
    - Def-Use pairs
    - Pairwise combinations

Question 1

9. Sources of information for functional testing include:
   - **Requirements Specification**
   - **User Manuals**
   - Program Source Code
   - **Domain Experts**
10. Category-Partition Testing technique requires identification of:
    - **Parameter characteristics**
    - **Representative values**
    - Def-Use pairs
    - Pairwise combinations

Question 1

11. Validation activities can only be performed once the complete system has been built.
    - True or False
12. Statement coverage criterion never requires as many test cases to satisfy as branch coverage criterion.
    - True or False
13. Requirement specifications are not needed for generating inputs to satisfy structural coverage of program code.
    - True or False
14. A system that fails to meet its user's needs may still be:
    - Correct with respect to its specification.
    - Safe to operate.
    - Robust in the presence of exceptional conditions.
    - Considered to have passed verification.

Question 1

11. Validation activities can only be performed once the complete system has been built.
    - True or **False**
12. Statement coverage criterion never requires as many test cases to satisfy as branch coverage criterion.
    - True or **False**
13. Requirement specifications are not needed for generating inputs to satisfy structural coverage of program code.
    - **True** or False
14. A system that fails to meet its user's needs may still be:
    - **Correct with respect to its specification.**
    - **Safe to operate.**
    - **Robust in the presence of exceptional conditions.**
    - **Considered to have passed verification.**

Question 2

Consider the software for air-traffic control at an airport.

Identify one performance, one availability, and one security requirement that you think would be necessary for this software and develop a quality scenario for each.

Question 2

Performance Requirement: Under normal load (< 500 aircraft), displayed aircraft positions shall be updated at least every 50 ms.

Performance Scenario:

- Overview: Check system responsiveness for displaying aircraft positions
- System and environment state: Deployment environment working correctly with less than 500 tracked aircraft.
- External stimulus: 50 Hz update of ATC system.
- System response: radar/sensor values are computed, new position is displayed to the air traffic controller with maximum error of 5 meters.
- Response measure: Display process completes in less than 50 ms.

Question 2

Availability Requirement: The system shall be able to tolerate the failure of any single server host, graphics card, display or network link.

Availability Scenario:

- Overview: One of the monitor display cards fails during transmission of a screen refresh.
- System and environment state: System is working correctly under normal load with no failures. No relevant environment factors.
- External stimulus: display card fails
- Required system response: display window manager system will detect failure within 10 ms and route display information through redundant graphics card with no user-discernable change to ATC aircraft display. Graphics card failure will be displayed as error message at bottom right

Question 2

Availability Requirement: The system shall be able to tolerate the failure of any single server host, graphics card, display or network link.

Availability Scenario:

- External stimulus: display card fails
- Required system response: display window manager system will detect failure within 10 ms and route display information through redundant graphics card with no user-discernable change to ATC aircraft display. Graphics card failure will be displayed as error message at bottom right hand of ATC display.
- Response measure: no loss in continuity of visual display and failover with visual warning completes within 1 s.

Question 2

Security Requirement: The system shall maintain audit logs of any logins to the ATC database.

Security Scenario:

- Overview: a malicious agent gains access to flight records database.
- System and environment state: System is working under normal load. No relevant environmental factors.
- External stimulus: malicious agent obtains access to the flight records database through password cracking, and downloads flight plans for commercial aircraft.

Question 2

Security Requirement: The system shall maintain audit logs of any logins to the ATC database.

Security Scenario:

- External stimulus: malicious agent obtains access to the flight records database through password cracking, and downloads flight plans for commercial aircraft.
- Required system response: audit log contains login and download information to support future prosecution of user.
- Response measure: system audit contains time, IP address, and related information w.r.t. download.

Question 3

- What is the difference between response time and throughput?
- Describe a situation where a system could display excellent throughput but poor response time and vice versa.

Question 3

What is the difference between response time and throughput?

- Response time is from the client's perspective.
  - How long does it take to service my request?
- Throughput is from the server's perspective.
  - How many requests can be processed in a given time period?

Question 3

A situation where a system could display excellent throughput/poor response time and vice versa.

- Several processors working in tandem to solve a particular problem.
  - (each segment takes time t, with number of segments s, so the total response time is t*s)
- Instead, imagine a single processor system that processes requests sequentially.
  - If there are few requests, it will have better response time than the pipelined system because there is no latency in servicing the request.
  - However, it will have very poor throughput under heavy load.

Question 4

You are building a web store that you feel will unseat Amazon as the king of online shops. Your marketing department has come back with figures stating that - to accomplish your goal - your shop will need an **availability** of at least 99%, a **probability of failure on demand** of less than 0.1, and a **rate of fault occurrence** of less than 2 failures per 8-hour work period.

You have recently finished a testing period of one week (seven full 24-hour days). During this time, 972 requests were served to the page. The product failed a total of 64 times. 37 of those resulted in a system crash, while the remaining 27 resulted in incorrect shopping cart totals. When the system crashes, it takes 2 minutes to restart it.

Question 4

Want: **availability** of at least 99%, a **probability of failure on demand** of less than 0.1, and a **rate of fault occurrence** of less than 2 failures per 8-hour work period.

Currently: 972 requests.. The product failed a total of 64 times (37 crashes, 27 incorrect computations). It takes 2 minutes to restart.

- What is the rate of fault occurrence?

Question 4

Want: **availability** of at least 99%, a **probability of failure on demand** of less than 0.1, and a **rate of fault occurrence** of less than 2 failures per 8-hour work period.

Currently: 972 requests.. The product failed a total of 64 times (37 crashes, 27 incorrect computations). It takes 2 minutes to restart.

- What is the rate of fault occurrence?
- **64/168 hours = 0.38/hour = 3.04/8 hour work day**

Question 4

Want: **availability** of at least 99%, a **probability of failure on demand** of less than 0.1, and a **rate of fault occurrence** of less than 2 failures per 8-hour work period.

Currently: 972 requests.. The product failed a total of 64 times (37 crashes, 27 incorrect computations). It takes 2 minutes to restart.

- What is the probability of failure on demand?

Question 4

Want: **availability** of at least 99%, a **probability of failure on demand** of less than 0.1, and a **rate of fault occurrence** of less than 2 failures per 8-hour work period.

Currently: 972 requests.. The product failed a total of 64 times (37 crashes, 27 incorrect computations). It takes 2 minutes to restart.

- What is the probability of failure on demand?
- **64/972 = 0.066**

Question 4

Want: **availability** of at least 99%, a **probability of failure on demand** of less than 0.1, and a **rate of fault occurrence** of less than 2 failures per 8-hour work period.

Currently: 972 requests.. The product failed a total of 64 times (37 crashes, 27 incorrect computations). It takes 2 minutes to restart.

- What is the availability?

Question 4

Want: **availability** of at least 99%, a **probability of failure on demand** of less than 0.1, and a **rate of fault occurrence** of less than 2 failures per 8-hour work period.

Currently: 972 requests.. The product failed a total of 64 times (37 crashes, 27 incorrect computations). It takes 2 minutes to restart.

- What is the availability?
- **It was down for (37*2) = 74 minutes out of 168 hours = 74/10089 minutes = 0.7% of the time. Availability = 99.3%**

Question 4

Want: **availability** of at least 99%, a **probability of failure on demand** of less than 0.1, and a **rate of fault occurrence** of less than 2 failures per 8-hour work period.

Currently: 972 requests.. The product failed a total of 64 times (37 crashes, 27 incorrect computations). It takes 2 minutes to restart.

- Is the product ready to ship? If not, why not?

Question 4

Want: **availability** of at least 99%, a **probability of failure on demand** of less than 0.1, and a **rate of fault occurrence** of less than 2 failures per 8-hour work period.

Currently: 972 requests.. The product failed a total of 64 times (37 crashes, 27 incorrect computations). It takes 2 minutes to restart.

- Is the product ready to ship? If not, why not?
- **No. Availability, POFOD are good. ROCOF is too low. How would you improve it?**

Question 5

You are testing the following method:

```
public double max(double a, double b);
```

Devise three executable test cases for this method in the JUnit notation. See the attached handout for a refresher on the notation.

```
@Test
  public void aLarger() {
    double a = 16.0;
    double b = 10.0;
    double expected = 16.0;
    double actual = max(a,b);
    assertTrue("should be larger", actual>b);
    assertEquals(expected, actual);
  }
@Test
  public void bLarger() {
    double a = 10.0;
    double b = 16.0;
    double expected = 16.0;
    double actual = max(a,b);
    assertThat("b should be larger", b>a);
    assertEquals(expected, actual);
  }
```

```
@Test
  public void bothEqual() {
    double a = 16.0;
    double b = 16.0;
    double expected = 16.0;
    double actual = max(a,b);
    assertEquals(a,b);
    assertEquals(expected, actual);
  }
@Test
  public void bothNegative() {
    double a = -2.0;
    double b = -1.0;
    double expected = -1.0;
    double actual = max(a,b);
    assertTrue("should be negative",actual<0);
    assertEquals(expected, actual);
  }
```

Question 6

After *carefully and thoroughly* developing a collection of requirements-based tests and running your test suite, you determine that you have achieved only 60% statement coverage. You are surprised (and saddened), since you had done a very thorough job developing the requirements-based tests and you expected the result to be closer to 100%.

Question 6

Briefly describe two (2) things that might have happened to account for the fact that 40% of the code was not exercised during the requirements-based tests.

- Poor job choosing test cases.
- Missing requirements.
- Dead or inactive code.
- Error-handling.
  - Code used only in special cases.

Question 6

Should you, in general, be able to expect 100% statement coverage through thorough requirements-based testing alone (why or why not)?

- No.
- There are almost always special cases not covered by requirements.
    - Code optimizations, debug code, exception handling.

Question 6

Some structural criteria, such as MC/DC, prescribe obligations that are impossible to satisfy. What are two reasons why a test obligation may be impossible to satisfy?

- Impossible combination of conditions
- Defensive programming (situations that may not happen in practice are planned for).
- Other situations that result in unused code (i.e., code implemented for future use that is not currently reachable).

Question 7

In class we discussed the importance of defining a test case for each requirement. What are the two primary benefits of defining this test case?

- Helps when performing integration testing. Can build test cases early, apply to code once it is written.
- Forces us to write testable requirements.

Question 8

- The airport connection check is part of a travel reservation system. It checks the validity of a single connection between two flights in an itinerary.
  - If the arrival airport of Flight A differs from the departure airport of Flight B, the connection is invalid.
  - If the departure time of Flight B is too close to the arrival time of Flight A, the connection is invalid.
  - If an airport doesn't exist, the connection is invalid…

Question 8

```
validConnection(Flight arrivingFlight, Flight departingFlight)
                returns ValidityCode
```

A Flight is a data structure consisting of:

- A unique identifying flight code (string, three characters followed by four numbers).
- The originating airport code (three character string).
- The scheduled departure time (in universal time).
- The destination airport code (three character string).
- The scheduled arrival time (in universal time).

There is also a flight database, where each record contains:

- Three-letter airport code (three character string).
- Airport country (two character string).
- Minimum connection times (integer, minimum number of minutes that must be allowed for flight connections).

Question 8

There is also a flight database, where each record contains:

- Three-letter airport code (three character string).
- Airport country (two character string).
- Minimum connection time (integer, minimum number of minutes that must be allowed for flight connections).

`ValidityCode` is an integer with value:

- 0 for OK
- 1 for invalid airport code
- 2 for a connection that is too short
- 3 for flights that do not connect (arrivingFlight does not land in the same location as departingFlight)
- 4 for any other errors (malformed input or any other unexpected errors).

Parameter: Arriving flight

Flight code:
- ***malformed***
- ***not in database***
- ***valid***

Originating airport code:
- ***malformed***
- ***not in database***
- ***valid city***

Scheduled departure time:
- ***syntactically malformed***
- ***out of legal range***
- ***legal***

Destination airport (transfer airport):
- ***malformed***
- ***not in database***
- ***valid city***

Scheduled arrival time (tA):
- ***syntactically malformed***
- ***out of legal range***
- ***legal***

Parameter: Departing flight

Flight code:
- ***malformed***
- ***not in database***
- ***valid***

Originating airport code:
- ***malformed***
- ***not in database***
- ***differs from transfer airport***
- ***same as transfer airport***

Scheduled departure time:
- ***syntactically malformed***
- ***out of legal range***
- ***before arriving flight time (tA)***
- ***between tA and tA + minimum connection time (CT)***
- ***equal to tA + CT***
- ***greater than tA + CT***

Destination airport code:
- ***malformed***
- ***not in database***
- ***valid city***

Scheduled arrival time:
- ***malformed***
- ***out of legal range***
- ***legal***

Parameter: Database record

This parameter refers to the database record corresponding to the transfer airport.

Airport code:
- ***malformed***
- ***blank***
- ***valid***

Airport country:
- ***malformed***
- ***blank***
- ***invalid (not a country)***
- ***valid***

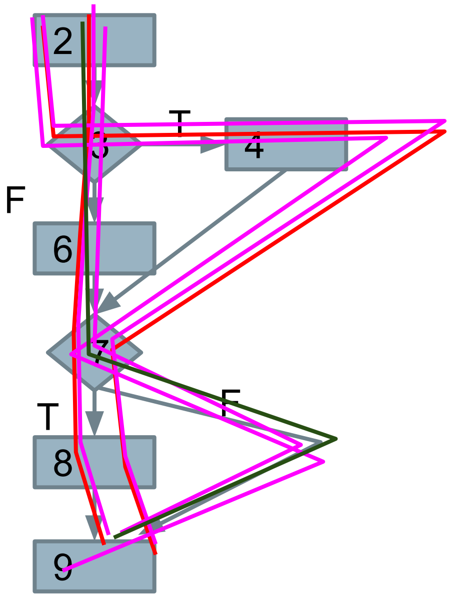
Minimum connection time:
- ***malformed***
- ***blank***
- ***invalid***
- ***valid***

Question 9

- Draw the control-flow graph for this method.
- Develop test input that will provide statement coverage.
- Develop test input that will provide branch coverage.
- Develop test input that will provide path coverage.

```
int findMax(int a, int b, int c) {
	int temp;
	if (a>b)
		temp=a;
	else
		temp=b;
	if (c>temp)
		temp = c;
	return temp;
}
```

Question 9




```
1. int findMax(int a, int b, int c) {
2.   int temp;
3.   if (a>b)
4.        temp=a;
5.   else
6.        temp=b;
7.   if (c>temp)
8.        temp = c;
9.   return temp;
10. }
```

Statement:
(3,2,4), (2,3,4)
Branch:
(3,2,4), (3,4,1)

Path:
(4,2,5), (4,2,1), (2,3,4), (2,3,1)

Question 9

- Modify the program to introduce a fault such that even path coverage could miss the fault.

```
int findMax(int a, int b, int c)
{
    int temp;
    if (a>b)
        temp=a;
    else
        temp=b;
    if (c>temp)
        temp = c;
    return temp;
}
```

Use (a >b+1) instead of (a>b) and the test input from the last slide:
(4,2,5), (4,2,1), (2,3,4), (2,3,1)
will not reveal the fault.

Question 10

- Identify all DU pairs and write test cases to achieve All DU Pair Coverage.
  - Hint - remember that there is a loop.

```
int doSomething(int x, int y)
{
    while(y > 0) {
        if(x > 0) {
            y = y - x;
        }else {
            x = x + 1;
        }
    }
    return x + y;
}
```

Question 10

```
1.  int doSomething(int x, int y)
2.  {
3.      while(y > 0) {
4.          if(x > 0) {
5.              y = y - x;
6.          }else {
7.              x = x + 1;
8.          }
9.      }
10.     return x + y;
11. }
```

Variable	Defs	Uses
x	1, 7	4, 5, 7, 10
y	1, 5	3, 5, 10

Variable	D-U Pairs
x	(1, 4), (1, 5), (1, 7), (1, 10), (7, 4), (7, 5), (7, 7), (7, 10)
y	(1, 3), (1, 5), (1, 10), (5, 3), (5, 5), (5, 10)

Question 10

```
1.  int doSomething(int x, int y)
2.  {
3.      while(y > 0) {
4.          if(x > 0) {
5.              y = y - x;
6.          }else {
7.              x = x + 1;
8.          }
9.      }
10.     return x + y;
11. }
```

Variable	D-U Pairs
x	~~(1, 4), (1, 5), (1, 7), (1, 10), (7, 4), (7, 5), (7, 7), (7, 10)~~
y	~~(1, 3), (1, 5), (1, 10), (5, 3), (5, 5), (5, 10)~~

Test 1: (x = 1, y = 2)
Covers lines 1, 3, 4, 5, 3, 4, 5, 3, 10
Test 2: (x = -1, y = 1)
Covers lines 1, 3, 4, 6, 7, 3, 4, 6, 7, 3, 4, 5, 3, 10
Test 3: (x = 1, y = 0)
Covers lines 1, 3, 8

Question 11

In a directed graph with a designated exit node, we say that a node **m** post-dominates another node **n**, if m appears on every path from n to the exit node.

Let us write *m pdom n* to mean that m post-dominates n, and *pdom(n)* to mean the set of all post-dominators of n, i.e., $\{m \mid m \; pdom \; n\}$.

Question 11

1. Does *b pdom b* hold true for all b?
2. Can both *a pdom b* and *b pdom a* hold true for two different nodes a and b?

1. Yes. Each node must appear on every path to the exit from itself.
2. Not if they are **different** nodes. If a pdom b, then b must be on all paths from a to the exit. Node a cannot appear after b at the same time.

Question 11

3. If both *c pdom b* and *b pdom a* hold true, what can you say about the relationship between c and a?
4. If both *c pdom a* and *b pdom a* hold true, what can you say about the relationship between c and b?

3. *c pdom a*. Node b appears on all paths from a to the exit. Node c must appear on the subpath from b to the exit.
4. Either *c pdom b* or *b pdom c*. Both b and c must appear on all paths from a to the exit. One will pdom the other.

Question 12

Consider the following function:

```
bSearch(A, value, start, end) {
    if (end <= start)
        return -1;
    mid = (start + end) / 2;
    if (A[mid] > value) {
        return bSearch(A, value, start, mid);
    } else if (value > A[mid]) {
        return bSearch(A, value, mid+1, end);
    } else {
        return mid;
    }
}
```

1. Create an equivalent mutant.

Question 12

Consider the following function:

```
bSearch(A, value, start, end) {
        if (end <= start)
                return -1;
        mid = (start + end) / 2;
        if (A[mid] > value) {
                return bSearch(A, value, start, mid);
        } else if (value > A[mid]) {
                return bSearch(A, value, mid+1, end);
        } else {
                return mid;
        }
}
```

1. Create an equivalent mutant.

```
} else if (value > A[mid]) {
        return bSearch(A, value,
mid+1, end);
} else {
}
return mid;
}
```

SES - End Block Shift

Question 12

Consider the following function:

```
bSearch(A, value, start, end) {
	if (end <= start)
		return -1;
	mid = (start + end) / 2;
	if (A[mid] > value) {
		return bSearch(A, value, start, mid);
	} else if (value > A[mid]) {
		return bSearch(A, value, mid+1, end);
	} else {
		return mid;
	}
}
```

2. Create an invalid mutant.

Question 12

Consider the following function:

```
bSearch(A, value, start, end) {
        if (end <= start)
                return -1;
        mid = (start + end) / 2;
        if (A[mid] > value) {
                return bSearch(A, value, start, mid);
        } else if (value > A[mid]) {
                return bSearch(A, value, mid+1, end);
        } else {
                return mid;
        }
}
```

2. Create an invalid mutant.

mid = (start + end) / 2;
~~if (A[mid] > value) {~~
return bSearch(A, value, start, mid);
} else if (value > A[mid]) {
return bSearch(A, value, mid+1, end);
} else {
return mid;
}
}

SDL - Statement Deletion

Question 12

Consider the following function:

```
bSearch(A, value, start, end) {
	if (end <= start)
		return -1;
	mid = (start + end) / 2;
	if (A[mid] > value) {
		return bSearch(A, value, start, mid);
	} else if (value > A[mid]) {
		return bSearch(A, value, mid+1, end);
	} else {
		return mid;
	}
}
```

3. Create a valid-but-not-useful mutant.

Question 12

Consider the following function:

```
bSearch(A, value, start, end) {
	if (end <= start)
		return -1;
	mid = (start + end) / 2;
	if (A[mid] > value) {
		return bSearch(A, value, start, mid);
	} else if (value > A[mid]) {
		return bSearch(A, value, mid+1, end);
	} else {
		return mid;
	}
}
```

3. Create a valid-but-not-useful mutant.

```
bSearch(A, value, start, end) {
	if (end > start)
		return -1;
	mid = (start + end) / 2;
```

ROR - Relational Operator Replacement

Question 12

Consider the following function:

```
bSearch(A, value, start, end) {
	if (end <= start)
		return -1;
	mid = (start + end) / 2;
	if (A[mid] > value) {
		return bSearch(A, value, start, mid);
	} else if (value > A[mid]) {
		return bSearch(A, value, mid+1, end);
	} else {
		return mid;
	}
}
```

3. Create a useful mutant.

```
	} else if (value > A[mid]) {
		return bSearch(A, value,
	mid+2, end);
	} else {
		return mid;
	}

}
```

CRP - Constant for Constant Replacement

Question 13

Suppose that finite state verification of an abstract model of some software exposes a counter-example to a property that is expected to hold for true for the system.

Briefly describe what follow-up actions would you take and why?

Question 13

Tells us one of the following is an issue:

- The model
  - Fault in the model, bad assumptions, incorrect interpretation of requirements
- The property
  - Property not formulated correctly.
- The requirements
  - Contradictory or incorrect requirements.

Question 14

Temporal Operators: A quick reference list.

- G p: p holds globally at every state on the path
- F p: p holds at some state on the path
- X p: p holds at the next (second) state on the path
- p U q: q holds at some state on the path and p holds at every state before the first state at which q holds.
- A: for all paths from a state, used in CTL as a modifier for the above properties (AG p)
- E: for some path from a state, used in CTL as a modifier for the above properties (EF p)

AG (pedestrian_light = walk -> traffic_light != green)

State variables:

- **traffic_light: {RED, YELLOW, GREEN}**
- **pedestrian_light: {WAIT, WALK, FLASH}**
- **button: {RESET, SET}**

Initially: **traffic_light = RED, pedestrian_light = WAIT, button = RESET**

Transitions:

pedestrian_light:

- **WAIT → WALK if traffic_light = RED**
- **WAIT → WAIT otherwise**
- **WALK → {WALK, FLASH}**
- **FLASH → {FLASH, WAIT}**

traffic_light:

- **RED → GREEN if button = RESET**
- **RED → RED otherwise**
- **GREEN → {GREEN, YELLOW} if button = SET**
- **GREEN → GREEN otherwise**
- **YELLOW→ {YELLOW, RED}**

button:

- **SET → RESET if pedestrian_light = WALK**
- **SET → SET otherwise**
- **RESET → {RESET, SET} if traffic_light = GREEN**
- **RESET → RESET otherwise**

G (traffic_light = RED & button = RESET -> F (traffic_light = green))

State variables:

- **traffic_light: {RED, YELLOW, GREEN}**
- **pedestrian_light: {WAIT, WALK, FLASH}**
- **button: {RESET, SET}**

Initially: **traffic_light = RED, pedestrian_light = WAIT, button = RESET**

Transitions:

pedestrian_light:

- **WAIT → WALK if traffic_light = RED**
- **WAIT → WAIT otherwise**
- **WALK → {WALK, FLASH}**
- **FLASH → {FLASH, WAIT}**

traffic_light:

- **RED → GREEN if button = RESET**
- **RED → RED otherwise**
- **GREEN → {GREEN, YELLOW} if button = SET**
- **GREEN → GREEN otherwise**
- **YELLOW→ {YELLOW, RED}**

button:

- **SET → RESET if pedestrian_light = WALK**
- **SET → SET otherwise**
- **RESET → {RESET, SET} if traffic_light = GREEN**
- **RESET → RESET otherwise**

Negate to get trap property:
G !(button = SET -> F (pedestrian_light = WALK))

State variables:

- **traffic_light: {RED, YELLOW, GREEN}**
- **pedestrian_light: {WAIT, WALK, FLASH}**
- **button: {RESET, SET}**

Initially: **traffic_light = RED, pedestrian_light = WAIT, button = RESET**

Transitions:

pedestrian_light:

- **WAIT → WALK if traffic_light = RED**
- **WAIT → WAIT otherwise**
- **WALK → {WALK, FLASH}**
- **FLASH → {FLASH, WAIT}**

traffic_light:

- **RED → GREEN if button = RESET**
- **RED → RED otherwise**
- **GREEN → {GREEN, YELLOW} if button = SET**
- **GREEN → GREEN otherwise**
- **YELLOW→ {YELLOW, RED}**

button:

- **SET → RESET if pedestrian_light = WALK**
- **SET → SET otherwise**
- **RESET → {RESET, SET} if traffic_light = GREEN**
- **RESET → RESET otherwise**

Question 15

Microwave controller

- Door: {Open, Closed} -- sensor input indicating state of the door
- Button: {None, Start, Stop} -- button press
- Timer: 0...999 -- (remaining) seconds to cook
- Cooking: Boolean -- state of the heating element

In CTL:

- The microwave shall never cook when the door is open.
- **AG (Door = Open -> !Cooking)**

Question 15

Microwave controller

- Door: {Open, Closed} -- sensor input indicating state of the door
- Button: {None, Start, Stop} -- button press
- Timer: 0...999 -- (remaining) seconds to cook
- Cooking: Boolean -- state of the heating element

In CTL:

- The microwave shall cook only as long as there is remaining cook time.
- **AG (Cooking -> Timer > 0)**

Question 15

Microwave controller

- Door: {Open, Closed} -- sensor input indicating state of the door
- Button: {None, Start, Stop} -- button press
- Timer: 0...999 -- (remaining) seconds to cook
- Cooking: Boolean -- state of the heating element

In CTL:

- If the stop button is pressed when the microwave is not cooking, the remaining cook time shall be cleared.
- **AG (Button = Stop & !Cooking -> AX (Timer = 0))**

Question 15

Microwave controller

- Door: {Open, Closed} -- sensor input indicating state of the door
- Button: {None, Start, Stop} -- button press
- Timer: 0...999 -- (remaining) seconds to cook
- Cooking: Boolean -- state of the heating element

In LTL:

- It shall never be the case that the microwave can continue cooking indefinitely.
- **G (Cooking -> F (!Cooking))**

Question 15

Microwave controller

- Door: {Open, Closed} -- sensor input indicating state of the door
- Button: {None, Start, Stop} -- button press
- Timer: 0...999 -- (remaining) seconds to cook
- Cooking: Boolean -- state of the heating element

In LTL:

- The only way to initiate cooking shall be pressing the start button when the door is closed and the remaining cook time is not zero.
- **G (!Cooking U ((Button = Start & Door = Closed) & (Timer > 0)))**

Question 15

Microwave controller

- Door: {Open, Closed} -- sensor input indicating state of the door
- Button: {None, Start, Stop} -- button press
- Timer: 0...999 -- (remaining) seconds to cook
- Cooking: Boolean -- state of the heating element

In LTL:

- The microwave shall continue cooking when there is remaining cook time unless the stop button is pressed or the door is opened.
- **G ((Cooking & Timer > 0) -> X (((Cooking | (!Cooking & Button = Stop)) | (!Cooking & Door = Open)))**

Any other questions?

Thank you for being a great class!

UNIVERSITY OF GOTHENBURG

CHALMERS

UNIVERSITY OF TECHNOLOGY