Examination
Model Based Testing
DIT848 / DAT260 / DAT261

Software Engineering and Management
Chalmers | University of Gothenburg

Wednesday June 1st, 2016

Time	08:30-12:30
Location	Johanneberg
Responsible teacher	Gerardo Schneider (031 772 6073)
Contact during exam	Jan Schröder (031-772 61 79)
Total number of pages	**XX** (including this page)
Long exam (7.5 HEC)	Max score: 100 pts
Grade limits (7.5 HEC)	3 (G): at least 50 pts
	4: at least 65 pts
	5 (VG): at least 80 pts
Short exam (4.5 HEC)	Max score: 60 pts
Grade limits (4.5 HEC)	3 (G): at least 30 pts
	4: at least 40 pts
	5 (VG): at least 50 pts

NOTE: You should have a minimum number of points in each of the tasks in order to pass the exam; see more details in the next page.

ALLOWED AID:

- One book on testing
- 1 page containing student's own notes
- English dictionary
- **NOT ALLOWED:** Anything else not explicitly mentioned above (including additional books, other notes, previous exams, or any form of electronic device: dictionaries, agendas, computers, mobile phones, etc.)

PLEASE OBSERVE THE FOLLOWING:

- Motivate your answers (a simple statement of facts not answering the question is considered to be invalid);
- Start each task on a new paper;
- Sort the tasks in order before handing them in;
- Write your student code on each page and put the number of the task on **every** paper;
- Read carefully the section "ABOUT THE FORMAT OF THE EXAM" to identify whether you are supposed to do the "short" (4.5 HEC) or the "long" (7.5 HEC) exam.

ABOUT THE FORMAT OF THE EXAM:

Please note that the tasks you are supposed to solve depend on whether you are taking the "short" exam modality (corresponding to 4.5 HEC), or the "long" one (corresponding to 7.5 HEC). See more information below.

EXAM 4.5 HEC

Who is entitled to take the short (4.5 HEC) exam?
All GU students registered for the MBT course in 2015 or later.
All Chalmers students registered for the MBT course in 2016 (DAT261).

Content of the short (4.5 HEC) exam. The exam consists of **3** tasks (**tasks 1-3**).

Information on grading scale for the short (4.5 HEC) exam.
Each task is worth 20 points. In order to reach the level to pass with **3** (**G**) you need **at least 30 points** out of the total, and **at least 5 points per task.** To pass with **4** you need **at least 40 points** out of the total, and **at least 7 points per task**.
In order to pass with distinction **5** (**VG**) you need to reach **at least 50 points** out of the total, and you must score **at least 12 points per task**.

IMPORTANT: Note that you should have a minimum number of points in each of the 3 tasks in order to pass the exam, so avoid letting unanswered tasks.

EXAM 7.5 HEC

Who should take the long (7.5 HEC) exam?
All GU students **NOT** registered (nor re-registered) for the MBT course in 2015-2016.
All Chalmers students registered for the MBT course in 2015 or before (those registered with course code DAT260), and **NOT** registered in 2016.

Content of the long (7.5 HEC) exam. The exam consists of **5** tasks (**tasks 1-5**).

Information on grading scale for the long (7.5 HEC) exam.
Each task is worth 20 points. In order to reach the level to pass with **3** (**G**) you need **at least 50 points** out of the total, and **at least 6 points per task.** To pass with **4** you need **at least 65 points** out of the total, and **at least 8 points per task**.
In order to pass with distinction **5** (**VG**) you need to reach **at least 80 points** out of the total, and you must score **at least 14 points per task**.

IMPORTANT: Note that you should have a minimum number of points in each of the 5 tasks in order to pass the exam, so avoid letting unanswered tasks.

Task 1 – Modeling: State machines

An *Online Social Network (OSN)* is a digital social structure made of users and relationships between users, allowing them to interact with each other through different provided activities or events. We will consider here an OSN called *Caralibro* where users can *post* messages on a public digital space called a *room*. Caralibro has been designed in such a way that interaction is not only strongly encouraged but also somehow mandatory. That means that it is not acceptable to keep posting unless other users having access to the room react to the posts.

1) In order to enforce the above constraint on interactions, the developers of Caralibro have decided to implement the following behavior into the system concerning the posting of a message in the room by a user.
   i) Each user can post up to 3 consecutive messages without getting a comment by some other user. (Other uses may write their own posts, but you are here modeling only 1 user, and what is relevant is other users' comments to the posts of the user you are modeling.)
   ii) If the user gets a comment after a post, then the constraint is "reset", meaning that the user can post again up to 3 consecutive messages without a comment.
   iii) When the user tries to post a new message after 3 posted messages without getting a comment, the post is blocked and the user gets a message saying that he/she will be blocked momentarily for undefined time. The user can post again after he/she is explicitly unblocked by the system, and the constraint (on the 3 consecutive posts) is applied again.

   Your task in this exercise is to **give an FSM to model behavior given by the specification above** (only for one user). **(8 pts)**

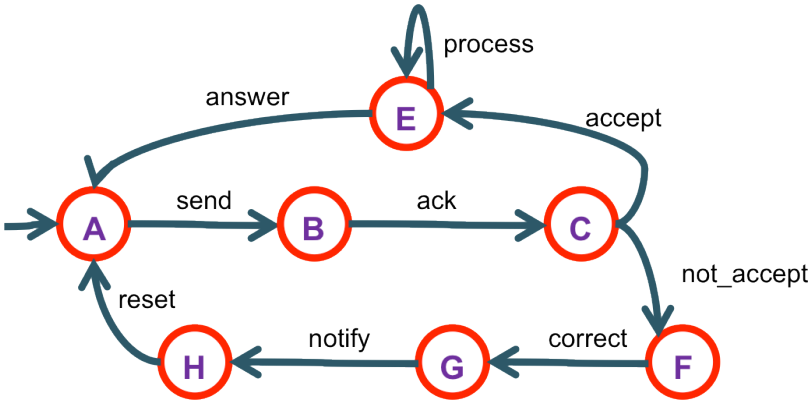
2) The developers of Caralibro realized that the above constraint on reacting to posts was not good enough, as very often a post didn't get a comment for many days. In order to enforce better interactions, they decided to implement the following constraint on posting.
   i) Each user can post up to 3 consecutive messages within 24 hours without getting a comment by some other user. (Other uses may write their own posts, but you are here modeling only 1 user, and what is relevant is other users' comments to the posts of the user you are modeling.)
   ii) At any moment, if the user gets a comment within 1 hour after a post then the constraint is "reset", meaning that the user can post again up to 3 consecutive messages without a comment within 24 hours.
   iii) When the user tries to post a new message after 3 posted messages without getting a comment, the post is not sent and the user is blocked till the timeout of 24 hours expires. The user gets a message saying that he/she will be blocked momentarily.
   iv) The user is unblocked after the 24 hours when the user tries to post a message. The user gets a message saying that he/she can post again, and the constraint (on the 3 consecutive posts within 24 hours, etc.) is applied again.

   Your task in this exercise is to **give an EFSM that models the behavior given by the above specification** (only for one user). **(12 pts)**

Note: Be sure you provide meaningful names for of each action, variable, state, etc., and provide a short explanation of each in case of ambiguity.

Task 2 - Coverage analysis

Let the following FSM represent the model of a SUT:



NOTE: **A** is the initial and the final state.

In what follows there are 10 statements on different *coverage criteria* issues or situations related to the above FSM. **Determine whether the statements are true or false**. **In each case justify your answer** giving clear arguments to defend your judgment. **In case of a false answer give the sequence of actions to be performed to satisfy the corresponding criteria if applicable**. **In case of a true answer, briefly explain why it is the case.** Note that your answer will not be considered complete if you don't justify it. **(20 pts – 2 pts each)**

a) The following are correct test cases achieving a full coverage according to the ***All-states*** *transition-based coverage criteria*:

- send, ack, accept, process, process, answer
- send, ack, not_accept, correct, notify, reset

b) Assuming we want to write only 1 test case aiming at achieving a full coverage according to the ***All-transitions*** *transition-based coverage criteria* in the above FSM, the minimum length of such test (number of actions in the test case) is 10.

c) The following test cases achieve full coverage according to the ***All-loop-free-paths*** *transition-based coverage criteria* in the above FSM:

- send, ack, accept, process, process, process, answer
- send, ack, not_accept, correct, notify, reset

d) The test cases given in exercise c) above achieve full coverage according to the ***All-one-loop-paths*** *transition-based coverage criteria* in the above FSM.

e) It is possible to achieve a full coverage on the above FSM according to the ***All- paths*** *transition-based coverage criteria*.

f) By using *stress testing* on the above model we could generate test cases where a lot of messages are sent ("send" action) overloading the system.

g) By using a *statistical test generation method* on the above FSM, it is possible to cover all transitions with the generated test cases.

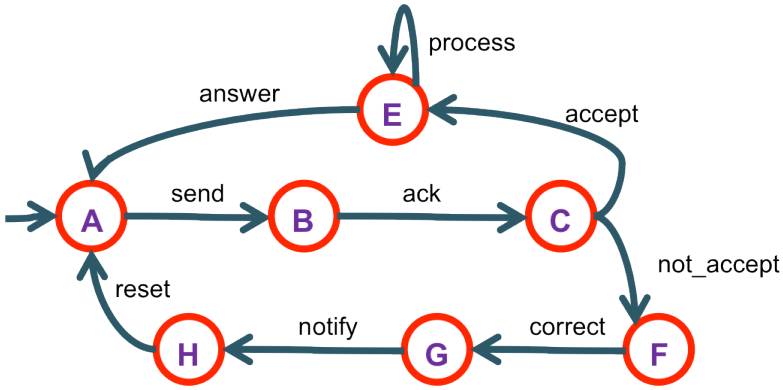
h) The *control-flow oriented* ***Decision/Condition Coverage*** *(D/CC) criteria* is not applicable to the FSM above.

i) By applying *data-coverage criteria* in the above FSM we could get a better analysis as we will have more precise tests that will consider boundary values.

j) From the above FSM it is certainly possible to extract test cases in order to achieve *full statement coverage* in the implementation under test (IUT).

Task 3 - Graph theory and MBT

For the FSM given in the figure, **determine whether the statements are true or false**. **In each case justify your answer** giving clear arguments to defend your judgment, **briefly explaining why it is the case**. Note that your answer will not be considered complete if you don't justify it. (**A** is the initial and the final state.) **(20 pts – 2 pts each)**



a) The FSM is *strongly connected* and *complete* since all the actions are present in the FSM.

b) There is no solution for the *Street Sweeper* problem since the graph cannot be Eulerized.

c) A solution for the problem of *testing combination of actions of length 2* by building the dual graph using the de Bruijn algorithm would give a graph with 9 states and 12 transitions (before the graph is Eulerized).

d) Assuming that each transition represents a task taking 1 minute, the *best time* that can be obtained for transition coverage by using parallelization is 5 minutes and using 2 machines.

e) If we add a transition from state A to state F, and assuming that we have a maximum of 3 machines, the best time to perform transition coverage by using parallelization would be 5 minutes (assuming that each transition represents a task taking 1 minute).

f) The number of branching of a FSM influences the number of machines one could used to perform parallel testing. In the above FSM we have only 1 branching and therefore it doesn't make sense to use more than 2 machines to perform parallel testing.

g) For the FSM given above a *Random Walk algorithm* would never give 100% transition coverage.

h) The above FSM cannot be used as a model for a more detailed EFSM because the Greedy algorithm would not work for extracting test cases.

i) Let's assume we add a probability of 0.1 to the transition labeled with "accept" and a probability of 0.9 to the transition labeled with "not_accept". If we then run 100 times a traversal algorithm to automatically generate test cases from the FSM (all finishing in the final state), most of the test cases will contain the transition labeled with "not_accept".

j) It is possible to combine Markov chain probabilities and graph traversal techniques to get an efficient traversal of the paths in the FSM.

Task 4 – Model-Based Testing (MBT) / ModelJUnit

You will find below 10 statements and situations about different issues related to model-based testing (MBT) and ModelJUnit. **Determine whether the statements are true or false**. **In each case justify your answer** giving clear arguments to defend your judgment **(if the answer is false provide the correct fact, if it is true write a short reason showing you understand why it is the case).** Note that your answer will not be considered complete if you don't justify it.
(20 pts – 2 pts each)

a) A model of the system is not needed in MBT if you want to perform offline testing.

b) ModelJUnit is a tool to help programmers to conceive and build their models for testing, from the implementation (SUT).

c) The most recommendable way to get a model for MBT is to get it from the control flow graph of the implementation (which might be automatically generated).

d) Without a graph traversal algorithm (like for instance the GreedyTester) it is not possible in ModelJUnit to automatically generate tests from the model.

e) ModelJUnit automatically generates test cases to cover those transitions that are not explicit in the model, allowing then the tester to check also those cases that should not pass.

f) An adaptor is needed in ModelJUnit in order to connect the model (EFSM) with the implementation (SUT). This adaptor uses, among other things, *Assert* statements to ensure the model and the SUT agree on the result of certain operations.

g) The *AllRoundTester* algorithm in ModelJUnit works well, and precisely gives a correct state coverage.

h) You need to have knowledge of Java and FSMs in order to apply MBT in industry.

i) In general it is not possible to write an adaptor without knowing the implementation (IUT), or at least its API.

j) In ModelJUnit each action method is responsible, among other things, for checking that any SUT outputs are correct, typically done via JUnit *assert* methods.

Task 5 – Property-based testing and QuickCheck

Assume that you have implemented a Module MultiSet that includes an implementation of *multisets* in Haskell with some additional operations besides the standard ones. (A *multiset* is a generalization of the notion of a set in which members are allowed to appear more than once.) The module introduces a parameterized data type, MultiSet a, such that for every type a, we have the type MultiSet a of multisets of a's (e.g. MultiSet Int is a multiset holding integers). This multiset implementation is pure, i.e. there are no side effects. As a consequence, the functions that manipulate multisets always return new multisets as their result, instead of modifying multisets in place. The interface for this module includes the following functions:

```
member :: (Eq a, Ord a) => a => MultiSet a -> Bool
    – tests if a value exists in the multiset
add   :: (Ord a) => MultiSet a -> a -> MultiSet a
    – adds a new element to the multiset
count  :: (Ord a) => MultiSet a -> a -> Int
    – returns the number of x in the multiset (0 if the element is not in)
union  :: (Ord a) => Multiset a -> MultiSet a -> MultiSet a
    – returns the union of two multisets
sum  :: (Ord a) => Multiset a -> MultiSet a -> MultiSet a
    – returns the sum of two multisets
empty  :: MultiSet a
    – the empty multiset
size   :: MultiSet a -> Int
    – the number of elements, including duplicates, in the multiset
toList   :: MultiSet a -> [a]
    – converts the multiset to a list (with duplicates)
```

Note that there are two kinds of "union" between multisets: union and sum. The union of two multiset is built by taking the maximum count of each element in either multiset, for instance: [1,1] `union` [1,2] == [1,1,2]. On the other hand, the sum of two multisets adds the count of each element, e.g., [1,1] `sum` [1,2] == [1,1,1,2].

You can assume that the type MultiSet a derives the class Eq (if a does).

Provide a solution to the following questions/situations regarding QuickCheck properties for the above module. (20 pts)

a. Write a simple property to verify that the empty multiset is an idempotent of union (the union of any multiset with the empty multiset gives you the original multiset) **(2 pts)**:

```
prop_idem ms  = ...
```

b. Write a property that checks that the sum of two multisets is bigger than the union of the same two multisets. (note that MultiSet a does *not* derive Ord!) **(2 pts)**:

```
prop_sum ms1 ms2 = ...
```

c. The following property checks that no element from a multiset is lost when doing a union:

```
prop_union ms1 ms2 = size ms1 > 0 ==> forAll (elements (toList ms1)) $ \x -> count x ms1
== count x (ms1 `union` ms2)
```

Is the property correct? If not say why and provide a correct property. **(2 pts)**

d. The union of multisets is idempotent (whenever applied to two equal values, it gives that value as the result). Write two QuickCheck properties, the first should check that union is idempotent and the second that sum is not (beware of the empty case!) **(4 pts)**:

```
prop_union_idem ms = ...
prop_sum_idem ms = ...
```

e. Write a simple property that check that if you add an element `x` to a multiset, then the element x should be a member of the resulting multiset **(2 pts)**:

```
prop_add_member ms x = ...
```

f. A programmer wants to check how the `add` operation and the `count` function interact, and for that she writes the following property:

```
prop_add ms a = count a (add a ms) == 1
```

Is the property correct? If not say why and provide a correct property. **(2 pts)**

g. Write a property to test that the multiset union is commutative **(2 pts)**

```
prop_commutativity ms1 ms2 = ...
```

h. A programmer wants to write a property about the size of the union of two multisets. She writes the following property:

```
prop_size ms1 ms2 = size (union ms1 ms2) > size ms1
```

Is the property correct? If not say what is wrong and give a correct property. **(2 pts)**

i. The following property about multisets is not correct since there is a typing problem **(2 pts)**:

```
prop_xxx a ms = member a (sum a ms)
```