Model-Based Testing

(DIT848 / DAT260)

Spring 2014

Lecture 16

Revision

Gerardo Schneider
Department of Computer Science and Engineering
Chalmers | University of Gothenburg

Revision requests…

- Go through a previous exam
- Give more examples of generators in QuickCheck

Exam MBT Disclaimer!

- Note that the following is only a sample of a previous exam!
- The precise content or format of the incoming exam might be slightly different!

Exam MBT (General issues)

- ALLOWED AID:
  - Books on testing
  - All lecture notes (including printouts of lectures' slides)
  - Students own notes
  - English dictionary
  - NOT ALLOWED: Any form of electronic device (dictionaries, agendas, computers, mobile phones, etc)

Exam MBT (General issues)

- 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 below "ABOUT THE FORMAT OF THE EXAM".

Exam MBT (General issues)

- ABOUT THE FORMAT OF THE EXAM:
  - The exam consists of 5 tasks, each one concerned with a specific part of the course content.
  - 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 per task in order to pass, so avoid letting unanswered tasks.

Exam MBT – May 21, 2012

- MBT-exam-2012-05-21.pdf

Task 1 -Test in general
Part 1

Solution

1. F – testing is always dynamic

2. T

3. F – debugging is testing + correcting the errors

4. F – This is the less advisable way to do it, according to many experts

5. F – No, you don't need a full implementation (you might use some mock code)

10 min

Task 1 -Test in general
Part 2

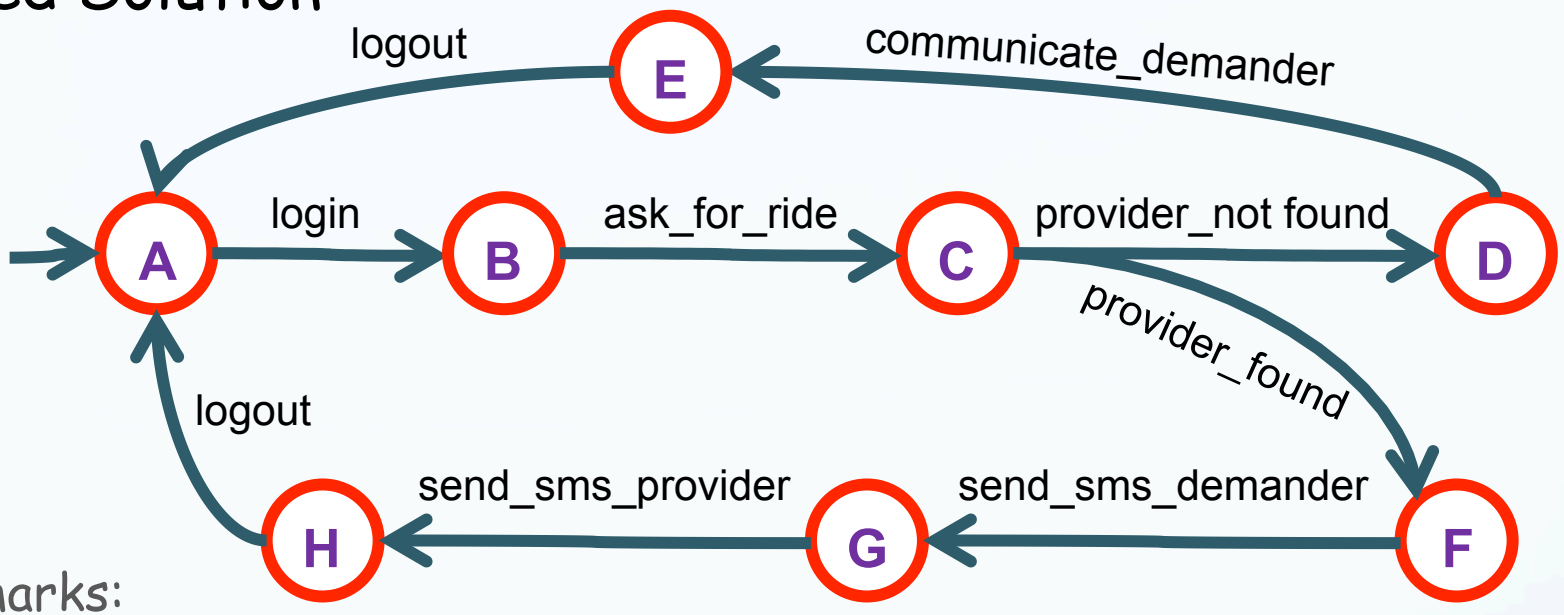
Solution:

1. Acceptance test (g) (also during system test - e)
2. stress/system test (e) and also acceptance (g)
3. Combination of coverage analysis (c) and unit tests (b)
4. timing response test (system test - e)
5. configuration test (system test - e)

10 min

Task 2 -State Machines Part 1

Proposed Solution



Some remarks:

- Many other solutions depending on how much do you abstract
  - A "good" solution should be abstract enough as to capture the informal description (but not too much as to be useless)
- "logout" could be eliminated (as it is automatic)
- No check on whether login is correct or not (not in the specification)
- Implicit loop in state "C" on "look_for_provider"

7 min

Task 2 -State Machines Part 2

Proposed Solution

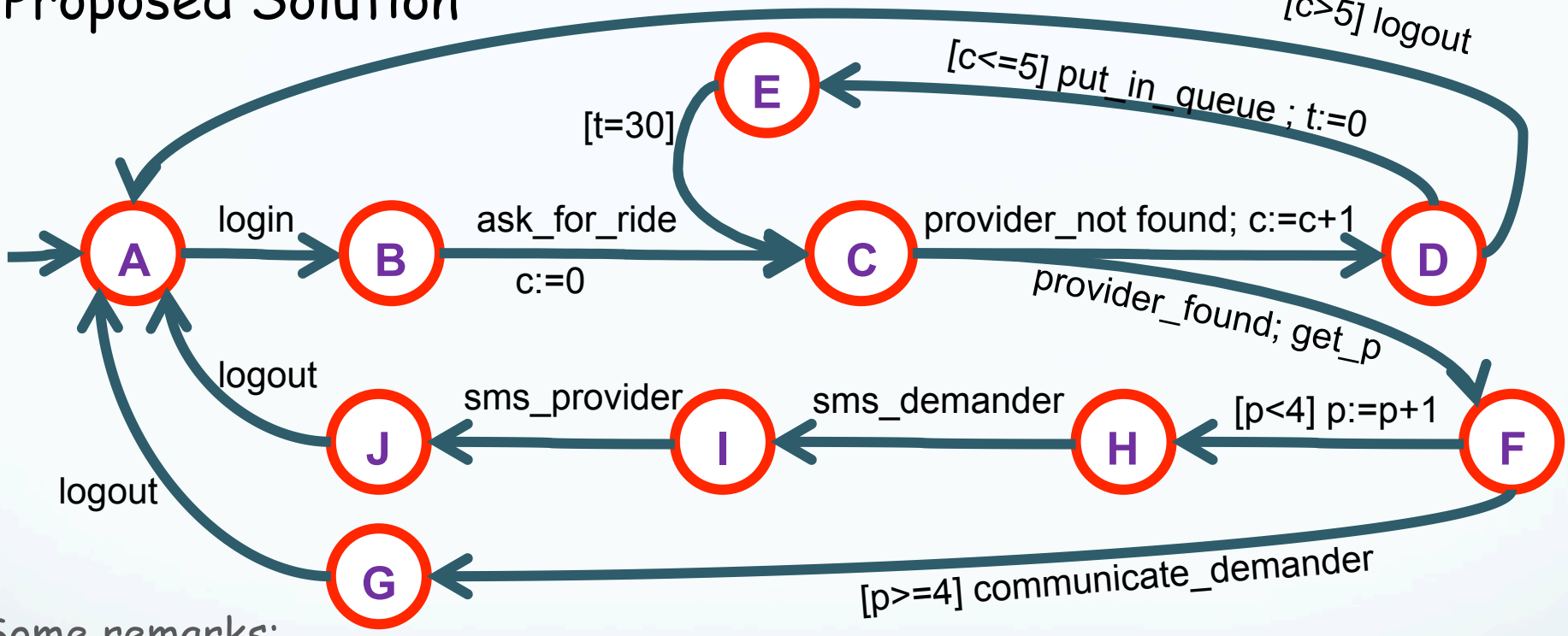
- Test cases you can extract:
  1. After login if there is provider then the demander gets an sms indicating that.
  2. If no provider exists for that ride then the user is logged out after getting a notification.

- Test cases you cannot extract:
  1. If a provider does exist for the ride, the user may still not get the guarantee of a ride due to overbooking.
  2. Any timing constraints in what concerns how much time to wait for getting a confirmation of a ride.

5 min

Task 2 -State Machines Part 3

Proposed Solution



Some remarks:

- Brackets ("[.]") are used as a short for "If ... then ..."
- t: timer; c: number of times a demander may request a ride; p: nr of passengers (stored in the DB; get using "get_p")
- Assumption: the timer is automatically incremented (implicit loop in state E)

10 min

Task 3 –White box testing and coverage Part 1

Solution

a. a-b-g (not finishing in the final state though -> a-c-d-e)

b. (Considering the state as being between the transitions)
s1: d-a, d-e
s2: a-b, a-c
s3: c-d, g-d
s4: e-g, e-f, b-g, b-f, f-f, f-g

c. e,
a-b

NOTE: The definition doesn't allow to repeat a configuration (state) so any other sequence is not included as they must pass through S1

d. Add to the above visiting "f" too

e. a-b-g-d-e-f,
a-c-d-e

15 min

Task 3 –White box testing and coverage Part 2

Solution

a. Deterministic (i), initially connected (ii), minimal (iii), strongly connected (iv)

b. Add copies of transitions a, g, d
(e.g: a-c-d-e-f-g-d'-a'-b-g'-d'')

c. Transform the graph using de Brujin's algorithm (dual graph) and then "Eulerize" it (see lecture 7)

15 min

Task 4 –MBT / ModelJUnit

Solution

1. F – you should aim at least at a 100% transition coverage
2. F – You might use transformation and adaptation.
3. F – you might need to change the code
4. F – this is the case for the transformation, not the adaptation
5. T
6. T
7. T
8. T
9. F – It doesn't as there might be many branches in the SUT abstracted away in the EFSM
10. F – Transition-based is control oriented, while pre/post is data-oriented.

15 min

Task 5 – Property-based test. and QuickCheck Part 1

Solution

a. prop_delete1 x t =
delete x (delete x t) == delete x t

b. prop_delete2 x t = not (member x t) ==>
flatten (delete x (insert x t)) == flatten t
(Note that the it is not necessarily true that you get the same tree!)

c. prop_delete3 x t = (member x t) ==>
(flatten (insert x (delete x t)) == flatten t)
(Note that the it is not necessarily true that you get the same tree!)

d. (The statement should be read as "Write a property that checks that 2 BSTs are not equal if they don't contain the same elements.")
prop_equal t1 t2 =
not (flatten t1 == flatten t2) ==> t1 /= t2

20 min

Task 5 – Property-based test. and QuickCheck Part 2

Solution

a. F – you write properties, not necessarily a full model.

b. T

c. F – There is no guarantee of getting the same tree. You should write:
prop_merge1 x y t1 t2 = flatten (merge (insert x t1) (insert y t2)) == flatten (insert x (insert y (merge t1 t2)))

d. F - The problem is that the symbols < and > are interchanged. You should make the following change:
"&& all (<y) (flatten lt) && all (>y) (flatten rt)"

20 min

2nd request: Generators in QuickCheck

Write a generator that generates non-empty lists of integers of arbitrary size satisfying the following constraints:

1. The list is sorted.
2. The first element of the list is a random number between 1 and 100.
3. Each element of the list is randomly generated in such a way that the element is bigger than the previous one and it differs at most in 100 from the previous one.

That is, if [a1, a2, …, an] is a list generated according to the above specification, then it should satisfy that:

0< a1 <= 100, and
0< ai+1 - ai <= 100 (for 0<i<n-1)

The following is a valid example of a generated list: [87, 122, 123, 222]. On the other hand, the lists [2, 104, 105, 200] , [105, 106, 110, 201] and [77, 56, 139, 150] are not valid.

* Task 2-2) of exam June 1st, 2013 (MBT-exam-2013-06-1.pdf)

20 min

Generators in QuickCheck
Proposed Solution (1)

```
import Test.QuickCheck
import Data.List


genListSorted :: Gen [Int]
genListSorted = do
    intlist <- listOf1 ( elements [1..100] )
    return ( tail ( map sum ( inits intlist ) ) )


main = sample genListSorted
```

* Solution by Grégoire Détrez

Generators in QuickCheck
Proposed Solution (2)

```
import Test.QuickCheck
import Test.QuickCheck.Gen
size = 10
arb :: Gen [Int]
arb = do
 x <- choose (1,100)
 l <- choose (1,size)
 xs <- arb' x
 return $ take l xs
```

```
arb' y = do
 z <- choose (y+1, y+100)
 ys <- arb' z
 return (y:ys)
```

* Solution by Magnus Ågren

Exam

- **May 28, at 08:30** (Lindholmen)

- Remember you need to have passed all the assignments in order to be able to take the exam!
  - Talk with Grégoire if for some reason you don't satisfy the above requirement