Sample Exam Testing Techniques

To make this exam, you are allowed to have a copy of the lecture notes and the slides. Nothing else. Indicate your name each seperate page that you hand in. You can earn 90 points with the following 4 exercises. The final score is equal to (your score + 10) divided by 10. We wish you a lot of success!

The division of the points is as follows:

exercise 1: 5
exercise 2: 30
exercise 3: 30
exercise 4: 25

What is testing?

Exercise 1 Extended static analysis (type checking, checking class invariants, etc) is sometimes called *static testing*.

1. Give an important difference between static analysis and testing (in the sense of this course).

2. Give an important similarity between static analysis and testing (in the sense of this course).

Classical testing techniques.

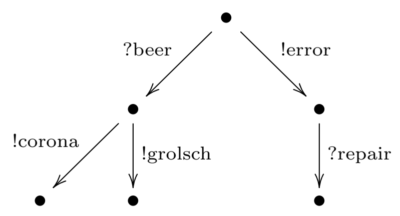
Exercise 2 The C function *saved* calculates the total sum of money that results after saving for years a certain fixed amount per year with a fixed interest rate. More precisely, on Januari 1st of each year a certain amount `amount` is put in a bank account. Each year on December 31st, the bank sents out an account summary indicating the total amount of money in the account. The bank uses the function `saved` below to compute the total amount of money after `years` years of saving.

The three inputs must be greater than or equal to 0; the output is a real value (double in C).

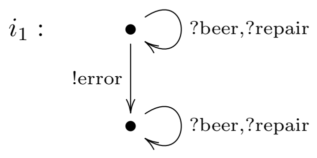
```
double saved(int amount, double rate, int years)
{
  int j;
  double s;

  j = 1;
  s = amount * (1.0 + rate/100);

  do
  {
    s = (s + amount) * (1.0 + rate/100.0);
    j = j+1;
  }
  while (j<years);

  return s;
}
```

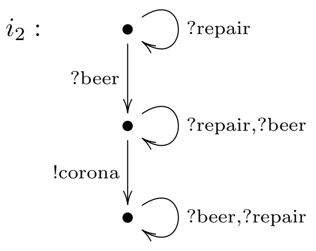
1. Give a formal specification for the function `saved` as pre- and postconditions, based on the informal description.
2. Use the equivalence partitioning technique to divide the input suitable equivalence classes.
3. Give a test set that covers all equivalence classes.
4. Extend the test set following the principle of boundary value analysis.
5. Which test cases from your test set in 4 does the implementation above pass?

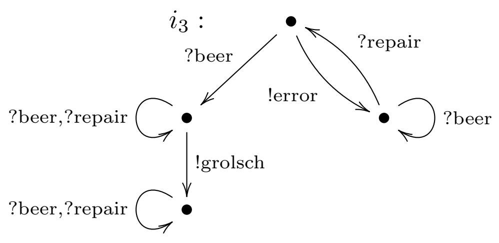
Exercise 3 (Test derivation for ioco) A company that produces beer tenders provides the following specification S, with $L_I = \{?\text{beer}, ?\text{repair}\}$ and $L_U = \{!\text{grolsch}, !\text{corona}, !\text{error}\}$.



1. Determine the quiescent states in S and construct the corresponding QTS.
2. Which of the following implementations i_1, i_2 and i_3 conform to S w.r.t. ioco? For those that are incorrect, provide a test case that exhibits the error.







3. Give an implementation IOLTS i_4 that is ioco-correct with respect to S and which is able to provide a corona after a repair.

Exercise 4 (True or false?) Are the following statements true? If so, give a proof otherwise give a counter example. Here, p, q are IOLTSs, p is input-enabled and u is an LTS. Do we have

1. p **ioco** $q \implies p \leq_{rf} q$
2. If $\mathit{Traces}(p) = \mathit{Traces}(q)$, then $\mathit{Straces}(p) = \mathit{Straces}(q)$.
3. If u **after** σ **refuses** A and $B \subseteq A$ then u **after** σ **refuses** B. [1]
4. If u **after** σ **refuses** A and $B \supseteq A$ then u **after** σ **refuses** B.

[1]Recall that s **after** σ **refuses** A is defined for a state s. We say that u **after** σ **refuses** A if the initial state s_0 of u meets s_0 **after** σ **refuses** A.