TRIAL EXAM A
Software Engineering using Formal Methods
TDA293 / DIT270

also serving as additional training material for the course
Formal Methods for Software Development, TDA294/DIT271

Assignment 1 PROMELA (12p)

In this assignment your task is to write a Promela program to solve the 15 puzzle. Typically the 15 puzzle is played with a little plastic board with small pieces numbered from 1 to 15, with one slot empty. The only possible move is to slide a piece into the empty slot. The goal is to reach the final state which looks as follows:

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

Write a PROMELA program which nonderministically chooses a correct move on the puzzle board. After each move, an assertion should check that the final goal state is *not* fulfilled. (This way, a violating path found by SPIN will correspond to a solution of the puzzle.) The move and assertion checking should be done in an infinite loop.

Hint: You can assume that the board is initialized so that it contains all the numbers from 0 to 15. The number 0 is used to represent the free slot. You can also assume that the free slot is in position 0,0 on the board when the game starts. The other numbers, 1 to 15, you can assume to be distributed arbitrarily. Again, this initialization you can *assume*, so do not waste time writing your own code for the initialisation. Also, recall that plain 2-dimensional arrays are not supported by PROMELA. One (but not the only) possibility is that you code everything in a single dimensional field of size 16.

Assignment 2 Linear Temporal Logic (LTL) (7p)

Are the following LTL formulas valid, unsatisfiable or neither? In case of valid or unsatisfiable, it is enough to write "valid" or "unsatisfiable", respectively. In case of neither, provide one satisfying and one non-satisfying run. The characters p and q are propositional variables.

1. $(\Diamond\Box p \wedge \Diamond\Box q) \rightarrow \Diamond\Box(p \wedge q)$

2. $(\Box\Diamond p \wedge \Box\Diamond q) \rightarrow \Box\Diamond(p \wedge q)$

3. $(\Box\Diamond p) \vee (\Box\Diamond\neg p)$

4. $(\Box(p \rightarrow \Diamond q)) \rightarrow (p \rightarrow \Box\Diamond q)$

5. $(\Box\text{false}) \rightarrow (\Diamond\text{false})$

Assignment 3 (First-Order Sequent Calculus) (8p)

Your task is to build a proof for the following sequent:

$\exists\ x; \forall\ y; p(y, x),$
$\forall\ x; \forall\ y; (p(x, y) \rightarrow q(y, x)),$
$\Rightarrow$
$\exists\ x; q(x, x)$

You are only allowed to use the rules presented in the lecture.

Provide the name of each rule used in your proof as well as the resulting sequent. (If you do not use a tree notation, make clear on which sequent you have applied the rule.) Every time you use a rule that has a side condition, justify that the side condition is fulfilled.
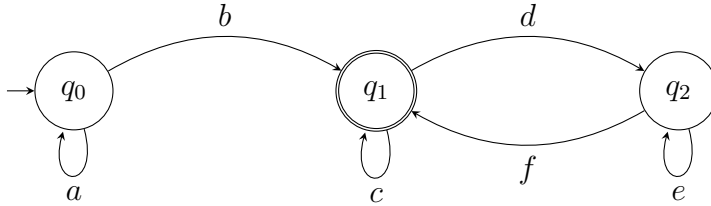
Hint: You may abbreviate formulas, but only if you clearly describe your abbreviations.

Assignment 4 (Büchi Automata and Model Checking) (9p)

(a) [3p]
Give the ω expression describing the language accepted by the following Büchi automaton:
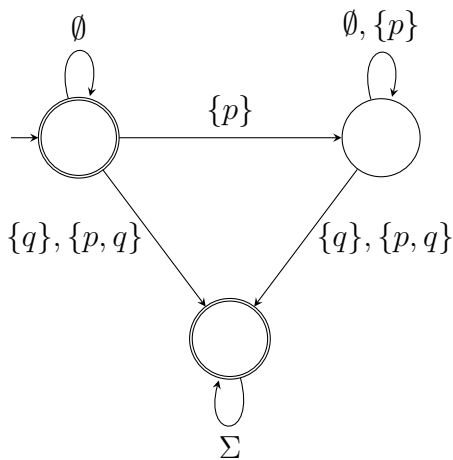

(b) [3p]
Give a Büchi automaton accepting exactly the ω-language given by the following ω-expression:

$$(a^*b^*c)^\omega$$

(c) [3p]
Give an LTL formula that is satisfied by exactly those runs which are accepted by the following Büchi automaton:

$\Sigma := 2^{\{p,q\}}$

Assignment 5 (Java Modeling Language) (12p)

We consider a small part of an e-book application. Each book is represented by an instance of a class `Book`, which is sketched below (see next page). A book always has an ISBN number and a number of pages. A book has a current page and a number of bookmarks (stored in an array). A reader can flip a page, jump to a marked page, and mark the current page. In the following we describe the properties and constraints for the books:

1. No two books have the same ISBN.

2. A book has at least one page.

3. Pages are numbered from 1 to the last page, and the current page must be within this range.

4. The bookmarks are initialized to the value 0, and can be assigned to valid page numbers.

5. There is no duplicate in bookmarks (i.e., no two assigned bookmarks point to the same page).

6. A reader can try to flip a page (i.e., `flip`) to move to the next page. If he/she is already at the last page, flipping throws a `BookException`, and keeps the book (i.e., ISBN and number of pages) and the current reading state (i.e., current page and all bookmarks) unchanged.

7. A reader can jump directly to a marked page (i.e., `jump`) by providing the array index of the bookmark.

8. A reader can mark the current page (i.e., `mark`) by storing the page number in the bookmark array, without changing existing bookmarks other than the location that is used for the new one. (*Hint*: We do not care about which array location is used for the new bookmark.)

Your assignment is to specify the `Book` class using JML to reflect the above properties plus any other properties that you deem necessary. In particular, specify:

- Class invariants to reflect the stated consistency properties (*Hint:* recall the difference between static and instance invariants),

- Method specifications for the `flip`, `jump`, and `mark` methods.

- Assignable clause for the methods.

You only have to *specify* the class. No implementation (neither for the constructor nor for any methods) is expected from you. Keep the pre- and postconditions as minimal as possible. In particular, avoid stating postconditions that can be inferred from the other method conditions or the class invariant.

```java
public class Book {
    private int lastPage;
    private int currentPage;
    private int isbn;
    private int[] bookmarks = new int[10];


    public Book(int numberOfPages, int isbn) {...}

    public void flip() {...}

    public void jump(int index) {...}

    public void mark() {...}
}
```

Assignment 6 (Contract Fulfillment) (8p)

Look at the following JML contract for a method **int m()** of a class **C**:

```
public int x, y;

/*@ public normal_behavior
  @  requires x>=0 && y>=0;
  @  ensures  \result == (x>=y ? x : y);
  @*/
public int m() { ... }
```

"(x>=y ? x : y)" is a conditional expression, returning x if x>=y, and y otherwise.

(a) [5p]
   Which of the following four implementations of **m()** respect the contract? Justify your answer in each case. You don't need to give a formal correctness proof. You can assume that the integers are unbound and no overflow occurs.

```
public int m() { // Implementation I1
  if (y >= 0) {
    if (x >= y) { return x;
    } else { return y; }
  } else return 0;
}


public int m() { // Implementation I2
  x = x - y;
  if (x >= 0) { return x + y;
  } else { return y; }
}


public int m() { // Implementation I3
  int t = x - y;
  if (t >= 0) { return t + y;
  } else { return y; }
}


public int m() { // Implementation I4
  if (x >= y) { return x;
  } else {
    if (y >= 0) { return y;
    } else { return x++; }
  }
}
```

(b) [3p]
   Choose one of the implementations of **m()** that does not respect its contract, and suggest a patch of the *contract* such that the (unchanged) implementation respects that contract.

Assignment 7 (Proof Obligations) (4p)

Take the *original* JML contract from the previous problem (i.e., not the one you wrote
to answer question 6b). The following dynamic logic formula is the proof obligation,
generated by KeY, for the correctness of the implementation of `m()` with respect to
that contract. However, we have left out (indicated by the empty rectangle) some part
of the DL formula. Your task is to write the part of the proof obligation formula which
should be there instead of the rectangle, such that the whole formula is the right proof
obligation generated from the given contract.

```
    wellFormed(heap)
  & !self = null
  & self.<created> = TRUE
  & C::exactInstance(self) = TRUE
  & measuredByEmpty
  & (self.x >= 0 & self.y >= 0 & self.<inv>)
 ->
```

```
            (  exc = null
             &   result
               = \if (self.x >= self.y)  \then (self.x)  \else (self.y)
             & self.<inv>
             & \forall Field f;
                 \forall java.lang.Object o;
                   (  (o, f) \in allLocs
                    | !o = null
                    & !o.<created>@heapAtPre = TRUE
                    | o.f = o.f@heapAtPre))
```

(total 60p)