Additional Exercises

TECHNISCHE UNIVERSITÄT DARMSTADT

Problem 1 First-Order Logics

Problem 1.1 FOL Calculus

Prove the validity of the following untyped first-order formula

$$\begin{array}{l}(\forall\, x; \forall\, y; \forall\, z; (p(x,y) \land p(y,z) \rightarrow p(x,z)) \land \\ \ \forall\, x; \forall\, y; (p(x,y) \rightarrow \neg p(y,x)) \land \\ \ \exists\, x; (p(a,x) \land p(x,b)) \\ ) \rightarrow \neg p(b,a)\end{array}$$

(with a, b being constant symbols and p a binary predicate symbol) using the sequent calculus. 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 and make clear on which sequent you have applied the rule. For the quantifier rules, justify that the side conditions on substituted terms are fulfilled.

To save space and time you can introduce abbreviations for some formulas.

Hint: If you want to generate more assignments of this type, try to formalize a (mathematical) property e.g. about relations in FOL and then verify it. You can check your solution by writing a .key file with your problem and by replaying your proof within KeY.

Problem 1.2 FOL Formalisation

People from the town of Liarsville always tell lies. Of the following people below, only one is not from Liarsville. Can you tell which one is the outsider based on the following statements made?

Mr. Applebee: I am very honest.

Mrs. Beatle: Dr. Doodle is not from Liarsville.

Ms. Casey: I am a liar.

Dr. Doodle: Mrs. Beatle does not lie.

Mr. Eastwood: I am from the East.

Formalize the above riddle as a set of axioms Ax as well as your solution Sol as a formula in first-order logic (with sort Person as only additional sort next to $\top$) such that $(\bigwedge_{\phi \in Ax} \phi) \rightarrow Sol$ is valid if and only if your solution is correct.

Try to prove your solution using KeY.

Problem 2 Dynamic Logics

Problem 2.1 Formalisation

Formalise the following statement in DL:

Let `A` be a class with an integer instance field `a` and let `o` be a program variable of type `A`.

- Execution of the program `p` terminates and in its final state the following holds: If for all created objects of type `A` their field a has a positive value then program `q` terminates.

Problem 2.2

Are the following rules sound and/or complete? Give a counterexample if the rule is unsound.

a)

$$\frac{\Gamma, A \Rightarrow \Delta}{\Gamma, A \wedge (B \vee C) \Rightarrow \Delta}$$

b)

$$\frac{\Gamma, A \wedge B \Rightarrow \Delta}{\Gamma, A \wedge (B \vee C) \Rightarrow \Delta}$$

Problem 2.3

Simplify the following formula (P is a binary predicate) as far as possible by only using the update simplification rules from the lecture. Provide each intermediate step.

$\{\mathtt{i} := \mathtt{j}\}\{\mathtt{j} := \mathtt{j} + \mathtt{i} \| \mathtt{i} := 4\}\mathtt{P}(\mathtt{i}, \mathtt{j})$

Problem 3 Java Modelling Language

All specifications have to be given in the Java Modeling Language (JML).

Consider the Java classes `Interval` and `IntervalSeq`:

```
public class Interval {
    private final int start, end;

    public Interval(int start, int end) {
        this.start = start;
        this.end = end;
    }

    public int getStart() {
        return start;
    }

    public int getEnd() {
        return end;
    }

    public boolean contains(int e) {
      // ...
    }
}

/** Class to represent sequence of intervals. */
public class IntervalSeq {

    protected int size = 0;
    protected Interval[] contents = new Interval[1000];

    /** Insert a new element in the sequence;
     *  it is not specified in which place the element will be inserted
     */
    public void insert(Interval iv) {
        // ...
    }
}
```

In the following, observe the usual restrictions under which Java elements can be used in JML specifications.

Problem 3.1

Specify in JML that for class `Interval` the value returned by `getEnd()` is always greater than the value returned by method `getStart()`.

Problem 3.2

Method `contains` shall return `true` if and only if the given element lies within the interval. An interval contains all elements which are between `start` (inclusive) and `end` (exclusive). The method terminates always normally and is side-effect-free.

Problem 3.3

In class `IntervalSeq`, the field `size` holds the number of `Interval` objects which have yet been inserted into the `IntervalSeq` object. All inserted `Interval` objects are stored contiguously from the beginning of the array. The remaining cells of the array are `null`.

Augment class `IntervalSeq` with a JML specification stating the following:

- The `size` field is never negative and smaller than the length of the `contents` array.
- The cells of the array `contents` which are stored below index `size` are never `null`.
- Each `IntervalSeq` object has a different `contents` array.
- If the value of `size` is strictly smaller than `contents.length`, then all of the following must hold:
  - `insert` terminates normally
  - `insert` increases `size` by one
  - After `insert(iv)`, the interval `iv` is stored in `contents` at some index `i` below `size`. Below index `i`, the array `contents` is unchanged. The elements stored in between `i` and `size` were shifted one index upwards (as compared to the old `contents`).
- If the `size` has reached `contents.length`, `insert` will throw an `IndexOutOfBoundsException`.
- Provide also the `assignable` clause as precise as possible.

Problem 4 Loop Invariants

Problem 4.1

The file `Snippet.java` contains a method with a loop. Provide a loop invariant, decreases term (variant) and assignable clause that are strong enough to prove the method contract.

Hint: `\old(e)` can be used in an invariant. It evaluates `e` always in the prestate of the method.

Creating your own examples: If you want to exercise more loops, write your own small examples and use KeY to ensure the correctness of your solution.

Problem 4.2

Given the following sequent with `i` denoting an integer typed program variable:

$$\Gamma, \mathtt{i} > 0 \Rightarrow \{\mathtt{i} := 10\}[\mathtt{while\ (i>0)\ \{i=\ i\ -1;\}}]\mathtt{i} \doteq 0, \Delta$$

Let $\mathtt{i} \geq 0$ be the invariant. Give the sequent for the preserves case of the improved invariant, as it looks directly after application of the loop invariant rule.

Problem 4.3 Recursion & Linked Data Structure

File `Tree` contains an implementation of a *binary* tree data structure, i.e., both children are either null or non-null. Specify all methods as complete as possible (incl. assignable). Specify as part of the invariant that the tree is actually a tree (i.e., no cycles). In case of recursion specify also a `measured_by` clause. Provide accessible clauses for the invariant and all methods.

Hint: Proving this specification is hard and requires some non-trivial interaction. So just check your spec against the provided solution and remember there is more than one way to express the same property. In doubt, use the forum to get opinions on your solution.

Problem 4.4

Explain in your own words under which circumstances and why using a dependency contract might ease verification that a method *m* preserves the invariant of its class if an accessible clause for the invariant is given.