Homework 8

Please submit your solution via email to the instructor with CC to ly603@nyu.edu.

The deadline for Homework 8 is April 8.

Problem 1 Dafny (3 Points)

Follow the instructions on the course web site to download and install Dafny. Get yourself familiar with the language and verifier.

Problem 2 Tax Payers (22 Points + 5 Bonus Points)

Before you start, you should read the hints at the end of this exercise.

Download the file `Taxpayer.dfy` from the course webpage. This Dafny class is used in an information system at the tax office to record information about persons.

(1) The tax office wants to ensure that the following properties hold:

   (a) Persons are either male or female.

   (b) Persons can only marry to people of the opposite sex. (A bit outdated maybe, but makes for interesting properties to specify).

   (c) A very obvious property, but easy to overlook: if person x is married to person y, then person y should of course also be married to person x.

   Your assignment is to

   - add class invariants to the code to express the properties mentioned above plus any other sensible properties you can think of;
   - use the Dafny program verifier to detect possible violations, which are reported as errors;
   - for the methods that Dafny complains about, fix them by
     - correcting the code, or
     - adding a missing modifies clause for the method, or
     - adding a (sensible) precondition for the method, using a requires clause, or
     - sometimes, adding a class invariant to record another property can help in verification. All class invariants should go into the predicate `ClassInvariantsHold`.

   In the end, Dafny should run without any complaints, meaning that it has succeeded in verifying that the code meets the formal specification. **(8 Points)**

(2) The tax system uses an income tax allowance:

(a) Every person has an income tax allowance, over which means they do not have to pay tax over the first part of their income. There is a default tax allowance of \$5,000.

(b) Married persons can pool their tax allowance, as long as the sum of their tax allowances remains the same. For example, the wife could have a tax allowance of \$10,000, and the husband a tax allowance of 0. Make sure that the tax allowances do not become negative.

Add class invariants that express these constraints, and if necessary fix/improve the code to ensure that they are not violated. **(6 Points)**

(3) The new government introduces a measure that people aged 65 and over have a higher tax allowance, of \$7,000. The rules for pooling tax allowances remain the same. Add invariants that express these constraints, and if necessary fix/improve the code to ensure that they are not violated. **(6 Points)**

(4) *WARNING: things may get a bit hairy here, so don't be too depressed if you don't manage to finish the bonus exercise.*

Legislation has passed a bill that introduces a new tax measure giving anyone with underage children an additional tax allowance of \$1,000. The taxpayer class is extended with an additional field to keep track of the youngest child

```
var youngestChild : Taxpayer; // youngest child of this person
```

and the following three lines are added to the constructor

```
method Init(isBoy: bool, mum : Taxpayer, dad : Taxpayer) {
    ...
    youngestChild := null; // NEW
    if(mum != null) mum.youngestChild := this; // NEW
    if(dad != null) dad.youngestChild := this; // NEW
}
```

Add the lines above to your code and add class invariant(s) to express the new government policy above. Run Dafny and fix the code to ensure that the new policy is ensured. **(5 Bonus Points)**

Hints on How to Use Dafny

Some hints to keep you out of troubleand help you use Dafny efficiently:

- Don't use method invocations inside methods for this exercise. Though, it might be useful to define auxiliary functions for the later parts.
- If you are using the tool on the command line and it complains about an error, it may produce a lot of feedback that might be hard to interpret. The crucial information to look at is

  - the line and column numbers saying which property is violated, and
  - the line and column numbers saying where the error occurs.

  Beware that calling a method on some object may break the invariant of another object; for instance, in the taxpayer exercise, calling a method on a Taxpayer may break the invariant of his or her spouse. So make sure that after each method call, the invariants of all modified objects are reestablished.

- The code of the methods `marry` and `divorce` in the class `Taxpayer` is not correct. Dafny should detect this and complain about these methods once you have added your invariants. You will have to add a few lines of code to fix these methods, and possibly also add a precondition by means of a requires clause. Also, the modifies clauses are missing and you need to add them.

- It is easiest to introduce one invariant at a time, then fix the errors this brings to light, and only then move on to the next one.

- If you introduce an if-then-else statement in a method, you can check one branch at a time by adding **`assume false;`** to the other branch. Just don't forget to remove all assume statements in the end.

- If the tool produces some warnings, i.e., if the tool thinks that some property is not satisfied by the code, this can have several causes:

  - There is an error in the code, which results in a violation. For example, an assignment to some field may be missing.
  - There is an error in your specification. For example, maybe you have written "and" in a specification where you meant "or", maybe you have written `age > 18` when you meant `age >= 18` in some constraint.
  - There may be some properties missing that are needed by the tool for the verification. Note that the tool does not know any of the things that may be obvious to you – for example, that if some person x is married to y then y is also married to x – and such properties may be needed for verification.
  - In general, a warning might be caused by the fact that some property is too difficult for the theorem prover to proof: an automated theorem prover can only ever prove properties of a certain, limited complexity. However, for the exercises here you should not run into this problem.

  To stop the tool from complaining, you can

  - correct the program; for the exercise here this should only involve adding some simple assignments to the offending method; for the bonus exercise these assignments might be a bit more complex.
  - correct the specification;
  - specify some additional properties, either as class invariants; method preconditions; or frame conditions.