# Formal Methods in Software Engineering
# Assignment 1 – *Alloy*

Due date: Feb. 6th, 2022, 11.59 pm.

Please upload into Teams a .zip file containing one .als file and one pdf file. The system will not accept submissions beyond 11.59pm and any emailed submissions will be ignored.

This assignment is about modeling a car rental system using Alloy. Create a .als file, and include in it the following code:

```
// The following signature models the set of all cars
  sig Car {  }
// Following signature models the current status of all cars
  one sig Rentals {
    available : set Car, // set of available cars, ready to be rented
    rented : set Car, // set of cars currently on rent
  }
```

Throughout this writeup, whenever we say that a car is in a state *st*, where *st* is `available` or `rented`, we mean that this car is an element of the corresponding set `Rentals.`*st*. In all "run" statements throughout the assignment, use a size-bound of 8.

**Problem 1.** Consider the following two properties

P1 : At any instant, every car is either `available` or `rented`

P2 : There is always at least one *available* car

Create a predicate `badInstances` that looks for instances that do *not* satisfy one or both of the properties listed above, and a "run" statement corresponding to this predicate. When executed, this "run" statement should return all the bad instances.

**Problem 2.** Add facts to ensure that all instances satisfy properties P1 and P2. Also, include a predicate "`pred Show {}`", and a "run" statement corresponding to this predicate. Make sure your system is not over-constrained. That is, *all* instances that satisfy the properties P1,P2 should get enumerated. Also, `run badInstances` should return no instances with the facts included, and should return all bad instances if the facts are commented out.

**Problem 3.** In this part we will add two "operations" to the model.

1. Model a *rent* operation as a predicate `rent`, taking parameters `toRent:Car`, `newAvail:Rentals -> set Car` and `newRented:Rentals -> set Car`. Write the predicate such that when it is run, Alloy will find solutions of the form (instance *i*, `toRent`, `newAvail`,

`newRented`, such that the last two arguments of the predicate represent the result of car `toRent` going from *available* state in instance $i$ to the *rented* state, if `toRent` is in the *available* state in instance $i$; otherwise, the last two arguments need to indicate no change from the instance $i$.

2. In the same way as above, model a *return* operation as a predicate `return`, taking the same parameters as the predicate `rent` (except that `toRent` is renamed as `toReturn`), and having the obvious meaning.

Some requirements from your solution are:

- Include a `run` command for each of the two operations above. (That is, you should let Alloy find solutions to the parameters required by the operations.)
- Each solution found by Alloy for an operation above intuitively represents an original instance (before the operation) as well as an updated instance (after the operation). Write each of your two operations in a way that the updated instances always satisfy properties P1,P2.
- Don't directly check in your operations whether the updated instance satisfies P1,P2. Rather, come up with and check suitable *pre-conditions* on the *original instance* and on the *first argument* to each operation such that *updated instance* (represented by the remaining arguments) is guaranteed to satisfy P1,P2.

**Problem 4.**
For each operation $X$ that you wrote above (i.e., $X$ is `rent` or `return`), write a checking predicate `findBugsIn`$X$, which invokes $X$ with an arbitrary set of arguments of the respective types and then checks whether the updated instance fails to satisfy any of the properties P1,P2. (These checking predicates should generate no solutions when run.) Each `findBugsIn`$X$ should take the same set of parameters as the corresponding $X$.

Your single .als file should contain answers to all the problems above. Use meaningful names for the various identifiers that you introduce. Write a comment for each construct you write, explaining in a sentence or two what it does, the meaning of its formal parameters, etc.

**Problem 5.** Assume a domain size of 1 for `Rentals` and 3 for `Car`. Using the procedure discussed in class for translating Alloy formulas to propositional formulas, provide the following propositional formulas:

1. A formula that checks the set of all cars is partitioned into the sets `Rentals.available` and `Rentals.rented`.
2. A formula that checks that there is at least one car in `Rentals.available`

For each part above, also write the matrix or formula for each sub-expression of the full expression, as we have done in the lecture slides. Attempt this entire problem on paper and scan into a pdf file.