Home Work 3

Goal: to improve writing simple TLA+ State Machine specification

A State Machine specification uses

```
Spec = Init /\ [][Next]_vars
```

.

where `Next` is the set of steps that can be made and `vars` is a sequence of variables that can be changed.

1 Card access Spec

A House consists of a set of Rooms and Doors along with a function mapping the doors to the rooms they connect. You cannot accurately or convincingly model such an abstract specification that copes with any house layout in one step. You will need to start with a very small and simplified fixed house that you can demonstrate and visualise, its correctness. Only when the simple specification is know to be correct do you refactor it, slowly introducing details and generality as you go.

1.1 Start with permissions that let you both ways through a door.

A `House` **has a set of** `Rooms` **"r1", "r2", ... and a set of** `Doors` **"d1", "d2", ...** There is an additional "room" representing the outside of the building and initially everyone is *outside*. The layout of the `House` is defined by a mapping from `Doors` to `Rooms`. `People` "p1", "p2" .. can mover around the house. The `Doors`, `Rooms`, `People` and `House` are constants defined in the construction of a model.

The specification makes use to two variables `where` and `perm`. Where a person is currently is defined by a mapping from `People` to `Rooms` called `where`. People can only pass through door to enter the room it is attached to by the given layout of the `House` and if they have the permission as define in the variable `perm`, a mapping from `Doors` to sets of `People`.

Let `d\in Doors` and `p \in People` in the following two steps;

addPerm(d,p) Gives a person permission to pass through a door.

enterDoor(d,p) Allows a person to pass through a door, if they have permission. Once some one passes through a door they enter the room defined by the `House` layout.

Note addPerm(d,p) can not be a step in `Next` because of the "free" variables `d` and `p`. Consequently these free variables need to be bound by an existential quantifier.

Initially hard code a small model one room, one door, one person

1. Build a model and correct your specification until it can run.
2. add invariants to check:
   (a) No one enters a room that they do not have permission to
   (b) A person only has permission to enter a room after the permission has been added to a relevant door. This is not easy to specify and would require some additional data being stored.

3. Inspect the State Graph to check the behaviour of your module.
4. After looking at the State Graph decide in what way is his model of the layout of a house and movement of people unrealistic. Then clone your model and improve it to satisfy the weaknesses you observed.

1.2 Progressively check larger models.

Repeat the above cycle of development each time reflecting on what invariants are needed.

1. a person is only in a room that they can reach from *outside* by passing through a sequence of doors that they have permission to use.

1.3 Refactor to allow one way permissions - report on lock in potential.

For prisons this would not be seen as a failure.

1.4 Refactor to capture ALL models with the Rooms, Doors and People as parameters

This will result in very very large models that can not be viewed as a state graph. You will need to carefully consider:

1. computational cost and prune the set of houses to being only "valid" houses.
2. what invariants can you guarantee

1. Every room in the house must be reachable by someone with the correct permissions.
2. No door goes from one room back to the same room
3. ..

2 Worker bees

Given a set of jobs and a set of processes we want to divide the jobs amongst the processes. When the jobs are completed return the results to one central place. The system should complete all the jobs even when one of the processes fails. Hence some means to redistributing the unfinished jobs of the failed process. No job should be completed more than once and all the jobs should be completed.

Write a master process that hands out jobs to workers and collects the results form the jobs. Given a set of jobs and a set of worker names the routine should build separate process for each of the workers. Then the jobs must be handed out one at a time to the workers. Each job only going to one of the workers processes. Once the worker has completed the job they must report back to the master process that the job has been completed. The master only need count the number of completed jobs.

Write a check invariant that can be used to verify that all the jobs have been finished (it is assumed that no errors occur). Model check your PlusCal specification, initially

using a single worker and single job. Only when this is working (view the State Graph to give your self confidence that you spec is correct) should you try to Model check using more workers and jobs.

A standard abstraction of a multy-process system is to specify a single process system that produces the same result although may be a lot slower. having built two TLA+ state machines one specifying the multy process system and the other the simpler and more abstract single process system. Then what is required is to establish that the more detailed state machine behaves like the abstract state machine when some of the details are over looked. This is more formally referred to as establishing a refinement relation between the two state machines.

End of Home Work