# Homework 3

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

The deadline for Homework 3 is February 25.

## Problem 1 Railway Switching (25 Points)

In this exercise, you will construct a simple model of a railway switching system, and you will check that a switching policy ensures no collisions. You will make some simplifying assumptions, for example, that a train occupies one track segment at a time, but you will learn techniques that apply in general, especially how to model a physical environment that allows many arbitrary behaviors (in this case the train movements), and how to separate the requirement (that no collisions occur) from assumptions (that drivers obey signals).

(a) Model the track layout as a collection *Segment* of track segments, with relation *next* from *Segment* to *Segment*. Segments are physically disjoint, touching each other only at their endpoints, and are directional, with trains assumed to travel from one endpoint to the other. The endpoints are not represented explicitly, though. Instead, we are representing the connection of the exit end of *s1* to the entrance end of *s2* by *next* mapping *s1* to *s2*. Generate two sample layouts; and obtain some nice visualizations using the Alloy Analyzer. **(2 Points)**

(b) To model the possibility of collisions, we might just say that two trains can collide only when they are on the same segment. For a more general notion, which allows for the possibility of a collision between trains on segments that are, for example, parallel but too close to allow a train on each, we can declare a relation *overlaps* that represents, very abstractly, the physical layout of the track, mapping a segment *s1* to a segment *s2* when it would be dangerous for one train to be on *s1* and another to be on *s2* at the same time. What properties would you expect this relation to have: is it reflexive, symmetric, transitive? Add the relation to your model, along with a fact *Overlaps* recording whichever of these properties you think should hold. **(3 Points)**

(c) Now you are going to introduce time varying state. Declare a signature *Train* to represent a set of trains, and a signature *TrainState*, with a relation *on* from *Train* to *Segment* to represent their positions. (Remember that each train can occupy only a single segment.) Define an additional field *occupied* in *TrainState* that holds the set of segments occupied by trains. Generate and visualize two sample states. **(2 Points)**

(d) To describe all physical possible train movements, introduce an operation *Move* on *TrainState* that takes as arguments two train states (representing the pre- and post-states), and a set of trains that move, and constraints the train states so that, in this step, each train that moves passes from a segment to one of its successors under the *next* relation. Generate and visualize two sample executions of the operation *Move*.

**(3 Points)**

(e) To model the signaling system, introduce a signature *GateState* with a field *closed* whose value is a set of segments, representing those segments beyond which a train is not supposed to travel. Note that there is no need to introduce gates or lights as explicit atoms. Write a predicate *LegalMovements* that captures legal movements and whose arguments are a *GateState*, two *TrainStates*, and a set of *Trains* that move. **(3 Points)**

(f) Write a safety condition *SafeTrainState* on *TrainState* saying that trains never occupy overlapping segments, and generate two sample states: one that satisfies the condition and one that violates it. **(2 Points)**

(g) The hardest part is designing the mechanism–the policy that determines when gates should be closed. Rather than prescribing exactly when and which gates should be closed, we want to write a condition that imposes some minimal constraints. In this way, we will actually be checking a whole family of possible mechanisms. Write the policy as a predicate *GatePolicy* that takes as arguments a *GateState* and a *TrainState*. It may say, for example, that if several occupied segments share a successor, then at most one can have an open gate. **(5 Points)**

(h) Finally, put all the parts together: write an assertion *SafeSwitching* that says that when the trains move, if the mechanism obeys the gate policy, and the train movements are legal, then a collision does not occur (that is, the system does not transition from a safe state to an unsafe state). Check this assertion, and if you find counterexamples, study them carefully, and adjust your model. Most likely, your gate policy will be at fault.

**(3 Points)**

(i) When you are satisfied that the gate policy works as expected (preventing collisions), make sure that you have not overconstrained the model, by generating and visualizing two interesting train movements. **(2 Points)**

*Hint: you can use Alloy to export the snapshots of generated instances to dot files, which you can process further using Graphviz. You can also use the snapshot viewer to adjust the coloring of atoms to improve the visualization. For instance, in part (c) you will probably want to use coloring to indicate the occupied segments.*