Petri Net Assignment

By Arno Deceuninck (s0181217) and Dogukan Altay (s0211552)

1 Petri-Net

The final model can be found in tapaal_codes/ptr_petri_light_v3.tapn

1.1 Describe and validly construct a single rail track/segment. What happens when this is a light?

Below graph shows an example of a track without a light used in the system. As clearly shown from the direction of the arcs it is a track moves right to left. The approach is built on top of the net shown in the paper here.
There are several key parts in a track segment. These are:

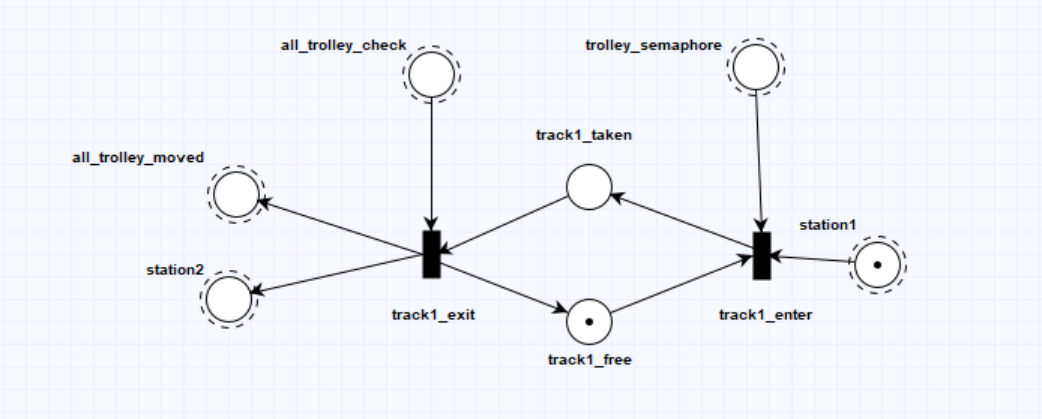
- Track free/taken logic
- Trolley semaphore
- All trolley check and all trolley moved.

Track free/taken logic is used for to accept the movement of a trolley between two stations. With this approach, we are able to control trolley movement for each 'clock' tick to ensure fairness in the system. Clock implementation explained further in the report.

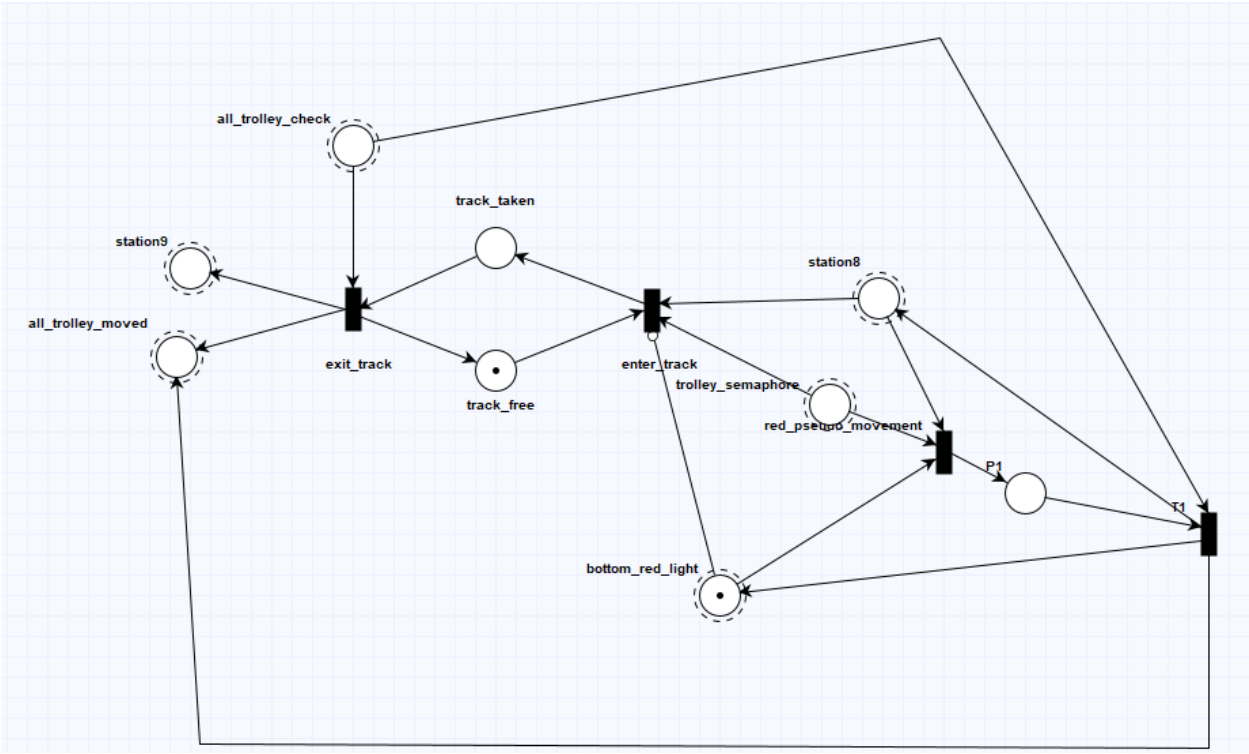
Trolley semaphore is a condition where only allows track_enter transitions to be fireable if the clock tick allows.

All trolley check is a condition where all trolleys entered a track and ready to reach destination

All trolley moved is a place where how many trolleys finished their movement successfully. This is a helper place for clock functionality.

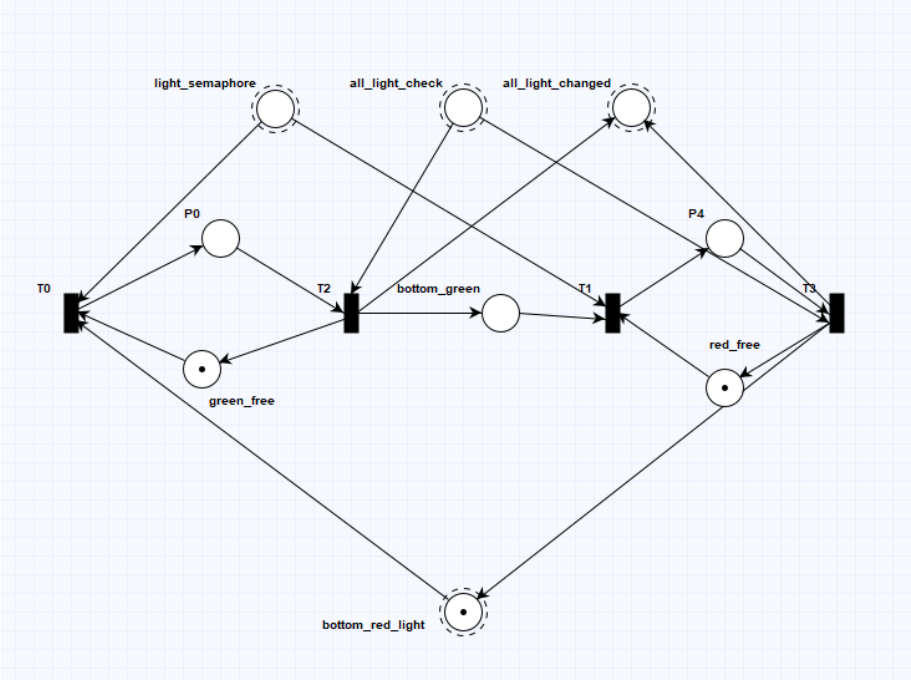


Below graph shows an example of a track with light used in the system. Overall flow of the net is same with a lightless track. However there is only one more condition called bottom_red_light for enter_track transition to be fireable.



Below graph shows an example light net used in the system. The shared places called; light_semaphore, all_light_check and all_light_changed have identical purposes with the trolley logic. The relation of light with the track is as follows:

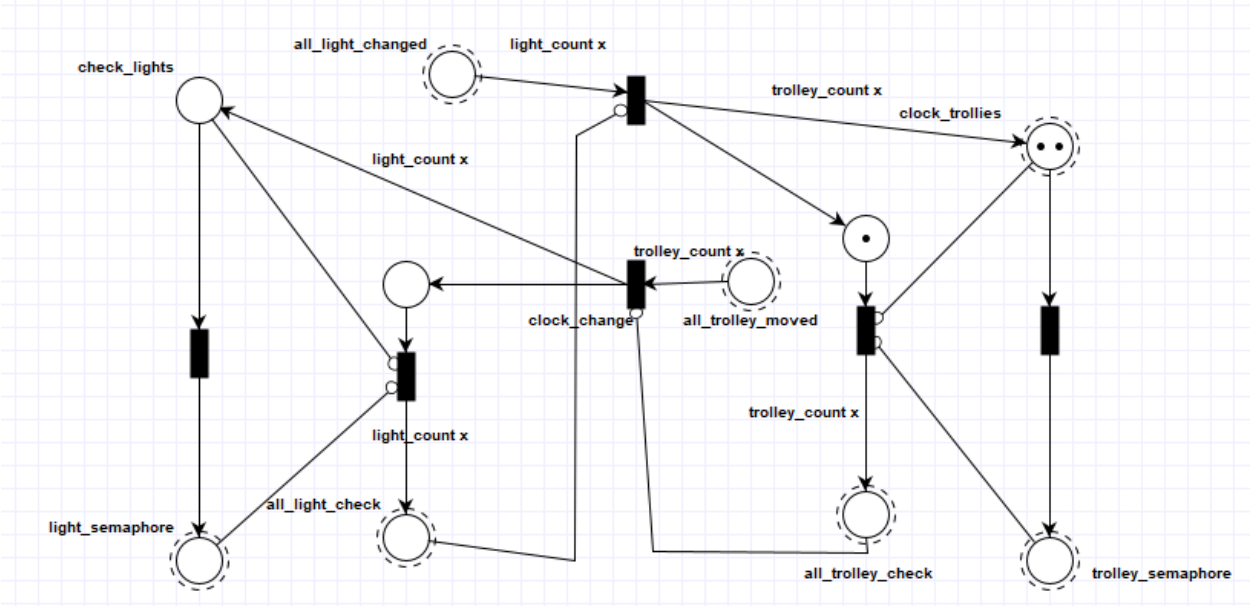
- Light alternates between red and light for each clock tick.
- Red light state is a shared place for track.
- To accommodate the logic and not lose tokens in the light net. We have used an inhibitor arc where if it is not red light, the trolley can move forward.
- In case a trolley stopped by red light, the system makes a pseudo movement for the trolley to be able to work with the clock system.



1.3 Clearly describe what is used to accomodate for the introduction of the clock.

Below graph shows the clock net used in the system. As clearly shown in the graph, it consist of two symmetrical smaller nets to alternate between trolley movement and light changes. The flow of the clock as follows:

- Initially we choose to move trolleys first. Therefore clock_trolleys place has two tokens initially since we have 2 trolleys in the system. (We couldn't use global constants as place markings)
- Since we want a clock and a system deterministic enough, all tokens have to be consumed and transferred to the trolley_semaphore place.
- As soon as trolley_semaphore has tokens, all trolleys' track_enter transitions become fireable and these transitions can only be called twice at a clock tick since we allow only 1 movement per tick per trolley.
- After all trolley_semaphore tokens consumed by trolleys another transition fires so that all_trolley_check place can receive 'trolley_count' much tokens.
- As soon as all_trolley_check place has tokens, trolleys which entered to the track can now exit to reach to destination.
- For each track_exit transition fire, all_trolley_moved place receives a token and as soon as the place has tokens equals to trolley_count the only transition available to fire is clock_change transition.
- As soon as the clock_change transition fires up, the clock switches to the light side(Left hand side of the net).
- All steps are the same for the lights.



With this clock approach, we ensured that only the same entities can move/change for each clock tick. Therefore assures system has fairness.

2 Traces

2.1 A trolley traveling for at least 2 tracks before passing through a light that is red when the trolley arrives there.

The trace named non_stop_2_track.trc can be found in the output_traces folder. The trolley that started in the station5 has satisfies the condition.

2.2 A trolley that travels in a single "8"-shape over the network, without ever seeing a red light.

The trace named 8_shape.trc can be found in the output_traces folder. The trolley started at station4 satisfies the condition. Its path is as below:
S4->S5->S6->S7->S8->S9->S4->S5->S6->S1->S2->S3->S4

3 Preliminary analysis

3.1 Reachability / Coverability

The markings are given w.r.t. the following places: ['all_light_changed', 'all_light_check', 'all_trolley_check',

'all_trolley_moved', 'bottom_light.P0', 'bottom_light.P4', 'bottom_light.bottom_green', 'bottom_light.green_free',
'bottom_light.red_free', 'bottom_red_light', 'clock.P14', 'clock.P22', 'clock.P24', 'clock_trollies', 'light_sema
phore', 'middle_red_light', 'middle_right.P0', 'middle_right.P4', 'middle_right.green_free', 'middle_right.red_fre
e', 'middle_right.top_green', 'station1', 'station2', 'station3', 'station4', 'station5', 'station6', 'station7',
'station8', 'station9', 'top_light.P0', 'top_light.P4', 'top_light.green_free', 'top_light.red_free', 'top_light.t
op_green', 'top_red_light', 'track1to2.track1_free', 'track1to2.track1_taken', 'track2to3light.P1', 'track2to3ligh
t.track_free', 'track2to3light.track_taken', 'track3to4.track_free', 'track3to4.track_taken', 'track4to5.track_fre
e', 'track4to5.track_taken', 'track5to6light.P1', 'track5to6light.track_free', 'track5to6light.track_taken', 'trac
k6to1.track_free', 'track6to1.track_taken', 'track6to7.track_free', 'track6to7.track_taken', 'track7to8.track_free
', 'track7to8.track_taken', 'track8to9light.P1', 'track8to9light.track_free', 'track8to9light.track_taken', 'track
9to4.track_free', 'track9to4.track_taken', 'trolley_semaphore'].

3.1.1 Argue why your solution would (not) produce an infinite reachability graph.

The trollies keep making 8's, so their is no given end station and they can infinitely keep driving around. However, the amount of tokens stays the same after each entire clock cycle, so the tokens don't grow to infinity, so the reachability graph is not infinite.

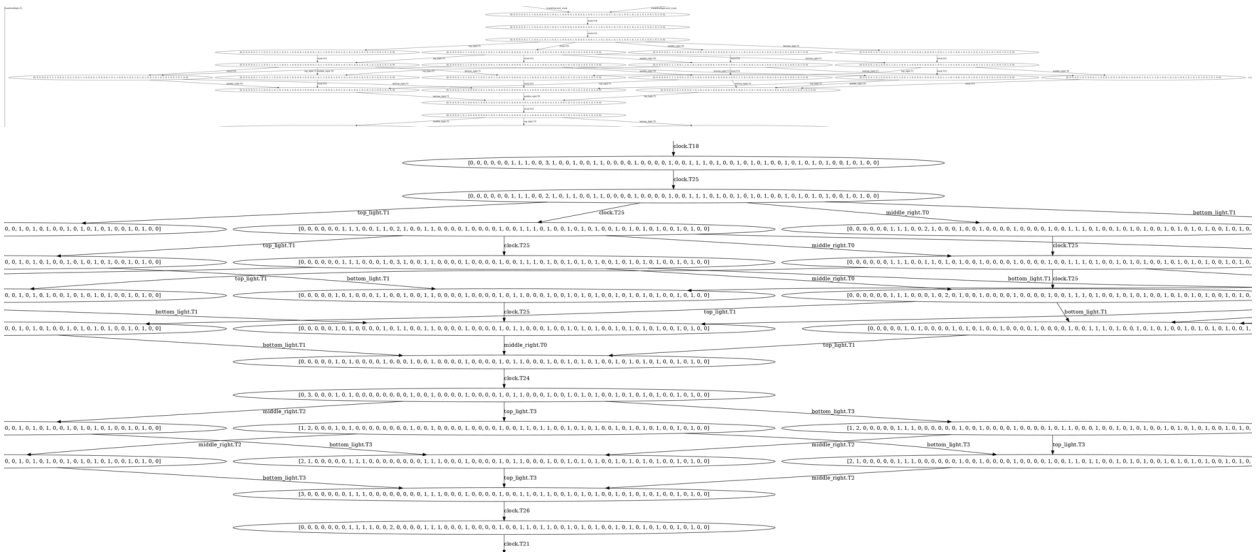
3.1.2 Generate the reachability graph and discuss any patterns you identify. If your reachability graph is infinite, generate the graph for a representative subset of your model instead.

```
python3 RC.py tapaal_codes/ptr_petri_light_v3.tapn reachability.dot reachability
dot -Tpng reachability.dot -o reachability.png
```

The reachability graph has 1013 nodes. A lot of the nodes are just clock transitions. The entire 13Mb picture of the graph can be found in reachability.png. This is a screenshot of the entire graph:

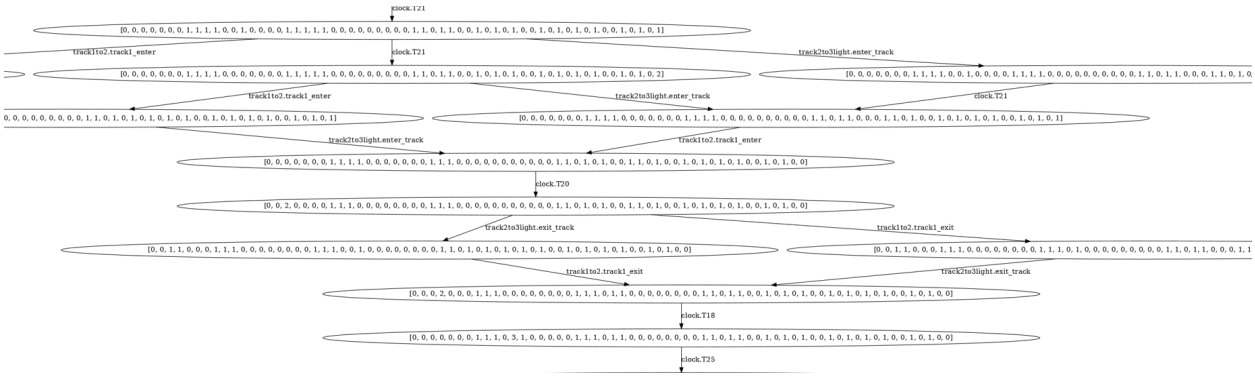
There are some patterns visible, for example the alternating trolley pattern and light pattern.

Light pattern



This is two times the same picture, but zoomed a bit in for readability. This pattern occurs after all trollies have moved to update all the lights (to switch their colors). There is a choice in which order you do this, but it all results finally in the same state.

Track pattern



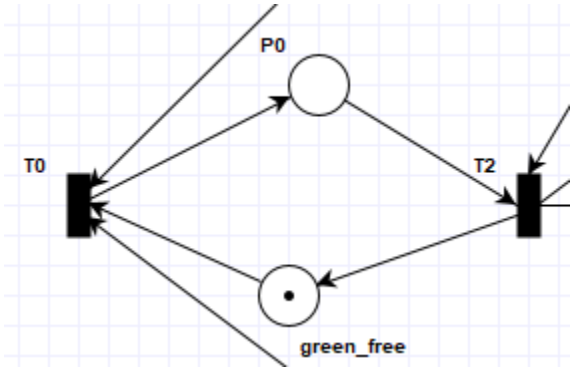
This patterns comes in front and after the light pattern. Here you must choose the order in which the tracks must be moved, but it results in the same state, whatever the order is you choose (unless of course it's at the split, where you need to choose which track to follow).

Tree structure

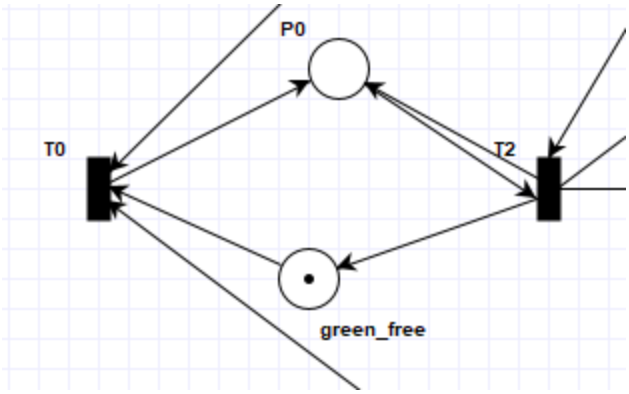
In the entire picture of the reachability graph, you can also see kind of a tree structure. This is because of the track choice you have to make from station 6 to station 1 or station 7. It isn't really a tree, because it later merges back to an earlier state.

3.1.3 Can you add/remove a place/transition such that the reachability graph becomes finite if it was infinite (or vice-versa)? Your new model does not have to conform to the requirements.

Yes, you can do this.
Before:



After:



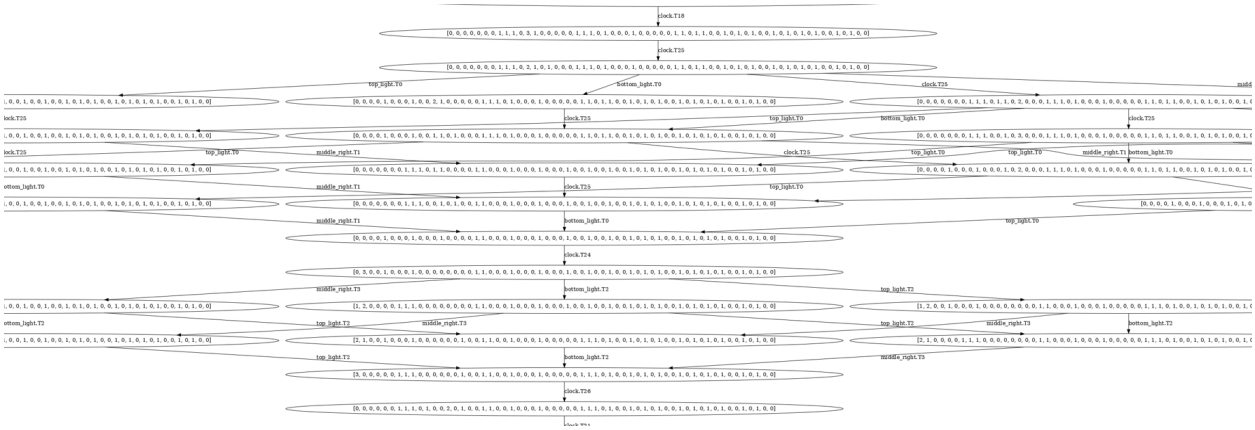
By adding this extra transition, the number of tokens increases, but the extra created token get never taken away, so while the trollies keep moving, the number of tokens build up to infinity, leading to an infinite reachability graph.

3.1.4 Generate the coverability graph and discuss any patterns you identify.

python3 RC.py tapaal_codes/ptr_petri_light_v3.tapn coverability.dot coverability
dot -Tpng coverability.dot -o coverability.png

This graph can be found in coverability.png.

You can again see a similar larger splits, representing the choice between going from station 6 to station 1 or to station 7.
Also the same light pattern and trolley pattern occurs, which you can both see below.
Light pattern:



Trolley pattern:

3.2 Invariant Analysis

3.2.1 Generate the P-invariants for the model and explain what they mean in terms of the railroad.

```
RC.py -p tapaal_codes/ptr_petri_light_v3.tapn deleteme.dot coverability > invariants.txt
```

Following invariants describe that a track is either free, or it's taken:
M(track5to6light.track_free) + M(track5to6light.track_taken) = 1
M(track6to1.track_free) + M(track6to1.track_taken) = 1
M(track3to4.track_free) + M(track3to4.track_taken) = 1

M(track9to4.track_free) + M(track9to4.track_taken) = 1
M(track2to3light.P1) + M(track5to6light.P1) = 1
M(track7to8.track_free) + M(track7to8.track_taken) = 1
M(track8to9light.track_free) + M(track8to9light.track_taken) = 1
M(track6to7.track_free) + M(track6to7.track_taken) = 1
M(track1to2.track1_free) + M(track1to2.track1_taken) = 1
M(track4to5.track_free) + M(track4to5.track_taken) = 1

Following invariants describe that it's either possible for a light to become red, or not possible:
M(middle_right.P4) + M(middle_right.red_free) = 1
M(bottom_light.P4) + M(bottom_light.red_free) = 1
M(top_light.P4) + M(top_light.red_free) = 1

Following invariants describe that it's either possible for a light to become green, or not possible:
M(middle_right.P0) + M(middle_right.green_free) = 1
M(bottom_light.P0) + M(bottom_light.green_free) = 1
M(top_light.P0) + M(top_light.green_free) = 1

Following invariants describe that a light is either red or green:
Note: P0 is green_taken and P4 is red_taken. Those are temporarily states to assure all lights update in a clock cycle, but it corresponds with being green or red when the clock cycle has finished. P1 is a temporarily state for the pseudo movement when the light is red. Note that it's strange that the top_light doesn't take the pseudo movements in account.

M(top_light.P0) + M(top_light.P4) + M(top_light.top_green) + M(top_red_light) + M(track2to3light.P1) = 1

M(middle_red_light) + M(middle_right.P0) + M(middle_right.P4) + M(middle_right.top_green) + M(track5to6light.P1) = 1

M(bottom_light.P0) + M(bottom_light.P4) + M(bottom_light.bottom_green) + M(bottom_red_light) + M(track8to9light.P1) = 1

Following invariants describe that there are two trolleys (in the tracks and stations together):
M(station1) + M(station2) + M(station3) + M(station4) + M(station5) + M(station6) + M(station7) + M(station8) + M(station9) + M(track1to2.track1_taken) + M(track2to3light.P1) + M(track2to3light.track_taken) + M(track3to4.track_taken) + M(track4to5.track_taken) + M(track5to6light.P1) + M(track5to6light.track_taken) + M(track6to1.track_taken) +

M(track6to7.track_taken) + M(track7to8.track_taken) + M(track8to9light.P1) + M(track8to9light.track_taken) + M(track9to4.track_taken) = 2

Following invariants are related to the fairness mechanism:

2 * M(all_light_changed) + 3 * M(all_trolley_check) + 3 * M(all_trolley_moved) + 4 * M(bottom_light.P0) + 4 * M(bottom_light.P4) + 2 * M(bottom_light.bottom_green) + 2 * M(bottom_red_light) + 6 * M(clock.P14) + 2 * M(clock.P22) + 2 * M(light_semaphore) + 2 * M(middle_right.P0) + 2 * M(middle_right.P4) + 2 * M(top_light.P0) + 2 * M(top_light.P4) + 2 * M(track8to9light.P1) = 8

2 * M(all_light_changed) + 2 * M(all_light_check) + 3 * M(all_trolley_check) + 3 * M(all_trolley_moved) + 6 * M(clock.P14) + 6 * M(clock.P24) = 6

2 * M(all_light_changed) + 2 * M(all_light_check) + 3 * M(all_trolley_moved) + 6 * M(clock.P24) + 3 * M(clock_trollies) + 3 * M(track1to2.track1_taken) + 3 * M(track2to3light.P1) + 3 * M(track2to3light.track_taken) + 3 * M(track3to4.track_taken) + 3 * M(track4to5.track_taken) + 3 * M(track5to6light.P1) + 3 * M(track5to6light.track_taken) + 3 * M(track6to1.track_taken) + 3 * M(track6to7.track_taken) + 3 * M(track7to8.track_taken) + 3 * M(track8to9light.P1) + 3 * M(track8to9light.track_taken) + 3 * M(track9to4.track_taken) + 3 * M(trolley_semaphore) = 6

2 * M(all_light_changed) + 2 * M(all_light_check) + 3 * M(all_trolley_check) + 3 * M(all_trolley_moved) + 2 * M(bottom_light.P0) + 2 * M(bottom_light.P4) + 2 * M(bottom_light.bottom_green) + 2 * M(bottom_red_light) + 6 * M(clock.P14) + 6 * M(clock.P24) + 2 * M(track8to9light.P1) = 8

2 * M(all_light_changed) + 2 * M(all_light_check) + 3 * M(all_trolley_moved) + 2 * M(bottom_light.P0) + 2 * M(bottom_light.P4) + 2 * M(bottom_light.bottom_green) + 2 * M(bottom_red_light) + 6 * M(clock.P24) + 3 * M(clock_trollies) + 3 * M(track1to2.track1_taken) + 3 * M(track2to3light.P1) + 3 * M(track2to3light.track_taken) + 3 * M(track3to4.track_taken) + 3 * M(track4to5.track_taken) + 3 * M(track5to6light.P1) + 3 * M(track5to6light.track_taken) + 3 * M(track6to1.track_taken) + 3 * M(track6to7.track_taken) + 3 * M(track7to8.track_taken) + 5 * M(track8to9light.P1) + 3 * M(track8to9light.track_taken) + 3 * M(track9to4.track_taken) + 3 * M(trolley_semaphore) = 8

2 * M(all_light_changed) + 3 * M(all_trolley_moved) + 2 * M(bottom_light.P0) + 2 * M(bottom_light.P4) + 2 * M(clock.P22) + 3 * M(clock_trollies) + 2 * M(light_semaphore) + 2 * M(middle_right.P0) + 2 * M(middle_right.P4) + 2 * M(top_light.P0) + 2 * M(top_light.P4) + 3 * M(track1to2.track1_taken) + 3 * M(track2to3light.P1) + 3 * M(track2to3light.track_taken) + 3 * M(track3to4.track_taken) + 3 * M(track4to5.track_taken) + 3 * M(track5to6light.P1) + 3 * M(track5to6light.track_taken) + 3 * M(track6to1.track_taken) + 3 * M(track6to7.track_taken) + 3 * M(track7to8.track_taken) + 3 * M(track8to9light.P1) + 3 * M(track8to9light.track_taken) + 3 * M(track9to4.track_taken) + 3 * M(trolley_semaphore) = 6

2 * M(all_light_changed) + 3 * M(all_trolley_check) + 3 * M(all_trolley_moved) + 2 * M(bottom_light.P0) + 2 * M(bottom_light.P4) + 6 * M(clock.P14) + 2 * M(clock.P22) + 2 * M(light_semaphore) + 2 * M(middle_right.P0) + 2 * M(middle_right.P4) + 2 * M(top_light.P0) + 2 * M(top_light.P4) = 6

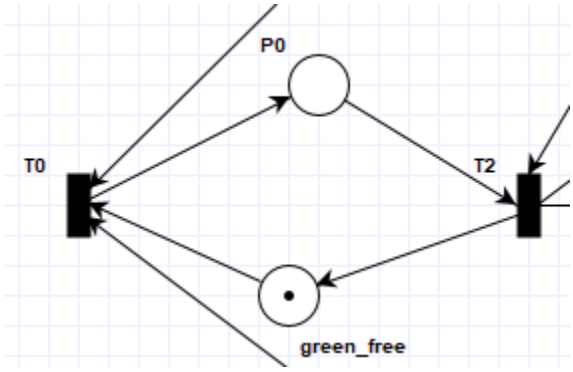
2 * M(all_light_changed) + 3 * M(all_trolley_moved) + 4 * M(bottom_light.P0) + 4 * M(bottom_light.P4) + 2 * M(bottom_light.bottom_green) + 2 * M(bottom_red_light) + 2 * M(clock.P22) + 3 * M(clock_trollies) + 2 * M(light_semaphore) + 2 * M(middle_right.P0) + 2 * M(middle_right.P4) + 2 * M(top_light.P0) + 2 * M(top_light.P4) + 3 * M(track1to2.track1_taken) + 3 * M(track2to3light.P1) + 3 * M(track2to3light.track_taken) + 3 * M(track3to4.track_taken) + 3 * M(track4to5.track_taken) + 3 * M(track5to6light.P1) + 3 * M(track5to6light.track_taken) + 3 * M(track6to1.track_taken) + 3 * M(track6to7.track_taken) + 3 * M(track7to8.track_taken) + 5 * M(track8to9light.P1) + 3 * M(track8to9light.track_taken) + 3 * M(track9to4.track_taken) + 3 * M(trolley_semaphore) = 8

3.2.2 Which invariants did you expect? Which ones are surprising? Why?

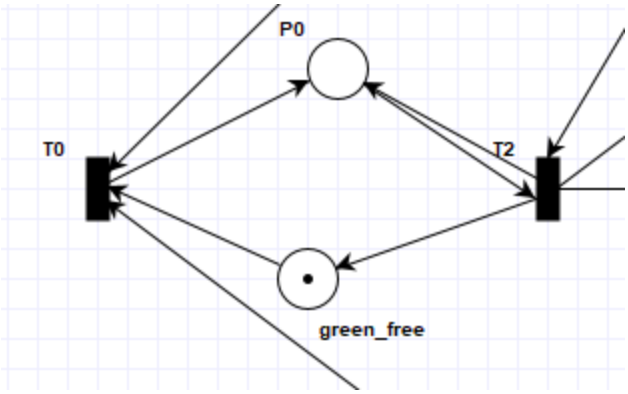
We didn't expect such a large equations related to the fairness mechanism, since it was a simple mechanism, but involves everything. We did expect the others.

3.2.3 Can you add a single place or transition to the model that changes these invariants (for the better or the worse)? Your new model does not have to conform to the requirements.

Yes, you can.
E.g. with the same transition as in the previous exercise
Before:



After:



This invariant doesn't hold anymore after adding this transition:
M(top_light.P0) + M(top_light.green_free) = 1
Since the sum of tokens in P0 and in green_free keeps growing, the sum can't stay 1.

4 TAPAAL queries

4.1 Boundedness

a) For the boundedness analysis of the system, we have used a query to check the k-boundedness of station places. Query used can be seen below. With this result we can clearly see that except our clock the system is k-bound to 2 tokens which means there is no place which has 2 tokens at a specific time for the initial state described at the assignment.
    i) Query: AG (station1 < 2 and station2 < 2 and station3 < 2 and station4 < 2 and station5 < 2 and station6 < 2 and station7 < 2 and station8 < 2 and station9 < 2)
b) However, for the clock net the clock_trollies and clock_light places will always have at most trolley number and light number of tokens in the system. With given initial state of the system, we can clearly see that whole net is k-bound to 4 due to clock_light state.

4.2 Deadlock

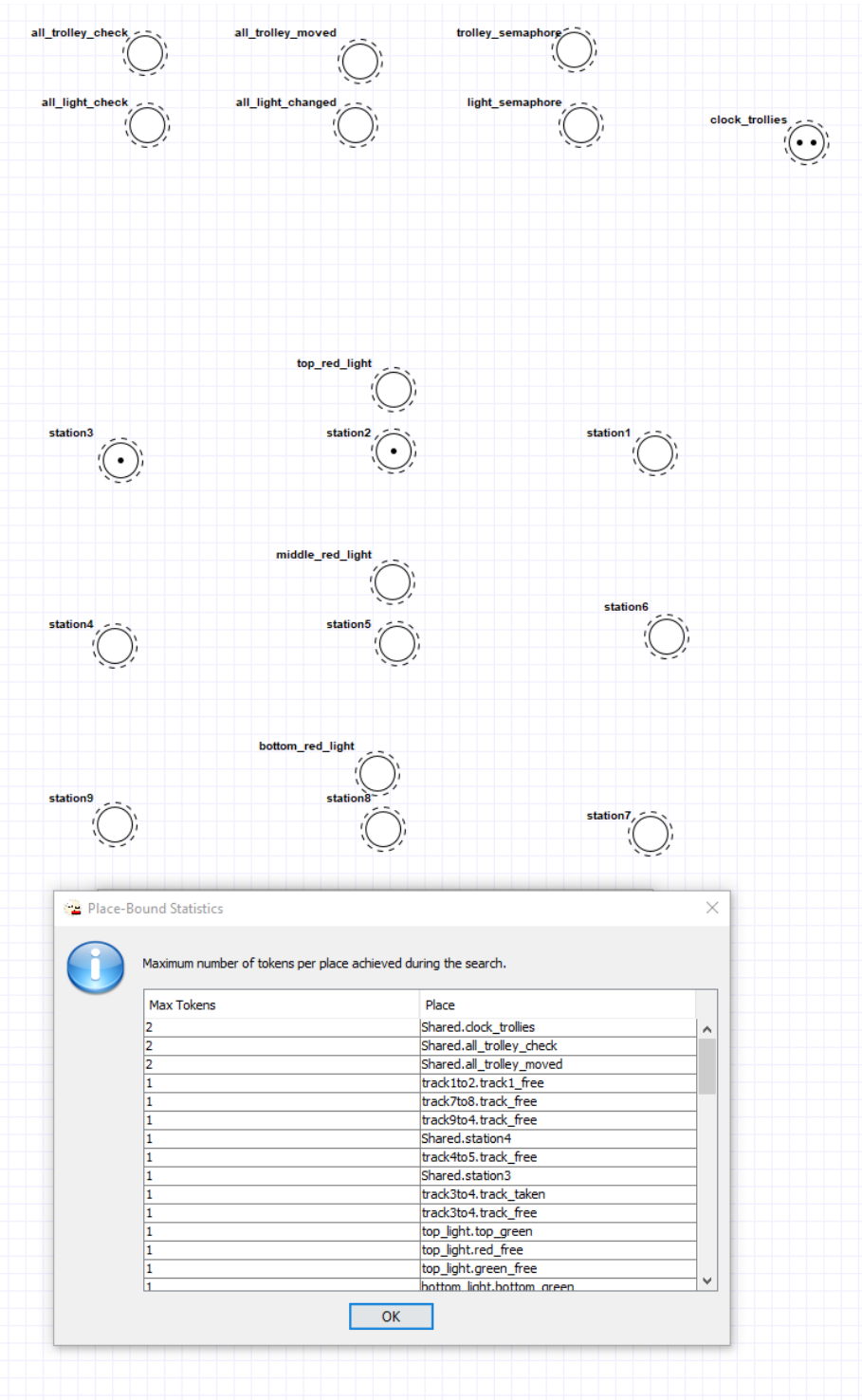
a) The "EF deadlock" query failed, meaning that there are no deadlocks possible in the system.
b) A deadlock is not possible, because the system is made such that there always is an option to move, since a trolley can always continue driving.
c) In the reachability graph, a deadlock would look like a state which doesn't have any outgoing transitions. You can't see a deadlock in a coverability graph.

4.3 Liveness

a) For transition clock.T21 we can see that the transition L2-Live with the query below. It means that if clock_trolley has some k tokens, the clock.T21 transition will fire k times. This ensures that clock.T21 is a transition with L2-Live liveness.
    i) Query: E (clock.T21 U clock_trollies > 0)
b) For this particular example we cannot change liveness of clock.T21 without altering the requirements of the system.

4.4 Fairness

a) Since we introduced a clock to the system. The fairness is already implemented by its nature. As clearly shown with output traces (hence 3 output trace) the clock is a proof of true fairness in the system.
b) By using our safety check query and using below initial markings, it is proven that true fairness is in place. If the query fails that means one of the trolleys moved twice at a single clock tick.

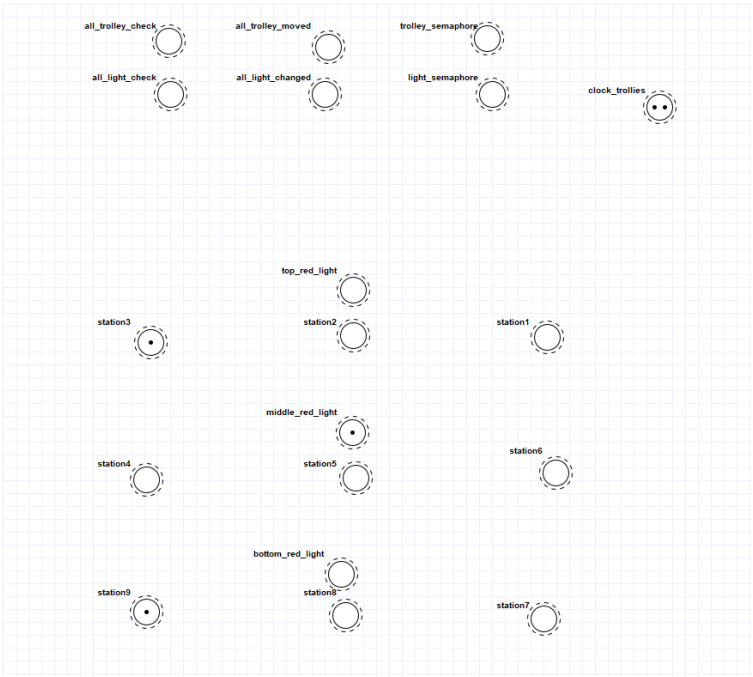

all_trolley_check
all_trolley_moved
trolley_semaphore
all_light_check
all_light_changed
light_semaphore
clock_trollies
top_red_light
station3
station2
station1
middle_red_light
station4
station5
station6
bottom_red_light
station9
station8
station7
Place-Bound Statistics
Maximum number of tokens per place achieved during the search.
Max Tokens
Place
2
Shared.clock_trollies
2
Shared.all_trolley_check
2
Shared.all_trolley_moved
1
track1to2.track1_free
1
track7to8.track_free
1
track9to4.track_free
1
Shared.station4
1
track4to5.track_free
1
Shared.station3
1
track3to4.track_taken
1
track3to4.track_free
1
top_light.top_green
1
top_light.red_free
1
top_light.green_free
OK

c)

4.5 Safety

a) For assessing safety, we have used a query which checks k-boundedness only on station places. If station places' k-boundness is less than two, this means that for a given petri net there is no possible trolley crash(no places with 2 tokens).
   i) Query: AG (station1 < 2 and station2 < 2 and station3 < 2 and station4 < 2 and station5 < 2 and station6 < 2 and station7 < 2 and station8 < 2 and station9 < 2)

b) However, the proposed petri net does not have any system to check and prevent trolley collusion. Such initial condition example can be given as below and the output trace can be found in output_traces folder.
   i) Since the railroad has a merge point, without any collusion prevention it is always possible to have a collusion in the system.



c)