Project 1 : Part 1

Task 1: For Layout 1, U = { T1, T2, R1, R2, R3, R4, S1, S2 },
write the formulas 1 through 6 in a file using the homework language.

Formula 1 (Task 1): expressed in the homework language.
This axiom represents the positive connections in the rail layout. In other words, these are the connections that do exist.

```
connects(R1, R2) ^ connects(R2, R3) ^
connects(R3, R4) ^ connects(R4, R1) ^
```

Formula 2 (Task 1) : expressed in the homework language.
These axioms represent the negative connections in the rail layout.
In other words, these are the connections that do not exist.
Basically, the above connections are the only connections that exist, and a rail is not connected to itself.

```
~connects(R1, R3) ^ ~connects(R1, R4) ^
~connects(R2, R4) ^ ~connects(R2, R1) ^
~connects(R3, R1) ^ ~connects(R3, R2) ^
~connects(R4, R2) ^ ~connects(R4, R3) ^
(all r ~connects(r, r)) ^
```

Formula 3 (Task 1): expressed in the homework language.
A situation is safe if and only if no 2 trains occupy the same rail.

```
(
      all s safe(s) <-> (
            all r all t1 all t2 (
                  on(t1,r,s) ^ on(t2,r,s) -> Equals(t1,t2)
            )
      )
)^
```

Formula 4 (Task 1) : expressed in the homework language.
This asserts that situation 1 is safe.

```
(safe(S1)) ^
```

Formula 5 (Task 1) : expressed in the homework language.
Dynamics: Only move to connected rails.

```
(
      all t all r2 (
            on(t,r2,S2) -> exists r1 (
                  on(t,r1,S1) ^ connects(r1,r2) ^ legal(t,r1,r2)
            )
      )
)^
```

Formula 6 (Task 1) : expressed in the homework language.

Definition of legal: A move is legal if there is no train on r2 in situation S1.

```
(
      all t1 all r1 all r2 (
            legal(t1,r1,r2) <-> (
                  ~(
                        exists t2 (on(t2,r2,S1))
                  )
            )
      )
)^
```

Task 2: You'll now need to add two more axioms to your file, specifying that:
2. The two new axioms, in the homework language.

The following axioms are generic because they do not mention the trains by name. They would therefore work for any universe of s, r, and t.

Axiom 1 (Task 2) : Every train is always on some rail.

```
(all t all s exists r on(t,r,s))^
```

The axiom essentially says: for any give t and s combination that train is on some rail.

Axiom 2 (Task 2) : No train is on two rails at the same time.

```
(
      all t all s all r1 (
            on(t,r1,s) -> (
                  all r2 (
                        on(t,r2,s) -> Equals(r1,r2)
                  )
            )
      )
)^
```

This axiom works because, we are specifying that if a train is on a given rail (in a given situation) then, if the train is on another rail, that rail is (equal to) the first rail.

Task 3: Find, print out and describe a satisfying assignment for the whole specification, using the provided code.

task3.txt.out

```
connects_R1_R1=false
connects_R1_R2=true
connects_R1_R3=false
connects_R1_R4=false
connects_R2_R1=false
connects_R2_R2=false
connects_R2_R3=true
connects_R2_R4=false
connects_R3_R1=false
connects_R3_R2=false
connects_R3_R3=false
connects_R3_R4=true
connects_R4_R1=true
connects_R4_R2=false
connects_R4_R3=false
connects_R4_R4=false
legal_T1_R1_R1=false
legal_T1_R1_R2=true
legal_T1_R1_R3=false
legal_T1_R1_R4=true
legal_T1_R2_R1=false
legal_T1_R2_R2=true
legal_T1_R2_R3=false
legal_T1_R2_R4=true
legal_T1_R3_R1=false
legal_T1_R3_R2=true
legal_T1_R3_R3=false
legal_T1_R3_R4=true
legal_T1_R4_R1=false
legal_T1_R4_R2=true
legal_T1_R4_R3=false
legal_T1_R4_R4=true
legal_T2_R1_R1=false
legal_T2_R1_R2=true
legal_T2_R1_R3=false
legal_T2_R1_R4=true
legal_T2_R2_R1=false
legal_T2_R2_R2=true
legal_T2_R2_R3=false
legal_T2_R2_R4=true
legal_T2_R3_R1=false
legal_T2_R3_R2=true
legal_T2_R3_R3=false
legal_T2_R3_R4=true
legal_T2_R4_R1=false
legal_T2_R4_R2=true
legal_T2_R4_R3=false
legal_T2_R4_R4=true
on_T1_R1_S1=true
on_T1_R1_S2=false
on_T1_R2_S1=false
on_T1_R2_S2=true
on_T1_R3_S1=false
on_T1_R3_S2=false
on_T1_R4_S1=false
on_T1_R4_S2=false
on_T2_R1_S1=false
on_T2_R1_S2=false
on_T2_R2_S1=false
on_T2_R2_S2=false
on_T2_R3_S1=true
on_T2_R3_S2=false
on_T2_R4_S1=false
on_T2_R4_S2=true
safe_S1=true
safe_S2=true
```

“on” Assignments:

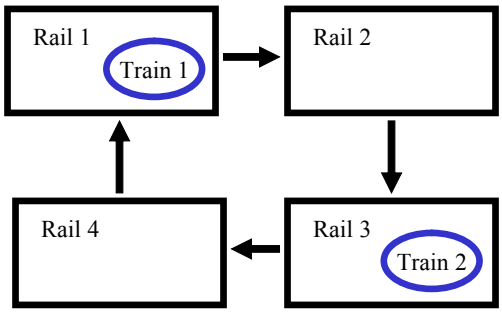
Where are the trains in situation 1 and situation2?

In situation 1,

train 1 is on rail 1 and
train 2 is on rail 3.

```
on_T1_R1_S1=true
on_T2_R3_S1=true
```

Diagram : Layout 1 : Situation 1

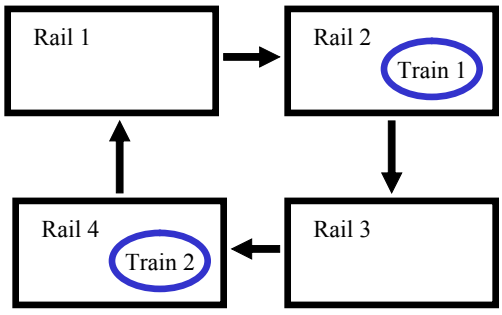


In situation 2,

train 1 is on rail 2 and
train 2 is on rail 4.

```
on_T1_R2_S2=true
on_T2_R4_S2=true
```

Diagram : Layout 1 : Situation 2



The location of the trains is reasonable.

The "dynamics" axiom (Formula 5) asserts that if a train is on a given rail in situation 2, then it was on a connecting rail in situation 1. The "connects" axioms assert that an axiom is not connected to itself. Therefore, the train must move (be on a different rail) from situation 1 to situation 2.

The legal assertion says that a train can only move to an empty rail. Therefore, in order for both trains to be able to move, they must be spaced out, with one rail in separating them. That way, they will both have an empty rail ahead of them.

The location of the trains satisfies this set of assertions and is therefore reasonable.

All of the other "on" assignments are false, as they should be.

```
on_T1_R1_S2=false
on_T1_R2_S1=false
on_T1_R3_S1=false
on_T1_R3_S2=false
on_T1_R4_S1=false
on_T1_R4_S2=false
on_T2_R1_S1=false
on_T2_R1_S2=false
on_T2_R2_S1=false
on_T2_R2_S2=false
on_T2_R3_S2=false
on_T2_R4_S1=false
```

Legal Assignments:

The definition of legal does not take into consideration whether or not the location of the train for which we are defining legality. It only considers the possible destination that we are considering. Therefore, it is legal for any train to move to an empty rail.

The definition of legal only pertains to situation 1.

It is legal for train 1 to move to rail 2, from any rail because rail 2 is not occupied it situation 1.

```
legal_T1_R1_R2=true
legal_T1_R2_R2=true
legal_T1_R3_R2=true
legal_T1_R4_R2=true
```

It is legal for train 2 to move to rail 2, from any rail because rail 2 is not occupied it situation 1.

```
legal_T2_R1_R2=true
legal_T2_R2_R2=true
legal_T2_R3_R2=true
legal_T2_R4_R2=true
```

It is legal for train 1 to move to rail 4, from any rail because rail 4 is not occupied it situation 1.

```
legal_T1_R1_R4=true
legal_T1_R2_R4=true
legal_T1_R3_R4=true
legal_T1_R4_R4=true
```

It is legal for train 2 to move to rail 4, from any rail because rail 4 is not occupied it situation 1.

```
legal_T2_R1_R4=true
legal_T2_R2_R4=true
legal_T2_R3_R4=true
legal_T2_R4_R4=true
```

It is not legal for any train to move to rail 1 or rail 3 because those rails are occupied it situation 1.

```
legal_T1_R1_R1=false
legal_T1_R1_R3=false
legal_T1_R2_R1=false
legal_T1_R2_R3=false
legal_T1_R3_R1=false
legal_T1_R3_R3=false
legal_T1_R4_R1=false
legal_T1_R4_R3=false
legal_T2_R1_R1=false
legal_T2_R1_R3=false
legal_T2_R2_R1=false
legal_T2_R2_R3=false
legal_T2_R3_R1=false
legal_T2_R3_R3=false
legal_T2_R4_R1=false
legal_T2_R4_R3=false
```

These 4 connections are true:

```
connects_R1_R2=true
connects_R2_R3=true
connects_R3_R4=true
connects_R4_R1=true
```

All of the other connections are false:

```
connects_R1_R1=false
connects_R1_R3=false
connects_R1_R4=false
connects_R2_R1=false
connects_R2_R2=false
connects_R2_R4=false
connects_R3_R1=false
connects_R3_R2=false
connects_R3_R3=false
connects_R4_R2=false
connects_R4_R3=false
connects_R4_R4=false
```

Safe Assignments:

The following are the safe assignments. Situation 1 is safe because we explicitly assigned it to be. Situation 2 is safe because there or no 2 trains on the same rail. While, the sat solver explicitly enforced that situation 1 would be safe, situation 2 is safe because of the rules that we axiomized. In other words, our rule cause situation 2 to be safe, assuming that situation 1 was safe. Task 4 proves this to be so.

```
safe_S1=true
safe_S2=true
```

Task 4: Now, add the assertion that there's a possible train wreck, ¬safe(S2), and use the code to show that there are no possible crashes in this domain. Would it have been okay to use WalkSAT for this job? Why or why not?

Prove no wrecks:

Method:
In order to show that there can be no crashes, I included the following assertion into the axiom set:

Assertion (Task 4) : There is a possible train wreck in situation 2.

```
~safe(S2)^
```

I then ran the sat solver (DPLL) on the axiom set. The result was null, which means that the sat solver is unable to find a satisfying solution of the previous rule set and the new axiom. Therefore, it is not possible for a crash to occur given the rule set.

To clarify that point; the new assertion says that there is a wreck in situation 2. However, the rule set that I created makes sure that there is no wreck in situation 2. Therefore, finding a satisfying assignment is impossible. Since DPLL will search the entire assignment space (complete), we know that there is no satisfying assignment if DPLL returns `null`.

Output:

```
task4.txt.out

null
```

Why not use WalkSAT?

It would not have been OK to use WalkSAT for this task because WalkSAT does not check every possible path to a solution and therefore cannot guarantee that a solution does not exist, even when it does not find one.

Likewise, WalkSAT is not guaranteed to find a solution if one exists. WalkSAT's advantage is that it is fast at finding a satisfying solution, in certain situations, if one exists. WalkSAT is fast if a satisfying solution correlates well with it's objective function (the number of satisfied clauses). If that is the case, then WalkSAT's downward search path towards more satisfied clauses will lead to a satisfying assignment.

WalkSAT can be initialized and terminated in a variety of ways. Some result in a more complete and robust search of the space. However, none of them guarantee a complete search of the space.

Task 5: Modify the definition of legal to include allowing the train to stay on the rail it's currently on. You may also need to modify the "connects" relation. Is this domain still safe? Demonstrate using the java code.

New Definition of Legal:

It is legal for a given train to move to a given rail if there is no train on the given rail, in situation S1, other than itself (the given train).

```
(
 all t1 all r1 all r2 (
      legal(t1,r1,r2) <-> (
            ~(
                  exists t2 (
                        on(t2,r2,S1) ^ ~Equals(t1,t2)
                  )
            )
      )
 )
)^
```

Connects Relations:

I modified the "connects" relation to allow all trains to be connected to themselves. First I removed the negation of this relation (all r ~connects(r,r)) from Formula 2.
Then I added this relation (all r connects(r,r)) to Formula 1. The result is as follows:

```
connects(R1, R2) ^ connects(R2, R3) ^
connects(R3, R4) ^ connects(R4, R1) ^
(all r connects(r,r)) ^

~connects(R1, R3) ^ ~connects(R1, R4) ^
~connects(R2, R4) ^ ~connects(R2, R1) ^
~connects(R3, R1) ^ ~connects(R3, R2) ^
~connects(R4, R2) ^ ~connects(R4, R3) ^
```

Is the domain is still safe?

Yes, there are many possible safe configurations.

How you used the code to show safeness?

I put the assertion that situation 2 is unsafe into the axiom set to see if it could generate an unsafe situation for situation 2.

Assertion : There is a train wreck in situation 2.

```
~safe(S2)^
```

I ran the DPLL sat solver with the "`~safe(S2)^`" assertion, and it returned "null," indicating that no solution was found. Since DPL is guaranteed to find a satisfying solution if one exists, then it is not possible to produce an unsafe condition in situation 2, given the axiom set.

Output:

```
task5nowrecks.txt.out

null
```

Finding a safe situation:

I then ran the sat solver without the "`~safe(S2)^`" assertion, and it found a safe situation where the trains were on rails 1 and 4 in situation 1, and did not move in situation 2. This makes sense; because it is now OK for 2 trains to be on consecutive rails due to the fact that it is legal for a train to "move" to a rail that it is currently on (AKA stay on the same rail) and all rails are connected to themselves. These to notions, together satisfy the dynamics axiom.

"on" assignments:

```
on_T1_R1_S1=true
on_T1_R1_S2=true
on_T2_R4_S1=true
on_T2_R4_S2=true
```

The following connects assignments reflect the new topology.

These reflect the original connections:

```
connects_R1_R2=true
connects_R2_R3=true
connects_R3_R4=true
connects_R4_R1=true
```

These reflect the new connections that a rail is connected to itself:

```
connects_R1_R1=true
connects_R2_R2=true
connects_R3_R3=true
connects_R4_R4=true
```

All of the other connections are false:

```
connects_R1_R3=false
connects_R1_R4=false
connects_R2_R1=false
connects_R2_R4=false
connects_R3_R1=false
connects_R3_R2=false
connects_R4_R2=false
connects_R4_R3=false
```

The following "legal" assignments show that it is legal for train 1 to move to any rail other than rail 4, which is the location of train 2 (in situation 1).

```
legal_T1_R1_R1=true
legal_T1_R1_R2=true
legal_T1_R1_R3=true
legal_T1_R1_R4=false
legal_T1_R2_R1=true
legal_T1_R2_R2=true
legal_T1_R2_R3=true
legal_T1_R2_R4=false
legal_T1_R3_R1=true
legal_T1_R3_R2=true
legal_T1_R3_R3=true
legal_T1_R3_R4=false
legal_T1_R4_R1=true
legal_T1_R4_R2=true
legal_T1_R4_R3=true
legal_T1_R4_R4=false
```

The following "legal" assignments show that it is legal for train 2 to move to any rail other than rail 1, which is the location of train 1 (in situation 1).

```
legal_T2_R1_R1=false
legal_T2_R1_R2=true
legal_T2_R1_R3=true
legal_T2_R1_R4=true
legal_T2_R2_R1=false
legal_T2_R2_R2=true
legal_T2_R2_R3=true
legal_T2_R2_R4=true
legal_T2_R3_R1=false
legal_T2_R3_R2=true
legal_T2_R3_R3=true
legal_T2_R3_R4=true
legal_T2_R4_R1=false
legal_T2_R4_R2=true
legal_T2_R4_R3=true
legal_T2_R4_R4=true
```

Safe assignment:

task5safe.txt.out

```
connects_R1_R1=true
connects_R1_R2=true
connects_R1_R3=false
connects_R1_R4=false
connects_R2_R1=false
connects_R2_R2=true
connects_R2_R3=true
connects_R2_R4=false
connects_R3_R1=false
connects_R3_R2=false
connects_R3_R3=true
connects_R3_R4=true
connects_R4_R1=true
connects_R4_R2=false
connects_R4_R3=false
connects_R4_R4=true
legal_T1_R1_R1=true
legal_T1_R1_R2=true
legal_T1_R1_R3=true
legal_T1_R1_R4=false
legal_T1_R2_R1=true
legal_T1_R2_R2=true
legal_T1_R2_R3=true
legal_T1_R2_R4=false
legal_T1_R3_R1=true
legal_T1_R3_R2=true
legal_T1_R3_R3=true
legal_T1_R3_R4=false
legal_T1_R4_R1=true
legal_T1_R4_R2=true
legal_T1_R4_R3=true
legal_T1_R4_R4=false
legal_T2_R1_R1=false
legal_T2_R1_R2=true
legal_T2_R1_R3=true
legal_T2_R1_R4=true
legal_T2_R2_R1=false
legal_T2_R2_R2=true
legal_T2_R2_R3=true
legal_T2_R2_R4=true
legal_T2_R3_R1=false
legal_T2_R3_R2=true
legal_T2_R3_R3=true
legal_T2_R3_R4=true
legal_T2_R4_R1=false
legal_T2_R4_R2=true
legal_T2_R4_R3=true
legal_T2_R4_R4=true
on_T1_R1_S1=true
on_T1_R1_S2=true
on_T1_R2_S1=false
on_T1_R2_S2=false
on_T1_R3_S1=false
on_T1_R3_S2=false
on_T1_R4_S1=false
on_T1_R4_S2=false
on_T2_R1_S1=false
on_T2_R1_S2=false
on_T2_R2_S1=false
on_T2_R2_S2=false
on_T2_R3_S1=false
on_T2_R3_S2=false
on_T2_R4_S1=true
on_T2_R4_S2=true
safe_S1=true
safe_S2=true
```

Task 6: Consider Layout 2, shown in figure 4. Demonstrate that it is unsafe, given your definition of legal from item 5.

The following is the homework language description of layout 2:

```
 connects(R1, R1) ^  connects(R1, R2) ^ ~connects(R1, R3) ^  connects(R1, R4) ^
 connects(R2, R1) ^  connects(R2, R2) ^ ~connects(R2, R3) ^ ~connects(R2, R4) ^
~connects(R3, R1) ^  connects(R3, R2) ^  connects(R3, R3) ^  connects(R3, R4) ^
~connects(R4, R1) ^ ~connects(R4, R2) ^  connects(R4, R3) ^  connects(R4, R4) ^
```

How do you show it is not safe?

Preliminary Test:

I tested this first without asserting that there will be a wreck in S2.
It generated a safe scenario where T1 started on R1 and T2 started on R4. Both trains stayed were they were. This is a safe assignment, however, this does not tell us if it is possible to have an unsafe assignment.

Conclusive Test:

I then added the assertion that there is a train wreck in S2.

```
~safe(S2)^
```

I ran the DPLL sat solver on it and it found an unsafe situation is S2.

In this unsafe assignment, T1 started on R1, T2 started on R3, and both trains proceeded to R2, producing a train wreck. This proves that the situation is not guaranteed to be safe.

```
on_T1_R1_S1=true
on_T2_R3_S1=true

on_T1_R2_S2=true
on_T2_R2_S2=true
```

It is legal for both trains to move to rail 2, since there is no train on that rail in situation 1.

```
legal_T1_R1_R2=true
legal_T2_R3_R2=true
```

Therefore, an unsafe situation was created (situation 2).

```
safe_S1=true
safe_S2=false
```

The following is the unsafe assignment that was produced, which demonstrates that this setup is not guaranteed to be safe:

task6_wreck.txt.out

```
connects_R1_R2=true
connects_R1_R3=false
connects_R1_R4=true
connects_R2_R1=true
connects_R2_R2=true
connects_R2_R3=false
connects_R2_R4=false
connects_R3_R1=false
connects_R3_R2=true
connects_R3_R3=true
connects_R3_R4=true
connects_R4_R1=false
connects_R4_R2=false
connects_R4_R3=true
connects_R4_R4=true
legal_T1_R1_R1=true
legal_T1_R1_R2=true
legal_T1_R1_R3=false
legal_T1_R1_R4=true
legal_T1_R2_R1=true
legal_T1_R2_R2=true
legal_T1_R2_R3=false
legal_T1_R2_R4=true
legal_T1_R3_R1=true
legal_T1_R3_R2=true
legal_T1_R3_R3=false
legal_T1_R3_R4=true
legal_T1_R4_R1=true
legal_T1_R4_R2=true
legal_T1_R4_R3=false
legal_T1_R4_R4=true
legal_T2_R1_R1=false
legal_T2_R1_R2=true
legal_T2_R1_R3=true
legal_T2_R1_R4=true
legal_T2_R2_R1=false
legal_T2_R2_R2=true
legal_T2_R2_R3=true
legal_T2_R2_R4=true
legal_T2_R3_R1=false
legal_T2_R3_R2=true
legal_T2_R3_R3=true
legal_T2_R3_R4=true
legal_T2_R4_R1=false
legal_T2_R4_R2=true
legal_T2_R4_R3=true
legal_T2_R4_R4=true
on_T1_R1_S1=true
on_T1_R1_S2=false
on_T1_R2_S1=false
on_T1_R2_S2=true
on_T1_R3_S1=false
on_T1_R3_S2=false
on_T1_R4_S1=false
on_T1_R4_S2=false
on_T2_R1_S1=false
on_T2_R1_S2=false
on_T2_R2_S1=false
on_T2_R2_S2=true
on_T2_R3_S1=true
on_T2_R3_S2=false
on_T2_R4_S1=false
on_T2_R4_S2=false
safe_S1=true
safe_S2=false
```

Task 7: Write a new definition of legal, which outlaws the bad behavior demonstrated in the previous task.
Prove that there will be no crashes in this domain.
Show that there are safe configurations for S1 and S2. What are they?
Your new definition should not involve knowledge of the legality of other trains' possible moves (that is, you shouldn't have legal on both the left and right hand side of your axiom).

Present the new definition of "legal" in the homework language and in English.

The following is a new definition of legal which outlaws the bad behavior demonstrated in Task 6.

The bad behavior demonstrated in task 6 was that the trains collided. The trains can collide because the legal rule allowed only checked to see if a destination the rail of a given train was empty (or the given train was not on it).

The new rule will also check to see if there is a train (other than the given train) on any of the rails that connect to the possible destination rail.

It is legal for a given train (t1) to move from a given rail (r1) to a possible destination rail (r2) if and only if the given train is on the given rail and the possible destination rail, or there is no train on any of the rails that connect to the possible destination rail (r2) other than the given train (t1).

Notes:

The possible destination rail (r2) also connects to the possible destination rail (r2), so this legal definition implicitly requires that the possible destination rail is empty of the given train (t1) is on it.

All of these rules assume that situation 1 is safe.

```
(
  all t1 all r1 all r2(
      legal(t1,r1,r2)<->( on(t1,r1,S1)^on(t1,r2,S1) v
            (all r3 all t2 (
                  (connects(r3,r2) ^ on(t2,r3,S1))
                  -> Equals(t1,t2)
            ))


      )
  )
)^
```

Is the domain safe?

Yes, the domain is safe. In order to prove this, I ran Layout 2 with the assertion that there will be a wreck in situation 2(~safe(S2)).
I tried to solve for all of the axioms using DPLL. It was impossible to find a satisfying solution for the axioms, in conjunction with the not safe in S2 assertion. Therefore, no solution was found, returning null.

```
task7_prove_no_wrecks.txt.out

null
```

Therefore, it is not possible to generate an unsafe situation, which is proof that the domain is safe.

Show a safe configuration that you found (or more, if you found more than one) for S1 and S2.

For this layout, there are essentially 12 ways permutations of where the trains can be located. However, due to symmetries in the layout and in how the trains operate, only 3 are logically distinct. I created all 3 in order to show that the rules work properly in all 3.

There is also the issue of all the permutations of were each train could move to in situation 2. This, however, doesn't really matter, because the legal assignments tell us where it is legal for the trains to move, which is what we are interested in.

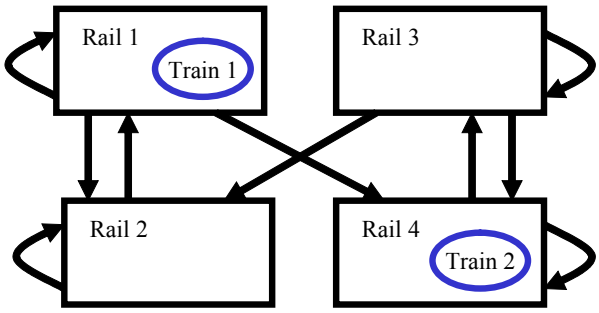
The first interpretation was produced by the axioms, without dictating where each train should start. The second and third interpretations dictated where the trains should start, so that we could look at the legal assignments in those interpretations.

First train location assignment:

Produced, T1 on R1, and T2 on R4.

```
on_T1_R1_S1=true
on_T2_R4_S1=true
```

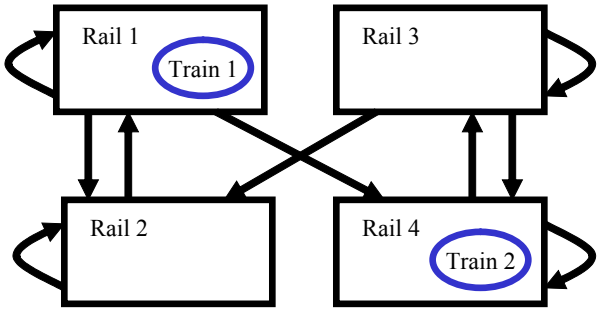
Task 4 Diagram : Layout 2 : Situation 1



Both trains stayed put.

```
on_T1_R1_S2=true
on_T2_R4_S2=true
```

Task 4 Diagram : Layout 2 : Situation 2



Legal Assignments:

It is legal for train 1 to move to Rail 2 or stay put.
Train 1 could not move to Rail 4 because train 2 is there.

```
legal_T1_R1_R1=true
legal_T1_R1_R2=true
legal_T1_R1_R3=false
legal_T1_R1_R4=false
```

It is legal for train 2 to move to Rail 3 or stay put.

```
legal_T2_R4_R1=false
legal_T2_R4_R2=false
legal_T2_R4_R3=true
legal_T2_R4_R4=true
```

The following is the entire assignment:

task7_find_safe_configuration2.txt.out

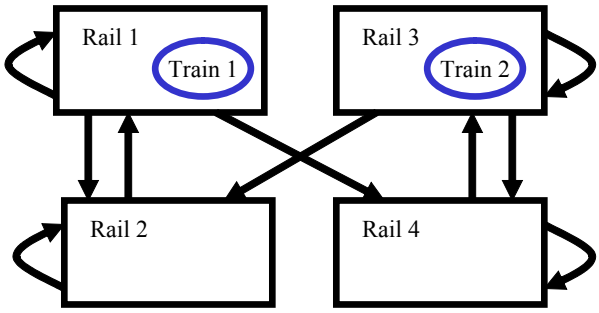
```
connects_R1_R1=true
connects_R1_R2=true
connects_R1_R3=false
connects_R1_R4=true
connects_R2_R1=true
connects_R2_R2=true
connects_R2_R3=false
connects_R2_R4=false
connects_R3_R1=false
connects_R3_R2=true
connects_R3_R3=true
connects_R3_R4=true
connects_R4_R1=false
connects_R4_R2=false
connects_R4_R3=true
connects_R4_R4=true
legal_T1_R1_R1=true
legal_T1_R1_R2=true
legal_T1_R1_R3=false
legal_T1_R1_R4=false
legal_T1_R2_R1=true
legal_T1_R2_R2=true
legal_T1_R2_R3=false
legal_T1_R2_R4=false
legal_T1_R3_R1=true
legal_T1_R3_R2=true
legal_T1_R3_R3=false
legal_T1_R3_R4=false
legal_T1_R4_R1=true
legal_T1_R4_R2=true
legal_T1_R4_R3=false
legal_T1_R4_R4=false
legal_T2_R1_R1=false
legal_T2_R1_R2=false
legal_T2_R1_R3=true
legal_T2_R1_R4=false
legal_T2_R2_R1=false
legal_T2_R2_R2=false
legal_T2_R2_R3=true
legal_T2_R2_R4=false
legal_T2_R3_R1=false
legal_T2_R3_R2=false
legal_T2_R3_R3=true
legal_T2_R3_R4=false
legal_T2_R4_R1=false
legal_T2_R4_R2=false
legal_T2_R4_R3=true
legal_T2_R4_R4=true
on_T1_R1_S1=true
on_T1_R1_S2=true
on_T1_R2_S1=false
on_T1_R2_S2=false
on_T1_R3_S1=false
on_T1_R3_S2=false
on_T1_R4_S1=false
on_T1_R4_S2=false
on_T2_R1_S1=false
on_T2_R1_S2=false
on_T2_R2_S1=false
on_T2_R2_S2=false
on_T2_R3_S1=false
on_T2_R3_S2=false
on_T2_R4_S1=true
on_T2_R4_S2=true
safe_S1=true
safe_S2=true
```

Second train location assignment:

Produced, T1 on R1, and T2 on R3.
T1 stayed on R1 and T2 stayed on R4.

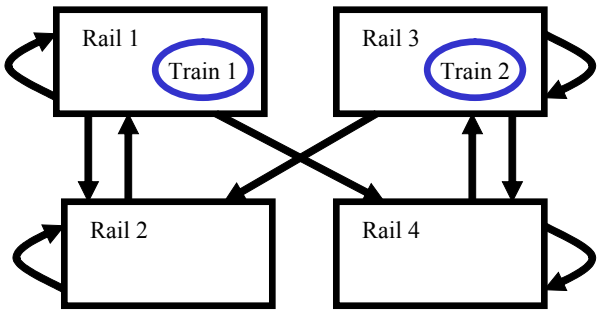
```
on_T1_R1_S1=true
on_T2_R3_S1=true
```

Task 4 Diagram : Layout 2 : Situation 1




```
on_T1_R1_S2=true
on_T2_R3_S2=true
```

Task 4 Diagram : Layout 2 : Situation 1



Legal Assignments:

Since, T1 is on R1 in S1 and T2 is on R3 in S1, the following are the only legal assignments that matter:

```
legal_T1_R1_R1=true
legal_T1_R1_R2=false
legal_T1_R1_R3=false
legal_T1_R1_R4=false
```

```
legal_T2_R3_R1=false
legal_T2_R3_R2=false
legal_T2_R3_R3=true
legal_T2_R3_R4=false
```

As you can see, in this setup, the only legal move for either train is to stay put. This is because, the other train (respectively) is also connected to R2 or R4, which are possible destination rails.

The following is the entire assignment:

task7_find_safe_configuration2.txt.out

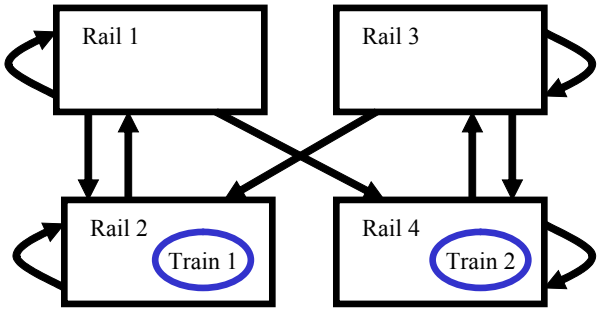
```
connects_R1_R1=true
connects_R1_R2=true
connects_R1_R3=false
connects_R1_R4=true
connects_R2_R1=true
connects_R2_R2=true
connects_R2_R3=false
connects_R2_R4=false
connects_R3_R1=false
connects_R3_R2=true
connects_R3_R3=true
connects_R3_R4=true
connects_R4_R1=false
connects_R4_R2=false
connects_R4_R3=true
connects_R4_R4=true
legal_T1_R1_R1=true
legal_T1_R1_R2=false
legal_T1_R1_R3=false
legal_T1_R1_R4=false
legal_T1_R2_R1=true
legal_T1_R2_R2=false
legal_T1_R2_R3=false
legal_T1_R2_R4=false
legal_T1_R3_R1=true
legal_T1_R3_R2=false
legal_T1_R3_R3=false
legal_T1_R3_R4=false
legal_T1_R4_R1=true
legal_T1_R4_R2=false
legal_T1_R4_R3=false
legal_T1_R4_R4=false
legal_T2_R1_R1=false
legal_T2_R1_R2=false
legal_T2_R1_R3=true
legal_T2_R1_R4=false
legal_T2_R2_R1=false
legal_T2_R2_R2=false
legal_T2_R2_R3=true
legal_T2_R2_R4=false
legal_T2_R3_R1=false
legal_T2_R3_R2=false
legal_T2_R3_R3=true
legal_T2_R3_R4=false
legal_T2_R4_R1=false
legal_T2_R4_R2=false
legal_T2_R4_R3=true
legal_T2_R4_R4=false
on_T1_R1_S1=true
on_T1_R1_S2=true
on_T1_R2_S1=false
on_T1_R2_S2=false
on_T1_R3_S1=false
on_T1_R3_S2=false
on_T1_R4_S1=false
on_T1_R4_S2=false
on_T2_R1_S1=false
on_T2_R1_S2=false
on_T2_R2_S1=false
on_T2_R2_S2=false
on_T2_R3_S1=true
on_T2_R3_S2=true
on_T2_R4_S1=false
on_T2_R4_S2=false
safe_S1=true
safe_S2=true
```

Third train location assignment:

Produced, T1 on R2, and T2 on R4.
T1 moved to R1, T2 stayed on R4.

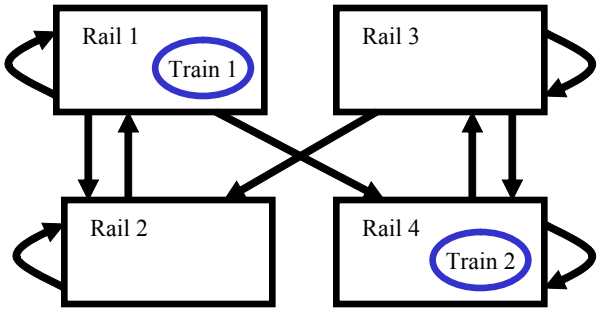
```
on_T1_R2_S1=true
on_T2_R4_S1=true
```

Task 4 Diagram : Layout 2 : Situation 1




```
on_T1_R1_S2=true
on_T2_R4_S2=true
```

Task 4 Diagram : Layout 2 : Situation 2



Legal Assignments:

Since, T1 is on R2 in S1 and T2 is on R4 in S1, the trains can move to any connected nodes. It is not legal to move to the other 2 rails, because the other train is wither on that rail, or on a rail connected to that rail.

```
legal_T1_R2_R1=true
legal_T1_R2_R2=true
legal_T1_R2_R3=false
legal_T1_R2_R4=false
```

```
legal_T2_R4_R1=false
legal_T2_R4_R2=false
legal_T2_R4_R3=true
legal_T2_R4_R4=true
```

The following is the entire assignment:

task7_find_safe_configuration2.txt.out

```
connects_R1_R1=true
connects_R1_R2=true
connects_R1_R3=false
connects_R1_R4=true
connects_R2_R1=true
connects_R2_R2=true
connects_R2_R3=false
connects_R2_R4=false
connects_R3_R1=false
connects_R3_R2=true
connects_R3_R3=true
connects_R3_R4=true
connects_R4_R1=false
connects_R4_R2=false
connects_R4_R3=true
connects_R4_R4=true
legal_T1_R1_R1=true
legal_T1_R1_R2=true
legal_T1_R1_R3=false
legal_T1_R1_R4=false
legal_T1_R2_R1=true
legal_T1_R2_R2=true
legal_T1_R2_R3=false
legal_T1_R2_R4=false
legal_T1_R3_R1=true
legal_T1_R3_R2=true
legal_T1_R3_R3=false
legal_T1_R3_R4=false
legal_T1_R4_R1=true
legal_T1_R4_R2=true
legal_T1_R4_R3=false
legal_T1_R4_R4=false
legal_T2_R1_R1=false
legal_T2_R1_R2=false
legal_T2_R1_R3=true
legal_T2_R1_R4=true
legal_T2_R2_R1=false
legal_T2_R2_R2=false
legal_T2_R2_R3=true
legal_T2_R2_R4=true
legal_T2_R3_R1=false
legal_T2_R3_R2=false
legal_T2_R3_R3=true
legal_T2_R3_R4=true
legal_T2_R4_R1=false
legal_T2_R4_R2=false
legal_T2_R4_R3=true
legal_T2_R4_R4=true
on_T1_R1_S1=false
on_T1_R1_S2=true
on_T1_R2_S1=true
on_T1_R2_S2=false
on_T1_R3_S1=false
on_T1_R3_S2=false
on_T1_R4_S1=false
on_T1_R4_S2=false
on_T2_R1_S1=false
on_T2_R1_S2=false
on_T2_R2_S1=false
on_T2_R2_S2=false
on_T2_R3_S1=false
on_T2_R3_S2=false
on_T2_R4_S1=true
on_T2_R4_S2=true
safe_S1=true
safe_S2=true
```

Task 8: One way to avoid crashes is never to move.
So, we'd also like to show that the system doesn't become deadlocked.
The system is deadlocked if there is no train that can move.
Add an axiom that constrains S1 to be deadlocked.
Using your definition of legal from item 7 show that there is no deadlock in layout 1.

The following is the axiom which asserts that S1 is deadlocked:

```
(
 all t1 all r1 (
      on(t1,r1,S1) ->   (
            all r2 (
                  (connects(r1,r2)^~Equals(r1,r2)) -> (
                        exists t2 exists r3 (
                          (connects(r3,r2)^on(t2,r3,S1)^~Equals(t1,t2))
                              )
                        )
                  )
            )
      )
)^
```

This works because the following must be true for all trains and all rails:

> If a given train is on a given rail is S1,
> then all rails which it is connected to (other than itself) there is another rail that connects to it, with another train (not itself) on it.

Notes:

If there is no train, on a given rail, then this axiom is satisfied (true) for that particular rail.

If there is a train on a given rail, but the rail is not connected to the particular r2 being considered, then the axiom is satisfied (true) for that particular t1, r1 and r2.

If there is a train on a given rail, and the rail is connected to the particular r2 being considered, but the r2 is really just r1, then the axiom is satisfied (true) for that particular t1, r1 and r2.

If there is a train on a given rail, and the rail is connected to the particular r2 being considered, and the r2 is not r1, then the axiom is satisfied (true) only if there exists a train (t2) on one of the rails that leads to r2 which is not the given train (t1).

Method for showing no deadlock:

The following is a description of the method I used to show no deadlock in layout 1:

I ran the DPLL sat solver on the domain to see if it could find a satisfying assignment. If it found a satisfying assignment, which would mean that it could create a situation that is deadlocked.

A satisfying assignment could not be found, therefore, there is no configuration that produces a deadlock.

```
task8_reduce.txt.out

null
```

This is correct because layout 1 is 4 rails in a cycle. Therefore, the trains are either next to each other or separated by 1 rail. Therefore, at least 1 of the trains will be able to move forward.

Diagram : Layout 1

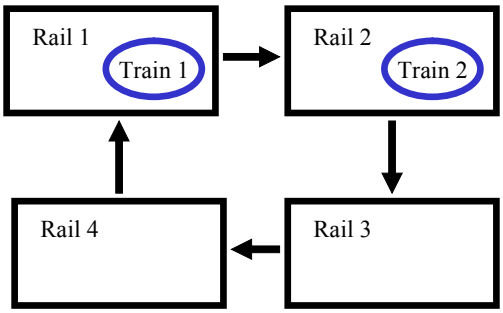
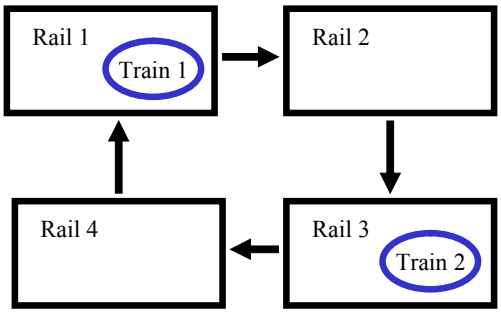


Diagram : Layout 1



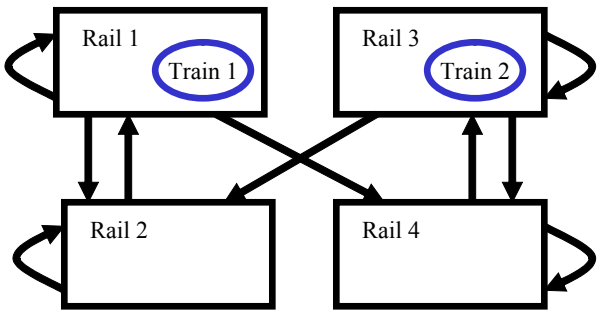
Task 9: Show that there can be a deadlock in layout 2. What is the on relation in your example?

```
To show that there can be a deadlock in layout 2, I ran the DPLL sat
solver with the axiom (from Task 8) that asserts that situation 1 is
deadlocked.  A satisfying deadlocked assignment was found which shows
that the domain can be deadlocked.
```

The following are the members of the ON relation, if your proof method produced an assignment.

```
on_T1_R1_S1=true
on_T1_R1_S2=true
on_T1_R2_S1=false
on_T1_R2_S2=false
on_T1_R3_S1=false
on_T1_R3_S2=false
on_T1_R4_S1=false
on_T1_R4_S2=false
on_T2_R1_S1=false
on_T2_R1_S2=false
on_T2_R2_S1=false
on_T2_R2_S2=false
on_T2_R3_S1=true
on_T2_R3_S2=true
on_T2_R4_S1=false
on_T2_R4_S2=false
```

Task 9 Diagram : Layout 2 : Situation 1



The sat solver produced and assignment with T1 on R1, and T2 on R3.

Since, T1 is on R1 in S1 and T2 is on R3 in S1, the only legal moves are for both trains to stay put:

```
legal_T1_R1_R1=true
legal_T1_R1_R2=false
legal_T1_R1_R3=false
legal_T1_R1_R4=false
legal_T2_R3_R1=false
legal_T2_R3_R2=false
legal_T2_R3_R3=true
legal_T2_R3_R4=false
```

As you can see, in this setup, the only legal move for either train is to stay put. This is because, the other train (respectively) is also connected to R2 or R4, which are possible destination rails.

A printout of the assignment, if you have one.

task9test.txt.out

```
connects_R1_R1=true
connects_R1_R2=true
connects_R1_R3=false
connects_R1_R4=true
connects_R2_R1=true
connects_R2_R2=true
connects_R2_R3=false
connects_R2_R4=false
connects_R3_R1=false
connects_R3_R2=true
connects_R3_R3=true
connects_R3_R4=true
connects_R4_R1=false
connects_R4_R2=false
connects_R4_R3=true
connects_R4_R4=true
legal_T1_R1_R1=true
legal_T1_R1_R2=false
legal_T1_R1_R3=false
legal_T1_R1_R4=false
legal_T1_R2_R1=true
legal_T1_R2_R2=false
legal_T1_R2_R3=false
legal_T1_R2_R4=false
legal_T1_R3_R1=true
legal_T1_R3_R2=false
legal_T1_R3_R3=false
legal_T1_R3_R4=false
legal_T1_R4_R1=true
legal_T1_R4_R2=false
legal_T1_R4_R3=false
legal_T1_R4_R4=false
legal_T2_R1_R1=false
legal_T2_R1_R2=false
legal_T2_R1_R3=true
legal_T2_R1_R4=false
legal_T2_R2_R1=false
legal_T2_R2_R2=false
legal_T2_R2_R3=true
legal_T2_R2_R4=false
legal_T2_R3_R1=false
legal_T2_R3_R2=false
legal_T2_R3_R3=true
legal_T2_R3_R4=false
legal_T2_R4_R1=false
legal_T2_R4_R2=false
legal_T2_R4_R3=true
legal_T2_R4_R4=false
on_T1_R1_S1=true
on_T1_R1_S2=true
on_T1_R2_S1=false
on_T1_R2_S2=false
on_T1_R3_S1=false
on_T1_R3_S2=false
on_T1_R4_S1=false
on_T1_R4_S2=false
on_T2_R1_S1=false
on_T2_R1_S2=false
on_T2_R2_S1=false
on_T2_R2_S2=false
on_T2_R3_S1=true
on_T2_R3_S2=true
on_T2_R4_S1=false
on_T2_R4_S2=false
safe_S1=true
safe_S2=true
```

Task 10: Revise your definition of legal so that you no longer can have a deadlock in layout 2. (Hint: you may need to add an extra relation on trains, which can mention them by name. However, your axioms for train movement, legality, etc. should not mention trains by name.) Give a satisfying assignment for the situation in which T1 is on R1 and T2 is on R3 in S1.

New definition of legal:

In my new definition of legal, it is legal for a given train to move to a given destination rail if and only if there are no trains on any of the rails which connect to the given destination rail, other than the given train, or there in no train on the given destination rail, and the given train has precedence (`iamit`).

```
(
 all t1 all r1 all r2(
      legal(t1,r1,r2)<->(
            all r3 (
                  connects(r3,r2)->(
                        ~(
                              exists t2 (
                                    on(t2,r3,S1)^~Equals(t1,t2)
                              )
                         )
                  )
            ) v
            (
                  all t3 ((~on(t3,r2,S1)) ^ deadlocked(S1) ^ iamit(t1))
            )
      )
 )
)^
```

Other new axioms or relations that I added:

In order to make the new definition of legal work, I gave train 1 precedence. In other words, if none of the trains have moves that would definitely be safe, train 1 has precedence to move to an empty rail. In configuration 2, this will work, if neither train has a move that is definitely safe, both trains are connected to a rail that is currently empty.

A more generic way to award precedence is to construct an axiom that says one train or the other has precedence. This technique, however, is flawed, because it leaves the decision up to the sat solver (DPLL algorithm) which then acts as an omniscient scheduler.

It would also be more generic to cause the precedence to change from situation to situation while defining the precedence for only situation 1. However, the legal definition does not take situation into consideration. It is applicable, really, only to situation 1.

Therefore, there is no more appropriate way to set precedence then to just define it for situation 1.

In order to be generic with respect to the number of trains, I set the precedence of T1 (to true), and require that the precedence of all of the other trains (really there is just 1 other train) to the opposite (false). That way, this would work, no matter how many trains there are.

The axiom works as follows. If `iamit(t1)` is false, then we don't care what `iamit(t2)` is. However, if `iamit(t1)` is true (i.e. it is T1), then all of the others will have to be false, for the axiom to be satisfied.

```
iamit(T1)^

(
      all t1 all t2 (
            ( iamit(t1)->~iamit(t2) ) v Equals(t1,t2)
      )
)^
```

Enforce movement

In order to make sure that the moves would be executed appropriately, I added an axiom that requires the train that has precedence must move. This was just for testing purposes. I ran the program with and without this axiom. The variables are assigned appropriately in both situations. When movement is not enforced, train 1 (with precedence) has legal moves, but chooses not to move. When movement is enforced, train 1 (with precedence) has legal moves, and chooses to move to rail 4.

```
(all t1 all r1 (
                  iamit(t1) -> (
                        on(t1,r1,S1) -> ~on(t1,r1,S2)
                  )
            )
)^
```

I rearranged my deadlock assertion for this task so that I could use it more easily. I named it, so that it could be used in the legal axiom and so that I could turn it on and off for testing.

The "nomoves" predicate defines whether or not a given train can move to a given rail in a given situation.

```
(
 all t1 all r1 all s nomoves(t1,r1,s) <-> (
      on(t1,r1,s) -> (
            all r2 (
                  ( connects(r1,r2)^~Equals(r1,r2) ) -> (
                        exists t2 exists r3 (
                        (connects(r3,r2)^on(t2,r3,s)^~Equals(t1,t2))
                        )
                  )
            )
      )
 )
)^
```

I named the "deadlocked" assertion, so that I could switch it on and off. The "deadlocked" assertion pertains to the entire situation.

```
(
      all s deadlocked(s) <-> (
            all t1 all r1 nomoves(t1,r1,s)
      )
)^
```

Here I switch it on, for situation 1.

```
deadlocked(S1)^
```

A printout of the assignment for the given arrangement.

The results are in the table below.
The results show that the trains started in R1 and R3.
Train 1 was given precedence; therefore, it was legal for Train 1 to move to R2 or R4 or R1.

When movement was enforced, certain interpretation variables changed for situation 1. Since train 1 moved to rail 4, train 2, on rail 3 had a legal move (to rail2) in situation 2. Therefore, situation 2 is not deadlocked.

Without enforcing movement task10v3.txt.out	**With enforcing movement:** task10v3.txt.out
connects_R1_R1=true	connects_R1_R1=true
connects_R1_R2=true	connects_R1_R2=true
connects_R1_R3=false	connects_R1_R3=false
connects_R1_R4=true	connects_R1_R4=true
connects_R2_R1=true	connects_R2_R1=true
connects_R2_R2=true	connects_R2_R2=true

connects_R2_R3=false	connects_R2_R3=false
connects_R2_R4=false	connects_R2_R4=false
connects_R3_R1=false	connects_R3_R1=false
connects_R3_R2=true	connects_R3_R2=true
connects_R3_R3=true	connects_R3_R3=true
connects_R3_R4=true	connects_R3_R4=true
connects_R4_R1=false	connects_R4_R1=false
connects_R4_R2=false	connects_R4_R2=false
connects_R4_R3=true	connects_R4_R3=true
connects_R4_R4=true	connects_R4_R4=true
deadlocked_S1=true	deadlocked_S1=true
deadlocked_S2=true	deadlocked_S2=false
iamit_T1=true	iamit_T1=true
iamit_T2=false	iamit_T2=false
legal_T1_R1_R1=true	legal_T1_R1_R1=true
legal_T1_R1_R2=true	legal_T1_R1_R2=true
legal_T1_R1_R3=false	legal_T1_R1_R3=false
legal_T1_R1_R4=true	legal_T1_R1_R4=true
legal_T1_R2_R1=true	legal_T1_R2_R1=true
legal_T1_R2_R2=true	legal_T1_R2_R2=true
legal_T1_R2_R3=false	legal_T1_R2_R3=false
legal_T1_R2_R4=true	legal_T1_R2_R4=true
legal_T1_R3_R1=true	legal_T1_R3_R1=true
legal_T1_R3_R2=true	legal_T1_R3_R2=true
legal_T1_R3_R3=false	legal_T1_R3_R3=false
legal_T1_R3_R4=true	legal_T1_R3_R4=true
legal_T1_R4_R1=true	legal_T1_R4_R1=true
legal_T1_R4_R2=true	legal_T1_R4_R2=true
legal_T1_R4_R3=false	legal_T1_R4_R3=false
legal_T1_R4_R4=true	legal_T1_R4_R4=true
legal_T2_R1_R1=false	legal_T2_R1_R1=false
legal_T2_R1_R2=false	legal_T2_R1_R2=false
legal_T2_R1_R3=true	legal_T2_R1_R3=true
legal_T2_R1_R4=false	legal_T2_R1_R4=false
legal_T2_R2_R1=false	legal_T2_R2_R1=false
legal_T2_R2_R2=false	legal_T2_R2_R2=false
legal_T2_R2_R3=true	legal_T2_R2_R3=true
legal_T2_R2_R4=false	legal_T2_R2_R4=false
legal_T2_R3_R1=false	legal_T2_R3_R1=false
legal_T2_R3_R2=false	legal_T2_R3_R2=false
legal_T2_R3_R3=true	legal_T2_R3_R3=true
legal_T2_R3_R4=false	legal_T2_R3_R4=false
legal_T2_R4_R1=false	legal_T2_R4_R1=false
legal_T2_R4_R2=false	legal_T2_R4_R2=false
legal_T2_R4_R3=true	legal_T2_R4_R3=true
legal_T2_R4_R4=false	legal_T2_R4_R4=false
nomoves_T1_R1_S1=true	nomoves_T1_R1_S1=true
nomoves_T1_R1_S2=true	nomoves_T1_R1_S2=true
nomoves_T1_R2_S1=true	nomoves_T1_R2_S1=true
nomoves_T1_R2_S2=true	nomoves_T1_R2_S2=true
nomoves_T1_R3_S1=true	nomoves_T1_R3_S1=true
nomoves_T1_R3_S2=true	nomoves_T1_R3_S2=true
nomoves_T1_R4_S1=true	nomoves_T1_R4_S1=true
nomoves_T1_R4_S2=true	nomoves_T1_R4_S2=true
nomoves_T2_R1_S1=true	nomoves_T2_R1_S1=true
nomoves_T2_R1_S2=true	nomoves_T2_R1_S2=true

nomoves_T2_R2_S1=true	nomoves_T2_R2_S1=true
nomoves_T2_R2_S2=true	nomoves_T2_R2_S2=true
nomoves_T2_R3_S1=true	nomoves_T2_R3_S1=true
nomoves_T2_R3_S2=true	nomoves_T2_R3_S2=false
nomoves_T2_R4_S1=true	nomoves_T2_R4_S1=true
nomoves_T2_R4_S2=true	nomoves_T2_R4_S2=true
on_T1_R1_S1=true	on_T1_R1_S1=true
on_T1_R1_S2=true	on_T1_R1_S2=false
on_T1_R2_S1=false	on_T1_R2_S1=false
on_T1_R2_S2=false	on_T1_R2_S2=false
on_T1_R3_S1=false	on_T1_R3_S1=false
on_T1_R3_S2=false	on_T1_R3_S2=false
on_T1_R4_S1=false	on_T1_R4_S1=false
on_T1_R4_S2=false	on_T1_R4_S2=true
on_T2_R1_S1=false	on_T2_R1_S1=false
on_T2_R1_S2=false	on_T2_R1_S2=false
on_T2_R2_S1=false	on_T2_R2_S1=false
on_T2_R2_S2=false	on_T2_R2_S2=false
on_T2_R3_S1=true	on_T2_R3_S1=true
on_T2_R3_S2=true	on_T2_R3_S2=true
on_T2_R4_S1=false	on_T2_R4_S1=false
on_T2_R4_S2=false	on_T2_R4_S2=false
safe_S1=true	safe_S1=true
safe_S2=true	safe_S2=true

Task 11: It might seem like the implication in formula 5 should really be a bi-conditional. Explain in English what that would mean, and why it would cause problems in our axiomatization.

The following is Formula 5 expressed in the homework language.

Dynamics: Only make legal moves to connected rails.

```
(
 all t all r2 (
      on(t,r2,S2) -> exists r1 (
            on(t,r1,S1) ^ connects(r1,r2) ^ legal(t,r1,r2)
      )
 )
) ^
```

The above axiom uses implication, rather than a bi-conditional. The original axiom requires that if a train is on a given rail in situation 2, then it was on a rail that is connected to the given rail is situation 1 and it is legal to move to from the connected rail to the given rail.

Changing Formula 5 to a bi-conditional changes its' meaning. Its' new meaning includes the former meaning. However, the bi-conditional also requires that if a train is on a given rail in situation 1, and it is connected to another rail, and it is legal to move to that rail, then it will be on that rail in situation 2. In other words, any legal move to a connected rail will be made.

Therefore, a given train will be on multiple rails in situation 2, if it has more than 1 possible move. This is actually impossible to satisfy, unless we allow deadlock, because of this axiom:

Axiom 2 (Task 2) : No train is on two rails at the same time.

```
(
 all t all s all r1 (
      on(t,r1,s) -> (
            all r2 (
                  on(t,r2,s)->Equals(r1,r2)
            )
      )
 )
)^
```

If deadlock was allowed, then the trains could stay on the same rail, which would be the one and only legal move to a connected rail. However, deadlock is not allowed, therefore, this would also cause unsafe situations for many assignments.

Our axioms are meant to represent something in the real world (i.e. a simulation). This change is akin to the manifestation of multiple futures in a single space-time.

Task 12 : Extra Credit: Our definition of liveness (lack of deadlock) in section 8 is weak, because there still might be some train that never gets to move. But requiring every train to be able to move on every step is too strong. How could you axiomatize the notion that every train eventually gets to move? Would it work to try to prove or disprove such a liveness condition using the methods of this assignment? Why or why not?

My answer to this question assumes that the axiom for liveness should be in a form akin to the way deadlock was axiomatized. In other words, I would require that a train will not always be in the same place, over all of the situations.

```
(all t1 all r1 all s1(
                        on(t1,r1,s1) -> (
                              exists s2 ~on(t1,r1,s2)
                        )
            )
)^
```

In order to test this in our situation, I would then include the negation of this axiom, in the axiom set, and attempt to solve in using DPLL sat solver. The sat solver could return null, which would mean that all trains eventually get to move. Otherwise, it could return a satisfying assignment where at least one train never gets to move.

This would not be appropriate if using the given methods for 2 reasons. First, since there are just 2 situations, this requirement is the same as requiring that every train move in every situation. This is explicitly not the goal, and this would also restrict the train locations, to locations where they could simultaneously move. For example, it would not be possible for a train to be on R1 and R3.

In sum, testing on just 2 situations would make the dynamics very constrained, and it just doesn't accomplish our goal.

Second, in order to test this we would therefore need to add multiple situations with an ordering (a time frame). The legal definition would need to be able to impose an ordering on the situations. Perhaps they could be joined in a series as follows:

```
next(S1,S2) ^
next(S2,S3) ^
next(S3,S4) ^
```

Then the legal axiom could use this ordering to determine if a move is legal in a given situation.

In theory, however, you would need an infinite number of states. You cannot have a universe of infinite size in the system that we are using. Even if you could, you could

never return a satisfying assignment (where some train never gets to move) because you would never know if nobody ever gets to move.

For practical purposes, in our small layout, we would be able to prove that it worked (every train eventually gets to move) for a small number of situations. But we would not be able to be sure that it does not work (no one ever gets to move).