Lawrence Bush
Project 1 - Part 2 (Theorem Prover)

6.825
Professor: Leslie Pack Kaelbling
October 13, 2004, 5:00 PM

Project 1, Part 2

Task1:

Formalized Axioms:

```
(all x man(x) -> mortal(x))
(all x mortal(x) -> boring(x))
~boring(Hera)
~(exists x ~man(x))
```

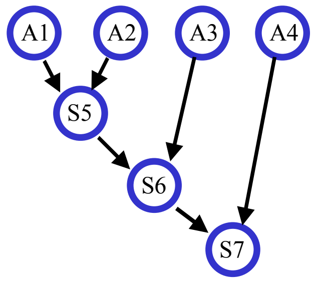
Proof, Version 1:

```
A1: ~man(x2) v mortal(x2)
A2: ~mortal(x4) v boring(x4)
A3: ~boring(CHera)
A4:  man(x6)
S5: ~man(x7) v boring(x7)      Res(1,2) {x2/x4}
S6: ~man(CHera)                Res(5,3) {x7/CHera}
S7:  F                         Res(6,4) {x6/CHera}
```

Proof Sketch:

Proving forward from "all men are mortal" to "Hera is not boring" can solve this proof. First, we resolve, "all men are mortal", with "all mortals are boring", to get "all men are boring." This is logically stated as "if you are a man then you are boring." We then resolve, "if you are a man then you are boring" with "Hera is not boring" to get "Hera is not a man." We conclude by resolving, "Hera is not a man" with the negation of the axiom "There exists someone who is not a man."

Diagram : Task 1 Proof Version 1


$$\begin{tikzcd}
A1 \arrow[dr] & & A2 \arrow[dl] & A3 \arrow[ddl] & A4 \arrow[dddl] \\
& S5 \arrow[dr] & & & \\
& & S6 \arrow[dr] & & \\
& & & S7 &
\end{tikzcd}$$

Key Steps:

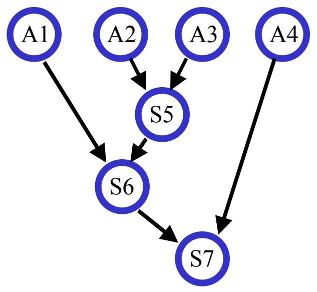
This proof is entirely linear. None of the proof steps are key.

Proof, Version 2:

```
A1: ~man(x2) v mortal(x2)
A2: ~mortal(x4) v boring(x4)
A3: ~boring(CHera)
A4:  man(x6)
S5: ~mortal(CHera)      Res(2,3) {x4/CHera}
S6: ~man(CHera)         Res(5,1) {x2/CHera}
S7:  F                  Res(6,4) {x6/CHera}
```

This can also be done, proving backward from to “Hera is not boring” to “all men are mortal.” First, we resolve, “all mortals are boring” with “Hera is not boring” to get “Hera is not mortal.” We then resolve “Hera is not mortal” with “all men are mortal” to get “Hera is not a man.” We conclude by resolving, “Hera is not a man” with the negation of the axiom “There exists someone who is not a man.”

Diagram : Task 1 Proof Version 2



Task 2:

Proof:

```
A1:  son(Father(CA), Father(CMe))
A2: ~sib(x14, CMe)
A3: ~sib(x18, y17) v Equals(Father(x18), Father(y17))
A4: ~sib(x20, y19) v ~Equals(x20, y19)
A5: ~Equals(Father(x22), Father(y21)) v Equals(x22, y21) v
      sib(x22, y21)
A6: ~son(x26, y25) v Equals(Father(x26), y25)
A7: ~Equals(Father(x28), y27) v son(x28, y27)
A8: ~son(CA, CMe)
S9: Equals(Father(Father(CA)), Father(CMe))
      Res(1,6) {y25/Father(CMe), x26/Father(CA)}
S10: Equals(Father(CA), CMe) v sib(Father(CA), CMe)
      Res(9,5) {y21/CMe, x22/Father(CA)}
S11: Equals(Father(CA), CMe)        Res(10,2) {x14/Father(CA)}
S12: son(CA, CMe)                   Res(11,7) {y27/CMe, x28/CA}
S13: F                              Res(12,8) {}
```

In this proof, “that man” is represented by the constant “A.”

Since that man’s father is my father’s son, and I don’t have any siblings, then that man’s father must be me, and that man must be my son. However, we have to use the sibling relationship to connect the son relationship with the fact that I have no siblings.

To resolve this proof, I started with the first axiom, “That man’s father is my father’s son.” This can only be resolved with A6: “If x is the son of y, then the father of x is y.” These resolve to “That man’s Grand father is my father.”

I then resolve, “That man’s Grand father is my father” with “if x and y have the same father, then x and y are siblings or x and y are the same person.” I then get, either “that man’s father is me,” or “that man’s father is my sibling.”

I then eliminate the possibility that that man’s father is my sibling by resolving it with “I don’t have any siblings” to get “that man’s father is me.” I then resolve this with “ if the father of x is y then x is the son of y” to get “that man is my son!”

This resolves with the conclusion of the proof.

Another way to show this is to first use the sibling relationship and the fact that I have no siblings to show that the only one who can have my father as their father is me.

```
S9: ~Equals(Father(x25), Father(CMe)) v Equals(x25, CMe)
      Res(2,5) {x14/x22, y21/CMe}
```

I then show that my son's grandfather is my father.

```
S10: ~Equals(Father(Father(x31)), Father(CMe)) v son(x31, CMe)
Res(9,7) {y27/CMe, x25/Father(x28)}
```

I then show that my son's father is the son of my father.

```
S11: son(x32, CMe) v ~son(Father(x32), Father(CMe))      Res(10,6)
{y25/Father(CMe), x26/Father(x31)}

S12: son(CA, CMe)      Res(11,1) {x32/CA}
```

I then show that A is my son which resolves the proof.

```
S13: F      Res(12,8) {}
```

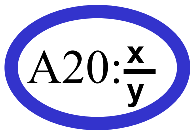
Task 3.

Printout of the Proof :

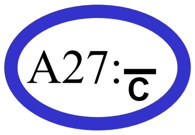
```
A1: ~Equals(Ca, Cb)
A2: ~Equals(Ca, Cc)
A3: ~Equals(Cb, Ca)
A4: ~Equals(Cb, Cc)
A5: ~Equals(Cc, Ca)
A6: ~Equals(Cc, Cb)
A7: ~Equals(Ca, Ctable)
A8: ~Equals(Cb, Ctable)
A9: ~Equals(Cc, Ctable)
A10: block(Ca)
A11: block(Cb)
A12: block(Cc)
A13: ~block(Ctable)
A14: ~on(x12, y11, s10) v ~on(x12, z13, s10) v Equals(y11, z13)
A15: ~on(x20, z19, s18) v ~on(y21, z19, s18) v Equals(x20, y21) v
Equals(z19, Ctable)
A16: ~clear(x27, s26) v ~block(y28) v ~on(y28, x27, s26) v Equals(x27,
Ctable)
A17: block(F_1(y31, x30, s29)) v clear(x30, s29)
A18: on(F_1(y24, x33, s32), x33, s32) v clear(x33, s32)
A19: ~Equals(x34, Ctable) v clear(x34, s35)
A20: ~clear(x41, s40) v ~clear(y42, s40) v ~on(x41, z43, s40) v on(x41,
y42, result(move(x41, y42), s40))
A21: ~clear(x45, s44) v ~clear(y46, s44) v ~on(x45, z47, s44) v
clear(z47, result(move(x45, y46), s44))
A22: ~on(x55, y54, s53) v Equals(x55, z56) v on(x55, y54,
result(move(z56, w52), s53))
A23: ~clear(x62, s61) v Equals(x62, z63) v clear(x62, result(move(y64,
z63), s61))
A24: on(Ca, Ctable, Cs0)
A25: on(Cc, Ca, Cs0)
A26: on(Cb, Ctable, Cs0)
A27: clear(Cc, Cs0)
A28: clear(Cb, Cs0)
A29: clear(Ctable, s66)
A30: ~on(Cb, Cc, s68) v answer(s68)
S31: ~clear(x69, Cs0) v ~on(x69, z70, Cs0) v on(x69, Cc,
result(move(x69, Cc), Cs0))      Res(20,27) {s40/Cs0, y42/Cc}
S32: ~on(Cb, z72, Cs0) v on(Cb, Cc, result(move(Cb, Cc), Cs0))
Res(31,28) {x69/Cb}
S33: on(Cb, Cc, result(move(Cb, Cc), Cs0))      Res(32,26) {z72/Ctable}
S34: answer(result(move(Cb, Cc), Cs0))      Res(33,30)
{s68/result(move(Cb, Cc), Cs0)}
```

Proof Description:

To resolve the proof, we need to refute the assertion that B cannot be on C in some situation. The following diagram outlines the process. The diagram shows a brief representation of what a given axiom says about the world. The For example,



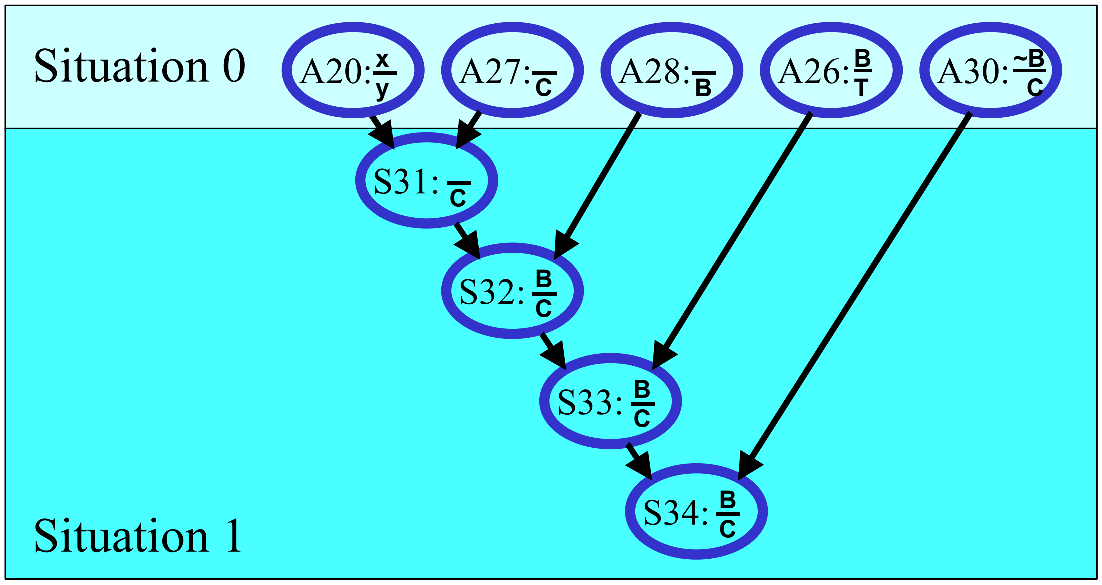
means that variable x is on variable y and



means that C is clear.

This is a shorthand representation, which aides in following the proof. To get the full and accurate meaning of a give axiom, you have to look at the axiom.

Diagram : Task 3



Ultimately, we want to prove Axiom 30:

```
      Axiom 30 : ~exists s on(b,c,s) ^ ~answer(s)
```

The proof checker represents it as follows:

```
      A30: ~on(Cb, Cc, s68) v answer(s68)
```

To do this, we satisfy the following axiom.

Axiom 20-21

If x and y are clear, and x is on z, then moving x to y results in x being on y and z being clear.

```
all s all x all y all z clear(x,s) ^ clear(y,s) ^ on (x,z,s) ->
    on(x,y,result(move(x,y),s)) ^ clear(z,result(move(x,y),s))
```

In other words, we are going to move B onto C. If we do, then the result is "B is on C." The axioms say that this can only be done if there is nothing on top of B or C and B is on z, where z can be anything. In our case, B is on the table.

The proof checker splits this axiom up into:

```
A20: ~clear(x41, s40) v ~clear(y42, s40) v ~on(x41, z43, s40) v on(x41,
y42, result(move(x41, y42), s40))
```

and

```
A21: ~clear(x45, s44) v ~clear(y46, s44) v ~on(x45, z47, s44) v
clear(z47, result(move(x45, y46), s44))
```

We only need to satisfy A20, because we want to show that x is on y (B is on C in particular). A21 would resolve to z is clear.

We have to show that B and C are both clear, in order to make the move.

```
A27: clear(Cc, Cs0)
A28: clear(Cb, Cs0)
```

However, we also have to show that B is on Z (in this case the table).

```
A26: on(Cb, Ctable, Cs0)
```

This seems unnecessary, but this requirement comes from the original axiom so that it can simultaneously declare that Z is clear (if we wanted that). In our case, we chose the subsequent clause that shows that B will be on C.

In essence, we have to resolve A26, A27 and A28 with A20.

A20 is the actual move. It is contingent on A26, A27 and A28.

```
A20: ~clear(x99, s98) v ~clear(y100, s98) v ~on(x99, z101, s98) v
on(x99, y100, result(move(x99, y100), s98))
```

The order doesn't matter, but the contingencies need to be resolved with the action.

Therefore, I resolved A20 with A27 to imply the move and show that C is clear. You have to make sure to choose the move which results in moving something to C.

```
S31: ~clear(x69, Cs0) v ~on(x69, z70, Cs0) v on(x69, Cc,
result(move(x69, Cc), Cs0))      Res(20,27) {s40/Cs0, y42/Cc}
```

I then resolved S31 with A28 to show that C is clear.

```
S32: ~on(Cb, z72, Cs0) v on(Cb, Cc, result(move(Cb, Cc), Cs0))
Res(31,28) {x69/Cb}
```

I then resolved S32 with A26 to show that B is on something (in this case the table).

```
S33: on(Cb, Cc, result(move(Cb, Cc), Cs0))      Res(32,26) {z72/Ctable}
```

I then resolved S33 with S30 (the answer) to get rid of the on statement and tuck the description of the move into the answer.

```
S34: answer(result(move(Cb, Cc), Cs0))      Res(33,30)
{s68/result(move(Cb, Cc), Cs0)}
```

Task 3 Extra Credit:

Printout of the proof:

```
A1: ~Equals(Ca, Cb)
A2: ~Equals(Ca, Cc)
A3: ~Equals(Cb, Ca)
A4: ~Equals(Cb, Cc)
A5: ~Equals(Cc, Ca)
A6: ~Equals(Cc, Cb)
A7: ~Equals(Ca, Ctable)
A8: ~Equals(Cb, Ctable)
A9: ~Equals(Cc, Ctable)
A10: block(Ca)
A11: block(Cb)
A12: block(Cc)
A13: ~block(Ctable)
A14: ~on(x168, y167, s166) v ~on(x168, z169, s166) v Equals(y167, z169)
A15: ~on(x176, z175, s174) v ~on(y177, z175, s174) v Equals(x176, y177)
v Equals(z175, Ctable)
A16: ~clear(x183, s182) v ~block(y184) v ~on(y184, x183, s182)
A17: ~clear(x186, s185) v ~Equals(x186, Ctable)
A18: block(F_3(y189, x188, s187)) v Equals(x188, Ctable) v clear(x188,
s187)
A19: on(F_3(y180, x191, s190), x191, s190) v Equals(x191, Ctable) v
clear(x191, s190)
A20: ~clear(x197, s196) v ~clear(y198, s196) v ~on(x197, z199, s196) v
on(x197, y198, result(move(x197, y198), s196))
A21: ~clear(x201, s200) v ~clear(y202, s200) v ~on(x201, z203, s200) v
clear(z203, result(move(x201, y202), s200))
A22: ~clear(x205, s204) v ~clear(y206, s204) v ~on(x205, z207, s204) v
clear(x205, result(move(x205, y206), s204))
A23: ~on(x215, y214, s213) v Equals(x215, z216) v on(x215, y214,
result(move(z216, w212), s213))
A24: ~clear(x222, s221) v Equals(x222, z223) v clear(x222,
result(move(y224, z223), s221))
A25: on(Ca, Ctable, Cs0)
A26: on(Cc, Ca, Cs0)
A27: on(Cb, Ctable, Cs0)
A28: clear(Cc, Cs0)
A29: clear(Cb, Cs0)
A30: clear(Ctable, s226)
A31: ~on(Ca, Cb, s228) v ~on(Cb, Cc, s228) v answer(s228)
S32: Equals(Ca, z45) v on(Ca, Ctable, result(move(z45, w46), Cs0))
Res(23,25) {y214/Ctable, x215/Ca, s213/Cs0}
S33: on(Ca, Ctable, result(move(Cc, w54), Cs0))      Res(32,2) {z45/Cc}
S34: Equals(Cb, z1) v on(Cb, Ctable, result(move(z1, w2), Cs0))
Res(23,27) {x215/Cb, s213/Cs0, y214/Ctable}
S35: on(Cb, Ctable, result(move(Cc, w3), Cs0))      Res(34,4) {z1/Cc}
S36: ~clear(Cc, Cs0) v ~clear(y4, Cs0) v clear(Ca, result(move(Cc, y4),
Cs0))      Res(21,26) {x201/Cc, z203/Ca, s200/Cs0}
S37: ~clear(y5, Cs0) v clear(Ca, result(move(Cc, y5), Cs0))
Res(36,28) {}
```

```
S38: clear(Ca, result(move(Cc, Ctable), Cs0))      Res(37,30)
{y5/Ctable, s226/Cs0}
S39: Equals(Cb, z6) v clear(Cb, result(move(y7, z6), Cs0))
Res(24,29) {s221/Cs0, x222/Cb}
S40: clear(Cb, result(move(y8, Ctable), Cs0))      Res(39,8)
{z6/Ctable}
S41: ~clear(x10, s9) v ~on(x10, z11, s9) v clear(x10, result(move(x10,
Ctable), s9))      Res(22,30) {s204/s226, y206/Ctable}
S42: ~on(Cc, z12, Cs0) v clear(Cc, result(move(Cc, Ctable), Cs0))
Res(41,28) {s9/Cs0, x10/Cc}
S43: clear(Cc, result(move(Cc, Ctable), Cs0))      Res(42,26) {z12/Ca}
S44: Equals(Ca, z13) v on(Ca, Ctable, result(move(z13, w212),
result(move(Cc, w14), Cs0)))      Res(33,23) {x215/Ca,
s213/result(move(Cc, w54), Cs0), y214/Ctable}
S45: on(Ca, Ctable, result(move(Cb, w16), result(move(Cc, w15), Cs0)))
Res(44,1) {z13/Cb}
S46: ~clear(Cb, result(move(Cc, w17), Cs0)) v ~clear(y18,
result(move(Cc, w17), Cs0)) v on(Cb, y18, result(move(Cb, y18),
result(move(Cc, w17), Cs0)))      Res(35,20) {z199/Ctable,
s196/result(move(Cc, w3), Cs0), x197/Cb}
S47: ~clear(Cb, result(move(Cc, Ctable), Cs0)) v on(Cb, Cc,
result(move(Cb, Cc), result(move(Cc, Ctable), Cs0)))      Res(43,46)
{w17/Ctable, y18/Cc}
S48: on(Cb, Cc, result(move(Cb, Cc), result(move(Cc, Ctable), Cs0)))
Res(40,47) {y8/Cc}
S49: Equals(Ca, z19) v clear(Ca, result(move(y20, z19), result(move(Cc,
Ctable), Cs0)))      Res(38,24) {s221/result(move(Cc, Ctable), Cs0),
x222/Ca}
S50: clear(Ca, result(move(y21, Cc), result(move(Cc, Ctable), Cs0)))
Res(49,2) {z19/Cc}
S51: ~clear(x22, result(move(Cc, Ctable), Cs0)) v ~on(x22, z23,
result(move(Cc, Ctable), Cs0)) v clear(x22, result(move(x22, Cc),
result(move(Cc, Ctable), Cs0)))      Res(43,22) {s204/result(move(Cc,
Ctable), Cs0), y206/Cc}
S52: ~clear(Cb, result(move(Cc, Ctable), Cs0)) v clear(Cb,
result(move(Cb, Cc), result(move(Cc, Ctable), Cs0)))      Res(35,51)
{z23/Ctable, x22/Cb, w3/Ctable}
S53: clear(Cb, result(move(Cb, Cc), result(move(Cc, Ctable), Cs0)))
Res(40,52) {y8/Cc}
S54: ~clear(Ca, result(move(Cb, w25), result(move(Cc, w24), Cs0))) v
~clear(y26, result(move(Cb, w25), result(move(Cc, w24), Cs0))) v on(Ca,
y26, result(move(Ca, y26), result(move(Cb, w25), result(move(Cc, w24),
Cs0))))      Res(45,20) {z199/Ctable, s196/result(move(Cb, w16),
result(move(Cc, w15), Cs0)), x197/Ca}
S55: ~clear(y27, result(move(Cb, Cc), result(move(Cc, Ctable), Cs0))) v
on(Ca, y27, result(move(Ca, y27), result(move(Cb, Cc), result(move(Cc,
Ctable), Cs0))))      Res(50,54) {w24/Ctable, y21/Cb, w25/Cc}
S56: on(Ca, Cb, result(move(Ca, Cb), result(move(Cb, Cc),
result(move(Cc, Ctable), Cs0))))      Res(53,55) {y27/Cb}
S57: Equals(Cb, z28) v on(Cb, Cc, result(move(z28, w29),
result(move(Cb, Cc), result(move(Cc, Ctable), Cs0))))      Res(48,23)
{x215/Cb, s213/result(move(Cb, Cc), result(move(Cc, Ctable), Cs0)),
y214/Cc}
S58: on(Cb, Cc, result(move(Ca, w30), result(move(Cb, Cc),
result(move(Cc, Ctable), Cs0))))      Res(57,3) {z28/Ca}
```

```
S59: ~on(Cb, Cc, result(move(Ca, Cb), result(move(Cb, Cc),
result(move(Cc, Ctable), Cs0)))) v answer(result(move(Ca, Cb),
result(move(Cb, Cc), result(move(Cc, Ctable), Cs0))))      Res(56,31)
{s228/result(move(Ca, Cb), result(move(Cb, Cc), result(move(Cc,
Ctable), Cs0)))}
S60: answer(result(move(Ca, Cb), result(move(Cb, Cc), result(move(Cc,
Ctable), Cs0))))      Res(58,59) {w30/Cb}
```

Proof Explanation: The following is an explanation of the proof steps.

We start in Situation 0 with the facts that:

```
      A is on the table,
      C is on A,
      B is on the table,
      C is clear,
      B is clear,
      The table is clear.
```

Note: The the table is always considered clear.

These facts are expressed in the following axioms:

```
A25: on(Ca, Ctable, Cs0)
A26: on(Cc, Ca, Cs0)
A27: on(Cb, Ctable, Cs0)
A28: clear(Cc, Cs0)
A29: clear(Cb, Cs0)
A30: clear(Ctable, s226)
```

We then rearrange the blocks. When we rearrange the blocks, we end up in a different situation. For convenience, I number the situations from 0 (initial situation) to 3 (final situation).

Really, we are just proving what the result of moving the blocks would be, based on the axioms. If we wanted to prove that the result of moving C onto the table is that C is on the table, we could do that. That was the result of the previous proof. However, that result is not something that we need to prove in this proof. Rather, we want to prove the following results (outlined below), which are needed later in the proof. For example, in order to prove that B is on C in Situation 2, we need to have already proven that B is clear, B is on the table, and C is clear in Situation 1.

Outline:

```
Move C to Table (Situation 1):
      Prove: A is on the table
      Prove: B is on the table
      Prove: A is clear
      Prove: B is clear
      Prove: C is clear

Move B to C (Situation 2):
      Prove: A is on the table
      Prove: B is on C
      Prove: A is clear
      Prove: B is clear

Move A onto B (Situation 3):

      Prove: A is on B
      Prove: B is on C

Resolve Answer (A is on B ^ B is on C):
      Prove: The final resolved answer.
```

The following is a diagram of the steps:

Diagram : Task 3 Extra Credit : Situation 0

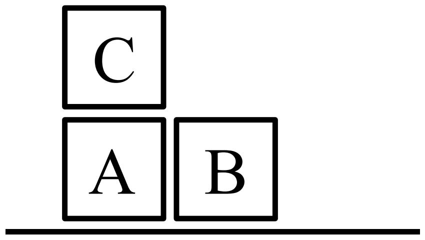


Diagram : Task 3 Extra Credit : Situation 1

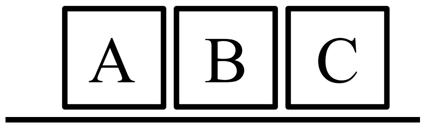


Diagram : Task 3 Extra Credit : Situation 2

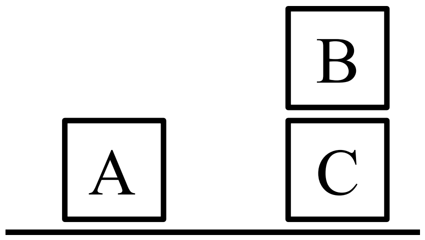
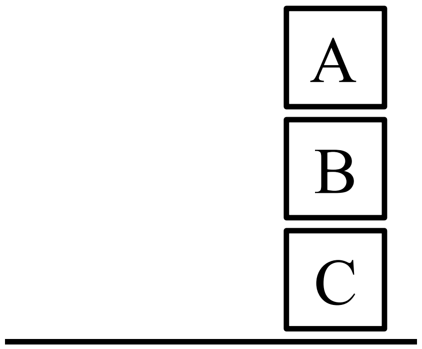


Diagram : Task 3 Extra Credit : Situation 3



Move C to Table (Situation 1):

If we move C to the table, we need to prove the following:

Prove: A is on the table

```
S32: Equals(Ca, z45) v on(Ca, Ctable, result(move(z45, w46), Cs0))
Res(23,25) {y214/Ctable, x215/Ca, s213/Cs0}

S33: on(Ca, Ctable, result(move(Cc, w54), Cs0))      Res(32,2) {z45/Cc}
```

Prove: B is on the table

```
S34: Equals(Cb, z1) v on(Cb, Ctable, result(move(z1, w2), Cs0))
Res(23,27) {x215/Cb, s213/Cs0, y214/Ctable}

S35: on(Cb, Ctable, result(move(Cc, w3), Cs0))      Res(34,4) {z1/Cc}
```

Prove: A is clear

```
S36: ~clear(Cc, Cs0) v ~clear(y4, Cs0) v clear(Ca, result(move(Cc, y4),
Cs0))      Res(21,26) {x201/Cc, z203/Ca, s200/Cs0}

S37: ~clear(y5, Cs0) v clear(Ca, result(move(Cc, y5), Cs0))
Res(36,28) {}

S38: clear(Ca, result(move(Cc, Ctable), Cs0))      Res(37,30)
{y5/Ctable, s226/Cs0}
```

Prove: B is clear

```
S39: Equals(Cb, z6) v clear(Cb, result(move(y7, z6), Cs0))
Res(24,29) {s221/Cs0, x222/Cb}

S40: clear(Cb, result(move(y8, Ctable), Cs0))      Res(39,8)
{z6/Ctable}
```

Prove: C is clear

```
S41: ~clear(x10, s9) v ~on(x10, z11, s9) v clear(x10, result(move(x10,
Ctable), s9))      Res(22,30) {s204/s226, y206/Ctable}

S42: ~on(Cc, z12, Cs0) v clear(Cc, result(move(Cc, Ctable), Cs0))
Res(41,28) {s9/Cs0, x10/Cc}

S43: clear(Cc, result(move(Cc, Ctable), Cs0))      Res(42,26) {z12/Ca}
```

Move B to C (Situation 2):

If we move B onto C, we need to prove the following:

Prove: A is on the table

S44: Equals(Ca, z13) v on(Ca, Ctable, result(move(z13, w212), result(move(Cc, w14), Cs0))) Res(33,23) {x215/Ca, s213/result(move(Cc, w54), Cs0), y214/Ctable}

S45: on(Ca, Ctable, result(move(Cb, w16), result(move(Cc, w15), Cs0))) Res(44,1) {z13/Cb}

Prove: B is on C

S46: ~clear(Cb, result(move(Cc, w17), Cs0)) v ~clear(y18, result(move(Cc, w17), Cs0)) v on(Cb, y18, result(move(Cb, y18), result(move(Cc, w17), Cs0))) Res(35,20) {z199/Ctable, s196/result(move(Cc, w3), Cs0), x197/Cb}

S47: ~clear(Cb, result(move(Cc, Ctable), Cs0)) v on(Cb, Cc, result(move(Cb, Cc), result(move(Cc, Ctable), Cs0))) Res(43,46) {w17/Ctable, y18/Cc}

S48: on(Cb, Cc, result(move(Cb, Cc), result(move(Cc, Ctable), Cs0))) Res(40,47) {y8/Cc}

Prove: A is clear

S49: Equals(Ca, z19) v clear(Ca, result(move(y20, z19), result(move(Cc, Ctable), Cs0))) Res(38,24) {s221/result(move(Cc, Ctable), Cs0), x222/Ca}

S50: clear(Ca, result(move(y21, Cc), result(move(Cc, Ctable), Cs0))) Res(49,2) {z19/Cc}

Prove: B is clear

S51: ~clear(x22, result(move(Cc, Ctable), Cs0)) v ~on(x22, z23, result(move(Cc, Ctable), Cs0)) v clear(x22, result(move(x22, Cc), result(move(Cc, Ctable), Cs0))) Res(43,22) {s204/result(move(Cc, Ctable), Cs0), y206/Cc}

S52: ~clear(Cb, result(move(Cc, Ctable), Cs0)) v clear(Cb, result(move(Cb, Cc), result(move(Cc, Ctable), Cs0))) Res(35,51) {z23/Ctable, x22/Cb, w3/Ctable}

S53: clear(Cb, result(move(Cb, Cc), result(move(Cc, Ctable), Cs0))) Res(40,52) {y8/Cc}

Move A onto B (Situation 3):

If we move A onto B, we need to prove the following:

Prove: A is on B

S54: ~clear(Ca, result(move(Cb, w25), result(move(Cc, w24), Cs0))) v ~clear(y26, result(move(Cb, w25), result(move(Cc, w24), Cs0))) v on(Ca, y26, result(move(Ca, y26), result(move(Cb, w25), result(move(Cc, w24), Cs0)))) Res(45,20) {z199/Ctable, s196/result(move(Cb, w16), result(move(Cc, w15), Cs0)), x197/Ca}

S55: ~clear(y27, result(move(Cb, Cc), result(move(Cc, Ctable), Cs0))) v on(Ca, y27, result(move(Ca, y27), result(move(Cb, Cc), result(move(Cc, Ctable), Cs0)))) Res(50,54) {w24/Ctable, y21/Cb, w25/Cc}

S56: on(Ca, Cb, result(move(Ca, Cb), result(move(Cb, Cc), result(move(Cc, Ctable), Cs0)))) Res(53,55) {y27/Cb}

Prove: B is on C

S57: Equals(Cb, z28) v on(Cb, Cc, result(move(z28, w29), result(move(Cb, Cc), result(move(Cc, Ctable), Cs0)))) Res(48,23) {x215/Cb, s213/result(move(Cb, Cc), result(move(Cc, Ctable), Cs0)), y214/Cc}

S58: on(Cb, Cc, result(move(Ca, w30), result(move(Cb, Cc), result(move(Cc, Ctable), Cs0)))) Res(57,3) {z28/Ca}

We then fill in the answer by resolving A is on B, and B is on C with the "Answer" axiom.

S59: ~on(Cb, Cc, result(move(Ca, Cb), result(move(Cb, Cc), result(move(Cc, Ctable), Cs0)))) v answer(result(move(Ca, Cb), result(move(Cb, Cc), result(move(Cc, Ctable), Cs0)))) Res(56,31) {s228/result(move(Ca, Cb), result(move(Cb, Cc), result(move(Cc, Ctable), Cs0)))}

Resolve Answer (A is on B ^ B is on C):

Prove: The final resolved answer.

The following is the final resolved answer:

S60: answer(result(move(Ca, Cb), result(move(Cb, Cc), result(move(Cc, Ctable), Cs0)))) Res(58,59) {w30/Cb}

Task 4.

Proof Printout:

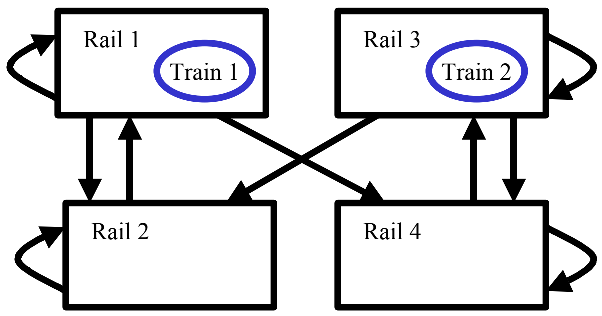
```
A1: rail(CR1)
A2: rail(CR2)
A3: rail(CR3)
A4: rail(CR4)
A5: ~rail(x2) v Equals(x2, CR1) v Equals(x2, CR2) v Equals(x2, CR3) v
Equals(x2, CR4)
A6: ~Equals(CR1, CR2)
A7: ~Equals(CR1, CR3)
A8: ~Equals(CR1, CR4)
A9: ~Equals(CR2, CR1)
A10: ~Equals(CR2, CR3)
A11: ~Equals(CR2, CR4)
A12: ~Equals(CR3, CR1)
A13: ~Equals(CR3, CR2)
A14: ~Equals(CR3, CR4)
A15: ~Equals(CR4, CR1)
A16: ~Equals(CR4, CR2)
A17: ~Equals(CR4, CR3)
A18: situation(CS1)
A19: situation(CS2)
A20: ~Equals(CS1, CS2)
A21: ~Equals(CS2, CS1)
A22: train(CT1)
A23: train(CT2)
A24: ~Equals(CT1, CT2)
A25: ~Equals(CT2, CT1)
A26: ~train(t4) v Equals(t4, CT1) v Equals(t4, CT2)
A27: connects(CR1, CR2)
A28: connects(CR3, CR2)
A29: connects(CR1, CR4)
A30: connects(CR3, CR4)
A31: connects(CR2, CR1)
A32: connects(CR4, CR3)
A33: ~connects(CR1, CR3)
A34: connects(CR1, CR1)
A35: ~connects(CR2, CR3)
A36: ~connects(CR2, CR4)
A37: connects(CR2, CR2)
A38: ~connects(CR3, CR1)
A39: connects(CR3, CR3)
A40: ~connects(CR4, CR2)
A41: ~connects(CR4, CR1)
A42: connects(CR4, CR4)
A43: ~on(t11, r110, s9) v ~on(t11, r212, s9) v Equals(r110, r212)
A44: rail(F_1(t17, s16))
A45: on(t19, F_1(t19, s18), s18)
A46: ~safe(s27) v ~on(t129, r28, s27) v ~on(t230, r28, s27) v
Equals(t129, t230)
A47: on(F_3(t223, t122, r21, s31), F_2(t223, t122, r21, s31), s31) v
safe(s31)
```

```
A48: on(F_4(t223, t122, r21, s32), F_2(t223, t122, r21, s32), s32) v
safe(s32)
A49: ~Equals(F_3(t236, t135, r34, s33), F_4(t236, t135, r34, s33)) v
safe(s33)
A50: ~on(t41, r240, CS2) v on(t41, F_5(r240, t41), CS1)
A51: ~on(t43, r242, CS2) v connects(F_5(r242, t43), r242)
A52: ~on(t45, r244, CS2) v legal(t45, F_5(r244, t45), r244)
A53: ~legal(t153, r152, r251) v Equals(r152, r251) v ~train(t254) v
~on(t254, r251, CS1)
A54: ~Equals(r156, r255) v legal(t157, r156, r255)
A55: train(F_6(t261, r260, r159, t158)) v legal(t158, r159, r260)
A56: on(F_6(t249, r262, r163, t164), r262, CS1) v legal(t164, r163,
r262)
A57: safe(CS1)
A58: on(CT1, CR1, CS1)
A59: on(CT2, CR3, CS1)
A60: ~legal(CT1, CR1, CR2)
S61: ~train(F_6(t212, r211, r110, t19)) v Equals(F_6(t212, r211, r110,
t19), CT2) v on(CT1, r211, CS1) v legal(t19, r110, r211)
Para(26,56) {t4/F_6(t249, r262, r163, t164)}
S62: ~train(F_6(t266, r265, r164, t163)) v on(CT1, r265, CS1) v
legal(t163, r164, r265) v on(CT2, r265, CS1) v legal(t163, r164, r265)
Para(61,56) {t19/t164, r211/r262, r110/r163, t212/t249}
S63: on(CT1, r267, CS1) v legal(t169, r168, r267) v on(CT2, r267, CS1)
v legal(t169, r168, r267) v legal(t169, r168, r267)      Res(62,55)
{r164/r159, t266/t261, r265/r260, t163/t158}
S64: on(CT1, r21, CS1) v on(CT2, r21, CS1) v legal(t13, r12, r21) v
legal(t13, r12, r21)      Fact(63) {}
S65: on(CT1, r24, CS1) v on(CT2, r24, CS1) v legal(t16, r15, r24)
Fact(64) {}
S66: ~on(t12, CR3, s11) v ~on(t12, CR2, s11)      Res(13,43) {r212/CR2,
r110/CR3}
S67: ~on(CT2, CR2, CS1)      Res(66,59) {t12/CT2, s11/CS1}
S68: ~on(t14, CR1, s13) v ~on(t14, CR2, s13)      Res(6,43) {r212/CR2,
r110/CR1}
S69: ~on(CT1, CR2, CS1)      Res(68,58) {s13/CS1, t14/CT1}
S70: on(CT1, CR2, CS1) v legal(t116, r115, CR2)      Res(65,67)
{r24/CR2}
S71: legal(t118, r117, CR2)      Res(69,70) {}
S72: F      Res(71,60) {t118/CT1, r117/CR1}
```

Proof Explanation:

In this proof, we wish to prove that it is legal for T1 to move to R2.
We know that Train 1 is on Rail 1 and Train 2 is on Rail 3. We also know that there are no other trains. The initial setup looks as follows:

Diagram : Initial Setup



This proof is broken into 2 main parts.

In the first part, we use the fact that there are only 2 trains (A26),

```
all t train(t) -> Equals(t,T1) v Equals(t,T2)
```

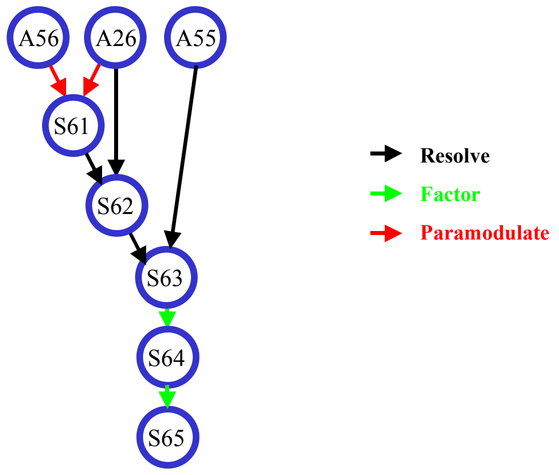
and if there is no train on a given track, then it is legal to move there,

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

to deduce that if neither T1 or T2 are on a given track, then it is legal to move there.

```
S65: on(CT1, r24, CS1) v on(CT2, r24, CS1) v legal(t16, r15, r24)
```

Diagram : Task 4 Part 1



The proof checker steps are as follows:

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

is broken up into A55 and A56 and is connected by the skolem funcion F_6.

```
A55: train(F_6(t261, r260, r159, t158)) v legal(t158, r159, r260)
A56: on(F_6(t249, r262, r163, t164), r262, CS1) v legal(t164, r163,
r262)

A26 says that there are only 2 trains.
A26: ~train(t4) v Equals(t4, CT1) v Equals(t4, CT2)
```

We paramodulate 56 with 26 twice because there are 2 trains and we will have to show that neither is on rail 2.

```
S61: ~train(F_6(t212, r211, r110, t19)) v Equals(F_6(t212, r211, r110,
t19), CT2) v on(CT1, r211, CS1) v legal(t19, r110, r211)
Para(26,56) {t4/F_6(t249, r262, r163, t164)}


S62: ~train(F_6(t266, r265, r164, t163)) v on(CT1, r265, CS1) v
legal(t163, r164, r265) v on(CT2, r265, CS1) v legal(t163, r164, r265)
Para(61,56) {t19/t164, r211/r262, r110/r163, t212/t249}
```

We then resolve this with A55, and factor it twice, to show that it is legal to move to a given rail if T1 and T2 are not on it.

```
S63: on(CT1, r267, CS1) v legal(t169, r168, r267) v on(CT2, r267, CS1)
v legal(t169, r168, r267) v legal(t169, r168, r267)      Res(62,55)
{r164/r159, t266/t261, r265/r260, t163/t158}


S64: on(CT1, r21, CS1) v on(CT2, r21, CS1) v legal(t13, r12, r21) v
legal(t13, r12, r21)      Fact(63) {}


S65: on(CT1, r24, CS1) v on(CT2, r24, CS1) v legal(t16, r15, r24)
Fact(64) {}
```

This result will be resolved after we generate the negation of the 2 “on” relations.

In the second part, we use the fact that 2 trains cannot be on the same track (A43), to prove that neither Train 1 nor Train 2 is on Rail 2 (the desired destination).

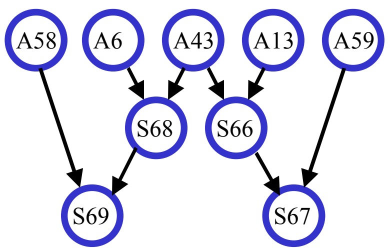
```
all s all t all r1 all r2 on(t, r1, s) ^ on(t, r2, s) -> Equals(r1, r2)
A43: ~on(t11, r110, s9) v ~on(t11, r212, s9) v Equals(r110, r212)
```

We know that Rail 1 and Rail 2 are not the same rail, and that Train 1 is on Rail 1 and Train 2 is on Rail 3.

```
A6:  ~Equals(CR1, CR2)
A13: ~Equals(CR3, CR2)
A58: on(CT1, CR1, CS1)
A59: on(CT2, CR3, CS1)
```

The proof checker steps are as follows:

Diagram : Task 4 Part 2



```
S66: ~on(t12, CR3, s11) v ~on(t12, CR2, s11)      Res(13,43) {r212/CR2,
r110/CR3}
S67: ~on(CT2, CR2, CS1)      Res(66,59) {t12/CT2, s11/CS1}
```

```
S68: ~on(t14, CR1, s13) v ~on(t14, CR2, s13)      Res(6,43) {r212/CR2,
r110/CR1}
S69: ~on(CT1, CR2, CS1)      Res(68,58) {s13/CS1, t14/CT1}
```

We then use these newly proven facts (that neither Train 1 nor Train 2 is on Rail 2) to resolve S65 from above.

```
S70: on(CT1, CR2, CS1) v legal(t116, r115, CR2)   Res(65,67) {r24/CR2}
S71: legal(t118, r117, CR2)      Res(69,70) {}
```

This result can resolve our proof:

```
S72: F      Res(71,60) {t118/CT1, r117/CR1}
```

Task 5:

The file trains2.txt contains the same axioms as the previous problem, but the conclusion is:

~(on(T1,R1,S1) ^ on(T2,R3,S1) -> ~safe(S2))

Now, we're asking you to show that, for this particular choice of initial conditions, the second situation is not safe.

Proof:

```
A1: rail(CR1)
A2: rail(CR2)
A3: rail(CR3)
A4: rail(CR4)
A5: ~rail(x2) v Equals(x2, CR1) v Equals(x2, CR2) v Equals(x2, CR3) v
Equals(x2, CR4)
A6: ~Equals(CR1, CR2)
A7: ~Equals(CR1, CR3)
A8: ~Equals(CR1, CR4)
A9: ~Equals(CR2, CR1)
A10: ~Equals(CR2, CR3)
A11: ~Equals(CR2, CR4)
A12: ~Equals(CR3, CR1)
A13: ~Equals(CR3, CR2)
A14: ~Equals(CR3, CR4)
A15: ~Equals(CR4, CR1)
A16: ~Equals(CR4, CR2)
A17: ~Equals(CR4, CR3)
A18: situation(CS1)
A19: situation(CS2)
A20: ~Equals(CS1, CS2)
A21: ~Equals(CS2, CS1)
A22: train(CT1)
A23: train(CT2)
A24: ~Equals(CT1, CT2)
A25: ~Equals(CT2, CT1)
A26: ~train(t4) v Equals(t4, CT1) v Equals(t4, CT2)
A27: connects(CR1, CR2)
A28: connects(CR3, CR2)
A29: connects(CR1, CR4)
A30: connects(CR3, CR4)
A31: connects(CR2, CR1)
A32: connects(CR4, CR3)
A33: ~connects(CR1, CR3)
A34: connects(CR1, CR1)
A35: ~connects(CR2, CR3)
A36: ~connects(CR2, CR4)
A37: connects(CR2, CR2)
A38: ~connects(CR3, CR1)
A39: connects(CR3, CR3)
```

```
A40: ~connects(CR4, CR2)
A41: ~connects(CR4, CR1)
A42: connects(CR4, CR4)
A43: ~on(t9, r18, CS1) v ~on(t9, r210, CS1) v Equals(r18, r210)
A44: rail(F_1(t15, s14))
A45: on(t17, F_1(t17, s16), s16)
A46: ~safe(s25) v ~on(t127, r26, s25) v ~on(t228, r26, s25) v
Equals(t127, t228)
A47: on(F_3(t221, t120, r19, s29), F_2(t221, t120, r19, s29), s29) v
safe(s29)
A48: on(F_4(t221, t120, r19, s30), F_2(t221, t120, r19, s30), s30) v
safe(s30)
A49: ~Equals(F_3(t234, t133, r32, s31), F_4(t234, t133, r32, s31)) v
safe(s31)
A50: ~on(t40, r239, CS2) v on(t40, F_5(r239, t40), CS1)
A51: ~on(t42, r241, CS2) v connects(F_5(r241, t42), r241)
A52: ~on(t44, r243, CS2) v legal(t44, F_5(r243, t44), r243)
A53: ~on(t46, r145, CS1) v ~connects(r145, r247) v ~legal(t46, r145,
r247) v on(t46, r247, CS2)
A54: ~legal(t155, r154, r253) v Equals(r154, r253) v ~train(t256) v
~on(t256, r253, CS1)
A55: ~Equals(r158, r257) v legal(t159, r158, r257)
A56: train(F_6(t263, r262, r161, t160)) v legal(t160, r161, r262)
A57: on(F_6(t251, r264, r165, t166), r264, CS1) v legal(t166, r165,
r264)
A58: safe(CS1)
A59: on(CT1, CR1, CS1)
A60: on(CT2, CR3, CS1)
A61: safe(CS2)
S62: ~train(F_6(t294, r293, r192, t191)) v Equals(F_6(t294, r293, r192,
t191), CT2) v on(CT1, r293, CS1) v legal(t191, r192, r293)
Para(26,57) {t4/F_6(t251, r264, r165, t166)}
S63: ~train(F_6(t2102, r2101, r1100, t199)) v on(CT1, r2101, CS1) v
legal(t199, r1100, r2101) v on(CT2, r2101, CS1) v legal(t199, r1100,
r2101)      Para(62,57) {t191/t166, r293/r264, t294/t251, r192/r165}
S64: on(CT1, r2103, CS1) v legal(t1105, r1104, r2103) v on(CT2, r2103,
CS1) v legal(t1105, r1104, r2103) v legal(t1105, r1104, r2103)
Res(63,56) {r1100/r161, t199/t160, r2101/r262, t2102/t263}
S65: ~on(t106, CR1, CS1) v ~on(t106, CR2, CS1)      Res(6,43)
{r210/CR2, r18/CR1}
S66: ~on(CT1, CR2, CS1)      Res(65,59) {t106/CT1}
S67: ~on(t107, CR3, CS1) v ~on(t107, CR2, CS1)      Res(13,43)
{r210/CR2, r18/CR3}
S68: ~on(CT2, CR2, CS1)      Res(67,60) {t107/CT2}
S69: legal(t1109, r1108, CR2) v on(CT2, CR2, CS1) v legal(t1109, r1108,
CR2) v legal(t1109, r1108, CR2)      Res(64,66) {r2103/CR2}
S70: legal(t1111, r1110, CR2) v legal(t1111, r1110, CR2) v legal(t1111,
r1110, CR2)      Res(68,69) {}
S71: legal(t1113, r1112, CR2) v legal(t1113, r1112, CR2)      Fact(70)
{}
S72: legal(t1115, r1114, CR2)      Fact(71) {}
S73: ~on(t117, r1116, CS1) v ~connects(r1116, CR2) v on(t117, CR2, CS2)
Res(72,53) {r247/CR2, r1114/r145, t1115/t46}
S74: ~connects(CR1, CR2) v on(CT1, CR2, CS2)      Res(73,59) {t117/CT1,
r1116/CR1}
```

```
S75: ~connects(CR3, CR2) v on(CT2, CR2, CS2)      Res(73,60) {t117/CT2,
r1116/CR3}
S76: on(CT1, CR2, CS2)      Res(74,27) {}
S77: on(CT2, CR2, CS2)      Res(75,28) {}
S78: ~safe(CS2) v ~on(t2118, CR2, CS2) v Equals(CT1, t2118)
Res(76,46) {s25/CS2, t127/CT1, r26/CR2}
S79: ~safe(CS2) v Equals(CT1, CT2)      Res(77,78) {t2118/CT2}
S80: ~safe(CS2)      Res(79,24) {}
S81: F      Res(80,61) {}
```

English explanation of the Proof:

For starters, the axioms in Trains1.txt and Trains2.txt differ for this particular axiom:

InTrains1.txt :

```
all s all t all r1 all r2 on(t, r1, s) ^ on(t, r2, s) -> Equals(r1, r2)
A43: ~on(t11, r110, s9) v ~on(t11, r212, s9) v Equals(r110, r212)

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

A50: ~on(t41, r240, CS2) v on(t41, F_5(r240, t41), CS1)
A51: ~on(t43, r242, CS2) v connects(F_5(r242, t43), r242)
A52: ~on(t45, r244, CS2) v legal(t45, F_5(r244, t45), r244)
```

In Trains2.txt :

This relation is restricted to S1, which means that the trains are not restricted to being on only 1 rail in S2.

```
all t all r1 all r2 on(t, r1, S1) ^ on(t, r2, S1) -> Equals(r1, r2)
A43: ~on(t9, r18, CS1) v ~on(t9, r210, CS1) v Equals(r18, r210)
```

This relation is made into a bi-conditional.

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

The bi-conditionality causes it to generate the following axioms, which includes an additional last "on" relation (A53),

```
A50: ~on(t40, r239, CS2) v on(t40, F_5(r239, t40), CS1)
A51: ~on(t42, r241, CS2) v connects(F_5(r241, t42), r241)
A52: ~on(t44, r243, CS2) v legal(t44, F_5(r243, t44), r243)
A53: ~on(t46, r145, CS1) v ~connects(r145, r247) v ~legal(t46, r145,
r247) v on(t46, r247, CS2)
```

Axioms 50 - 52 essentially say that if a train is on a given node in situation 2, then it was on connecting rail in situation 1, and the move was legal. These 3 axioms are connected by the skolem function F_5.

The additional axiom 53 says 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.

Axiom 54 combined with axiom 43 mean that a given train will be on multiple rails in situation 2, if it has more than 1 possible moves. This really doesn't make any sense. However, this is why we can prove that train 1 will be on rail 2 and rail 4 in situation 2, and train 2 will be on rail 2 and rail 4 in situation 2.

This set of axioms, also allows us to prove that situation 2 is not safe. Without the bi-conditional, trains could move to the same rail, but they wouldn't have to.

Proof:

I constructed the proof as follows:

Steps 62 – 72, essentially mirror the proof from Task 4, which results in the axiom that it is legal to move to Rail 2.

```
S62: ~train(F_6(t294, r293, r192, t191)) v Equals(F_6(t294, r293, r192,
t191), CT2) v on(CT1, r293, CS1) v legal(t191, r192, r293)
Para(26,57) {t4/F_6(t251, r264, r165, t166)}
S63: ~train(F_6(t2102, r2101, r1100, t199)) v on(CT1, r2101, CS1) v
legal(t199, r1100, r2101) v on(CT2, r2101, CS1) v legal(t199, r1100,
r2101)      Para(62,57) {t191/t166, r293/r264, t294/t251, r192/r165}
S64: on(CT1, r2103, CS1) v legal(t1105, r1104, r2103) v on(CT2, r2103,
CS1) v legal(t1105, r1104, r2103) v legal(t1105, r1104, r2103)
Res(63,56) {r1100/r161, t199/t160, r2101/r262, t2102/t263}
S65: ~on(t106, CR1, CS1) v ~on(t106, CR2, CS1)      Res(6,43)
{r210/CR2, r18/CR1}
S66: ~on(CT1, CR2, CS1)      Res(65,59) {t106/CT1}
S67: ~on(t107, CR3, CS1) v ~on(t107, CR2, CS1)      Res(13,43)
{r210/CR2, r18/CR3}
S68: ~on(CT2, CR2, CS1)      Res(67,60) {t107/CT2}
```

```
S69: legal(t1109, r1108, CR2) v on(CT2, CR2, CS1) v legal(t1109, r1108,
CR2) v legal(t1109, r1108, CR2)      Res(64,66) {r2103/CR2}
S70: legal(t1111, r1110, CR2) v legal(t1111, r1110, CR2) v legal(t1111,
r1110, CR2)      Res(68,69) {}
S71: legal(t1113, r1112, CR2) v legal(t1113, r1112, CR2)      Fact(70)
{}
S72: legal(t1115, r1114, CR2)      Fact(71) {}
```

Next, we resolve this with A53 to show that the train that made this legal move was on a connecting rail, in S1.

```
S73: ~on(t117, r1116, CS1) v ~connects(r1116, CR2) v on(t117, CR2, CS2)
Res(72,53) {r247/CR2, r1114/r145, t1115/t46}
```

We then resolve this (separately) with the axioms that Train 1 was on Rail 1 and Train 2 was on Rail 3.

```
S74: ~connects(CR1, CR2) v on(CT1, CR2, CS2)      Res(73,59) {t117/CT1,
r1116/CR1}
S75: ~connects(CR3, CR2) v on(CT2, CR2, CS2)      Res(73,60) {t117/CT2,
r1116/CR3}
```

We then resolve these, respectively, with the axioms that Rail 1 is connected to Rail 2 and Rail 3 is connected to Rail 2.

```
S76: on(CT1, CR2, CS2)      Res(74,27) {}
S77: on(CT2, CR2, CS2)      Res(75,28) {}
```

These 2 results can then be used to resolve the "save" axiom.

```
S78: ~safe(CS2) v ~on(t2118, CR2, CS2) v Equals(CT1, t2118)
Res(76,46) {s25/CS2, t127/CT1, r26/CR2}
S79: ~safe(CS2) v Equals(CT1, CT2)      Res(77,78) {t2118/CT2}
S80: ~safe(CS2)      Res(79,24) {}
```

This result resolves our negation to False.

```
S81: F      Res(80,61) {}
```

6. Extra Credit: Estimate the size of the search space in the last problem. Can you think of an example where theorem proving would be a better approach than model-checking and vice versa?

The size of the search space is exponential, with regard to the number of axioms. The depth of my proof is 9. In other words, if an automatic theorem prover used breadth first search, it would solve the proof in the 9th layer of the search tree.

The size of the tree depends on how many new axioms can be created from a pair of axioms. In our proof, there were situations that could resolve to 2 different axioms. This could be higher. If that factor is B, then the size of the search space is:

O((B^9) * (31^9))

This is a worst-case scenario because many of the axioms don't resolve.
something like the number of each type of variable (rails, trains, situations) times the number of instances of that type of variable (4, 2, 2 respectively).

A theorem prover is better than a model checker when there are a small set of axioms, a large universe and a short proof. For example, in the "all men are mortal" proof, if there was a large universe (more than just Hera), the model checker must consider all people, in order to show conclusively that there is no x that is not a man. In this case, x could include all men, animals, objects, and etcetera. However, the theorem prover would solve it in just a few steps.

This is because the search space for a model checker is 2^number of variables, rather than (number of axioms) ^ (depth of tree).

A model checker is better when there are many axioms, but a small universe.
The train example is a case in point. It has only 4 rails, 2 trains and 2 situations. However, it has many axioms. For this situation, it would be more efficient to test it using a theorem prover.

7. Extra Credit: Suppose you were writing a fully automatic resolution-refutation theorem prover that tries to derive a contradiction from the entire space of axioms. What kinds of heuristics might be useful for such a theorem-prover in searching for clauses to resolve with each other?

First we need to define what we mean by fully automatic. The input axioms for a theorem prover can be divided into sets, one set, could be important facts about the problem. In this case, every resolution step would resolve a member of this set, against another axiom. This is a forward search strategy. However, you could consider this not to be fully automatic, because the prover knows what facts about the problem are important.

Another form would be to use a backward search from the axiom that we are trying to refute. This does not require knowledge of what axioms are important.

We need a heuristic function that eliminates some sub goals that are less interesting. To do this, the heuristic could correlate to the size of the clause or the difficulty in resolving it. For example, a unit clause would be resolved first and the longest clause would be resolved last. This makes sense because we are trying to end up with an empty clause that resolves to false.
The prover would work something like this. Start with the axiom, which is the refutation of what we want to prove. Then, put the other axioms into a map, where the axiom has a key that corresponds to each negated sub clause. This will allow us to pick out the axioms that will resolve with the negated axiom that we wish to refute. This is not the primary heuristic, however.

We then, resolve the negated axiom with all of the other axioms which are keyed to the negated sub clause. The axioms that are produced are put into the map of axioms. We get the smallest axiom and attempt to resolve it against all of the other axioms. We set aside axioms, as they cannot be resolved. We terminate when we get an empty axiom that resolves to false.

This strategy, while does not address paramodulation, attempts to avoid those axioms that just get longer an longer. For example, in the siblings proof, we can generate very long axioms, by combining siblings, their father, their father's father, etc. that just get farther and farther from resolving the proof