1.a.
The two columns on the very left should have the same T/F assignments

A	B	C	D	$(A \wedge B \rightarrow C \wedge D)$	$(A \wedge B \rightarrow C)$
T	T	T	T	T	T
T	T	T	F	F	F
T	T	F	T	F	T
T	T	F	F	F	T
T	F	T	T	T	T
T	F	T	F	T	T
T	F	F	T	T	T
T	F	F	F	T	T
F	T	T	T	T	T
F	T	T	F	T	T
F	T	F	T	T	T
F	T	F	F	T	T
F	F	T	T	T	T
F	F	T	F	T	T
F	F	F	T	T	T
F	F	F	F	T	T

All assignments that satisfied $(A \wedge B \rightarrow C \wedge D)$ also satisfied $(A \wedge B \rightarrow C)$.(All instances in which the left side were true also led to the right side being true as well)

1.b. $(A \wedge B \rightarrow C \wedge D) \models (A \wedge B \rightarrow C)$ using **Natural Deduction.**

1.b. $A \wedge B \rightarrow C \wedge D$ ← premises.

1. $\neg(A \wedge B) \vee (C \wedge D)$ Implication Elimination.

2. $(C \vee \neg(A \wedge B)) \wedge (D \vee \neg(A \wedge B))$ Distributivity.

3. $(A \wedge B \rightarrow C) \wedge (A \wedge B \rightarrow D)$ Implication elimination (done on both sides).

4. $(A \wedge B \rightarrow C)$ And Elimination.

1.c. $A \wedge B \rightarrow C \wedge D$ premise.

$(\neg A \vee \neg B) \vee (C \wedge D)$ Implication Elimination

$(\neg A \vee \neg B \vee C) \wedge (\neg A \vee \neg B \vee D)$ Distribution / pushing negation inwards.

$(\neg A \vee \neg B \vee C), (\neg A \vee \neg B \vee D)$ ← KB

Query = $Q = A \wedge B \rightarrow C = \neg(A \wedge B) \vee C = (\neg A \vee \neg B) \vee C = \neg A \vee \neg B \vee C$

$\neg Q = \neg(\neg A \vee \neg B \vee C) = A \wedge B \wedge \neg C$

So clauses are

$(\neg A \vee \neg B \vee C), (\neg A \vee \neg B \vee D), A, B, \neg C$

These clauses contradict eachother

So proof is done

2.a
O1Y -> not (L1Y)
O1W -> not (L1W)
O1W -> not (C1Y)
O1Y -> not (C1W)
O1Y ^ O1W -> C1B
O1Y ^ O1W -> not (L1B)

O2Y -> not (L2Y)
O2W -> not (L2W)
O2W -> not (C2Y)
O2Y -> not (C2W)
O2Y ^ O2W -> C2B
O2Y ^ O2W -> not (L2B)

O3Y -> not (L3Y)
O3W -> not (L3W)
O3W -> not (C3Y)
O3Y -> not (C3W)
O3Y ^ O3W -> C3B
O3Y ^ O3W -> not (L3B)

O1W ^ O2W -> C3Y
O1Y ^ O2Y -> C3W

O2W ^ O3W -> C1Y
O2Y ^ O3Y -> C1W

O1W ^ O3W -> C2Y
O1Y ^ O3Y -> C2W

2.b

Facts {O1Y, L1W, O2W, L2Y, O3Y, L3B}

O1Y and O3Y from knowledge base gives us C2W.

This is from the rule

O1Y ^ O3Y -> C2W

From the knowledge base

3.

a. CanBikeToWork → CanGetToWork
b. CanDriveToWork → CanGetToWork
c. CanWalkToWork → CanGetToWork
d. HaveBike WorkCloseToHome ^ Sunny → CanBikeToWork
e. HaveMountainBike → HaveBike
f. HaveTenSpeed → HaveBike
g. OwnCar → CanDriveToWork
h. OwnCar → MustGetAnnualInspection
i. OwnCar → MustHaveValidLicense
j. CanRentCar → CanDriveToWork
k. HaveMoney CarRentalOpen → CanRentCar
l. HertzOpen→ CarRentalOpen
m. AvisOpen→ CarRentalOpen
n. EnterpriseOpen→ CarRentalOpen
o. CarRentalOpen → IsNotAHoliday
p. HaveMoney TaxiAvailable → CanDriveToWork
q. Sunny ^ WorkCloseToHome → CanWalkToWork
3
r. HaveUmbrella ^ WorkCloseToHome → CanWalkToWork
s. Sunny → StreetsDry }

Facts: { Rainy, HaveMoutainBike, EnjoyPlayingSoccer, WorkForUniversity, WorkCloseToHome, HaveMoney, HertzClosed, AvisOpen, McDonaldsOpen

Facts serve as agenda

Facts: { Rainy, HaveMoutainBike, EnjoyPlayingSoccer, WorkForUniversity, WorkCloseToHome, HaveMoney, HertzClosed, AvisOpen, McDonaldsOpen

Worked ou (highlighted words are “crossed out”)t:

m. AvisOpen→ CarRentalOpen
So we add carRentalOpen to facts
Facts: { Rainy, HaveMoutainBike, EnjoyPlayingSoccer, WorkForUniversity, WorkCloseToHome, HaveMoney, HertzClosed, AvisOpen, McDonaldsOpen, carRentalOpen}

HaveMoney and CarRentalOpen mean we can rent a car:
k. HaveMoney CarRentalOpen → CanRentCar

Facts: { Rainy, HaveMoutainBike, EnjoyPlayingSoccer, WorkForUniversity, WorkCloseToHome, HaveMoney, HertzClosed, AvisOpen, McDonaldsOpen, carRentalOpen, canRentCar}

CanRentCar means `j. CanRentCar → CanDriveToWork`

Facts: { `Rainy, HaveMoutainBike, EnjoyPlayingSoccer, WorkForUniversity, WorkCloseToHome, HaveMoney, HertzClosed, AvisOpen, McDonaldsOpen, carRentalOpen, canRentCar, canDriveToWork}`

canDriveToWork means we can get to work:
`b. CanDriveToWork → CanGetToWork`

Facts: { `Rainy, HaveMoutainBike, EnjoyPlayingSoccer, WorkForUniversity, WorkCloseToHome, HaveMoney, HertzClosed, AvisOpen, McDonaldsOpen, carRentalOpen, canRentCar, canDriveToWork, canGetToWork}`

`canGetToWork` was generated, so we're done.

4.

FirstElement is the top of the stack
Stack: firstElement, secondElement, …..

Working out the problem steps:

Stack: canGetToWork

canGetToWork has canDriveToWork as antecedent (rule b)

Stack: canDriveToWork

canDriveToWork has canRentCar as antecedent (rule j)

Stack: CanRentCar

CanRentCar has two antecedents: HaveMoney and CarRentalOpen (rule k)

Stack: HaveMoney, CarRentalOpen

HaveMoney is a known fact

Stack: CarRentalOpen

CarRentalOpen has AvisOpen as an antecedent (rule m)

AvisOpen is a fact

Stack is empty, so canGetToWork is true.

5. It seems that backward chaining should be used in the instance that the rules in the knowledge base generate inferences that make it difficult to reach a conclusion from. Forward chaining also provides rules in the form of definite clauses, so ambiguity in the clauses through the use of or (like one of these clauses might work) will not be permitted to using forward chaining with if that is the case. Additionally, if the clauses seem that they will be easy to generate, maybe they are generic, then it could be of advantage to use backward chaining. Also if the knowledge base is really large, backward chaining might be better given there are so many choices to work from if you do forward chaining.