1a. Prove that (A^B→C^D) ⊢ (A^B→C) ("conjunctive rule splitting") is a **sound rule-of inference** using a truth table.

A rule R is sound iff for all sentences, a, b if a⊢b then a ⊨ b

ANSWER

A	**B**	**C**	**D**	**A^B**	**C^D**	**A^B→C^D**	**A^B→C**
T	T	T	T	T	T	T	T
T	T	T	F	T	F	F	T
T	T	F	T	T	F	F	F
T	T	F	F	T	F	F	F
T	F	T	T	F	T	T	T
T	F	T	F	F	F	T	T
T	F	F	T	F	F	T	T
T	F	F	F	F	F	T	T
F	T	T	T	F	T	T	T
F	T	T	F	F	F	T	T
F	T	F	T	F	F	T	T
F	T	F	F	F	F	T	T
F	F	T	T	F	T	T	T
F	F	T	F	F	F	T	T
F	F	F	T	F	F	T	T
F	F	F	F	F	F	T	T

We see that all models that satisfy (A^B→C^D) also satisfy (A^B→C) and thus this is a sound rule of inference.

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

(Hint: it might help to use a ROI for "Implication Introduction". If you have a Horn clause, with 1 positive literal and n-1 negative literals, like $(\neg X \vee Z \vee \neg Y)$, you can transform it into a rule by collecting the negative literals as positive antecedents, e.g. $X \wedge Y \rightarrow Z$. This is a truthpreserving operation (hence sound), which you could prove to yourself using a truth table.)

ANSWER

1. $(A \wedge B \rightarrow C \wedge D)$ [Premise]
2. $C \wedge D$ [Premise]
3. $\neg(A \wedge B) \vee (C \wedge D)$ [Implication Elimination, 1]
4. $(\neg A \vee \neg B) \vee (C \wedge D)$ [De Morgan, 3]
5. $(\neg A \vee \neg B) \vee C$ [And Elimination, 4, 2]
6. $A \wedge B \rightarrow C$ [Implication Introduction, 5]

1c. Also prove $(A \wedge B \rightarrow C \wedge D) \models (A \wedge B \rightarrow C)$ using Resolution.

We will use resolution refutation. First we convert our KB into CNF:

1. $(A \wedge B \rightarrow C \wedge D)$ [KB]
2. $\neg(A \wedge B) \vee (C \wedge D)$ [Implication Elimination, 1]
3. $(\neg A \vee \neg B) \vee (C \wedge D)$ [De Morgan, 2]
4. $(\neg A \vee \neg B \vee C) \wedge (\neg A \vee \neg B \vee D)$ [Distribution, 3]

Our new KB = $\{(\neg A \vee \neg B \vee C), (\neg A \vee \neg B \vee D)\}$

Now, we negate our query and add it to the KB

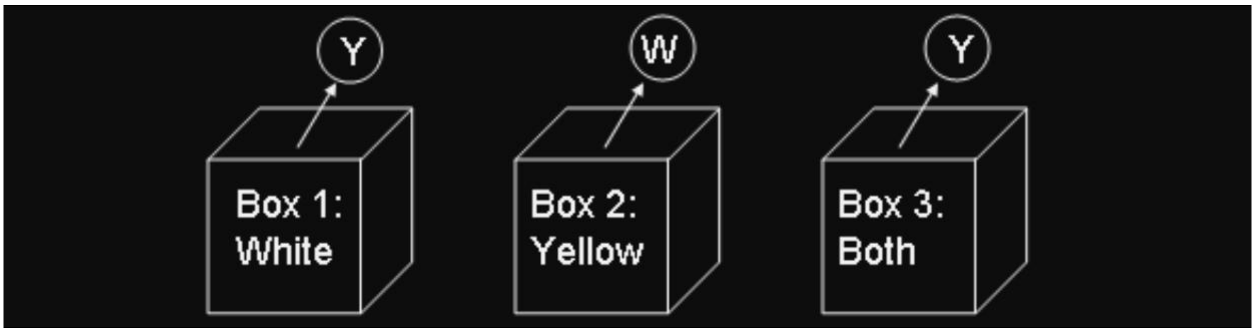
1. $\neg(A \wedge B \rightarrow C)$ [Negation]
2. $\neg(\neg(A \wedge B) \vee C)$ [Implication Elimination, 1]
3. $\neg\neg(A \wedge B) \wedge \neg C$ [DeMorgan, 2]
4. $A \wedge B \wedge \neg C$ [DN, 3]

Now our KB = $\{(\neg A \vee \neg B \vee C), (\neg A \vee \neg B \vee D), A, B, \neg C\}$. From this we should be able to derive an inconsistency.

1. A [KB]
2. B [KB]
3. $\neg C$ [KB]
4. $(\neg A \vee \neg B \vee C)$ [KB]
5. $(\neg B \vee C)$ [Resolve 1, 4]
6. (C) [Resolve 2, 5]
7. $\varnothing$ [Resolve, 3, 6]

2. **Sammy’s Sport Shop**

You are the proprietor of Sammy’s Sport Shop. You have just received a shipment of three boxes filled with tennis balls. One box contains only yellow tennis balls, one box contains only white tennis balls, and one contains both yellow and white tennis balls. You would like to stock the tennis balls in appropriate places on your shelves. Unfortunately, the boxes have been labeled incorrectly; the manufacturer tells you that you have exactly one box of each, but that each box is definitely labeled wrong. You draw one ball from each box and observe its color. Given the initial (incorrect) labeling of the boxes above, and the three observations, use Propositional Logic to infer the correct contents of the middle box.



Use propositional symbols in the following form: O1Y means a yellow ball was drawn (observed) from box 1, L1W means box 1 was initially labeled white, C1W means box 1 contains (only) white balls, and C1B means box 1 actually contains both types of tennis balls. Note, there is no ‘O1B’, etc, because you can't directly "observe both". When you draw a tennis ball, it will either be white or yellow. The initial facts describing this particular situation are: {O1Y, L1W, O2W, L2Y, O3Y, L3B}

2a. Using these propositional symbols, write a propositional knowledge base (sammy.kb) that captures the knowledge in this domain (i.e. implications of what different observations or labels mean, as well as constraints inherent in this problem, such as that all boxes have different contents). Do it in a complete and general way, writing down all the rules and constraints, not just the ones needed to make the specific inference about the middle box. Do not include derived knowledge that depends on the particular labeling of this instance shown above; stick to what is stated in the problem description above. Your KB should be general enough to reason about any alternative scenario, not just the one given above (e.g. with different observations and labels and box contents).

ANSWER

KB = {

a. O1Y → C1Y v C1B
b. O1W → C1W v C1B
c. O2Y → C2Y v C2B
d. O2W → C2W v C2B
e. O3Y → C3Y v C3B
f. O3W → C3W v C3B
g. L1W → (C1Y v C1B) ^ ¬C1W
h. L1Y → (C1W v C1B) ^ ¬C1Y
i. L1B → (C1Y v C1W) ^ ¬C1B
j. L2W → (C2Y v C2B) ^ ¬C2W
k. L2Y → (C2W v C2B) ^ ¬C2Y
l. L2B → (C2Y v C2W) ^ ¬C2B
m. L3W → (C3Y v C3B) ^ ¬C3W
n. L3Y → (C3W v C3B) ^ ¬C3Y
o. L3B → (C3Y v C3W) ^ ¬C3B
p. ¬(C1W ^ C2W) ^ ¬(C1W ^ C3W) ^ ¬(C2W ^ C3W)
q. ¬(C1Y ^ C2Y) ^ ¬(C1Y ^ C3Y) ^ ¬(C2Y ^ C3Y)
r. ¬(C1B ^ C2B) ^ ¬(C1B ^ C3B) ^ ¬(C2B ^ C3B)
s. ¬(L1W ^ L2W) ^ ¬(L1W ^ L3W) ^ ¬(L2W ^ L3W)
t. ¬(L1Y ^ L2Y) ^ ¬(L1Y ^ L3Y) ^ ¬(L2Y ^ L3Y)
u. ¬(L1B ^ L2B) ^ ¬(L1B ^ L3B) ^ ¬(L2B ^ L3B)
v. ¬(L1W ^ L1Y) ^ ¬(L1W ^ L1B) ^ ¬(L1Y ^ L1B)
w. ¬(L2W ^ L2Y) ^ ¬(L2W ^ L2B) ^ ¬(L2Y ^ L2B)
x. ¬(L3W ^ L3Y) ^ ¬(L3W ^ L3B) ^ ¬(L3Y ^ L3B)
y. ¬O1B ^ ¬O2B ^ ¬O3B
z. ¬(O1W ^ O1Y) ^ ¬(O2W ^ O2Y) ^ ¬(O3W ^ O3Y)

}

a-f: Consequence of observables

g-o: Consequence of labels

p-r: Constraints (content of boxes)

s-x: Constraints (labels of boxes, one per box)

y-z: Constraints (observables)

2b. Prove that box 2 must contain white balls (C2W) using Natural Deduction.

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

1. O3Y → C3Y v C3B [e]

2. L3B → (C3Y v C3W) ^ ¬C3B [o]
3. L3B [Fact]
4. (C3Y v C3W) ^ ¬C3B [MP, 2, 3]
5. ¬C3B [AE, 2, 6]
6. C3Y [Simp, 1, 5] (Established that box 3 is yellow)
7. O1Y → C1Y v C1B
8. O1Y [Fact]
9. C1Y v C1B [MP, 7, 8]
10. C1B [q, 6, 9] (Constraint) (Established that box 1 is both)
11. O2W → C2W v C2B [d]
12. O2W [Fact]
13. C2W v C2B [MP, 11, 12]
14. C2W [r, 10, 13] (Constraint) (box 2 is white)

2c. Convert your KB to CNF.

CNF(KB) = {

1. ¬O1Y v C1Y v C1B
2. ¬O1W v C1W v C1B
3. ¬O2Y v C2Y v C2B
4. ¬O2W v C2W v C2B
5. ¬O3Y v C3Y v C3B
6. ¬O3W v C3W v C3B
7. (¬L1W ∨ C1Y ∨ C1B)
8. (¬L1W ∨ ¬C1W)
9. (¬L1Y ∨ C1W ∨ C1B)
10. (¬L1Y ∨ ¬C1Y)
11. (¬L1B ∨ C1Y ∨ C1W)
12. (¬L1B ∨ ¬C1B)
13. (¬L2W ∨ C2Y ∨ C2B)
14. (¬L2W ∨ ¬C2W)
15. (¬L2Y ∨ C2W ∨ C2B)
16. (¬L2Y ∨ ¬C2Y)
17. (¬L2B ∨ C2Y ∨ C2W)
18. (¬L2B ∨ ¬C2B)
19. (¬L3W ∨ C3Y ∨ C3B)
20. (¬L3W ∨ ¬C3W)
21. (¬L3Y ∨ C3W ∨ C3B)
22. (¬L3Y ∨ ¬C3Y)
23. (¬L3B ∨ C3Y ∨ C3W)
24. (¬L3B ∨ ¬C3B)
25. (¬C1W v ¬C2W)
26. (¬C1W v ¬C3W)

27. (¬C2W v ¬C3W)
28. (¬C1Y v ¬C2Y)
29. (¬C1Y v ¬C3Y)
30. (¬C2Y v ¬C3Y)
31. (¬C1B v ¬C2B)
32. (¬C1B v ¬C3B)
33. (¬C2B v ¬C3B)
34. (¬L1W v L2W)
35. ¬L1W v ¬L3W
36. ¬L2W v ¬L3W
37. ¬L1Y v ¬L2Y
38. ¬L1Y v ¬L3Y
39. ¬L2Y v ¬L3Y
40. ¬L1B v ¬L2B
41. ¬L1B v ¬L3B
42. ¬L2B v ¬L3B
43. ¬L1W v ¬L1Y
44. ¬L1W v ¬L1B
45. ¬L1Y v ¬L1B
46. ¬L2W v ¬L2Y
47. ¬L2W v ¬L2B
48. ¬L2Y v ¬L2B
49. ¬L3W v ¬L3Y
50. ¬L3W v ¬L3B
51. ¬L3Y v ¬L3B
52. ¬O1B
53. ¬O2B
54. ¬O3B
55. ¬O1W v ¬O1Y
56. ¬O2W v ¬O2Y
57. ¬O3W v ¬O3Y

}

2d. Prove C2W using Resolution.

Facts: {O1Y, L1W, O2W, L2Y, O3Y, L3B} (also added to KB)

We add ¬C2W to the KB.

1. ¬O2W v C2W v C2B [KB, 4]
2. C2B [Resolve 1, facts]
3. (¬C1B v ¬C2B) [KB, 31]
4. ¬C1B [Resolve 2,3]
5. (¬L1W ∨ C1Y ∨ C1B) [KB, 7]
6. C1Y [Resolve, facts, 4, 5]
7. (¬C1Y v ¬C3Y) [KB, 29]
8. ¬C3Y Resolve[6,7]
9. (¬L3B ∨ C3Y ∨ C3W) [KB, 23]
10. C3W [Resolve facts, 8, 9]
11. (¬C2W v ¬C3W) [KB, 27]
12. ¬C2W [Resolve, 10, 11]
13. (¬L2Y ∨ C2W ∨ C2B) [KB, 15]
14. ∅ [Resolve facts, 2, 12]

3. Do Forward Chaining for the CanGetToWork KB below. You don't need to follow the formal FC algorithm (with agenda/queue and counts array). Just indicate which rules are triggered (in any order), and keep going until all consequences are generated. Show the final list of all inferred propositions at the end. Is CanGetToWork among them?

KB = {

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

r. HaveUmbrella ^ WorkCloseToHome → CanWalkToWork

s. Sunny → StreetsDry }

Facts: { Rainy, HaveMoutainBike, EnjoyPlayingSoccer, WorkForUniversity,

WorkCloseToHome, HaveMoney, HertzClosed, AvisOpen, McDonaldsOpen }

ANSWER:

Irrelevant facts (negations are not allowed in FC): { Rainy, EnjoyPlayingSoccer, WorkForUniversity, HertzClosed, McDonaldsOpen}

Relevant facts: {HaveMoutainBike, WorkCloseToHome, HaveMoney, AvisOpen }

HaveMountainBike → HaveBike

AvisOpen→ CarRentalOpen

CarRentalOpen → IsNotAHoliday

HaveMoney ^ CarRentalOpen → CanRentCar

CanRentCar → CanDriveToWork

CanDriveToWork → CanGetToWork

Final list: {HaveBike, CarRentalOpen, IsNotAHoliday, CanRentCar, CanDriveToWork, CanGetToWork}

CanGetToWork is among them.

1. Do Backward Chaining for the CanGetToWork KB. In this case, you should follow the BC algorithm closely (the pseudocode for the propositional version of Back-chaining is given in the lecture slides). Important: when you pop a subgoal (proposition) from the goal stack, you should systematically go through all rules that can be used to prove it IN THE ORDER THEY APPEAR IN THE KB. In some cases, this will lead to back-tracking, which you should show. Also, the sequence of results depends on order in which antecedents are pushed onto the stack. If you have a rule like A^B→C, and you pop C off the stack, push the antecedents in reverse order, so B goes in first, then A; in the next iteration, A would be the next subgoal popped off the stack. 5. In what kinds of problems would it be better to use forward-chaining? When would it be better to use backward-chaining?

- CanGetToWork // initialize with query
- CanBikeToWork // pop CanGetToWork, push CanBikeToWork
- Sunny, WorkCloseToHome, HaveBike // pop CanBikeToWork, push antecedents in reverse order
- Sunny, WorkCloseToHome, HaveMountainBike // pop HaveBike, push antecedents
- Sunny, WorkCloseToHome // pop HaveMountainBike (fact)
- Sunny // pop WorkCloseToHome (fact)
- CanDriveToWork // (pop) Sunny is not provable, backtrack. Push CanDriveToWork
- OwnCar // pop CanDriveToWork, push antecedents
- CanRentCar // (pop) OwnCar is not provable, backtrack. Push CanRentCar
- CarRentalOpen, HaveMoney // pop CanRentCar, push antecedents in reverse order
- CarRentalOpen // pop HaveMoney (fact)
- HertzOpen // pop CarRentalOpen, push antecedents
- AvisOpen // (pop) HertzOpen is not provable, backtrack. Push AvisOpen
- ∅ // pop AvisOpen (fact)

Forward-chaining is generally better when the goal or query is not clearly defined or there are multiple goals to be proven. Backward-chaining is generally better when there is a specific, well-defined goal or query to be proven.