AI- Course

Assignemnt#3

Muhammad Usman Awan

F223378   6E

Contents

[Question No. 1: [CSP] 0](#_Toc194256278)

[(a) 0](#_Toc194256279)

[ANSWER: 0](#_Toc194256280)

[(b) 2](#_Toc194256281)

[CODE: 2](#_Toc194256282)

[OUTPUT: 4](#_Toc194256283)

[Question No. 2 [AC-3] 4](#_Toc194256284)

[CODE: 5](#_Toc194256285)

[OUTPUT: 8](#_Toc194256286)

[Question No. 3: [Entailment and Sentence Validity] 9](#_Toc194256287)

[(a) Prove each of the following assertions: 9](#_Toc194256288)

[(b) Prove, or find a counterexample to, each of the following assertions: 11](#_Toc194256289)

[Question No. 4: [Inference in Propositional Logic] 12](#_Toc194256290)

# Question No. 1: [CSP]

## (a)

Consider the following logic puzzle: In five houses, each with a different color, live five persons of different nationalities, each of whom prefers a different brand of candy, a different drink, and a different pet. Given the following facts, the questions to answer are "Where does the zebra live, and in which house do they drink water?"

1. The Englishman lives in the red house.
2. The Spaniard owns the dog.
3. The Norwegian lives in the first house on the left.
4. The green house is immediately to the right of the ivory house.
5. The man who eats Hershey bars lives in the house next to the man with the fox.
6. Kit Kats are eaten in the yellow house.
7. The Norwegian lives next to the blue house.
8. The Smarties eater owns snails.
9. The Snickers eater drinks orange juice.
10. The Ukrainian drinks tea.
11. The Japanese eats Milky Ways.
12. Kit Kats are eaten in a house next to the house where the horse is kept.
13. Coffee is drunk in the green house.
14. Milk is drunk in the middle house.

### ANSWER:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **House #** | **Color** | **Nationality** | **Drink** | **Candy** | **Pet** |
| 1 |  |  |  |  |  |
| 2 |  |  |  |  |  |
| 3 |  |  |  |  |  |
| 4 |  |  |  |  |  |
| 5 |  |  |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **House #** | **Color** | **Nationality** | **Drink** | **Candy** | **Pet** |
| 1 |  | Norwegian |  |  |  |
| 2 | Blue |  |  |  |  |
| 3 | |  | | --- | | Ivory |  |  | | --- | |  | |  | Milk |  |  |
| 4 | |  | | --- | | Green |  |  | | --- | |  | |  | Coffee |  |  |
| 5 |  |  |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **House #** | **Color** | **Nationality** | **Drink** | **Candy** | **Pet** |
| 1 | Yellow | Norwegian | Water | Kit Kats |  |
| 2 | Blue | Ukrainian | Tea |  | Horse |
| 3 | |  | | --- | | Ivory |  |  | | --- | |  | | Spaniard | Milk |  | Dog |
| 4 | |  | | --- | | Green |  |  | | --- | |  | | Japanese | Coffee | Milky Way | Zebra |
| 5 | Red | Englishman |  |  |  |

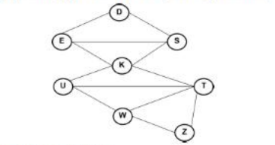
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **House #** | **Color** | **Nationality** | **Drink** | **Candy** | **Pet** |
| 1 | Yellow | Norwegian | Water | Kit Kats | Fox |
| 2 | Blue | Ukrainian | Tea | Hershey | Horse |
| 3 | |  | | --- | | Ivory |  |  | | --- | |  | | Spaniard | Milk | Snickers | Dog |
| 4 | |  | | --- | | Green |  |  | | --- | |  | | Japanese | Coffee | Milky Way | Zebra |
| 5 | Red | Englishman | Orange Juice | Smarties | Snails |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **House #** | **Color** | **Nationality** | **Drink** | **Candy** | **Pet** |
| 1 | Yellow | Norwegian | Water | Kit Kats | Fox |
| 2 | Blue | Ukrainian | Tea | Hershey | Horse |
| 3 | |  | | --- | | Ivory |  |  | | --- | |  | | Spaniard | Milk | Snickers | Dog |
| 4 | |  | | --- | | Green |  |  | | --- | |  | | Japanese | Coffee | Milky Way | Zebra |
| 5 | Red | Englishman | Orange Juice | Smarties | Snails |

1. **Zebra lives in House 4** (Japanese).
2. **Water is drunk in House 1** (Norwegian).

## (b)

Implement the constraint satisfaction problem using Forward checking with backtracking of the following map coloring problem. The state space is represented as,



Variables: D, S, E, K, U,T,W,Z

Domains: Di= (red; green; blue}

Constraints: adjacent regions must have different colours.

### CODE:

variables = ['D', 'S', 'E', 'K', 'U', 'T', 'W', 'Z']

domains = {var: ['red', 'green', 'blue'] for var in variables}

neighbors = {

    'D': ['S', 'E'],

    'S': ['D', 'K', 'E'],

    'E': ['D', 'K', 'S'],

    'K': ['E', 'S', 'U', 'T'],

    'U': ['K', 'T', 'W'],

    'T': ['K', 'U', 'W'],

    'W': ['U', 'T', 'Z'],

    'Z': ['T', 'W']

}

def is\_consistent(var, value, assignment):

    for neighbor in neighbors[var]:

        if neighbor in assignment and assignment[neighbor] == value:

            return False

    return True

def forward\_check(domains, var, value):

    new\_domains = {v: list(domains[v]) for v in domains}

    for neighbor in neighbors[var]:

        if value in new\_domains[neighbor]:

            new\_domains[neighbor].remove(value)

            if not new\_domains[neighbor]:

                return None

    return new\_domains

def backtrack(assignment, domains):

    if len(assignment) == len(variables):

        return assignment

    unassigned = [v for v in variables if v not in assignment]

    var = min(unassigned, key=lambda v: len(domains[v]))

    for value in domains[var]:

        if is\_consistent(var, value, assignment):

            assignment[var] = value

            new\_domains = forward\_check(domains, var, value)

            if new\_domains:

                result = backtrack(assignment, new\_domains)

                if result:

                    return result

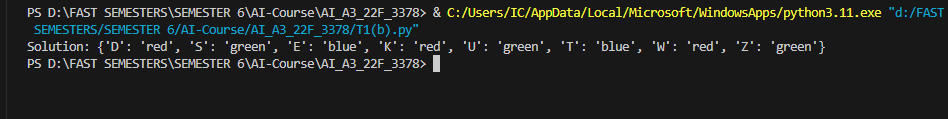
            del assignment[var]

    return None

solution = backtrack({}, domains)

print("Solution:", solution)

### OUTPUT:



# Question No. 2 [AC-3]

Implement the AC-3 algorithm to show that are consistency can detect the inconsistency of the partial assignment (WA-green, V =red) for the problem shown in Figure 1.

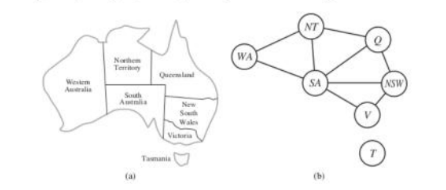


Figure 1: (a) The principal states and territories of Australia. Coloring this map can be viewed as a constraint satisfaction problem (CSP). The goal is to assign colors to each region so that no neighboring regions have the same color. (b) The map-coloring problem represented as a constraint graph.

## CODE:

import matplotlib.pyplot as plt

import networkx as nx

from collections import deque

variables = ['WA', 'NT', 'SA', 'Q', 'NSW', 'V', 'T']

domains = {var: ['red', 'green', 'blue'] for var in variables}

domains['WA'] = ['green']

domains['V'] = ['red']

neighbors = {

    'WA': ['NT', 'SA'],

    'NT': ['WA', 'SA', 'Q'],

    'SA': ['WA', 'NT', 'Q', 'NSW', 'V'],

    'Q': ['NT', 'SA', 'NSW'],

    'NSW': ['SA', 'Q', 'V'],

    'V': ['SA', 'NSW'],

    'T': []

}

def ac3(domains, neighbors):

    queue = deque([(xi, xj) for xi in variables for xj in neighbors[xi]])

    while queue:

        xi, xj = queue.popleft()

        if revise(domains, xi, xj):

            if not domains[xi]:

                return False

            for xk in neighbors[xi]:

                if xk != xj:

                    queue.append((xk, xi))

    return True

def revise(domains, xi, xj):

    revised = False

    for x in domains[xi][:]:

        if not any(x != y for y in domains[xj]):

            domains[xi].remove(x)

            revised = True

    return revised

def show\_graph(domains, neighbors):

    G = nx.Graph()

    for var in variables:

        for neighbor in neighbors[var]:

            G.add\_edge(var, neighbor)

    color\_map = {

        'red': '#e74c3c',

        'green': '#2ecc71',

        'blue': '#3498db'

    }

    node\_colors = []

    for node in G.nodes:

        if len(domains[node]) == 1:

            node\_colors.append(color\_map[domains[node][0]])

        else:

            node\_colors.append('#bdc3c7')

    pos = nx.spring\_layout(G, seed=42)

    plt.figure(figsize=(8, 6))

    nx.draw(

        G, pos, with\_labels=True,

        node\_color=node\_colors,

        node\_size=800,

        font\_color='white',

        font\_weight='bold'

    )

    plt.title("Australia Map Coloring CSP (Partial Assignment)")

    plt.show()

consistent = ac3(domains, neighbors)

if consistent:

    print("AC-3: The CSP is consistent with the partial assignment.")

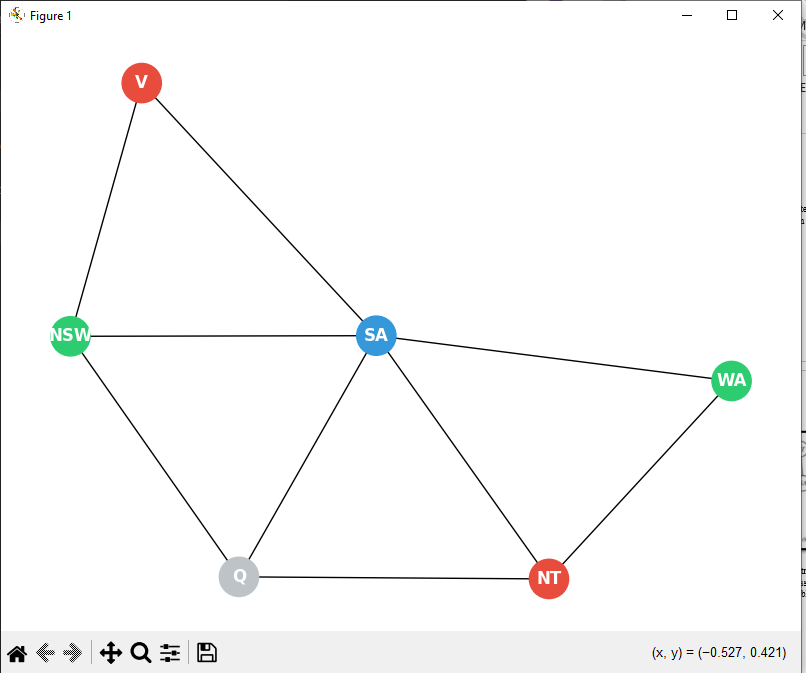
else:

    print("AC-3: Inconsistency detected with the partial assignment!")

show\_graph(domains, neighbors)

### OUTPUT:





# Question No. 3: [Entailment and Sentence Validity]

## (a) Prove each of the following assertions:

**i.**   α is valid if and only if True ⊨ α.

**Proof**:  
A sentence **α is valid** if it is **true in all possible models**.  
The notation True ⊨ α means that **in all models where True is true**, α is also true.

Since True is true in **all models**, this implies that α must be true in all models — which is exactly the definition of **validity**.

**Hence, α is valid ⇔ True ⊨ α.**

**ii.**   For any α, False ⊨ α.

**Proof**:  
The notation False ⊨ α means that **in all models where False is true, α must be true**.

But False is **never true in any model**, so the implication holds **vacuously** — there is no counterexample to False ⊨ α.

**Hence, this is always true for any α.**

**iii.**  α ⊨ β if and only if the sentence (α ⇒ β) is valid.

**Proof**:

* α ⊨ β means that **in every model where α is true, β is also true**.
* The implication (α ⇒ β) is **true in every model** if:
  + either α is false, or
  + both α and β are true.

This matches the condition for α ⊨ β. So the implication being valid means α entails β.

**Thus, α ⊨ β ⇔ (α ⇒ β) is valid.**

**iv.**  α = β if and only if the sentence (α ⇔ β) is valid.

**Proof**:

* α = β means α and β are **logically equivalent** (have the same truth value in all models).
* (α ⇔ β) is valid if it is **true in all models**, which happens exactly when α and β are logically equivalent.

**Hence, α = β ⇔ (α ⇔ β) is valid.**

**v.**   α ⊨ β if and only if the sentence (α ∧ ¬β) is unsatisfiable.

**Proof**:

* α ⊨ β means there is **no model where α is true and β is false**.
* So (α ∧ ¬β) would be **false in all models**, i.e., unsatisfiable.

**Thus, α ⊨ β ⇔ (α ∧ ¬β) is unsatisfiable.**

## (b) Prove, or find a counterexample to, each of the following assertions:

**i.**   If α ⊨ γ or β ⊨ γ (or both) then (α ∧ β) ⊨ γ

**Counterexample**:

Let:

* α = True
* β = False
* γ = True

Then:

* α ⊨ γ (True ⊨ True)
* β ⊨ γ is irrelevant here
* But (α ∧ β) = False, so (α ∧ β) ⊨ γ is **vacuously true**, but **does not imply** γ logically follows from both.

Now try this:

* Let α = p, β = ¬p, γ = q

Then:

* α ⊨ γ: False
* β ⊨ γ: False
* (α ∧ β) = (p ∧ ¬p) → contradiction → entails anything vacuously

So we can’t guarantee γ logically follows from α ∧ β unless both entail γ separately.

**Hence, this statement is False.**

**ii.**  If α ⊨ (β ∧ γ) then α ⊨ β and α ⊨ γ.

**Proof**:  
If α entails (β ∧ γ), then in every model where α is true, both β and γ are true.  
So naturally, α entails β, and α entails γ.

**Thus, the statement is True.**

**iii.**  If α ⊨ (β ∨ γ) then α ⊨ β or α ⊨ γ (or both).

**Counterexample**:

Let:

* α = True
* β = False
* γ = True

Then:

* β ∨ γ = True
* α ⊨ (β ∨ γ)
* But α ⊨ β → False, and α ⊨ γ → False is also possible in other models

Or more cleanly:

Let α = True

* β = A
* γ = ¬A

Then β ∨ γ is always true (a tautology), so α ⊨ (β ∨ γ), but α ⊨ β is **not necessarily** true, nor α ⊨ γ.

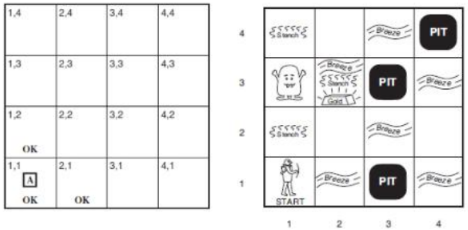
**Hence, the statement is False.**

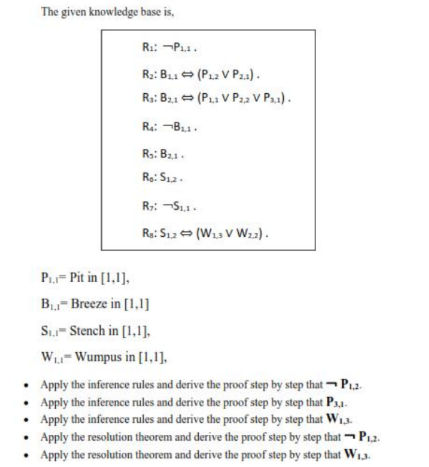
# Question No. 4: [Inference in Propositional Logic]

**[Wumpus World]:**

Consider the Wumpus world example with the following rules:

* The agent can move **Forward**, **TurnLeft by 90°**, **TurnRight by 90°**, **grab**, **shoot**.
* The square adjacent **directly (not diagonally)** to the square containing the **Wumpus**, the agent will perceive a **Stench**.
* The squares adjacent to a **pit**, the agent will perceive a **Breeze**.
* The square with **gold**, the agent will perceive a **Glitter**.
* An agent **walks into a wall**, it will perceive a **Bump**.
* ☠️ When the **Wumpus is killed**, it emits a woeful **Scream**.



****

**Knowledge Base:**

* R1: ~P11
* R2: B11 <=> (P12 v P21)
* R3: B21 <=> (P11 v P22 v P31)
* R4: ~B11
* R5: B21
* R6: S12
* R7: ~S11
* R8: S12 <=> (W13 v W22)

**Inference-Based Proofs**

**1. Prove ~P12**

From R2: B11 <=> (P12 v P21)

Given R4: ~B11

Thus: ~B11 => ~(P12 v P21) => ~P12 ^ ~P21

Therefore, ~P12 is proven.

**2. Prove P31**

From R3: B21 <=> (P11 v P22 v P31)

Given R1: ~P11 and (2,2) is safe => ~P22

Given R5: B21 is true

Therefore, P31 must be true.

**3. Prove W13**

From R8: S12 <=> (W13 v W22)

Given R6: S12, and square (2,2) is safe => ~W22

Therefore, W13 must be true.

**Resolution-Based Proofs**

**4. Prove ~P12 using Resolution**

From R2: B11 <=> (P12 v P21)

CNF: ~B11 v P12 v P21

From R4: ~B11

Resolve: ~B11 and (~B11 v P12 v P21) => P12 v P21

From earlier: ~P21 => P12

But we also have ~P12 from earlier => contradiction

Therefore, ~P12 is proven by resolution.

**5. Prove W13 using Resolution**

From R8: S12 <=> (W13 v W22)

CNF: ~S12 v W13 v W22

Given: S12 and ~W22

Resolve: S12 and (~S12 v W13 v W22) => W13 v W22

Resolve with ~W22 => W13

Therefore, W13 is proven by resolution.