Adrien Protzel

2/16/2025

Exercise 6: Designing Factored States & CSPs

# Problem 1

Consider the problems of Vacuum-Cleaner World (module 2) and the Eight-Puzzle problem. For each of these two problems, illustrate your specification of them as CSPs. Explain why, or why not, these problems are appropriate to consider as a CSP.

## Vacuum-Cleaner World:

Variables: {dirt1, ..., dirtk}

Domain:

Ddirti => {True, False}

Constraints: < = False>

## Eight-Puzzle:

Variables: {L00, L01, L02, L10, L11, L12, L20, L21, L22}

Domain:

DL00 => {None, 1, 2, 3, 4, 5, 6, 7, 8}

DL01 => {None, 1, 2, 3, 4, 5, 6, 7, 8}

DL02 => {None, 1, 2, 3, 4, 5, 6, 7, 8}

DL10 => {None, 1, 2, 3, 4, 5, 6, 7, 8}

DL11 => {None, 1, 2, 3, 4, 5, 6, 7, 8}

DL12 => {None, 1, 2, 3, 4, 5, 6, 7, 8}

DL20 => {None, 1, 2, 3, 4, 5, 6, 7, 8}

DL21 => {None, 1, 2, 3, 4, 5, 6, 7, 8}

DL22 => {None, 1, 2, 3, 4, 5, 6, 7, 8}

Constraints: <(L00, L01, L02, L10, L11, L12, L20, L21, L22), {None, 1, 2, 3, 4, 5, 6, 7, 8}>

There is only one solution for each of these problems, therefore it does not need constraints and should not be considered a good CSP. These problems only need a pattern to follow to get to THE solution. There are no other possible outcomes from these simple puzzles.

# Problem 2

## CSP:

Variables: {K1, …, Kk}

Domain:

DKi => {(x, y) | 1 <= x,y <= n}

Constraints: <|xi - xj| \* |yi - yj| != 2> WHERE Ki = (xi, yi), Kj = (xj, yj)

The variables are all knights 1 through k. The domain is all spots x,y on the board nxn. Finally the constraints are that no two knights are within an L shape of each other, by which their coordinates do not equal 2.

## Functions:

ACTIONS:

Loop through the board.

Check for an open space.

Check if the position is safe (no other knight can attack this position).

Add the knight if the position is safe; otherwise, return false)

RESULT:

Save and return the board configuration after ACTIONS.

OBJECTIVE:

Place knights until the ACTIONS.add\_knight function returns false.

# Problem 3

## Design 1:

Variables:

{Englishman, Norwegian, Spaniard, Ukrainian, Japanese, Red, Green, Ivory, Yellow, Blue, Dog, Fox, Horse, Snails, Zebra, Herseys, Kit Kat, Smarties, Snickers, Milky Ways, Orange Juice, Tea, Coffee, Milk, Water}

Domain:

Di = {1, 2, 3, 4, 5}

Constraints:

<(Englishman, Red), Englishman = Red>

<(Spaniard, Dog), Spaniard= Dog>

<(Norwegian), Norwegian = 1>

<(Ivory, Green), Green = Ivory+1>

<(Hershey’s, Fox), Hershey's = Fox+n>

<(Kit Kats, Yellow), Kit Kats = Yellow>

<(Norwegian, Blue), Norwegian = Blue+-1>

<(Smarties, Snails), Smarties = Snails>

<(Snickers, Orange Juice), Snickers = Orange Juice>

<(Ukrainian, Teas), Ukrainian = Tea>

<(Japanese, Milky Ways), Japanese = Milky Ways>

<(Kit Kats), Kit Kats = Horse+-1>

<(Coffee, Green), Coffee = Green>

<(Milk), Milk = 3>

This design lists all variables individually and only lists the houses as domains. This makes things fluid by allowing the variables to float around and the houses to have order. The constraints also try to tie variables to create a starting order. This design is very simple and compact. The drawback is that each variable would need to be tested and assigned individually which means a lock of checking and backtracking.

## Design 2:

Variables:

House: {H1, H2, H3, H4, H5}

Color: {Red, Green, Ivory, Yellow, Blue}

Person: {Englishman, Spaniard, Norwegian, Ukrainian, Japanese}

Candy: {Hershey, Kit Kat, Smarties, Snickers, Milky Way}

Drink: {Coffee, Tea, Milk, Orange Juice, Water}

Pet: {Dog, Fox, Snails, Horse, Zebra}

Domain:

DHi => {Color, Person, Candy, Drink, Pet}

Constraints:

Where X is some number between 1 and 5 that is not already taken by an H. Where \_ is any value of that type.

<(H1), {\_, Norwegian, \_, \_, \_}>

<(H2), {Blue, \_, \_, \_, \_}>

<(H3), {\_, \_, \_, Milk, \_}>

<(Hx(Hx = {Horse}+-1)), {\_, \_, Kit Kat, \_, \_}>

<(H(Hx = {Fox }+-1)), {\_, \_, Hershey, \_, \_}>

<(H(Hx = {Ivory}+1)), {Green, \_, \_, Coffee, \_}>

<(Hx), {Ivory, \_, \_, \_, \_}>

<(Hx), {Red, Englishman, \_, \_, \_}>

<(Hx), {Yellow, \_, Kit Kat, \_, \_}>

<(Hx), {\_, Spaniard, \_, \_, Dog}>

<(Hx), {\_, \_, Smarties, \_, Snails}>

<(Hx), {\_, \_, Snickers, Orange Juice, \_}>

<(Hx), {\_, Ukrainian, \_, Tea, \_}>

<(Hx), {\_, Japanese, Milky Way, \_, \_}>

This design attempts to add modularity to the houses, by saying each house has a certain set of things and then saying what those things could be. This makes it easier to modify or add or remove variables. The downside is that there are overall more variables to check through. The constraints also have a number of redundancies and fill in the blanks. This second design attempts to be more reader friendly with its constraints and saying what each house should keep.