CONSTRAINT SATISFACTION PROBLEMS (CSPs):

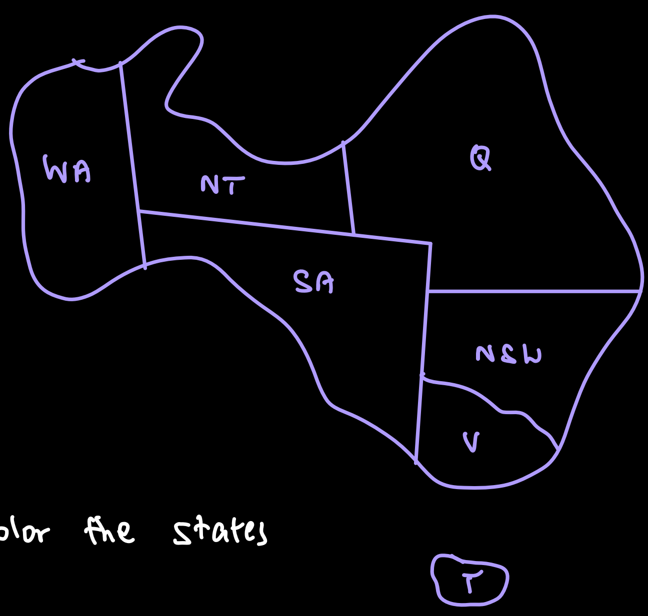
- Standard formulation a search
- Standard search strategy
- Improvements: Six total

→ Set of variables $x_1 \ldots x_n$ (discrete variable)

with a set of values they can take $D_1, \ldots D_n$
(Domain)

→ Constraints: allowable combinations of value.

EXAMPLE:



Goal: Color the states

Variables, x_i : {WA, NT, SA, Q, NSW, V, T}

Domain, D_i : {Red, Blue, Green}

Constraints:

WA ≠ NT

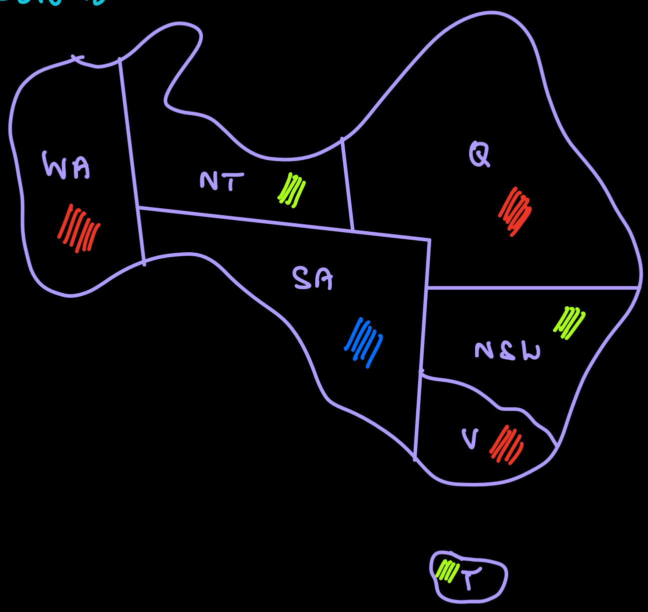
WA ≠ SA

⋮

V ≠ NSW.

No neighboring states should have same color.

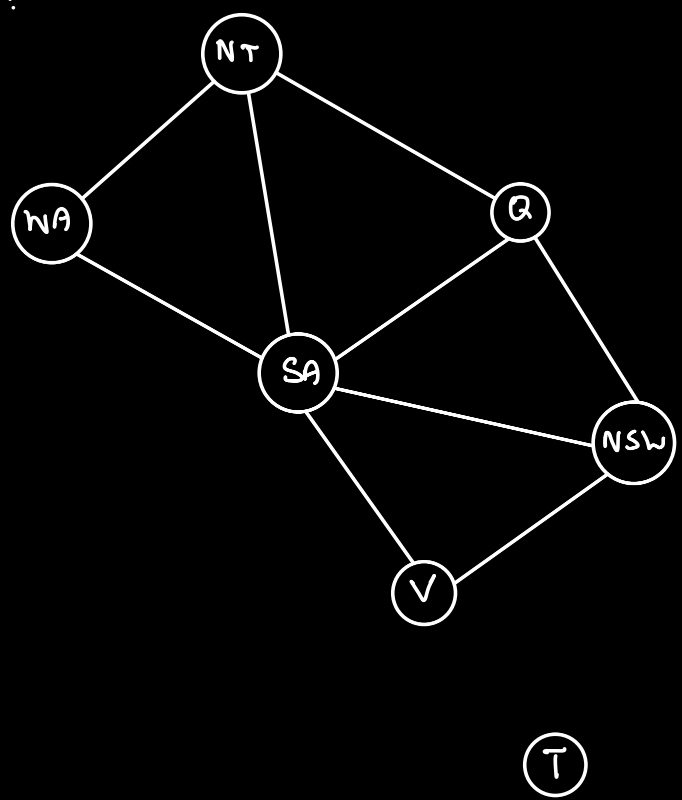
Solution:



Constraint Graph:

* Add a node for every variable
* Add edges if 2 variables are involved in a constraint.

Primal graph:



* If the constraint graph is a tree, then the CSP can be solved efficiently (Polynomial time).

TYPES OF CONSTRAINTS:

* Unary Constraints (One variable)

  SA ≠ green

* Binary Constraints (Binary CSP)

  SA ≠ WA

* Higher constraints

$\boxed{\text{* Hard Constraints}}$: Used here

* Soft Constraints

→ Preferences

I prefer red to green.

→ Weights on constraints

Constraint 1: w_1

Constraint 2: w_2

⋮

Satisfiability (SAT):

Variables : $x_1 \dots x_n$

Values : True , False

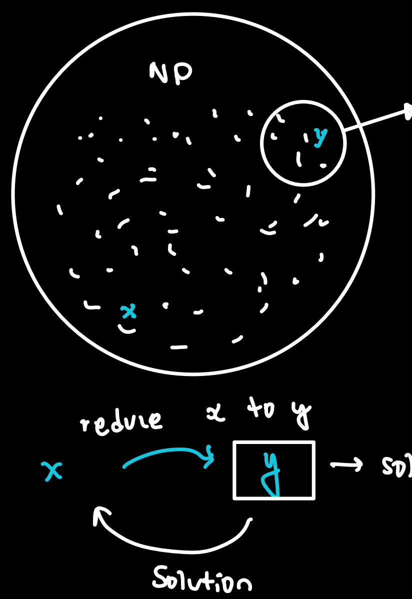
1 0

Constraints :

$$\left.\begin{array}{l} * \ x_1 \vee \neg x_2 \vee x_4 \\ * \ x_2 \vee x_5 \\ \vdots \end{array}\right] \text{Clauses}$$

NP - Complete:

3 SAT : every clause has three variables.



Hardest (complete for that class)

SAT is prototypical for NP.

CSP AS SEARCH:

* Initial State

* Final/ Goal State : " Final/Goal State Test "

* Actions / Successor function.

State : Partial variable assignment.

Initial State : Empty variable assignment

eg :

State : {x : True, z = False} x, y, z

State : { x : True, y : False , z = False }

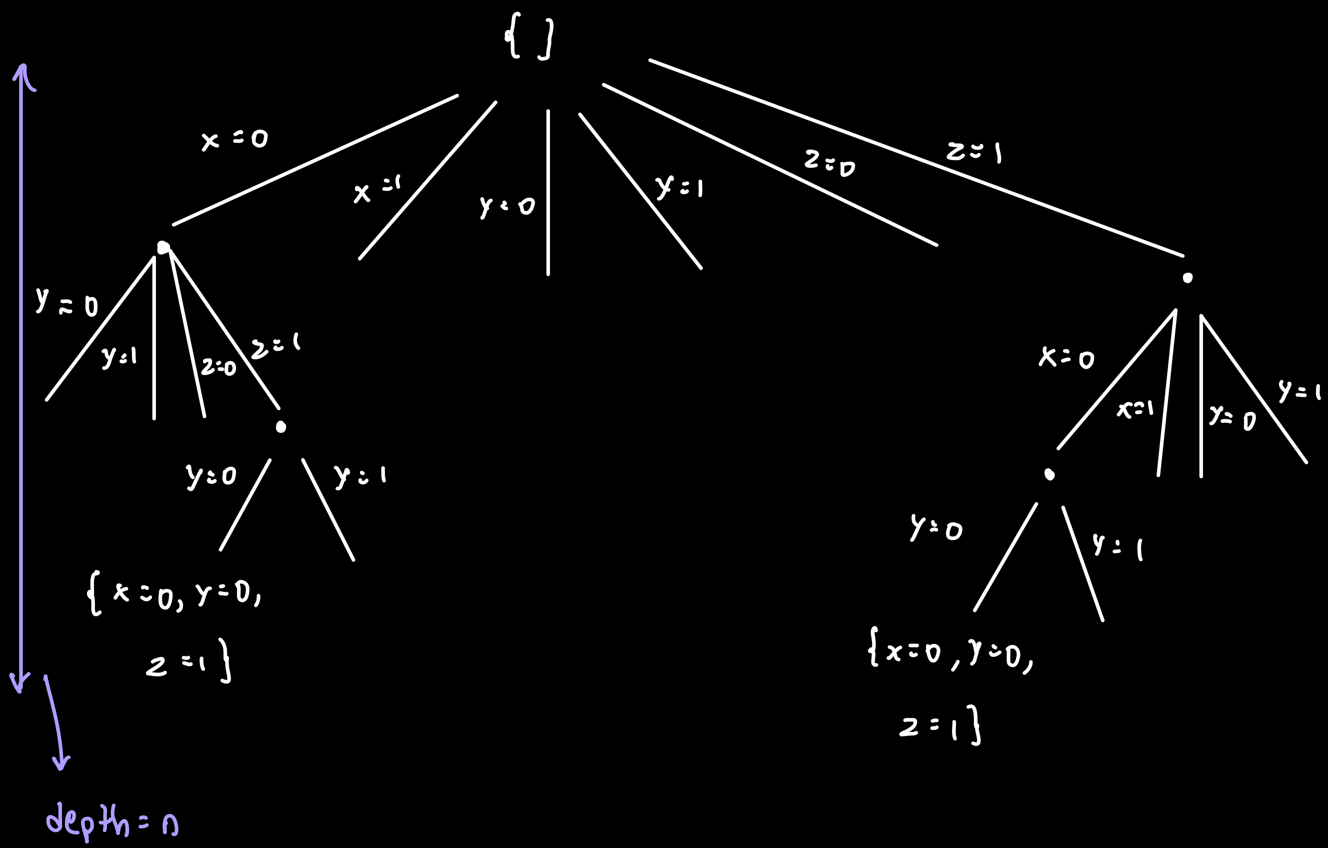
State {y : True}

Successors

- $x:0$ → {y : 1, x=0}
- $x=1$ → {y=1, x=1}
- $z=0$ → {y=1, z=0}
- $z=1$ → { y=1, z = 1}

Variables x, y, z $\quad n = 3$

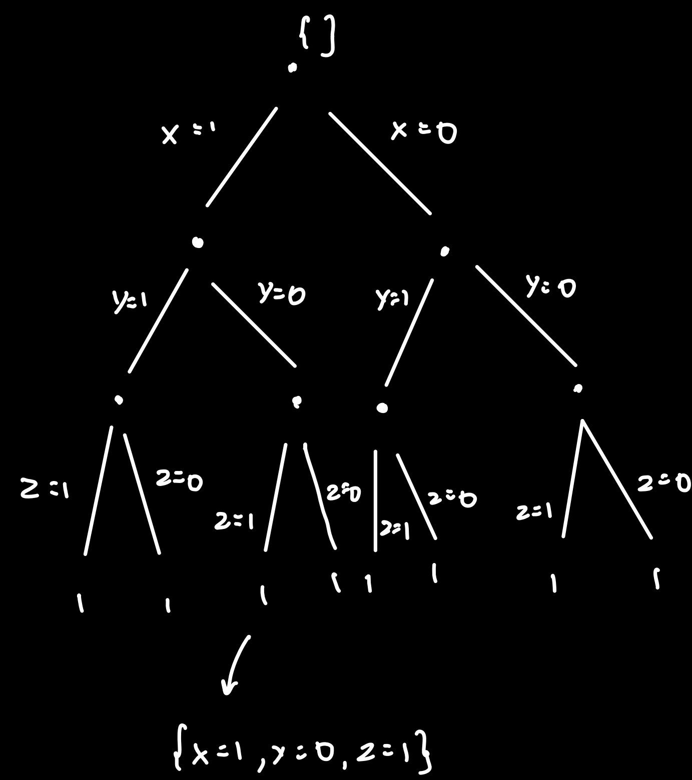
Values $0, 1$ $\quad d = 2$



$\boxed{n!\, d^n \text{ leaves}}$

$6 \times 2^3 = 48$

$\Rightarrow d^n$ distinct states



leaves $= d^n$ $(2^3 = 8)$

distinct variable assignment $= d^n$

Best search strategy:

DFS.

→ Best space complexity.

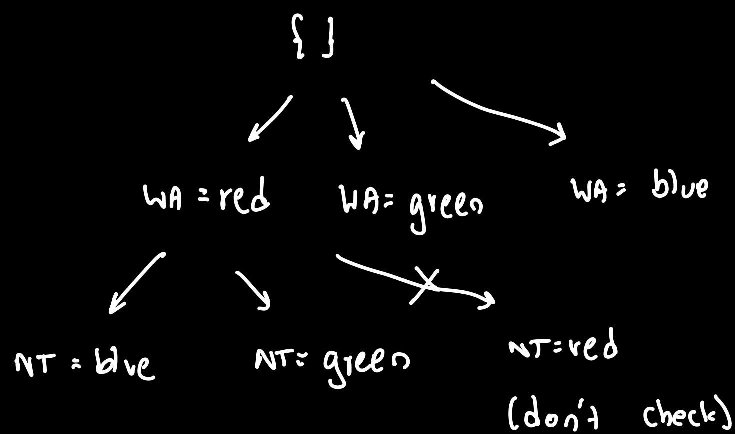
→ Complete: Finite depth

→ Optimality: Any of the solution is fine.

IMPROVEMENTS OF DFS FOR CSP:

1. Backtrack search

    Avoid children that causes constraint violation.



2. Value ordering
3. Variable ordering
4. Forwarding Checking
5. Arc Consistency

Detects bad states like Backtrack Search.

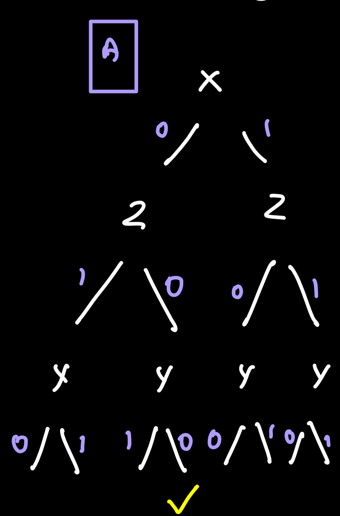
VARIABLE AND VALUE ORDERING:

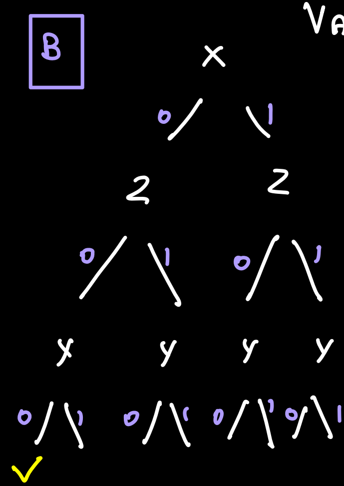
Variables x, y, z

Values $0, 1$

Constraints $x = y$; $y = z$

VAL ORDERING



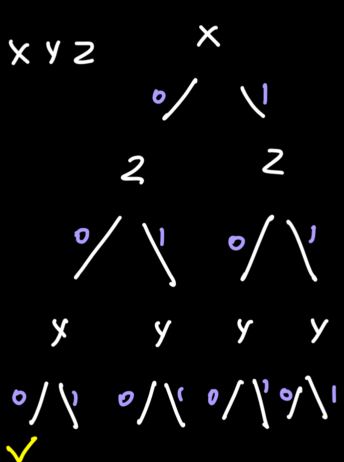


Better due to value ordering.

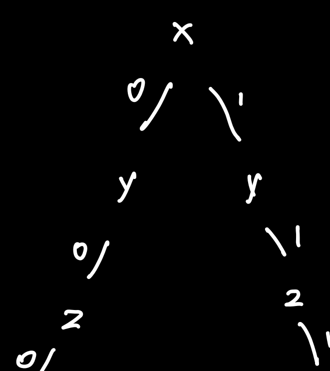
=> Order of visiting changes.

VAR ORDERING:

x y z



x y z

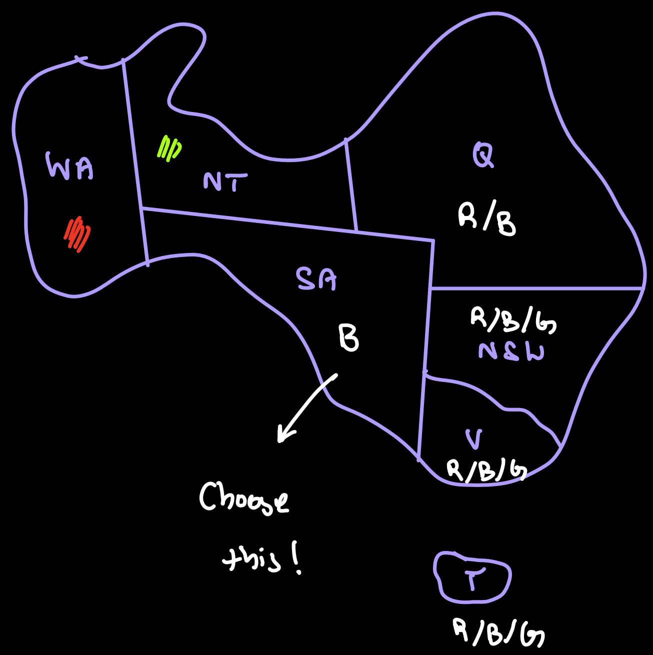


=> Size of tree changes.

VARIABLE ORDERING HEURISTIC:

MOST CONSTRAINED VARIABLE:

Choose the variable with the fewest legal values.

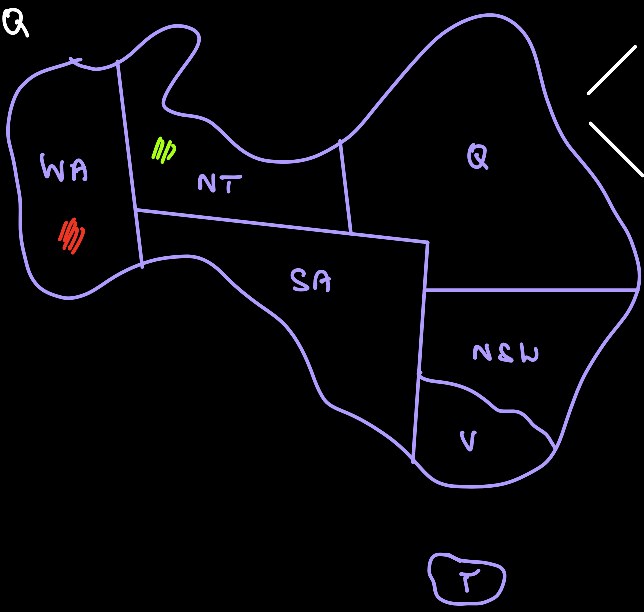


VALUE ORDERING HEURISTIC:

LEAST CONSTRAINING VALUE:

Choose value that rules out the fewest values of remaining variables.

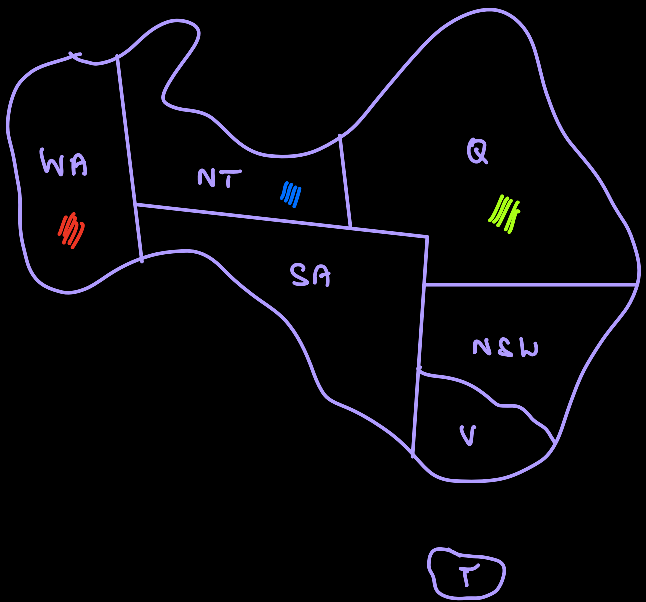
Next variable: Q



- allows 1 value for SA (B) ✓

- allows 0 values for SA.

FORWARD CHECKING (FC):



This is allowed state as per Backtrack Search.

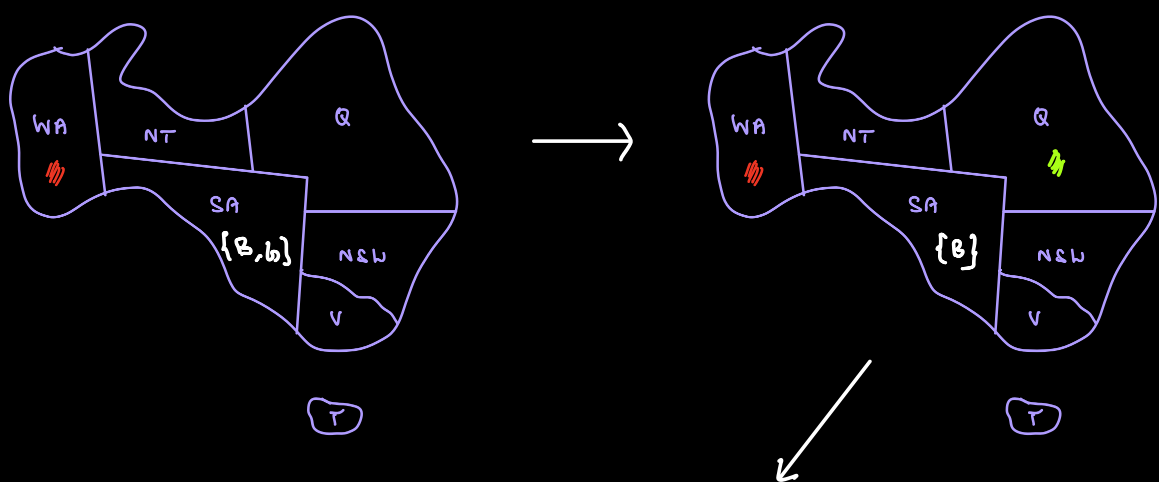
But we can see that this won't work, both NT, SA needs to be Blue.

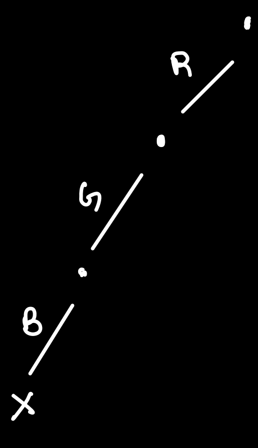
Forward Checking detects this!

* Maintain set of possible values for each variable.

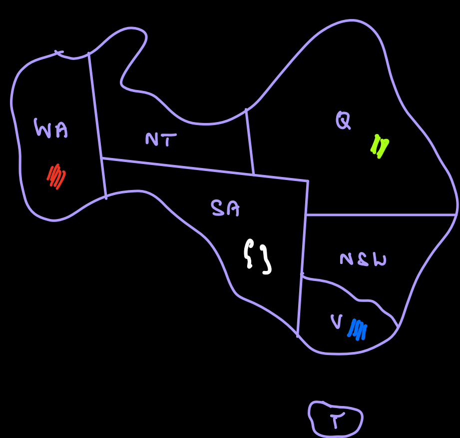
* When you assign a value to a variable, update possible values for other variable.

* Declare "bad state" if some variable looses all its values



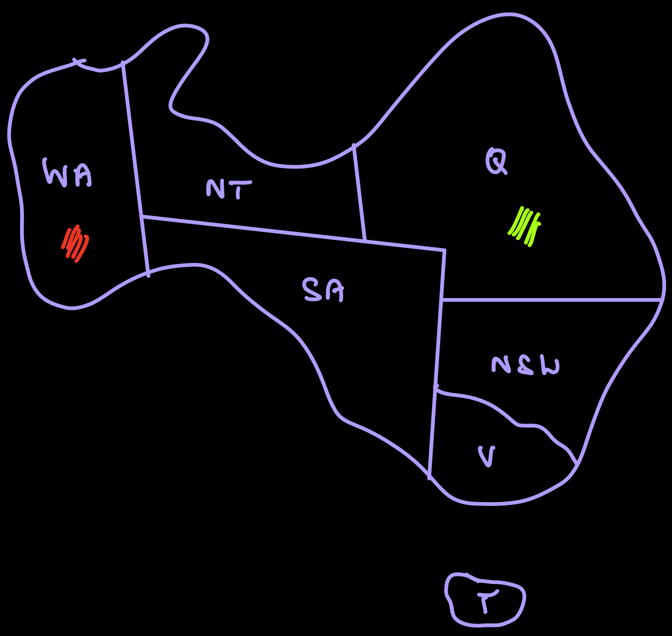


Bad State



BAD STATE!

Arc Consistency (AC):



Forward Checking detects Bad State only when NT/SA/NSW is colored blue.

Arc Consistency can detect now!

→ Arcs to be consistent $x \to y$

$$\left\{ \begin{array}{l} \text{for every possible value of } x \\ \text{there is a compatible value for } y \end{array} \right\}.$$