Constraint Satisfaction

Chapter 6
Part 1

Slides courtesy of Andrea Thomaz

About Search so far...

- Searching in a space of **states**
- Evaluate states with heuristics and goal test to guide search and see when we're done
- States have no specific structure

- Constraint Satisfaction Problems
- States and goal test have specific structure
- This allows us to use general purpose rather than problem specific heuristics!

Constraint Satisfaction Problem

State is a set of variables:

X_1	X_2	...	X_N
$\{d_1 \ldots . d_m\}$	$\{d_1 \ldots . d_m\}$		$\{d_1 \ldots . d_m\}$

Each can be one of a domain of values:

* Goal Test: constraints specifying allowable combinations of values for variables

Constraint Satisfaction Problem

- Formal Representation Language
- Allows general-purpose algorithms with more power than standard search algs

Example: Carpool

* Carpool scheduling (2 cars, 4 people)

Variables = P_1, ,P_2 , P_3, P_4

Domain = $\{C_1,C_2\}$

Constraints = $P_1=C_1$ $P_2=C_2$

$P_1 \neq P_3$ $P_2=P_4$

Ex Solution = $C_1=\{P_1\}$; $C_2=\{P_2,P_4,P_3\}$

Example: Class Sched

* Prof A and B, two classes, three class times

State vars = A_1, A_2, B_1, B_2

Domain = 10am, 11am, noon

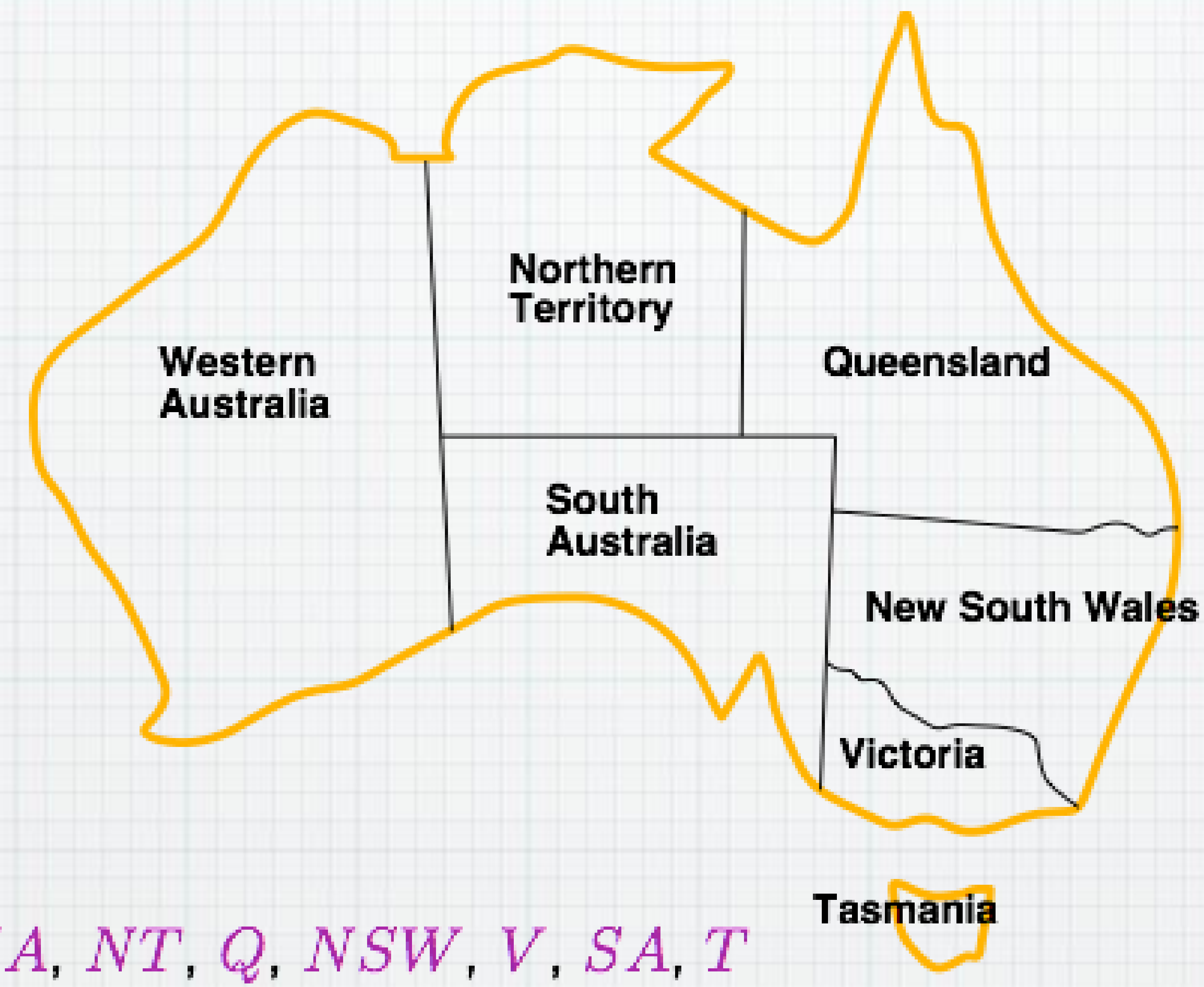
Constraints = $(A_1 \neq A_2)$; $(B_1 \neq B_2)$

Ex Solution = $\{A_1=10am; A_2=11am;$
$B_1=11am; B_2=noon\}$

Solving CSPs

* Each state of the problem is a possible assignment to some or all of the variables
* legal (consistent) assignment: no violations
* complete assignment: every var assigned

Map Example from Ch6



Variables WA, NT, Q, NSW, V, SA, T

Domains $D_i = \{red, green, blue\}$

Constraints: adjacent regions must have different colors

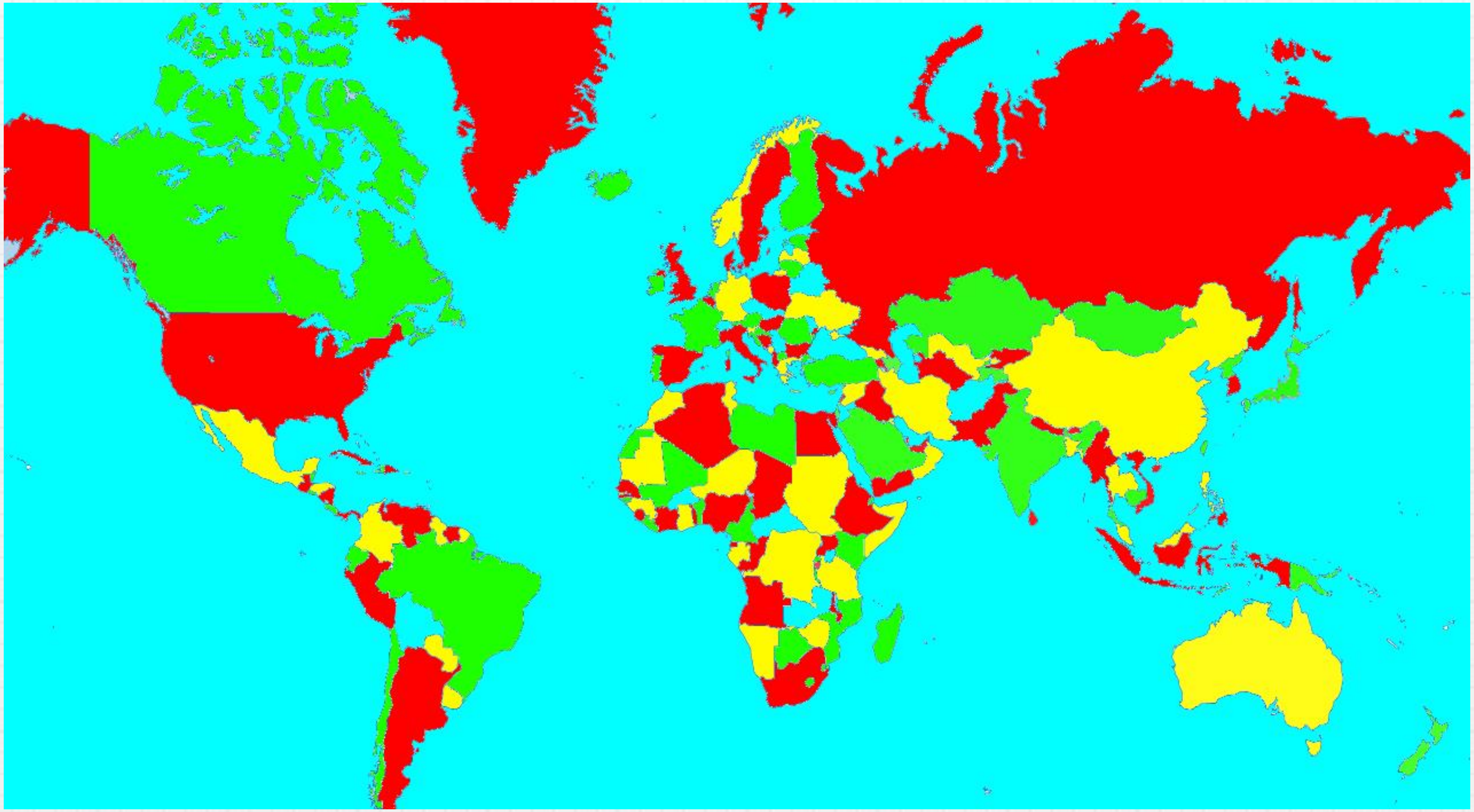
e.g., $WA \neq NT$ (if the language allows this), or

$(WA, NT) \in \{(red, green), (red, blue), (green, red), (green, blue), \ldots\}$

Map Example from Ch6

Solutions are assignments satisfying all constraints, e.g.,
$\{WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green\}$

4 Color Theorem



Only 4 colors are needed to color a planar map

Searching for a Solution

* Local search: wander state space, change whole state, test for goal state
* Now we know states have variables, and goals are constraints on those variables
* Use general purpose algorithms about constraints to construct a goal state

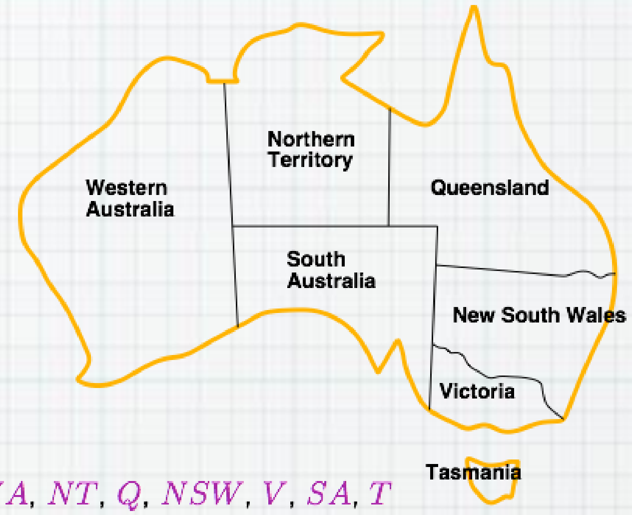
Constraint Graph

- Constraint Graph is a data structure we will use to represent the problem

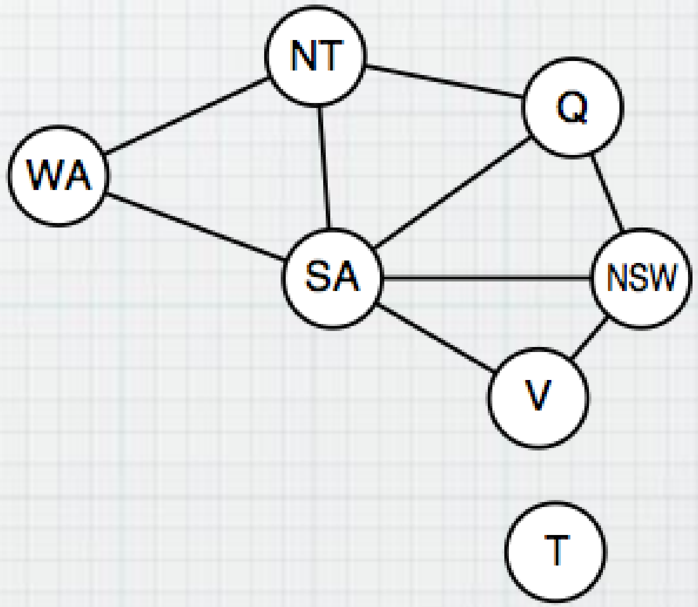
- Nodes are variables

- Arcs show which variables are constrained

- Binary CSP: each constraint relates only 2 variables



Variables WA, NT, Q, NSW, V, SA, T
Domains $D_i = \{red, green, blue\}$
Constraints: adjacent regions must have different colors
e.g., $WA \neq NT$ (if the language allows this), or
$(WA, NT) \in \{(red, green), (red, blue), (green, red), (green, blue), \ldots\}$



Varieties of CSPs

- Different kinds of variables
  - Discrete vars with finite domains size d: $O(d^n)$ complete assignments
  - Discrete vars, infinite domains, need constraint language: $Job_1 + 5 < Job_2$
  - Continuous vars, Ex: start/end times for scheduling problem

Varieties of CSPs

* Different kinds of constraints
  * Unary: single variable SA != green
  * Binary: 2 vars SA != WA
  * Higher order: 3 or more vars
  * Preferences (soft constraints): red better than green

CSP that you know!

- Vars = each square
- Domain $d = \{1...9\}$
- Constraints = all row squares different, all col squares different, all box squares different
- $O(9^{81})$ complete variable assignments

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

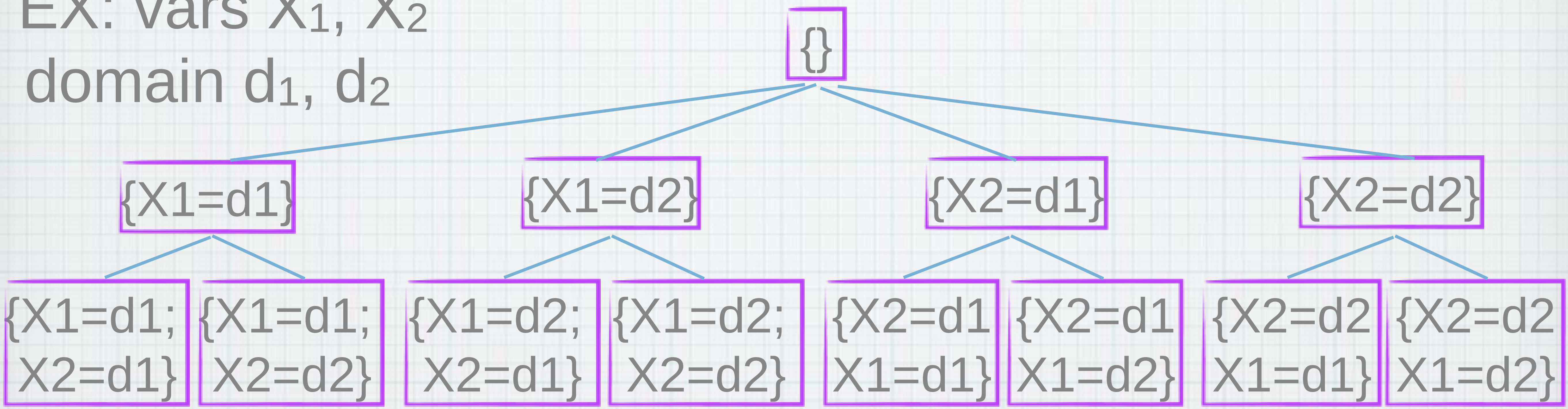
Search for CSP Solution

* Init State = {}
* Successor() = assign value (consistent with constraints) to any unassigned variable
* Goal Test = all vars are assigned
* Fail if no legal assignment to do
* Every solution appears at depth n, so DFS is complete
* Same formulation for every CSP problem, yea!
* Problem: n vars, with d values, branch factor at root is **nd**, then **(n-1)d** **...**terrible!

How to fix it...

- This standard formulation ignores key property = Communicativity

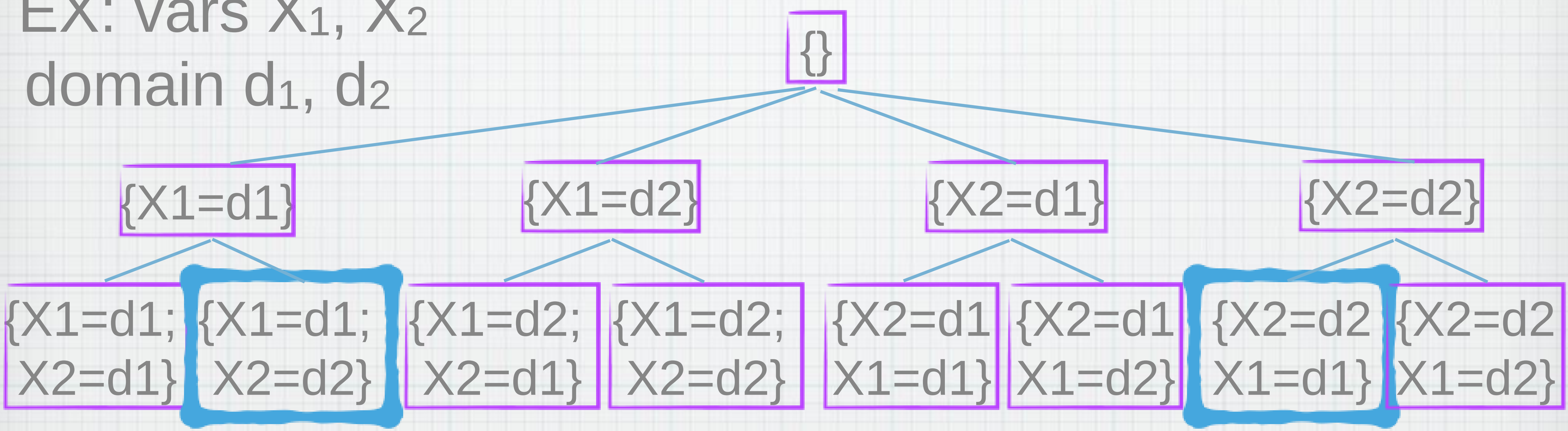
EX: vars X_1, X_2
domain d_1, d_2



How to fix it...

- This standard formulation ignores key property = Communicativity

EX: vars X_1, X_2
domain d_1, d_2



How to fix it...

* Order of var assignment doesn’t matter
* So consider a single var at each node of the search tree
* This with DFS is, Backtracking Search
* Do one var at a time and “backs up” if a previous assignment becomes infeasible

Backtracking Search

select a var to assign next

find value consistent with constraints; recurse to assign another

```
function BACKTRACKING-SEARCH(csp) returns solution/failure
    return RECURSIVE-BACKTRACKING({ }, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
        if value is consistent with assignment given CONSTRAINTS[csp] then
            add {var = value} to assignment
            result ← RECURSIVE-BACKTRACKING(assignment, csp)
            if result ≠ failure then return result
            remove {var = value} from assignment
    return failure
```

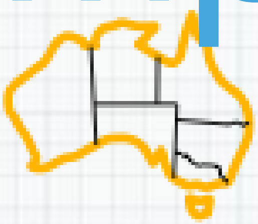
if no consistent assignment exists, return failure, which causes another value to be tried

Backtracking Search

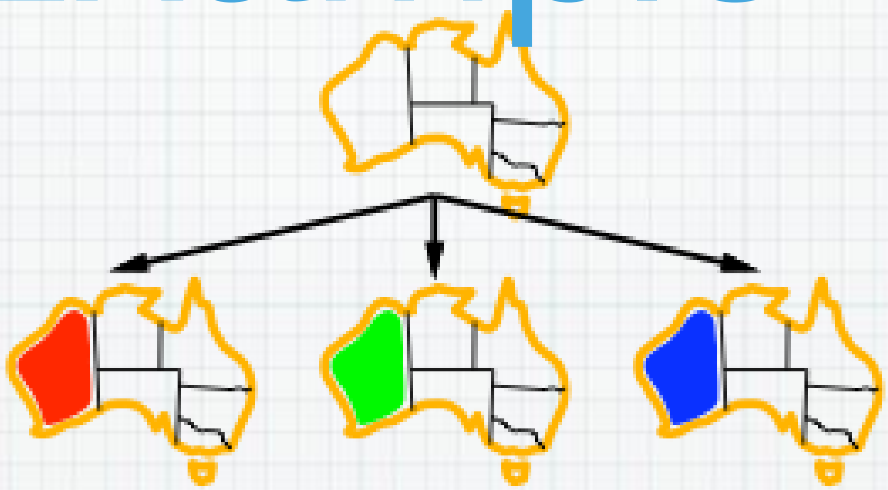
```
function BACKTRACKING-SEARCH(csp) returns solution/failure
    return RECURSIVE-BACKTRACKING({ }, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
        if value is consistent with assignment given CONSTRAINTS[csp] then
            add {var = value} to assignment
            result ← RECURSIVE-BACKTRACKING(assignment, csp)
            if result ≠ failure then return result
            remove {var = value} from assignment
    return failure
```

only keeps a single representation of the assigned state

Backtracking Example

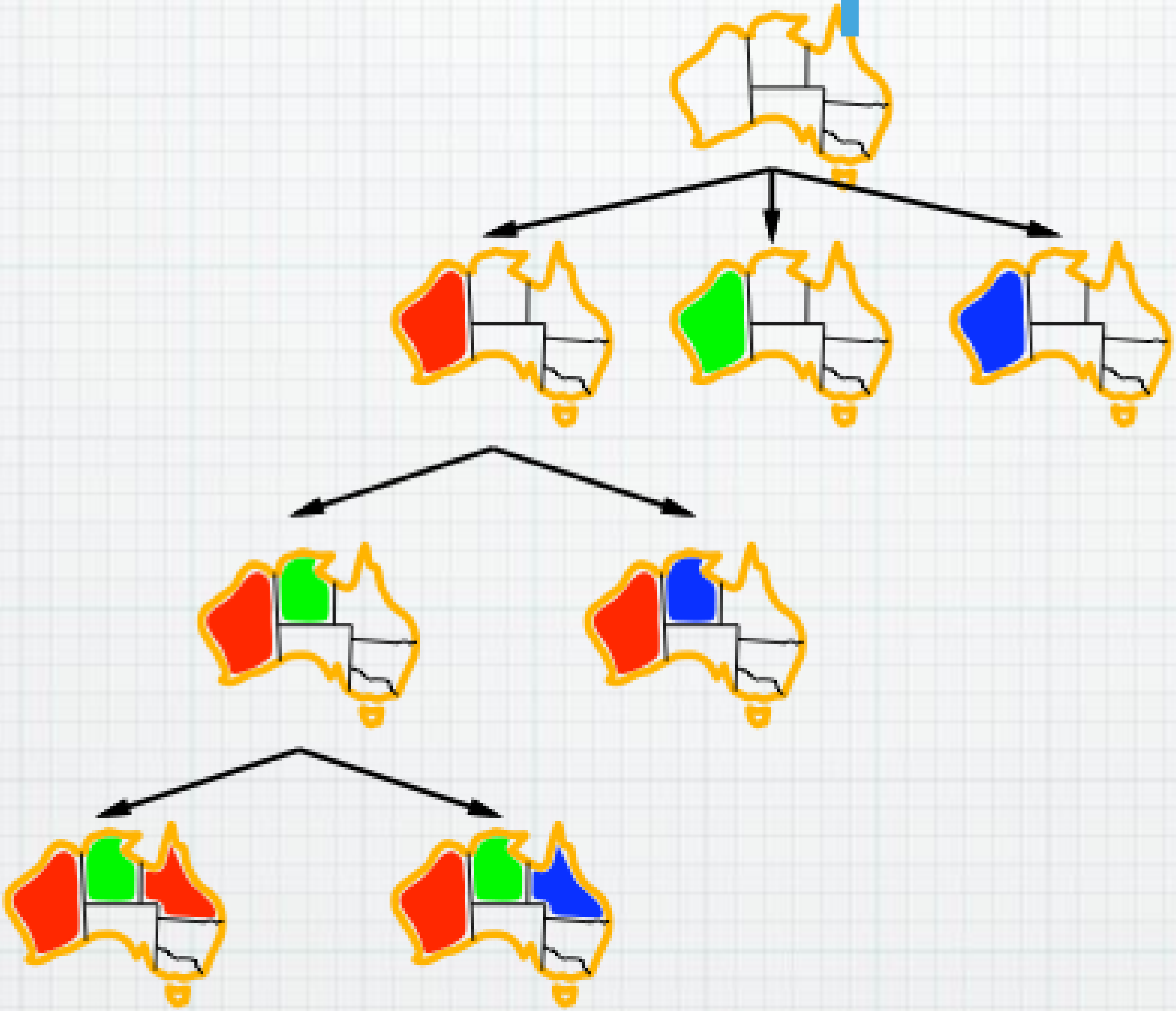


Backtracking Example



Backtracking Example

Backtracking Example



Improving Backtracking

* Plain backtracking is uninformed search...
* Usual improvement is to add heuristics
* Now we have general purpose heuristics related to the structure of CSPs instead of heuristics based on domain knowledge!

Improving Backtracking

* What variable to assign next?
* What order to try the domain values?
* What inference can be made to detect failure early?
* Can we take advantage of the problem structure?

Improving Backtracking

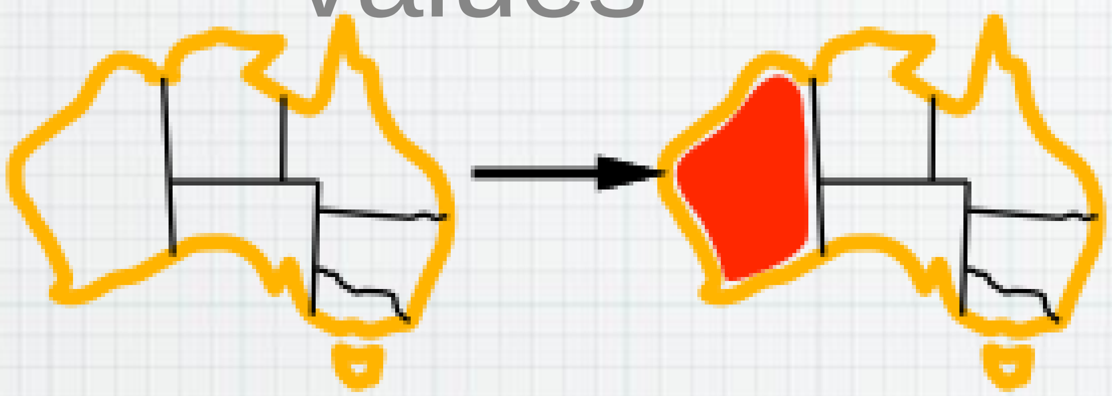
* What variable to assign next?
* What order to try the domain values?
* What inference can be made to detect failure early?
* Can we take advantage of the problem structure?

What var to do next?

* Simplest: some fixed order, or random
* Better idea: choose the most constrained variable as the next one to assign so it doesn't run out of options

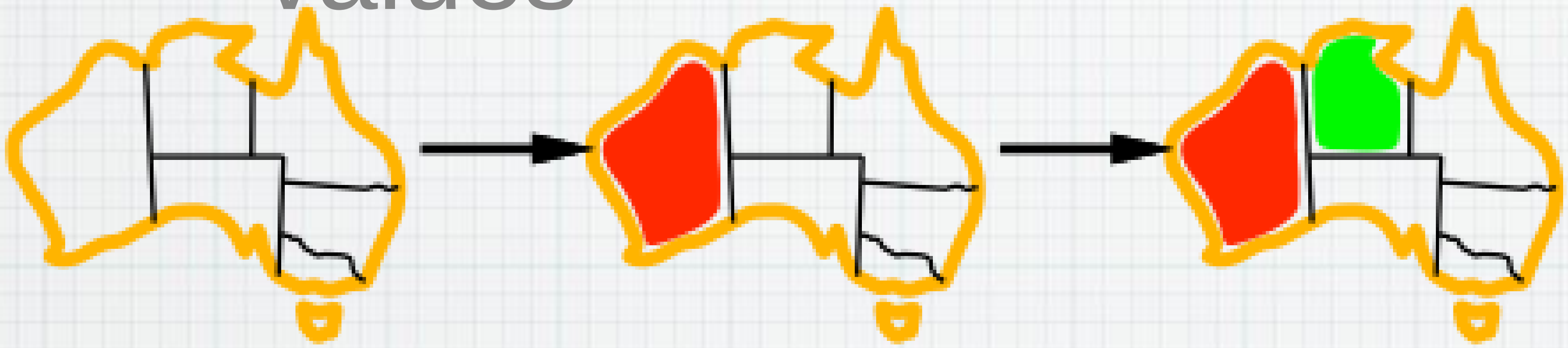
Minimum Remaining Values (MRV)

* Choose var with the fewest legal values



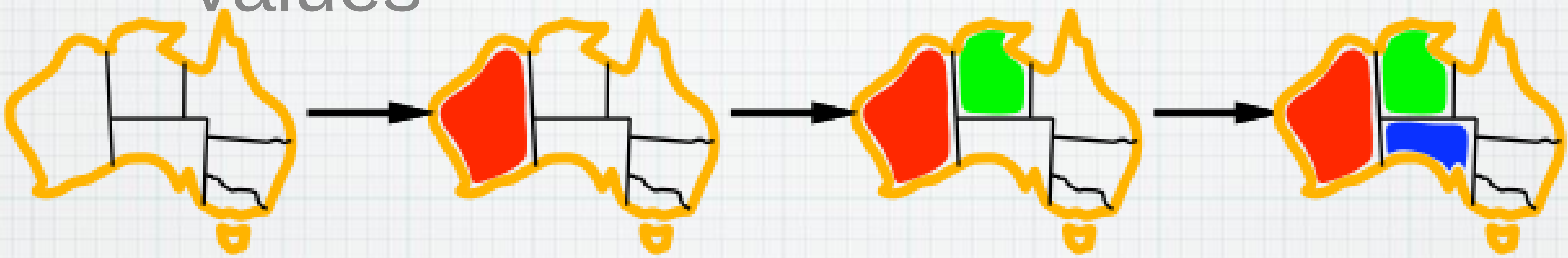
Minimum Remaining Values (MRV)

- Choose var with the fewest legal values



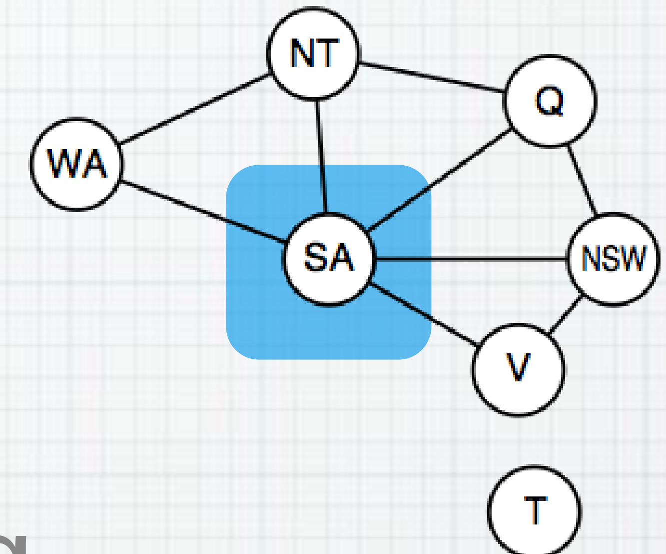
Minimum Remaining Values (MRV)

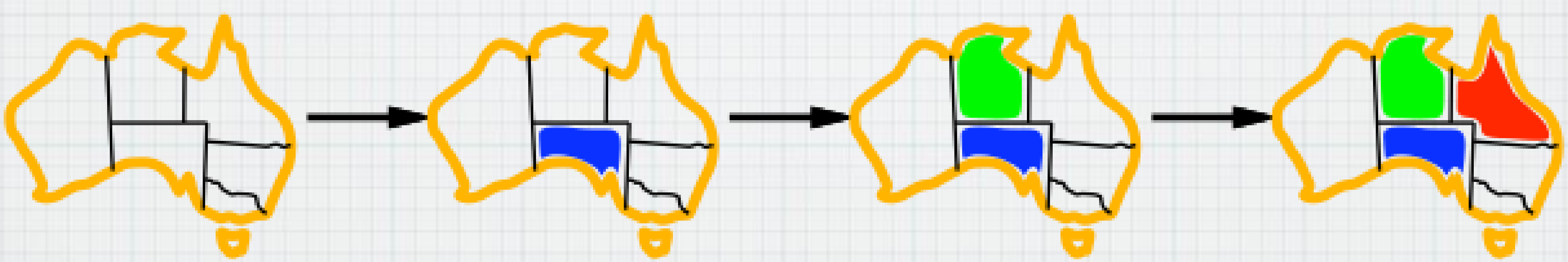
* Choose var with the fewest legal values



Degree Heuristic

* Tie breaker for MRV
* Choose var with most constraints on remaining variables



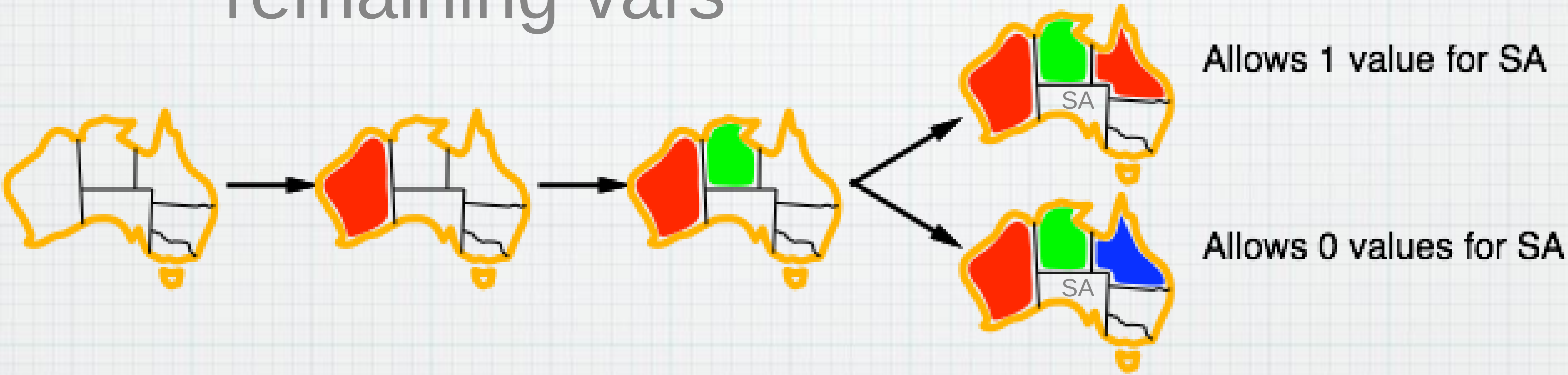


Improving Backtracking

* What variable to assign next?
* What order to try the domain values?
* What inference can be made to detect failure early?
* Can we take advantage of the problem structure?

Least Constraining Value

* Given a variable, choose value that rules out the least values in remaining vars



Improving Backtracking

* Variable ordering -- fail first, to minimize nodes in the search tree
* Value ordering -- fail last, we only need one solution so keep options open
* Can we interleave search and inference to narrow our choices even further?