CSC442: Introduction to Artificial Intelligence

Lecture 2.1

Constraint Satisfaction

The Problem With States

- Representation of states is specific to a problem domain
- Functions on states are specific to the state representation
- Heuristic functions are both!
- Many design choices, many opportunities for errors

Representation

Approach

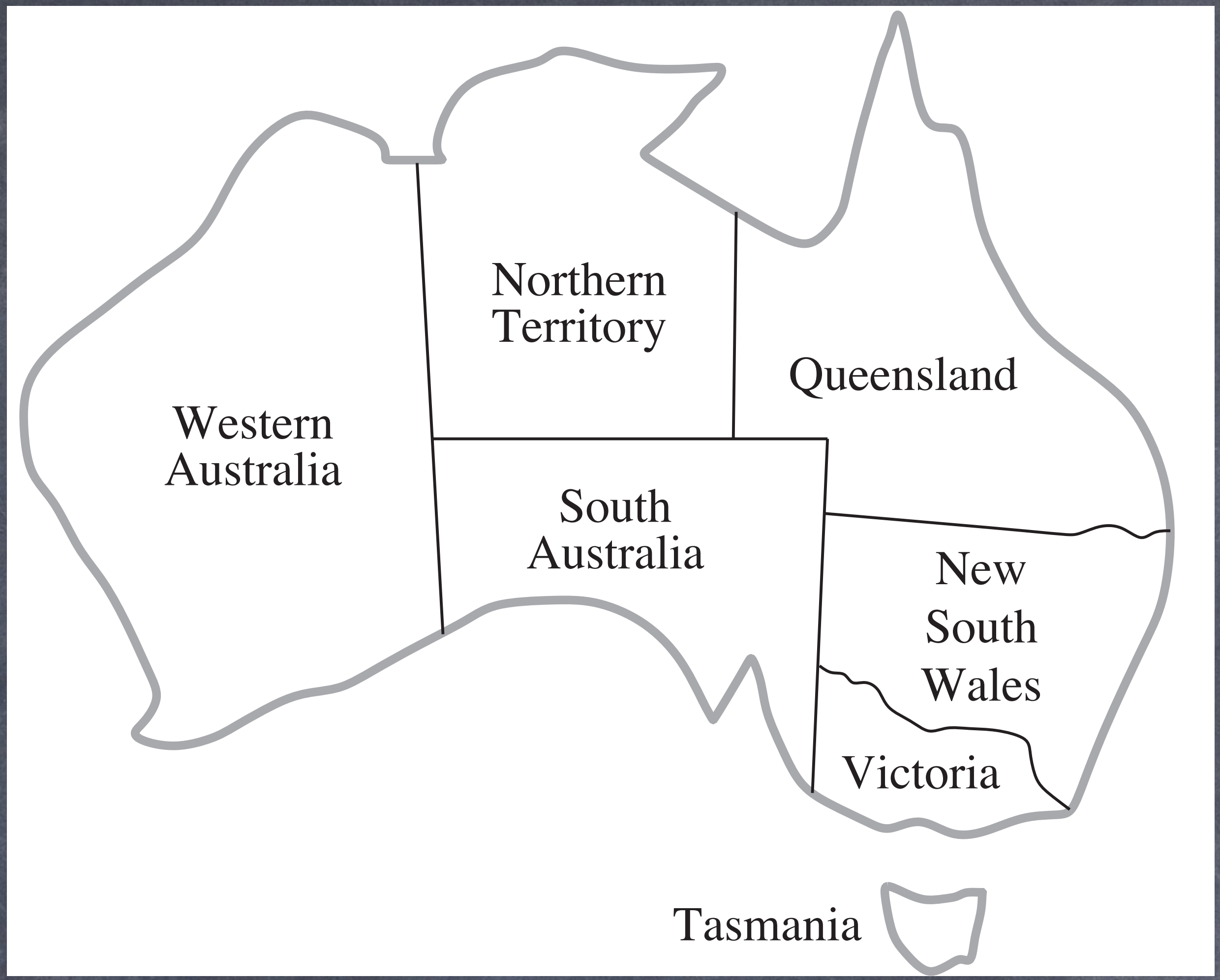
- Impose a structure on the representation of states
- Using that representation, successor generation and goal tests are domain-independent
- Can also develop effective problem- and domain-independent heuristics

Bottom Line

Represent State This Way → Write No Code! → Have No Bugs!

(By using off-the-shelf algorithms/tools).

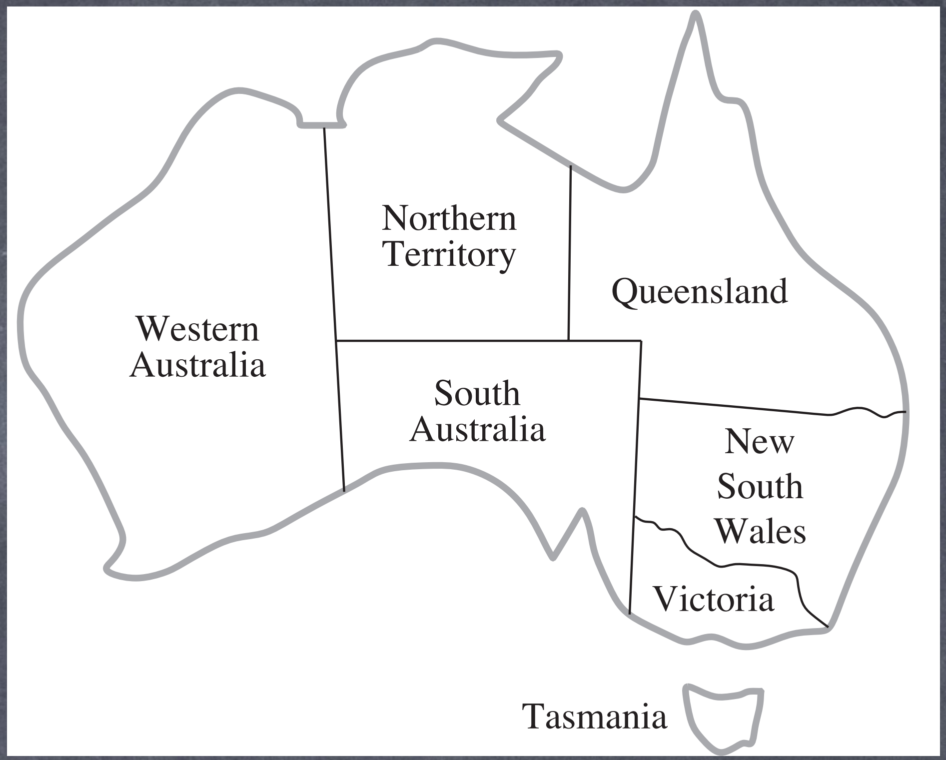
Example



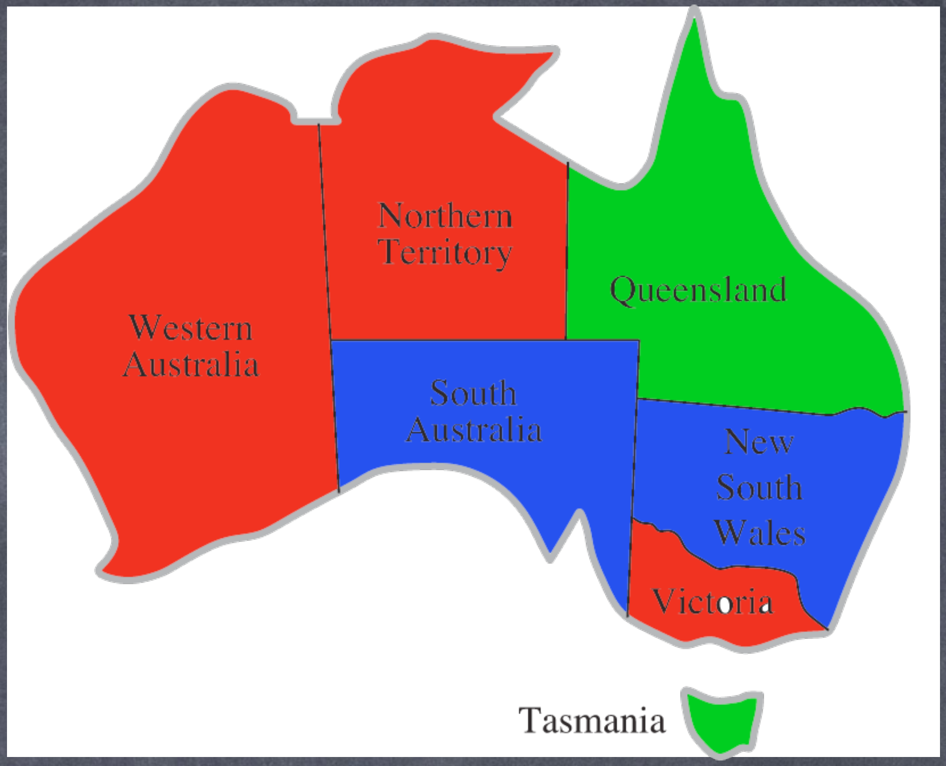
Assign a color to each region such that no two neighboring regions have the same color

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

enum Color = red, green, blue



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

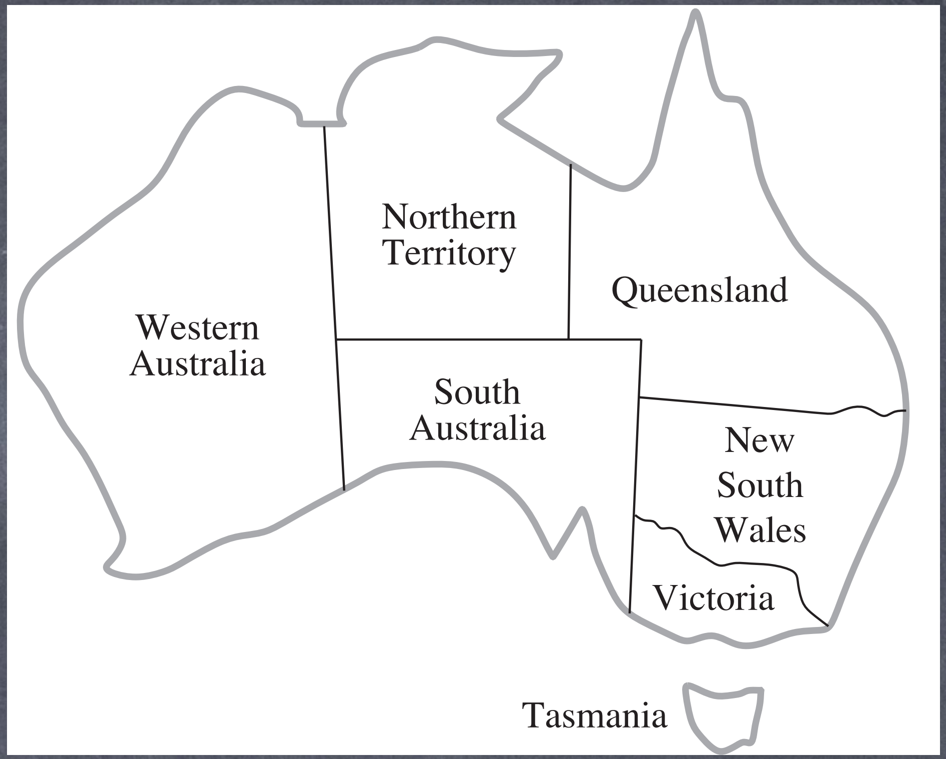


enum Color = red, green, blue

WA=red, NT=red, Q=green, NSW=blue

V=red, SA=blue, T=green

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

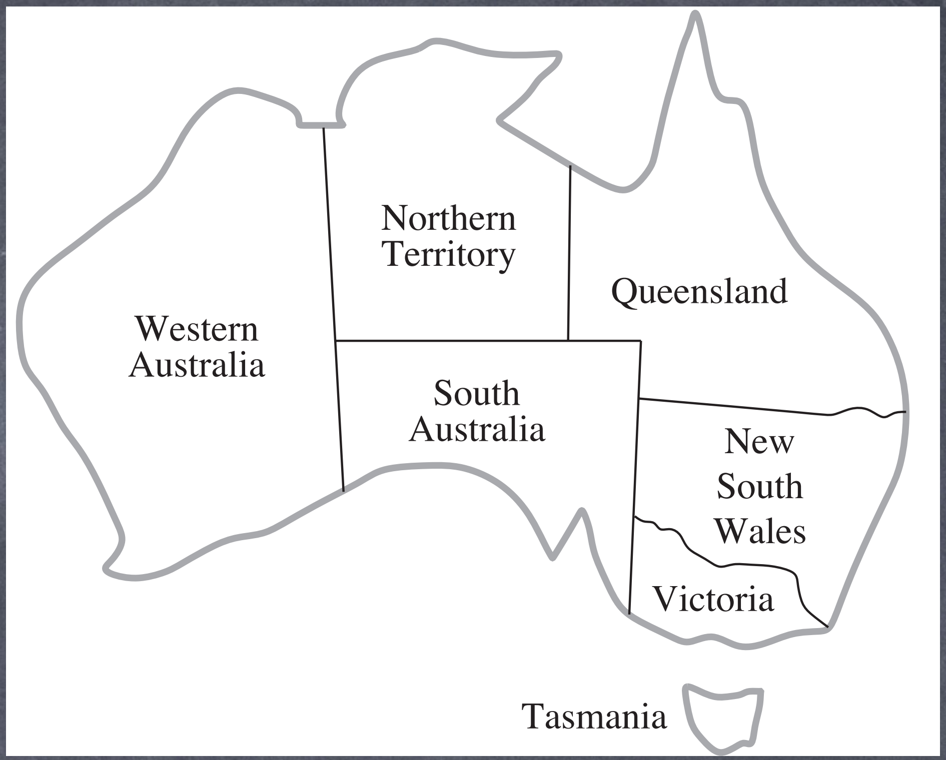


enum Color = red, green, blue

State: assignment of colors to regions

Action: pick an unassigned region and assign it a color

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



enum Color = red, green, blue

State: assignment of colors to regions

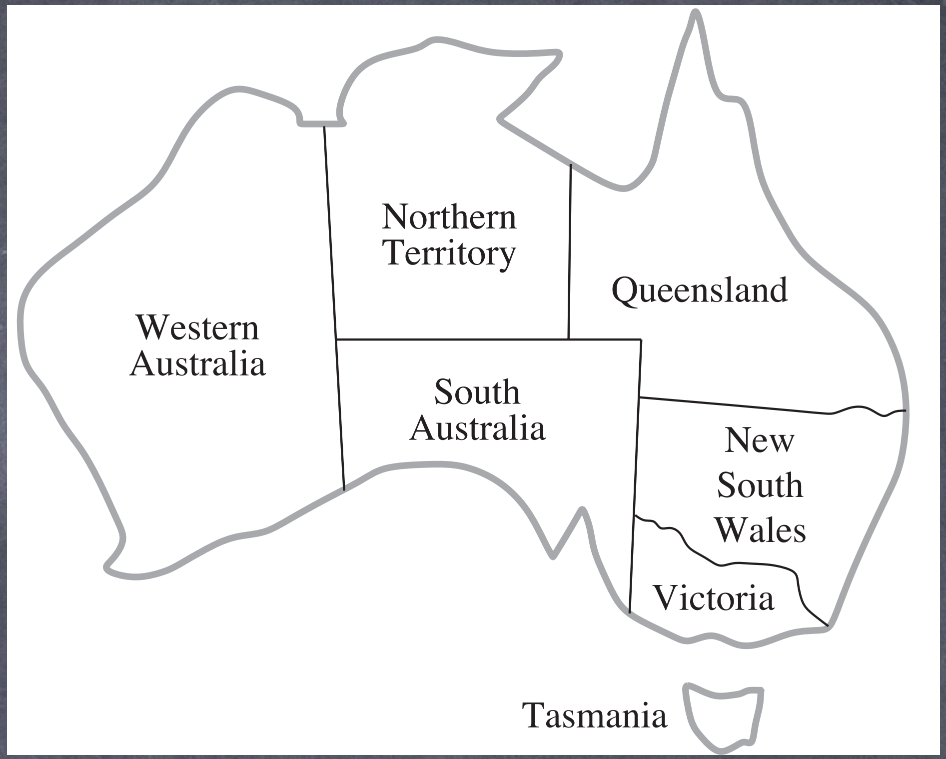
Action: pick an unassigned region and assign it a color

7*3*6*3*5*3*4*3*3*3*2*3*1*3 =

7!*3^7 = 11,022,480

$n!d^n$

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



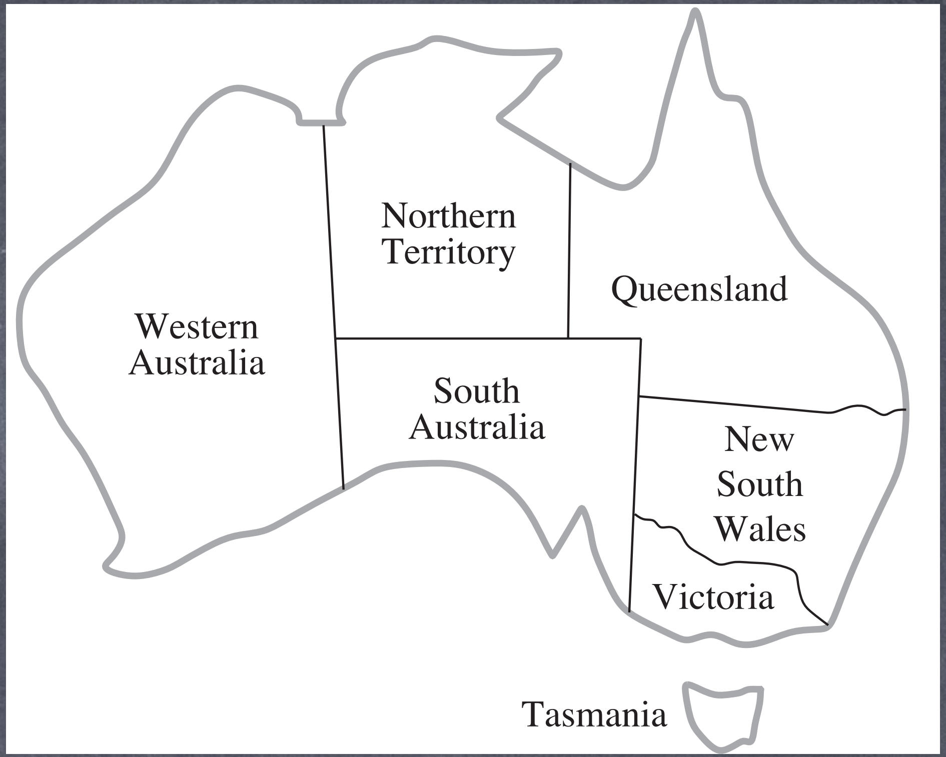
enum Color = red, green, blue

State: assignment of colors to regions

Goal test: All regions assigned and no adjacent regions have the same color

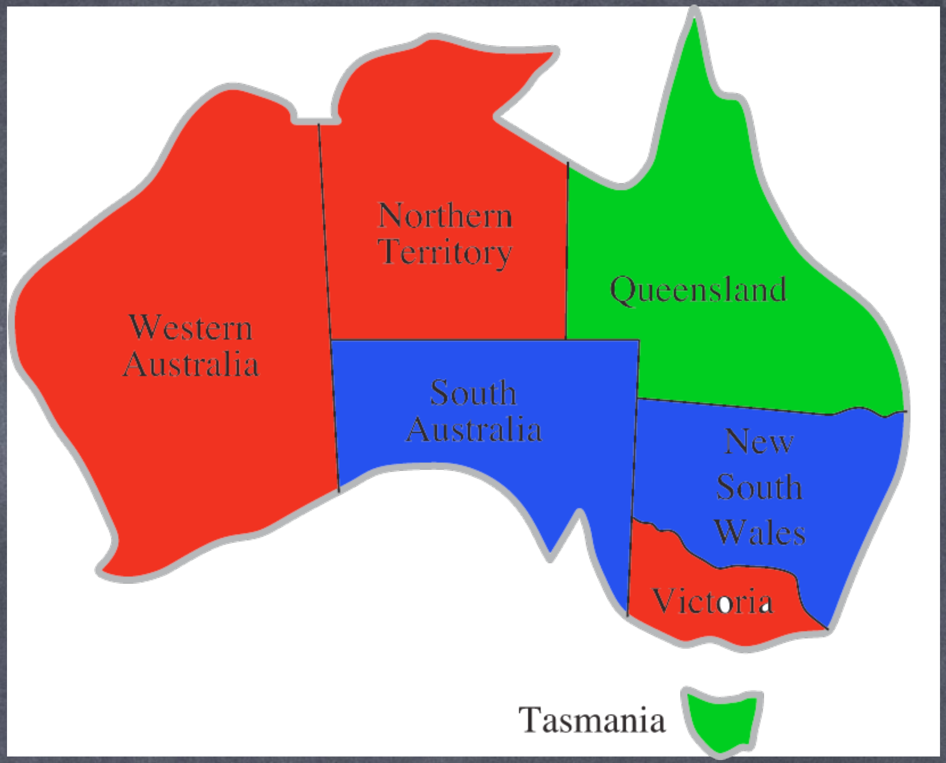
Color WA, NT, Q, NSW, V, SA, T

enum Color = red, green, blue




```
boolean isGoal(State s) {
  if (s.WA != s.SA && s.WA != s.NT && s.NT != s.Q && s.NT != s.SA &&
      s.SA != s.Q && s.SA != s.NSW && s.SA != s.V && s.Q != s.NSW &&
      s.NSW != s.V) {
    return true;
  } else {
    return false;
  }
}
```

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

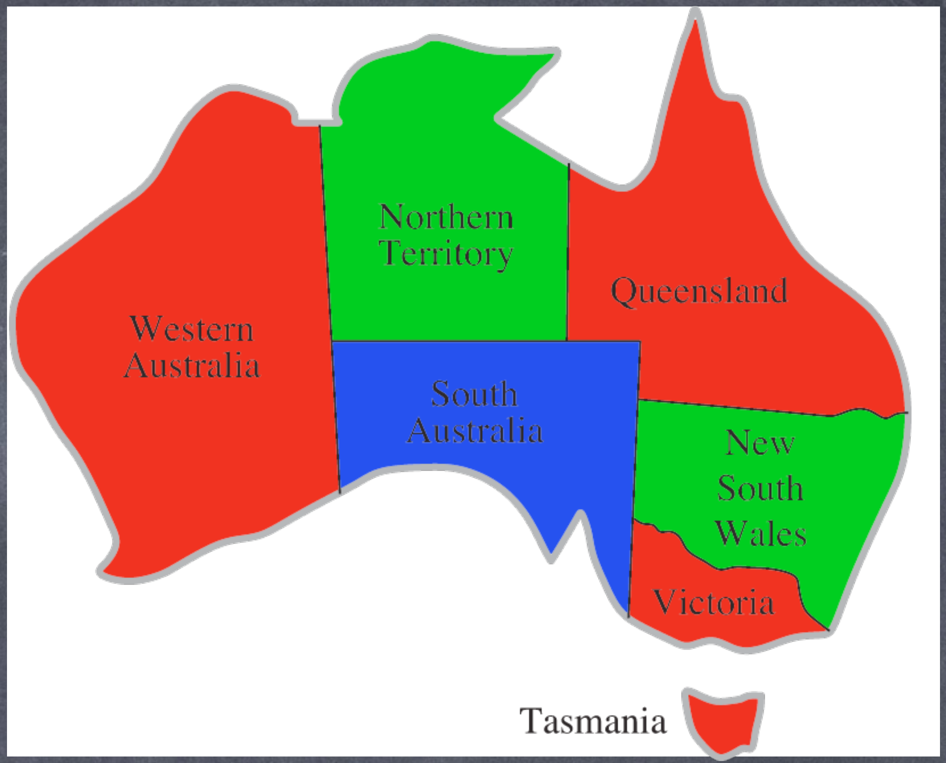


enum Color = red, green, blue

WA=red, NT=red, Q=green, NSW=blue

V=red, SA=blue, T=green

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



enum Color = red, green, blue

WA=red, NT=green, Q=red, NSW=green
V=red, SA=blue, T=red

Success!

Observation:

the states are compositional!

New Framework:

Constraint Satisfaction

Problem Solving State

- X: Set of variables { X_1, ..., X_n }
- Problem solving state is assignment of values to variables
- Assignment must satisfy constraints

Factored Representation

- Splits a state into variables (or attributes) that can have values
- Factored states can be more or less similar (unlike atomic states)
- Can also represent uncertainty (don't know the value of some attribute)

Constraint Satisfaction Problem (CSP)

- X: Set of variables $\{ X_1, ..., X_n \}$
- D: Set of domains $\{ D_1, ..., D_n \}$
  - Each D_i : set of values $\{ v_1, ..., v_k \}$
- C: Set of constraints $\{ C_1, ..., C_m \}$
- Solution: Assign to each X_i a value from D_i such that all the C_i are satisfied

Australia Map CSP

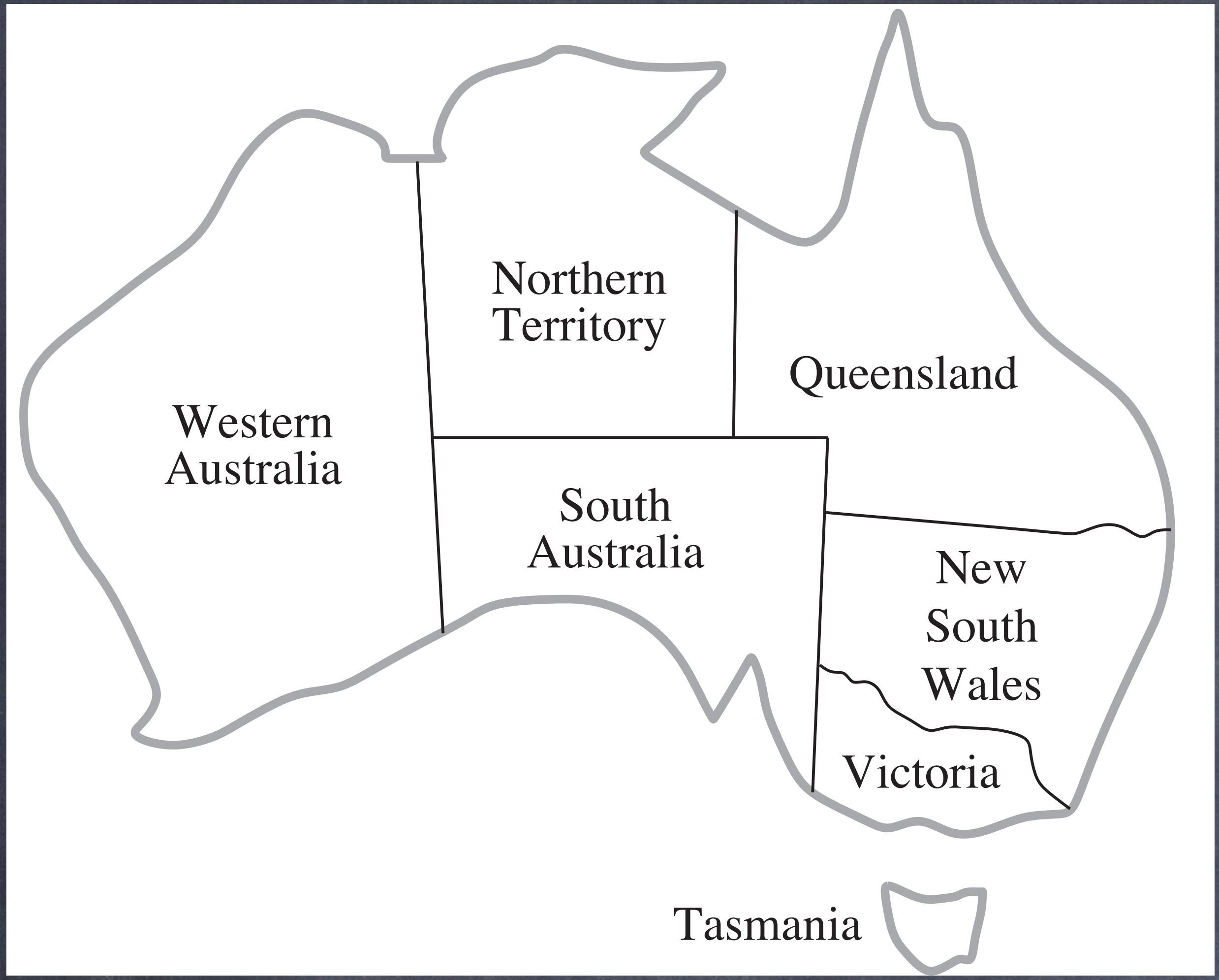
- Variables:
  $\{ X_i \} = \{$ WA, NT, Q, NSW, V, SA, T $\}$
- Domains: Each $D_i = \{$ red, green, blue $\}$
- Constraints:{ SA≠WA, SA≠NT, SA≠Q, SA≠NSW, SA≠V, WA≠NT, NT≠Q, Q≠NSW, VSW≠V }

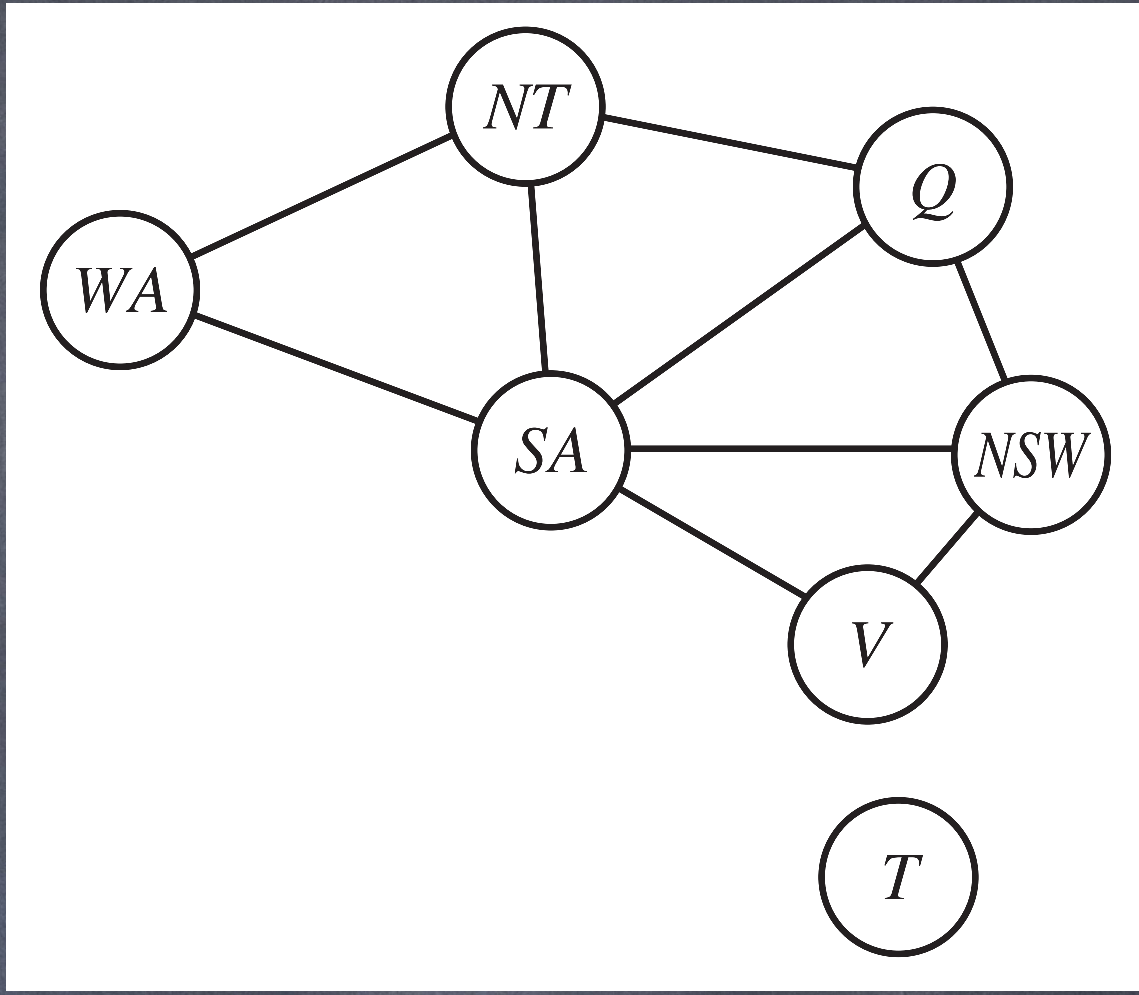
More CSP Terminology

- Assignments, Partial Assignment, Complete Assignment, Consistency, Solutions

More CSP Terminology

- Assignment: { $X_i = v_i$, ... } where $v_i \in D_i$
- Consistent: does not violate any constraints
- Partial: some variables are unassigned
- Complete: every variable is assigned
- Solution: consistent, complete assignment


Northern Territory
Queensland
Western Australia
South Australia
New South Wales
Victoria
Tasmania


NT
Q
WA
SA
NSW
V
T

Constraints

- Unary constraint: one variable
  - e.g., NSW ≠ red, X_i is even, $X_i = 2$
- Binary constraint: two variables
  - e.g., NSW ≠ WA, $X_i > X_j$, $X_i+X_j = 2$
- "Global" constraint: more than two vars
  - e.g., X_i is between X_j and X_k, AllDiff(X_i,X_j,X_k)
  - Can be reduced to set of binary constraints (possibly inefficiently)

Constraint Satisfaction Problem (CSP)

- X: Set of variables $\{ X_1, \ldots, X_n \}$
- D: Set of domains $\{ D_1, \ldots, D_n \}$
- Each D_i : set of values $\{ v_1, \ldots, v_k \}$
- C: Set of constraints $\{ C_1, \ldots, C_m \}$
- Solution: Assign to each X_i a value from D_i such that all the C_i are satisfied

Search for CSPs

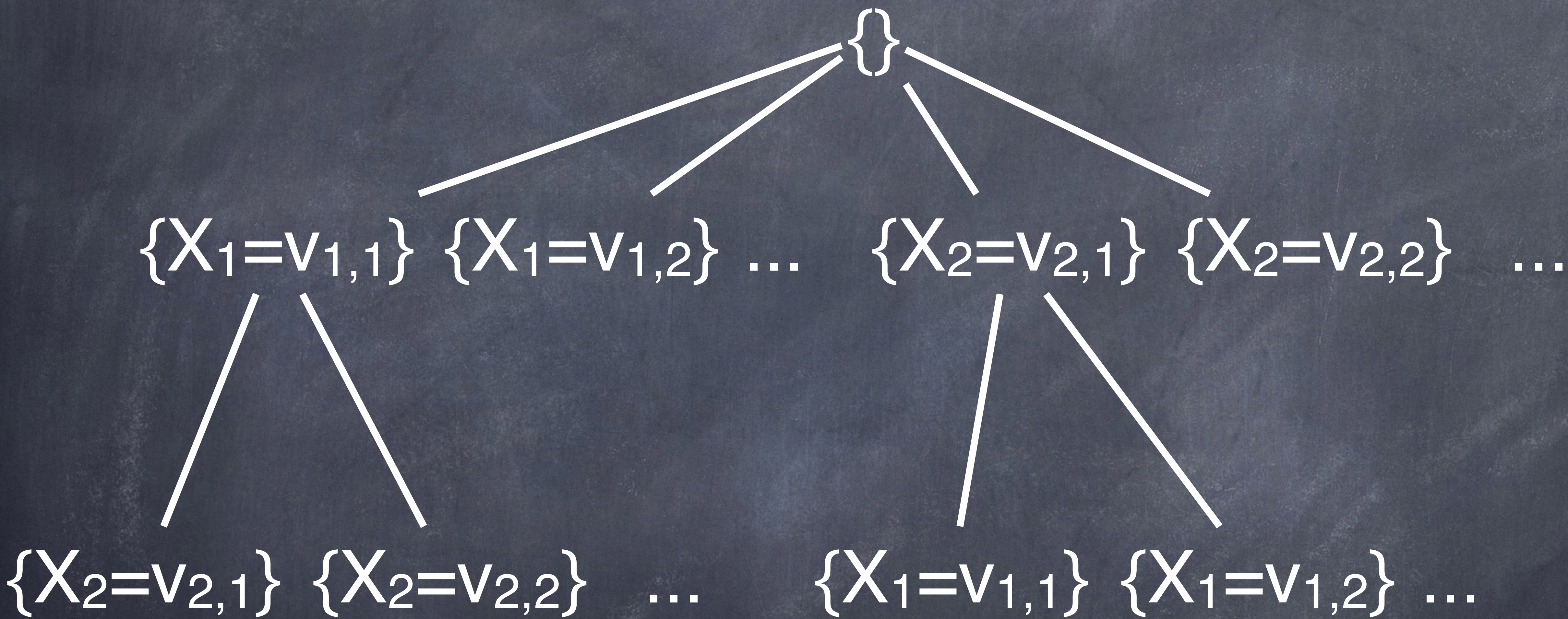
{}

Search for CSPs

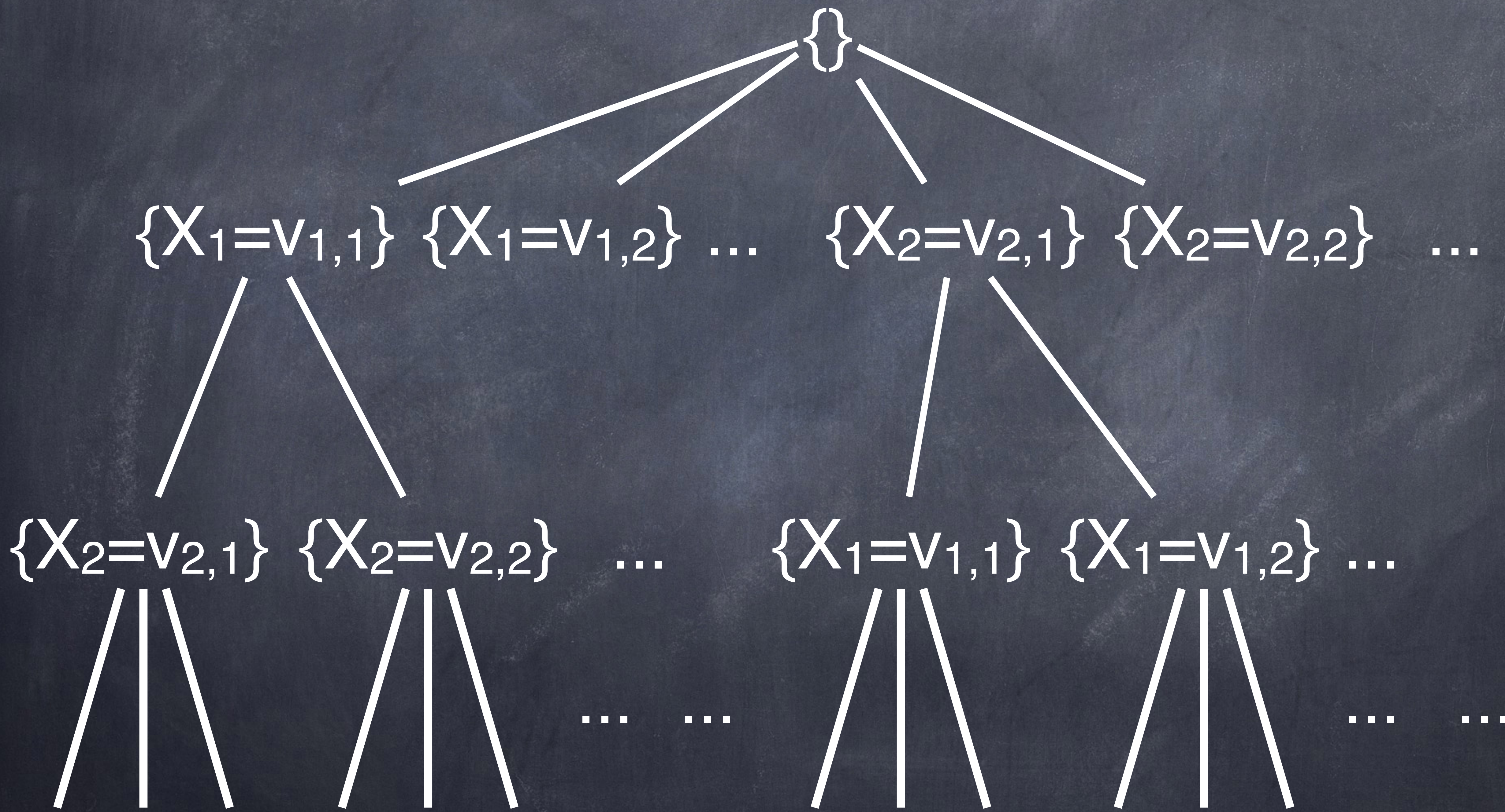
$$\{\}$$

$$\{X_1=v_{1,1}\} \quad \{X_1=v_{1,2}\} \quad \ldots \quad \{X_2=v_{2,1}\} \quad \{X_2=v_{2,2}\} \quad \ldots$$

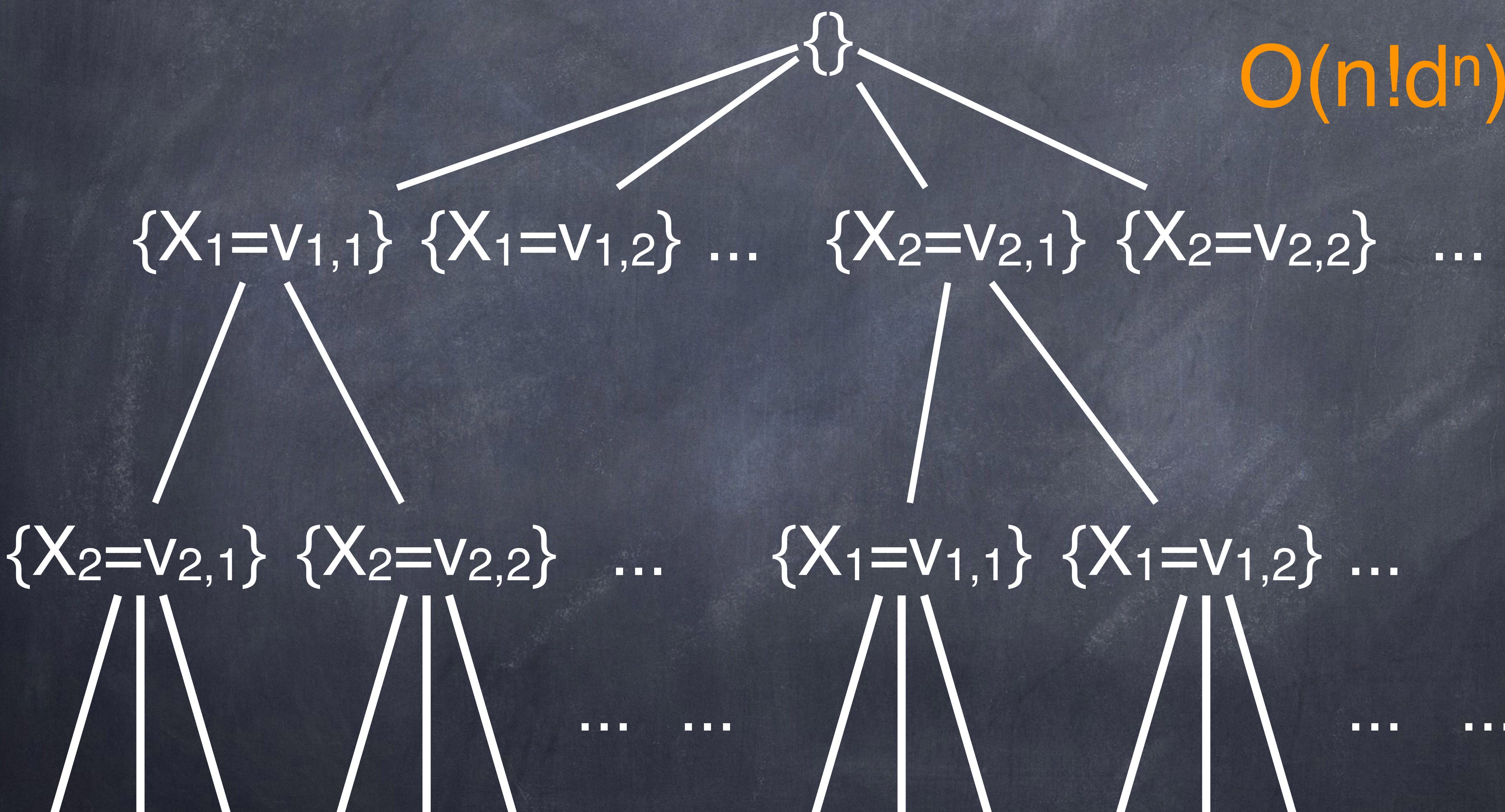
Search for CSPs



Search for CSPs

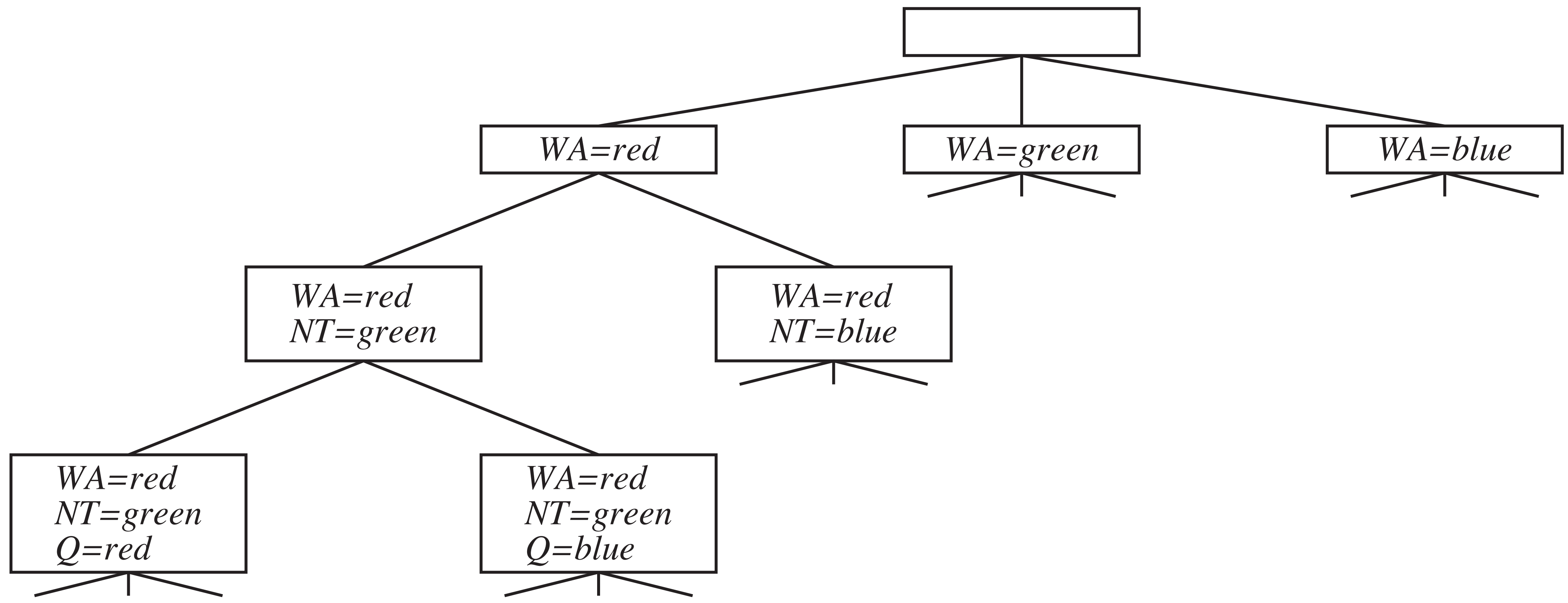


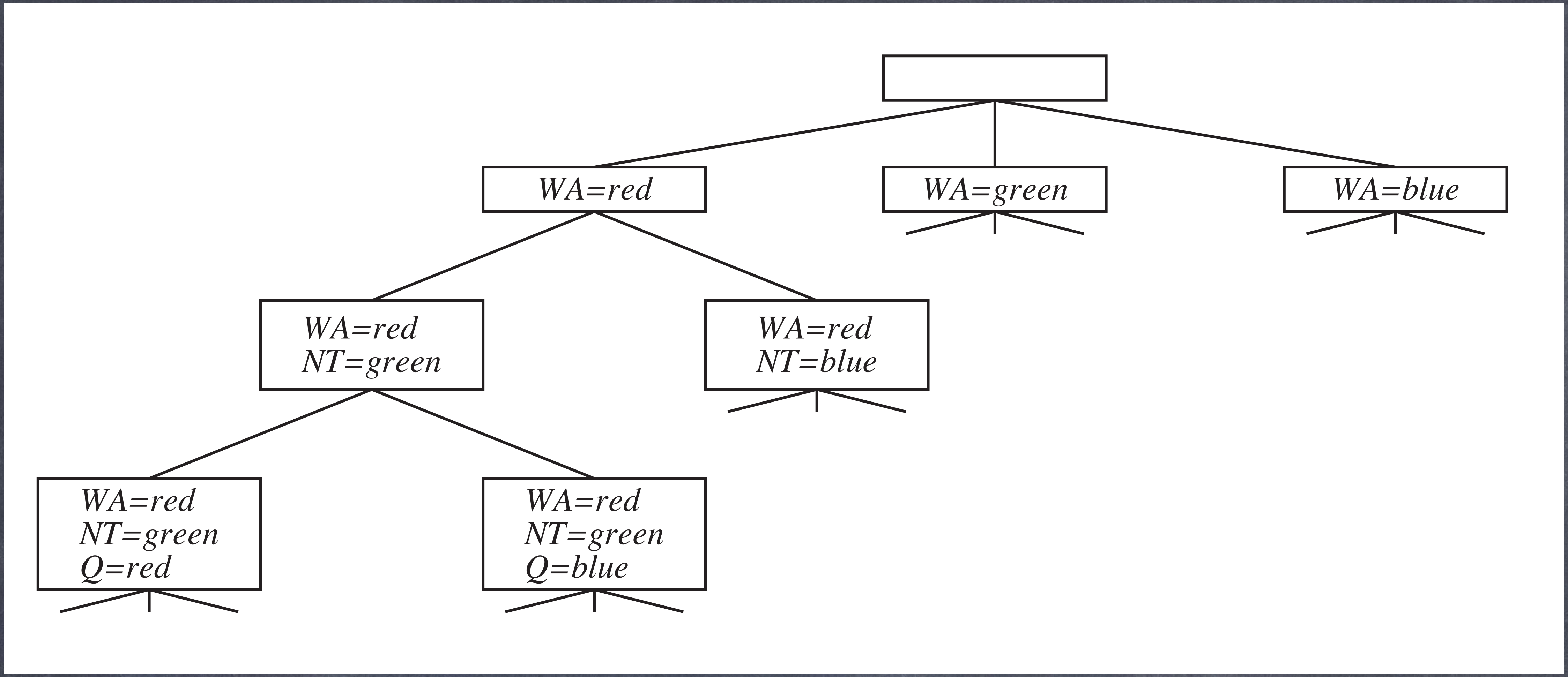
Search for CSPs



CSPs are Commutative

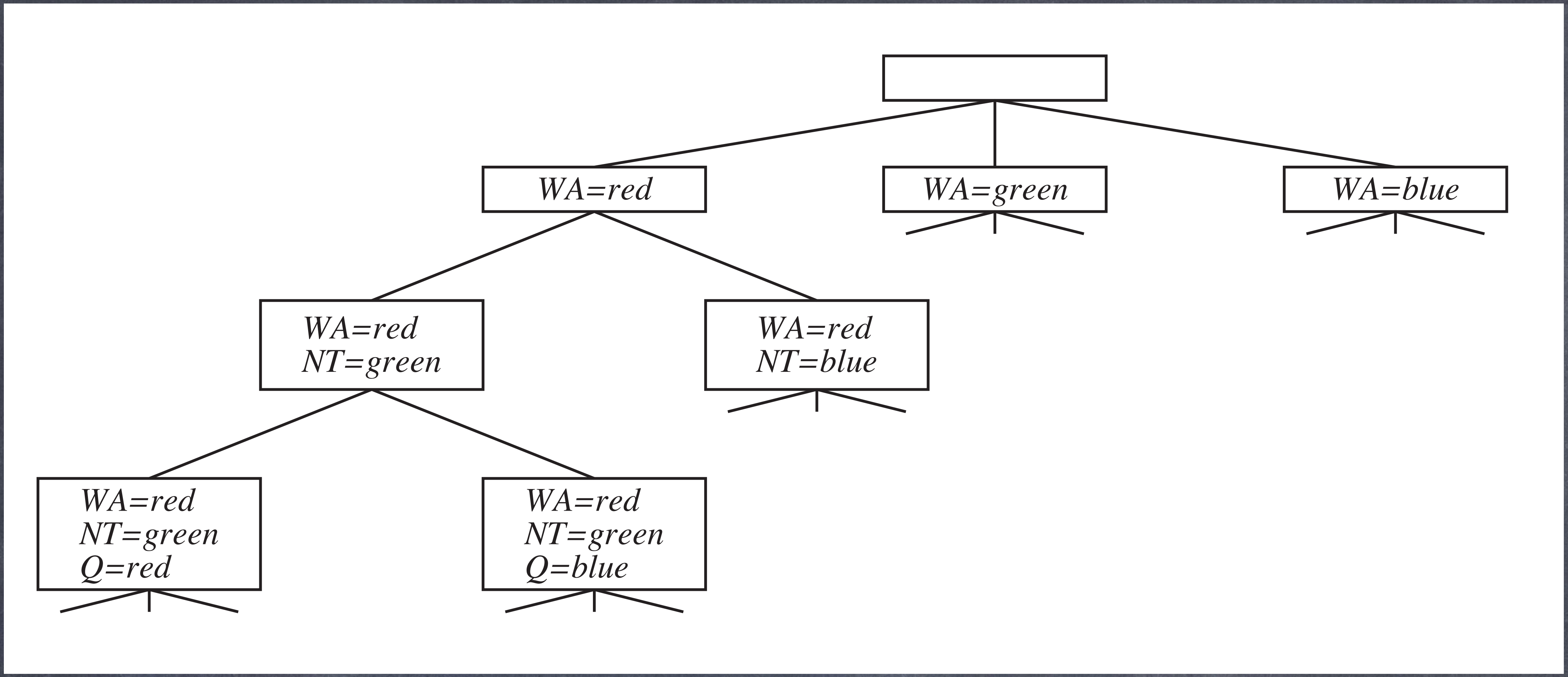
- CSPs are commutative because we reach the same partial assignment regardless of order
- Need only consider assignment to a single variable at each node in the search tree


WA=red
WA=green
WA=blue
WA=red
NT=green
WA=red
NT=blue
WA=red
NT=green
Q=red
WA=red
NT=green
Q=blue



n levels (one per variable), at most d nodes per level:

$$O(d^n)$$



- If we have no legal choice or an empty domain
- Then a state cannot be extended to a complete, consistent assignment

Prune!

Backtracking Search

```
function BT(csp)
  return backtrack({}, csp)

function backtrack(assignment, csp)
  if (assignment is complete)
    return assignment
  var = SelectUnassignedVar(csp)
  foreach value in OrderDomainValues(var,assignment,csp)
    if (value is consistent with assignment)
      add <var,value> to assignment
      result = backtrack(assignment, csp)
      if (result != failure)
        return result
      else
        remove <var,value> from assignment
  return failure
```

Backtracking Search

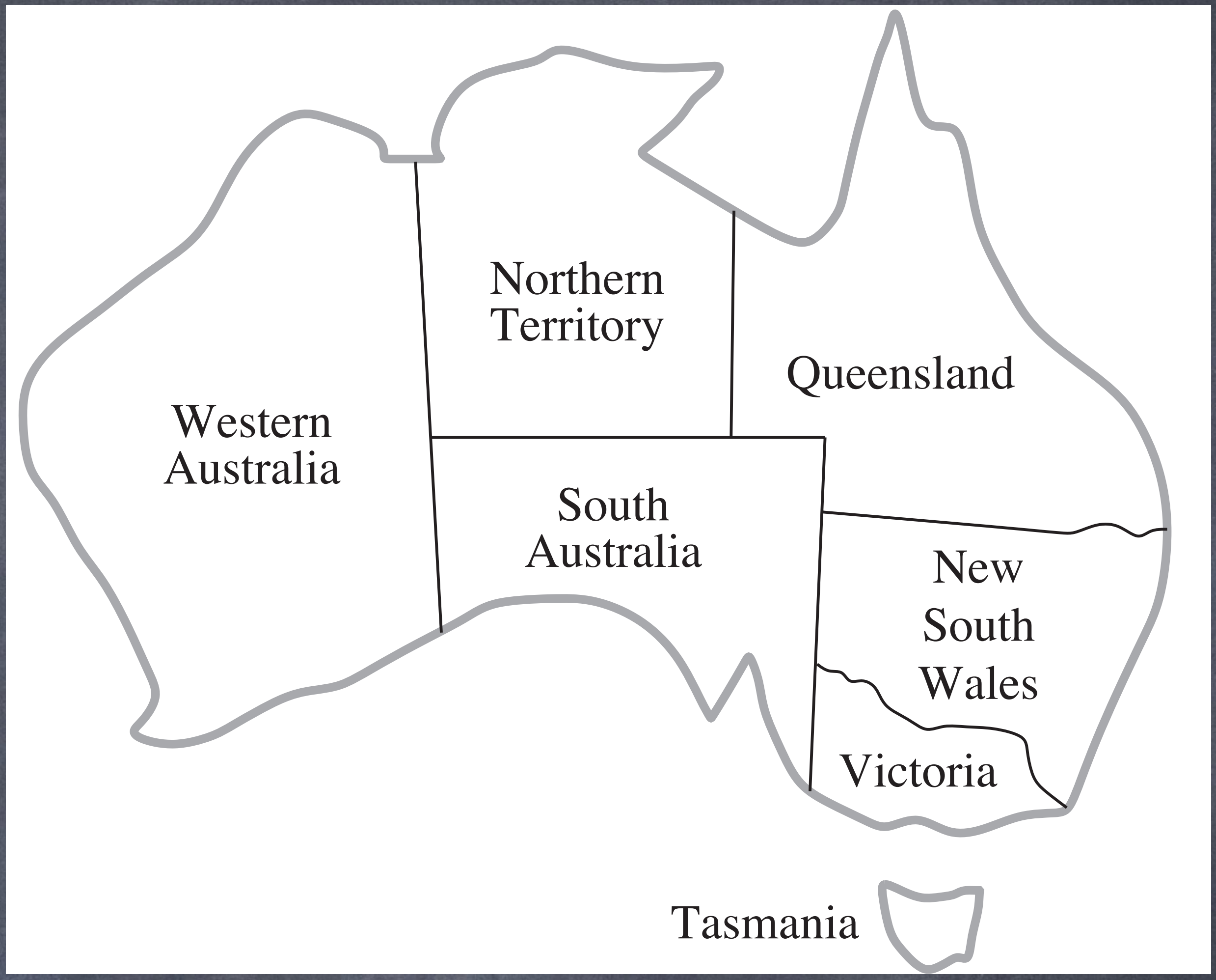
- DFS search through the space of assignments
- Assign one variable at a time
- Because the representation of CSPs is standardized, no need to supply initial state, actions, transition model, or goal test — nice!

But wait, there's more!

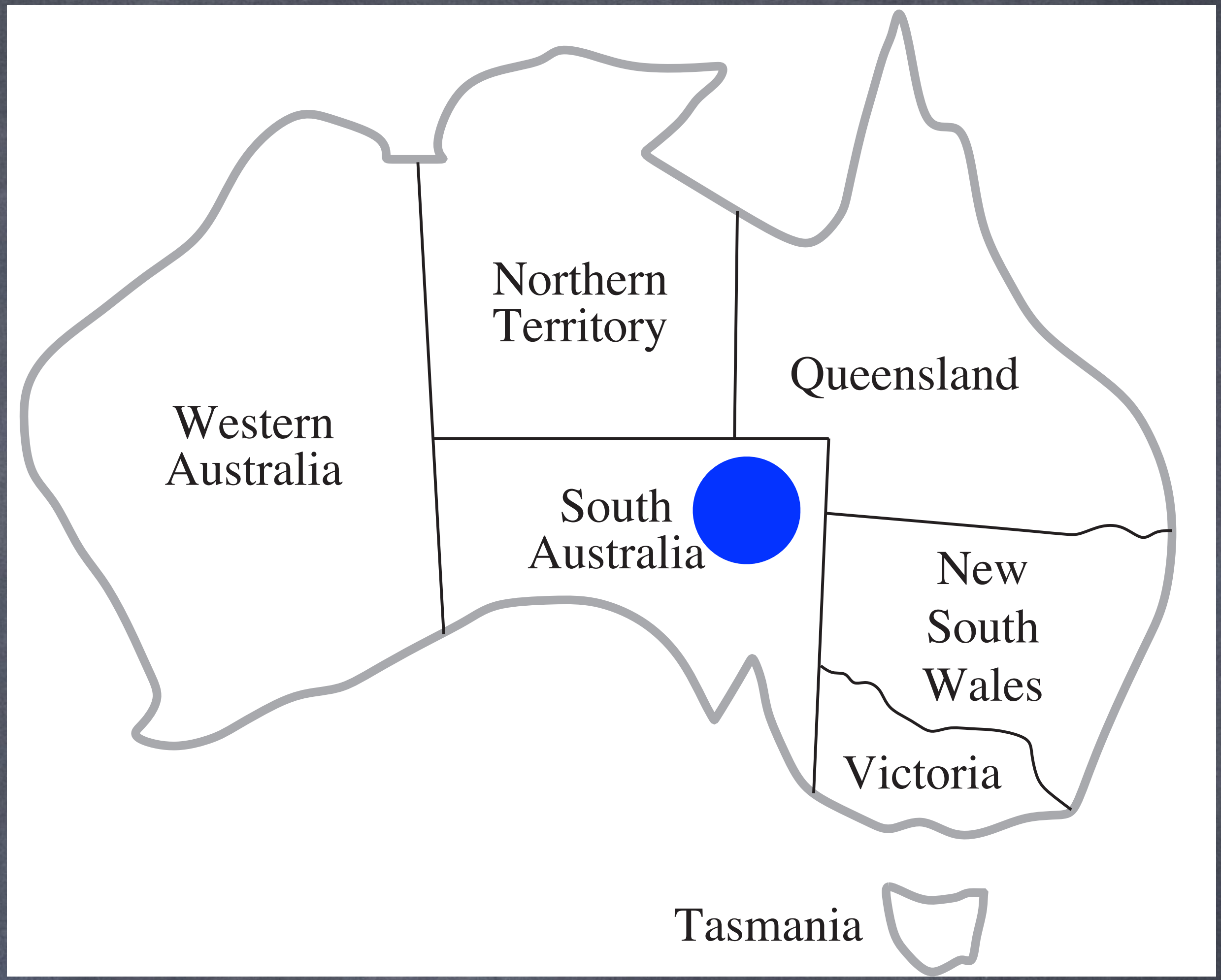
Heuristics for CSPs

- Minimum-remaining values (most constrained variable)
- Degree heuristic (variable involved in most constraints with unassigned variables)
- Least constraining value (if we only want to find one solution)

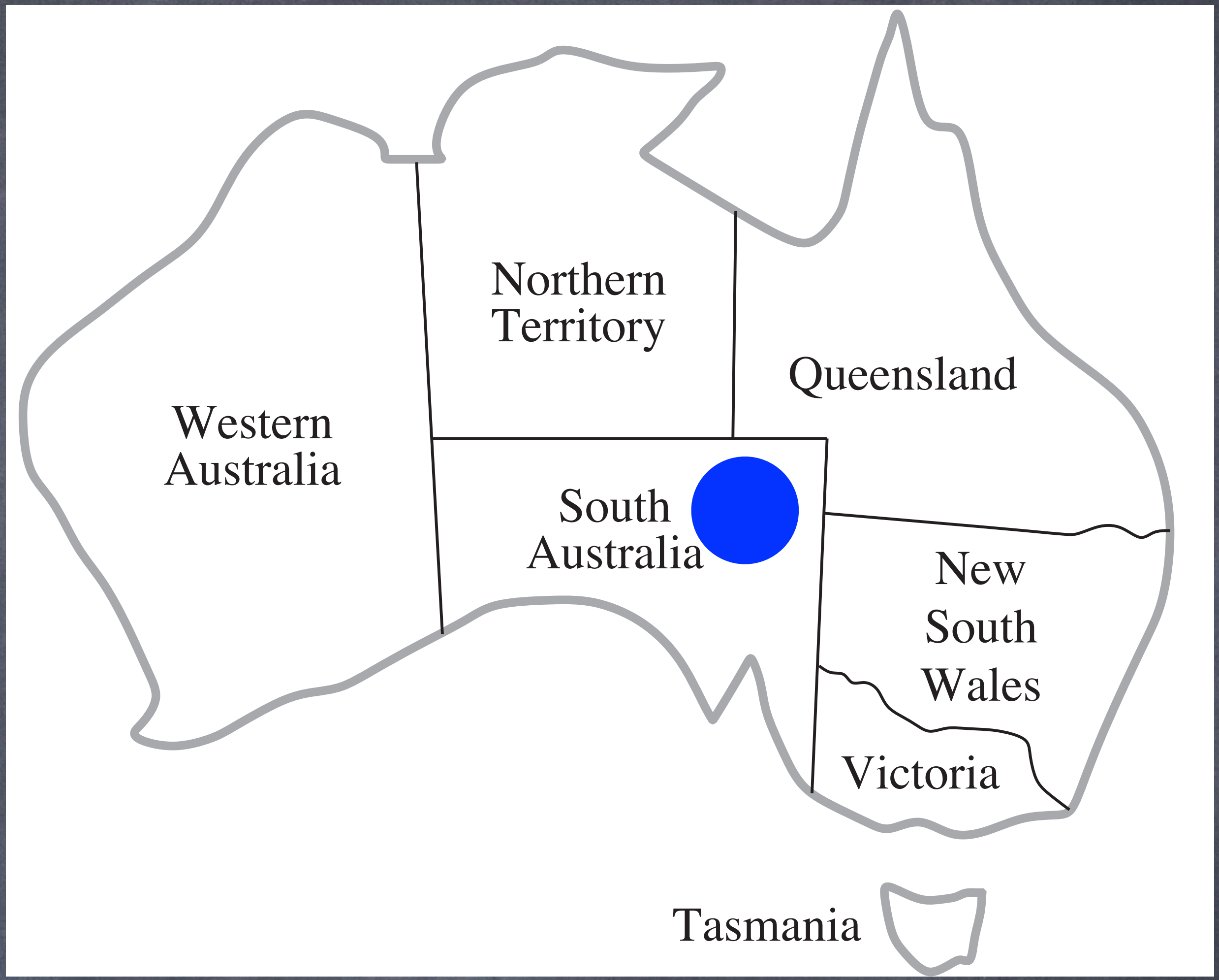
WA	R, G, B
NT	R, G, B
SA	R, G, B
Q	R, G, B
NSW	R, G, B
V	R, G, B
T	R, G, B



WA	R, G, B
NT	R, G, B
SA	B
Q	R, G, B
NSW	R, G, B
V	R, G, B
T	R, G, B

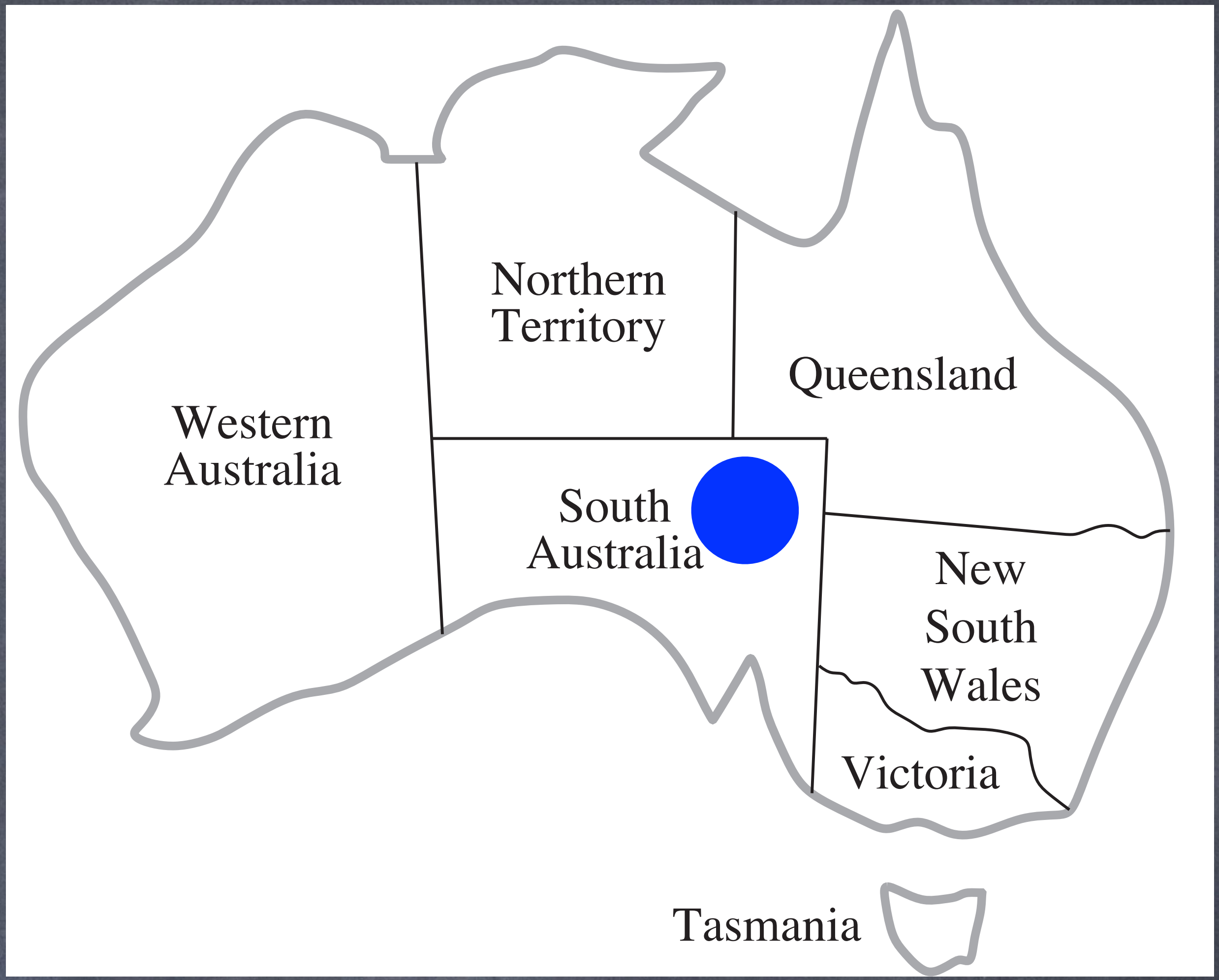


WA	R, G, B
NT	R, G, B
SA	B
Q	R, G, B
NSW	R, G, B
V	R, G, B
T	R, G, B



Remaining possibilities: $3^6 = 729$

WA	R, G
NT	R, G
SA	B
Q	R, G
NSW	R, G
V	R, G
T	R, G, B



Remaining possibilities: $3*2^5 = 96$

Constraint Propagation

- Using the constraints to reduce the set of legal values of a variable, which can in turn reduce the legal values of another variable, and so on
- Not a search process!
- A type of inference:
  making implicit information explicit

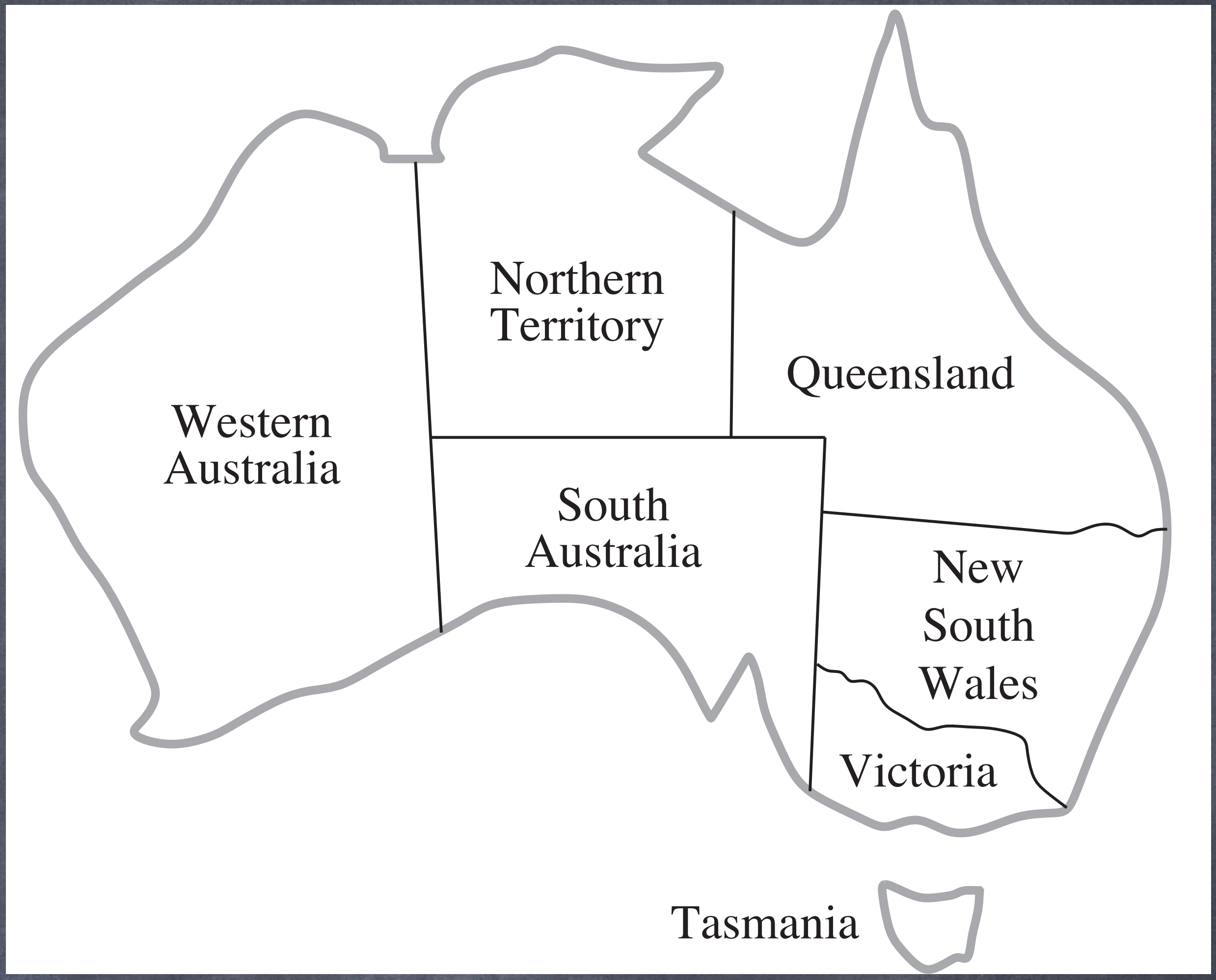
Constraint Propagation

- Good:
  - Can significantly reduce the space of assignments left to search
- (Possibly) Bad:
  - How long does it take to do the propagation?

Constraints

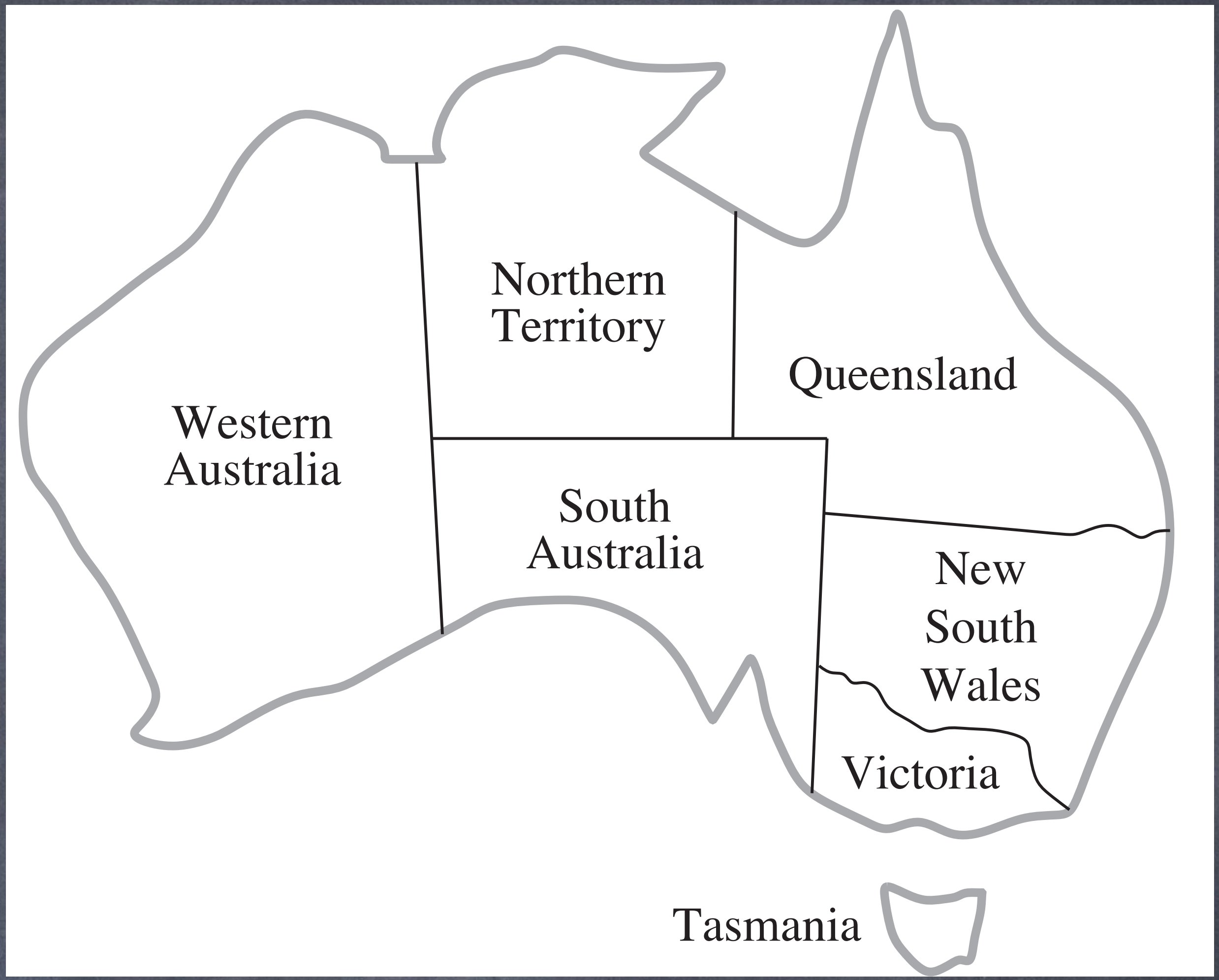
- Unary constraint: one variable
  - e.g., NSW ≠ red, X_i is even, $X_i = 2$
- Binary constraint: two variables
  - e.g., NSW ≠ WA, $X_i > X_j$, $X_i+X_j = 2$
- "Global" constraint: more than two vars
  - e.g., X_i is between X_j and X_k, AllDiff(X_i,X_j,X_k)
  - Can be reduced to set of binary constraints (possibly inefficiently)

WA	R, G, B
NT	R, G, B
SA	R, G, B
Q	R, G, B
NSW	R, G, B
V	R, G, B
T	R, G, B



SA ≠ green

WA	R, G, B
NT	R, G, B
SA	R, B
Q	R, G, B
NSW	R, G, B
V	R, G, B
T	R, G, B



SA ≠ green

Node Consistency

- Every possible value of every variable is consistent with the unary constraints

WA	R, G, B
NT	R, G, B
SA	
Q	R, G, B
NSW	R, G, B
V	R, G, B
T	R, G, B

INCONSISTENT

Constraints:

- SA ≠ green
- SA ≠ red
- SA ≠ blue

Inconsistency

- Empty domain for any variable
- No possible values for that variable
- No possible assignment including that variable
- No possible solution!

Node Consistency

- Apply all unary constraints
- If problem is not inconsistent, then we can always propagate unary constraints at the start
- And then we can ignore them
- Complexity: Each variable, each value, each unary constraint

Constraints

- Unary constraint: one variable
  - e.g., NSW $\neq$ red, X_i is even, $X_i = 2$
- Binary constraint: two variables
  - e.g., NSW $\neq$ WA, $X_i > X_j$, $X_i+X_j = 2$
- "Global" constraint: more than two vars
  - e.g., X_i is between X_j and X_k, AllDiff(X_i,X_j,X_k)
  - Can be reduced to set of binary constraints (possibly inefficiently)

Arc Consistency

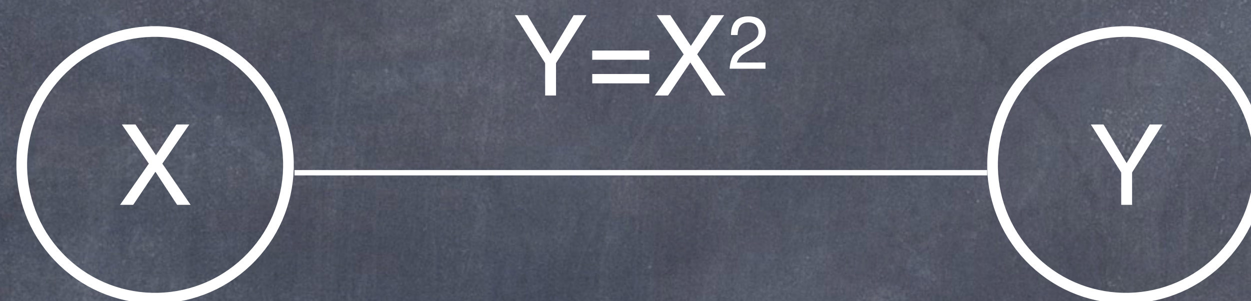
X_m is arc-consistent w.r.t. X_n if

for <u>every</u> value in the domain D_m,

there is <u>some</u> value in the domain D_n

that satisfies the binary constraint on the arc (X_m, X_n)

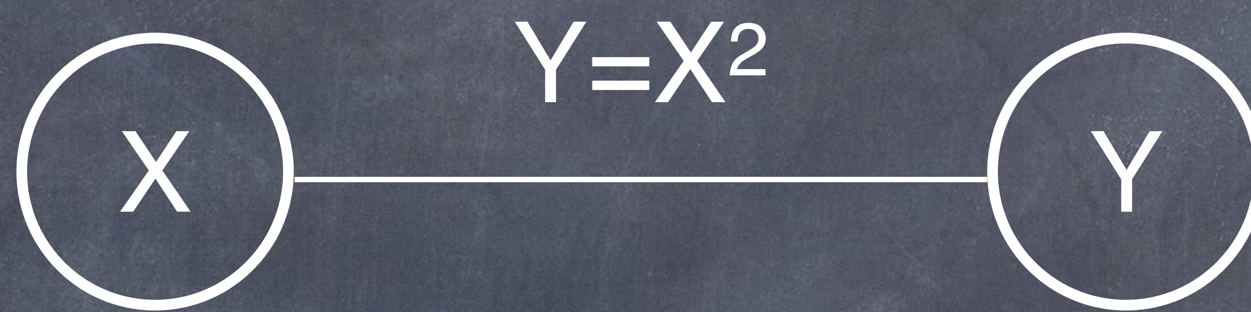
Arc Consistency



$\{ 0,1,2,3,4,5,6,7,8,9 \}$ $\{ 0,1,2,3,4,5,6,7,8,9 \}$

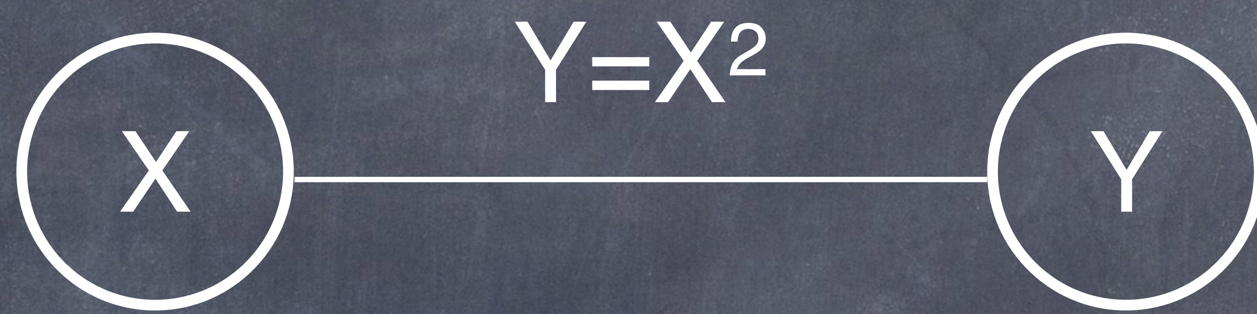
possible assignments: $10 \times 10 = 100$

Arc Consistency



{ 0,1,2,3,4,5,6,7,8,9 }   { 0,1,2,3,4,5,6,7,8,9 }

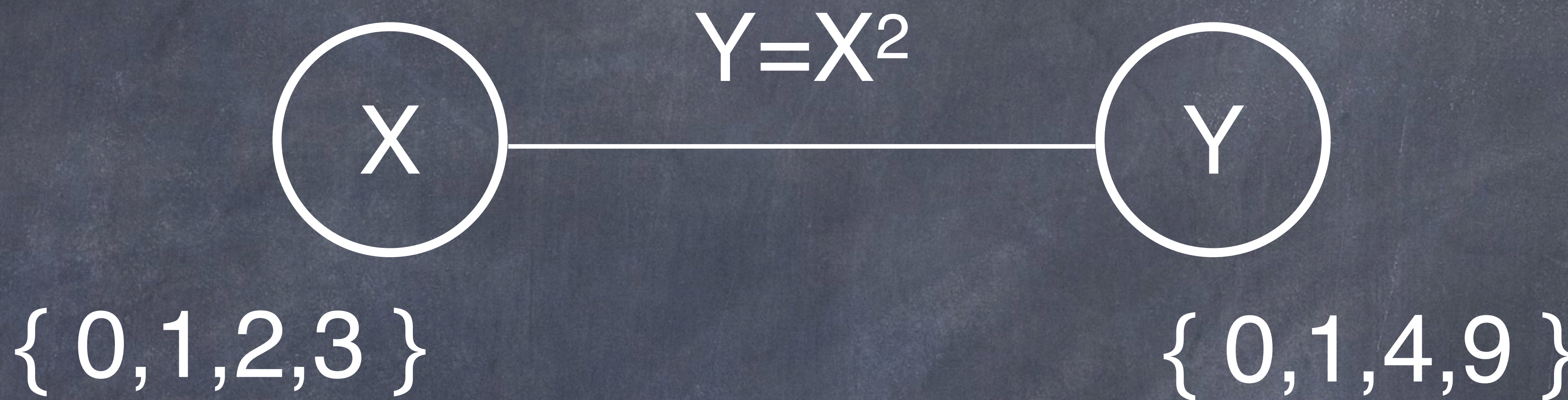
Arc Consistency



$\{0,1,2,3\}$ $\{0,1,2,3,4,5,6,7,8,9\}$

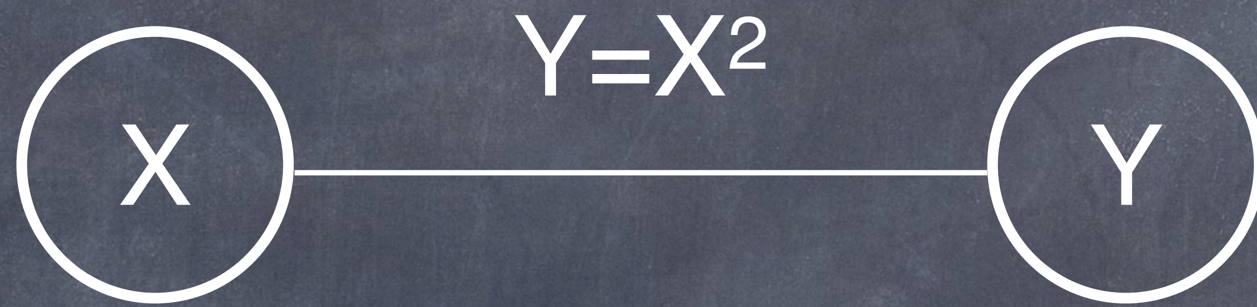
X arc-consistent with respect to Y

Arc Consistency



Y arc-consistent with respect to X

Arc Consistency



$\{0,1,2,3\}$ $\{0,1,4,9\}$

possible assignments: $4 \times 4 = 16$

AC-3

```
boolean AC3(csp) {
  Set queue = all arcs in csp
  while (queue is not empty) {
    <i,j> = queue.removeFirst()
    if (revise(csp, i, j)) {
      if Di is empty {
        return false
      }
      foreach k in neighbors(i) {
        add <k,i> to queue
      }
    }
  }
  return true
}
```

```
boolean revise(csp, i, j) {
  boolean changed = false
  foreach vi in Di {
    boolean ok = false
    foreach vj in Dj {
      if (<vi,vj> satisfies Cij )
        ok = true
      }
    }
    if (!ok) {
      delete vi from Di
      changed = true
    }
  }
  return changed
}
```

AC-3 Analysis

- CSP with n variables, domain size $\leq d$, c constraints (arcs)
- Each arc can be inserted in the queue at most d times.
- Checking a single arc takes $O(d^2)$ time
- Total time: $O(cd^3)$
  - Independent of n

More Constraint Propagation

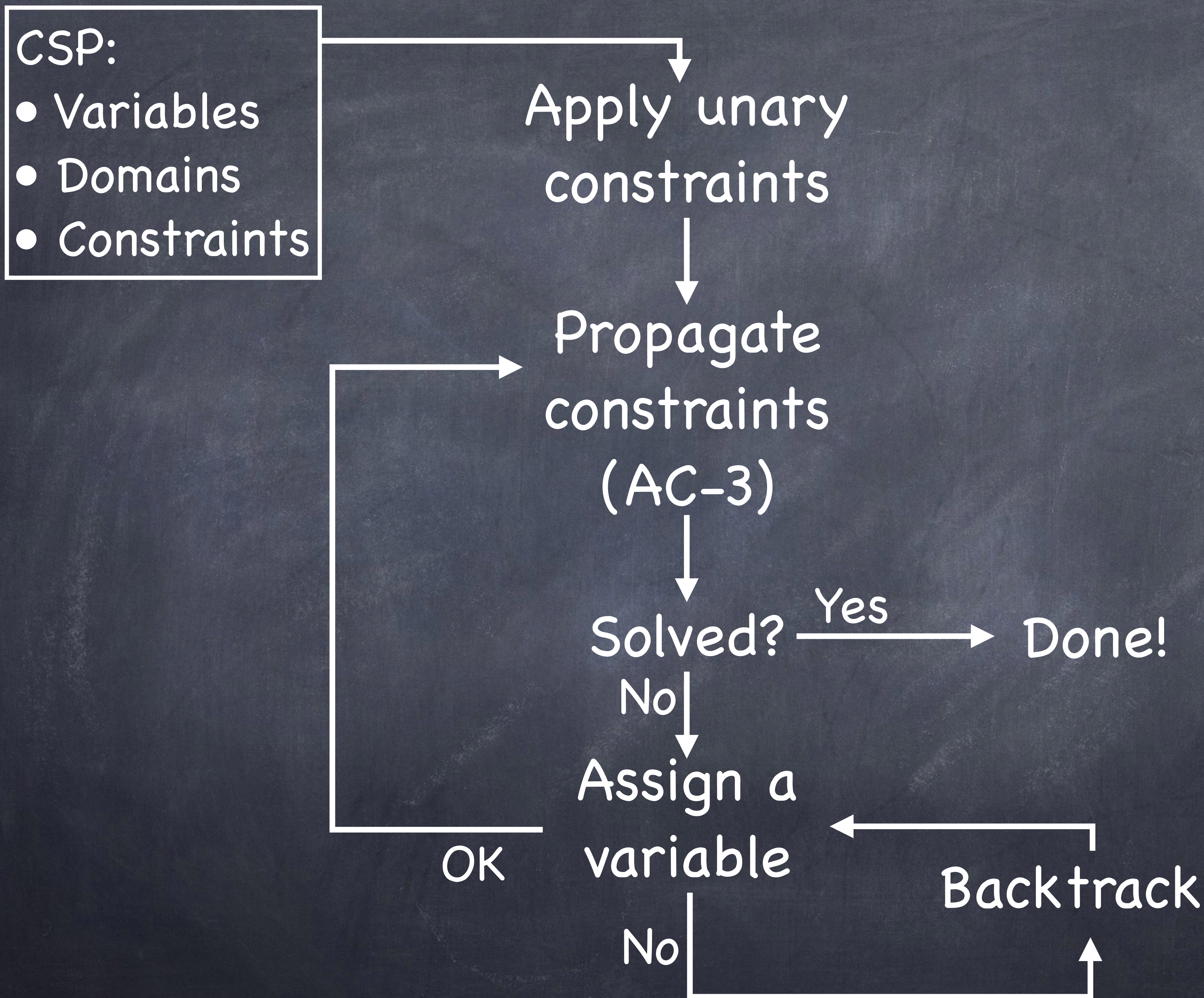
- Path consistency
- k-consistency
  - Generalization of node (1-), arc (2-), and path (3-) consistency
  - Establishing k-consistency is exponential in k
  - Typically use node-and arc-consistency and rarely path-consistency

Constraint Propagation

- Bottom line: "After constraint propagation, we are left with a CSP that is equivalent to the original CSP—they both have the same solutions—but the new CSP will in most cases be faster to search because its variables have smaller domains."

Interleaving Search and Inference

- After each choice during search, we can perform inference to reduce future search


CSP:
• Variables
• Domains
• Constraints
Apply unary constraints
Propagate constraints (AC-3)
Solved?
Yes
Done!
No
Assign a variable
OK
No
Backtrack

CSP Secret Sauce

- Factored representation of state:
  - Variables, Domains, Constraints
- Allows:
  - Early pruning of inconsistent states
  - Inference during search to reduce alternatives

Constraint Satisfaction

- Impose a structure on the representation of states: Variables, Domains, Constraints
- Backtracking (DFS) search for complete, consistent assignment of values to variables
- Inference (constraint propagation) can reduce the domains of variables
  - Preprocessing and/or interleaved with search
- Useful problem-independent heuristics

For next time:

AIMA 7.0 - 7.4

Slide credits:
Prof George Ferguson
(used with permission)