CSC242: Introduction to Artificial Intelligence

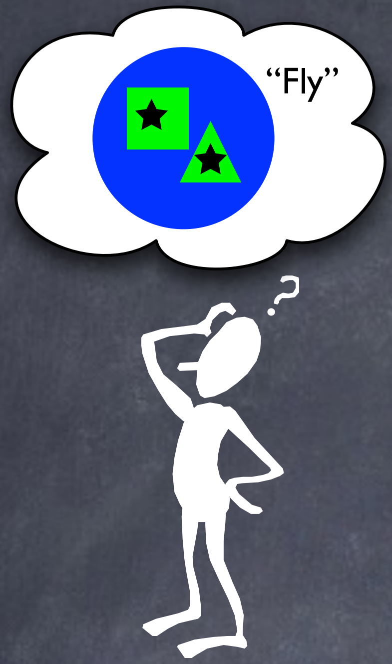
Prof. George Ferguson
Office: CSB731
Email: george.ferguson@rochester.edu

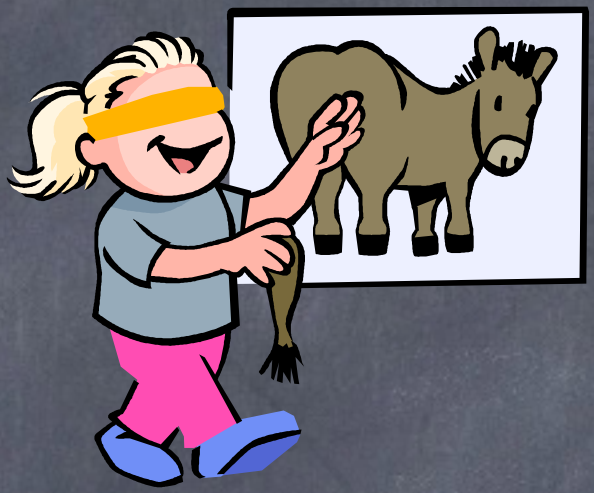
Lecture 9

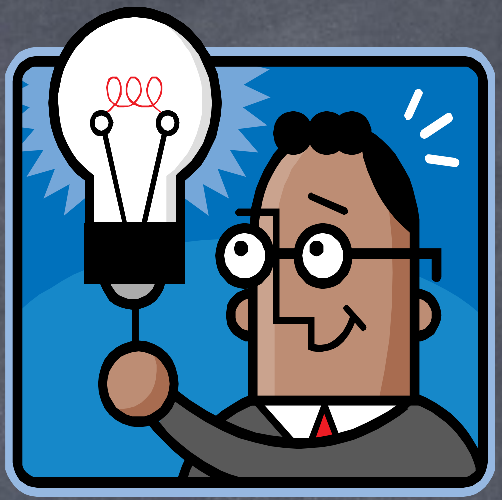
Announcements

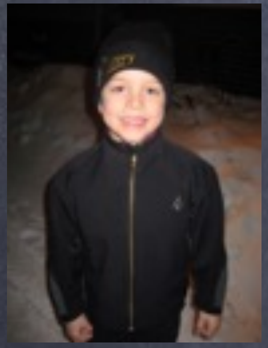
- Project 1 due Mon 15 Feb 1159PM
  - DO NOT WAIT TO GET STARTED
- No office hours Mon 15 Feb

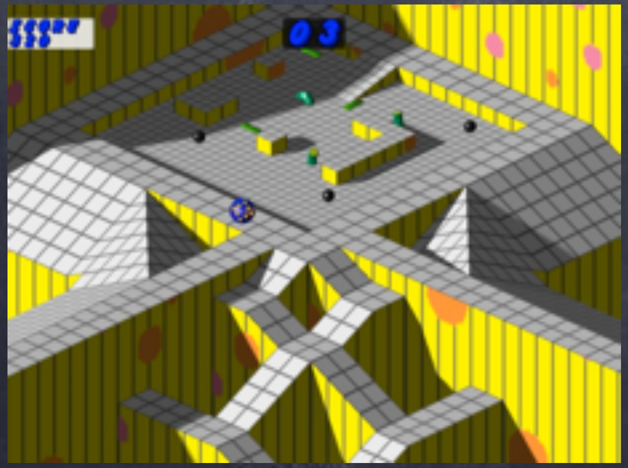
Constraint Satisfaction

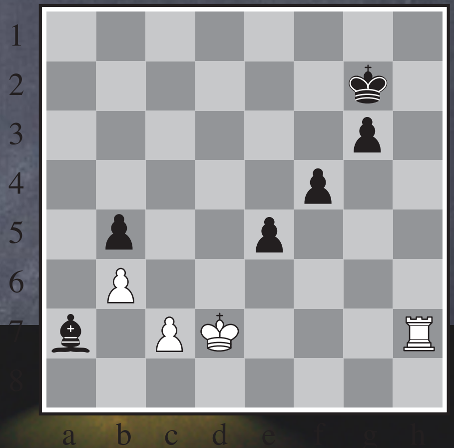

"Fly"

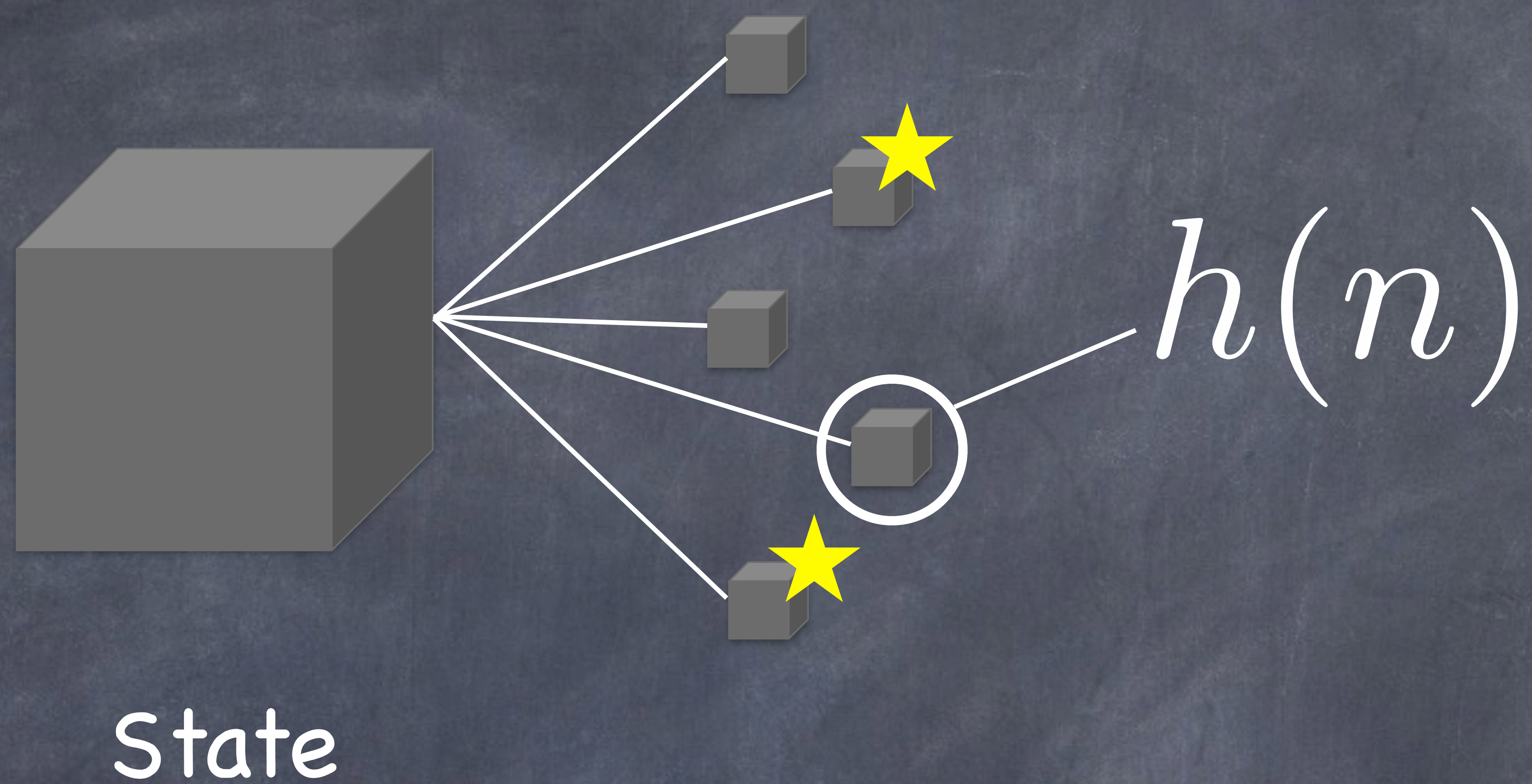









1
2
3
4
5
6
a
b
c
d



$h(n)$
State

```java
public class Board {
  protected Player[][] grid;
  public Board(int n) {
    grid = new Player[n][n];
  }
  ...
}

public class State {
    protected Board board;
    protected Player nextPlayer;
    ...
}
```

```java
public Player getWinner() {
  Player p;
  p = checkHorizontals();
  if (p != null) {
    return p;
  }
  p = checkVerticals();
  if (p != null) {
    return p;
  }
  p = checkDiagonals();
  if (p != null) {
    return p;
  }
  return null;
}

protected Player checkHorizontals() {
  for (int y=0; y < size; y++) {
    Player p = checkHorizontal(y);
    if (p != null) {
```

The Problem With States

- State representation is specific to a given problem (or domain of problems)
- Functions on states (successor generation, goal test) are specific to the state representation
- Heuristic functions are both problem-specific and dependent on the state representation
- Many design choices, many opportunities for coding errors

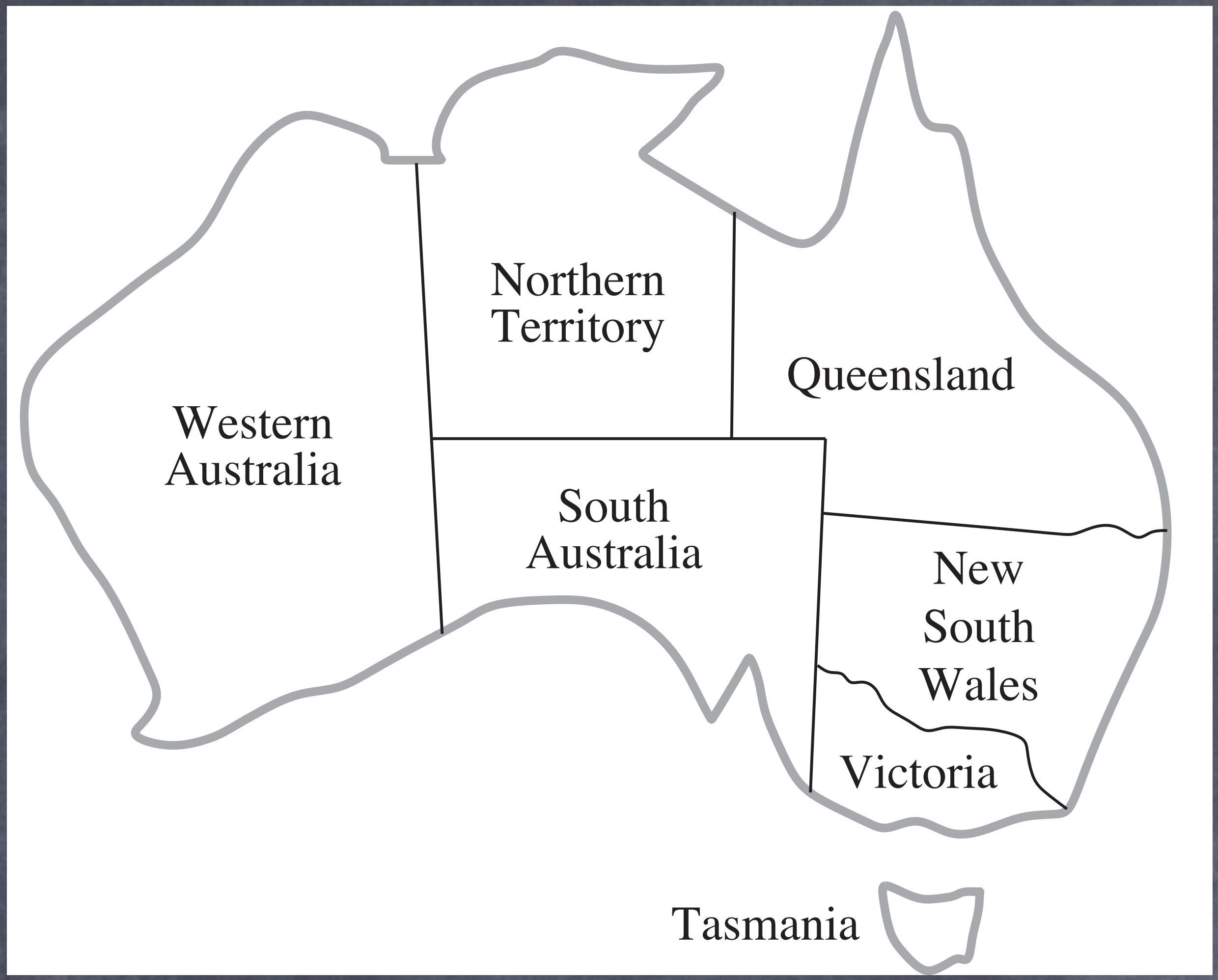
Approach

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

Bottom Line

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

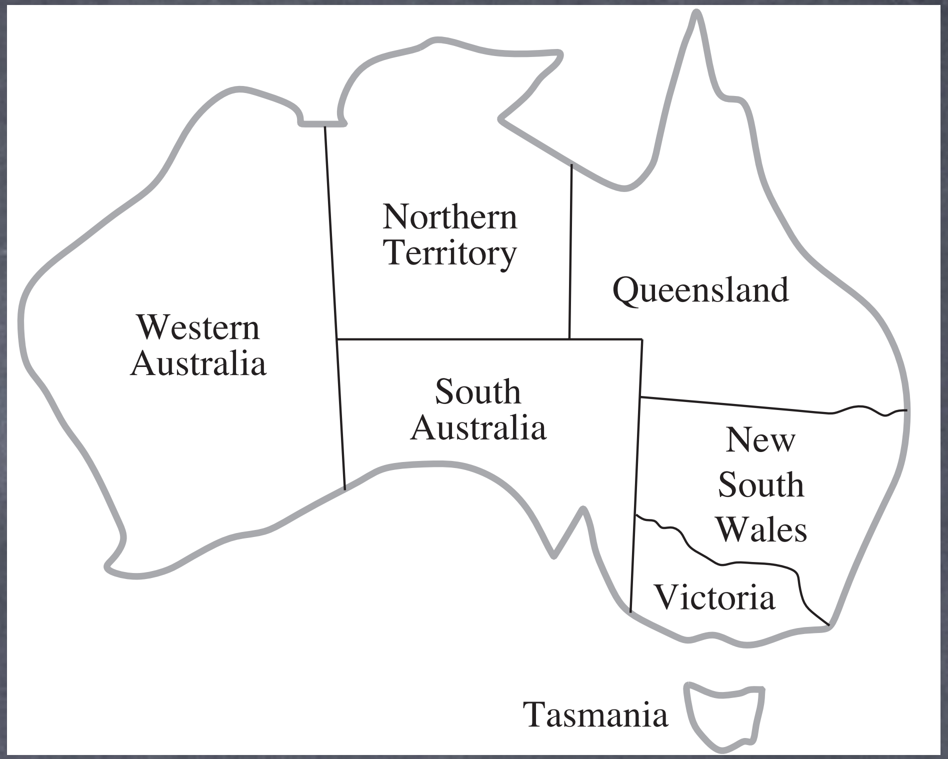
Example



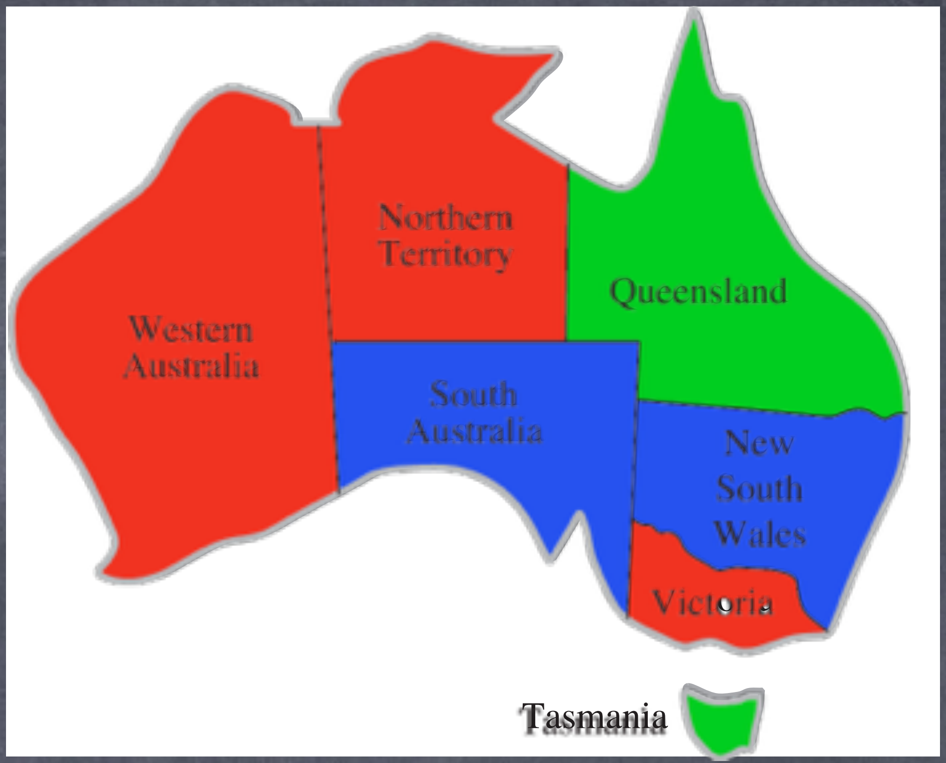
Assign a color to each region such that no two neighboring regions have the same color

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

enum Color = red, green, blue



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



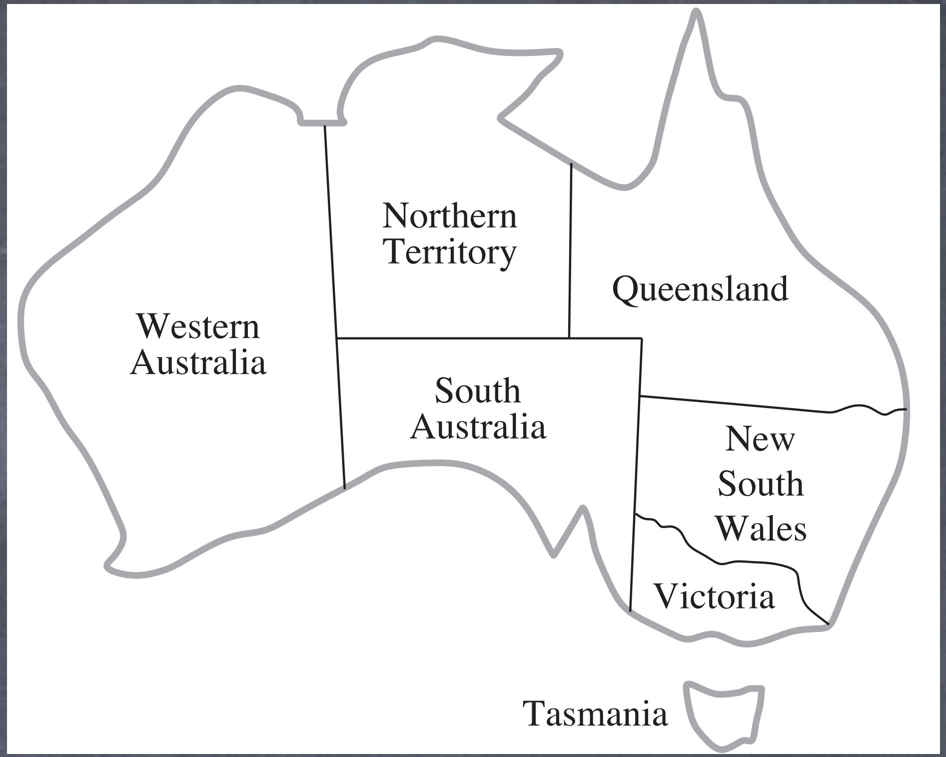
enum Color = red, green, blue

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

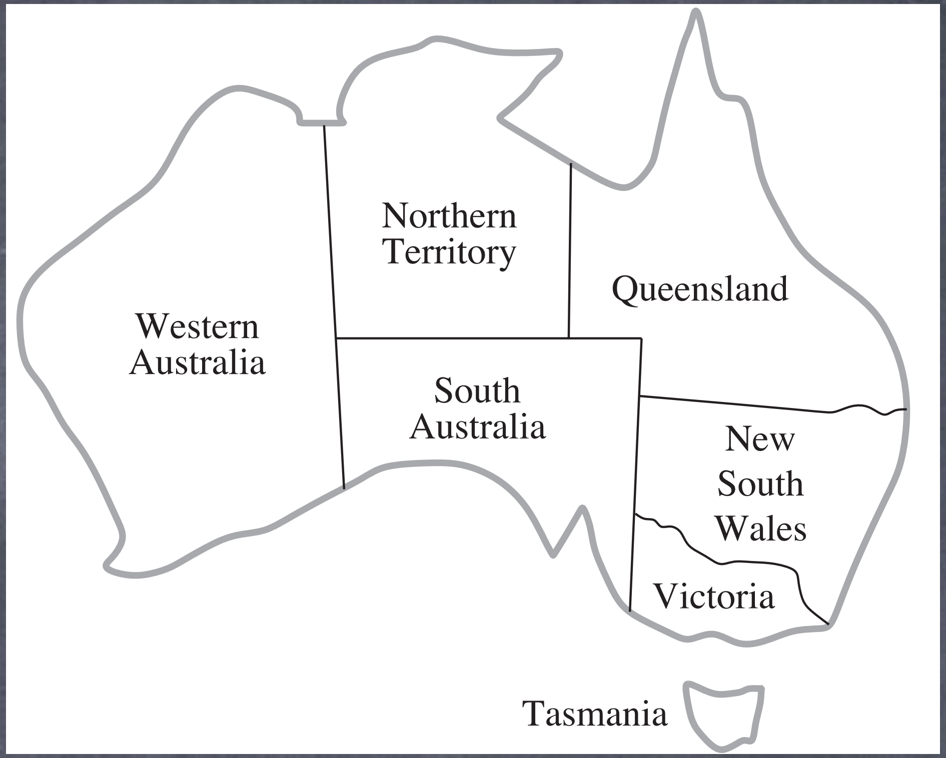
Action: pick an unassigned region and assign it a color

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

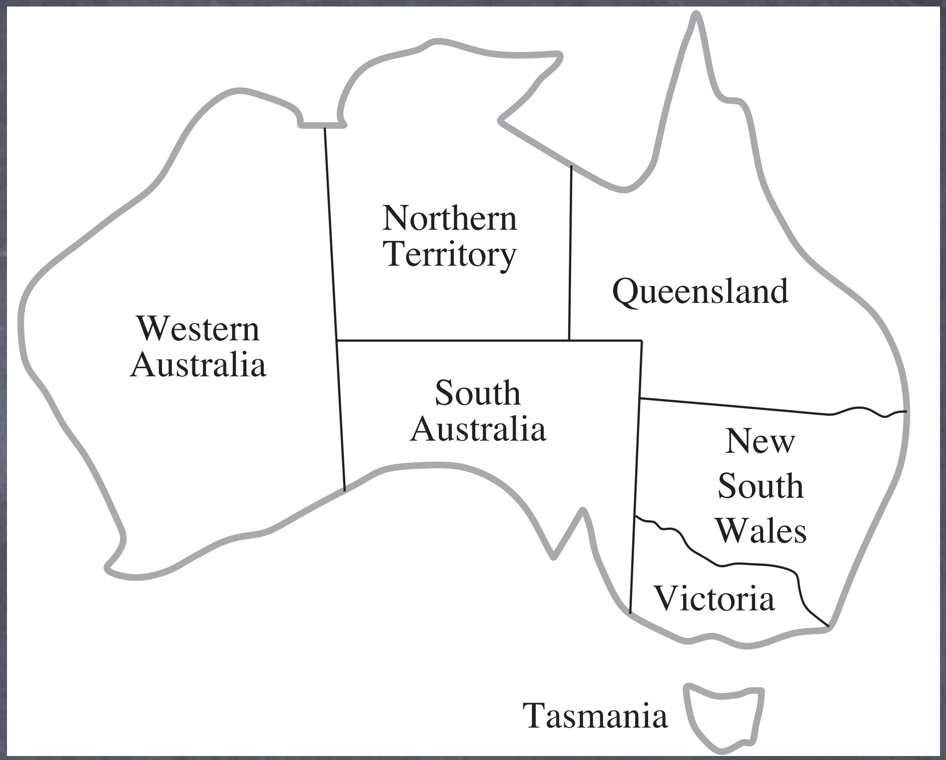
State: assignment of colors to regions

Successor function: pick an unassigned region and assign it a color

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

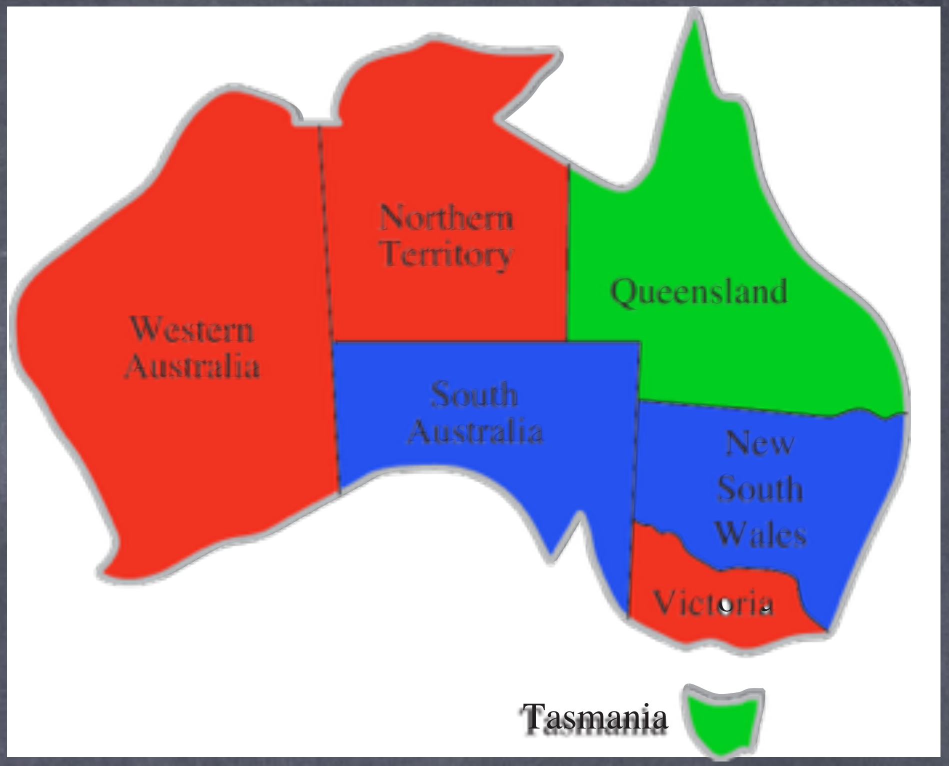
Color WA, NT, Q, NSW, V, SA, T

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

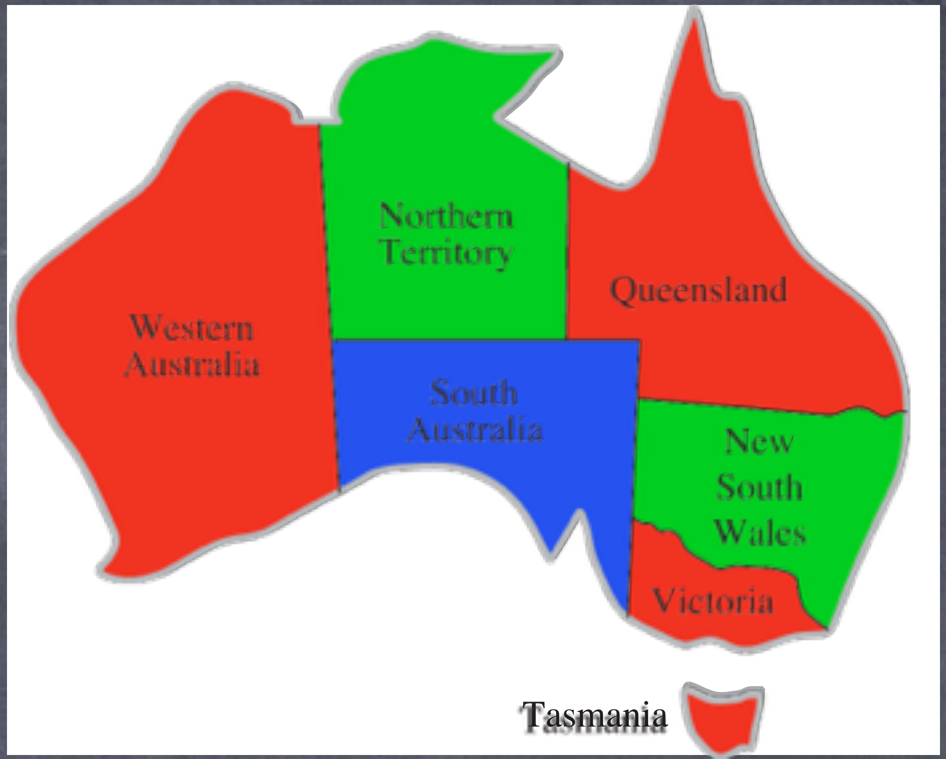


enum Color = red, green, blue

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

V=red, SA=blue, T=green

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

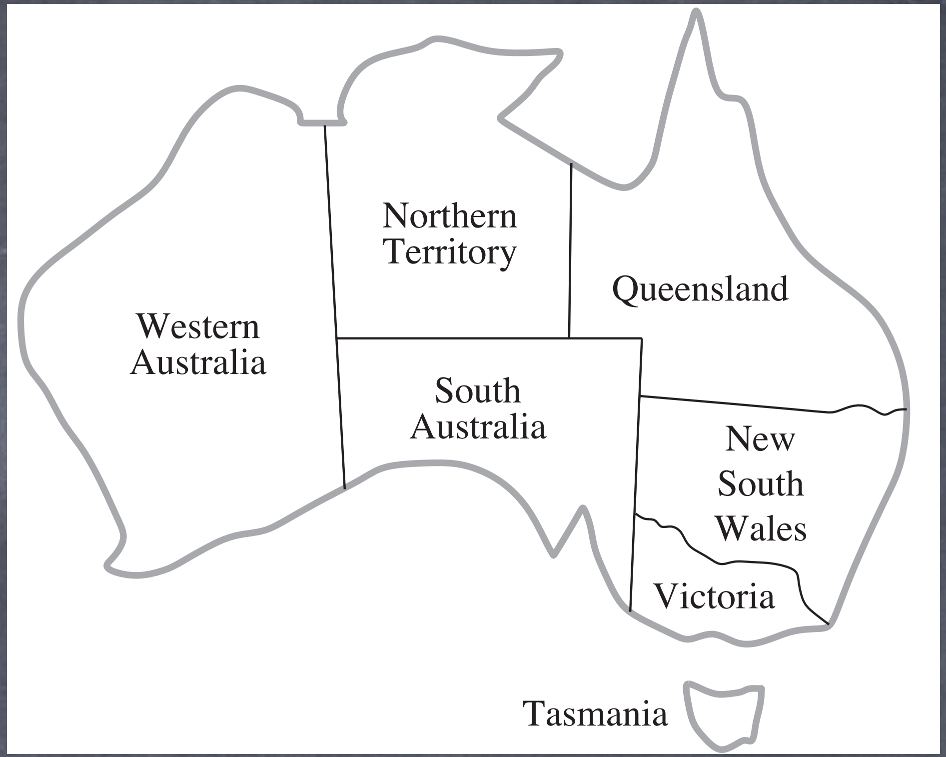


enum Color = red, green, blue

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

V=red, SA=blue, T=red

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



enum Color = red, green, blue

State: assignment of colors to regions

Successor function: pick an unassigned region and assign it a color

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

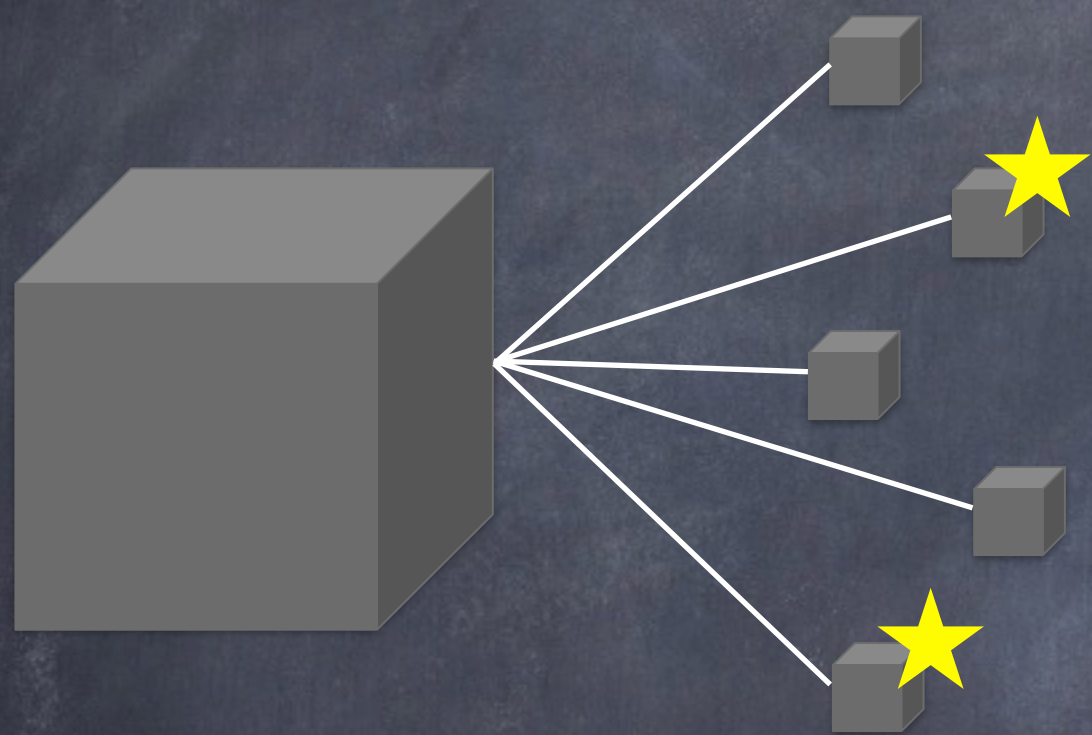
Constraint Satisfaction Problem (CSP)

- 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 $\{ C_1, ..., C_m \}$

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 value of some attribute)

State Representation



Atomic

$X_1 = v_1$
$X_2 = v_2$
$X_3 = v_3$

$X_1 = v_1$
$X_2 = v_3$
$X_3 = v_2$

$X_1 = v_1$
$X_2 = v_3$
$X_3 = ?$

Factored

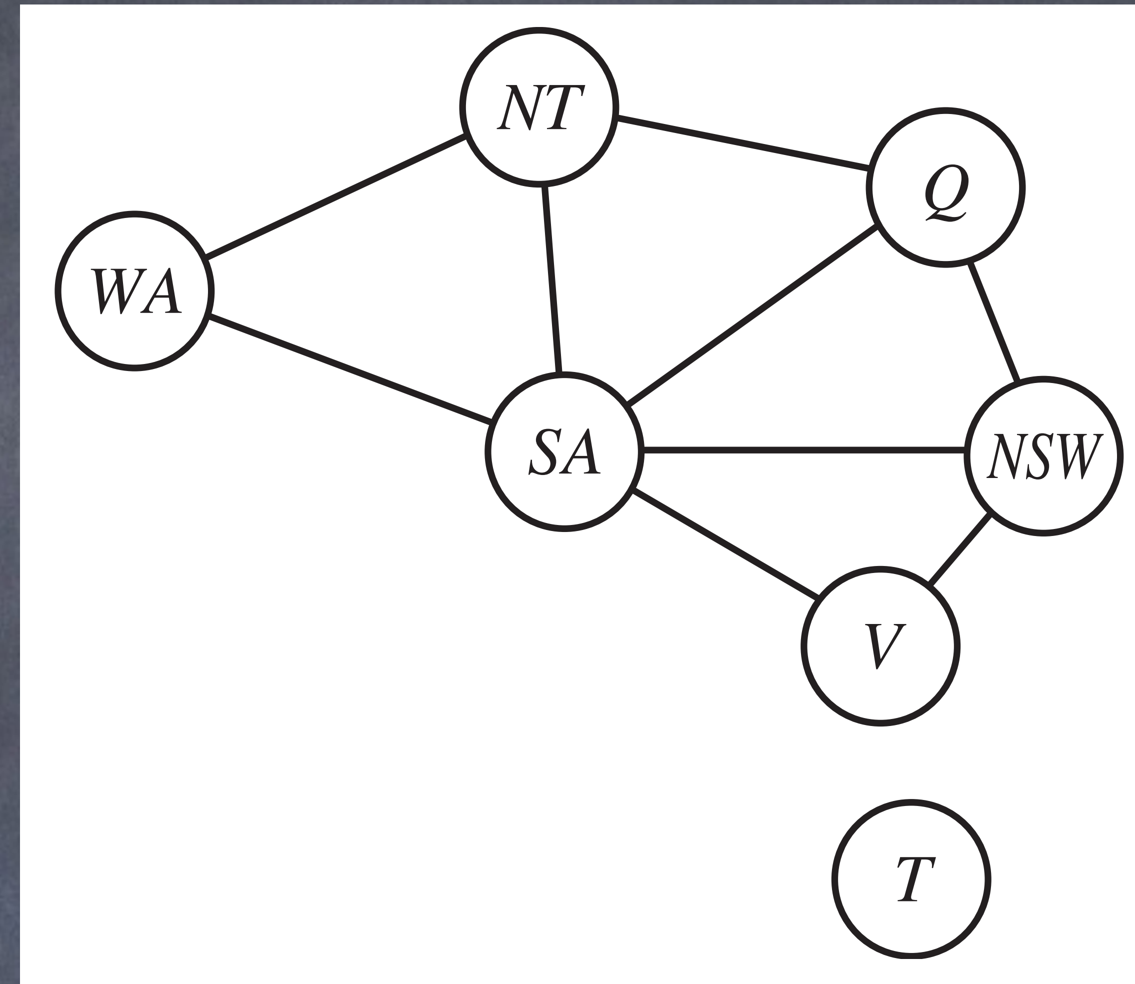
Australia Map CSP

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

More CSP Terminology

- Assignment: $\{ X_i = v_i, X_j = v_j, \ldots \}$
- 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

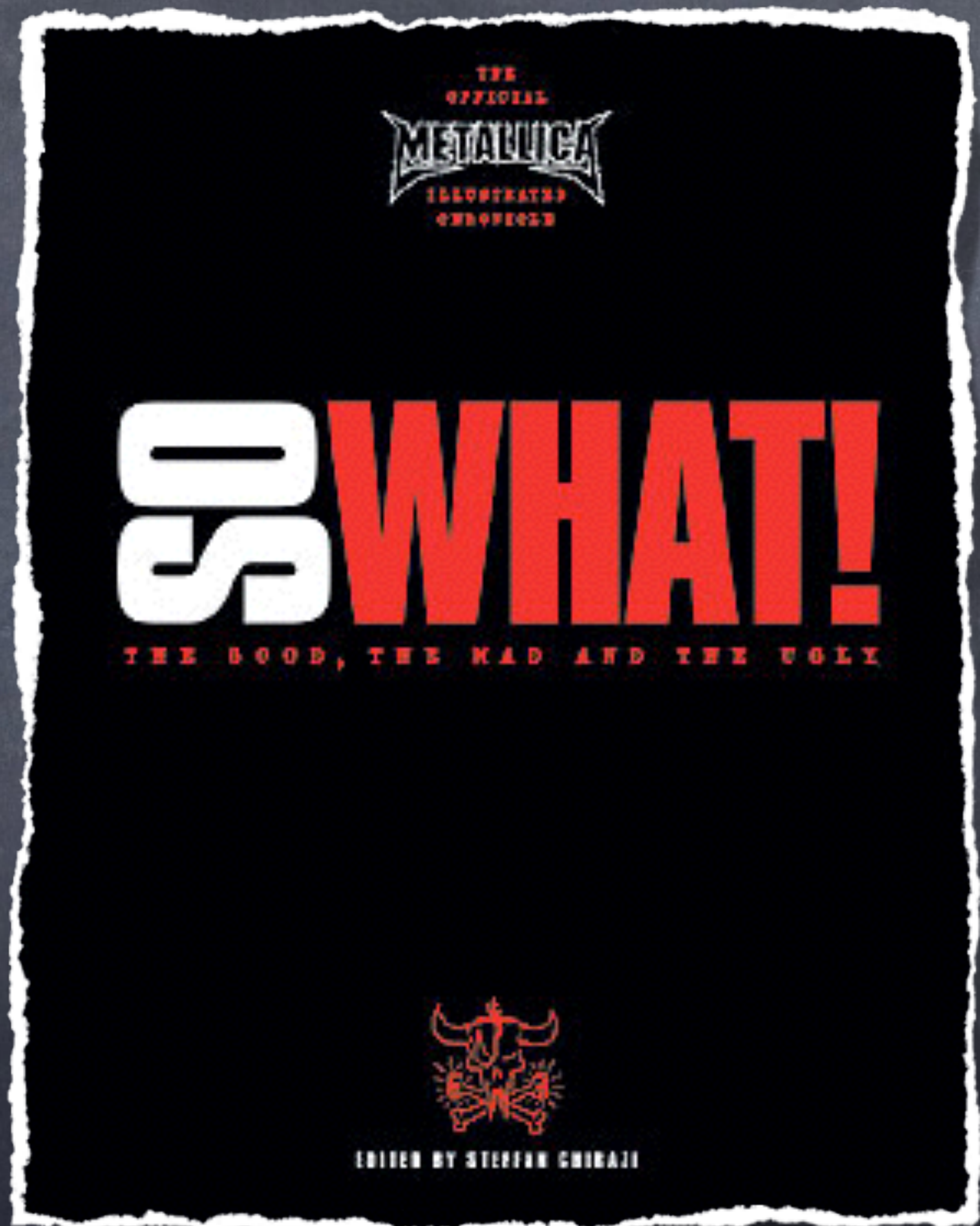

NT
Q
WA
SA
NSW
V
T

Constraints

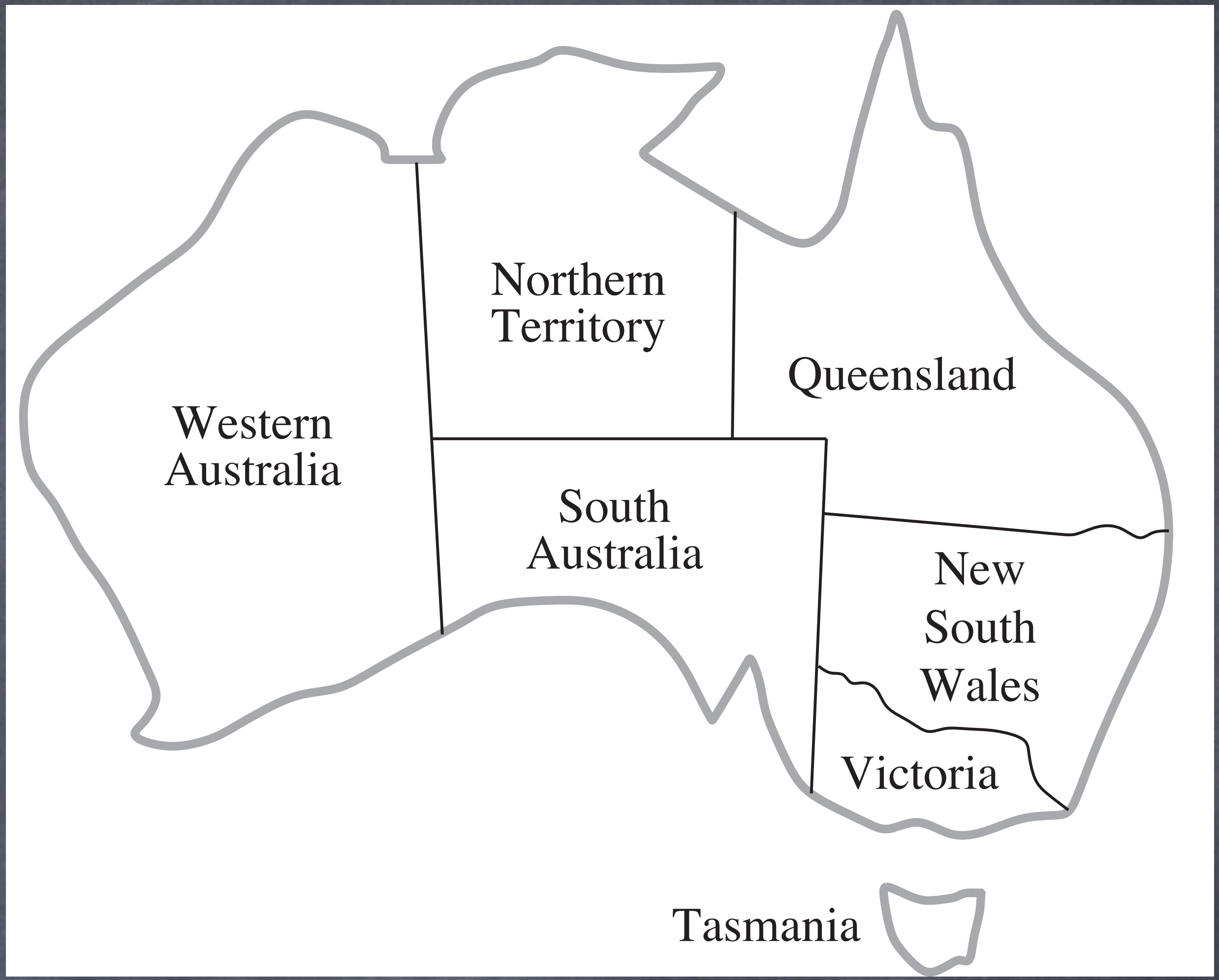
- Unary constraint: one variable
  - e.g., $NSW \neq red$, X_i is even, $X_i = 2$
- Binary constraint: two variables
  - e.g., $NSW \neq WA$, $X_i > X_j$, $X_i + X_j = 2$
- "Global" constraint: more than two vars
  - e.g., X_i is between X_j and X_k, $AllDiff(X_i, X_j, X_k)$
  - Can be reduced to set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

Constraint Satisfaction Problem (CSP)

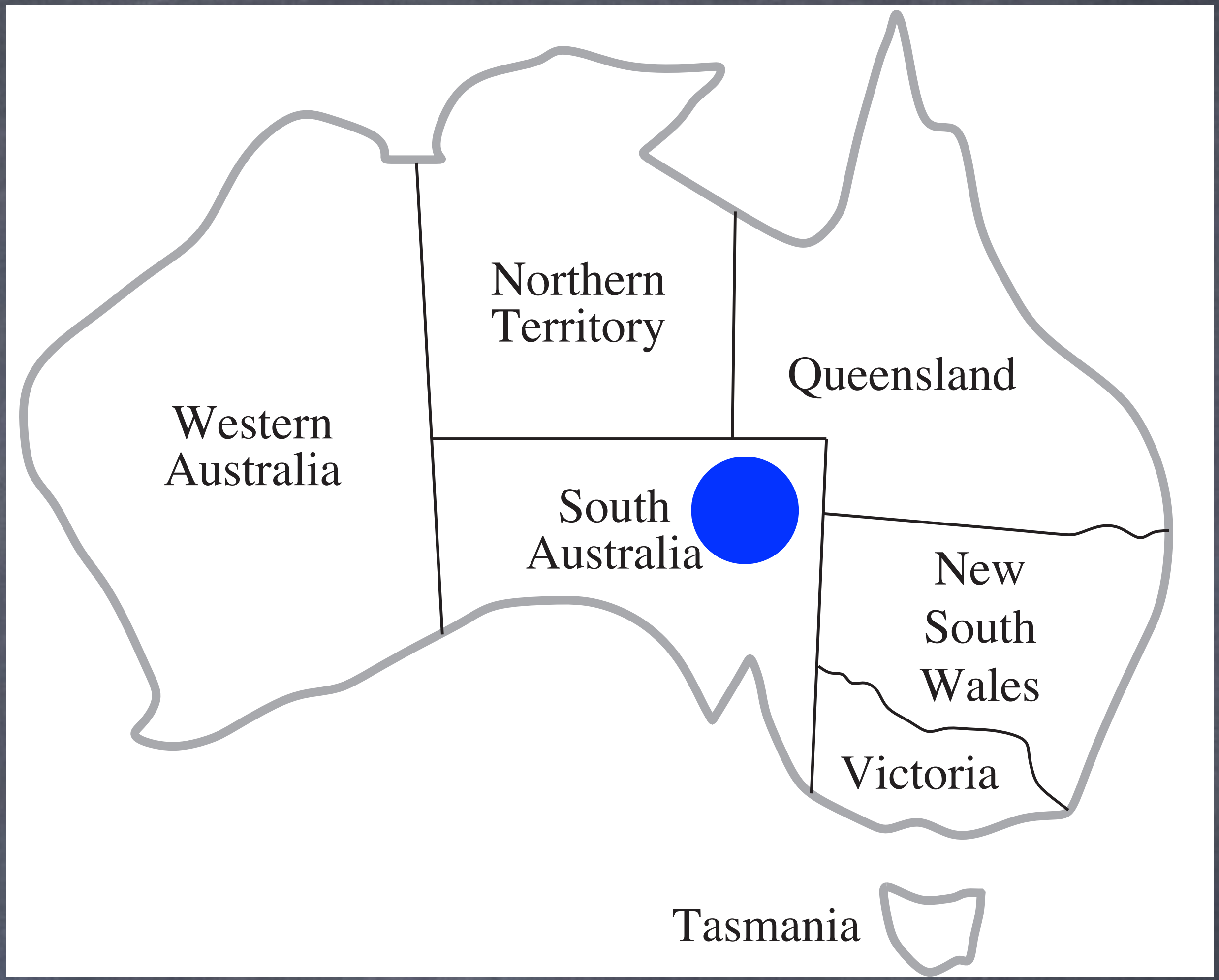
- X: Set of variables $\{ X_1, ..., X_n \}$
- D: Set of domains $\{ D_1, ..., D_n \}$
- Each D_i : set of values $\{ v_1, ..., v_k \}$
- C: Set of constraints $\{ C_1, ..., C_m \}$


THE OFFICIAL
METALLICA
ILLUSTRATED CHRONICLE
SO WHAT!
THE GOOD, THE MAD AND THE UGLY
EDITED BY STEFFAN CHIRAZI

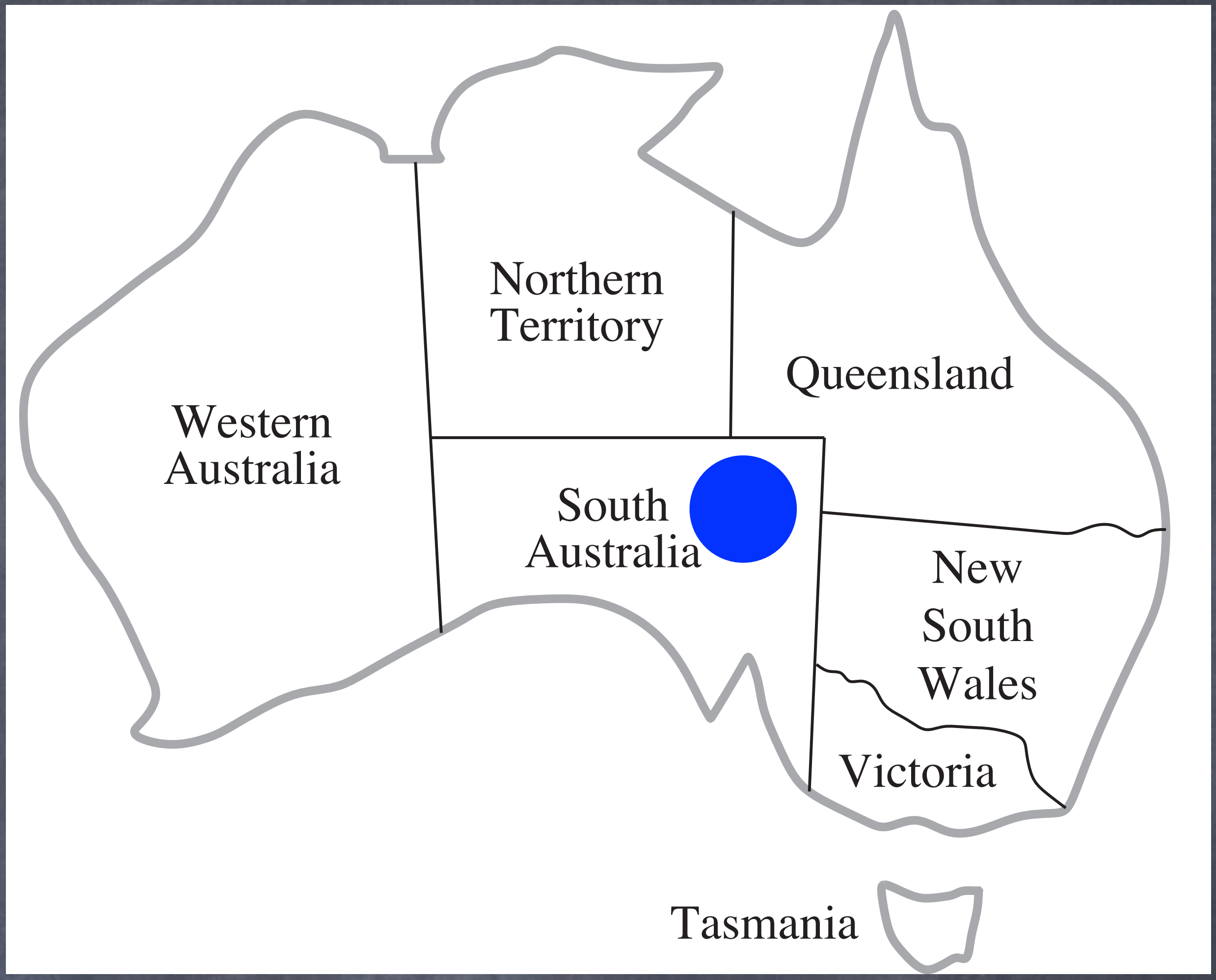
WA	R, G, B
NT	R, G, B
SA	R, G, B
Q	R, G, B
NSW	R, G, B
V	R, G, B
T	R, G, B



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

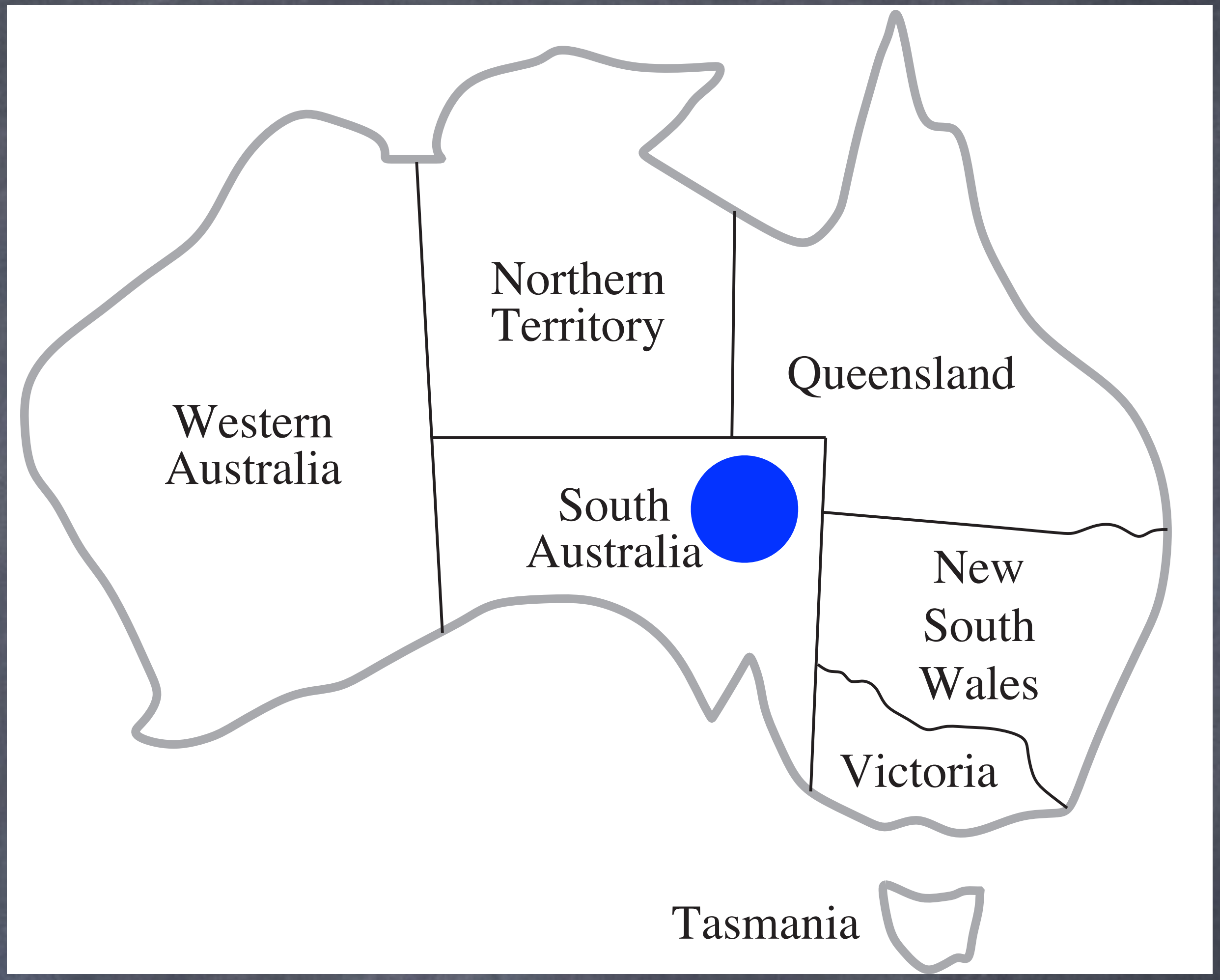


WA	R, G, B
NT	R, G, B
SA	B
Q	R, G, B
NSW	R, G, B
V	R, G, B
T	R, G, B



Remaining possibilities: $3^5 = 243$

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



Remaining possibilities: $2^5 = 32$

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!
- Part of state update in state-space search
- A type of <u>inference</u>: making implicit information explicit

Constraint Propagation

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

Constraint
Propagation
(inference)

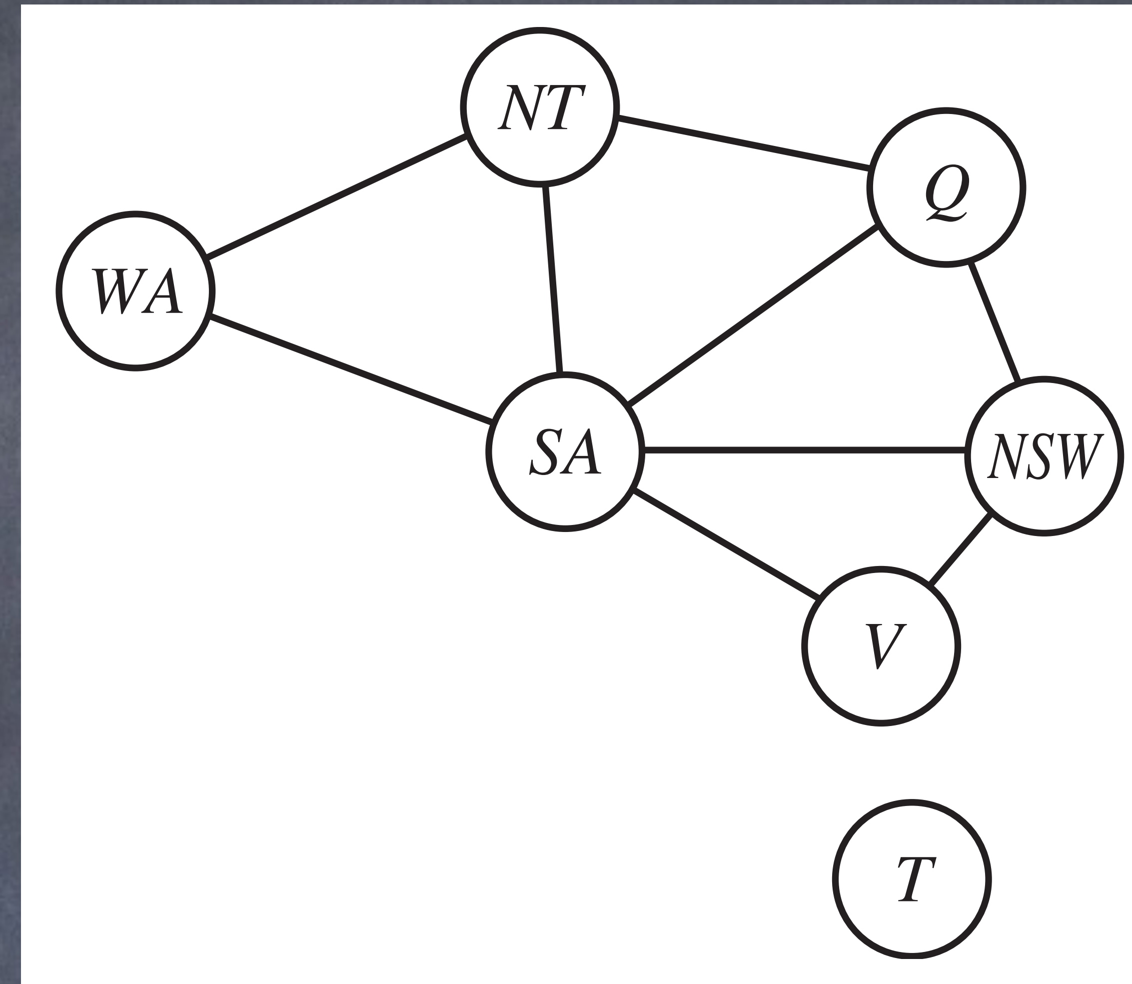
↔

State-Space
Search

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)

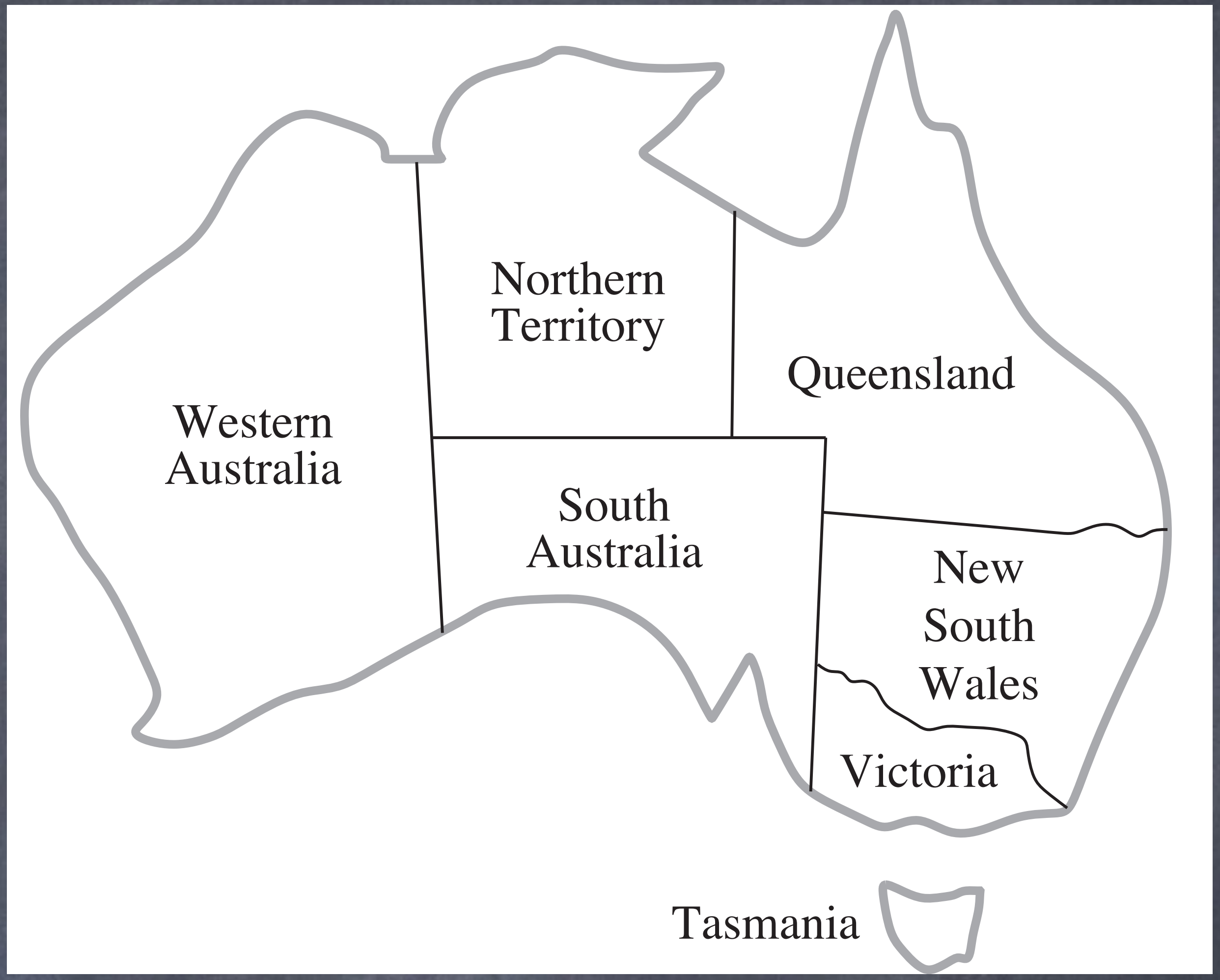

Northern Territory
Western Australia
Queensland
South Australia
New South Wales
Victoria
Tasmania


NT
Q
WA
SA
NSW
V
T

Constraint Propagation

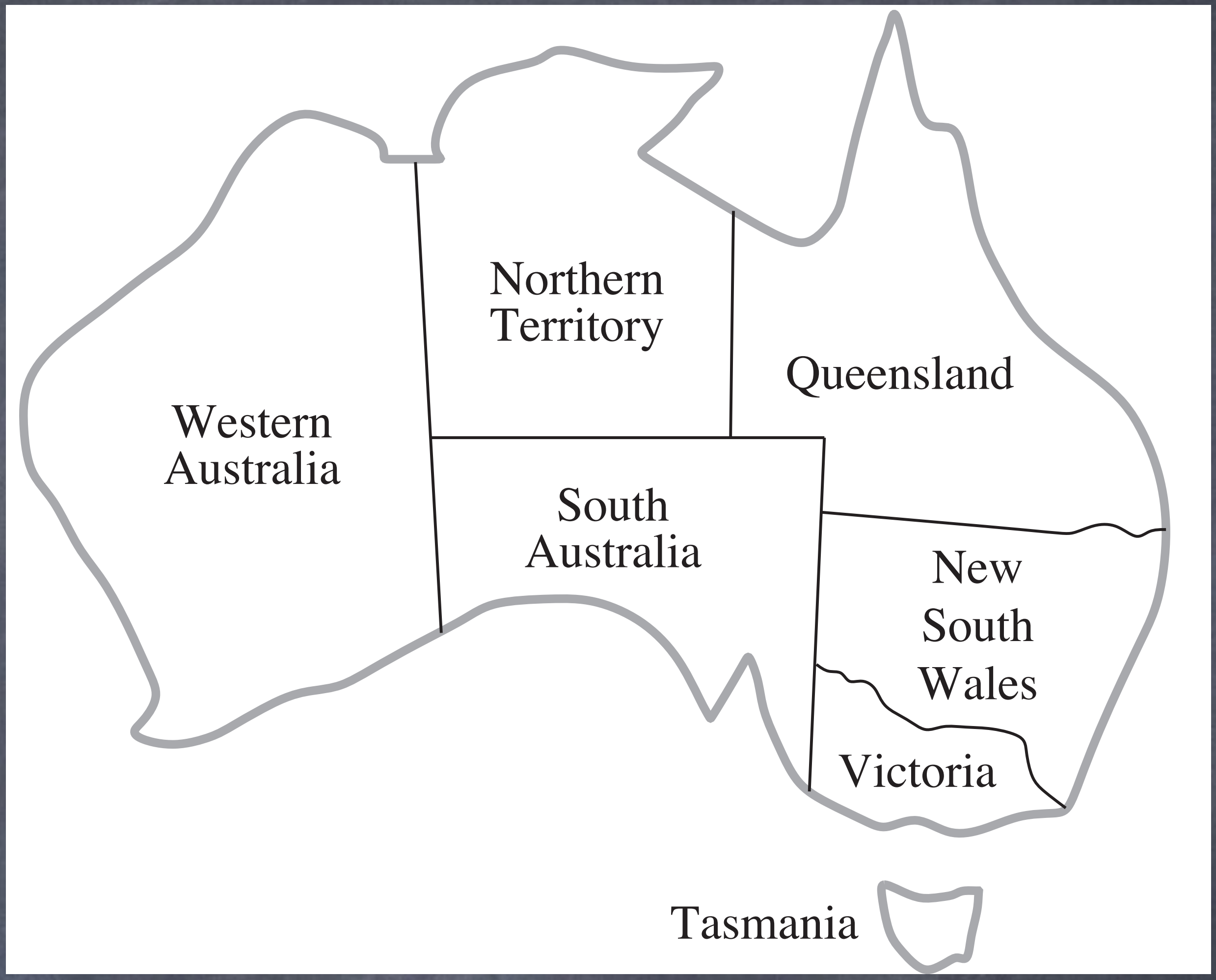
- Apply all unary constraints

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

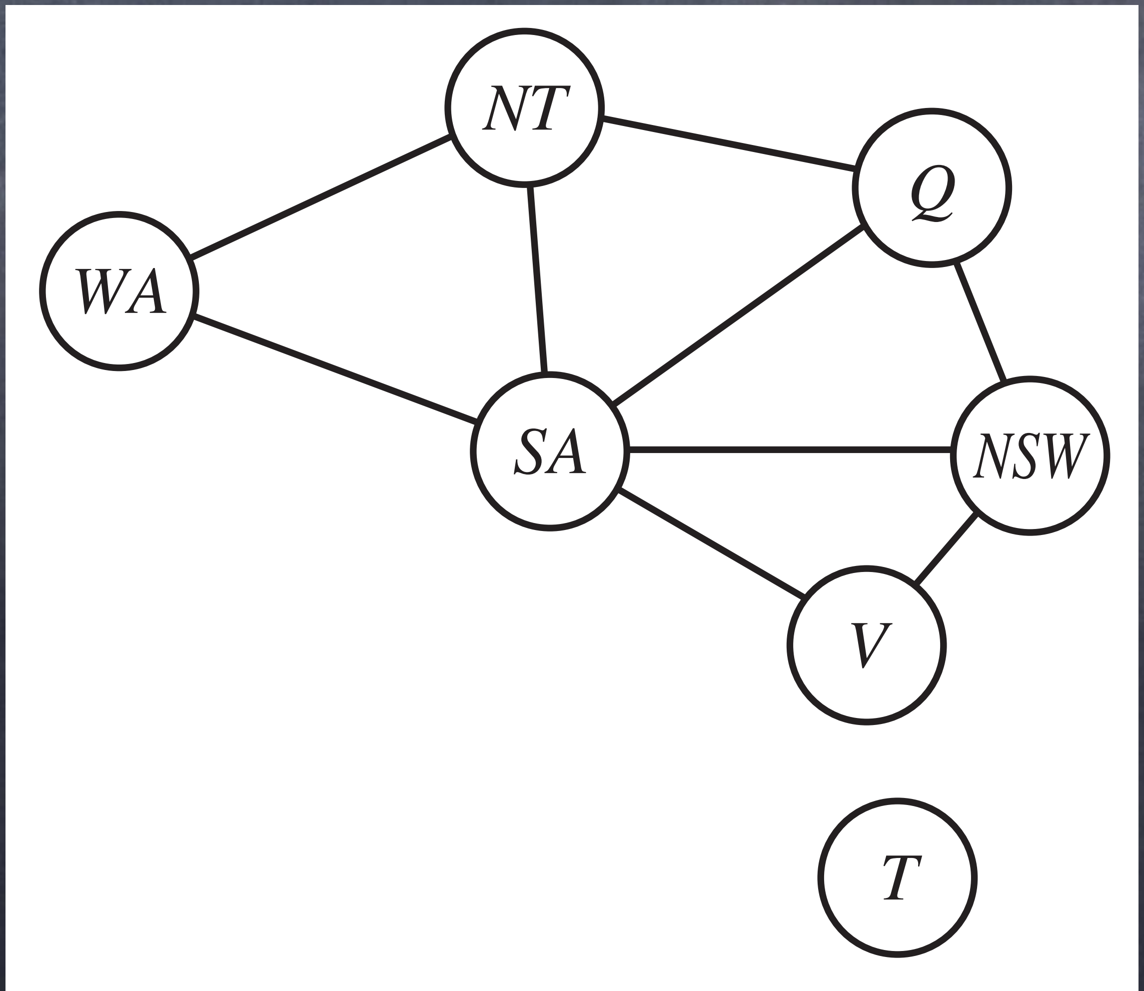
WA	R, G, B
NT	R, G, B
SA	R, B
Q	R, G, B
NSW	R, G, B
V	R, G, B
T	R, G, B



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

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

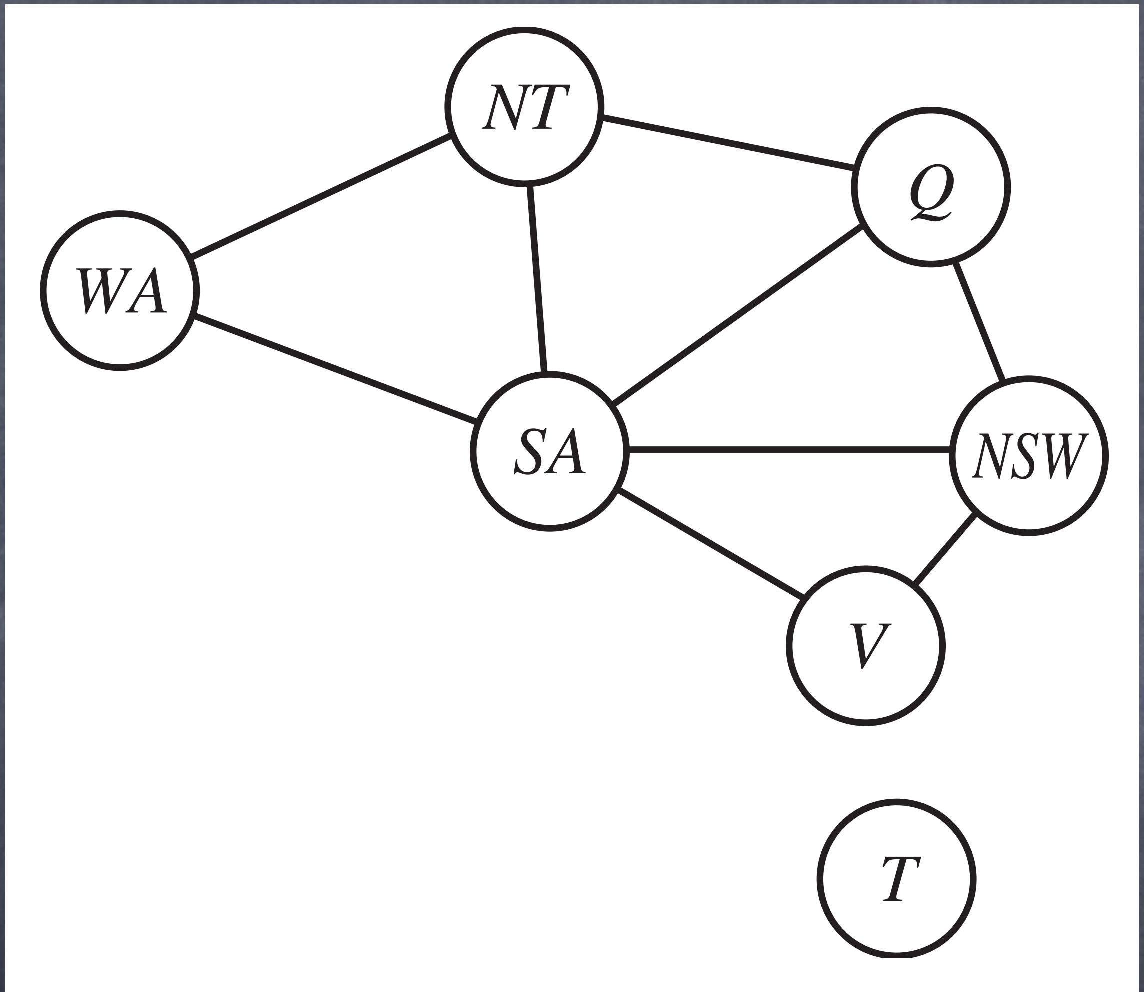
- Apply all unary constraints
- If problem is not inconsistent, then we can always propagate unary constraints at the start
- And then we can ignore them

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
  - But only do it once at the start

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)


NT
Q
WA
SA
NSW
V
T

Arc Consistency

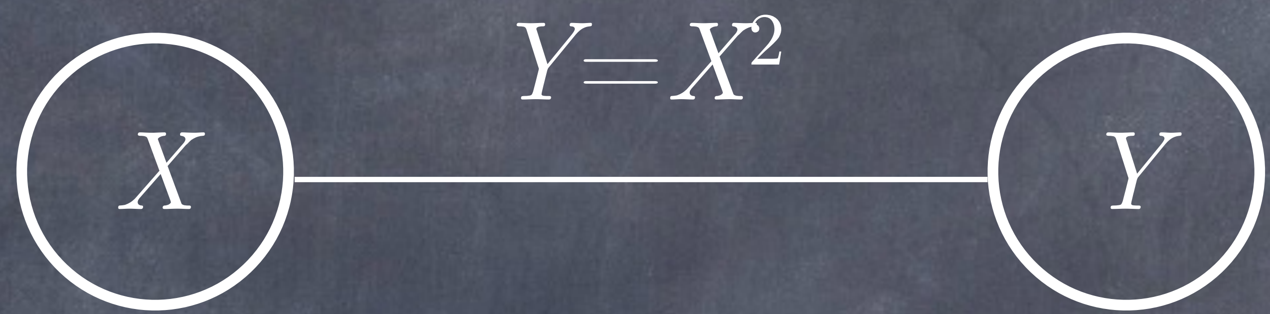
X_i is arc-consistent w.r.t. X_j if

for every value in the domain D_i,

there is some value in the domain D_j

that satisfies the binary constraint on the arc (X_i, X_j)

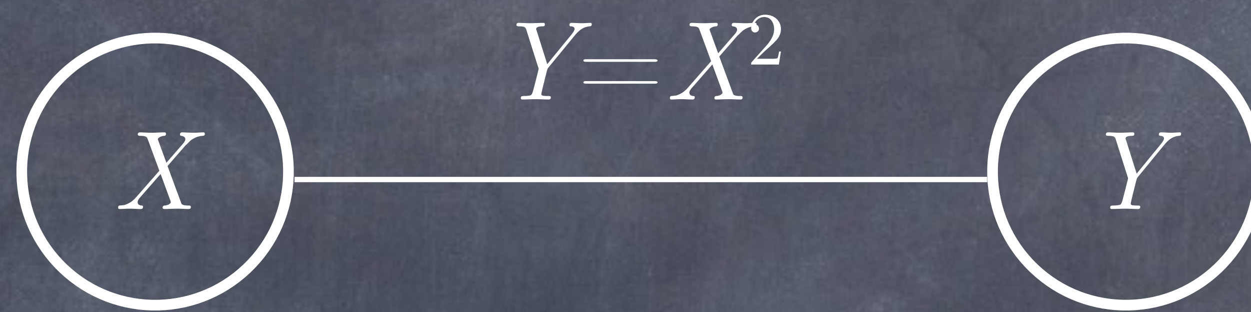
Arc Consistency



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

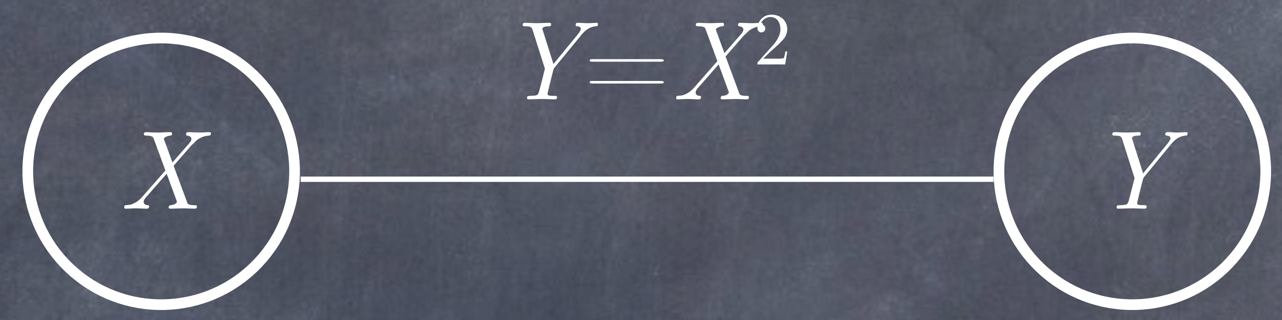
possible assignments: $10 \times 10 = 100$

Arc Consistency



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

Arc Consistency

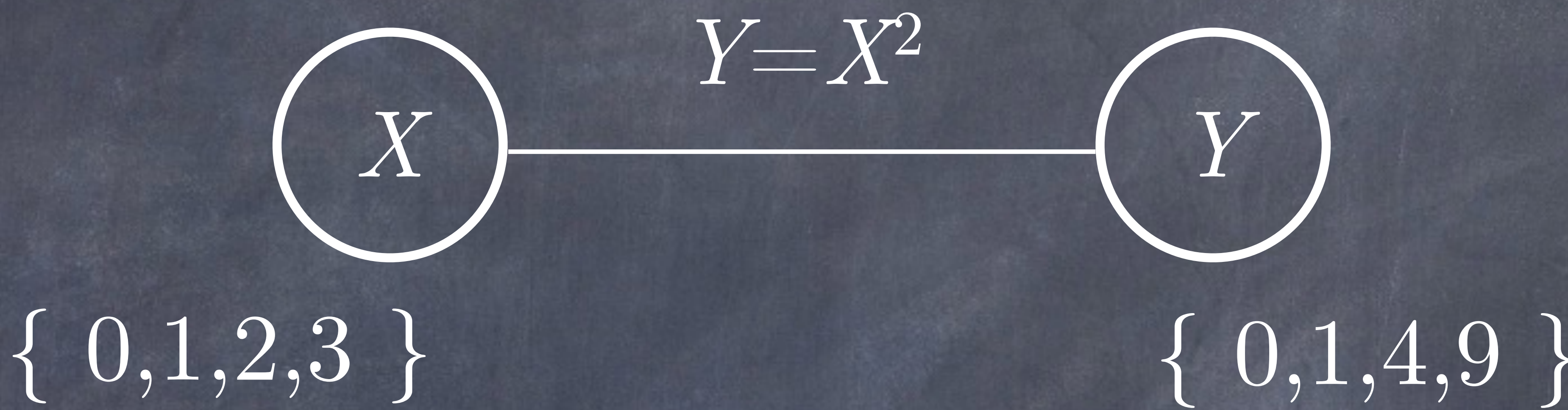


$\{ 0,1,2,3 \}$

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

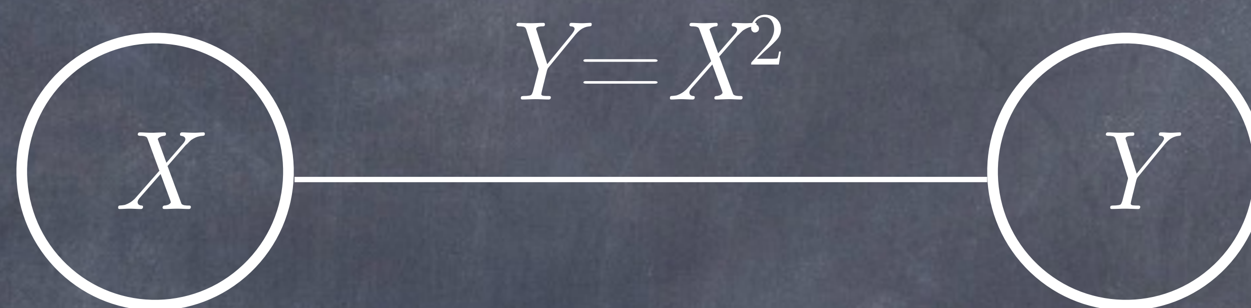
X arc-consistent with respect to Y

Arc Consistency



Y arc-consistent with respect to X

Arc Consistency



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

possible assignments: 4×4 = 100

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 ≤ 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 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.”

Constraint Propagation (inference)

↔

State-Space Search

State-Space Search for CSPs

- State: assignment of values to variables
- Initial state: all variables unassigned
- Action: Assign value to variable

$$X_1 = \varnothing, X_2 = \varnothing, \ldots, X_n = \varnothing$$

$X_1 = v_1, X_2 = \varnothing, \ldots, X_n = \varnothing$ $\quad$ $X_1 = \varnothing, X_2 = v_1, \ldots, X_n = \varnothing$ $\quad$ $X_1 = \varnothing, X_2 = \varnothing, \ldots, X_n = v_1$

$X_1 = v_2, X_2 = \varnothing, \ldots, X_n = \varnothing$ $\quad$ $X_1 = \varnothing, X_2 = v_2, \ldots, X_n = \varnothing$ $\quad$ $X_1 = \varnothing, X_2 = \varnothing, \ldots, X_n = v_2$

$\ldots$ $\quad$ $\ldots$ $\quad$ $\ldots$

Total number of nodes searched:

$$n \cdot d \times (n-1) \cdot d \times \ldots \times 1 \cdot d = n! \cdot d^n$$

# of possible complete assignments: d^n

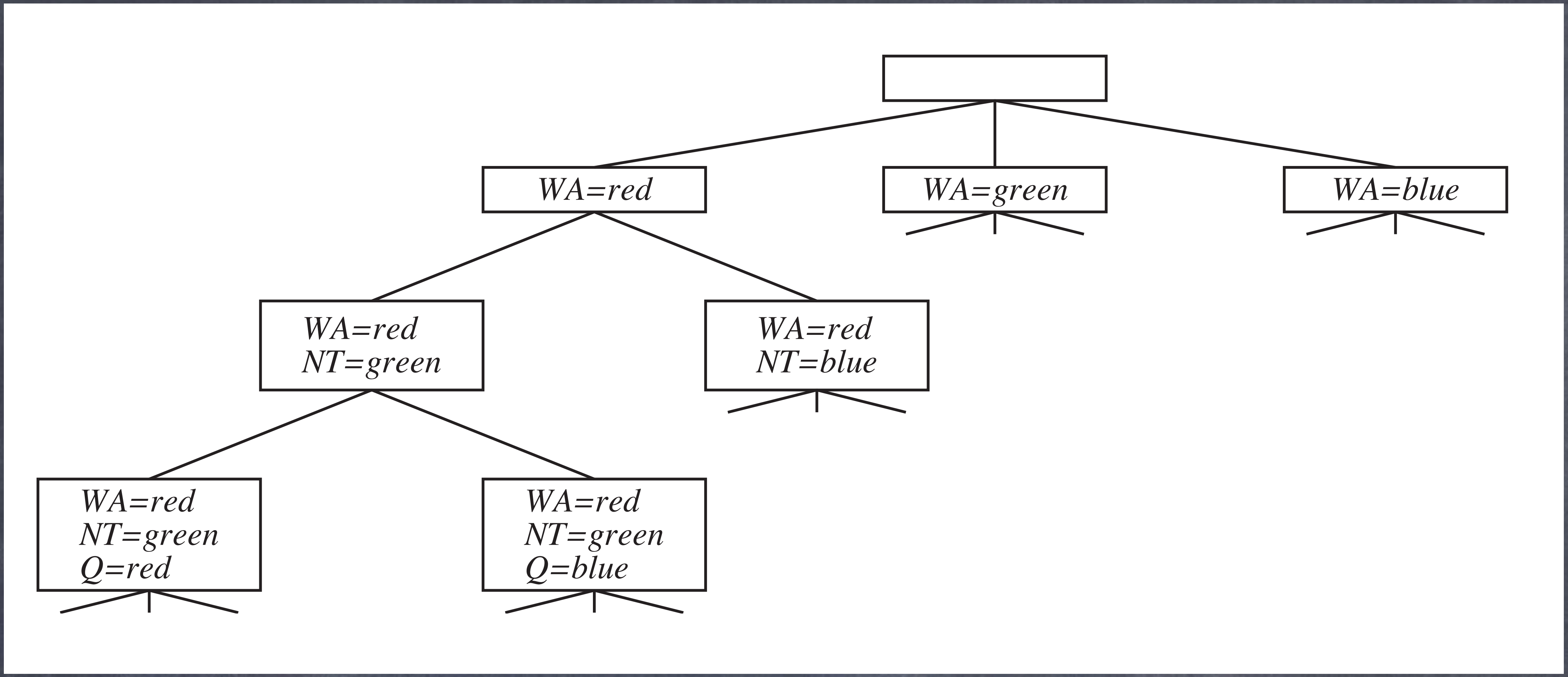
Commutativity

commutative |ˈkämyəˌtātiv; kəˈmyoōtətiv|

adjective Mathematics
involving the condition that a group of quantities connected by operators gives the same result whatever the order of the quantities involved, e.g., $a \times b = b \times a$.

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



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

$$d^n$$