2. Problem Solving

Artificial Intelligence and Neural Network (AINN)

(Part III)

Dr. Udaya Raj Dhungana
Assist. Professor
Pokhara University, Nepal
Guest Faculty
Hochschule Darmstadt University of Applied Sciences, Germany
E-mail: udaya@pu.edu.np and udayas.epost@gmail.com

Overview

- Constraint Satisfaction Problems
- Constraint Propagation
- Backtracking Search- Game Playing.
- Cryptarithmetic Problem

Constraint Satisfaction Problems

- Standard search problem:
  - state is a "black box“ – any data structure that supports successor function, heuristic function, and goal test
  - Problems can be solved by searching in a space of states
- CSP:
  - state is defined by variables X_i with values V_i from domain D_i
  - goal test is a set of constraints specifying allowable combinations of values for subsets of variables

Constraint Satisfaction Problems

- a way to solve a wide variety of problems more efficiently.
- We use a **factored representation** for each state: ***a set of variables***, ***each of which has a value***.
- A problem is solved when each variable has a value that satisfies all the constraints on the variable.
- A problem described this way is called a **constraint satisfaction problem**, or **CSP**.

Constraint Satisfaction Problems

- CSP search algorithms
    - **Idea:** eliminate large portions of the search space all at once by identifying variable/value combinations that violate the constraints.

Constraint Satisfaction Problems

- A constraint satisfaction problem is defined by 3 components (X, D, C):
  - X is **a set of variables**, {X_1,...,X_n}.
  - D is **a set of domain containing** allowable values {v_1,...,v_k}, one value for each variable X_i.
  - C is **a set of constraints** that specify allowable combinations of values.
    - C = set of ⟨scope , rel ⟩ where scope is a tuple of variables that participate in the constraint and rel is a relation that defines the values that those variables can take on.

Constraint Satisfaction Problems

- Then, in CSP,
  - state is defined by variables X_i with values V_i from domain D_i
  - goal test is a set of constraints specifying allowable combinations of values for subsets of variables
  - For example, if X1 and X2 both have the domain {A,B}, then the constraint saying the two variables must have different values can be written as
    - $\langle(X1, X2), [(A, B), (B, A)]\rangle$ or
    - $\langle(X1, X2), X1 \neq X2\rangle$.

Solution to CSP

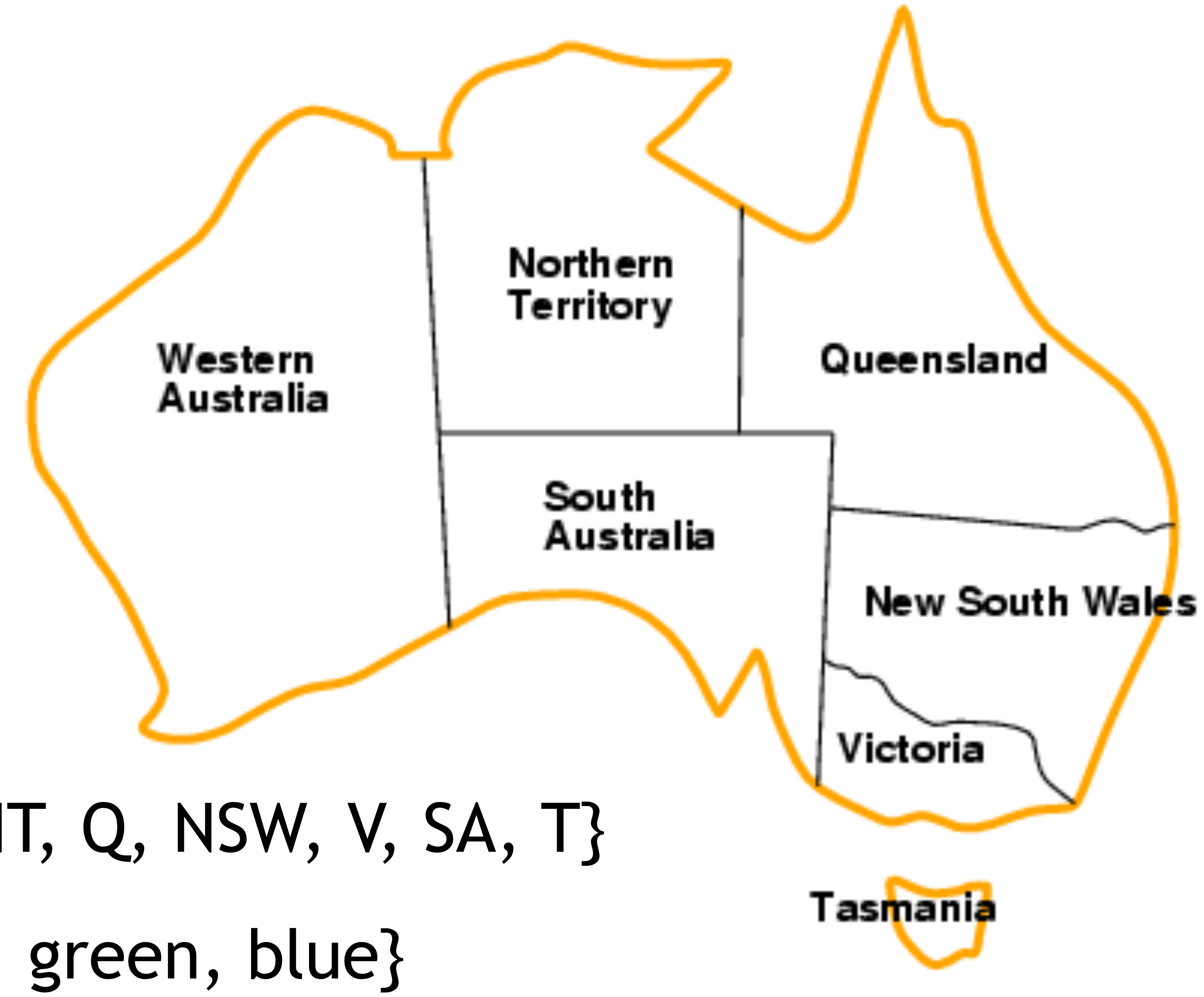
- Each state in a CSP is defined by an **assignment** of values to some or all of the variables, $\{X_i = v_i, X_j = v_j, \ldots\}$.

- An assignment that does not violate any constraints is called a **consistent** or legal assignment.

- A **complete assignment** is one in which every variable is assigned, and

- a **solution** to a CSP is a consistent, complete assignment.

Example: Map Coloring



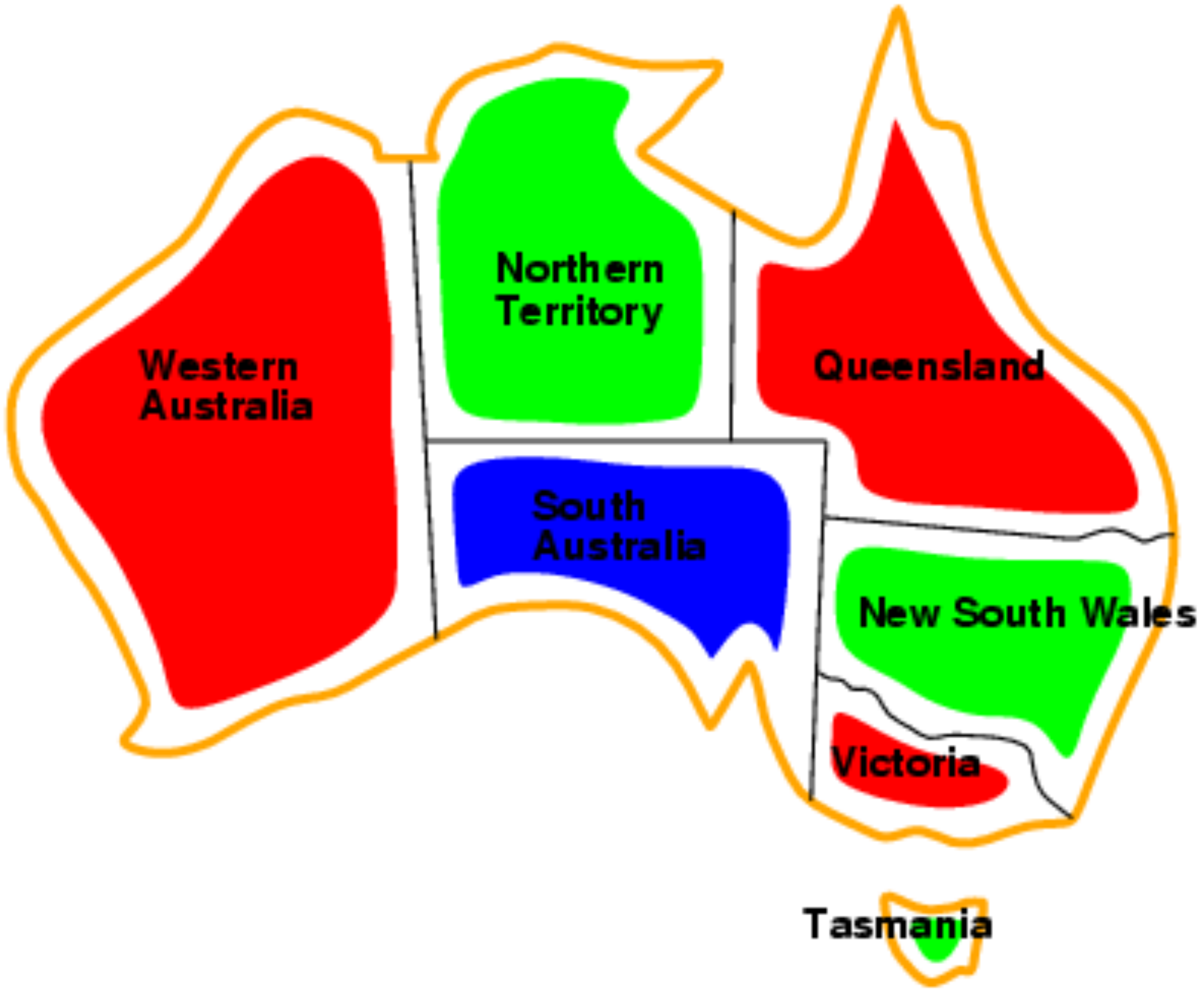
Problem:

- Variables = {WA, NT, Q, NSW, V, SA, T}
- Domains D_i = {red, green, blue}
- Constraints: adjacent regions must have different colors
- e.g., WA ≠ NT, or (WA,NT) in {(red, green),(red, blue),(green, red), (green, blue),(blue, red),(blue, green)}

Solution = ?

Example: Map Coloring

Solution:

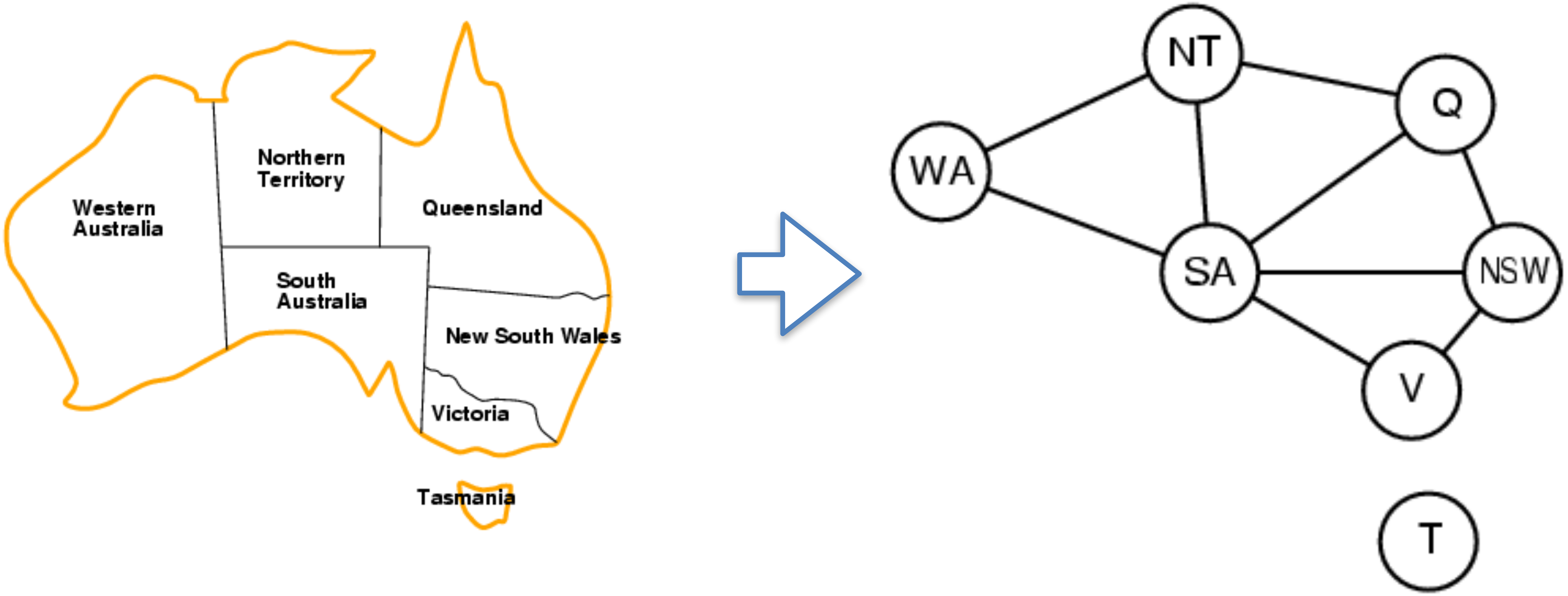


- Solutions are complete and consistent assignments,
- e.g., WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green

Constraint Graph

- Binary CSP: each constraint relates two variables

- Constraint graph: nodes are variables, arcs are constraints



Constraint Graph

CSP as a Search Problem

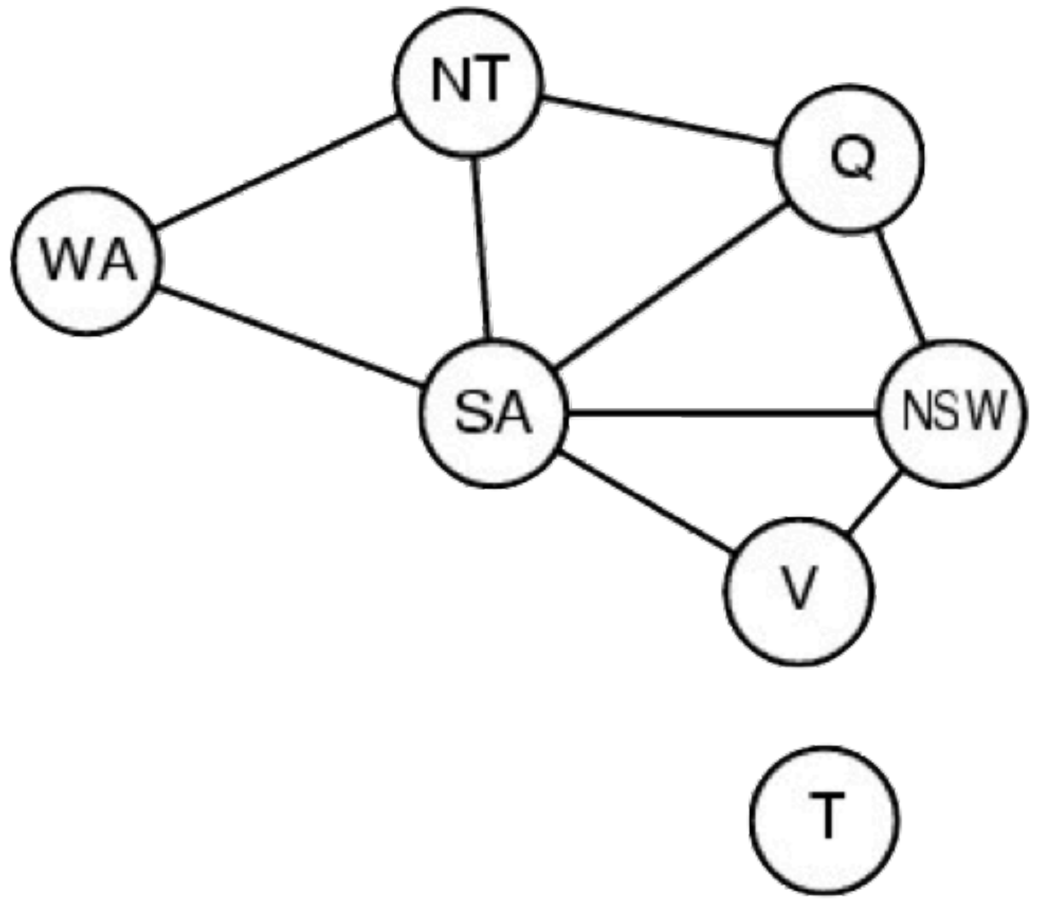
- **Initial state:**
  - the empty assignment {} – all variables are unassigned
- **Successor function:**
  - a value is assigned to one of the unassigned variables with no conflict
  - fail if no legal assignments
- **Goal test:**
  - a complete assignment: all variables have a value and none of the constraints is violated.
- **Path cost:**
  - a constant cost for each step
- Solution appears at depth n if there are n variables
- Depth-first or local search methods work well
- Path is irrelevant, so can also use complete-state formulation

CSP Solvers Can be Faster

CSP solver can quickly eliminate large part of search space

If {SA = blue}

Then 3^5 assignments can be reduced to 2^5 assignments, a reduction of 87%



In a CSP, if a partial assignment is not a solution, we can immediately discard further refinements of it

Types of Variables

- **Discrete variables**
  - finite domains:
    - n variables, domain size d → $O(d^n)$ complete assignments
    - e.g., Boolean CSPs, incl.~Boolean satisfiability (NP-complete)
  - infinite domains:
    - integers, strings, etc.
    - e.g., job scheduling, variables are start/end days for each job
    - need a constraint language, e.g., $StartJob_1 + 5 \leq StartJob_3$
- **Continuous variables**
  - e.g., start/end times for Hubble Space Telescope observations
  - linear constraints solvable in polynomial time by linear programming

Types of Constraints

- Unary constraints involve a single variable,
  - e.g., SA ≠ green

- Binary constraints involve pairs of variables,
  - e.g., SA ≠ WA

- Higher-order constraints involve 3 or more variables,
  - e.g., cryptarithmetic column constraints

Real-World CSPs

- Assignment problems
  - e.g., who teaches what class
- Timetabling problems
  - e.g., which class is offered when and where?
- Transportation scheduling
- Factory scheduling

What Search Algorithm to Use?

- Since we can formulate CSP problems as standard search problems, we can apply any search algorithms
- If breadth-first search were applied
  - branching factor? nd
  - tree size? $nd * (n\text{-}1)d * \ldots * d = n! * d^n$ leaves
  - complete assignments? d^n
- A crucial property to all CSPs: commutativity
  - the order of application of any given set of actions has no effect on the outcome
  - Variable assignments are commutative, i.e., [WA = red then NT = green] same as [NT = green then WA = red]

Backtracking Search

- Only need to consider assignments to a single variable at each node → b = d and there are dn leaves
- Backtracking search is used for a depth-first search that chooses values for one variable at a time and backtracks when a variable has no legal values left to assign
- Backtracking search is the basic uninformed algorithm for CSPs

Backtracking Search

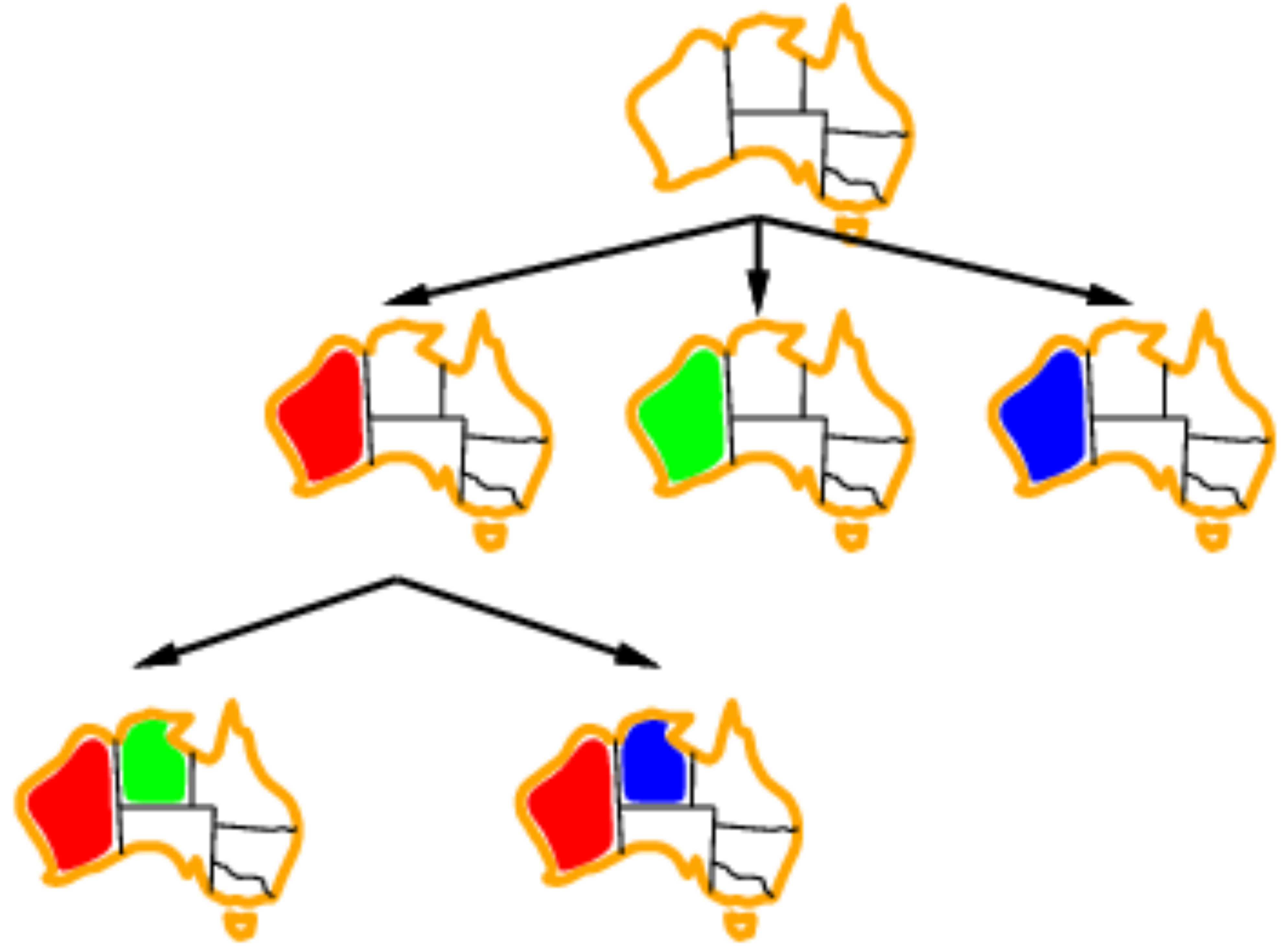
```
function BACKTRACKING-SEARCH( csp) returns a solution, or failure
   return RECURSIVE-BACKTRACKING({}, csp)

function RECURSIVE-BACKTRACKING( assignment, csp) returns a solution, or
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 according to Constraints[csp] then
         add { var = value } to assignment
         result ← RECURSIVE-BACKTRACKING(assignment, csp)
         if result ≠ failue then return result
         remove { var = value } from assignment
   return failure
```

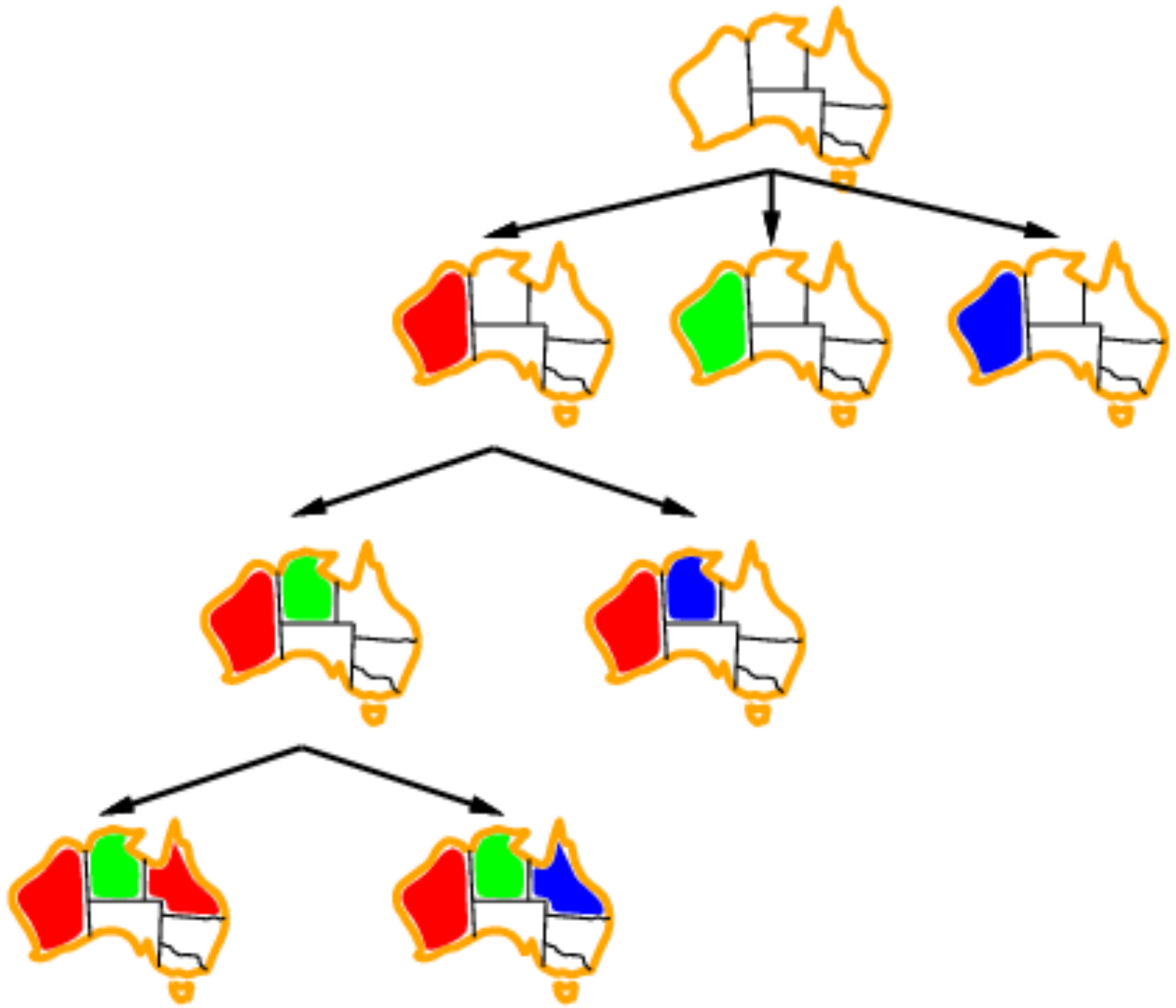
Backtracking Example

Backtracking Example

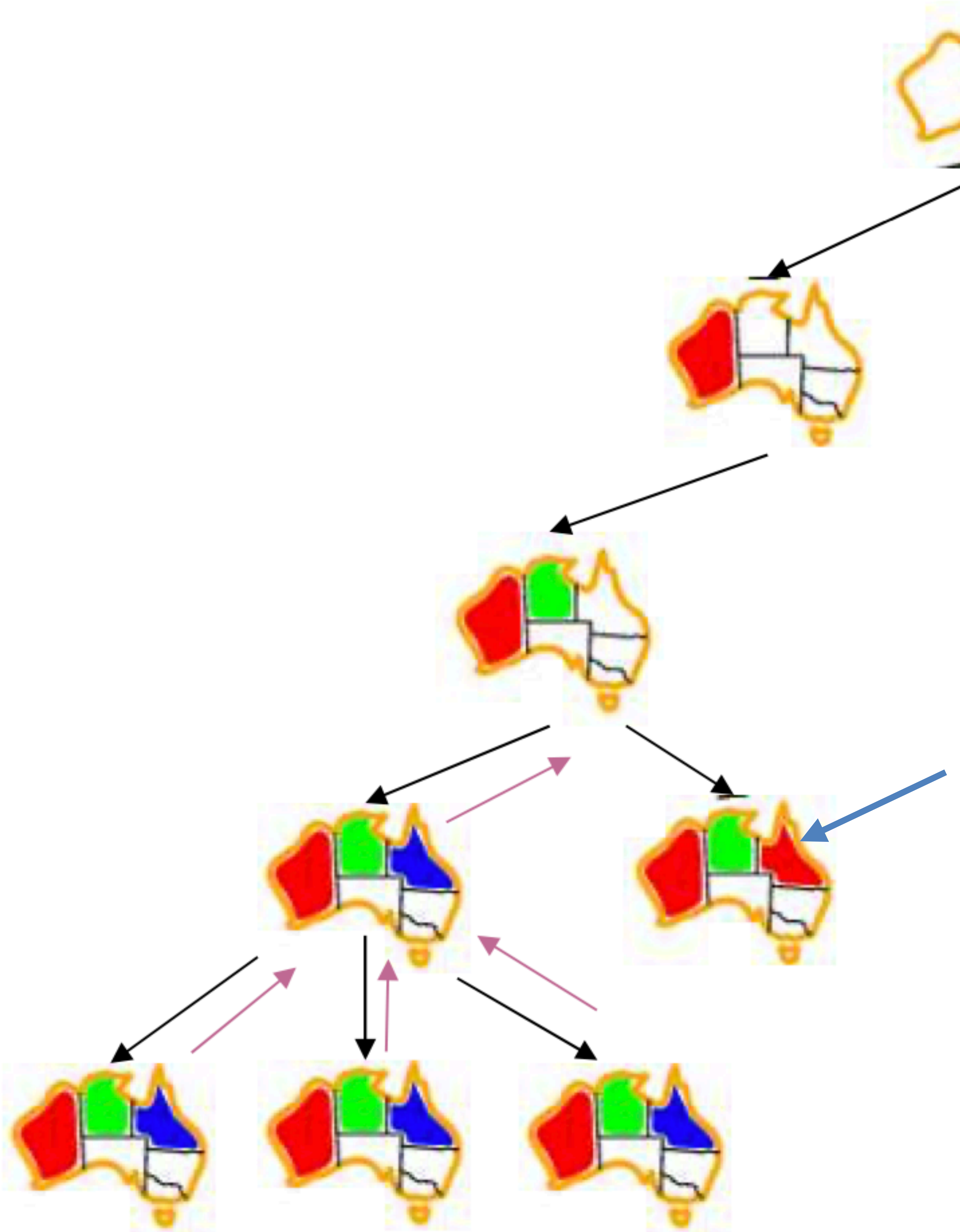
Backtracking Example



Backtracking Example



Backtracking Example



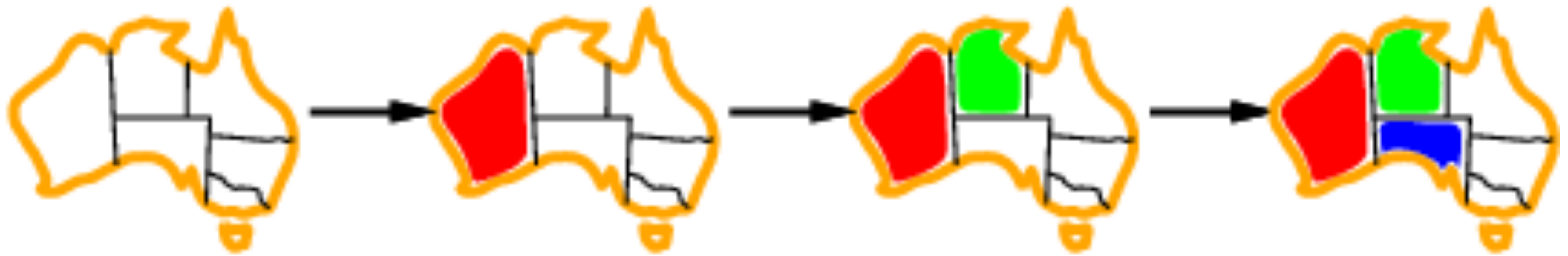
Backtracked and assigned red color for Queensland

Improving Backtracking Efficiency

- General-purpose methods can give huge gains in speed:
  - Which variable should be assigned next?
  - In what order should its values be tried?
  - Can we detect inevitable failure early?

Most Constrained Variable

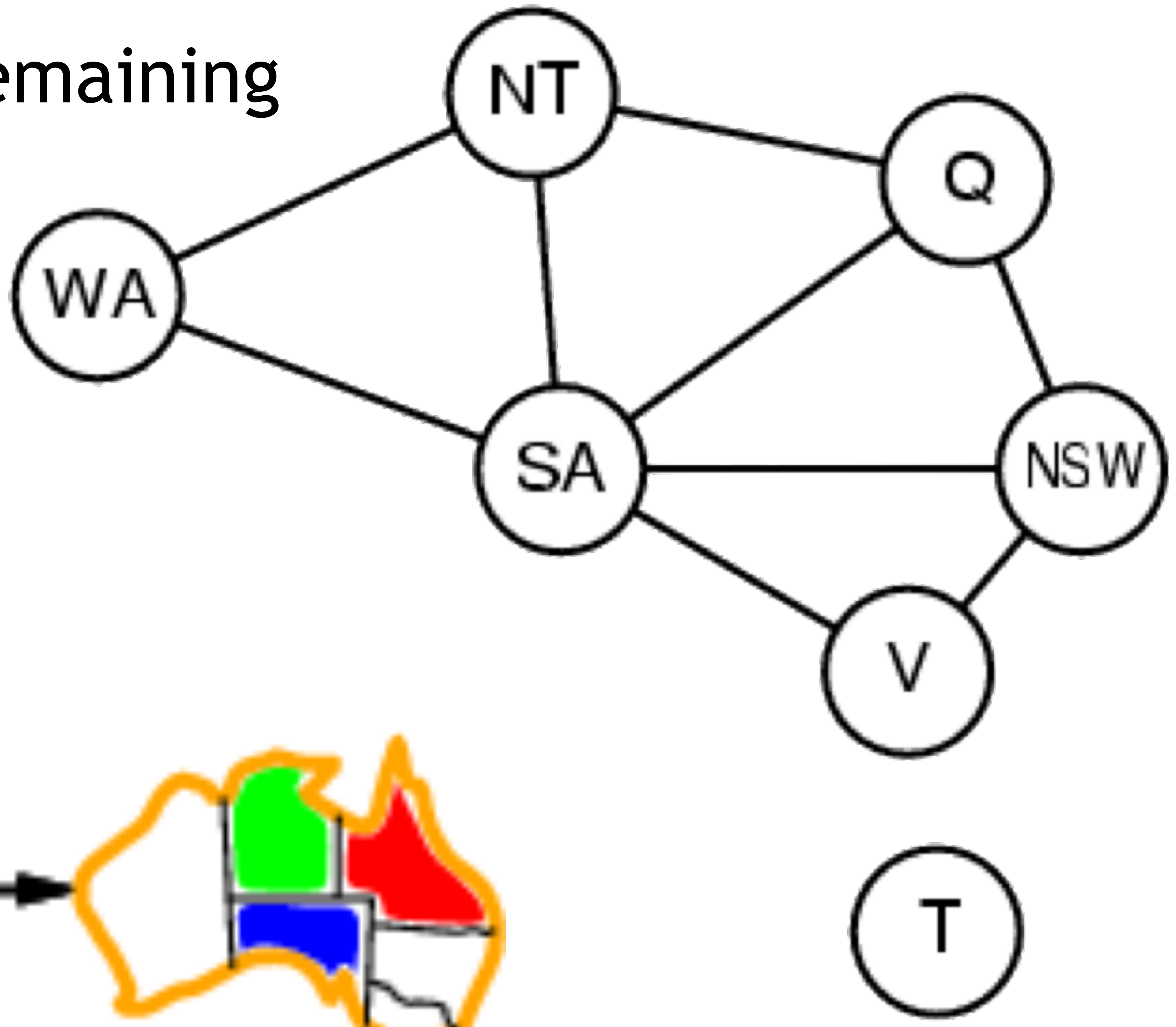
- Most constrained variable:
  - choose the variable with the fewest legal values

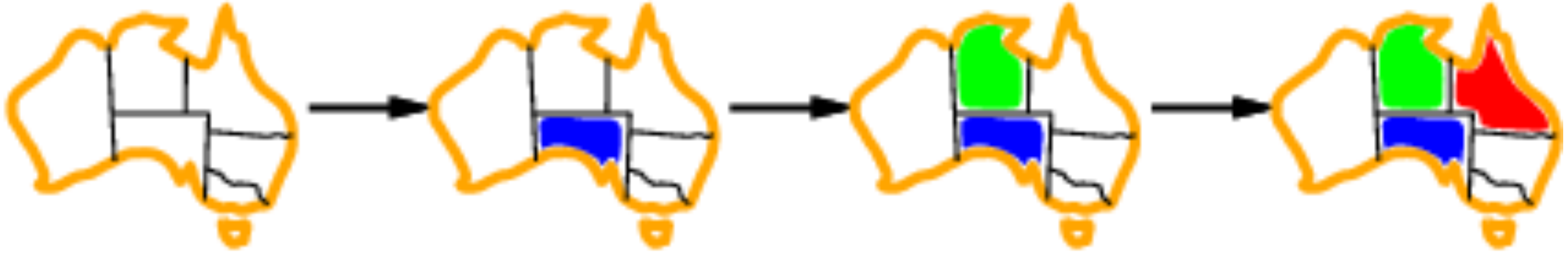


- Also known as minimum remaining values (MRV) or **fail-first** heuristic
- Picks a variable which will cause failure as soon as possible, allowing the tree to be pruned.

Most Constraining Variable

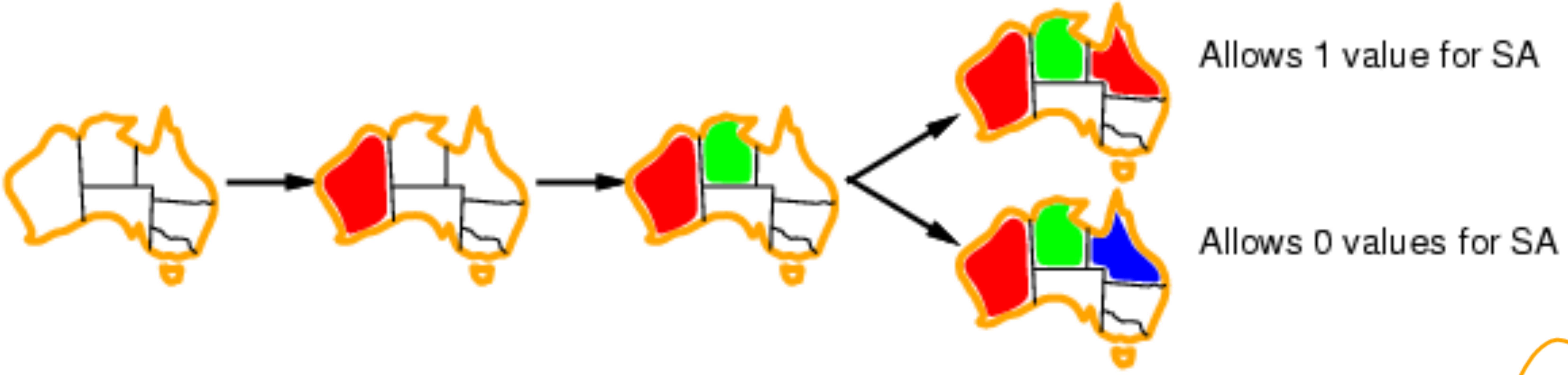
- Most constraining variable:
  - choose the variable with the most constraints on remaining variables (most edges in graph i.e. SA)
  - also called degree heuristics
- Tie-breaker among most constrained variables





Least Constraining Value

- Given a variable, choose the least constraining value:
  - the one that rules out the fewest values in the remaining variables



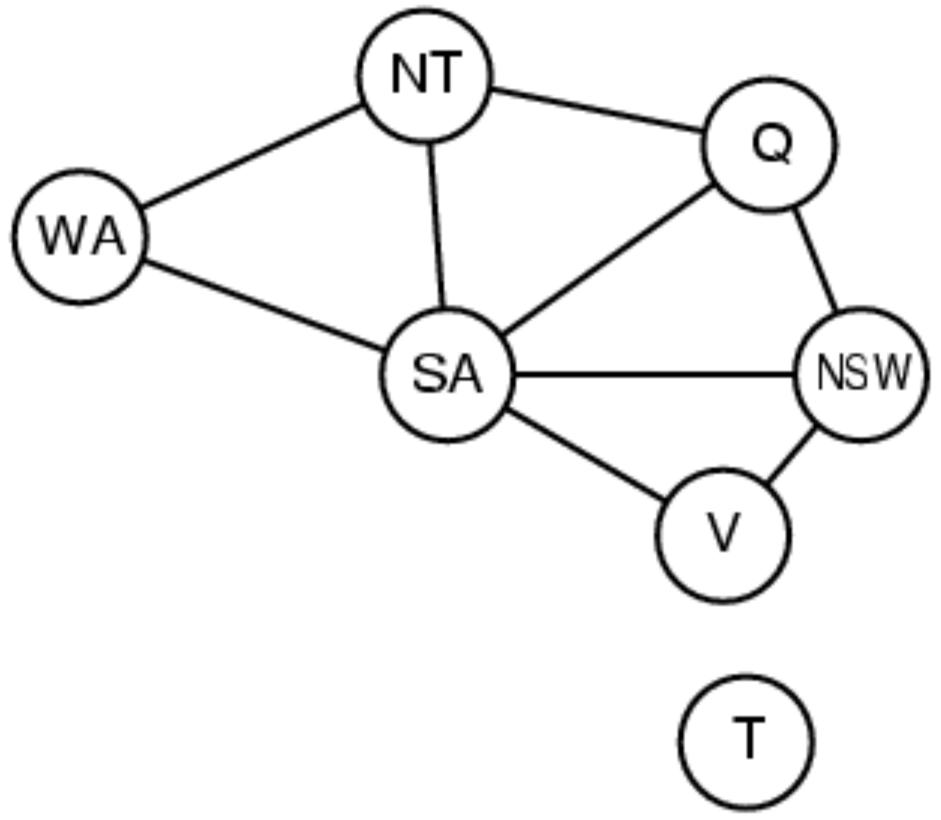
- Leaves maximal flexibility for a solution.
- Combining these heuristics makes 1000 queens feasible

Forward Checking

(Propagating Information through Constraints)

Idea:

- Keep track of remaining legal values for unassigned variables
- Terminate search when any variable has no legal values

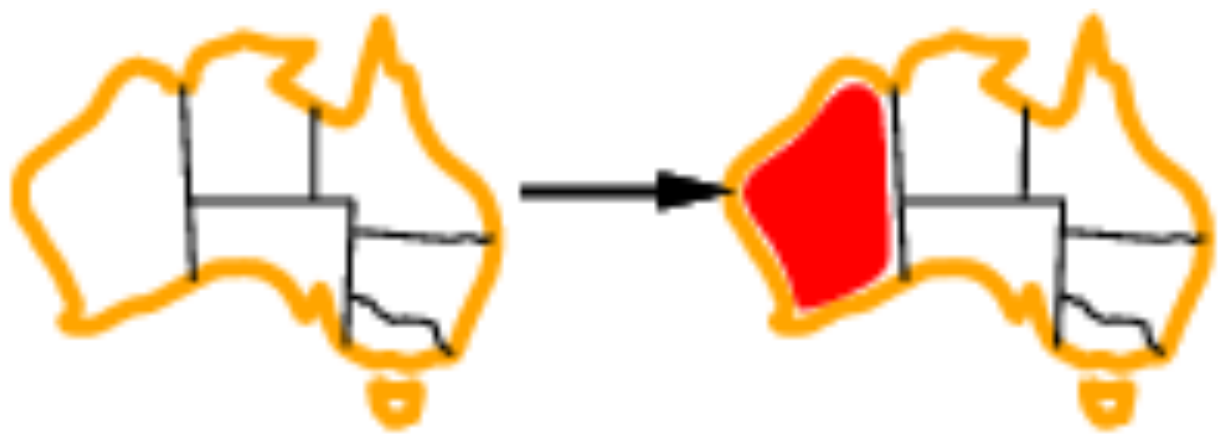


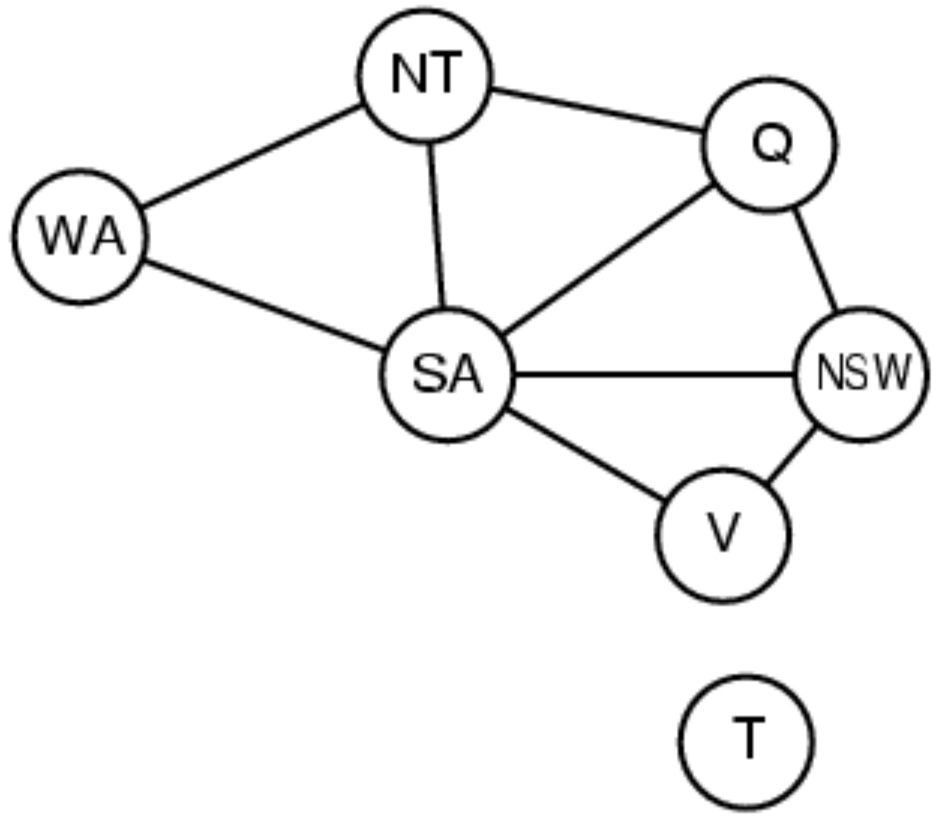
Forward Checking

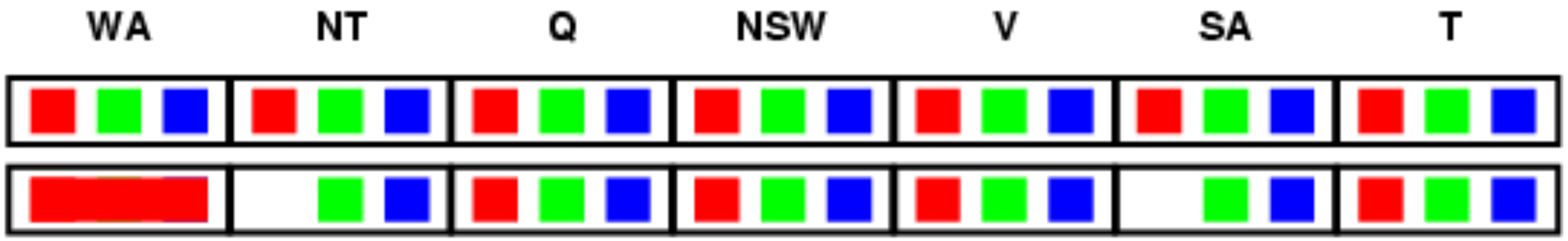
(Propagating Information through Constraints)

Idea:

- Keep track of remaining legal values for unassigned variables
- Terminate search when any variable has no legal values





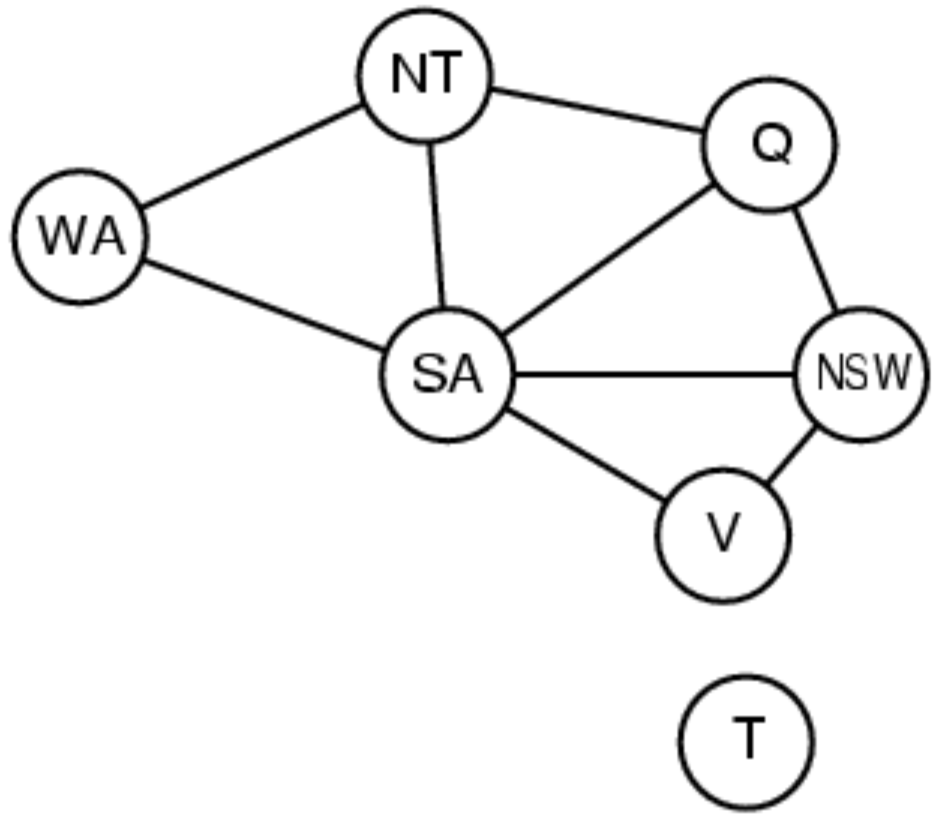


Forward Checking

(Propagating Information through Constraints)

Idea:

- Keep track of remaining legal values for unassigned variables
- Terminate search when any variable has no legal values

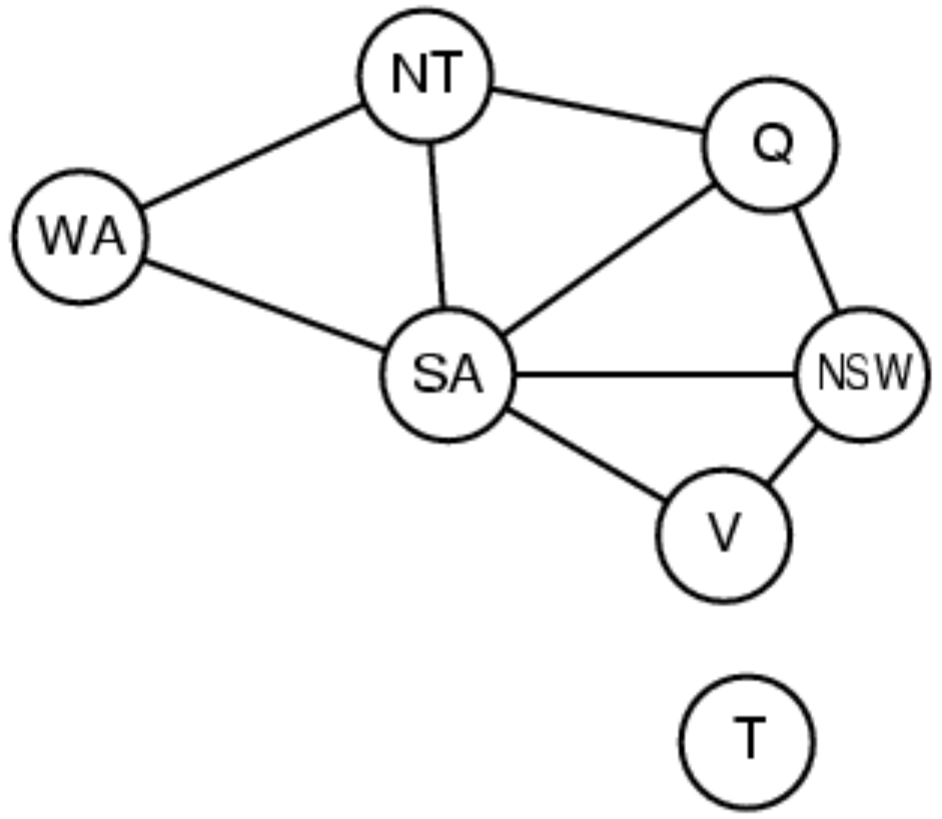


Forward Checking

(Propagating Information through Constraints)

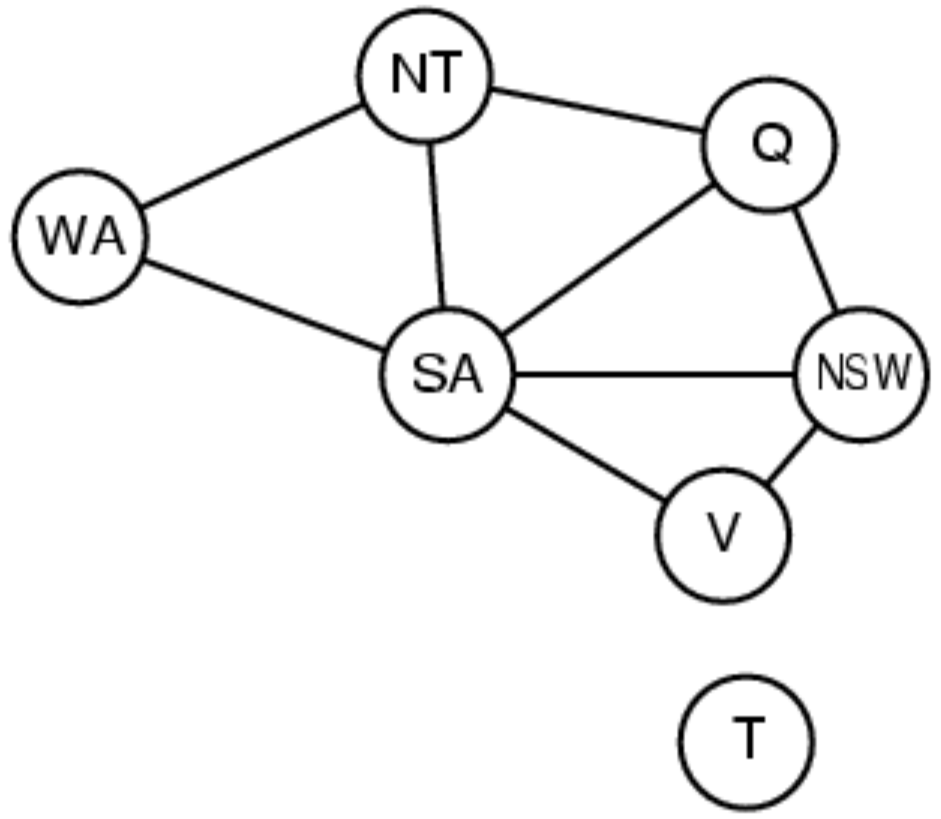
Idea:

- Keep track of remaining legal values for unassigned variables
- Terminate search when any variable has no legal values



Constraint Propagation

- Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures:



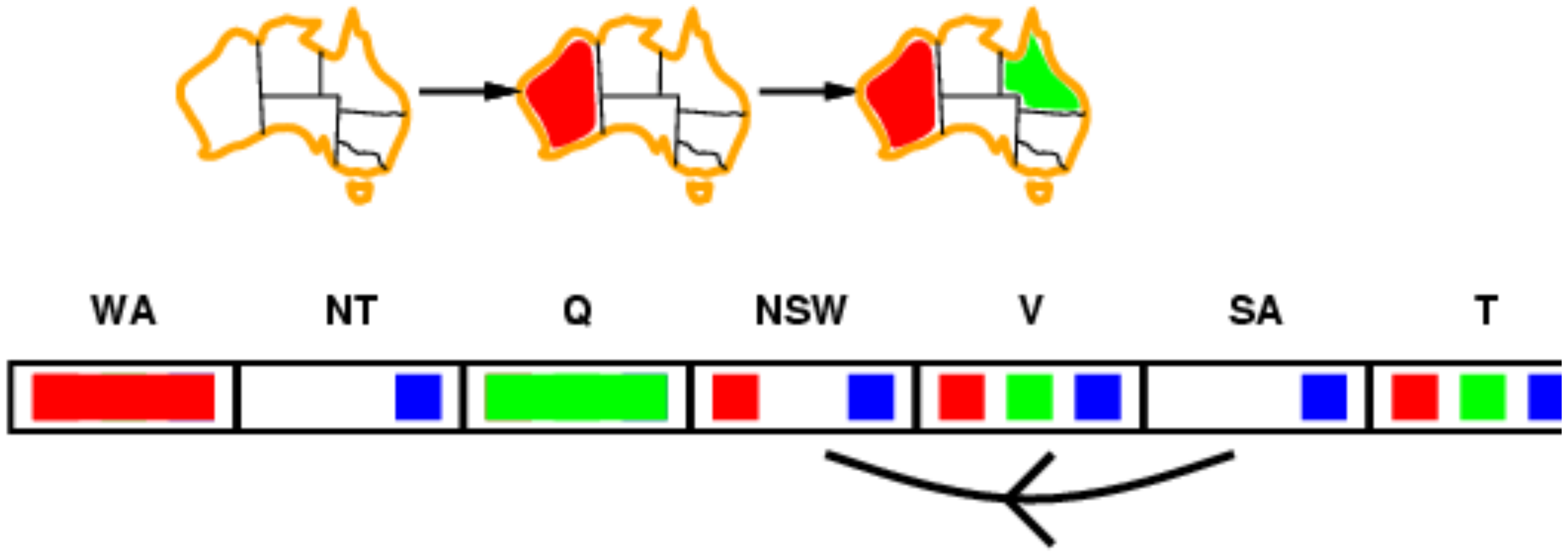
- NT and SA cannot both be blue!
- Constraint propagation repeatedly enforces constraints locally by propagating implications of a constraint of one variable onto other variables

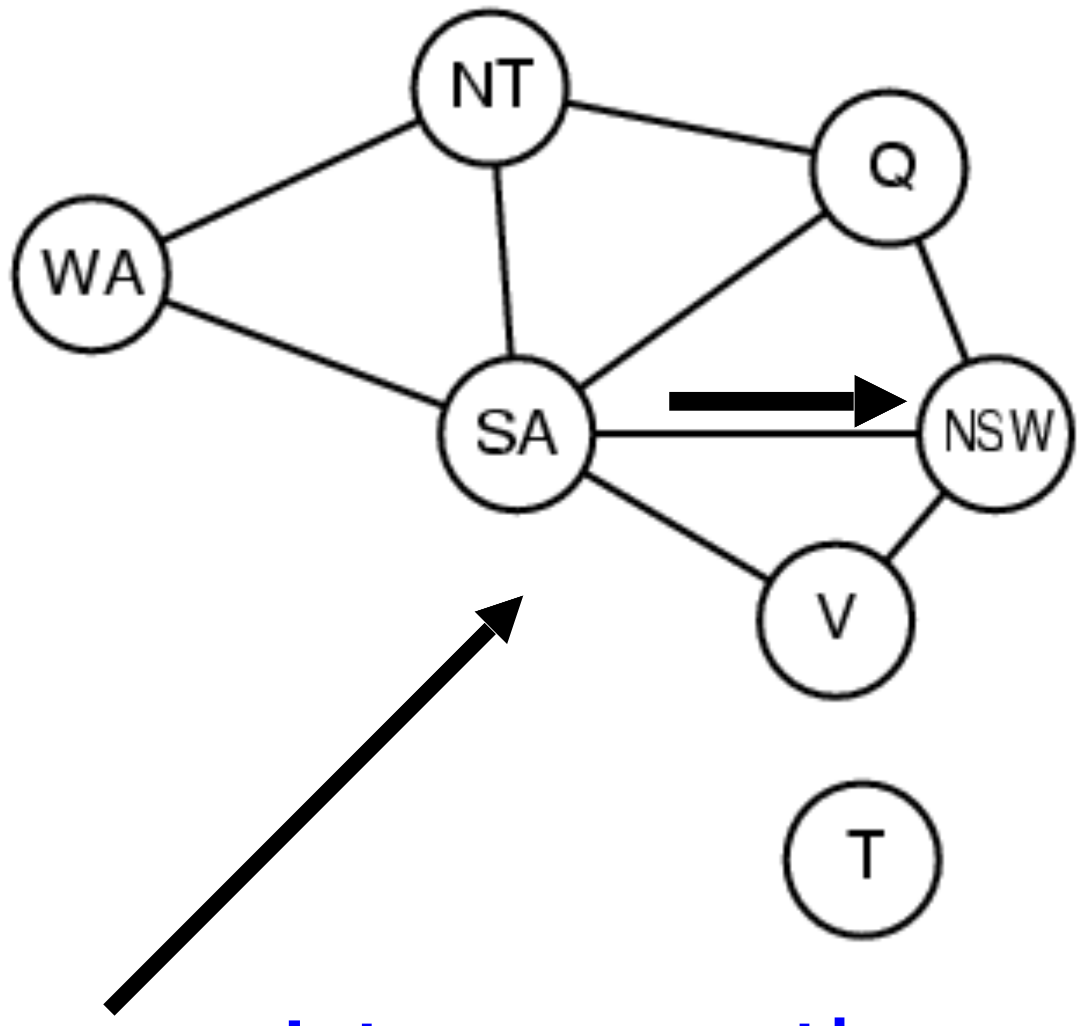
Node Consistency

- A single variable is node-consistent if all the values in the variable's domain satisfy the variable's unary constraints
- For example, SA dislikes green
- A network is node-consistent if every variable in the network is node-consistent

Arc Consistency

- Simplest form of propagation makes each arc consistent
- X →Y is consistent iff
  - for every value x of X there is some allowed y

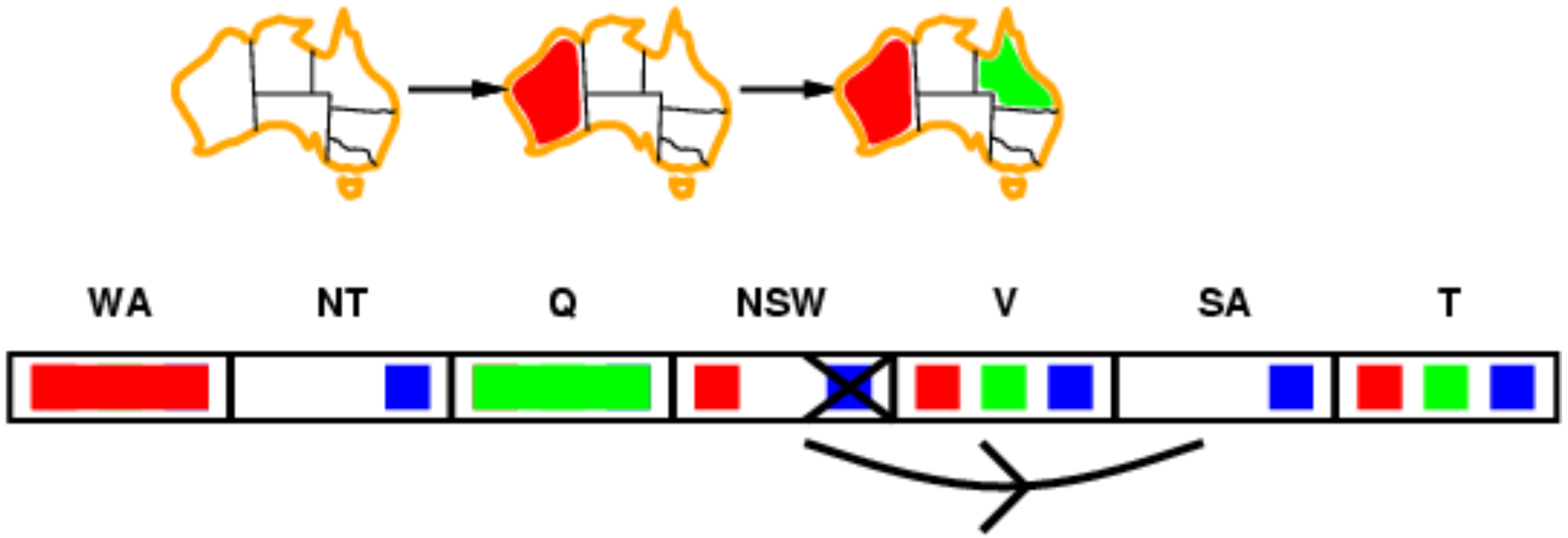


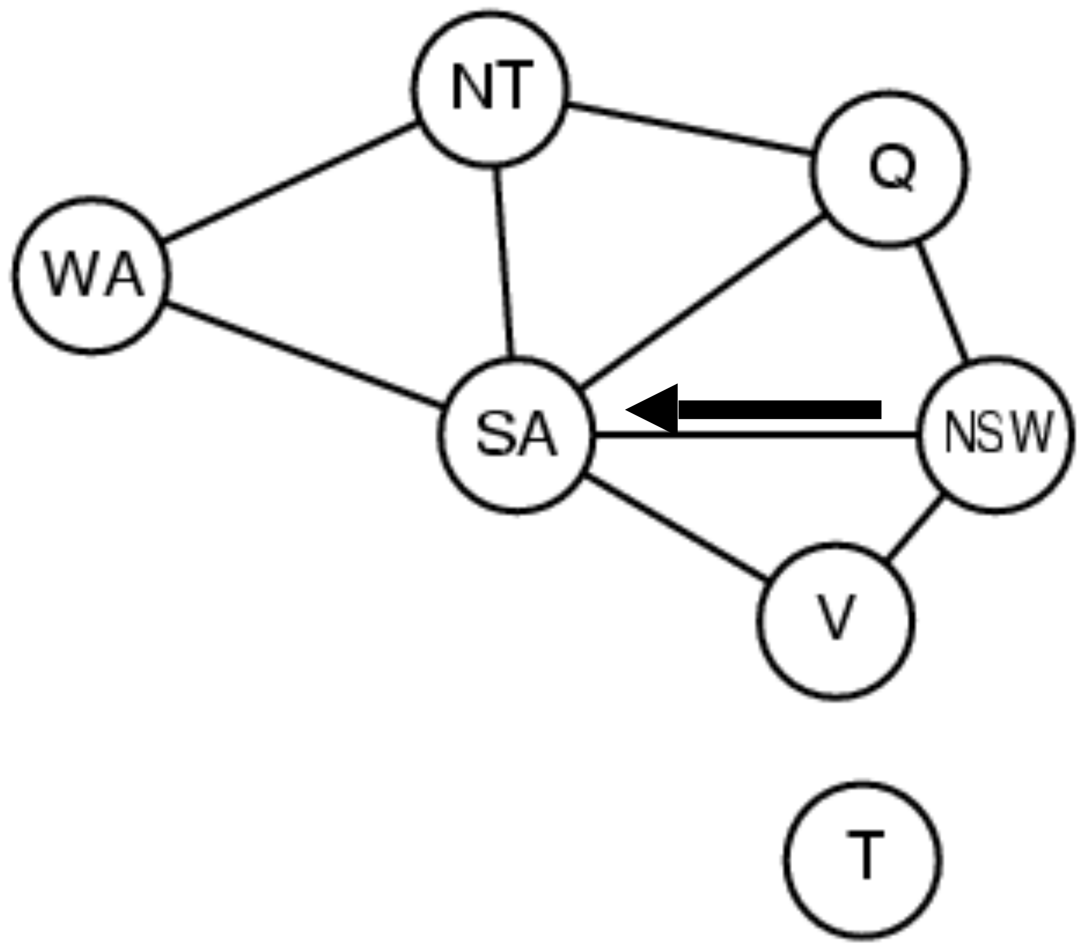


constraint propagation propagates arc consistency on the graph.

Arc Consistency

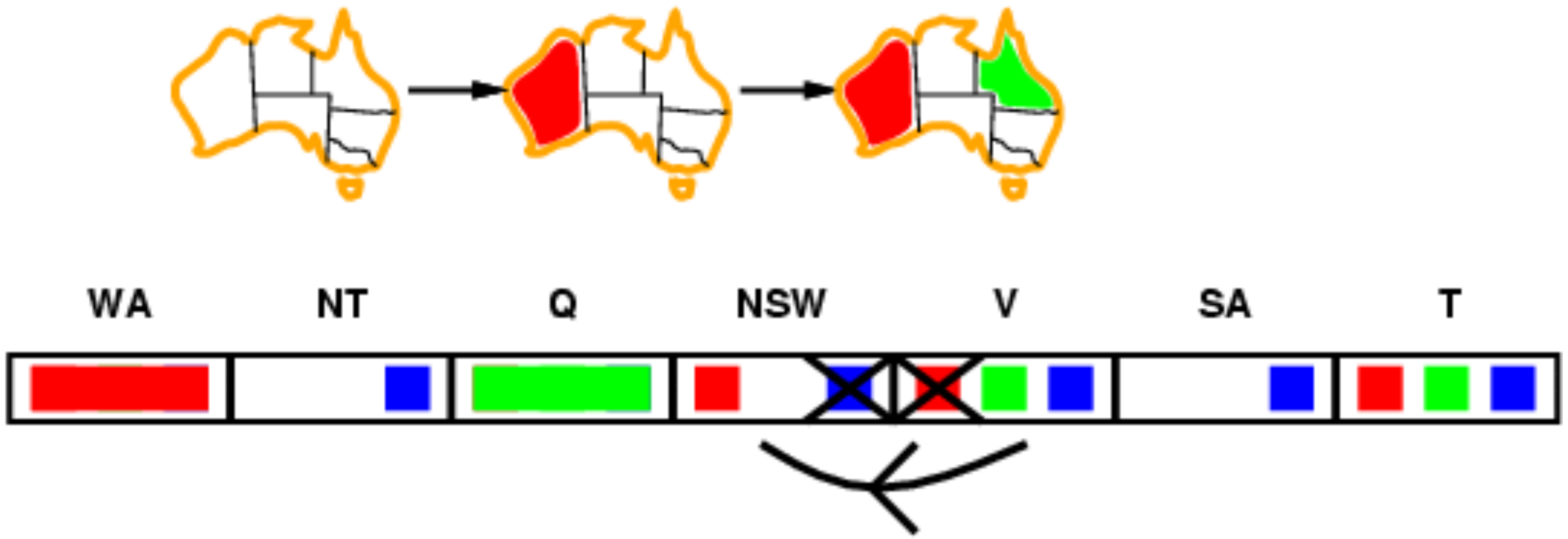
- Simplest form of propagation makes each arc consistent
- X →Y is consistent iff
  - for every value x of X there is some allowed y

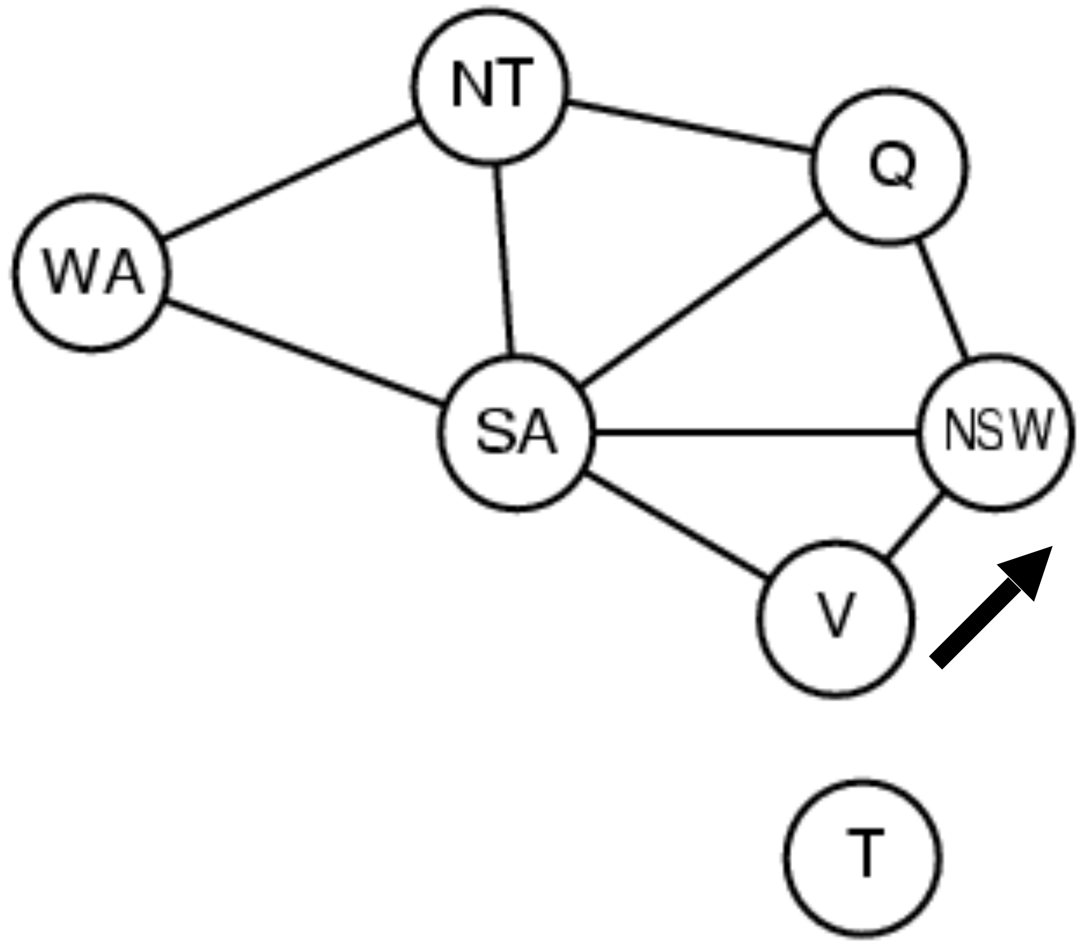




Arc Consistency

- Simplest form of propagation makes each arc consistent
- X →Y is consistent iff
  - for every value x of X there is some allowed y

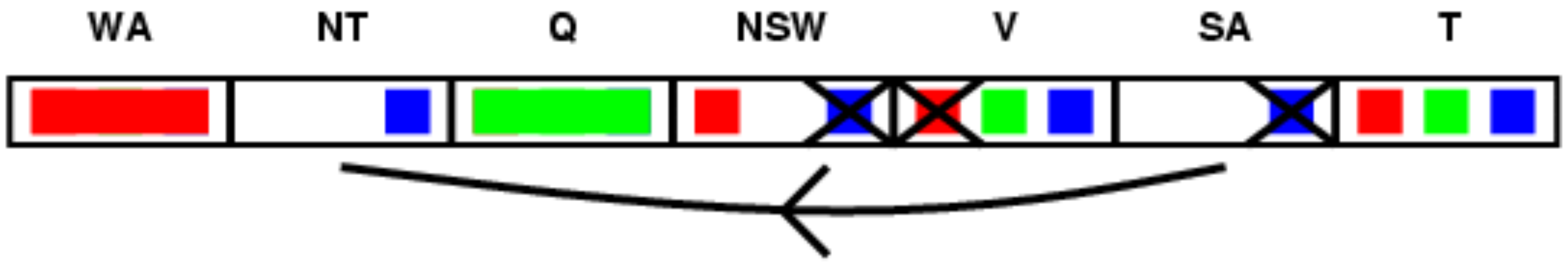


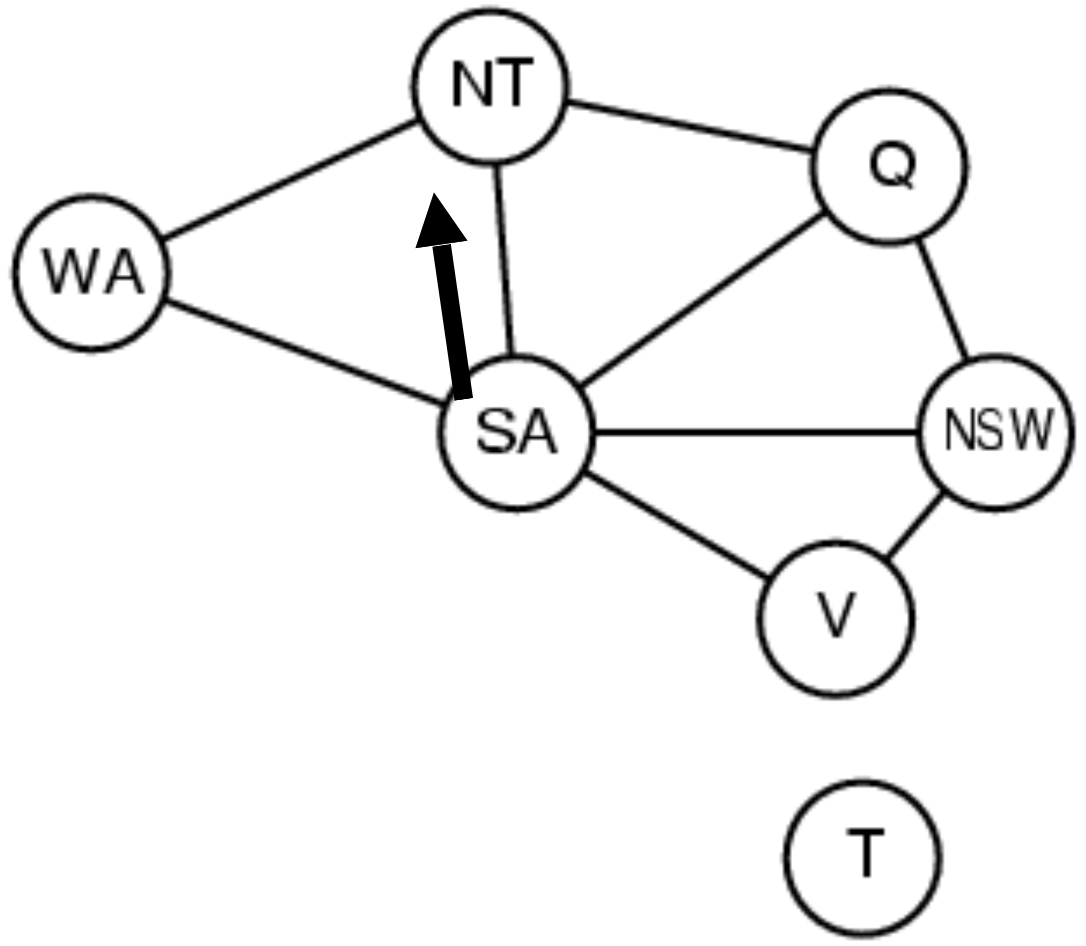


- If X loses a value, neighbors of X need to be rechecked

Arc Consistency

- Simplest form of propagation makes each arc consistent
- X →Y is consistent iff
  - for every value x of X there is some allowed y





- If X loses a value, neighbors of X need to be rechecked
- Arc consistency detects failure earlier than forward checking
- Can be run as a preprocessor or after each assignment

Arc Consistency Algorithm AC-3

```
function AC-3( csp) returns the CSP, possibly with reduced domains
   inputs: csp, a binary CSP with variables {X1, X2, ..., Xn}
   local variables: queue, a queue of arcs, initially all the arcs in csp

   while queue is not empty do
      (Xi, Xj) ← REMOVE-FIRST(queue)
      if RM-INCONSISTENT-VALUES(Xi, Xj) then
         for each Xk in NEIGHBORS[Xi] do
            add (Xk, Xi) to queue
```

```
function RM-INCONSISTENT-VALUES( Xi, Xj) returns true iff remove a value
   removed ← false
   for each x in DOMAIN[Xi] do
      if no value y in DOMAIN[Xj] allows (x,y) to satisfy constraint(Xi, Xj)
         then delete x from DOMAIN[Xi];  removed ← true
   return removed
```

Time complexity: $O(n^2d^3)$

Local Search for CSPs

- Hill-climbing, simulated annealing typically work with "complete" states, i.e., all variables assigned

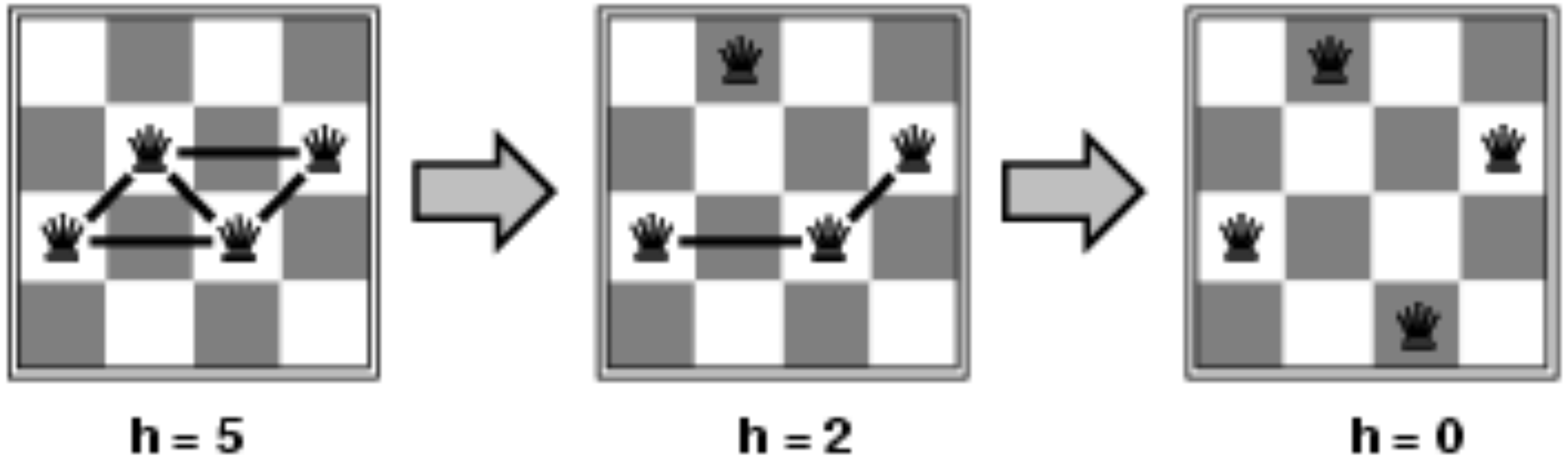
- To apply to CSPs:
  - allow states with unsatisfied constraints
  - operators reassign variable values

- Variable selection: randomly select any conflicted variable

- Value selection by min-conflicts heuristic:
  - choose value that violates the fewest constraints
  - i.e., hill-climb with h(n) = total number of violated constraints

Example: 4 Queens

- States: 4 queens in 4 columns (4^4 = 256 states)
- Actions: move queen in column
- Goal test: no attacks
- Evaluation: h(n) = number of attacks



- Min-conflicts is quite effective for many CSPs.
- Given random initial state, can solve n-queens in almost constant time for arbitrary n with high probability (e.g., n = 10,000,000)

See the below videos

- Very Important Videos:
  - CSP Example of Map cploring:
    - https://www.youtube.com/watch?v=lCrHYT_EhDs
    - https://www.youtube.com/watch?v=udOfKqeLVSg
  - Cryptarithmetic Problem with an Example SEND + MORE = MONEY:
    - https://www.youtube.com/watch?v=HC6Y49iTg1k
    - https://www.cpp.edu/~ftang/courses/CS420/notes/CSP.pdf

Summary

- CSPs are a special kind of problem:
  - states defined by values of a fixed set of variables
  - goal test defined by constraints on variable values
- Backtracking = depth-first search with one variable assigned per node
- Variable ordering and value selection heuristics help significantly
- Forward checking prevents assignments that guarantee later failure
- Constraint propagation (e.g., arc consistency) does additional work to constrain values and detect inconsistencies
- Iterative min-conflicts is usually effective in practice

Questions

1. What are constraint satisfaction problems? How are they different from others?

2. What are the applications of CSPs in real world? Describe with examples.

3. What are the different components of CSPs? Describe in detail.

4. Explain the Map coloring example using CSP.

5. Describe constraint propagation with example.

6. Why do you need backtracking in CSPs? How can these backtracking be efficient? Explain with example.

Cryptarithmetic Problem

- Cryptarithmetic Problem is a type of constraint satisfaction problem where the game is about digits and its unique replacement either with alphabets or other symbols.

- In cryptarithmetic problem, the digits (0-9) get substituted by some possible alphabets or symbols.

- The task in cryptarithmetic problem is to substitute each digit with an alphabet to get the result arithmetically correct.

The rules or constraints on a cryptarithmetic problem

- There should be a unique digit to be replaced with a unique alphabet.
- The result should satisfy the predefined arithmetic rules, i.e., 2+2 =4, nothing else.
- Digits should be from 0-9 only.
- There should be only one carry forward, while performing the addition operation on a problem.
- The problem can be solved from both sides, i.e., lefthand side (L.H.S), or righthand side (R.H.S)

Example: Cryptarithmetic Problem

- Variables: F T U W R O
- Domains: {0,1,2,3,4,5,6,7,8,9}
- Constraints: Alldiff (F,T,U,W,R,O)

$$O + O = R + 10 \cdot C_{10}$$

$$C_{10} + W + W = U + 10 \cdot C_{100}$$

$$C_{100} + T + T = O + 10 \cdot C_{1000}$$
$$C_{1000} = F,\ T \neq 0,\ F \neq 0$$

where C10, C100, and C1000 are auxiliary variables representing the digit carried over into the tens, hundreds, or thousands column.

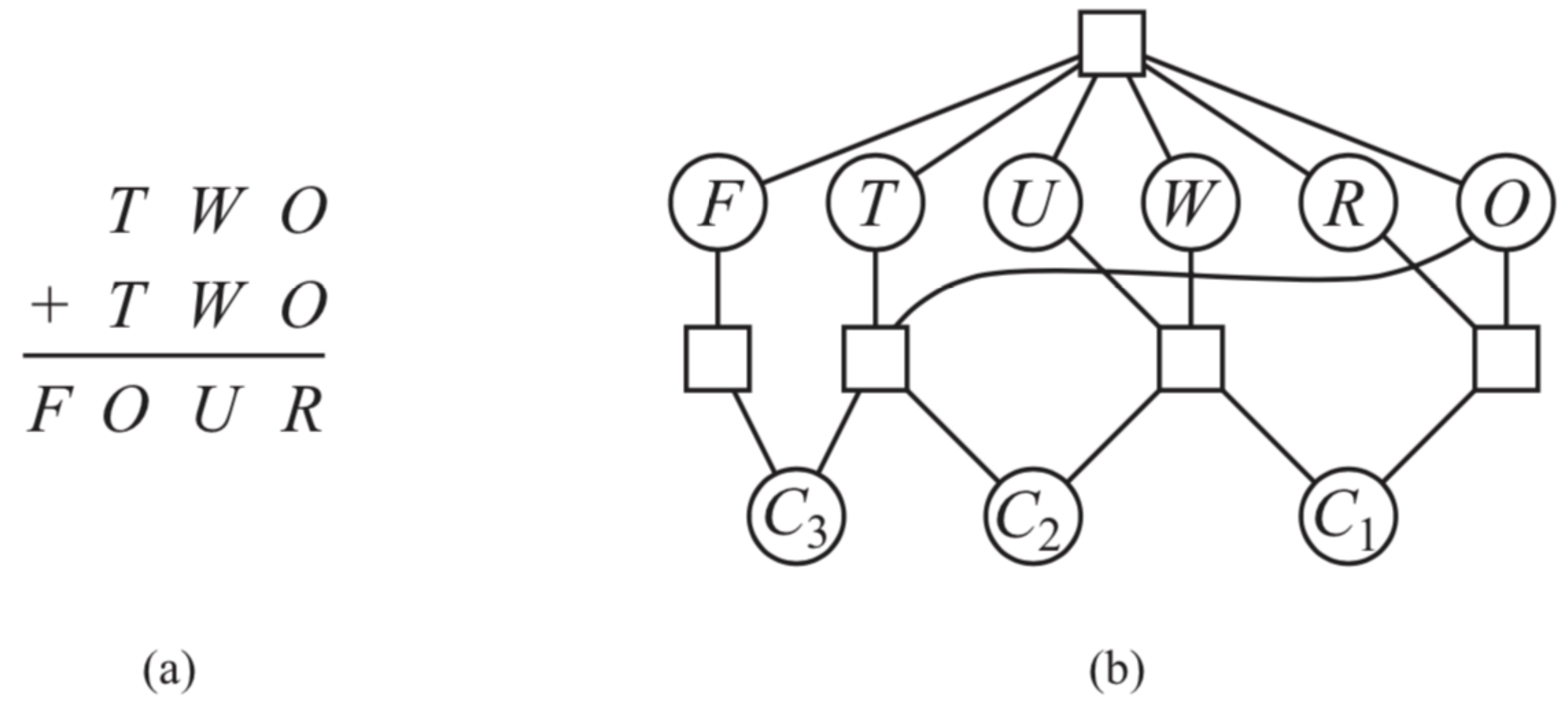


Figure 6.2 (a) A cryptarithmetic problem. Each letter stands for a distinct digit; the aim is to find a substitution of digits for letters such that the resulting sum is arithmetically correct, with the added restriction that no leading zeroes are allowed. (b) The constraint hypergraph for the cryptarithmetic problem, showing the *Alldiff* constraint (square box at the top) as well as the column addition constraints (four square boxes in the middle). The variables C_1, C_2, and C_3 represent the carry digits for the three columns.

One of the most common global constraints is Alldiff, which says that all of the variables involved in the constraint must have different values.

Example: cryptarithmetic problem

- Given a cryptarithmetic problem: **S E N D + M O R E = M O N E Y**

$$
\begin{array}{cccccc}
 & & S & E & N & D \\
+ & & M & O & R & E \\
\hline
 & M & O & N & E & Y
\end{array}
$$

Solution: S E N D + M O R E = M O N E Y

- https://www.youtube.com/watch?v=HC6Y49iTg1k

Character	Code
S	9
E	
N	
D	
M	1
O	0
R	
Y	

Solution: S E N D + M O R E = M O N E Y

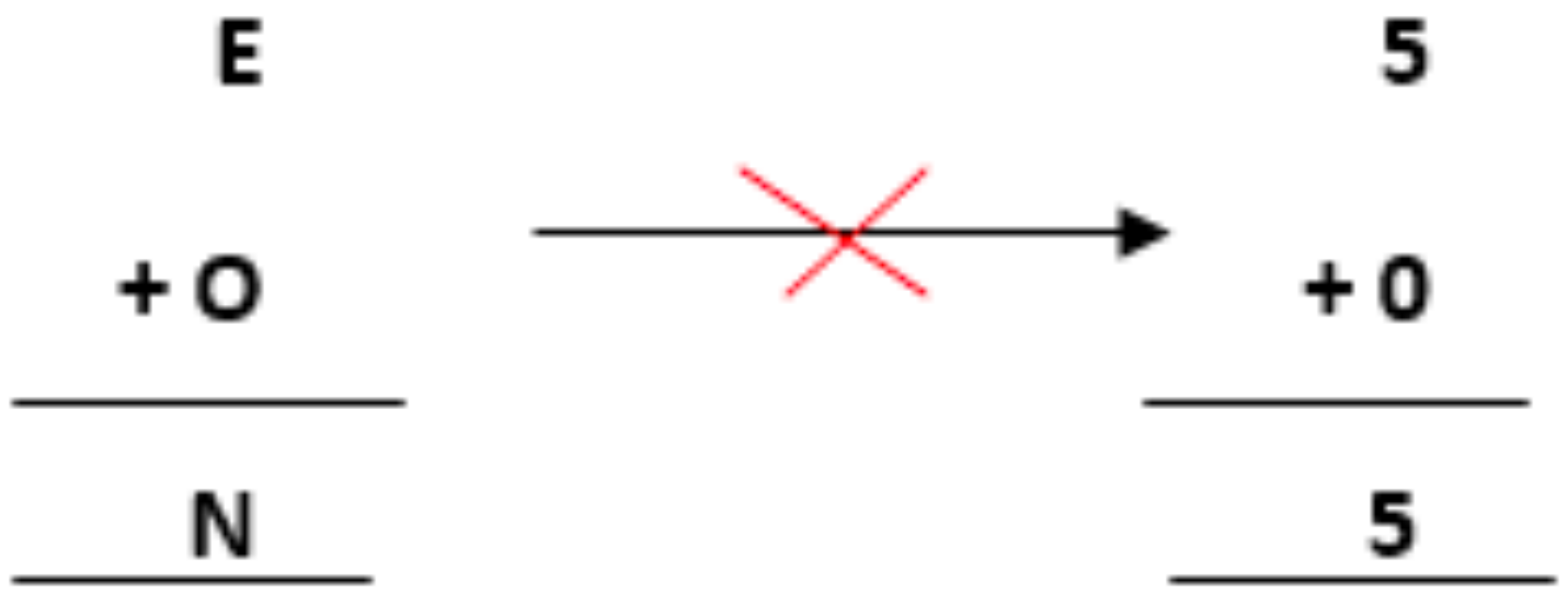
- Starting from the left hand side (L.H.S) , the terms are S and M. Assign a digit which could give a satisfactory result. Let's assign S=9 and M=1.

$$\begin{array}{r} S \\ +M \\ \hline MO \end{array} \longrightarrow \begin{array}{r} 9 \\ +1 \\ \hline 10 \end{array} \qquad\qquad \begin{array}{rr} & SEND \\ + & MORE \\ \hline & MONEY \end{array}$$

- Hence, we get a satisfactory result by adding up the terms and got an assignment for **O** as **O=0** as well.

Solution: S E N D + M O R E = M O N E Y

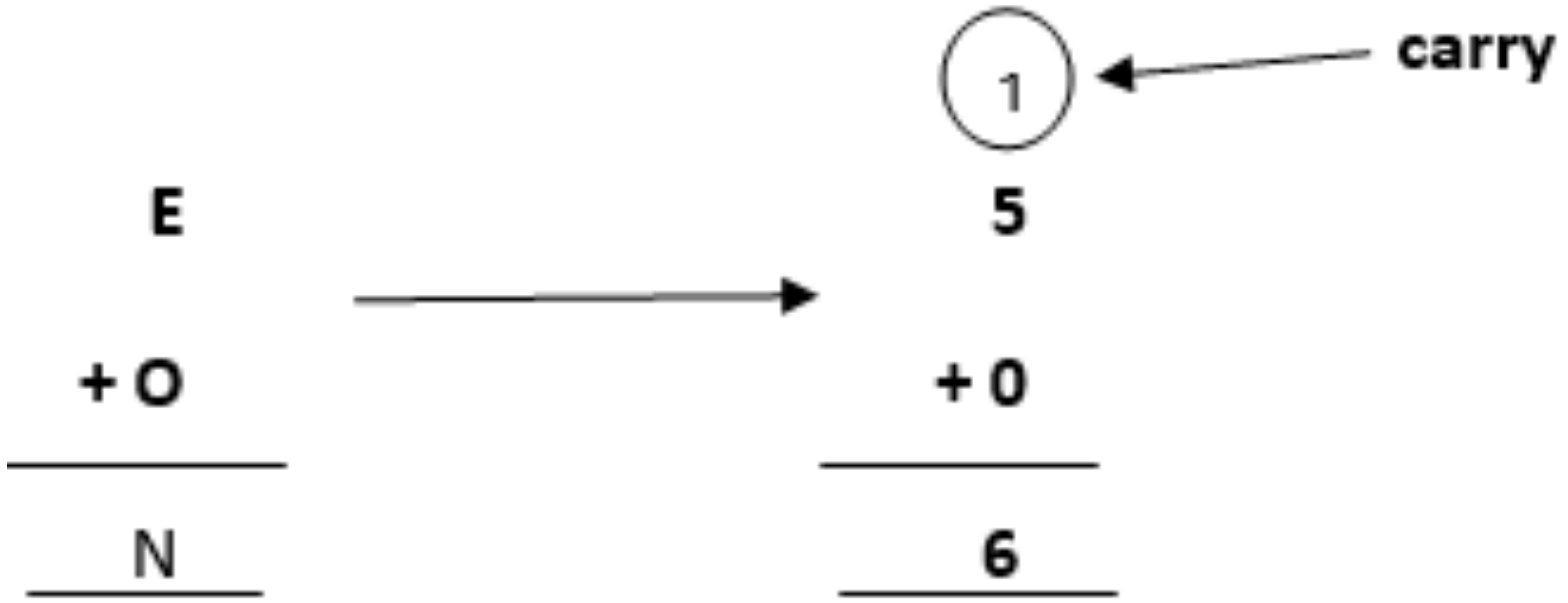
- Now, move ahead to the next terms **E** and **O** to get **N** as its output.



- Hence, **Adding E and O, which means 5+0=0, which is not possible because** according to cryptarithmetic constraints, we cannot assign the same digit to two letters. So, we need to think more and assign some other value.

Solution: S E N D + M O R E = M O N E Y

- At this time, we assume there is a carry from N + R



- Hence, the answer will be satisfied.

Solution: S E N D + M O R E = M O N E Y

- Further, adding the next two terms **N** and **R** (try R = 8 so that N+R produce a carry) we get,

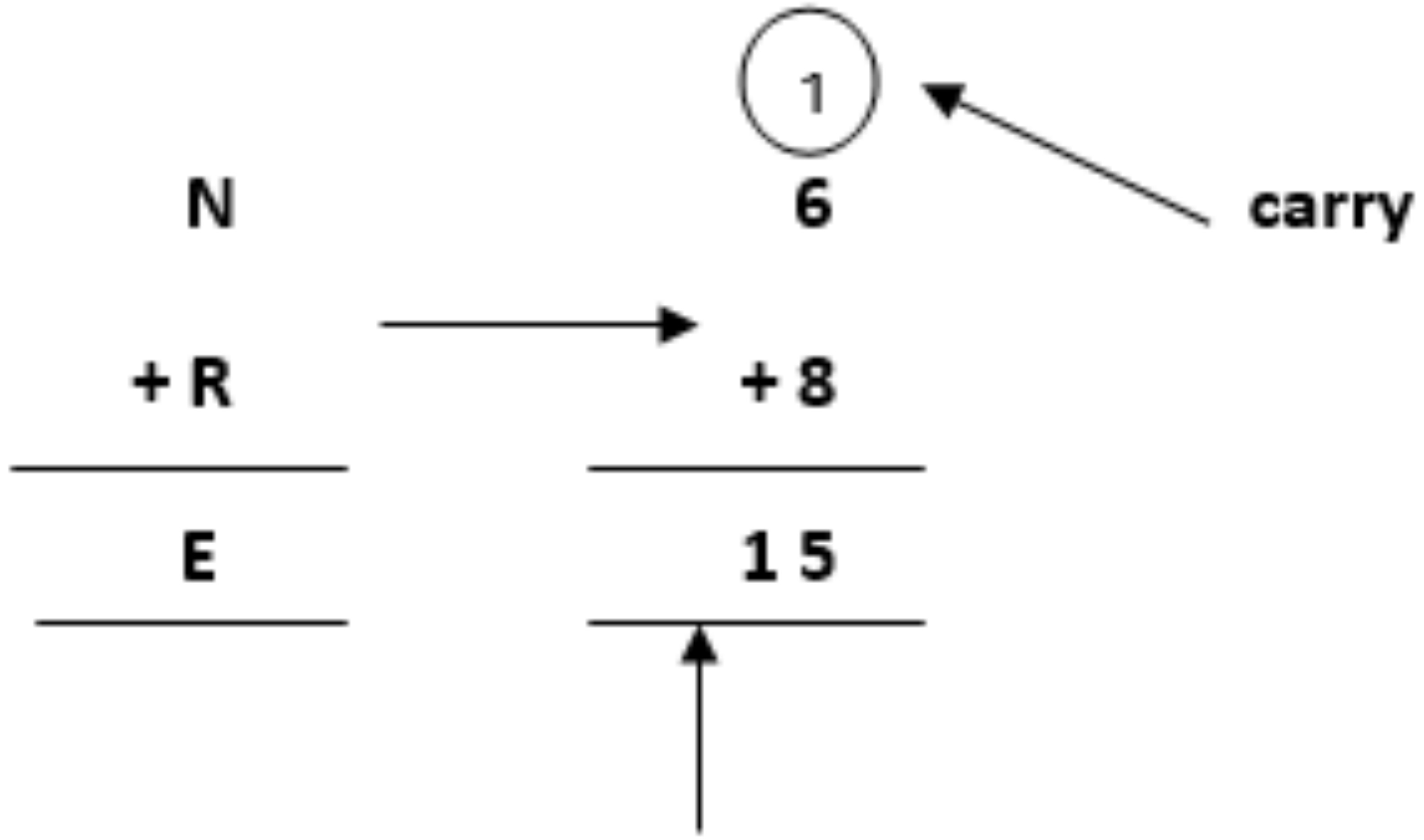
$$\begin{array}{r} N \\ +R \\ \hline E \\ \hline \end{array} \quad \not\longrightarrow \quad \begin{array}{r} 6 \\ +8 \\ \hline 14 \\ \hline \end{array}$$

$$\begin{array}{r} S\,E\,N\,D \\ +\quad M\,O\,R\,E \\ \hline M\,O\,N\,E\,Y \end{array}$$

- But, we have already assigned **E = 5**. Thus, the above result does not satisfy the values.

Solution: S E N D + M O R E = M O N E Y

- Try assuming D + E produces a carry. So,

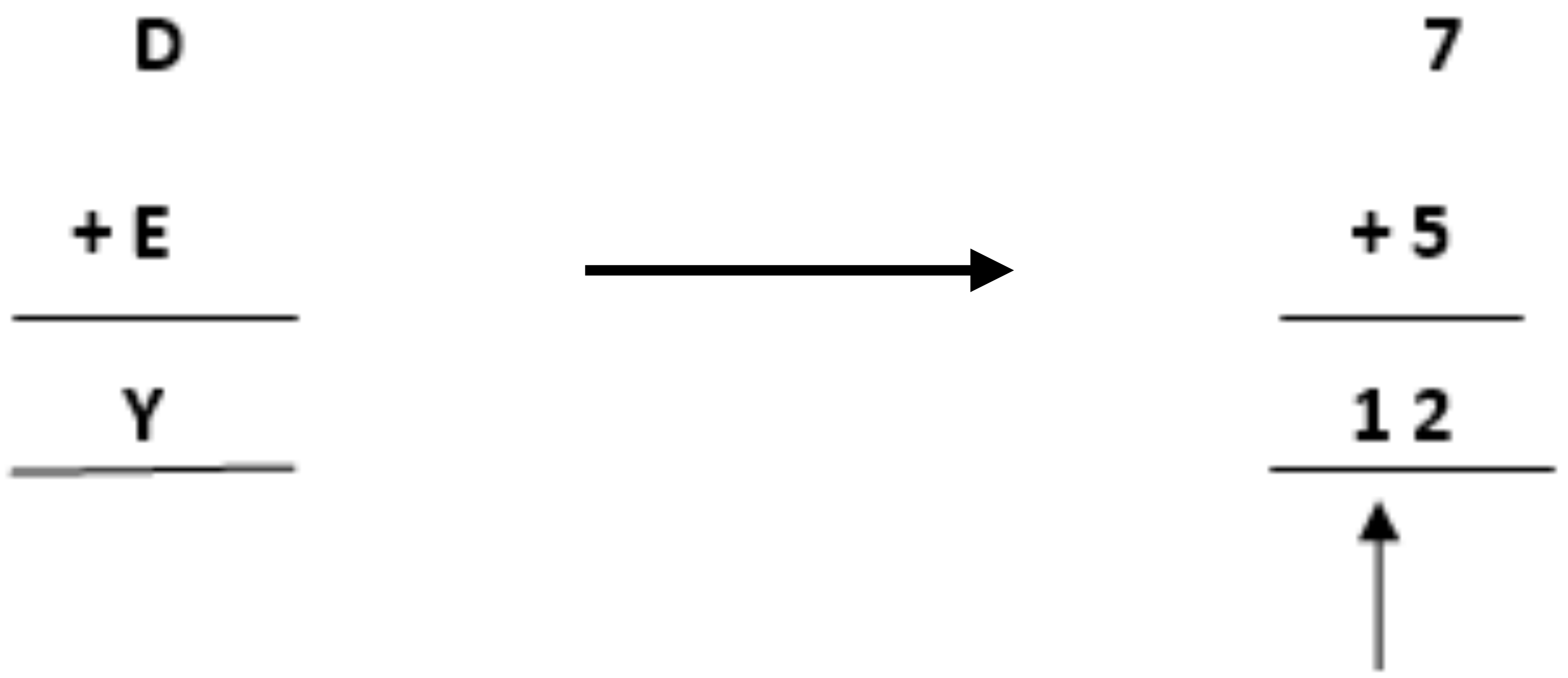



```
  S E N D
+ M O R E
---------
M O N E Y
```

- Hence, R = 8 is satisfied.

Solution: S E N D + M O R E = M O N E Y

- Try on adding the last two terms, i.e., the rightmost terms **D** and **E**, we get **Y** as its result (assuming D = 7 so that D+E will produce a carry).




```
  S E N D
+ M O R E
---------
M O N E Y
```

- Hence, D = 7 is satisfied. All the variables are assigned values by satisfying all constraint. Thus the problem is solved.

Solution: S E N D + M O R E = M O N E Y

- Solution:

S	9
E	5
N	6
D	7
M	1
O	0
R	8
y	2

```
    S E N D
+   M O R E
-----------
  M O N E Y
```

```
    9 5 6 7
  + 1 0 8 5
  ---------
  1 0 6 5 2
```

Practice for Students

1.

```
   B A S E
 + B A L L
 _________
 G A M E S
```

B	7
A	4
S	8
E	3
L	5
G	1
M	9

Practice for Students

2.

$$\begin{array}{r} YOUR \\ +\ YOU \\ \hline HEART \end{array}$$

Y	9
O	4
U	2
R	6
H	1
E	0
A	3
T	8

Practice for Students

3.

$$\begin{array}{r} C\ R\ O\ S\ S \\ R\ O\ A\ D\ S \\ \hline D\ A\ N\ G\ E\ R \end{array} \longrightarrow \begin{array}{r} 96233 \\ 62513 \\ \hline 158746 \end{array}$$

THANK YOU

End of Chapter