Course Topics

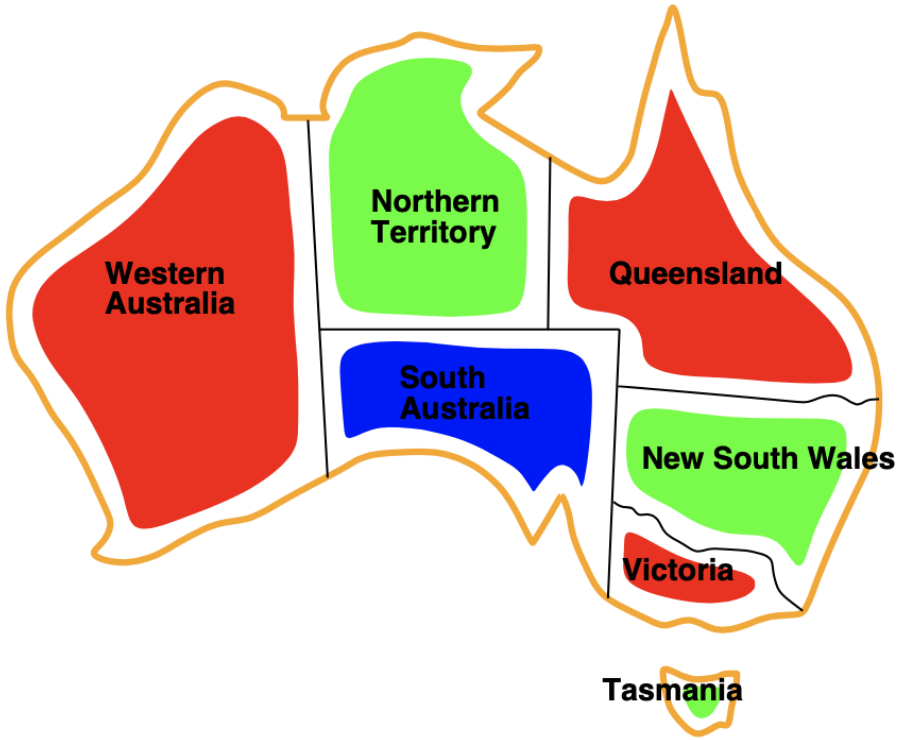
- Intelligent agents (AIMA Ch. 2)
- Search (AIMA Ch. 3, 4, 6, 5)
- Reasoning (AIMA Ch. 7-9, 12-15)
- Machine learning (AIMA Ch. 19-20)
- Deep learning (AIMA Ch. 22)
- Natural language processing (AIMA Ch. 24)
- Generative AI (Optional)

Example: Map Coloring

- Coloring each region either red, green, or blue in such a way that **no two neighboring regions have the same color**

Example: Map Coloring

- Coloring each region either red, green, or blue in such a way that **no two neighboring regions have the same color**



Constraint Satisfaction Problems

L.-Y. Wei

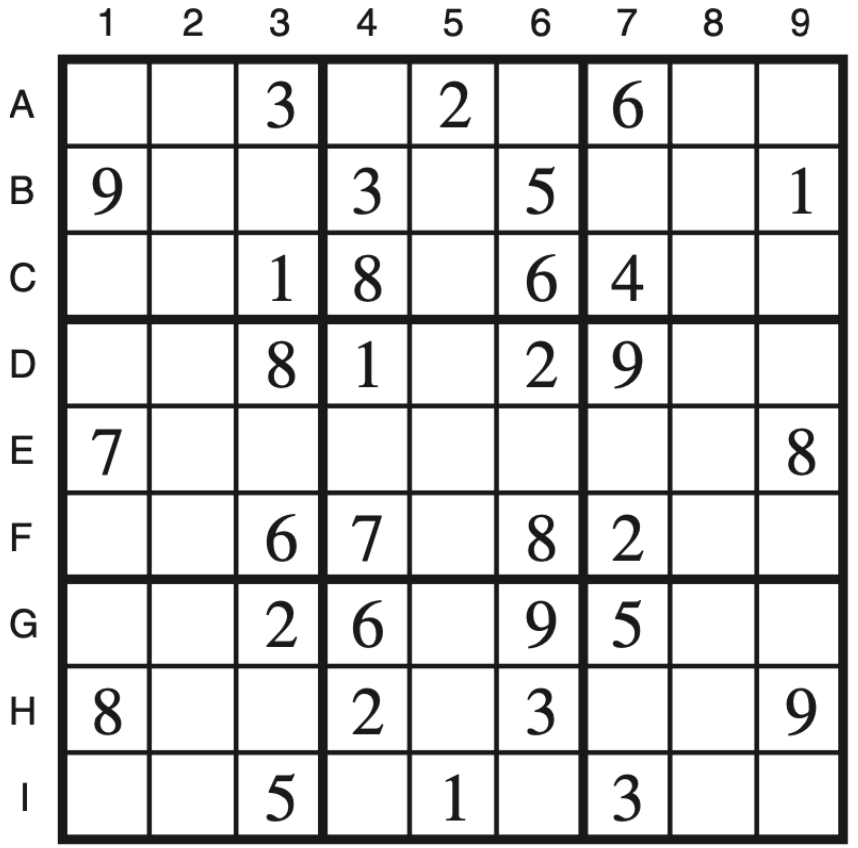
Spring 2024

Constraint Satisfaction Problems (CSPs)

- **State**
  - A set of **variables**, each of which has a **value**
- **Goal test**
  - Each variable has a value that **satisfies all the constraints** on the variable

Example: Sudoku

- A Sudoku board consists of 81 squares, some of which are initially filled with digits from 1 to 9
- Fill in all the remaining squares such that **no digit appears twice in any row, column, or 3x3 box**



	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

Example: Sudoku

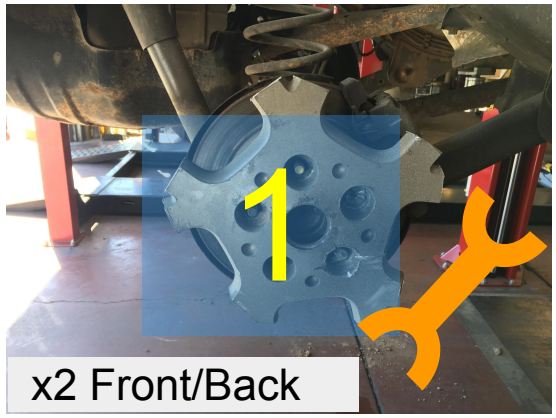
- A Sudoku board consists of 81 squares, some of which are initially filled with digits from 1 to 9
- Fill in all the remaining squares such that **no digit appears twice in any row, column, or 3x3 box**

	1	2	3	4	5	6	7	8	9
A	4	8	3	9	2	1	6	5	7
B	9	6	7	3	4	5	8	2	1
C	2	5	1	8	7	6	4	9	3
D	5	4	8	1	3	2	9	7	6
E	7	2	9	5	6	4	1	3	8
F	1	3	6	7	9	8	2	4	5
G	3	7	2	6	8	9	5	1	4
H	8	1	4	2	5	3	7	6	9
I	6	9	5	4	1	7	3	8	2

Example: Job-Shop Scheduling

Source: https://preview.redd.it/kibs4oby22h21.jpg?auto=webp&s=4d6c60c59c833c7f5f690fdef30bd7f5ab5cb18f
https://qph.cf2.quoracdn.net/main-qimg-77cb54f1511fbe301818b29c18f501ee.webp
https://www.forbes.com/wheels/wp-content/uploads/2021/09/LugNuts_Featured.jpg
http://discussions.texasbowhunter.com/attachment.php?s=2b3d4c66d31aa88b9aaafd9739285027&attachmentid=483370&stc=1&d=1366221409
https://m.media-amazon.com/images/S/aplus-media/vc/f90e29c0-81b3-43b4-905a-4d6e516e3ef6._SR300,300_.jpg

Example: Job-Shop Scheduling



Source: https://preview.redd.it/kibs4oby22h21.jpg?auto=webp&s=4d6c60c59c833c7f5f690fdef30bd7f5ab5cb18f
https://qph.cf2.quoracdn.net/main-qimg-77cb54f1511fbe301818b29c18f501ee.webp
https://www.forbes.com/wheels/wp-content/uploads/2021/09/LugNuts_Featured.jpg
http://discussions.texasbowhunter.com/attachment.php?s=2b3d4c66d31aa88b9aaafd9739285027&attachmentid=483370&stc=1&d=1366221409
https://m.media-amazon.com/images/S/aplus-media/vc/f90e29c0-81b3-43b4-905a-4d6e516e3ef6._SR300,300_.jpg

Example: Cryptarithmetic Puzzle

- Each letter in a cryptarithmetic puzzle represents a different digit

$$\begin{array}{cccc} & T & W & O \\ + & T & W & O \\ \hline F & O & U & R \end{array}$$

Example: Cryptarithmetic Puzzle

- Each letter in a cryptarithmetic puzzle represents a **different digit**

$$\begin{array}{cccc} & T & W & O \\ + & T & W & O \\ \hline F & O & U & R \end{array}$$

Example: Cryptarithmetic Puzzle

- Each letter in a cryptarithmetic puzzle represents a **different digit**

	T	W	O		7	3	4
$+$	T	W	O		7	3	4
F	O	U	R	1	4	6	8

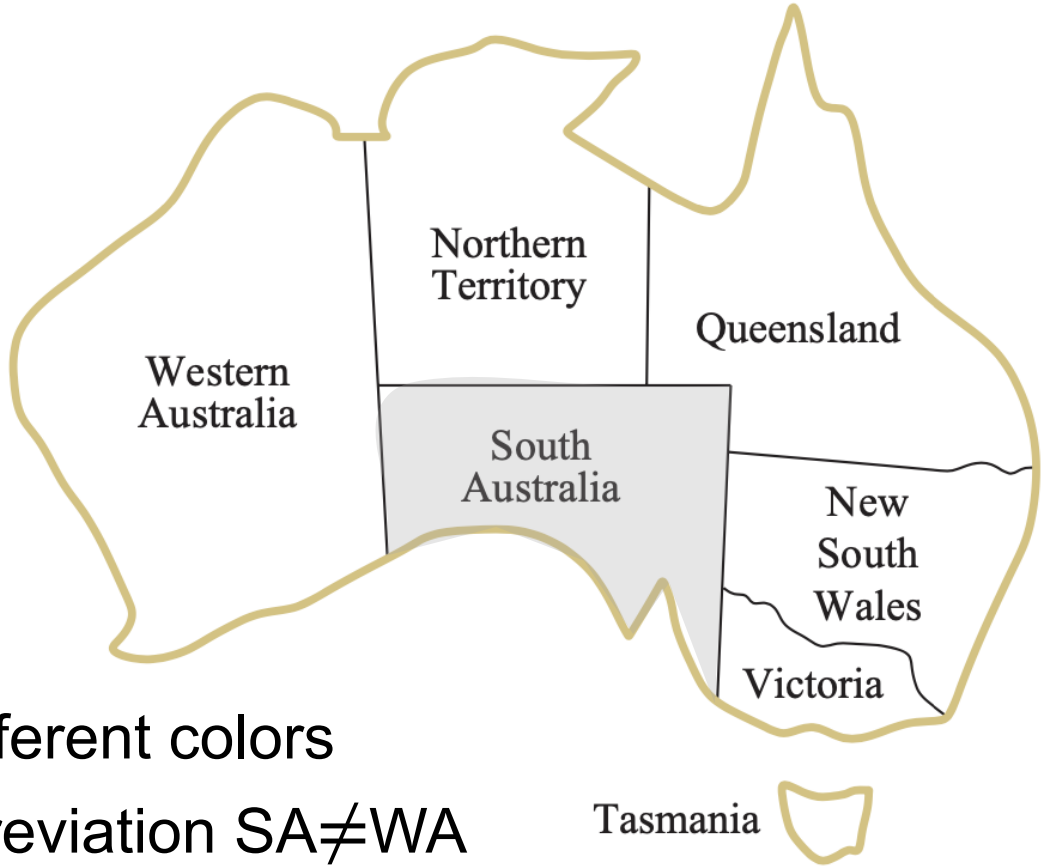
Constraint Satisfaction Problems

- A CSP consists of three components:
  - A set of **variables** $\{X_1, X_2, \ldots, X_n\}$
  - A set of **domains** $\{D_1, D_2, \ldots, D_n\}$
    - A domain, D_i, consists of a set of allowable values, $\{v_1,...,v_k\}$, for variable X_i
  - A set of **constraints**
    - Each constraint C_j consists of a pair <*scope*,*rel*>
      - *scope*: a tuples of variables that participate in the constraint
      - *rel*: a relation that defines the values that those variables can take on

Example: Map Coloring

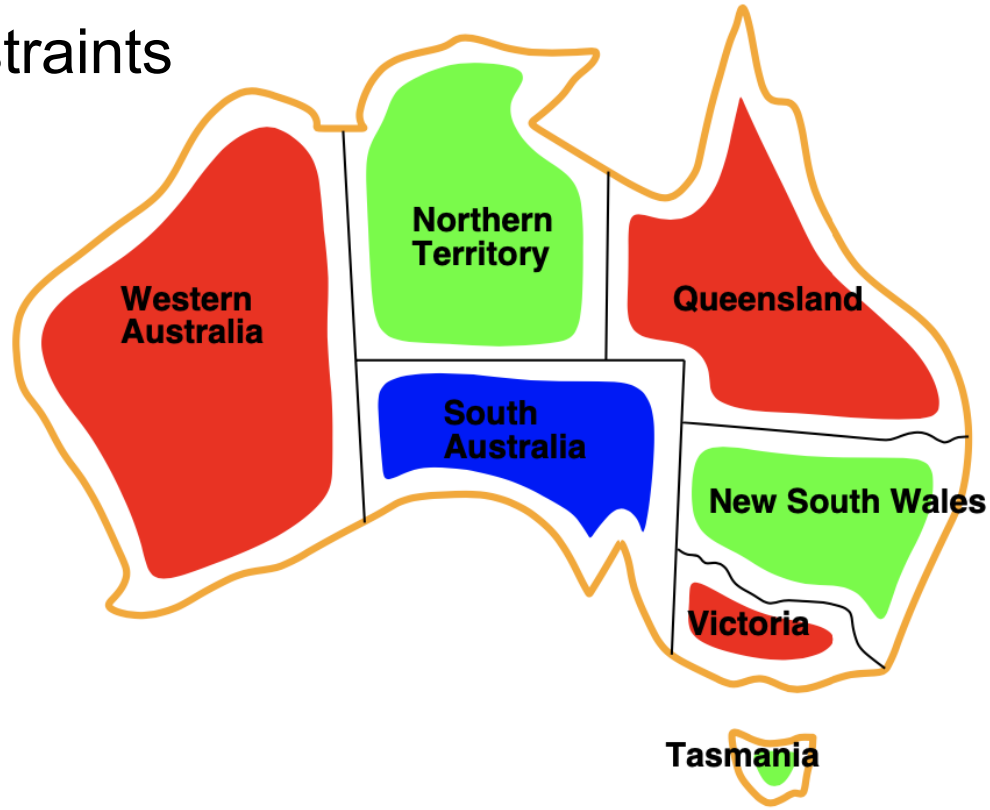
- Variables
  - {WA, NT, SA, Q, NSW, V, T}
- Domains
  - D_i = {red,green,blue}
- Constraints
  - Adjacent regions must have different colors
  - e.g., <(SA,WA),SA≠WA>: abbreviation SA≠WA

$$\{SA \neq WA, SA \neq NT, SA \neq Q, SA \neq NSW, SA \neq V,$$
$$WA \neq NT, NT \neq Q, Q \neq NSW, NSW \neq V\}.$$



Example: Map Coloring

- Solutions
  - Assignments satisfying all constraints
  - e.g., {WA=red,
    NT=green,
    SA=blue,
    Q=red,
    NSW=green,
    V=red,
    T=green}



Example: Sudoku

- Variables
  - {A1,A2,....,I8,I9}
- Domains
  - Empty squares: {1,2,...,9}
  - Pre-filled squares: a single value
- Constraints

rows
$$Alldiff(A1,A2,A3,A4,A5,A6,A7,A8,A9)$$
$$Alldiff(B1,B2,B3,B4,B5,B6,B7,B8,B9)$$
$$\ldots$$

columns
$$Alldiff(A1,B1,C1,D1,E1,F1,G1,H1,I1)$$
$$Alldiff(A2,B2,C2,D2,E2,F2,G2,H2,I2)$$
$$\ldots$$

3x3 boxes
$$Alldiff(A1,A2,A3,B1,B2,B3,C1,C2,C3)$$
$$Alldiff(A4,A5,A6,B4,B5,B6,C4,C5,C6)$$
$$\ldots$$

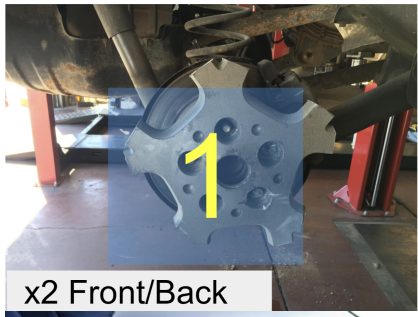
	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

Example: Job-Shop Scheduling

- Variables

$$\{Axle_F, Axle_B, Wheel_{RF}, Wheel_{LF}, Wheel_{RB}, Wheel_{LB}, Nuts_{RF},$$
$$Nuts_{LF}, Nuts_{RB}, Nuts_{LB}, Cap_{RF}, Cap_{LF}, Cap_{RB}, Cap_{LB}, Inspect\}$$

  where the value of each variable is the time that **the task starts**, expressed as **an integer number of minutes**



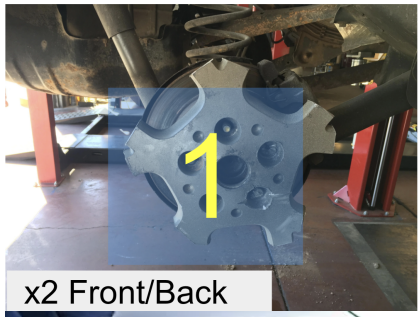
Source: https://qph.cf2.quoracdn.net/main-qimg-77cb54f1511fbe301818b29c18f501ee.webp
https://www.forbes.com/wheels/wp-content/uploads/2021/09/LugNuts_Featured.jpg
http://discussions.texasbowhunter.com/attachment.php?s=2b3d4c66d31aa88b9aaafd9739285027&attachmentid=483370&stc=1&d=1366221409
https://m.media-amazon.com/images/S/aplus-media/vc/f90e29c0-81b3-43b4-905a-4d6e516e3ef6._SR300,300_.jpg

Example: Job-Shop Scheduling

- Constraints
  - Precedence constraints

$$Axle_F + 10 \leq Wheel_{RF};\ Wheel_{RF} + 1 \leq Nuts_{RF};\ Nuts_{RF} + 2 \leq Cap_{RF};$$
$$Axle_F + 10 \leq Wheel_{LF};\ Wheel_{LF} + 1 \leq Nuts_{LF};\ Nuts_{LF} + 2 \leq Cap_{LF};$$
$$Axle_B + 10 \leq Wheel_{RB};\ Wheel_{RB} + 1 \leq Nuts_{RB};\ Nuts_{RB} + 2 \leq Cap_{RB};$$
$$Axle_B + 10 \leq Wheel_{LB}.\ Wheel_{LB} + 1 \leq Nuts_{LB};\ Nuts_{LB} + 2 \leq Cap_{LB}.$$

$X + d_X \leq Inspect$ for every variable X



x2 Front/Back

x4 RF,LF,RB,LB

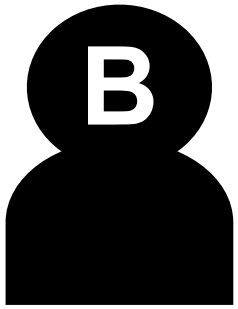
x4 RF,LF,RB,LB

x4 RF,LF,RB,LB

Source: https://qph.cf2.quoracdn.net/main-qimg-77cb54f1511fbe301818b29c18f501ee.webp
https://www.forbes.com/wheels/wp-content/uploads/2021/09/LugNuts_Featured.jpg
http://discussions.texasbowhunter.com/attachment.php?s=2b3d4c66d31aa88b9aaafd9739285027&attachmentid=483370&stc=1&d=1366221409
https://m.media-amazon.com/images/S/aplus-media/vc/f90e29c0-81b3-43b4-905a-4d6e516e3ef6._SR300,300_.jpg

Example: Job-Shop Scheduling

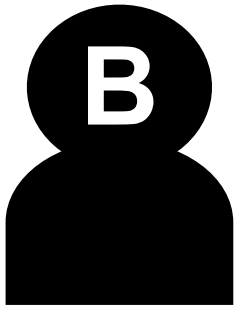
Get the whole assembly done in 30 minutes!



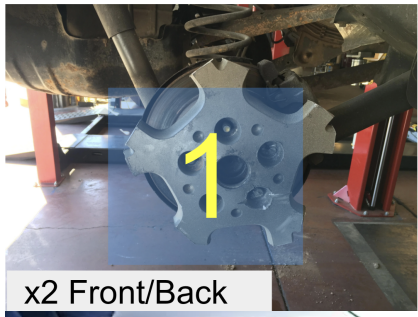
Source: https://qph.cf2.quoracdn.net/main-qimg-77cb54f1511fbe301818b29c18f501ee.webp
https://www.forbes.com/wheels/wp-content/uploads/2021/09/LugNuts_Featured.jpg
http://discussions.texasbowhunter.com/attachment.php?s=2b3d4c66d31aa88b9aaafd9739285027&attachmentid=483370&stc=1&d=1366221409
https://m.media-amazon.com/images/S/aplus-media/vc/f90e29c0-81b3-43b4-905a-4d6e516e3ef6._SR300,300_.jpg

Example: Job-Shop Scheduling

Get the whole assembly done in 30 minutes!



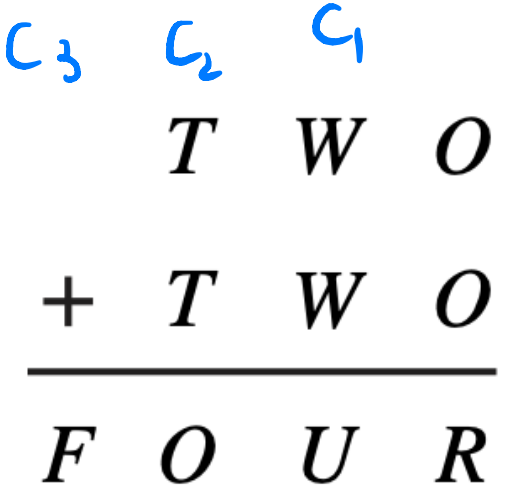
- Domain of all variables
  - $D_i = \{0,1, 2, 3, ..., 30\}$



Source: https://qph.cf2.quoracdn.net/main-qimg-77cb54f1511fbe301818b29c18f501ee.webp
https://www.forbes.com/wheels/wp-content/uploads/2021/09/LugNuts_Featured.jpg
http://discussions.texasbowhunter.com/attachment.php?s=2b3d4c66d31aa88b9aaafd9739285027&attachmentid=483370&stc=1&d=1366221409
https://m.media-amazon.com/images/S/aplus-media/vc/f90e29c0-81b3-43b4-905a-4d6e516e3ef6._SR300,300_.jpg

Example: Cryptarithmetic Puzzle

- Variables
  - {F, T, U, W, R, O, C_1, C_2, C_3}
- Domains
  - {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
- Constraints
  - Alldiff(F,T,U,W,R,O)
  - $O + O = R + 10 \cdot C_1$
  - $C_1 + W + W = U + 10 \cdot C_2$
  - $C_2 + T + T = O + 10 \cdot C_3$
  - $C_3 = F$



Varieties of CSPs

- Discrete variables
  - Finite domains
    - e.g., map-coloring problems, scheduling with time limits, and 8-queens problem
  - Infinite domains
    - e.g., a set of integers/strings etc.
    - e.g., job scheduling problem without a deadline
      - Variables are start/end days for each job
      - Implicit constraint: $\text{StartJob}_1 + 5 \leq \text{StartJob}_3$
- Continuous variables
  - e.g., linear programming problems

Varieties of Constraints

- **Unary constraint** restricts the value of a single variable
  - e.g., SA ≠ green
- **Binary constraint** relates two variable
  - e.g., SA ≠ WA
- High-order constraints involve 3 or more variables

Varieties of Constraints (Cont.)

- **Preference constraint**
  - e.g., in a university class-scheduling problem, Prof. M might prefer teaching in the morning, whereas Prof. A prefers teaching in the afternoon
  - Can be encoded as costs on individual variable assignments
    - e.g., assigning an afternoon slot for Prof. M costs 2 points and assigning a morning slot costs 1.
    - CSPs with preferences can be solved with optimization search methods
      (**constrained optimization problems**, COPs)

Node Consistency

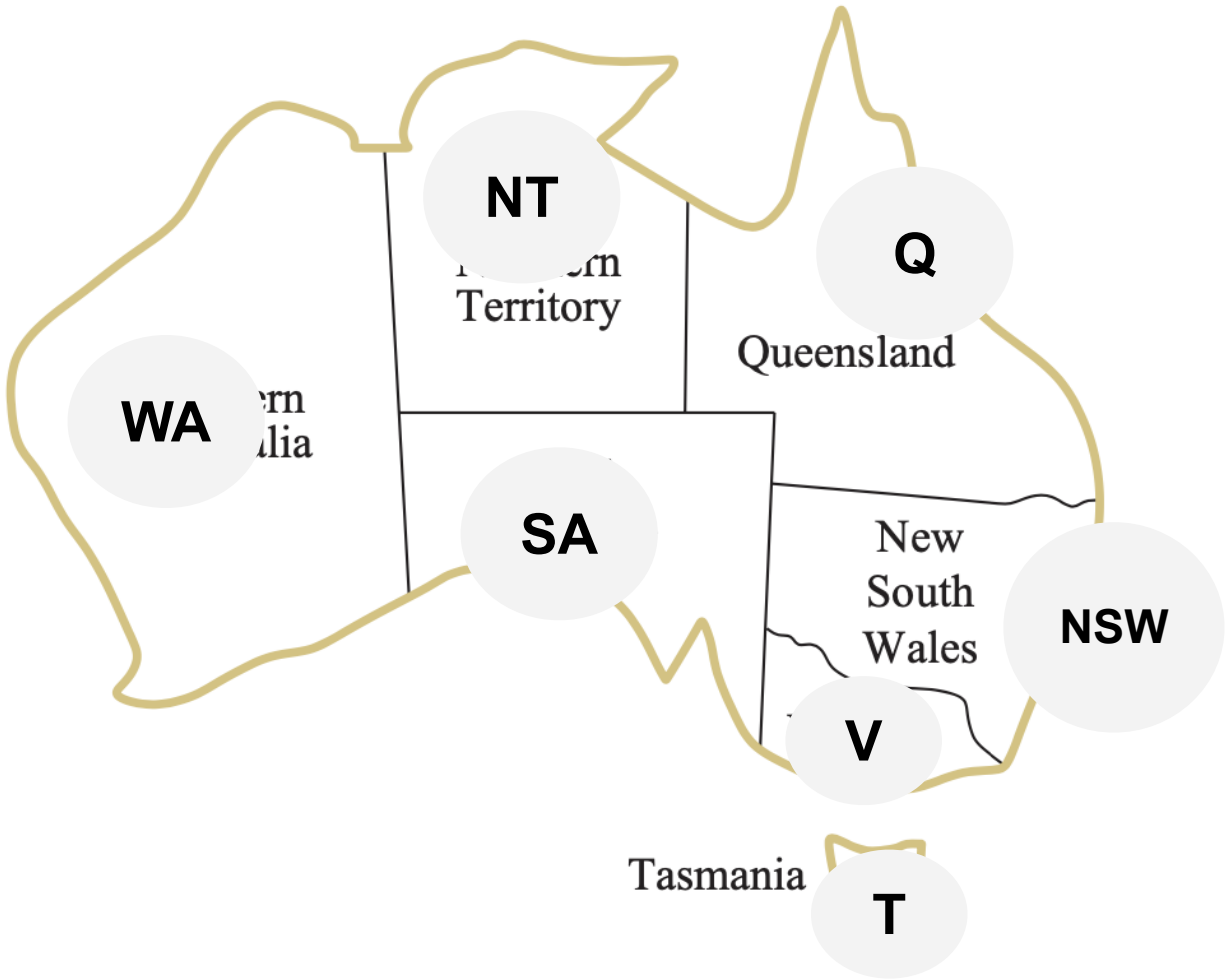
- A single variable (corresponding to a node in the CSP graph) is **node-consistent** if all the values in the variable's domain satisfy the variable's unary constraints
- Example
  - The variable SA starts with domain {red, green, blue} but a constraint {SA ≠ green}
  - Make it node-consistent by eliminating "green", i.e., SA with the reduced domain {red, blue}
- A graph is node-consistent if every variable in the graph is node-consistent

Binary CSP

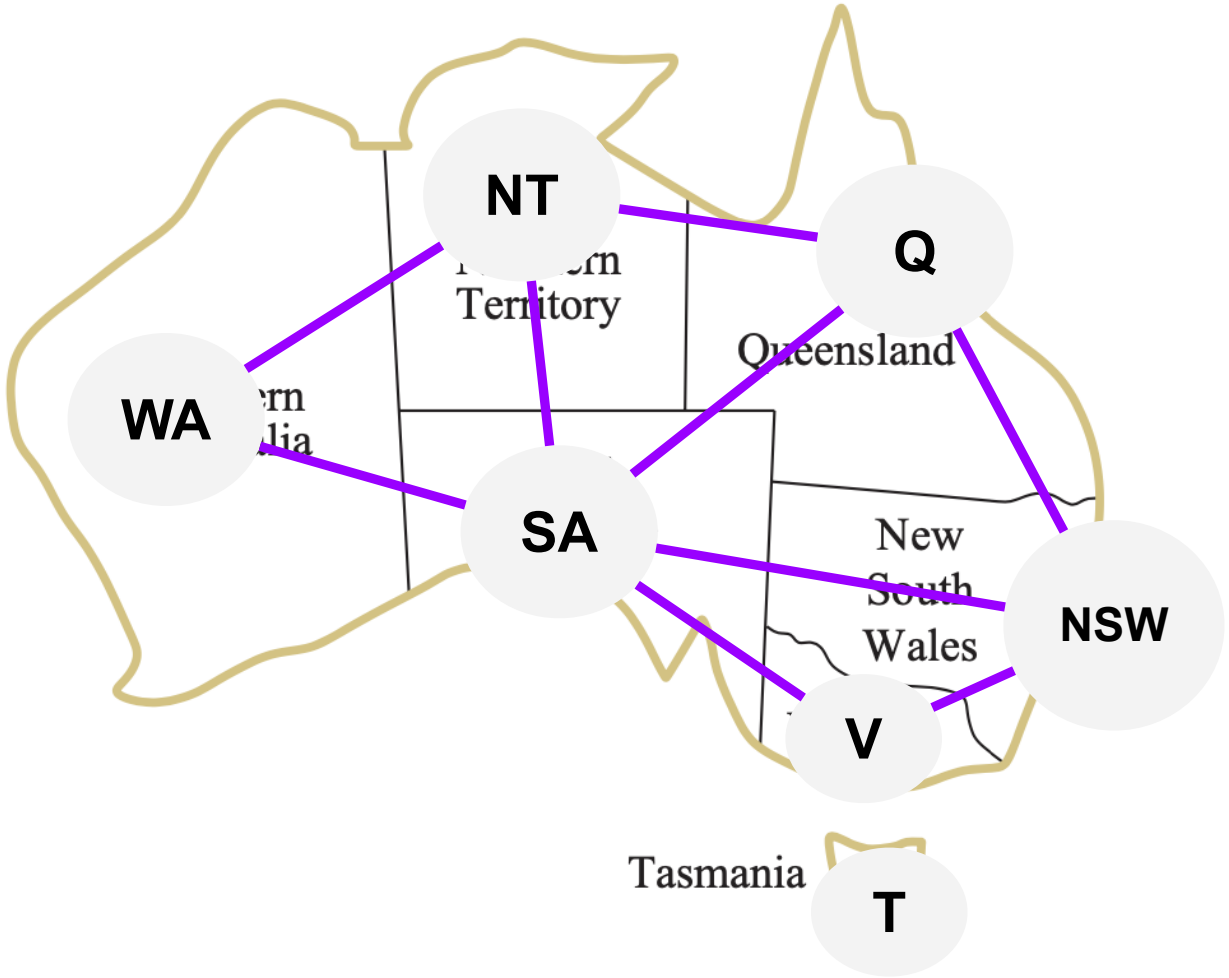
- Constraint types of binary CSPs
  - Unary constraints
  - Binary constraints
- Binary CSP can be represented as a constraint graph
- **Constraint graph**
  - Nodes are variables of the CSP
  - Edges show constraints
    - Edges connect any two variables that participate in a constraint

Constraint Graph of Map Coloring Problem

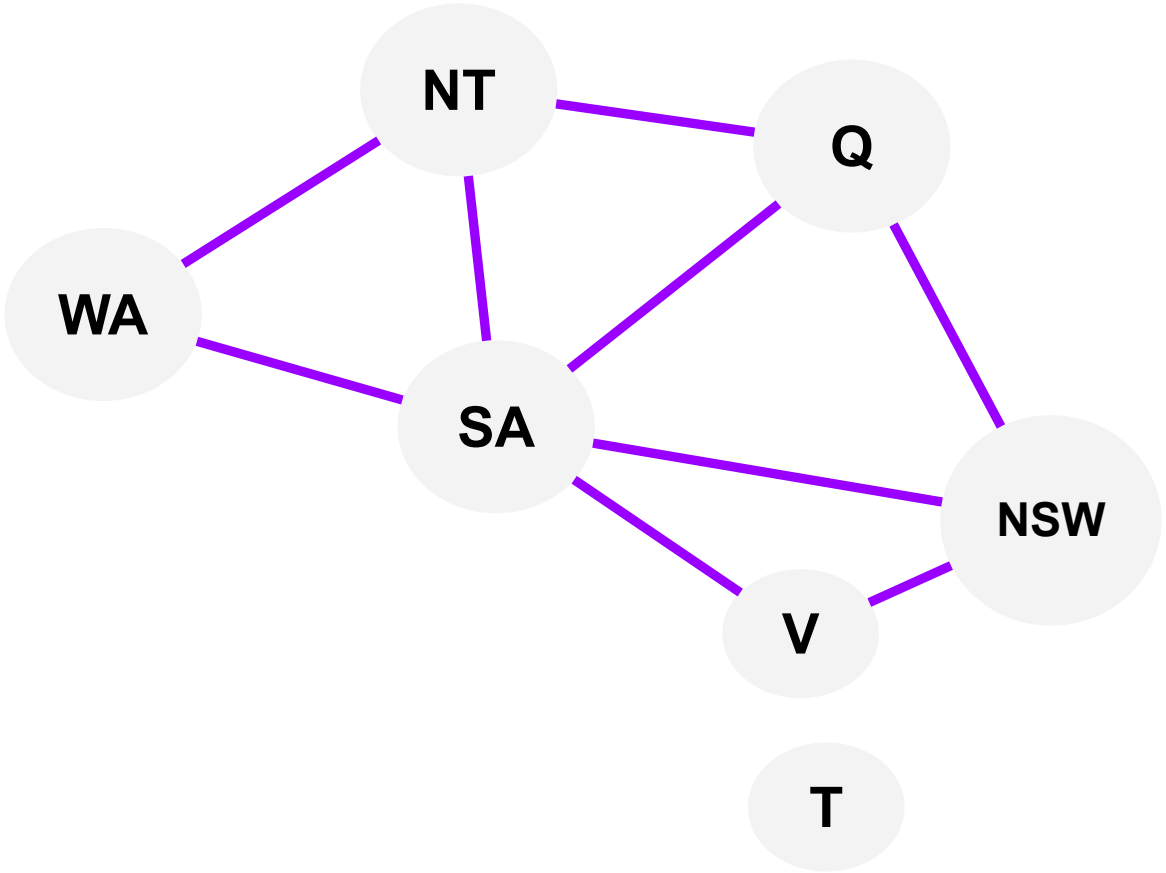
Constraint Graph of Map Coloring Problem



Constraint Graph of Map Coloring Problem



Constraint Graph of Map Coloring Problem



Constraint Hypergraph of Cryptarithmetic Puzzle Problem

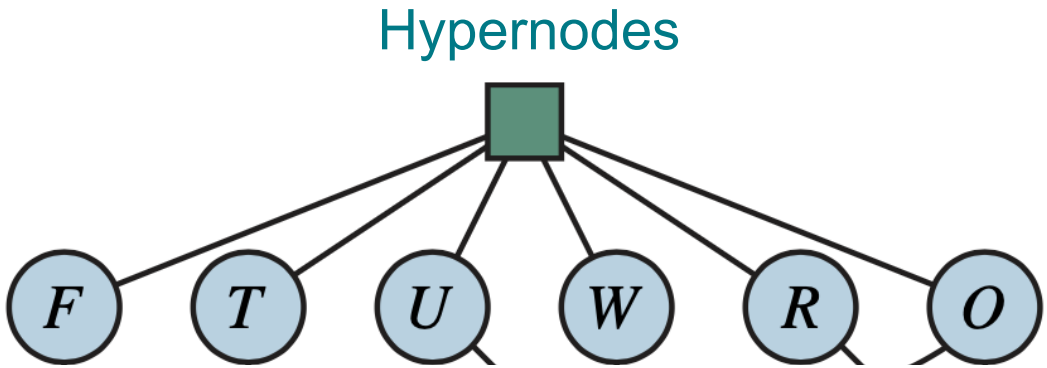
- Variables
  - {F, T, U, W, R, O, C_1, C_2, C_3}
- Domains
  - {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
- Constraints
  - Alldiff(F, T, U, W, R, O)
  - $O + O = R + 10 \cdot C_1$
  - $C_1 + W + W = U + 10 \cdot C_2$
  - $C_2 + T + T = O + 10 \cdot C_3$
  - $C_3 = F$

high-order constraints

Ordinary Nodes

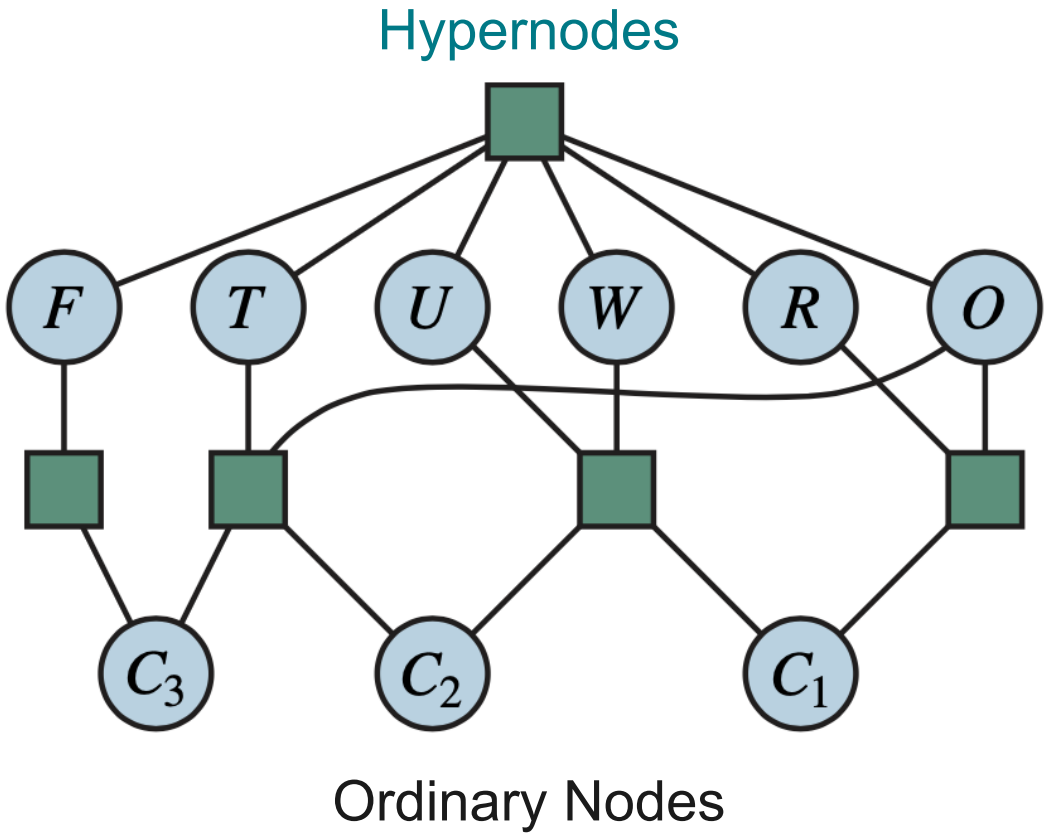
Constraint Hypergraph of Cryptarithmetic Puzzle Problem

- Variables
  - {F, T, U, W, R, O, C_1, C_2, C_3}
- Domains
  - {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
- Constraints
  - Alldiff(F, T, U, W, R, O)
  - $O + O = R + 10 \cdot C_1$
  - $C_1 + W + W = U + 10 \cdot C_2$
  - $C_2 + T + T = O + 10 \cdot C_3$
  - $C_3 = F$



Constraint Hypergraph of Cryptarithmetic Puzzle Problem

- Variables
  - {F, T, U, W, R, O, C_1, C_2, C_3}
- Domains
  - {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
- Constraints
  - Alldiff(F, T, U, W, R, O)
  - $O + O = R + 10 \cdot C_1$
  - $C_1 + W + W = U + 10 \cdot C_2$
  - $C_2 + T + T = O + 10 \cdot C_3$
  - $C_3 = F$



Real-World CSPs

- Scheduling problems
  - e.g., when can we all meet?
- Timetabling problem
  - e.g., which class is offered when and where?
- Assignment problems
  - e.g., who teaches what class
- Transportation scheduling
- Circuit layout
- Factory scheduling
- …

Search Problem

State space

Goal

Actions

Transition model

Path cost

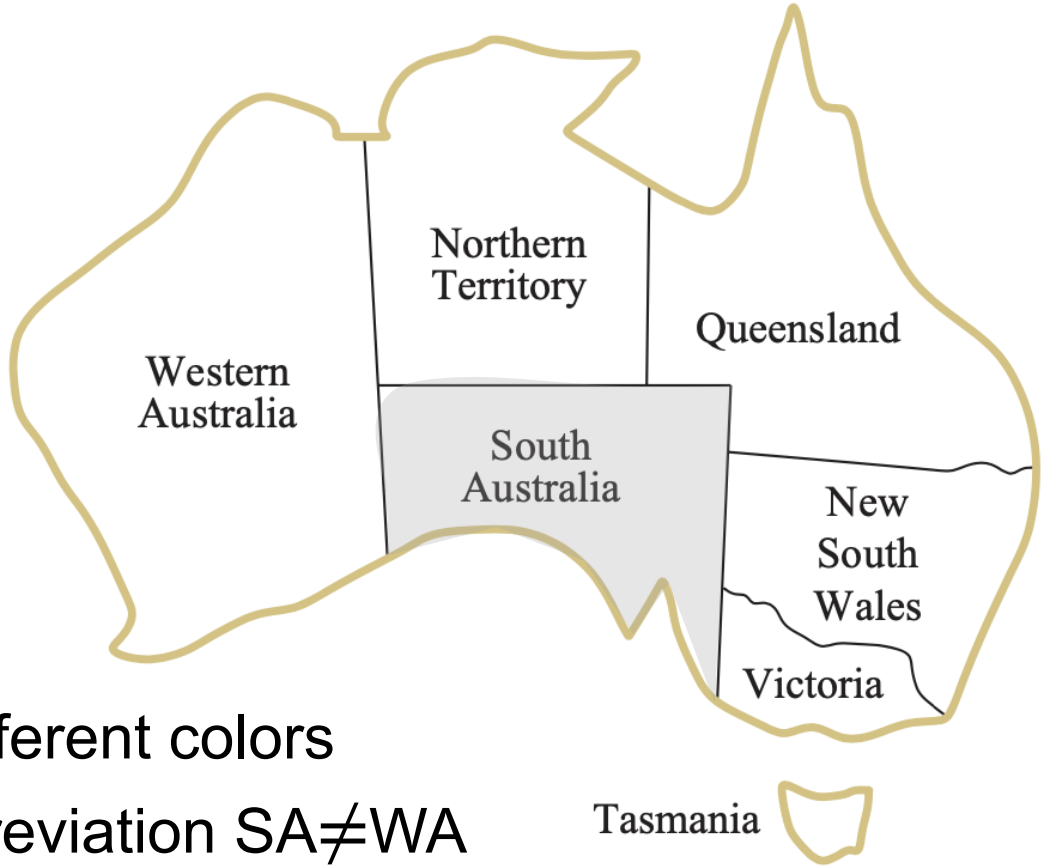
CSPs as Search Problems

- Initial state
  - The empty assignment, {}
- States
  - Defined by the values assigned so far (partial assignments, i.e., some variables unassigned)
- Actions
  - Add a {variable = value} to assignment
- Transition model
  - Show how adding an assignment changes the assignment
- Goal test
  - Check the current assignment is **complete** and **consistent**
    - A complete assignment is one in which every variable is assigned a value
    - An assignment that does not violate any constraints is called a consistent or legal assignment
- Path cost
  - All paths have same cost

Example: Map Coloring

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

$$\{SA \neq WA, SA \neq NT, SA \neq Q, SA \neq NSW, SA \neq V,$$
$$WA \neq NT, NT \neq Q, Q \neq NSW, NSW \neq V\}.$$



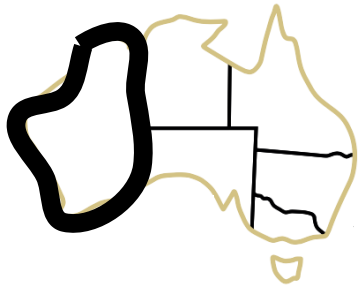
Search Methods

- What would BFS do?
- What would DFS do?

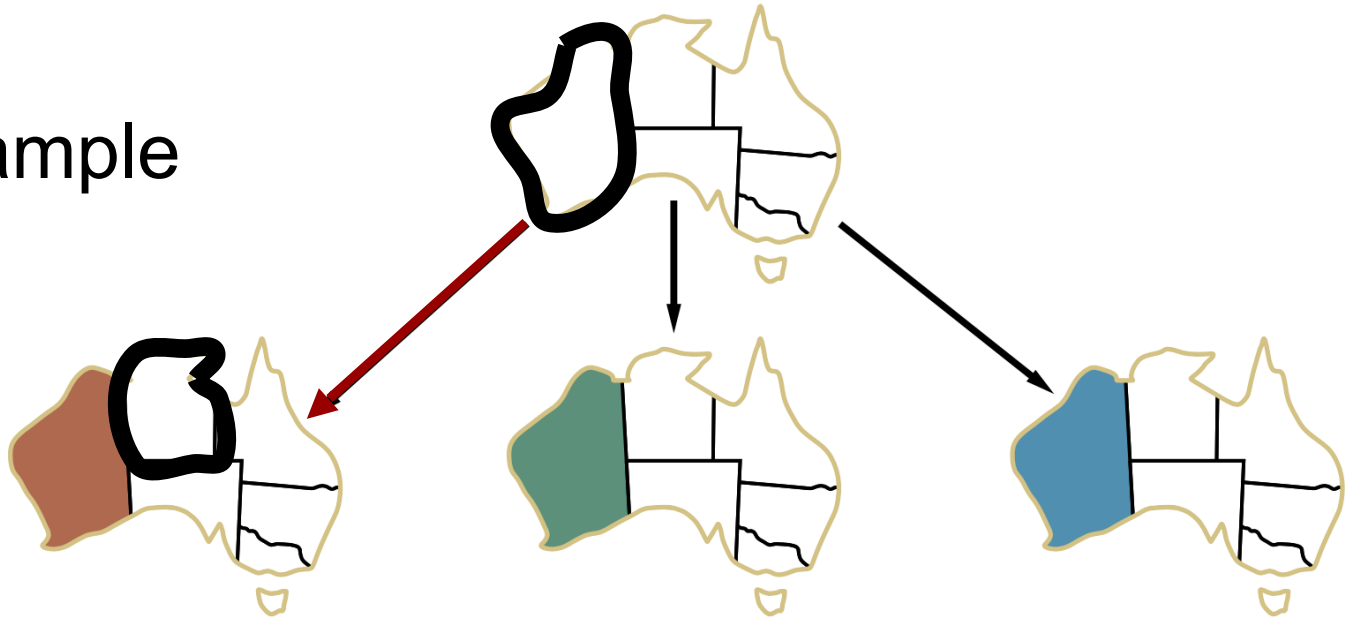
Backtracking Search for CSPs

- Backtracking search is the basic uninformed algorithm for solving CSPs
- Idea 1: One variable at a time
  - CSP is commutative
    - If a problem is **commutative** if the order of application of any given set of actions does not matter
    - e.g., [WA=red then NT=green] same as [NT=green and WA=red]
  - Consider assignments to a single variable at each step
- Idea 2: Check constraints as you go
  - Consider only values which do not conflict with previous assignments
  - Might have to do some computation to check the constraints
- Depth-first search with these two improvements is called **backtracking search**

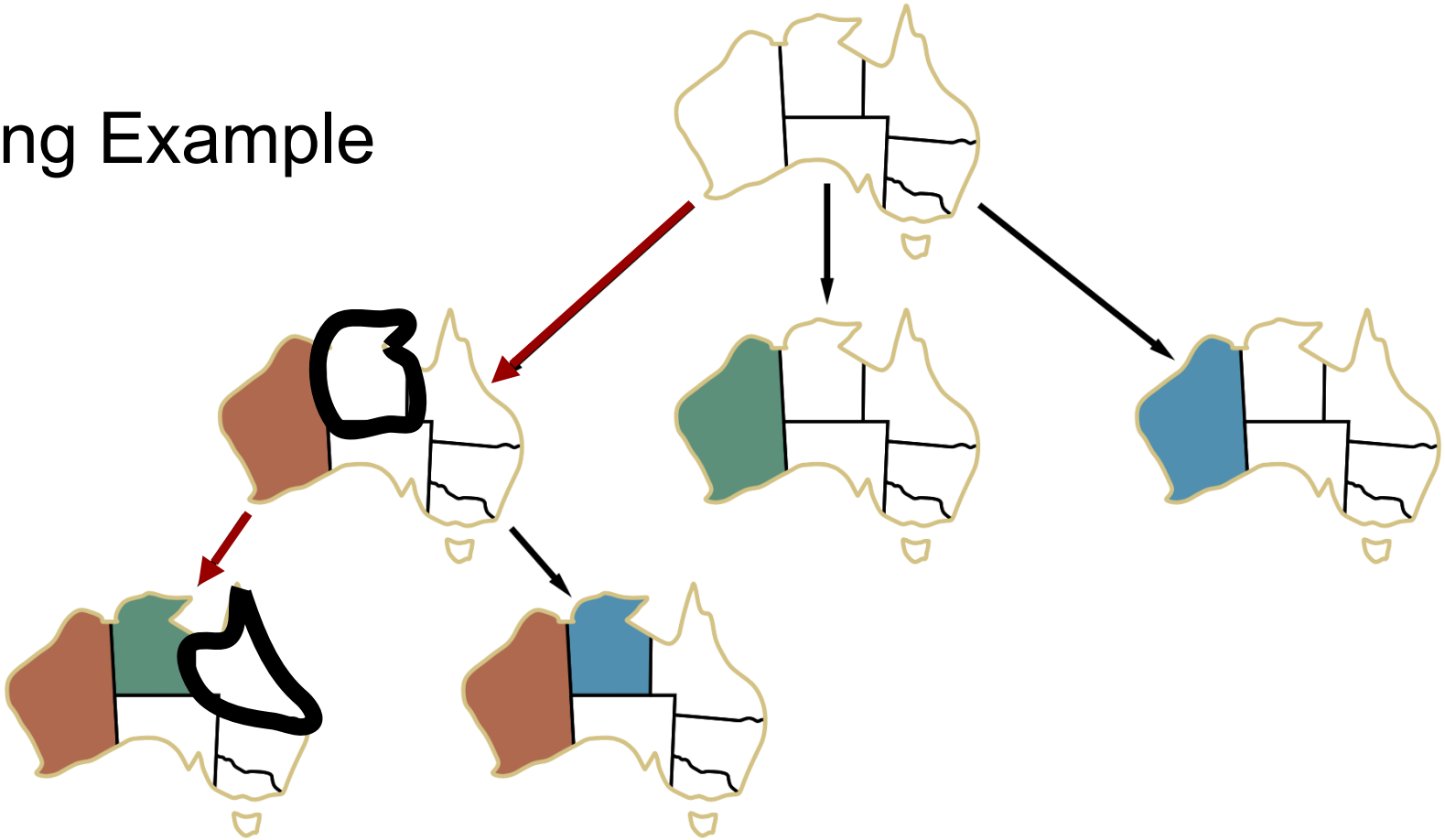
Backtracking Example



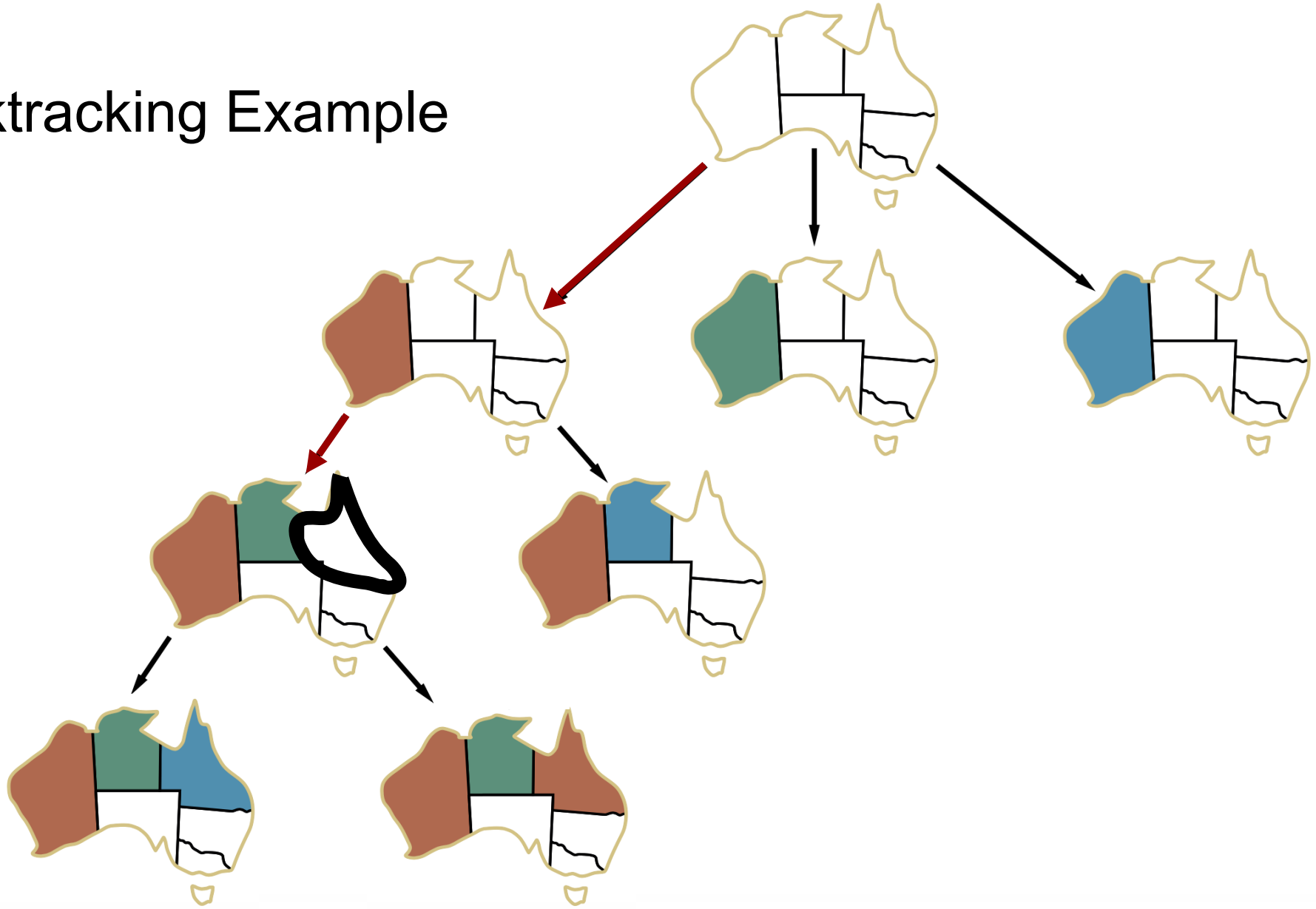
Backtracking Example



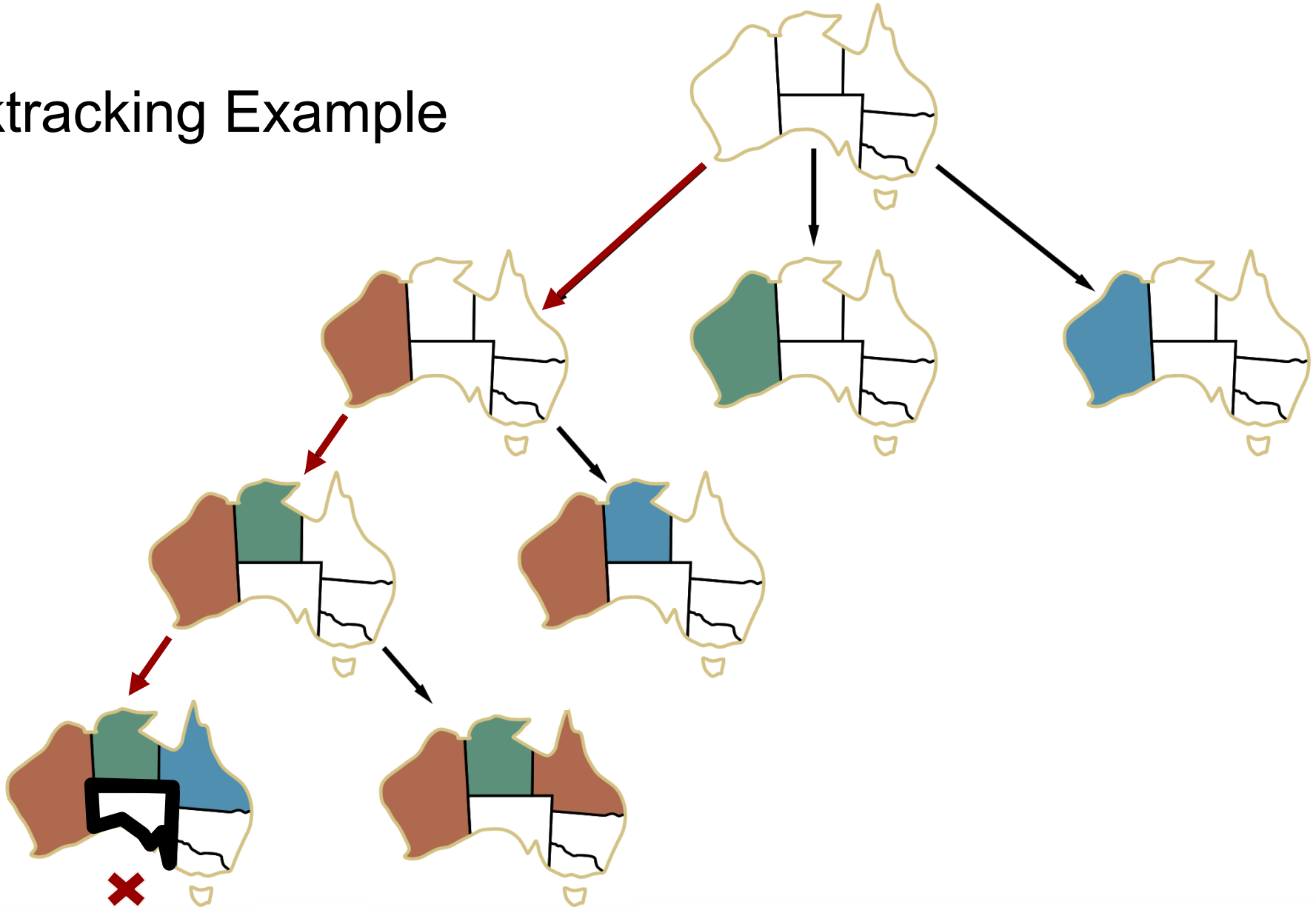
Backtracking Example



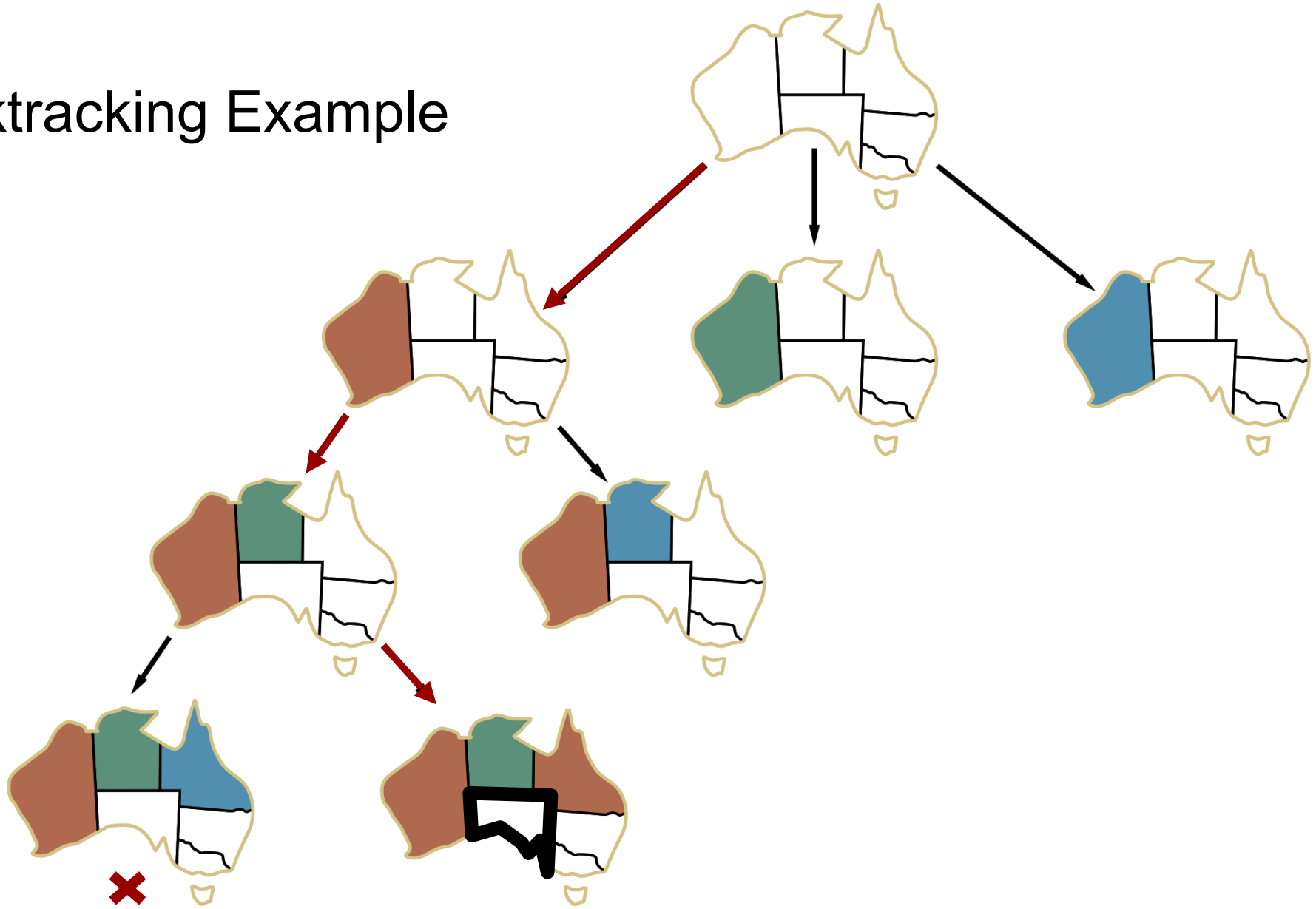
Backtracking Example



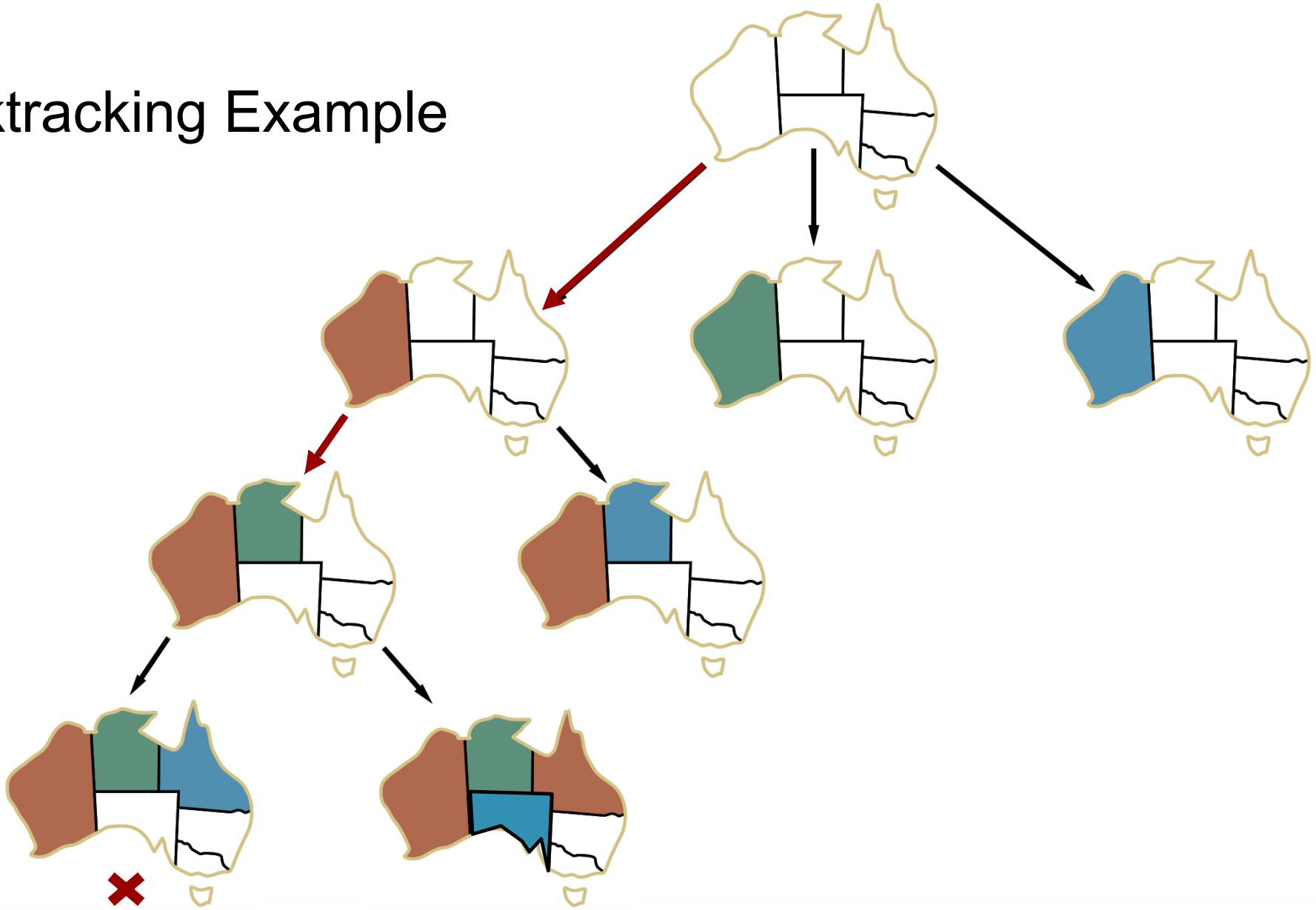
Backtracking Example



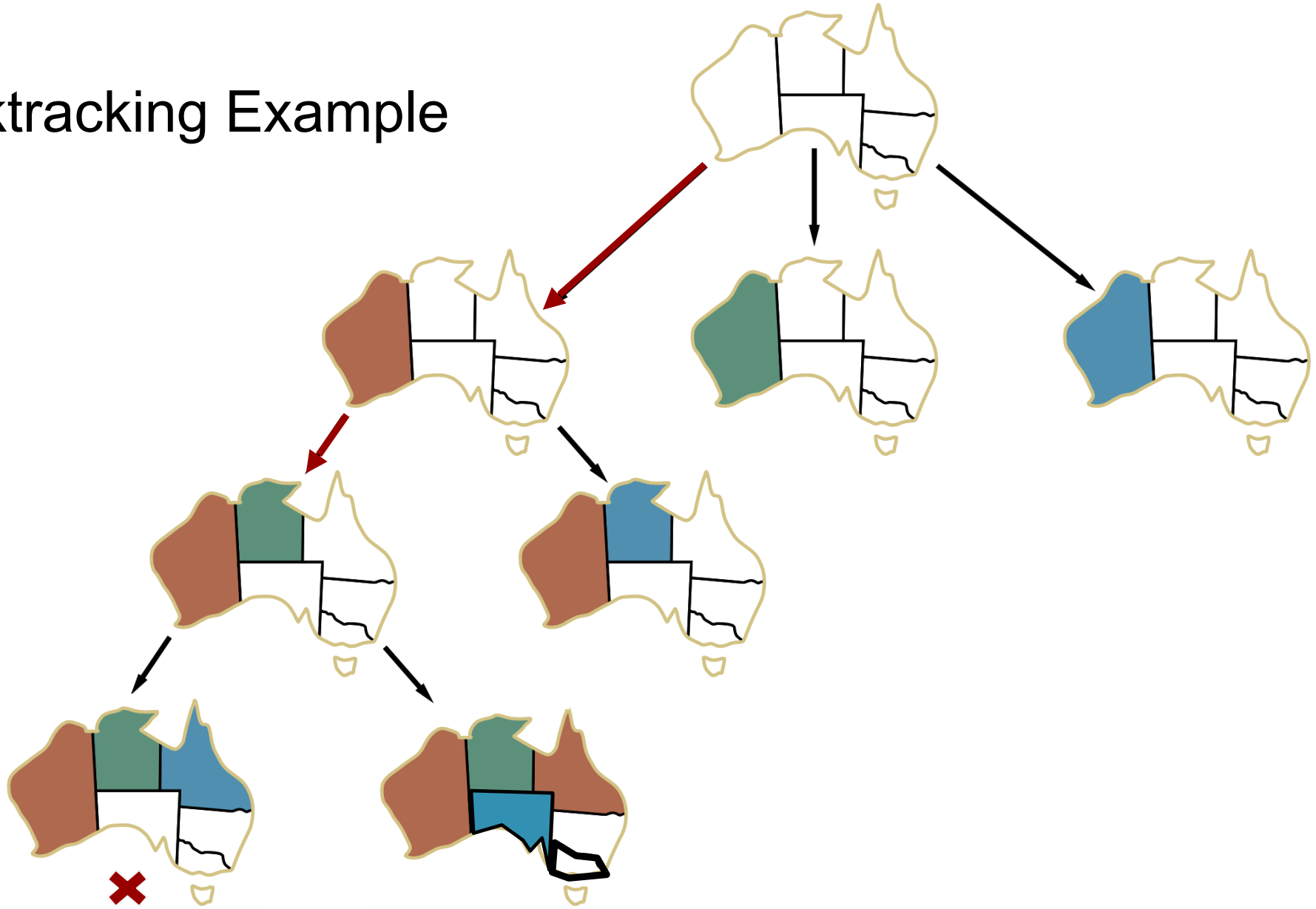
Backtracking Example



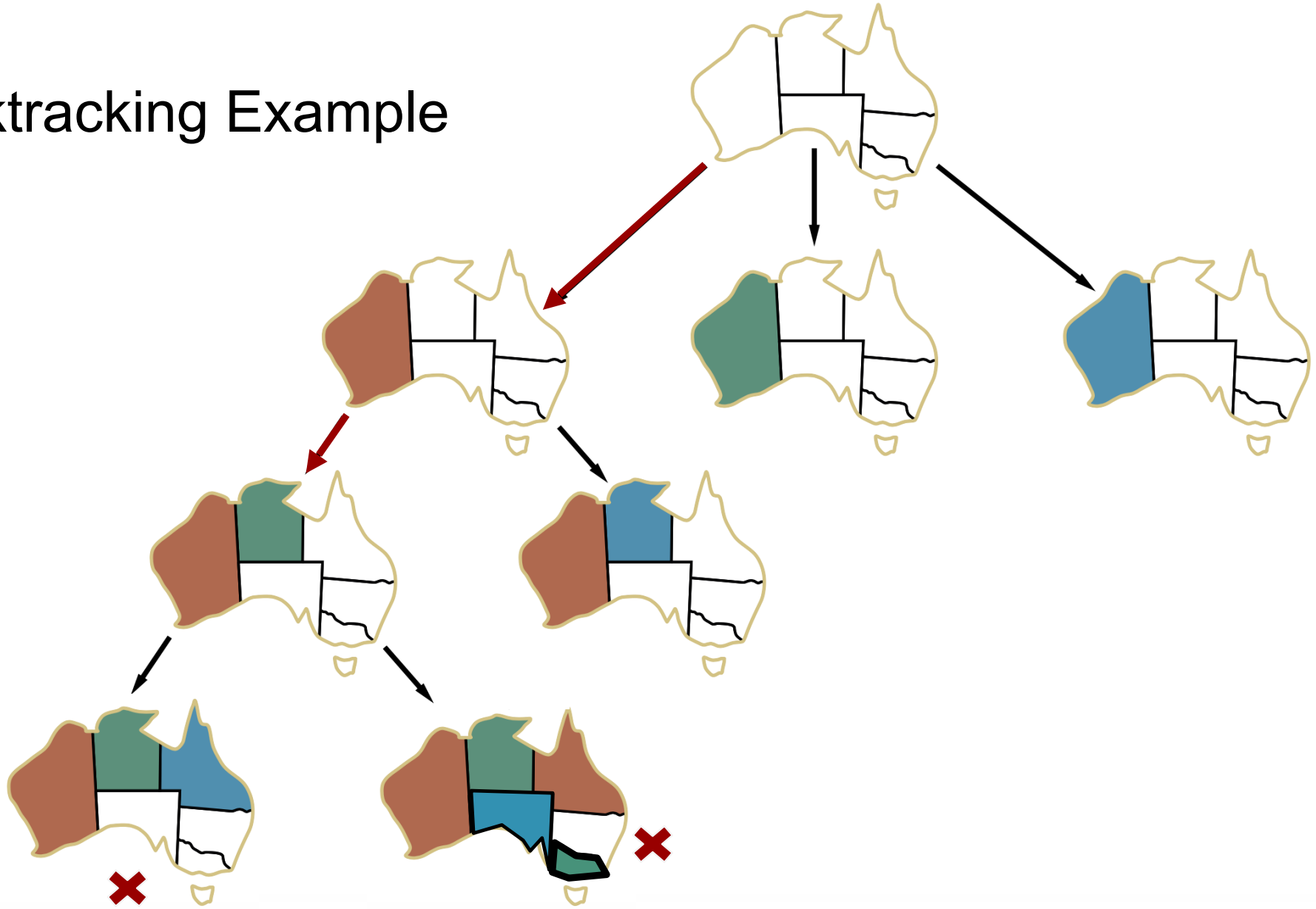
Backtracking Example



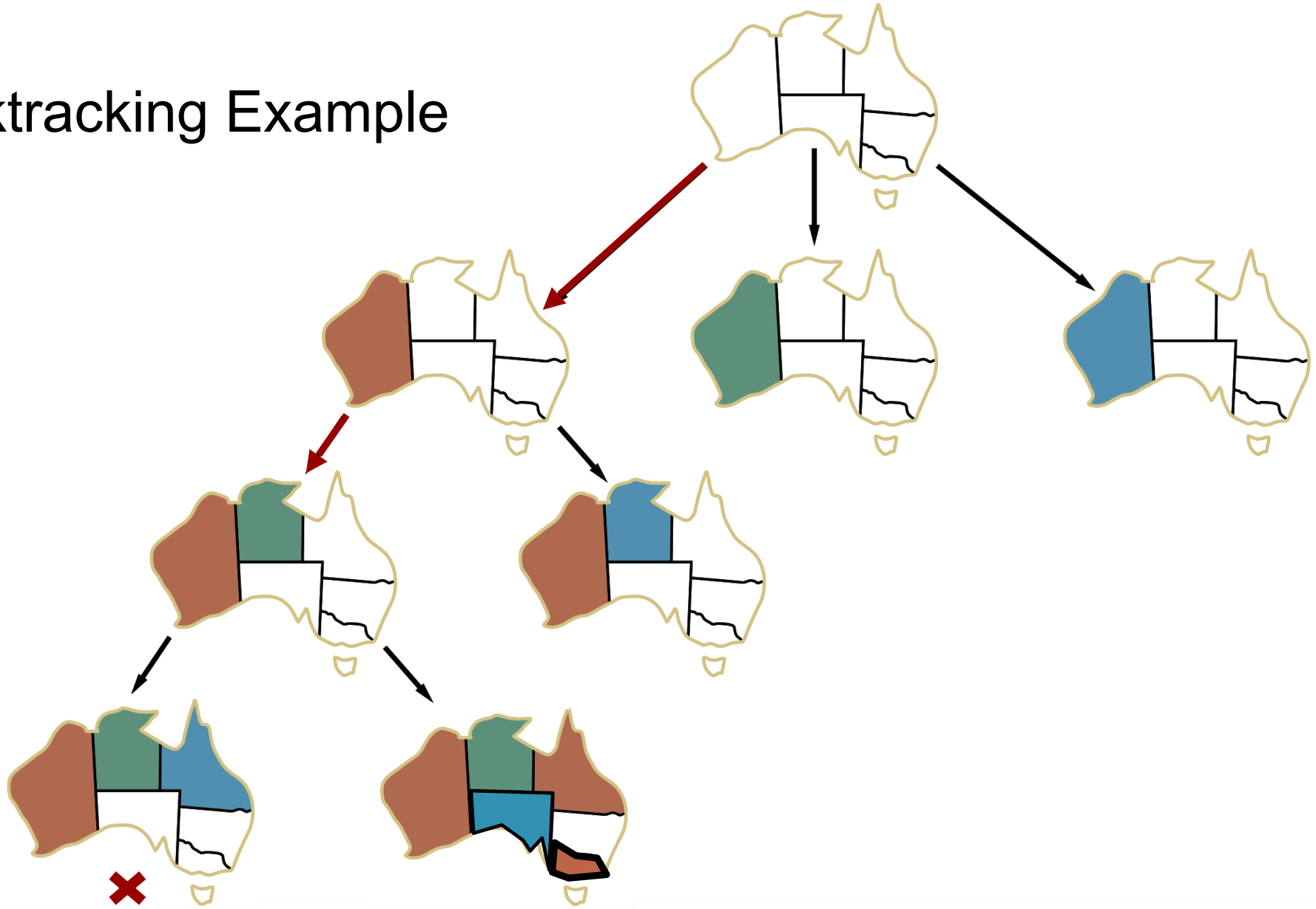
Backtracking Example



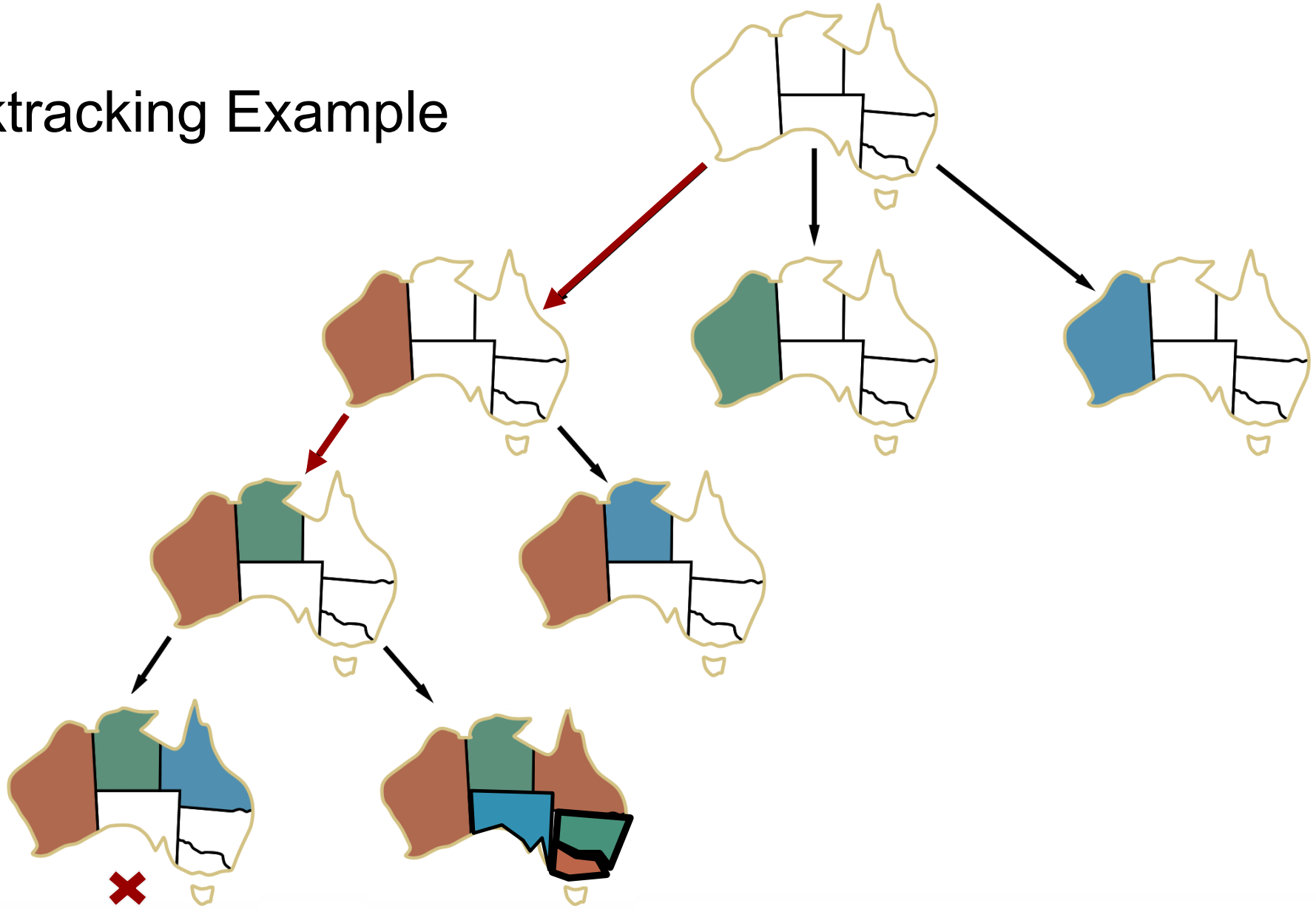
Backtracking Example



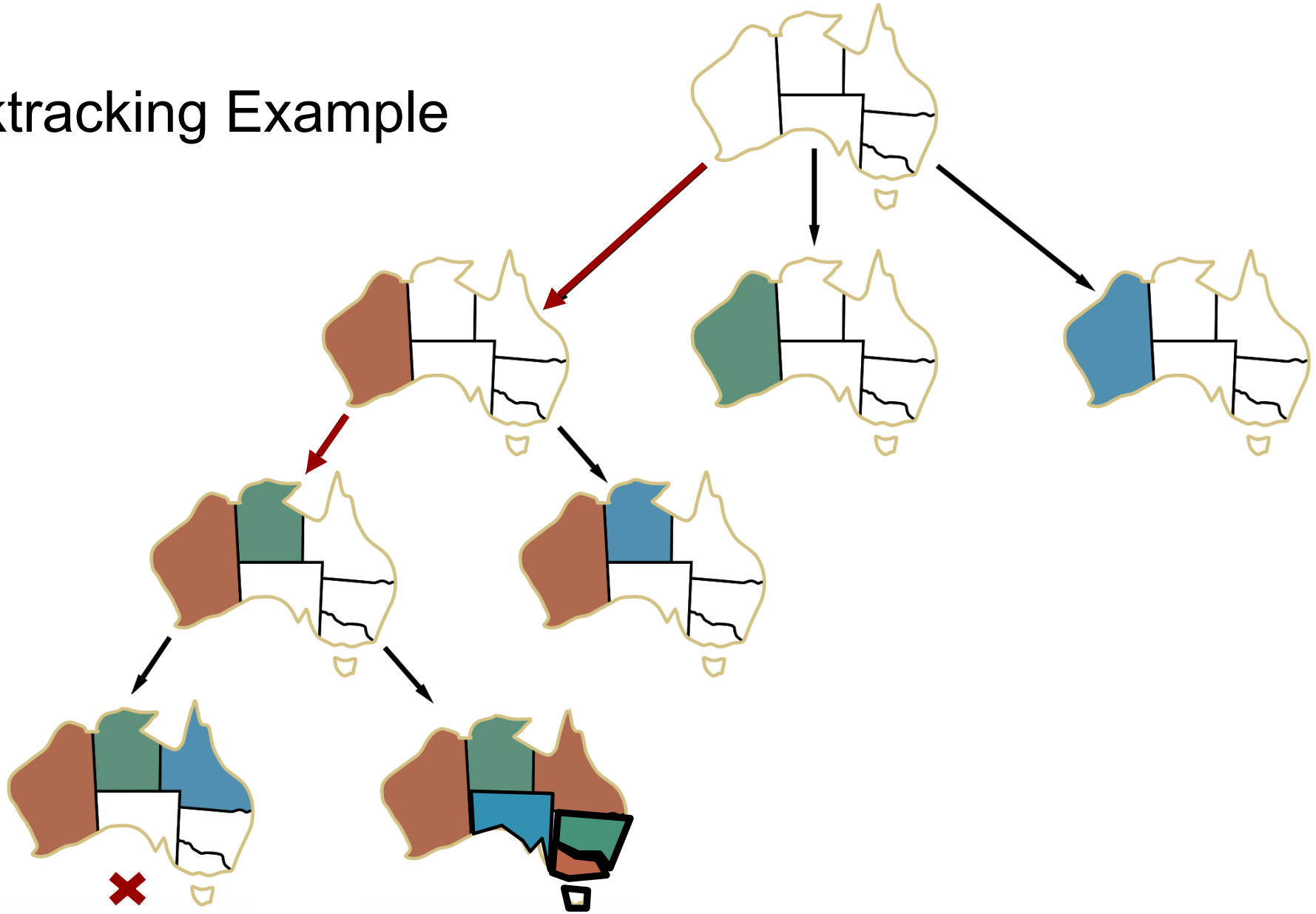
Backtracking Example



Backtracking Example



Backtracking Example



Simple Backtracking Algorithm for CSPs

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

function BACKTRACK(csp, assignment) returns a solution or failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(csp, assignment)
    for each value in DOMAIN-VALUES(csp, var, assignment) do
        if value is consistent with assignment then
            add {var = value} to assignment
            result ← BACKTRACK(csp, assignment)
            if result ≠ failure then return result
            remove {var = value} from assignment
    return failure
```

recursion

Efficiency Improvement for Backtracking Search

- Which variable should be assigned next?
- In what order should its values be tried?
- Can we detect inevitable failure early?

Efficiency Improvement for Backtracking Search

- **Which variable should be assigned next?**
- In what order should its values be tried?
- Can we detect inevitable failure early?

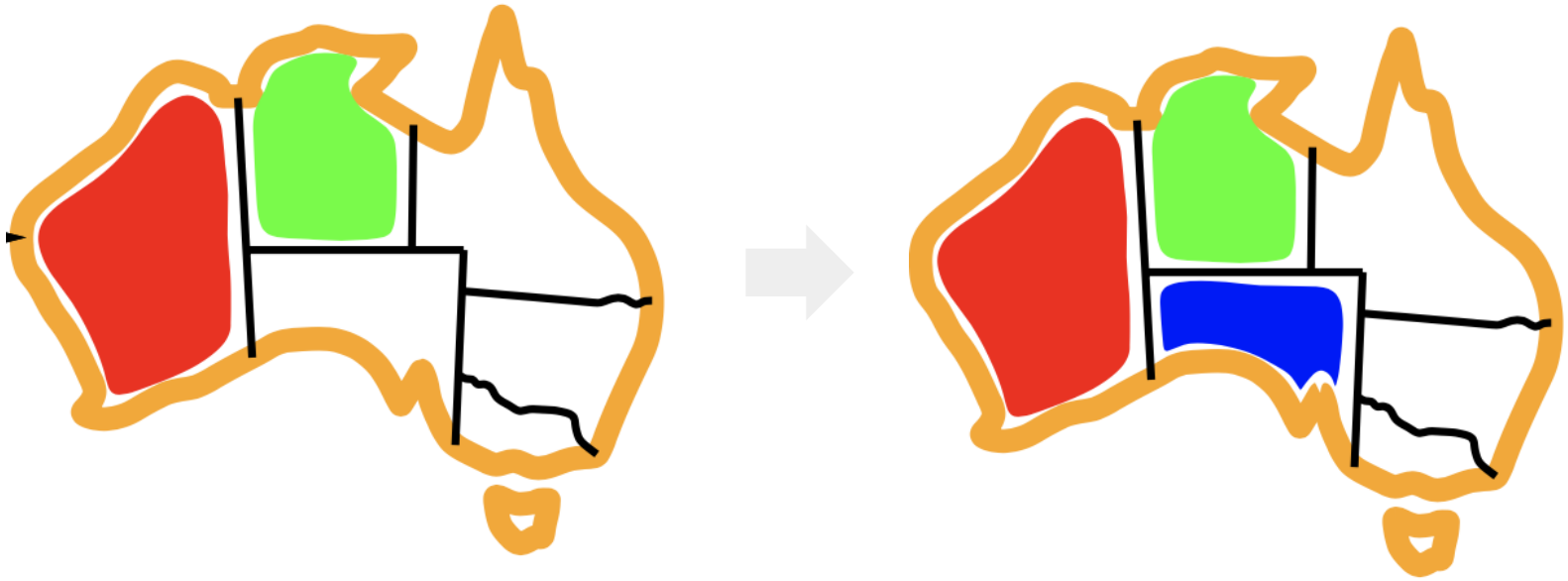
Simple Backtracking Algorithm for CSPs

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

function BACKTRACK(csp, assignment) returns a solution or failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(csp, assignment)
    for each value in DOMAIN-VALUES(csp, var, assignment) do
        if value is consistent with assignment then
            add {var = value} to assignment
            result ← BACKTRACK(csp, assignment)
            if result ≠ failure then return result
            remove {var = value} from assignment
    return failure
```

Select-Unassigned-Variable

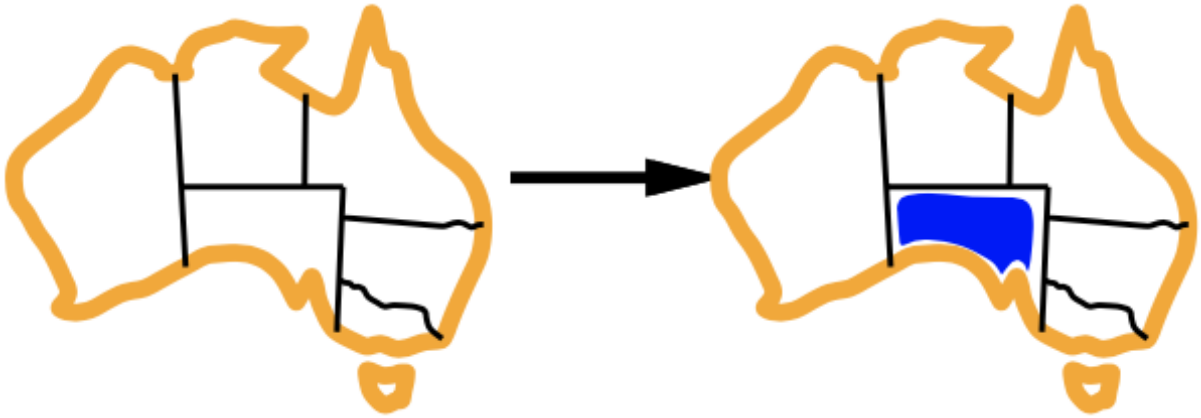
- Minimum-remaining-values (MRV) heuristic
  - Select the variable with the fewest legal values

    (most constrained variable or fail-first heuristic)



Select-Unassigned-Variable (Cont.)

- Degree heuristic
  - Idea: Reduce the branching factor on future choices
  - Select the variable that is involved in the largest number of constraints on other unassigned variables
    i.e., Select the variable that has the highest degree



Efficiency Improvement for Backtracking Search

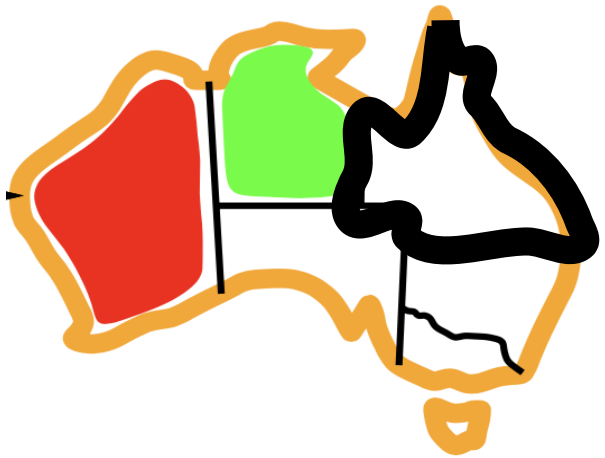
- Which variable should be assigned next?
- **In what order should its values be tried?**
- Can we detect inevitable failure early?

Simple Backtracking Algorithm for CSPs

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

function BACKTRACK(csp, assignment) returns a solution or failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(csp, assignment)
    for each value in DOMAIN-VALUES(csp, var, assignment) do
        if value is consistent with assignment then
            add {var = value} to assignment
            result ← BACKTRACK(csp, assignment)
            if result ≠ failure then return result
            remove {var = value} from assignment
    return failure
```

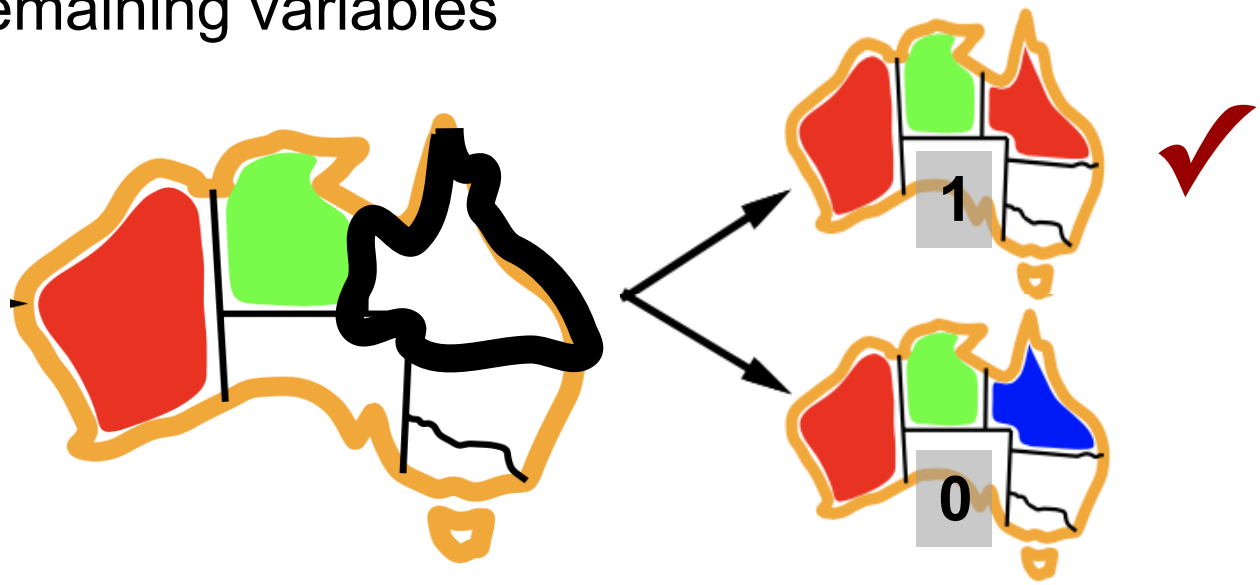
Domain-Values

- Least-constraining-value heuristic
  - Idea: Leave the maximum flexibility for subsequent variable assignments
  - It prefers the value that rules out the fewest values in the remaining variables



Domain-Values

- Least-constraining-value heuristic
  - Idea: Leave the maximum flexibility for subsequent variable assignments
  - It prefers the value that rules out the fewest values in the remaining variables



Backtracking Algorithm for CSPs

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

function BACKTRACK(csp, assignment) returns a solution or failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(csp, assignment)
    for each value in ORDER-DOMAIN-VALUES(csp, var, assignment) do
        if value is consistent with assignment then
            add {var = value} to assignment
            result ← BACKTRACK(csp, assignment)
            if result ≠ failure then return result
            remove {var = value} from assignment
    return failure
```

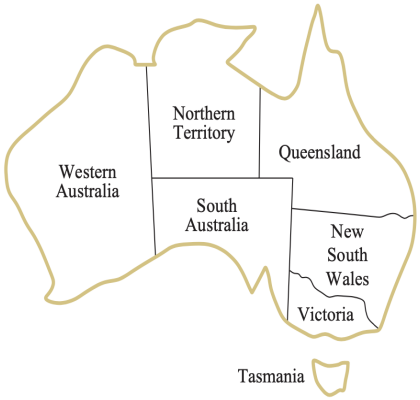
Efficiency Improvement for Backtracking Search

- Which variable should be assigned next?
- In what order should its values be tried?
- **Can we detect inevitable failure early?**

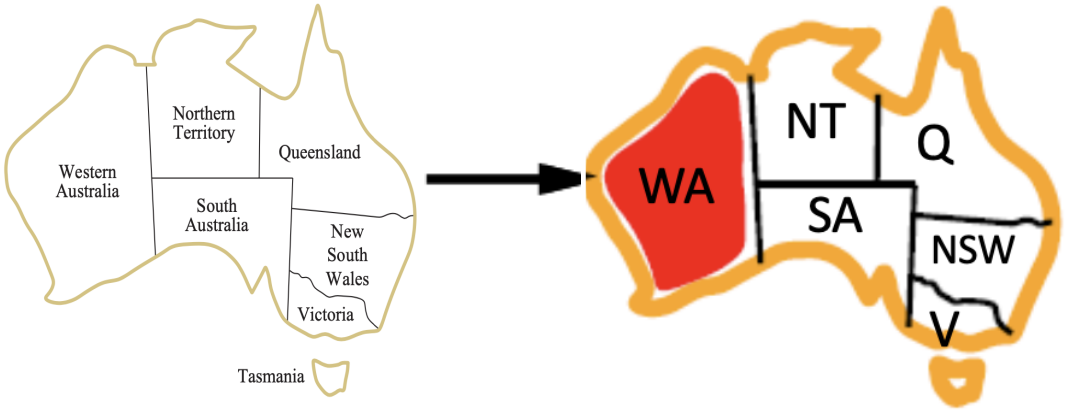
Inference by Forward Checking

- Filtering
  - Keep track of remaining legal values for unassigned variables and cross off bad options
  - Terminate search when any variable has no legal values
- Forward checking
  - Propagate information from assigned to unassigned variables
  - Cross off values that violate a constraint when added to the existing assignment
  - When a variable X is assigned, the forward-checking process establishes **arc consistency** for it
    - For each unassigned variable Y that is connected to X by a constraint, delete from Y's domain any value that is inconsistent with the value chosen for X

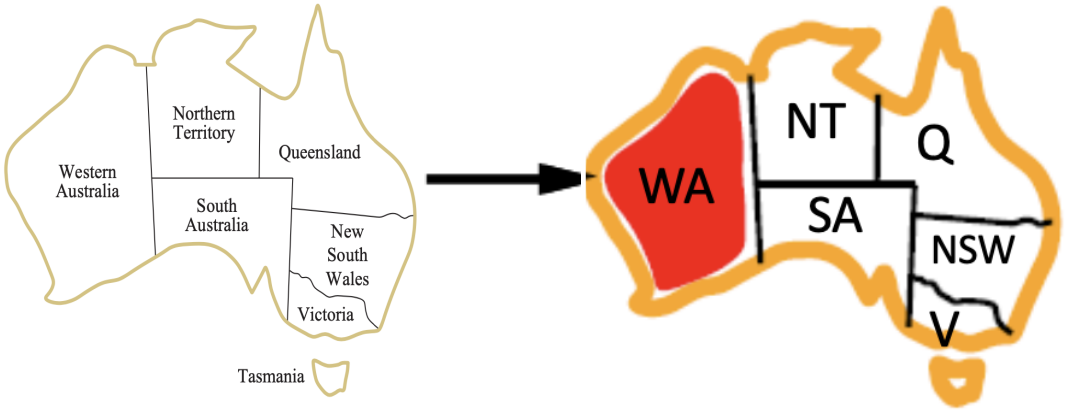
Example: Forward Checking



Example: Forward Checking



Example: Forward Checking



Backtracking Algorithm for CSPs

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

function BACKTRACK(csp, assignment) returns a solution or failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(csp, assignment)
    for each value in ORDER-DOMAIN-VALUES(csp, var, assignment) do
        if value is consistent with assignment then
            add {var = value} to assignment
            result ← BACKTRACK(csp, assignment)
            if result ≠ failure then return result
            remove {var = value} from assignment
    return failure
```

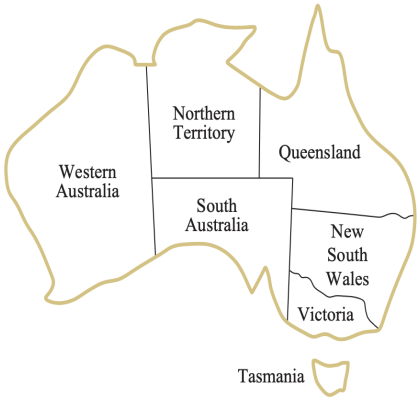
Backtracking Algorithm for CSPs

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

function BACKTRACK(csp, assignment) returns a solution or failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(csp, assignment)
    for each value in ORDER-DOMAIN-VALUES(csp, var, assignment) do
        if value is consistent with assignment then
            add {var = value} to assignment

    return failure
```

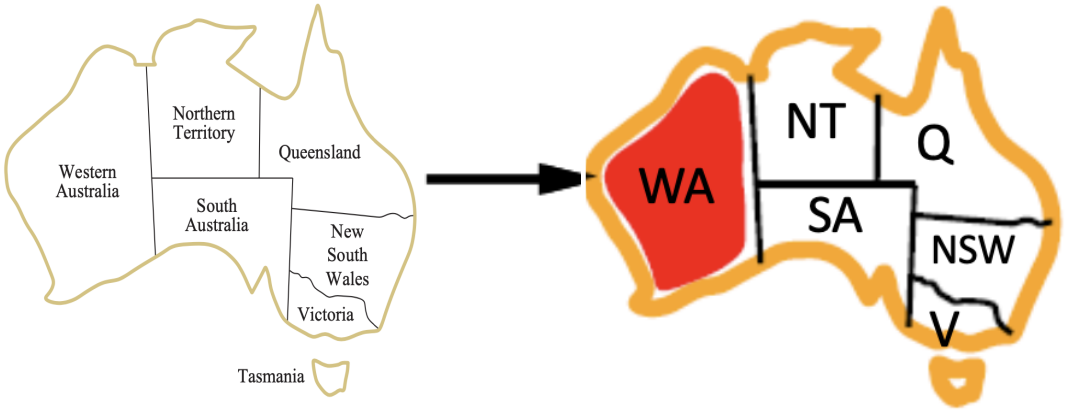
Backtracking Algorithm for CSPs

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

function BACKTRACK(csp, assignment) returns a solution or failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(csp, assignment)
    for each value in ORDER-DOMAIN-VALUES(csp, var, assignment) do
        if value is consistent with assignment then
            add {var = value} to assignment
            inferences ← INFERENCE(csp, var, assignment)
            if inferences ≠ failure then
                add inferences to csp
                result ← BACKTRACK(csp, assignment)
                if result ≠ failure then return result
                remove inferences from csp
            remove {var = value} from assignment
    return failure
```

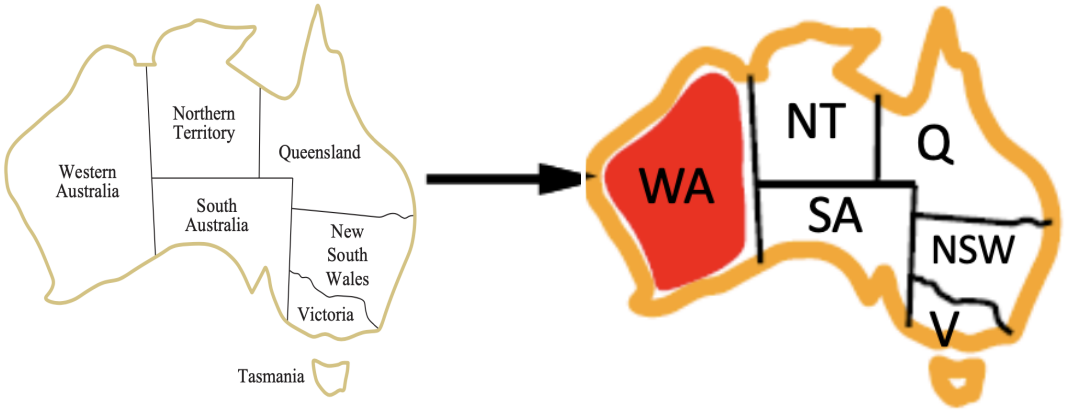
Example: Forward Checking



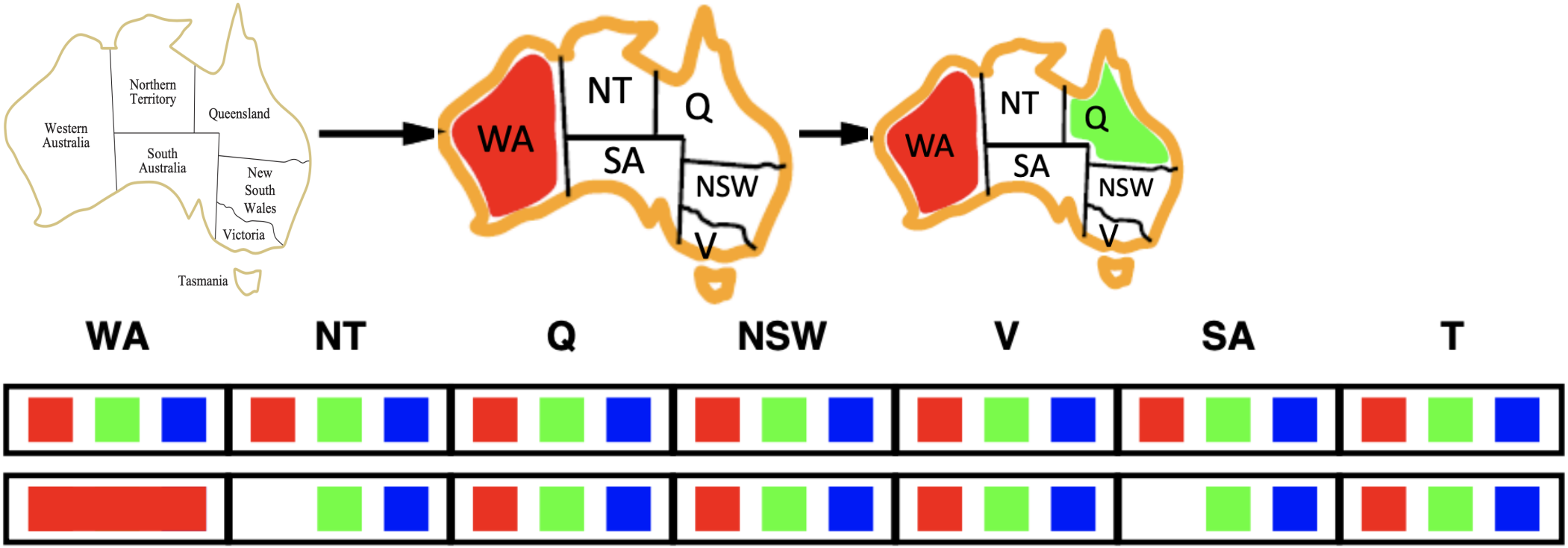
Example: Forward Checking



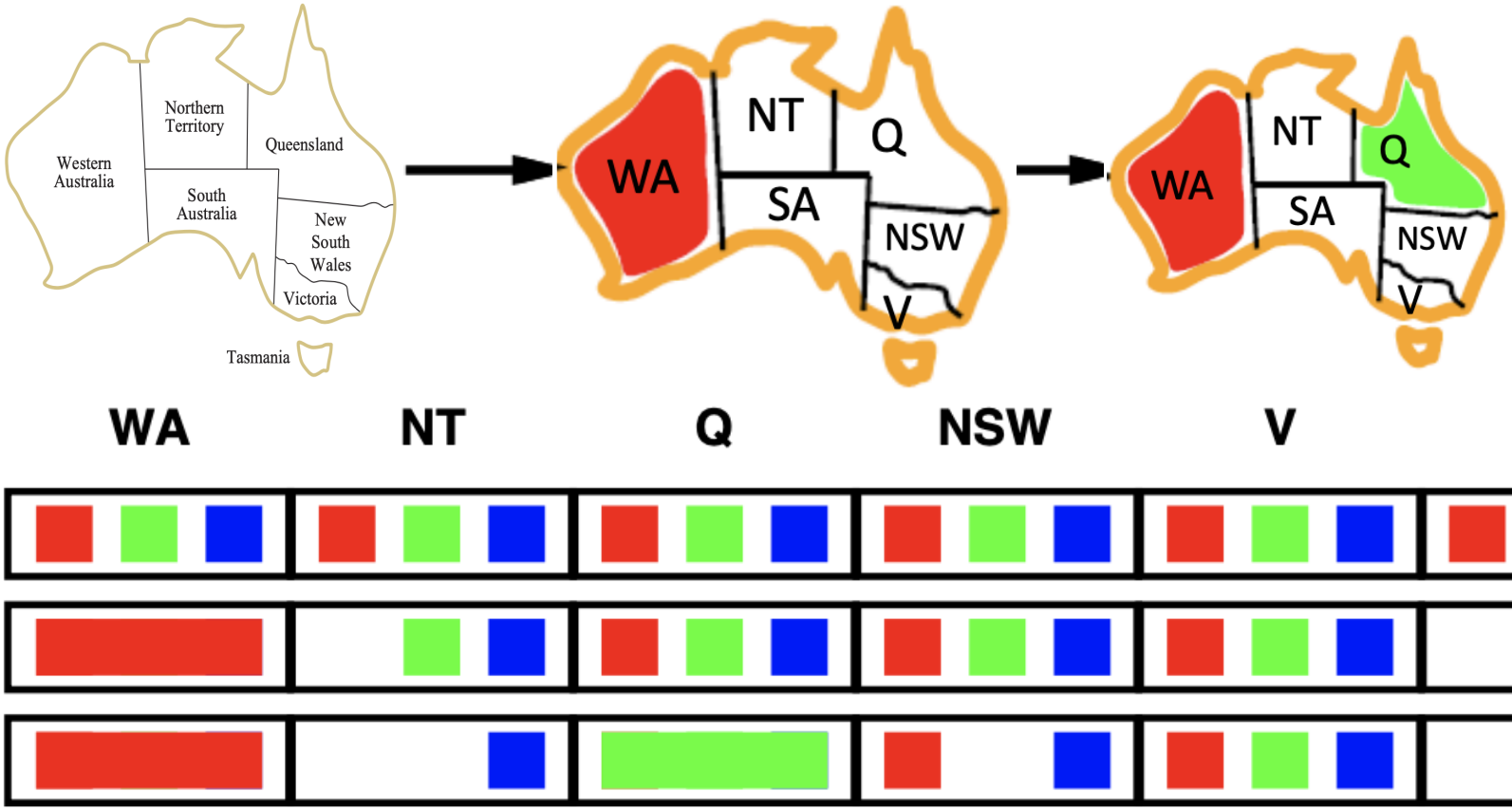
Example: Forward Checking



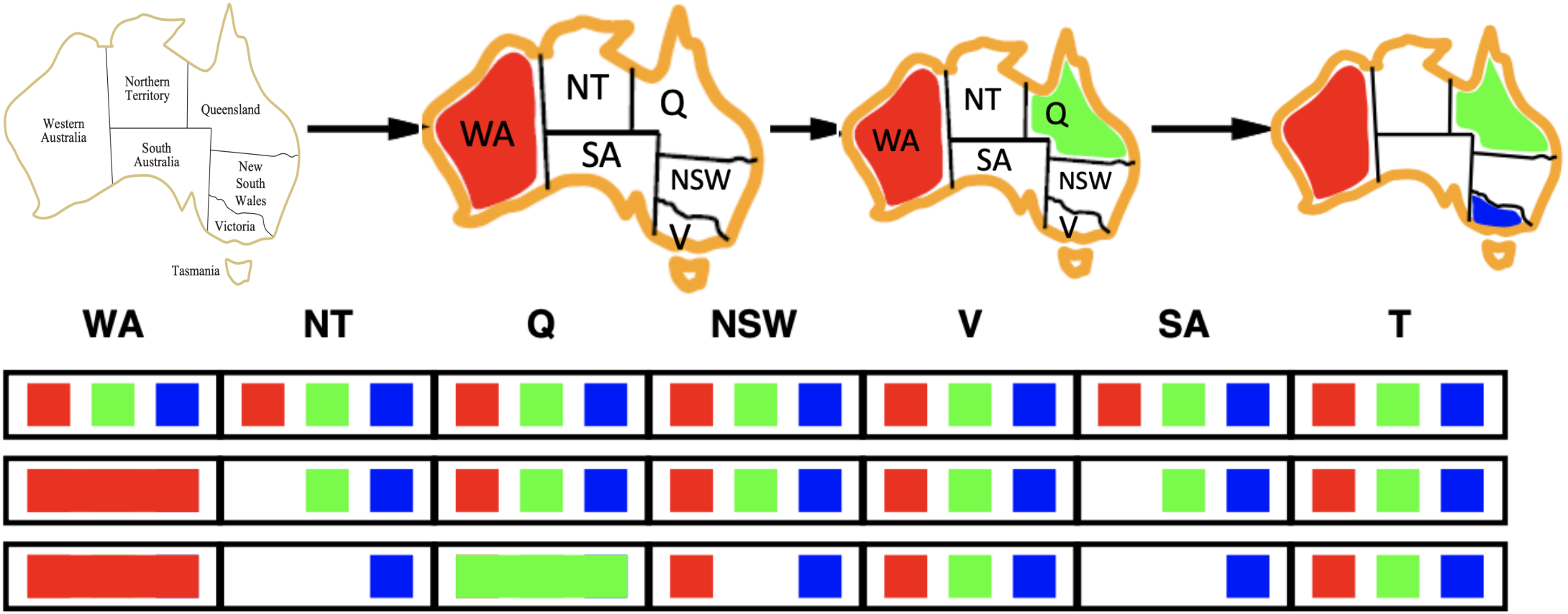
Example: Forward Checking



Example: Forward Checking

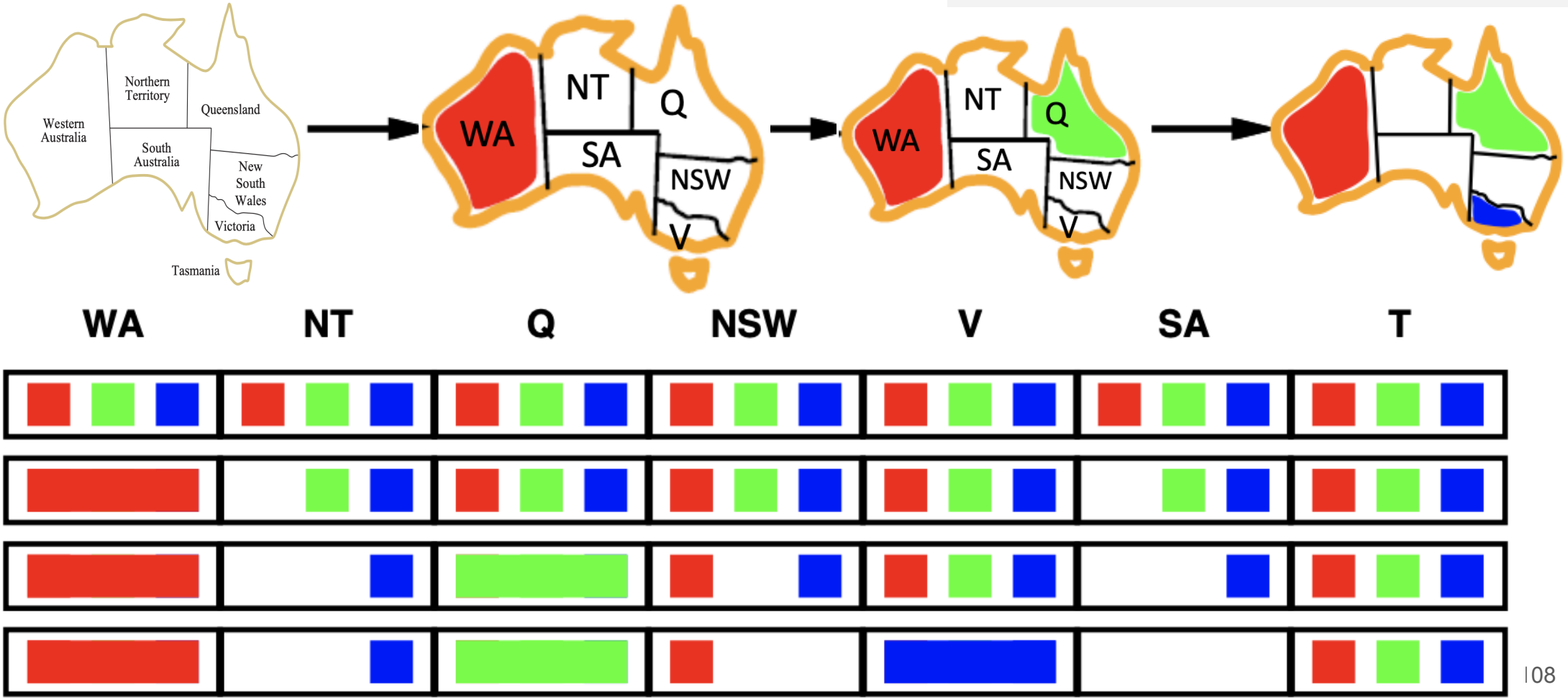


Example: Forward Checking



Example: Forward Checking

{WA=red, Q=green,V=blue}
SA: no legal values!



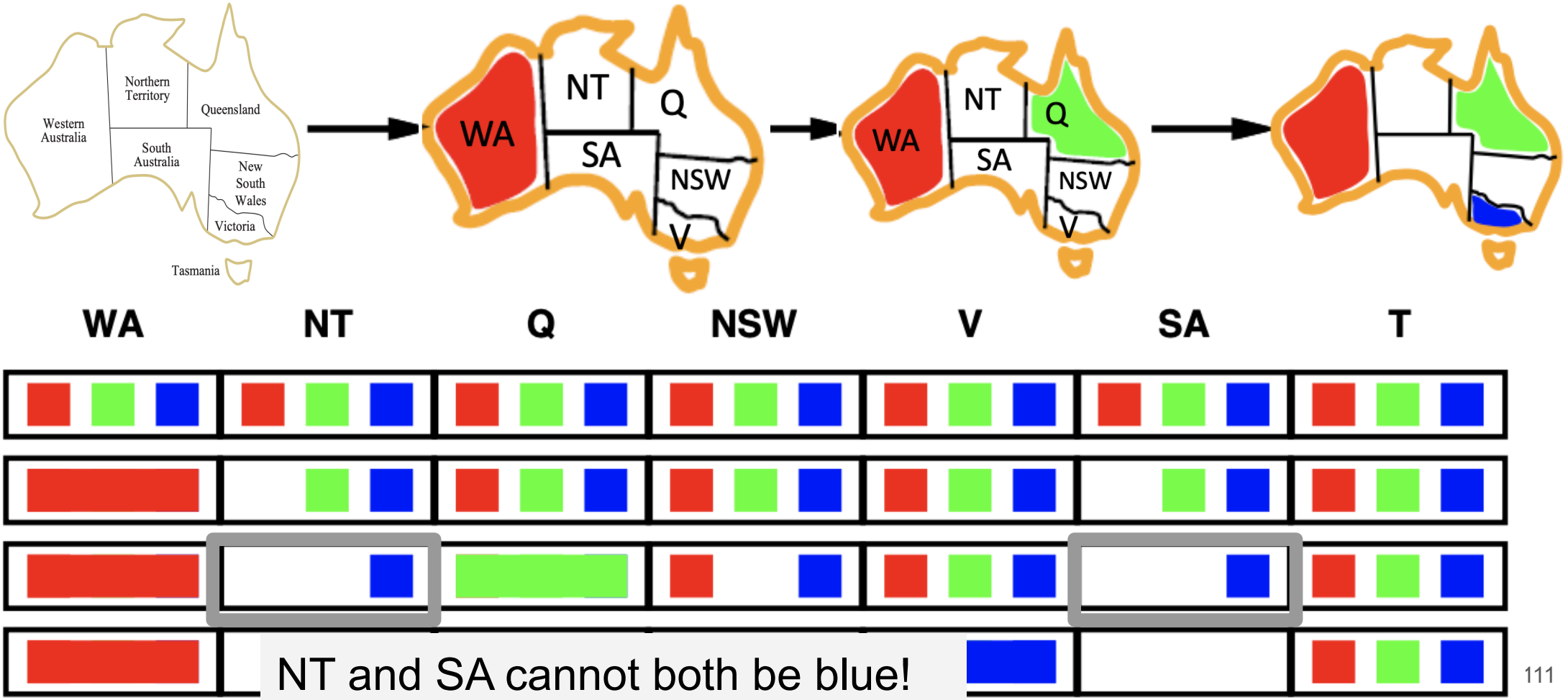
Backtracking Algorithm for CSPs

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

function BACKTRACK(csp, assignment) returns a solution or failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(csp, assignment)
    for each value in ORDER-DOMAIN-VALUES(csp, var, assignment) do
        if value is consistent with assignment then
            add {var = value} to assignment
            inferences ← INFERENCE(csp, var, assignment)
            if inferences ≠ failure then
                add inferences to csp
                result ← BACKTRACK(csp, assignment)
                if result ≠ failure then return result
                remove inferences from csp
            remove {var = value} from assignment
    return failure
```

Weakness of Forward Checking

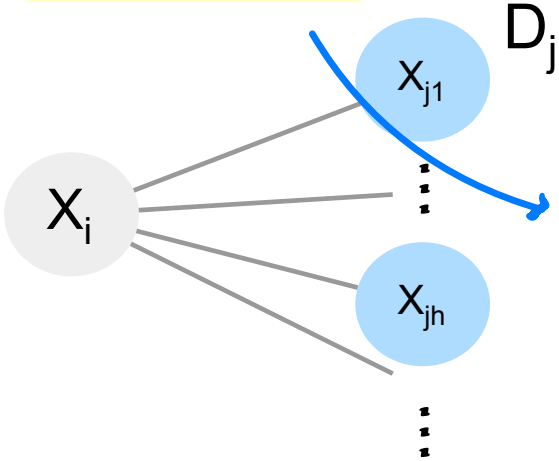
- Forward checking propagates information from assigned to unassigned variables, but does not provide **early** detection for all failures

Example: Forward Checking



MAC (Maintaining Arc Consistency) Algorithm

- After a variable X_i is assigned a value, the INFERENCE procedure calls AC-3 to detect inconsistencies
  - Idea: we start with only the arcs (X_j, X_i) for all X_j that are unassigned variables that are neighbors of X_i, then do **constraint propagation**



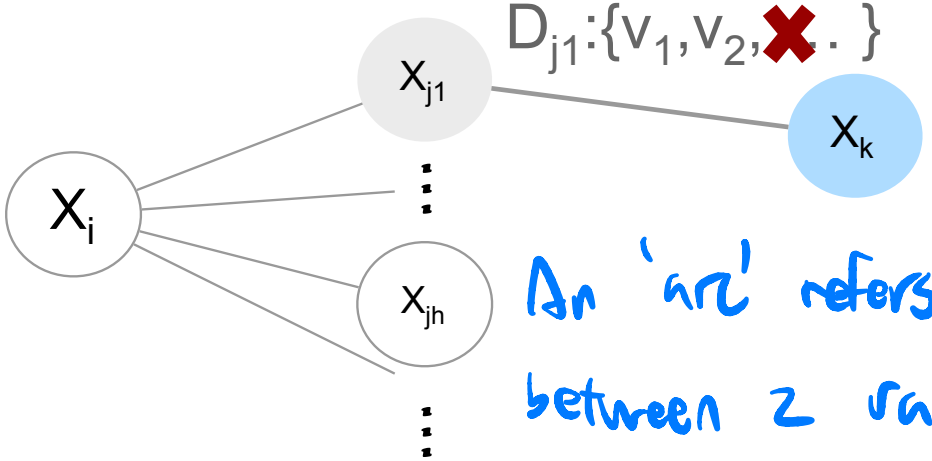
revised!

D_{j1}:{v_1,v_2,✖. }

When the domain of a variable is updated, the information is propagated throughout the system, leading to further domain reductions for other variables.

MAC (Maintaining Arc Consistency) Algorithm

- After a variable X_i is assigned a value, the INFERENCE procedure calls AC-3 to detect inconsistencies
  - Idea: we start with only the arcs (X_j, X_i) for all X_j that are unassigned variables that are neighbors of X_i, then do **constraint propagation**

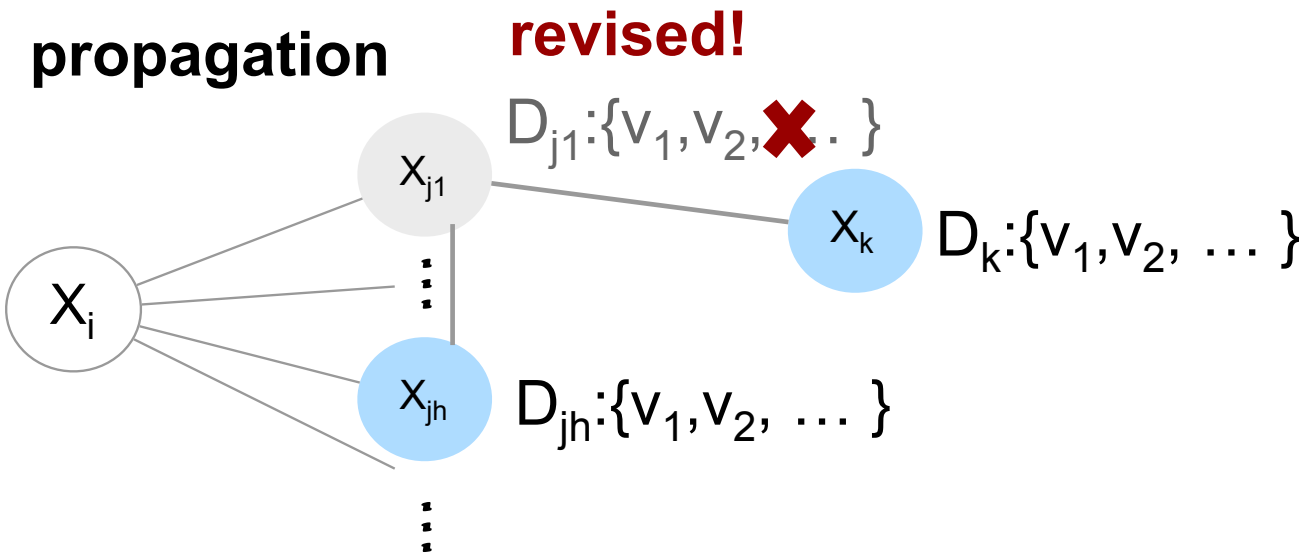


$A \neq B$ becomes $A \neq B$ & $B \neq A$

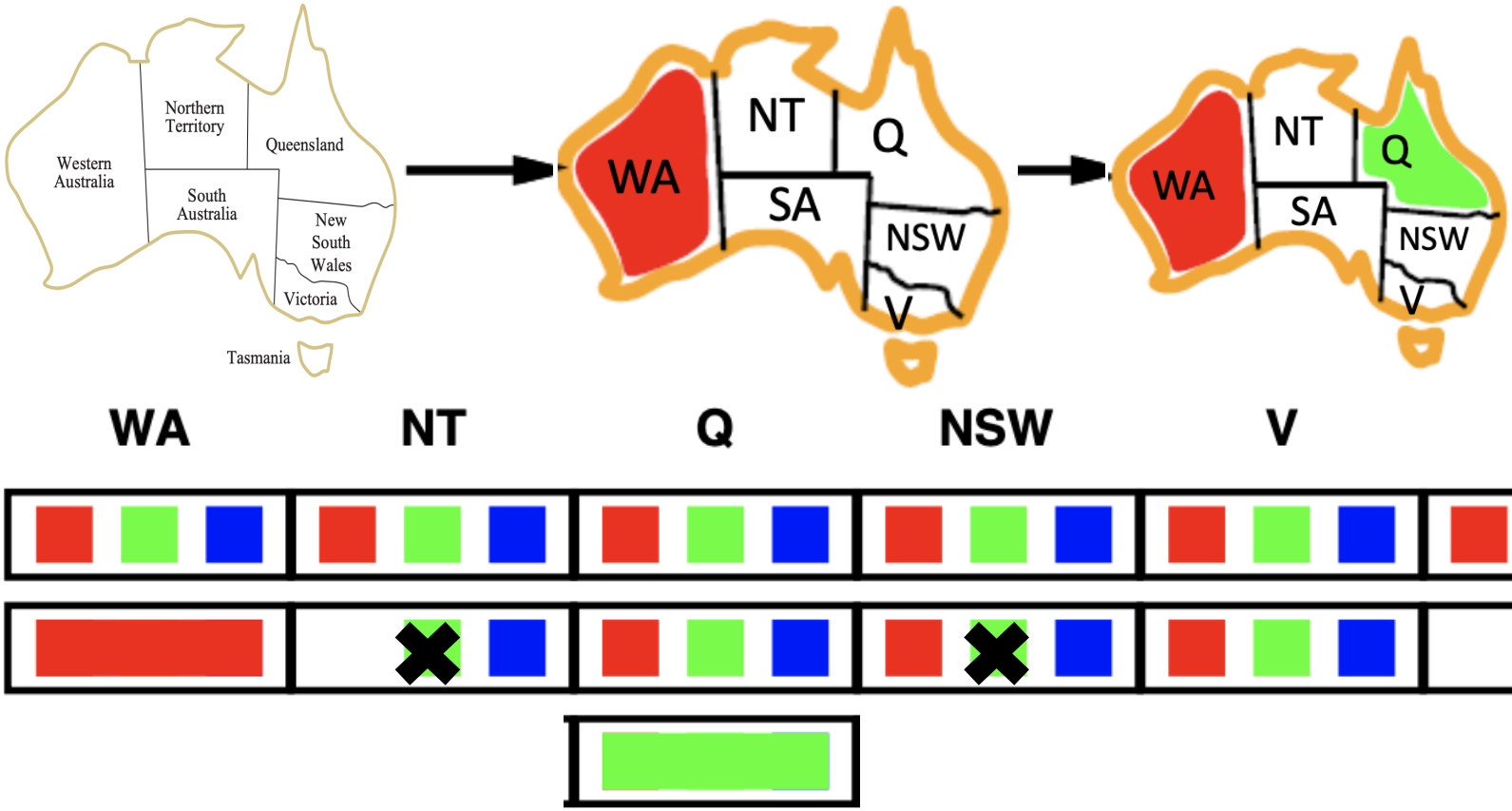
An 'arc' refers to a directional relationship between 2 variables that are subject to a constraint. e.g. An arc from X_i to X_j would represent the value of X_i must be compatible to X_j

MAC (Maintaining Arc Consistency) Algorithm

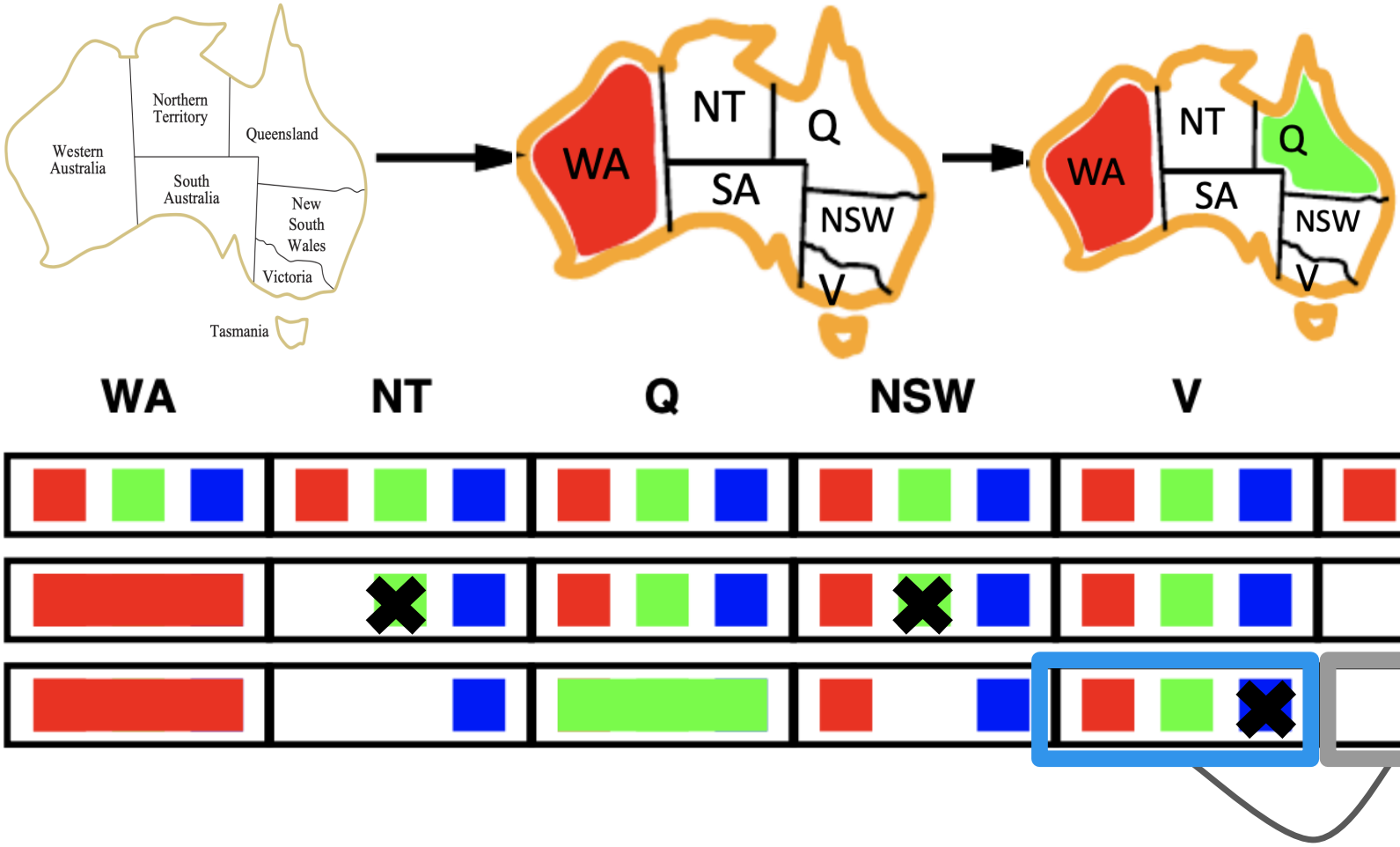
- After a variable X_i is assigned a value, the INFERENCE procedure calls AC-3 to detect inconsistencies
  - Idea: we start with only the arcs (X_j,X_i) for all X_j that are unassigned variables that are neighbors of X_i, then do **constraint propagation**



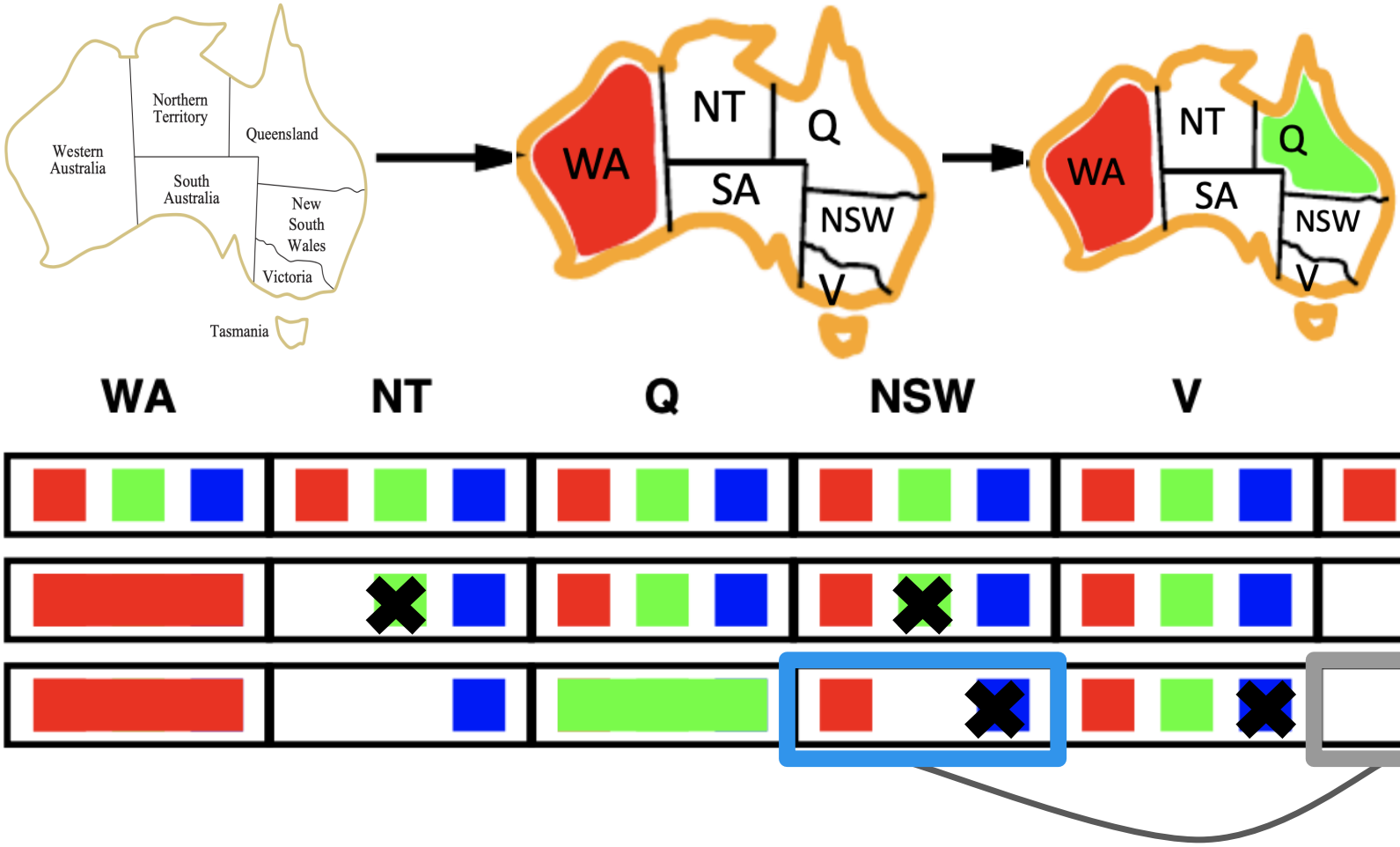
Example: Forward Checking



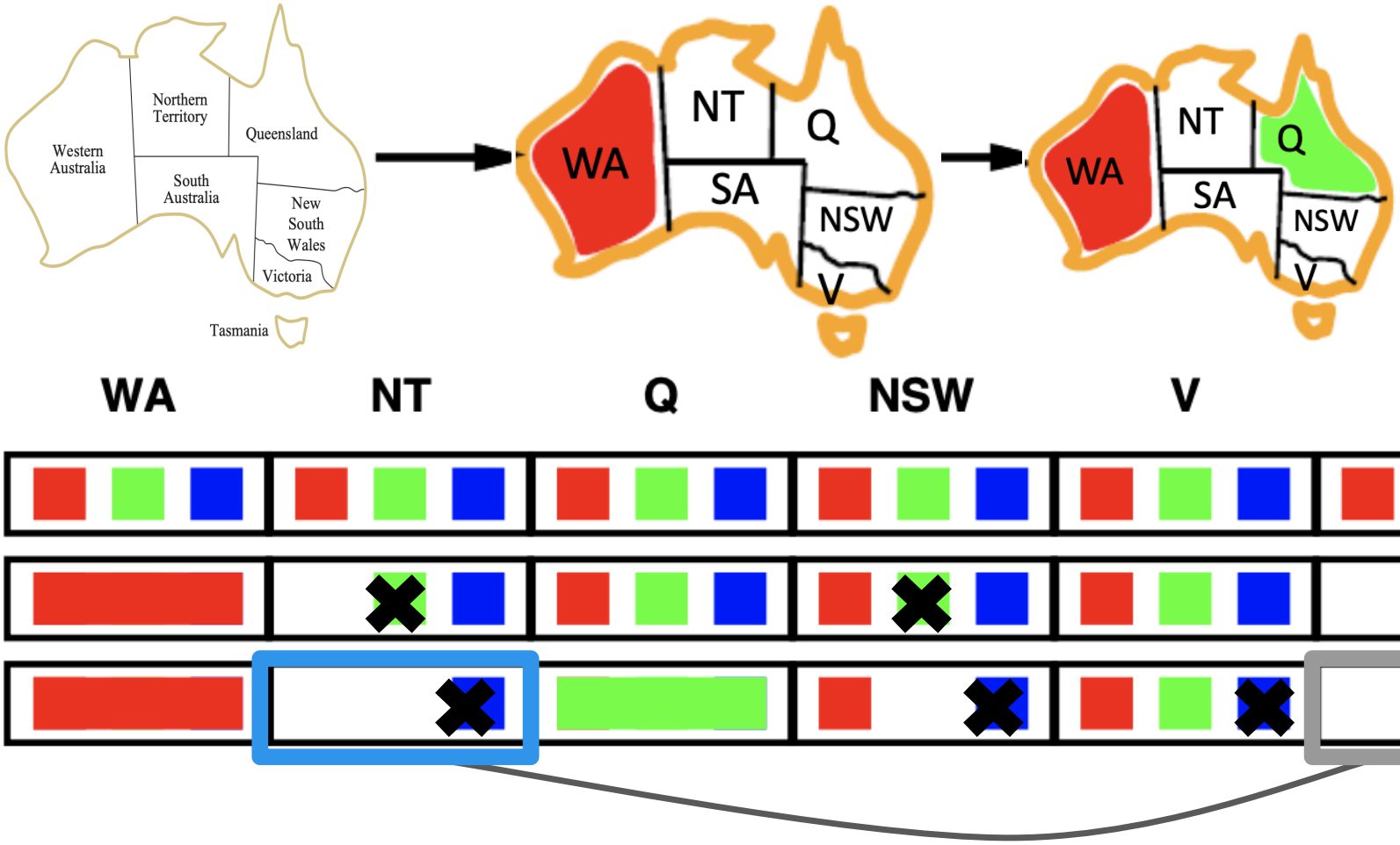
Example: Forward Checking



Example: Forward Checking



Example: Forward Checking



Arc Consistency Algorithm

```
function AC-3(csp) returns false if an inconsistency is found and true otherwise
    queue ← a queue of arcs, initially all the arcs in csp

    while queue is not empty do
        (X_i, X_j) ← POP(queue)
        if REVISE(csp, X_i, X_j) then
            if size of D_i = 0 then return false
            for each X_k in X_i.NEIGHBORS - {X_j} do
                add (X_k, X_i) to queue
    return true
```

inconsistency ⇒ no solution exists

```
function REVISE(csp, X_i, X_j) returns true iff we revise the domain of X_i
    revised ← false
    for each x in D_i do
        if no value y in D_j allows (x,y) to satisfy the constraint between X_i and X_j then
            delete x from D_i
            revised ← true
    return revised
```