CS 471/571(Fall 2023): Introduction to Artificial Intelligence

Lecture 6: Constraint Satisfaction Problems (Part 2)

Thanh H. Nguyen

Source: http://ai.berkeley.edu/home.html

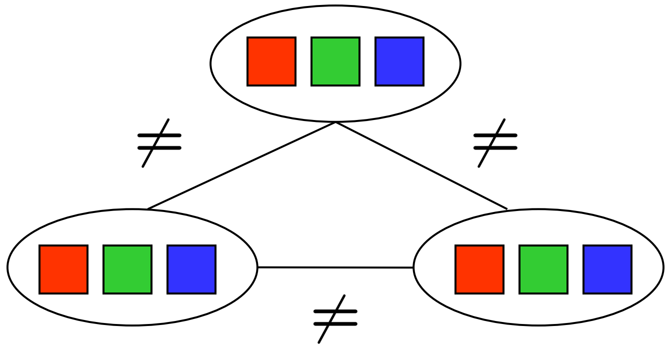
Reminder

- Written assignment 1:
  - Deadline: Oct 11th, 2023

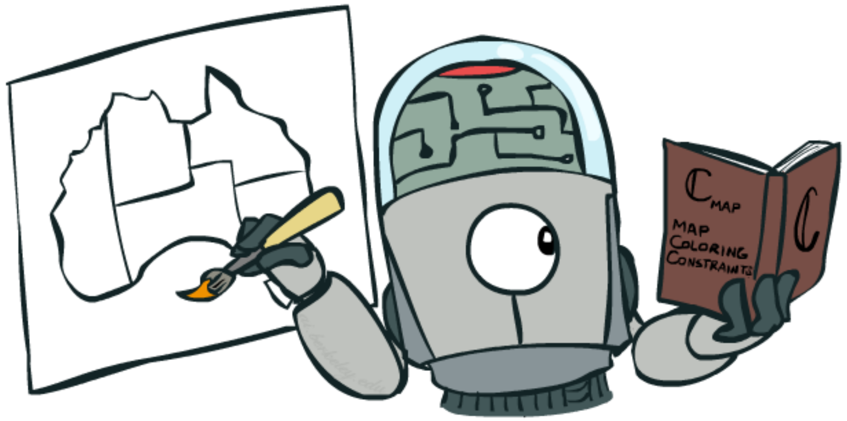
- Project 1:
  - Deadline: Oct 16th, 2023

Reminder: CSPs

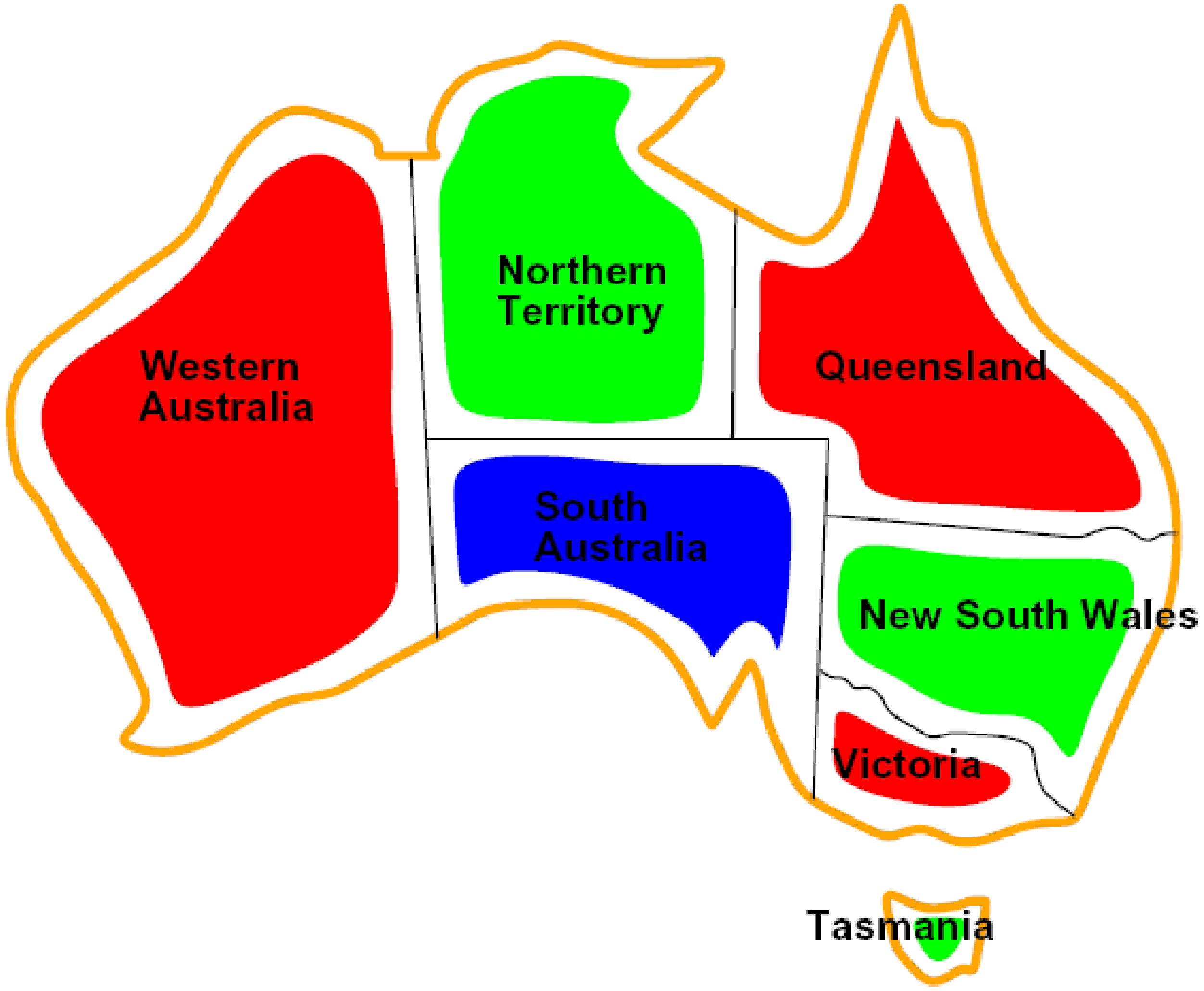
- CSPs:
  - Variables
  - Domains
  - Constraints
    - Implicit (provide code to compute)
    - Explicit (provide a list of the legal tuples)
    - Unary / Binary / N-ary

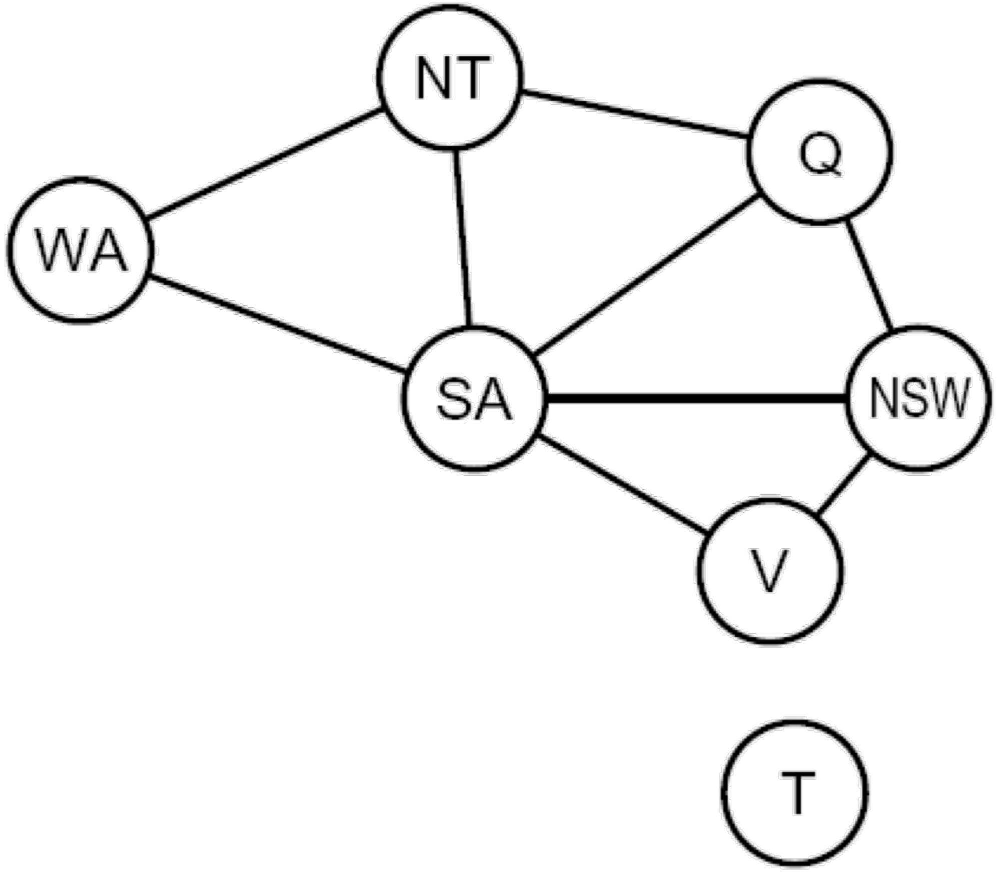


- Goals:
  - Here: find any solution
  - Also: find all, find best, etc.



Example: Map Coloring





Backtracking Search

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

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

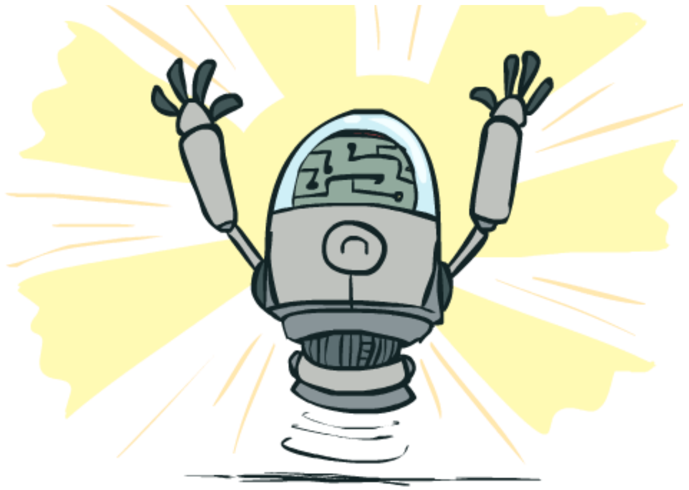
Improving Backtracking

- General-purpose ideas give huge gains in speed

- Filtering: Can we detect inevitable failure early?
  - Arc consistency
  - Forward checking
  - Constraint propagation

- Ordering:
  - Which variable should be assigned next?
  - In what order should its values be tried?

- Structure: Can we exploit the problem structure?

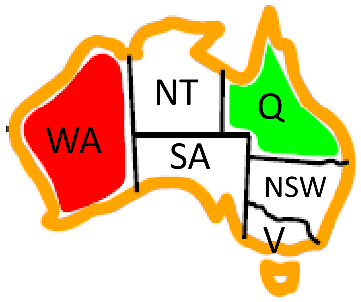


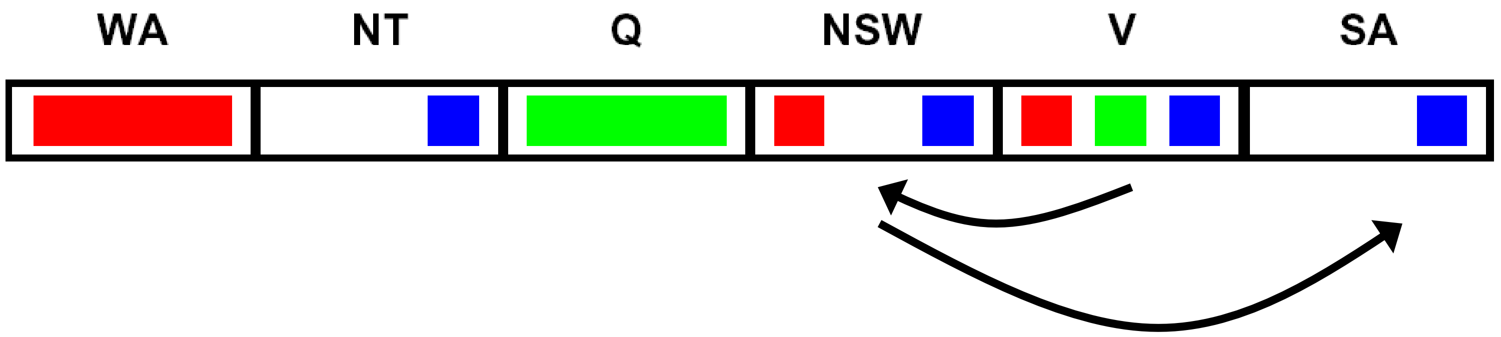
Arc Consistency

- An arc X → Y is consistent iff for *every* x in the tail there is *some* y in the head which could be assigned without violating a constraint
- Enforcing consistency of X → Y: filter values of the tail X to make X → Y consistent

Constraint Propagation

- Constraint propagation: enforce arc consistency of entire CSP
  - Maintain a queue of arcs to enforce consistency
- Important: If X loses a value, neighbors of X need to be rechecked!
  - After enforcing consistency on X → Y, if X loses a value, all arcs pointing to X need to be added back to the queue





Enforcing Arc Consistency in a CSP

```
function AC-3( csp) returns the CSP, possibly with reduced domains
    inputs: csp, a binary CSP with variables $\{X_1, X_2, \ldots, X_n\}$
    local variables: queue, a queue of arcs, initially all the arcs in csp

    while queue is not empty do
        $(X_i, X_j) \leftarrow$ REMOVE-FIRST(queue)
        if REMOVE-INCONSISTENT-VALUES($X_i, X_j$) then
            for each $X_k$ in NEIGHBORS[$X_i$] do
                add $(X_k, X_i)$ to queue
```

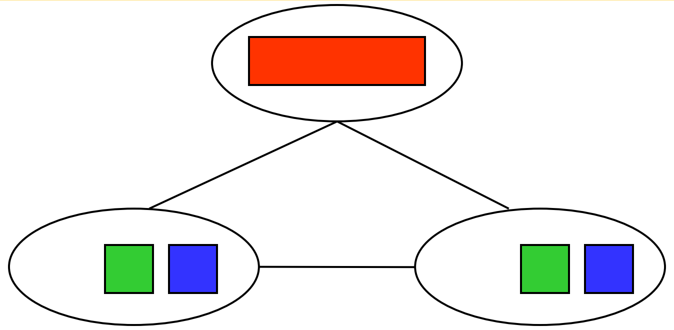
```
function REMOVE-INCONSISTENT-VALUES( $X_i, X_j$) returns true iff succeeds
    removed $\leftarrow$ false
    for each $x$ in DOMAIN[$X_i$] do
        if no value $y$ in DOMAIN[$X_j$] allows $(x,y)$ to satisfy the constraint $X_i \leftrightarrow X_j$
            then delete $x$ from DOMAIN[$X_i$];  removed $\leftarrow$ true
    return removed
```

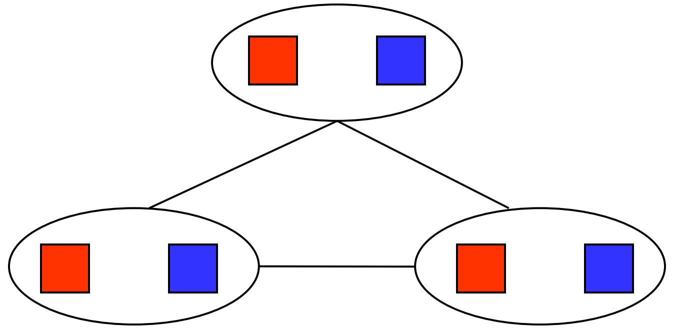
- Runtime: $O(n^2d^3)$

Limitations of Arc Consistency

- After enforcing arc consistency:
  - Can have one solution left
  - Can have multiple solutions left
  - Can have no solutions left (and not know it)

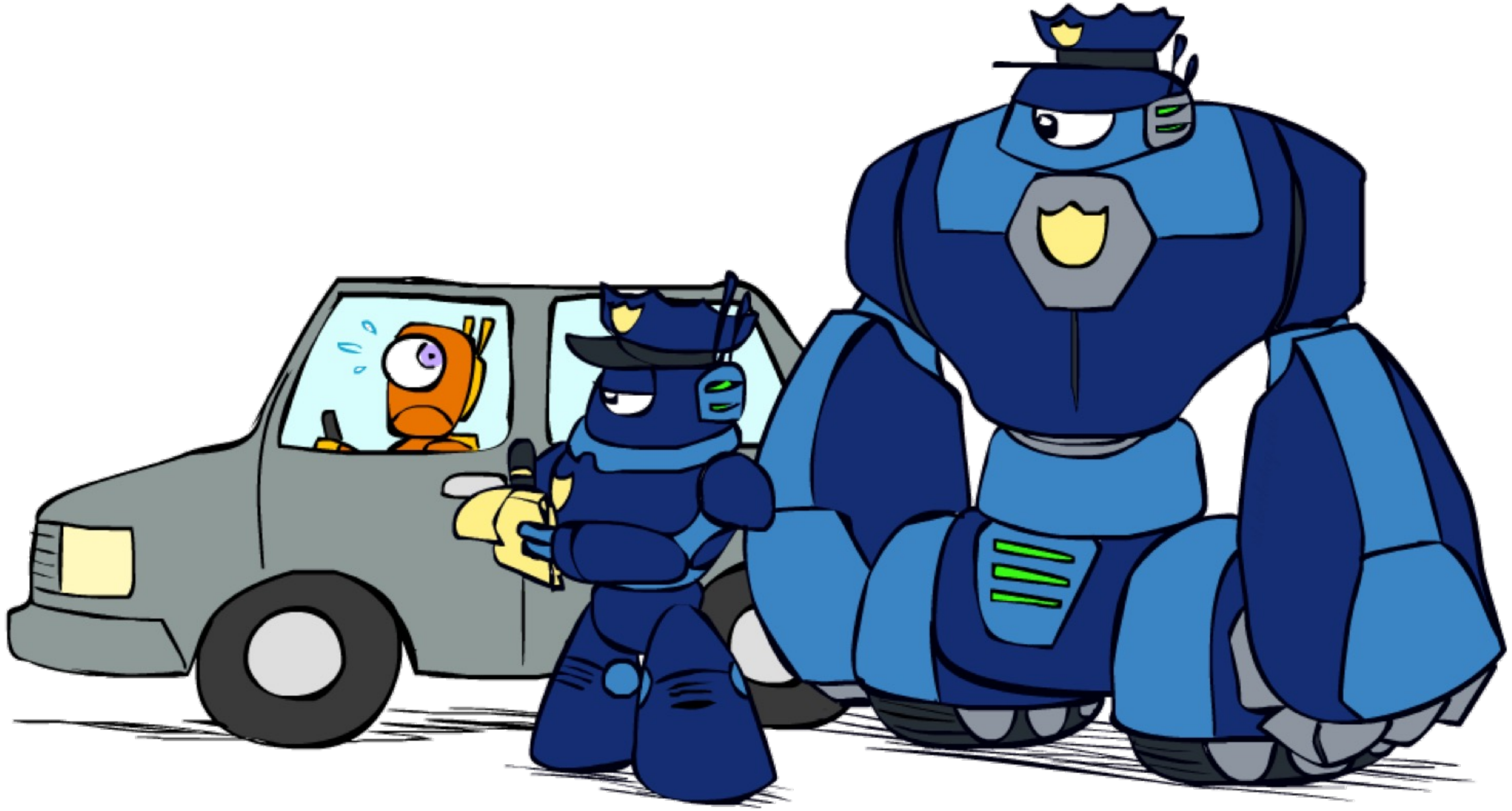
- Arc consistency still runs inside a backtracking search!





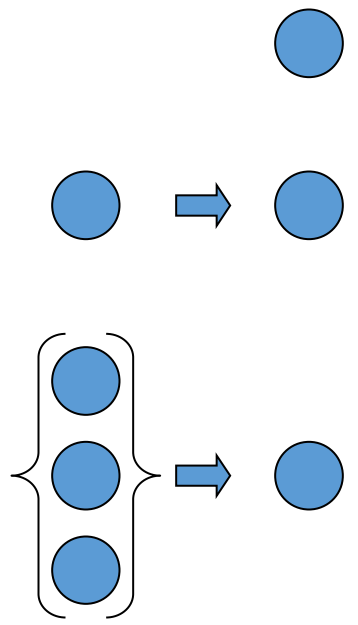
What went wrong here?

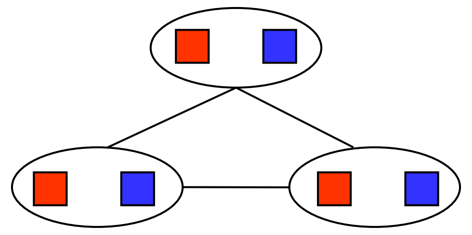
K-Consistency



K-Consistency

- Increasing degrees of consistency
  - 1-Consistency (Node Consistency): Each single node's domain has a value which meets that node's unary constraints
  - 2-Consistency (Arc Consistency): For each pair of nodes, any consistent assignment to one can be extended to the other
  - K-Consistency: For each k nodes, any consistent assignment to k-1 can be extended to the kth node.
- Higher k more expensive to compute
- (You need to know the k=2 case: arc consistency)





Strong K-Consistency

- Strong k-consistency: also k-1, k-2, … 1 consistent

- Claim: strong n-consistency means we can solve without backtracking!

- Why?
  - Choose any assignment to any variable
  - Choose a new variable
  - By 2-consistency, there is a choice consistent with the first
  - Choose a new variable
  - By 3-consistency, there is a choice consistent with the first 2
  - …

- Lots of middle ground between arc consistency and n-consistency! (e.g. k=3, called path consistency)

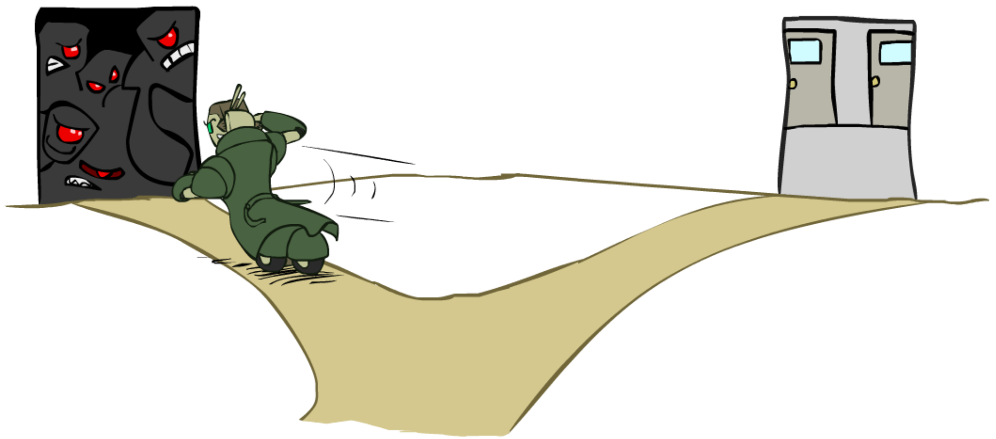
Ordering

Ordering: Minimum Remaining Values

- Variable Ordering: Minimum remaining values (MRV):
  - Choose the variable with the fewest legal left values in its domain

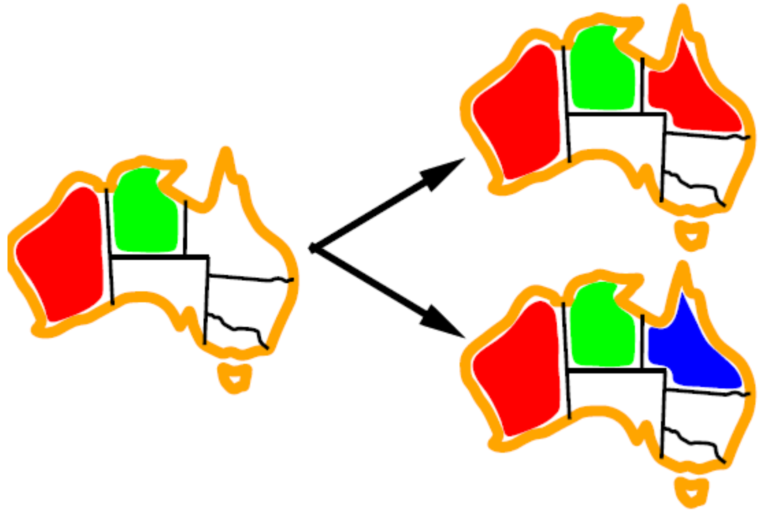
WA
NT
SA
Q
NSW
V

- Why min rather than max?
- Also called "most constrained variable"
- "Fail-fast" ordering



Ordering: Least Constraining Value

- Value Ordering: Least Constraining Value
  - Given a choice of variable, choose the *least constraining value*
  - I.e., the one that rules out the fewest values in the remaining variables
  - Note that it may take some computation to determine this! (E.g., rerunning filtering)



- Why least rather than most?

- Combining these ordering ideas makes 1000 queens feasible