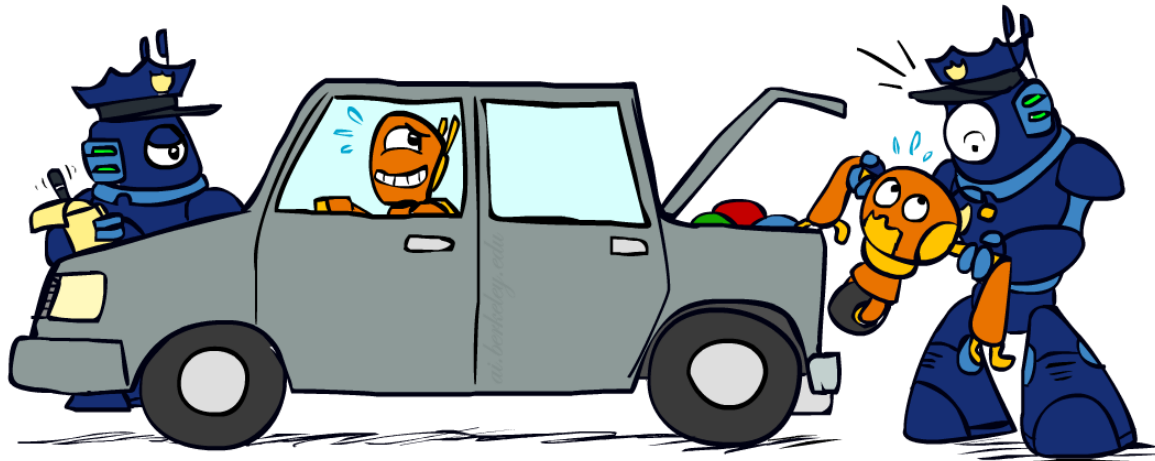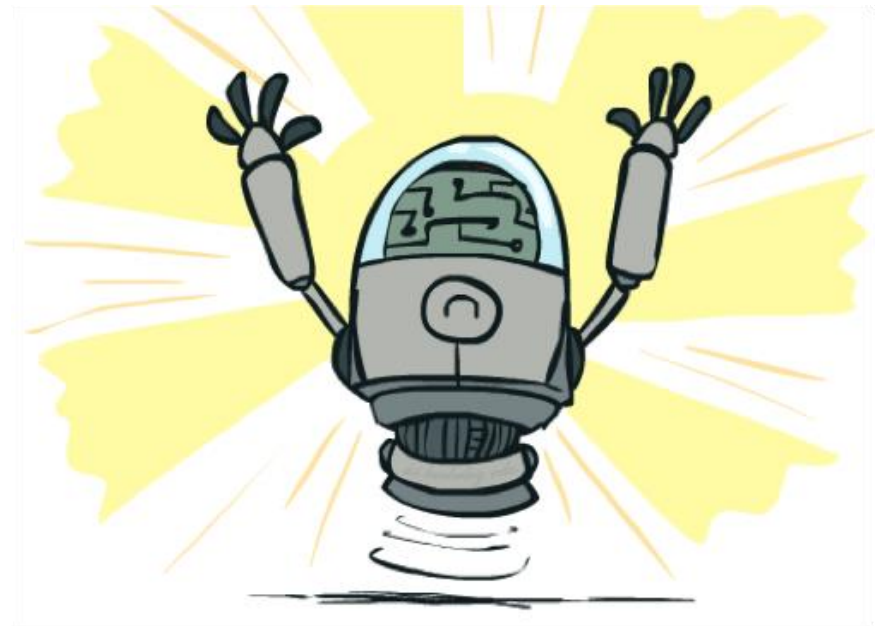# Constraint Satisfaction Problems II
Filtering, ordering

Improving Backtracking

- General-purpose ideas give huge gains in speed

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

- Filtering: Can we detect inevitable failure early?

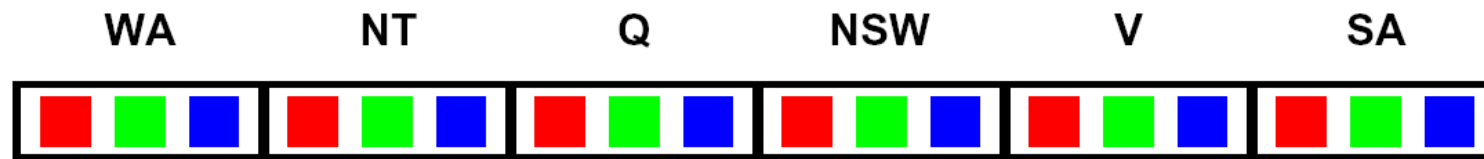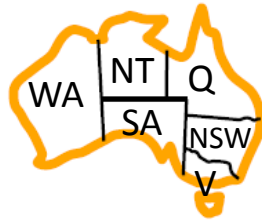- Structure: Can we exploit the problem structure?

Filtering

Forward Checking

- Filtering: Keep track of domains for unassigned variables and cross off bad options
- Forward checking: Cross off values that violate a constraint when added to the existing assignment

| WA	NT	Q	NSW	V	SA
🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦

Filtering: Constraint Propagation

- Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures:
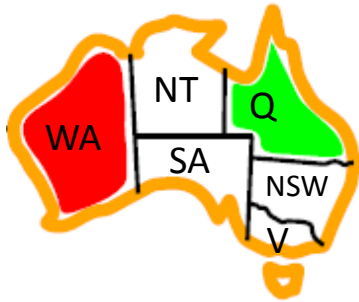
- NT and SA cannot both be blue!
- Why didn't we detect this yet?
- *Constraint propagation:* reason from constraint to constraint

Consistency of A Single Arc

- 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
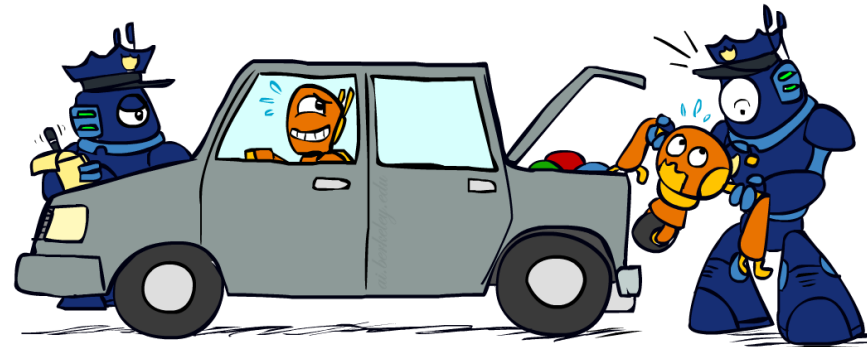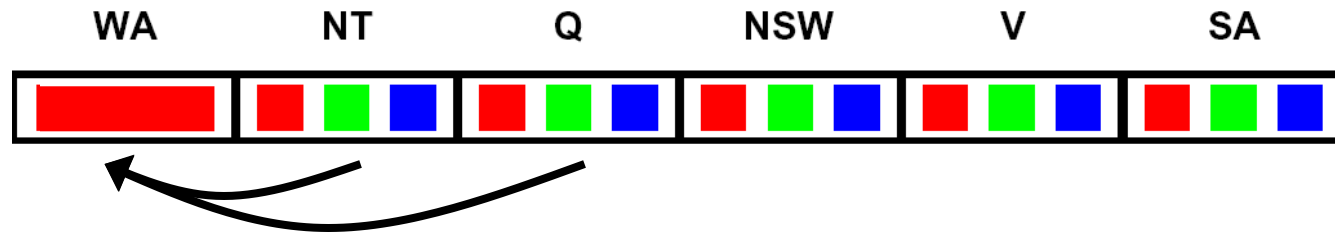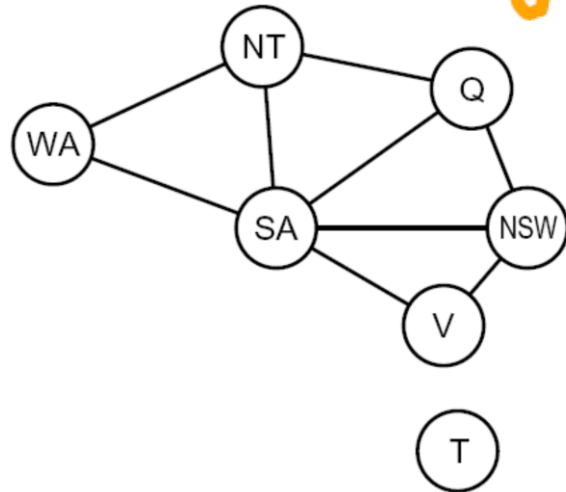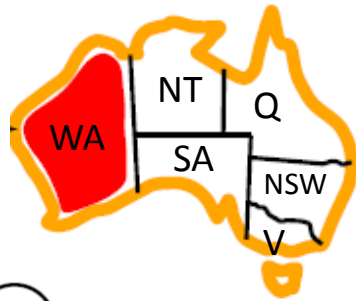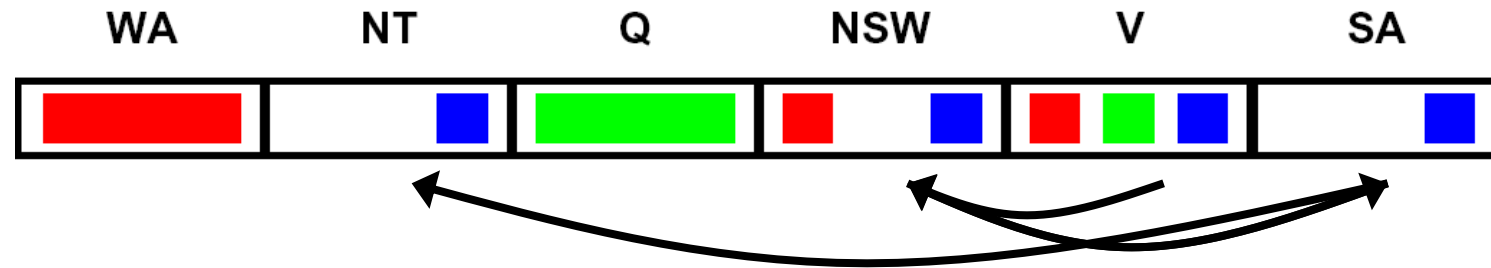

Delete from the tail!

- Forward checking: Enforcing consistency of arcs pointing to each new assignment

Arc Consistency of an Entire CSP

- A simple form of propagation makes sure **all** arcs are consistent:
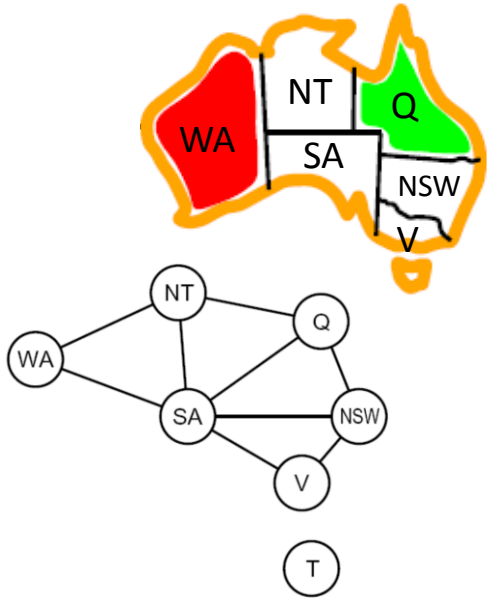
- Important: If X loses a value, neighbors of X need to be rechecked!
- Arc consistency detects failure earlier than forward checking
- Can be run as a preprocessor or after each assignment
- What's the downside of enforcing arc consistency?

Remember: Delete from the tail!

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: $O(n^2d^3)$, can be reduced to $O(n^2d^2)$
- … but detecting all possible future problems is NP-hard – why?

[Demo: CSP applet (made available by aispace.org) -- n-queens]

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!
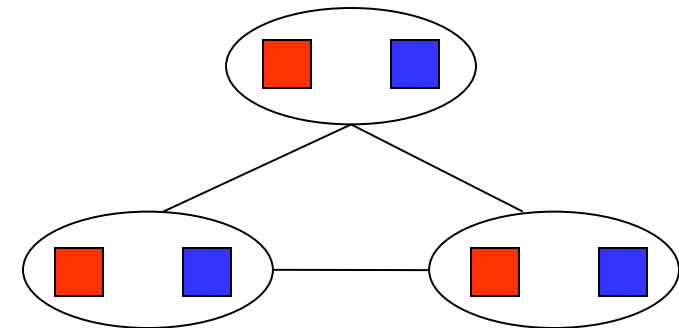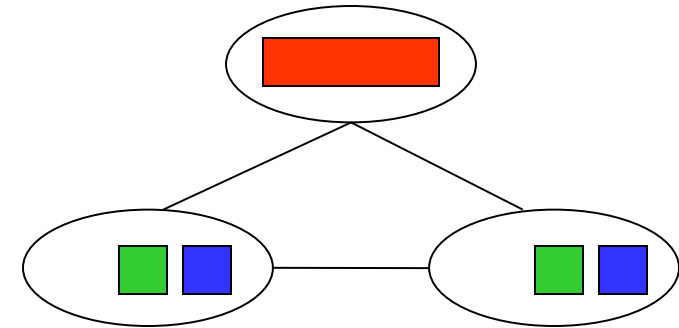
What went wrong here?

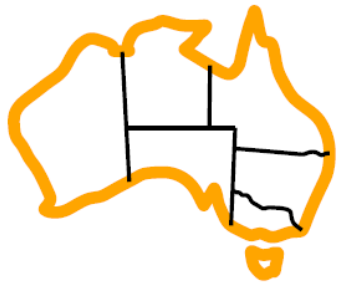[Demo: coloring -- forward checking]

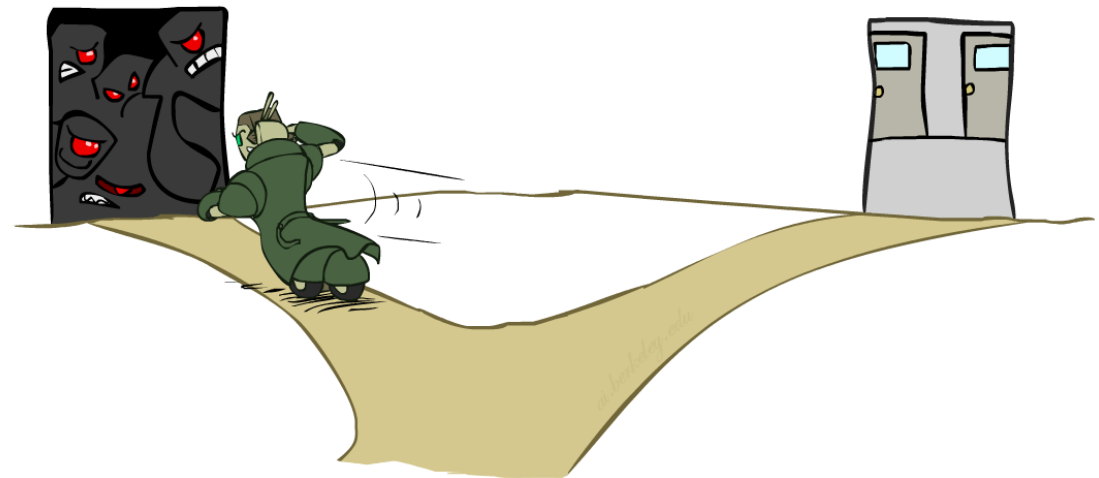[Demo: coloring -- arc consistency]

Ordering

Ordering: Minimum Remaining Values

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

- 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**