Constraint Satisfaction Problems

Russell and Norvig: Chapter 6
CSE 240: Winter 2023
Lecture 10
Guest Lecturer: Prof. Razvan Marinescu

Announcements

- This week: Prof. Marinescu will lecture (Prof. Gilpin at AAAI)
- Assignment 3 is out.
- Prof. Gilpin will still hold remote office hours.

Agenda and Topics

- Constraint Satisfaction Problems
  - Solving CSPs

Standard Search Formulation

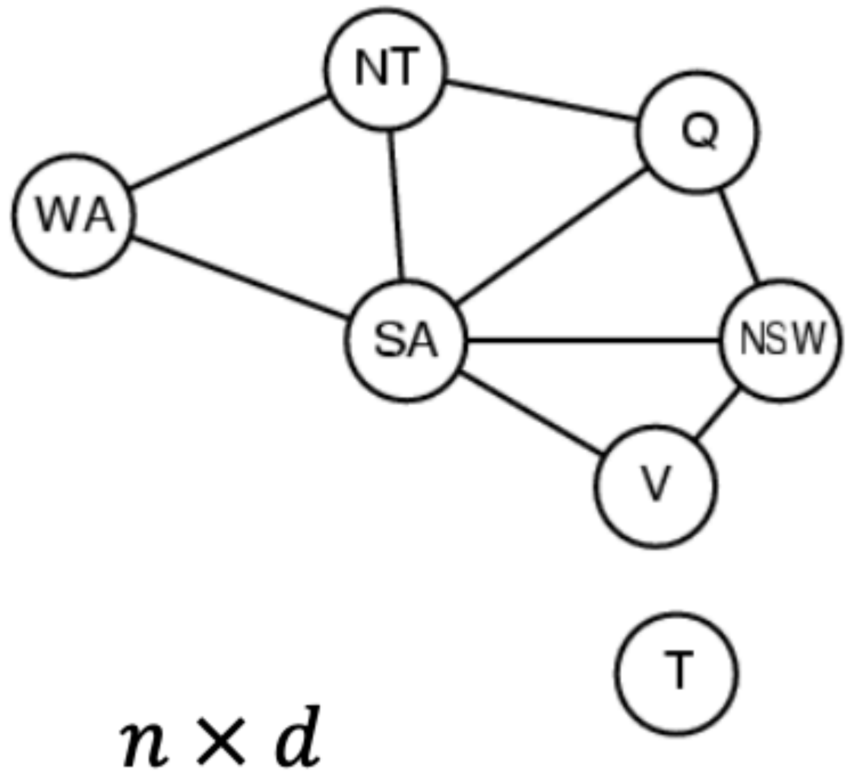
- Standard search formulation of CSPs
- States defined by the values assigned so far (partial assignments)
  - Initial state: the empty assignment, {}
  - Successor function (Actions): assign a value to an unassigned variable
  - Goal test: the current assignment is complete (all variables have assigned values) and consistent (satisfies all constraints)
  - Path cost: not important
- We'll start with the straightforward, naïve approach, then improve it

Properties of CSP

- Every solution appears at depth n with n variables
- Which search algorithm is appropriate?
  - Depth-limited search
- Branching factor is nd at the top level, $b = (n - l)d$ at depth l, hence there are $n!d^n$ leaves
- However there are only n^d complete assignments.

Assignment

When assigning values to variables, we reach the same partial assignment regardless of the order of the variables.

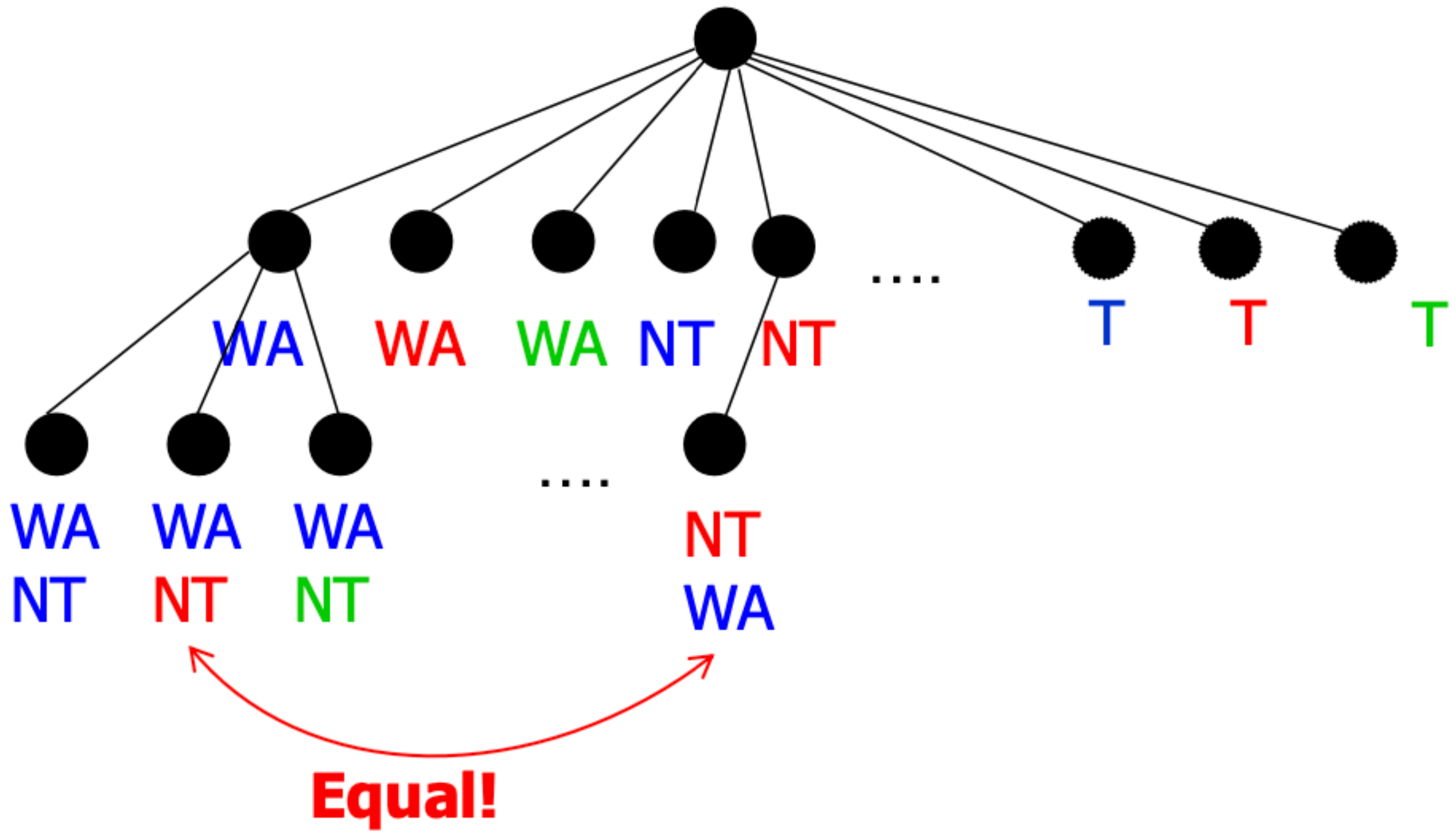


$n \times d$

$(n \times d) \times ((n-1) \times d)$

…

$n! \times d^n$

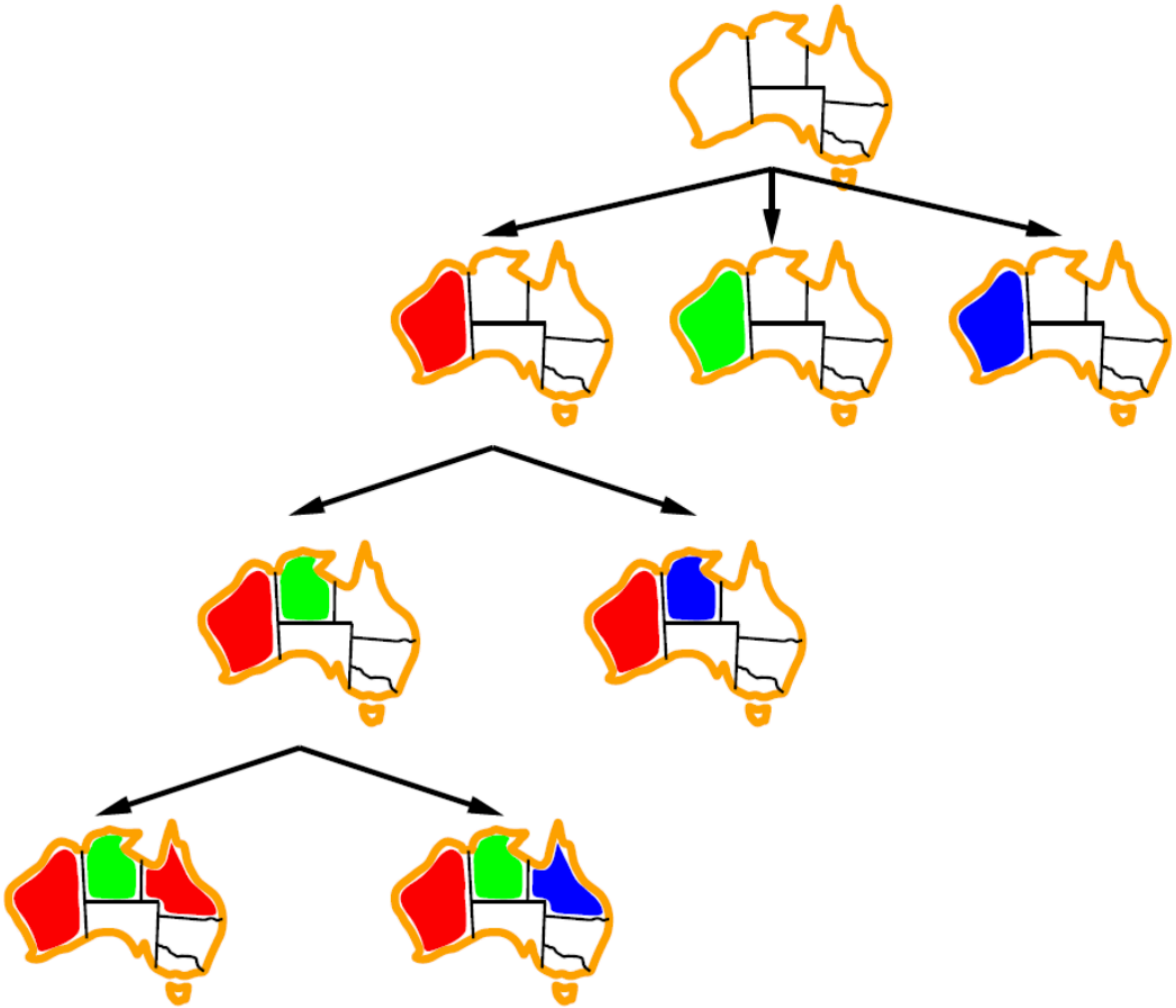


Backtracking Search

Backtracking Search

- Idea 1: Only consider a single variable at each point
  - Variable assignments are commutative, so fix ordering
  - I.e., [WA = red then NT = green] same as [NT = green then WA = red]
  - Only need to consider assignments to a single variable at each step
- Idea 2: Only allow legal assignments at each point
  - I.e. consider only values which do not conflict previous assignments
  - Might have to do some computation to figure out whether a value is ok
  - "Incremental goal test"
- Depth-first search for CSPs with these two improvements is called **backtracking search**
- Backtracking search is the basic uninformed algorithm for CSPs
- Can solve n-queens for $n \approx 25$

Backtracking Example



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 ← 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 ← RECURSIVE-BACKTRACKING(assignment, csp)
            if result ≠ failure then return result
            remove {var = value} from assignment
    return failure
```

- Backtracking = DFS + variable-ordering + fail-on-violation
- What are the choice points?

Improving Backtracking

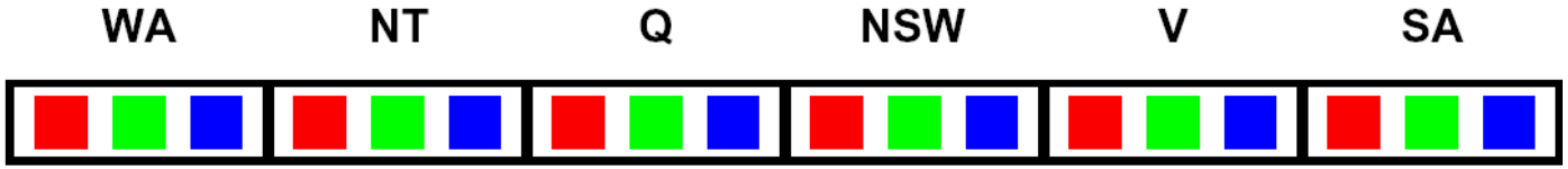
- General-purpose ideas give huge gains in speed
- Filtering: Can we detect inevitable failure early?
- Ordering:
  - Which variable should be assigned next?
  - In what order should its values be tried?

Filtering

Keep track of domains for unassigned variables and cross off bad options

Filtering: Forward Checking

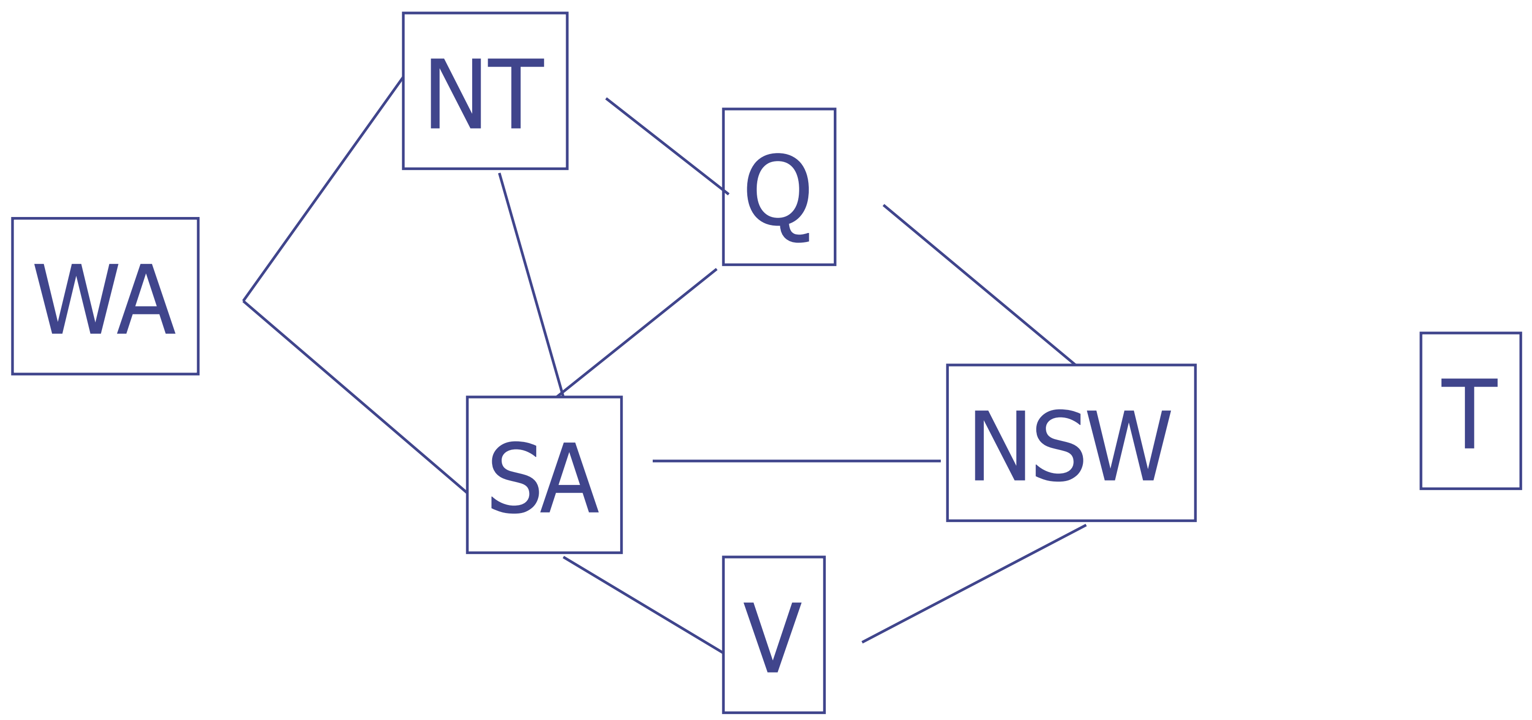
- Filtering: Keep track of domains for unassigned variables and cross off bad options
- Forward checking: Cross off values that violate a constraint when added to the existing assignment



Forward Checking

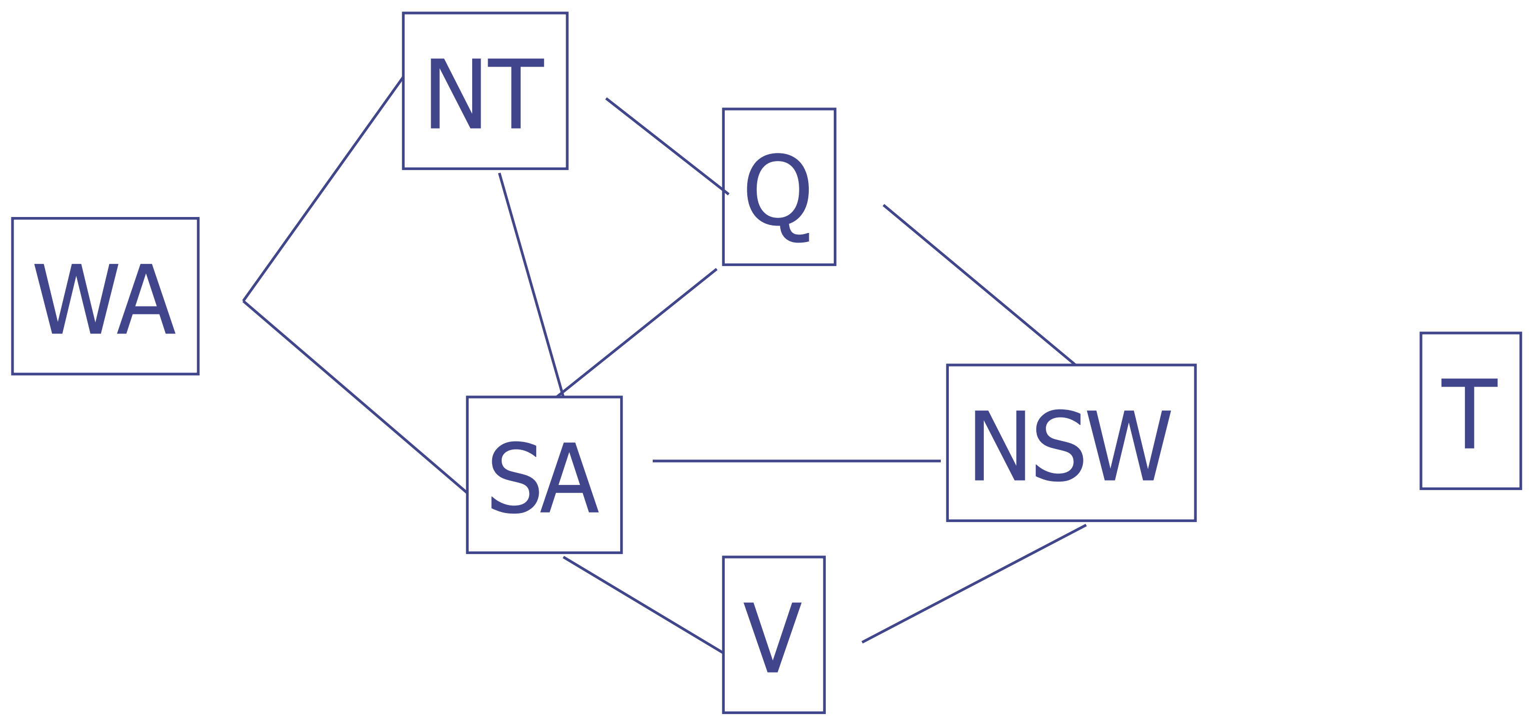
After a variable X is assigned a value v, look at each unassigned variable Y that is connected to X by a constraint and deletes from Y's domain any value that is inconsistent with v

Map Coloring: FC



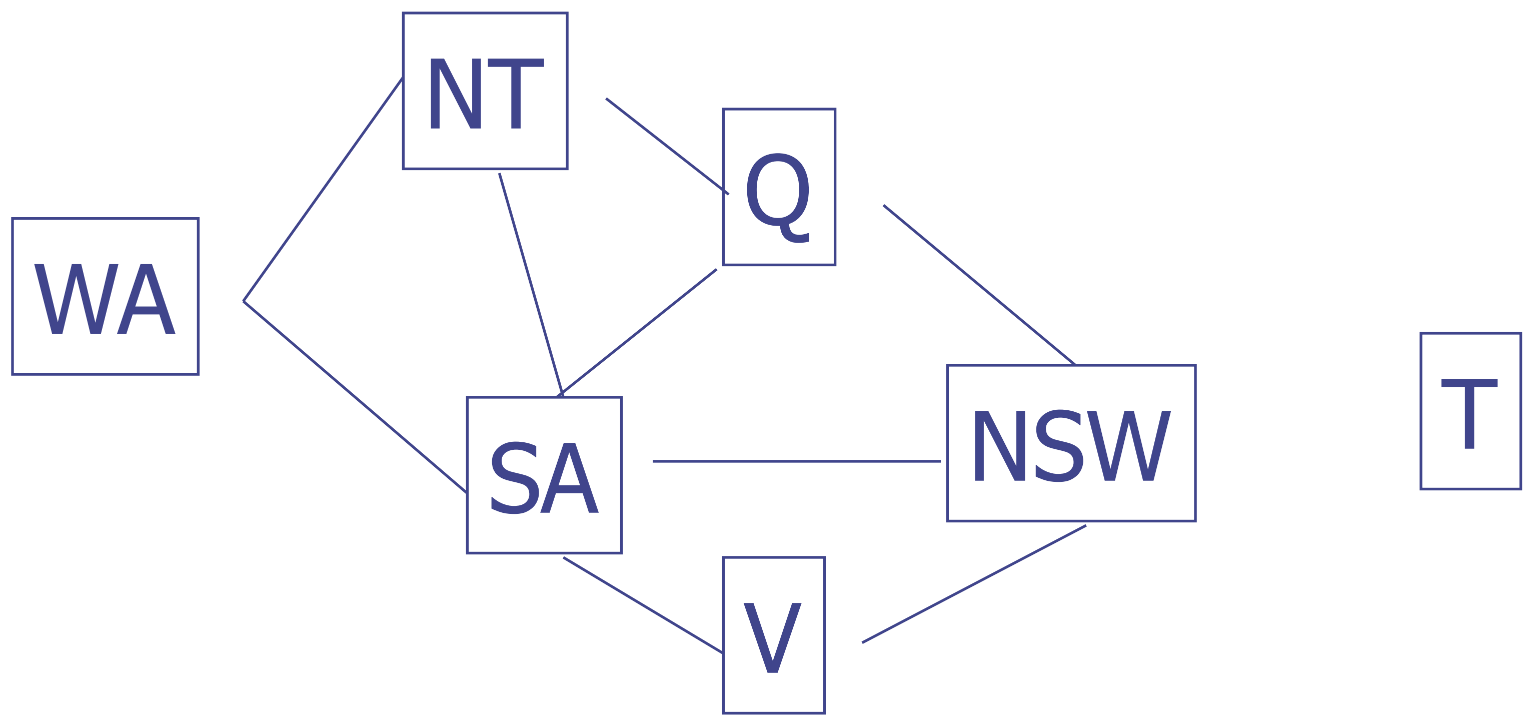
WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB

Map Coloring: FC



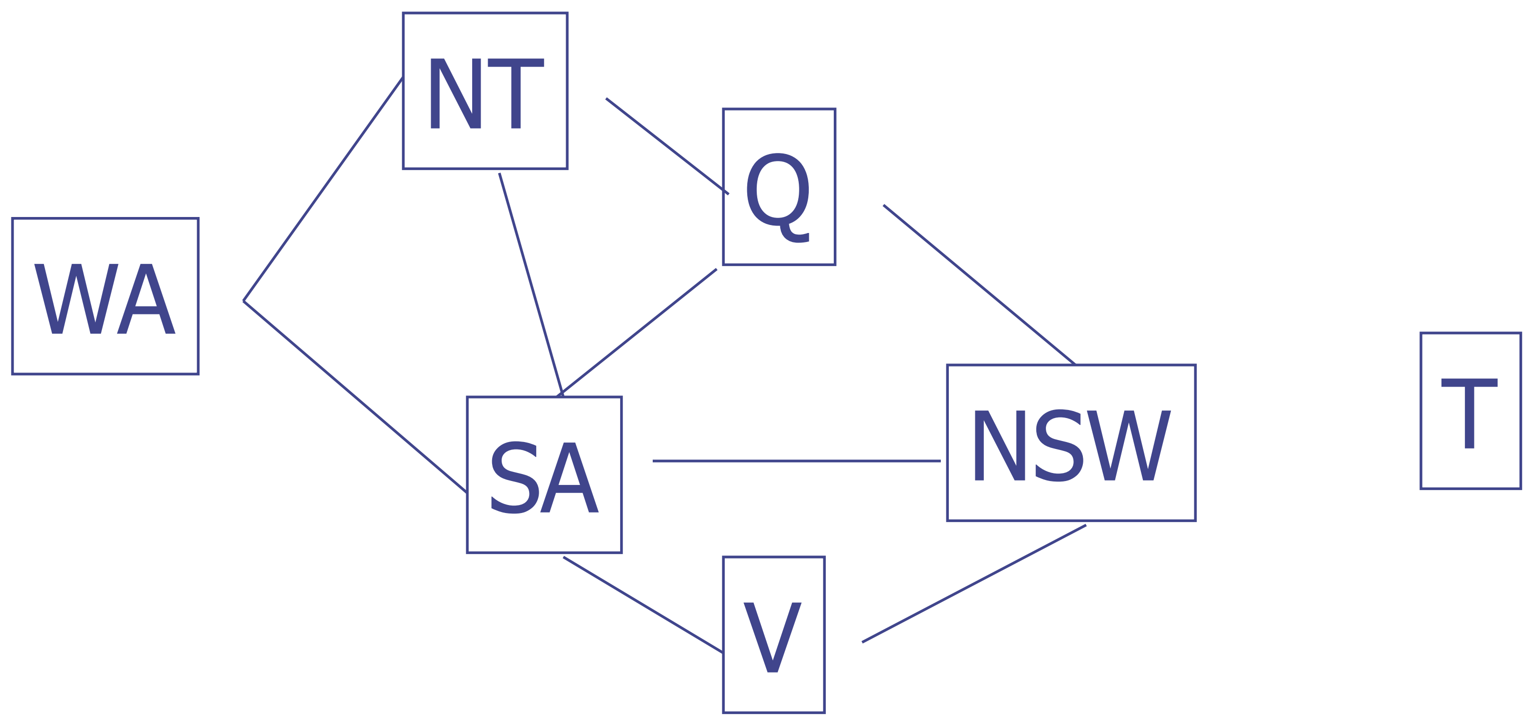
WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
1: R	RGB	RGB	RGB	RGB	RGB	RGB

Map Coloring: FC



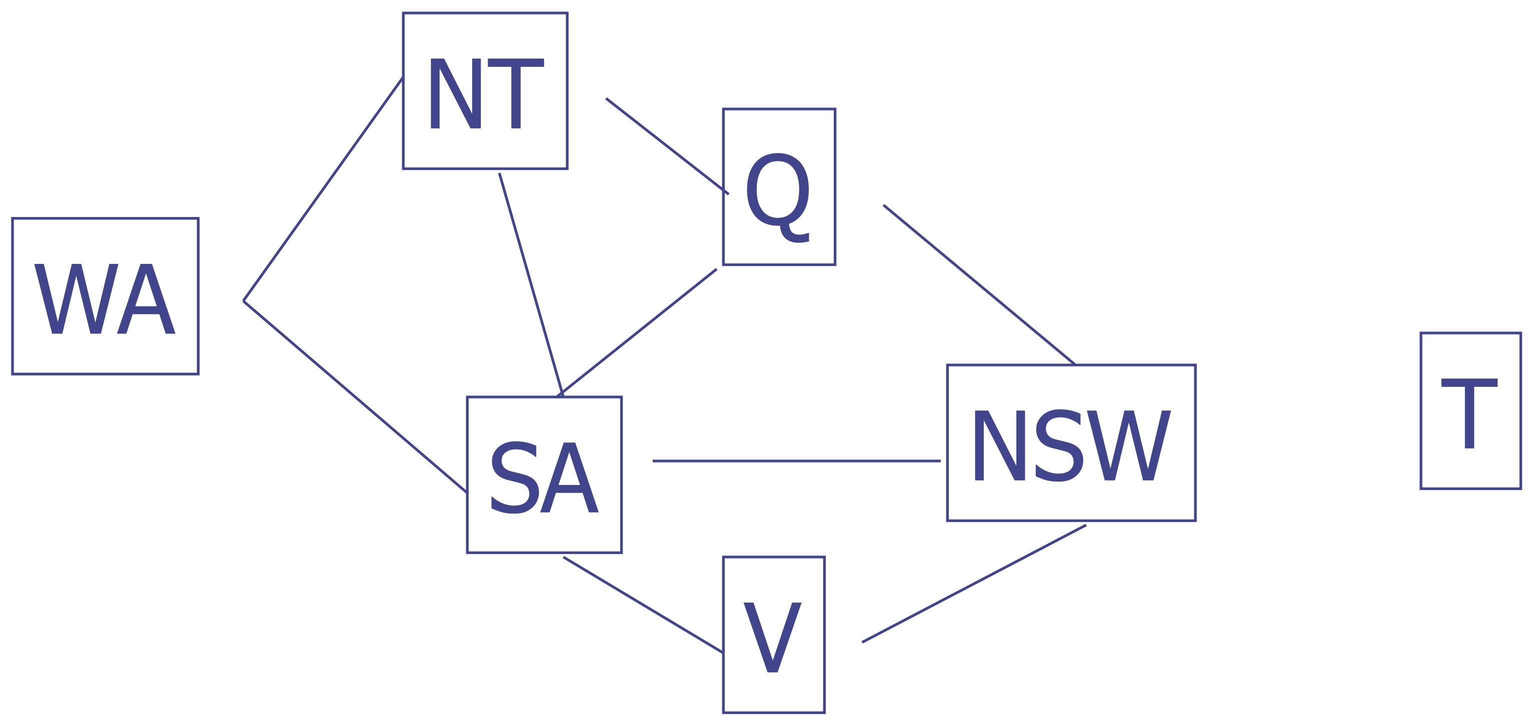
WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
1: R	~~R~~GB	RGB	RGB	RGB	~~R~~GB	RGB

Map Coloring: FC



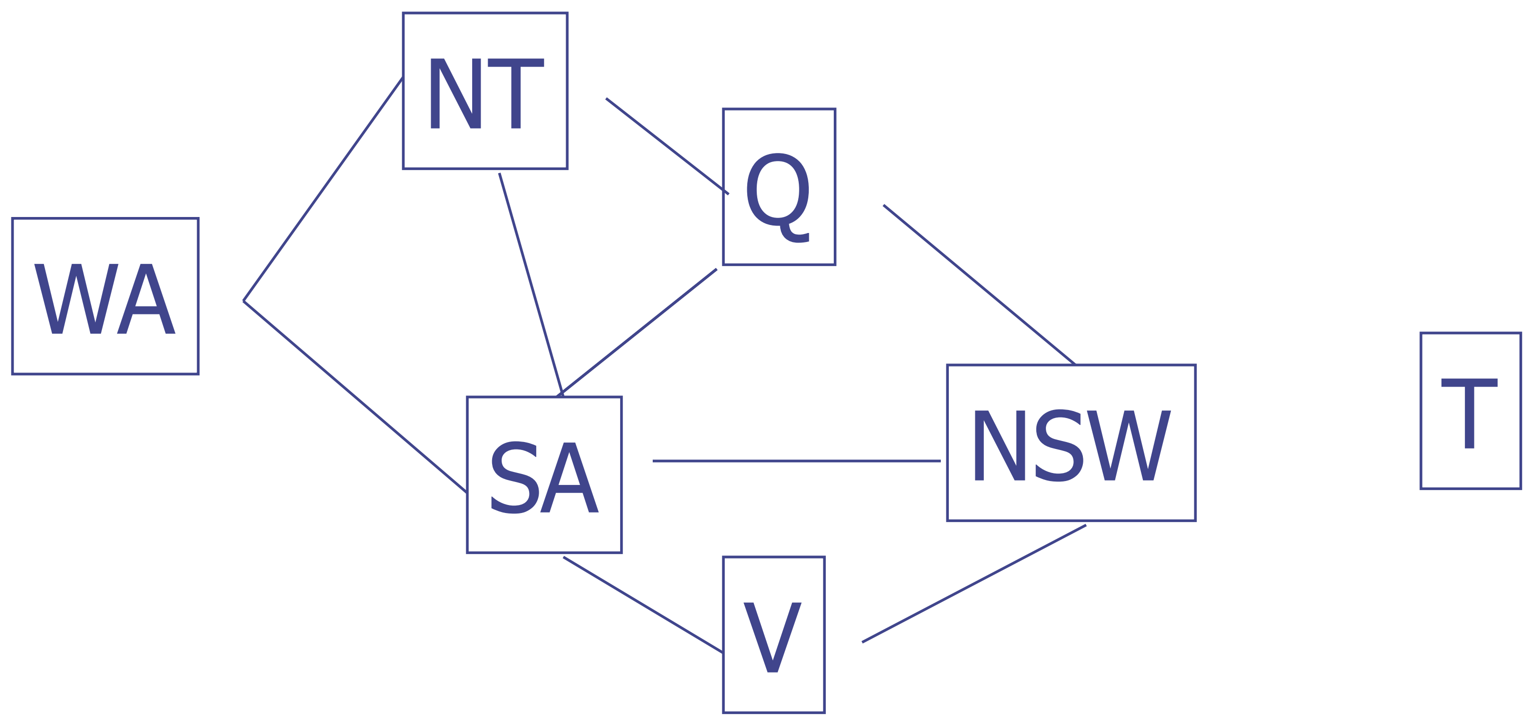
WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	~~R~~GB	RGB	RGB	RGB	~~R~~GB	RGB
R	~~R~~GB	2: G	RGB	RGB	~~R~~GB	RGB

Map Coloring:FC



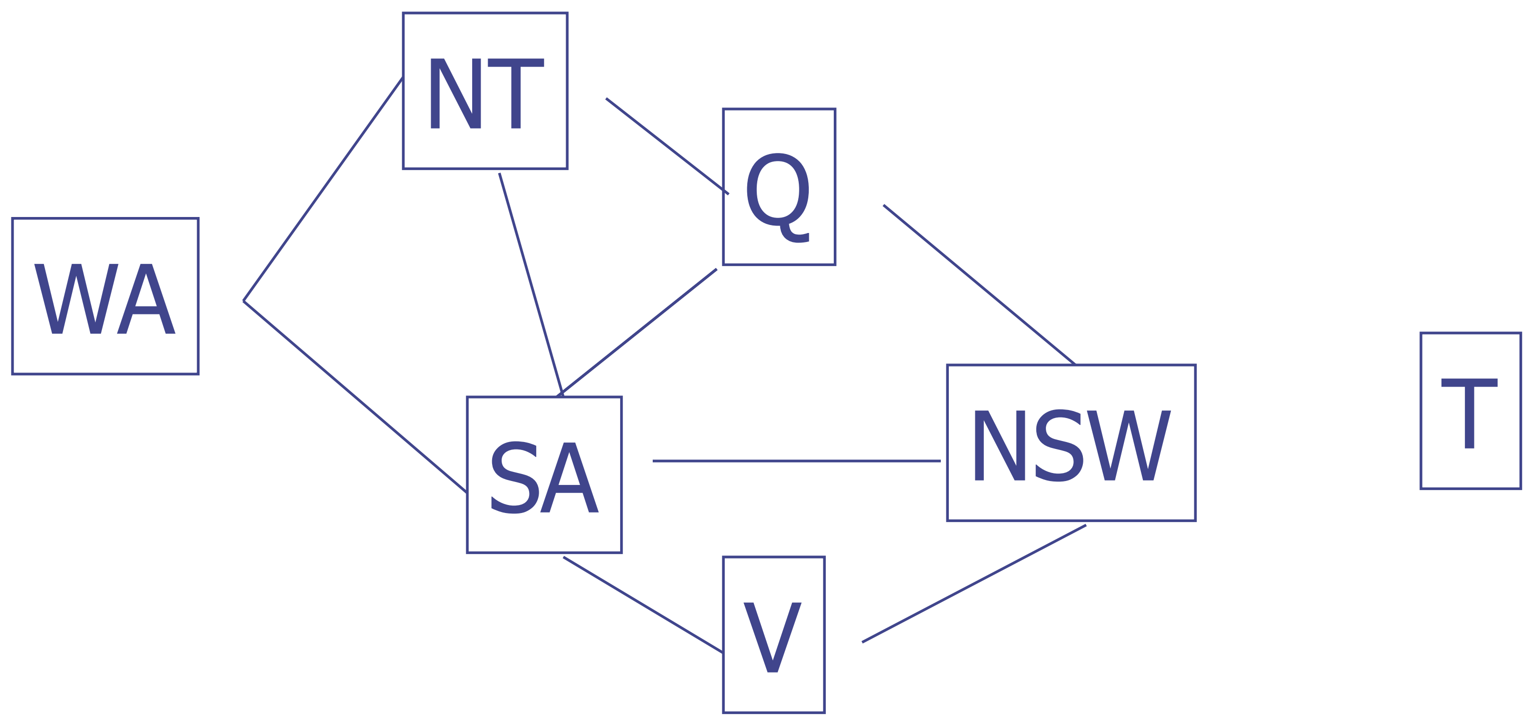
WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	~~R~~GB	RGB	RGB	RGB	~~R~~GB	RGB
R	~~R~~~~G~~B	2: G	R~~G~~B	RGB	~~R~~~~G~~B	RGB

Map Coloring: FC



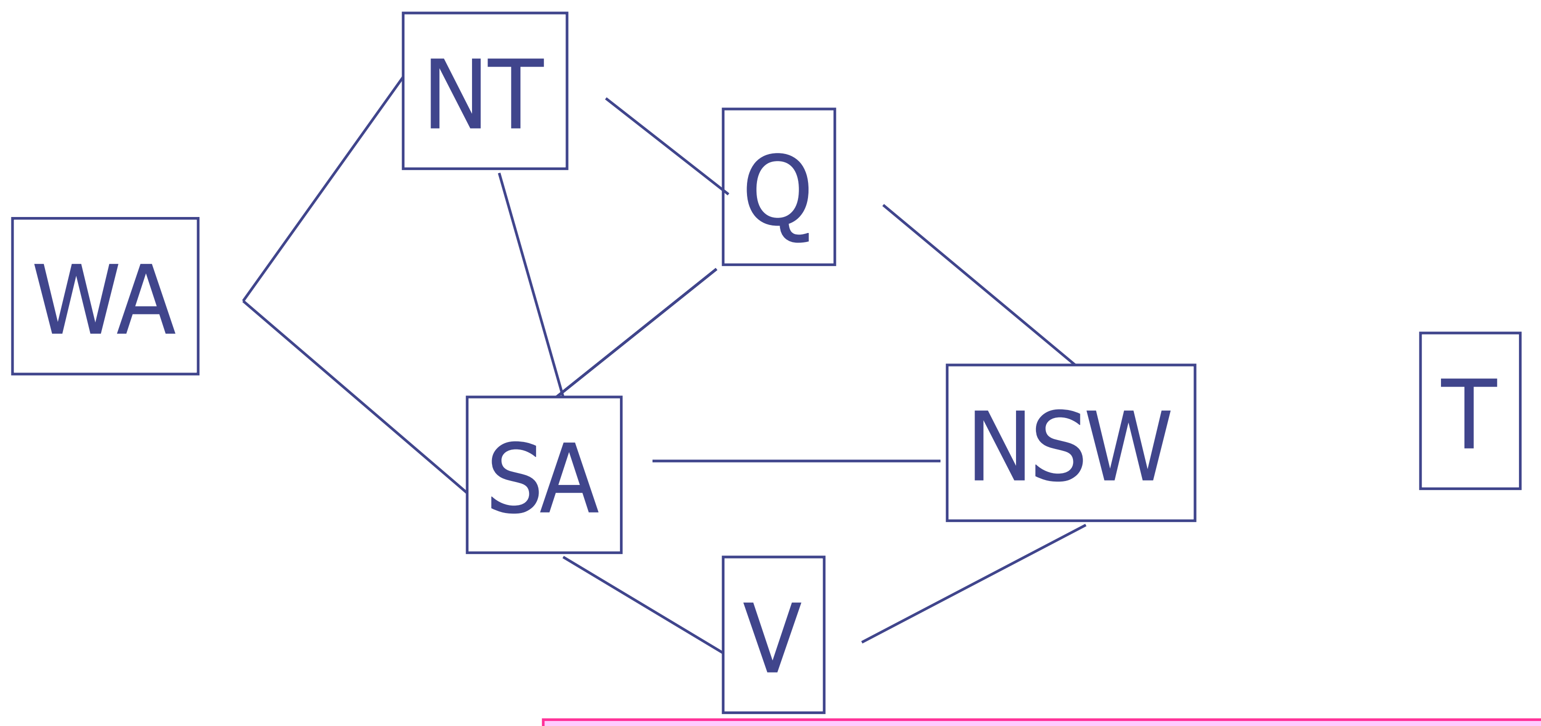
WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	~~R~~GB	RGB	RGB	RGB	~~R~~GB	RGB
R	~~R~~~~G~~B	G	R~~G~~B	RGB	~~R~~~~G~~B	RGB
R	~~R~~~~G~~B	G	R~~G~~B	3:B	~~R~~~~G~~B	RGB

Map Coloring: FC



WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	~~R~~GB	RGB	RGB	RGB	~~R~~GB	RGB
R	~~R~~~~G~~B	G	R~~G~~B	RGB	~~R~~~~G~~B	RGB
R	~~R~~~~G~~B	G	R~~G~~~~B~~	3:B	~~R~~~~G~~~~B~~	RGB

Map Coloring: FC



Impossible assignments that forward checking does not detect

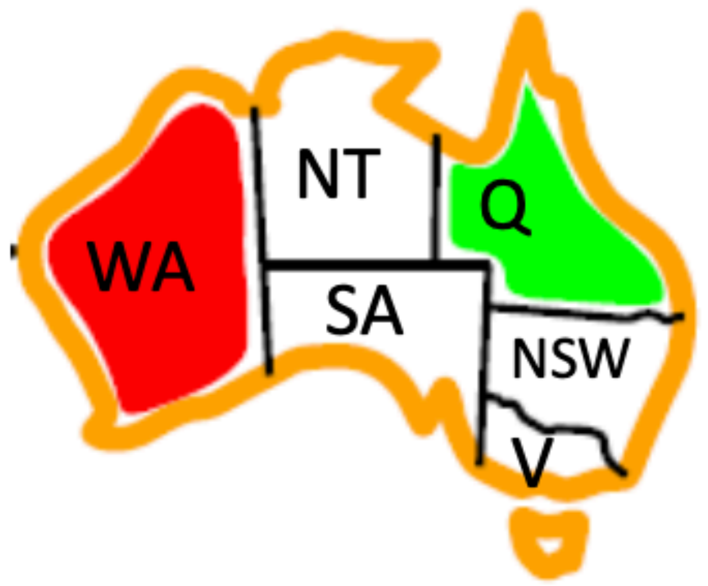
WA	NT	Q				
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	~~R~~GB	RGB	RGB	RGB	~~R~~GB	RGB
R	~~RG~~B	G	R~~G~~B	RGB	~~RG~~B	RGB
R	~~RG~~B	G	R~~GB~~	3:B	~~RGB~~	RGB

CE 10: What to Change

- What changes could we make to forward checking to make it more robust?

Filtering: Constraint Propagation

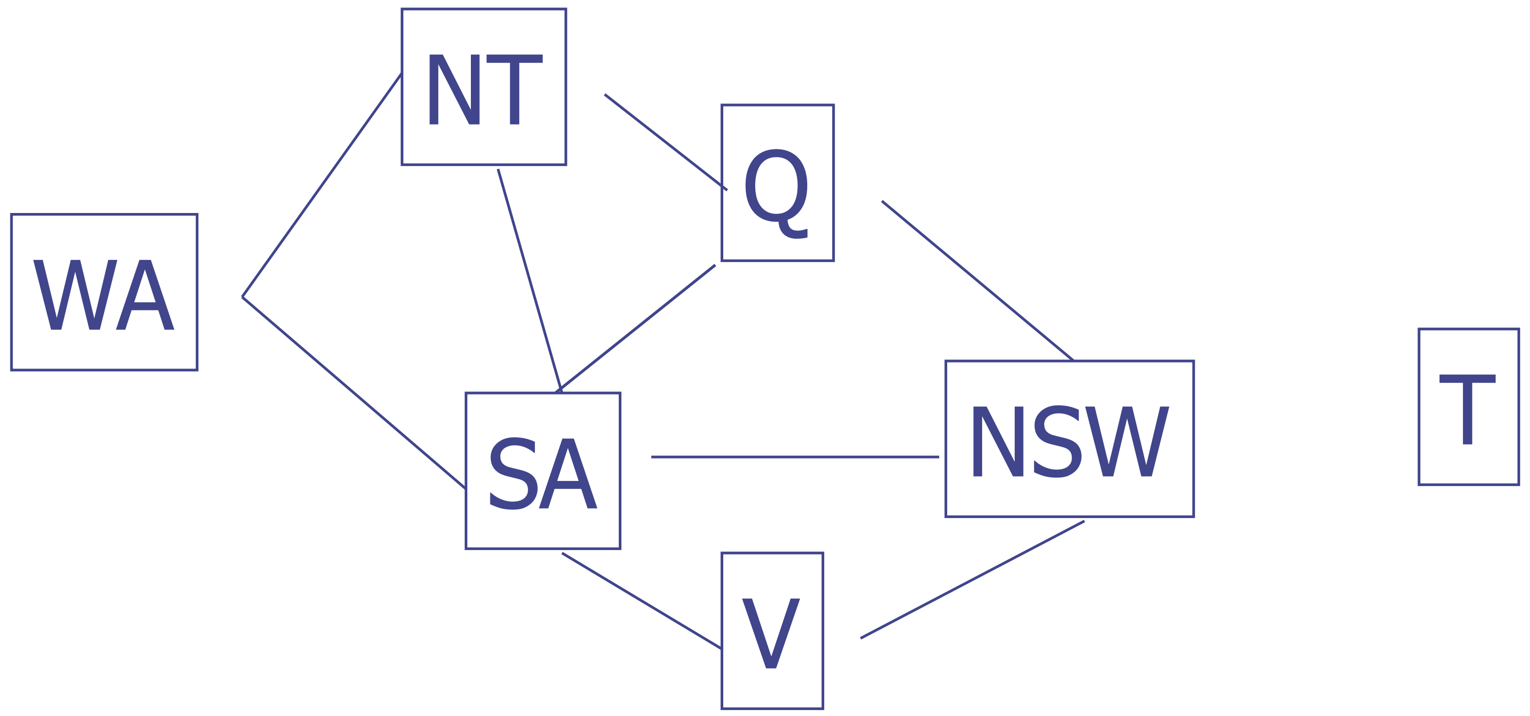
Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures:

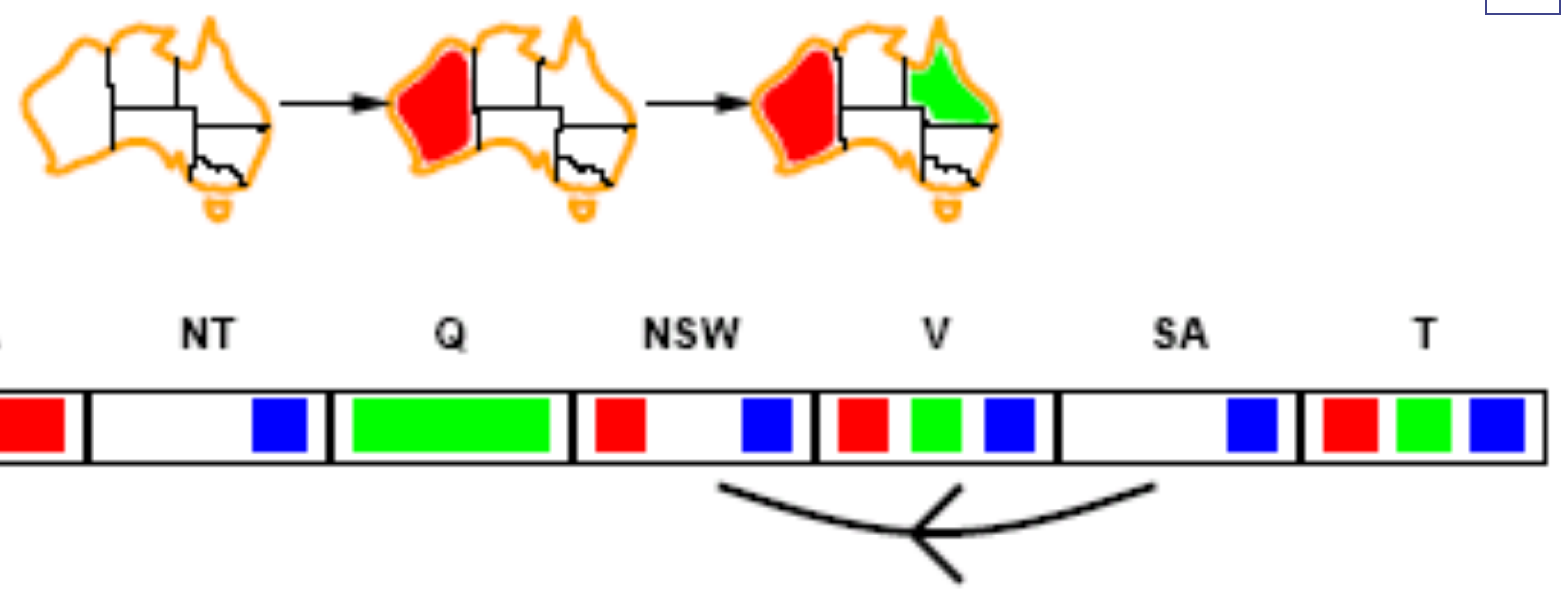


NT and SA cannot both be blue!

- Why didn’t we detect this yet?
- *Constraint propagation:* reason from constraint to constraint

Arc consistency





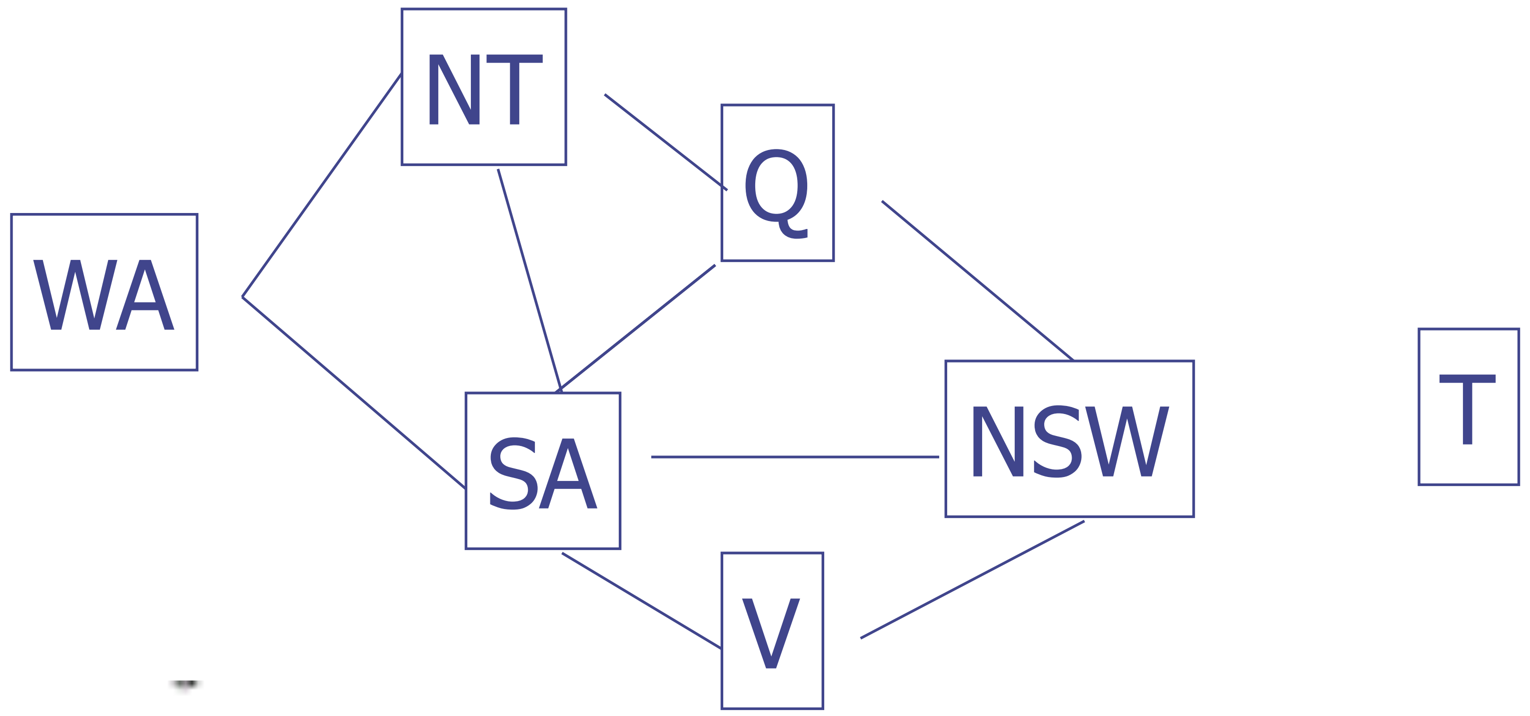
X → Y is consistent iff

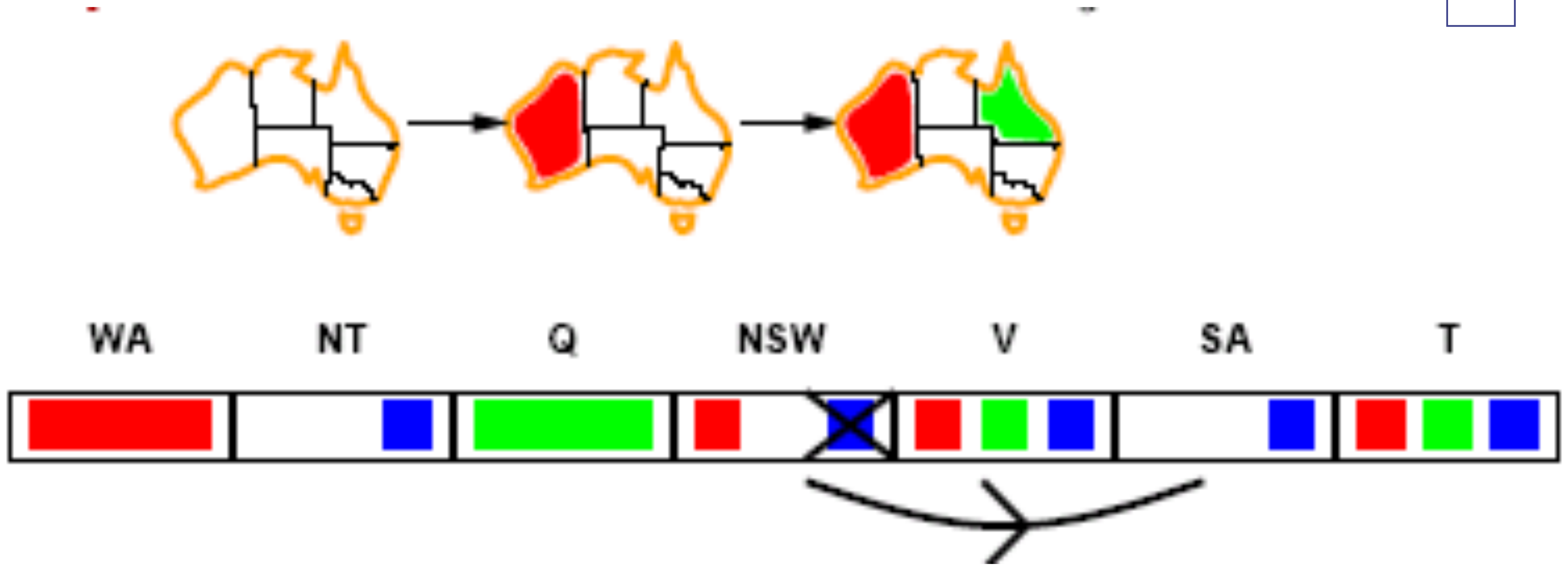
for every value x of X there is some allowed y

SA → NSW is consistent iff

SA=blue and NSW=red

Arc consistency





X → Y is consistent iff

for every value x of X there is some allowed y

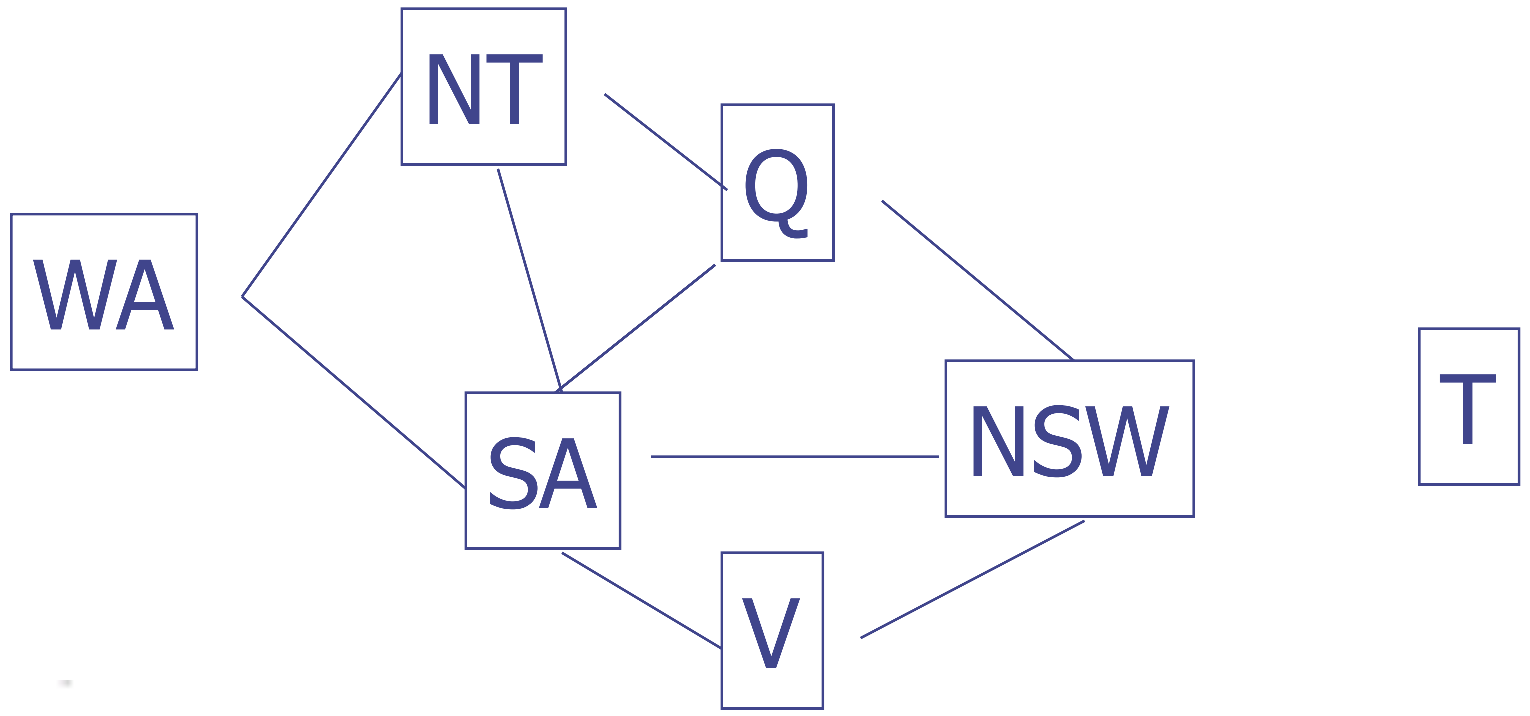
NSW → SA is consistent iff

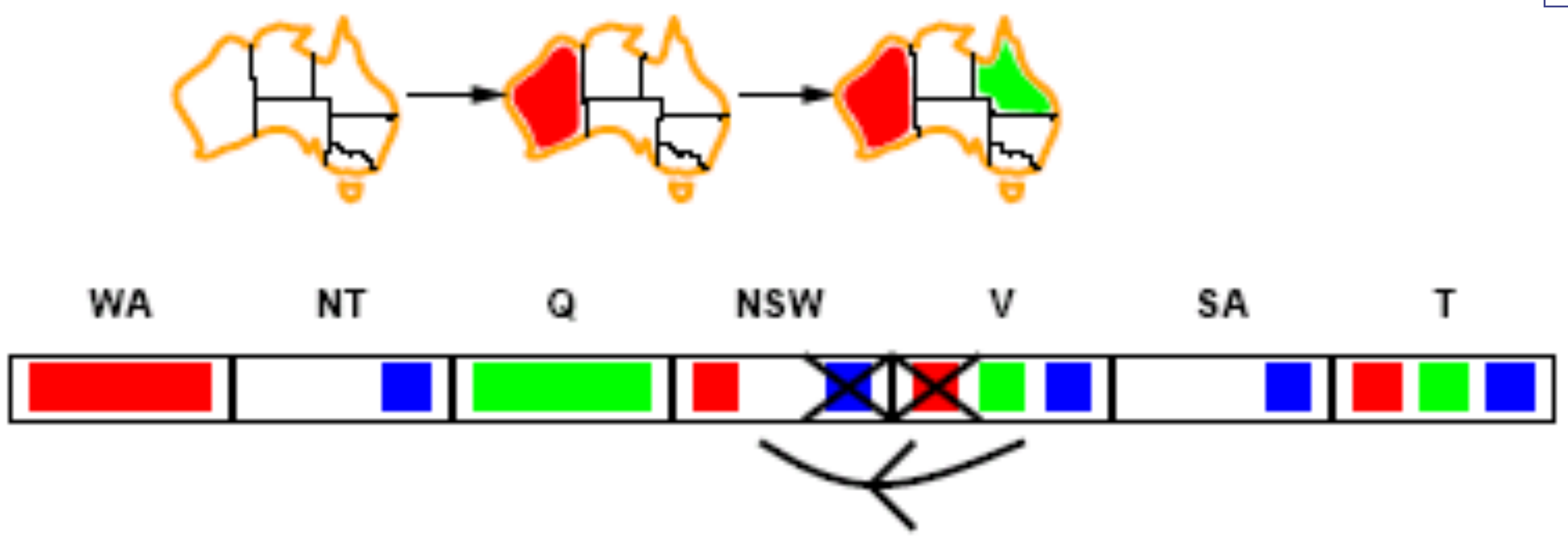
NSW=red and SA=blue

NSW=blue and SA=???

Arc can be made consistent by removing blue from NSW

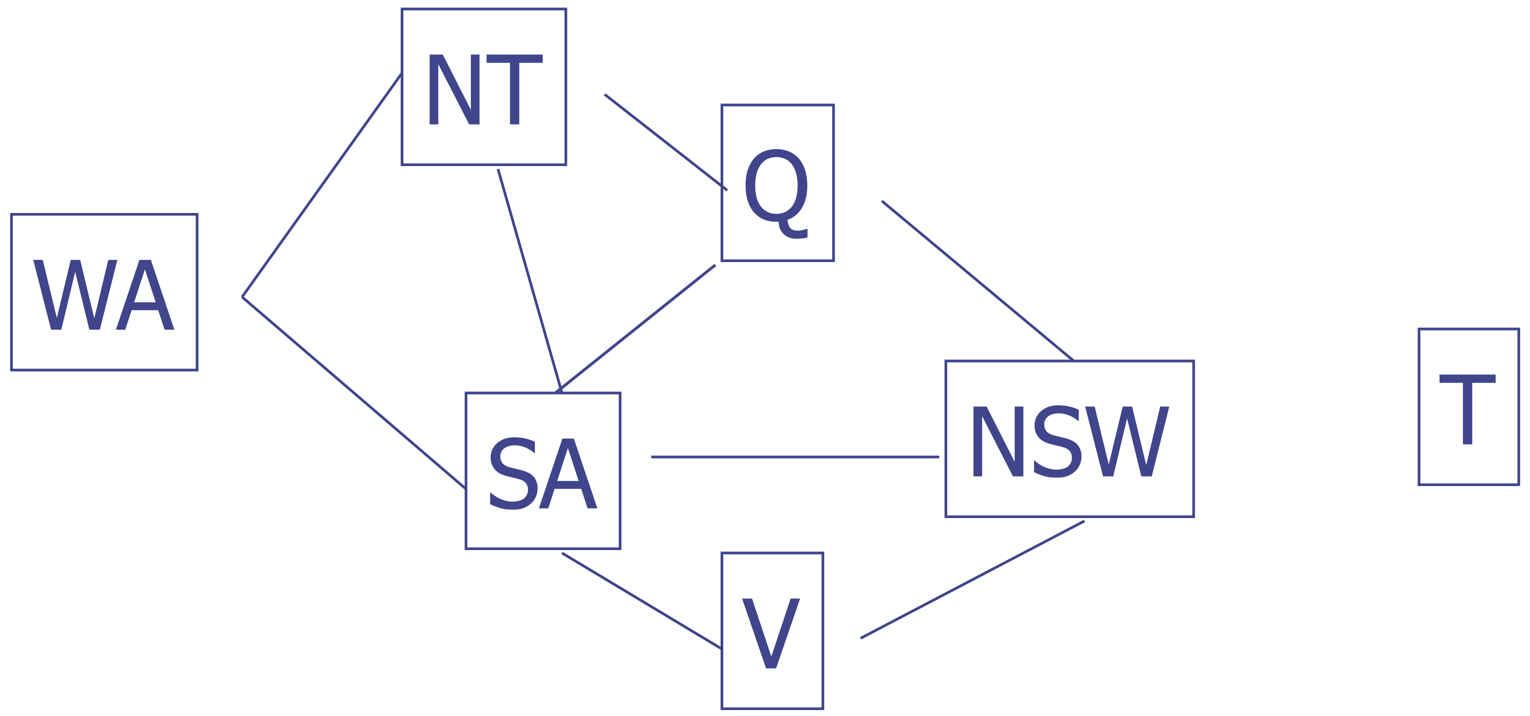
Arc consistency

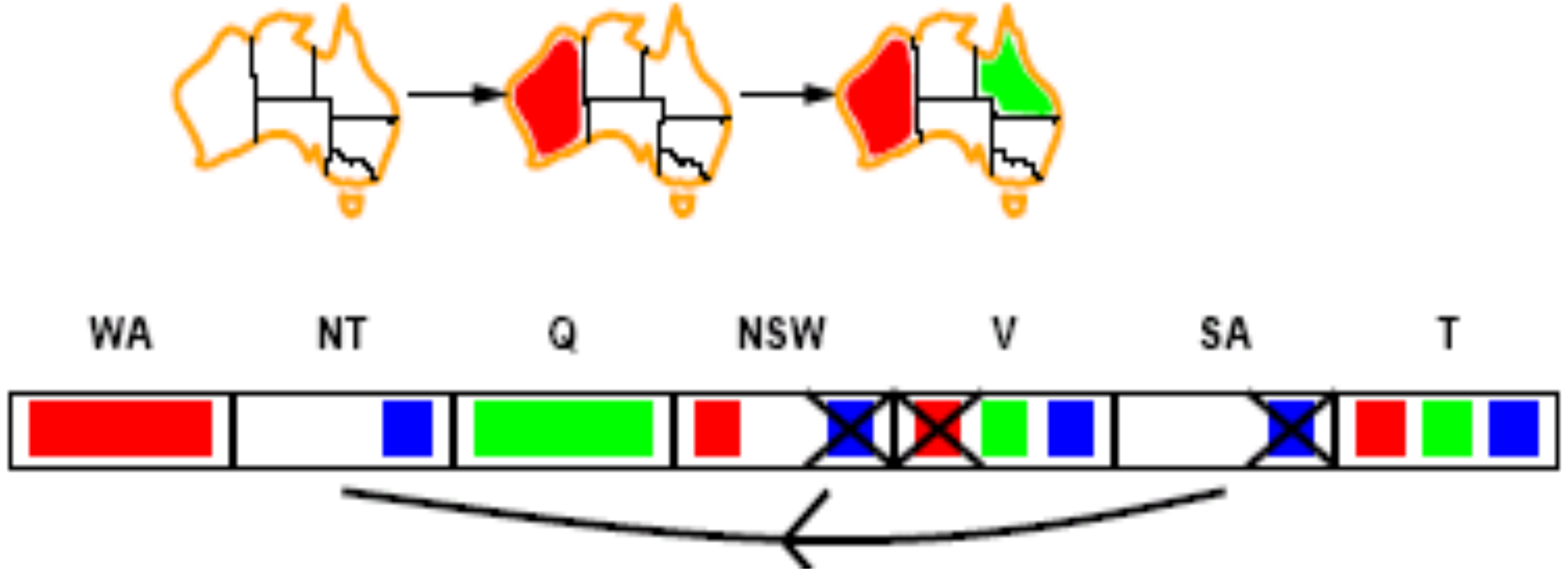




- Arc can be made consistent by removing blue from NSW
- RECHECK neighbors of NSW!!
  - Remove red from V

Arc consistency



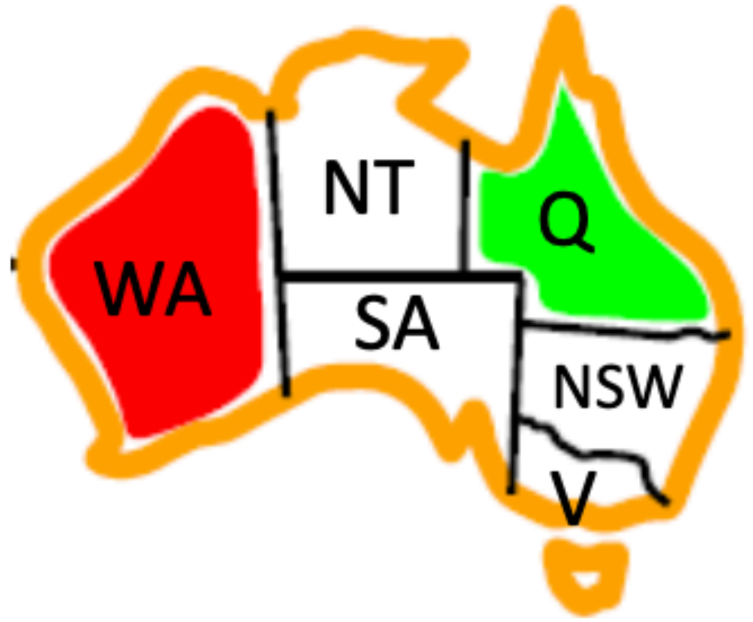


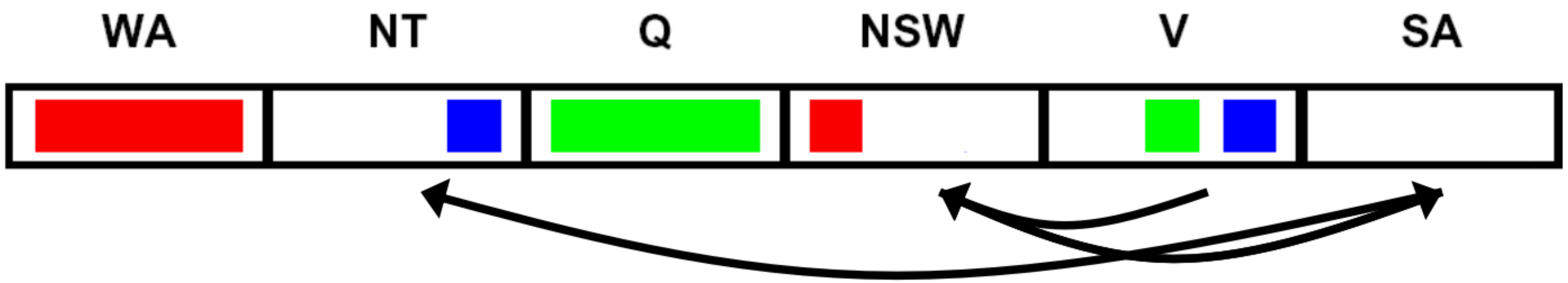
- Arc can be made consistent by removing blue from NSW
- RECHECK neighbors of NSW!!
  - Remove red from V
- Arc consistency detects failure earlier than FC
- Can be run as a preprocessor or after each assignment.
  - Repeated until no inconsistency remains

Arc Consistency Example

- X_i is arc consistent with respect to X_j if for every value in D_i there is a consistent value in D_j
- Example
  - Variables: $X = \{X_1, X_2\}$
  - Domain: $D_i = \{0,1,2,3,4,5,6,7,8,9\}$
  - Constraint: $X_1 = X_2^2$
- Is X_1 arc-consistent with respect to X_2?
  - No, to be arc-consistent $Domain(D_1) = \{0,1,4,9\}$
- Is X_2 arc-consistent with respect to X_1?
  - No, to be arc-consistent $Domain(D_2) = \{0,1,2,3\}$

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?

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: $\boxed{queue}$, 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 $\boxed{\text{REMOVE-INCONSISTENT-VALUES}(X_i,\ X_j)}$ **then**
      for each X_k **in** NEIGHBORS$[X_i]$ **do**
        add $\boxed{(X_k,\ X_i)}$ to *queue*

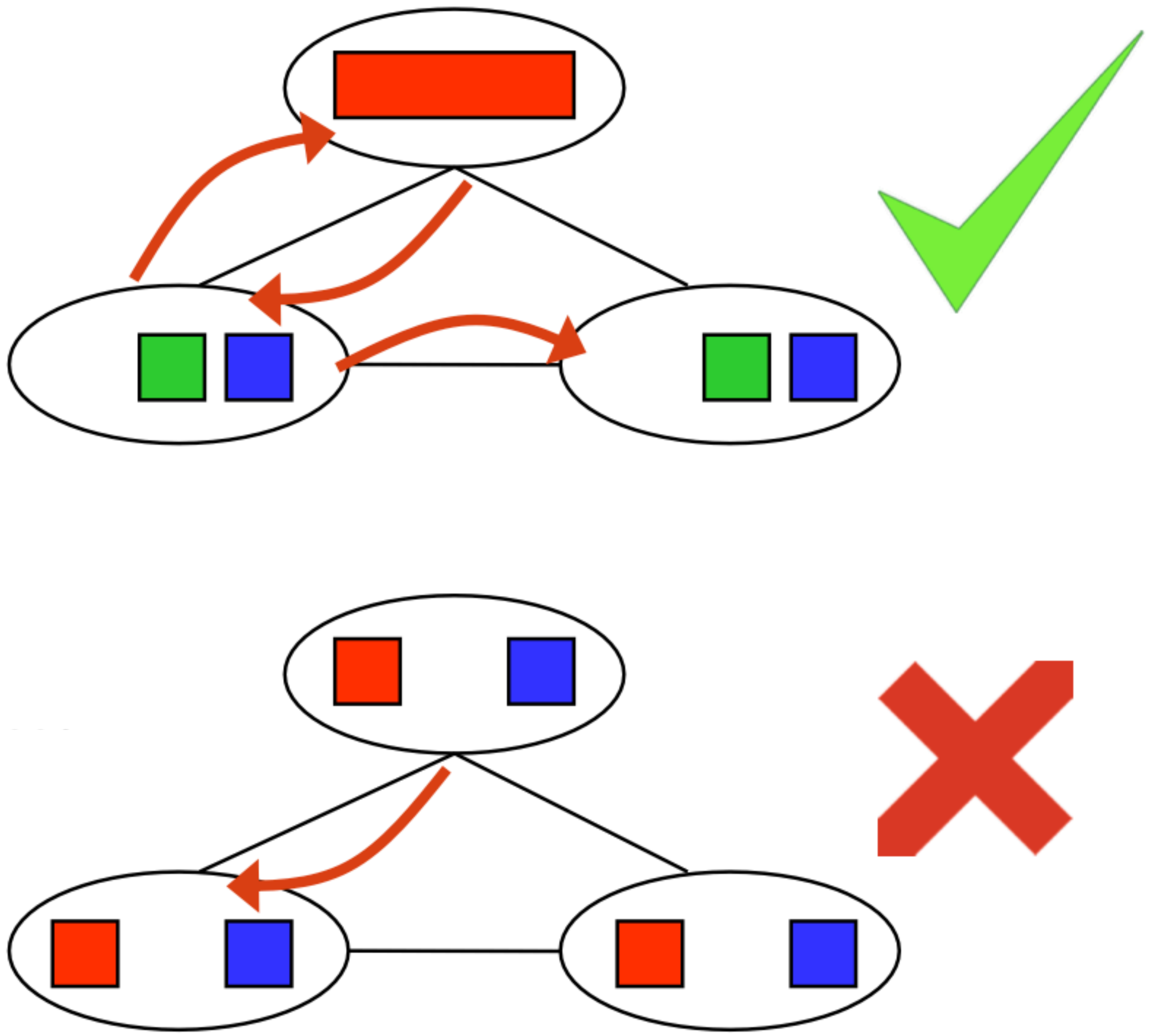
---

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*

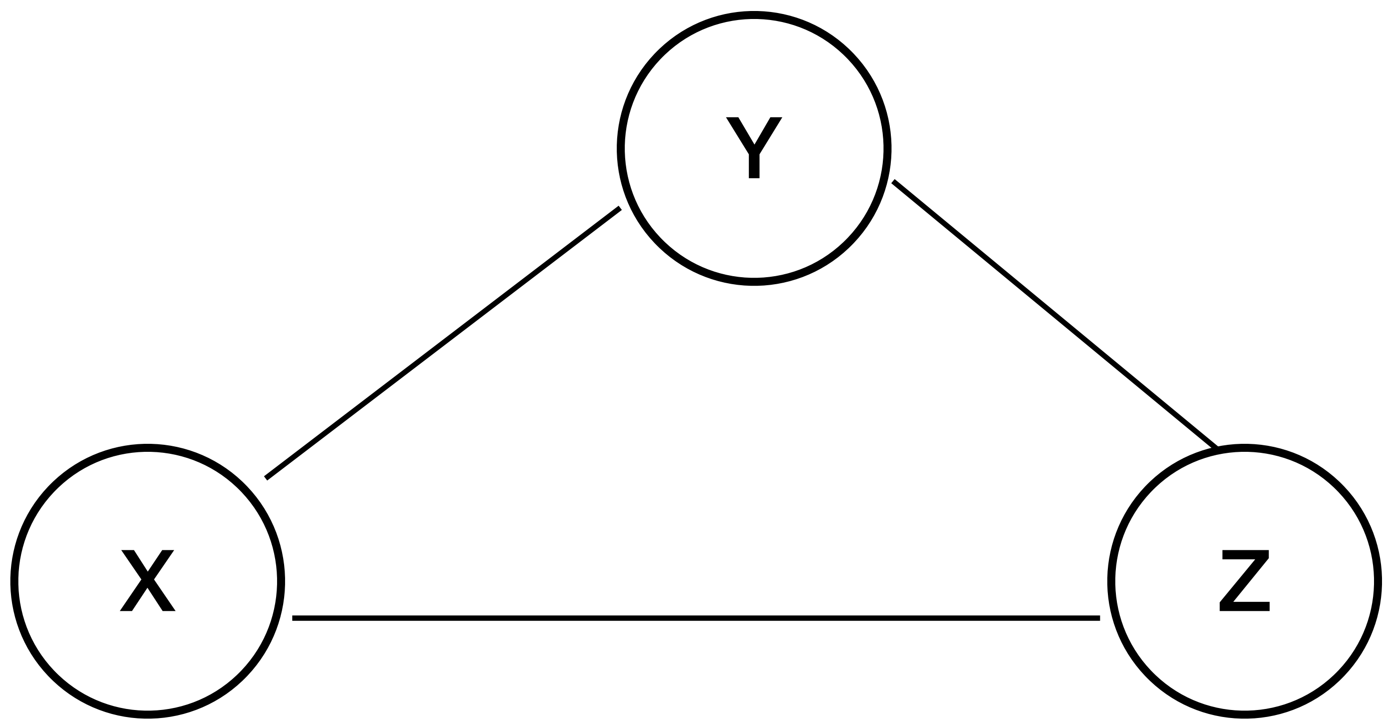
- Runtime: $O(n^2d^3)$

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!



K-consistency



X != Y
Y != Z
X != Z

{1,2}

- Arc consistency does not detect all inconsistencies:
  - Think of simplest example possible….
- Stronger forms of propagation can be defined using the notion of k-consistency.
- A CSP is k-consistent if for any set of k-1 variables and for any consistent assignment to those variables, a consistent value can always be assigned to any kth variable.
  - E.g. 1-consistency or node-consistency
  - E.g. 2-consistency or arc-consistency
  - E.g. 3-consistency or path-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)

Summary and Next Time

- Constraint Satisfaction Problems
  - Algorithms
    - Backtracking
    - Arc Consistency
    - Path Consistency

- Next week (Prof. Gilpin is back)
  - Continue CSPs
    - Heuristics
  - Naives Bayes