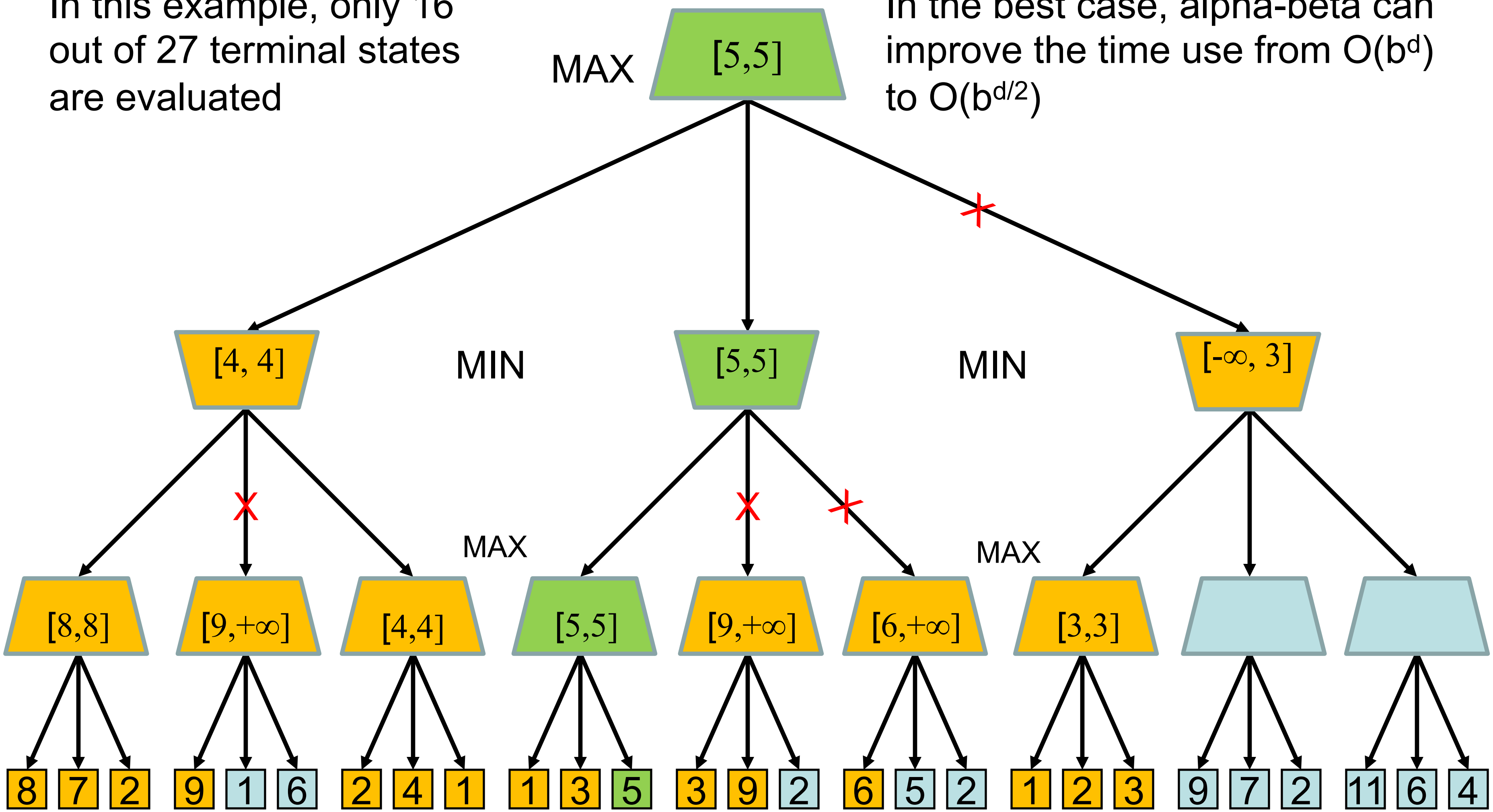
Constraint Satisfaction Problems

Alpha-Beta Pruning Example

In this example, only 16 out of 27 terminal states are evaluated

In the best case, alpha-beta can improve the time use from $O(b^d)$ to $O(b^{d/2})$



Sudoku

4						8		5
	3							
			7					
	2						6	
				8		4		
	4			1				
			6		3		7	
5		3	2		1			
1		4						

Example puzzle with a unique solution

4	1	7	3	6	9	8	2	5
6	3	2	1	5	8	9	4	7
9	5	8	7	2	4	3	1	6
8	2	5	4	3	7	1	6	9
7	9	1	5	8	6	4	3	2
3	4	6	9	1	2	7	5	8
2	8	9	6	4	3	5	7	1
5	7	3	2	9	1	6	8	4
1	6	4	8	7	5	2	9	3

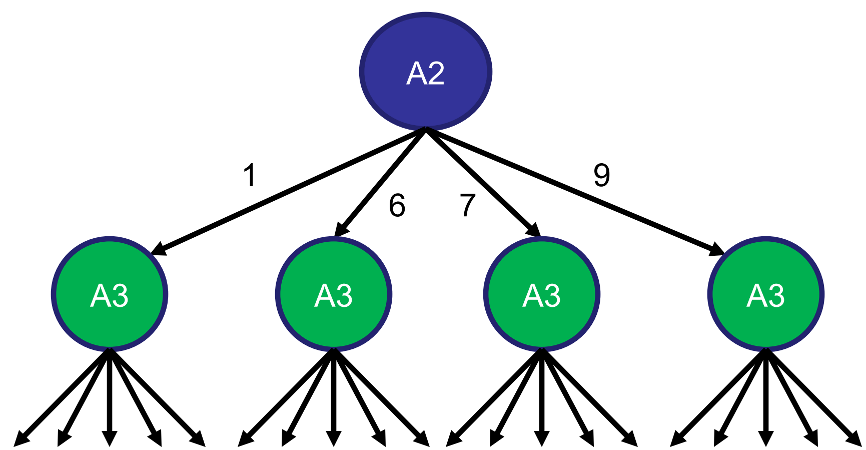
No duplicates in row, column, or 3x3 box

Solving Sudoku via Search

4	A2	A3				8		5
	3							
			7					
	2						6	
				8		4		
	4			1				
			6		3		7	
5		3	2		1			
1		4						

- 61 depth, max 8 branching factor
- 4.6 x 10^{38} possibilities
- Even on 1 million 10GHz, 1024 core machines, this is 1300 billion years!

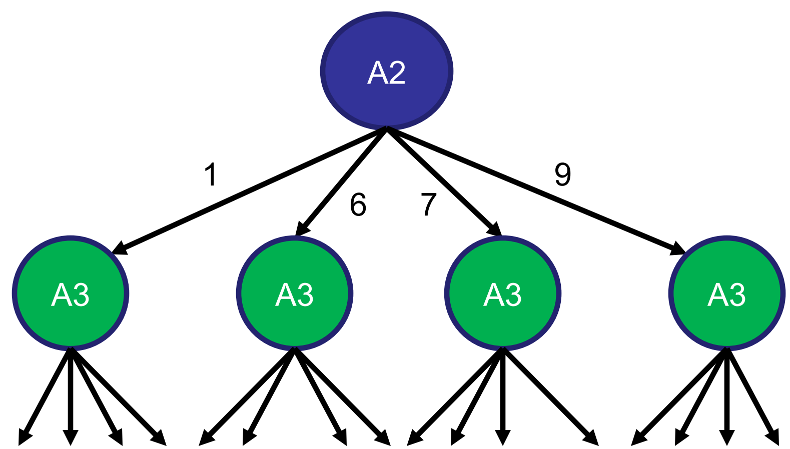
- 20 squares fixed and 61 need to be solved
- Find possible entries
  - A2: 1 ~~2~~ ~~3~~ ~~4~~ ~~5~~ 6 7 ~~8~~ 9
  - A3: 1 2 ~~3~~ ~~4~~ ~~5~~ 6 7 ~~8~~ 9
- Build a tree:



A Smarter Way

4	A2	A3				8		5
	3							
			7					
	2						6	
				8		4		
	4			1				
			6		3		7	
5		3	2		1			
1		4						

- Find possible entries
  - A2: 1 ~~2~~ ~~3~~ ~~4~~ ~~5~~ 6 7 ~~8~~ 9
  - A3: 1 2 ~~3~~ ~~4~~ ~~5~~ 6 7 ~~8~~ 9
- Once we choose A2, that further limits our choices



Constraint Satisfaction Problems

- In a typical search problem
  - state is a "black box" – any data structure that supports successor function, heuristic function, and goal test
- In a constraint satisfaction problem (CSP):
  - state is an assignment of values from a domain D_i to a set of variables X_i
  - goal test is a set of constraints specifying allowable combinations of values for subsets of variables
- A solution to a CSP is one that is complete (all variables are assigned) and consistent (no constraints are violated)
- Simple example of a formal representation language

Sudoku as a CSP

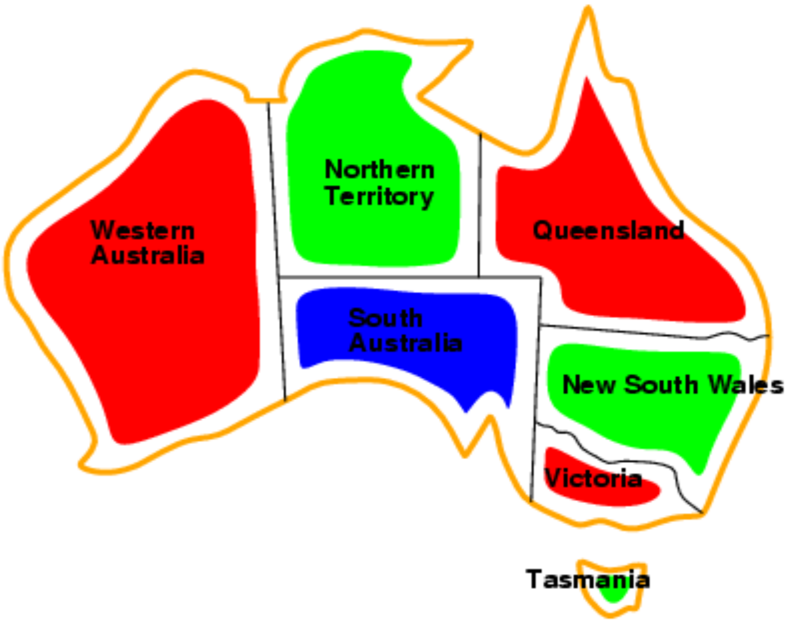
4						8		5
	3							
			7					
	2						6	
				8		4		
	4			1				
			6		3		7	
5		3	2		1			
1		4						

- Domain = {1, 2, 3, 4, 5, 6, 7, 8, 9}
- Variables = { A1, A2, ... A9,
  B1, B2, … B9,
  …
  I1, I2, … I9 }
- Constraints from row, column, and 3x3 cell restrictions
- Constraints =
  {A1≠A2, A1≠A3, A1≠A4, …
  A1≠B1, A1≠C1, A1≠D1, …
  A1≠B2, A1≠B3, A1≠C1, …}

Simpler Example: Map Coloring

- Variables V_i = {*WA, NT, Q, NSW, V, SA, T* }
- Domain 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)}

Simpler Example: Map Coloring



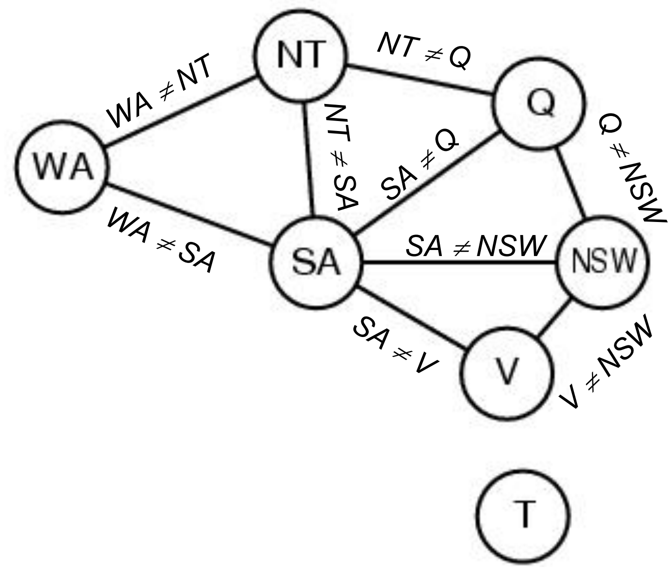
- Solutions are complete and consistent assignments
- One solution is shown above

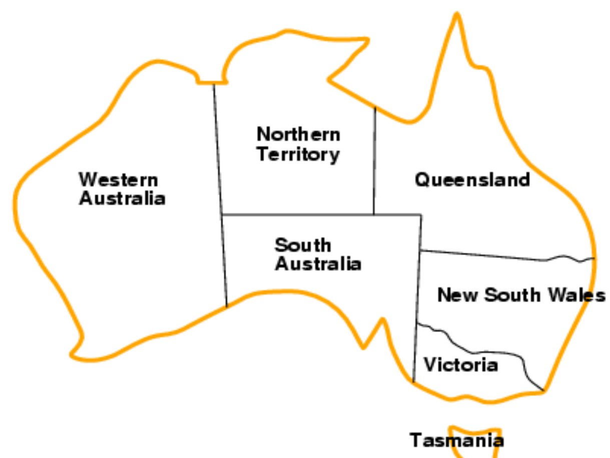
  WA = red, NT = green, Q = red, NSW = green,

  V = red, SA = blue, T = green

Constraint Graph

- Constraint graph:
  - nodes are variables
  - arcs are constraints
- CSP benefits
  - Standard representation pattern: variables with values
  - Generic goal, successor functions
  - Generic heuristics (no domain specific expertise)
  - Graph can simplify search.
    - e.g. Tasmania is an independent subproblem.



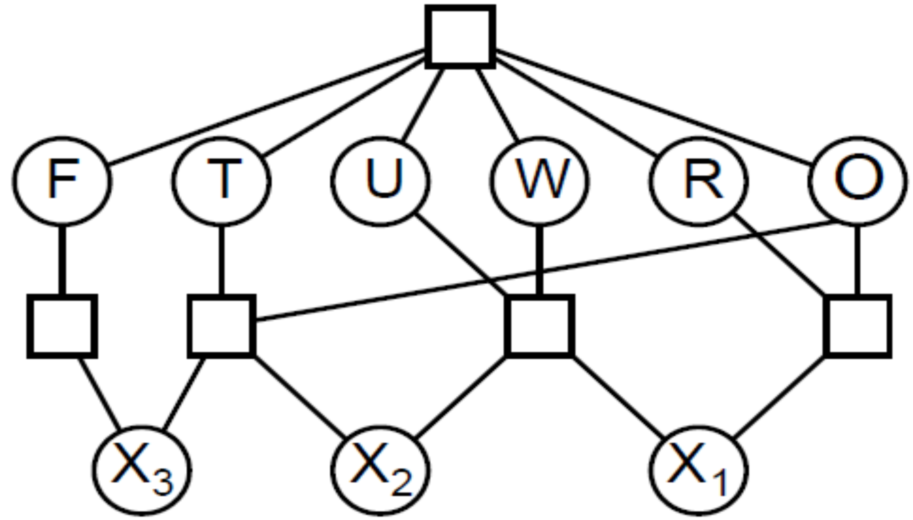


Another Example: Cryptarithmetic

```
  T W O
+ T W O
-------
F O U R
```

Another Example: Cryptarithmetic

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



Variables: $F, O, U, R, T, W, X_1, X_2, X_3$

Domain: $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

Constraints: $Alldiff\ (F, O, U, R, T, W)$

$$O + O = R + 10 \cdot X_1$$

$$X_1 + W + W = U + 10 \cdot X_2$$

$$X_2 + T + T = O + 10 \cdot X_3$$

$$X_3 = F,\ T \neq 0,\ F \neq 0$$

Varieties of CSPs

- Discrete variables
  - finite domains:
    - n variables, domain size $d \rightarrow O(d^n)$ 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

Varieties 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
- Circuit layout

- Notice that many real-world problems involve real-valued variables

Solving CSPs

- Let's start with a straightforward approach, then fix it.
- Just like we did with Sudoku, let's treat this as a search problem.
  - Initial state: the empty assignment { }
  - Successor function: assign a value to an unassigned variable that does not conflict with current assignment

    → fail if no legal assignments
  - Goal test: the current assignment is complete

Backtracking search

- Variable assignments are commutative, i.e.,

  [WA = red] followed by [NT = green] is the same as
  [NT = green] followed by [WA = red]
- Only need to consider assignments to a single variable at each depth of the tree
- Depth-first search for CSPs with single-variable assignments is called backtracking search
- Backtracking search is the basic uninformed algorithm for CSPs
- Can solve *n*-queens for $n \approx 25$

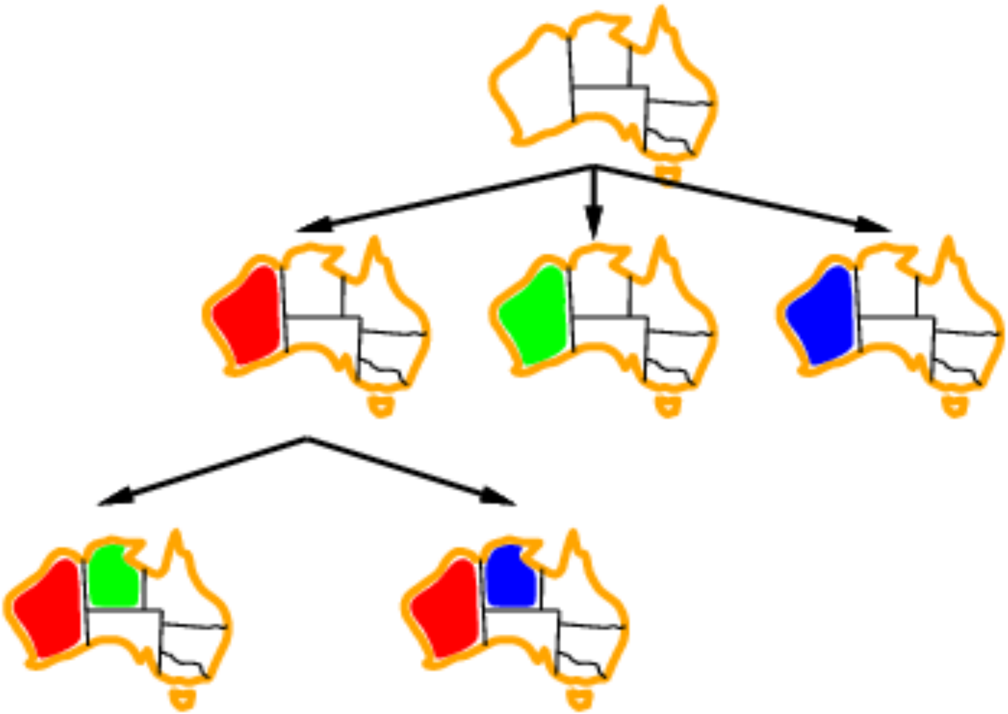
Backtracking search

```
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
```

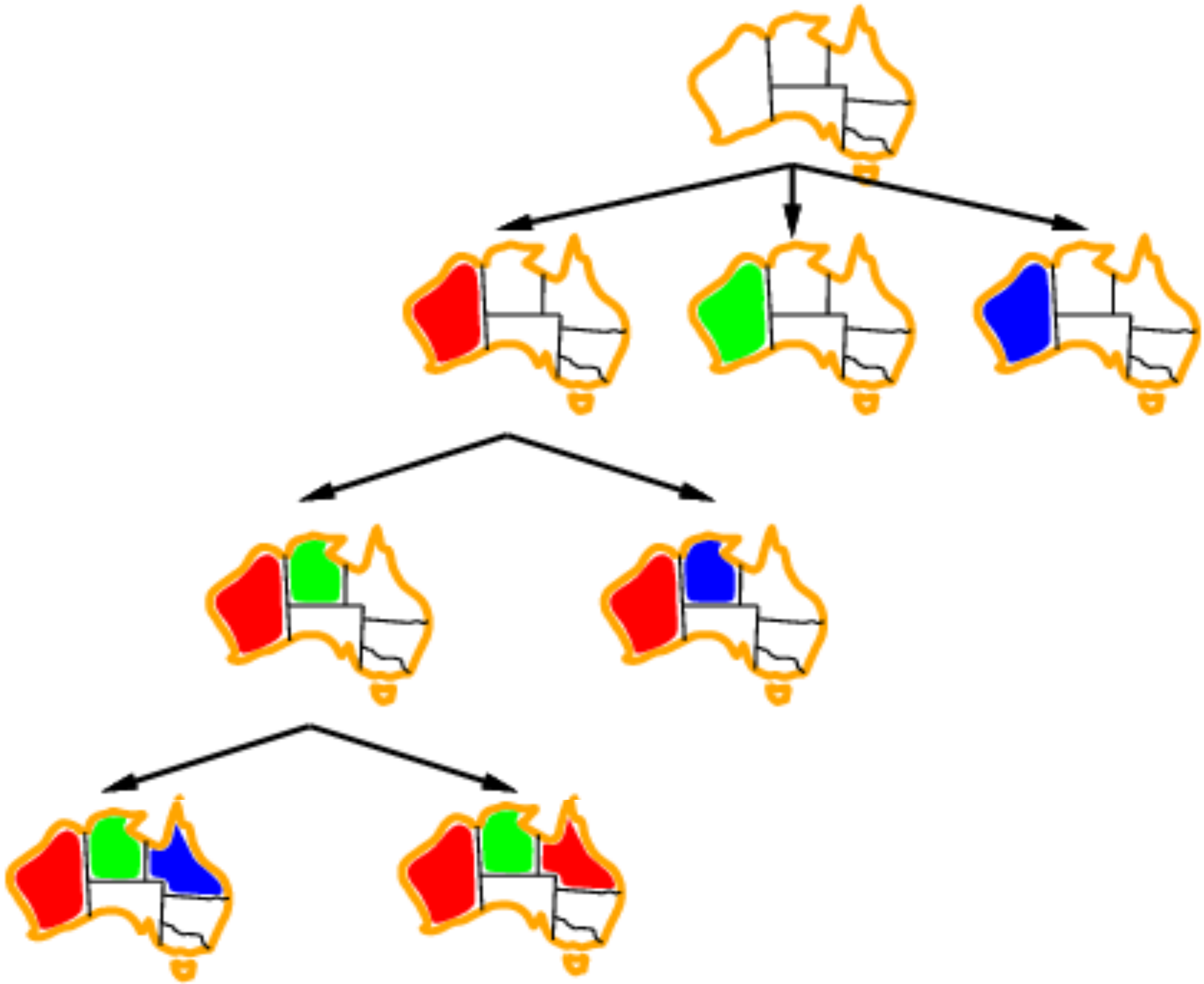
Backtracking example

Backtracking example

Backtracking example



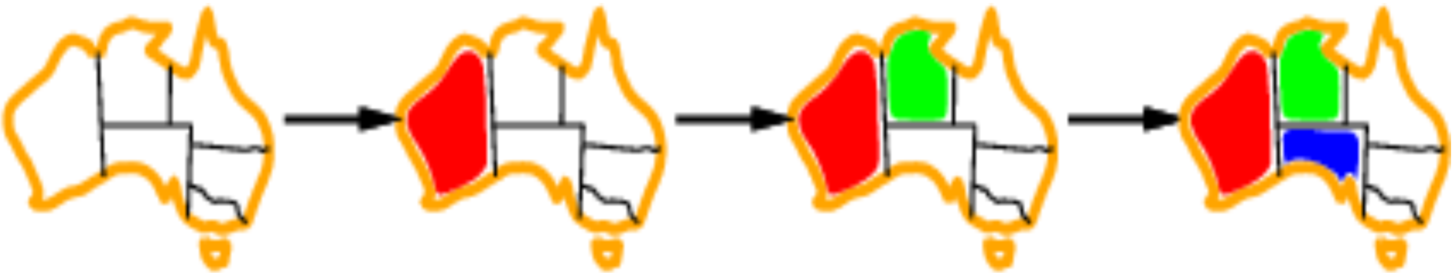
Backtracking example



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?
- Heuristics:
  1. Most constrained variable
  2. Most constraining variable
  3. Least constraining value
  4. Forward checking

H1: Most constrained variable

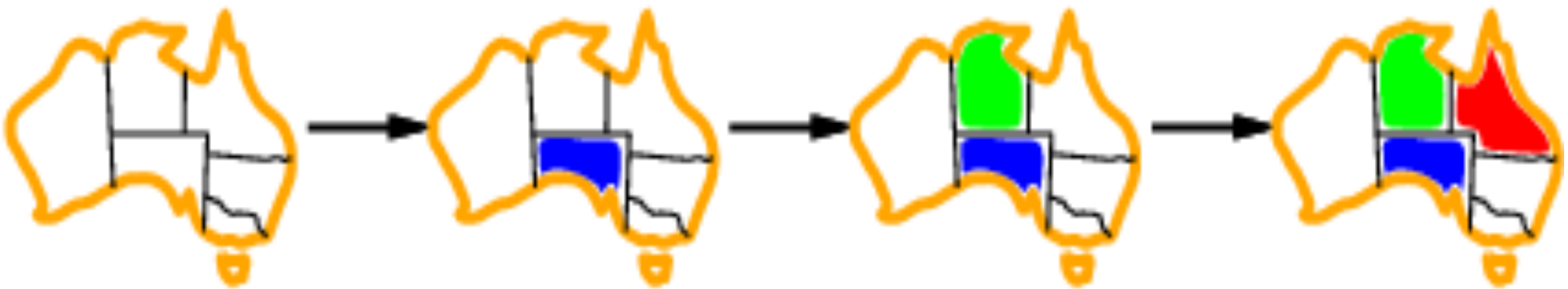


- Most constrained variable:

  choose the variable with the fewest legal values

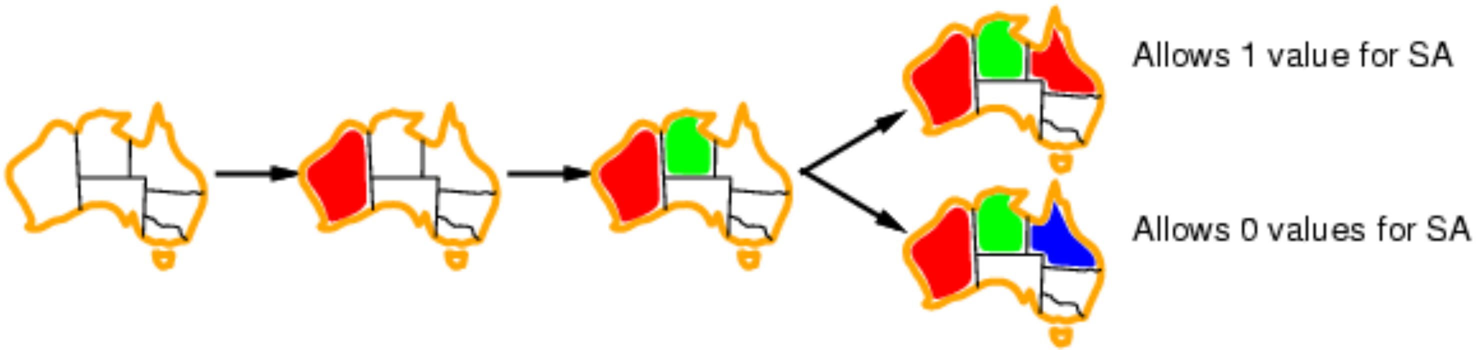
- a.k.a. minimum remaining values (MRV) heuristic

H2: Most constraining variable



- Tie-breaker among most constrained variables
- Most constraining variable:
  - choose the variable with the most constraints on remaining variables

H3: Least constraining value

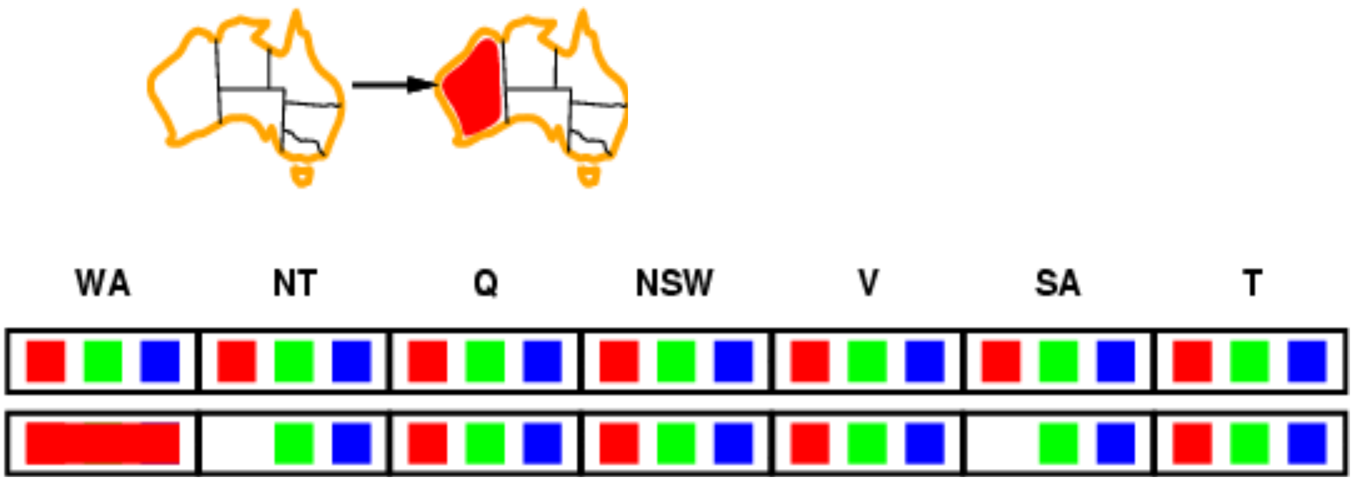


- Given a variable, choose the least constraining value:
  - the one that rules out the fewest values in the remaining variables
- Combining these heuristics makes 1000 queens feasible

H4: Forward checking

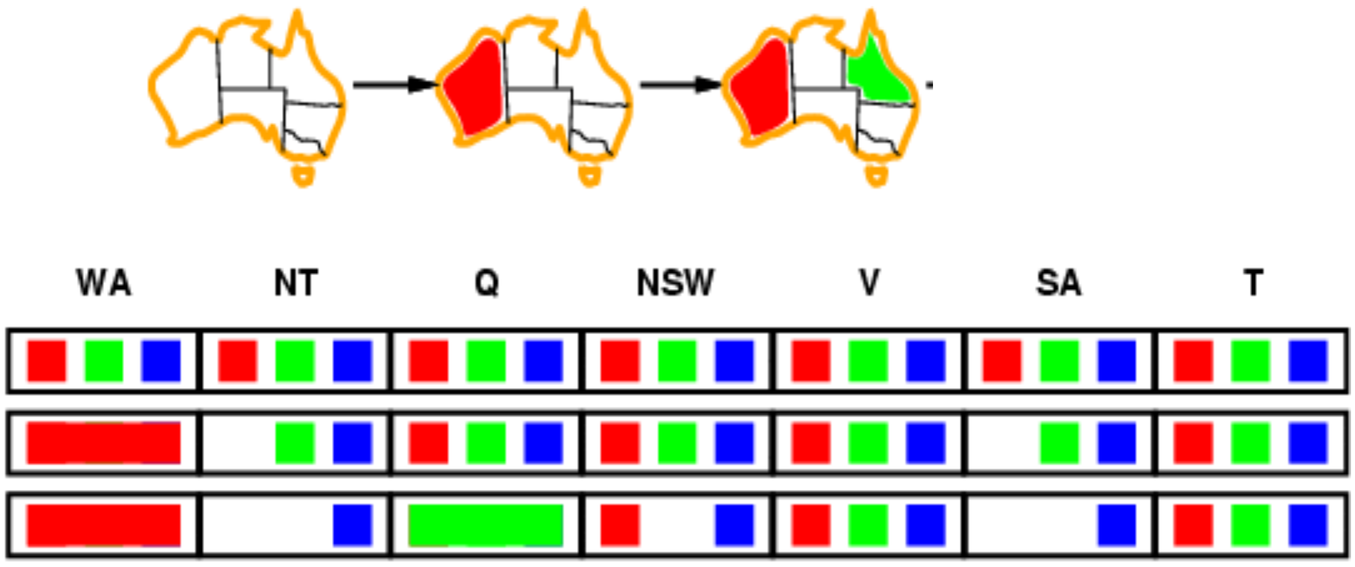
- Idea:
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values

H4: Forward checking



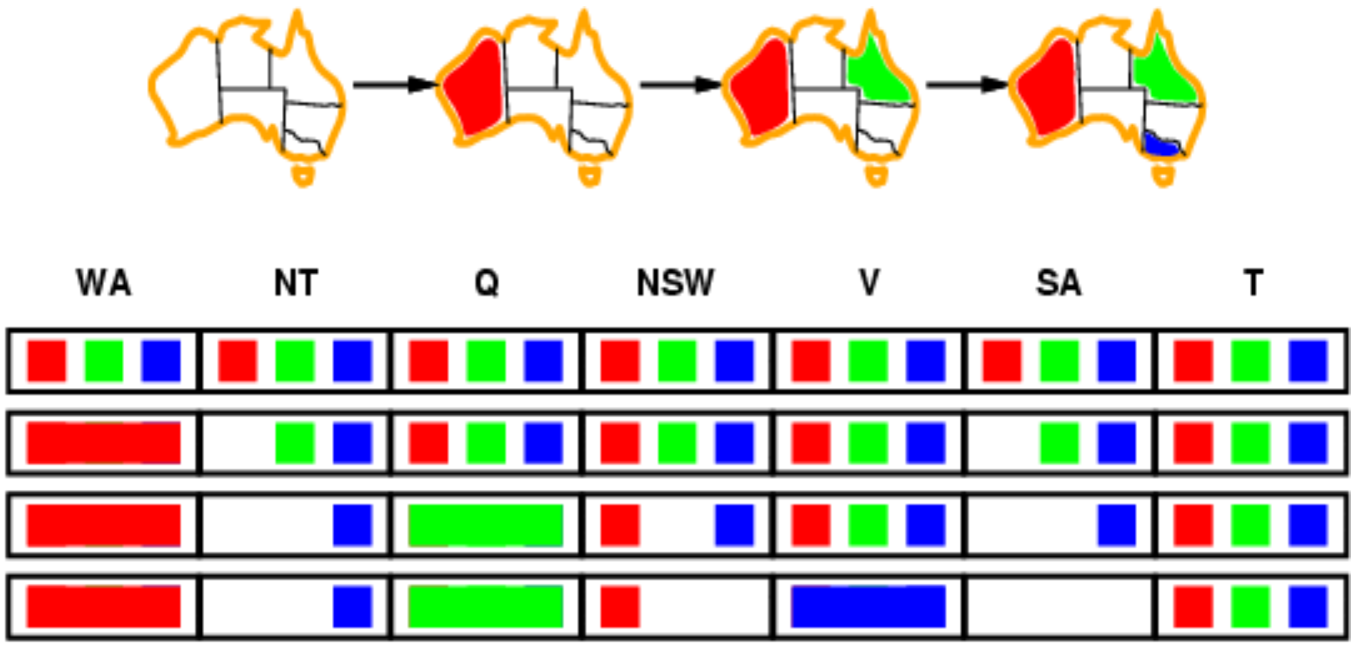
- Idea:
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values

H4: Forward checking



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

H4: Forward checking

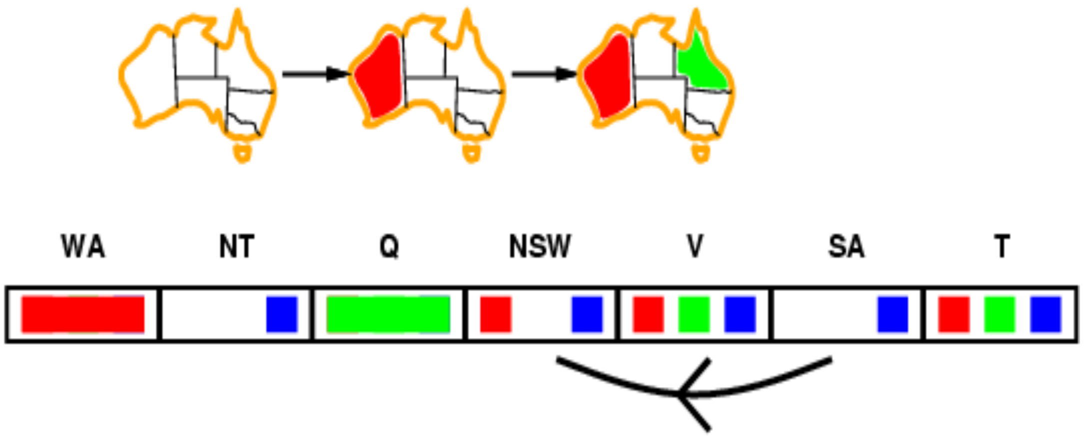


- Idea:
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values

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

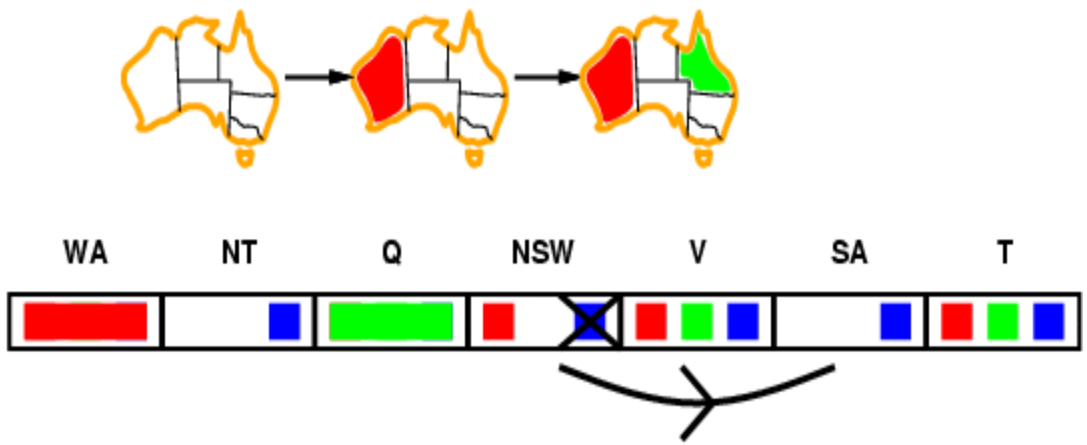
Arc consistency



- Simplest form of propagation makes each arc consistent
- *X* → *Y* is consistent iff

  for every value *x* of *X* there is some allowed *y*

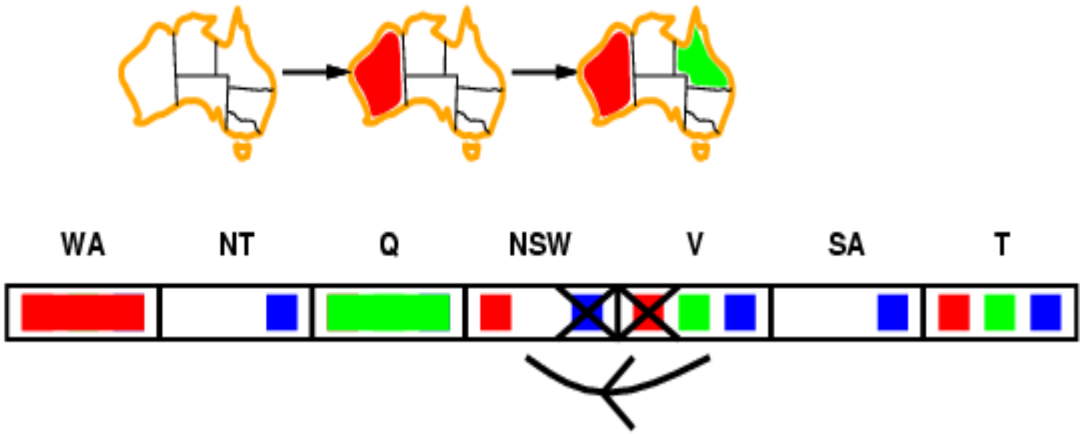
Arc consistency



- Simplest form of propagation makes each arc consistent
- $X \rightarrow Y$ is consistent iff

  for every value x of X there is some allowed y

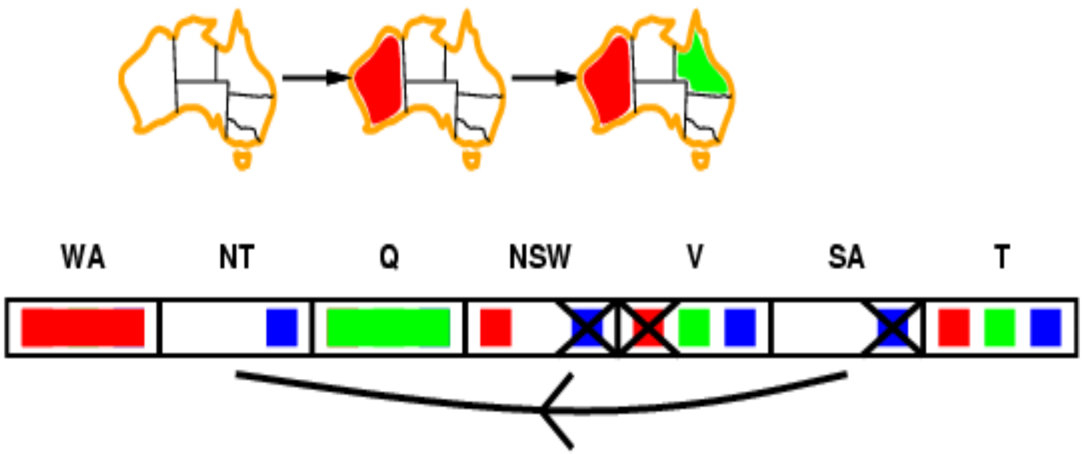
Arc consistency



- Simplest form of propagation makes each arc consistent
- $X \rightarrow Y$ 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

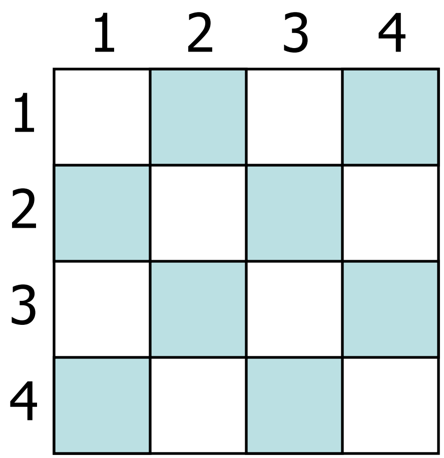
Arc consistency algorithm AC-3

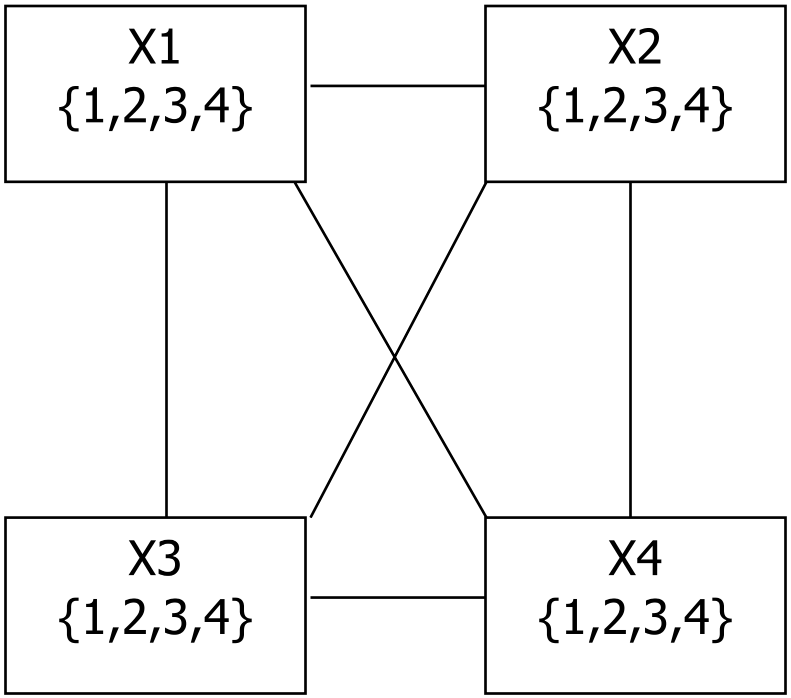
```
function AC-3( csp) returns the CSP, possibly with reduced domains
   inputs: csp, a binary CSP with variables {X1, X2, ..., Xn}
   local variables: queue, a queue of arcs, initially all the arcs in csp

   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
```

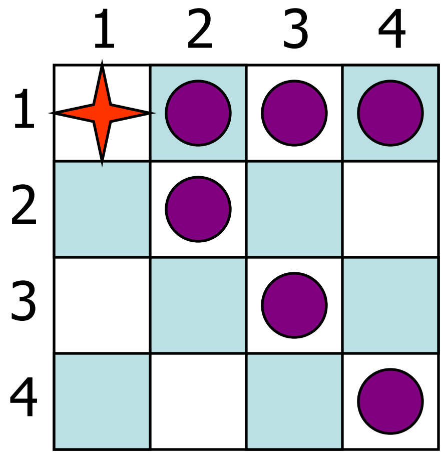
- Time complexity: $O(n^2d^3)$

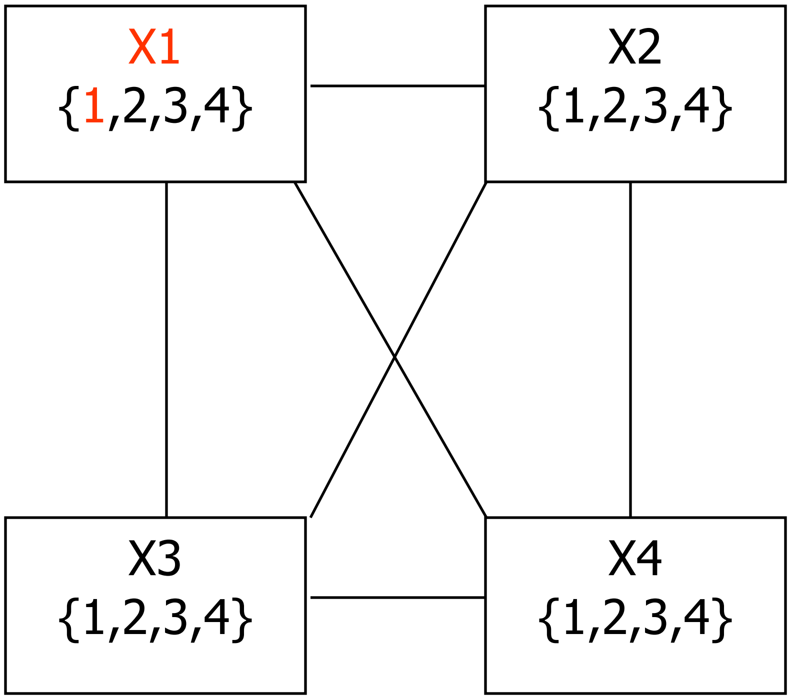
Example: 4-Queens Problem



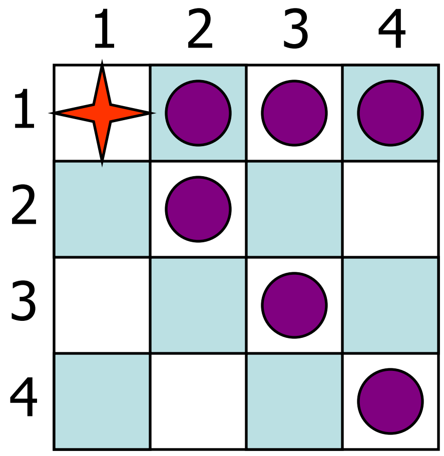


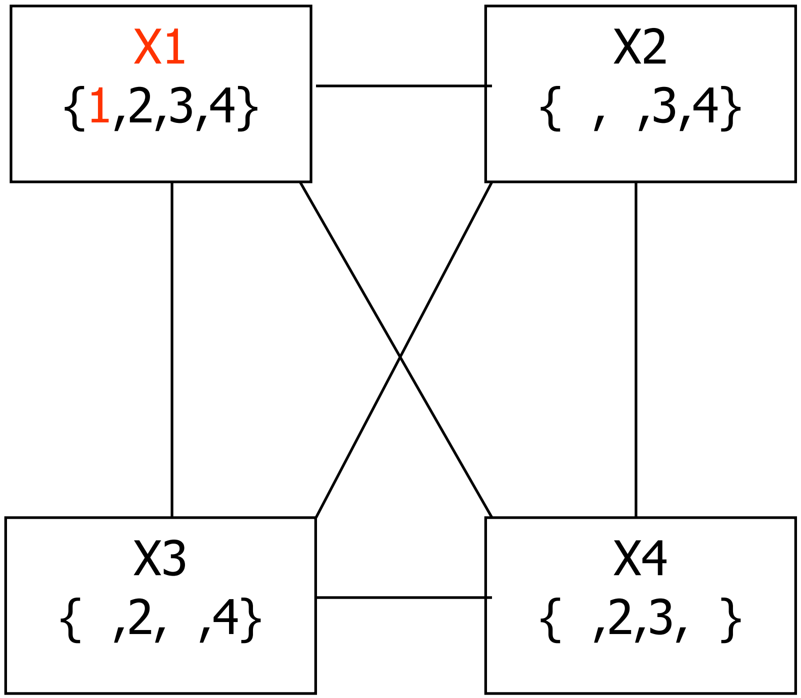
Example: 4-Queens Problem



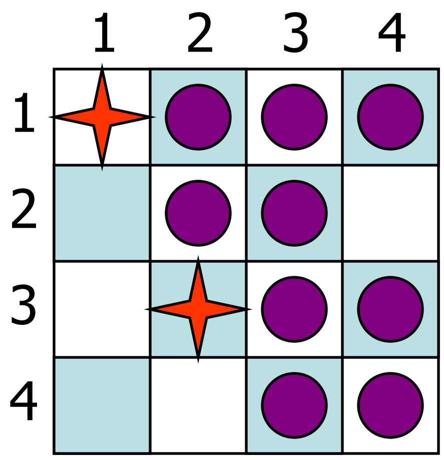


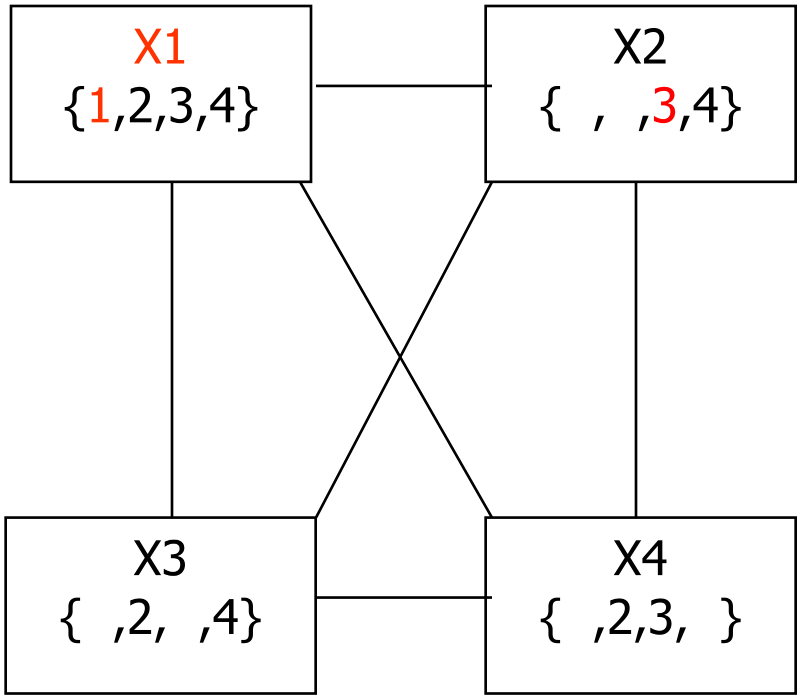
Example: 4-Queens Problem



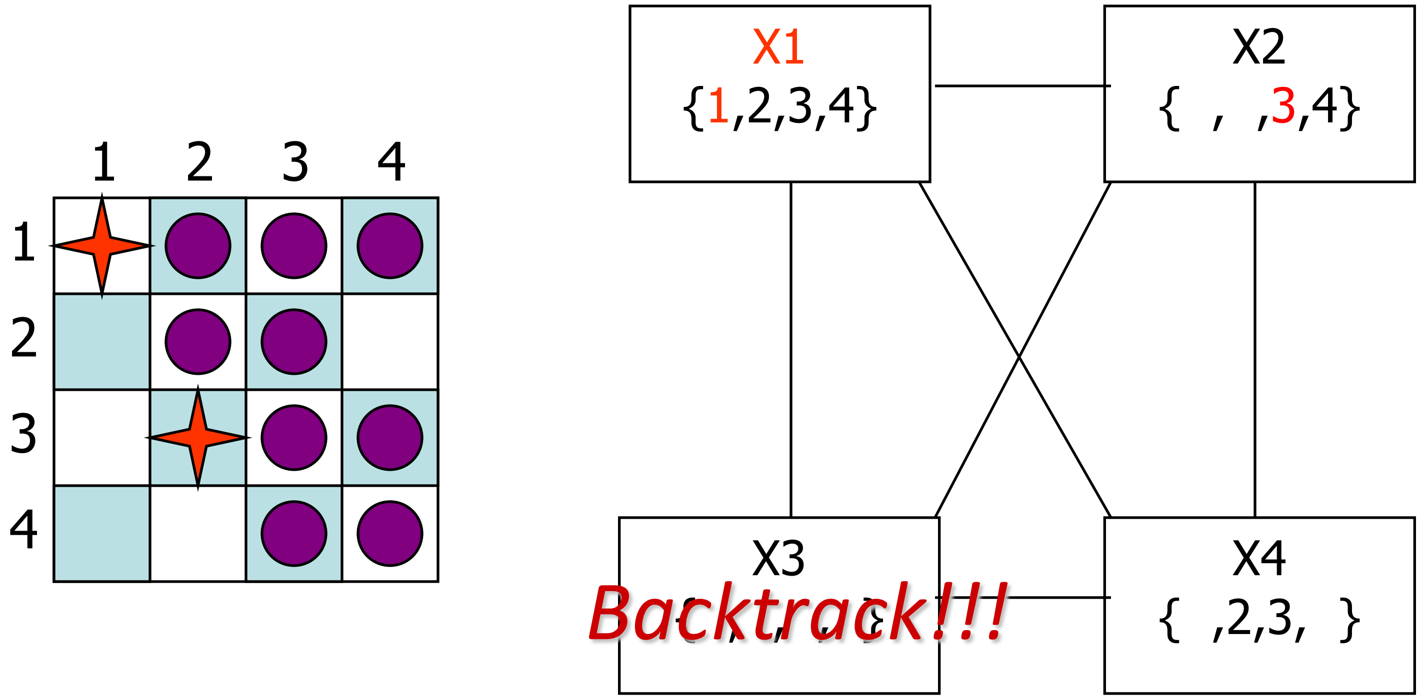


Example: 4-Queens Problem



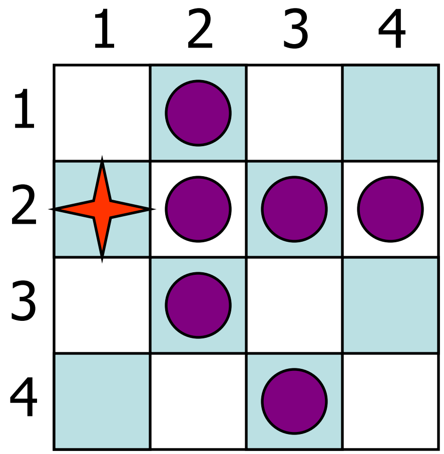


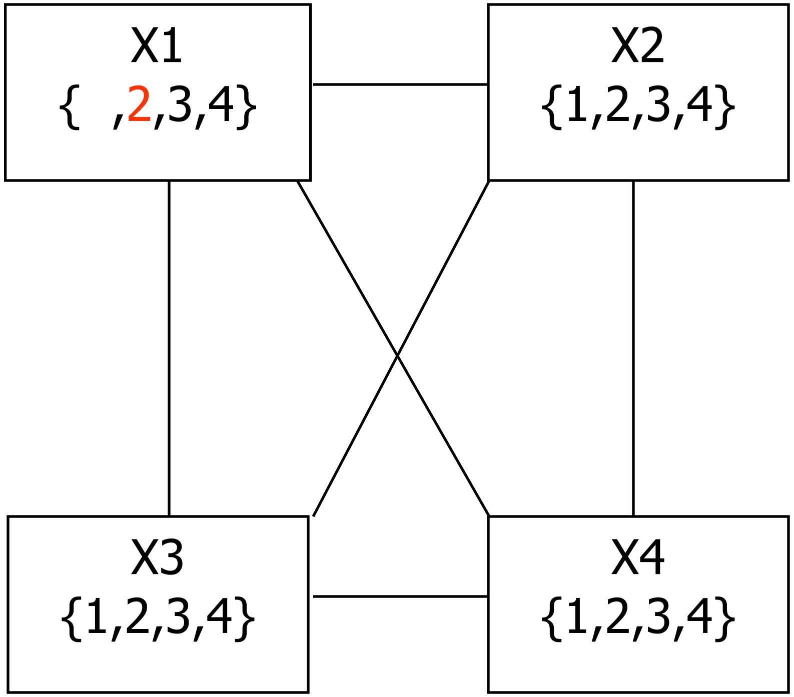
Example: 4-Queens Problem



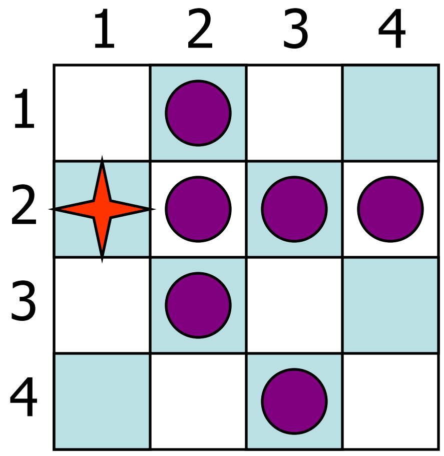
Example: 4-Queens Problem

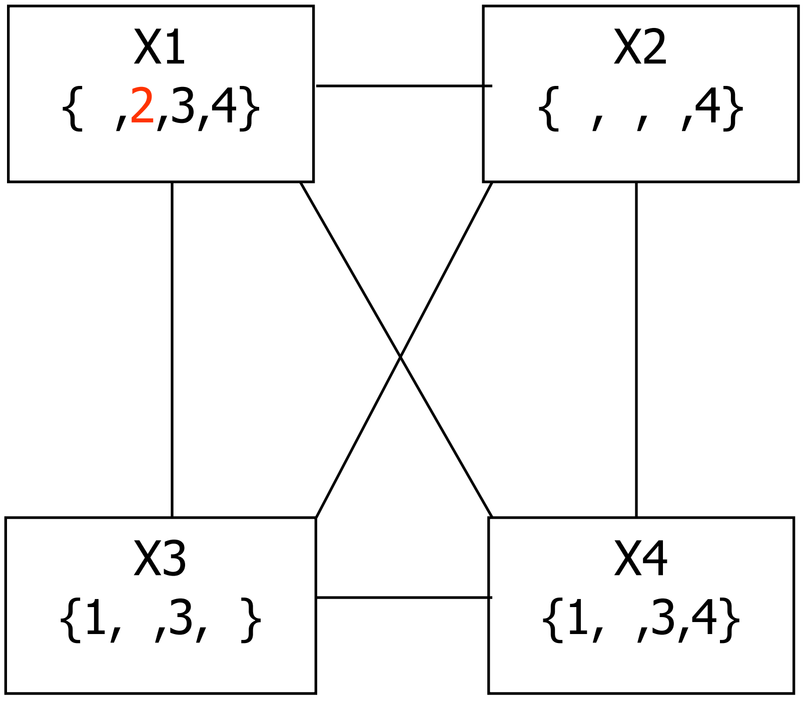
Picking up a little later after two steps of backtracking….



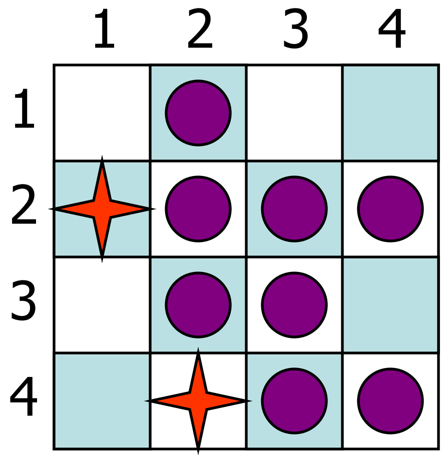


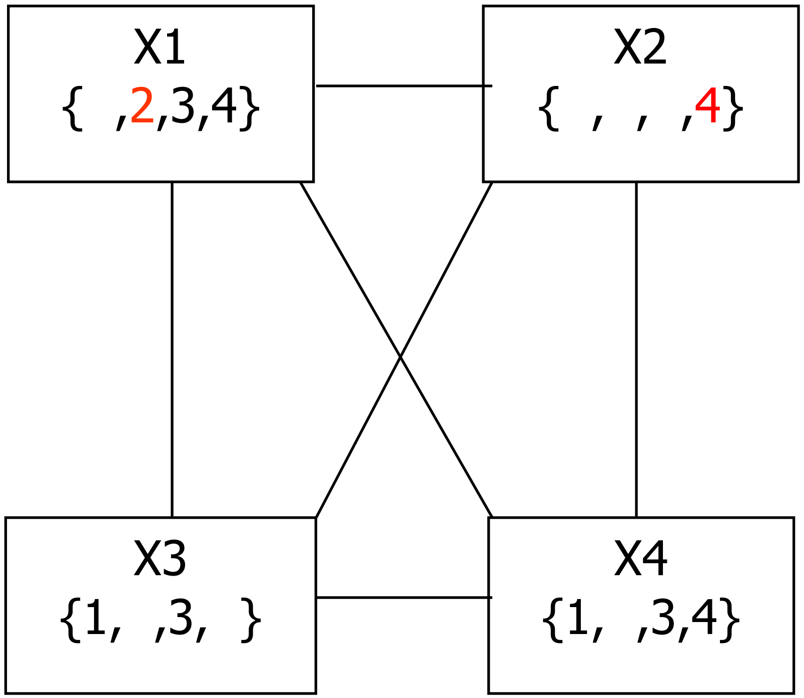
Example: 4-Queens Problem



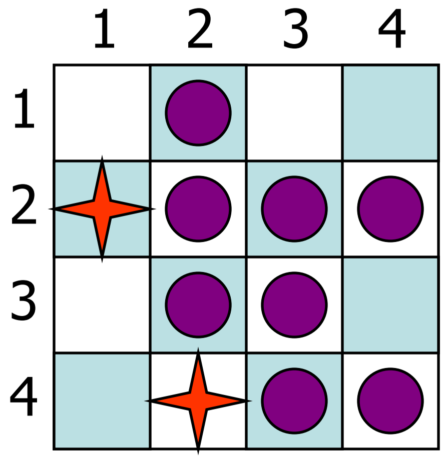


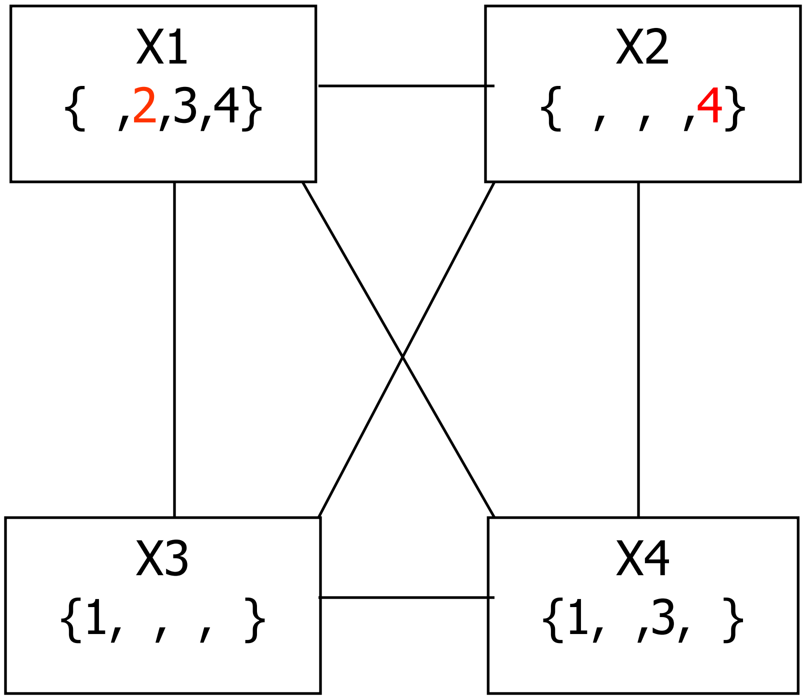
Example: 4-Queens Problem



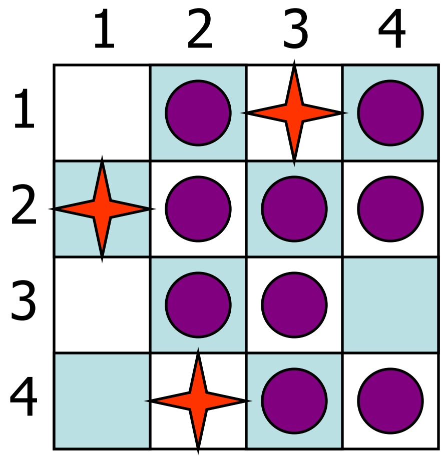


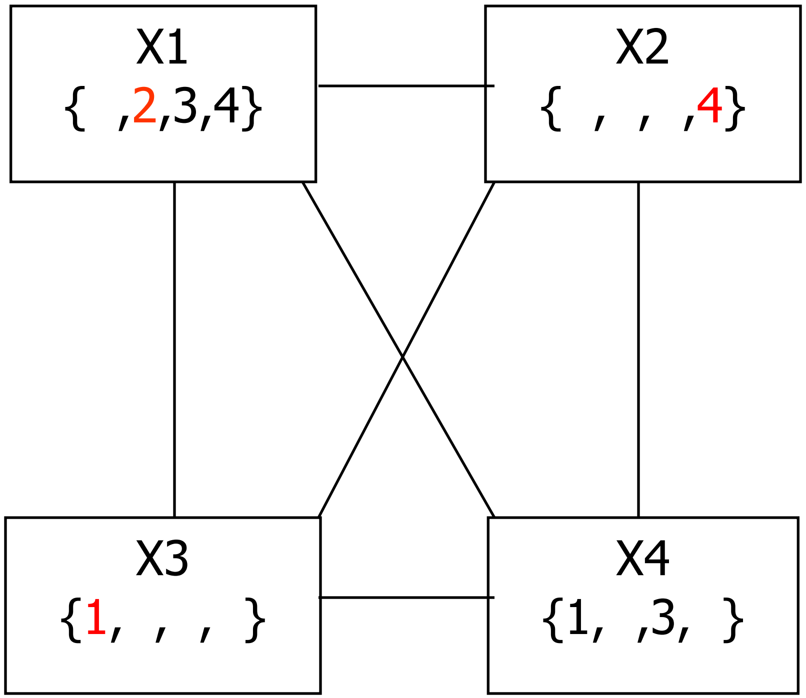
Example: 4-Queens Problem



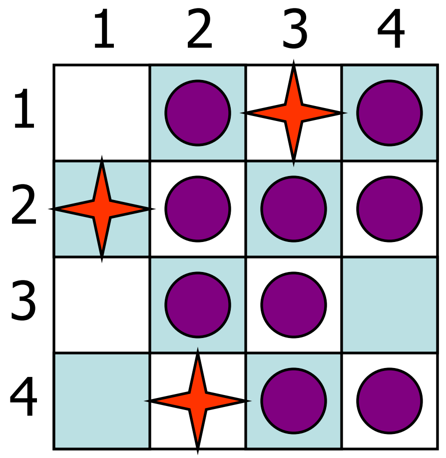


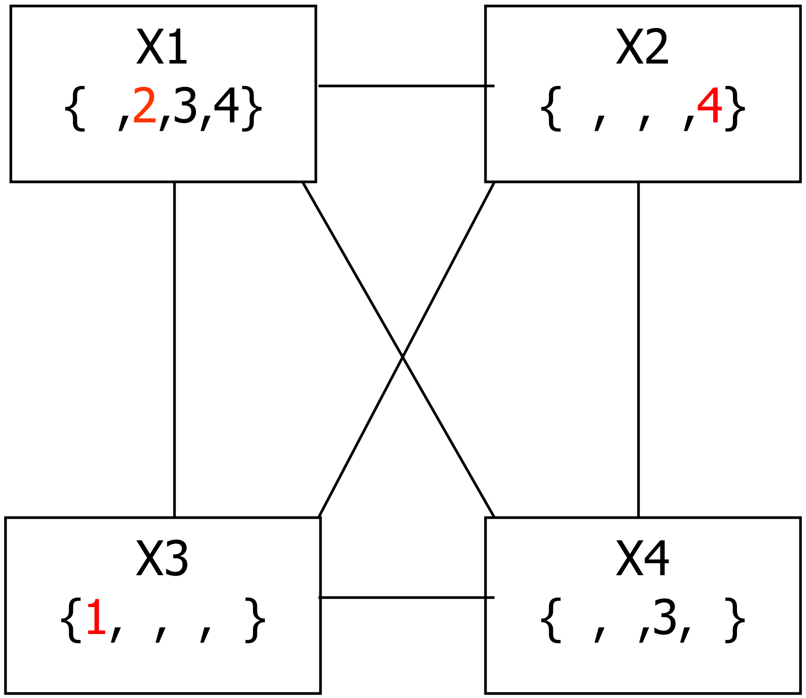
Example: 4-Queens Problem



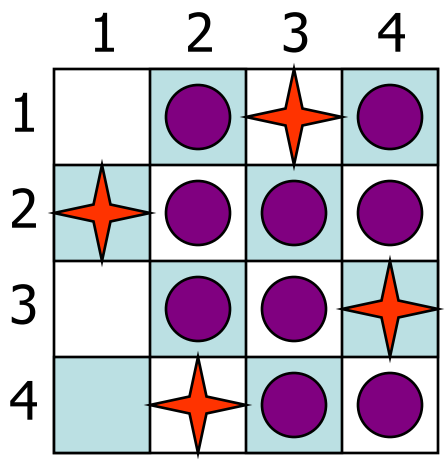


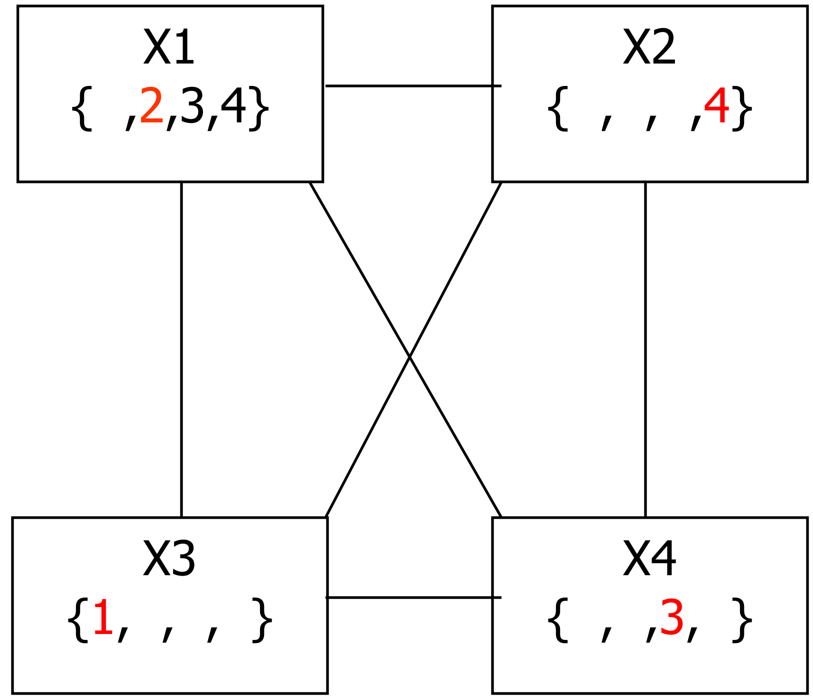
Example: 4-Queens Problem





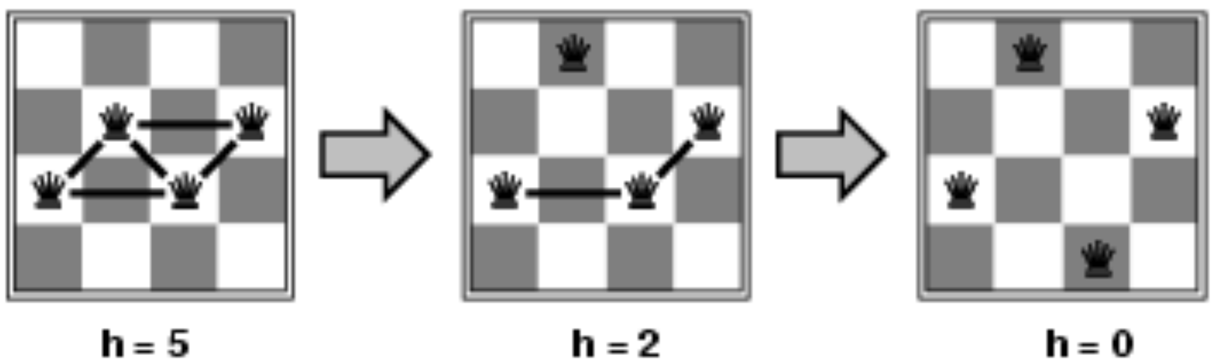
Example: 4-Queens Problem





Example: n-queens

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



- Given random initial state, AC-3 can solve *n*-queens in almost constant time for arbitrary *n* with high probability (e.g., *n* = 10,000,000)

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
- 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

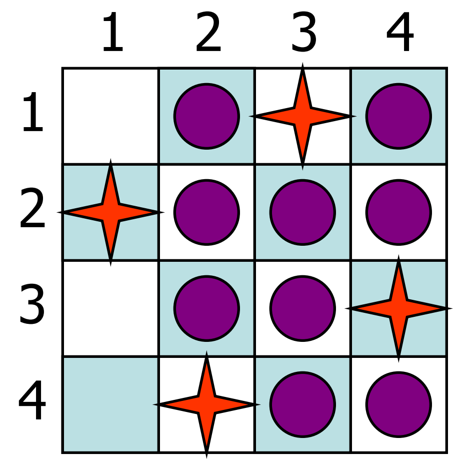
Propositional Logic

CPSC 470 – Artificial Intelligence

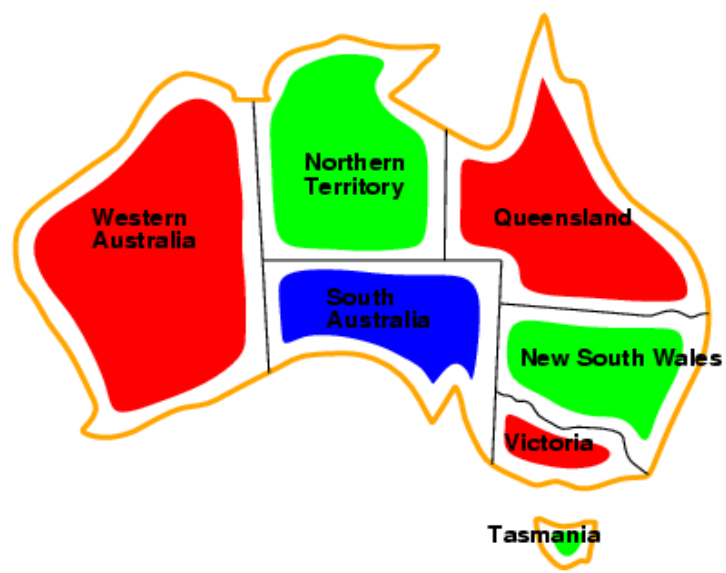
Brian Scassellati

Constraint Satisfaction Problems

```
  T W O
+ T W O
-------
F O U R
```



4						8		5
	3							
			7					
	2						6	
				8		4		
	4			1				
			6		3		7	
5		3	2		1			
1		4						



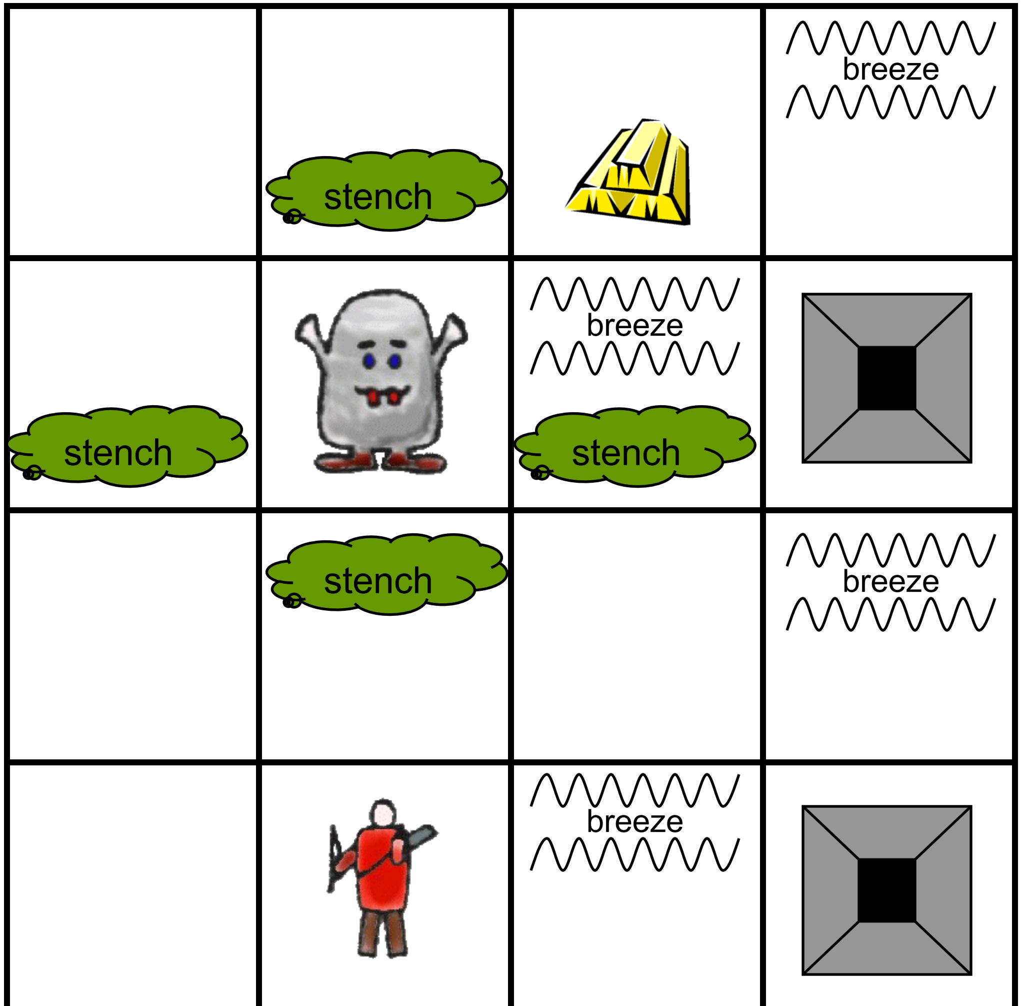
World Characterization

	Search	**CSP**
Fully Observable	Yes	Yes
Deterministic	Yes	Yes
Episodic	No	No
Static	Yes	Yes
Discrete	Yes	Mostly

World Characterization

	Search	**CSP**	**Today**
Fully Observable	Yes	Yes	No
Deterministic	Yes	Yes	Yes
Episodic	No	No	No
Static	Yes	Yes	Yes
Discrete	Yes	Mostly	Yes

The Wumpus World

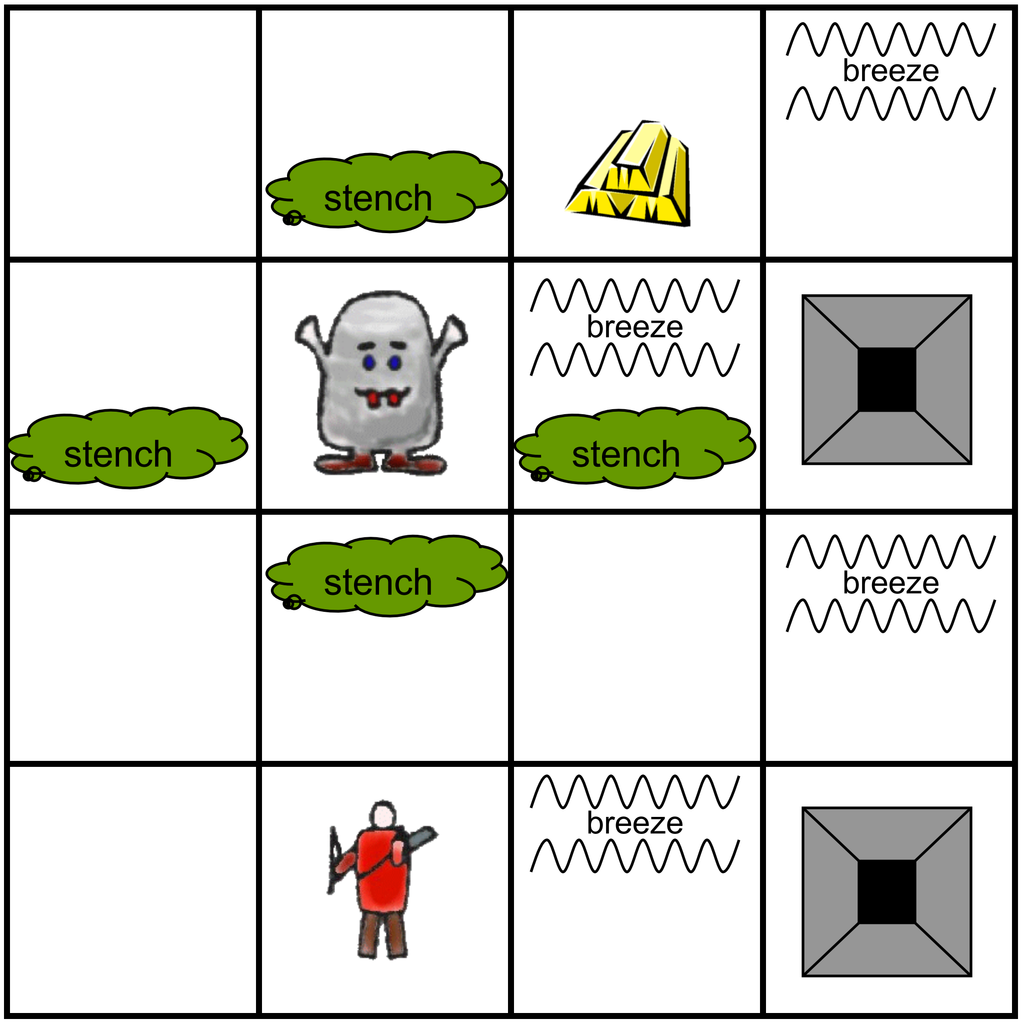


- Grid-like world

- Noble hero
- Horrible wumpus
- Bottomless pits
- Gold

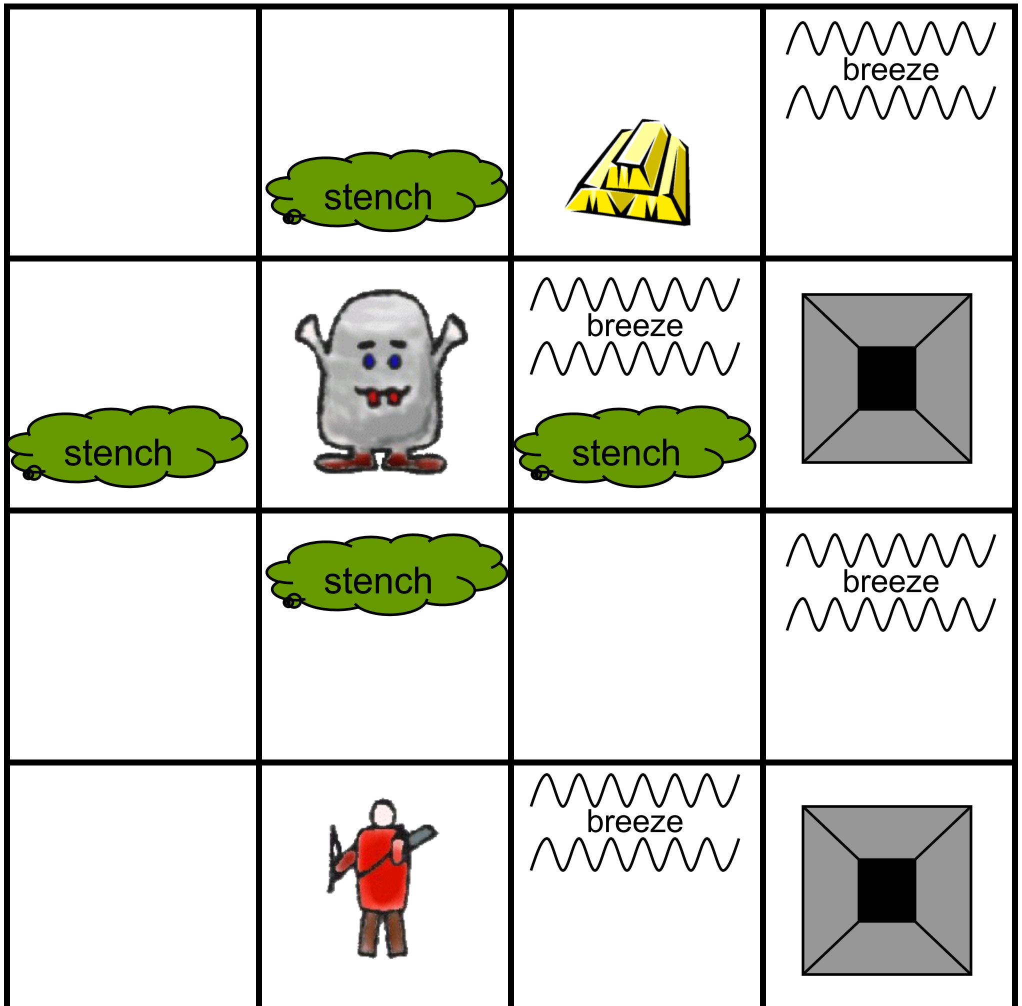
- Breeze
- Stench

Actions in the Wumpus World



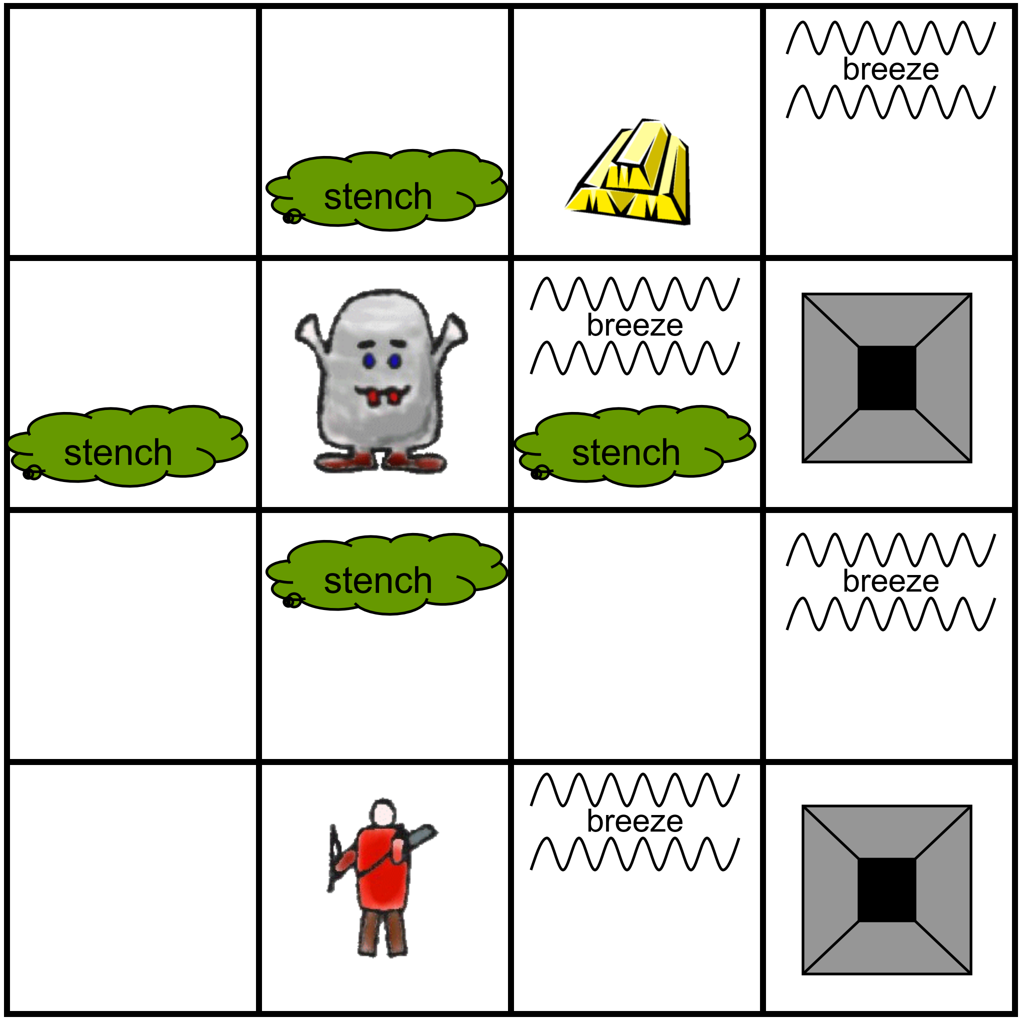
- Goals:
  - find the gold
  - kill the wumpus
  - go home
- Actions
  - Move N,S,E,W
  - Grab
  - Shoot(N,S,E,W)
    - Only one arrow!

The Wumpus World



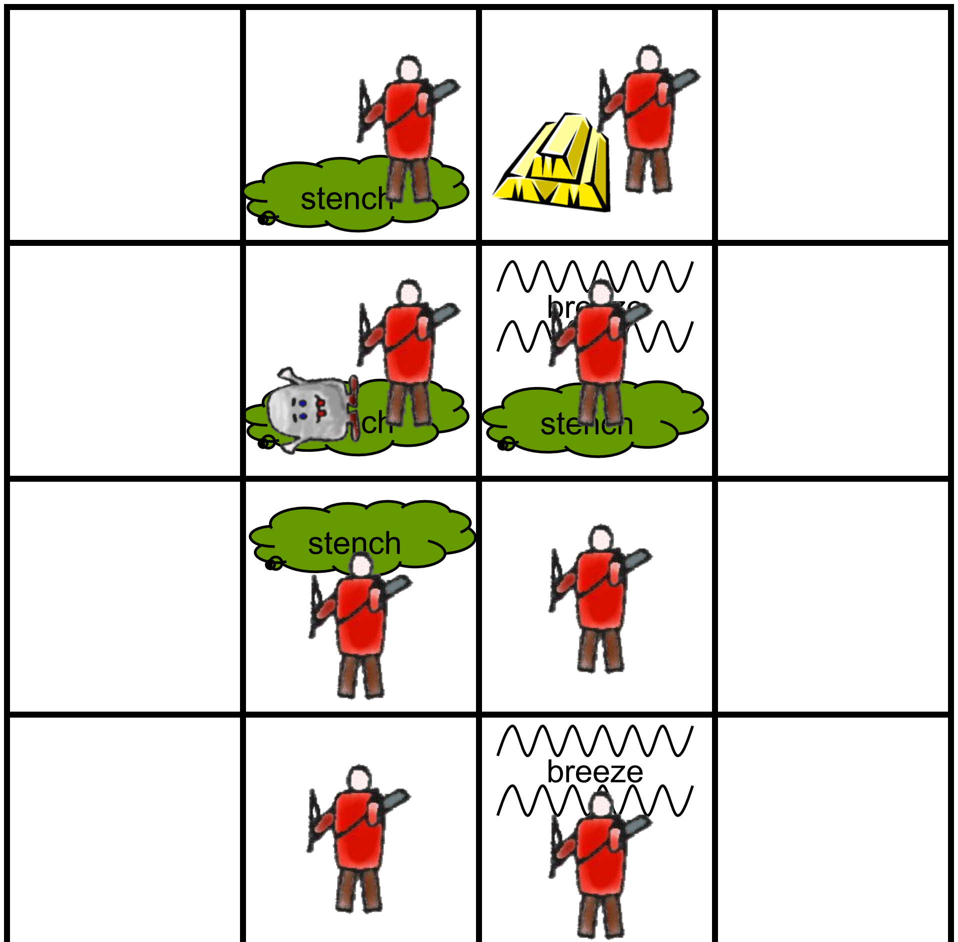
- If we had complete knowledge of the world, then we could simply build a search tree
- What if our perceptions are limited?

Incomplete Knowledge of the World



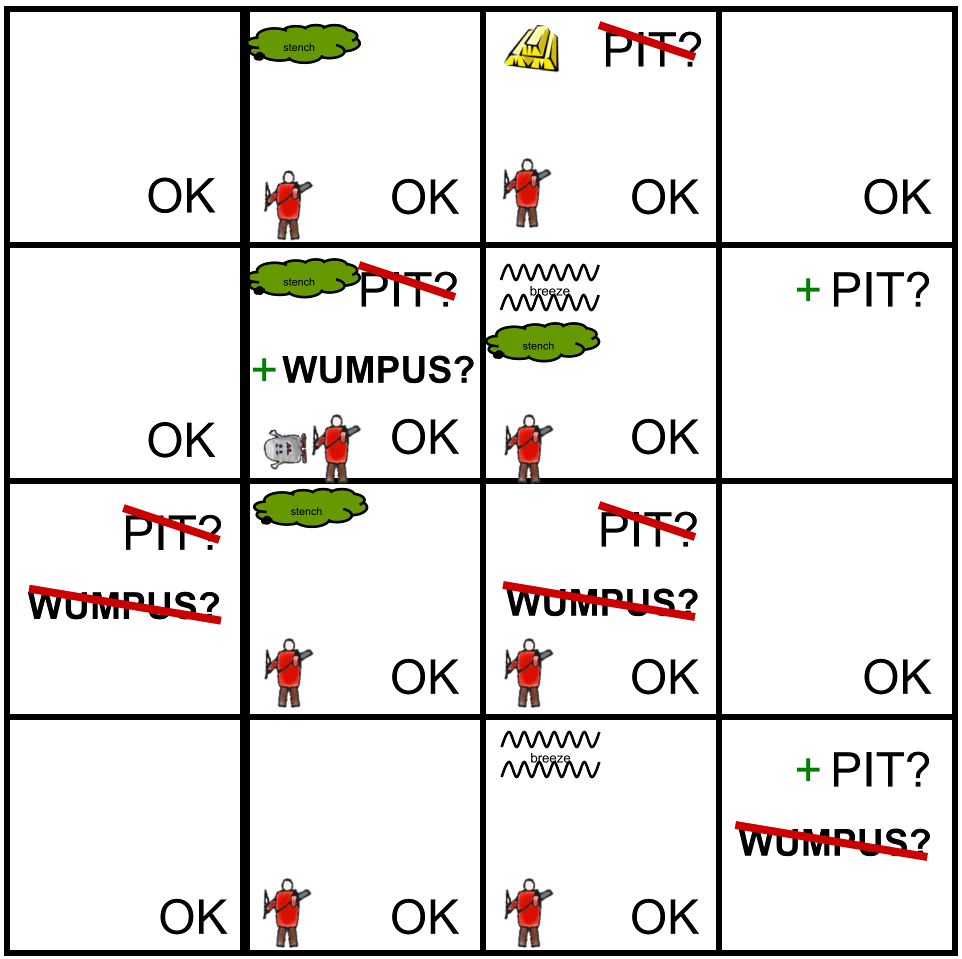
- Agent's percepts:
  - Stench
  - Breeze
  - Glitter
  - Bump
  - Scream
- Other than the agent, the world is static

Our First Wumpus Hunt



stench	breeze	glitter	bump	scream	
No	No	No	No	No	**South**
No	No	No	Yes	No	**East**
No	Yes	No	No	No	**West**
No	No	No	No	No	**North**
Yes	No	No	No	No	**East**
No	No	No	No	No	**North**
Yes	Yes	No	No	No	**Shoot(W)**
Yes	Yes	No	No	Yes	**West**
Yes	No	No	No	No	**North**
Yes	No	No	No	No	**East**
No	No	Yes	No	No	**Grab**

Annotated Wumpus Hunt



stench	breeze	glitter	bump	scream	
No	No	No	No	No	**South**
No	No	No	**Yes**	No	**East**
No	**Yes**	No	No	No	**West**
No	No	No	No	No	**North**
Yes	No	No	No	No	**East**
No	No	No	No	No	**North**
Yes	**Yes**	No	No	No	**Shoot(W)**
Yes	**Yes**	No	No	**Yes**	**West**
Yes	No	No	No	No	**North**
Yes	No	No	No	No	**East**
No	No	**Yes**	No	No	**Grab**

Today we will see how to build an agent that can perform this reasoning

Representing Beliefs

- In most programming languages, it is easy to specify statements like this…
  - *There is a pit in square [3,1]*
- But it is difficult to specify statements like these…
  - *There is a pit in either square [3,1] or [2,2]*
  - *There is no wumpus in square [2,2]*
  - *Because there was no breeze in square [1,2], there is a pit in square [3,1]*
- Require an agent that can represent this knowledge and perform the reasoning to infer new conclusions

Components of a Logic

- A formal system for representing the state of affairs
  - A sentence is a representation of a fact about the world
  - A syntax that describes how to make sentences
  - A semantics that gives constraints on how sentences relate to the state of affairs
  - A proof theory – a set of rules for deducing the entailments of a set of sentences

Entailment means that one thing follows from another

Properties of Logical Inference

- Inference is complete if it can find a proof for any sentence that is entailed
- A sentence is valid or necessarily true if and only if it is true under all possible interpretations in all possible worlds

  There is a stench in [1,1] or there is not a stench in [1,1]

- A sentence is satisfiable if and only if there is some interpretation in some world for which it is true

  There is a wumpus at [1,1]

Types of Commitment

Language	Ontological Commitment (What exists in the world)	Epistemological Commitment (What an agent believes about facts)
Propositional logic	facts	true/false/unknown
First-order logic	facts, objects, relations	true/false/unknown
Temporal logic	facts, objects, relations, times	true/false/unknown
Probability theory	facts	degree of belief $0 \ldots 1$
Fuzzy logic	degree of truth	degree of belief $0 \ldots 1$

- We make assumptions about
  - the world (ontological commitments)
  - the beliefs that an agent can hold (epistemological commitments)

Propositional Logic Syntax

- Basic Units (sentences)
  - *True* and *False*
  - Propositions *P, Q, …*
- Connectives

  $P \wedge Q$ and (conjunction)

  Returns true if both P and Q are true

  $P \vee Q$ or (disjunction)

  Returns true if either P or Q is true

  $P \Rightarrow Q$ implication

  If P is true then Q is also true

  $P \Leftrightarrow Q$ equivalence

  P is true exactly when Q is true

  $\neg P$ negation

  Returns true when P is false

Propositional Logic Grammar

- BNF (Backus-Naur form) for PL Grammar:

$$
\begin{aligned}
&Sentence \rightarrow AtomicSentence \mid ComplexSentence \\
&AtomicSentence \rightarrow True \mid False \mid P \mid Q \mid \ldots \\
&ComplexSentence \rightarrow (Sentence) \mid \\
&\qquad\qquad\qquad\qquad Sentence\ Connective\ Sentence \mid \\
&\qquad\qquad\qquad\qquad \neg Sentence \\
&Connective \rightarrow \wedge \mid \vee \mid \Rightarrow \mid \Leftrightarrow
\end{aligned}
$$

- Also require an order of precedence

From highest to lowest: $\neg \quad \wedge \quad \vee \quad \Rightarrow \quad \Leftrightarrow$

Propositional Logic Semantics

- Propositions can have any semantic meaning:

  P = “Paris is the capital of France”

  Q = “The wumpus is dead”

  R = “Bill Gates is the US President”

- Compound functions can be derived from a truth table:

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
False	*False*	*True*	*False*	*False*	*True*	*True*
False	*True*	*True*	*False*	*True*	*True*	*False*
True	*False*	*False*	*False*	*True*	*False*	*False*
True	*True*	*False*	*True*	*True*	*True*	*True*

Validity and Inference

$$((P \vee H) \wedge \neg H) \Rightarrow P$$

P	H	$P \vee H$	$(P \vee H) \wedge \neg H$	$((P \vee H) \wedge \neg H) \Rightarrow P$
False	*False*			
False	*True*			
True	*False*			
True	*True*			

- Truth tables can also be used to test validity of a sentence
- Remember to read implications as conditionals:

  $P \Rightarrow Q$ is read as "if P then Q"

Inference Rules for Propositional Logic

- Modus Ponens (Implication-Elimination)
  - From an implication and its premise, infer conclusion

$$\frac{\alpha \Rightarrow \beta\ ,\ \alpha}{\beta}$$

- And-Elimination
  - From a conjunction, you can infer any conjunct

$$\frac{\alpha_1 \wedge \alpha_2 \wedge \alpha_3 \wedge \ldots \wedge \alpha_n}{\alpha_i}$$

Inference Rules for Propositional Logic

- And-Introduction
  - From a list of sentences, you can infer the conjunct

$$\frac{\alpha_1, \alpha_2, \alpha_3, \ldots \alpha_n}{\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n}$$

- Or-Introduction
  - From a sentence, infer its disjunction with anything

$$\frac{\alpha_i}{\alpha_1 \vee \alpha_2 \vee \ldots \vee \alpha_n}$$

Inference Rules for Propositional Logic

- Double-Negative Elimination
  - From a double negation, infer the positive sentence

$$\frac{\neg\neg\alpha}{\alpha}$$

- Unit Resolution
  - From a disjunction in which one is false, then you can infer the other is true

$$\frac{\alpha \vee \beta\ ,\ \ \neg\beta}{\alpha}$$

Inference Rules for Propositional Logic

- Resolution
  - Since beta cannot be both true and false, one of the disjuncts must be true

$$\frac{\alpha \vee \beta \ , \neg \beta \vee \gamma}{\alpha \vee \gamma}$$

  - Implication is transitive

$$\frac{\neg \alpha \Rightarrow \beta \quad , \quad \beta \Rightarrow \gamma}{\neg \alpha \Rightarrow \gamma}$$

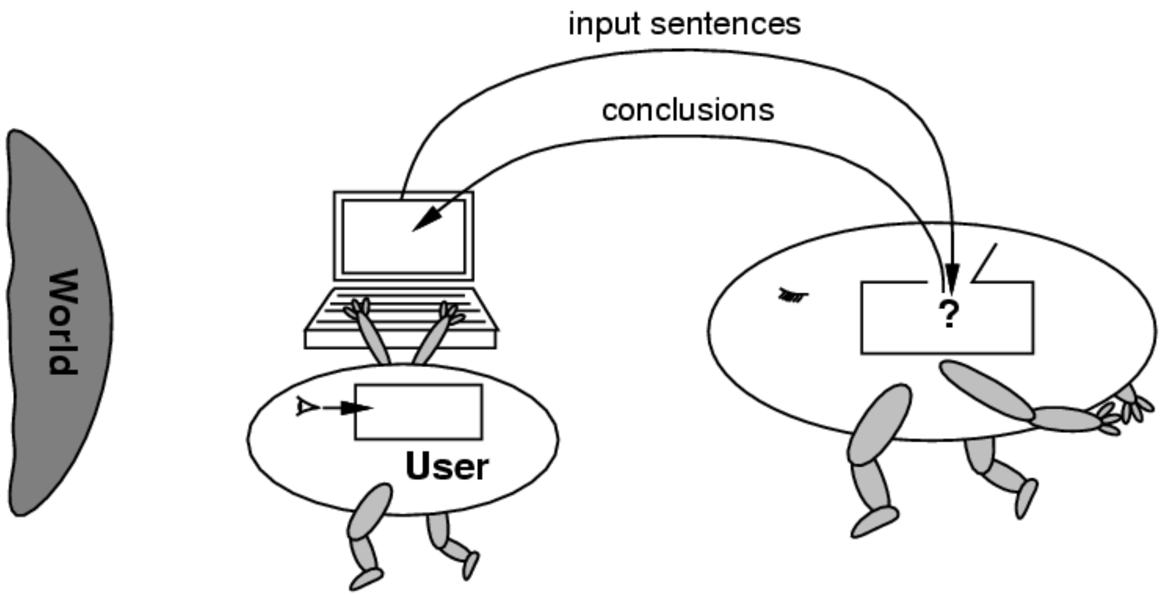
Truth Table for Resolution

α	β	γ	$\alpha \vee \beta$	$\neg\beta \vee \gamma$	$\alpha \vee \gamma$
False	*False*	*False*	*False*	*True*	*False*
False	*False*	*True*	*False*	*True*	*True*
False	*True*	*False*	*True*	*False*	*False*
<u>*False*</u>	<u>*True*</u>	<u>*True*</u>	<u>*True*</u>	<u>*True*</u>	<u>*True*</u>
<u>*True*</u>	<u>*False*</u>	<u>*False*</u>	<u>*True*</u>	<u>*True*</u>	<u>*True*</u>
<u>*True*</u>	<u>*False*</u>	<u>*True*</u>	<u>*True*</u>	<u>*True*</u>	<u>*True*</u>
True	*True*	*False*	*True*	*False*	*True*
<u>*True*</u>	<u>*True*</u>	<u>*True*</u>	<u>*True*</u>	<u>*True*</u>	<u>*True*</u>

- Truth tables can also be used to verify the inference rules

$$\frac{\alpha \vee \beta\ ,\ \neg\beta \vee \gamma}{\alpha \vee \gamma}$$

Logical Agents



- Input sentences can come from the user perceiving the world, or from a machine-readable representation of the world
- Infer new statements about the world that are valid

An Agent for the Wumpus World

- Convert perceptions into sentences:

  "In square [1,1], there is no breeze and no stench" … becomes…

  $\neg B_{11} \wedge \neg S_{11}$

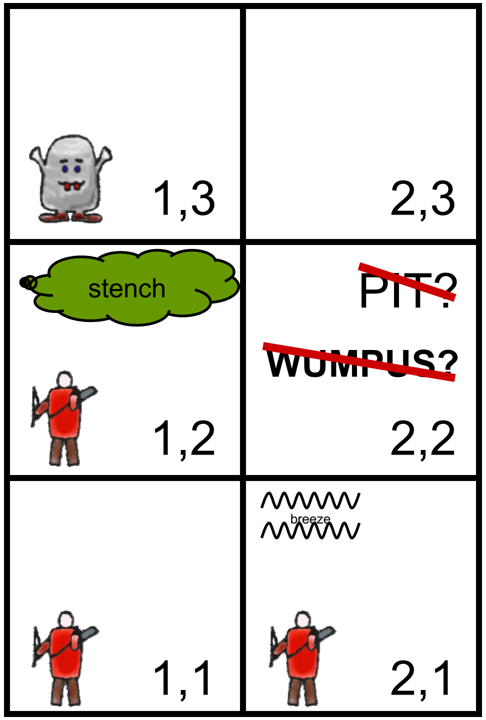
- Start with some knowledge of the world (in the form of rules)

  $R1 : \neg S_{11} \Rightarrow \neg W_{11} \wedge \neg W_{12} \wedge \neg W_{21}$

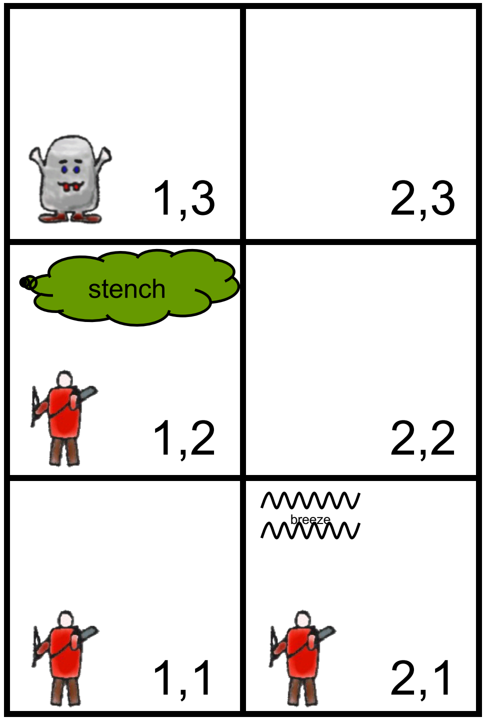
  $R2 : \neg S_{21} \Rightarrow \neg W_{11} \wedge \neg W_{21} \wedge \neg W_{22}$

  ….

  $R4 : S_{12} \Rightarrow W_{13} \vee W_{12} \vee W_{22} \vee W_{11}$



Finding the Wumpus



Percepts:

$\neg S_{11}$

$\neg S_{21}$

S_{12}

1. Apply modus ponens and and-elimination to $\neg S_{11} \Rightarrow \neg W_{11} \wedge \neg W_{12} \wedge \neg W_{21}$ to get

   $\neg \mathbf{W_{11}}$ $\neg \mathbf{W_{12}}$ $\neg \mathbf{W_{21}}$

2. Apply modus ponens and and-elimination to $\neg S_{21} \Rightarrow \neg W_{11} \wedge \neg W_{21} \wedge \neg W_{22}$ to get

   $\neg \mathbf{W_{22}}$ $\neg \mathbf{W_{21}}$ $\neg \mathbf{W_{31}}$

3. Apply modus ponens to
   $S_{12} \Rightarrow W_{13} \vee W_{12} \vee W_{22} \vee W_{11}$ to get

   $\mathbf{W_{13}} \vee \mathbf{W_{12}} \vee \mathbf{W_{22}} \vee \mathbf{W_{11}}$

4. Apply unit resolution to #3 and #1

   $\mathbf{W_{13}} \vee \mathbf{W_{22}}$

5. Apply unit resolution to #4 and #2

   $\mathbf{W_{13}}$

The wumpus is in square [1,3]!!!

Problems with Propositional Logic

- Too many propositions!
  - How can you encode a rule such as “don’t go forward if the wumpus is in front of you”?
  - In propositional logic, this takes (16 squares * 4 orientations) = 64 rules!
- Truth tables become unwieldy quickly
  - Size of the truth table is 2^n where n is the number of propositional symbols

More Problems with Propositional Logic

- No good way to represent changes in the world
  - How do you encode the location of the agent?
- What kinds of practical applications is this good for?
  - Relatively little

Coming Up…

- More powerful logic!
  - First-order logic (also known as First Order Predicate Calculus)