Midterm Review

AIMA CH 2-7

Chapter 1&2

Environments

* Observable / Partially Observable
* Deterministic / Stochastic
* Episodic / Sequential
* Static / Dynamic
* Discrete / Continuous
* Single-agent / Multi-agent

Chapter 1&2

Agent Designs

* Reflex
* Model-based
* Goal-based
* Utility

Chapter 3

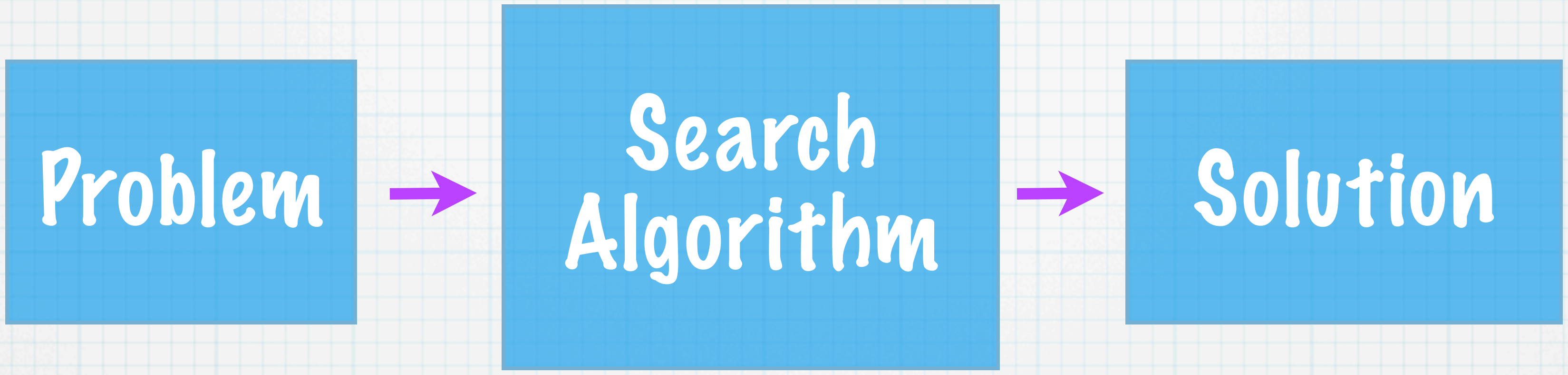
Classical Search

* Problem Formulation
* Tree Search
* Uninformed Strategies
* Informed Strategies

Problem Formulation

* The design decision of how to represent the agent's: actions, states, costs

Problem Solving Steps



- Formulate the **Goal**
- Formulate the **States,Actions,Costs**
- Find **Solution**
- **Execute** sequence of actions

Tree Search

* Root node is init state
* Transition model tells next states
* Goal test? yes -> done, no->expand more

Search Strategy: Which leaf node to expand 1st?

Tree Search

What's in my node representation?

* **State** this node represents
* **Parent** node that generated this
* **Action** that generated this
* **Cost** of path from init to here
* **Depth** of path from init to here

Node != State
represents path
to a state

General Tree Search

```
TreeSearch(problem) returns solution
    frontier={init-state}
    loop:
        1. if frontier  empty return failure
        2. choose leaf node, remove from frontier
        3. if node.contains(goal)  return
        node.solution
        4. node.expand(), add children to
        frontier
```

How they get added into queue is important

Evaluating Search Strategies

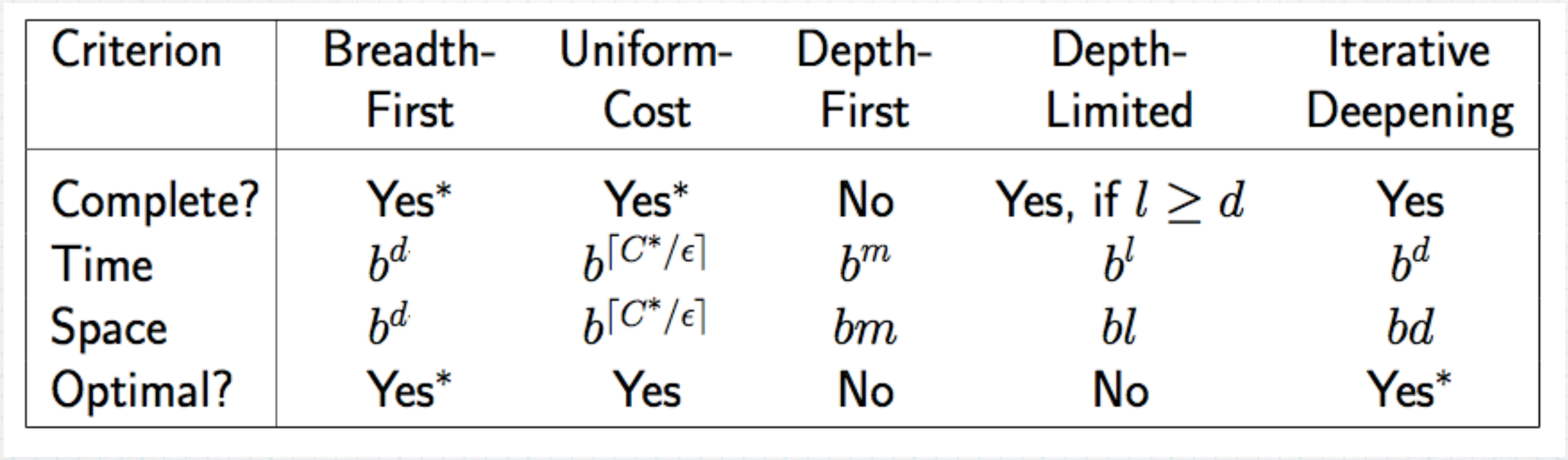
- Strategy = order of node expansion
  - Complete: finds solution if one exists
  - Optimal: always finds least cost solution
  - Time Complexity: # nodes generated
  - Space Complexity: max nodes ever in memory

Uninformed Search

Use only the info in the problem definition

* Breadth-first search
* Uniform-cost search
* Depth-first search
* Depth-limited search
* Iterative-deepening search

Comparison of Algs



Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes*	Yes*	No	Yes, if $l \geq d$	Yes
Time	b^d	$b^{\lceil C^*/\epsilon \rceil}$	b^m	b^l	b^d
Space	b^d	$b^{\lceil C^*/\epsilon \rceil}$	bm	bl	bd
Optimal?	Yes*	Yes	No	No	Yes*

- BFS vs. DFS
  - memory
- DFS vs. D-limited vs. Iterative-D
  - time

Summary for Uninformed

* Variety of search strategies
* Iterative deepening uses only linear space and not much more time than other algorithms
* Graph search can be exponentially more efficient than tree search

Informed Search

- What if you know more...
  - Designer knows something about the problem to help the agent
  - Domain knowledge
- Use this to expand the **BEST** node first

Best-First Search

* Tree search + Evaluation Function
* f(n) = desirability of node n

Search Strategy: How to define eval function

Best-First Search

* Heuristic function h(n)
    * estimated cheapest path, n to goal
    * estimated future path cost from n

Greedy Best-First

- $f(n) = h(n)$: expand node that looks closest from here
- Example -- a common heuristic for route planning: straight line distance to goal

A* Search

- Most widely known Best-First alg
- f(n) = g(n) + h(n)
    - g(n) = cost to get to this node
    - h(n) = estimated cost from here
- Minimizes total solution cost

Admissible Heuristic

- h(n) =
  - under-estimate of cost to goal
  - zero for any goal state
  - non-zero for all others
- Makes A* Optimal & Complete!

Dominance

If $h_2(n) \geq h_1(n)$ for all n (both admissible)
then h_2 dominates h_1 and is better for search

Typical search costs:

$d = 14$	IDS $=$ 3,473,941 nodes
	A$^*(h_1)$ $=$ 539 nodes
	A$^*(h_2)$ $=$ 113 nodes
$d = 24$	IDS $\approx$ 54,000,000,000 nodes
	A$^*(h_1)$ $=$ 39,135 nodes
	A$^*(h_2)$ $=$ 1,641 nodes

Given any admissible heuristics h_a, h_b,

$$h(n) = \max(h_a(n), h_b(n))$$

is also admissible and dominates h_a, h_b

Chapter 4

Local Search

- Classical Search
  - Solution = path to goal state
- Local Search
  - Solution = goal state itself
  - How to search for a solution when the path doesn't matter?

Local Search

* **Complete** = always finds a goal state if there is one
* **Optimal** = always find the global max

Local Search Algs

* Hill-Climbing
* Simulated Annealing
* Local Beam Search
* Genetic Algorithms

Online Search

* Offline: simulate the world and reason about a plan to get to a goal
* Online: solve the search problem while executing actions
* Interleaves search and execution

Chapter 6

Constraint Satisfaction Problems

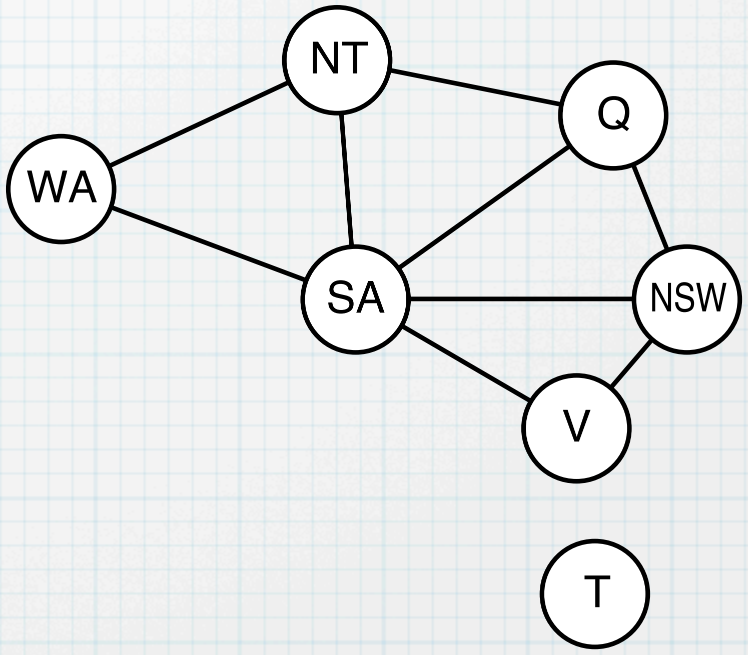
- Problem Formulation
- Backtracking Search
    - Variable order heuristics
    - Value order heuristics
    - Constraint Propagation

Constraint Satisfaction Problem

- State: variables X_i, values from domain D_i
- Goal Test: constraints specifying allowable combinations of values for variables
- Formal Representation Language
- Allows general-purpose algorithms with more power than standard search algs

Constraint Graph

* Nodes are variables
* Arcs show constraints
* Binary CSP: each constraint relates only 2 variables
* CSP algorithms use the graph structure to speed up search for a goal state configuration



Search for CSP Solution

* Init State = {}
* Successor() = assign value (consistent with constraints) to any unassigned variable
* Goal Test = all vars are assigned
* Fail if no legal assignment to do

* Same formulation for every CSP problem, yea!
* Problem: n vars, with d values, branch factor at root is nd, then (n-1)d ...terrible!
* Order doesn't matter, consider 1 var at a time

Backtracking Search

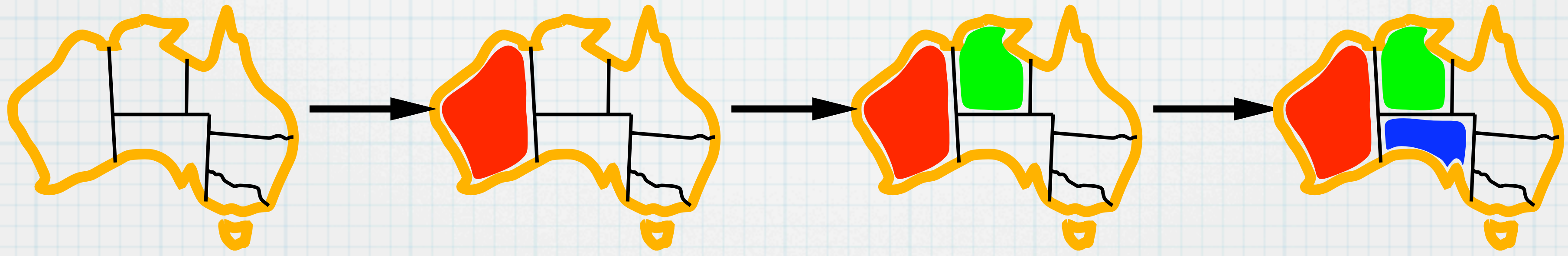
```
function BACKTRACKING-SEARCH($csp$) returns solution/failure
    return RECURSIVE-BACKTRACKING({ }, $csp$)

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 $failure$
```

* Repeatedly choose value for unassigned var, return fail if inconsistency detected

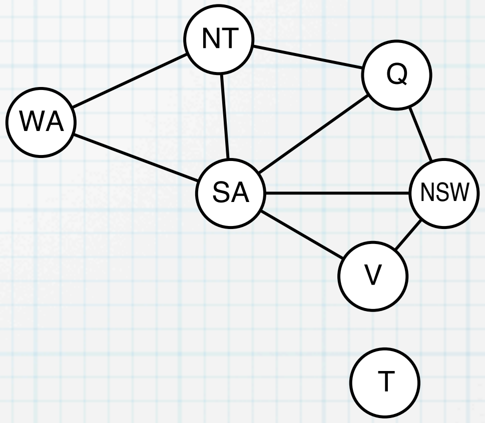
Minimum Remaining Values (MRV)

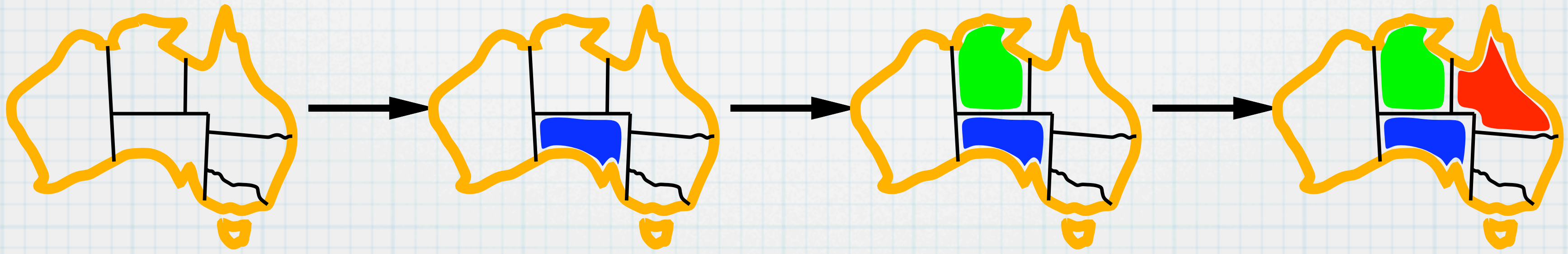
* What variable to do next?
* Choose var with the fewest legal values



Degree Heuristic

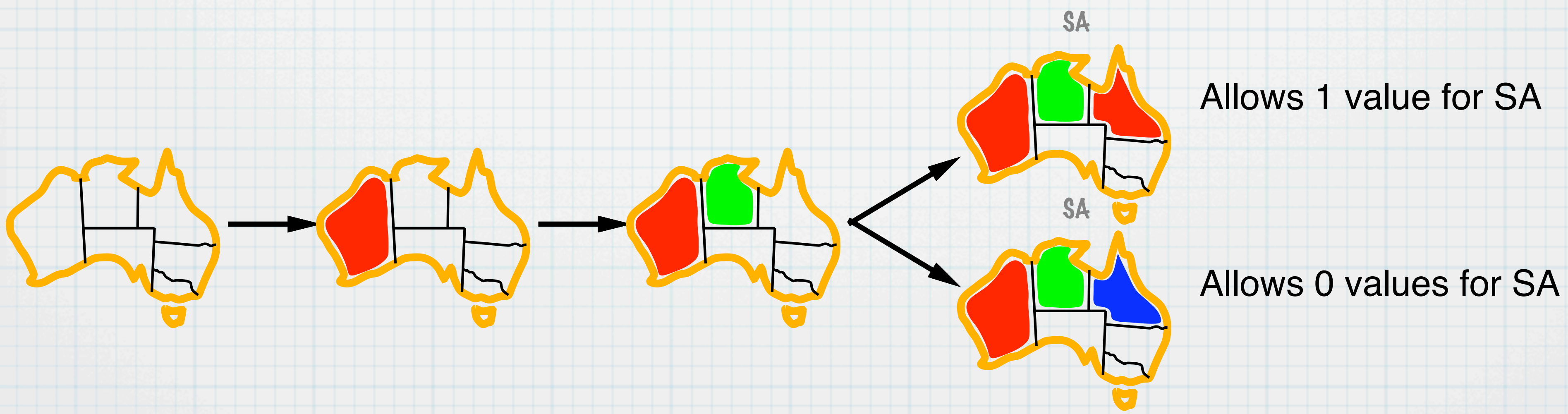
* Tie breaker for MRV
* Choose var with most constraints on remaining variables





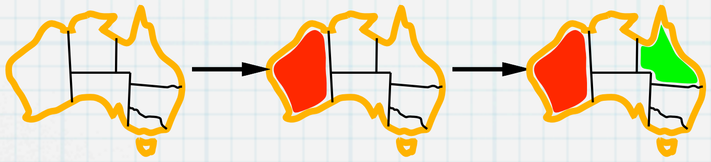
Least Constraining Value

* What value to try next?
* Given a variable, choose value that rules out the least values in remaining vars



Constraint Propagation

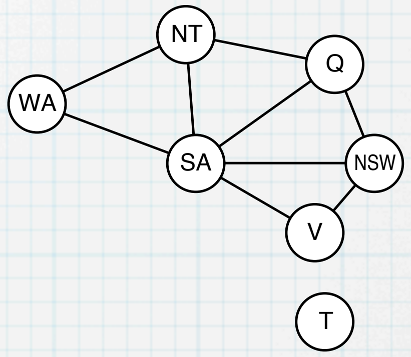
- Can stop a branch even earlier by propagating constraints and values
- After deleting neighbors run constraints



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

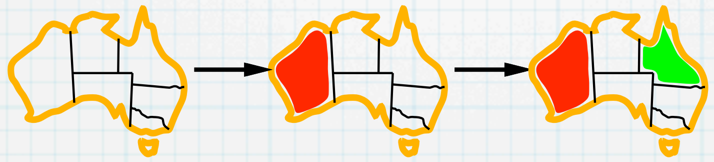
NT and SA cannot both be blue!

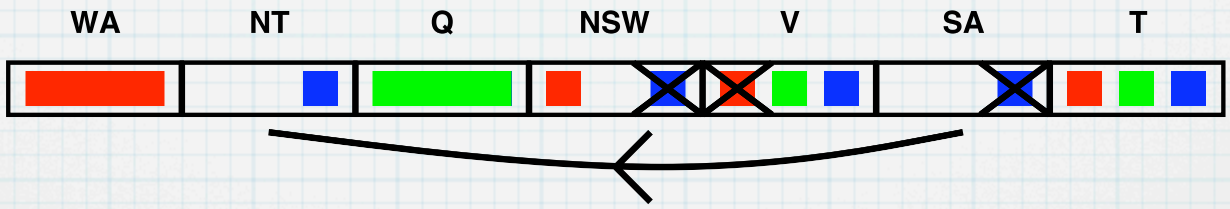
Arc Consistency



Simplest form of propagation makes each arc consistent

$X \to Y$ 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

Chapter 7

Logical Agents

* Propositional Logic
* Inference in Propositional Logic
    * by Enumeration
    * Forward/Backward Chaining
    * Resolution

Simple KB-agent

```
**function** KB-AGENT( percept) **returns** an action
    **static**: KB, a knowledge base
            t, a counter, initially 0, indicating time

    TELL(KB, MAKE-PERCEPT-SENTENCE( percept, t))
    action ← ASK(KB, MAKE-ACTION-QUERY(t))
    TELL(KB, MAKE-ACTION-SENTENCE(action, t))
    t ← t + 1
    **return** action
```

The agent must be able to:

- Represent states, actions, etc.
- Incorporate new percepts
- Update internal representations of the world
- Deduce hidden properties of the world
- Deduce appropriate actions

Propositional Logic: Syntax

Propositional logic is the simplest logic—illustrates basic ideas

The proposition symbols P_1, P_2 etc are sentences

If S is a sentence, $\neg S$ is a sentence (negation)

If S_1 and S_2 are sentences, $S_1 \wedge S_2$ is a sentence (conjunction)

If S_1 and S_2 are sentences, $S_1 \vee S_2$ is a sentence (disjunction)

If S_1 and S_2 are sentences, $S_1 \;\Rightarrow\; S_2$ is a sentence (implication)

If S_1 and S_2 are sentences, $S_1 \;\Leftrightarrow\; S_2$ is a sentence (biconditional)

Truth Table for Connectives

Model		Truth value w.r.t. given Model				
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*

Inference by Enumeration

```
function TT-ENTAILS?($KB, \alpha$) returns $true$ or $false$
    inputs: $KB$, the knowledge base, a sentence in propositional logic
            $\alpha$, the query, a sentence in propositional logic

    $symbols \leftarrow$ a list of the proposition symbols in $KB$ and $\alpha$
    return TT-CHECK-ALL($KB, \alpha, symbols, [\,]$)
```

```
function TT-CHECK-ALL($KB, \alpha, symbols, model$) returns $true$ or $false$
    if EMPTY?($symbols$) then
        if PL-TRUE?($KB, model$) then return PL-TRUE?($\alpha, model$)
        else return $true$
    else do
        $P \leftarrow$ FIRST($symbols$); $rest \leftarrow$ REST($symbols$)
        return TT-CHECK-ALL($KB, \alpha, rest$, EXTEND($P, true, model$)) and
                   TT-CHECK-ALL($KB, \alpha, rest$, EXTEND($P, false, model$))
```

* DFS enumeration of all variables
* Checking if query T everywhere KB is T

Truth Tables for Inference

Model							KB sentences					
$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	R_1	R_2	R_3	R_4	R_5	KB
false	*false*	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*false*	*false*
false	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*false*	*true*	*false*	*false*
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
false	*true*	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*false*	*true*	*true*	*false*
false	*true*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*false*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*true*	*false*	*false*	*true*	*false*	*false*	*true*	*true*	*false*
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
true	*true*	*true*	*true*	*true*	*true*	*true*	*false*	*true*	*true*	*false*	*true*	*false*

Enumerate rows (different assignments to symbols),
if KB is true in row, check that α is too

Inference as Search

* **Init State:** initial KB
* **Transition model:** all logical inference rules and resulting additions to the KB
* **Goal:** KB that contains the sentence we are trying to prove

Forward and Backward Chaining

Horn Form (restricted)

KB = **conjunction** of **Horn clauses**

Horn clause =

- ♢ proposition symbol; or
- ♢ (conjunction of symbols) $\Rightarrow$ symbol

E.g., $C \wedge (B \Rightarrow A) \wedge (C \wedge D \Rightarrow B)$

Modus Ponens (for Horn Form): complete for Horn KBs

$$\frac{\alpha_1, \ldots, \alpha_n, \qquad \alpha_1 \wedge \cdots \wedge \alpha_n \Rightarrow \beta}{\beta}$$

Can be used with forward chaining or backward chaining.
These algorithms are very natural and run in **linear** time

Forward Chaining

```
function PL-FC-ENTAILS?($KB$, $q$) returns $true$ or $false$
  inputs: $KB$, the knowledge base, a set of propositional Horn clauses
          $q$, the query, a proposition symbol
  local variables: $count$, a table, indexed by clause, initially the number of premises
                   $inferred$, a table, indexed by symbol, each entry initially $false$
                   $agenda$, a list of symbols, initially the symbols known in $KB$

  while $agenda$ is not empty do
      $p \leftarrow$ POP($agenda$)
      unless $inferred[p]$ do
          $inferred[p] \leftarrow true$
          for each Horn clause $c$ in whose premise $p$ appears do
              decrement $count[c]$
              if $count[c] = 0$ then do
                  if HEAD[$c$] $= q$ then return $true$
                  PUSH(HEAD[$c$], $agenda$)
  return $false$
```

Resolution

Conjunctive Normal Form (CNF—universal)

$$\textbf{conjunction} \text{ of } \underbrace{\textbf{disjunctions} \text{ of } \textbf{literals}}_{\textbf{clauses}}$$

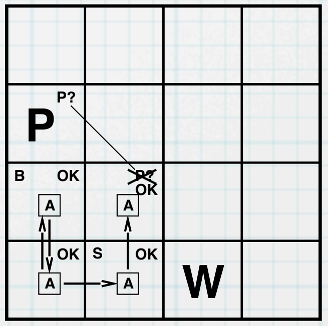
E.g., $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$

Resolution inference rule (for CNF): complete for propositional logic

$$\frac{\ell_1 \vee \cdots \vee \ell_k, \qquad m_1 \vee \cdots \vee m_n}{\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n}$$

where ℓ_i and m_j are complementary literals. E.g.,

$$\frac{P_{1,3} \vee P_{2,2}, \qquad \neg P_{2,2}}{P_{1,3}}$$



Resolution is sound and complete for propositional logic

Example CNF Convert

- Great! Can we make everything CNF?
- Convert sentences to Conjunctive Normal Form with our rules of logical equivalence

$(\alpha \land \beta)$	$\equiv$	$(\beta \land \alpha)$	commutativity of $\land$
$(\alpha \lor \beta)$	$\equiv$	$(\beta \lor \alpha)$	commutativity of $\lor$
$((\alpha \land \beta) \land \gamma)$	$\equiv$	$(\alpha \land (\beta \land \gamma))$	associativity of $\land$
$((\alpha \lor \beta) \lor \gamma)$	$\equiv$	$(\alpha \lor (\beta \lor \gamma))$	associativity of $\lor$
$\neg(\neg\alpha)$	$\equiv$	α	double-negation elimination
$(\alpha \Rightarrow \beta)$	$\equiv$	$(\neg\beta \Rightarrow \neg\alpha)$	contraposition
$(\alpha \Rightarrow \beta)$	$\equiv$	$(\neg\alpha \lor \beta)$	implication elimination
$(\alpha \Leftrightarrow \beta)$	$\equiv$	$((\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha))$	biconditional elimination
$\neg(\alpha \land \beta)$	$\equiv$	$(\neg\alpha \lor \neg\beta)$	De Morgan
$\neg(\alpha \lor \beta)$	$\equiv$	$(\neg\alpha \land \neg\beta)$	De Morgan
$(\alpha \land (\beta \lor \gamma))$	$\equiv$	$((\alpha \land \beta) \lor (\alpha \land \gamma))$	distributivity of $\land$ over $\lor$
$(\alpha \lor (\beta \land \gamma))$	$\equiv$	$((\alpha \lor \beta) \land (\alpha \lor \gamma))$	distributivity of $\lor$ over $\land$

Example CNF Convert

$$\mathrm{B}_{11} \Leftrightarrow (P_{12} \vee P_{21})$$

Biconditional Elimination

$$\mathrm{B}_{11} \Rightarrow (P_{12} \vee P_{21}) \wedge (P_{12} \vee P_{21}) \Rightarrow B_{11}$$

Implication Elimination

$$(\neg B_{11} \vee P_{12} \vee P_{21}) \wedge (\neg(P_{12} \vee P_{21}) \vee B_{11})$$

Move neg. in, DeMorgans

$$(\neg B_{11} \vee P_{12} \vee P_{21}) \wedge ((\neg P_{12} \wedge \neg P_{21}) \vee B_{11})$$

Distribute over and/or

$$(\neg B_{11} \vee P_{12} \vee P_{21}) \wedge (\neg P_{12} \vee B_{11}) \wedge (\neg P_{21} \vee B_{11})$$

Resolution Algorithm

Proof by contradiction, i.e., show $KB \wedge \neg\alpha$ unsatisfiable

```
function PL-Resolution($KB, \alpha$) returns true or false
    inputs: $KB$, the knowledge base, a sentence in propositional logic
            $\alpha$, the query, a sentence in propositional logic

    clauses ← the set of clauses in the CNF representation of $KB \wedge \neg\alpha$
    new ← { }
    loop do
        for each $C_i$, $C_j$ in clauses do
            resolvents ← PL-Resolve($C_i, C_j$)
            if resolvents contains the empty clause then return true
            new ← new $\cup$ resolvents
        if new $\subseteq$ clauses then return false
        clauses ← clauses $\cup$ new
```