

# Inf2D 01: Intelligent Agents and their Environments

Peggy Series & Michael Herrmann

University of Edinburgh, School of Informatics

17/01/2017

**informatics**



Credits: The content of this lecture was prepared by Michael Rovatsos and follows R&N

# Structure of Intelligent Agents

An agent:

- Perceives its **environment**,
- Through its **sensors**,
- Then achieves its **goals**
- By acting on its environment via **actuators**.

# Examples of Agents 1

- **Agent:** mail sorting robot
- **Environment:** conveyor belt of letters
- **Goals:** route letter into correct bin
- **Percepts:** array of pixel intensities
- **Actions:** route letter into bin

# Examples of Agents 2

- **Agent:** intelligent house
- **Environment:**
  - occupants enter and leave house,
  - occupants enter and leave rooms;
  - daily variation in outside light and temperature
- **Goals:** occupants warm, room lights are on when room is occupied, house energy efficient
- **Percepts:** signals from temperature sensor, movement sensor, clock, sound sensor
- **Actions:** room heaters on/off, lights on/off

# Examples of Agents 3

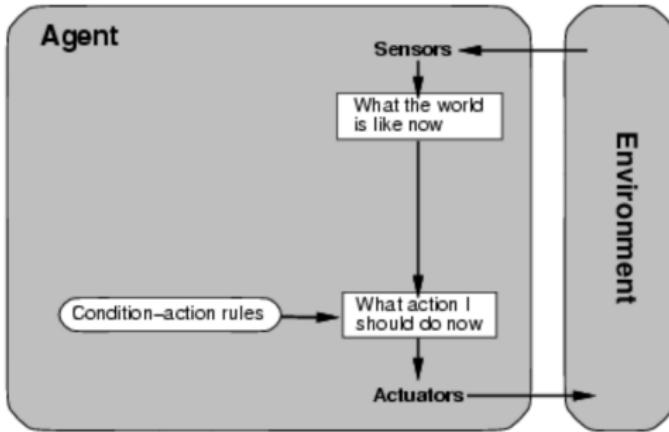
- **Agent**: automatic car.
- **Environment**: streets, other vehicles, pedestrians, traffic signals/lights/signs.
- **Goals**: safe, fast, legal trip.
- **Percepts**: camera, GPS signals, speedometer, sonar.
- **Actions**: steer, accelerate, brake.

Side info: [http://en.wikipedia.org/wiki/2005\\_DARPA\\_Grand\\_Challenge](http://en.wikipedia.org/wiki/2005_DARPA_Grand_Challenge)

# Simple Reflex Agents

- Action depends only on immediate percepts.
- Implement by condition-action rules.
- Example:
  - Agent: Mail sorting robot
  - Environment: Conveyor belt of letters
  - Rule: e.g.  $city=Edin \rightarrow put\ Scotland\ bag$

# Simple Reflex Agents



```
function SIMPLE-REFLEX-AGENT(percept)
returns action

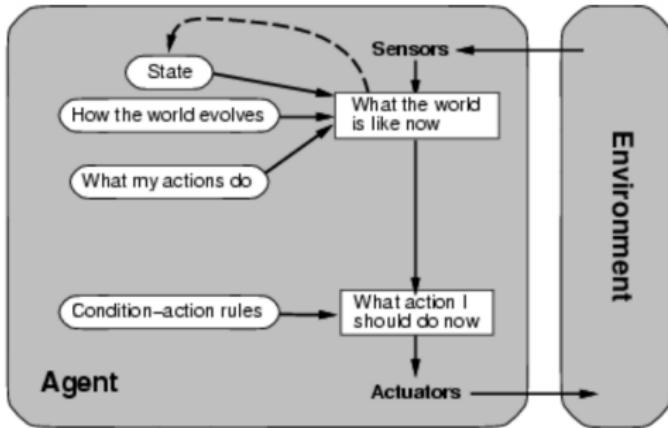
persistent: rules (set of condition-action rules)
    state  $\leftarrow$  INTERPRET-INPUT(percept)
    rule  $\leftarrow$  RULE-MATCH(state, rules)
    action  $\leftarrow$  rule.ACTION

return action
```

# Model-Based Reflex Agents

- Action may depend on history or unperceived aspects of the world.
- Need to maintain **internal world model**.
- **Example:**
  - **Agent:** robot vacuum cleaner
  - **Environment:** dirty room, furniture.
  - **Model:** map of room, which areas already cleaned.
  - Sensor/model trade-off.

# Model-Based Agents

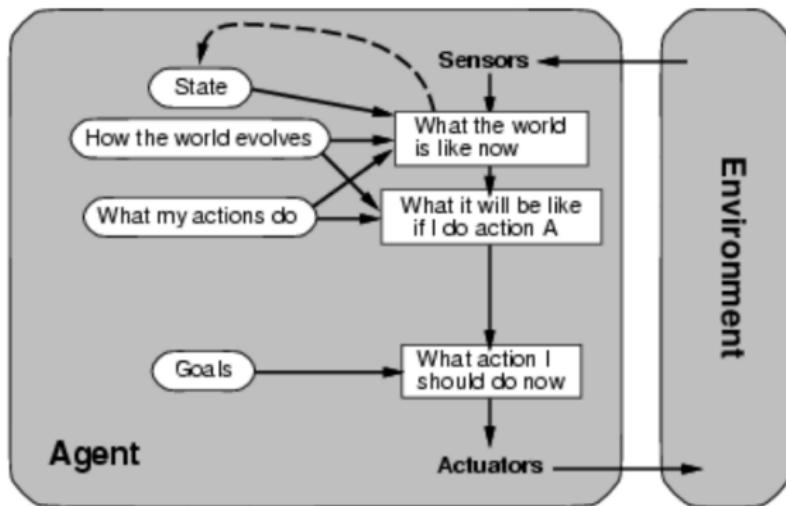


```
function REFLEX-AGENT-WITH-STATE(percept)
  returns action
  persistent: state, description of current world state
  model, description of how the next state depends on
    current state and action
  rules, a set of condition-action rules
  action, the most recent action, initially none
  state  $\leftarrow$  UPDATE-STATE(state, action, percept, model)
  rule  $\leftarrow$  RULE-MATCH(state, rules)
  action  $\leftarrow$  rule.ACTION
  return action
```

# Goal-Based Agents

- Agents so far have fixed, implicit goals.
- We want agents with variable goals.
- Forming plans to achieve goals is later topic.
- **Example:**
  - **Agent:** household service robot
  - **Environment:** house & people.
  - **Goals:** clean clothes, tidy room, table laid, etc.

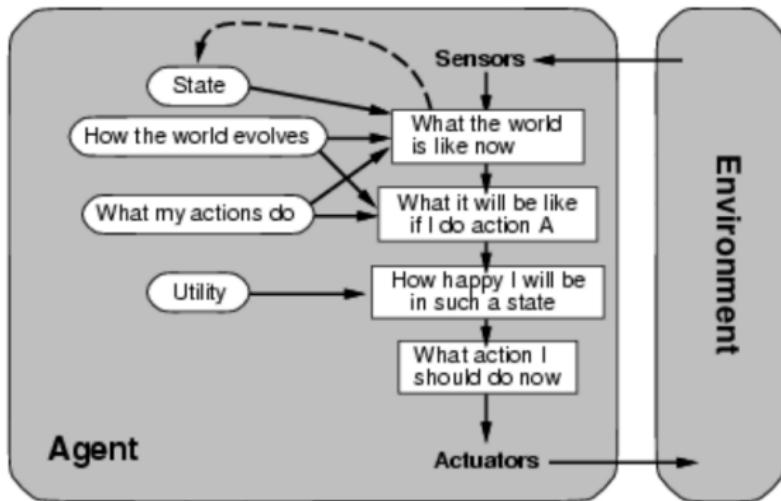
# Goal-Based Agents



# Utility-Based Agents

- Agents so far have had a single goal.
- Agents may have to juggle conflicting goals.
- Need to optimise utility over a range of goals.
- Utility: measure of *goodness* (a real number).
- Combine with probability of success to get *expected utility*.
- Example:
  - Agent: automatic car.
  - Environment: roads, vehicles, signs, etc.
  - Goals: stay safe, reach destination, be quick, obey law, save fuel, etc.

# Utility-Based Agents



We will not be covering utility-based agents, but this topic is discussed in Russell & Norvig, Chapters 16 and 17.

# Learning Agents

How do agents improve their performance in the light of experience?

- Generate problems which will test performance.
- Perform activities according to rules, goals, model, utilities, etc.
- Monitor performance and identify non-optimal activity.
- Identify and implement improvements.

We will not be covering learning agents, but this topic is discussed in Russell & Norvig, Chapters 18-21.

# Mid-Lecture Problem

Consider a chess playing program.

What sort of agent would it need to be?

# Solution(s)

- **Simple-reflex agent**: but some actions require some memory (e.g. castling in chess - <http://en.wikipedia.org/wiki/Castling>).
- **Model-based reflex agent**: but needs to reason about future.
- **Goal-based agent**: but only has one goal.
- **Utility-based agent**: might consider multiple goals with limited lookahead.

# Types of Environment 1

- Fully Observable vs. Partially Observable:
  - Observable: agent's sensors describe environment fully.
  - Playing chess with a blindfold.
- Deterministic vs. Stochastic:
  - Deterministic: next state fully determined by current state and agent's actions.
  - Chess playing in a strong wind.

An environment may appear stochastic if it is only partially observable.

# Types of Environment 2

- Episodic vs. Sequential:

- Episodic: next episode does not depend on previous actions.
- Mail-sorting robot vs. crossword puzzle.

- Static vs. Dynamic:

- Static: environment unchanged while agent deliberates.
- Crossword puzzle vs. chess.
- Industrial robot vs. robot car

# Types of Environment 3

- Discrete vs. Continuous:
  - Discrete: percepts, actions and episodes are discrete.
  - Chess vs. robot car.
- Single Agent vs. Multi-Agent:
  - How many objects must be modelled as agents.
  - Crossword vs. poker.

Element of choice over which objects are considered agents.

# Types of Environment 4

- An agent might have any combination of these properties:
  - from “benign” (i.e., fully observable, deterministic, episodic, static, discrete and single agent)
  - to “chaotic” (i.e., partially observable, stochastic, sequential, dynamic, continuous and multi-agent).
- What are the properties of the environment that would be experienced by
  - a mail-sorting robot?
  - an intelligent house?
  - a car-driving robot?

# Summary

- Simple reflex agents
- Model-based reflex agents
- Goal-based agents
- Utility-based agents
- Learning agents
- Properties of environments

# Inf2D 02: Problem Solving by Searching

Michael Herrmann

University of Edinburgh, School of Informatics

19/01/2017

**informatics**



Credits: The content of this lecture was prepared by Michael Rovatsos and follows R&N

# Outline

- Problem-solving agents
- Problem types
- Problem formulation
- Example problems
- Basic search algorithms

# Problem-solving agents

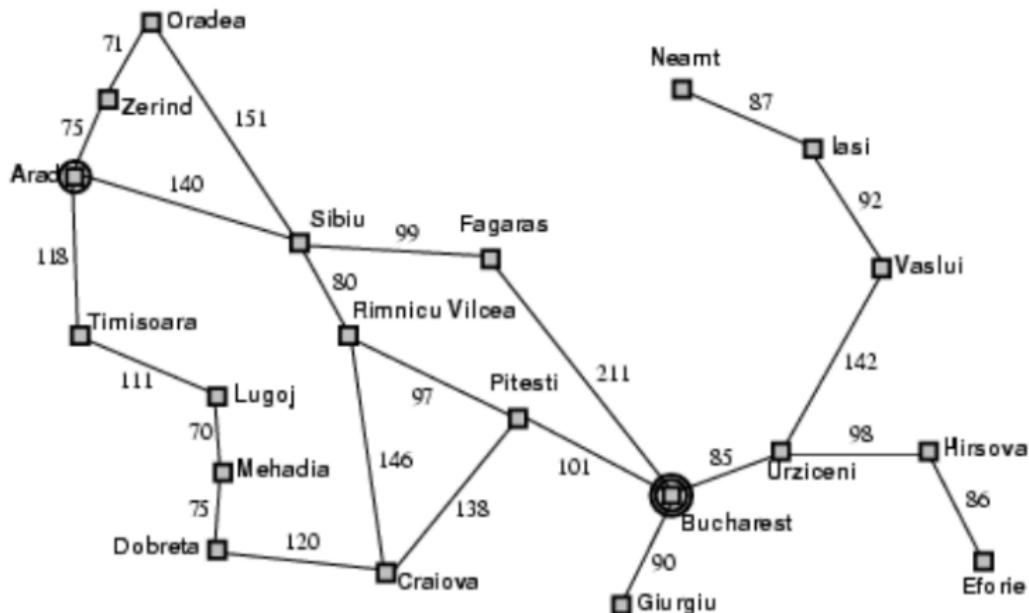
```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
    persistent: seq, an action sequence, initially empty
                state, some description of the current world state
                goal, a goal, initially null
                problem, a problem formulation
    state  $\leftarrow$  UPDATE-STATE(state, percept)
    if seq is empty then do
        goal  $\leftarrow$  FORMULATE-GOAL(state)
        problem  $\leftarrow$  FORMULATE-PROBLEM(state, goal)
        seq  $\leftarrow$  SEARCH(problem)
        if seq = failure then return a null action
        action  $\leftarrow$  FIRST(seq)
        seq  $\leftarrow$  REST(seq)
    return action
```

Agent has a “Formulate, Search, Execute” design

## Example: Romania

- On holiday in Romania; currently in Arad.
- Flight leaves tomorrow from Bucharest
- Formulate goal:
  - be in Bucharest
- Formulate problem:
  - states: various cities
  - actions: drive between cities
- Find solution:
  - sequence of cities, e.g., Arad, Sibiu, Fagaras, Bucharest

# Example: Romania



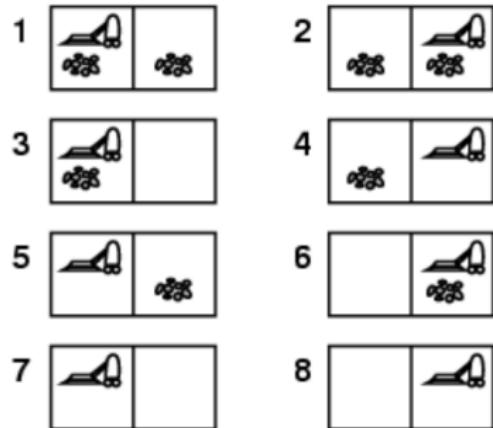
# Problem types

- Deterministic, fully observable → single-state problem
  - Agent knows exactly which state it will be in; solution is a sequence
- Non-observable → sensorless problem (conformant problem)
  - Agent may have no idea where it is; solution is a sequence
- Nondeterministic and/or partially observable → contingency problem –
  - percepts provide new information about current state
  - often interleave search, execution
- Unknown state space → exploration problem

# Example: vacuum world

- Single-state, start in #5.

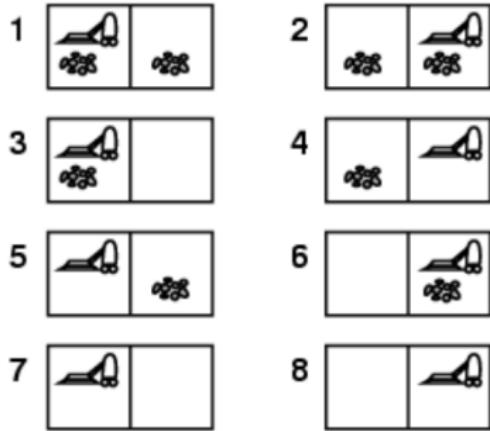
Solution?



# Example: vacuum world

- Single-state, start in #5.

Solution: [Right, Suck]



- Sensorless, start in  $\{1, 2, 3, 4, 5, 6, 7, 8\}$  e.g., Right goes to  $\{2, 4, 6, 8\}$

Solution?

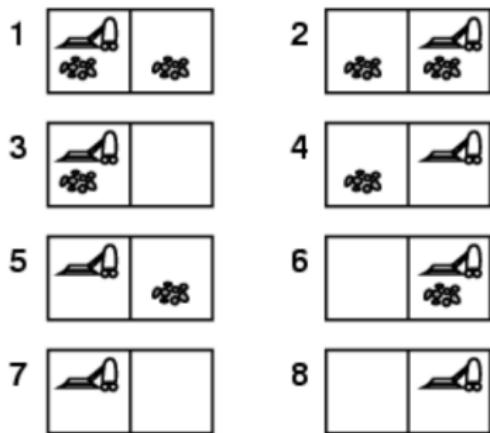
# Example: vacuum world

- Sensorless, start in  $\{1, 2, 3, 4, 5, 6, 7, 8\}$  e.g., Right goes to  $\{2, 4, 6, 8\}$

Solution:  $[Right, Suck, Left, Suck]$

- Contingency

- Nondeterministic:  
Suck may dirty a  
clean carpet
- Partially observable:  
location, dirt at  
current location.
- Percept:  $[L, Clean]$ ,  
i.e., start in #5 or #7

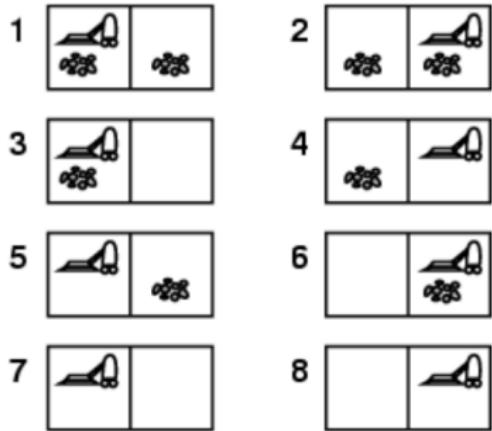


Solution?

# Example: vacuum world

- Contingency

- Nondeterministic:  
Suck may dirty a  
clean carpet
- Partially observable:  
location, dirt at  
current location.
- Percept:  $[L, Clean]$ ,  
i.e., start in #5 or #7



Solution: [Right, if dirt then Suck]

# Single-state problem formulation

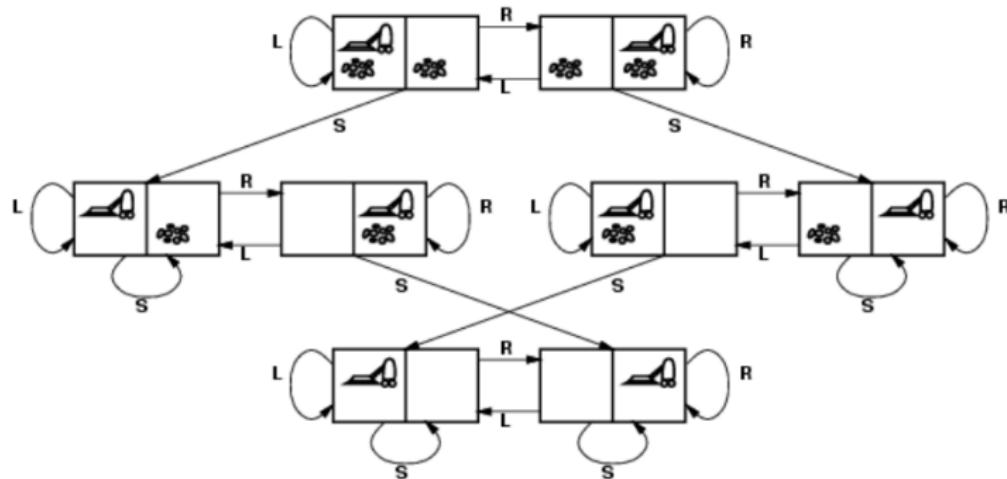
A problem is defined by four items:

- ① initial state e.g., "in Arad"
  - ② actions or successor function  $S(x) = \text{set of action-state pairs}$ 
    - e.g.,  $S(\text{Arad}) = \{\langle \text{Arad} \rightarrow \text{Zerind}, \text{Zerind} \rangle, \dots\}$
  - ③ goal test, can be
    - explicit, e.g.,  $x = \text{"in Bucharest"}$
    - implicit, e.g.,  $\text{Checkmate}(x)$
  - ④ path cost (additive)
    - e.g., sum of distances, number of actions executed, etc.
    - $c(x, a, y)$  is the step cost of taking action  $a$  in state  $x$  to reach state  $y$ , assumed to be  $\geq 0$
- A solution is a sequence of actions leading from the initial state to a goal state

# Selecting a state space

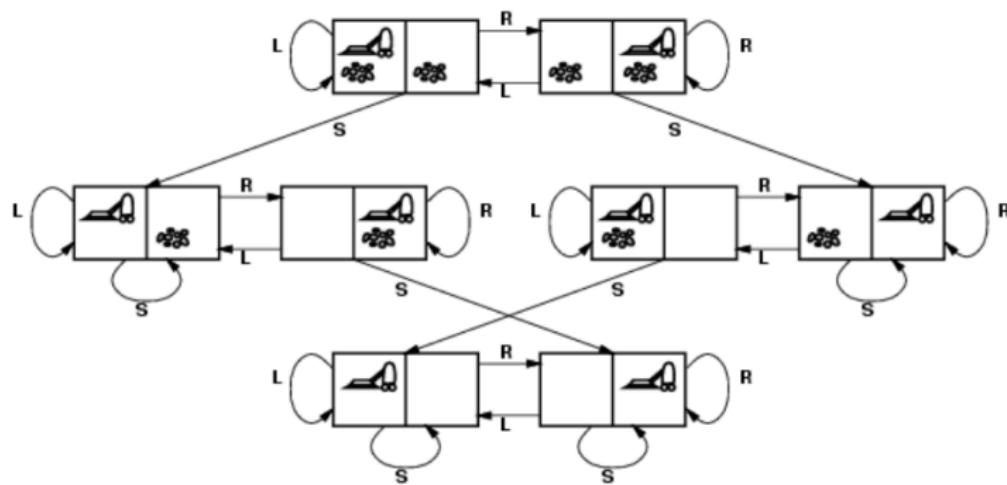
- Real world is absurdly complex → state space must be abstracted for problem solving
- (Abstract) state = set of real states
- (Abstract) action = complex combination of real actions
  - e.g., "Arad → Zerind" represents a complex set of possible routes, detours, rest stops, etc.
- For guaranteed realizability, **any** real state "in Arad" must get to some real state "in Zerind"
- (Abstract) solution =
  - set of real paths that are solutions in the real world
- Each abstract action should be "easier" than the original problem

# Vacuum world state space graph



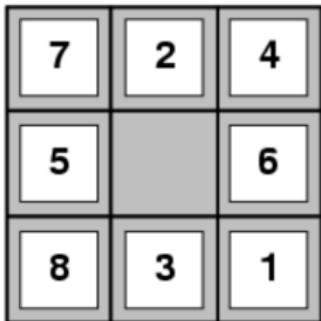
- states?
- actions?
- goal test?
- path cost?

# Vacuum world state space graph

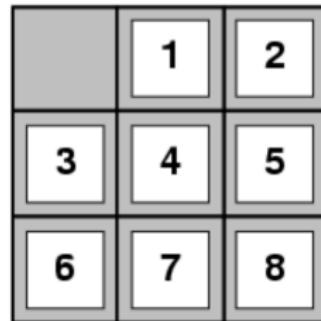


- **states?** Pair of dirt and robot locations
- **actions?** Left, Right, Suck
- **goal test?** no dirt at any location
- **path cost?** 1 per action

# Example: The 8-puzzle



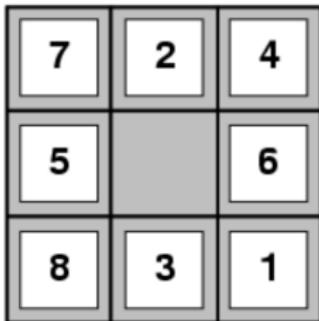
Start State



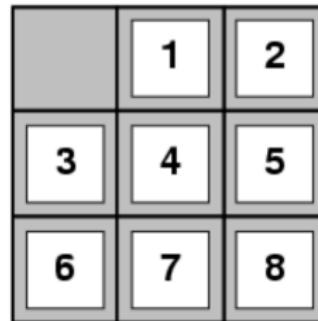
Goal State

- states?
- actions?
- goal test?
- path cost?

# Example: The 8-puzzle



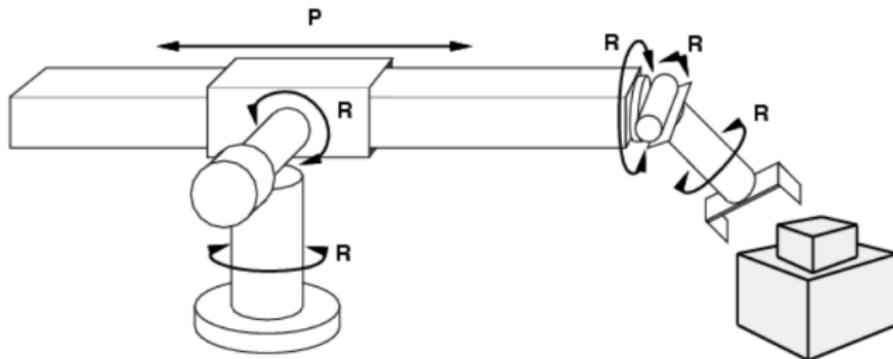
Start State



Goal State

- states? locations of tiles
- actions? move blank left, right, up, down
- goal test? = goal state (given)
- path cost? 1 per move

# Example: robotic assembly



- **states?**: real-valued coordinates of robot joint angles & parts of the object to be assembled
- **actions?**: continuous motions of robot joints
- **goal test?**: complete assembly
- **path cost?**: time to execute

# Tree search algorithms

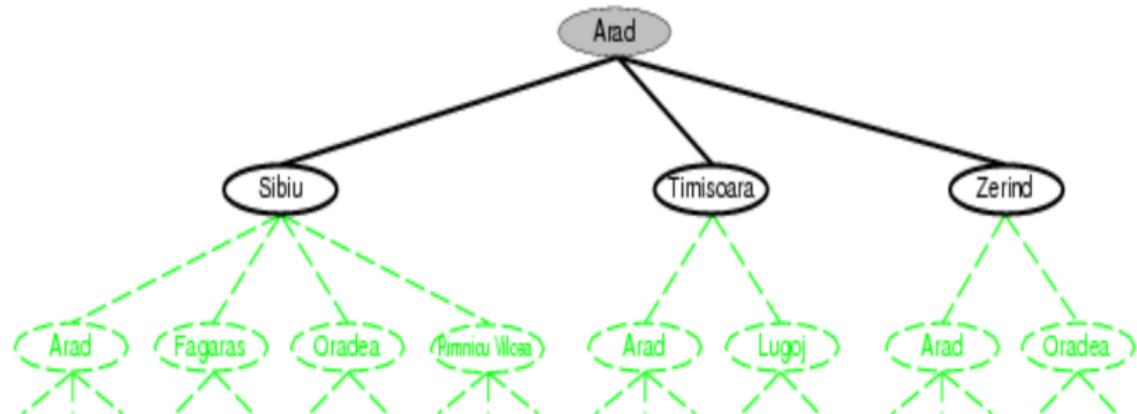
- Basic idea:
  - offline, simulated exploration of state space by generating successors of already-explored states (a.k.a. expanding states)

```
function TREE-SEARCH(problem) returns a solution, or failure
    initialize the frontier using the initial state of problem
    loop do
        if the frontier is empty then return failure
        choose a leaf node and remove it from the frontier
        if the node contains a goal state then return the corresponding solution
        expand the chosen node, adding the resulting nodes to the frontier
```

# Tree search example



# Tree search example



# Tree search example



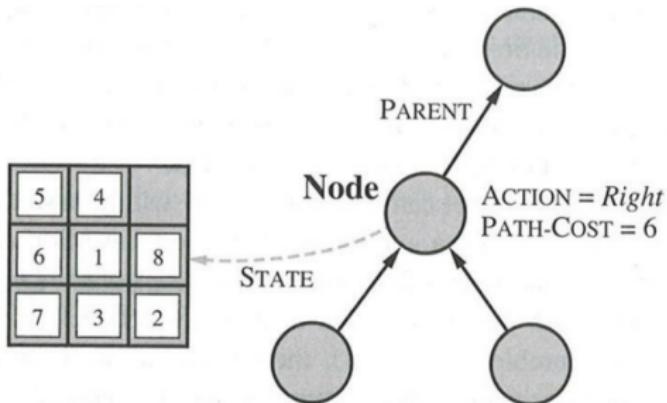
# Implementation: general tree search

```
function TREE-SEARCH(problem) returns a solution, or failure
    initialize the frontier using the initial state of problem
    loop do
        if the frontier is empty then return failure
        choose a leaf node and remove it from the frontier
        if the node contains a goal state then return the corresponding solution
        expand the chosen node, adding the resulting nodes to the frontier
```

```
function CHILD-NODE(problem, parent, action) returns a node
    return a node with
        STATE = problem.RESULT(parent.STATE, action),
        PARENT = parent, ACTION = action,
        PATH-COST = parent.PATH-COST + problem.STEP-COST(parent.STATE, action)
```

# Implementation: states vs. nodes

- A state is a (representation of) a physical configuration
- A node is a book-keeping data structure constituting part of a **search tree** includes state, parent node, action, path cost



- Using these it is easy to compute the components for a child node. (The CHILD-NODE function)

# Summary

- Problem formulation usually requires abstracting away real-world details to define a state space that can feasibly be explored.

# Inf2D 03: Search Strategies

Michael Herrmann

University of Edinburgh, School of Informatics

20/01/2017

**informatics**



Credits: The content of this lecture was prepared by Michael Rovatsos and follows R&N

# Outline

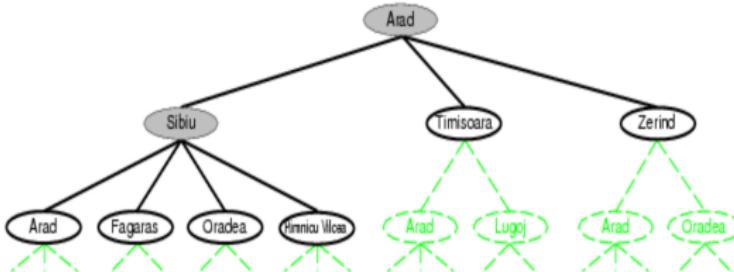
- Uninformed search strategies use only information in problem definition
- Breadth-first search
- Depth-first search
- Depth-limited and iterative deepening search

# Search strategies

- A **search strategy** is defined by picking the order of node expansion – nodes are taken from the *frontier*
- Strategies are evaluated along the following dimensions:
  - **completeness**: does it always find a solution if one exists?
  - **time complexity**: number of nodes generated
  - **space complexity**: maximum number of nodes in memory
  - **optimality**: does it always find a least-cost solution?
- Time and space complexity are measured in terms of
  - $b$ : maximum branching factor of the search tree
  - $d$ : depth of the least-cost solution
  - $m$ : maximum depth of the state space (may be  $\infty$ )

# Recall: Tree Search

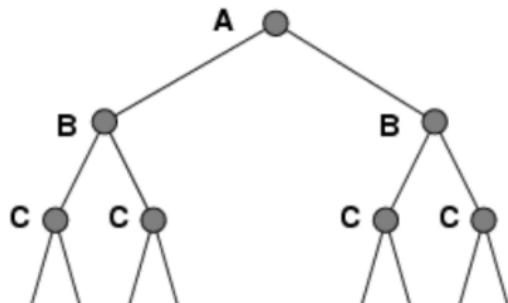
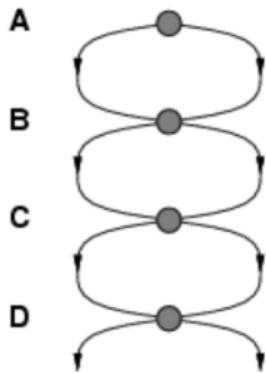
```
function TREE-SEARCH(problem) returns a solution, or failure
    initialize the frontier using the initial state of problem
    loop do
        if the frontier is empty then return failure
        choose a leaf node and remove it from the frontier
        if the node contains a goal state then return the corresponding solution
        expand the chosen node, adding the resulting nodes to the frontier
```



“Arad” is a repeated state!

# Repeated states

- Failure to detect repeated states can turn a **linear** problem into an **exponential** one!



# Graph search

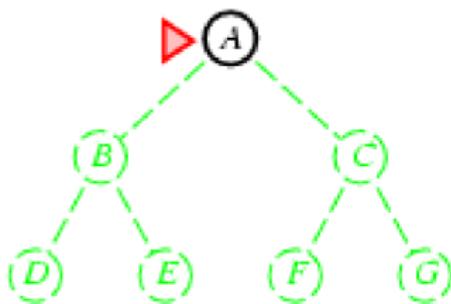
```
function GRAPH-SEARCH(problem) returns a solution, or failure
    initialize the frontier using the initial state of problem
    initialize the explored set to be empty
    loop do
        if the frontier is empty then return failure
        choose a leaf node and remove it from the frontier
        if the node contains a goal state then return the corresponding solution
        add the node to the explored set
        expand the chosen node, adding the resulting nodes to the frontier
        only if not in the frontier or explored set
```

Augment TREE-SEARCH with a new data-structure:

- the explored set (closed list), which remembers every expanded node
- newly expanded nodes already in explored set are discarded

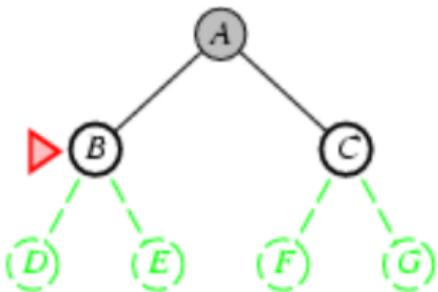
# Breadth-first search

- Expand shallowest unexpanded node
- Implementation:
  - frontier is a FIFO queue, i.e., new successors go at end



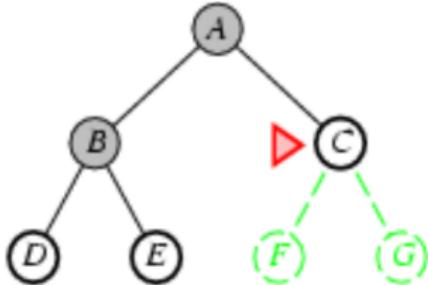
# Breadth-first search

- Expand shallowest unexpanded node
- Implementation:
  - frontier is a FIFO queue, i.e., new successors go at end



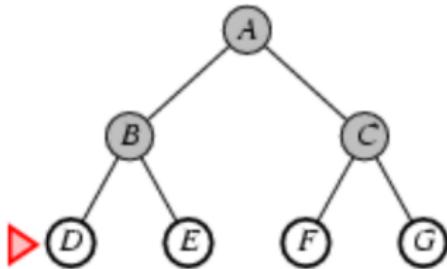
# Breadth-first search

- Expand shallowest unexpanded node
- Implementation:
  - frontier is a FIFO queue, i.e., new successors go at end



# Breadth-first search

- Expand shallowest unexpanded node
- Implementation:
  - frontier is a FIFO queue, i.e., new successors go at end



# Breadth-first search algorithm

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
    node  $\leftarrow$  a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
    if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
    frontier  $\leftarrow$  a FIFO queue with node as the only element
    explored  $\leftarrow$  an empty set
    loop do
        if EMPTY?(frontier) then return failure
        node  $\leftarrow$  POP(frontier) /* chooses the shallowest node in frontier */
        add node.STATE to explored
        for each action in problem.ACTIONS(node.STATE) do
            child  $\leftarrow$  CHILD-NODE(problem, node, action)
            if child.STATE is not in explored or frontier then
                if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
                frontier  $\leftarrow$  INSERT(child, frontier)
```

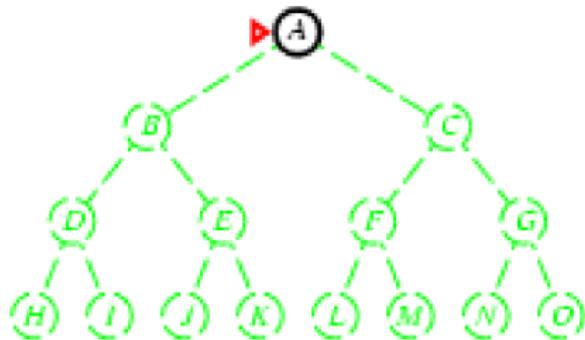
# Properties of breadth-first search

- **Complete?** Yes (if  $b$  is finite)
- **Time?**  $b + b^2 + b^3 + \dots + b^d = O(b^d)$  (worst-case: regular  $b$ -ary tree of depth  $d$ )
- **Space?**  $O(b^d)$  (keeps every node in memory)
- **Optimal?** Yes (if cost = 1 per step, then a *solution* is optimal if it is closest to the start node)

Space is the bigger problem (more than time)

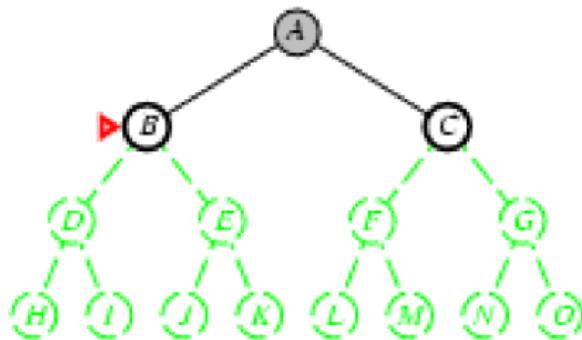
# Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - frontier = LIFO queue, i.e., put successors at front



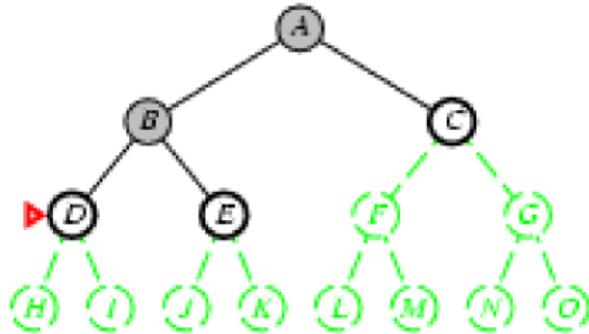
# Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - frontier = LIFO queue, i.e., put successors at front



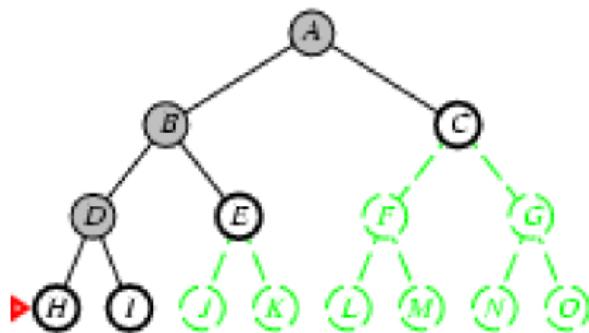
# Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - frontier = LIFO queue, i.e., put successors at front



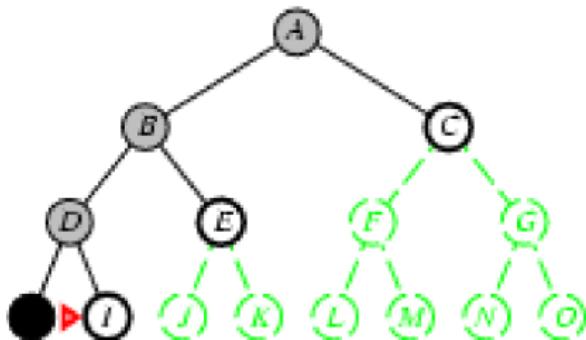
# Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - frontier = LIFO queue, i.e., put successors at front



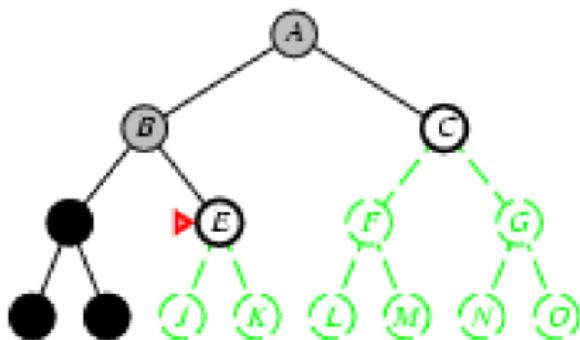
# Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - frontier = LIFO queue, i.e., put successors at front



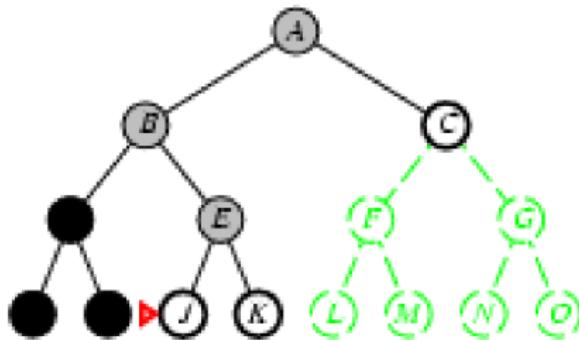
# Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - frontier = LIFO queue, i.e., put successors at front



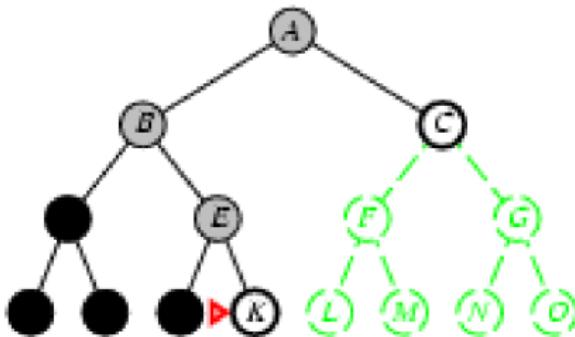
# Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - frontier = LIFO queue, i.e., put successors at front



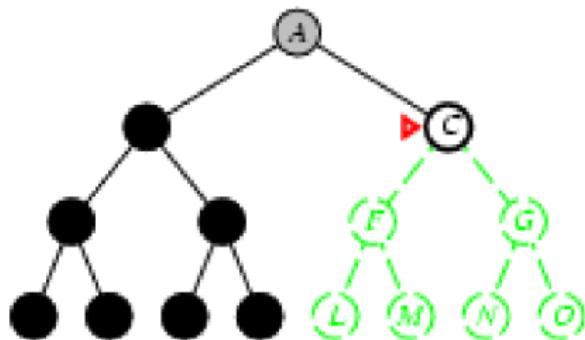
# Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - frontier = LIFO queue, i.e., put successors at front



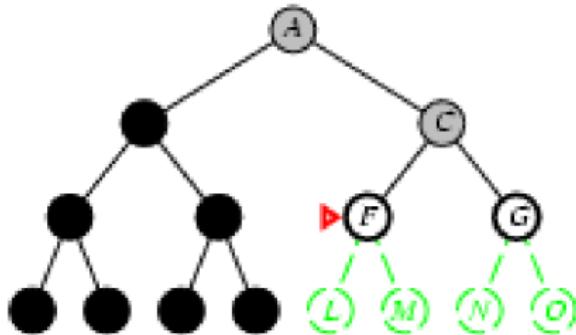
# Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - frontier = LIFO queue, i.e., put successors at front



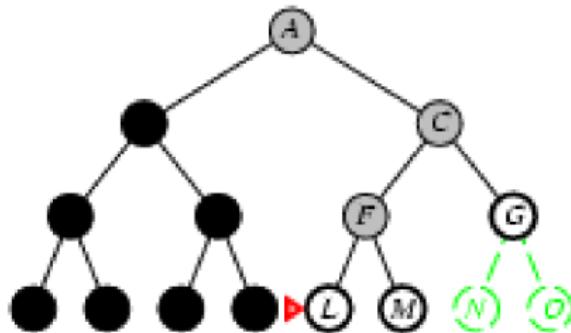
# Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - frontier = LIFO queue, i.e., put successors at front



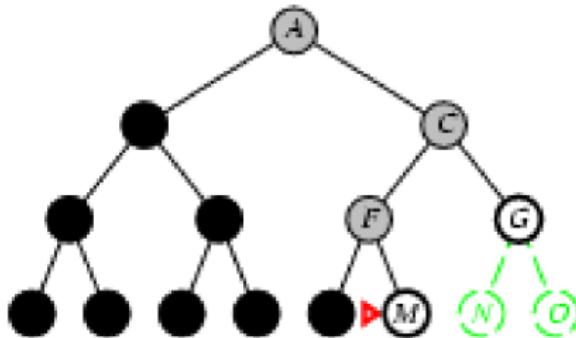
# Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - frontier = LIFO queue, i.e., put successors at front



# Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - frontier = LIFO queue, i.e., put successors at front



# Properties of depth-first search

- **Complete?** No: fails in infinite-depth spaces, spaces with loops
  - Modify to avoid repeated states along path
  - Complete in finite spaces
- **Time?**  $O(b^m)$ : terrible if  $m$  is much larger than  $d$ 
  - but if solutions are dense, may be much faster than breadth-first
- **Space?**  $O(bm)$ , i.e., linear space!
- **Optimal?** No

## Mid-Lecture Problem

- Compare breadth-first and depth-first search.
  - When would breadth-first be preferable?
  - When would depth-first be preferable?

# Solution

- Breadth-First:
  - When completeness is important.
  - When optimal solutions are important.
- Depth-First:
  - When solutions are dense and low-cost is important, especially space costs.

# Depth-limited search

This is depth-first search with **depth limit  $l$** , i.e., nodes at depth  $l$  have no successors

Recursive implementation:

```
function DEPTH-LIMITED-SEARCH(problem, limit) returns a solution, or failure/cutoff
    return RECURSIVE-DLS(MAKE-NODE(problem.INITIAL-STATE), problem, limit)

function RECURSIVE-DLS(node, problem, limit) returns a solution, or failure/cutoff
    if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
    else if limit = 0 then return cutoff
    else
        cutoff_occurred? ← false
        for each action in problem.ACTIONS(node.STATE) do
            child ← CHILD-NODE(problem, node, action)
            result ← RECURSIVE-DLS(child, problem, limit - 1)
            if result = cutoff then cutoff_occurred? ← true
            else if result ≠ failure then return result
        if cutoff_occurred? then return cutoff else return failure
```

# Iterative deepening search

```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution, or failure
  for depth = 0 to  $\infty$  do
    result  $\leftarrow$  DEPTH-LIMITED-SEARCH(problem, depth)
    if result  $\neq$  cutoff then return result
```

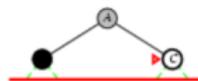
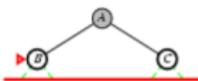
# Iterative deepening search / = 0

Limit = 0



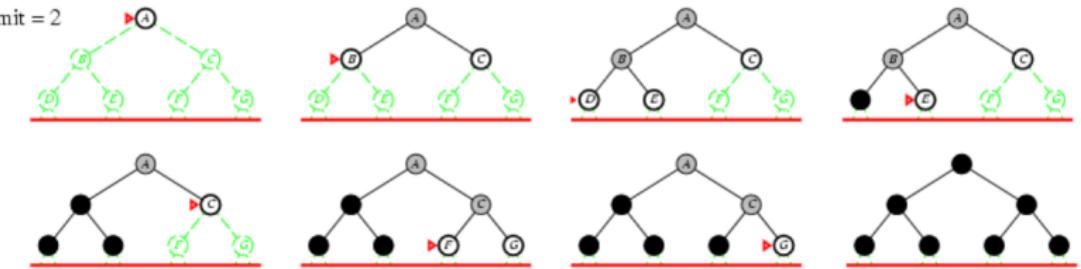
# Iterative deepening search / = 0

Limit = 1



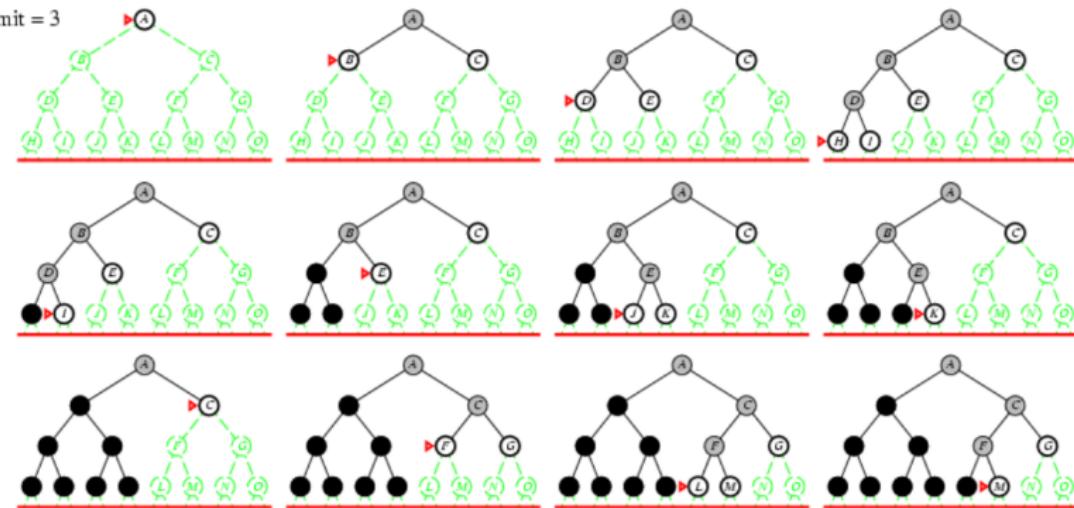
# Iterative deepening search / = 0

Limit = 2



# Iterative deepening search / = 0

Limit = 3



# Iterative deepening search

- Number of nodes generated in an iterative deepening search to depth  $d$  with branching factor  $b$ :

$$N_{\text{IDS}} = (d)b + (d - 1)b^2 + \cdots + (2)b^{d-1} + (1)b^d$$

- Some cost associated with generating upper levels multiple times
- Example: For  $b = 10$ ,  $d = 5$ ,
  - $N_{\text{BFS}} = 10 + 100 + 3,000 + 10,000 + 100,000 = 111,110$
  - $N_{\text{IDS}} = 50 + 400 + 3,000 + 20,000 + 100,000 = 123,450$
- Overhead =  $(123,450 - 111,110)/111,110 = 11\%$

# Properties of iterative deepening search

- Complete? Yes
- Time?  $(d)b + (d - 1)b^2 + \dots + (1)b^d = O(b^d)$
- Space?  $O(bd)$
- Optimal? Yes, if step cost = 1

# Uniform cost search (UCS)

Step costs are not uniform.

Details: home work.

# Summary of algorithms

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

# Summary

- Variety of *uninformed* search strategies:
  - breadth-first
  - depth-first
  - depth limited
  - iterative deepening
- Iterative deepening search uses only linear space and not much more time than other uninformed algorithms

# Inf2D 04: Adversarial Search

Michael Herrmann

University of Edinburgh, School of Informatics

24/01/2017

**informatics**



Credits: The content of this lecture was prepared by Michael Rovatsos and follows R&N

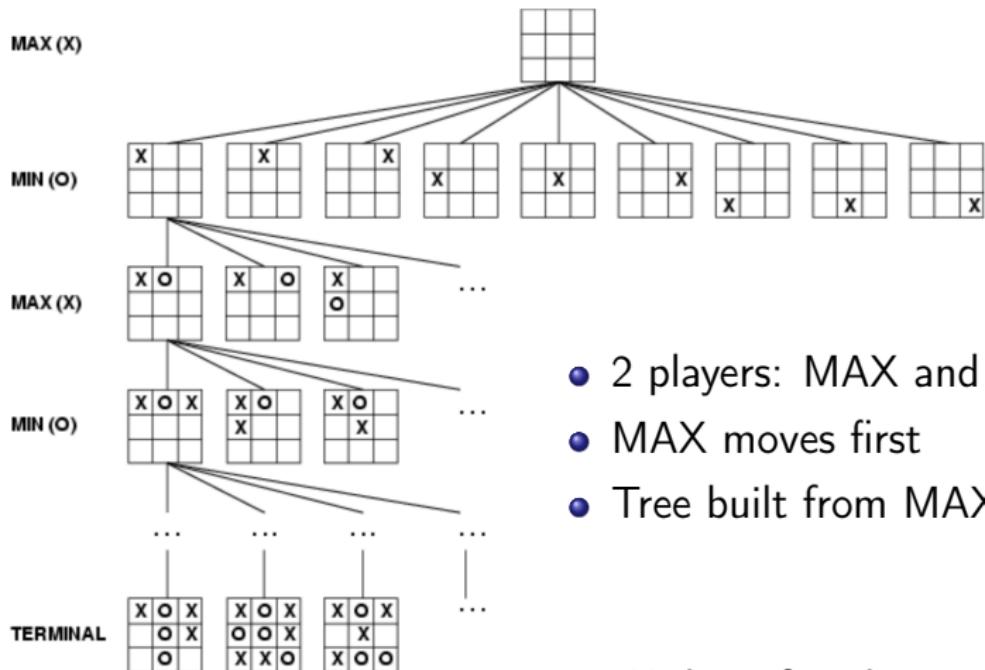
# Outline

- Games
- Optimal decisions
- $\alpha$ - $\beta$  pruning
- Imperfect, real-time decisions

# Games vs. search problems

- We are (usually) interested in zero-sum games of perfect information
  - Deterministic, fully observable
  - Agents act alternately
  - Utilities at end of game are equal and opposite
- “Unpredictable” opponent → specifying a move for every possible opponent reply
- Time limits → unlikely to find goal, must approximate

# Game tree (2-player, deterministic, turns)



- 2 players: MAX and MIN
- MAX moves first
- Tree built from MAX's POV

← Utility of each terminal state from MAX's point of view.

# Optimal Decisions

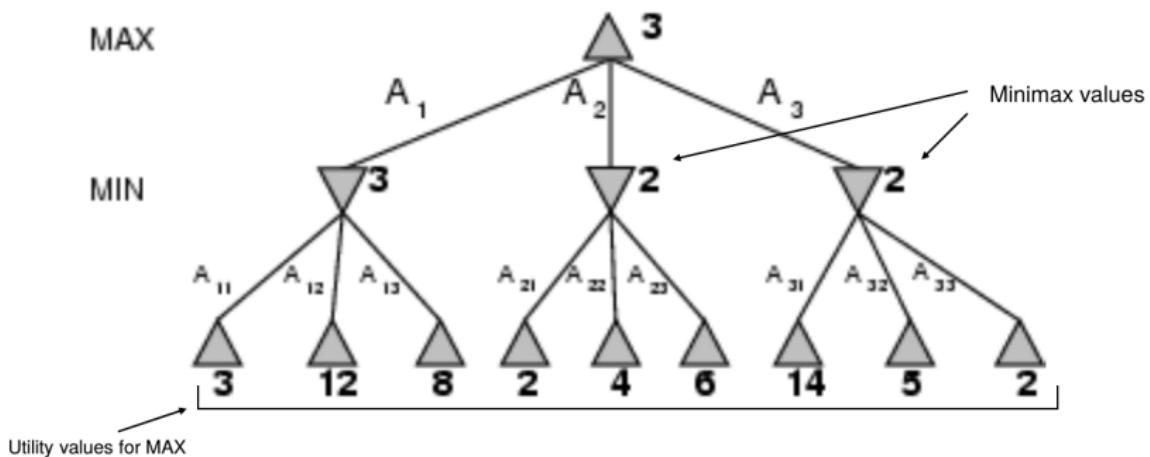
- Normal search: optimal decision is a sequence of actions leading to a goal state (i.e. a winning terminal state)
- Adversarial search:
  - MIN has a say in game
  - MAX needs to find a contingent strategy which specifies:
    - MAX's move in initial state then ...
    - MAX's moves in states resulting from every response by MIN to the move then ...
    - MAX's moves in states resulting from every response by MIN to all those moves, etc. ...

minimax value of a node=utility for MAX of being in corresponding state:  
 $\text{MINIMAX}(s) =$

$$\begin{cases} \text{UTILITY}(s) & \text{if } \text{TERMINAL-TEST}(s) \\ \max_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MAX} \\ \min_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MIN} \end{cases}$$

# Minimax

- Perfect play for deterministic games
- Idea: choose move to position with highest minimax value  
= best achievable payoff against best play
- Example: 2-ply game:



# Minimax algorithm

```
function MINIMAX-DECISION(state) returns an action
    return arg maxa ∈ ACTIONS(s) MIN-VALUE(RESULT(s, a))
```

```
function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(s) then return UTILITY(s)
    v ← −∞
    for each a in ACTIONS(s) do
        v ← MAX(v, MIN-VALUE(RESULT(s, a)))
    return v
```

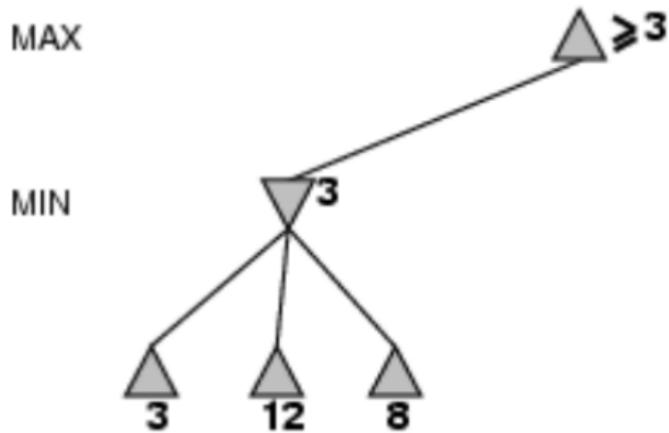
```
function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(s) then return UTILITY(s)
    v ← ∞
    for each a in ACTIONS(s) do
        v ← MIN(v, MAX-VALUE(RESULT(s, a)))
    return v
```

Idea: Proceed all the way down to the leaves of the tree then  
minimax values are backed up through tree

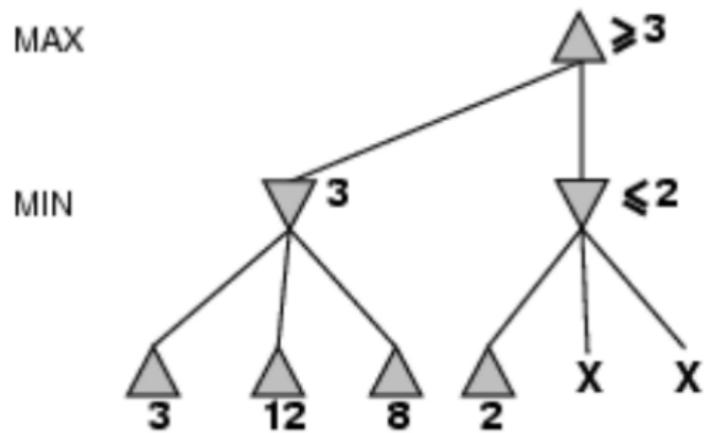
# Properties of minimax

- **Complete?** Yes (if tree is finite)
- **Optimal?** Yes (against an optimal opponent)
- **Time complexity?**  $O(b^m)$
- **Space complexity?**  $O(bm)$  (depth-first exploration)
- For chess,  $b \approx 35$ ,  $m \approx 100$  for “reasonable” games
  - exact solution completely infeasible!
  - would like to eliminate (large) parts of game tree

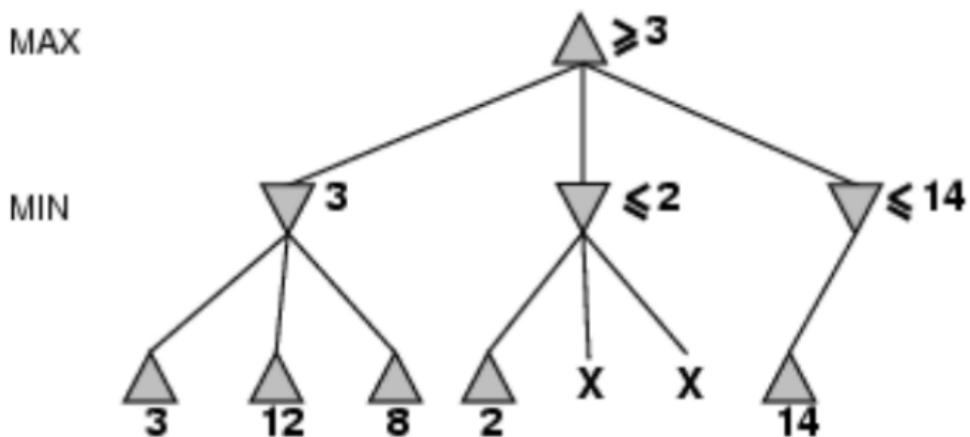
# $\alpha$ - $\beta$ pruning example



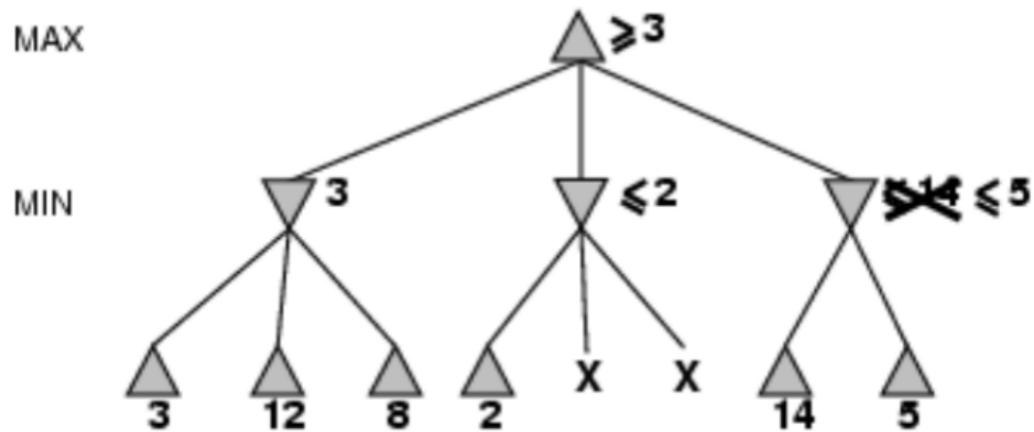
# $\alpha$ - $\beta$ pruning example



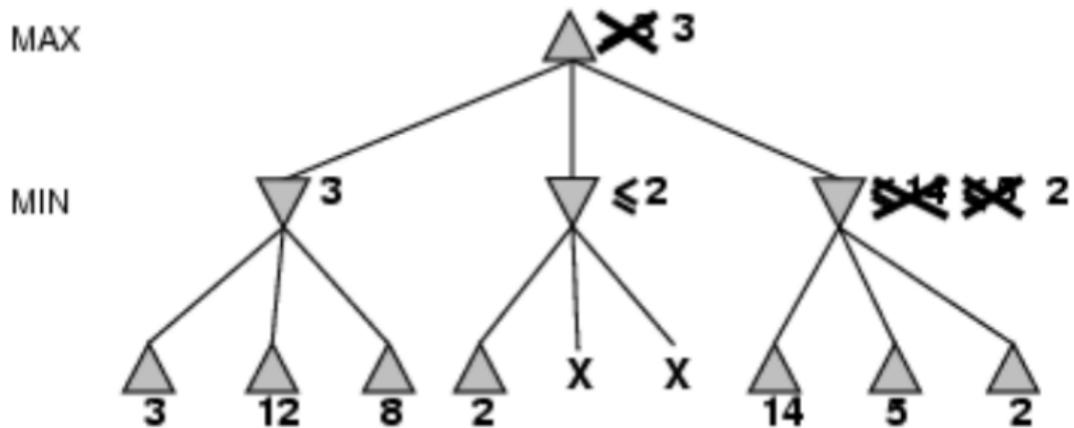
# $\alpha$ - $\beta$ pruning example



# $\alpha$ - $\beta$ pruning example



# $\alpha$ - $\beta$ pruning example



## $\alpha$ - $\beta$ pruning example

- Are minimax value of root and, hence, minimax decision independent of pruned leaves?
- Let pruned leaves have values  $u$  and  $v$ , then

$$\begin{aligned} \text{MINIMAX}(\text{root}) &= \max(\min(3, 12, 8), \min(2, u, v), \min(14, 5, 2)) \\ &= \max(3, \min(2, u, v), 2) \\ &= \max(3, z, 2) \text{ where } z \leq 2 \\ &= 3 \end{aligned}$$

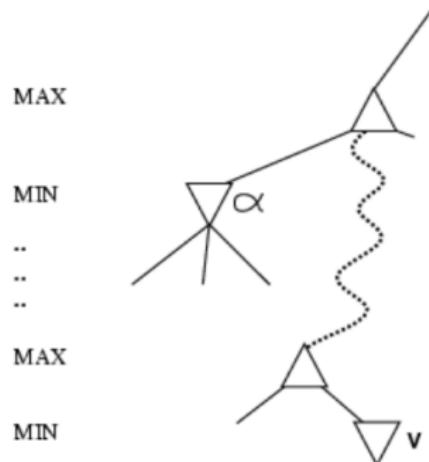
- Yes!

# Properties of $\alpha$ - $\beta$

- Pruning does not affect final result (as we saw for example)
- Good move ordering improves effectiveness of pruning (How could previous tree be better?)
- With “perfect ordering”, time complexity  $O(b^{m/2})$ 
  - branching factor goes from  $b$  to  $\sqrt{b}$
  - (alternative view) doubles depth of search compared to minimax
- A simple example of the value of reasoning about which computations are relevant (a form of meta-reasoning)

# Why is it called $\alpha$ - $\beta$ ?

- $\alpha$  is the value of the best (i.e., highest-value) choice found so far at any choice point along the path for MAX
- If  $v$  is worse than  $\alpha$ , MAX will avoid it  
→ prune that branch
- Define  $\beta$  similarly for MIN



# The $\alpha$ - $\beta$ algorithm

```
function ALPHA-BETA-SEARCH(state) returns an action
    v ← MAX-VALUE(state, -∞, +∞)
    return the action in ACTIONS(state) with value v
```

```
function MAX-VALUE(state,  $\alpha$ ,  $\beta$ ) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← -∞
    for each a in ACTIONS(state) do
        v ← MAX(v, MIN-VALUE(RESULT(s, a),  $\alpha$ ,  $\beta$ ))
        if  $v \geq \beta$  then return v
         $\alpha \leftarrow \text{MAX}(\alpha, v)$ 
    return v
```

Prune as this value is worse for MIN and so won't ever be chosen by MIN!

- $\alpha$  is value of the best i.e. highest-value choice found so far at any choice point along the path for MAX
- $\beta$  is value of the best i.e. lowest-value choice found so far at any choice point along the path for MIN

# The $\alpha$ - $\beta$ algorithm

```
function MIN-VALUE(state,  $\alpha$ ,  $\beta$ ) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
   $v \leftarrow +\infty$ 
  for each a in ACTIONS(state) do
     $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(s, a), \alpha, \beta))$ 
    if  $v \leq \alpha$  then return  $v$  ←
     $\beta \leftarrow \text{MIN}(\beta, v)$ 
  return v
```

Prune as this value is worse for MAX and so won't ever be chosen by MAX!

# Resource limits

- Suppose we have 100 secs, explore  $10^4$  nodes/sec  
→  $10^6$  nodes per move
- Standard approach:
  - cutoff test: e.g., depth limit (perhaps add quiescence search, which tries to search interesting positions to a greater depth than quiet ones)
  - evaluation function  
= estimated desirability of position

# Evaluation functions

- For chess, typically linear weighted sum of **features**

$$EVAL(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s)$$

where each  $w_i$  is a weight and each  $f_i$  is a feature of state  $s$

- Example
  - queen = 1, king = 2, etc.
  - $f_i$ : number of pieces of type  $i$  on board
  - $w_i$ : value of the piece of type  $i$

# Cutting off search

- Minimax Cutoff is identical to MinimaxValue except
  - ① TERMINAL-TEST is replaced by CUTOFF
  - ② UTILITY is replaced by EVAL
- Does it work in practice?  
 $b^m = 10^6$ ,  $b = 35 \rightarrow m = 4$
- 4-ply lookahead is a hopeless chess player!
  - 4-ply  $\approx$  human novice
  - 8-ply  $\approx$  typical PC, human master
  - 12-ply  $\approx$  Deep Blue, Kasparov

# Deterministic games in practice

- Checkers: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used a precomputed endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 444 billion positions.
- Chess: Deep Blue defeated human world champion Garry Kasparov in a six-game match in 1997. Deep Blue searches 200 million positions per second, uses very sophisticated evaluation, and undisclosed methods for extending some lines of search up to 40 ply.
- Othello: human champions refuse to compete against computers, who are too good.
- Go: human champions used to refuse to compete against computers, who are too bad. In Go,  $b > 300$ , so most programs use pattern knowledge bases to suggest plausible moves. 2016: AlphaGo

# Summary

- Games are fun to work on!
- They illustrate several important points about AI
- Perfection is unattainable → must approximate
- Good idea to think about what to think about

# Inf2D 05: Informed Search and Exploration for Agents

Michael Herrmann

University of Edinburgh, School of Informatics

26/01/2017

**informatics**



Credits: The content of this lecture was prepared by Michael Rovatsos and follows R&N

# Outline

- Best-first search
- Greedy best-first search
- $A^*$  search
- Heuristics
- Admissibility

# Review: Tree search

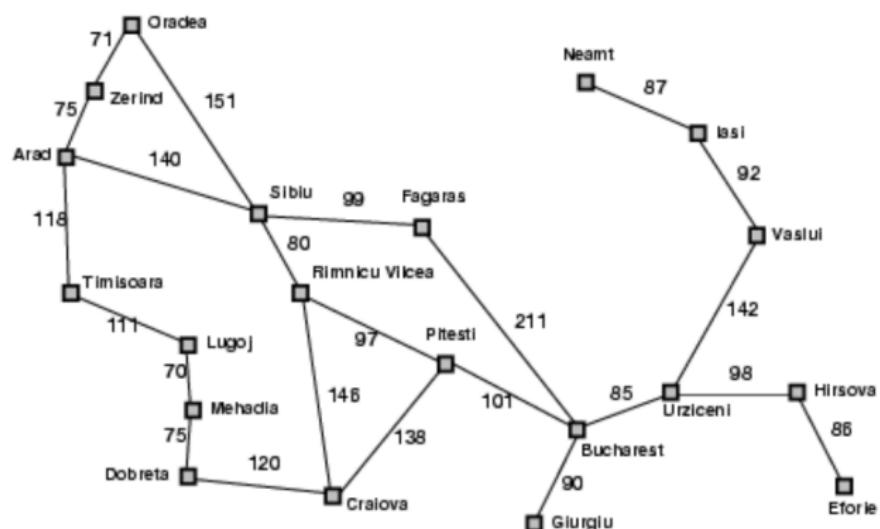
```
function TREE-SEARCH(problem) returns a solution, or failure
    initialize the frontier using the initial state of problem
    loop do
        if the frontier is empty then return failure
        choose a leaf node and remove it from the frontier
        if the node contains a goal state then return the corresponding solution
        expand the chosen node, adding the resulting nodes to the frontier
```

A search strategy is defined by picking the order of node expansion from the frontier

# Best-first search

- An instance of general TREE-SEARCH or GRAPH-SEARCH
- Idea: use an evaluation function  $f(n)$  for each node  $n$ 
  - estimate of “desirability”  
→ Expand most desirable unexpanded node, usually the node with the lowest evaluation
- Implementation:  
Order the nodes in frontier in decreasing order of desirability
- Special cases:
  - Greedy best-first search
  - A\*search

# Romania with step costs in km



Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

# Greedy best-first search

- Evaluation function  $f(n) = h(n)$  (heuristic)
- $h(n)$ : **estimated cost** of cheapest path from state at node  $n$  to a goal state
  - e.g.,  $h_{SLD}(n)$ : straight-line distance from  $n$  to goal (Bucharest)
  - Greedy best-first search expands the node that **appears** to be closest to goal

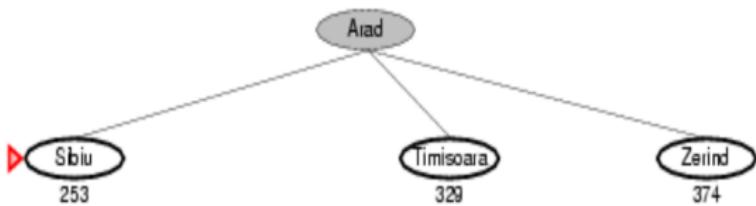
## What is a heuristic?

- From the greek word “*heuriskein*” meaning “*to discover*” or “*to find*”
- A heuristic is any method that is believed or practically proven to be useful for the solution of a given problem, although there is no guarantee that it will always work or lead to an optimal solution.
- Here we will use heuristics to guide tree search. This may not change the worst case complexity of the algorithm, but can help in the average case.
- We will introduce conditions (admissibility, consistency, see below) in order to identify good heuristics, i.e. those which actually lead to an improvement over uninformed search.
- See also: <https://en.wikipedia.org/wiki/Heuristic>

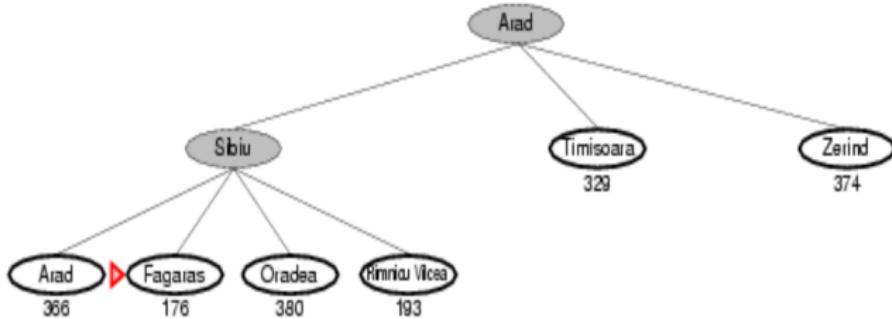
# Greedy best-first search example



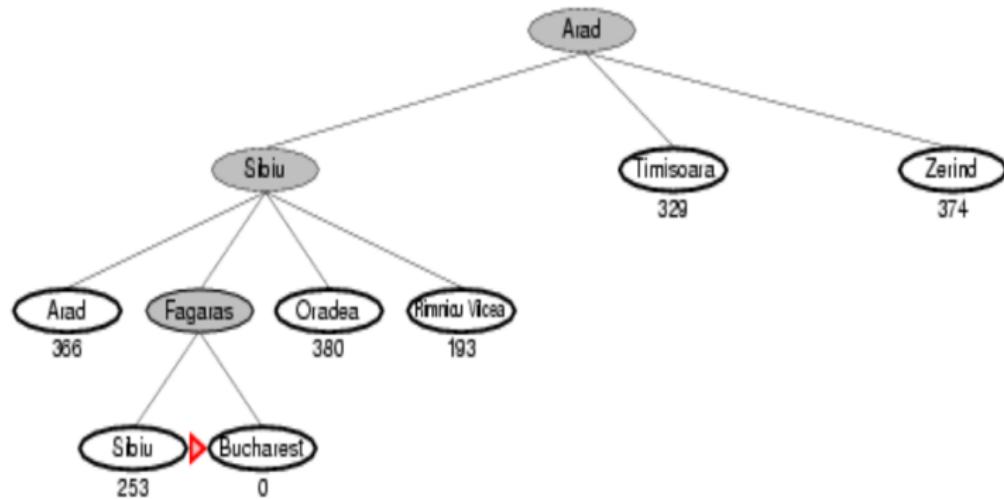
# Greedy best-first search example



# Greedy best-first search example



# Greedy best-first search example



# Properties of greedy best-first search

- **Complete?** No – can get stuck in loops
  - Graph search version is complete in finite space, but not in infinite ones
- **Time?**  $O(b^m)$  for tree version, but a good heuristic can give dramatic improvement
- **Space?**  $O(b^m)$  – keeps all nodes in memory
- **Optimal?** No

# $A^*$ search

- Idea: avoid expanding paths that are already expensive
- Evaluation function  $f(n) = g(n) + h(n)$ 
  - $g(n)$ : cost so far to reach  $n$
  - $h(n)$ : estimated cost from  $n$  to goal
  - $f(n)$ : estimated total cost of path through  $n$  to goal
- $A^*$  is both complete and optimal if  $h(n)$  satisfies certain conditions

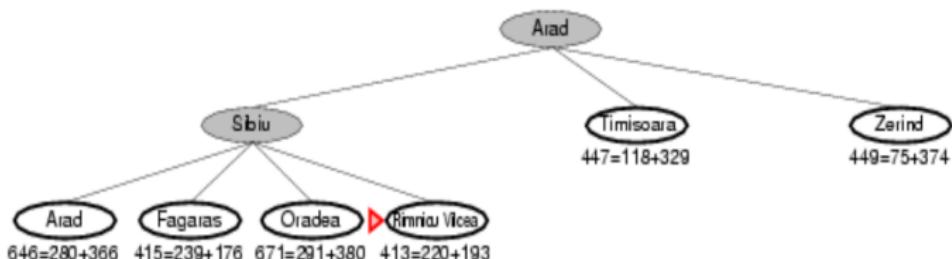
# $A^*$ search example



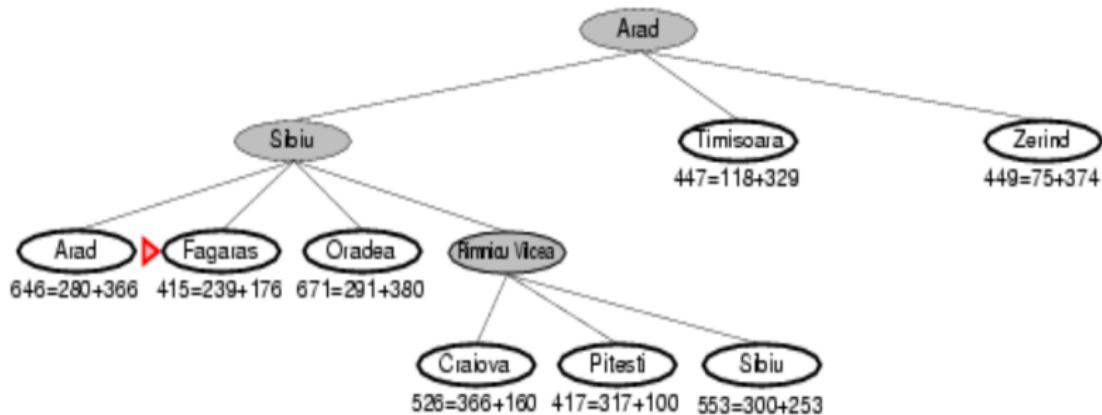
# $A^*$ search example



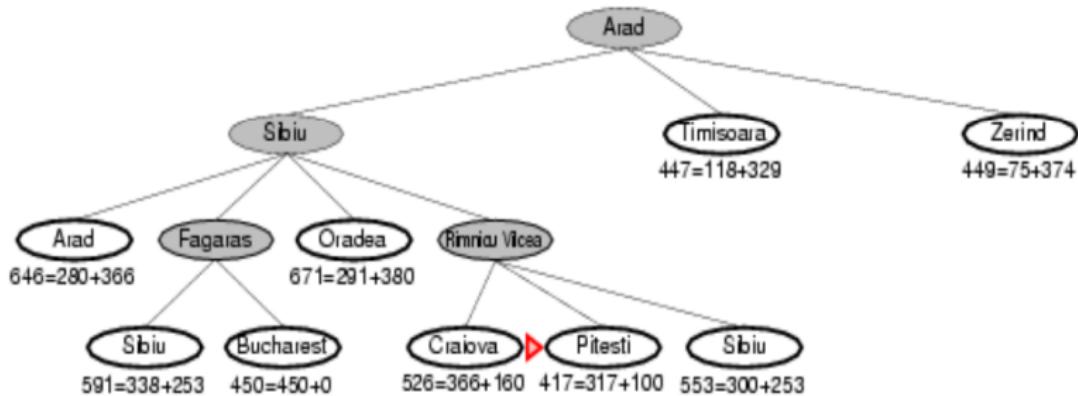
# $A^*$ search example



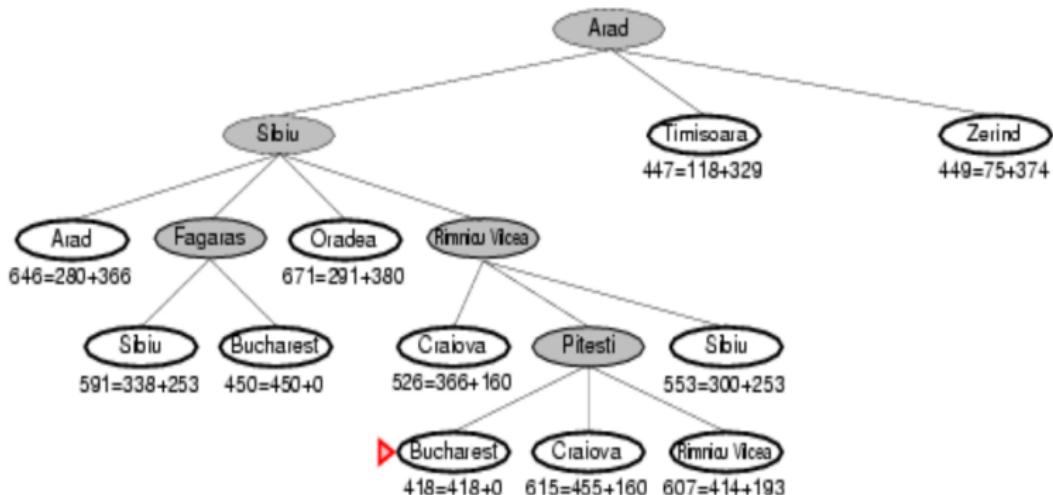
# $A^*$ search example



# $A^*$ search example



# $A^*$ search example

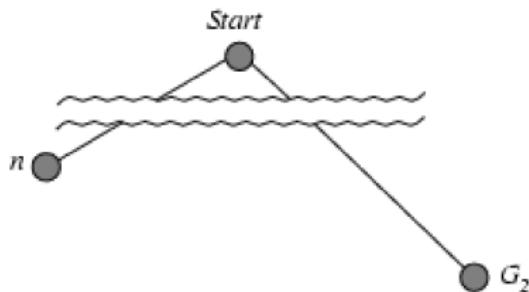


# Admissible heuristics

- A heuristic  $h(n)$  is admissible if for every node  $n$ ,  $h(n) \leq h^*(n)$ , where  $h^*(n)$  is the **true** cost to reach the goal state from  $n$ .
- An admissible heuristic never overestimates the cost to reach the goal, i.e., it is optimistic
  - Thus,  $f(n) = g(n) + h(n)$  **never** overestimates the true cost of a solution
- Example:  $h_{SLD}(n)$  (never overestimates the actual road distance)
- **Theorem:** If  $h(n)$  is admissible,  $A^*$  using TREE-SEARCH is **optimal**.

# Optimality of $A^*$ (proof)

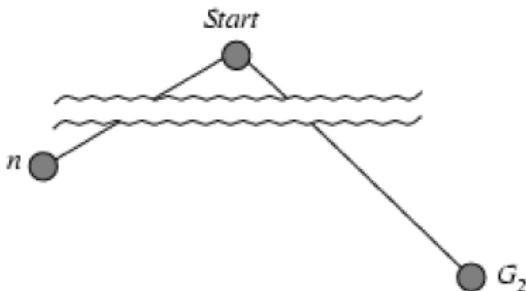
- Suppose some **suboptimal** goal  $G_2$  has been generated and is in the frontier. Let  $n$  be an unexpanded node in the frontier such that  $n$  is on a **shortest path** to an **optimal** goal  $G$ .



- $f(G_2) = g(G_2)$  since  $h(G_2) = 0$
- $g(G_2) > g(G)$  since  $G_2$  is suboptimal
- $f(G) = g(G)$  since  $h(G) = 0$
- $f(G_2) > f(G)$  from above

# Optimality of $A^*$ (proof cntd.)

- Suppose some **suboptimal** goal  $G_2$  has been generated and is in the **fringe**. Let  $n$  be an unexpanded node in the fringe such that  $n$  is on a **shortest path** to an **optimal** goal  $G$ .



- $f(G) < f(G_2)$  from above ( $G_2$  is suboptimal)
- $h(n) \leq h^*(n)$  since  $h$  is admissible
- $g(n) + h(n) \leq g(n) + h^*(n) = f(G)$
- $f(n) \leq f(G)$

Hence  $f(n) < f(G_2) \Rightarrow A^*$  will never select  $G_2$  for expansion.

# Consistent heuristics

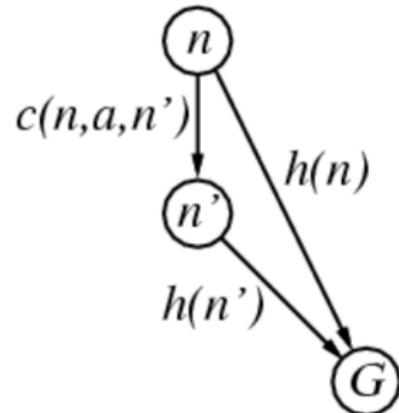
- A heuristic is **consistent** if for every node  $n$ , every successor  $n'$  of  $n$  generated by any action  $a$ ,

$$h(n) \leq c(n, a, n') + h(n')$$

- If  $h$  is consistent, we have

$$\begin{aligned}f(n') &= g(n') + h(n') \\&= g(n) + c(n, a, n') + h(n') \\&\geq g(n) + h(n) \\&\geq f(n)\end{aligned}$$

- i.e.,  $f(n)$  is non-decreasing along any path.

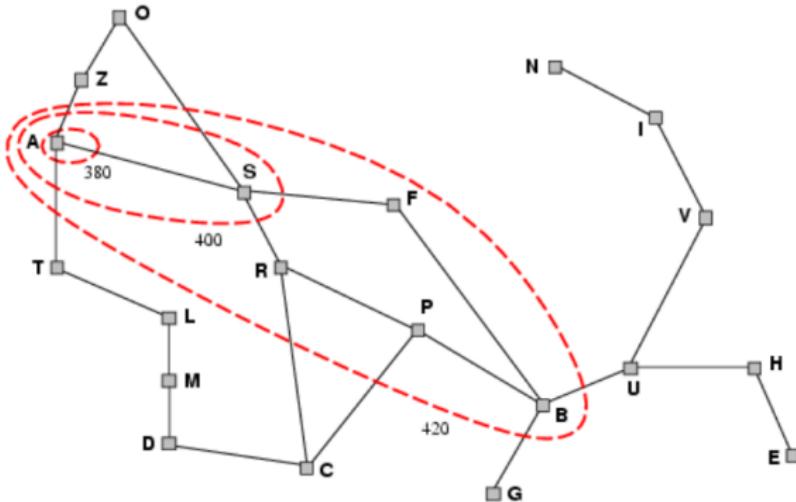


**Theorem:**

If  $h(n)$  is consistent,  $A^*$  using GRAPH-SEARCH is optimal.

# Optimality of $A^*$

- $A^*$  expands nodes in order of increasing  $f$  value
- Gradually adds “ $f$ -contours” of nodes
- Contour  $i$  has all nodes with  $f = f_i$ , where  $f_i < f_{i+1}$



# Properties of $A^*$

- **Complete?** Yes (unless there are infinitely many nodes with  $f \leq f(G)$ )
- **Time?** Exponential
- **Space?** Keeps all nodes in memory
- **Optimal?** Yes

# Admissible heuristics

Example:

- for the 8-puzzle:
  - $h_1(n)$ : number of misplaced tiles
  - $h_2(n)$ : total Manhattan distance

(i.e., no. of squares from desired location of each tile)

Exercise: Calculate  
these two values:

- $h_1(S) = ?$
- $h_2(S) = ?$

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

# Dominance

- If  $h_2(n) \geq h_1(n)$  for all  $n$  (both admissible) then
  - $h_2$  dominates  $h_1$
  - $h_2$  is better for search
- Typical search costs (average number of nodes expanded):
  - $d = 12$       IDS = 3,644,035 nodes  
 $A^*(h_1) = 227$  nodes  
 $A^*(h_2) = 73$  nodes
  - $d = 24$       IDS = too many nodes  
 $A^*(h_1) = 39,135$  nodes  
 $A^*(h_2) = 1,641$  nodes

# Relaxed problems

- A problem with fewer restrictions on the actions is called a **relaxed problem**
- The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem
- If the rules of the 8-puzzle are relaxed so that a tile can move anywhere,
  - then  $h_1(n)$  gives the shortest solution
- If the rules are relaxed so that a tile can move to any adjacent square,
  - then  $h_2(n)$  gives the shortest solution
- Can use relaxation to automatically generate admissible heuristics

# Summary

Smart search based on **heuristic scores**.

- Best-first search
- Greedy best-first search
- $A^*$  search
- Admissible heuristics and optimality.

# Inf2D 07: Effective Propositional Inference

Michael Herrmann

University of Edinburgh, School of Informatics

31/01/2017

**informatics**



Credits: The content of this lecture was prepared by Michael Rovatsos and follows R&N

# Outline

Two families of efficient algorithms for propositional inference:

- Complete backtracking search algorithms
  - DPLL algorithm (Davis, Putnam, Logemann, Loveland)
- Incomplete local search algorithms
  - WalkSAT algorithm

# Clausal Form: CNF

- DPLL and WalkSAT manipulate formulae in conjunctive normal form (CNF).
- Sentence is formula whose satisfiability is to be determined.
  - conjunction of clauses.
- Clause is disjunction of literals
- Literal is proposition or negated proposition
- Example:  $(A, \neg B), (B, \neg C)$  representing  $(A \vee \neg B) \wedge (B \vee \neg C)$

# Conversion to CNF

$$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$$

- ① Eliminate  $\Leftrightarrow$  replacing  $\alpha \Leftrightarrow \beta$  by  $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$   
 $(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$
- ② Eliminate  $\Rightarrow$ , replacing  $\alpha \Rightarrow \beta$  by  $\neg\alpha \vee \beta$   
 $(\neg B_{1,1} \vee (P_{1,2} \vee P_{2,1})) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1})$
- ③ Move  $\neg$  inwards using de Morgan's rules  
 $(\neg B_{1,1} \vee (P_{1,2} \vee P_{2,1})) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$   
possibly also eliminating double-negation: replacing  $\neg(\neg\alpha)$  by  $\alpha$
- ④ Apply distributivity law ( $\vee$  over  $\wedge$ ) and flatten:  
 $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$

# The DPLL algorithm

Determine if an input propositional logic sentence (in CNF) is **satisfiable**.

**Improvements** over truth table enumeration:

- ① Early termination
- ② Pure symbol heuristic
- ③ Unit clause heuristic

# Early termination

- A clause is true if one of its literals is true,
  - e.g. if  $A$  is true then  $(A \vee \neg B)$  is true.
- A sentence is false if **any** of its clauses is false,
  - e.g. if  $A$  is false and  $B$  is true then  $(A \vee \neg B)$  is false, so sentence containing it is false.

# Pure symbol heuristic

- Pure symbol: always appears with the same “sign” or polarity in all clauses.
  - e.g., in the three clauses  $(A \vee \neg B)$ ,  $(\neg B \vee \neg C)$ ,  $(C \vee A)$   $A$  and  $B$  are pure,  $C$  is impure.
- Make literal containing a pure symbol true.
  - e.g. (for satisfiability) Let  $A$  and  $\neg B$  both be true

# Unit clause heuristic

**Unit clause:** only one literal in the clause, e.g.  $(A)$

- The only literal in a unit clause must be true.
  - e.g.  $A$  must be true.
- Also includes clauses where **all but one** literal is false,
  - e.g.  $(A, B, C)$  where  $B$  and  $C$  are false since it is equivalent to  $(A, \text{false}, \text{false})$  i.e.  $(A)$ .

# The DPLL algorithm

**function** DPLL-SATISFIABLE?(*s*) **returns** true or false

**inputs:** *s*, a sentence in propositional logic

*clauses*  $\leftarrow$  the set of clauses in the CNF representation of *s*

*symbols*  $\leftarrow$  a list of the proposition symbols in *s*

**return** DPLL(*clauses, symbols, []*)

---

**function** DPLL(*clauses, symbols, model*) **returns** true or false

**if** every clause in *clauses* is true in *model* **then return** true

**if** some clause in *clauses* is false in *model* **then return** false

*P, value*  $\leftarrow$  FIND-PURE-SYMBOL(*symbols, clauses, model*)

**if** *P* is non-null **then return** DPLL(*clauses, symbols-P, [P = value | model]*)

*P, value*  $\leftarrow$  FIND-UNIT-CLAUSE(*clauses, model*)

**if** *P* is non-null **then return** DPLL(*clauses, symbols-P, [P = value | model]*)

*P*  $\leftarrow$  FIRST(*symbols*); *rest*  $\leftarrow$  REST(*symbols*)

**return** DPLL(*clauses, rest, [P = true | model]*) **or**

         DPLL(*clauses, rest, [P = false | model]*)

# Tautology Deletion (Optional)

- Tautology: both a proposition and its negation in a clause.
  - e.g.  $(A, B, \neg A)$
- Clause bound to be true.
  - e.g. whether  $A$  is true or false.
  - Therefore, can be deleted.

## Mid-Lecture Exercise

- Apply DPLL heuristics to the following sentence:

$(S_{2,1}), (\neg S_{1,1}), (\neg S_{1,2}),$   
 $(\neg S_{2,1}, W_{2,2}), (\neg S_{1,1}, W_{2,2}), (\neg S_{1,2}, W_{2,2}),$   
 $(\neg W_{2,2}, S_{2,1}, S_{1,1}, S_{1,2}).$

- Use **case splits** if model not found by these heuristics.

# Solution

Symbols:  $S_{1,1}$ ,  $S_{1,2}$ ,  $S_{2,1}$ ,  $W_{2,2}$

- Pure symbol heuristic:
  - No literal is pure.
- Unit clause heuristic:
  - $S_{2,1}$  is true;  $S_{1,1}$  and  $S_{1,2}$  are false.  
 $(S_{2,1}), (\neg S_{1,1}), (\neg S_{1,2}),$   
 $(\neg S_{2,1}, W_{2,2}),$   
 $(\neg S_{1,1}, W_{2,2}), (\neg S_{1,2}, W_{2,2}),$   
 $(\neg W_{2,2}, S_{2,1}, S_{1,1}, S_{1,2}).$
- Early termination heuristic:
  - $(\neg S_{1,1}, W_{2,2}), (\neg S_{1,2}, W_{2,2})$  are both true.
  - $(\neg W_{2,2}, S_{2,1}, S_{1,1}, S_{1,2})$  is true.
- Unit clause heuristic:
  - $\neg S_{2,1}$  is false, so  $(\neg S_{2,1}, W_{2,2})$  becomes unit clause.
  - $W_{2,2}$  must be true.

# The WalkSAT algorithm

- Incomplete, local search algorithm
- Evaluation function: The min-conflict heuristic of minimizing the number of unsatisfied clauses
- Balance between greediness and randomness

# The WalkSAT algorithm

```
function WALKSAT(clauses, p, max-flips) returns a satisfying model or failure
    inputs: clauses, a set of clauses in propositional logic
            p, the probability of choosing to do a “random walk” move
            max-flips, number of flips allowed before giving up
    model  $\leftarrow$  a random assignment of true/false to the symbols in clauses
    for i = 1 to max-flips do
        if model satisfies clauses then return model
        clause  $\leftarrow$  a randomly selected clause from clauses that is false in model
        with probability p flip the value in model of a randomly selected symbol
            from clause
        else flip whichever symbol in clause maximizes the number of satisfied clauses
    return failure
```

Algorithm checks for satisfiability by randomly flipping the values of variables

# Hard satisfiability problems

- Consider random 3-CNF sentences: 3SAT problem
- Example:

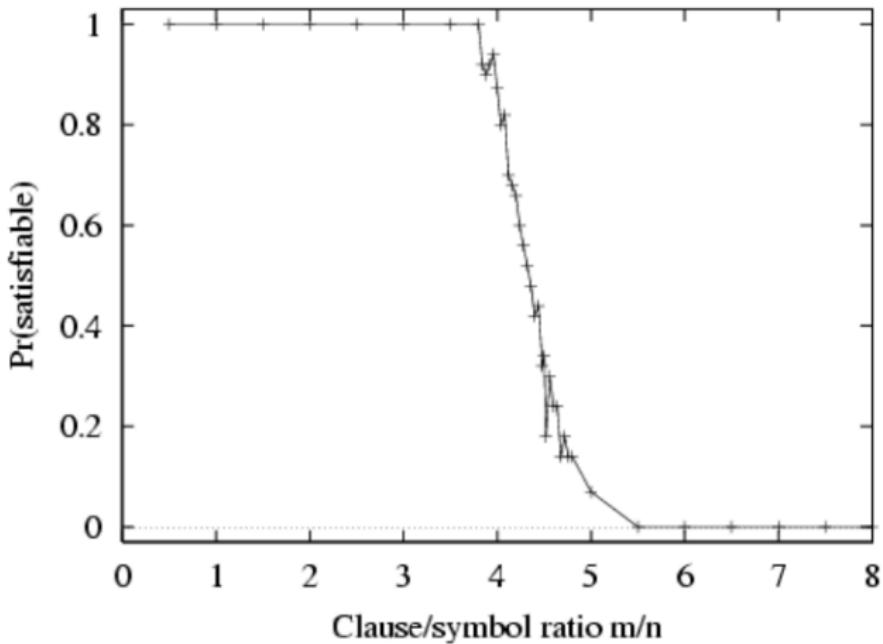
$$(\neg D \vee \neg B \vee C) \wedge (B \vee \neg A \vee \neg C) \wedge (\neg C \vee \neg B \vee E) \wedge (E \vee \neg D \vee B) \wedge (B \vee E \vee \neg C)$$

$m$ : number of clauses

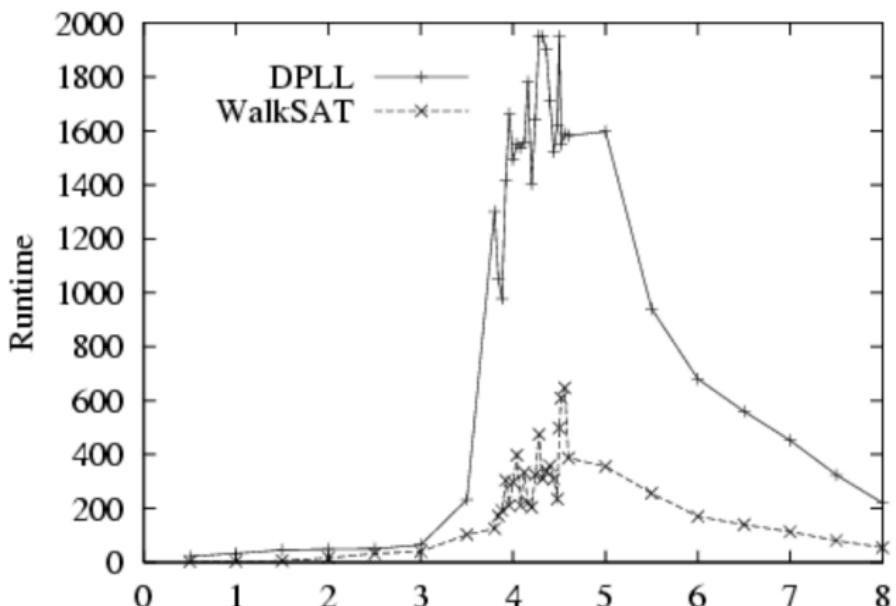
$n$ : number of symbols

- Hard problems seem to cluster near  $m/n = 4.3$  (critical point)

# Hard satisfiability problems



# Hard satisfiability problems



Median runtime for 100 satisfiable random 3-CNF sentences,  
 $n = 50$

# Inference-based agents in the wumpus world

- A wumpus-world agent using propositional logic:

$$\neg P_{1,1}$$

$$\neg W_{1,1}$$

$$B_{x,y} \Leftrightarrow (P_{x,y+1} \vee P_{x,y-1} \vee P_{x+1,y} \vee P_{x-1,y})$$

$$S_{x,y} \Leftrightarrow (W_{x,y+1} \vee W_{x,y-1} \vee W_{x+1,y} \vee W_{x-1,y})$$

$$W_{1,1} \vee W_{1,2} \vee \dots \vee W_{4,4}$$

$$\neg W_{1,1} \vee \neg W_{1,2}$$

$$\neg W_{1,1} \vee \neg W_{1,3}$$

...

⇒ 64 distinct proposition symbols, 155 sentences

# The Wumpus Agent (1)

```
function HYBRID-WUMPUS-AGENT (percept) returns an action
  inputs: percept, a list, [stench, breeze, glitter, bump, scream]
  persistent: KB, a knowledge base, initially the atemporal "wumpus physics"
    t, a counter, initially 0, indicating time
    plan, an action sequence, initially empty
  TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
  TELL the KB the temporal "physics" sentences for time t
  safe ← {[x,y] : ASK (KB,OKtx,y) = true}
  if ASK(KB, Glittert) = true then
    plan ← [Grab] + PLAN-ROUTE(current, {[1,1]}, safe) + [Climb]
  if plan is empty then
    unvisited ← {[x,y] : ASK(KB,Lt'x,y) = false for all t' ≤ t }
    plan ← PLAN-ROUTE(current, unvisited ∩ safe, safe)
  if plan is empty and ASK(KB, HaveArrowt) = true then
    possible_wumpus ← {[x,y] : ASK(KB,¬Wx,y) = false }
    plan ← PLAN-SHOT(current, possible_wumpus, safe)
  if plan is empty then // no choice but to take a risk
    not_unsafe ← {[x,y] : ASK(KB,¬OKtx,y) = false }
    plan ← PLAN-ROUTE(current, unvisited ∩ not_unsafe, safe)
  if plan is empty then
    plan ← PLAN-ROUTE(current, {[1,1]}, safe) + [Climb]
  action ← POP(plan)
  TELL(KB, MAKE-ACTION-SENTENCE(action, t))
  t ← t+1
  return action
```

# The Wumpus Agent (2)

```
function PLAN-ROUTE(current, goals, allowed) returns an action sequence
```

**inputs:** *current*, the agent's current position

*goals*, a set of squares; try to plan a route to one of them

*allowed*, a set of squares that can form part of the route

```
problem  $\leftarrow$  ROUTE-PROBLEM(current, goals, allowed)
```

```
return A*-GRAPH-SEARCH(problem)
```

We will look at this later on.

# We need more!

- Effect axioms:

$$L_{1,1}^0 \wedge \text{FacingEast}^0 \wedge \text{Forward}^0 \Rightarrow L_{2,1}^1 \wedge \neg L_{1,1}^1$$

- We need extra axioms about the world.
- Representational frame problem

- Frame axioms:

$$\text{Forward}^t \Rightarrow (\text{HaveArrow}^t \Leftrightarrow \text{HaveArrow}^{t+1})$$

$$\text{Forward}^t \Rightarrow (\text{WumpusAlive}^t \Leftrightarrow \text{WumpusAlive}^{t+1})$$

- Inferential frame problem

- Successor-state axioms:

$$\text{HaveArrow}^{t+1} \Leftrightarrow (\text{HaveArrow}^t \wedge \neg \text{Shoot}^t)$$

# Expressiveness limitation of propositional logic

- KB contains “physics” sentences for every single square
- For every time  $t$  and every location  $[x, y]$ ,

$$L_{x,y}^t \wedge \text{FacingRight}^t \wedge \text{Forward}^t \Rightarrow L_{x+1,y}^{t+1}$$

- Rapid proliferation of clauses

# Summary

- Logical agents apply inference to a knowledge base to derive new information and make decisions.
- Two algorithms: DPLL & WalkSAT
- Hard satisfiability problems
- Applications to Wumpus World.
- Propositional logic lacks expressive power

# Inf2D 06: Logical Agents: Knowledge Bases and the Wumpus World

Michael Herrmann

University of Edinburgh, School of Informatics

27/01/2017

**informatics**



Credits: The content of this lecture was prepared by Michael Rovatsos and follows R&N

# Outline

- Knowledge-based agents
- Wumpus world
- Logic in general – models and entailment
- Propositional (Boolean) logic
- Equivalence, validity, satisfiability

# Knowledge bases

Inference engine      ← domain-independent algorithms

Knowledge base      ← domain-specific content

- Knowledge base (KB): Set of **sentences** in a formal language
- **Declarative** approach to building a KB:
  - Tell it what it needs to know
- Then the agent can Ask the KB what to do
  - answers should follow from the KB
- KB can be part of agent or be accessible to many agents
- The agent's KB can be viewed at the **knowledge level**  
i.e., what it knows, regardless of how implemented
- Or at the **implementation level**
  - i.e., data structures in KB and algorithms that manipulate them

# A simple knowledge-based agent

```
function KB-AGENT(percept) returns an action
  persistent KB, a knowledge base
            t, a counter, initially 0, indicating time
  TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
  action  $\leftarrow$  ASK(KB, MAKE-ACTION-QUERY(t))
  TELL(KB, MAKE-ACTION-SENTENCE(action, t))
  t  $\leftarrow$  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

# Wumpus World PEAS description

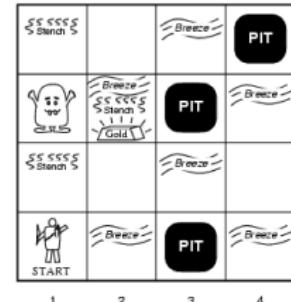
- Performance measure

- gold +1000, death -1000
- -1 per step, -10 for using arrow

- Environment

- Squares adjacent to Wumpus are smelly
- Squares adjacent to pits are breezy
- Glitter iff gold is in the same square
- Shooting kills Wumpus if you are facing it
- Shooting uses up the only arrow
- Grabbing picks up gold if in same square
- Releasing drops the gold in same square

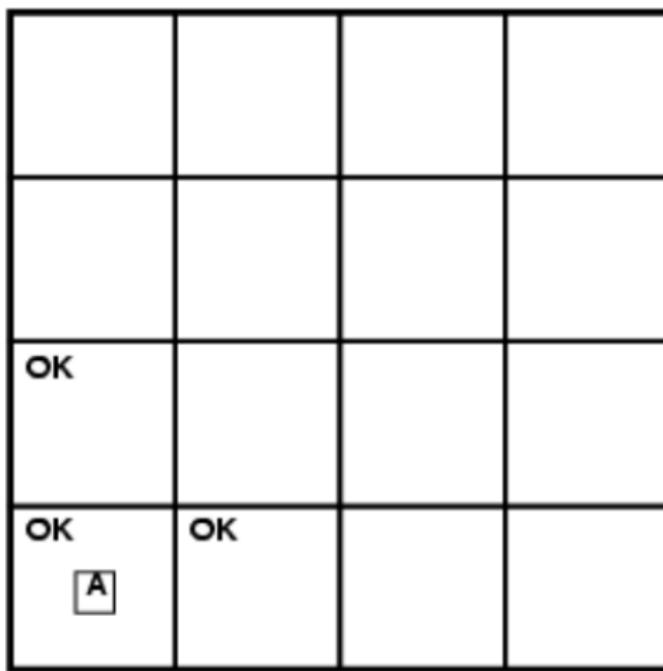
- **Actuators:** Left turn, Right turn, Forward, Grab, Release, Shoot
- **Sensors:** Stench, Breeze, Glitter, Bump, Scream



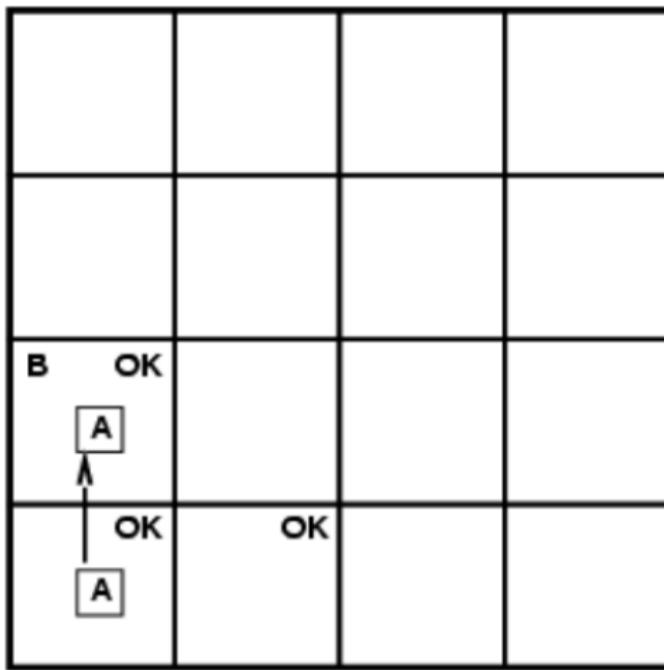
# Wumpus world characterization

- Fully Observable? No – only local perception
- Deterministic? Yes – outcomes exactly specified
- Episodic? No – sequential at the level of actions
- Static? Yes – Wumpus and Pits do not move
- Discrete? Yes
- Single-agent? Yes – Wumpus is essentially a natural feature

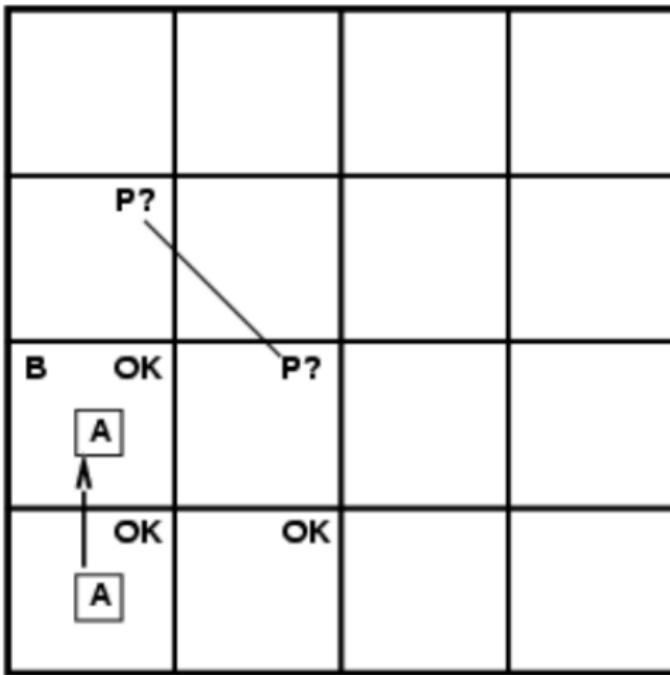
# Exploring a Wumpus world



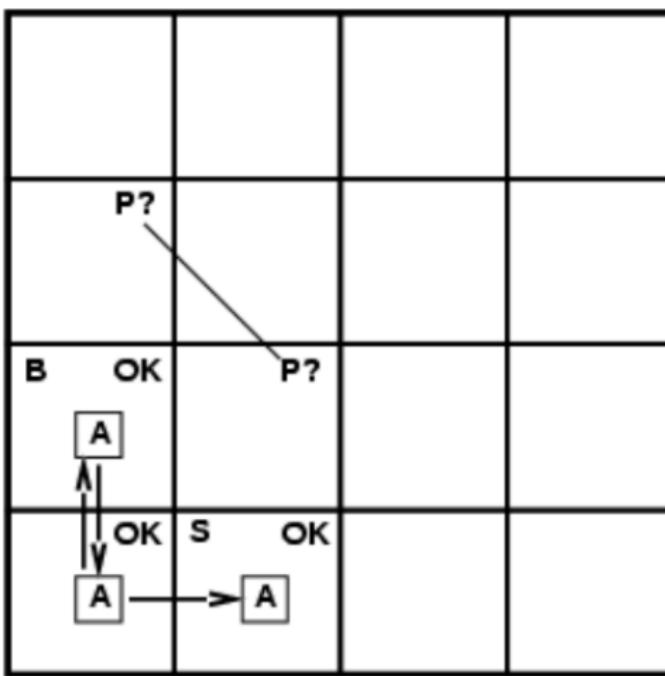
# Exploring a Wumpus world



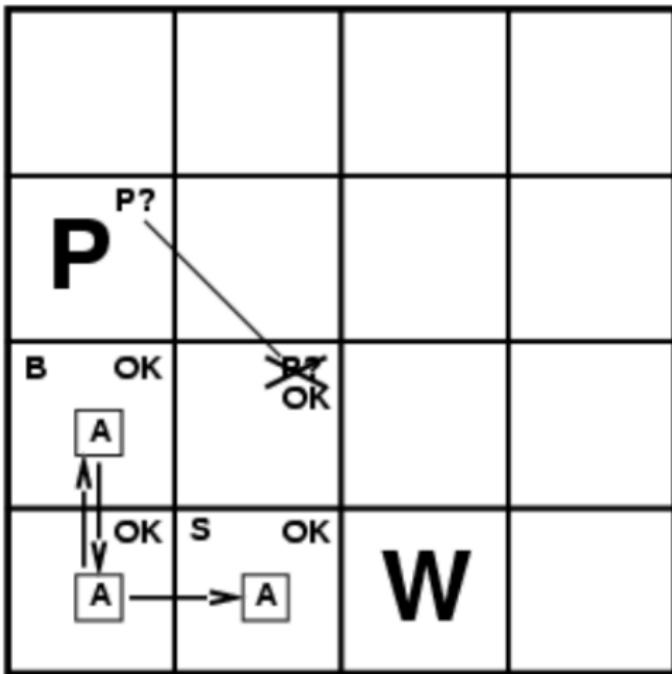
# Exploring a Wumpus world



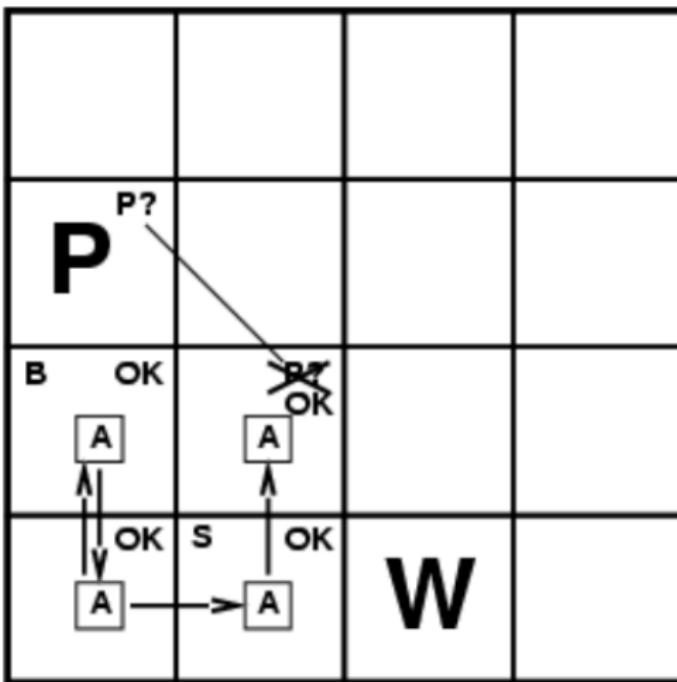
# Exploring a Wumpus world



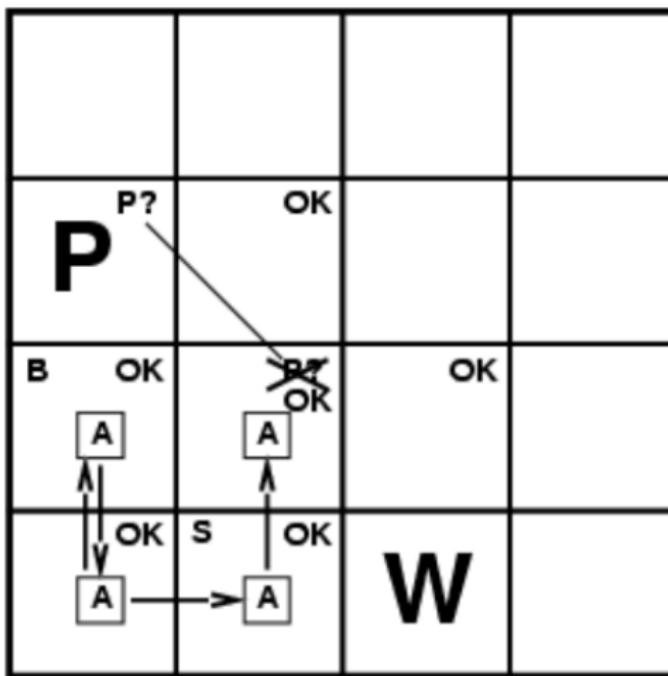
# Exploring a Wumpus world



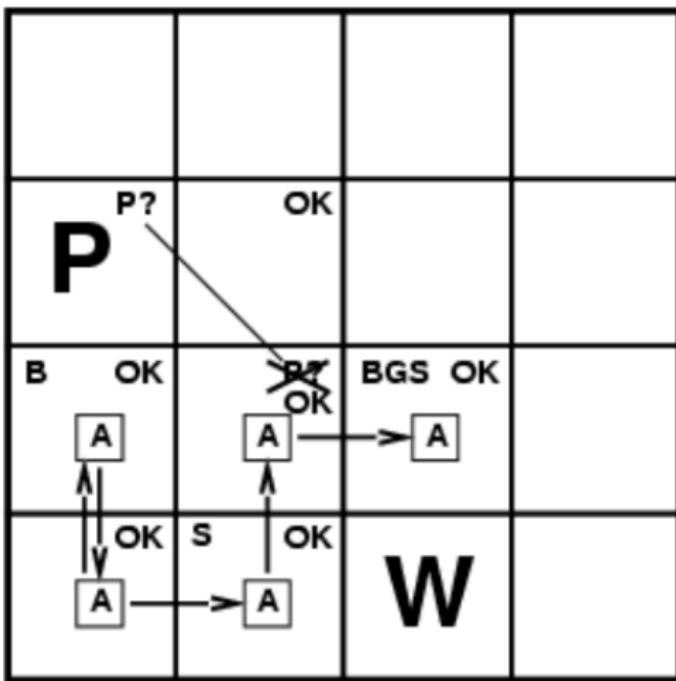
# Exploring a Wumpus world



# Exploring a Wumpus world



# Exploring a Wumpus world



# Logic in general

- **Logics** are formal languages for representing information such that conclusions can be drawn
- **Syntax** defines the sentences in the language
- **Semantics** defines the “meaning” of sentences
  - i.e., define truth of a sentence in a world
- E.g., the language of arithmetic
  - $x + 2 \geq y$  is a sentence;  $x2 + y >$  is not a sentence
  - $x + 2 \geq y$  is true iff the number  $x + 2$  is no less than the number  $y$
  - $x + 2 \geq y$  is true in a world where  $x = 7, y = 1$
  - $x + 2 \geq y$  is false in a world where  $x = 0, y = 6$

# Entailment

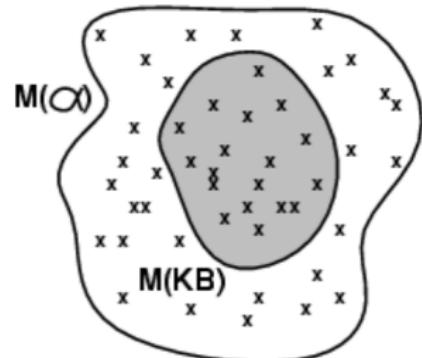
- Entailment means that one thing follows from another:

$$KB \models \alpha$$

- Knowledge base  $KB$  entails sentence  $\alpha$  if and only if  $\alpha$  is true in all worlds where  $KB$  is true
  - e.g., the  $KB$  containing “Celtic won” and “Hearts won” entails “Celtic won or Hearts won”
  - Considering only worlds where Celtic plays Hearts (and no draws) it entails “Either Celtic won or Hearts won”
  - e.g.,  $x + y = 4$  entails  $4 = x + y$
  - Entailment is a relationship between sentences (i.e., syntax) that is based on semantics

# Models

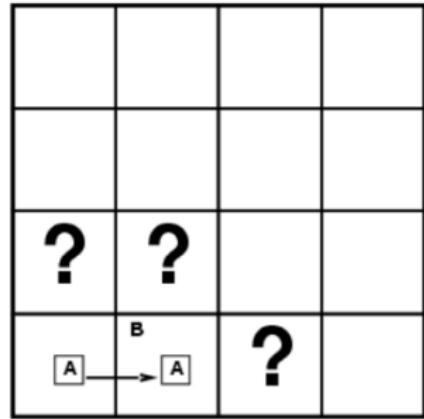
- Logicians typically think in terms of **models**, which are formally structured worlds with respect to which truth can be evaluated
- We say  $m$  is a **model** of a sentence  $\alpha$  if  $\alpha$  is true in  $m$
- $M(\alpha)$  is the set of all models of  $\alpha$
- Then  $KB \models \alpha$  iff  $M(KB) \subseteq M(\alpha)$
- The *stronger* an assertion, the fewer models it has.



# Entailment in the Wumpus world

Situation after detecting nothing in [1,1], moving right, breeze in [2,1]

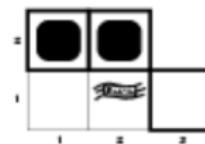
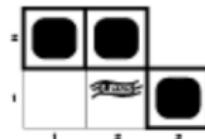
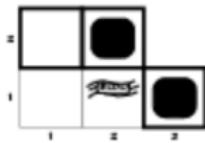
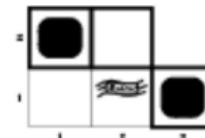
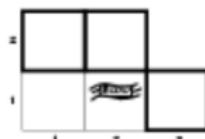
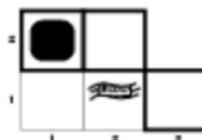
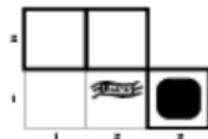
Consider possible models for KB assuming only pits



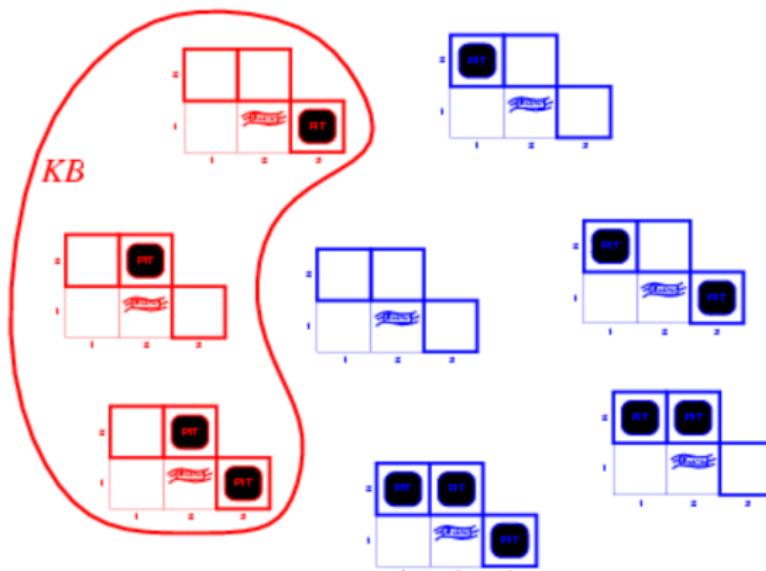
3 Boolean choices  $\Rightarrow$  8 possible models

Mid-lecture Exercise: What are these 8 models?

# Wumpus models

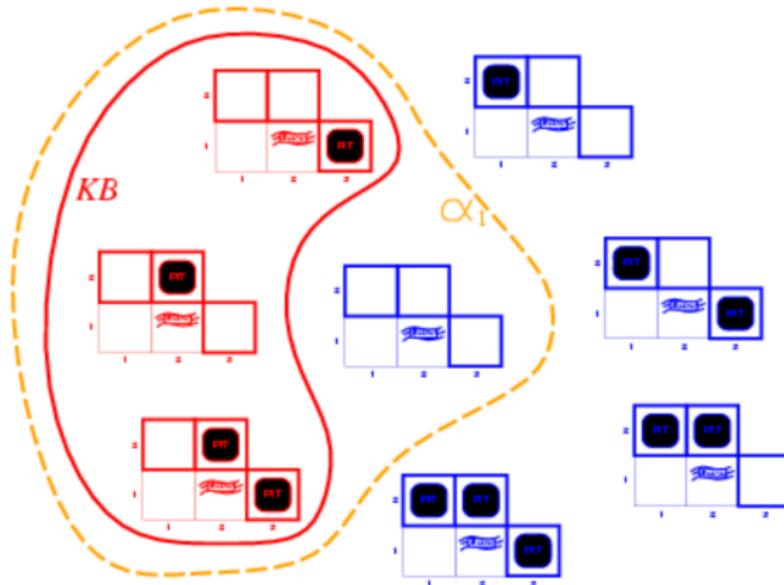


# Wumpus models



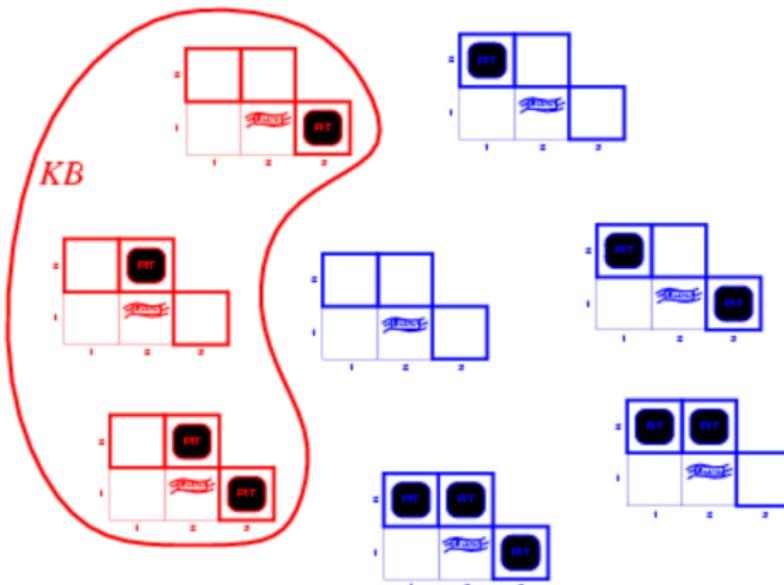
- $KB = \text{Wumpus-world rules} + \text{observations}$

# Wumpus models



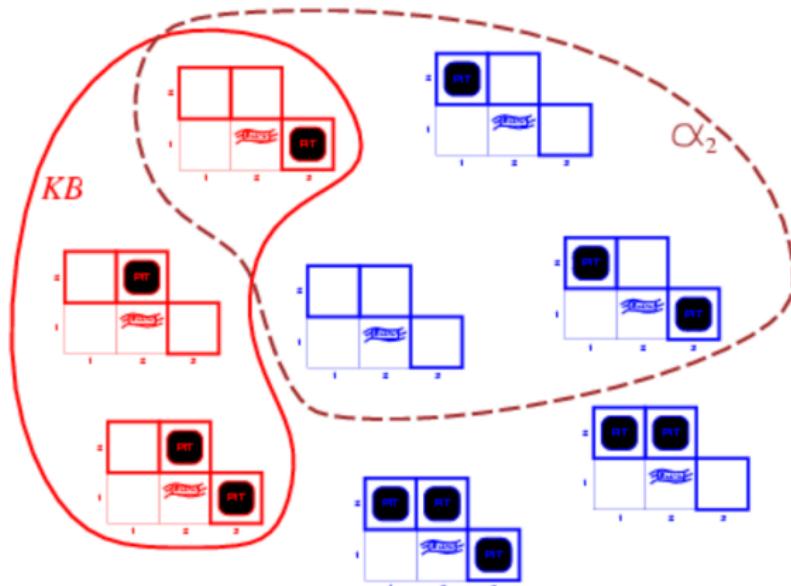
- $KB$  = Wumpus-world rules + observations
- $\alpha_1 = "[1,2] \text{ has no pit}"$ ,  $KB \models \alpha_1$ , proved by model checking
  - In every model in which  $KB$  is true,  $\alpha_1$  is also true

# Wumpus models



- $KB = \text{Wumpus-world rules} + \text{observations}$

# Wumpus models



- $KB = \text{Wumpus-world rules} + \text{observations}$
- $\alpha_2 = "[2,2] \text{ has no pit}", KB \not\models \alpha_2$ 
  - In some models in which  $KB$  is true,  $\alpha_2$  is also true

# Inference

- $KB \vdash_i \alpha$ : sentence  $\alpha$  can be derived from  $KB$  by an inference procedure  $i$
- Soundness:  $i$  is sound if whenever  $KB \vdash_i \alpha$ , it is also true that  $KB \vDash \alpha$
- Completeness:  $i$  is complete if whenever  $KB \vDash \alpha$ , it is also true that  $KB \vdash_i \alpha$
- Preview: we will define first-order logic:
  - expressive enough to say almost anything of interest,
  - sound and complete inference procedure exists.
  - But first...

# 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)

# Propositional logic: Semantics

- Each model specifies true/false for each proposition symbol
  - e.g.  $P_{1,2}$   $P_{2,2}$   $P_{3,1}$   
false true false

With these symbols, 8 possible models, can be enumerated automatically.

- Rules for evaluating truth with respect to a model  $m$ :

$\neg S$	is true iff	$S$ is false
$S_1 \wedge S_2$	is true iff	$S_1$ is true and $S_2$ is true
$S_1 \vee S_2$	is true iff	$S_1$ is true or $S_2$ is true
$S_1 \Rightarrow S_2$ i.e.	is true iff	$S_1$ is false or $S_2$ is true $S_1$ is true and $S_2$ is false
$S_1 \Leftrightarrow S_2$	is true iff	$S_1 \Rightarrow S_2$ is true and $S_2 \Rightarrow S_1$ is true

- Simple recursive process evaluates an arbitrary sentence, e.g.,

$$\neg P_{1,2} \wedge (P_{2,2} \vee P_{3,1}) = \text{true} \wedge (\text{true} \vee \text{false}) = \text{true} \wedge \text{true} = \text{true}$$

# Truth tables for connectives

$P$	$Q$	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
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

# Wumpus world sentences

Let  $P_{i,j}$  be true if there is a pit in  $[i,j]$ .

Let  $B_{i,j}$  be true if there is a breeze in  $[i,j]$ .

$$\neg P_{1,1}$$

$$\neg B_{1,1}$$

$$B_{2,1}$$

"Pits cause breezes in adjacent squares"

$$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$$

$$B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$$

Recall:  $\alpha_1 = "[1,2] \text{ has no pit}"$ ,

# Truth tables for inference

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	$KB$	$\alpha_1$
false	false	true						
false	false	false	false	false	false	true	false	true
:	:	:	:	:	:	:	:	:
true	true	false	false	false	false	false	false	true
false	true	false	false	false	false	true	true	true
false	true	false	false	false	true	false	true	true
false	true	false	false	false	true	true	true	true
false	true	false	false	true	false	false	false	true
:	:	:	:	:	:	:	:	:
true	false	false						

# Inference by enumeration

- Depth-first enumeration of all models is sound and complete

```
function TT-ENTAILS?(KB,  $\alpha$ ) returns true or false
    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))
```

- PL-TRUE? returns true if a sentence holds within a model
- EXTEND( $P$ ,  $val$ ,  $model$ ) returns a new partial model in which  $P$  has value  $val$
- For  $n$  symbols, time complexity:  $O(2^n)$ , space complexity:  $O(n)$

# Logical equivalence

- Two sentences are logically equivalent iff true in the same models:  $\alpha \equiv \beta$  iff  $\alpha \models \beta$  and  $\beta \models \alpha$

$$(\alpha \wedge \beta) \equiv (\beta \wedge \alpha) \text{ commutativity of } \wedge$$

$$(\alpha \vee \beta) \equiv (\beta \vee \alpha) \text{ commutativity of } \vee$$

$$((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma)) \text{ associativity of } \wedge$$

$$((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma)) \text{ associativity of } \vee$$

$$\neg(\neg \alpha) \equiv \alpha \text{ double-negation elimination}$$

$$(\alpha \Rightarrow \beta) \equiv (\neg \beta \Rightarrow \neg \alpha) \text{ contraposition}$$

$$(\alpha \Rightarrow \beta) \equiv (\neg \alpha \vee \beta) \text{ implication elimination}$$

$$(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) \text{ biconditional elimination}$$

$$\neg(\alpha \wedge \beta) \equiv (\neg \alpha \vee \neg \beta) \text{ de Morgan}$$

$$\neg(\alpha \vee \beta) \equiv (\neg \alpha \wedge \neg \beta) \text{ de Morgan}$$

$$(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) \text{ distributivity of } \wedge \text{ over } \vee$$

$$(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) \text{ distributivity of } \vee \text{ over } \wedge$$

# Validity and satisfiability

- A sentence is **valid** if it is true in **all** models,

e.g. true,  $A \vee \neg A$ ,  $A \Rightarrow A$ ,  $(A \wedge (A \Rightarrow B)) \Rightarrow B$

- Validity is connected to inference via the **Deduction Theorem**:

$KB \vDash \alpha$  if and only if  $(KB \Rightarrow \alpha)$  is valid

- A sentence is **satisfiable** if it is true in **some** model

e.g.,  $A \vee B$ ,  $C$

- A sentence is **unsatisfiable** if it is true in **no** models

e.g.,  $A \wedge \neg A$

- Satisfiability is connected to inference via the following:

$KB \vDash \alpha$  if and only if  $(KB \wedge \neg \alpha)$  is unsatisfiable

# Proof methods

Proof methods divide into (roughly) two kinds:

- Application of inference rules

- Legitimate (sound) generation of new sentences from old
- Proof = a sequence of inference rule applications.  
Can use inference rules as operators in a standard search algorithm
- Typically require transformation of sentences into a normal form
- Example: resolution

- Model checking

- truth table enumeration (always exponential in  $n$ )
- improved backtracking, e.g.,  
Davis-Putnam-Logemann-Loveland (DPLL) method
- heuristic search in model space (sound but incomplete)  
e.g., min-conflicts-like hill-climbing algorithms

# Summary

- Logical agents apply inference to a knowledge base to derive new information and make decisions
- Basic concepts of logic:
  - **syntax**: formal structure of sentences
  - **semantics**: truth of sentences w.r.t. models
  - **entailment**: necessary truth of one sentence given another
  - **inference**: deriving sentences from other sentences
  - **soundness**: derivations produce only entailed sentences
  - **completeness**: derivations can produce all entailed sentences
- Wumpus world requires the ability to represent partial and negated information, reason by cases, etc.
- Does propositional logic provide enough expressive power for statements about the real world?

# Inf2D 08: Smart Searching Using Constraints

Michael Herrmann

University of Edinburgh, School of Informatics

02/02/2017

**informatics**



Credits: The content of this lecture was prepared by Michael Rovatsos and follows R&N

# Outline

- Constraint Satisfaction Problems (CSP)
- Backtracking search for CSPs
- Making this more efficient

# Constraint satisfaction problems (CSPs)

- Standard search problem:
  - state is a ‘black box’ – any data structure that supports successor function, heuristic function and goal test.
- CSP:
  - state is defined by variables  $X_i$  with values from domain  $D_i \quad (i = 1, \dots, n)$
  - goal test is a set of constraints specifying allowable combinations of values for subsets of variables.
  - Simple example of a *formal representation language*.
  - Allows useful general-purpose algorithms with more power than standard search algorithms.

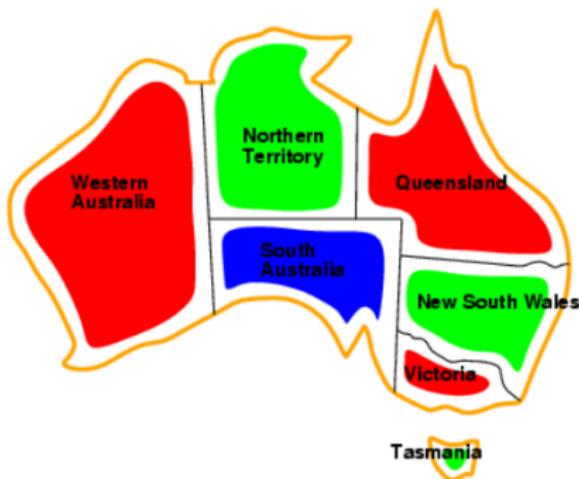
# Example: Map-Colouring



- Variables  $WA, NT, Q, NSW, V, SA, T$
- Domains  $D_i = \text{red}, \text{green}, \text{blue}$
- Constraints: adjacent regions must have different colours,
  - e.g.  $WA \neq NT$ ,
  - or  $(WA, NT)$  in

$$\{(\text{red}, \text{green}), (\text{red}, \text{blue}), (\text{green}, \text{red}), (\text{green}, \text{blue}), (\text{blue}, \text{red}), (\text{blue}, \text{green})\}.$$

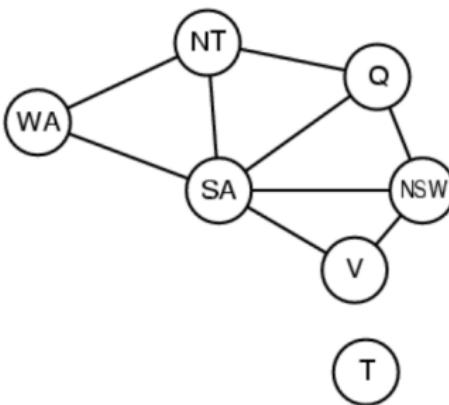
# Example: Map-Colouring



- Solutions are complete and consistent assignments,
  - e.g. WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green.

# Constraint graph

- **Binary CSP:** each constraint relates two variables.
- **Constraint graph:**
  - nodes are variables,
  - arcs (edges) represent constraints relating two nodes each.



# Varieties of CSPs

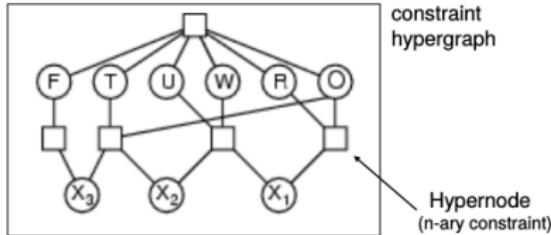
- Discrete variables:
  - finite domains:
    - $n$  variables, domain size  $d \rightarrow O(d^n)$ , complete assignments.
    - e.g. Boolean CSPs, incl. Boolean satisfiability (NP-complete).
  - infinite domains:
    - integers, strings, etc.
    - e.g. job scheduling, variables are start/end days for each job.
    - need a constraint language, e.g.  
 $\text{StartJob}_1 + 5 \leq \text{StartJob}_3$
- 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.  $SA \neq \text{green}$ .
- **Binary** constraints involve pairs of variables,
  - e.g.  $SA \neq WA$ .
- **Higher-order** constraints involve 3 or more variables,
  - e.g. crypt-arithmetic column constraints.
- **Global** constraints involve an arbitrary number of variables

# Example: Crypt-arithmetic

$$\begin{array}{r} \text{T} \ \text{W} \ \text{O} \\ + \ \text{T} \ \text{W} \ \text{O} \\ \hline \text{F} \ \text{O} \ \text{U} \ \text{R} \end{array}$$



- Variables:  $F, T, U, W, R, O, X_1, X_2, X_3$
- Domains:  $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- Constraints:  
  - $O + O = R + 10 \cdot X_1$
  - $X_1 + W + W = U + 10 \cdot X_2$
  - $X_2 + T + T = O + 10 \cdot X_3$ .
  - $X_3 = F, T \neq 0, F \neq 0.$← global constraint

# 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.

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

# Standard search formulation (incremental)

Let's start with the straightforward approach, then adapt it.

States are defined by the values assigned so far.

- **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.
- 
- 1 This is the same for all CSPs.
  - 2 Every solution appears at depth  $n$  with  $n$  variables → use depth-first search.

# Backtracking search

- Variable assignments are **commutative**,
  - e.g. [ WA = red then NT = green ] same as [ NT = green then WA = red ].
- Only need to consider assignments to a single variable at each node →  $b = d$  and there are  $d^n$  leaves.
- 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

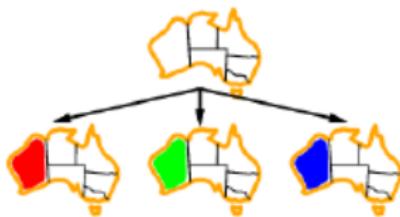
```
function BACKTRACKING-SEARCH(csp) returns a solution, or failure
    return BACKTRACK({ }, csp)

function BACKTRACK(assignment, csp) returns a solution, or failure
    if assignment is complete then return assignment
    var  $\leftarrow$  SELECT-UNASSIGNED-VARIABLE(csp)
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
        if value is consistent with assignment then
            add {var = value} to assignment
            inferences  $\leftarrow$  INFERENCE(csp, var, value)  $\leftarrow$  Optional: Can be used to
                impose arc-consistency
                (more on this later)
            if inferences  $\neq$  failure then
                add inferences to assignment
                result  $\leftarrow$  BACKTRACK(assignment, csp)
                if result  $\neq$  failure then
                    return result
                remove {var = value} and inferences from assignment
            return failure
    
```

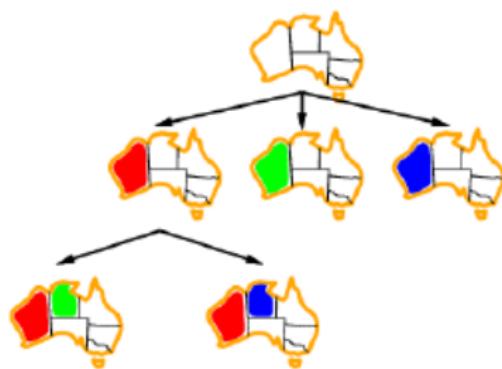
# Backtracking example



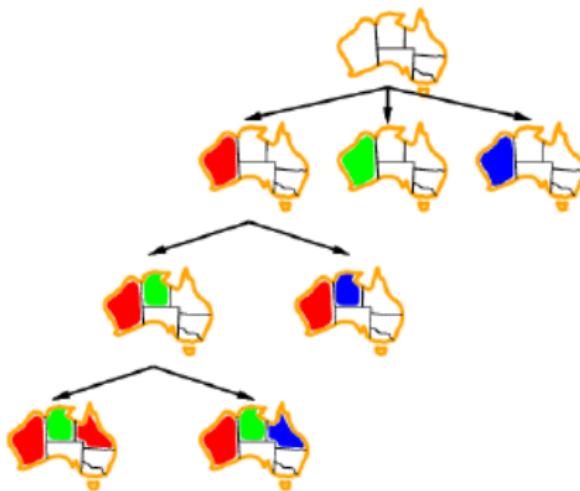
# Backtracking example



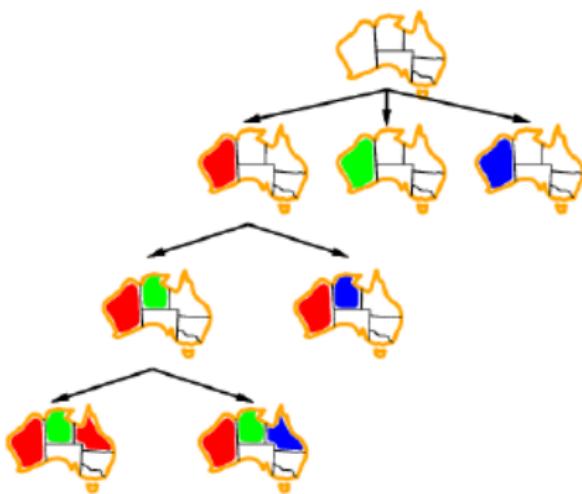
# Backtracking example



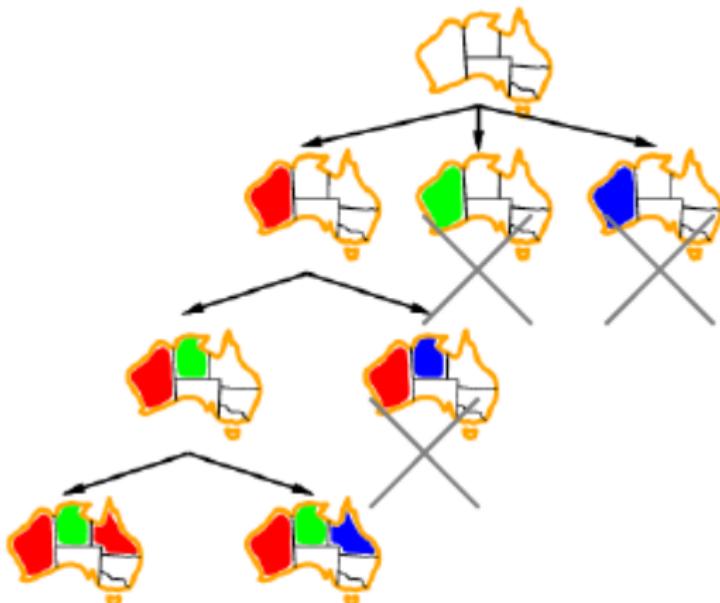
# Backtracking example



Which nodes can be eliminated on symmetry grounds?



# Solution



# Improving backtracking efficiency

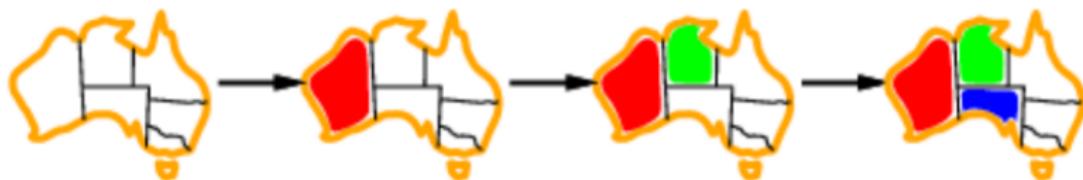
General-purpose methods can give huge gains in speed:

- Which **variable** should be assigned next?
  - **SELECT-UNASSIGNED-VARIABLE**
- Then, in what order should its **values** be tried?
  - **ORDER-DOMAIN-VALUES**
- What inferences should be performed at each step of the search?
  - **INFERENCE**
- Can we detect inevitable failure early?

# Most constrained variable

$\text{var} \leftarrow \text{SELECT-UNASSIGNED-VARIABLE}(csp)$

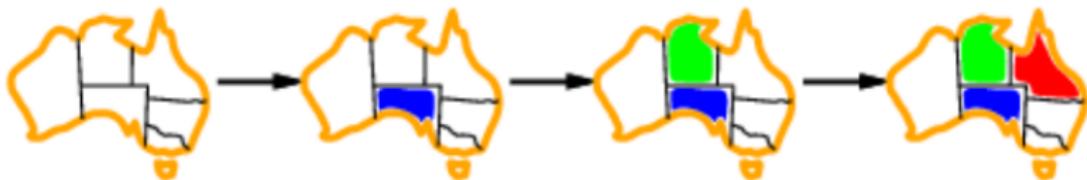
- Most constrained variable: choose the variable with the fewest legal values.



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

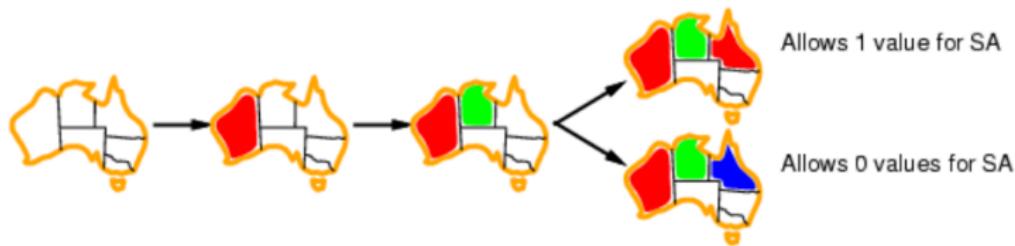
# Most constraining variable

- Tie-breaker among most constrained variables.
- Most constraining variable:
  - choose the variable with the most constraints on remaining variables – thus reducing branching.
- a.k.a. degree heuristic



# 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.

# Inference: Forward checking

Idea:

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

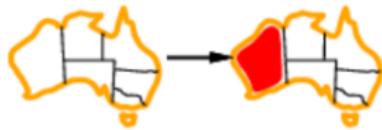


WA	NT	Q	NSW	V	SA	T
█ Red	█ Green	█ Blue	█ Red	█ Green	█ Blue	█ Red

# Forward checking

Idea:

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

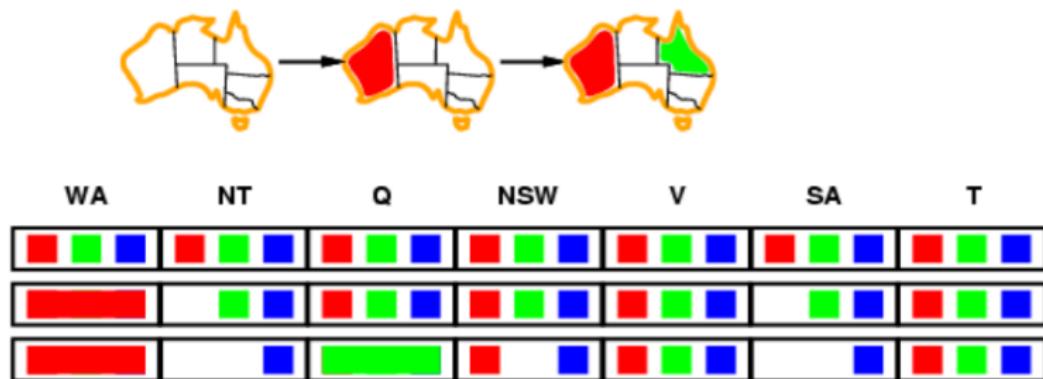


WA	NT	Q	NSW	V	SA	T
█ Red █ Green █ Blue						
█ Red █ █ █	█ Green █ Blue	█ Red █ Green █ Blue	█ Red █ Green █ Blue	█ Red █ Green █ Blue	█ Green █ Blue	█ Red █ Green █ Blue

# Forward checking

Idea:

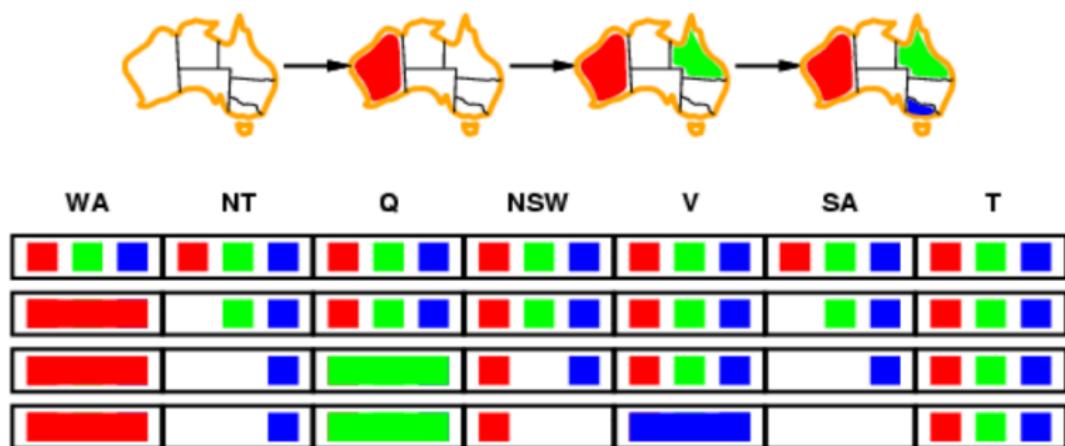
- Keep track of remaining legal values for unassigned variables.
- Terminate search when any variable has no legal values.



# Forward checking

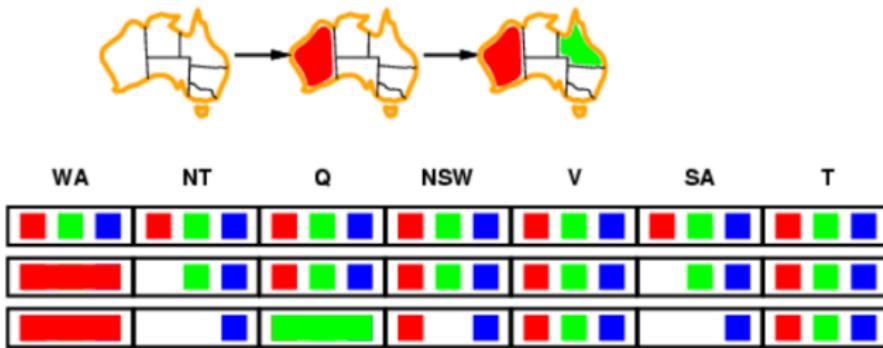
Idea:

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



# Constraint propagation

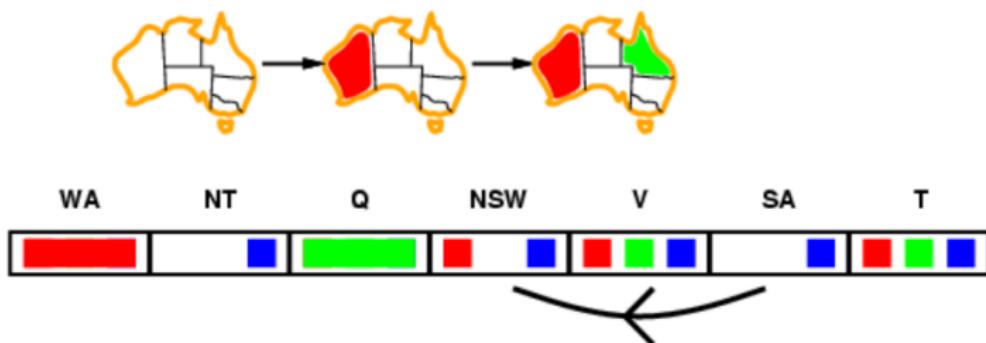
- Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures:



- NT and SA cannot both be blue!
- Constraint propagation repeatedly enforces constraints locally.

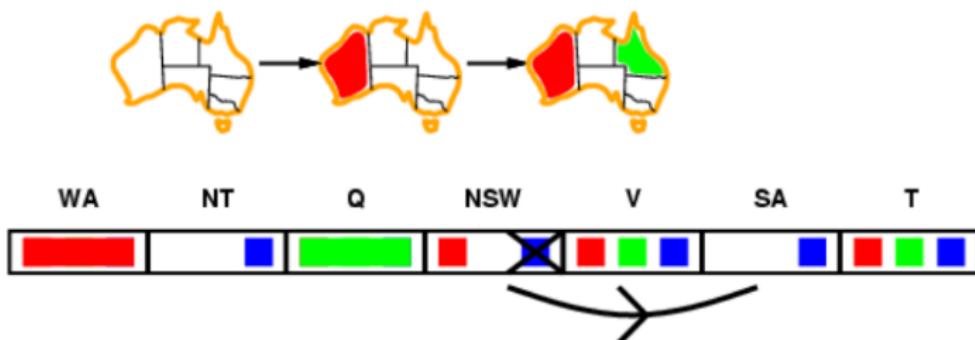
# Arc consistency

- Simplest form of propagation makes each arc **consistent**.
- $X \rightarrow Y$  is consistent iff
  - for **every** value  $x$  of in the domain of  $X$  there is **some** allowed  $y$  in the domain of  $Y$ .



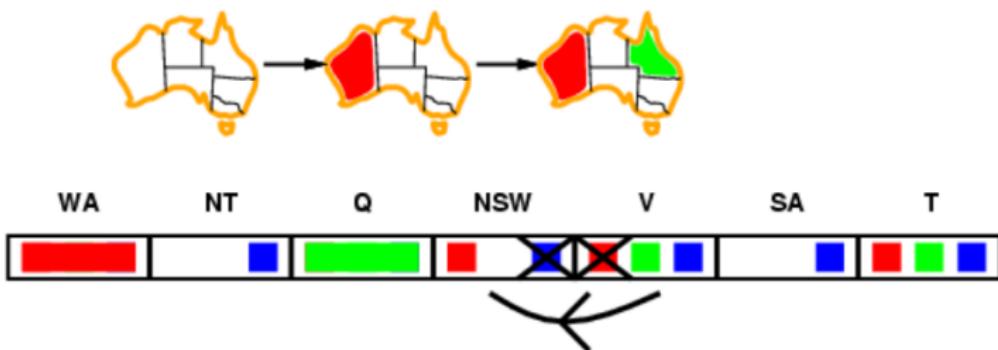
# Arc consistency

- Simplest form of propagation makes each arc **consistent**.
- $X \rightarrow Y$  is consistent iff
  - for **every** value  $x$  of in the domain of  $X$  there is **some** allowed  $y$  in the domain of  $Y$ .



# Arc consistency

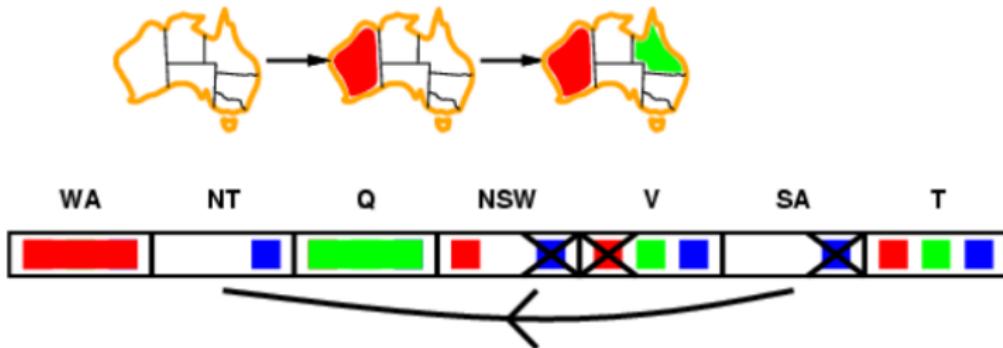
- Simplest form of propagation makes each arc **consistent**.
- $X \rightarrow Y$  is consistent iff
  - for **every** value  $x$  of in the domain of  $X$  there is **some** allowed  $y$  in the domain of  $Y$ .



- If  $X$  loses a value, neighbours of  $X$  need to be rechecked

# Arc consistency

- Simplest form of propagation makes each arc **consistent**.
- $X \rightarrow Y$  is consistent iff
  - for **every** value  $x$  of in the domain of  $X$  there is **some** allowed  $y$  in the domain of  $Y$ .



- If  $X$  loses a value, neighbours 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 false if an inconsistency is found and true otherwise
  inputs: csp, a binary CSP with components (X, D, C)
  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 REVISE(csp, Xi, Xj) then ← Make Xi arc-consistent with respect to Xj
      if size of Di = 0 then return false ← No consistent value left for Xi so fail
      for each Xk in Xi.NEIGHBORS - {Xj} do
        add (Xk, Xi) to queue ← Since revision occurred, add all neighbours
                                         of Xi for consideration (or reconsideration)
    return true
```

---

```
function REVISE(csp, Xi, Xj) returns true iff we revise the domain of Xi
  revised ← false
  for each x in Di do
    if no value y in Dj allows (x,y) to satisfy the constraint between Xi and Xj then
      delete x from Di
      revised ← true
  return revised
```

- Time complexity:  $O(cd^3)$ , where  $d$  is maximum size of each domain and  $c$  is the number of binary constraints (arcs).
- Space complexity  $O(c)$

# 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
- Variable ordering and value selection 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

# Inf2D 10: First-Order Logic

Michael Herrmann

University of Edinburgh, School of Informatics

07/02/2017

**informatics**



Credits: The content of this lecture was prepared by Michael Rovatsos and follows R&N

# Outline

- Why FOL?
- Syntax and semantics of FOL
- Using FOL
- Wumpus world in FOL

# Pros and cons of propositional logic

- Propositional logic is declarative
- Propositional logic allows partial/disjunctive/negated information
  - unlike most data structures and databases
- Propositional logic is compositional:
  - The meaning of  $B_{1,1} \wedge P_{1,2}$  is derived from that of  $B_{1,1}$  and of  $P_{1,2}$
- Meaning in propositional logic is context-independent
  - unlike natural language, where meaning depends on context
- Propositional logic has very limited expressive power
  - unlike natural language
  - for example, we cannot say “pits cause breezes in adjacent squares”, except by writing one sentence for each square

# First-order logic

Whereas propositional logic assumes the world contains facts, first-order logic (like natural language) assumes the world contains:

- **Objects**: people, houses, numbers, colours, football games, wars, ...
- **Relations**: red, round, prime, brother of, bigger than, part of, comes between, ...
- **Functions**: father of, best friend, one more than, plus, ...

# Syntax of FOL: Basic elements

- Constants KingJohn, 2, UoE,...
- Predicates Brother, >, ...
- Functions Sqrt, LeftLegOf, ...
- Variables Variables  $x, y, a, b, \dots$
- Connectives  $\neg, \Rightarrow, \wedge, \vee, \Leftrightarrow$
- Equality =
- Quantifiers  $\forall, \exists$

# Atomic formulae

Atomic formula = predicate ( $\text{term}_1, \dots, \text{term}_n$ )

or  $\text{term}_1 = \text{term}_2$

Term = function ( $\text{term}_1, \dots, \text{term}_n$ )

or constant or variable

Examples:

- Brother(KingJohn, RichardTheLionheart)
- >(Length(LeftLegOf(Richard)), Length(LeftLegOf(KingJohn)))



predicate



functions



constants



# Complex formulae

Complex formulae are made from atomic formulae using connectives

$$\neg S, S_1 \wedge S_2, S_1 \vee S_2, S_1 \Rightarrow S_2, S_1 \Leftrightarrow S_2$$

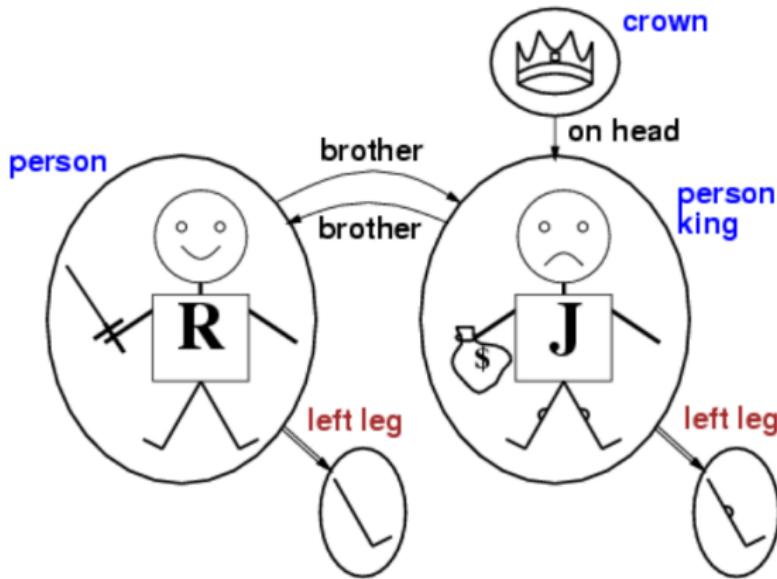
Examples:

- $\text{Sibling}(\text{KingJohn}, \text{Richard}) \Rightarrow \text{Sibling}(\text{Richard}, \text{KingJohn})$
- $>(1, 2) \vee \leq(1, 2)$
- $>(1, 2) \wedge \neg >(1, 2)$

# Semantics of first-order logic

- Formulae are mapped to an interpretation
  - An interpretation is called a model of a set of formulae when all the formulae are true in the interpretation.
- Interpretation contains objects (domain elements) and relations between them
- Mapping specifies referents for
  - constant symbols  $\mapsto$  objects
  - predicate symbols  $\mapsto$  relations
  - function symbols  $\mapsto$  functions
- An atomic formula  $\text{predicate}(\text{term}_1, \dots, \text{term}_n)$  is true iff the objects referred to by  $\text{term}_1, \dots, \text{term}_n$  are in the relation referred to by predicate.

# Interpretations for FOL: Example



# Universal quantification

- $\forall <variables>. <formula>$ 
  - i.e. will often write  $\forall x, y.P$  for  $\forall x. \forall y. P$
  - Example: Everyone at UoE is smart:  
 $\forall x. \text{At}(x, \text{UoE}) \Rightarrow \text{Smart}(x)$
- $\forall x. P$  is true in an interpretation  $m$  iff  $P$  is true with  $x$  being each possible object in the interpretation.
- Roughly speaking, equivalent to the conjunction of instantiations of  $P$

$$\begin{aligned} & \text{At}(\text{KingJohn}, \text{UoE}) \Rightarrow \text{Smart}(\text{KingJohn}) \\ & \wedge \text{At}(\text{Richard}, \text{UoE}) \Rightarrow \text{Smart}(\text{Richard}) \\ & \wedge \text{At}(\text{UoE}, \text{UoE}) \Rightarrow \text{Smart}(\text{UoE}) \\ & \wedge \dots \quad \dots \quad \dots \end{aligned}$$

# A common mistake to avoid

- Typically,  $\Rightarrow$  is the main connective with  $\forall$
- Common mistake: using  $\wedge$  as the main connective with  $\forall$ :

$$\forall x. At(x, UoE) \wedge Smart(x)$$

means “Everyone is at UoE and everyone is smart”

# Existential quantification

- $\exists <variables>. <formula>$ 
  - i.e. will often write  $\exists x, y.P$  for  $\exists x. \exists y. P$
  - Example: Someone at UoE is smart:  
 $\exists x. \text{At}(x, \text{UoE}) \wedge \text{Smart}(x)$
- $\exists x. P$  is true in an interpretation  $m$  iff  $P$  is true with  $x$  being some possible object in the interpretation.
- Roughly speaking, equivalent to the disjunction of instantiations of  $P$

$$\begin{aligned} & \text{At(KingJohn, UoE)} \wedge \text{Smart(KingJohn)} \\ \vee & \text{At(Richard, UoE)} \wedge \text{Smart(Richard)} \\ \vee & \text{At(UoE, UoE)} \wedge \text{Smart(UoE)} \\ \vee & \dots \quad \dots \quad \dots \end{aligned}$$

## Another common mistake to avoid

- Typically,  $\wedge$  is the main connective with  $\exists$
- Common mistake: using  $\Rightarrow$  as the main connective with  $\exists$ :

$$\exists x. At(x, UoE) \Rightarrow Smart(x)$$

is true if there is anyone who is not at UoE!

# Properties of quantifiers

- $\forall x.\forall y.$  is the same as  $\forall y.\forall x.$
- $\exists x.\exists y.$  is the same as  $\exists y.\exists x.$
- $\exists x.\forall y.$  is **not** the same as  $\forall y.\exists x.$ 
  - $\exists x.\forall y.Loves(x, y)$   
“There is a person who loves everyone in the world”
  - $\forall y.\exists x.Loves(x, y)$   
“Everyone in the world is loved by at least one person”
- **Quantifier duality:** each can be expressed using the other:
  - $\forall x.Likes(x, IceCream) \equiv \neg\exists x.\neg Likes(x, IceCream)$
  - $\exists x.Likes(x, Broccoli) \equiv \neg\forall x.\neg Likes(x, Broccoli)$

# Equality

- $\text{term}_1 = \text{term}_2$  is true under a given interpretation if and only if  $\text{term}_1$  and  $\text{term}_2$  refer to the same object.
- Definition: If  $\text{term}_1$  and  $\text{term}_2$  yield the same truth values for any predicate.
- Example: Definition of *Sibling* in terms of *Parent*:

$$\forall x, y. \text{Sibling}(x, y) \Leftrightarrow (\neg(x = y) \wedge \exists m, f. \neg(m = f)$$

$$\wedge \text{Parent}(m, x) \wedge \text{Parent}(f, x) \wedge \text{Parent}(m, y) \wedge \text{Parent}(f, y))$$

# Using FOL

Example: The kinship domain:

- Brothers are siblings

$$\forall x, y. \text{Brother}(x, y) \Rightarrow \text{Sibling}(x, y)$$

- One's mother is one's female parent

$$\forall m, c. \text{Mother}(c) = m \Leftrightarrow (\text{Female}(m) \wedge \text{Parent}(m, c))$$

- “Sibling” is symmetric

$$\forall x, y. \text{Sibling}(x, y) \Leftrightarrow \text{Sibling}(y, x)$$

# Using FOL

The set domain:

- $\forall s. Set(s) \Leftrightarrow (s = \{\}) \vee (\exists x, s_2. Set(s_2) \wedge s = \{x|s_2\})$
- $\neg \exists x, s. \{x|s\} = \{\}$
- $\forall x, s. (x \in s \Leftrightarrow \exists s_2. s = \{x|s_2\})$
- $\forall x, s. x \in s \Leftrightarrow [\exists y, s_2. (s = \{y|s_2\} \wedge (x = y \vee x \in s_2))]$
- $\forall s_1, s_2. s_1 \subseteq s_2 \Leftrightarrow (\forall x. x \in s_1 \Rightarrow x \in s_2)$
- $\forall s_1, s_2. (s_1 = s_2) \Leftrightarrow (s_1 \subseteq s_2 \wedge s_2 \subseteq s_1)$
- $\forall x, s_1, s_2. x \in (s_1 \cap s_2) \Leftrightarrow (x \in s_1 \wedge x \in s_2)$
- $\forall x, s_1, s_2. x \in (s_1 \cup s_2) \Leftrightarrow (x \in s_1 \vee x \in s_2)$

# Interacting with FOL KBs

- Suppose a wumpus-world agent is using an FOL KB and perceives a smell and a breeze (but no glitter) at  $t = 5$ :  
 $\text{Tell}(\text{KB}, \text{Percept}([\text{Smell}, \text{Breeze}, \text{None}], 5))$   
 $\text{Ask}(\text{KB}, \exists a. \text{BestAction}(a, 5))$ 
  - i.e., does the KB entail some best action at  $t = 5$ ?
  - Answer: Yes,  $\{a/\text{Shoot}\} \leftarrow \text{substitution}$  (binding list)
  - Given a sentence  $S$  and a substitution  $\sigma$ ,
  - $S\sigma$  denotes the result of “plugging”  $\sigma$  into  $S$ ; e.g.,  
 $S = \text{Smarter}(x, y)$   
 $\sigma = \{x/\text{Clinton}, y/\text{Trump}\}$   
 $S\sigma = \text{Smarter}(\text{Clinton}, \text{Trump})$
  - $\text{Ask}(\text{KB}, S)$  returns some or all  $\sigma$  such that  $\text{KB} \models S\sigma$

# Knowledge base for the wumpus world

- Perception
  - $\forall t, s, b. \text{Percept}([s, b, \text{Glitter}], t) \Rightarrow \text{Glitter}(t)$
- Reflex
  - $\forall t. \text{Glitter}(t) \Rightarrow \text{BestAction}(\text{Grab}, t)$

# Deducing hidden properties

- $\forall x, y, a, b. \text{Adjacent}([x, y], [a, b]) \Leftrightarrow [a, b] \in \{[x + 1, y], [x - 1, y], [x, y + 1], [x, y - 1]\}$
- $\forall s, t. \text{At}(\text{Agent}, s, t) \wedge \text{Breeze}(t) \Rightarrow \text{Breezy}(s)$
- Squares are breezy near a pit:
  - Diagnostic rule: infer cause from effect  
 $\forall s. \text{Breezy}(s) \Rightarrow \exists r. \text{Adjacent}(r, s) \wedge \text{Pit}(r)$
  - Causal rule: infer effect from cause  
 $\forall r. \text{Pit}(r) \Rightarrow [\forall s. \text{Adjacent}(r, s) \Rightarrow \text{Breezy}(s)]$

# Summary

- First-order logic:
  - objects and relations are semantic primitives
  - syntax: constants, functions, predicates, equality, quantifiers
- Increased expressive power: sufficient to define the Wumpus world

# Inf2D 09: CW1 (12.5%)

Michael Herrmann

University of Edinburgh, School of Informatics

02/02/2017

**informatics**



Credits: The content of this lecture was prepared by Michael Rovatsos and follows R&N

# Overview

- Constraint Satisfaction Problems (CSP)
- WalkSAT
- DPLL
- A bit of Haskell

- Understand inference and satisfiability problems in Propositional Logic
- Familiarize yourself with some algorithms for solving inference/SAT problems
- Implement satisfiability algorithms using Haskell

# Datatypes (t.b.r.)

- **Atom:** a.k.a. “variable”, e.g.  $P$ .
- **Literal:** a.k.a. “symbol”: either atoms (for example  $P$ ) or negations of atoms (for example  $\neg P$ ).
- **Clause:** A Clause is a disjunction of literals, for example  $P \vee Q \vee R \vee \neg S$ .
- **Formula:** a.k.a. “sentence”: conjunction of clauses, for example  $(P \vee Q) \wedge (R \vee P \vee \neg Q) \wedge (\neg P \vee \neg R)$ .
- **Assignment:** assignment of a truth value (True or False) to a particular Atom.
- **Model:** A (partial) Model is a (partial) assignment of truth values to the Atoms in a Formula (a list of Assignments).
- **Node:** The Node is the datatype describing a node in the search space
  - Formula
  - list of unassigned Atoms
  - Model containing the current variable assignments.

# Coursework Overview

- Task 1: General Helper Functions (10 marks)
- Task 2: WalkSAT (20 marks)
- Task 3: DPLL (30 Marks)
- Task 4: Improving the efficiency of DPLL (15 Marks)
- Task 5: Evaluation (15 Marks)
- Task 6: Report (10 Marks)

# Task 1: General Helper Functions (10 marks)

Functions defined in this task will be useful in the remaining tasks.

- `lookupAssignment::Symbol->Model->Maybe Bool`
- `negateSymbol::Symbol->Symbol`
- `isNegated::Symbol->Bool`
- `getUnsignedSymbol::Symbol->Symbol`
- `getSymbols::[Sentence]->[Symbol]`

## Task 2: WalkSAT (20 marks)

- Incomplete, local search algorithm
  - Evaluation function: The min-conflict heuristic of minimizing the number of unsatisfied clauses
  - Balance between greediness and randomness

# The WalkSAT algorithm

```
function WALKSAT(clauses, p, max-flips) returns a satisfying model or failure
    inputs: clauses, a set of clauses in propositional logic
            p, the probability of choosing to do a “random walk” move
            max-flips, number of flips allowed before giving up
    model  $\leftarrow$  a random assignment of true/false to the symbols in clauses
    for i = 1 to max-flips do
        if model satisfies clauses then return model
        clause  $\leftarrow$  a randomly selected clause from clauses that is false in model
        with probability p flip the value in model of a randomly selected symbol
            from clause
        else flip whichever symbol in clause maximizes the number of satisfied clauses
    return failure
```

Algorithm checks for satisfiability by randomly flipping the values of variables

## Task 3: DPLL (30 Marks)

Determine if an input propositional logic sentence (in CNF) is **satisfiable**.

**Improvements** over truth table enumeration:

- ① Early termination
- ② Pure symbol heuristic
- ③ Unit clause heuristic

# Early termination

- A clause is true if one of its literals is true,
  - e.g. if  $A$  is true then  $(A \vee \neg B)$  is true.
- A sentence is false if **any** of its clauses is false,
  - e.g. if  $A$  is false and  $B$  is true then  $(A \vee \neg B)$  is false, so sentence containing it is false.

# Pure symbol heuristic

- Pure symbol: always appears with the same “sign” or polarity in all clauses.
  - e.g., in the three clauses  $(A \vee \neg B)$ ,  $(\neg B \vee \neg C)$ ,  $(C \vee A)$   $A$  and  $B$  are pure,  $C$  is impure.
- Make literal containing a pure symbol true.
  - e.g. (for satisfiability) Let  $A$  and  $\neg B$  both be true

# Unit clause heuristic

**Unit clause:** only one literal in the clause, e.g.  $(A)$

- The only literal in a unit clause must be true.
  - e.g.  $A$  must be true.
- Also includes clauses where **all but one** literal is false,
  - e.g.  $(A, B, C)$  where  $B$  and  $C$  are false since it is equivalent to  $(A, \text{false}, \text{false})$  i.e.  $(A)$ .

# Tautology Deletion

- Tautology: both a proposition and its negation in a clause.
  - e.g.  $(A, B, \neg A)$
- Clause bound to be true.
  - e.g. whether  $A$  is true or false.
  - Therefore, can be deleted.
- Can be extended across clauses ...

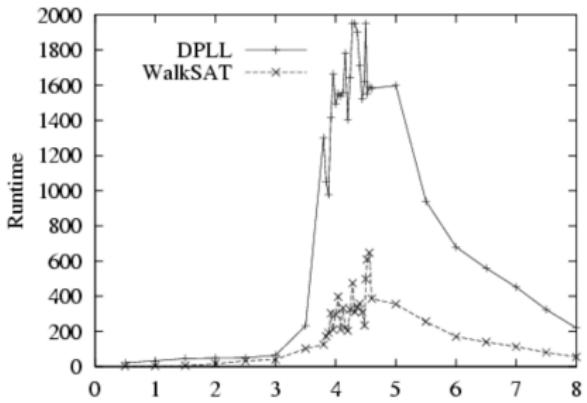
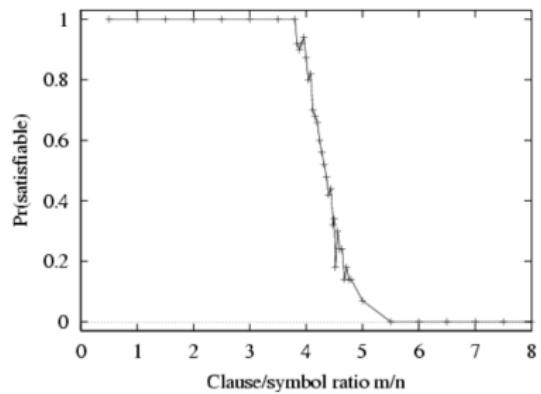
## Task 4: Improving DPLL (15 Marks)

- Implement an improved variable selection heuristic, i.e. a better choice function for dpll.
- Improvement is possible in various dimensions:
  - time, space, exploitation of bias or inhomogeneities
  - specify what is improved (most likely time, i.e. efficiency)
  - any drawbacks?
- Partial credit will be given for any relevant attempt at a solution for this part.

## Task 5: Evaluation (15 Marks)

- Study performance of WalkSAT for different  $p$
- Compare WalkSAT, DPLL, DPLLv2 based on sample inputs provided
- \*Compare WalkSAT, DPLL, DPLLv2 for different  $m/n$ .

# Hard satisfiability problems



## Task 6: Report (10 Marks)

You should write a brief report

- Either as a comment in your code (in the section provided at the end of the file)
- If you want to add figures, then submit the report as an additional pdf file (add a note in the section provided at the end of the file)
- You will answer specific questions discussing the results and your experiences in implementing the algorithms.

- Read the assignment sheet carefully before starting.
- Code clarity is important. Comment your code adequately.
- Reuse functions as much as possible.
- You can add your own helper functions, but you should explain why they are necessary in your comments.
- Test code and make sure they run before submitting.

# Help and support

- Coursework clinics: Fridays from 11am (FH 1.B31)
- Ask you tutor
- Piazza
- e-mail to lecturer

9th March 2017 @ 4pm

[Official start: 9th February]

Good Luck!!!

# Haskell Refresher

- Purely functional! : “Everything is a function”
- Main topics:
  - Recursion
  - Currying
  - Higher-order functions
  - List processing functions such as map, filter, foldl, sortBy, etc.
  - The Maybe monad
- More on Haskell:  
<https://wiki.haskell.org/Haskell>

# Types

- Unlike other programming languages like Java, Haskell has type inference.
- However, type declarations ensure that you are specific about the input arguments of your function and the output values.
- Example:  
`pLogicEvaluate :: Sentence -> Model -> Bool`
- The `pLogicEvaluate` function takes arguments of type `Sentence` and `Model` and returns a `Bool` type.

# Type Synonyms

```
type Symbol = String
```

```
type Model = [(Symbol, Bool)]
```

- The type `Symbol` is a synonym for `String`, and type `Model` is a synonym for a list of `(Symbol,Bool)` tuples.
- Types synonyms are good for code clarity.

# Recursion

- Important role in Haskell.
- A function is recursive when one part of its definition includes the function itself again.
- It is important to have a termination condition to avoid infinite loop.

```
length :: [a] -> Int
length [] = 0
length (x:xs) = 1 + length xs
```

# Currying

- The process of creating intermediate functions when feeding arguments into a complex function.
- Note: all functions in Haskell really only take one argument
- Example:  
 $2 * 3$  in Haskell:
  - (\*) function takes first argument 2, and returns an intermediate function (2\*)
  - The new function (2\*) takes one argument, 3, and completes the multiplication
- Applying only one parameter to a function that takes two parameters returns a function that takes one parameter

# Higher-Order Functions

- Functions are just like any other value in Haskell.
- Functions can take functions as parameters and also return functions.

```
map :: (a -> b) -> [a] -> [b]
map _ [] = []
map f (x:xs) = f x : map f xs
```

- Map takes a function and list and applies that function to every element in the list.

# List Processing Functions (map, filter, foldl, etc.)

- map: takes a function and list and applies that function to every element in the list.

`map :: (a -> b) -> [a] -> [b]`

- filter: takes a predicate (function that returns true or false) and list and then returns the list of all elements that satisfy the predicate.

`filter:: (a -> Bool) -> [a] -> [a]`

- foldl: takes a binary function, an accumulator and a list. It 'folds' up the items in the list and return a single value.

`foldl:: (a -> b -> a) -> a -> [b] -> a`

# List Comprehension

- Build more specific sets out of general sets
- Example: to create a list of integers that are multiples of 2 and greater than than 20:

```
[x*2 | x <- [1..25] , x*2 >= 20]
```

↑

↑

↑

Output  
function

Elements  
bound to x

Condition or  
Predicate

# Maybe Monad

- The Maybe monad represents computations which might “go wrong” by not returning a value.
- If a value is returned, it uses `Just a`, where `a` is the type of the value.
- If no value is available, it returns `Nothing`.
- Example:

```
safeDiv :: Double -> Double -> Maybe Double
safeDiv x y
  | y == 0 = Nothing
  | otherwise = Just (x/y)
```

# Inf2D 11: Unification and Generalised Modus Ponens

Michael Herrmann

University of Edinburgh, School of Informatics

09/02/2017

**informatics**



Credits: The content of this lecture was prepared by Michael Rovatsos and follows R&N

# Outline

- Reducing first-order inference to propositional inference
- Unification
- Generalized Modus Ponens

# Universal instantiation (UI)

- Every instantiation of a universally quantified formula  $\alpha$  is entailed by it:

$$\frac{\forall v.\alpha}{\alpha \{v/g\}}$$

for any variable  $v$  and ground term  $g$



Contains no variables

Example:

$\forall x.King(x) \wedge Greedy(x) \Rightarrow Evil(x)$  yields:

$King(John) \wedge Greedy(John) \Rightarrow Evil(John)$

$King(Richard) \wedge Greedy(Richard) \Rightarrow Evil(Richard)$

$King(Father(John)) \wedge Greedy(Father(John)) \Rightarrow Evil(Father(John))$

etc...

# Existential instantiation (EI)

- For any formula  $\alpha$ , variable  $v$ , and some constant symbol  $k$  that does **not** appear elsewhere in the knowledge base:

$$\frac{\exists v. \alpha}{\alpha \{v/k\}}$$

- Example.**  $\exists x. Crown(x) \wedge OnHead(x, John)$  yields:

$$Crown(C_1) \wedge OnHead(C_1, John)$$

- provided  $C_1$  is a new constant symbol, called a **Skolem constant**

# Reduction to propositional inference

- Suppose the KB contains just the following:

$\forall x. \text{King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x)$

$\text{King}(\text{John})$

$\text{Greedy}(\text{John})$

$\text{Brother}(\text{Richard}, \text{John})$

- Instantiating the universal sentence in all possible ways, we have:

$\text{King}(\text{John}) \wedge \text{Greedy}(\text{John}) \Rightarrow \text{Evil}(\text{John})$

$\text{King}(\text{Richard}) \wedge \text{Greedy}(\text{Richard}) \Rightarrow \text{Evil}(\text{Richard})$

$\text{King}(\text{John})$

$\text{Greedy}(\text{John})$

Note: universal sentence can then be discarded

$\text{Brother}(\text{Richard}, \text{John})$

- The new KB is **propositionalised**: proposition symbols are

$\text{King}(\text{John})$ ,  $\text{Greedy}(\text{John})$ ,  $\text{Evil}(\text{John})$ , **King(Richard)**, etc.

note: it's not in KB as a fact

## Reduction contd.

- Every FOL KB can be propositionalised so as to **preserve** entailment
  - A ground sentence is entailed by new KB iff entailed by original KB
- Idea: propositionalise KB and query, apply DPLL (or some other complete propositional method), return result
- Problem: with function symbols, there are infinitely many ground terms,
  - e.g., Father(Father(Father(John)))

## Reduction contd.

Theorem: Herbrand (1930). If a sentence  $\alpha$  is entailed by a FOL KB, it is entailed by a finite subset of the propositionalised KB

- Idea: For  $n = 0$  to  $\infty$  do  
create a propositional KB by instantiating with depth- $n$  terms see if  $\alpha$  is entailed by this KB
- Problem: works if  $\alpha$  is entailed, loops forever if  $\alpha$  is not entailed

Theorem: Turing (1936), Church (1936). Entailment for FOL is **semi-decidable** (i.e. algorithms exist that say yes to every entailed sentence, but no algorithm exists that also says no to every non-entailed sentence.)

# Problems with propositionalisation

- Propositionalisation seems to generate lots of irrelevant sentences.
- Example
  - From:  
 $\forall x. King(x) \wedge Greedy(x) \Rightarrow Evil(x)$   
 $King(John)$   
 $\forall y. Greedy(y)$   
 $Brother(Richard, John)$
  - It seems obvious that  $Evil(John)$ , but propositionalisation produces lots of facts such as  $Greedy(Richard)$  that are irrelevant
- With  $p$   $k$ -ary predicates and  $n$  constants, there are  $p \cdot k^n$  instantiations.

# Unification

- We can get the inference immediately if we can find a **substitution**  $\theta$  such that  $King(x)$  and  $Greedy(x)$  match  $King(John)$  and  $Greedy(y)$ 
  - $\theta = \{x/John, y/John\}$  works
- Unify  $(\alpha, \beta) - \theta$  iff  $\alpha\theta = \beta\theta$

$\alpha$	$\beta$	$\theta$
Knows(John,x)	Knows(John,Jane)	
Knows(John,x)	Knows(y,OJ)	
Knows(John,x)	Knows(y,Mother(y))	
Knows(John,x)	Knows(x, Richard)	

- Standardizing variables apart eliminates overlap of variables, e.g., Change  $Knows(x, Richard)$  to  $Knows(z_{17}, Richard)$  and then we succeed with  $\theta = \{z_{17}/John, x/Richard\}$  for the last case

# Unification

- We can get the inference immediately if we can find a **substitution**  $\theta$  such that  $King(x)$  and  $Greedy(x)$  match  $King(John)$  and  $Greedy(y)$ 
  - $\theta = \{x/John, y/John\}$  works
- Unification of two sentences means to find a substitution  $\theta$  such that the sentences become identical.
- Symbolically, this substitution is produced by the *Unify* operator:  $Unify(\alpha, \beta) = \theta$  iff  $\alpha\theta \equiv \beta\theta$

$\alpha$	$\beta$	$\theta$
Knows(John, $x$ )	Knows(John, Jane)	$\{x/Jane\}$
Knows(John, $x$ )	Knows( $y$ , OJ)	$\{x/OJ, y/John\}$
Knows(John, $x$ )	Knows( $y$ , Mother( $y$ ))	$\{y/John, x/Mother(John)\}$
Knows(John, $x$ )	Knows( $x$ , Richard)	[fail]

# Standardizing variables apart

- We cannot unify the sentences  $\text{Knows}(\text{John}, x)$  and  $\text{Knows}(x, \text{Richard})$  if the two “ $x$ ” are referring to the same variable.
- If the two “ $x$ ” are unrelated and just due to inconsiderate naming of an unknown, then we can rename one of them and avoid thus the clash of variables. This is called:
- Standardizing variables apart eliminates overlap of variables, e.g., change  $\text{Knows}(x, \text{Richard})$  to  $\text{Knows}(z_{17}, \text{Richard})$  and then we succeed with  $\theta = \{z_{17}/\text{John}, x/\text{Richard}\}$  for the above example.

# Unification

- To unify  $\text{Knows}(\text{John}, x)$  and  $\text{Knows}(y, z)$ ,  
 $\theta = \{y/\text{John}, x/z\}$  or  $\theta = \{y/\text{John}, x/\text{John}, z/\text{John}\}$
- The first unifier is **more general** than the second.
- FOL: There is a single **most general unifier** (MGU) that is unique up to renaming of variables.
  - $\text{MGU} = \{y/\text{John}, x/z\}$
- Can be viewed as an **equation solving** problem.
  - i.e. solve  $\text{Knows}(\text{John}, x) \equiv \text{Knows}(y, z)$

## Example

What is the most general unifier, if any, of the following pairs of formulae?

- $\text{Loves}(\text{John}, x) \equiv \text{Loves}(y, \text{Mother}(y))$ .
- $\text{Loves}(\text{John}, \text{Mother}(x)) \equiv \text{Loves}(y, y)$ .

# Solution

- $\text{Loves}(\text{John}, x) \equiv \text{Loves}(y, \text{Mother}(y)).$ 
  - $\{x/\text{Mother}(\text{John}), y/\text{John}\}$
- $\text{Loves}(\text{John}, \text{Mother}(x)) \equiv \text{Loves}(y, y).$ 
  - Fails

# Finding the MGU

- Can be broken-down into a series of steps
  - Decomposition
  - Conflict
  - Eliminate
  - Delete
  - Switch
  - Coalesce
  - Occurs Check
- Other presentations of algorithm are possible (see R&N)

# Decomposition

- Given:  $\text{Knows}(\text{John}, x) \equiv \text{Knows}(y, z)$ .
- Replace with  $\text{John} \equiv y$  and  $x \equiv z$ .
- In general, given

$$f(s_1, \dots, s_n) \equiv f(t_1, \dots, t_n)$$

- Replace with  $s_1 \equiv t_1, \dots, s_n \equiv t_n$ .

# Conflict

- Given:  $\text{Knows}(\text{John}, x) \equiv \text{Greedy}(y)$ .
- Then fail.

- In general, given:

$$f(s_1, \dots, s_m) \equiv g(t_1, \dots, t_n), \text{ where } f \neq g$$

- Then fail.

## Eliminate

- Given:  $\text{Knows}(\text{John},x) \equiv \text{Knows}(y,z)$  and  $z \equiv \text{Richard}$ .
- Replace with  $\text{Knows}(\text{John},x) \equiv \text{Knows}(y,\text{Richard})$  and  $z \equiv \text{Richard}$ .
- In general, given:  $P$  and  $x \equiv t$ , where  $x$  occurs in  $P$  but not in  $t$ , and  $t$  is not a variable
- Replace with  $P\{x/t\}$  and  $x \equiv t$ .

# Delete

- Given:  $\text{Greedy}(\text{John}) \equiv \text{Greedy}(\text{John})$ .
  - Remove this equation.
- 
- In general, given  $P$  and  $s \equiv s$
  - Then replace with  $P$ .

- Given:  $\text{Knows}(\text{John},x) \equiv \text{Knows}(y,z)$  and  $\text{Richard} \equiv z$ .
- Replace with  $\text{Knows}(\text{John},x) \equiv \text{Knows}(y,z)$  and  $z \equiv \text{Richard}$ .
- In general, given:  $P$  and  $s \equiv x$ , where  $x$ , but not  $s$ , is a variable
- Replace with  $P$  and  $x \equiv s$ .

- Given:  $\text{Knows}(\text{John},x) \equiv \text{Knows}(y,z)$  and  $y \equiv z$ .
  - Replace with  $\text{Knows}(\text{John},x) \equiv \text{Knows}(z,z)$  and  $y \equiv z$ .
- 
- In general, given  $P$  and  $x \equiv y$ , where  $x$  and  $y$  are variables occurring in  $P$ .
  - Replace with  $P\{x/y\}$  and  $x \equiv y$ .

## Occurs Check

- Given:  $x \equiv \text{Father}(x)$ .
- Then fail, else eliminate will loop.
  - $P(x)$  and  $x \equiv \text{Father}(x) \mapsto P(\text{Father}(\text{Father}(\dots)))$ .
- In general, given  $x \equiv s$ , where  $x$  occurs in  $s$  and  $s$  is not a variable
- Then fail.

# Example

$$\text{Loves}(\text{John},x) \equiv \text{Loves}(y,\text{Mother}(y))$$

↓ Decompose

$$\text{John} \equiv y \wedge x \equiv \text{Mother}(y)$$

↓ Switch

$$y \equiv \text{John} \wedge x \equiv \text{Mother}(y)$$

↓ Eliminate

$$y \equiv \text{John} \wedge x \equiv \text{Mother}(\text{John})$$

# Generalized Modus Ponens (GMP)

$$\frac{p'_1, p'_2, \dots, p'_n (p_1 \wedge p_2 \wedge \dots \wedge p_n \Rightarrow q)}{q\theta} \text{ when } p'_i\theta \equiv p_i\theta \forall i$$

Example       $p'_1$  is King (John)       $p_1$  is King( $x$ )  
                 $p'_2$  is Greedy( $y$ )       $p_2$  is Greedy( $x$ )  
                 $\theta$  is ( $x/John, y/John$ )       $q$  is Evil( $x$ )  
                 $q\theta$  is Evil(John)

- GMP used with KB of **definite clauses** (exactly one positive literal)
- All variables assumed universally quantified

# Soundness of GMP

- Need to show that

$$p'_1, \dots, p'_n, (p_1 \wedge \dots \wedge p_n \Rightarrow q) \models q\theta$$

provided that  $p'_i\theta = p_i\theta$  for all  $i$

- Lemma: For any sentence  $p$ , we have  $p \models p\theta$  by UI
  - ①  $(p_1 \dots p_n \Rightarrow q) \models (p_1 \dots p_n \Rightarrow q)\theta = (p_1\theta \dots p_n\theta \Rightarrow q\theta)$
  - ②  $p'_1, \dots, p'_n \models p'_1 \dots p'_n \models p'_1\theta \dots p'_n\theta = p_1\theta \dots p_n\theta$   
since by definition of GMP  $p'_i\theta = p_i\theta$  for all  $i$
  - ③ From 1 and 2,  $q\theta$  follows by ordinary Modus Ponens

# Example Knowledge Base

- The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.
- Prove that Colonel West is a criminal

## Example Knowledge Base (cntd.)

... it is a crime for an American to sell weapons to hostile nations:

$$\text{American}(x) \wedge \text{Weapon}(y) \wedge \\ \text{Sells}(x,y,z) \wedge \text{Hostile}(z) \Rightarrow \text{Criminal}(x)$$

Nono ... has some missiles, i.e.,  $\exists x \text{ Owns}(\text{Nono},x) \wedge \text{Missile}(x)$ : ...

$$\text{Owns}(\text{Nono},M_1) \text{ and } \text{Missile}(M_1)$$

... all of its missiles were sold to it by Colonel West

$$\text{Missile}(x) \wedge \text{Owns}(\text{Nono},x) \Rightarrow \text{Sells}(\text{West},x,\text{Nono})$$

Missiles are weapons:

$$\text{Missile}(x) \Rightarrow \text{Weapon}(x)$$

An enemy of America counts as "hostile":

$$\text{Enemy}(x,\text{America}) \Rightarrow \text{Hostile}(x)$$

West, who is American ...

$$\text{American}(\text{West})$$

The country Nono, an enemy of America ...

$$\text{Enemy}(\text{Nono},\text{America})$$

# Summary

- Rules for quantifiers.
- Reducing FOL to PL.
- Unification as equation solving.
- Generalized modus ponens

# Inf2D 13: Resolution-Based Inference

Michael Herrmann

University of Edinburgh, School of Informatics

14/02/2017

**informatics**



# Last lecture: Forward and backward chaining

- Backward chaining:
    - If Goal is known  
(goal directed)
    - Can query for data
  - Forward chaining:
    - If specific Goal is not known, but system needs to react to new facts (data driven)
    - Can make suggestions
  - Also relevant:
    - What do users expect from the system?
    - Which direction has the larger branching factor?
- Combination of forward and backward chaining:  
Opportunistic reasoning

# Limitations (due to restriction to definite clauses)

In order to apply GMP:

- Premises of all rules contain only non-negated symbols
- Conclusions of all rules is a non-negated symbol
- Facts are non-negated propositions

Possible solution: Introduce more variables, e.g.  $Q := \neg P$

What about: "If we cannot prove  $A$ , then  $\neg A$  is true"?

(works only if there is a rule for each variable)

# Last lecture: Resolution

- Negate query  $\alpha$
- Convert everything to CNF
- Repeat: Choose clauses and resolve (based on unification)
- If resolution results in empty clause,  $\alpha$  is proved
- Return all substitutions (or Fail)

## Mid-lecture question

- Is the following sentence unsatisfiable?

$$P(x, t) , \quad \neg P(s, x)$$

- with  $x$ : variable,  $s \neq t$ : constants

# Answer

- Is the following set of clauses unsatisfiable?

$$P(x, t) , \quad \neg P(s, x)$$

- Not necessarily, if this was a propositionalisation of

$$\forall x. (P(x, t) \wedge \neg P(s, x))$$

- It is unsatisfiable, if this was a propositionalisation of

$$(\forall x. P(x, t)) , \quad (\forall x. \neg P(s, x))$$

Here we can standardise-away the clash of the variablex (which we should have done in the first place!) and obtain a contradiction by instantiation.

# Binary Resolution and Modus Ponens

Ground binary resolution

$$\frac{C \vee P \quad D \vee \neg P}{C \vee D}$$

Suppose  $C = \text{False}$

$$\frac{P \quad \neg P \vee D}{D}$$

i.e.  $P$  and  $P \Rightarrow D$  entails  $D$ .

Modus ponens is a special case of binary resolution.

# Full Resolution and Generalised Modus Ponens

GMP with  $p'_i\theta \equiv p_i\theta \forall i$

$$\frac{p'_1, \dots, p'_n \quad (p_1 \wedge \dots \wedge p_n \Rightarrow q)}{q\theta}$$

$$\frac{p'_1, \dots, p'_n \quad q \vee \neg p_1 \vee \dots \vee \neg p_n}{q\theta}$$

Full resolution with  $\theta$  MGU of all  $P_i$  and  $P'_i\theta \equiv P_i\theta \forall i$

$$\frac{C \vee P_1 \vee \dots \vee P_m \quad D \vee \neg P'_1 \vee \dots \vee \neg P'_n}{(C \vee D)\theta}$$

# Resolution in Implication Form

Ground binary resolution

$$\frac{C \vee P \quad D \vee \neg P}{C \vee D}$$

Set  $C = \neg A$

$$\frac{A \Rightarrow P \quad P \Rightarrow D}{A \Rightarrow D}$$

## Example: Quacks and Doctors

- Some patients like all doctors

$$F_1 \equiv \exists x. P(x) \wedge \forall y. D(y) \Rightarrow Likes(x, y)$$

- No patient likes any quack

$$F_2 \equiv \forall x. P(x) \wedge \forall y. Q(y) \Rightarrow \neg Likes(x, y)$$

- Show: No doctor is a quack

$$F \equiv \forall x. D(x) \Rightarrow \neg Q(x)$$

# Example: Quacks and Doctors

CNF

$$\begin{aligned}F_1 &\equiv \exists x. P(x) \wedge \forall y. D(y) \Rightarrow Likes(x, y) \\&\equiv \exists x. P(x) \wedge \forall y. \neg D(y) \vee Likes(x, y)\end{aligned}$$

$$\begin{aligned}F_2 &\equiv \forall x. P(x) \wedge \forall y. Q(y) \Rightarrow \neg Likes(x, y) \\&\equiv \forall x. P(x) \wedge \forall y. \neg Q(y) \vee \neg Likes(x, y)\end{aligned}$$

$$\begin{aligned}F &\equiv \forall x. D(x) \Rightarrow \neg Q(x) \\&\equiv \forall x. \neg D(x) \vee \neg Q(x)\end{aligned}$$

# Example: Quacks and Doctors

## Propositionalisation

$$\begin{aligned} F_1 &\equiv \exists x. P(x) \wedge \forall y. \neg D(y) \vee Likes(x, y) \\ &\Rightarrow P(G) \wedge (\neg D(y) \vee Likes(G, y)) \end{aligned}$$

$$\begin{aligned} F_2 &\equiv \forall x. P(x) \wedge \forall y. \neg Q(y) \vee \neg Likes(x, y) \\ &\Rightarrow P(w) \wedge \neg Q(z) \vee \neg Likes(w, z) \end{aligned}$$

$$\begin{aligned} F &\equiv \forall x. \neg D(x) \vee \neg Q(x) \\ &\equiv \neg D(x) \vee \neg Q(x) \end{aligned}$$

# Example: Quacks and Doctors

## Unification

$$F'_1 \equiv P(G) \wedge \neg D(y) \vee Likes(G, y)$$

$$F'_2 \equiv P(w) \wedge \neg Q(z) \vee \neg Likes(w, z)$$

$$w/G \Rightarrow P(G) \wedge \neg Q(z) \vee \neg Likes(G, z)$$

## Negation of proof goal

$$\neg(\neg D(x) \vee \neg Q(x)) \equiv D(x) \wedge Q(x)$$

# Example: Quacks and Doctors

## Resolution

$$\begin{aligned} P(G) \wedge \neg D(y) \vee Likes(G, y), \\ P(G) \wedge \neg Q(z) \vee \neg Likes(G, z), \\ D(x) \wedge Q(x) \end{aligned}$$

Clauses:  $P(G)$ ,  $D(y)$ ,  $Q(z)$ ,  
 $\neg D(y) \vee Likes(G, y)$ ,  $\neg Q(z) \vee \neg Likes(G, z)$

$$\frac{\neg D(y) \vee Likes(G, z) \quad \neg Q(z) \vee \neg Likes(G, z)}{\neg D(y) \vee \neg Q(z)}$$

Substitute  $y/x$  and  $z/x$

$$\frac{\neg D(x) \vee \neg Q(x) \quad D(x)}{\neg Q(x)} \text{ and } \frac{\neg Q(x) \quad Q(x)}{\square}$$

Therefore:  $\neg D(x) \vee \neg Q(x)$ , i.e.  $D(x) \Rightarrow \neg Q(x)$  for any  $x$  such that  $\forall x. D(x) \Rightarrow \neg Q(x)$

## Example: Quacks and Doctors (Variant)

- Some patients like all doctors

$$F_1 \equiv \exists x. P(x) \wedge \forall y. D(y) \Rightarrow Likes(x, y)$$

- No patient likes any quack

$$F_2 \equiv \forall x. P(x) \xrightarrow{\text{red}} \forall y. Q(y) \Rightarrow \neg Likes(x, y)$$

- Show: No doctor is a quack

$$F \equiv \forall x. D(x) \Rightarrow \neg Q(x)$$

# Example: Quacks and Doctors (Variant)

CNF

$$\begin{aligned}F_1 &\equiv \exists x. P(x) \wedge \forall y. D(y) \Rightarrow Likes(x, y) \\&\equiv \exists x. P(x) \wedge \forall y. \neg D(y) \vee Likes(x, y)\end{aligned}$$

$$\begin{aligned}F_2 &\equiv \forall x. P(x) \Rightarrow \forall y. Q(y) \Rightarrow \neg Likes(x, y) \\&\equiv \forall x. \neg P(x) \vee \forall y. \neg Q(y) \vee \neg Likes(x, y)\end{aligned}$$

$$\begin{aligned}F &\equiv \forall x. D(x) \Rightarrow \neg Q(x) \\&\equiv \forall x. \neg D(x) \vee \neg Q(x)\end{aligned}$$

# Example: Quacks and Doctors (Variant)

## Propositionalisation

$$\begin{aligned} F_1 &\equiv \exists x. P(x) \wedge \forall y. \neg D(y) \vee Likes(x, y) \\ &\Rightarrow P(G) \wedge (\neg D(y) \vee Likes(G, y)) \end{aligned}$$

$$\begin{aligned} F_2 &\equiv \forall x. \neg P(x) \vee \forall y. \neg Q(y) \vee \neg Likes(x, y) \\ &\Rightarrow \neg P(w) \vee \neg Q(z) \vee \neg Likes(w, z) \end{aligned}$$

$$\begin{aligned} F &\equiv \forall x. \neg D(x) \vee \neg Q(x) \\ &\equiv \neg D(x) \vee \neg Q(x) \end{aligned}$$

# Example: Quacks and Doctors (Variant)

## Unification

$$F'_1 \equiv P(G) \wedge \neg D(y) \vee Likes(G, y)$$

$$\begin{aligned} F'_2 &\equiv \neg P(w) \vee \neg Q(z) \vee \neg Likes(w, z) \\ w/G &\Rightarrow \neg P(G) \vee \neg Q(z) \vee \neg Likes(G, z) \end{aligned}$$

Negation of proof goal

$$\neg(\neg D(x) \vee \neg Q(x)) \equiv D(x) \wedge Q(x)$$

# Example: Quacks and Doctors (Variant)

Resolution

$$\begin{aligned} P(G) \wedge \neg D(y) \vee \text{Likes}(G, y), \\ \neg P(G) \vee \neg Q(z) \vee \neg \text{Likes}(G, z), \\ D(x) \wedge Q(x) \end{aligned}$$

Clauses:  $P(G)$ ,  $D(y)$ ,  $Q(z)$ ,

$$\neg D(y) \vee \text{Likes}(G, y), \quad \neg P(G) \vee \neg Q(z) \vee \neg \text{Likes}(G, z)$$

$$\frac{P(G) \quad \neg P(G) \vee \neg Q(z) \vee \neg \text{Likes}(G, z)}{\neg Q(z) \vee \neg \text{Likes}(G, z)}$$

$$\frac{\neg D(y) \vee \text{Likes}(G, z) \quad \neg Q(z) \vee \neg \text{Likes}(G, z)}{\neg D(y) \vee \neg Q(z)}$$

Substitute  $y/x$  and  $z/x$

$$\frac{\neg D(x) \vee \neg Q(x) \quad D(x)}{\neg Q(x)} \quad \text{and} \quad \frac{\neg Q(x) \quad Q(x)}{\square}$$

Therefore:  $\neg D(x) \vee \neg Q(x)$ , i.e.  $D(x) \Rightarrow \neg Q(x)$  for any  $x$  such that  $\forall x. D(x) \Rightarrow \neg Q(x)$

# Resolution: Soundness and completeness

- Resolution is sound and complete: If a set of clauses is unsatisfiable, then one can derive an empty clause from this set.
- Soundness is evident since the conclusion of any inference rule is a logical consequence of its premises.
- Completeness can be proved using completeness of propositional resolution and [lifting](#) (see the following slides, but full proof is beyond the scope of this course).

# Reminder: Factoring

$$\frac{A \vee B \quad A \vee \neg B}{A \vee A}$$

More generally

$$\frac{C \vee P_1 \vee \dots \vee P_m}{(C \vee P_1) \theta}$$

where  $\theta$  is the MGU of the  $P_i$

**Soundness:** by universal instantiation and deletion of duplicates.

# Resolvent

Let  $C_1$  and  $C_2$  be two clauses. A resolvent is a

- binary resolvent of  $C_1$  and  $C_2$
- binary resolvent of a factor of  $C_1$  and  $C_2$
- binary resolvent of  $C_1$  and a factor of  $C_2$
- binary resolvent of a factor of  $C_1$  and a factor of  $C_2$

## Factoring example

$$C_1 = P(x) \vee P(f(y)) \vee R(g(y))$$

$$C_2 = \neg P(f(g(a))) \vee Q(b)$$

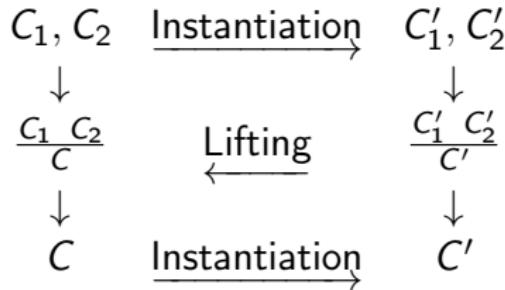
$C'_1 = P(f(y)) \vee R(g(y))$  is a factor of  $C_1$

$C = R(g(g(a))) \vee Q(b)$  is a resolvent of  $C'_1$  and  $C_2$  and thus a resolvent of  $C_1$  and  $C_2$ .

Choose  $\theta = \{x/f(y), y/g(a)\}$

# The lifting lemma

If  $C_1$  and  $C_2$  are two clauses with no shared variables  
and  $C'_1$  and  $C'_2$  their respective instances  
and  $C'$  is a resolvent of  $C'_1$  and  $C'_2$ ,  
then there exists a clause  $C$  such that both  
 $C$  is a resolvent of  $C_1$  and  $C_2$  and  
 $C'$  is an instance of  $C$ .



# Lifting lemma: Example

- Clauses
  - $C_1 = P(x) \vee Q(x)$
  - $C_2 = \neg P(f(y)) \vee R(y)$
- Resolving  $C_1$  and  $C_2$  gives  $C = Q(f(y)) \vee R(y)$
- Instances:
  - $C'_1 = P(f(a)) \vee Q(f(a))$  using  $x/f(a)$
  - $C'_2 = \neg P(f(a)) \vee R(a)$  using  $y/a$
- Resolving  $C'_1$  and  $C'_2$  gives  $C' = Q(f(a)) \vee R(a)$
- $C'$  is an instance of  $C$  as predicted by the Lifting Lemma!

# Remark: Resolution for 3SAT

$$\underbrace{(X_i \vee \neg X_j \vee X_k) \wedge (X_j \vee \neg X_r \vee X_s) \wedge \cdots \wedge (\neg X_k \vee X_l \vee \neg X_m)}_{M \text{ clauses}}$$

with  $i, j, k, r, s, l, m, \dots \in \{1, \dots, N\}$

Assuming  $X_i = 0$ :

$$\frac{\neg X_i \quad X_i \vee \neg X_j \vee X_k}{\neg X_j \vee X_k}$$

(we can't really use the  
(negated) goal at this stage)

First two clauses:

$$\frac{\neg X_j \vee X_k \quad X_j \vee \neg X_r \vee X_s}{X_k \vee \neg X_r \vee X_s}$$

More assumptions? More symbols per clause?

# Efficient algorithms for resolution

Heuristics to make resolution more efficient (compare DPLL!)

- Unit preference: Prefer clauses with only one symbol
- Pure clauses: Pure clauses contains symbol  $A$  which does not occur in any other clause: Cannot lead to contradiction.
- Tautology: Clauses containing  $A$  and  $\neg A$
- Set of support: Identify “useful” rules and ignore the rest.
- Input resolution: Intermediately generated sentences can only be combined with original inputs or original rules.
- Subsumption: If a clause contains another one, use only the shorter clause. Prune unnecessary facts from the database.

Including heuristics, resolution is more efficient than DPLL.

# Summary

- Forward chaining
- Backward chaining
- Resolution

# Inf2D 12: Resolution-Based Inference

Michael Herrmann

University of Edinburgh, School of Informatics

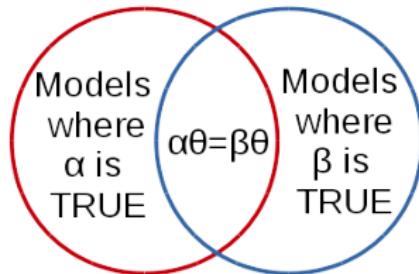
10/02/2017

**informatics**



Credits: The content of this lecture was prepared by Michael Rovatsos and follows R&N

- Unification: Given  $\alpha$  and  $\beta$ , find  $\theta$  such that  $\alpha\theta = \beta\theta$
- Most general unifier (MGU)



- Generalised Modus Ponens

$$\frac{p'_1, p'_2, \dots, p'_n (p_1 \wedge p_2 \wedge \dots \wedge p_n \Rightarrow q)}{q\theta} \text{ when } p'_i\theta \equiv p_i\theta \forall i$$

# Outline

- Forward chaining
- Backward chaining
- Resolution

# Forward chaining algorithm

```
function FOL-FC-ASK( $KB, \alpha$ ) returns a substitution or false
    inputs:  $KB$ , the knowledge base, a set of first-order definite clauses
            $\alpha$ , the query, an atomic sentence
    local variables:  $new$ , the new sentences inferred on each iteration

    repeat until  $new$  is empty
         $new \leftarrow \{ \}$ 
        for each rule in  $KB$  do
             $(p_1 \wedge \dots \wedge p_n \Rightarrow q) \leftarrow \text{STANDARDIZE-VARIABLES}(rule)$ 
            for each  $\theta$  such that  $\text{SUBST}(\theta, p_1 \wedge \dots \wedge p_n) = \text{SUBST}(\theta, p'_1 \wedge \dots \wedge p'_n)$ 
                for some  $p'_1, \dots, p'_n$  in  $KB$ 
                     $q' \leftarrow \text{SUBST}(\theta, q)$                                      ← Pattern-matching
                    if  $q'$  does not unify with some sentence already in  $KB$  or  $new$  then
                        add  $q'$  to  $new$                                          ←
                         $\phi \leftarrow \text{UNIFY}(q', \alpha)$                          Facts irrelevant to the goal can be generated
                        if  $\phi$  is not fail then return  $\phi$ 
                    add  $new$  to  $KB$ 
    return false
```

# 'West' Knowledge Base

$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$

$Owns(Nono, M_1)$  and  $Missile(M_1)$

$Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$

$Missile(x) \Rightarrow Weapon(x)$

$Enemy(x, America) \Rightarrow Hostile(x)$

$American(West)$  and  $Enemy(Nono, America)$

# Forward chaining proof

$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$

*Owns(Nono, M<sub>1</sub>) and Missile(M<sub>1</sub>)*

$Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$

$Missile(x) \Rightarrow Weapon(x)$

$Enemy(x, America) \Rightarrow Hostile(x)$

*American(West) and Enemy(Nono, America)*

American(West)

Missile(M<sub>1</sub>)

Owns(Nono, M<sub>1</sub>)

Enemy(Nono, America)

# Forward chaining proof

$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$

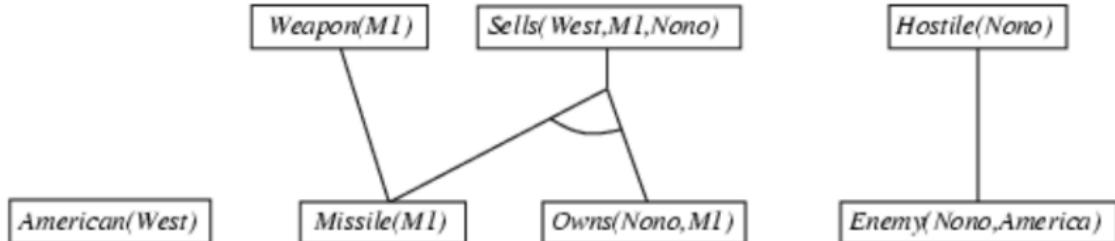
$Owns(Nono, M_1)$  and  $Missile(M_1)$

$Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$

$Missile(x) \Rightarrow Weapon(x)$

$Enemy(x, America) \Rightarrow Hostile(x)$

$American(West)$  and  $Enemy(Nono, America)$



# Forward chaining proof

$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$

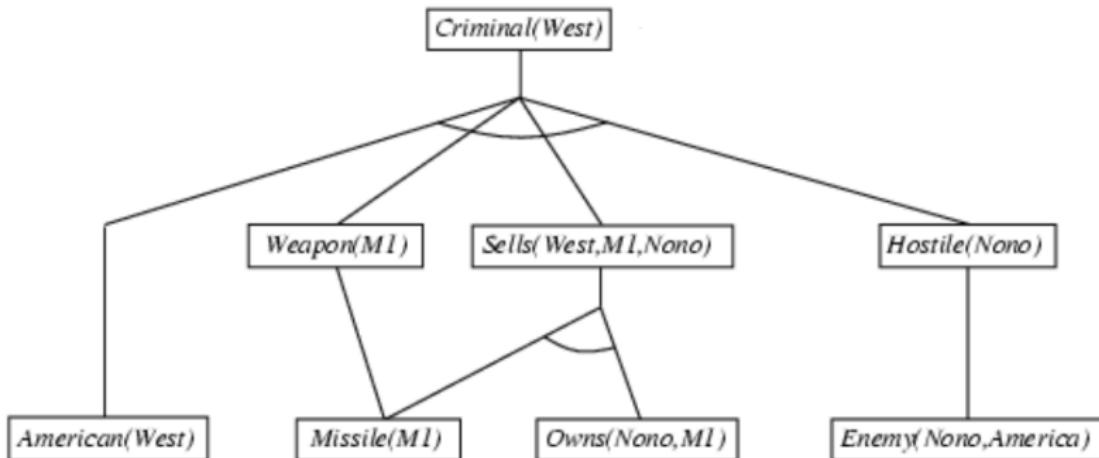
$Owns(Nono, M_1)$  and  $Missile(M_1)$

$Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$

$Missile(x) \Rightarrow Weapon(x)$

$Enemy(x, America) \Rightarrow Hostile(x)$

$American(West)$  and  $Enemy(Nono, America)$



# Properties of forward chaining II

- Sound and complete for first-order definite clauses
  - Definite clause = exactly one positive literal.
- Datalog = first-order definite clauses + no functions
  - FC terminates for Datalog in finite number of iterations
- May not terminate in general if  $\alpha$  is not entailed
- This is unavoidable: entailment with definite clauses is semi-decidable

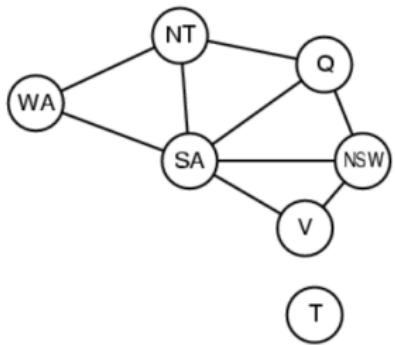
# Efficiency of forward chaining I

- Incremental forward chaining: no need to match a rule on iteration  $k$  if a premise wasn't added on iteration  $k - 1$   
⇒ match each rule whose premise contains a newly added positive literal
- Matching itself can be expensive:  
**Database indexing** allows  $O(1)$  retrieval of known facts
  - e.g., query  $\text{Missile}(x)$  retrieves  $\text{Missile}(M1)$
- Forward chaining is widely used in **deductive databases**

# Efficiency of forward chaining II

- Pattern Matching
  - For each  $\theta$  such that
$$SUBST(\theta, p_1 \wedge \cdots \wedge p_n) = SUBST(\theta, p'_1 \wedge \cdots \wedge p'_n)$$
 for some  $p'_1, \dots, p'_n$  in KB
  - Finding all possible unifiers can be very expensive
- Example
  - $Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$
  - Can find each object owned by Nono in constant time and then check if it is admissible
  - But what if Nono owns many objects but very few missiles?
  - Conjunct Ordering: Better (cost-wise) to find all missiles first and then check whether they are owned by Nono
  - Optimal ordering is NP-hard. Heuristics available: e.g. MRV from CSP if each conjunct is viewed as a constraint on its vars

# Hard matching example



$Diff(wa, nt) \wedge Diff(wa, sa) \wedge Diff(nt, q) \wedge Diff(nt, sa) \wedge Diff(q, nsw) \wedge Diff(q, sa) \wedge Diff(nsw, v) \wedge Diff(nsw, sa) \wedge Diff(v, sa) \Rightarrow Colourable$

$Diff(Red, Blue)$   $Diff(Blue, Green)$   
 $Diff(Green, Red)$   $Diff(Green, Blue)$   
 $Diff(Blue, Red)$   $Diff(Blue, Green)$

- Every finite domain CSP can be expressed as a single definite clause + ground facts
- Colourable is inferred iff the CSP has a solution
- CSPs include 3SAT as a special case, hence matching is NP-hard

# Backward chaining algorithm

A function that returns multiple times, each time giving one possible result

```
function FOL-BC-ASK( $KB$ ,  $query$ ) returns a generator of substitutions
    return FOL-BC-OR( $KB$ ,  $query$ , { })

generator FOL-BC-OR( $KB$ ,  $goal$ ,  $\theta$ ) yields a substitution
    for each rule ( $lhs \Rightarrow rhs$ ) in FETCH-RULES-FOR-GOAL( $KB$ ,  $goal$ ) do
        ( $lhs$ ,  $rhs$ )  $\leftarrow$  STANDARDIZE-VARIABLES(( $lhs$ ,  $rhs$ ))
        for each  $\theta'$  in FOL-BC-AND( $KB$ ,  $lhs$ , UNIFY( $rhs$ ,  $goal$ ,  $\theta$ )) do
            yield  $\theta'$ 

generator FOL-BC-AND( $KB$ ,  $goals$ ,  $\theta$ ) yields a substitution
    if  $\theta = failure$  then return
    else if LENGTH( $goals$ ) = 0 then yield  $\theta$ 
    else do
         $first, rest \leftarrow FIRST(goals), REST(goals)$ 
        for each  $\theta'$  in FOL-BC-OR( $KB$ , SUBST( $\theta$ ,  $first$ ),  $\theta$ ) do
            for each  $\theta''$  in FOL-BC-AND( $KB$ ,  $rest$ ,  $\theta'$ ) do
                yield  $\theta''$ 
```

Fetch rules that might unify

$$SUBST(COMPOSE(\theta_1, \theta_2), p) = SUBST(\theta_2, SUBST(\theta_1, p))$$

# Backward chaining example

$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$

$Owns(Nono, M_1)$  and  $Missile(M_1)$

$Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$

$Missile(x) \Rightarrow Weapon(x)$

$Enemy(x, America) \Rightarrow Hostile(x)$

$American(West)$  and  $Enemy(Nono, America)$

$Criminal(West)$

# Backward chaining example

$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$

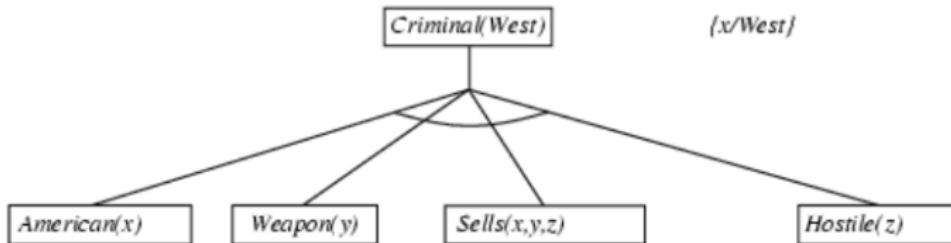
$Owns(Nono, M_1) \text{ and } Missile(M_1)$

$Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$

$Missile(x) \Rightarrow Weapon(x)$

$Enemy(x, America) \Rightarrow Hostile(x)$

$American(West) \text{ and } Enemy(Nono, America)$



# Backward chaining example

$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$

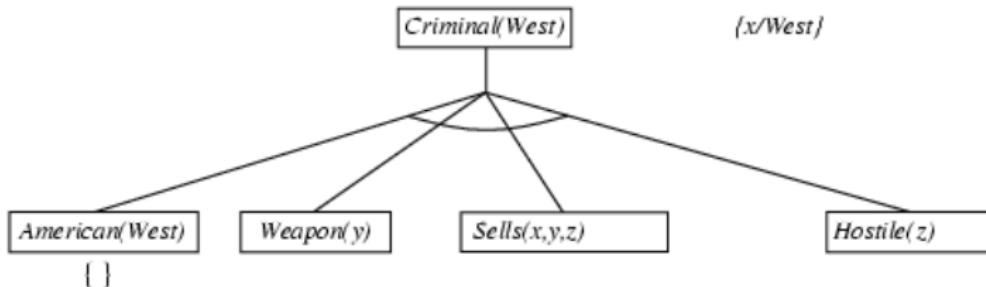
$Owns(Nono, M_1) \text{ and } Missile(M_1)$

$Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$

$Missile(x) \Rightarrow Weapon(x)$

$Enemy(x, America) \Rightarrow Hostile(x)$

**American(West)** and **Enemy(Nono, America)**



# Backward chaining example

$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$

$Owns(Nono, M_1)$  and  $Missile(M_1)$

$Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$

$\text{Missile}(x) \Rightarrow \text{Weapon}(x)$

$Enemy(x, America) \Rightarrow Hostile(x)$

$American(West)$  and  $Enemy(Nono, America)$



# Backward chaining example

$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$

Owns(Nono, M<sub>1</sub>) and **Missile(M<sub>1</sub>)**

$Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$

$Missile(x) \Rightarrow Weapon(x)$

$Enemy(x, America) \Rightarrow Hostile(x)$

$American(West) \text{ and } Enemy(Nono, America)$



# Backward chaining example

$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$

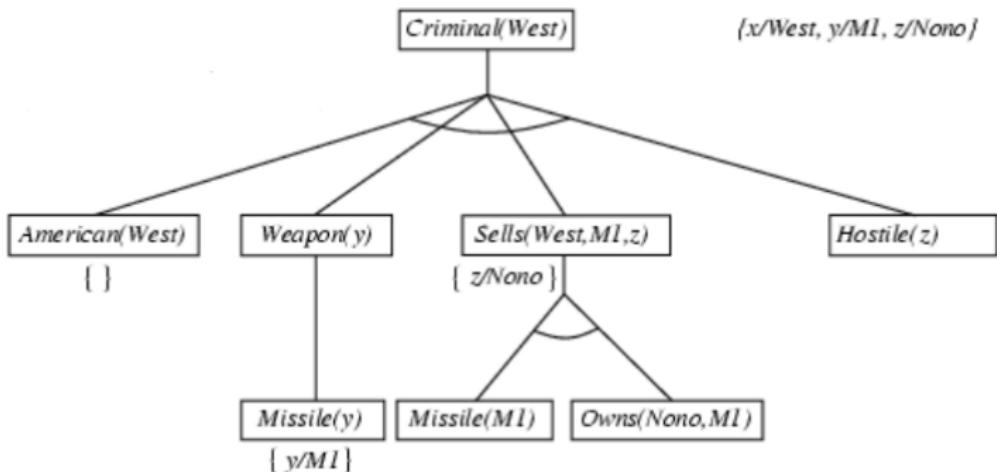
$Owns(Nono, M_1)$  and  $Missile(M_1)$

$Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$

$Missile(x) \Rightarrow Weapon(x)$

$Enemy(x, America) \Rightarrow Hostile(x)$

$American(West)$  and  $Enemy(Nono, America)$



# Backward chaining example

$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$

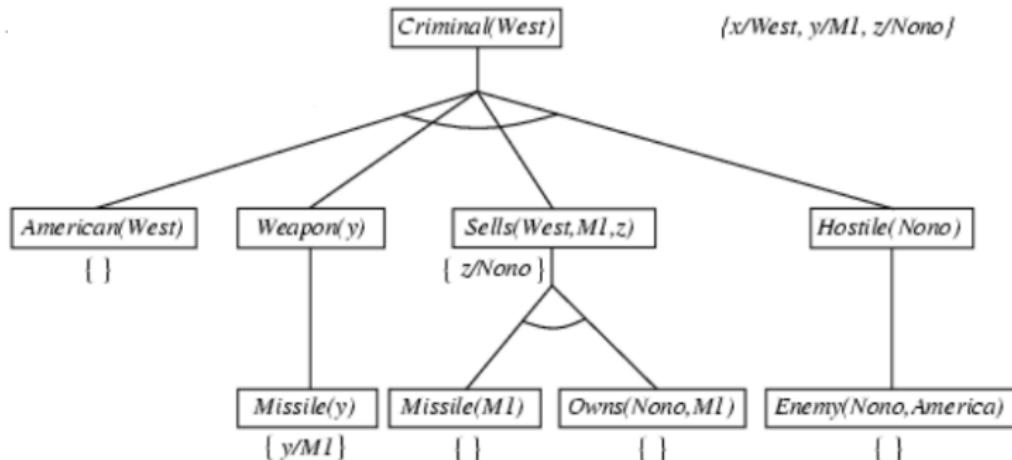
Owes(Nono, M<sub>1</sub>) and Missle(M<sub>1</sub>)

Missle(x)  $\wedge$  Owes(Nono, x)  $\Rightarrow$  Sells(West, x, Nono)

Missle(x)  $\Rightarrow$  Weapon(x)

Enemy(x, America)  $\Rightarrow$  Hostile(x)

American(West) and Enemy(Nono, America)



# Properties of backward chaining

- Depth-first recursive proof search: space is linear in size of proof
- Incomplete due to infinite loops
  - partial fix by checking current goal against every goal on stack
- Inefficient due to repeated subgoals (both success and failure)
  - fix using caching of previous results (extra space)
- Widely used for logic programming

# Resolution

- A method for telling whether a propositional formula is satisfiable and for proving that a first-order formula is unsatisfiable.
- Yields a **complete** inference algorithm
- If a set of clauses if unsatisfiable, then the resolution closure of those clauses contains the empty clause (propositional logic)

# Ground Binary Resolution

$$\frac{C \vee P \quad D \vee \neg P}{C \vee D}$$

Soundness:

$$C \vee P \text{ iff } \neg C \Rightarrow P$$

$$D \vee \neg P \text{ iff } P \Rightarrow D$$

Therefore,  $\neg C \Rightarrow D$ ,

which is equivalent to  $C \vee D$

Note: if both  $C$  and  $D$  are empty then resolution deduces the **empty clause**, i.e. false.

# Non-Ground Binary Resolution

$$\frac{C \vee P \quad D \vee \neg P'}{(C \vee D) \theta}$$

where  $\theta$  is the MGU of  $P$  and  $P'$

- The two clauses are assumed to be **standardized apart** so that they share **no** variables.
- **Soundness:** apply  $\theta$  to premises then appeal to ground binary resolution.

$$\frac{C\theta \vee P\theta \quad D\theta \vee \neg P\theta}{C\theta \vee D\theta}$$

## Example

$$\frac{\neg \text{Rich}(x) \vee \text{Unhappy}(x) \quad \text{Rich}(\text{Ken})}{\text{Unhappy}(\text{Ken})}$$

with  $\theta = \{x/\text{Ken}\}$

# Factoring

$$\frac{A \vee B \quad A \vee \neg B}{A \vee A}$$

More generally

$$\frac{C \vee P_1 \vee \dots \vee P_m}{(C \vee P_1) \theta}$$

where  $\theta$  is the MGU of the  $P_i$

**Soundness:** by universal instantiation and deletion of duplicates.

# Full Resolution

$$\frac{C \vee P_1 \vee \dots \vee \neg P_m \quad D \vee \neg P'_1 \vee \dots \vee \neg P'_n}{(C \vee D) \theta}$$

where  $\theta$  is MGU of all  $P_i$  and  $P'_j$

- Soundness: by combination of factoring and binary resolution.
- To prove  $\alpha$ : apply resolution steps to  $CNF(KB \wedge \neg\alpha)$ 
  - complete for FOL, if full resolution or binary resolution + factoring is used

# Conversion to CNF

Example:

Everyone who loves all animals is loved by someone:

$$\forall x. [\forall y. \text{Animal}(y) \Rightarrow \text{Loves}(x, y)] \Rightarrow [\exists y. \text{Loves}(y, x)]$$

- ① Eliminate all biconditionals and implications

$$\forall x. \neg[\forall y. \neg\text{Animal}(y) \vee \text{Loves}(x, y)] \vee [\exists y. \text{Loves}(y, x)]$$

- ② Move  $\neg$  inwards, use:  $\neg\forall x.p \equiv \exists x.\neg p$ ,  $\neg\exists x.p \equiv \forall x.\neg p$ , etc.

$$\forall x. [\exists y. \neg(\neg\text{Animal}(y) \vee \text{Loves}(x, y))] \vee [\exists y. \text{Loves}(y, x)]$$

$$\forall x. [\exists y. \neg\neg\text{Animal}(y) \wedge \neg\text{Loves}(x, y)] \vee [\exists y. \text{Loves}(y, x)]$$

$$\forall x. [\exists y. \text{Animal}(y) \wedge \neg\text{Loves}(x, y)] \vee [\exists y. \text{Loves}(y, x)]$$

# Conversion to CNF contd.

- ① Standardize variables apart: each quantifier should use a different one

$$\forall x. [\exists y. \text{Animal}(y) \wedge \neg \text{Loves}(x, y)] \vee [\exists z. \text{Loves}(z, x)]$$

- ② Skolemize: a more general form of existential instantiation

Each existential variable is replaced by a Skolem function of the enclosing universally quantified variables<sup>1)</sup>:

$$\forall x. [\text{Animal}(F(x)) \wedge \neg \text{Loves}(x, F(x))] \vee \text{Loves}(G(x), x)$$

- ③ Drop universal quantifiers:

$$[\text{Animal}(F(x)) \wedge \neg \text{Loves}(x, F(x))] \vee \text{Loves}(G(x), x)$$

- ④ Distribute  $\vee$  over  $\wedge$  :

$$[\text{Animal}(F(x)) \vee \text{Loves}(G(x), x)] \wedge [\neg \text{Loves}(x, F(x)) \vee \text{Loves}(G(x), x)]$$

<sup>1)</sup> No enclosing universal quantifier? Just replace with Skolem constant, i.e. a function with no argument.

# 'West' Clauses

$\neg American(x) \vee \neg Weapon(y) \vee \neg Sells(x, y, z) \vee$   
 $\neg Hostile(z) \vee Criminal(x)$

*Owns(Nono, M<sub>1</sub>) and Missile(M<sub>1</sub>)*

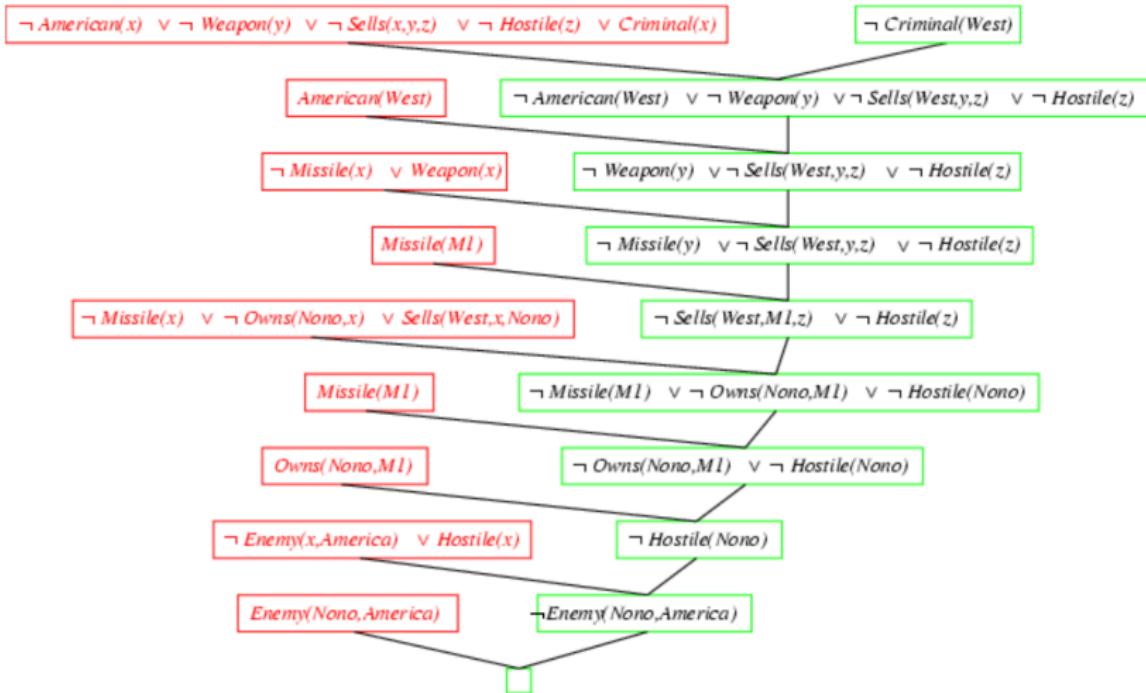
$\neg Missile(x) \vee \neg Owns(Nono, x) \vee Sells(West, x, Nono)$

$\neg Missile(x) \vee Weapon(x)$

$\neg Enemy(x, America) \vee Hostile(x)$

*American(West) and Enemy(Nono, America)*

# Resolution proof: definite clauses



# Summary

- Forward chaining
- Backward chaining
- Resolution

# Inf2D 15: Introduction to Planning

Michael Herrmann

University of Edinburgh, School of Informatics

17/02/2017

**informatics**



Credits: The content of this lecture was prepared by Alex Lascarides and follows R&N

# Where are we?

The first two blocks of the course dealt with . . .

- Basic notions of agency
- Intelligent problem-solving
- Heuristic search, constraints
- Logic & logical reasoning
- Reasoning about actions and time

In the remainder of the course we will talk about . . .

- Planning
- Uncertainty

# What is planning?

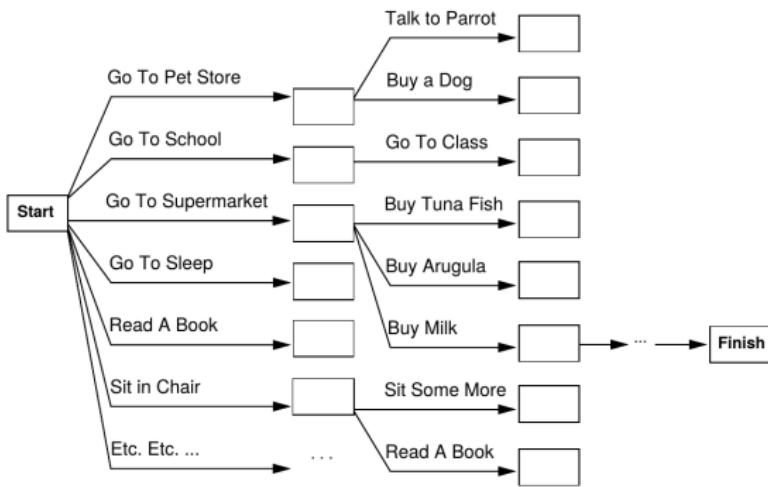
- Planning is the task of coming up with a sequence of actions that will achieve a goal
- We are only considering classical planning in which environments are
  - fully observable (accessible),
  - deterministic,
  - finite,
  - static (up to agents' actions),
  - discrete (in actions, states, objects and events).
- Some of these assumptions will be relaxed in the “uncertainty” part of this course

# Why planning?

- So far we have dealt with two types of agents:
  - ① Search-based problem-solving agents
  - ② Logical planning agents
- Do these techniques work for solving planning problems?

# Why planning?

- Consider a search-based problem-solving agent in a robot shopping world
- Task: Go to the supermarket and get milk, bananas and a cordless drill
- What would a search-based agent do?



# Problems with search

- No goal-directedness.
- No problem decomposition into sub-goals that build on each other
  - May undo past achievements
  - May go to the store 3 times!
- Simple goal test doesn't allow for the identification of milestones
- How do we find a good heuristic function?  
How do we model the way humans perceive complex goals and the quality of a plan?

# How about logic & deductive inference?

- Generally a good idea, allows for “opening up” representations of states, actions, goals and plans
- If  $Goal = \text{Have(Bananas)} \wedge \text{Have(Milk)}$  this allows achievement of sub-goals (if independent)
- Current state can be described by properties in a compact way (e.g.  $\text{Have(Drill)}$  stands for hundreds of states)
- Allows for compact description of actions, for example

$$\text{Object}(x) \Rightarrow \text{Can}(a, \text{Grab}(x))$$

- Allows for representing a plan hierarchically, e.g.  
 $\text{GoTo(Supermarket)} = \text{Leave(House)} \wedge$   
 $\text{ReachLocationOf(Supermarket)} \wedge \text{Enter(Supermarket)}$   
then decompose further into sub-plans

# How about logic & deductive inference?

- Problems:
  - In its general form either awkward (propositional logic) or tractability problems (first-order logic), high complexity
  - If  $p$  is a sequence that achieves the goal, then so is  $[a, a^{-1}|p]$ !
- Solutions: We need
  - ① To reduce complexity to allow scaling up.
  - ② To allow reasoning to be guided by plan 'quality'/efficiency.
- Do ① today; ② later.

# Representing planning problems

- Need a language expressive enough to cover interesting problems, restrictive enough to allow efficient algorithms.
- Planning Domain Definition Language (PDDL)
- PDDL will allow you to express:
  - ① states
  - ② actions: a description of transitions between states
  - ③ and goals: a (partial) description of a state.

# Representing States and Goals in PDDL

- States represented as conjunctions of propositional or function-free first order positive literals:
  - $\text{Happy} \wedge \text{Sunshine}$
  - $\text{At}(\text{Plane}_1, \text{Melbourne}) \wedge \text{At}(\text{Plane}_2, \text{Sydney})$
- These aren't states:
  - $\text{At}(x, y)$  (no variables allowed),
  - $\text{Love}(\text{Father}(\text{Fred}), \text{Fred})$  (no function symbols allowed)
  - $\neg \text{Happy}$  (no negation allowed).

Closed-world assumption! (i.e. non-derivable sentences are assumed to be false)

- A goal is a partial description of a state, and you can use negation, variables etc. to express that description.
  - $\neg \text{Sunshine}, \text{At}(x, \text{EDI}), \text{Love}(\text{Father}(\text{Fred}), \text{Fred}), \dots$

# Actions in PDDL

Action( $Fly(p, from, to)$ ,

Precond:  $At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$

Effect:  $\neg At(p, from) \wedge At(p, to)$ )

- Actually **action schemata**, as they may contain variables
- Action name and parameter list serves to identify action
- **Precondition**: defines states in which action is **executable**:
  - Conjunction of positive and negative literals, where all variables must occur in action name.
- **Effect**: defines how literals in the input state get changed (anything not mentioned stays the same).
  - Conjunction of positive and negative literals, with all its variables also in the preconditions.
  - Often positive and negative effects are divided into **add list** and **delete list**

# The Semantics of PDDL

## States and their Descriptions

$s \models At(P_1, EDI)$  iff  $At(P_1, EDI) \in s$

$s \models \neg At(P_1, EDI)$  iff  $At(P_1, EDI) \notin s$

$s \models \varphi(x)$  iff there is a ground term  $d$  such that  $s \models \varphi[x/d]$ .

$s \models \varphi \wedge \psi$  iff  $s \models \varphi$  and  $s \models \psi$

# The Semantics of PDDL

## Applicable Actions

- Any action is applicable in any state that satisfies the precondition with an appropriate substitution for parameters.
- Example: State

$$\begin{aligned} & At(P_1, Melbourne) \wedge At(P_2, Sydney) \wedge Plane(P_1) \wedge Plane(P_2) \\ & \wedge Airport(Sydney) \wedge Airport(Melbourne) \wedge Airport(Heathrow) \end{aligned}$$

satisfies precondition

$$At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$$

with substitution (among others)

$$\{p/P_2, \text{from}/Sydney, \text{to}/Heathrow\}$$

# The semantics of PDDL: The Result of an Action

- **Result** of executing action  $a$  in state  $s$  is state  $s'$  with any positive literal  $P$  in  $a$ 's **Effects** added to the state and every negative literal  $\neg P$  removed from it (under the given substitution) .
- In our example  $s'$  would be

$$At(P_1, Melbourne) \wedge At(P_2, Heathrow) \wedge Plane(P_1) \wedge Plane(P_2)$$
$$\wedge Airport(Sydney) \wedge Airport(Melbourne) \wedge Airport(Heathrow)$$

- “PDDL assumption”: every literal not mentioned in the effect remains unchanged (cf. frame problem)
- **Solution** = action sequence that leads from the initial state to a state that satisfies the goal.

# Blocks world example

- Given: A set of cube-shaped blocks sitting on a table
- Can be stacked, but only one on top of the other
- Robot arm can move around blocks (one at a time)
- Goal: to stack blocks in a certain way
- Formalisation in PDDL:
  - $On(b, x)$  to denote that block  $b$  is on  $x$  (block/table)
  - $Move(b, x, y)$  to indicate action of moving  $b$  from  $x$  to  $y$
  - Precondition for this action: nothing must be stacked on  $y$ :  $Clear(y)$  etc. ( $x$  was considered irrelevant in the previous Move function)

# Blocks world example

- Action schema:

$$Action(Move(b, x, y),$$
$$\text{Precond} : On(b, x) \wedge Clear(b) \wedge Clear(y)$$
$$\text{Effect} : On(b, y) \wedge Clear(x) \wedge \neg On(b, x) \wedge \neg Clear(y))$$

- Problem: When  $x = Table$  or  $y = Table$  we infer that the table is clear when we have moved a block from it (not true) and require that table is clear to move something on it (not true)

- Solution: Introduce another action

$$Action(MoveToTable(b, x),$$
$$\text{Precond: } On(b, x) \wedge Clear(b)$$
$$\text{Effect: } On(b, Table) \wedge Clear(x) \wedge \neg On(b, x))$$

# Does this Work?

- Interpret  $\text{Clear}(b)$  as “there is space on  $b$  to hold a block” (thus  $\text{Clear}(\text{Table})$  is always true)
- But without further modification, planner can still use  $\text{Move}(b, x, \text{Table})$ :
  - Needlessly increases search space  
(not a big problem here, but can be)
- So part of solution is to also add  $\text{Block}(b) \wedge \text{Block}(y)$  to precondition of  $\text{Move}$

# Summary

- Defined the planning problem
- Discussed problems with search/logic
- Introduced PDDL: a special representation language for planning
- Blocks world example as a famous application domain
- Next time: Algorithms for planning!

State-Space Search and Partial-Order Planning

# Inf2D 14: Situation Calculus

Michael Herrmann

University of Edinburgh, School of Informatics

16/02/2017

**informatics**



Credits: The content of this lecture was prepared by Michael Rovatsos and follows R&N

# Outline

- Planning
- Situations
- Frame problem

# Using Logic to Plan

- We need ways of:
  - representing the world.
  - representing the goal.
  - representing how actions change the world.
- We haven't said much about the last.
  - **Difficulty:** After an action, new things are true, and some previously true facts are no longer true.

# Situations

Idea:

- **Situations** extend the concept of a **state** by additional **logical terms**
  - Consist of initial situation (usually called  $S_0$ ) and all situations generated by applying an action to a situation.
- Provide facts about situations.
  - By relativising predications to situations.
  - E.g., instead of saying just  $On(A, B)$ , say (somehow)  $On(A, B)$  in situation  $S_0$
- Actions are thus
  - performed in a situation, and
  - produce new situations with new facts.
  - Examples: *Forward* and *Turn(Right)*

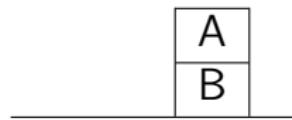
# Representing Predications Relative to a Situation

- Can add an argument for a situation to each predicate that can change.
  - E.g., instead of  $On(A, B)$ , write  $On(A, B, S_0)$
- Alternatively, introduce a predicate **Holds**
  - On etc. become now functions
  - E.g.,  $Holds(On(A, B), S_0)$
  - What do things like  $On(A, B)$  now mean?  
Either a category of situations, in which  $A$  is **on**  $B$ , or a set of those situations.

# How This Will Work

- Before some action, we might have in our KB:

- $On(A, B, S_0)$
- $On(B, Table, S_0)$



- After an action that moves A to the table, say, we add

- $Clear(B, S_1)$
- $On(A, Table, S_1)$



- All these propositions are true. We have dealt with the issue of change, by keeping track of what is true when.

# Same Thing, Slightly Different Notation

- Before

- $Holds(On(A, B), S_0)$
- $Holds(On(B, Table), S_0)$



- After, add

- $Holds(Clear(B), S_1)$
- $Holds(On(A, Table), S_1)$



# Representing Actions

- Need to represent:
  - Results of doing an action
  - Conditions that need to be in place to perform an action.
- For convenience, we will define **functions** to abbreviate actions:
  - E.g.,  $Move(A, B)$  denotes the **action type** of moving  $A$  onto  $B$ .
  - These are action types, because actions themselves are specific to time, etc.
- Now, introduce a **function Result**, designating “the situation resulting from doing an action type in some situation”.
  - E.g.,  $Result(Move(A, B), S_0)$  means “the situation resulting from doing an action of type  $Move(A, B)$  in situation  $S_0$ ”.

# How This Works

- Keep in mind that things like  
 $\text{Result}(\text{Move}(A, B), S_0)$   
are terms, and denote **situations**.  
They can appear anywhere we would expect a situation.
- So we can say things like  
 $S_1 = \text{Result}(\text{Move}(A, B), S_0),$   
 $\text{On}(A, B, \text{Result}(\text{Move}(A, B), S_0)) \equiv \text{On}(A, B, S_1),$   
etc.
- Alternatively,  
 $\text{Holds}(\text{On}(A, B), \text{Result}(\text{Move}(A, B), S_0)),$   
etc.

# Axiomatising Actions

- Now, we can describe the results of actions, together with their **preconditions**.
- E.g., “If nothing is on  $x$  and  $y$ , then one can move  $x$  to on top of  $y$ , in which case  $x$  will then be on  $y$ .  
$$\begin{aligned} & \forall x, y, s \ Clear(x, s) \wedge \Clear(y, s) \\ & \Rightarrow \On(x, y, \text{Result}(\text{Move}(x, y), s)) \end{aligned}$$
- Alternatively:  
$$\begin{aligned} & \forall x, y, s \ Holds(\Clear(x), s) \wedge \Holds(\Clear(y), s) \\ & \Rightarrow \Holds(\On(x, y), \text{Result}(\text{Move}(x, y), s)) \end{aligned}$$
- This is an **effect axiom**.  
It includes a precondition as well.

# Situation Calculus

- This approach is called the situation calculus.
- We axiomatise all our actions, then use a general theorem prover to prove that a situation exists in which our goal is true.
- The actions in the proof would comprise our plan.

# A Very Simple Example

- KB:

$On(A, Table, S_0)$

$On(B, C, S_0)$

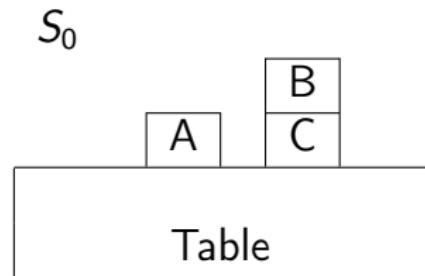
$On(C, Table, S_0)$

$Clear(A, S_0)$

$Clear(B, S_0)$

and axioms about actions, etc.

- Goal:  $\exists s'. On(A, B, s')$



# What happens?

- We try to prove  $On(A, B, s')$  for some  $s'$ 
  - Find axiom
$$\forall x, y, s. \text{Clear}(x, s) \wedge \text{Clear}(y, s) \\ \Rightarrow On(x, y, \text{Result}(\text{Move}(x, y), s))$$
  - By chaining, e.g., goal would be true if we could prove  $\text{Clear}(A, s) \wedge \text{Clear}(B, s)$  by backward chaining.
  - But both are true in  $S_0$ , so we can conclude
$$On(A, B, \text{Result}(\text{Move}(A, B), S_0))$$
- We are done!
- We look in the proof and see only one action,  $\text{Move}(A, B)$ , which is executed in situation  $S_0$ , so this is our **plan**.

# Example: Same Initial Situation, Harder Goal<sup>1</sup>

- KB:

$On(A, Table, S_0)$

$On(B, C, S_0)$

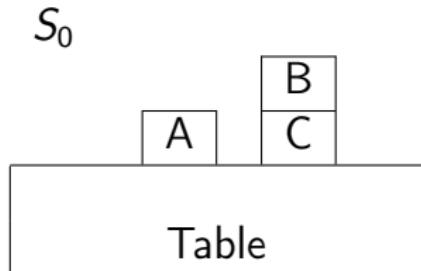
$On(C, Table, S_0)$

$Clear(A, S_0)$

$Clear(B, S_0)$

and axioms about actions, etc.

- Goal:  $\exists s'. On(A, B, s') \wedge On(B, C, s')$



---

<sup>1</sup>It's not really harder,  $B$  is already on  $C$ , and we just showed how to put  $A$  on  $B$

# With Goal $On(A, B, s') \wedge On(B, C, s')$

- Suppose we try to prove the first subgoal,  $On(A, B, s')$ .
  - Use same axiom
$$\forall x, y, s. Clear(x, s) \wedge Clear(y, s) \Rightarrow On(x, y, Result(Move(x, y), s))$$
  - Again, by chaining, we can conclude
$$On(A, B, Result(Move(A, B), S_0))$$
  - Abbreviating  $Result(Move(A, B), S_0)$  as  $S_1$ , we have
$$On(A, B, S_1)$$
- Substituting for  $s'$  in our other subgoal makes that  $On(B, C, S_1)$ . If this is true, we're done.
- But we have **no reason to believe this is true!**
- Sure,  $On(B, C, S_0)$ , but how does the planner know this is still true, i.e.,  $On(B, C, S_1)$ ?
- In fact, it doesn't, so it fails to find an answer!

# The Frame Problem

- We have failed to express the fact that everything that isn't changed by an action stays the same.
- Can fix by adding frame axioms. E.g.:  
 $\forall x, y, z, s. \text{Clear}(x, s) \Rightarrow \text{Clear}(x, \text{Result}(\text{Paint}(x, y), s))$   
... ... ...
- There are lots of these!
- Is this a big problem?

# Better Frame Axioms

- Can fix with neater formulation:

$$\begin{aligned} \forall x, y, s, a. & \text{ } On(x, y, s) \wedge (\forall z. a = Move(x, z) \Rightarrow y = z) \\ & \Rightarrow On(x, y, Result(a, s)) \end{aligned}$$

- Can combine with effect axioms to get **successor-state axioms**:

$$\begin{aligned} \forall x, y, s, a. & \text{ } On(x, y, Result(a, s)) \Leftrightarrow \\ & On(x, y, s) \wedge (\forall z. a = Move(x, z) \Rightarrow y = z) \\ & \vee (Clear(x, s) \wedge Clear(y, s) \wedge a = Move(x, y)) \end{aligned}$$

# How Does This Help Our Example?

- We want to prove  $On(B, C, Result(Move(A, B), S_0))$  given that  $On(B, C, S_0)$
- Axiom says  $\forall x, y, s, a. On(x, y, Result(a, s)) \Leftrightarrow$   
 $On(x, y, s) \wedge (\forall z. a = Move(x, z) \Rightarrow y = z)$   
 $\vee (Clear(x, s) \wedge Clear(y, s) \Rightarrow a = Move(x, y))$
- So need to show  
 $On(B, C, S_0) \wedge (\forall z. Move(A, B) = Move(B, z) \Rightarrow C = z)$  is true, which is easy
  - The first conjunct is in the KB.
  - The second one is true since actions are the same only if they have the same name and involve the exact same objects i.e.\*  
 $A(x_1, \dots, x_m) = A(y_1, \dots, y_m)$  iff  $x_1 = y_1 \wedge \dots \wedge x_m = y_m$   
so  $Move(A, B) = Move(B, z)$  is false.

Note: Another assumption\* in KB:  $A(x_1, \dots, x_m) \neq B(y_1, \dots, y_n)$

\*These are known as Unique Action Axioms

# For Refutation Theorem Proving: (Dual) Skolemisation

- Suppose  $\forall x. \exists y. G(x, y)$  is goal in resolution refutation.
- So, we need to **negate** the goal:  
$$\neg \forall x. \exists y. G(x, y) \equiv \exists x. \forall y. \neg G(x, y)$$
- Then skolemise (i.e drop existential quantifier):  
$$\neg G(X_0, y)$$
- Intuition:
  - $y$  is to be unified to construct **witness**.
  - $X_0$  must **not** be instantiated.
- Similar story for GMP, but goal not negated, i.e.  
 $G(X_0, y)$ , for some  $y$ , is used as the goal.

# KB and Axioms as Clauses

Constants  $A, B, C, S_0$   
Variables:  $a, x, y, s$

- Initial State

$On(A, Table, S_0)$

$On(B, C, S_0)$

$On(C, Table, S_0)$

$Clear(A, S_0)$

$Clear(B, S_0)$

- Goal

$\neg On(A, B, s') \vee \neg On(B, C, s')$

- Effect Axiom

$\neg Clear(x, s) \vee \neg Clear(y, s) \vee$

$On(x, y, Result(Move(x, y), s))$

- Frame Axioms

Skolem function



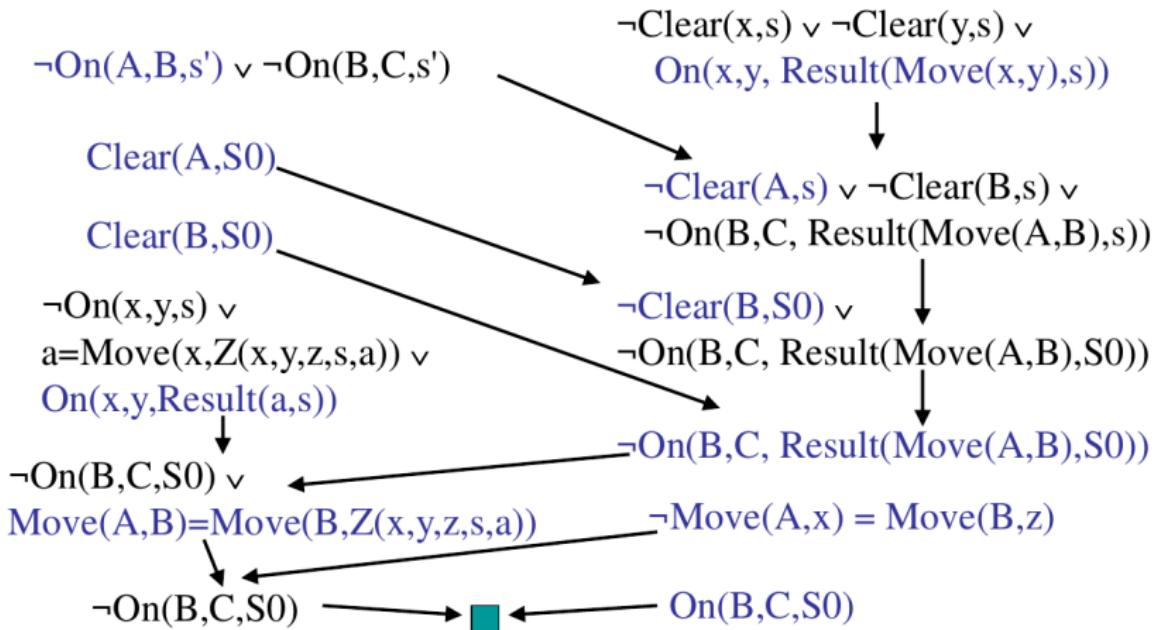
$\neg On(x, y, s) \vee a = Move(x, Z(x, y, z, s, a)) \vee On(x, y, Result(a, s))$

$\neg On(x, y, s) \vee \neg y = Z(x, y, z, s, a) \vee On(x, y, Result(a, s))$

- Unique Action Axioms:  $\neg Move(A, B) = Move(B, z)$ , etc.

- Unique Name Axiom: disequality for every pair of constants in KB

# Resolution Refutation



# Frame problem partially solved

- This solves the representational part of the frame problem.
- Still have to compute that everything that was true that wasn't changed is still true.
- Inefficient (as is general theorem proving).
- Solution: Special purpose representations, special purpose algorithms, called **planners**.

# Summary

- Planning
- Situations
- Frame problem

# Inf2D 17: State-Space Search and Partial-Order Planning

Michael Herrmann

University of Edinburgh, School of Informatics

02/03/2017

**informatics**



Credits: The content of this lecture was prepared by Alex Lascarides and follows R&N

# Where are we?

Last time . . .

- we defined the planning problem
- discussed problem with using search and logic in planning
- introduced representation languages for planning
- looked at blocks world example

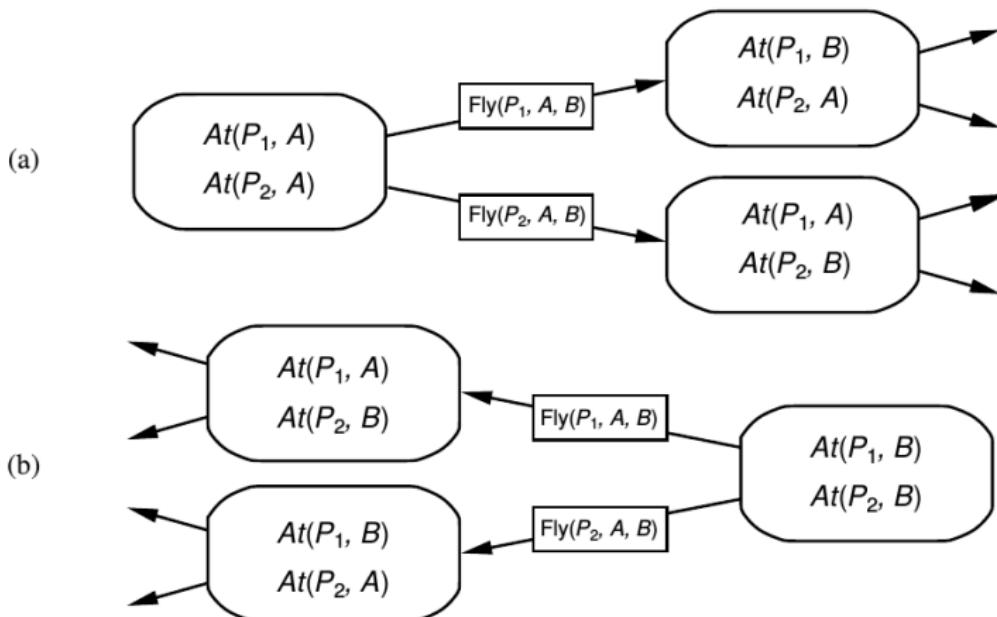
Today . . .

- State-space search and partial-order planning

# Planning with state-space search

- Most straightforward way to think of planning process:  
**search the space of states using action schemata**
- Since actions are defined both in terms of preconditions and effects we can search in both directions
- Two methods:
  - ① **forward state-space search:** Start in initial state; consider action sequences until goal state is reached.
  - ② **backward state-space search:** Start from goal state; consider action sequences until initial state is reached

# Planning with state-space search



# Forward state-space search

- Also called progression planning
- Formulation of planning problem:
  - Initial state of search is initial state of planning problem (= set of positive literals)
  - Applicable actions are those whose preconditions are satisfied
  - Single successor function works for deterministic planning problems (consequence of action representation)
  - Goal test = checking whether state satisfies goal of planning problem
  - Step cost usually 1, but different costs can be allowed

# Forward state-space search

- Search space is finite in the absence of function symbols
- Any complete graph search algorithm (like A\* ) will be a complete graph planning algorithm
- Forward search does not solve problem of irrelevant actions (all actions considered from each state)
- Efficiency depends largely on quality of heuristics
- Example:
  - Air cargo problem, 10 airports with 5 planes each, 20 pieces of cargo
  - Task: move all 20 pieces of cargo at airport  $A$  to airport  $B$
  - Each of 50 planes can fly to 9 airports, each of 200 packages can be unloaded or loaded (individually)
  - Assuming an average of about 1000 actions at every airport, we have  $1000^{81}$  nodes in the search tree (although solution trivial)!

# Backward state-space search

- In normal search, backward approach hard because goal described by a set of constraints (rather than being listed explicitly)
- Problem of how to generate predecessors, but planning representations allow us to consider only **relevant** actions
- Exclusion of irrelevant actions decreases branching factor
- In example, only about 20 actions working backward from goal
- **Regression planning** = computing the states from which applying a given action leads to the goal
- Must ensure that actions are **consistent**, i.e. they don't undo any desired literals

# Air cargo domain example

- Goal can be described as

$$At(C_1, B) \wedge At(C_2, B) \wedge \cdots \wedge At(C_{20}, B)$$

- To achieve  $At(C_1, B)$  there is only one action,  
 $Unload(C_1, p, B)$  ( $p$  unspecified)
- Can do this action only if its preconditions are satisfied.
- So the predecessor to the goal state must include  
 $In(C_1, p) \wedge At(p, B)$ , and should not include  $At(C_1, B)$   
(otherwise irrelevant action)
- Full predecessor:  $In(C_1, p) \wedge At(p, B) \wedge \cdots \wedge At(C_{20}, B)$
- $Load(C_1, p)$  would be inconsistent (negates  $At(C_1, B)$ )

# Backward state-space search

- General process of constructing predecessors for backward search given goal description  $G$ , relevant and consistent action  $A$ :
  - Any positive effects of  $A$  that appear in  $G$  are deleted
  - Each precondition of  $A$  is added unless it already appears
- Any standard search algorithm can be used, terminates when predecessor description is satisfied by initial (planing) state
- First-order case may require additional substitutions which must be applied to actions leading from state to goal

# Heuristics for state-space search

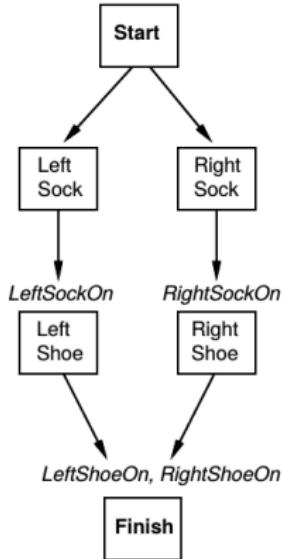
- Two possibilities:
  - ① Divide and Conquer ([subgoal decomposition](#))
  - ② Derive a [Relaxed Problem](#)
- Subgoal decomposition is . . .
  - optimistic (admissible) if negative interactions exist (e.g. subplan deletes goal achieved by other subplan)
  - pessimistic (inadmissible) if positive interactions exist (e.g. subplans contain redundant actions)
- Relaxations:
  - drop all preconditions (all actions always applicable, combined with subgoal independence makes prediction even easier)
  - remove all negative effects (and count minimum number of actions so that union satisfies goals)
  - empty delete lists approach (involves running a simple planning problem to compute heuristic value)

# Partial-order planning

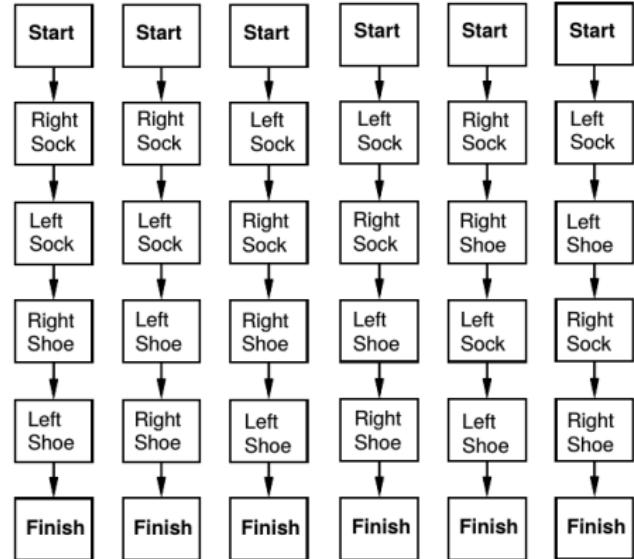
- State-space search planning algorithms consider **totally ordered** sequences of actions
- Better not to commit ourselves to complete chronological ordering of tasks (**least commitment** strategy)
- **Basic idea:**
  - Add actions to a plan without specifying which comes first unless necessary
  - Combine 'independent' subsequences afterwards
- Partial-order solution will correspond to one or several **linearisations** of partial-order plan
- Search in **plan space** rather than state spaces (because your search is over ordering constraints on actions, as well as transitions among states).

# Example: Put your socks and shoes on

Partial-Order Plan:



Total-Order Plans:



# Partial-order planning (POP) as a search problem

Define POP as search problem over plans consisting of:

- **Actions:** initial plan contains dummy actions  
*Start*(no preconditions, effect = initial state) and  
*Finish*(no effects, precondition = goal literals)
- **Ordering constraints** on actions  $A \prec B$  ( $A$  must occur before  $B$ ); contradictory constraints prohibited
- **Causal links** between actions  $A \xrightarrow{p} B$  express  $A$  achieves  $p$  for  $B$  ( $p$  precondition of  $B$ , effect of  $A$ , must remain true between  $A$  and  $B$ ); inserting action  $C$  ( $A \prec C$  and  $C \prec B$ ) with effect  $\neg p$  would lead to conflict
- **Open preconditions:** set of conditions not yet achieved by the plan (planners try to make open precondition set empty without introducing contradictions)

# The POP algorithm

- **Final plan** for socks and shoes example (without trivial ordering constraints):

Actions: {RightSock, RightShoe, LeftSock, LeftShoe, Start, Finish}

Orderings: {RightSock  $\prec$  RightShoe, LeftSock  $\prec$  LeftShoe}

Links: {RightSock  $\xrightarrow{\text{RightSockOn}}$  RightShoe,  
LeftSock  $\xrightarrow{\text{LeftSockOn}}$  LeftShoe,  
RightShoe  $\xrightarrow{\text{RightShoeOn}}$  Finish,  
LeftShoe  $\xrightarrow{\text{LeftShoeOn}}$  Finish}

Open preconditions: {}

- **Consistent plan** = plan without cycles in orderings and conflicts with links
- **Solution** = consistent plan without open preconditions
- Every linearisation of a partial-order solution is a total-order solution (implications for execution!)

# The POP algorithm

- Initial plan:  
Actions:  $\{Start, Finish\}$ , Orderings:  $\{Start \prec Finish\}$ ,  
Links:  $\{ \}$ , Open preconditions: Preconditions of *Finish*
- Pick  $p$  from open preconditions on some action  $B$ ,  
generate a consistent successor plan for every  $A$  that  
achieves  $p$
- Ensuring consistency:
  - Add  $A \xrightarrow{p} B$  and  $A \prec B$  to plan. If  $A$  new, add  $A$  and  
 $Start \prec A$  and  $A \prec Finish$  to plan
  - Resolve conflicts between the new link and all actions  
and between  $A$  (if new) and all links as follows:  
If conflict between  $A \xrightarrow{p} B$  and  $C$ , add  $B \prec C$  or  $C \prec A$
  - Goal test: check whether there are open preconditions  
(only consistent plans are generated)

# Partial-order planning example (1)

*Init(At(Flat, Axle)  $\wedge$  At(Spare, Trunk)). Goal(At(Spare, Axle)).*

*Action(Remove(Spare, Trunk),*

*Precond: At(Spare, Trunk)*

*Effect:  $\neg$ At(Spare, Trunk)  $\wedge$  At(Spare, Ground))*

*Action(Remove(Flat, Axle),*

*Precond: At(Flat, Axle)*

*Effect:  $\neg$ At(Flat, Axle)  $\wedge$  At(Flat, Ground))*

*Action(PutOn(Spare, Axle),*

*Precond: At(Spare, Ground)  $\wedge$   $\neg$ At(Flat, Axle)*

*Effect:  $\neg$ At(Spare, Ground)  $\wedge$  At(Spare, Axle))*

*Action(LeaveOvernight,*

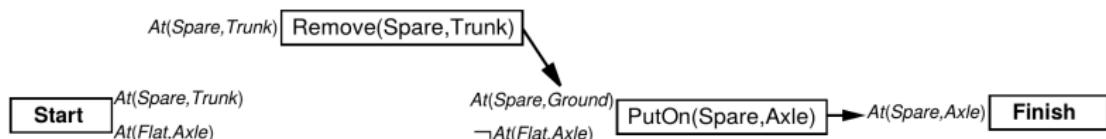
*Precond:*

*Effect:  $\neg$ At(Spare, Ground)  $\wedge$   $\neg$ At(Spare, Axle)  $\wedge$*

*$\neg$ At(Spare, Trunk)  $\wedge$   $\neg$ At(Flat, Ground)  $\wedge$   $\neg$ At(Flat, Axle))*

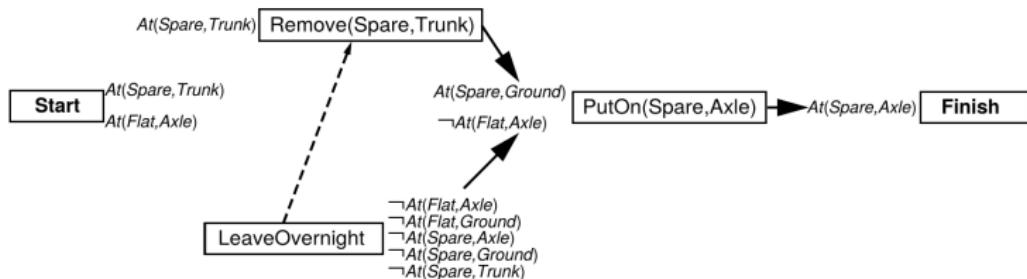
# Partial-order planning example (2)

- Pick (only) open precondition  $At(Spare, Axle)$  of *Finish*  
Only applicable action =  $PutOn(Spare, Axle)$
- Pick  $At(Spare, Ground)$  from  $PutOn(Spare, Axle)$   
Only applicable action =  $Remove(Spare, Trunk)$
- Situation after two steps:



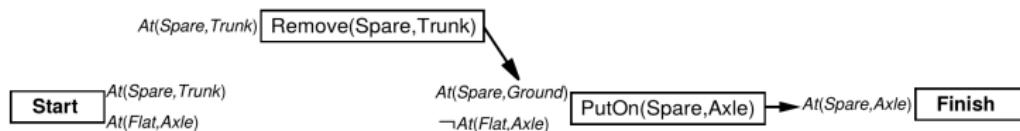
# Partial-order planning example (3)

- Pick  $\neg At(Flat, Axle)$  precondition of  $PutOn(Spare, Axle)$   
Choose  $LeaveOvernight$ , effect  $\neg At(Spare, Ground)$
- Conflict with link  
 $Remove(Spare, Trunk) \xrightarrow{At(Spare, Ground)} PutOn(Spare, Axle)$
- Resolve by adding  
 $LeaveOvernight \prec Remove(Spare, Trunk)$   
Why is this the only solution?



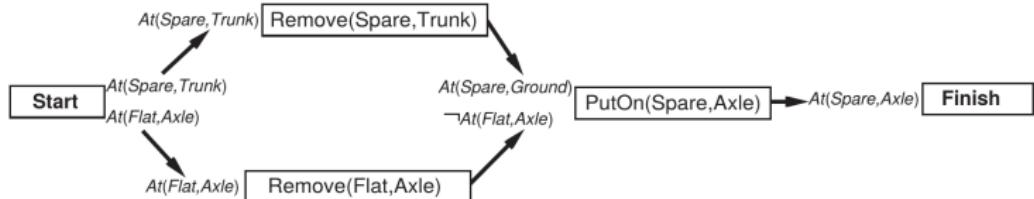
# Partial-order planning example (4)

- Remaining open precondition  $At(Spare, Trunk)$ , but conflict between  $Start$  and  $\neg At(Spare, Trunk)$  effect of  $LeaveOvernight$
- No ordering before  $Start$  possible or after  $Remove(Spare, Trunk)$  possible
- No successor state, backtrack to previous state and remove  $LeaveOvernight$ , resulting in this situation:



# Partial-order planning example (5)

- Now choose  $\text{Remove}(\text{Flat}, \text{Axe})$  instead of  $\text{LeaveOvernight}$
- Next, choose  $\text{At}(\text{Spark}, \text{Trunk})$  precondition of  $\text{Remove}(\text{Spare}, \text{Trunk})$   
Choose *Start* to achieve this
- Pick  $\text{At}(\text{Flat}, \text{Axe})$  precondition of  $\text{Remove}(\text{Flat}, \text{Axe})$ , choose *Start* to achieve it
- Final, complete, consistent plan:



# Dealing with unbound variables

- In first-order case, unbound variables may occur during planning process
- Example:

$Action(Move(b, x, y),$

Precond:  $On(b, x) \wedge Clear(b) \wedge Clear(y)$

Effect:

$On(b, y) \wedge Clear(x) \wedge \neg On(b, x) \wedge \neg Clear(y))$

achieves  $On(A, B)$  under substitution  $\{b/A, y/B\}$

- Applying this substitution yields

$Action(Move(A, x, B),$

Precond:  $On(A, x) \wedge Clear(A) \wedge Clear(B)$

Effect:

$On(A, B) \wedge Clear(x) \wedge \neg On(A, x) \wedge \neg Clear(B))$

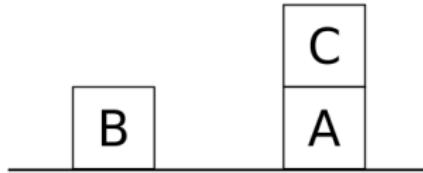
and  $x$  is still unbound (another side of the least commitment approach)

# Dealing with unbound variables

- Also has an effect on links, e.g. in example above  
 $Move(A, x, B) \xrightarrow{On(A,B)} Finish$  would be added
- If another action has effect  $\neg On(A, z)$  then this is only a conflict if  $z = B$
- Solution: insert **inequality constraints** (in example:  $z \neq B$ ) and check these constraints whenever applying substitutions
- Remark on heuristics: Even harder than in total-order planning, e.g. adapt most-constrained-variable approach from CSPs

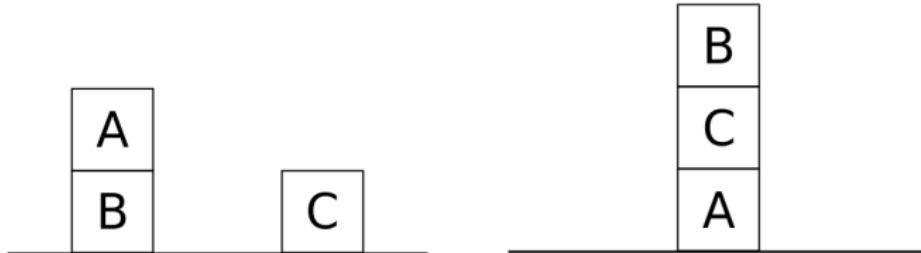
# Sussman anomaly

Start:



Goal 1:  $On(A, B)$

Goal 2:  $On(B, C)$



After achieving one of the goals, the agent cannot now pursue the other goal without undoing the first one.

# Summary

- State-space search approaches (forward/backward)
- Heuristics for state-space search planning
- Partial-order planning
- The POP algorithms
- POP as search in planning space
- POP example
- POP with unbound variables
- Next time: Planning and Acting in the Real World I

# Inf2D 18: – Planning and Acting in the Real World I

Michael Herrmann

University of Edinburgh, School of Informatics

03/03/2017

**informatics**



Credits: The content of this lecture was prepared by Alex Lascarides and follows R&N

# Where are we?

Last time . . .

- Discussed planning with state-space search
- Identified weaknesses of this approach
- Introduced partial-order planning
  - Search in plan space rather than state space
  - Described the POP algorithm and examples

Today . . .

- Planning and acting in the real world I

# Planning/acting in Nondeterministic Domains

- So far only looked at **classical** planning, i.e. environments are fully observable, static, deterministic
- Also assumed that action descriptions are correct and complete
- Unrealistic in many real-world applications:
  - Don't know everything; may even hold incorrect information
  - Actions can go wrong
- Distinction: **bounded** vs. **unbounded** indeterminacy: can possible preconditions and effects be listed at all?
- Unbounded indeterminacy related to qualification problem

# Methods for handling indeterminacy

- **Sensorless/conformant planning:** achieve goal in all possible circumstances, relies on coercion
- **Contingency planning:** for partially observable and non-deterministic environments; includes sensing actions and describes different paths for different circumstances
- **Online planning and replanning:** check whether plan requires revision during execution and replan accordingly

## Example Problem: Paint table and chair same colour

- **Initial State:** We have two cans of paint and table and chair, but colours of paint and of furniture is **unknown**:

$$\text{Object}(\text{Table}) \wedge \text{Object}(\text{Chair}) \wedge \text{Can}(C_1) \wedge \text{Can}(C_2) \wedge \text{InView}(\text{Table})$$

- **Goal State:** Chair and table same colour:

$$\text{Color}(\text{Chair}, c) \wedge \text{Color}(\text{Table}, c)$$

- **Actions:** To look at something; to open a can; to paint.

# Formal Representation of the Three Actions

Now we allow variables in preconditions that are not part of the actions's variable list!

- Action(*RemoveLid(can)*,  
Precond: *Can(can)*  
Effect: *Open(can)*)
- Action(*Paint(x, can)*,  
Precond: *Object(x) ∧ Can(can) ∧ Color(can, c) ∧ Open(can)*  
Effect: *Color(x, c)*)
- Action(*LookAt(x)*,  
Precond: *InView(y) ∧ (x ≠ y)*  
Effect: *InView(x) ∧ ¬InView(y)*)

# Sensing with Percepts

- A **percept schema** models the agent's sensors.
- It tells the agent what it knows, given certain conditions about the state it's in.
- $\text{Percept}(\text{Color}(x, c),$   
Precond:  $\text{Object}(x) \wedge \text{InView}(x))$
- $\text{Percept}(\text{Color}(\text{can}, c),$   
Precond:  $\text{Can}(\text{can}) \wedge \text{Open}(\text{can}) \wedge \text{InView}(\text{can}))$
- A fully observable environment has a percept axiom for each fluent with no preconditions!
- A sensorless planner has no percept schemata at all!

# Planning

- One could coerce the table and chair to be the same colour by painting them both—a **sensorless planner** would have to do this!
- But a **contingent planner** can do better than this:
  - ① Look at the table and chair to sense their colours.
  - ② If they're the same colour, you're done.
  - ③ If not, look at the paint cans.
  - ④ If one of the can's is the same colour as one of the pieces of furniture, then apply that paint to the other piece of furniture.
  - ⑤ Otherwise, paint both pieces with one of the cans.
- Let's now look at these types of planning in more detail ...

# How to represent belief states

- ① Sets of state representations, e.g.  $(2^n \text{ states!})$

$$\{(AtL \wedge CleanR \wedge CleanL), (AtL \wedge CleanL)\}$$

- ② Logical sentences can capture a belief state, e.g.

$AtL \wedge CleanL$  shows ignorance about  $CleanR$  by not mentioning it!

- This often offers a more compact representation, but
- Many equivalent sentences; need **canonical** representation to avoid general theorem proving; E.g:
  - All representations are ordered conjunctions of literals (under open-world assumption)
  - But this does not capture everything (e.g.  $AtL \vee CleanR$ )

- ③ Knowledge propositions, e.g.  $K(AtR) \wedge K(CleanR)$  (closed-world assumption)

- Will use second method, but clearly loss of expressiveness

# Sensorless Planning: The Belief States

- There are no *InView* fluents, because there are no sensors!
- There are unchanging facts:  
 $Object(Table) \wedge Object(Chair) \wedge Can(C_1) \wedge Can(C_2)$
- And we know that the objects and cans have colours:  
 $\forall x \exists c. Color(x, c)$
- After Skolemisation this gives an initial belief state:

$$b_0 = Color(x, C(x))$$

- A belief state corresponds exactly to the set of possible worlds that satisfy the formula – **open world assumption**.

# The Plan

$[RemoveLid(C_1), Paint(Chair, C_1), Paint(Table, C_1)]$

Rules:

- You can only apply actions whose preconditions are satisfied by your current belief state  $b$ .
- The **update of a belief state  $b$  given an action  $a$**  is the set of all states that result (in the physical transition model) from doing  $a$  in each possible state  $s$  that satisfies belief state  $b$ :

$$b' = \text{Result}(b, a) = \{s' : s' = \text{Result}_P(s, a) \wedge s \in b\}$$

- ① If action adds  $l$ ,  $l$  is in  $b'$ .
- ② If action deletes  $l$ ,  $\neg l$  is in  $b'$ .
- ③ If action says nothing about  $l$ , it retains its  $b$ -value.

# Showing the Plan Works

$$b_0 = \text{Color}(x, C(x))$$

$$b_1 = \text{Result}(b_0, \text{RemoveLid}(C_1))$$

$$= \text{Color}(x, C(x)) \wedge \text{Open}(C_1)$$

$$b_2 = \text{Result}(b_1, \text{Paint}(\text{Chair}, C_1))$$

(binding  $\{x/C_1, C(C_1)\}$  satisfies Precond

$$= \text{Color}(x, C(x)) \wedge \text{Open}(C_1) \wedge \text{Color}(\text{Chair}, C(C_1))$$

$$b_3 = \text{Result}(b_1, \text{Paint}(\text{Table}, C_1))$$

$$= \text{Color}(x, C(x)) \wedge \text{Open}(C_1) \wedge$$

$$\text{Color}(\text{Chair}, C(C_1)) \wedge \text{Color}(\text{Table}, C(C_1))$$

# Conditional Effects

- So far, we have only considered actions that have the **same effects** on all states where the preconditions are satisfied.
- This means that any initial belief state that is a conjunction is updated by the actions to a belief state that is also a conjunction.
- But some actions are best expressed with conditional effects.
- This is especially true if the effects are non-deterministic, but in a bounded way.

# Extending action representations

- Disjunctive effects:  
 $Action(Left, \text{Precond} : AtR, \text{Effect} : AtL \vee AtR)$
  - Conditional effects:  
 $Action(Suck,$   
    Precond:  
    Effect: (**when**  $AtL : CleanL$ )  $\wedge$  (**when**  $AtR : CleanR$ ))
  - Combination:  
 $Action(Left,$   
    Precond:  $AtR$   
    Effect:  $AtL \vee (AtL \wedge (\text{when } CleanL : \neg CleanL))$ )
- Conditional steps: **if**  $AtL \wedge CleanL$  **then** *Right* **else** *Suck*

# Contingent Planning: Using the Percepts

The formal representation of the plan we saw earlier:

[*LookAt(Table)*, *LookAt(Chair)*)

**if** *Color(Table, c)  $\wedge$  Color(Chair, c)* **then** *NoOp*

**else** [*RemoveLid(C<sub>1</sub>)*, *LookAt(C<sub>1</sub>)*, *RemoveLid(C<sub>2</sub>)*, *LookAt(C<sub>2</sub>)*,

**if** *Color(Chair, c)  $\wedge$  Color(can, c)* **then** *Paint(Table, can)*

**elseif** *Color(Table, c)  $\wedge$  Color(can, c)* **then** *Paint(Chair, can)*

**else** [*Paint(Chair, C<sub>1</sub>)*, *Paint(Table, C<sub>1</sub>)*]]]

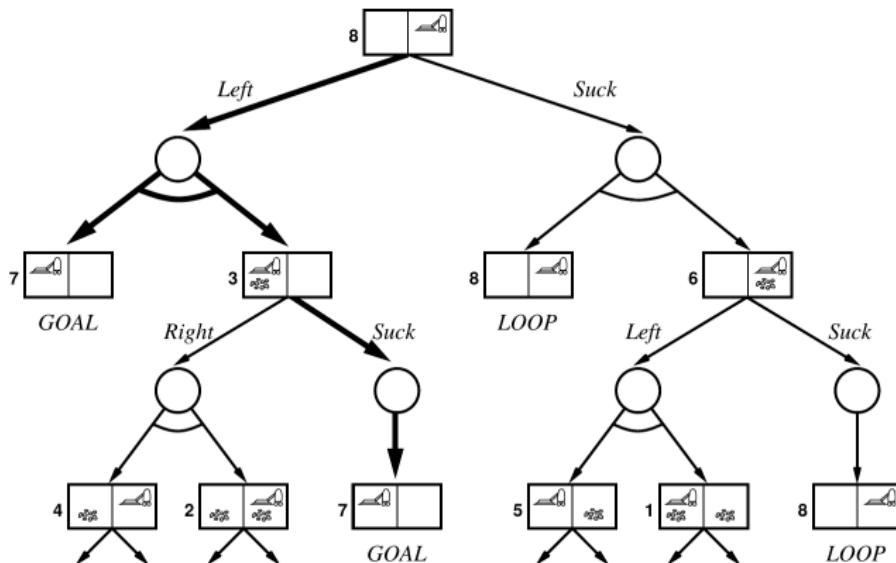
- Variables (e.g., *c*) are **existentially quantified**.

# Games against nature

- Conditional plans should succeed regardless of circumstances
- Nesting conditional steps results in trees
- Similar to adversarial search, [games against nature](#)
- Game tree has state nodes and chance nodes where nature determines the outcome
- Definition of solution: A subtree with
  - a goal node at every leaf
  - specifies one action at each state node
  - includes every outcome at chance node
- AND-OR graphs can be used in similar way to the minimax algorithm (basic idea: find a plan for every possible result of a selected action)

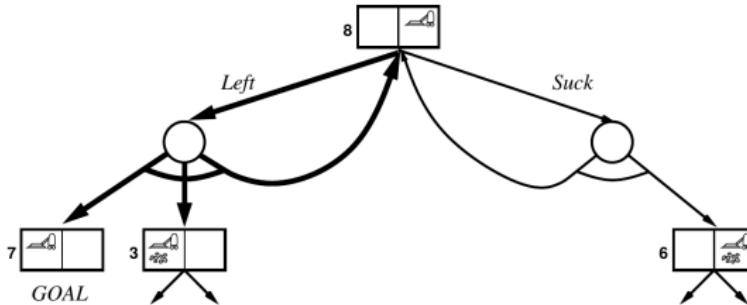
# Example: “double Murphy” vacuum cleaner

- This wicked vacuum cleaner sometimes deposits dirt when moving to a clean destination or when sucking in a clean square
- Solution: [Left, if CleanL; then [] else Suck]



# Acyclic vs. cyclic solutions

- If identical state is encountered (on same path), terminate with failure (if there is an acyclic solution it can be reached from previous incarnation of state)
- However, sometimes all solutions are cyclic!
- E.g., “triple Murphy” (also) sometimes fails to move.
- Plan [Left, if CleanL then [] else Suck] does not work anymore
- Cyclic plan:  
[ $L : Left, \text{ if } AtR \text{ then } L \text{ elseif } CleanL \text{ then } [] \text{ else } Suck$ ]

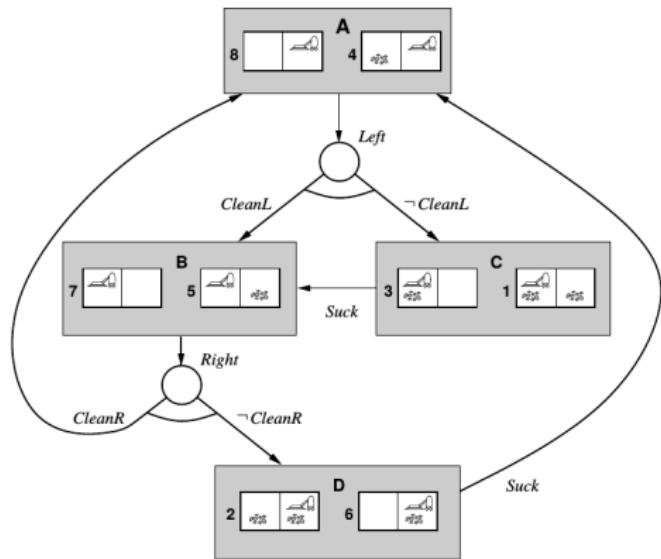


# Nondeterminism and partially observable environments

“alternate double Murphy”:

- Vacuum cleaner can sense cleanliness of square it's in, but not the other square, and
- dirt can sometimes be left behind when leaving a clean square.
- Plan in fully observable world: “Keep moving left and right, sucking up dirt whenever it appears, until both squares are clean and in the left square”
- But now goal test cannot be performed!

# Housework in partially observable worlds



# Conditional planning, partial observability

- Basically, we can apply our AND-OR-search to belief states (rather than world states)
- Full observability is special case of partial observability with singleton belief states
- Is it really that easy?
- Not quite, need to describe
  - representation of belief states
  - how sensing works
  - representation of action descriptions

# Summary

- Methods for planning and acting in the real world
- Dealing with indeterminacy
- Contingent planning: use percepts and conditionals to cater for all contingencies.
- Fully observable environments: AND-OR graphs, games against nature
- Partially observable environments: belief states, action and sensing
- Next time: Planning and acting in the real world II

# Inf2D 20: CW2 (12.5%)

Michael Herrmann

University of Edinburgh, School of Informatics

09/03/2017

**informatics**



Credits: The content of this lecture was prepared by Michael Rovatsos and follows R&N

# Overview

- Golog and Prolog
- The Taxi Problem

- Algorithms in Logics
- Language based on Situation Calculus
  - Situations
  - Actions
  - Predicates (atemporal, fluent)
  - Axioms (descriptive, precondition, successor state)
- How to automatize inference in this language?
  - We have Prolog
- Golog interpreter is written in Prolog
  - Both have similar syntax

- Declarative language (relations)
  - Knowledge base (facts and rules)
  - Queries (can you prove/satisfy this?)
- User can provide a KB and ask a query.
- Prolog uses resolution (backward chaining) over KB to answer.
- Why Prolog?
  - First Prolog compiler by David H. D. Warren (Edinburgh)
  - The “Edinburgh Prolog” dialect serves as the basis for the syntax of most modern implementations.
  - Prolog has been used in Watson (IBM).

# The Syntax of Prolog

- Predicates & constants start with lower-case.
- Variables start with upper-case.
- A **Term** is a constant, variable or composite from other terms.
- No quantifiers
  - variables in KB are implicitly universal
  - queries ask for satisfaction (existential-like)

,	conjunction
;	disjunction
:-	if (rev. implication)
.	end of sentence
=	unification (also prefix)
_	anonymous variable

# Example of Prolog syntax

- **KB:**

```
horse(bluebeard).
```

```
offspring(charlie, bluebeard).
```

```
horse(H) :- offspring(H,X), horse(X).
```

- **Queries:**

```
horse(charlie).
```

```
horse(X).
```

```
lion(X).
```

“:-” is only used in rules. Rules cannot be queries.

# Prolog Demo

- Create a KB file, e.g., `demo.pl`
- Load the file in Prolog with command `[demo].`
- Useful predicates:
  - `listing/0` : list all facts in KB
  - `assert/1` : save a fact into KB
  - `retract/1` : remove a fact from KB
  - `halt/0` : exit Prolog

# Prolog Block World Demo

- You can run `./plan.sh sample-blocks.pl`
- Runs on DICE machines or on Linux machines with installed SWI Prolog (“`swipl`”).
- Modify the program in order to find a solution of the Susman anomaly (sorry, no marks for this).

# Prolog Demo

KB

edge/2.

edge(a,b).

edge(a,e).

edge(b,d).

edge(b,c).

edge(c,a).

edge(e,b).

Query1

edge(a,b).

Query2

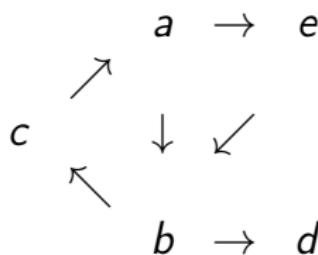
edge(a,b) , edge(b,c).

Query3

edge(a,b) , edge(a,c).

Query4

edge(a,b) ; edge(a,c).



Query5

edge(a,X).

# Prolog Rules

- An atom in prolog is anything that can be represented internally by a single unit (e.g. string, variable, empty set)
- Rule:

Atom :- Atom1, Atom2, ..., AtomN.

e.g.

```
tedge(X,Y) :- edge(X,Z) , edge(Z,Y).
```

nodes X and Y have distance 2

```
path(X,Y) :- edge(X,Y).  
path(X,Y) :- edge(X,Z) , path(Z,Y).
```

there is a path from X to Y.

# Numbers in Prolog

- Numbers are constants.
- Some arithmetic operators:

+ - / \* > < >= <= is ==

**KB:**

```
price(book,10).  
price(coffee,3).
```

**Queries:**

```
canBuy(2,Item).
```

```
canBuy(6,Item).
```

```
canBuy(Money,Item) :-  
    price(X,P), Money>=P, Item=X.
```

```
canBuy(12,Item).
```

# Numbers in Prolog

Create alternative definition of numbers:

Use a predicate “s” that stands for successor!

Rule:

```
can(A,X) :- A = move, X = s(s( ));  
          A = jump, X = s(s(s( ))).
```

Queries:

```
can(jump, s(s(s(s(s(0)))))).  
can(move, s(0)).  
can(X, s(s(s(s(s(0)))))).
```

# Taxi Dispatch Agent

You are asked to:

- ① Formalise the problem in the Situation Calculus
- ② Implement in Golog

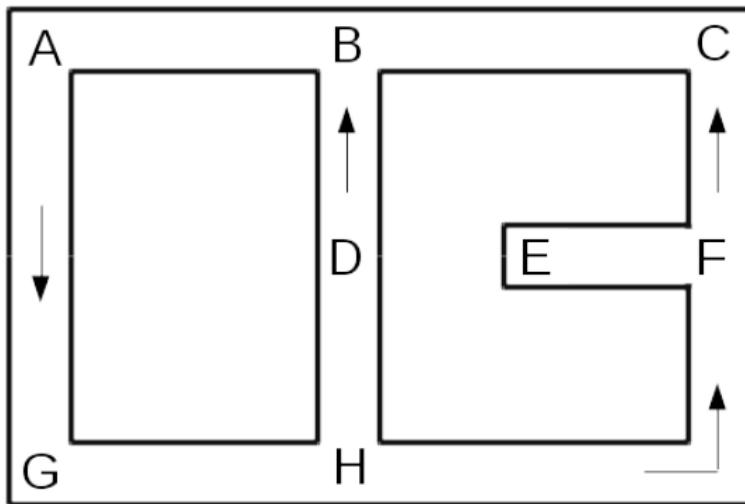
# Tips

- Start with: The first two chapters of “Learn Prolog Now!” by Patrick Blackburn, Johan Bos, and Kristina Striegnitz (online version);
- Play with the block example;
- Start with the simple tasks and build up.

# Problem Description

- Steps
  - ① Find and list the facts
  - ② Represent those facts in a compact representation
- Facts
  - Map is discrete
  - 8 locations with streets connecting them
  - Taxi is in a locations room (e.g. A)
  - Passenger is static and its location is known
  - Dispatcher can be reached at all times for updated information

# Problem Description



Street map with pick-up and drop-off locations.

# Modelling

- Create a model using situation calculus
- atemporal predicates, fluents
- as compact as possible
  - remove unused predicates or fluents

# Part 1: Formalise

- The environment:
  - which locations are connected?
  - where are passengers and taxis?
  - what is the state of the taxi?
- The actions:
  - what can the taxi do?
  - when can it do it? (preconditions)
  - how do actions change the environment? (effects)

# Tips

- If locations  $A$  and  $B$  are bidirectionally connected the taxi can go from  $A$  to  $B$  and from  $B$  to  $A$
- If locations  $A$  and  $B$  are unidirectionally connected the taxi can go either  $A$  to  $B$  or from  $B$  to  $A$
- To pick up a passenger, the taxi and the passenger must be in the same location

## Parts 2 & 3: Implement and extend

- Part 2:
  - Translate Part 1 into the syntax of Golog
  - Test on some initial states & goals
- Part 3:
  - Add more predicates
  - Change your axioms to suit the new predicates
  - Test

# Assignment package

- 8 files
- A Golog-based planner - only available on DICE machines
- Domain and problem template files
- All text should go in answer.txt
- Blocks world example.

- Being concise will help you:
  - Remove unused predicates
  - Remove arguments in predicates that don't need them.
- Balance abstraction:
  - Don't add a new predicate for every action.
  - Don't use same predicates for things that are essentially different.

# Submission Package

All original files

- Part 1 answers should be in answer.txt
- Part 2
  - a domain file: domain-task21.pl
  - problem instances:  
instance-task22.pl, instance-task23.pl, instance-task24.pl
- Part 3
  - domain-task31.pl, instance-task31.pl
  - domain-task32.pl, instance-task32.pl
  - domain-task33.pl, instance-task33a.pl and  
instance-task33b.pl

## To conclude

- Worth 12.5% of the final mark
- Deadline: 4pm on Monday 28st March
- Electronic submission through submit software (available on any DICE machine)

# Inf2D 19: – Planning and Acting in the Real World II

Michael Herrmann

University of Edinburgh, School of Informatics

07/03/2017

**informatics**



Credits: The content of this lecture was prepared by Alex Lascarides and follows R&N

# Where are we?

Last time . . .

- Looked at methods for real-world planning
- Sensorless planning and contingent planning
- Fully and partially observable environments

Today . . .

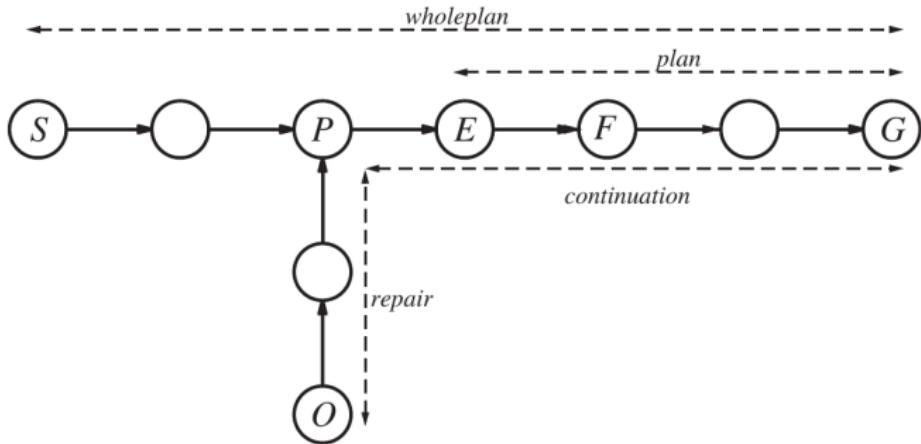
- Planning and acting in the real world II

# Execution monitoring and replanning

- Execution monitoring = checking whether things are going according to plan (necessitated by unbounded indeterminacy in realistic environments)
  - Action monitoring = checking whether previous action was successful and next action is feasible
  - Plan monitoring = checking whether plan was followed so far and remainder of plan is feasible
- Replanning = ability to find new plan when things go wrong (often *repairing* the old plan)
- Taken together these methods yield powerful planning abilities

# Action monitoring and replanning

While attempting to get from  $S$  to  $G$ , a problem is encountered in  $E$ , agent discovers actual state is  $O$  and plans to get to  $P$  and to execute the rest of the original plan.



# Plan monitoring

- Action monitoring can result in suboptimal behaviour
  - executes everything until actual failure
  - can get stuck in loops (going back to  $P$  in the previous example can move the agent again to  $O$  instead of  $E$ )
- Plan monitoring checks preconditions for entire remaining plan
- Can also take advantage of serendipity (unexpected circumstances might make remaining plan easier)
  - If the problem had sent agent to  $F$  instead of  $O$  it should continue and not go back to  $P$ .
- In partially observable environments things are more complex (sensing actions have to be planned for, they can fail in turn, etc.)

# Hierarchical decomposition in planning

- Hierarchical decomposition seems a natural idea to improve planning capabilities.
- Key idea: at each level of the hierarchy, activity involves only small number of steps (i.e. small computational cost)
- Hierarchical task network (HTN) planning: initial plan provides only high-level description, refined by action refinements
- Refinement process continued until plan consists only of primitive actions

# Representing action decompositions

- Each **high level action** (HLA) has (at least) one **refinement** into a sequence of actions.
- The actions in the sequence may be HLAs or primitive.
  - So HLAs form a hierarchy!
- If they're all primitive, then that's an **implementation** of the HLA.

# Example: Go to SF Airport

A plan can have multiple refinements

- Refinement( $Go(Home, SFO)$ ,  
Precond:  $At(Car, Home)$   
Steps: [ $Drive(Home, SFOLongTermParking)$ ,  
 $Shuttle(SFOLongTermParking, SFO)$ ])
- Refinement( $Go(Home, SFO)$ ,  
Precond:  $Cash, At(Home)$   
Steps: [ $Taxi(Home, SFO)$ ])

# Refinements can be Recursive

- Refinement( $\text{Navigate}([a, b], [x, y])$ ,  
Precond:  $a = x, b = y$   
Steps: [ ])
- Refinement( $\text{Navigate}([a, b], [x, y])$ ,  
Precond:  $\text{Connected } [a, b], [a - 1, b]$   
Steps:  $\text{Left}, \text{Navigate } [a - 1, b], [x, y]$ )
- Refinement( $\text{Navigate}([a, b], [x, y])$ ,  
Precond:  $\text{Connected } [a, b], [a + 1, b]$   
Steps:  $\text{Right}, \text{Navigate } [a + 1, b], [x, y]$ )

# High-Level Plans

- High-Level Plans (HLP) are a sequence of HLAs.
- An implementation of a High Level Plan is the concatenation of an implementation of each of its HLAs.
- An HLP achieves the goal from an initial state if **at least one** of its implementations does this.
- Not all implementations of an HLP have to reach the goal state!
- The agent gets to decide which implementation of which HLAs to execute.

# Searching for Primitive Solutions

- The HLA plan library is a hierarchy:
  - (Ordered) Daughters to an HLA are the sequences of actions provided by one of its refinements;
  - Because a given HLA can have more than one refinement, there can be more than one node for a given HLA in the hierarchy.
- This hierarchy is essentially a search space of action sequences that conform to knowledge about how high-level actions can be broken down.
- So you can search this state space for a plan!

# Searching for Primitive Solutions: Breadth First

- Start your plan  $P$  with the HLA [Act]
- Take the first HLA  $A$  in  $P$  (recall that  $P$  is an *action sequence*).
- Do a **breadth-first search** in your hierarchical plan library, to find a refinement of  $A$  whose preconditions are satisfied by the outcome of the action in  $P$  that is prior to  $A$ .
- Replace  $A$  in  $P$  with the steps of this refinement.
- Keep going until your plan  $P$  has no HLAs and either:
  - ① Your plan  $P$ 's outcome is the goal, in which case return  $P$ , or
  - ② Your plan  $P$ 's outcome is not the goal, in which case return *failure*.

# Problems

- Like forward search, you consider lots of irrelevant actions.
- The algorithm essentially refines HLAs right down to primitive actions so as to determine if a plan will succeed.
- This contradicts **common sense!**
- Sometimes you know an HLA will work *regardless* of how it's broken down!
- We don't need to know which route to take to *SFOParking* to know this plan works:

$[Drive(Home, SFOParking), Shuttle(SFOParking, SFO)]$

- We can capture this if we add to HLAs *themselves* a set of preconditions and effects.

# Adding Preconditions and Effects to HLAs

- One challenge in specifying preconditions and effects of an HLA is that the HLA may have more than one refinement, each one with slightly different preconditions and effects!
  - If you refine  $Go(Home, SFO)$  with Taxi action: you need Cash.
  - If you refine it with *Drive*, you don't!
  - This difference may affect your **choice** on how to refine the HLA!
- Recall that an HLA achieves a goal if **one** of its refinements does this.
- And you can choose the refinement!

# Getting Formal

- $s' \in \text{Reach}(s, h)$  iff  $s'$  is reachable from at least one of HLA  $h$ 's refinements, given (initial) state  $s$ .

$$\text{Reach}(s, [h_1, h_2]) = \bigcup_{s' \in \text{Reach}(s, h_1)} \text{Reach}(s', h_2)$$

- HLP  $p$  achieves goal  $g$  given initial state  $s$  iff  $\exists s' \text{ s.t.}$   
$$s' \models g \text{ and } s' \in \text{Reach}(s, p)$$
- So we should search HLPs to find a  $p$  with this relation to  $g$ , and then focus on refining it.
- But a pre-requisite to this algorithm is to define  $\text{Reach}(s, h)$  for each  $h$  and  $s$ .
- In other words, we still need to determine how to represent effects (and preconditions) of HLAs ...

# Defining Reach

- A primitive action makes a fluent true, false, or leaves it unchanged.
- But with HLAs you sometimes get to **choose**, by choosing a particular refinement!
- We add new notation to reflect this:

$\overset{\sim}{+} A$ : you can possibly add  $A$  (or leave  $A$  unchanged)

$\overset{\sim}{-} A$ : you can possibly delete  $A$  (or leave  $A$  unchanged)

$\overset{\sim}{\pm} A$ : you can possibly add  $A$ , or possibly delete  $A$   
(or leave  $A$  unchanged)

- You should now *derive* the correct preconditions and effects from its refinements!

# Our SFO Example

- Refinement( $Go(Home, SFO)$ ,  
Precond:  $At(Car, Home)$   
Steps: [ $Drive(Home, SFOLongTermParking)$ ,  
 $Shuttle(SFOLongTermParking, SFO)]$ )
- Refinement( $Go(Home, SFO)$ ,  
Precond:  $Cash, At(Home)$   
Steps: [ $Taxi(Home, SFO)]$ )

# The 'Primitive' Actions

- Action( $Taxi(a, b)$ ,  
Precond:  $Cash, At(Taxi, a)$   
Effect:  $\neg Cash, \neg At(Taxi, a), At(Taxi, b)$ )
- Action( $Drive(a, b)$ ,  
Precond:  $At(Car, a)$   
Effect:  $\neg At(Car, a), At(Car, b)$ )
- Action( $Shuttle(a, b)$ ,  
Precond:  $At(Shuttle, a)$   
Effect:  $\neg At(Shuttle, a), At(Shuttle, b)$ )

# Deriving the Preconds and Effects of the HLA

- $\neg \text{Cash}$  is Effect of one HLA refinement, but not the other.
- So  $\approx \neg \text{Cash}$  in HLA Effect!

Not so Simple!

- Similar argument for  $\text{At}(\text{Car}, \text{SFOParking})$
- But you can't choose the combination:  
 $\neg \text{Cash} \wedge \text{At}(\text{Car}, \text{SFOParking})$
- Solution is to write approximate descriptions.

# Approximate Descriptions

## Optimistic Description: $\text{Reach}^+(s, h)$

- Take union of all possible outcomes from all refinements.
- So this includes  $\neg \text{Cash}$  and  $\overset{\sim}{+} \text{At}(\text{Car}, \text{SFOParking})$ .
- This overgenerates reachable states.

## Pessimistic Description: $\text{Reach}^-(s, h)$

- Only states that satisfy effects from *all* refinements survive.
- So this does not include  $\neg \text{Cash}$  or  $\overset{\sim}{+} \text{At}(\text{Car}, \text{SFOParking})$ .
- This undergenerates reachable states.

$$\text{Reach}^-(s, h) \subseteq \text{Reach}(s, h) \subseteq \text{Reach}^+(s, h)$$

# Algorithm for Finding a Plan

- Two Important Facts:
  - ① If  $\exists s' \in \text{REACH}^-(s, h)$  s.t.  $s' \models g$ , you know  $h$  can succeed.
  - ② If  $\neg \exists s' \in \text{REACH}^+(s, h)$  s.t.  $s' \models g$ , you know  $h$  will fail!
- The Algorithm:
  - Do breadth first search as before.
  - But now you can **stop searching** and **implement instead** when you reach an  $h$  where (1) is true.
  - And you can **drop**  $h$  (and all its refinements) when (2) is true.
  - If (1) and (2) are both false for the current  $h$ , then you don't know if  $h$  will succeed or fail, but you can find out by refining it.

# Summary

- Execution monitoring: checking success of execution
- Replanning: repairing plans in case of failure
- HLAs and HLPs
- Using refinements and preconditions and effects of primitive actions to *approximate* which states are reachable.
- Such approximate descriptions of HLAs help to inform search and when to refine an HLP so as to reach a goal.
- Next time: **Acting under Uncertainty** (Peggy Seriés)



# **Inf2D-Reasoning and Agents**

## Spring 2017

Lecture (2)1: Acting Under Uncertainty

Peggy Seriès, [pseries@inf.ed.ac.uk](mailto:pseries@inf.ed.ac.uk)

---

**THERE IS NOTHING  
CERTAIN, BUT THE  
UNCERTAIN**

---

Proverb

# Acting under uncertainty

---

- So far we have always assumed that propositions are assumed to be true, false, or unknown
- But in reality, we have **hunches rather than complete ignorance or absolute knowledge**
- Approaches like conditional planning and replanning handle things that might go wrong
- But they don't tell us how **likely** it is that something might go wrong. ...
- **Rational decisions** (i.e. ‘the right thing to do’) depend on the relative importance of various goals and the likelihood that (and degree to which) they will be achieved

# Uncertain Reasoning: the toothache example

---



- A rule like :  
 $\text{Toothache} \Rightarrow \text{cavity}$   
is clearly wrong
- $\text{Toothache} \Rightarrow \text{Cavity} \vee \text{Gum Disease} \vee \text{Abscess} \dots$   
That list could be unlimited...
- Causal rules like:  $\text{Cavity} \Rightarrow \text{Toothache}$   
can also cause problems
- Cavity and toothache can even be unconnected ...

# Uncertain Reasoning: the toothache example

---

- Logic fails for 3 reasons:
  - **Complexity/ Laziness:** (can be impractical to include all antecedents and consequents in rules, and/or too hard to use them)
  - **Theoretical ignorance:** (don't know a rule completely)
  - **Practical ignorance:** (don't know the current state)
- One possible approach: express **degrees of belief** in propositions using probability theory
- Probability can **summarise the uncertainty** that comes from our 'laziness' and ignorance
- Probabilities between 0 and 1 express the degree to which we believe a proposition to be true

# Acting under uncertainty

- Goal: deliver a passenger on time to the airport.
- Airport is 5 miles away
- What time should the taxi come?
- A90? A180? A1440?
- Rational decision: maximise the agent's performance measure, given some knowledge of the uncertainty in the environment



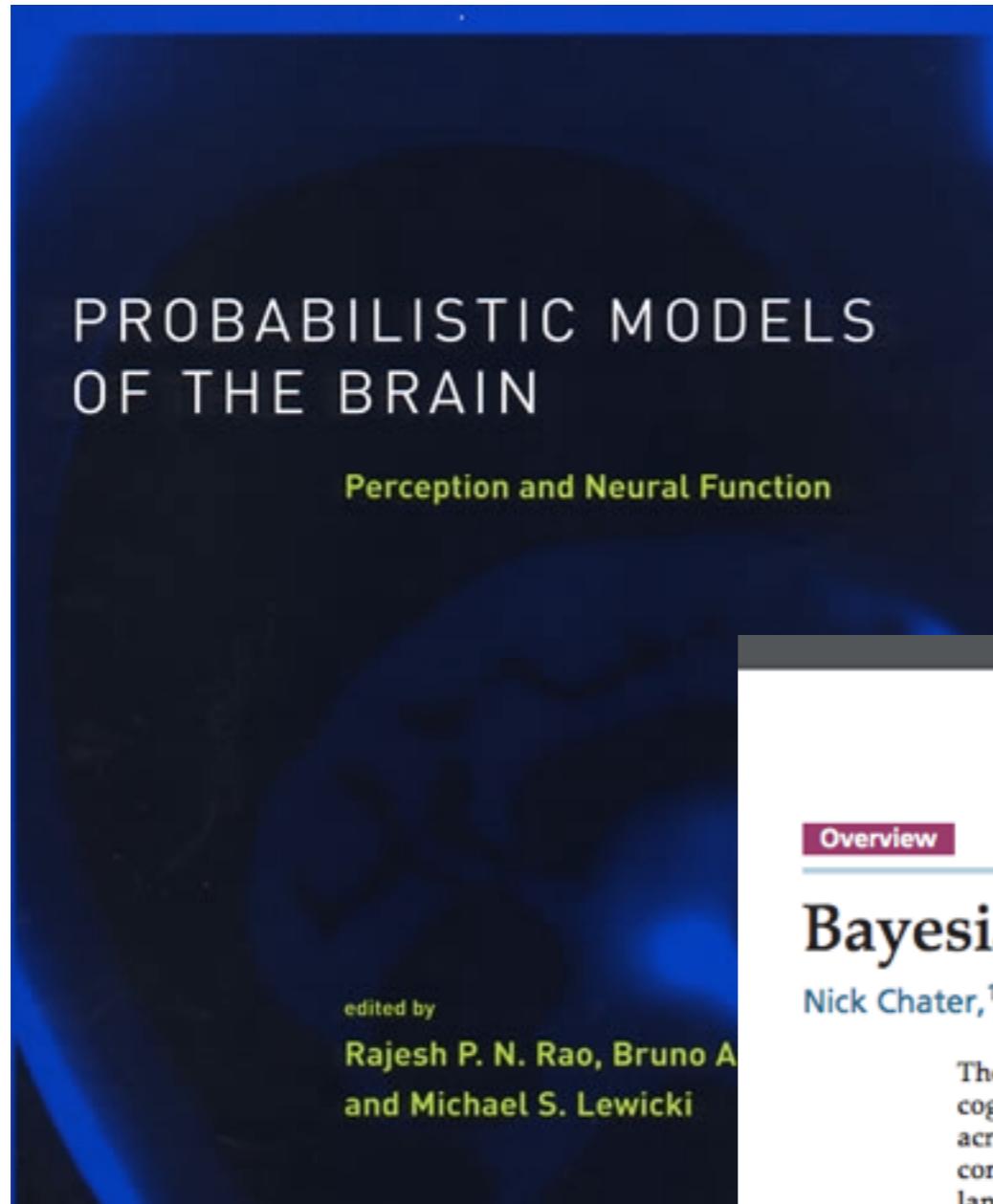
- Probability about states  
= **Probability theory**
- Uses: preferences about outcomes and states (e.g. cost of missing the flight, cost of long wait, stuck in traffic ..)  
= **Utility theory**

# Uncertainty and rational decisions

---

- A general theory of rational decision making
- Decision theory = probability theory + utility theory
- Foundation of decision theory:  
An agent is rational if and only if it chooses the action that yields the highest expected utility, i.e. averaged over all possible outcomes of the action
- Principle of Maximum Expected Utility
- General principle for building agents able to cope with real-world environments

# Are we (humans) rational?



- The human brain is thought to approximate **probabilistic reasoning** and behaviour can often be modelled as **maximising expected utility**.

## Overview

### Bayesian models of cognition

Nick Chater,<sup>1\*</sup> Mike Oaksford,<sup>2</sup> Ulrike Hahn<sup>3</sup> and Evan Heit<sup>4</sup>



There has been a recent explosion in research applying Bayesian models to cognitive phenomena. This development has resulted from the realization that across a wide variety of tasks the fundamental problem the cognitive system confronts is coping with uncertainty. From visual scene recognition to on-line language comprehension, from categorizing stimuli to determining to what degree an argument is convincing, people must deal with the incompleteness of the information they possess to perform these tasks, many of which have important survival-related consequences. This paper provides a review of Bayesian models

# Are we (humans) rational?

---



- but not always ..  
distorted views  
about probabilities,  
or utilities ...



# Are we (humans) rational?

---

- What do you prefer:
  - A: 100% chance of £3000
  - B: 80% chance of £4000
- C: 25% chance of £3000
- D: 20% chance of £4000
- 80% of you chose lottery A over lottery B.
- 89% of you chose lottery D over lottery C.
- So lots of you chose A and D, which is irrational!
- If  $U(3000) > 0.8 * U(4000)$ , then  $0.25 * U(3000) > 0.2 * U(4000)!!$
- Our ability to MEU also affected by emotion, social relationships
- In fact, we're predictably irrational.



# Probabilities

---

- **Unconditional/prior probability** = degree of belief in a proposition in the absence of any other information
- Can be between 0 and 1, write as  $P(\text{Cavity} = \text{true}) = 0.1$  or  $P(\text{cavity}) = 0.1$
- Most of the time we have some information = **evidence**:  
 $P(\text{cavity} | \text{toothache})$ .  
= **conditional probability**
- When making decisions, an agent has to condition *on all the information available*.  
 $P(\text{cavity} | \text{toothache})=0.6$   
means :  
Whenever toothache is true and we *have no further information* conclude that cavity is true with probability 0.6

# Probabilities

---

- **Random variable**, a part of the world whose status is unknown, with a **domain** = set of values it can take (e.g. Cavity with domain {true, false})
- Domains can be boolean, discrete or continuous
- Can compose complex **propositions** from statements about random variables (e.g.  $P(\text{cavity} \mid \neg \text{toothache} \wedge \text{teen}) = 0.1$ )
- $P(\text{Weather} = \text{sunny}) = 0.7$   
 $P(\text{Weather} = \text{rain}) = 0.2$   
 $P(\text{Weather} = \text{cloudy}) = 0.1$   
Write  $\mathbf{P}(\text{Weather}) = \{0.7, 0.2, 0.1\}$

**P** defines a **probability distribution** for the RV Weather

- For continuous variables, we use **probability density function a.k.a pdfs** (we cannot enumerate values) e.g.

$$P(\text{NoonTemp} = x) = \text{Uniform}_{[18C, 26C]}(x)$$

# Probabilities

---

- For a mixture of several variables, we obtain a **joint probability distribution (JPD)** – cross-product of individual distributions  
e.g.  $P(\text{Weather}, \text{Cavity})$
- A JPD (“joint”) describes one’s uncertainty about the world, and completely determines a probability model.

# Conditional Probabilities

---

- Can be defined using unconditional probabilities:  $P(a|b) = P(a \wedge b)/P(b)$
- Often written as product rule:  $P(a \wedge b) = P(a|b)P(b)$

Intuitively, for  $a \wedge b$  to be true, we need  $b$  to be true, and  $a$  to be true if  $b$  is true

- Good for describing JPDs (which then become “CPDs”) as  $P(X, Y) = P(X|Y)P(Y)$
- Set of equations, not matrix multiplication (!):  
 $P(X = x_1 \wedge Y = y_1) = P(X = x_1|Y = y_1)P(Y = y_1)$   
 $P(X = x_1 \wedge Y = y_2) = P(X = x_1|Y = y_2)P(Y = y_2)$   
.  
.  
 $P(X = x_n \wedge Y = y_n) = P(X = x_n|Y = y_n)P(Y = y_n)$

# The axioms of Probability

---

- **Kolmogorov's axioms** define basic semantics for probabilities:
  1.  $0 \leq P(a) \leq 1$  for any proposition  $a$
  2.  $P(\text{true}) = 1$  and  $P(\text{false}) = 0$
  3.  $P(a \vee b) = P(a) + P(b) - P(a \wedge b)$
- From this, a number of useful facts can be derived, e.g:  
 $P(\neg a) = 1 - P(a)$
- For variable  $D$  with domain  $\{d_1, \dots, d_n\}$ ,  $\sum_{i=1}^n P(D = d_i) = 1$
- And so any JPD over finite variables sums to 1

# Summary

---

- Explained why logic in itself is insufficient to model uncertainty
- Discussed principles of decision making under uncertainty
- Decision theory, MEU principle
- Probability theory provides useful tools for quantifying degree of belief/uncertainty in propositions
- propositions, random variables, Probability distributions, conditional probabilities
- Axioms of probability
- Next time: Bayes and Inference



# **Inf2D-Reasoning and Agents**

## Spring 2017

Lecture (2)2: Probabilities and Bayes

Peggy Seriès, [pseries@inf.ed.ac.uk](mailto:pseries@inf.ed.ac.uk)

Based on previous slides by A. Lascarides

# Where are we?

---

- Last time . . .  
Talked about uncertainty, **random variables**, **probability distributions**
- Introduced basics of **decision theory** (probability theory + utility)
- Introduced basic probability notation and **axioms**
- Today . . .  
Probabilities and Bayes' Rule

# Inference with joint probability distributions

---

- Problem:  
Given some observed **evidence** and a **query proposition**,  
how can we compute the **posterior probability** of that  
**proposition**?
- e.g. given that we have a toothache and that the dentist's probe is catching on something on my tooth, what is the probability I have a cavity? (diagnosis)

$$P(\text{cavity} \mid \text{toothache}, \text{catch})=?$$

- If we have a knowledge base, e.g. in the form of a **joint probability distributions** table (JPDs), we can answer such questions.

## Example



- Domain consisting only of Boolean variables:  
Toothache, Cavity and Catch
- The JPD is a  $2 \times 2 \times 2 = 2^3$  table — our “knowledge base”.

	toothache		$\neg$ toothache	
	catch	$\neg$ catch	catch	$\neg$ catch
cavity	0.108	0.012	0.072	0.008
$\neg$ cavity	0.016	0.064	0.144	0.576

- e.g.  $p(\text{cavity}, \text{catch}, \text{toothache}) = 0.108$ .
- Probabilities (table entries) sum to 1.
- We can compute probability of any proposition, e.g.  
 $P(\text{catch} \vee \text{cavity})$   
 $= 0.108 + 0.016 + 0.072 + 0.144 + 0.012 + 0.008 = 0.36$

# Marginalisation, conditioning & normalisation

---

	<i>toothache</i>		$\neg$ <i>toothache</i>		
	<i>catch</i>	$\neg$ <i>catch</i>	<i>catch</i>	$\neg$ <i>catch</i>	
<i>cavity</i>	0.108	0.012	0.072	0.008	
$\neg$ <i>cavity</i>	0.016	0.064	0.144	0.576	

- How do we compute  $P(\text{cavity})=?$
- We **sum up** the probability for each possible value of the other variables

$$\begin{aligned}P(\text{cavity}) &= P(\text{cavity}, \text{toothache}, \text{catch}) + P(\text{cavity}, \text{toothache}, \neg\text{catch}) \\&\quad + P(\text{cavity}, \neg\text{toothache}, \text{catch}) + P(\text{cavity}, \neg\text{toothache}, \neg\text{catch}) \\&= 0.108 + 0.012 + 0.072 + 0.008 = 0.2\end{aligned}$$

- This is called **marginalisation**:  $P(Y) = \sum_z P(Y, z)$

- **Conditioning** – variant using the product rule:

$$P(Y) = \sum_z P(Y|z)P(z)$$

# Marginalisation, conditioning & normalisation

	toothache		¬toothache	
	catch	¬catch	catch	¬catch
cavity	0.108	0.012	0.072	0.008
¬cavity	0.016	0.064	0.144	0.576

- Computing **conditional probabilities**

$$\begin{aligned} P(\text{cavity}|\text{toothache}) &= \frac{P(\text{cavity} \wedge \text{toothache})}{P(\text{toothache})} && \text{definition of conditional probability.} \\ &= \frac{0.108 + 0.012}{0.108 + 0.012 + 0.016 + 0.064} = 0.6 \end{aligned}$$

- Denominator = **Normalisation** ensures probabilities sum to 1.  
Normalisation constants often denoted by  $\alpha$

$$\begin{aligned} \mathbf{P}(\text{Cavity}|\text{toothache}) &= \alpha \mathbf{P}(\text{Cavity}, \text{toothache}) \\ &= \alpha [\mathbf{P}(\text{Cavity}, \text{toothache}, \text{catch}) + \mathbf{P}(\text{Cavity}, \text{toothache}, \neg\text{catch})] \\ &= \alpha [\langle 0.108, 0.016 \rangle + \langle 0.012, 0.064 \rangle] = \alpha \langle 0.12, 0.08 \rangle = \langle 0.6, 0.4 \rangle \end{aligned}$$

# General Inference Procedure

---

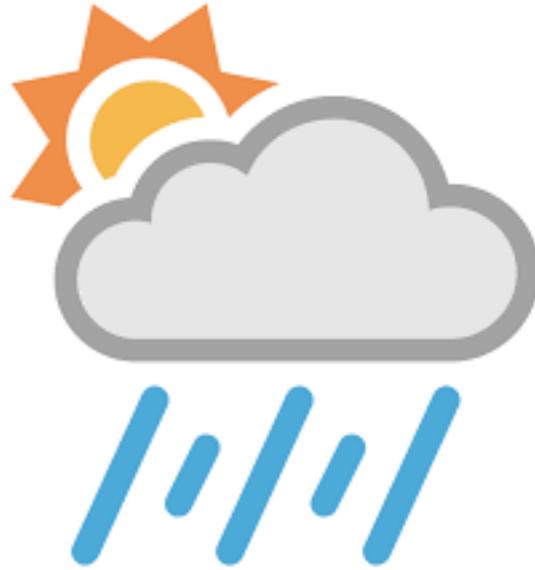
- Let  $X$  be a **query** variable (e.g. Cavity),  $E$  set of **evidence** variables (e.g. Toothache) and  $e$  their observed values,  $Y$  remaining unobserved variables
- Query evaluation:

$$P(X|e) = \alpha P(X, e) = \alpha \sum_y P(X, e, y)$$

- $X$ ,  $E$ , and  $Y$  constitute complete set of variables, i.e.  $P(x, e, y)$  simply a subset of probabilities from the JPD
- For every value  $x_i$  of  $X$ , sum over all values of every variable in  $Y$  and normalise the resulting probability vector
- Does not scale well ...requires input table of size  $O(2^n)$  and takes  $O(2^n)$  steps for  $n$  Boolean variables

# Independence

---

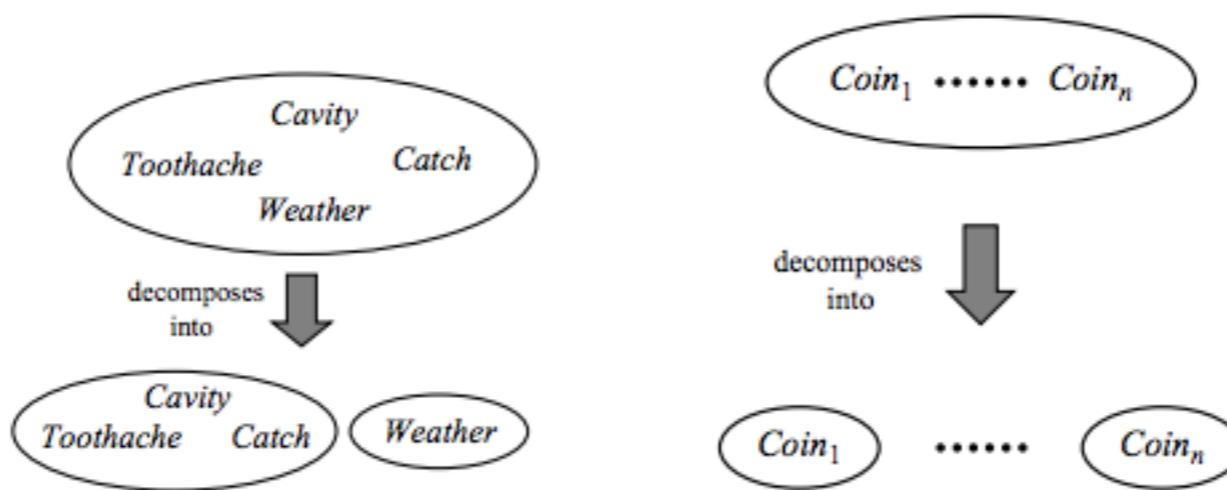


- We extend our JPD with variable Weather={sunny, rain, cloudy, snow}.  
 $P(Toothache, \text{Catch}, \text{Cavity}, \text{Weather})$  now has  $2 \times 2 \times 2 \times 4$  entries= 32.
- But .. what is the relationship between old and new JPD?
- Can compute  
 $P(\text{toothache}, \text{catch}, \text{cavity}, \text{cloudy})$   
=  $P(\text{cloudy}|\text{toothache}, \text{catch}, \text{cavity}) P(\text{toothache}, \text{catch}, \text{cavity})$
- And since **the weather does not depend on dental stuff**, we expect that:  **$P(\text{cloudy}|\text{toothache}, \text{catch}, \text{cavity}) = P(\text{cloudy})$**
- So:  $P(\text{Toothache}, \text{Catch}, \text{Cavity}, \text{Weather}) = P(\text{Weather})P(\text{Toothache}, \text{Catch}, \text{Cavity})$
- One 4-element and one 8-element table rather than a 32-table!

# Independence

---

- This is called independence, usually written as:  
 $P(X|Y) = P(X)$  or  $P(Y|X) = P(Y)$  or  $P(X,Y) = P(X)P(Y)$
- Depends on **domain knowledge**; can factor joint distributions



- Such independence assumptions can help to dramatically **reduce complexity**
- Independence assumptions are sometimes *necessary* even when not entirely justified, so as to make probabilistic reasoning in the domain practical (more later).

# Bayes' Rule

---



- Bayes' rule is derived by writing the **product rule** in two forms and equating them:

$$\left. \begin{array}{l} P(a \wedge b) = P(a|b)P(b) \\ P(a \wedge b) = P(b|a)P(a) \end{array} \right\} \Rightarrow P(b|a) = \frac{P(a|b)P(b)}{P(a)}$$

- General case for multivaried variables using background evidence e:

$$P(Y|X, e) = \frac{P(X|Y, e)P(Y|e)}{P(X|e)}$$

- Useful because often we have good estimates for three terms on the right and are interested in the fourth

# Bayes' Rule for Diagnosis



- Bayes' rule particularly useful for diagnosis

$$P(\text{disease}|\text{symptom}) = \frac{P(\text{symptom}|\text{disease})P(\text{disease})}{P(\text{symptom})}$$

↓                      ↓                      ↓

Likelihood that if you have the disease  
you'll develop the symptom      Prevalence of the disease  
Prevalence of the symptom

- Because causal knowledge (from cause to symptoms) is usually more robust than diagnostic knowledge (from symptoms to causes) (e.g.  $P(d|s)$  will go up if  $P(d)$  goes up due to epidemic)
  - Bayes rule underlies most modern AI systems for probabilistic inference !  
<https://www.youtube.com/watch?v=BcvLAv-JRss>

# Applying Bayes Rule

---



- Example: meningitis causes stiff neck with 50%, prevalence of meningitis  $P(m)=1/50000$ , prevalence of stiff neck  $P(s)=1/20$ ,

I have a stiff neck. What is the probability of having meningitis?

$$P(m|s) = \frac{P(s|m)P(m)}{P(s)} = \frac{\frac{1}{2} \times \frac{1}{50000}}{\frac{1}{20}} = \frac{1}{5000}$$

- Previously, we were able to avoid calculating probability of evidence ( $P(s)$ ) by using normalisation
- With Bayes' rule:  $P(M|s) = \alpha < P(s|m)P(m), P(s|\neg m)P(\neg m) >$
- Usefulness of this depends on whether  $P(s|\neg m)$  is easier to calculate than  $P(s)$

# Combining Evidence

	<i>toothache</i>		$\neg$ <i>toothache</i>	
	<i>catch</i>	$\neg$ <i>catch</i>	<i>catch</i>	$\neg$ <i>catch</i>
<i>cavity</i>	0.108	0.012	0.072	0.008
$\neg$ <i>cavity</i>	0.016	0.064	0.144	0.576

- What if I have **2 sources of evidence** available for my diagnosis?
- $P(\text{cavity}|\text{toothache} \wedge \text{catch})=?$
- Easy to compute if I know the JPD model  
 $P(\text{Cavity}|\text{toothache} \wedge \text{catch}) = \alpha <0.108, 0.016> \approx <0.871, 0.129>$
- but if we don't have JPD, we can use Bayes:  
 $P(\text{Cavity}|\text{toothache} \wedge \text{catch}) = \alpha P(\text{toothache} \wedge \text{catch}|\text{Cavity})P(\text{Cavity})$
- This is basically almost as hard as JPD calculation
- **Conditional independence:** Toothache and Catch are independent given presence/absence of Cavity (both caused by cavity, no effect on each other)
- $P(\text{toothache} \wedge \text{catch}|\text{Cavity}) = P(\text{toothache}|\text{Cavity})P(\text{catch}|\text{Cavity})$

# Conditional Independence

---

- Two variables X and Y are **conditionally independent** given Z if  $P(X,Y|Z) = P(X|Z)P(Y|Z)$
- Equivalent forms  $P(X|Y,Z) = P(X|Z)$  and  $P(Y|X,Z) = P(Y|Z)$
- So in our example:

$$P(\text{Cavity}|\text{toothache} \wedge \text{catch}) = \alpha P(\text{toothache}|\text{Cavity})P(\text{catch}|\text{Cavity})P(\text{Cavity})$$

- As before, this allows us to decompose large JPD tables into smaller ones, grows as  $O(n)$  instead of  $O(2^n)$
- This is what makes probabilistic reasoning methods scalable at all!

# Conditional Independence

---

- Conditional independence assumptions much more often reasonable than absolute independence assumptions
- Naive Bayes model:

$$P(Cause, Effect_1, \dots, Effect_n) = P(Cause) \prod_i P(Effect_i | Cause)$$

© International Statistical Institute

- Based on the cause variable
- Also called Ba
- Works surprisi

he

## Idiot's Bayes—Not So Stupid After All?

David J. Hand<sup>1</sup> and Keming Yu<sup>2</sup>

<sup>1</sup>Department of Mathematics, Imperial College, London, UK. E-mail: d.j.hand@ic.ac.uk

<sup>2</sup>University of Plymouth, UK

### Summary

Folklore has it that a very simple supervised classification rule, based on the typically false assumption that the predictor variables are independent, can be highly effective, and often more effective than sophisticated rules. We examine the evidence for this, both empirical, as observed in real data applications, and theoretical, summarising explanations for why this simple rule might be effective.

*Key words:* Supervised classification; Independence model; Naïve Bayes; Simple Bayes; Diagnosis.

# Summary

---

- Probabilistic inference with full JPDs
- Independence and conditional independence
- Bayes' rule and its applications problems with fairly simple techniques
- Next time: Probabilistic Reasoning with Bayesian Networks



# **Inf2D-Reasoning and Agents**

## Spring 2017

Lecture (2)4:- Exact Inference with Bayesian Networks

Peggy Seriès, [pseries@inf.ed.ac.uk](mailto:pseries@inf.ed.ac.uk)

Based on previous slides by A. Lascarides

# Where are we?

---

- Introduced Bayesian networks
- Allow for compact representation of JPDs
- Methods for efficient representations of CPTs
- But how hard is inference in BNs?
- Today . . . Inference in Bayesian networks

# Inference in BNs

---

- Basic task: compute **posterior distribution for set of query variables given some observed event** (i.e. assignment of values to evidence variables)
- Formally: determine  $P(X|e)$  given query variables X, evidence variables E (and non-evidence or hidden variables Y)
- Example:  
 $P(\text{Burglary}|\text{JohnCalls} = \text{true}, \text{MaryCalls} = \text{true}) = ?$

First we will discuss **exact algorithms** for computing posterior probabilities then **approximate methods** later

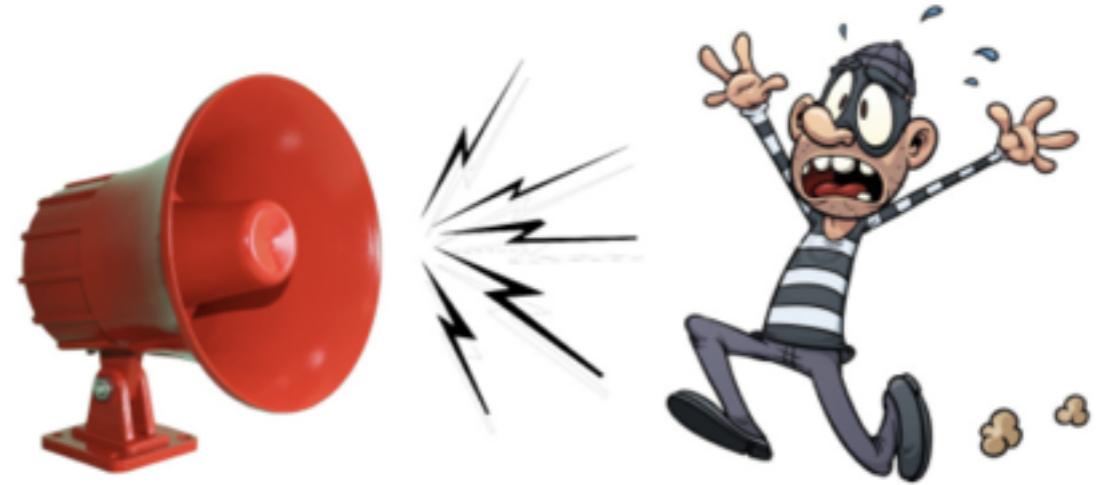
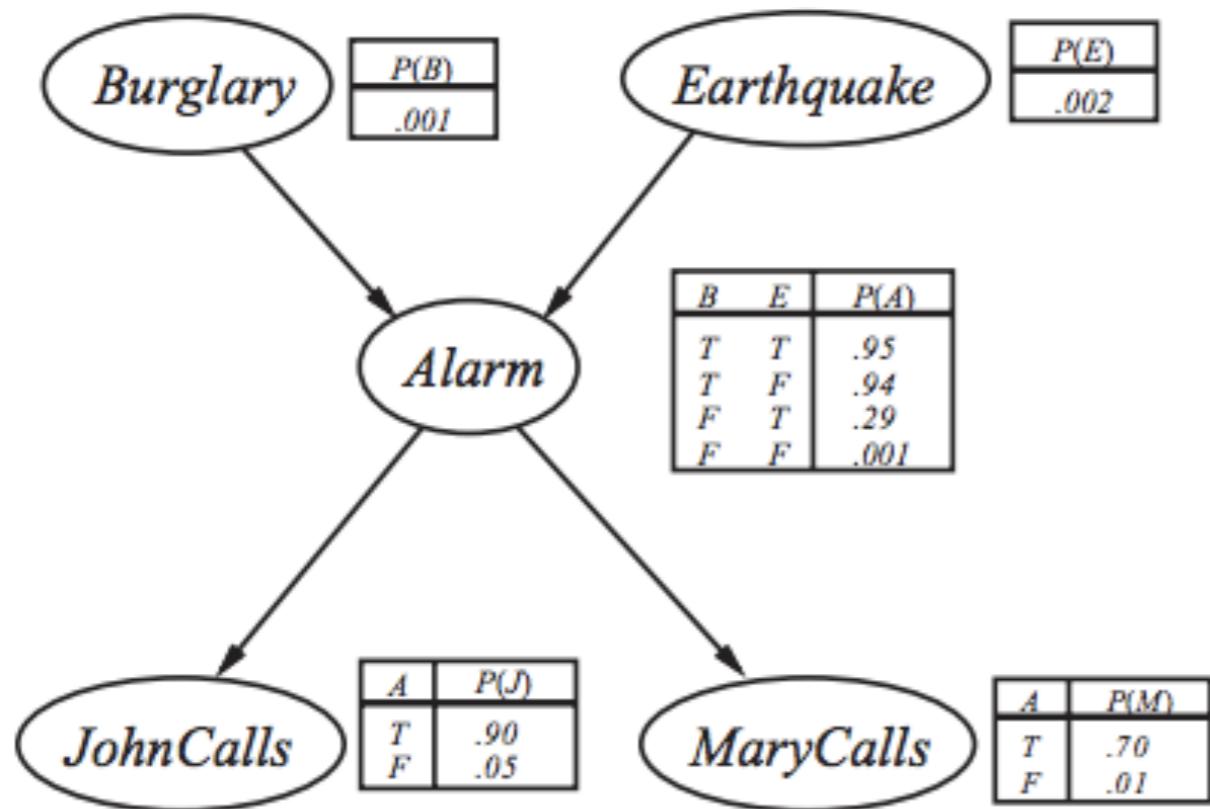
# Inference by enumeration

---

- We have seen that any conditional probability can be computed from a full JPD by summing terms:  
$$P(X|e) = \alpha P(X, e) = \alpha \sum_y P(X, e, y)$$
- Since BN gives complete representation of full JPD, we must be able to answer a query by computing **sums of products of conditional probabilities from the BN**
- Consider query  
$$P(\text{Burglary}|\text{JohnCalls} = \text{true}, \text{MaryCalls} = \text{true}) = P(B|j, m)$$

$$\begin{aligned} P(B|j, m) &= \alpha P(B, j, m) = \alpha \sum_e \sum_a P(B, e, a, j, m) \\ &= \alpha \sum_e \sum_a P(b)P(e)P(a|b, e)P(j|a)P(m|a) \end{aligned}$$

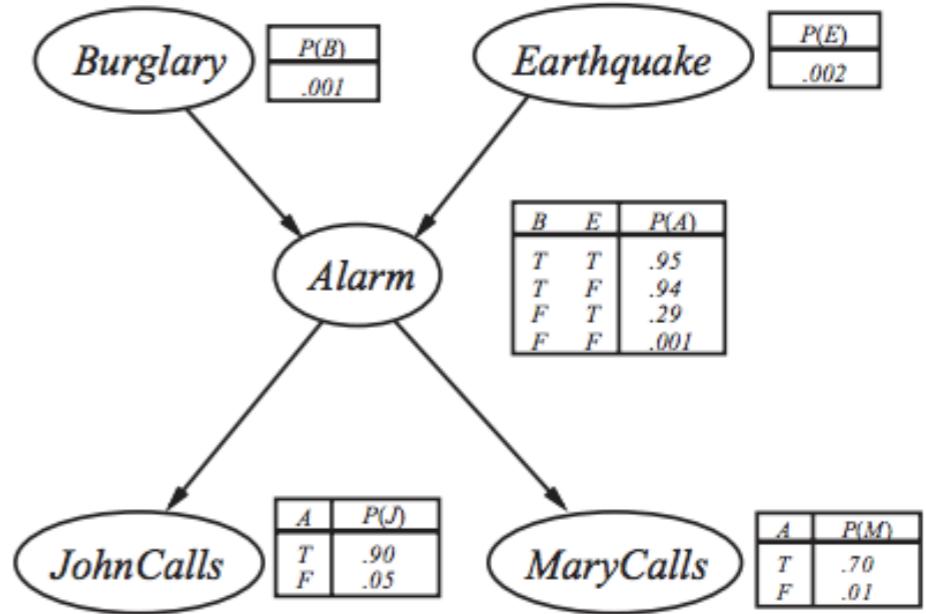
# Example



- New burglar alarm has been fitted, fairly reliable but sometimes reacts to earthquakes
- Neighbours John and Mary promise to call when they hear alarm
- John sometimes mistakes phone for alarm, and Mary listens to loud music and sometimes doesn't hear alarm

# Inference by enumeration

---



- $P(B|j, m)$
- $= \alpha P(B, j, m)$
- $= \alpha \sum_e \sum_a P(B, e, a, j, m)$  marginalisation
- $= \alpha \sum_e \sum_a P(B)P(e)P(a|B, e)P(j|a)P(m|a)$  using BN
- we can improve efficiency of this by moving terms outside that don't depend on sums

$$P(b|j, m) = \alpha P(b) \sum_e P(e) \sum_a P(a|b, e) P(j|a) P(m|a)$$

$$\begin{aligned} P(b|j, m) &= \alpha P(b) (P(e)[P(a|b, e)P(j|a)P(m|a) + P(\neg a|b, e)P(j|\neg a)P(m|\neg a)]) \\ &\quad + P(\neg e)[(P(a|b, \neg e)P(j|a)P(m|a) + P(\neg a|b, \neg e)P(j|\neg a)P(m|\neg a))] \end{aligned}$$

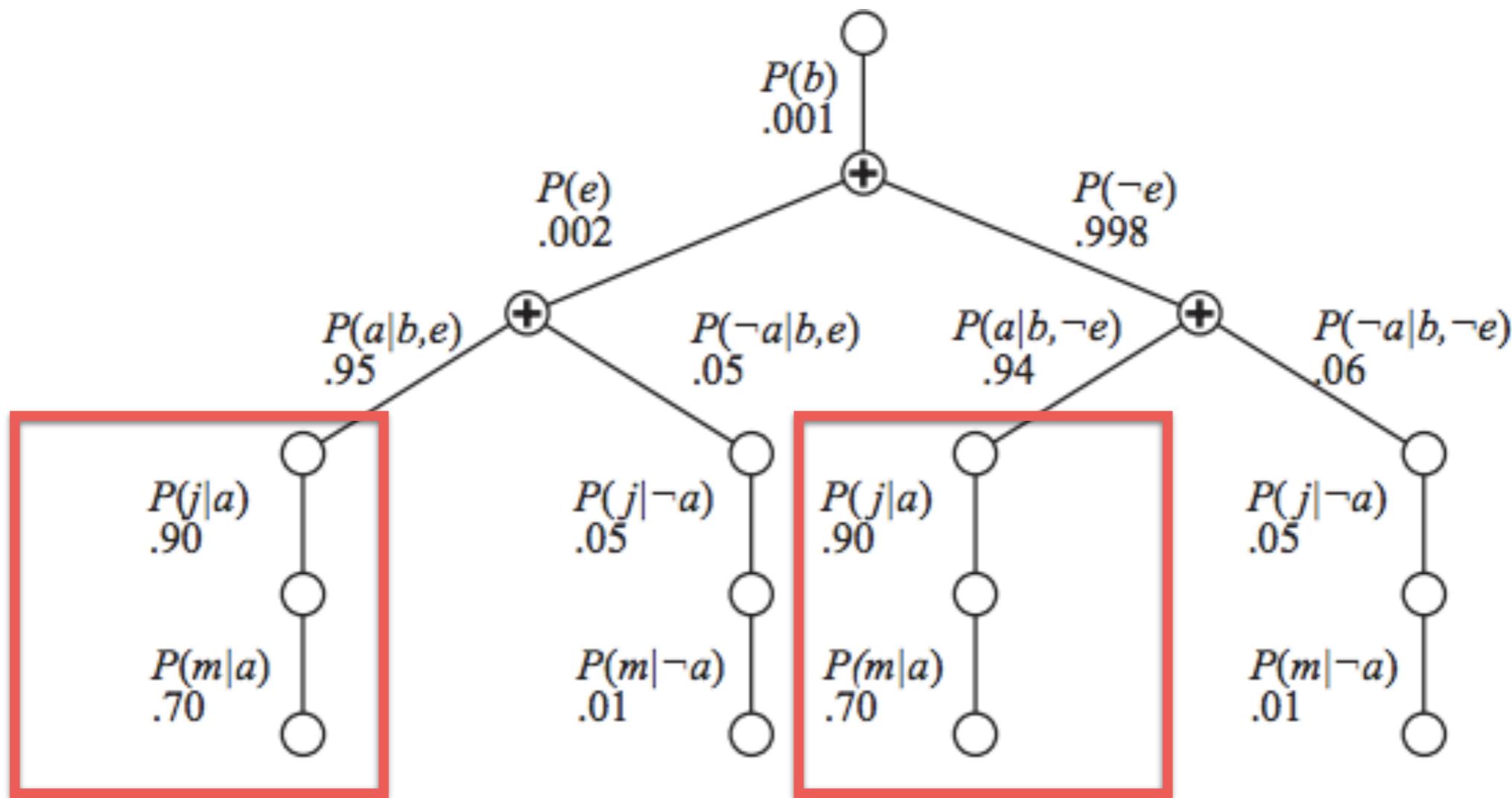
# Inference by enumeration

---

- To compute this, we need to loop through variables in order and multiply CPT entries; for each summation we need to loop over variable's possible values
- Enumeration method is computationally quite hard.
- You often compute the same thing several times;  
e.g.  $P(j|a)P(m|a)$  and  $P(j|\neg a)P(m|\neg a)$  for each value of  $e$
- Recursive depth-first enumeration:  $O(n)$  space,  $O(d^n)$  time

# Inference by enumeration

- Evaluation of expression  $P(b|j, m) = \alpha P(b)[(P(e)(P(a|b, e)P(j|a)P(m|a)+P(\neg a|b, e)P(j|\neg a)P(m|\neg a))+P(\neg e)(P(a|b, \neg e)P(j|a)P(m|a)+P(\neg a|b, \neg e)P(j|\neg a)P(m|\neg a))]$
- shown in the following tree:



# The variable elimination algorithm

---

- Idea of variable elimination: avoid repeated calculations
- Basic idea: **store results after doing calculation once**
- Works **bottom-up** by evaluating subexpressions
- Assume we want to evaluate:

$$\mathbf{P}(B|j, m) = \alpha \underbrace{\mathbf{P}(B)}_{f_1(B)} \sum_e \underbrace{\mathbf{P}(e)}_{f_2(E)} \sum_a \underbrace{\mathbf{P}(a|B, e)}_{f_3(A, B, E)} \underbrace{\mathbf{P}(j|a)}_{f_4(A)} \underbrace{\mathbf{P}(m|a)}_{f_5(A)}$$

- We've annotated each part with a **factor**.

# The variable elimination algorithm

---

$$P(B|j, m) = \alpha \underbrace{P(B)}_{f_1(B)} \sum_e \underbrace{P(e)}_{f_2(E)} \sum_a \underbrace{P(a|B, e)}_{f_3(A, B, E)} \underbrace{P(j|a)}_{f_4(A)} \underbrace{P(m|a)}_{f_5(A)}$$

- A factor is a **matrix**, indexed with its argument variables.
- E.g: Factor  $f_5(A)$  corresponds to  $P(m|a)$  and depends just on  $A$  because  $m$  is fixed – it's a  $2 \times 1$  matrix:  
 $f_5(A) = \langle P(m|a), P(m|\neg a) \rangle = \langle 0.70, 0.01 \rangle$
- $f_4(A) = \langle P(j|a), P(j|\neg a) \rangle = \langle 0.90, 0.05 \rangle$
- $f_3(A, B, E)$  is a  $2 \times 2 \times 2$  matrix for  $P(a|B, e)$   
its “first” element is  $P(a|b, e) = 0.95$  and “last”  $P(\neg a|\neg b, \neg e) = 0.999$

# The variable elimination algorithm

---

$$P(B|j, m) = \alpha \underbrace{P(B)}_{f_1(B)} \sum_e \underbrace{P(e)}_{f_2(E)} \sum_a \underbrace{P(a|B, e)}_{f_3(A, B, E)} \underbrace{P(j|a)}_{f_4(A)} \underbrace{P(m|a)}_{f_5(A)}$$

- Eliminate/Summing out A produces a  $2 \times 2$  matrix (via pointwise product):

$$\begin{aligned} f_6(B, E) &= \sum_a f_3(A, B, E) \times f_4(A) \times f_5(A) \\ &= (f_3(a, B, E) \times f_4(a) \times f_5(a)) + (f_3(\neg a, B, E) \times f_4(\neg a) \times f_5(\neg a)) \end{aligned}$$

with  $f_3(a, B, E) = (P(a|b, e), P(a|\neg b, e); P(a|b, \neg e), P(a|\neg b, \neg e))$

- So now we have  $P(B|j, m) = \alpha f_1(B) \times \sum_e f_2(E) \times f_6(B, E)$
- Eliminating E in the same way:  
 $f_7(B) = (f_2(e) \times f_6(B, e)) + (f_2(\neg e) \times f_6(B, \neg e))$
- Using  $f_1(B) = P(B)$ , we can finally compute  $P(B|j, m) = \alpha f_1(B) \times f_7(B)$
- Remains to define point-wise product and summing out

# Pointwise Products

---

- Pointwise product of 2 factors  $f_1$  and  $f_2$  yields a new factor  $f$
- whose variables are the union of the variables in  $f_1$  and  $f_2$
- and whose elements are given by the product of the corresponding elements in the 2 factors.

$$f_1(X, Y) \ f_2(Y, Z) = f(X, Y, Z)$$

A	B	$f_1(A, B)$	B	C	$f_2(B, C)$	A	B	C	$f(A, B, C)$
T	T	0.3	T	T	0.2	T	T	T	$0.3 \times 0.2$
T	F	0.7	T	F	0.8	T	T	F	$0.3 \times 0.8$
F	T	0.9	F	T	0.6	T	F	T	$0.7 \times 0.6$
F	F	0.1	F	F	0.4	T	F	F	$0.7 \times 0.4$
						F	T	T	$0.9 \times 0.2$
						F	T	F	$0.9 \times 0.8$
						F	F	T	$0.1 \times 0.6$
						F	F	F	$0.1 \times 0.4$

- For example  $f(T, T, F) = f_1(T, T) \times f_2(T, F)$

## Summing out

---

- **Summing out** is similarly straightforward
- Summing out is done by adding the sub matrices formed by fixing the variable to each of its values in turn.
- Trick: any factor that does not depend on the variable to be summed out

$$\begin{aligned} & \sum_e \mathbf{f}_2(E) \times \mathbf{f}_3(A, B, E) \times \mathbf{f}_4(A) \times \mathbf{f}_5(A) \\ &= \mathbf{f}_4(A) \times \mathbf{f}_5(A) \times \sum_e \mathbf{f}_2(E) \times \mathbf{f}_3(A, B, E) \end{aligned}$$

- Matrices are only multiplied when we need to sum out a variable from the accumulated product

Another example: :  $\mathbf{P}(J|b) = \langle P(j|b), P(\neg j|b) \rangle$

---

- $P(J|b) = \alpha \sum_e \sum_a \sum_m P(J, b, e, a, m)$

Another example: :  $\mathbf{P}(J|b) = \langle P(j|b), P(\neg j|b) \rangle$

---

- $P(J|b) = \alpha \sum_e \sum_a \sum_m \mathbf{P}(J, b, e, a, m)$   
 $= \alpha \sum_e \sum_a \sum_m P(b)P(e)P(a|b, e)\mathbf{P}(J|a)P(m|a)$

Another example: :  $\mathbf{P}(J|b) = \langle P(j|b), P(\neg j|b) \rangle$

---

- $P(J|b)$    =    $\alpha \sum_e \sum_a \sum_m \mathbf{P}(J, b, e, a, m)$   
              =    $\alpha \sum_e \sum_a \sum_m P(b)P(e)P(a|b, e)\mathbf{P}(J|a)P(m|a)$   
              =    $\alpha' \sum_e \underbrace{P(e)}_{\mathbf{f}_1(E)} \sum_a \underbrace{P(a|b, e)}_{\mathbf{f}_2(A, E)} \underbrace{\mathbf{P}(J|a)}_{\mathbf{f}_3(J, A)} \underbrace{\sum_m P(m|a)}_m$

Another example: :  $\mathbf{P}(J|b) = \langle P(j|b), P(\neg j|b) \rangle$

---

$$\begin{aligned} \bullet \quad \mathbf{P}(J|b) &= \alpha \sum_e \sum_a \sum_m \mathbf{P}(J, b, e, a, m) \\ &= \alpha \sum_e \sum_a \sum_m P(b) P(e) P(a|b, e) \mathbf{P}(J|a) P(m|a) \\ &= \alpha' \sum_e \underbrace{P(e)}_{\mathbf{f}_1(E)} \sum_a \underbrace{P(a|b, e)}_{\mathbf{f}_2(A, E)} \underbrace{\mathbf{P}(J|a)}_{\mathbf{f}_3(J, A)} \underbrace{\sum_m P(m|a)}_{= 1} \end{aligned}$$

- M was irrelevant to this query  
The result of this query is unchanged if we remove *MaryCalls* altogether
- In general, every variable that is not an accents of a query variable or evidence variable is irrelevant to the query

# Summary

---

- Inference in Bayesian Networks
- Exact methods: enumeration, variable elimination algorithm
- In general, the complexity of exact inference will depend strongly **on the structure of the network** :
- Computationally intractable in the worst case
- Linear time in the best cases (tree structures) (as a function of number of CPT entries)
- Next time: Approximate inference in Bayesian Networks



# **Inf2D-Reasoning and Agents**

## Spring 2017

Lecture (2)3:- Probabilistic Reasoning with Bayesian Networks

Peggy Sériès, [pseries@inf.ed.ac.uk](mailto:pseries@inf.ed.ac.uk)

Based on previous slides by A. Lascarides

# Where are we?

---

- Using JPD tables for probabilistic inference
- Concepts of absolute and conditional independence
- Bayes' rule
- Today . . .

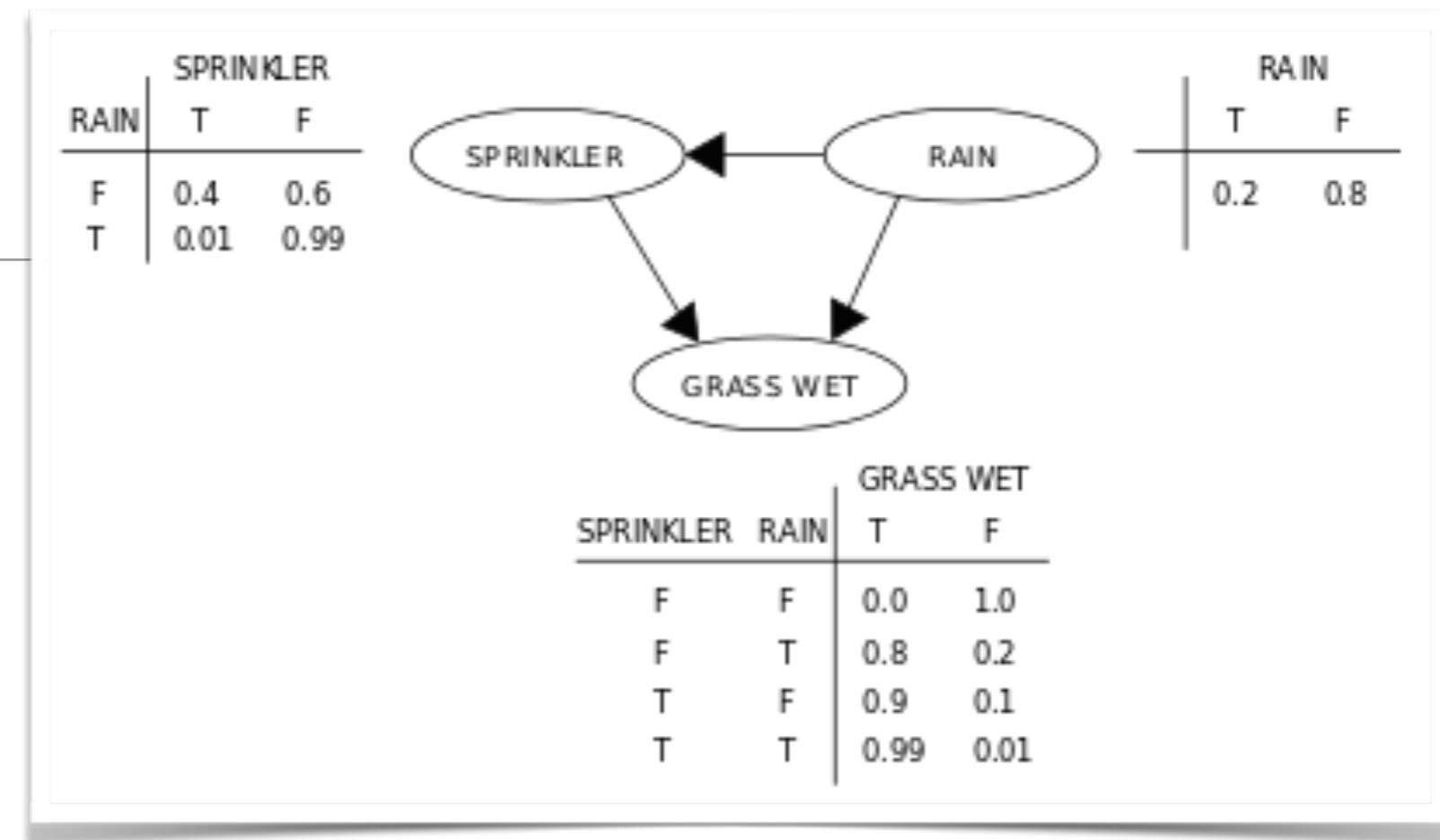
Probabilistic Reasoning with Bayesian Networks

# Representing Knowledge in an uncertain domain

---

- **Full joint probability distributions** can become intractably large very quickly
- **Conditional independence** helps to reduce the number probabilities required to specify the JPD
- Now we will introduce **Bayesian networks (BNs)** to systematically describe dependencies between random variables
- Roughly speaking, BNs are **graphs** that connect nodes representing variables with each other whenever they depend on each other

# Bayesian Networks

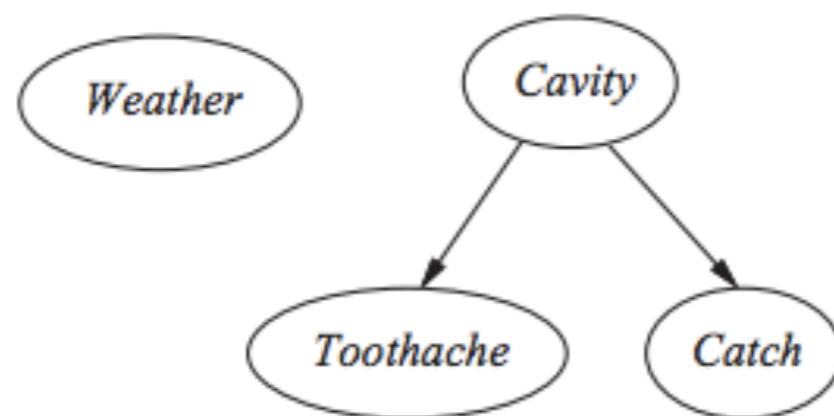


- A BN is a **directed acyclic graph** (DAG) with nodes annotated with probability information
- The **nodes** represent random variables (discrete/continuous)
- **Links** connect nodes. If there's an arrow from X to Y, we call X a **parent** of Y
- Each node  $X_i$  has a **conditional probability distribution** (CPD) attached to it
- The CPD describes how  $X_i$  depends on its parents, i.e. its entries describe  $P(X_i|Parents(X_i))$ ; rows in CPD sum to 1.

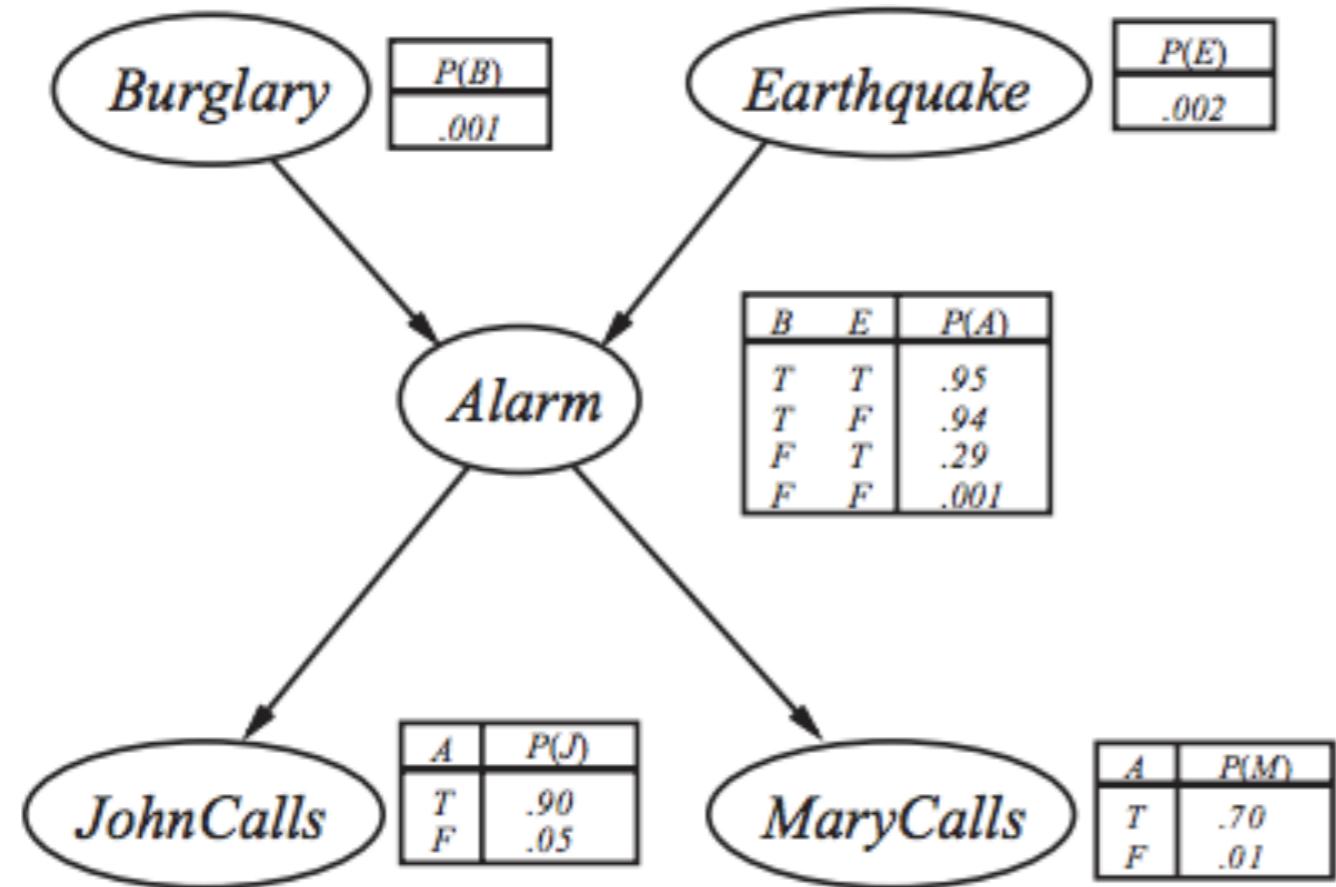
# Bayesian Networks

---

- Topology of graphs describes conditional independence relationships
- Intuitively, links describe direct effects of variables on each other in the domain
- Assumption: anything that is not directly connected does not directly depend on each other
- In previous dentist/weather example:



## Example

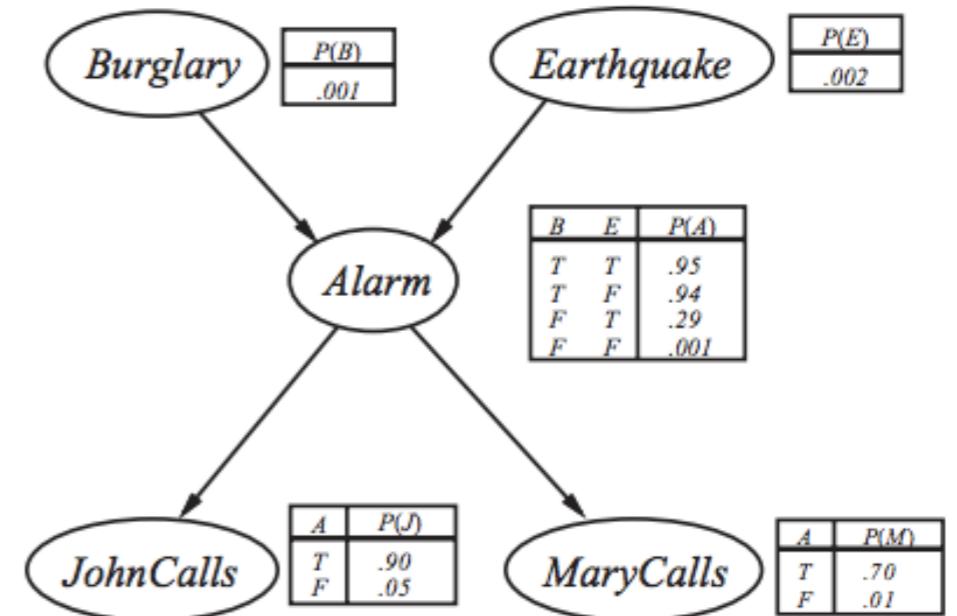


- New **burglar alarm** has been fitted, fairly reliable but sometimes reacts to earthquakes
- Neighbours John and Mary promise to call when they hear alarm
- John sometimes mistakes phone for alarm, and Mary listens to loud music and sometimes doesn't hear alarm

# Example

---

- No perception of earthquake by John or Mary
- No explicit modelling of phone ring confusing John, or of Mary's loud music (summarised in uncertainty regarding their reaction)
- Actually this uncertainty summarises any kind of failure
- Each row in CPTs contains **a conditioning case** (configuration of parent values)
- For  $k$  boolean parents,  $2^k$  possible cases
- We often omit  $P(\neg x_i || \text{Parents}(X_i))$  from CPT for node  $X_i$  (computes as  $1 - P(x_i | \text{Parents}(X_i))$ )

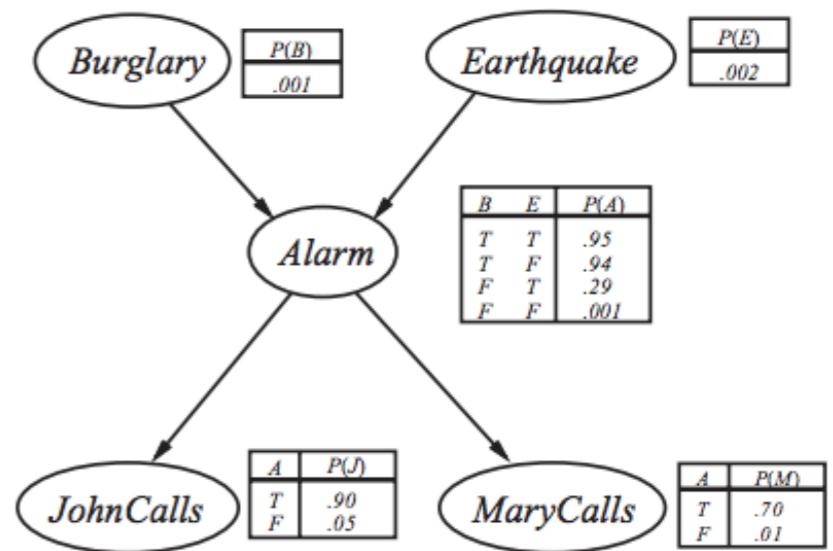


# The Semantics of Bayesian Networks

- Two views:
  - 1) BN as **representation of JPD** (useful for constructing BNs)
  - 2) BN as collection of **conditional independence statements** (useful for designing inference procedures)
- Every entry  $P(X_1 = x_1 \wedge \dots \wedge X_n = x_n)$  in the JPD can be calculated from a BN (abbreviate by  $P(x_1, \dots, x_n)$ )

$$\cdot P(x_1, \dots, x_n) = \prod_{i=1}^n P(x_i | \text{parents}(X_i))$$

- Example:
$$\begin{aligned} & P(j \wedge m \wedge a \wedge \neg b \wedge \neg e) \\ & = P(j|a)P(m|a)P(a|\neg b \wedge \neg e)P(\neg b)P(\neg e) \\ & = 0.9 \times 0.7 \times 0.001 \times 0.999 \times 0.998 = 0.00062 \end{aligned}$$
- As before, this can be used to answer any query



# A Method for Constructing BNs

---

- Recall **product rule** for  $n$  variables:

$$P(x_1, \dots, x_n) = P(x_n|x_{n-1}, \dots, x_1)P(x_{n-1}, \dots, x_1)$$

- Repeated application of this yields the so-called **chain rule**:

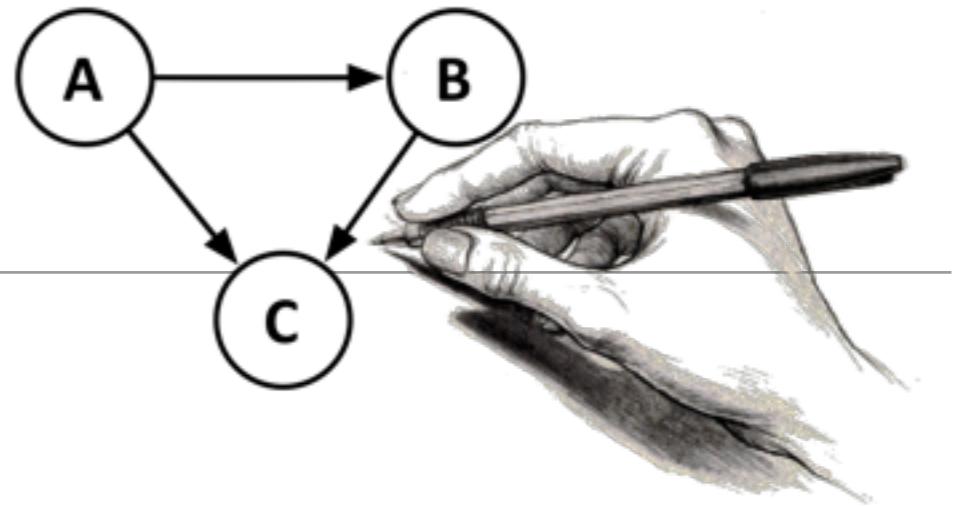
$$\begin{aligned} P(x_1, \dots, x_n) &= P(x_n|x_{n-1}, \dots, x_1)P(x_{n-1}|x_{n-2}, \dots, x_1) \cdots P(x_2|x_1)P(x_1) \\ &= \prod_{i=1}^n P(x_i|x_{i-1}, \dots, x_1) \end{aligned}$$

- We also have  $P(X_i|X_{i-1}, \dots, X_1) = P(X_i|\text{Parents}(X_i))$  as long as  $\text{Parents}(X_i) \subseteq \{X_{i-1}, \dots, X_1\}$  (ensured by labelling nodes appropriately)
- For example, it is reasonable to assume that  
 $P(\text{MaryCalls}|\text{JohnCalls}, \text{Alarm}, \text{Earthquake}, \text{Burglary}) = P(\text{MaryCalls}|\text{Alarm})$

$$P(x_1, \dots, x_n) = \prod_{i=1}^n P(x_i|\text{parents}(X_i))$$

# A Method for Constructing BNs

---



## Nodes

1. First determine the set of **variables** that are required to model the domain.
2. **Order them** in such a way that causes precede effects.

## Links

1. Choose a minimal set of **parents** for each variable  $X_i$ , that directly influence  $X_i$ .
2. For each parent, **insert a link** from the parent to  $X_i$
3. Write down the **conditional probability** table  $P(X_i|Parents(X_i))$

# Compactness and Node Ordering

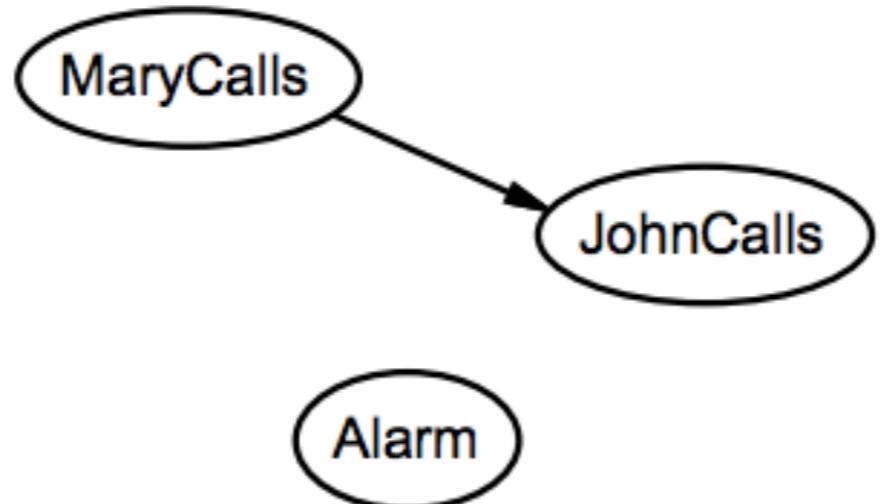
---

- BNs examples of **locally structured (sparse)** systems:  
subcomponents only interact with small number of other components
- E.g. if 30 boolean nodes and every node depends on 5 nodes,  
BN will have  $30 \times 2^5 = 960$  probabilities stored in the CPDs,  
while JPD would have  $2^{30} \approx 10003$  entries
- But remember that this is based on designer's **independence assumptions!**
- Also not trivial to determine **good BN structure**:

*Add “root causes” first, then variables they influence, and so on, until we reach “leaves” which have no influence on other variables*

## Example

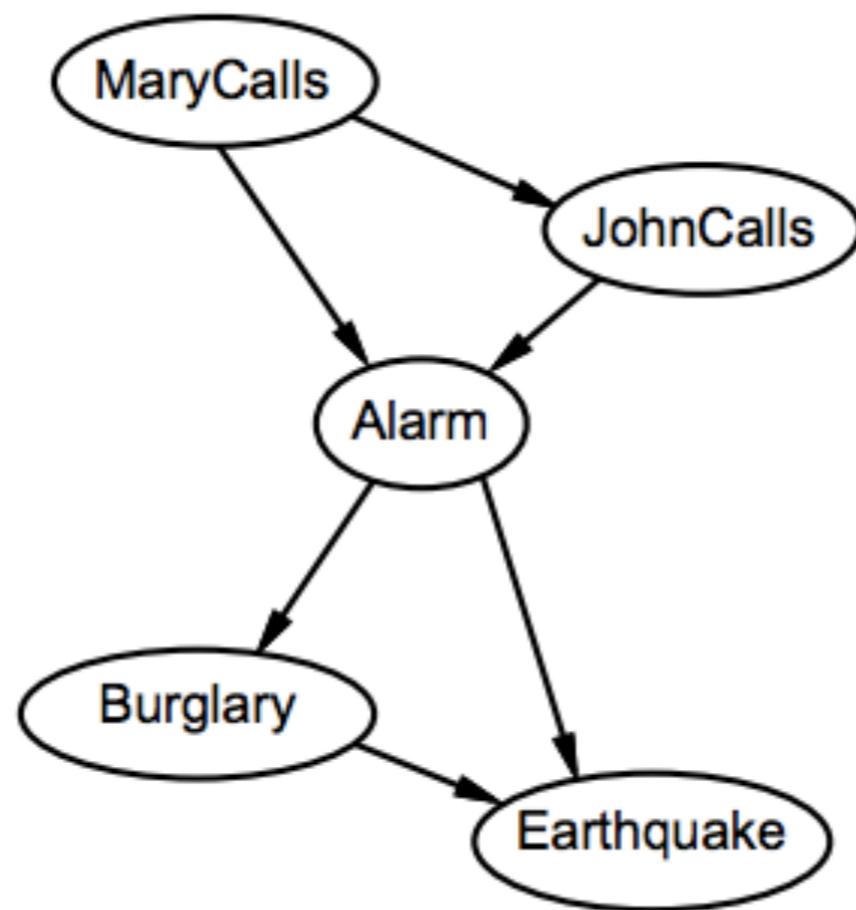
Suppose we choose the ordering  $M, J, A, B, E$



$$P(J|M) = P(J)? \text{ No}$$

$$P(A|J, M) = P(A|J)? \quad P(A|J, M) = P(A)?$$

## Example contd.



Deciding conditional independence is hard in noncausal directions

(Causal models and conditional independence seem hardwired for humans!)

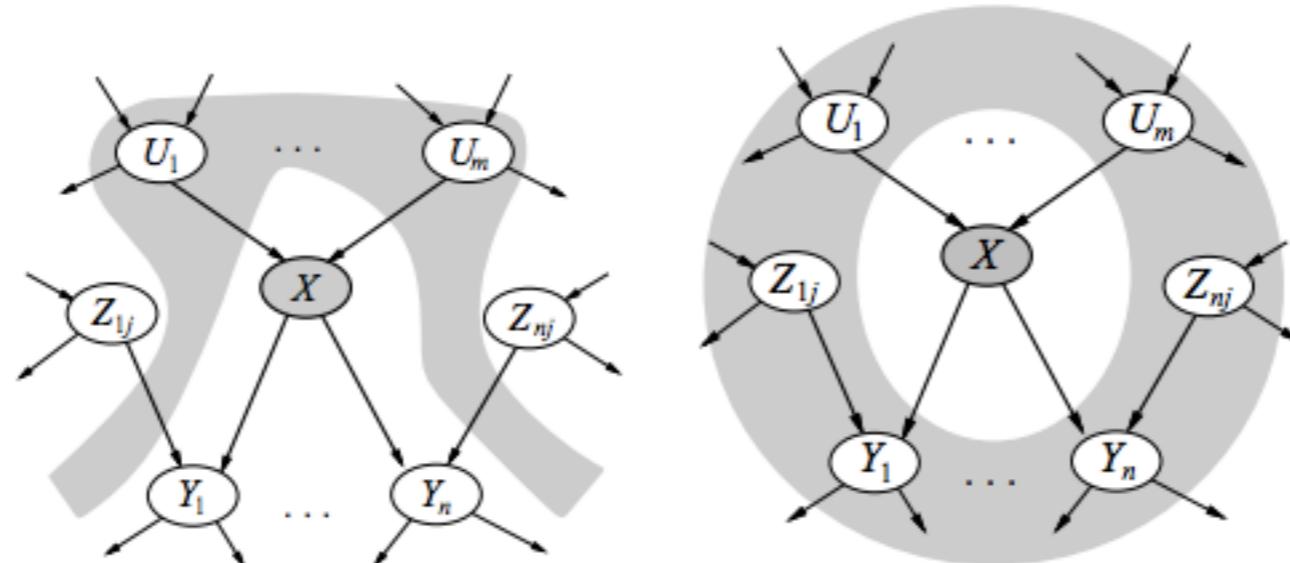
Assessing conditional probabilities is hard in noncausal directions

Network is less compact:  $1 + 2 + 4 + 2 + 4 = 13$  numbers needed

# Conditional independence relations in BNs

---

- 1. A node is conditionally independent of its non-descendants, given its parents
- 2. A node is conditionally independent of all other nodes, given its parents, children and children's parents, i.e. its **Markov blanket**.  
This means that the Markov blanket of a node is the only knowledge needed to predict the behavior of that node.



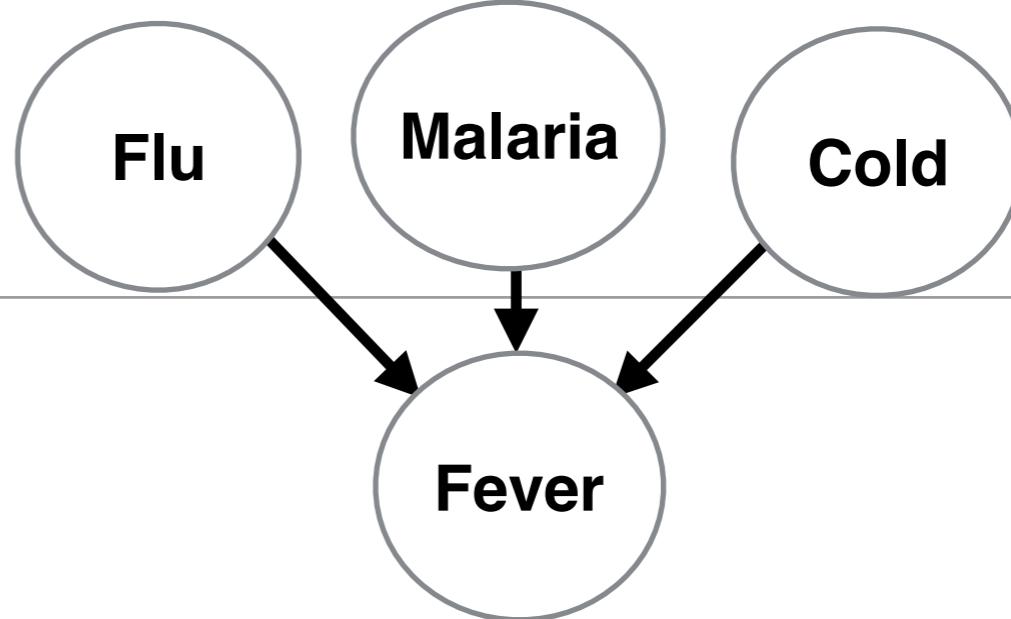
# Efficient representation of conditional distributions

---

- Even the  $2^k$  ( $k$  parents) conditioning cases that have to be provided require **a great deal of experience and knowledge** of the domain
- Can we **avoid having to specify every entry** in the joint conditional pdf?
- By specifying pattern by a few parameters we can save a lot of space!
- Simplest case: **deterministic node** that can be directly inferred from values of parents
- For example, logical or mathematical functions

# Noisy-OR relationships

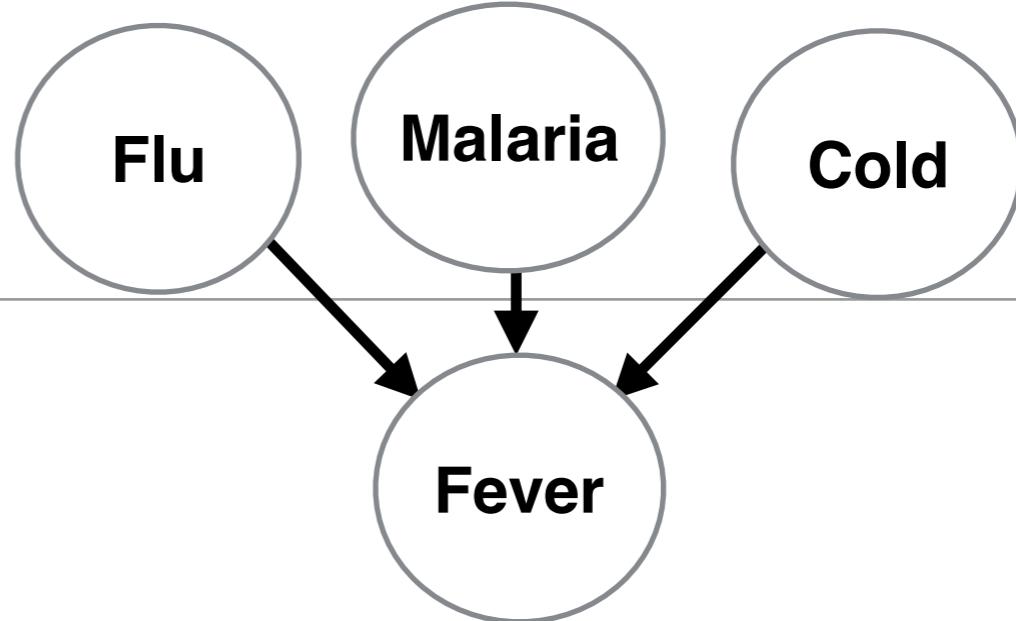
---



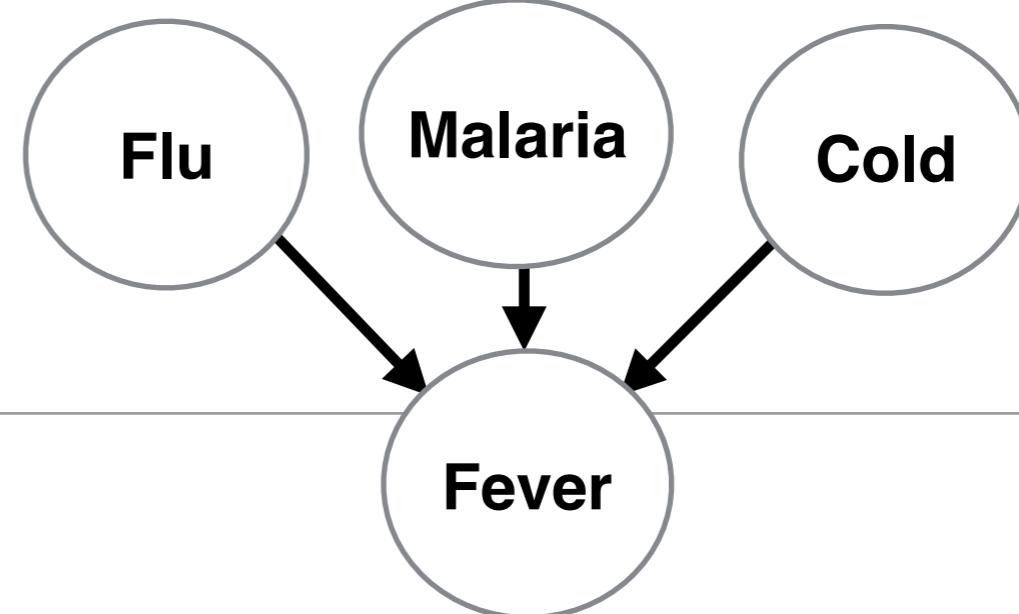
- Any cause can make effect true, but won't necessarily — effect is then said to be **inhibited**, e.g. Flu has a chance of not causing Fever.
- Assumes **all possible causes are listed**.
- Assumes **inhibitions are mutually conditionally independent**: Whatever inhibits Flu from causing Fever is independent of what inhibits Malaria to cause Fever.
- Fever is false only if each of its parents are inhibited. We can compute this likelihood from product of probabilities for each individual cause inhibiting Fever.
- How does this help?

# Noisy-OR relationships

---



- Fever is caused by Cold, Flu or Malaria and that's all.
- Inhibitions of Cold, Flu and Malaria are mutually conditionally independent
- Likelihood that Cold is inhibited from causing Fever is  $P(\neg \text{fever} | \text{cold}, \neg \text{flu}, \neg \text{malaria})$  (similarly for other causes)
- Individual inhibition probabilities:  
 $P(\neg \text{fever} | \text{cold}, \neg \text{flu}, \neg \text{malaria}) = 0.6$   
 $P(\neg \text{fever} | \neg \text{cold}, \text{flu}, \neg \text{malaria}) = 0.2$   
 $P(\neg \text{fever} | \neg \text{cold}, \neg \text{flu}, \text{malaria}) = 0.1$
- **Inhibitions mutually independent**, so:  
 $P(\neg \text{fever} | \text{cold}, \text{flu}, \neg \text{malaria}) =$   
 $P(\neg \text{fever} | \text{cold}, \neg \text{flu}, \neg \text{malaria})P(\neg \text{fever} | \neg \text{cold}, \text{flu}, \neg \text{malaria}) = 0.6 \times 0.2$

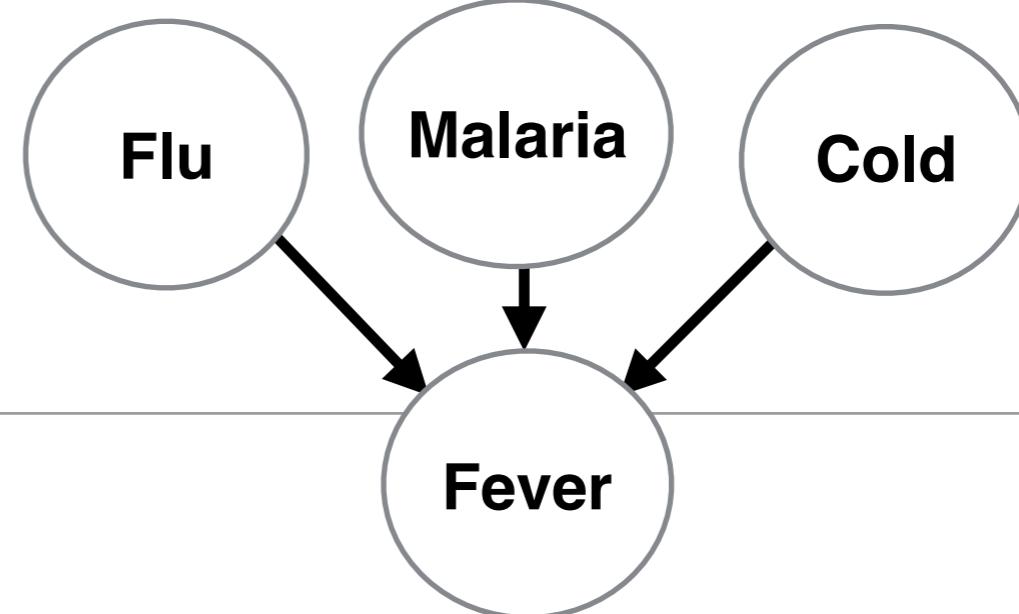


## Noisy-OR relationships

---

- We can construct entire CPT from this information (3 numbers)

<i>Cold</i>	<i>Flue</i>	<i>Malaria</i>	$P(Fever)$	$P(\neg Fever)$
F	F	F		
F	F	T		<b>0.1</b>
F	T	F		<b>0.2</b>
F	T	T		
T	F	F		<b>0.6</b>
T	F	T		
T	T	F		
T	T	T		

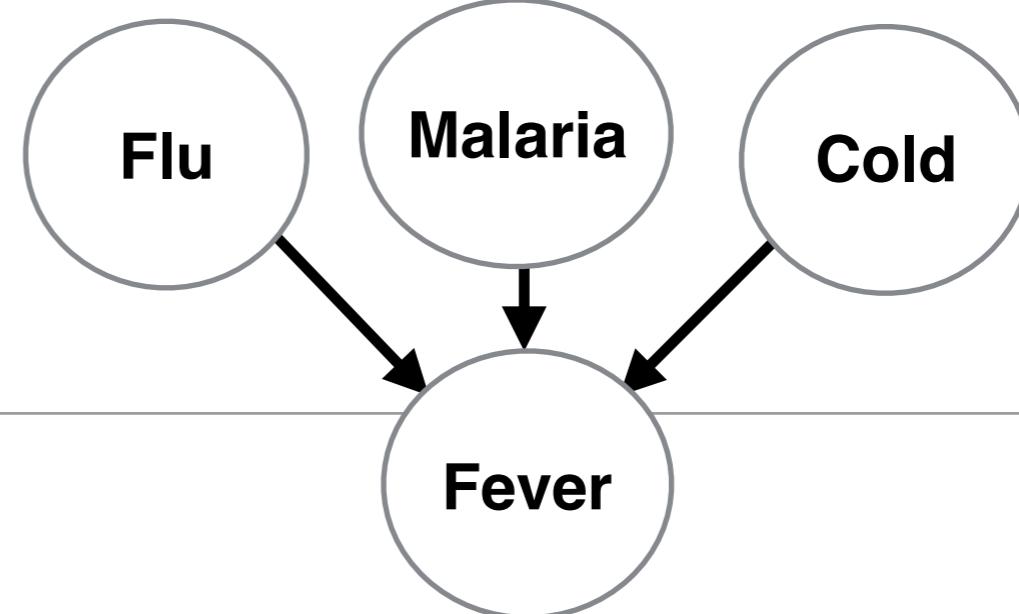


## Noisy-OR relationships

---

- We can construct entire CPT from this information (3 numbers)

<i>Cold</i>	<i>Flue</i>	<i>Malaria</i>	$P(Fever)$	$P(\neg Fever)$
F	F	F		
F	F	T	0.9	<b>0.1</b>
F	T	F	0.8	<b>0.2</b>
F	T	T		
T	F	F	0.4	<b>0.6</b>
T	F	T		
T	T	F		
T	T	T		

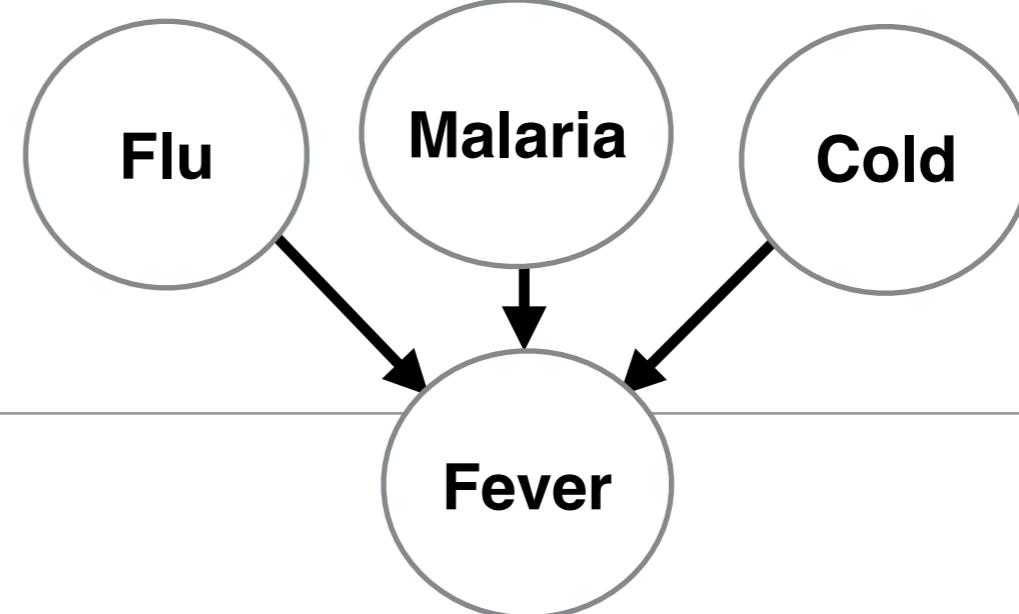


## Noisy-OR relationships

---

- We can construct entire CPT from this information (3 numbers)

<i>Cold</i>	<i>Flue</i>	<i>Malaria</i>	$P(Fever)$	$P(\neg Fever)$
F	F	F		
F	F	T	0.9	<b>0.1</b>
F	T	F	0.8	<b>0.2</b>
F	T	T		$0.02 = 0.2 \times 0.1$
T	F	F	0.4	<b>0.6</b>
T	F	T		
T	T	F		
T	T	T		

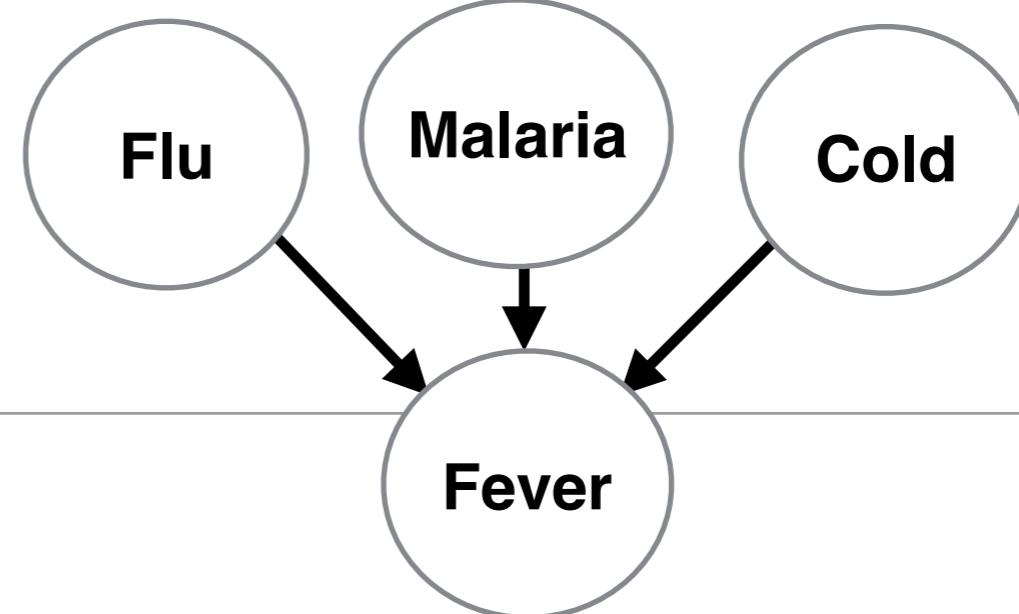


## Noisy-OR relationships

---

- We can construct entire CPT from this information (3 numbers)

<i>Cold</i>	<i>Flue</i>	<i>Malaria</i>	$P(Fever)$	$P(\neg Fever)$
F	F	F		
F	F	T	0.9	<b>0.1</b>
F	T	F	0.8	<b>0.2</b>
F	T	T		$0.02 = 0.2 \times 0.1$
T	F	F	0.4	<b>0.6</b>
T	F	T		$0.06 = 0.6 \times 0.1$
T	T	F		$0.12 = 0.6 \times 0.2$
T	T	T		

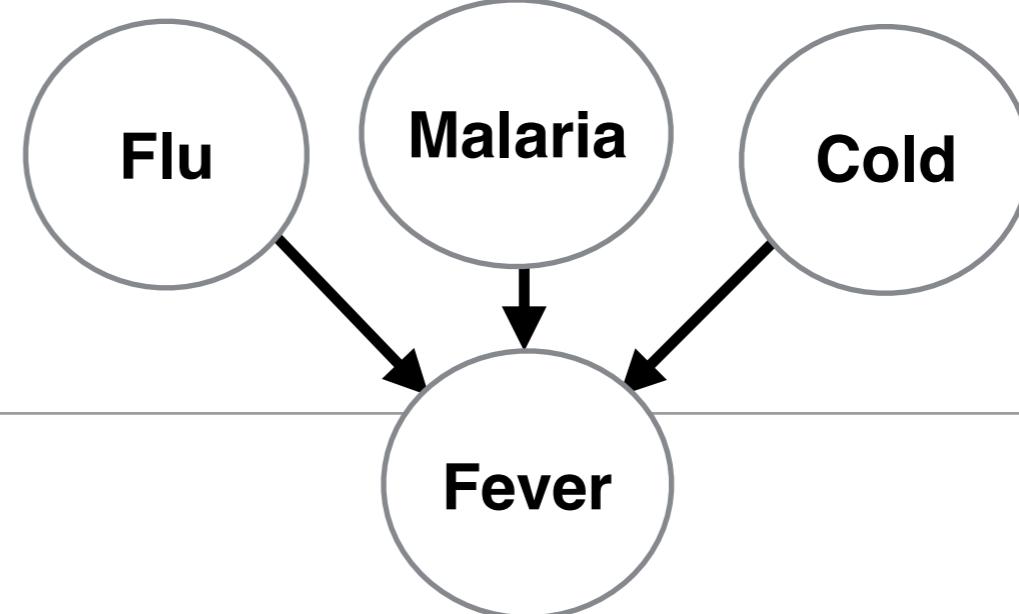


## Noisy-OR relationships

---

- We can construct entire CPT from this information (3 numbers)

<i>Cold</i>	<i>Flue</i>	<i>Malaria</i>	$P(Fever)$	$P(\neg Fever)$
F	F	F		
F	F	T	0.9	<b>0.1</b>
F	T	F	0.8	<b>0.2</b>
F	T	T		$0.02 = 0.2 \times 0.1$
T	F	F	0.4	<b>0.6</b>
T	F	T		$0.06 = 0.6 \times 0.1$
T	T	F		$0.12 = 0.6 \times 0.2$
T	T	T		$0.012 = 0.6 \times 0.2 \times 0.1$

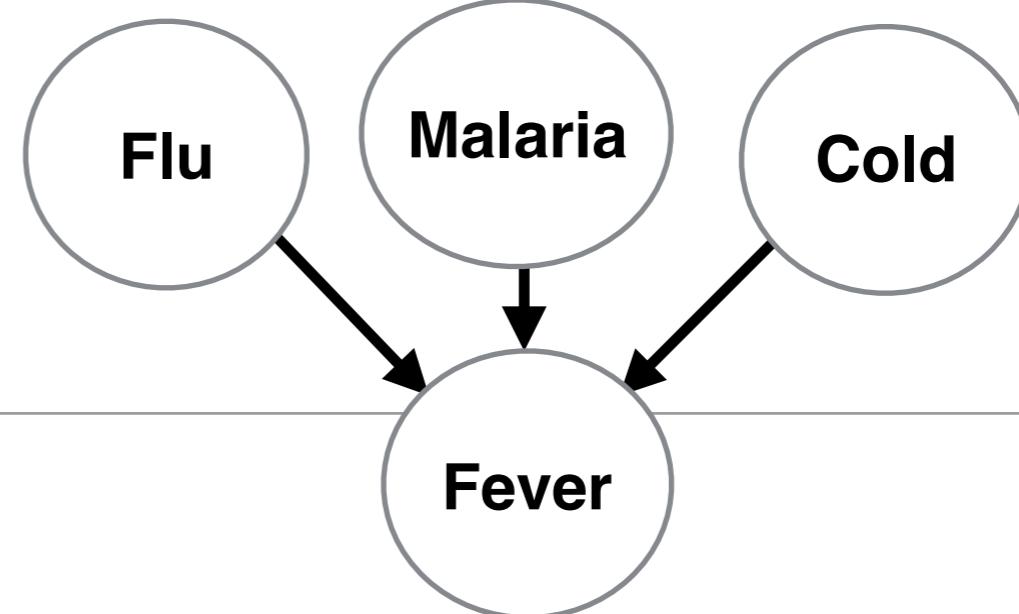


## Noisy-OR relationships

---

- We can construct entire CPT from this information (3 numbers)

<i>Cold</i>	<i>Flue</i>	<i>Malaria</i>	$P(Fever)$	$P(\neg Fever)$
F	F	F		
F	F	T	0.9	<b>0.1</b>
F	T	F	0.8	<b>0.2</b>
F	T	T	0.98	$0.02 = 0.2 \times 0.1$
T	F	F	0.4	<b>0.6</b>
T	F	T	0.94	$0.06 = 0.6 \times 0.1$
T	T	F	0.88	$0.12 = 0.6 \times 0.2$
T	T	T	0.988	$0.012 = 0.6 \times 0.2 \times 0.1$



## Noisy-OR relationships

---

- We can construct entire CPT from this information (3 numbers)

<i>Cold</i>	<i>Flue</i>	<i>Malaria</i>	$P(Fever)$	$P(\neg Fever)$
F	F	F	0.0	1.0
F	F	T	0.9	<b>0.1</b>
F	T	F	0.8	<b>0.2</b>
F	T	T	0.98	$0.02 = 0.2 \times 0.1$
T	F	F	0.4	<b>0.6</b>
T	F	T	0.94	$0.06 = 0.6 \times 0.1$
T	T	F	0.88	$0.12 = 0.6 \times 0.2$
T	T	T	0.988	$0.012 = 0.6 \times 0.2 \times 0.1$

- Encodes CPT with  $k$  instead of  $2^k$  values

# BN with continuous variables

---

- Often variables range over **continuous domains**
- **Discretisation** one possible solution but often leads to inaccuracy or requires a lot of discrete values
- Other solution: use of **standard families of probability distributions** specified in terms of a few parameters
- Example: normal/Gaussian distribution  $N(\mu, \sigma^2)(x)$  defined in terms of mean  $\mu$  and variance  $\sigma^2$  (needs just two parameters)
- **Hybrid Bayesian Networks** use mixture of discrete and continuous variables (special methods to deal with links between different types – not discussed here)

# Summary

---

- Introduced Bayesian Networks as a structured way of reasoning under uncertainty using probabilities and independence
- Defined their semantics in terms of JPD representation, and conditional independence statements
- Talked about issues of efficient representation of CPTs
- Noisy-OR model
- Discussed continuous variables and hybrid networks
- Next time: Exact Inference in Bayesian Networks



# **Inf2D-Reasoning and Agents**

## Spring 2017

Lecture (2)5:- Approximate Inference with Bayesian Networks

Peggy Seriès, [pseries@inf.ed.ac.uk](mailto:pseries@inf.ed.ac.uk)

Based on previous slides by A. Lascarides

# Where are we?

---

- Last time . . .
- Inference in Bayesian Networks  $P(X|e)$
- Exact methods: enumeration, variable elimination algorithm involved evaluation large sums of products.
- Computationally intractable in the worst case
- Today . . . Approximate Inference in Bayesian Networks

# Approximate Inference in BNs

---

- Exact inference computationally very hard
- Approximate methods important, here **randomised sampling** algorithms
- **Monte Carlo** algorithms
- We will talk about two types of MC algorithms:
  1. Direct sampling methods
  2. Markov chain sampling

# Direct sampling methods

---

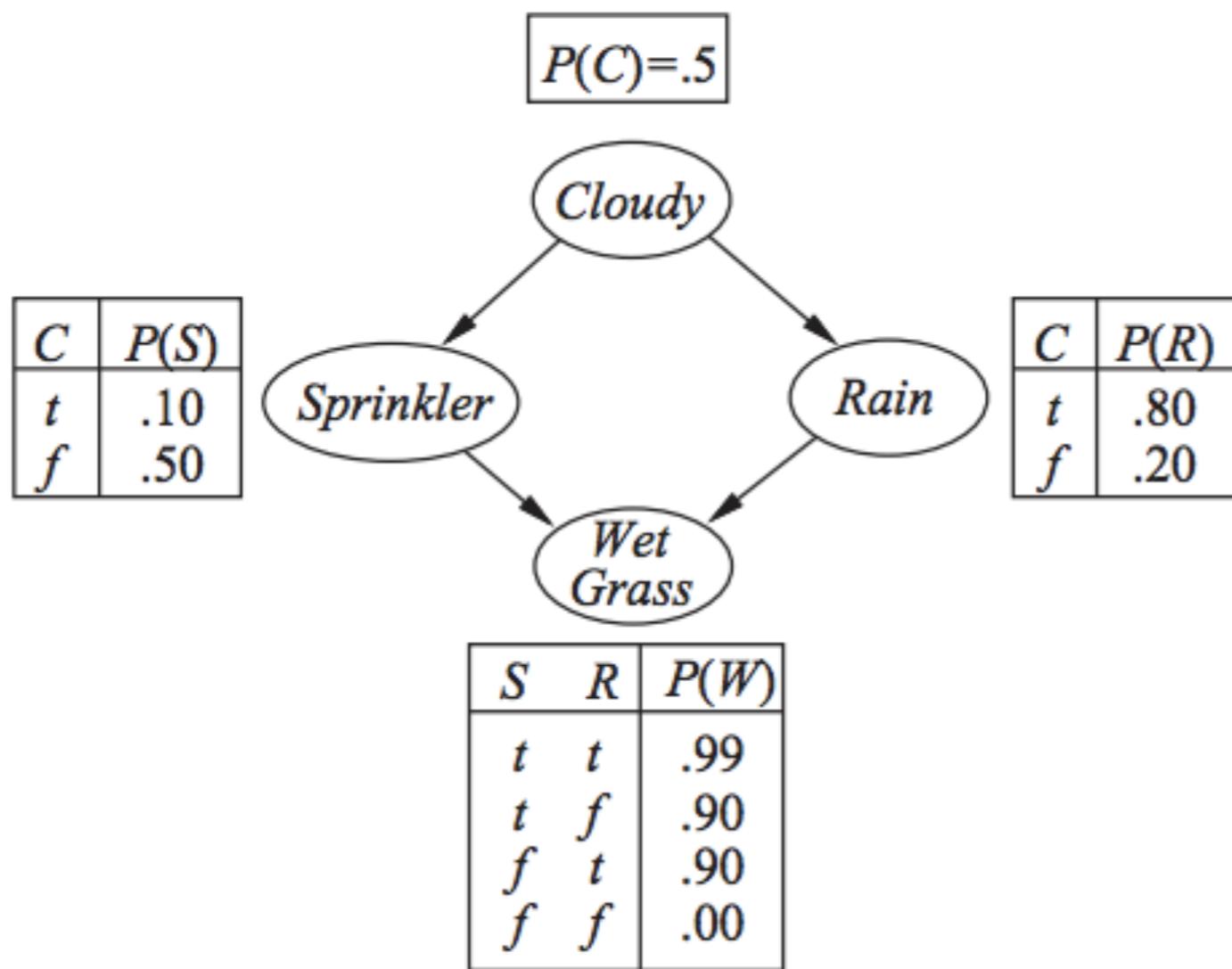


- Basic idea: generate samples from a known probability distribution
- Consider an unbiased coin as a random variable – sampling from the distribution is like flipping the coin
- It is possible to sample any distribution on a single variable given a set of random numbers from  $[0,1]$  - **biased coin**
- Simplest method: generate events from network **without evidence**
  - **Sample each variable in ‘topological order’**
  - Probability distribution for sampled value is conditioned on values assigned to parents

# Example

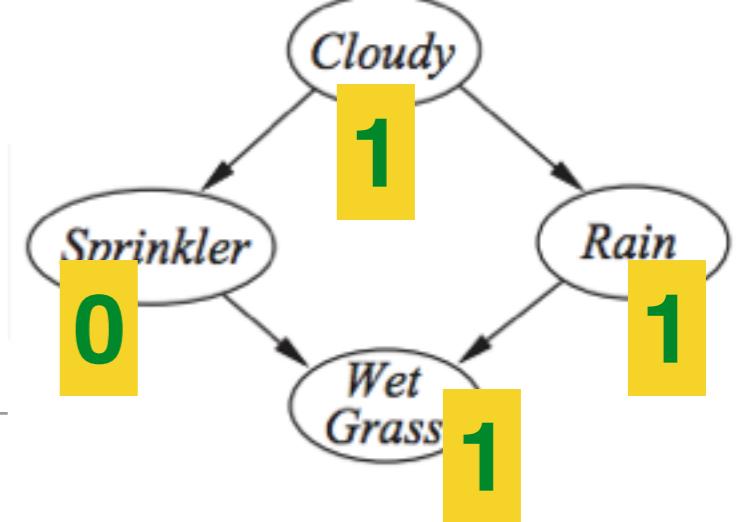


- Consider the following BN and ordering [Cloudy, Sprinkler, Rain, WetGrass]



## Example

---



- Direct sampling process:
- Sample from  $\mathbf{P}(\text{Cloudy}) = \langle 0.5, 0.5 \rangle$ , suppose this returns true
- Sample from  $\mathbf{P}(\text{Sprinkler}|\text{Cloudy} = \text{true}) = \langle 0.1, 0.9 \rangle$ , suppose this returns false.
- Sample from  $\mathbf{P}(\text{Rain}|\text{Cloudy} = \text{true}) = \langle 0.8, 0.2 \rangle$ , suppose this returns true.
- Sample from  $\mathbf{P}(\text{WetGrass}|\text{Sprinkler} = \text{false}, \text{Rain} = \text{true}) = \langle 0.9, 0.1 \rangle$ , suppose this returns true.
- Event returned=[true, false, true, true].  
This is a sample from the joint.
- Repeat.

# Direct Sampling Methods

---

- Generates samples with probability  $S(x_1, \dots, x_n)$

$$S(x_1, \dots, x_n) = P(x_1, \dots, x_n) = \prod_{i=1}^n P(x_i | parents(X_i))$$

- Answers are computed by **counting** the number  $N(x_1, \dots, x_n)$  of the times event  $x_1, \dots, x_n$  was generated and dividing by total number  $N$  of all samples
- In the **limit**, we should get

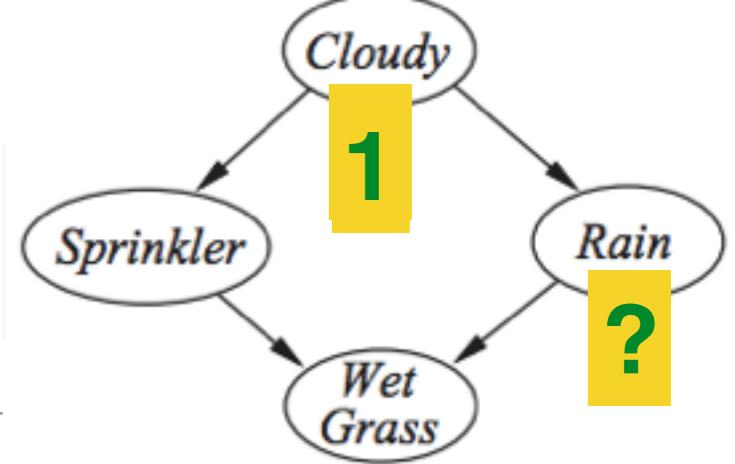
$$\lim_{n \rightarrow \infty} \frac{N(x_1, \dots, x_n)}{N} = S(x_1, \dots, x_n) = P(x_1, \dots, x_n)$$

- When the estimated probability becomes exact in the limit we call the estimate **consistent** and we write “ $\approx$ ” in this sense,

$$P(x_1, \dots, x_n) \approx N(x_1, \dots, x_n)/N$$

# Example $P_{\text{est}}(\text{Rain}=\text{true})?$

---



- Repeat 100 times :
  - Sample from  $P(\text{Cloudy}) = \langle 0.5, 0.5 \rangle$ .
  - Given the result, Sample from  $P(\text{Rain}|\text{Cloudy})$
- Count the number of samples where Rain =1 and divide by 100 samples
- $P_{\text{est}}(\text{Rain}=\text{true}) \sim 0.511$

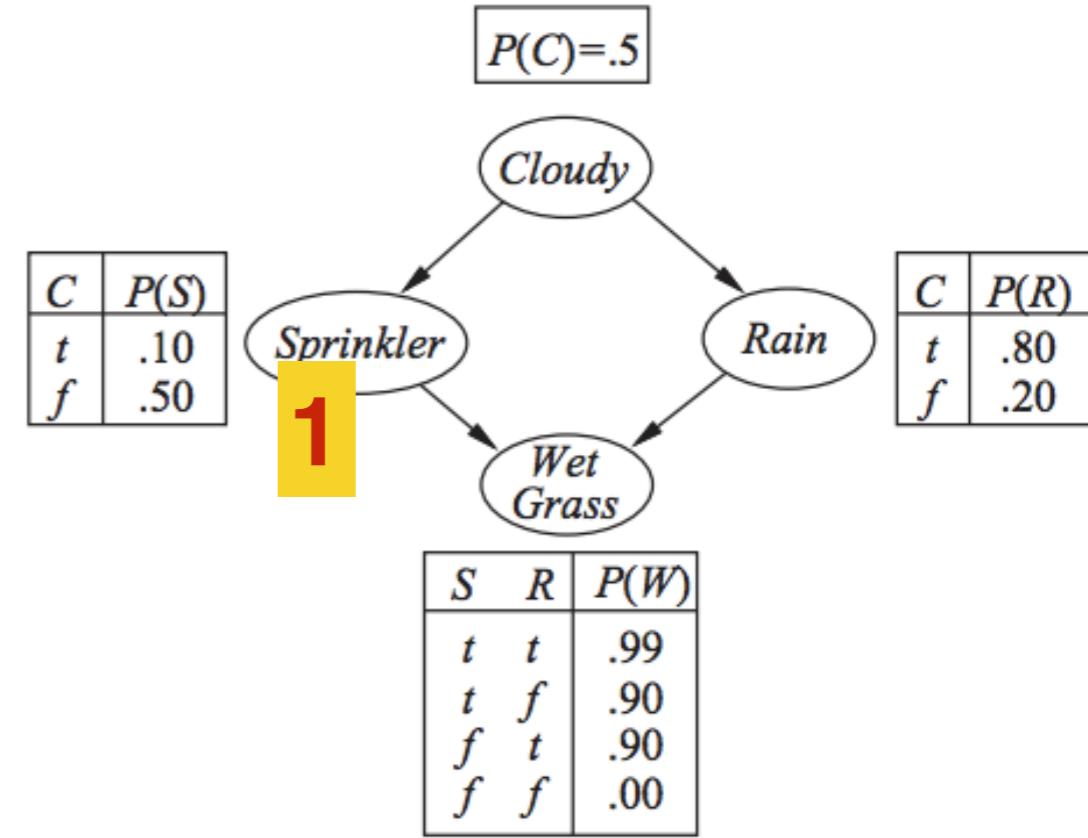
# Rejection Sampling

---

- What if you wanted to compute  $P(X|e)$  — with some **evidence**?
- Rejection Sampling: A general method to produce samples for hard-to-sample distribution from easy-to-sample distribution
- To **determine  $P(X|e)$**  generate samples from the prior distribution specified by the BN first
- Then **reject those that do not match the evidence**
- The estimate  $P(X = x|e)$  is obtained by **counting** how often  $X = x$  occurs **in the remaining samples**
- Rejection sampling is consistent because, by definition:

$$\hat{P}(X|e) = \frac{\mathbf{N}(X, e)}{N(e)} \approx \frac{\mathbf{P}(X, e)}{P(e)} = \mathbf{P}(X|e)$$

## Back to our example



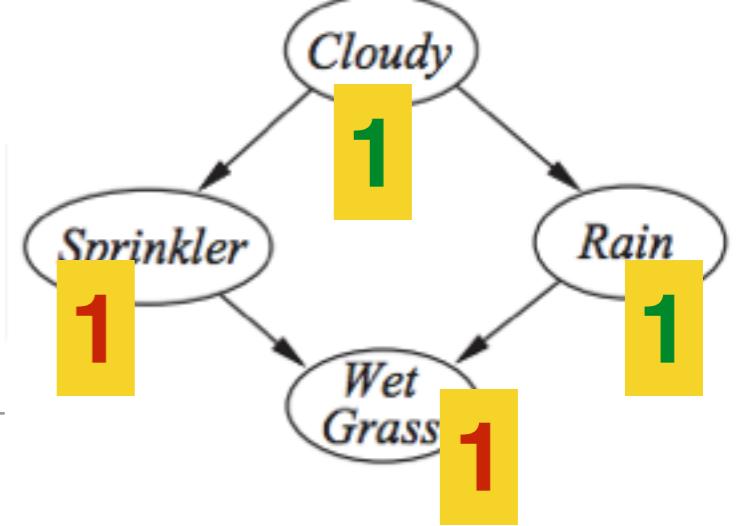
- Assume we want to estimate  $P(\text{Rain}|\text{Sprinkler} = \text{true})$ , using 100 samples
- 73 have  $\text{Sprinkler} = \text{false}$  (rejected), 27 have  $\text{Sprinkler} = \text{true}$
- Of these 27, 8 have  $\text{Rain} = \text{true}$  and 19 have  $\text{Rain} = \text{false}$
- $P(\text{Rain}|\text{Sprinkler} = \text{true}) \approx \alpha <8, 19> = <0.296, 0.704>$
- True answer would be  $<0.3, 0.7>$
- But the procedure **rejects too many samples** that are not consistent with e (exponential in number of variables)
- Not really usable (similar to naively estimating conditional probabilities from observation)

# Likelihood weighting

---

- Avoids inefficiency of rejection sampling by **generating only samples consistent with evidence**
- Fixes the values for evidence variables **E** and samples only the remaining variables **X** and **Y**
- Since not all events are equally probable, each event has to be **weighted by its likelihood** that it accords to the evidence
- Likelihood is measured by product of conditional probabilities for each evidence variable, given its parents.

# Likelihood weighting



- Consider query  $\mathbf{P}(\text{Rain}|\text{Sprinkler} = \text{true}, \text{WetGrass} = \text{true})$  in our example;
- Initially set weight  $w = 1$ , then event is generated:
- Sample from  $\mathbf{P}(\text{Cloudy}) = <0.5, 0.5>$ , suppose this returns true
- Sprinkler is evidence variable with value true, we set  
 $w \leftarrow w \times P(\text{Sprinkler} = \text{true}|\text{Cloudy} = \text{true}) = 0.1$
- Sample from  $\mathbf{P}(\text{Rain}|\text{Cloudy} = \text{true}) = <0.8, 0.2>$ , suppose this returns true
- WetGrass is evidence variable with value true, we set  
 $w \leftarrow w \times P(\text{WetGrass} = \text{true}|\text{Sprinkler} = \text{true}, \text{Rain} = \text{true}) = w \times 0.99 = 0.099$
- **Sample returned=[true,true,true,true] with weight 0.099 tallied under Rain = true**

# Likelihood weighting: why it works

---

- Define all the non-evidence variables  $\mathbf{Z} = \mathbf{X} \cup \mathbf{Y}$  ( $\mathbf{X}$  the query,  $\mathbf{Y}$  the hidden) and consider sampling distribution  $S$

$$S(\mathbf{z}, \mathbf{e}) = \prod_{i=1}^l P(z_i | \text{parents}(Z_i))$$

- $S$ 's sample values for each  $Z_i$  is influenced by the evidence among  $Z_i$ 's ancestors
- But  $S$  pays no attention when sampling  $Z_i$ 's value to evidence from  $Z_i$ 's non-ancestors; so it's not sampling from the true posterior probability distribution!
- But the likelihood weight  $w$  makes up for the difference

$$w(\mathbf{z}, \mathbf{e}) = \prod_{i=1}^m P(e_i | \text{parents}(E_i))$$

# Likelihood weighting: why it works

---

- Since two products cover all the variables in the network, we can write:

$$P(\mathbf{z}, \mathbf{e}) = \underbrace{\prod_{i=1}^l P(z_i | \text{parents}(Z_i))}_{S(\mathbf{z}, \mathbf{e})} \underbrace{\prod_{i=1}^m P(e_i | \text{parents}(E_i))}_{w(\mathbf{z}, \mathbf{e})}$$

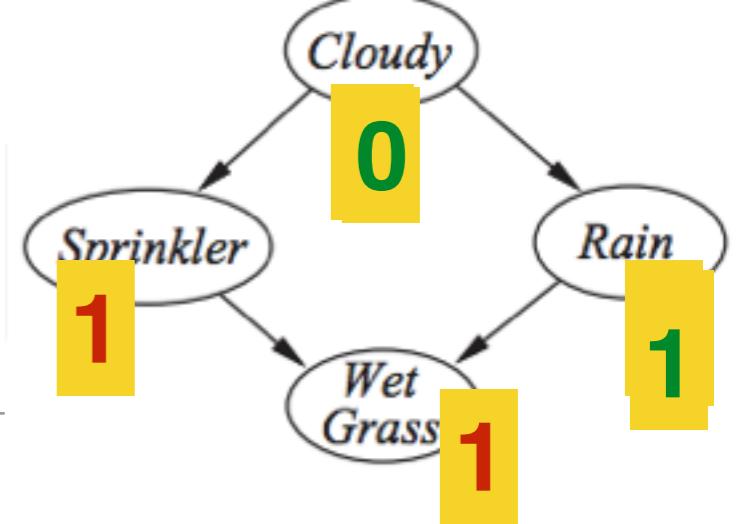
- With this, it is easy to derive that likelihood weighting is **consistent** (tutorial exercise)
- much more efficient than rejection sampling but efficiency gets worse when number of evidence variables increases
- Problem: most samples will have **very small weights** as the number of evidence variables increases
- The weighted estimate will be dominated by tiny fraction of samples that give more than infinitesimal likelihood to the evidence

# Markov Chain Monte-Carlo (MCMC) Algorithm

---

- MCMC algorithm: create an event from a previous event, rather than generate all events from scratch
- Helpful to think of the BN as having a current **state** specifying a value for each variable
- Consecutive state is generated by **sampling a value for one of the non-evidence variables  $X_i$  conditioned on the current values of variables in the Markov blanket of  $X_i$**
- Recall that **Markov blanket** consists of parents, children, and children's parents
- Algorithm randomly wanders around state space **flipping one variable at a time** and keeping evidence variables fixed

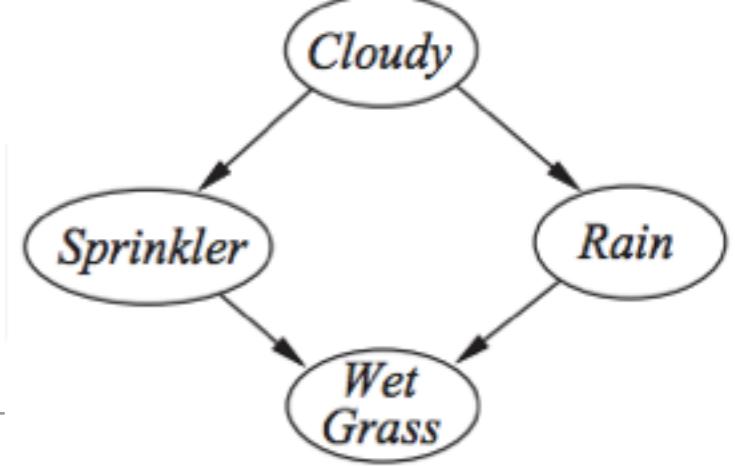
# MCMC/Gibbs Algorithm



- Consider query  $\mathbf{P}(\text{Rain}|\text{Sprinkler} = \text{true}, \text{WetGrass} = \text{true})$  once more
- Sprinkler and WetGrass (evidence variables) are fixed to their observed values, hidden variables Cloudy and Rain are initialised randomly (e.g. true and false)
- Assume Initial state is [true,true, false,true]
- Execute repeatedly:
- Sample Cloudy given values of Markov blanket, i.e. sample from  $\mathbf{P}(\text{Cloudy}|\text{Sprinkler} = \text{true}, \text{Rain} = \text{false})$
- Suppose result is false, new state is [false,true, false,true]
- Sample Rain given values of Markov blanket, i.e. sample from  $\mathbf{P}(\text{Rain}|\text{Sprinkler} = \text{true}, \text{Cloudy} = \text{false}, \text{WetGrass} = \text{true})$
- Suppose we obtain Rain = true, new state [false,true,true,true]

# MCMC Algorithm

---



- Each visited state is a sample
- If there were 20 states where Rain=1 and 60 states where Rain=0,  
 $P_{\text{est}}(\text{Rain}|\text{Sprinkler} = \text{true}, \text{WetGrass} = \text{true}) = <20/80, 60/80>$
- Basic idea of proof that MCMC is consistent:  
The sampling process settles into a “dynamic equilibrium” in which the long-term **fraction of time spent in each state is exactly proportional to its posterior probability**
- MCMC is a very powerful method used for all kinds of things involving probabilities

# Summary

---

- Exact inference is in general intractable
- Stochastic approximation techniques such as likelihood weighting and MCMC can give **reasonable estimates** of the true posterior probabilities in a network and can cope with **much larger networks** than can exact algorithms
- There are 2 important families of approximation methods that were not covered here: **variational approximations and belief propagation.**
- Next time: Time and Uncertainty I



# **Inf2D-Reasoning and Agents**

## Spring 2017

Lecture (2)6:-Time and Uncertainty 1

Peggy Seriès, [pseries@inf.ed.ac.uk](mailto:pseries@inf.ed.ac.uk)

Based on previous slides by A. Lascarides

# Where are we?

---

- Last time . . .
- Completed our account of Bayesian Networks
- Dealt with methods for exact and approximate inference in BNs
- Enumeration, variable elimination, sampling, MCMC
- Today . . . Time and uncertainty I

# Time and uncertainty

---

- So far we have only seen methods for describing uncertainty in **static** environments
- Every variable had a fixed value, we assumed that nothing changes during evidence collection or diagnosis
- Many practical domains involve uncertainty about **dynamic** processes that can be modelled with probabilistic methods
- Basic idea straightforward: imagine **one BN model of the problem for every time step** and reason about changes between them

# States and observations

---

- series of snapshots (**time slices**) will be used to describe process of change
- Snapshots consist of **observable** random variables  $\mathbf{E}_t$  and **non-observable** ones  $\mathbf{X}_t$
- For simplicity, we assume same subset of variables is observable in each time slice (but this is not necessary)
- Observation at  $t$  will be  $\mathbf{E}_t = \mathbf{e}_t$  for some set of values  $\mathbf{e}_t$
- Assume that states start at  $t = 0$  and evidence starts arriving at  $t = 1$

# States and observations

---

- Example: underground security guard wants to predict whether it is raining but only observes every morning whether director comes in carrying umbrella
- For each day,  $\mathbf{E}_t$  contains variable  $U_t$  (whether the umbrella appears) and  $\mathbf{X}_t$  contains state variable  $R_t$  (whether it's raining)
- Evidence  $U_1, U_2, \dots$ , state variables  $R_0, R_1, \dots$
- Use notation  $a : b$  to denote sequences of integers,  
e.g.  $U_1, U_2, U_3 = U_{1:3}$



# Stationary processes and the Markov assumption

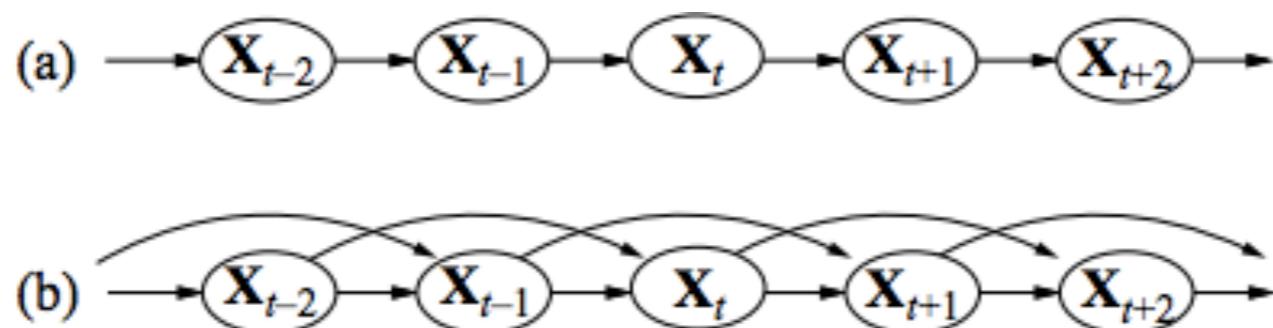
---

- How do we specify dependencies among variables?
- Natural to arrange them in temporal order (causes usually precede effects)
- Problem: set of variables is unbounded (one for each time slice), so we would have to
  - specify unbounded number of conditional probability tables
  - specify an unbounded number of parents for each of these
- Solution to first problem: we assume that **changes are caused by a stationary process** – the laws that govern the process do not change themselves when shifted in time (not to be confused with “static”)
- For example,  $P(U_t|Parents(U_t))$  does not depend on  $t$

# Stationary processes and the Markov assumption

---

- Solution to second problem: **Markov assumption** – the current state only depends on a finite history of previous states
- Such processes are called **Markov processes** or **Markov chains**
- Simplest form: **first-order** Markov processes, every state depends only on predecessor state
- We can write this as  $P(X_t | X_{0:t-1}) = P(X_t | X_{t-1})$
- This conditional distribution is called **transition model**
- Difference between **first-order** and **second-order** Markov processes:



# Stationary processes and the Markov assumption

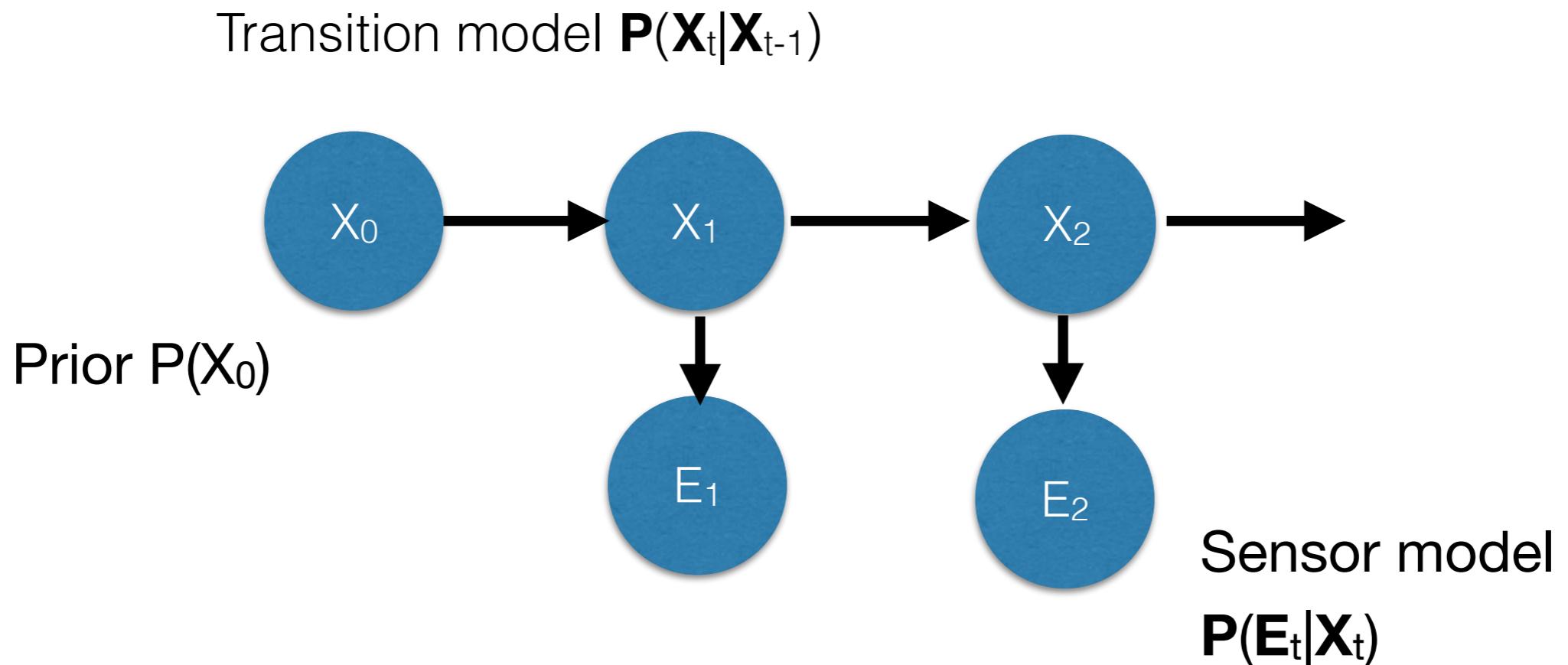
---

- Additionally, we will assume that **evidence variables depend only on current state (sensor Markov assumption)**:  
$$P(E_t | X_{0:t}, E_{0:t-1}) = P(E_t | X_t)$$
- This is called the **sensor model (observation model)** of the system
- Notice direction of dependence: state causes evidence (but inference goes in other direction!)
- In umbrella world, rain causes umbrella to appear
- Finally, we need a **prior distribution** over initial states  $P(X_0)$
- These three distributions give a specification of the complete JPD

$$P(X_0, X_1, \dots, X_t, E_1, \dots, E_t) = P(X_0) \prod_{i=1}^t P(X_i | X_{i-1}) P(E_i | X_i)$$

# Stationary processes and the Markov assumption

---

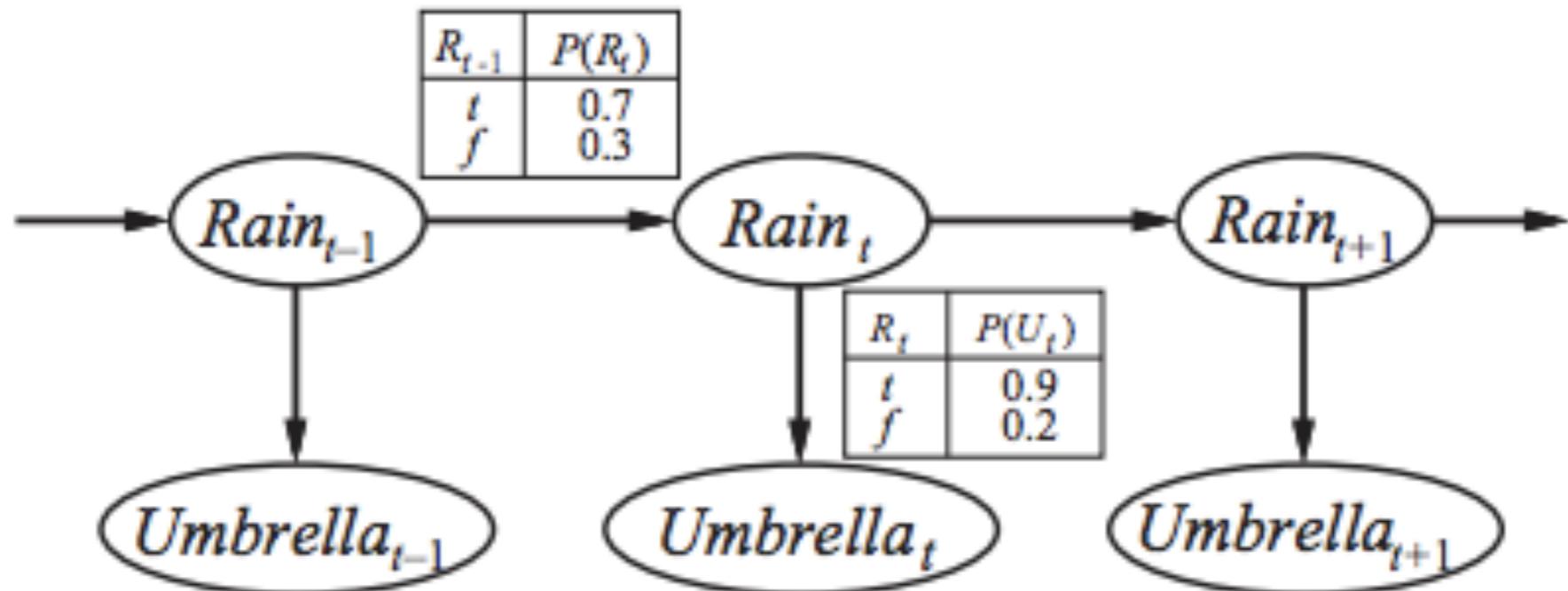


$$\mathbf{P}(\mathbf{X}_0, \mathbf{X}_1, \dots, \mathbf{X}_t, \mathbf{E}_1, \dots, \mathbf{E}_t) = \mathbf{P}(\mathbf{X}_0) \prod_{i=1}^t \mathbf{P}(\mathbf{X}_i|\mathbf{X}_{i-1}) \mathbf{P}(\mathbf{E}_i|\mathbf{X}_i)$$

# Umbrella World Example



- Transition model  $P(\text{Rain}_t | \text{Rain}_{t-1})$ , sensor model  $P(\text{Umbrella}_t | \text{Rain}_t)$



- Rain depends only on rainfall on previous day, whether this is reasonable depends on domain!

# Stationary processes and the Markov assumption

---

- If Markov assumptions seems too simplistic for some domains (and hence, inaccurate), two measures can be taken:
  - We can **increase the order** of the Markov process model
  - We can **increase the set of state variables**
- For example, add information about season, pressure or humidity
- But this will also increase prediction requirements (problem alleviated if we add new sensors)
- Example: dependency of predicting movement of robot on battery power level:
  - add battery level sensor

# Inference tasks in temporal models

---

- we will need inference methods for a number of tasks:
- **Filtering/monitoring or state estimation**: compute **current state** given evidence to date, i.e.  $\mathbf{P}(\mathbf{X}_t | \mathbf{e}_{1:t})$
- Interestingly, an almost identical calculation yields the likelihood of the evidence sequence  $\mathbf{P}(\mathbf{e}_{1:t})$
- **Prediction**: computing posterior distribution over a **future state** given evidence to date:  $\mathbf{P}(\mathbf{X}_{t+k} | \mathbf{e}_{1:t})$
- **Smoothing/hindsight**: compute posterior distribution of **past state**,  $\mathbf{P}(\mathbf{X}_k | \mathbf{e}_{1:t})$ ,  $0 \leq k < t$
- **Most likely explanation**: compute  $\arg \max_{\mathbf{x}_{1:t}} \mathbf{P}(\mathbf{x}_{1:t} | \mathbf{e}_{1:t})$  i.e. the most likely sequence of states given evidence

# Filtering and Prediction

---

- Done by **recursive estimation**: compute result for  $t+1$  by doing it for  $t$  and then updating with new evidence  $e_{t+1}$ . That is, for some function  $f$  :

$$P(\mathbf{X}_{t+1} | \mathbf{e}_{1:t+1}) = f(\mathbf{e}_{t+1}, P(\mathbf{X}_t | \mathbf{e}_{1:t}))$$

$$P(\mathbf{X}_{t+1} | \mathbf{e}_{1:t+1}) = P(\mathbf{X}_{t+1} | \mathbf{e}_{1:t}, \mathbf{e}_{t+1})$$

decomposing 1:t and t+1

# Filtering and Prediction

---

- Done by **recursive estimation**: compute result for  $t+1$  by doing it for  $t$  and then updating with new evidence  $e_{t+1}$ . That is, for some function  $f$  :

$$P(\mathbf{X}_{t+1} | \mathbf{e}_{1:t+1}) = f(\mathbf{e}_{t+1}, P(\mathbf{X}_t | \mathbf{e}_{1:t}))$$

$$\begin{aligned} P(\mathbf{X}_{t+1} | \mathbf{e}_{1:t+1}) &= P(\mathbf{X}_{t+1} | \mathbf{e}_{1:t}, \mathbf{e}_{t+1}) \\ &= \alpha P(\mathbf{e}_{t+1} | \mathbf{X}_{t+1}, \mathbf{e}_{1:t}) P(\mathbf{X}_{t+1} | \mathbf{e}_{1:t}) \end{aligned} \quad (\text{Bayes' rule})$$

$$P(\text{alc}) = \alpha P(\text{cla})P(\text{a})$$

$$P(\text{alb}, \text{c}) = \alpha P(\text{cla}, \text{b})P(\text{alb})$$

# Filtering and Prediction

---

- Done by **recursive estimation**: compute result for  $t+1$  by doing it for  $t$  and then updating with new evidence  $e_{t+1}$ . That is, for some function  $f$  :

$$P(\mathbf{X}_{t+1} | \mathbf{e}_{1:t+1}) = f(\mathbf{e}_{t+1}, P(\mathbf{X}_t | \mathbf{e}_{1:t}))$$

$$\begin{aligned} P(\mathbf{X}_{t+1} | \mathbf{e}_{1:t+1}) &= P(\mathbf{X}_{t+1} | \mathbf{e}_{1:t}, \mathbf{e}_{t+1}) \\ &= \alpha P(\mathbf{e}_{t+1} | \mathbf{X}_{t+1}, \mathbf{e}_{1:t}) P(\mathbf{X}_{t+1} | \mathbf{e}_{1:t}) && (\text{Bayes' rule}) \\ &= \alpha P(\mathbf{e}_{t+1} | \mathbf{X}_{t+1}) P(\mathbf{X}_{t+1} | \mathbf{e}_{1:t}) && (\text{Markov property}) \end{aligned}$$

# Filtering and Prediction

---

- Done by **recursive estimation**: compute result for  $t+1$  by doing it for  $t$  and then updating with new evidence  $e_{t+1}$ . That is, for some function  $f$  :

$$P(\mathbf{X}_{t+1} | \mathbf{e}_{1:t+1}) = f(\mathbf{e}_{t+1}, P(\mathbf{X}_t | \mathbf{e}_{1:t}))$$

$$\begin{aligned} P(\mathbf{X}_{t+1} | \mathbf{e}_{1:t+1}) &= P(\mathbf{X}_{t+1} | \mathbf{e}_{1:t}, e_{t+1}) \\ &= \alpha P(e_{t+1} | \mathbf{X}_{t+1}, \mathbf{e}_{1:t}) P(\mathbf{X}_{t+1} | \mathbf{e}_{1:t}) && (\text{Bayes' rule}) \\ &= \alpha P(e_{t+1} | \mathbf{X}_{t+1}) P(\mathbf{X}_{t+1} | \mathbf{e}_{1:t}) && (\text{Markov property}) \\ &= \alpha P(e_{t+1} | \mathbf{X}_{t+1}) \sum_{\mathbf{x}_t} P(\mathbf{X}_{t+1} | \mathbf{x}_t, \mathbf{e}_{1:t}) P(\mathbf{x}_t | \mathbf{e}_{1:t}) && (\text{conditioning}) \end{aligned}$$

$\sum_{\mathbf{x}_t} P(\mathbf{X}_{t+1}, \mathbf{x}_t | \mathbf{e}_{1:t})$  conditioning  
 $= \sum_{\mathbf{x}_t} P(\mathbf{X}_{t+1} | \mathbf{x}_t, \mathbf{e}_{1:t}) P(\mathbf{x}_t | \mathbf{e}_{1:t})$   
using product rule  $P(a,b) = P(alb)P(b)$

# Filtering and Prediction

---

- Done by **recursive estimation**: compute result for  $t+1$  by doing it for  $t$  and then updating with new evidence  $e_{t+1}$ . That is, for some function  $f$  :

$$P(\mathbf{X}_{t+1} | \mathbf{e}_{1:t+1}) = f(\mathbf{e}_{t+1}, P(\mathbf{X}_t | \mathbf{e}_{1:t}))$$

$$\begin{aligned} P(\mathbf{X}_{t+1} | \mathbf{e}_{1:t+1}) &= P(\mathbf{X}_{t+1} | \mathbf{e}_{1:t}, e_{t+1}) \\ &= \alpha P(e_{t+1} | \mathbf{X}_{t+1}, \mathbf{e}_{1:t}) P(\mathbf{X}_{t+1} | \mathbf{e}_{1:t}) && \text{(Bayes' rule)} \\ &= \alpha P(e_{t+1} | \mathbf{X}_{t+1}) P(\mathbf{X}_{t+1} | \mathbf{e}_{1:t}) && \text{(Markov property)} \\ &= \alpha P(e_{t+1} | \mathbf{X}_{t+1}) \sum_{\mathbf{x}_t} P(\mathbf{X}_{t+1} | \mathbf{x}_t, \mathbf{e}_{1:t}) P(\mathbf{x}_t | \mathbf{e}_{1:t}) && \text{(conditioning)} \\ &= \alpha P(e_{t+1} | \mathbf{X}_{t+1}) \sum_{\mathbf{x}_t} P(\mathbf{X}_{t+1} | \mathbf{x}_t) P(\mathbf{x}_t | \mathbf{e}_{1:t}) && \text{(Markov assumption)} \end{aligned}$$

# Filtering Equation

---

$$P(\mathbf{X}_{t+1}|\mathbf{e}_{1:t+1}) = \alpha P(\mathbf{e}_{t+1}|\mathbf{X}_{t+1}) \sum_{\mathbf{x}_t} P(\mathbf{X}_{t+1}|\mathbf{x}_t) P(\mathbf{x}_t|\mathbf{e}_{1:t})$$

**Sensor model**

**Transition model**

**Recursion**

The diagram illustrates the filtering equation with three main components:

- Sensor model:** Represented by a red arrow pointing to the term  $P(\mathbf{e}_{t+1}|\mathbf{X}_{t+1})$ .
- Transition model:** Represented by a green arrow pointing to the term  $\sum_{\mathbf{x}_t} P(\mathbf{X}_{t+1}|\mathbf{x}_t)$ .
- Recursion:** Represented by a blue arrow pointing to the term  $P(\mathbf{x}_t|\mathbf{e}_{1:t})$ .

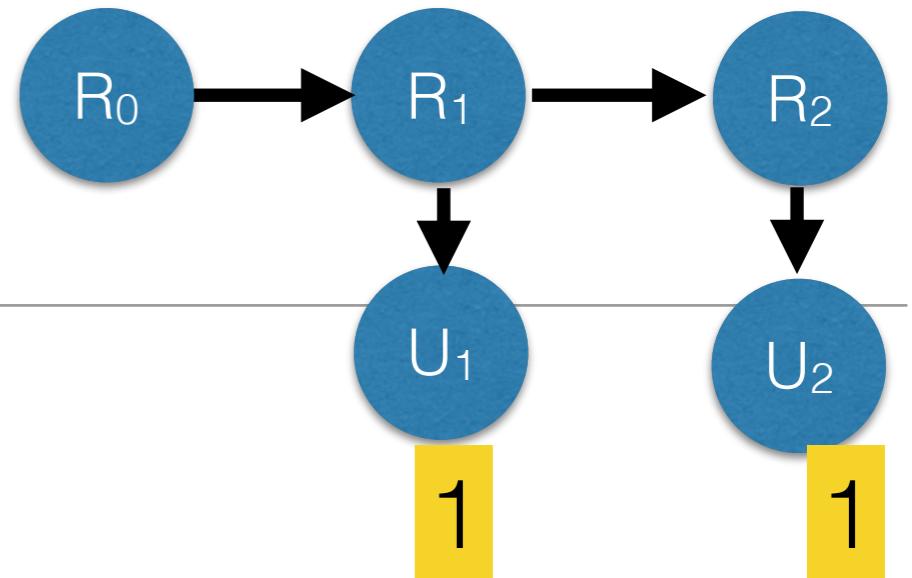
# Filtering and Prediction

---

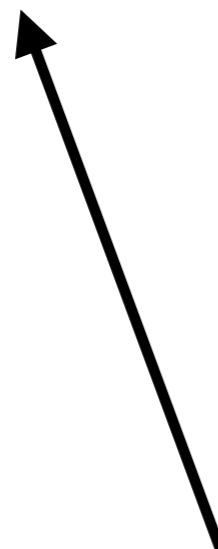
- We can view estimate  $\mathbf{P}(\mathbf{X}_t | \mathbf{e}_{1:t})$  as “**message**”  $f_{1:t}$  propagated along the sequence, modified by each transition, and updated by each observation.
- We write this process as  $f_{1:t+1} = \alpha \text{Forward}(f_{1:t}, \mathbf{e}_{t+1})$
- **Time and space requirements** for this are **constant** regardless of length of sequence
- This is extremely important for agent design!

## Example

- Compute  $P(R_2|u_{1:2})$ ,  $U_1 = \text{true}$ ,  $U_2 = \text{true}$
- Suppose  $\mathbf{P}(R_0) = \langle 0.5, 0.5 \rangle$
- Recursive equations:



$$\mathbf{P}(R_2|u_1, u_2) = \alpha \mathbf{P}(u_2|R_2) \sum_{r_1} \mathbf{P}(R_2|r_1) P(r_1|u_1)$$

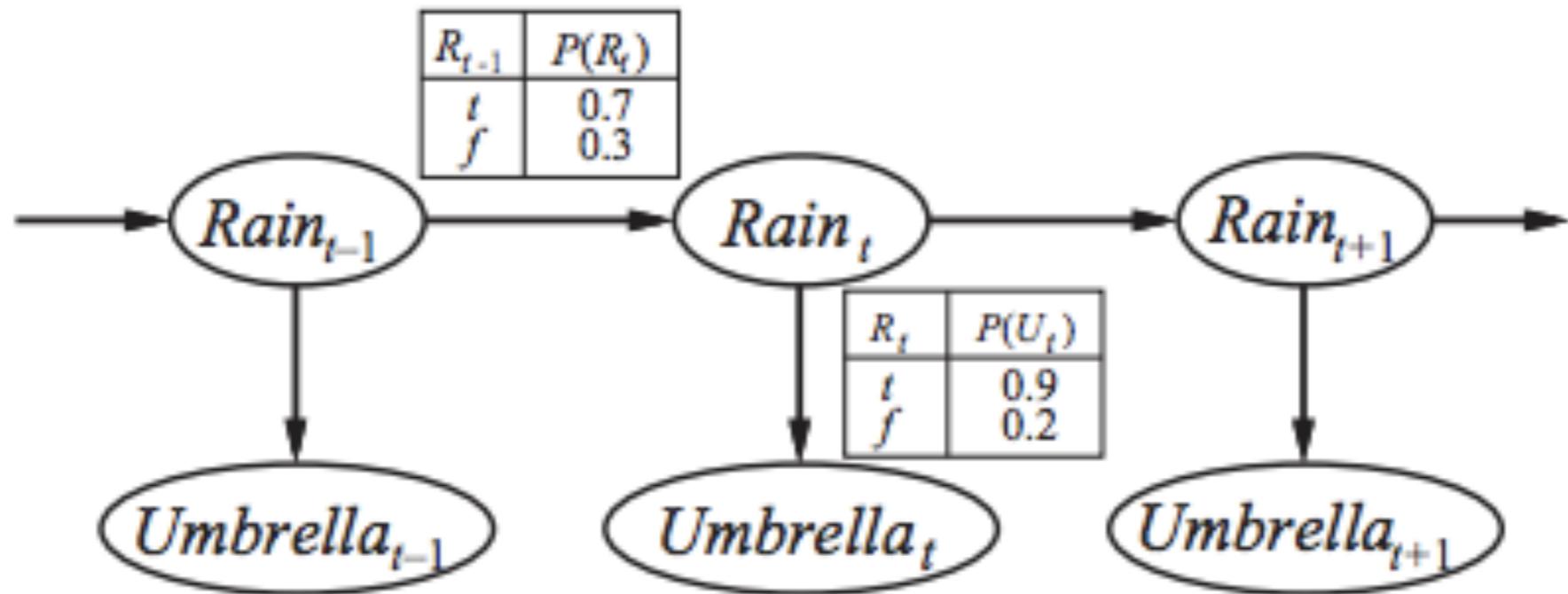


$$\begin{aligned}\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t+1}) &= \mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t}, \mathbf{e}_{t+1}) \\ &= \alpha \mathbf{P}(\mathbf{e}_{t+1}|\mathbf{X}_{t+1}, \mathbf{e}_{1:t}) \mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t}) && (\text{Bayes' rule}) \\ &= \alpha \mathbf{P}(\mathbf{e}_{t+1}|\mathbf{X}_{t+1}) \mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t}) && (\text{Markov property}) \\ &= \alpha \mathbf{P}(\mathbf{e}_{t+1}|\mathbf{X}_{t+1}) \sum_{\mathbf{x}_t} \mathbf{P}(\mathbf{X}_{t+1}|\mathbf{x}_t, \mathbf{e}_{1:t}) P(\mathbf{x}_t|\mathbf{e}_{1:t}) && (\text{conditioning}) \\ &= \alpha \mathbf{P}(\mathbf{e}_{t+1}|\mathbf{X}_{t+1}) \sum_{\mathbf{x}_t} \mathbf{P}(\mathbf{X}_{t+1}|\mathbf{x}_t) P(\mathbf{x}_t|\mathbf{e}_{1:t}) && (\text{Markov assumption})\end{aligned}$$

# Umbrella World Example



- Transition model  $P(\text{Rain}_t | \text{Rain}_{t-1})$ , sensor model  $P(\text{Umbrella}_t | \text{Rain}_t)$

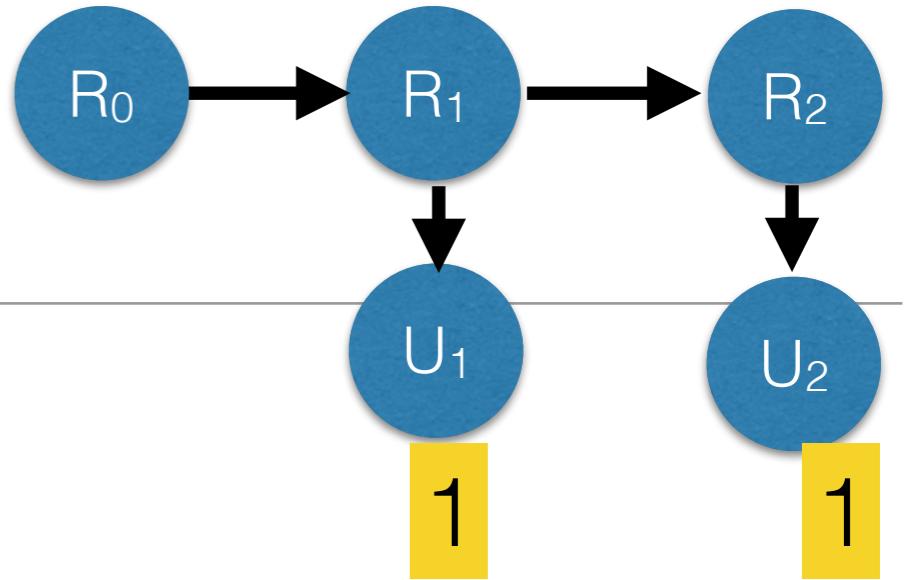


- Rain depends only on rainfall on previous day, whether this is reasonable depends on domain!

## Example

---

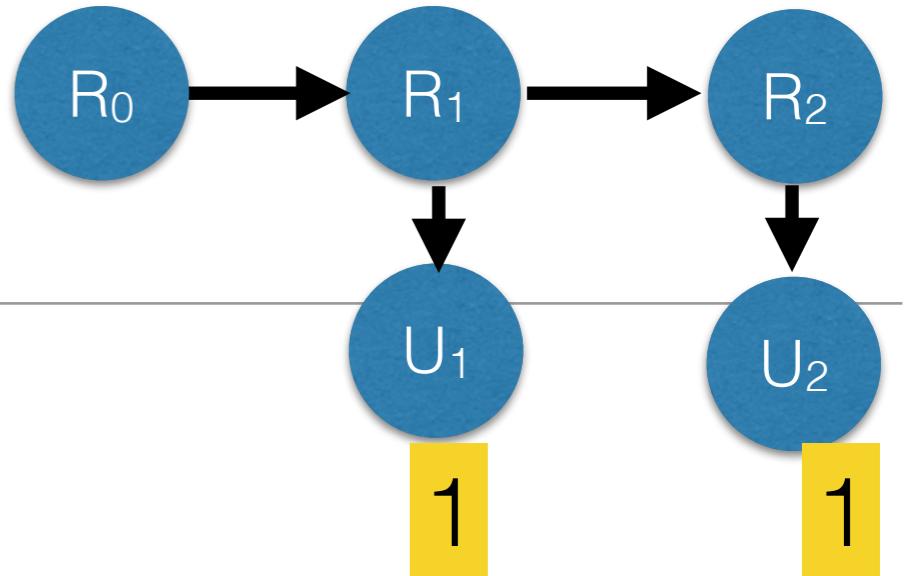
- $\mathbf{P}(R_0) = \langle 0.5, 0.5 \rangle$
- $\mathbf{P}(R_1) = \sum_{r_0} \mathbf{P}(R_1|r_0)P(r_0)$
- $\mathbf{P}(R_1) = \langle 0.7, 0.3 \rangle \times 0.5 + \langle 0.3, 0.7 \rangle \times 0.5 = \langle 0.5, 0.5 \rangle$



## Example

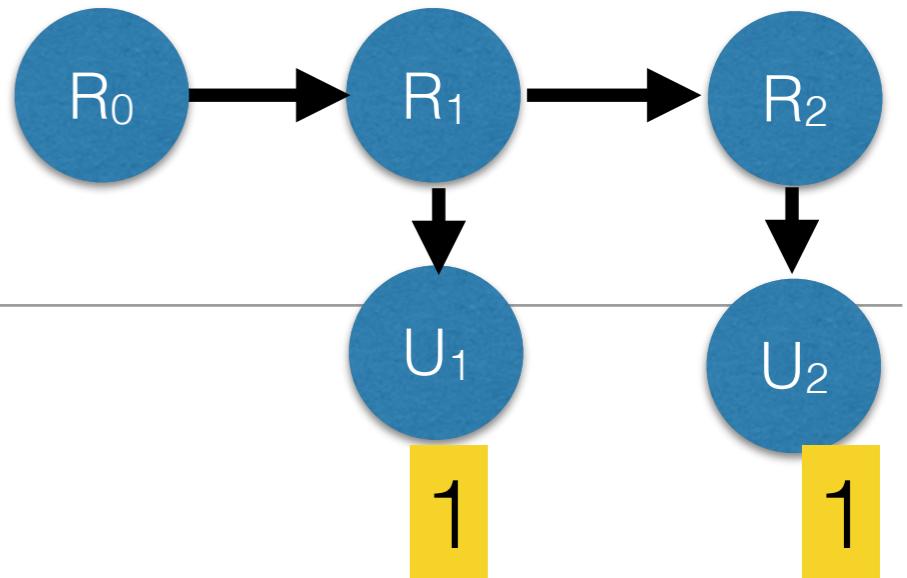
---

- $\mathbf{P}(R_0) = \langle 0.5, 0.5 \rangle$
- $\mathbf{P}(R_1) = \sum_{r_0} \mathbf{P}(R_1|r_0)P(r_0)$
- $\mathbf{P}(R_1) = \langle 0.7, 0.3 \rangle \times 0.5 + \langle 0.3, 0.7 \rangle \times 0.5 = \langle 0.5, 0.5 \rangle$



$$\begin{aligned}\mathbf{P}(R_1|u_1) &= \alpha' \mathbf{P}(u_1|R_1) \sum_{r_0} \mathbf{P}(R_1|r_0)P(r_0) \\ &= \alpha' \langle 0.9, 0.2 \rangle (\langle 0.7, 0.3 \rangle \times 0.5 + \langle 0.3, 0.7 \rangle \times 0.5) \\ &= \alpha' \langle 0.9, 0.2 \rangle \langle 0.5, 0.5 \rangle \\ &= \langle 0.818, 0.182 \rangle\end{aligned}$$

## Example



- $\mathbf{P}(R_0) = \langle 0.5, 0.5 \rangle$
- $\mathbf{P}(R_1) = \sum_{r_0} \mathbf{P}(R_1|r_0)P(r_0)$
- $\mathbf{P}(R_1) = \langle 0.7, 0.3 \rangle \times 0.5 + \langle 0.3, 0.7 \rangle \times 0.5 = \langle 0.5, 0.5 \rangle$

$$\begin{aligned}\mathbf{P}(R_1|u_1) &= \alpha' \mathbf{P}(u_1|R_1) \sum_{r_0} \mathbf{P}(R_1|r_0)P(r_0) \\ &= \alpha' \langle 0.9, 0.2 \rangle (\langle 0.7, 0.3 \rangle \times 0.5 + \langle 0.3, 0.7 \rangle \times 0.5) \\ &= \alpha' \langle 0.9, 0.2 \rangle \langle 0.5, 0.5 \rangle \\ &= \langle 0.818, 0.182 \rangle\end{aligned}$$

$$\mathbf{P}(R_2|u_1, u_2) = \alpha \mathbf{P}(u_2|R_2) \sum_{r_1} \mathbf{P}(R_2|r_1)P(r_1|u_1)$$

$$\begin{aligned}\mathbf{P}(R_2|u_1, u_2) &= \alpha \langle 0.9, 0.2 \rangle (\langle 0.7, 0.3 \rangle \times 0.818 + \langle 0.3, 0.7 \rangle \times 0.182) \\ &= \alpha \langle 0.9, 0.2 \rangle \langle 0.627, 0.373 \rangle \\ &= \alpha \langle 0.565, 0.075 \rangle \\ &= \langle 0.883, 0.117 \rangle\end{aligned}$$

# Filtering and Prediction

---

- Prediction works like filtering without new evidence
- Computation involves **only transition model** and not sensor model:

$$P(\mathbf{X}_{t+k+1} | \mathbf{e}_{1:t}) = \sum_{\mathbf{x}_{t+k}} P(\mathbf{X}_{t+k+1} | \mathbf{x}_{t+k}) P(\mathbf{x}_{t+k} | \mathbf{e}_{1:t})$$

- As we predict further and further into the future, distribution of rain converges to the prior  $\langle 0.5, 0.5 \rangle$
- This is called the **stationary distribution** of the Markov process (the more uncertainty, the quicker it will converge)

# Filtering and Prediction

---

- We can use the above method to compute **likelihood** of evidence sequence  $\mathbf{P}(\mathbf{e}_{1:t})$
- Useful so as to compare different temporal models that might have produced the same evidence sequence
- Use a likelihood message  $\mathbf{l}_{1:t} = \mathbf{P}(\mathbf{X}_t, \mathbf{e}_{1:t})$  and compute

$$\mathbf{l}_{1:t+1} = \alpha \text{FORWARD}(\mathbf{l}_{1:t}, \mathbf{e}_{t+1})$$

- ▶ Once we compute  $\mathbf{l}_{1:t}$ , summing out yields likelihood

$$L_{1:t} = P(\mathbf{e}_{1:t}) = \sum_{\mathbf{x}_t} \mathbf{l}_{1:t}(\mathbf{x}_t, \mathbf{e}_{1:t})$$

# Summary

---

- Time and uncertainty (states and observations)
- Stationarity and Markov assumptions
- Inference in temporal models: filtering, prediction, smoothing ...
- Filtering equation and prediction equation
- Next time: Time and Uncertainty II



# **Inf2D-Reasoning and Agents**

## Spring 2017

Lecture (2)7:-Time and Uncertainty 2

Peggy Seriès, [pseries@inf.ed.ac.uk](mailto:pseries@inf.ed.ac.uk)

Based on previous slides by A. Lascarides

# Where are we?

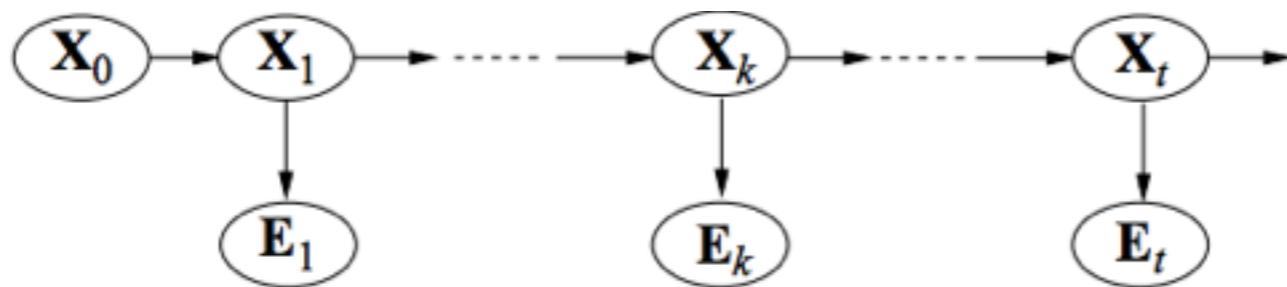
---

- Last time . . .
- Time in BN, reasoning about uncertainty
- Markov assumption, stationarity
- Algorithms for reasoning about temporal processes
- Filtering and prediction
- Today . . . Time and uncertainty II

# Smoothing

---

- Smoothing is computation of distribution of **past states given current evidence**, i.e.  $P(X_k | e_{1:t})$ ,  $1 \leq k < t$



- Easiest to view as **2-step process** (up to  $k$ , then  $k+1$  to  $t$ )

$$\begin{aligned} P(X_k | e_{1:t}) &= P(X_k | e_{1:k}, e_{k+1:t}) \\ &= \alpha P(X_k | e_{1:k}) P(e_{k+1:t} | X_k, e_{1:k}) && \text{(Bayes' rule)} \\ &= \alpha P(X_k | e_{1:k}) P(e_{k+1:t} | X_k) && \text{(conditional independence)} \\ &= \alpha f_{1:k} b_{k+1:t} \end{aligned}$$

- Here “backward” message is  $b_{k+1:t} = P(e_{k+1:t} | X_k)$  analogous to “forward” message

# Forward message (Filtering)

---

- Formula for **forward message**:

$$P(\mathbf{X}_{t+1} | \mathbf{e}_{1:t+1}) = \alpha P(\mathbf{e}_{t+1} | \mathbf{X}_{t+1}) \sum_{\mathbf{x}_t} P(\mathbf{X}_{t+1} | \mathbf{x}_t) P(\mathbf{x}_t | \mathbf{e}_{1:t})$$

The diagram shows the forward message formula with three annotations:

- A red arrow points to the term  $P(\mathbf{e}_{t+1} | \mathbf{X}_{t+1})$  with the label **Sensor model**.
- A green arrow points to the term  $P(\mathbf{X}_{t+1} | \mathbf{x}_t)$  with the label **Transition model**.
- A blue arrow points to the summation symbol ( $\sum$ ) with the label **Recursion**.

- The forward message is computed **from t=1 to k** (forward!)

# Backward Message

---

- Formula for **backward message**:

$$P(\mathbf{e}_{k+1:t} | \mathbf{x}_k) = \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1} | \mathbf{x}_{k+1}) P(\mathbf{e}_{k+2:t} | \mathbf{x}_{k+1}) P(\mathbf{x}_{k+1} | \mathbf{x}_k)$$

Sensor model                          Recursion                          Transition model

- Define  $\mathbf{b}_{k+1:t} = \text{Backward}(\mathbf{b}_{k+2:t}, \mathbf{e}_{k+1:t})$
- The backward message is computed **from t to k+1** (backwards!)
- The backward phase has to be **initialised** with  $b_{t+1:t} = P(\mathbf{e}_{t+1:t} | \mathbf{x}_t) = 1$  (a vector of 1s) because probability of observing empty sequence is 1

## Backward message (proof)

---

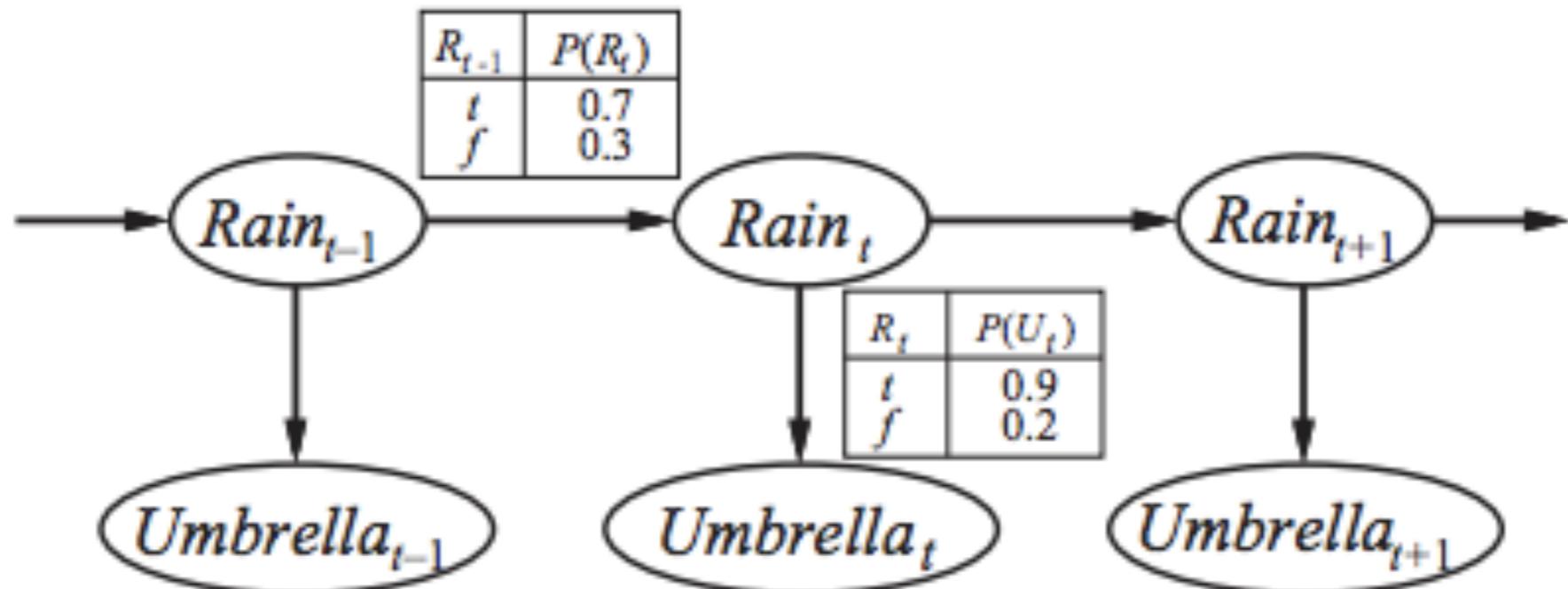
$$\begin{aligned}\mathbf{P}(\mathbf{e}_{k+1:t} \mid \mathbf{X}_k) &= \sum_{\mathbf{x}_{k+1}} \mathbf{P}(\mathbf{e}_{k+1:t} \mid \mathbf{X}_k, \mathbf{x}_{k+1}) \mathbf{P}(\mathbf{x}_{k+1} \mid \mathbf{X}_k) \quad (\text{conditioning on } \mathbf{X}_{k+1}) \\ &= \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1:t} \mid \mathbf{x}_{k+1}) \mathbf{P}(\mathbf{x}_{k+1} \mid \mathbf{X}_k) \quad (\text{by conditional independence}) \\ &= \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1}, \mathbf{e}_{k+2:t} \mid \mathbf{x}_{k+1}) \mathbf{P}(\mathbf{x}_{k+1} \mid \mathbf{X}_k) \\ &= \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1} \mid \mathbf{x}_{k+1}) P(\mathbf{e}_{k+2:t} \mid \mathbf{x}_{k+1}) \mathbf{P}(\mathbf{x}_{k+1} \mid \mathbf{X}_k),\end{aligned}\tag{15.9}$$

re the last step follows by the conditional independence of  $\mathbf{e}_{k+1}$  and  $\mathbf{e}_{k+2:t}$ , given  $\mathbf{X}_{k+1}$ .  
he three factors in this summation, the first and third are obtained directly from the mode  
the second is the “recursive call.” Using the message notation, we have

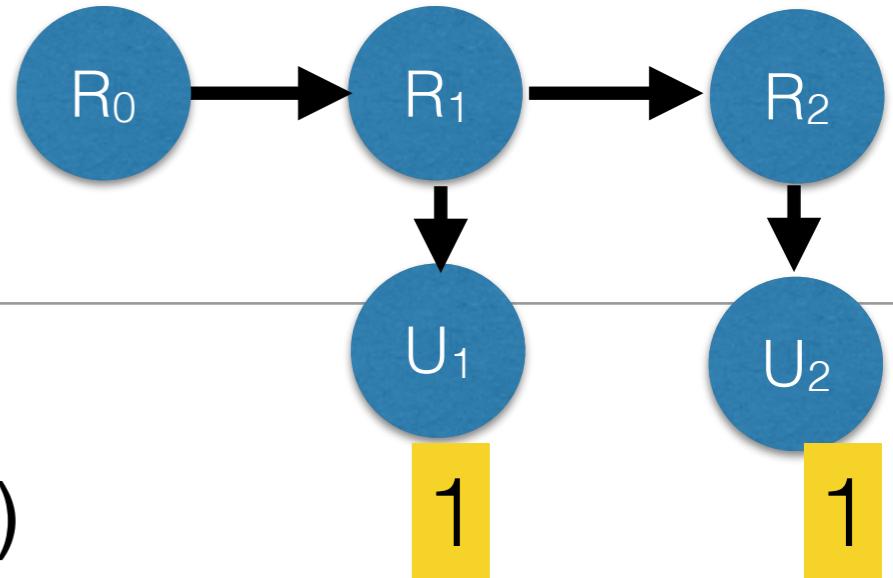
# Umbrella World Example



- Transition model  $P(\text{Rain}_t | \text{Rain}_{t-1})$ , sensor model  $P(\text{Umbrella}_t | \text{Rain}_t)$



Example: Compute  $\mathbf{P}(R_1|u_1, u_2)$

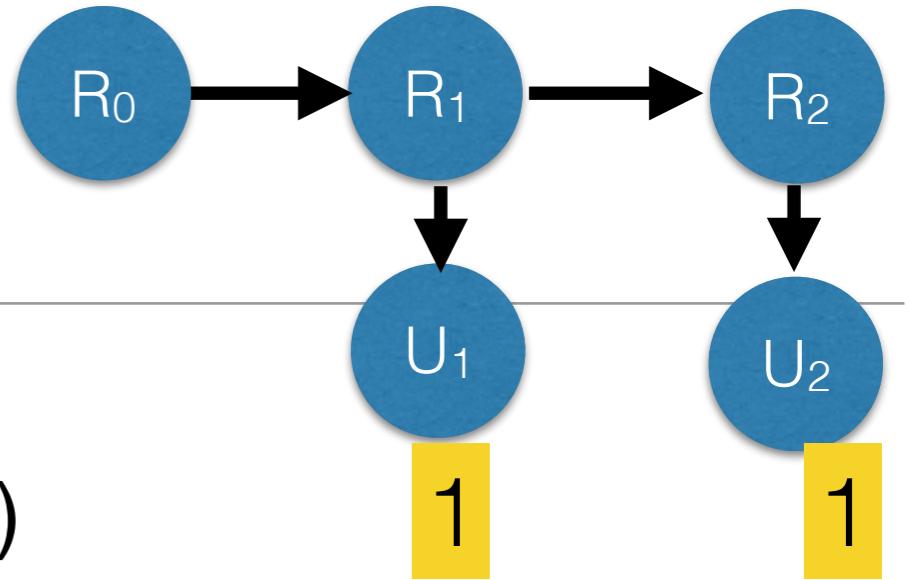


- We have  $\mathbf{P}(R_1|u_1, u_2) = \alpha \mathbf{P}(R_1|u_1) \mathbf{P}(u_2|R_1)$

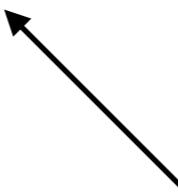
$$\begin{aligned}\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:t}) &= \mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:k}, \mathbf{e}_{k+1:t}) \\ &= \alpha \mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:k}) \mathbf{P}(\mathbf{e}_{k+1:t}|\mathbf{X}_k, \mathbf{e}_{1:k}) \\ &= \alpha \mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:k}) \mathbf{P}(\mathbf{e}_{k+1:t}|\mathbf{X}_k) \\ &= \alpha \mathbf{f}_{1:k} \mathbf{b}_{k+1:t}\end{aligned}$$



Example: Compute  $\mathbf{P}(R_1|u_1, u_2)$



- We have  $\mathbf{P}(R_1|u_1, u_2) = \alpha \mathbf{P}(R_1|u_1) \mathbf{P}(u_2|R_1)$

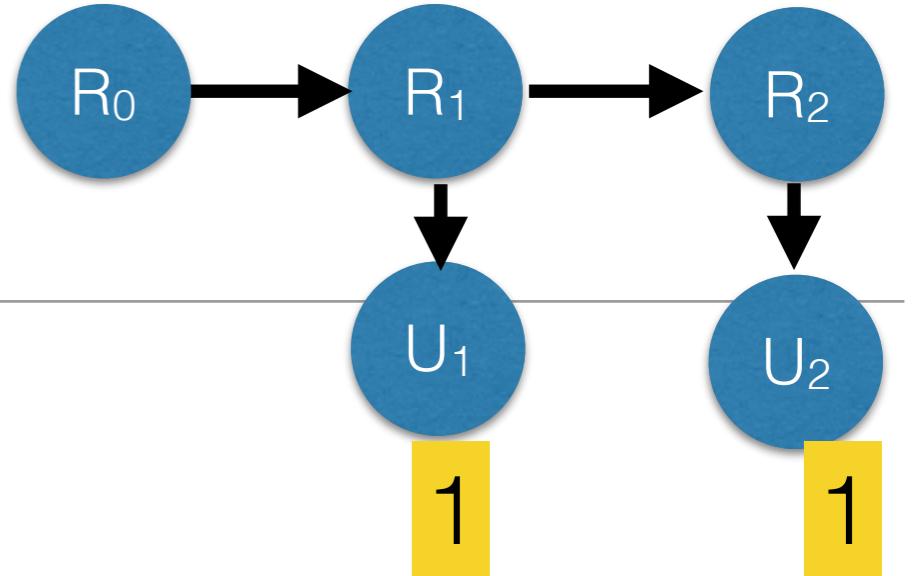


$$\begin{aligned}\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:t}) &= \mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:k}, \mathbf{e}_{k+1:t}) \\ &= \alpha \mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:k}) \mathbf{P}(\mathbf{e}_{k+1:t}|\mathbf{X}_k, \mathbf{e}_{1:k}) \\ &= \alpha \mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:k}) \mathbf{P}(\mathbf{e}_{k+1:t}|\mathbf{X}_k) \\ &= \alpha \mathbf{f}_{1:k} \mathbf{b}_{k+1:t}\end{aligned}$$

- So we'll need to remind ourselves of  $\mathbf{P}(R_1|u_1)$  from last lecture:

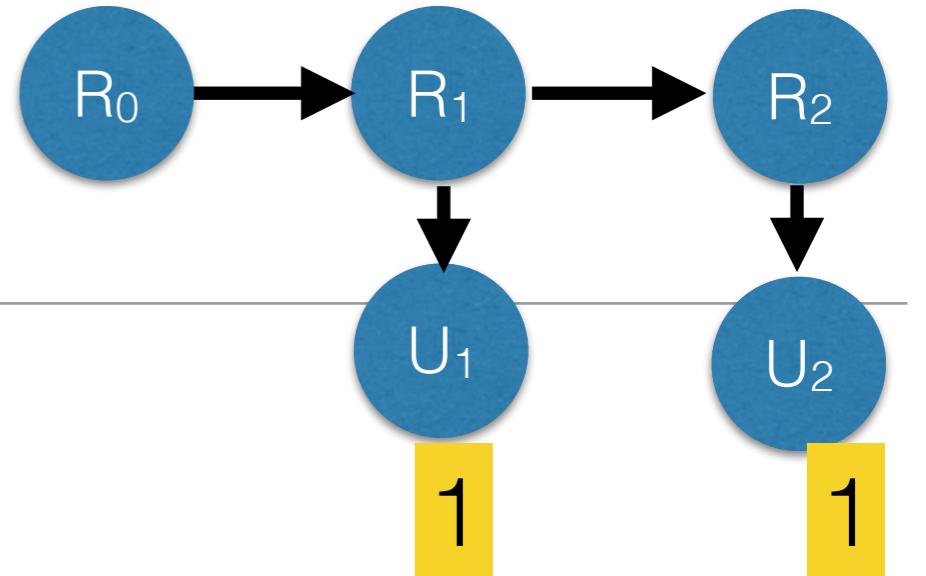
$$\begin{aligned}\mathbf{P}(R_1|u_1) &= \alpha' \mathbf{P}(u_1|R_1) \sum_{r_0} \mathbf{P}(R_1|r_0) P(r_0) \\ &= \alpha' \langle 0.9, 0.2 \rangle (\langle 0.7, 0.3 \rangle \times 0.5 + \langle 0.3, 0.7 \rangle \times 0.5) \\ &= \alpha' \langle 0.9, 0.2 \rangle \langle 0.5, 0.5 \rangle \\ &= \langle 0.818, 0.182 \rangle\end{aligned}$$

Example: Compute  $\mathbf{P}(R_1|u_1, u_2)$



- $\mathbf{P}(u_2|R_1)$  is the **backward message**, we use the formula

$$\mathbf{P}(\mathbf{e}_{k+1:t}|\mathbf{x}_k) = \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1}|\mathbf{x}_{k+1})P(\mathbf{e}_{k+2:t}|\mathbf{x}_{k+1})\mathbf{P}(\mathbf{x}_{k+1}|\mathbf{x}_k)$$



Example: Compute  $\mathbf{P}(R_1|u_1, u_2)$

- $\mathbf{P}(u_2|R_1)$  is the **backward message**, we use the formula

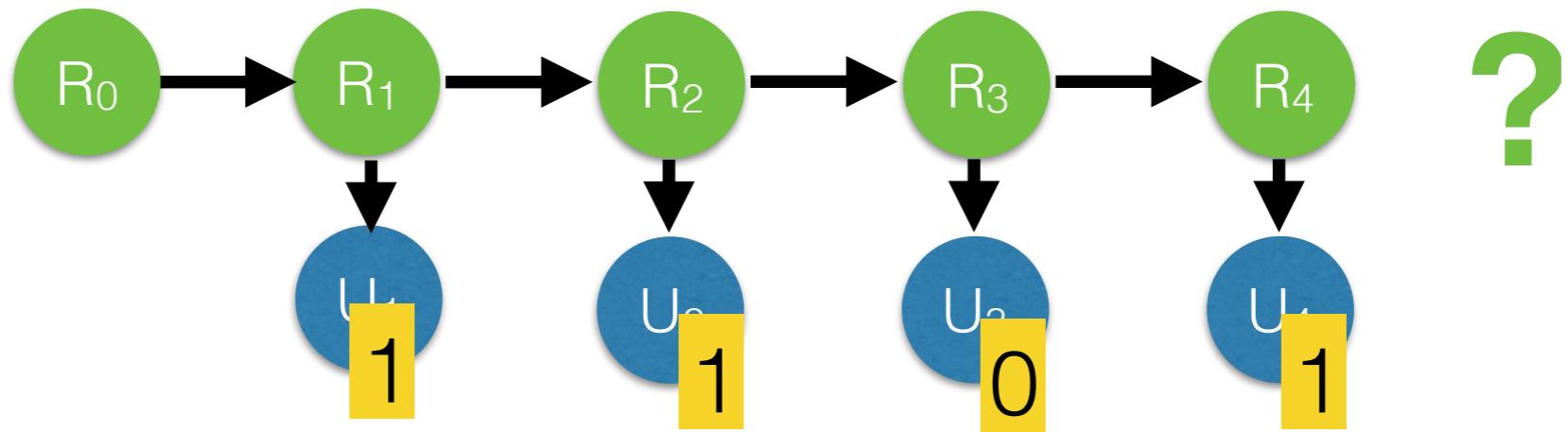
$$\mathbf{P}(\mathbf{e}_{k+1:t}|\mathbf{x}_k) = \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1}|\mathbf{x}_{k+1})P(\mathbf{e}_{k+2:t}|\mathbf{x}_{k+1})\mathbf{P}(\mathbf{x}_{k+1}|\mathbf{x}_k)$$

$$\begin{aligned}\mathbf{P}(u_2|R_1) &= \sum_{r_2} P(u_2|r_2)P(|r_2)\mathbf{P}(r_2|R_1) \\ &= (0.9 \times 1 \times \langle 0.7, 0.3 \rangle) + (0.2 \times 1 \times \langle 0.3, 0.7 \rangle) = \langle 0.69, 0.41 \rangle\end{aligned}$$

- So finally:  $\mathbf{P}(R_1|u_1, u_2) = \alpha \langle 0.818, 0.182 \rangle \langle 0.69, 0.41 \rangle = \langle 0.883, 0.117 \rangle$
- So our confidence that it rained on Day 1 increases when we see the umbrella on the second day as well as the first.
- A simple improved version of this that stores results runs in linear time (**forward-backward algorithm**)

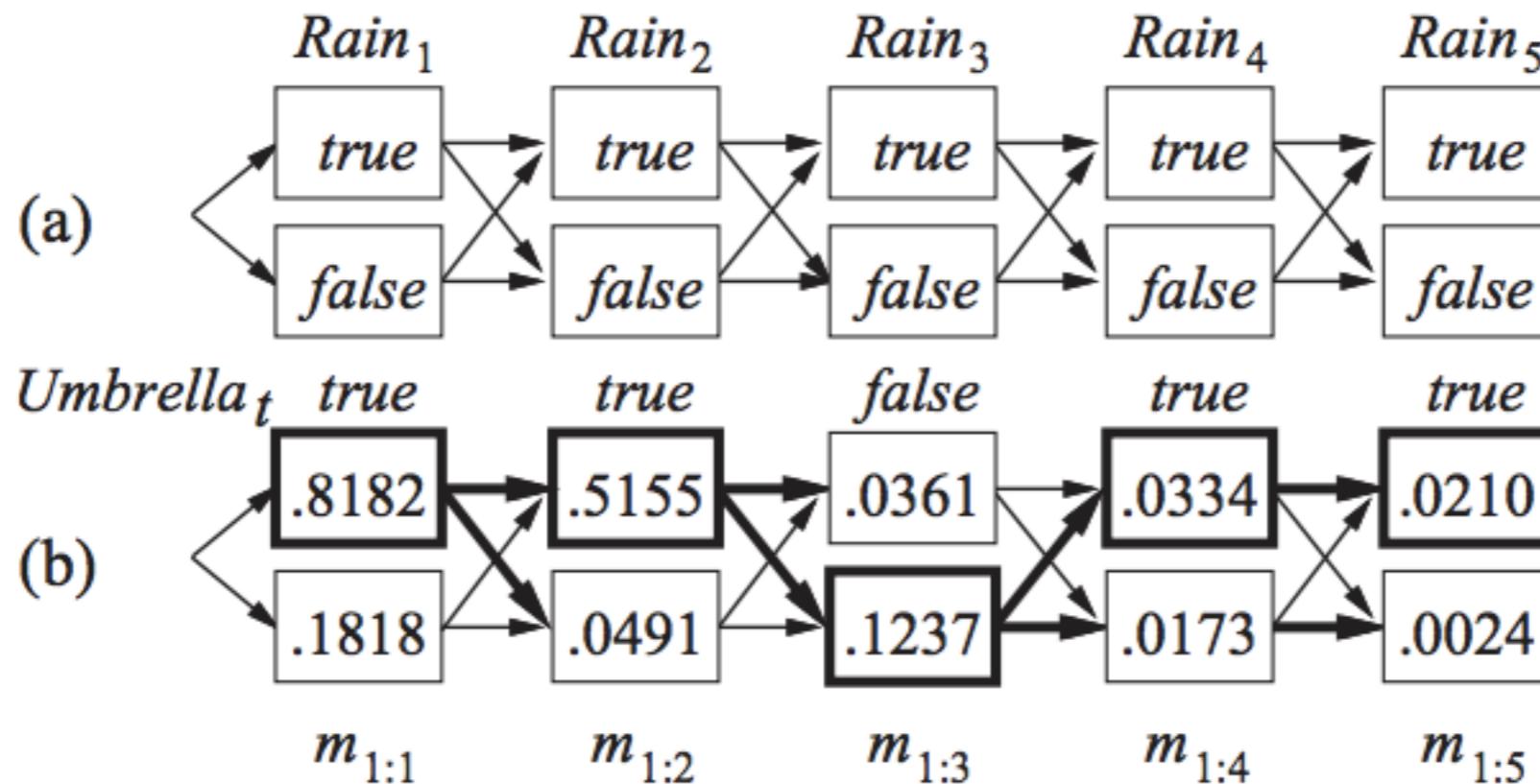
# Finding the most likely sequence

---

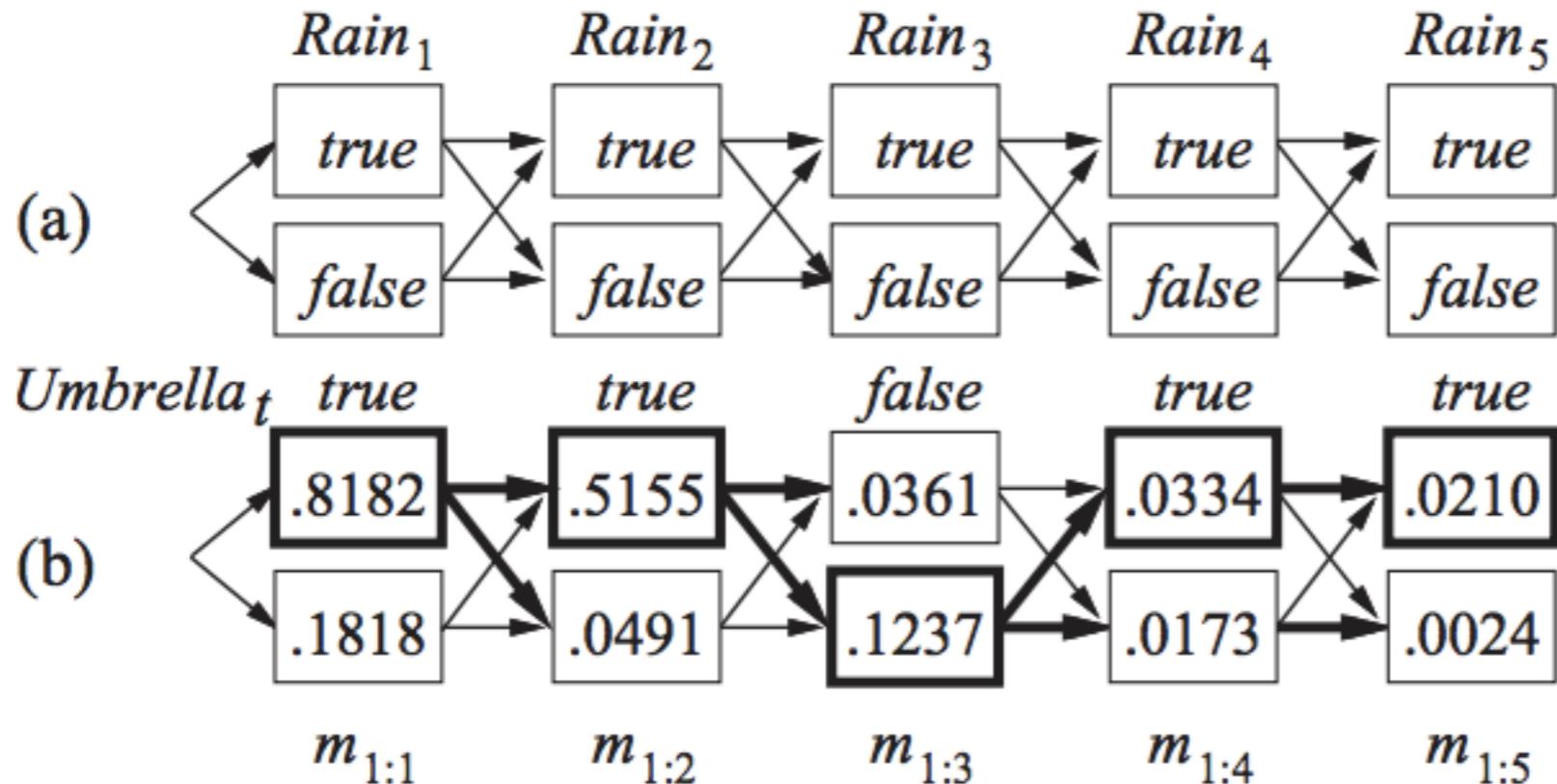


- Suppose [true,true, false,true,true] is the umbrella sequence for first five days, what is the **most likely weather sequence** that caused it?
- Could use smoothing procedure to find posterior distribution for weather at each step and then use most likely weather at each step to construct sequence
- NO! Smoothing considers distributions over individual time steps, but we **must consider joint probabilities over all time steps**
- Actual algorithm is based on viewing each sequence as path through a graph (nodes=states at each time step)

# Finding the most likely sequence



- Look at states with  $Rain_5 = \text{true}$
- Because of Markov property, most likely path to this state consists of most likely path to some state at time 4 followed by transition to  $Rain_5 = \text{true}$
- **Recursive:** compute most likely paths to state  $x_{t+1}$  using most likely path to state  $x_t$



Define **message** as

$$\mathbf{m}_{1:t} = \max_{x_1, \dots, x_{t-1}} \mathbf{P}(x_1, \dots, x_{t-1}, X_t | \mathbf{e}_{1:t}) = \begin{bmatrix} M_{1:t}^1 \\ M_{1:t}^2 \end{bmatrix} \rightarrow \begin{cases} \text{for } X_t = t \\ \text{for } X_t = f \end{cases}$$

- $m_1 = P(R_1 | u_1) = <0.8182, 0.1818>$  choose true
- assuming we've taken true for R1;  
 $m_{1:2} = P(u_2 | R_2) P(R_2 | r_1) P(r_1 | u_1) = <0.9 \times 0.7 \times 0.8182, 0.2 \times 0.3 \times 0.8182> = <0.5155, 0.049>$

# Finding the most likely sequence

---

- There is a recursive relationship between most likely paths to  $x_{t+1}$  and most likely paths to each state  $x_t$

$$\begin{aligned} & \max_{x_1 \dots x_t} P(x_1, \dots, x_t, X_{t+1} | e_{1:t+1}) \\ &= \alpha P(e_{t+1} | X_{t+1}) \max_{x_t} (P(X_{t+1} | x_t) \max_{x_1 \dots x_{t-1}} P(x_1, \dots, x_{t-1}, x_t | e_{1:t})) \end{aligned}$$

- This is like **filtering** only that the forward message is replaced by

$$m_{1:t} = \max_{x_1 \dots x_{t-1}} P(x_1, \dots, x_{t-1}, X_t | e_{1:t})$$

- And summation is now replaced by maximisation
- **Viterbi Algorithm**

# Viterbi Algorithm

---



- Andrew Viterbi 1967
- has found **universal application** in decoding the convolutional codes used in both CDMA and GSM digital cellular, dial-up modems, satellite, deep-space communications, and 802.11 wireless LANs.
- also commonly used in **speech recognition**, speech synthesis, keyword spotting, computational linguistics, and bioinformatics.
- For example, in **speech-to-text**, the acoustic signal is treated as the observed sequence of events, and a string of text is considered to be the "hidden cause" of the acoustic signal. The Viterbi algorithm finds the most likely string of text given the acoustic signal.

# Hidden Markov Models

---

- So far, we have seen a general model for temporal probabilistic reasoning (independent of transition/sensor models)
- In following lecture we are going to look at more concrete models and applications
- **Hidden Markov Models (HMMs)**: temporal probabilistic model in which state of the process is described by a **single variable**
- Like our umbrella example (single variable  $\text{Rain}_t$ )
- More than one variable can be accommodated, but only by combining them into a single “mega-variable”
- We will see that the restricted structure of HMMs allows for a very simple and elegant **matrix implementation** of basic algorithms

## Applications [ edit ]

HMMs can be applied in many fields where the goal is to recover a data sequence that is not immediately observable. These include:

- Computational finance<sup>[18]</sup>
- Single Molecule Kinetic analysis<sup>[19]</sup>
- Cryptanalysis
- Speech recognition
- Speech synthesis
- Part-of-speech tagging
- Document Separation in scanning solutions
- Machine translation
- Partial discharge
- Gene prediction
- Handwriting Recognition
- Alignment of bio-sequences
- Time Series Analysis
- Activity recognition
- Protein folding<sup>[20]</sup>
- Metamorphic Virus Detection<sup>[21]</sup>
- DNA Motif Discovery<sup>[22]</sup>

# Summary

---

- The forward-backward algorithm
- Finding the most likely sequence (Viterbi algorithm)
- Talked about HMMs
- HMMs: single state variable, simplifies algorithms (see other courses for these)
- Huge significance, for example in speech recognition:  
$$P(\text{words}|\text{signal}) = \alpha P(\text{signal}|\text{words})P(\text{words})$$
- Vast array of applications, but also limits.
- Next time: Dynamic Bayesian Networks



# **Inf2D-Reasoning and Agents**

## Spring 2017

Lecture (2)8:- Dynamic Bayesian Networks

Peggy Seriès, [pseries@inf.ed.ac.uk](mailto:pseries@inf.ed.ac.uk)

Based on previous slides by A. Lascarides

# Where are we?

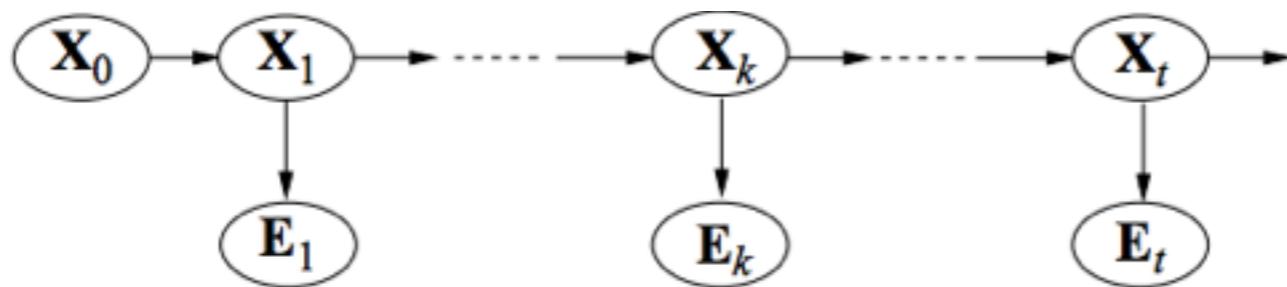
---

- Last time . . .
- Inference in temporal models: **smoothing and most likely sequence**
- Discussed general model (**forward-backward algorithm** for smoothing, **Viterbi algorithm** for most likely sequence)
- Specific instances: **HMMs**
- today . . . bit more about HMM, Kalman filters and mostly Dynamic Bayesian Networks

# Smoothing

---

- Smoothing is computation of distribution of **past states given current evidence**, i.e.  $P(X_k | e_{1:t})$ ,  $1 \leq k < t$



- Easiest to view as **2-step process** (up to  $k$ , then  $k+1$  to  $t$ )

$$\begin{aligned} P(X_k | e_{1:t}) &= P(X_k | e_{1:k}, e_{k+1:t}) \\ &= \alpha P(X_k | e_{1:k}) P(e_{k+1:t} | X_k, e_{1:k}) && \text{(Bayes' rule)} \\ &= \alpha P(X_k | e_{1:k}) P(e_{k+1:t} | X_k) && \text{(conditional independence)} \\ &= \alpha f_{1:k} b_{k+1:t} \end{aligned}$$

- Here “backward” message is  $b_{k+1:t} = P(e_{k+1:t} | X_k)$  analogous to “forward” message

# Forward message (Filtering)

---

- Formula for **forward message**:

$$P(\mathbf{X}_{t+1} | \mathbf{e}_{1:t+1}) = \alpha P(\mathbf{e}_{t+1} | \mathbf{X}_{t+1}) \sum_{\mathbf{x}_t} P(\mathbf{X}_{t+1} | \mathbf{x}_t) P(\mathbf{x}_t | \mathbf{e}_{1:t})$$

The diagram illustrates the forward message formula. It shows the equation  $P(\mathbf{X}_{t+1} | \mathbf{e}_{1:t+1}) = \alpha P(\mathbf{e}_{t+1} | \mathbf{X}_{t+1}) \sum_{\mathbf{x}_t} P(\mathbf{X}_{t+1} | \mathbf{x}_t) P(\mathbf{x}_t | \mathbf{e}_{1:t})$ . Three arrows point to different parts of the equation: a red arrow points to  $P(\mathbf{e}_{t+1} | \mathbf{X}_{t+1})$  with the label "Sensor model"; a green arrow points to  $\sum_{\mathbf{x}_t} P(\mathbf{X}_{t+1} | \mathbf{x}_t) P(\mathbf{x}_t | \mathbf{e}_{1:t})$  with the label "Transition model"; and a blue arrow points to  $P(\mathbf{x}_t | \mathbf{e}_{1:t})$  with the label "Recursion".

- The forward message is computed **from t=1 to k** (forward!)

# Backward Message

---

- Formula for **backward message**:

$$P(\mathbf{e}_{k+1:t} | \mathbf{x}_k) = \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1} | \mathbf{x}_{k+1}) P(\mathbf{e}_{k+2:t} | \mathbf{x}_{k+1}) P(\mathbf{x}_{k+1} | \mathbf{x}_k)$$

Sensor model                          Recursion                          Transition model

- Define  $\mathbf{b}_{k+1:t} = \text{Backward}(\mathbf{b}_{k+2:t}, \mathbf{e}_{k+1:t})$
- The backward message is computed **from t to k+1** (backwards!)
- The backward phase has to be **initialised** with  $\mathbf{b}_{t+1:t} = P(\mathbf{e}_{t+1:t} | \mathbf{X}_t) = 1$  (a vector of 1s) because probability of observing empty sequence is 1

# Hidden Markov Models

---

- **Hidden Markov Models (HMMs)**: temporal probabilistic model in which state of the process is described by a single variable
- Like our umbrella example (single variable  $\text{Rain}_t$ )
- More than one variable can be accommodated, but only by combining them into a single “mega-variable”
- the restricted structure of HMMs allows for a very simple and elegant **matrix implementation** of basic algorithms

# Hidden Markov Models

---

- $X_t$  is a single, discrete variable (usually  $E_t$  is too)
- Domain of  $X_t$  is  $\{1, \dots, S\}$  ( $S$  possible states)
- The transition model  $T = P(X_t | X_{t-1})$  becomes a  $S \times S$  **Transition matrix**,  
 $T_{ij} = P(X_t = j | X_{t-1} = i)$
- For example the Transition matrix for the umbrella world is

$$T = P(X_t | X_{t-1}) = \begin{pmatrix} 0.7 & 0.3 \\ 0.3 & 0.7 \end{pmatrix}$$

- The **sensor model** can also expressed as a matrix. In this case, the value of  $E_t$  is known at time  $t$  (call it  $e_t$ ), so we need only specify, for each state  $i$ ,  $P(e_t | X_t = i)$ .
- For mathematical convenience we place these values into an  $S \times S$  **diagonal matrix**,  $O_t$ , whose  $i^{\text{th}}$  diagonal entry is  $P(e_t | X_t = i)$  and whose other entries are 0.

# Hidden Markov Models

---

- For example, if on day 1 we have  $U_1=\text{true}$  and on day 3 we have  $U_3=\text{false}$

$$\mathbf{O}_1 = \begin{pmatrix} 0.9 & 0 \\ 0 & 0.2 \end{pmatrix}; \quad \mathbf{O}_3 = \begin{pmatrix} 0.1 & 0 \\ 0 & 0.8 \end{pmatrix}$$

- **Smoothing** : Now if we use column vectors to represent the forward and the backward message, all the computations become simple matrix-vector operations.
- The forward message can be written:

$$\mathbf{f}_{1:t+1} = \alpha \mathbf{O}_{t+1} \mathbf{T}^\top \mathbf{f}_{1:t}$$

- The backward message can be written:

$$\mathbf{b}_{k+1:t} = \mathbf{T} \mathbf{O}_{k+1} \mathbf{b}_{k+2:t}$$

- The product between matrices  $MN$  is defined only if the second matrix has the same number of rows as the first has columns:  
if  $M$  is of size  $a \times b$ , then  $N$  must be of size  $b \times c$ , and resulting matrix is of size  $a \times c$ .  
If the matrices are of appropriate size, then:

$$MN = \sum_j m_{ij}n_{jk}$$

In the example:

$$M + N = \begin{pmatrix} 1 & 2 \\ 2 & 3 \end{pmatrix} \begin{pmatrix} 3 & 4 \\ 5 & 2 \end{pmatrix} = \begin{pmatrix} 1 \times 3 + 2 \times 5 & 1 \times 4 + 2 \times 2 \\ 2 \times 3 + 3 \times 5 & 2 \times 4 + 3 \times 2 \end{pmatrix} = \begin{pmatrix} 13 & 8 \\ 21 & 14 \end{pmatrix}$$

- The transpose of a matrix  $M$  is written  $M^T$  and is formed turned rows into columns.  
For example:

$$N^T = \begin{pmatrix} 3 & 4 \\ 5 & 2 \end{pmatrix}^T = \begin{pmatrix} 3 & 5 \\ 4 & 2 \end{pmatrix}$$

# Hidden Markov Models

---

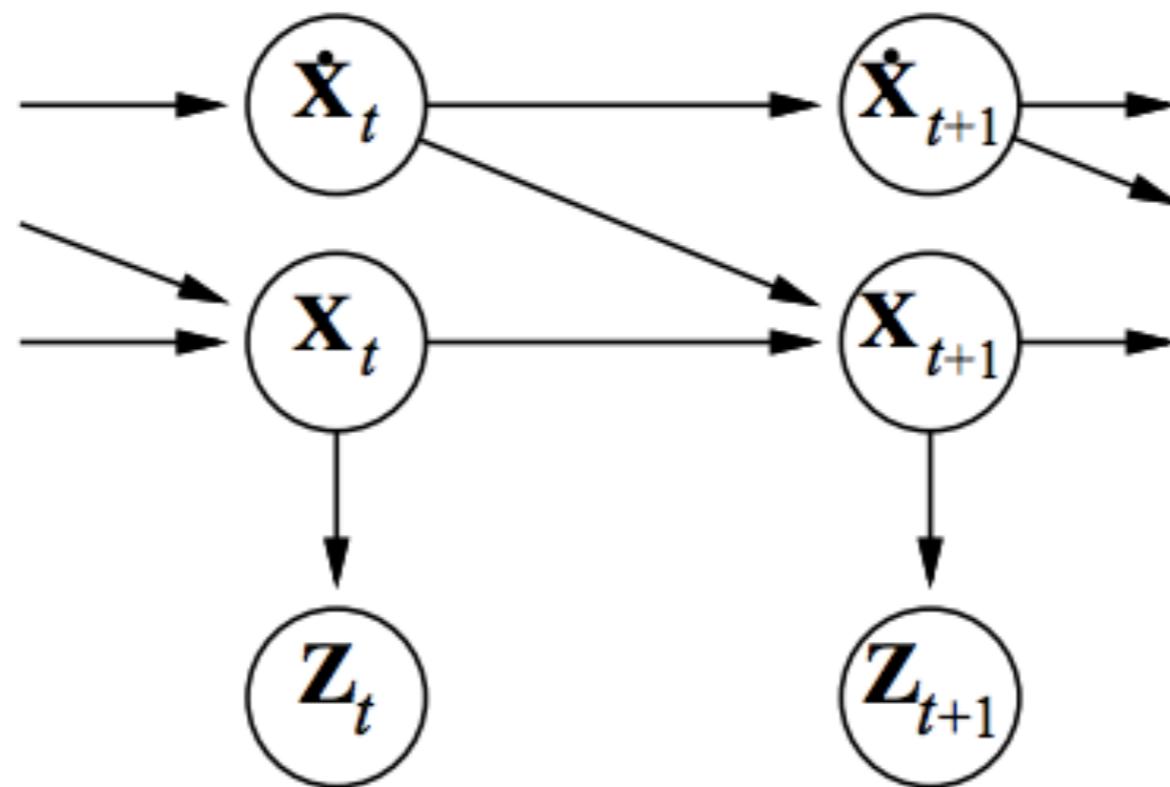
- The **time complexity** of the forward–backward algorithm applied to a sequence of length  $t$  is  $O(S^2t)$ , because each step requires multiplying an  $S$ -element vector by an  $S \times S$  matrix.
- The **space requirement** is  $O(St)$ , because the forward pass stores  $t$  vectors of size  $S$ .
- Besides providing an **elegant description of the filtering and smoothing algorithms** for HMMs, the matrix formulation reveals opportunities for **improved algorithms**.

# a note about Kalman Filters

---

Modelling systems described by a set of continuous variables,  
e.g., tracking a bird flying— $\mathbf{X}_t = X, Y, Z, \dot{X}, \dot{Y}, \dot{Z}$ .

Airplanes, robots, ecosystems, economies, chemical plants, planets, . . .



Gaussian prior, linear Gaussian transition model and sensor model

# Constructing DBNs

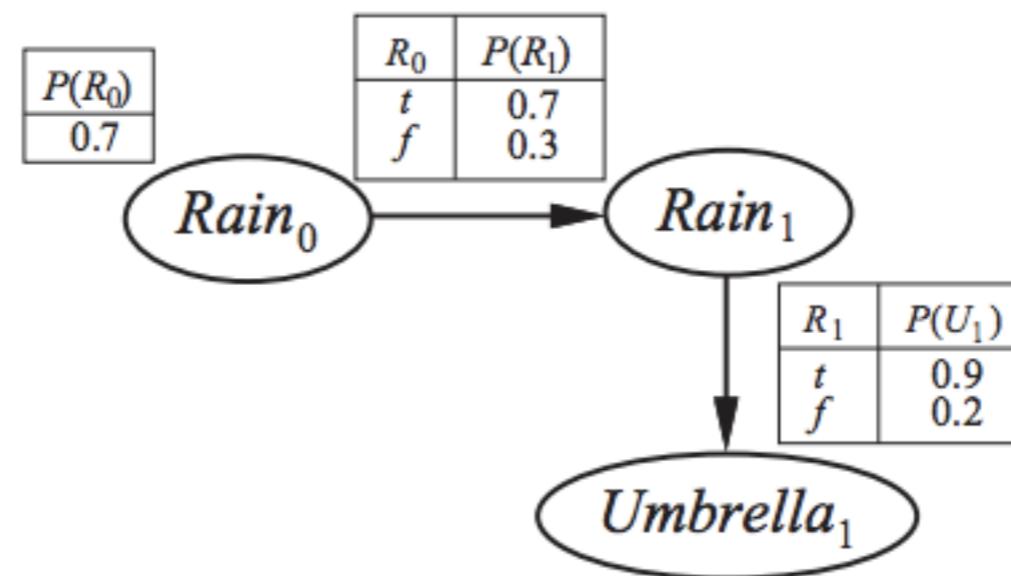
---

- A DBN is a BN describing a **temporal probability model** that can have **any number of state variables  $X_t$  and evidence variables  $E_t$**
- HMMs are DBNs with a **single state and a single evidence variable**
- But recall that one can combine a set of discrete (evidence or state) variables into a single variable (whose values are tuples).
- So every discrete-variable DBN can be described as a HMM.
- So why bother with DBNs?
- Because decomposing a complex system into constituent variables, as a DBN does, ameliorates **sparseness** in the temporal probability model

# Constructing DBNs

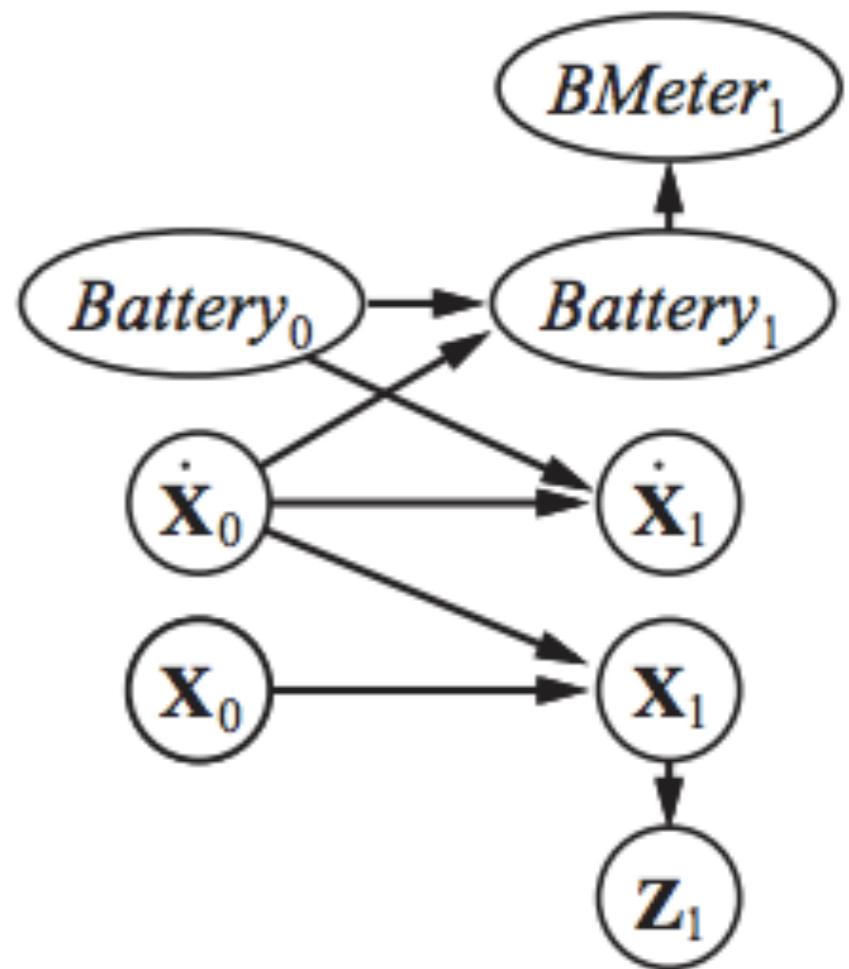
---

- We have to specify **prior distribution** of state variables  $\mathbf{P}(\mathbf{X}_0)$ ,
- **transition model**  $\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{X}_t)$ , and **sensor model**  $\mathbf{P}(\mathbf{E}_t|\mathbf{X}_t)$
- Also, we have to fix **topology** of nodes
- Stationarity assumption
- most convenient to specify topology for first slice
- Umbrella world example:



# The art of representing complex problems: An example

- Consider a **battery-driven robot** moving in the  $X \times Y$  plane
- Let  $\mathbf{X}_t = (X_t, Y_t)$  and  $\dot{\mathbf{X}}_t = (\dot{X}_t, \dot{Y}_t)$  state variables for position and velocity, and  $Z_t$  measurements of position (e.g. GPS)
- Add  $Battery_t$  for battery charge level and  $BMeter_t$  for the measurement of it
- We obtain the following basic model:



# Modeling Failure

---

- Assume  $Battery_t$  and  $BMeter_t$  take on discrete values (e.g. integer between 0 and 5)  
These variables should be identical (CPT=identity matrix) unless error creeps in
- One way to model error is through **Gaussian error model**, i.e. a small Gaussian error is added to the meter reading – We can approximate this also for the discrete case through an appropriate distribution
- **Problem:** This model will lead to very wrong estimation if **sensor failure rather than inaccurate measurements ...**

# Transient failure

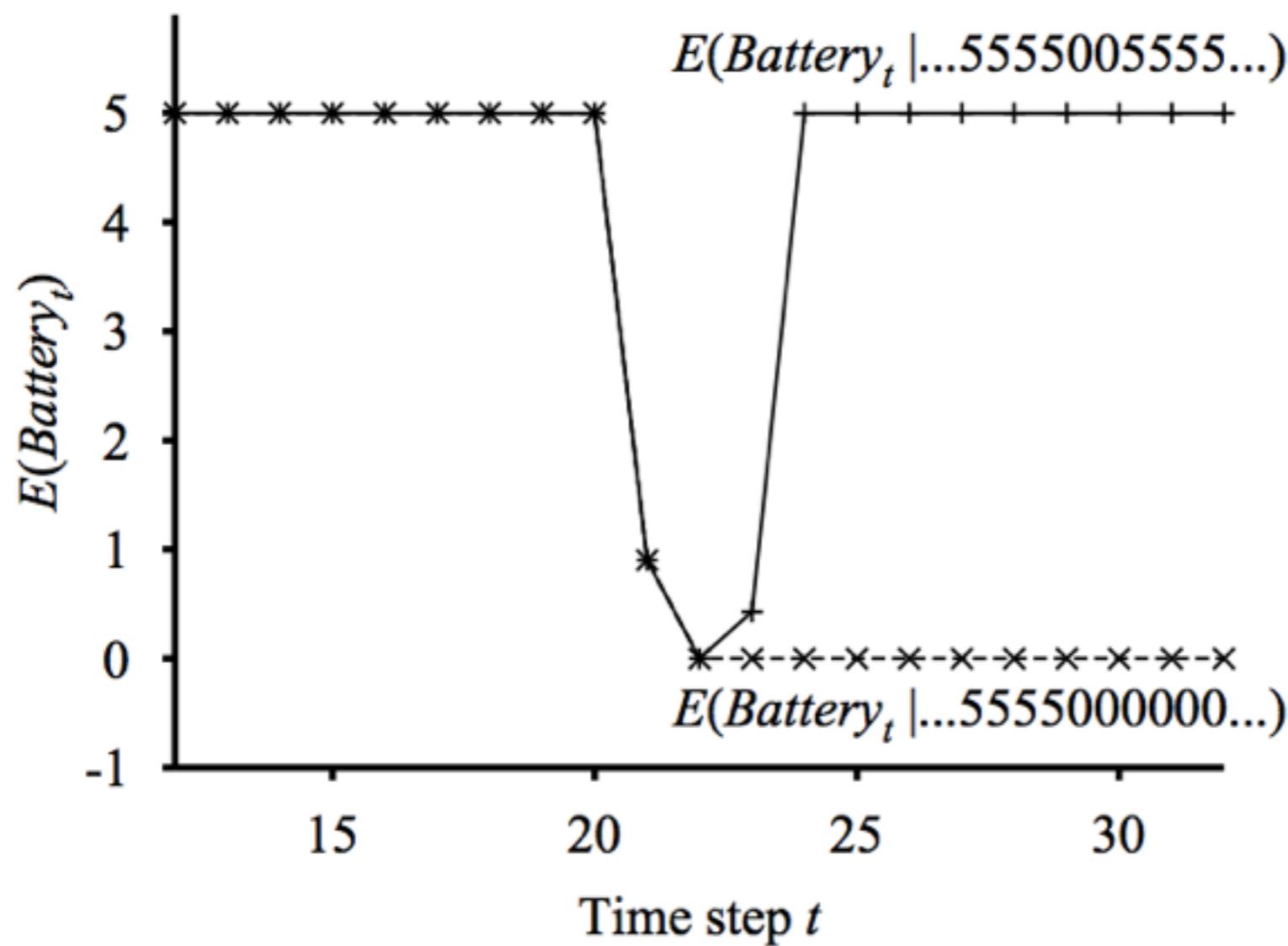
---

- **Transient failure:** sensor occasionally sends inaccurate data,
- Robot example: after 20 consecutive readings of 5 suddenly  $\text{BMeter}_{21} = 0$
- **How should this be interpreted?**
- In Gaussian error model, belief about  $\text{Battery}_{21}$  depends on:
  - Sensor model:  $P(\text{BMeter}_{21} = 0 | \text{Battery}_{21})$  and
  - Prediction model:  $P(\text{Battery}_{21} | \text{BMeter}_{1:20})$
- If probability of large sensor error is smaller than sudden transition to 0, then with high probability battery is considered empty
- A measurement of 0 at  $t = 22$  will make this (almost) certain
- After a reading of 5 at  $t = 23$  the probability of full battery will go back to high level
- which of course is impossible ... robot made completely wrong judgement . .

# Transient failure model

---

- Curves for prediction depending on whether BMeter<sub>t</sub> is only 0 for t = 22/23 or whether it stays 0 indefinitely



# Transient failure model

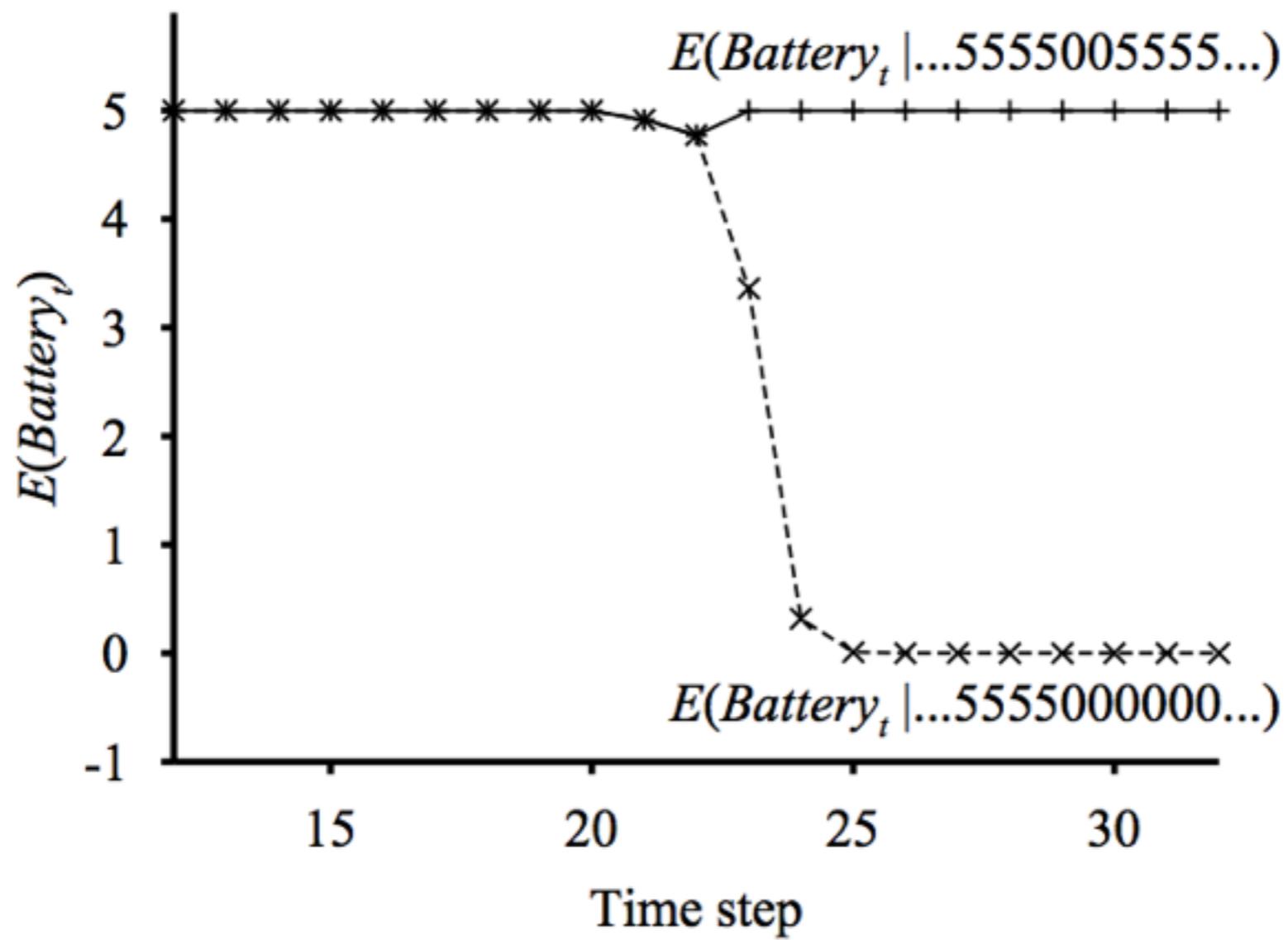
---

- To handle failure properly, sensor model must include possibility of failure
- Simplest failure model: assign small probability to incorrect values, e.g.  
 $P(B\text{Meter}_t = 0 | \text{Battery}_t = 5) = 0.03$
- When faced with 0 reading, provided that predicted probability of empty battery is much less than 0.03, best explanation is failure
- This model is much less susceptible to failure, because an explanation is available
- However, it cannot cope with persistent failure either

# Transient failure model

---

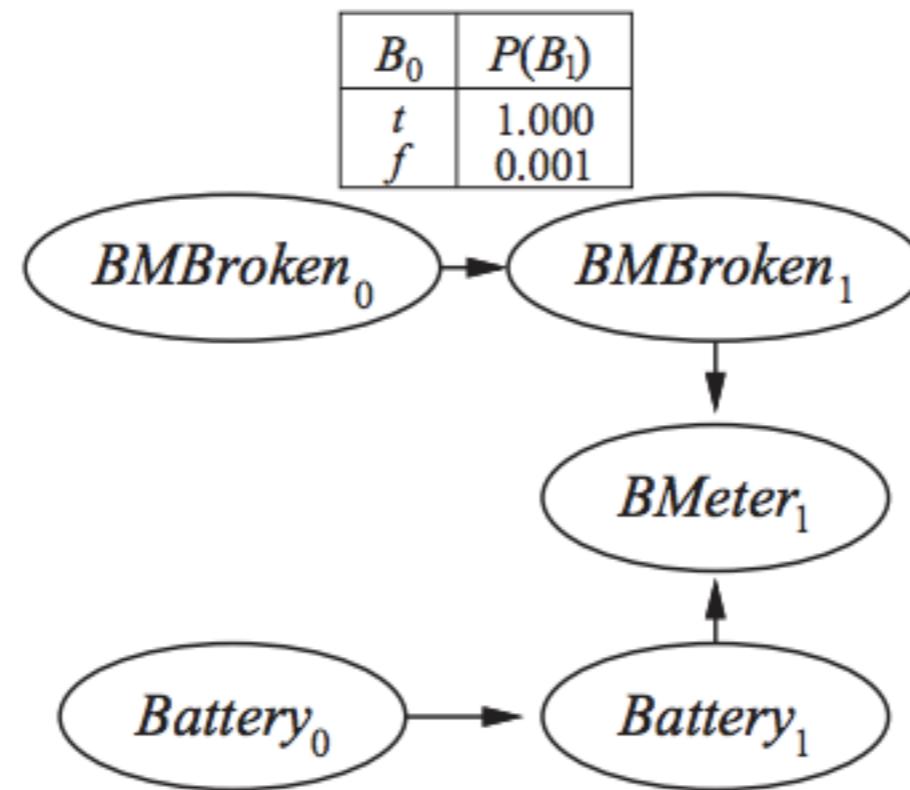
- Handles transient failure
- Problem: In case of permanent failure the robot will (wrongly) believe the battery is empty



# Persistent failure model

---

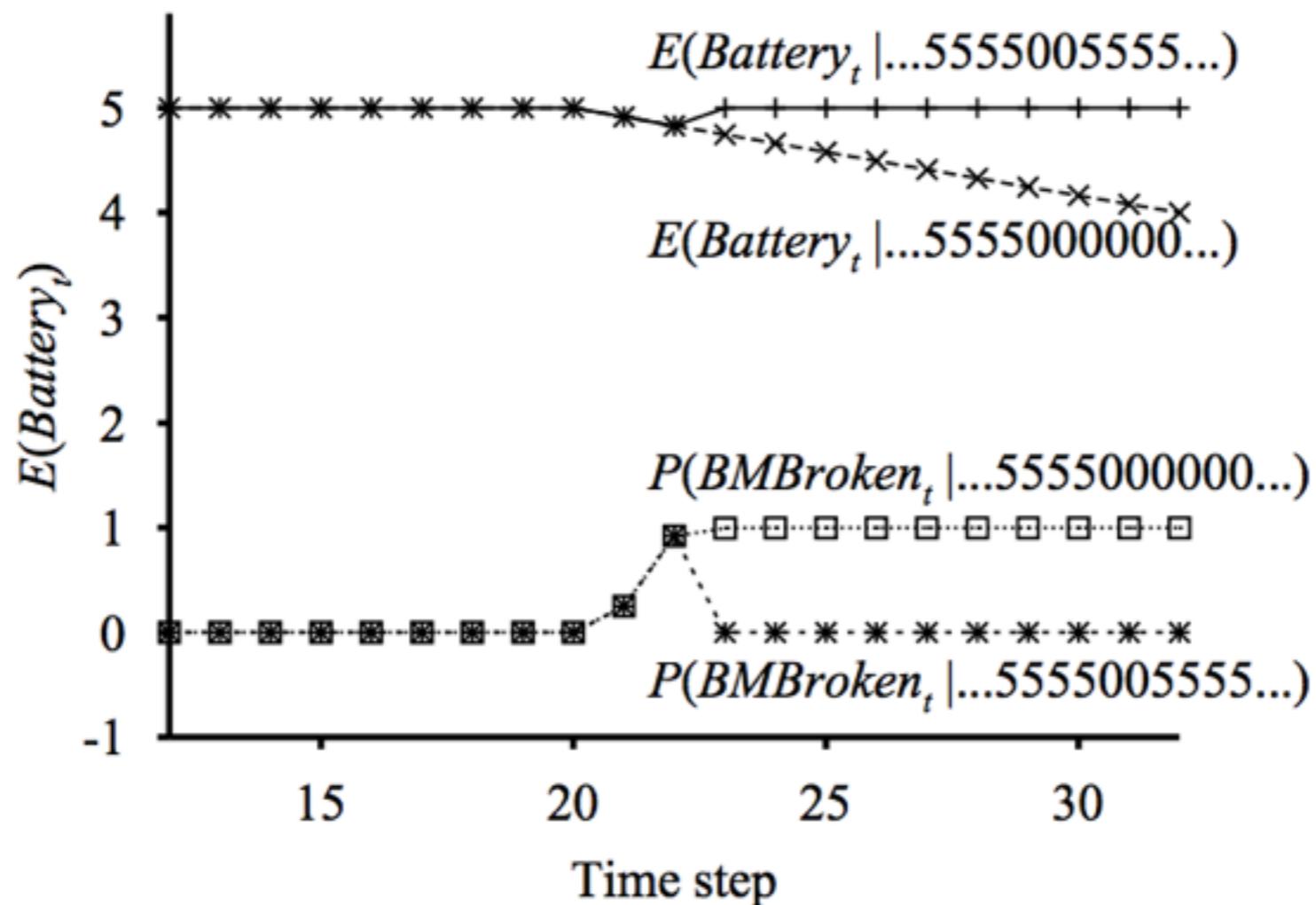
- Persistent failure models describe how sensor behaves under normal conditions and after failure
- Add additional variable BMBroken, and CPT to next BMBroken state has a very small probability if not broken, but 1.0 if broken before.
- When BMBroken is true, BMeter will be 0 regardless of Battery:



# Persistent failure model

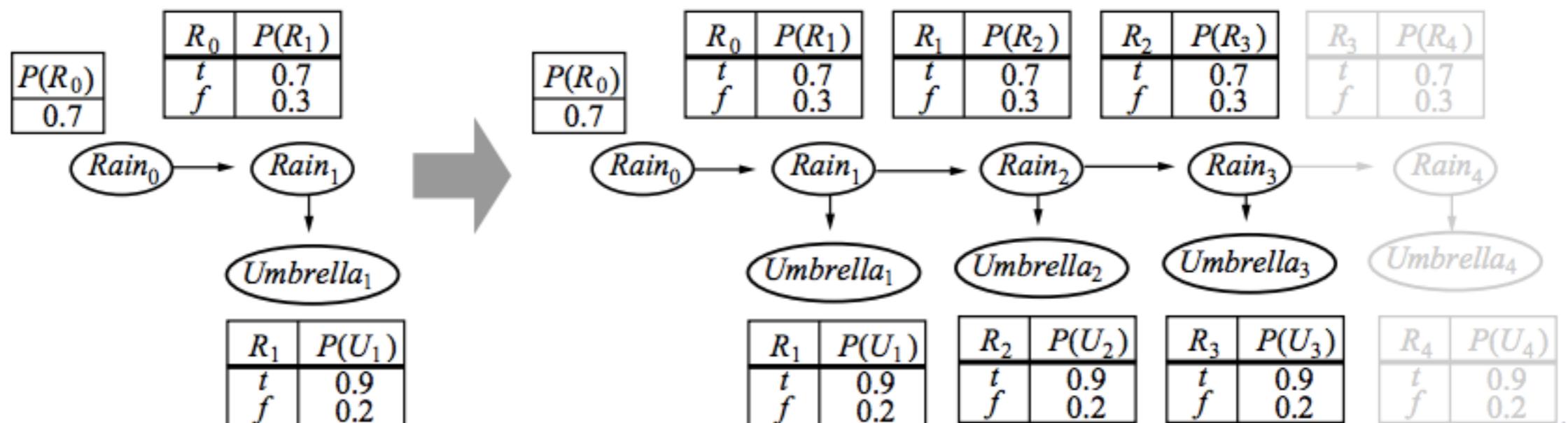
---

- In case of persistent failure, robot assumes discharge of battery at “normal” rate



# Exact Inference in DBNs

- Since DBNs are BNs, we already have inference algorithms like **variable elimination**
- Essentially DBN equivalent to infinite “unfolded” BN, but slices beyond required inference period are irrelevant
- Unrolling: reproducing basic time slice to accommodate observation sequence



# Exact Inference in DBNs

---

- Exact inference in DBNs is intractable, and this is a major problem.
- There are **approximate inference methods** that work well in practice, likelihood weighting, MCMC, particle filtering.
- This issue is currently a hot topic in AI. ..

# Summary

---

- Account of time and uncertainty complete
- HMMs and matrix operations.
- Kalman Filters
- DBNs as general case
- the art of representing complex processes, robot example: choosing a transition model and sensor models.
- Quite intractable / exact inference, but powerful
- Next time: Decision Making under Uncertainty



# **Inf2D-Reasoning and Agents**

## Spring 2017

Lecture (2)9:- Decision Making Under Uncertainty

Peggy Seriès, [pseries@inf.ed.ac.uk](mailto:pseries@inf.ed.ac.uk)

Based on previous slides by A. Lascarides

# Where are we?

---

- Last time . . . Looked at **Dynamic Bayesian Networks**
- General, powerful method for describing temporal probabilistic problems
- Unfortunately exact inference computationally too hard
- Methods for approximate inference (particle filtering)
- Today . . . **Decision Making under Uncertainty**  
**(Basics of Utility theory)**

# Combining beliefs and desires

---

- Rational agents do things that are an optimal tradeoff between:
  - the **likelihood** of reaching a particular resultant state (given one's actions) and
  - The **desirability** of that state
- So far we have done the ‘likelihood’ bit: we know how to evaluate the probability of being in a particular state at a particular time.
- But we’ve not looked at an agent’s **preferences or desires**
- Now we will discuss **utility theory** to obtain a full picture of **decision-theoretic agent design**

# Utility theory & Utility Functions

---

- Agent's preferences between world states are described using a **utility function**
- UF assigns some numerical value  $U(S)$  to each state  $S$  to express its **desirability** for the agent
- Actions determine states : Nondeterministic action  $a$  has results  $Result(a)$  and probabilities  $P(Result(a) = s'|a, e)$  summarise agent's knowledge about its effects given evidence observations  $e$ .
- **Expected utility of action** is the average utility value of the outcomes weighted by the probability that they occur:

$$EU(A|E) = \sum_{s'} P(Result(a) = s'|a, e) U(s')$$

# Utility theory & utility functions

---

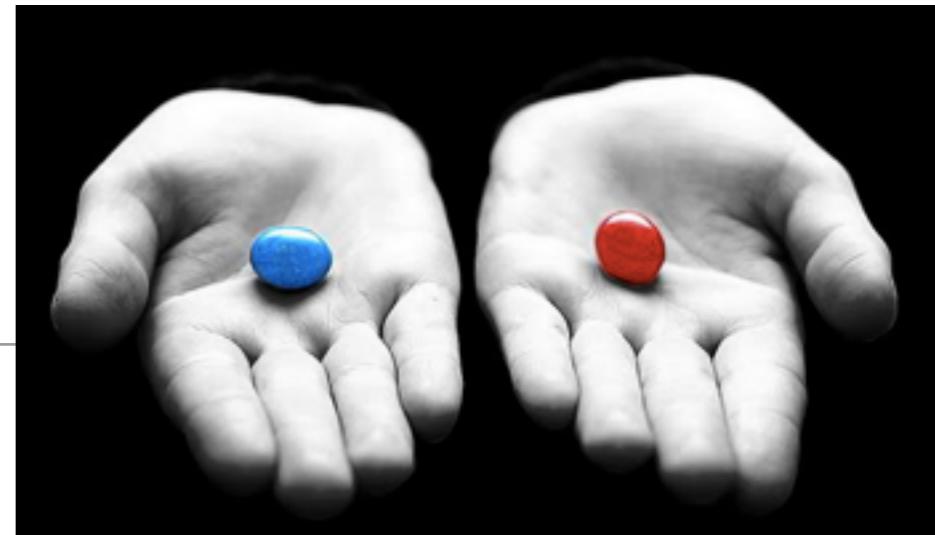
- Principle of maximum expected utility (MEU) says a **rational agent** should **use action that maximises expected utility**
- In a sense, this summarises the whole endeavour of AI:

*If agent maximises utility function that correctly reflect the performance measure applied to it, then optimal performance will be achieved by averaging over all environments in which agent could be placed*

- Of course, this doesn't tell us how to define utility function or how to determine probabilities for any sequence of actions in a complex environment (in reality this is hard to learn).
- For now we will only look at **one-shot decisions**, not sequential decisions (next lecture)

# Constraints on rational preferences

---



- MEU sounds reasonable, but why should this be the best quantity to maximise? Why are numerical utilities sensible? Why single number?
- Can be answered by looking at **constraints on preferences that rational agent should have** - then showing that the MEU can be derived from those constraints.
- Notation:
  - A > B A is preferred to B
  - A ~ B the agent is indifferent between A and B
  - A >~ the agent prefers A to B or is indifferent between them
- What are A and B? **Lotteries: set of outcomes for each action.** Possible outcomes  $C_1 \dots C_n$  and accompanying probabilities  
 $L=[p_1, C_1; p_2, C_2; \dots; p_n, C_n]$
- The outcome of a lottery is either a state or another lottery

# Constraints on rational preferences

---



- The following are considered reasonable **axioms of utility theory**
- **Orderability:** Given 2 lotteries, one needs to decide preference:  
$$(A > B) \vee (B > A) \vee (A \sim B)$$
- **Transitivity:** If agent prefers A over B and B over C then he must prefer A over C: 
$$(A > B) \wedge (B > C) \Rightarrow (A > C)$$
- Example: Assume  $A > B > C > A$  and A, B, C are goods
  - Agent might trade A and some money for C if he has A
  - We then offer B for C and some cash and then trade A for B
  - Agent would lose all his money over time

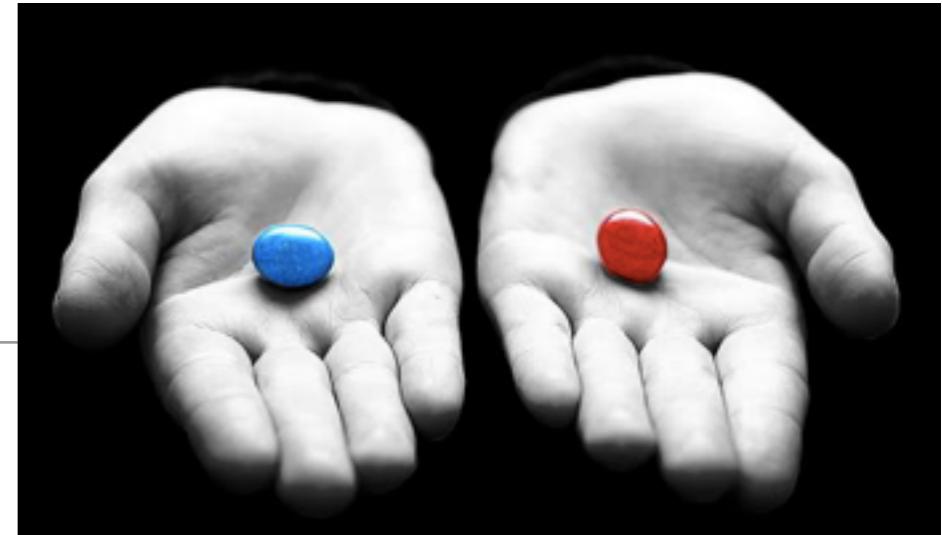
# Constraints on rational preferences

---

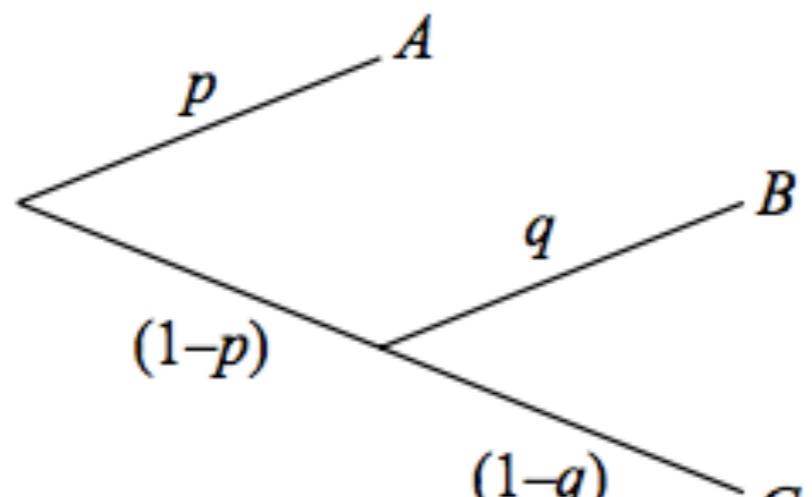


- **Continuity:** If B is between A and C in preference, then with some probability p, agent will be indifferent between getting B for sure and a lottery over A and C  
$$A > B > C \Rightarrow \exists p [p, A; 1 - p, C] \sim B$$
- **Substitutability:** Indifference between lotteries leads to indifference between complex lotteries built from them  
$$A \sim B \Rightarrow [p, A; 1 - p, C] \sim [p, B; 1 - p, C]$$
- **Monotonicity:** Preferring A to B implies preference for any lottery that assigns higher probability to A  
$$A > B \Rightarrow (p \geq q \Leftrightarrow [p, A; 1 - p, B] \geq [q, A; 1 - q, B])$$

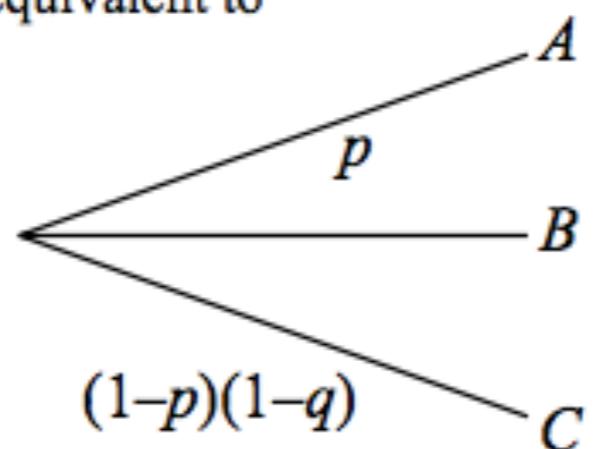
# Decomposability Example



- **Decomposability:** Compound lotteries can be reduced to simpler one  
 $[p, A; 1 - p, [q, B; 1 - q, C]] \sim [p, A; (1 - p)q, B; (1 - p)(1 - q), C]$



is equivalent to



# From Preferences to Utility

---

- From the above axioms on preference, we can derive the following consequences (**axioms of utility**):
- **Utility principle:** there exists a function such that :  
 $U(A) > U(B) \Leftrightarrow A > B$   
 $U(A) = U(B) \Leftrightarrow A \sim B$
- **Maximum Expected Utility principle:** utility of lottery is sum of probability of outcomes times their utilities

$$U([p_1, S_1; \dots; p_n, S_n]) = \sum_i p_i U(S_i)$$

- But an agent might use this but not know explicitly his own utilities!
- But you can work out his (or even your own!) utilities by observing his (your) behaviour and assuming that he (you) chooses to MEU

# From Preferences to Utility

---

- According to the above axioms, any arbitrary preferences can be expressed by utility functions
- I prefer to have a prime number of £ in my bank account; when I have £10 I will give away £3.
- But fortunately usually preferences are more systematic, a typical example being money (roughly, we like to maximise our money)
- Agents exhibit **monotonic preference** toward money, but how about lotteries involving money?

# From Preferences to Utility

---

- You've just won 1 Million dollars !!



- keep 1 million or risk it with the prospect of getting 3 millions at the toss of a (fair) coin ??



# Utility of Money

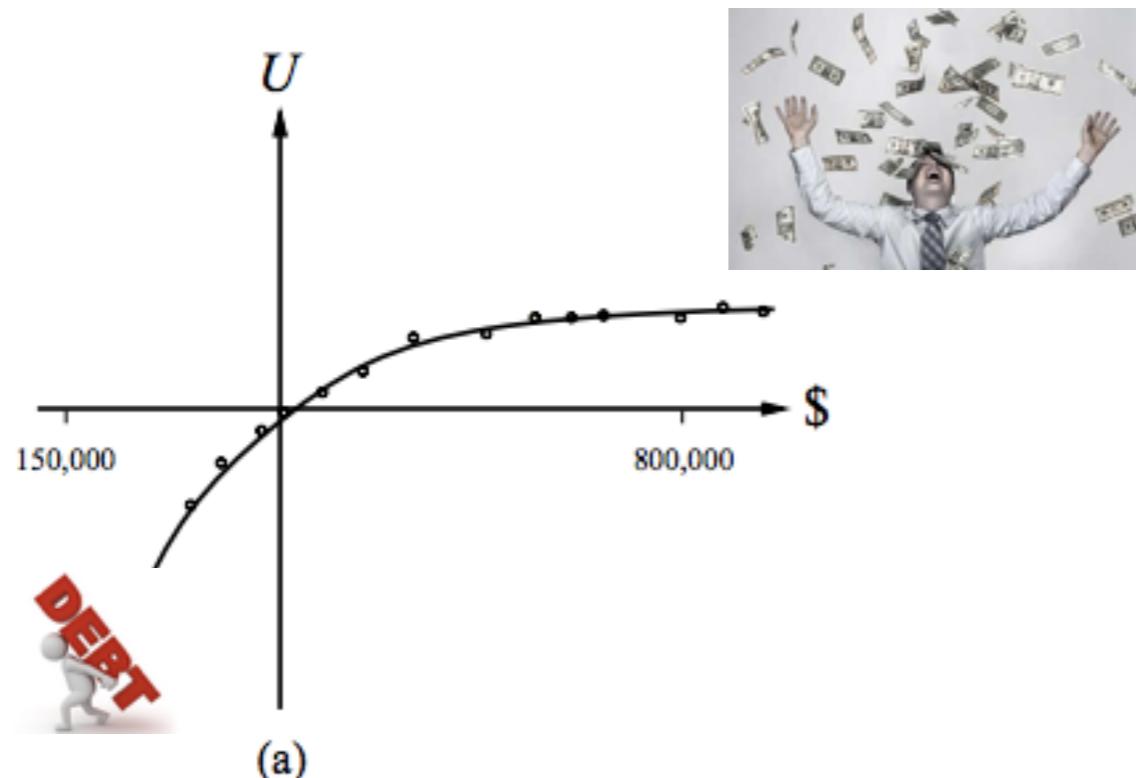
---

- The **Expected Monetary Value** of accepting gamble is  $0.5 \times 0 + 0.5 \times 3,000,000$  which is greater than 1,000,000
- but ... that doesn't mean this is going to be the preferred decision.  
**Utility is not always proportional to the EMV.**
- Use  $S_n$  to denote state of possessing wealth “n dollars”, current wealth  $S_k$
- Expected utilities become:  
$$EU(\text{Accept}) = 0.5 * U(S_{k+0}) + 0.5 * U(S_{k+3,000,000})$$
$$EU(\text{Decline}) = U(S_{k+1,000,000})$$
- But it all depends on utility values you assign to levels of monetary wealth  
(are you already a billionaire? is the second million worth as much as the first?)

# Utility of Money (empirical study)

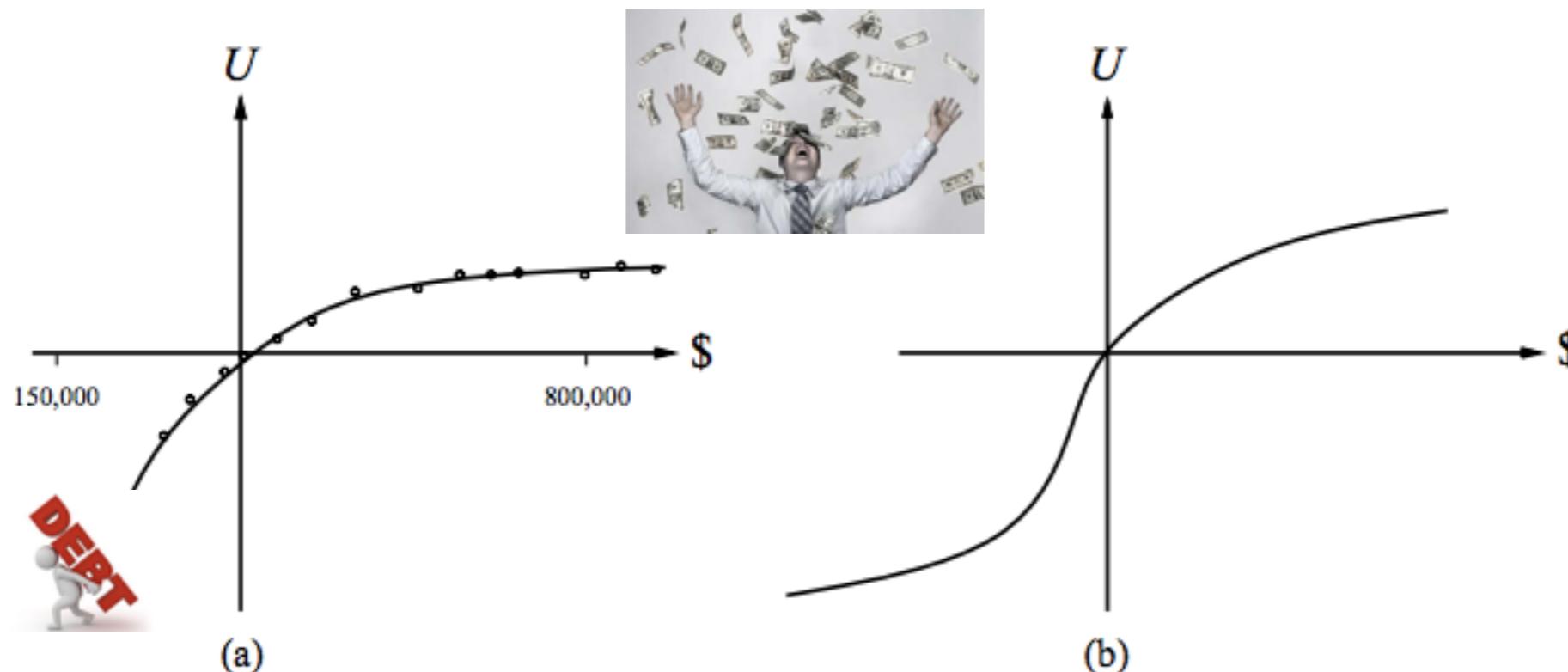
---

- It turns out that for most people utility of money is usually concave, showing that going into debt is considered disastrous relative to small gains in money—**risk averse**.



# Utility of Money (empirical study)

- It turns out that for most people utility of money is usually concave, showing that going into debt is considered disastrous relative to small gains in money—**risk averse**.



- But if you're already \$10M in debt, losing more millions doesn't make much difference—**risk seeking** when desperate!

# Utility Scales

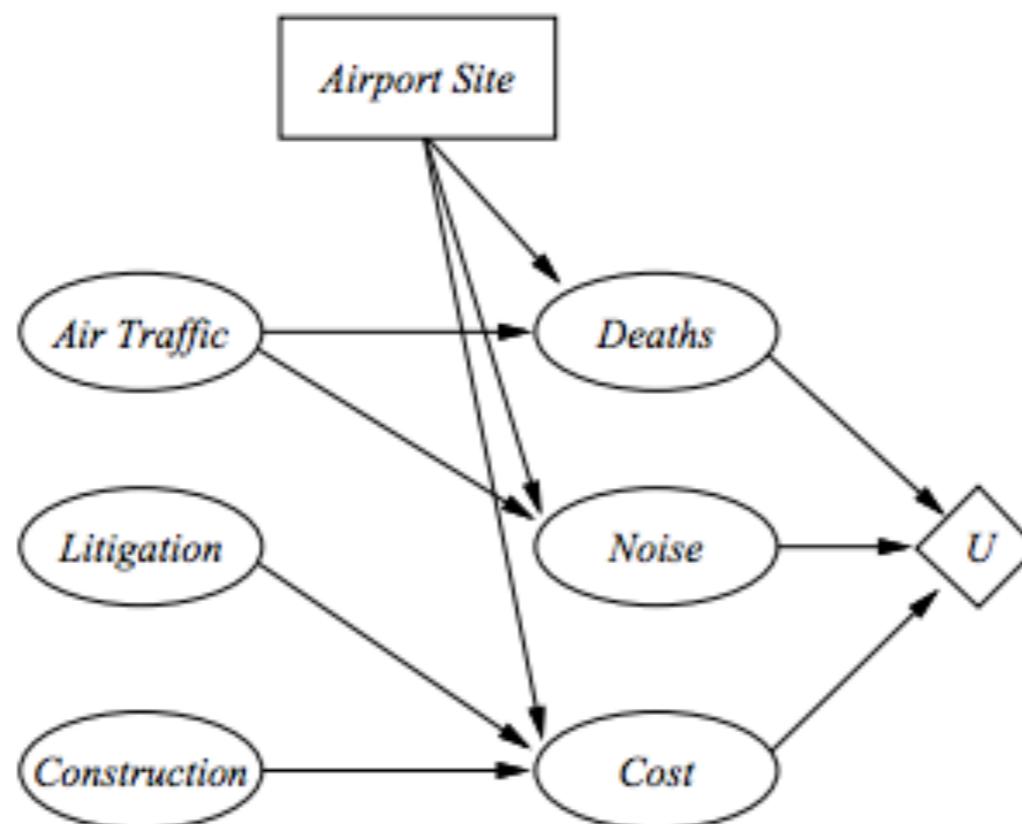
---

- Axioms don't say anything about **scales**
- For example transformation of  $U(S)$  into  $U'(S) = k_1 + k_2 U(S)$  ( $k_2$  positive) doesn't affect behaviour
- In deterministic contexts behaviour is unchanged by any monotonic transformation
- One procedure for assessing utilities is to use **normalised utility** between “**best possible prize**” ( $u^T = 1$ ) and “**worst possible catastrophe**” ( $u_\perp = 0$ )
- Ask agent to indicate preference between  $S$  and the standard lottery  $[p, u^T : (1 - p), u_\perp]$ , adjust  $p$  until agent is indifferent between  $S$  and standard lottery, set  $U(S) = p$

# Decision Networks

---

- What we now need is a way of integrating utilities into our view of probabilistic reasoning
- **Decision networks** (a.k.a **influence diagrams**) combine BNs with additional node types for actions and utilities
- Illustrate with airport siting problem - where to place the airport?



# Representing decision problems with DNs

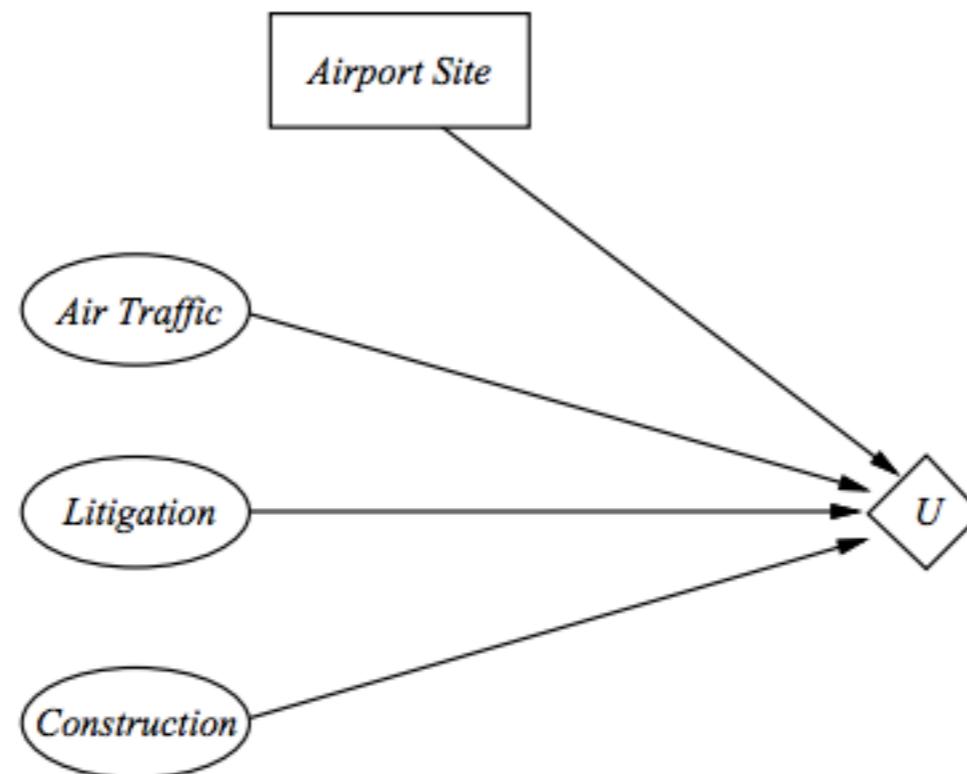
---

- **Chance nodes** (ovals) represent random variables with CPTs, parents can be decision nodes
- **Decision nodes** (rectangles) represent decision-making points at which actions are available
- **Utility nodes** (diamond) represent utility function connected to all nodes that affect utility directly
- Often nodes describing outcome states are omitted and *expected utility* associated with actions is expressed (rather than states) – **action-utility tables**

# Representing decision problems with DNs

---

- Simplified version with action-utility tables (Q-values)
- Less flexible but simpler (like pre-compiled version of general case)



# Evaluating decision networks

---

- Evaluation of a DN works by **setting decision node to every possible value**
- “Algorithm”:
  1. Set evidence variables for current state
  2. For each value of decision node:
    - 2.1 Set decision node to that value
    - 2.2 Calculate **posterior probabilities for parents of utility node**
    - 2.3 Calculate resulting (expected) utility for action
  3. **Return action with highest (expected) utility**
- Using any algorithm for BN inference, this yields a simple framework for building agents that make single-shot decisions

# Summary

---

- Probability theory describes what an agent should believe based on the basis of evidence; Utility theory describes what an agent wants
- Decision theory puts the two together to describe what an agent should do. → rational agent.
- An agent whose preferences are consistent with axioms of utility theory can be described as possessing a utility function. The agent selects actions as if maximising expected utility.
- Decision networks natural extensions of Bayesian Networks framework
- Only looked at one-shot decisions so far ... Next time: **Markov Decision Processes**



# **Inf2D-Reasoning and Agents**

## Spring 2017

Lecture (3)0:- Markov decision processes

Peggy Seriès, [pseries@inf.ed.ac.uk](mailto:pseries@inf.ed.ac.uk)

Based on previous slides by A. Lascarides

# Where are we?

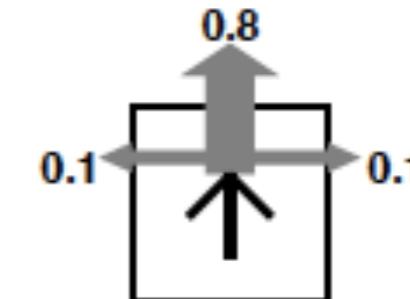
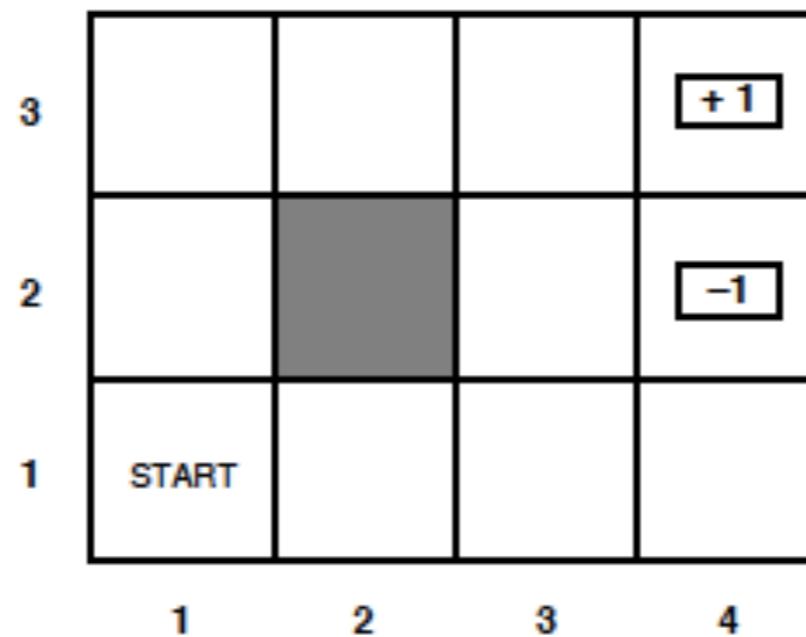
---

- Talked about decision making under uncertainty
- Discussed axioms of utility theory
- Described different utility functions
- Introduced decision networks
- Today . . . **Markov Decision Processes**

# Sequential decision problems

---

- So far we have only looked at one-shot decisions, but decision process are often **sequential**
- Example scenario: a 4x3-grid in which agent moves around (fully observable) and obtains utility of +1 or -1 in terminal states



- Actions are somewhat unreliable (in deterministic world, solution would be trivial)

# Markov decision processes

---

- To describe such worlds, we can use a **(transition) model**  $T(s, a, s')$  denoting the probability that action  $a$  in  $s$  will lead to state  $s'$
- Model is **Markovian**: probability of reaching  $s'$  depends only on  $s$  and not on history of earlier states
- Think of  $T$  as big three-dimensional table (actually a DBN)
- Utility function now depends on **environment history**:
  - Agent receives a **reward  $R(s)$**  in each state  $s$  (e.g. -0.04 apart from terminal states in our example)
  - (for now) **utility of environment history is the sum of state rewards**

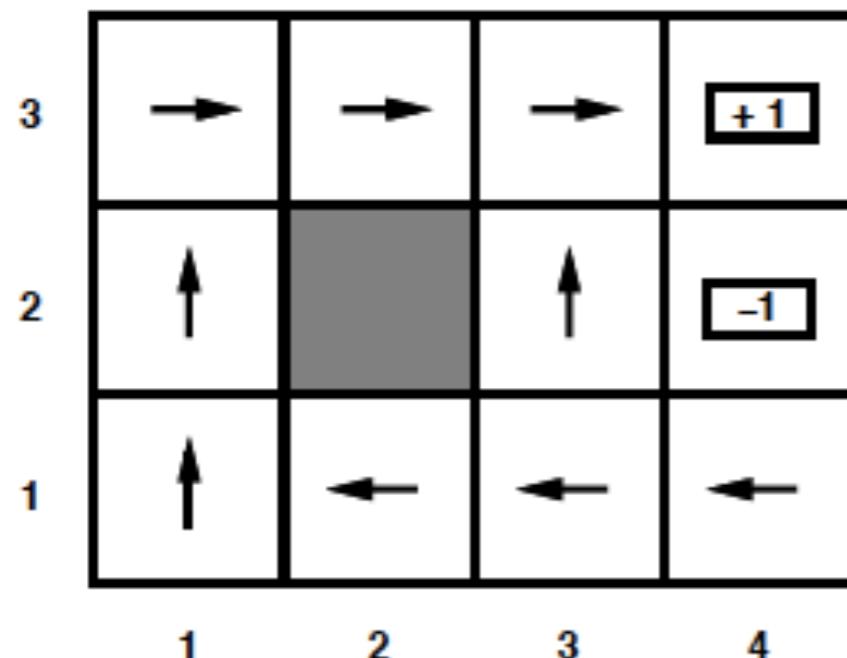
# Markov decision processes

---

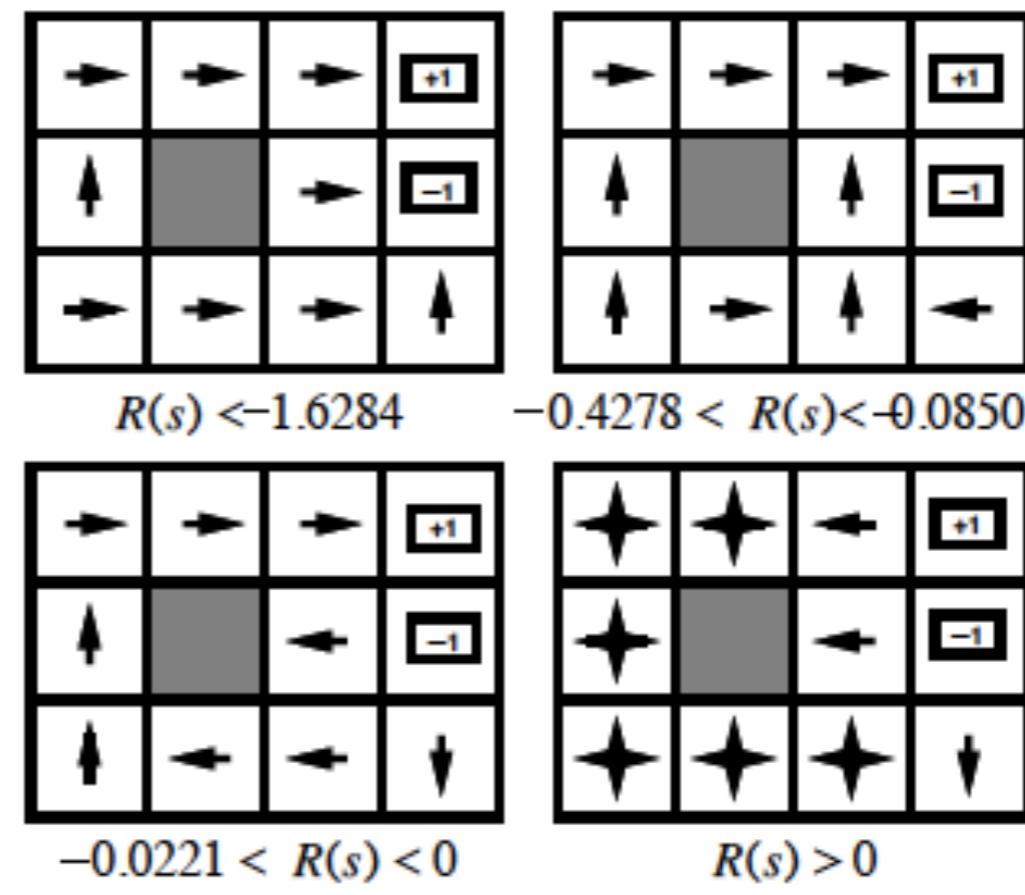
- Definition of a Markov Decision Process (MDP):
  - Initial state:  $S_0$
  - Transition model:  $P(s'|a, s)$
  - Utility function: sum of  $R(s)$
- Solution should describe what agent does in every state. This is called **policy**, written as  $\Pi(s)$ .  $\Pi(s)$  for an individual state describes **which action should be taken** in  $s$
- **Optimal policy** is one that yields the highest expected utility (denoted by  $\Pi^*$ )

## Example: optimal policy: balancing risks and rewards

- Optimal policies in the 4x3-grid environment:
  - (a) With cost of -0.04 per intermediate state,  $\Pi^*$  is conservative for (3,1)
  - (b) Different cost induces direct run to terminal state/shortcut at (3,1)/no risk/avoid both exits



(a)



(b)

# Optimality in sequential decision problems

---

- We have used **sum of rewards** as utility of environment history until now, but what are the alternatives?
- First question: **finite horizon or infinite horizon?**
- Finite means there is a fixed time  $N$  after which nothing matters (game's over)  
$$\forall k \quad U_h([s_0, s_1, \dots, s_{N+k}]) = U_h([s_0, s_1, \dots, s_N])$$
- This leads to **non-stationary optimal policies** ( $t$  matters)

# Optimality in sequential decision problems

---

- With **infinite** horizon, we get **stationary** optimal policies (time at state doesn't matter) → We are mainly going to use infinite horizon utility functions
- NOTE: sequences to terminal states can be finite even under infinite horizon utility calculation
- Second issue: how to calculate **utility of sequences**
- **Stationarity** here is reasonable assumption: if you prefer one future to another starting tomorrow, then you should still prefer that future if it were to start today instead.  
if  $(s_0 = s'_0)$  and  $[s_0, s_1, s_2 \dots] > [s'_0, s'_1, s'_2, \dots]$  ⇒  $[s_1, s_2 \dots] > [s'_1, s'_2, \dots]$

# Utilities of sequences of states

---

- There are only two ways to assign utilities to sequences under stationarity assumptions:

- **Additive rewards:**

$$U_h([s_0, s_1, s_2 \dots]) = R(s_0) + R(s_1) + R(s_2) + \dots$$

- **Discounted rewards (for discount factor  $0 \leq \gamma \leq 1$ )**

$$U_h([s_0, s_1, s_2 \dots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \dots$$

- Discount factor makes more distant future rewards less significant
- We will mostly use discounted rewards in what follows

# Utilities of sequences of states

---

- Choosing infinite horizon rewards creates a problem:  
Some sequences will be infinite with infinite (additive) reward, how do we compare them?
- Solution 1 (preferred): with **discounted rewards** the utility of an infinite sequence is bounded

$$U_h([s_0, s_1, s_2 \dots]) = \sum_{t=0}^{\infty} \gamma^t R(s_t) \leq \sum_{t=0}^{\infty} \gamma^t R_{\max} = R_{\max}/(1 - \gamma)$$

- Solution 2: under “**proper policies**”, i.e. if agent will eventually visit terminal state, additive rewards are finite
- Solution 3: compare infinite sequences using **average reward** per time step

# What is the optimal policy? Value Iteration

---

- Value iteration is an algorithm for **calculating optimal policy** in MDPs  
*Calculate the utility of each state and then select optimal action based on these utilities*
- Since discounted rewards seemed to create no problems, we will use

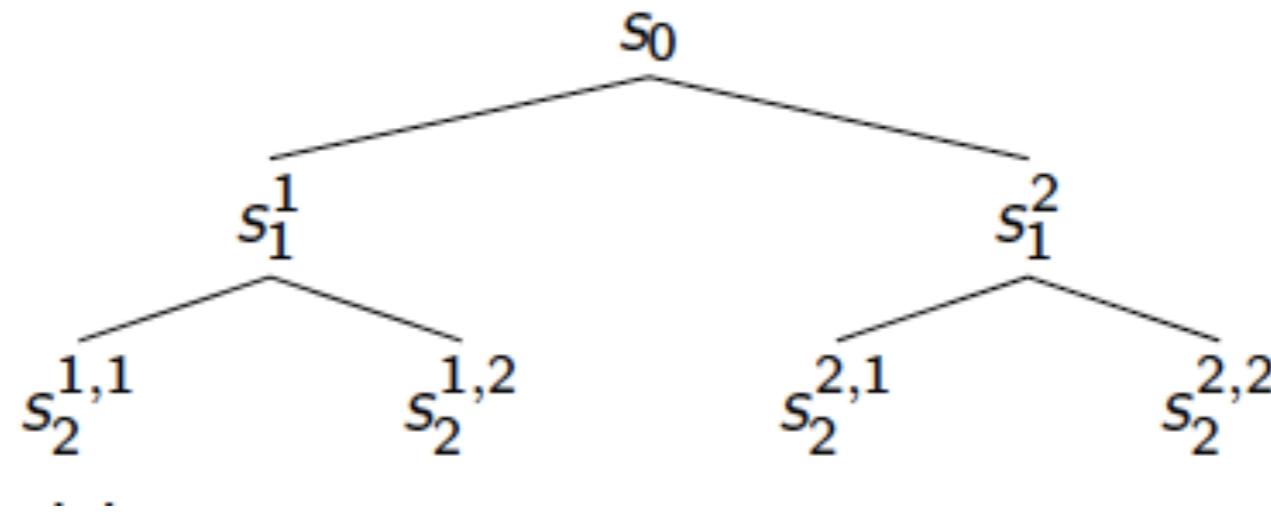
$$\pi^* = \arg \max_{\pi} E \left[ \sum_{t=0}^{\infty} \gamma^t R(s_t) | \pi \right]$$

as a criterion for **optimal policy**. The expectation is with respect to the probability distribution over state sequences determined by  $s$  and  $\Pi$ .

Explaining

$$\pi^* = \arg \max_{\pi} E \left[ \sum_{t=0}^{\infty} \gamma^t R(s_t) | \pi \right]$$

- Each policy  $\Pi$  yields a tree, with root node  $s_0$ , and daughters to a node  $s$  are the possible successor states given the action  $\Pi(s)$ .
- $P(s'|a, s)$  gives the probability of traversing an arc from  $s$  to daughter  $s'$ .



- $E$  is computed by:
  - (a) For each path  $p$  in the tree, getting the product of the (joint) probability of the path in this tree with its discounted reward, and then
  - (b) Summing over all the products from (a)
- So this is just a generalisation of single shot decision theory.

# Utility of States : $U(s) \neq R(s)!$

---

- $R(s)$  is reward for being in  $s$  now.
- By making  $U(s)$  the utility of the states that might follow it,  $U(s)$  captures long-term advantages from being in  $s$   
 $U(s)$  reflects what you can do from  $s$ ;  
 $R(s)$  does not.
- States that follow depend on  $\Pi(s)$ . So utility of  $s$  given  $\Pi(s)$  is:

$$U^\pi(s) = E \left[ \sum_{t=0}^{\infty} \gamma^t R(s_t) | \pi, s_0 = s \right]$$

- With this, “true” utility  $U(s)$  is  $U^{\pi^*}(s)$  (expected sum of discounted rewards if executing optimal policy)

# Utilities in our example

---

- $U(s)$  computed for our example from algorithms to come.
- $\gamma = 1, R(s) = -0.04$  for nonterminals.

3	0.812	0.868	0.918	+ 1
2	0.762		0.660	-1
1	0.705	0.655	0.611	0.388

# Utilities of States

---

- Given  $U(s)$ , we can easily determine **optimal policy** at each  $s$ :

$$\pi^*(s) = \operatorname{argmax}_{a \in A(s)} \sum_{s'} P(s' | s, a) U(s')$$

- Direct relationship between:
  - utility of a state and that of its neighbours:  
Utility of a state is immediate reward plus expected utility of subsequent states if agent chooses optimal action
- This can be written as the famous **Bellman equations**:

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' | s, a) U(s')$$

# The value iteration algorithm: calculating optimal policy

---

- idea: calculate the utility of each state and then use the state utilities to select an optimal action in each state
- For  $n$  states we have  $n$  Bellman equations with  $n$  unknowns (utilities of states)
- **Value iteration** is an iterative approach to solving the  $n$  equations.

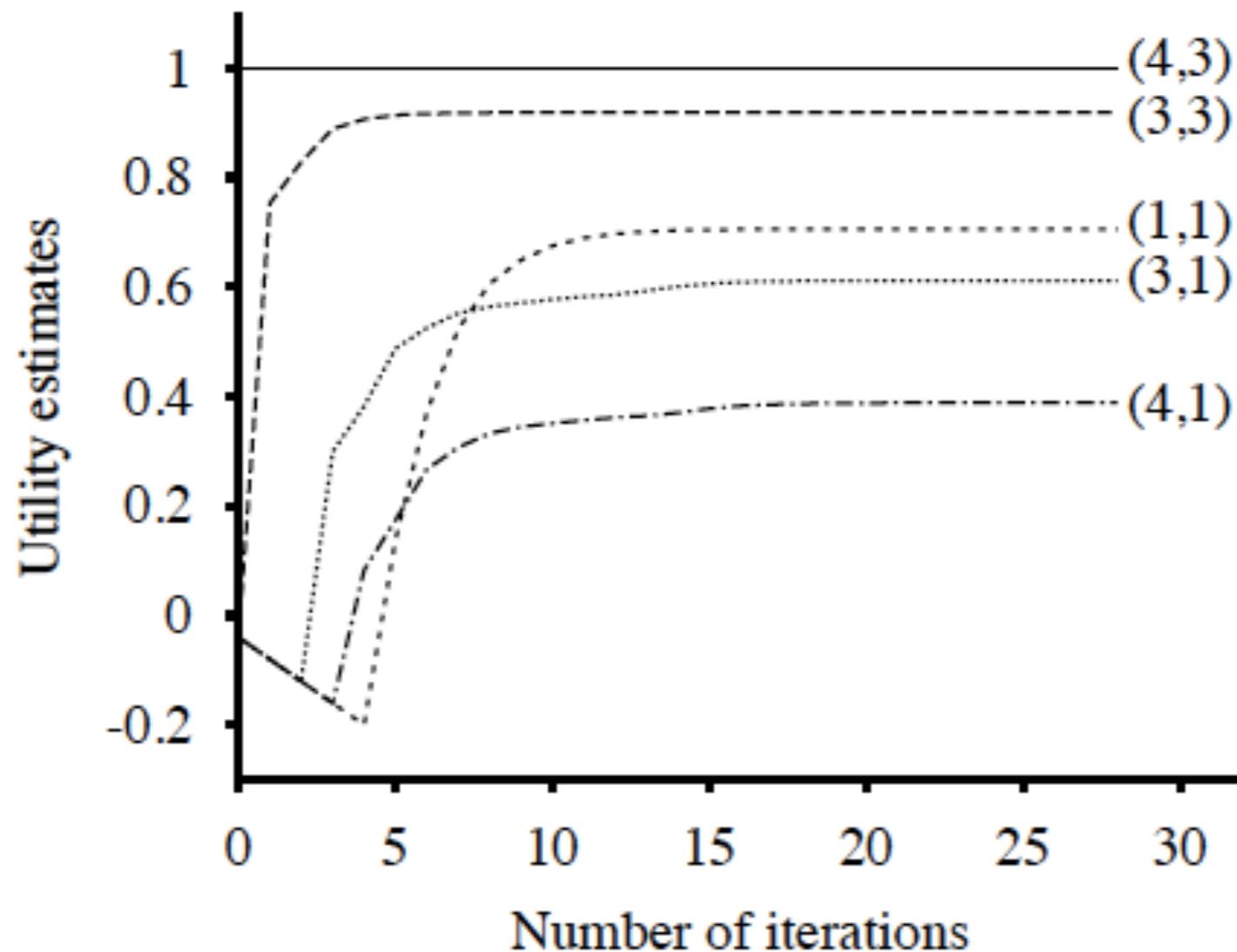
$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' | s, a) U_i(s') ,$$

- The algorithm converges to right and unique solution
- Like propagating values through network or utilities, by means of local updates.

# The value iteration algorithm

---

- Value iteration in our example: evolution of utility values of states



# Decision-theoretic agents

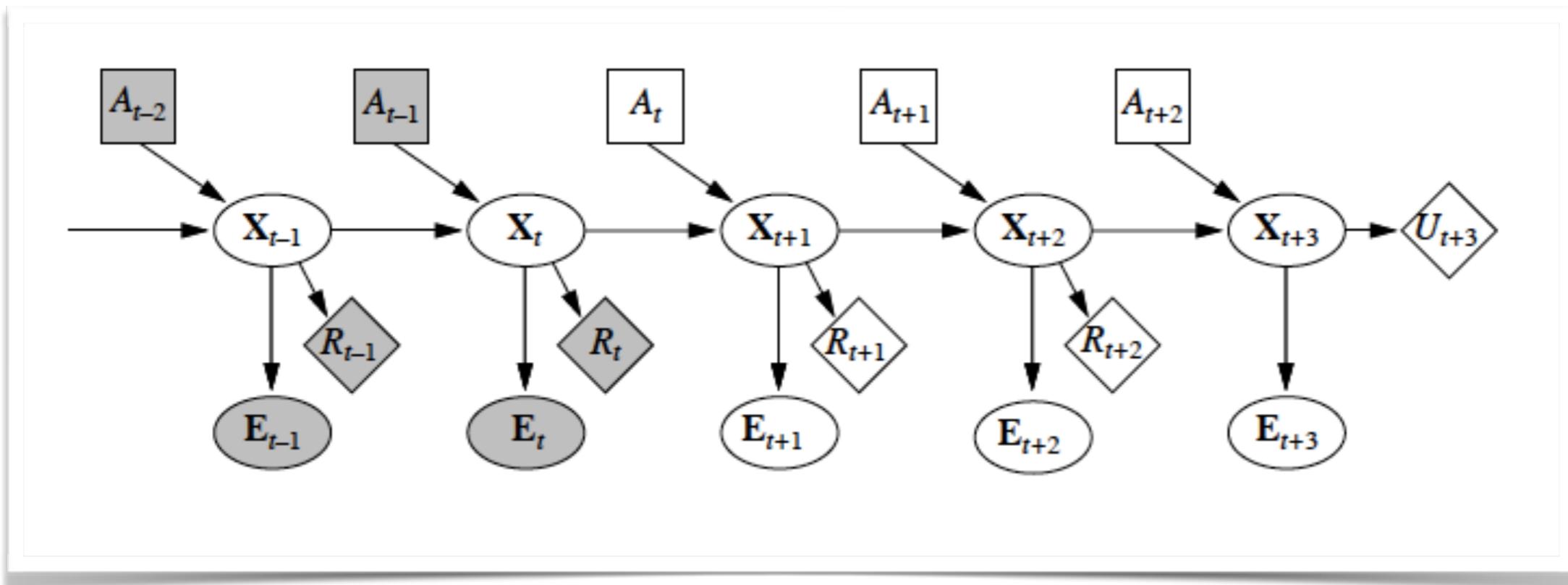
---

- We now have (tediously) gathered all the ingredients to build decision-theoretic agents
- POMDPs: Partially observable MDPs.
- Transition and observation models will be described by a DBN
- They will be augmented by decision and utility nodes to obtain a **dynamic DN**
- Decisions will be made by projecting forward possible action sequences and choosing the best one
- Practical design for a **utility-based agent**

# Decision-theoretic agents

---

- Dynamic decision networks look something like this
- General form of everything we have talked about in uncertainty part



# Summary

---

- Sequential decision problems in uncertain environments, also called **Markov decision processes**, are defined by a **transition model** specifying the probabilistic outcomes of actions and a **reward function** specifying the reward in each state.
- The **utility of a state sequence** is the sum of all the rewards over the sequence, possibly discounted over time. The solution of an MDP is a **policy** that associates a decision with every state that the agent might reach. An **optimal policy** maximizes the utility of the state sequences encountered when it is executed.
- The utility of a state is the expected utility of the state sequences encountered when an optimal policy is executed, starting in that state. The **value iteration algorithm** for solving MDPs works by iteratively solving the equations relating the utility of each state to those of its neighbours.

# Summary

---

- A **decision theoretical agent** can be constructed for **POMDP** environments.
- The agent uses a **dynamic decision network** to represent the transition and sensor models, to update its belief state and to project forward possible action sequences.
- multiple agents: **game theory**.
- Learning optimal policy from observed rewards: **reinforcement learning**. (Agent doesn't know the transition model and reward function).

Thank you ! :)

---

