CSCE 580: Introduction to AI

Week 7 - Lectures 13 and 14: Symbolic - Representation and Logic; Search

PROF. BIPLAV SRIVASTAVA, AI INSTITUTE

30TH SEP AND 2ND OCT 2025

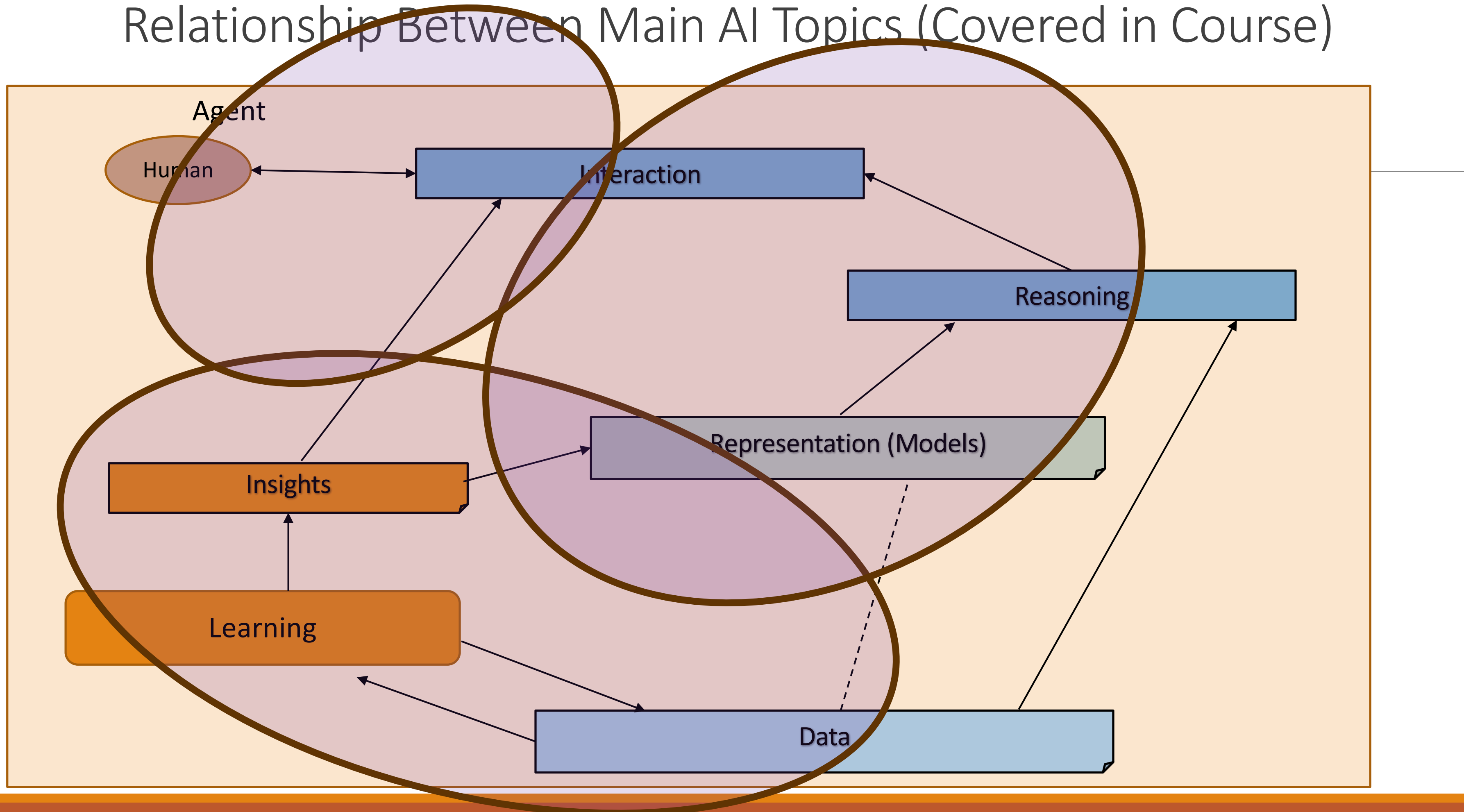
Carolinian Creed: "I will practice personal and academic integrity."

Credits: **Copyrights of all material reused acknowledged**

Organization of Week 7 - Lectures 13, 14

- Introduction Section
  - Recap from Week 5 (Lectures 9 and 10)
  - AI news
- Main Section
  - L13: Logic and Inference - First Order
  - L14: Search, Search - Uninformed
- Concluding Section
  - About next week – W8: Lectures 15, 16
  - Ask me anything

Relationship Between Main AI Topics (Covered in Course)



Recap of Week 6

We talked about

- AI/ ML Trust
  - Explainability
  - Trust ratings
- Representation and Logic
  - Propositional

- Week 1: Introduction, Aim: Chatbot / Intelligence Agent
- Weeks 2-3: Data: Formats, Representation and the Trust Problem
- Week 3: Machine Learning – Supervised (Classification)
- Week 4: Machine Learning - Unsupervised (Clustering) –
- Topic 5: Learning neural network, deep learning, Adversarial attacks
- Week 6: Large Language Models – Representation and Usage issues
- Weeks 7-8: Search, Heuristics - Decision Making
- Week 9: Constraints, Optimization – Decision Making
- Topic 10: Markov Decision Processes, Hidden Markov models - Decision making
- Topic 11-12: Planning, Reinforcement Learning – Sequential decision making
- Week 13: Trustworthy Decision Making: Explanation, AI testing
- Week 14: AI for Real World: Tools, Emerging Standards and Laws; Safe AI/ Chatbots

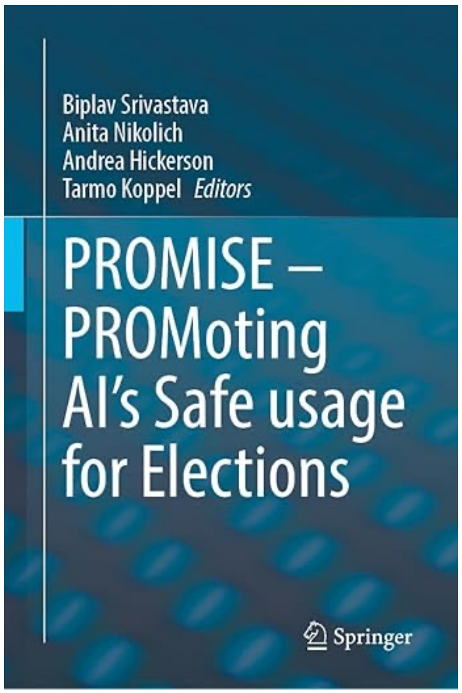
Upcoming Evaluation Milestones

- Projects B: Sep 30 – Nov 20
- Quiz 2: Oct 7
- Quiz 3: Nov 11
- Paper presentation (grad students only) : Nov 18
- Finals: Dec 11

AI News

#1 NEWS – PROMISE Book - PROMoting AI's Safe usage for Elections

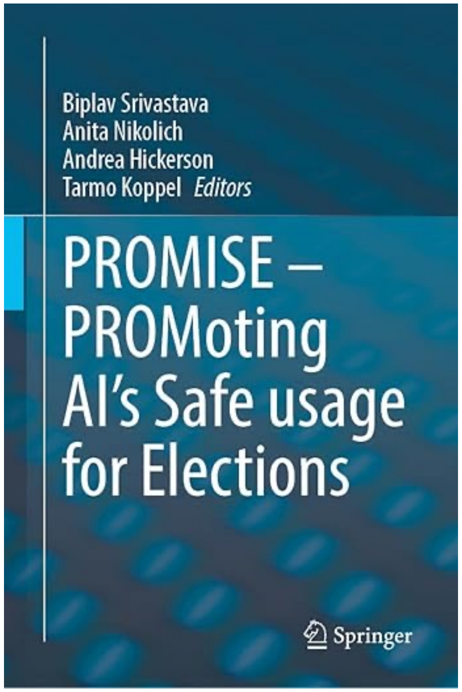
- Link: https://www.linkedin.com/posts/biplav-srivastava_promise-promoting-ais-safe-usage-for-elections-activity-7377405499815911424-nlgU
- Press: https://www.nbcnews.com/tech/security/hacker-used-ai-automate-unprecedented-cybercrime-spree-anthropic-says-rcna227309



- It is an edited book based on four years of work that explores AI's role in enhancing credible, data-driven electoral processes.
- This all started with a few of us brainstorming on how AI could make elections better that lead to academic workshops at Neurips 2021, AAAI 2023, AAAI 2024, a special issue of the AI Magazine on the topic, and a blue-sky paper at AAAI 2025.
- The book has contributions from over 30 authors that are drawn from academia, industry, non-profits, and government, from around the world. They bring and combine technical perspectives from the lens of computer science and AI, security, journalism, law, and political science, and consider elections in all continents - Asia (India), Africa (Ghana, Nigeria, Kenya), Europe (Estonia, UK), North America (Canada, US) and Latin America (Brazil).
- The book consists of a mix of article types - research papers, interviews and essays, touching on impact potentials of AI technologies like chatbots, large language models, game theory and machine learning, for voters, candidates, and election commissions, and **ends with a code of ethics for those working in AI and election space using relevant guidance from computing and journalism fields**.
- It offers practical guidelines for researchers, teachers, practitioners, students and government officials.

#1 NEWS – PROMISE Book - PROMoting AI's Safe usage for Elections

- Link: https://www.linkedin.com/posts/biplav-srivastava_promise-promoting-ais-safe-usage-for-elections-activity-7377405499815911424-nlgU
- Press: https://www.nbcnews.com/tech/security/hacker-used-ai-automate-unprecedented-cybercrime-spree-anthropic-says-rcna227309



Technology in Elections: Code of Ethics – The 7 Easy Reckoner

Promoting Computing: (ACM – https://www.acm.org/code-of-ethics)
•Contribute to society and to human well-being, acknowledging that all people are stakeholders in computing
•Maintain high standards of professional competence, conduct, and ethical practice

Promoting Communication: (https://www.spj.org/ethicscode.asp)
•Seek truth and report it
•Be accountable and transparent

Promoting Model Citizenship Responsibility
•Minimize harm
•Respect everyone's view and give them space to express them
•Honor people and their free will to vote

#2 NEWS – Insights for Irmo Fire

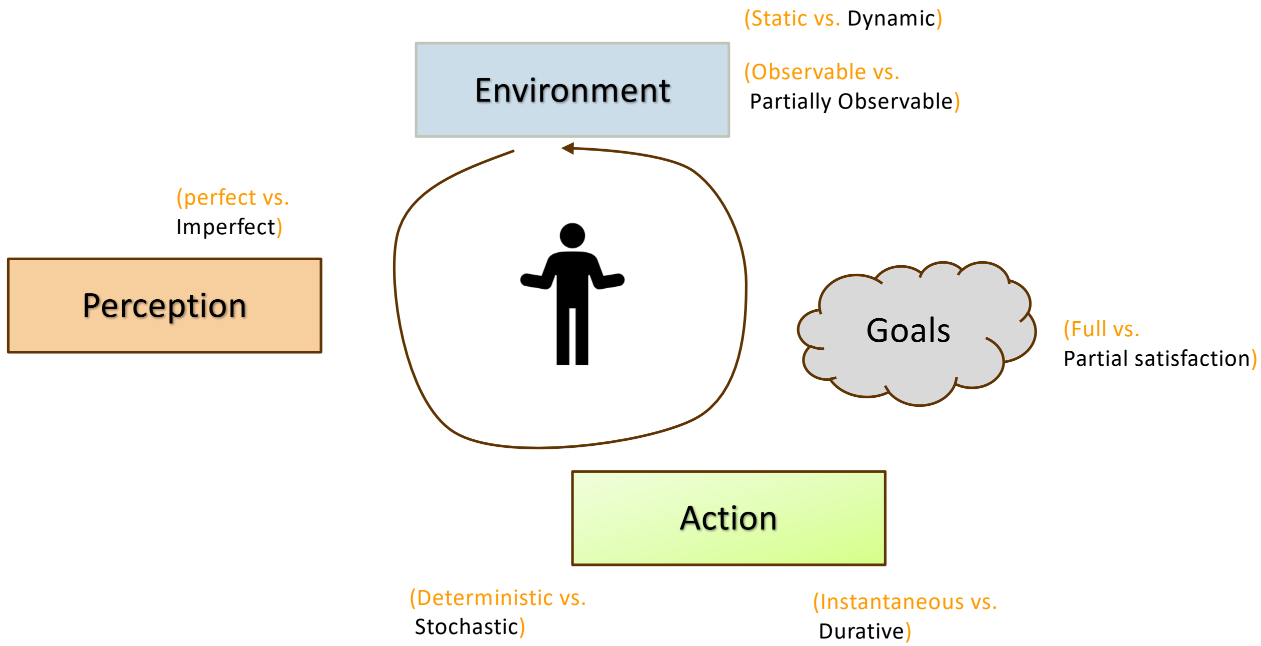
- Based on results from their data and Quiz1 analysis

Introduction Section

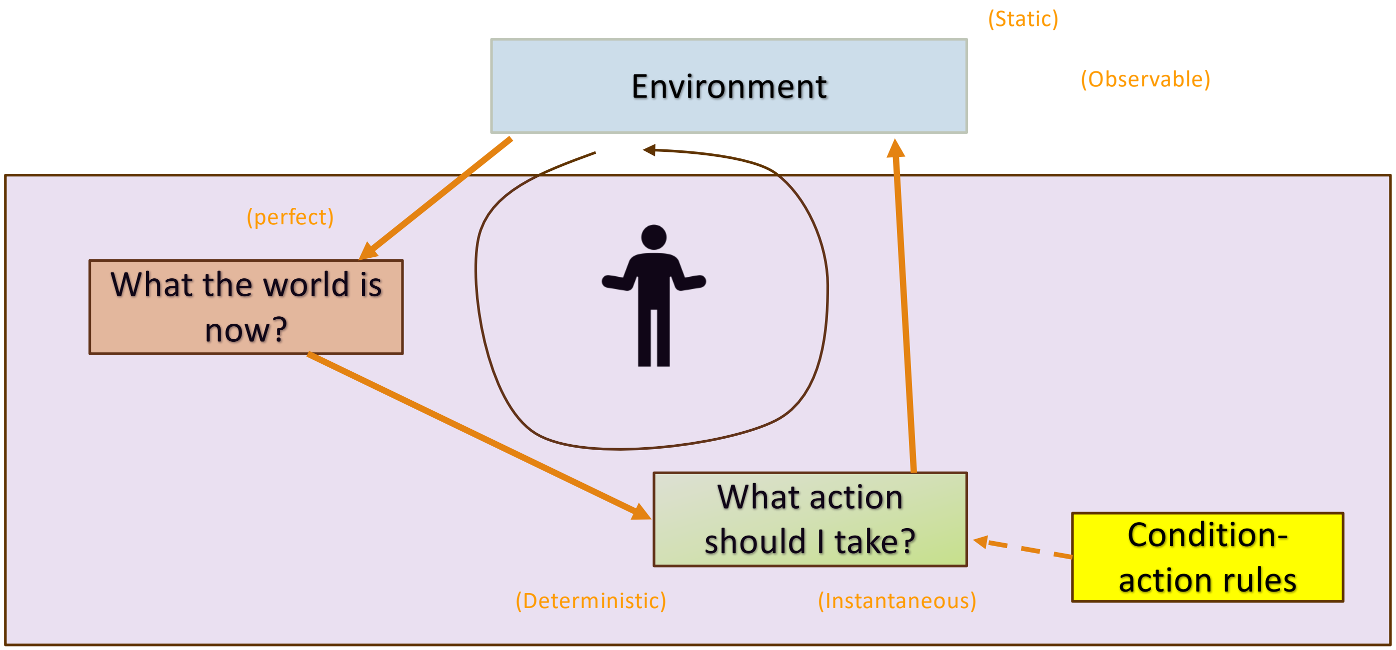
Main Section

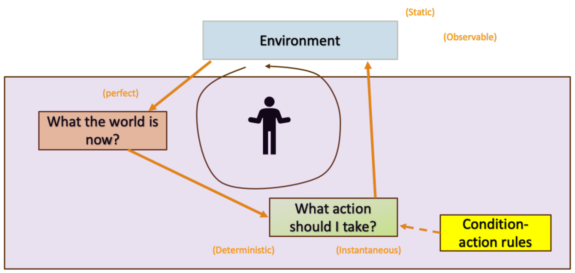
Lecture 13:
Logic and Inference - First Order

Intelligent Agent Model



Intelligent Agent – Simple Knowledge Based





KB Agent Procedure

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

   TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))      // Report (check)
   action ← ASK(KB, MAKE-ACTION-QUERY(t))           // Generate (ask)
   TELL(KB, MAKE-ACTION-SENTENCE(action, t))        // Report (check)
   t ← t + 1
   return action
```

Source: Russell & Norvig, AI: A Modern Approach

First Order Predicate Logic (FOPL)

Concepts

Constants: a, b, student123, teacher94
- Name of a specific object.

Variables: X, Y.
- Refer to an object without naming it.

Predicates: Father, Before
- Relationships between objects. May be many and may not be unique. Objects are specified as arguments (arity of a predicate).

Functions: father-of
- Mapping from objects to objects. Mapping must be present and be unique. Objects are specified as arguments (arity of a predicate).

Terms: dad-of(organism33), leftLeg(John)
- A logical expression that refers to an object

Atomic Sentences: in(dad-of(dog33), food6)
- Can be true or false
- Correspond to propositional symbols P, Q

Objects

Relations

Functions

Adapted from:
a) Dan Weld's AI course (CSE 573, Univ. of Washington)
b) Russell & Norvig, AI: A Modern Approach

FOPL - Syntax

Sentence — *AtomicSentence*
| *Sentence Connective Sentence*
| *Quantifier Variable,. . . Sentence*
| ¬ *Sentence*
| *(Sentence)*

AtomicSentence — *Predicate(Term,. . .)* | *Term = Term*

Term → *Function(Term,. . .)*
| *Constant*
\ *Variable*

Connective → ⇒ | A V I ⇔
Quantifier → V I 3
Constant → *A* \ *X*\ \ *John* | · · ·
Variable → a | *x* *s* • • •
Predicate → *Before* \ *HasColor* \ *Raining* \ · · ·
Function → *Mother* \ *LeftLegOf* \ · · ·

BNF (Backus-Naur Form) grammar
of sentences in FOPL

Source: Russell & Norvig, AI: A Modern Approach

Connectives and Quantifiers

Logical connectives: and, or, not, =>

Quantifiers:

- ∀ : Forall
- ∃ : There exists

Examples:

1. All students: ∀ **students**
2. All students are university members:
   ∀x Student(x) => UniversityMember(x)
   (For all x, if x is a student, then x is a UniversityMember)
3. A phone: **∃x Phone(x)**
4. John has a phone:
   ∃x Phone(x) ∧ Owns(John,x)
   (There exists a phone such that John owns it.)

Connections / Equivalences

$$
\begin{array}{llllll}
\forall x\ \neg P & = & \neg\exists x\ P & \qquad \neg P \wedge \neg Q & = & \neg(P \vee Q) \\
\neg\forall x\ P & = & 3x\ \neg P & \qquad \neg(P \wedge Q) & = & \neg P \vee \neg Q \\
\forall x\ P & = & \neg\exists x\ \neg P & \qquad P \wedge Q & = & \neg(\neg P \vee \neg Q) \\
\exists x\ \mathrm{P} & = & \neg\forall x\ \neg P & \qquad P \vee Q & = & \neg(\neg P \wedge \neg Q)
\end{array}
$$

Derivable from De Morgan's law about sets:
(A ∪ B)' = A' ∩ B' and (A ∩ B)' = A' ∪ B'

Source: Russell & Norvig, AI: A Modern Approach

Comparing Syntax - FOPL and Propositional Logic

Sentence → *AtomicSentence* | *ComplexSentence*

AtomicSentence → **True** | **False**
| *P* | *Q* | *R* | ...

ComplexSentence → (*Sentence*)
| *Sentence Connective Sentence*
| ¬*Sentence*

Connective → A | V | ⇔ | ⇒

Sentence → *AtomicSentence*
| *Sentence Connective Sentence*
| *Quantifier Variable*, . . . *Sentence*
| ¬ *Sentence*
| (*Sentence*)

AtomicSentence → *Predicate*(*Term*, . . .) | *Term* = *Term*

Term → *Function*(*Term*, . . .)
| *Constant*
| *Variable*

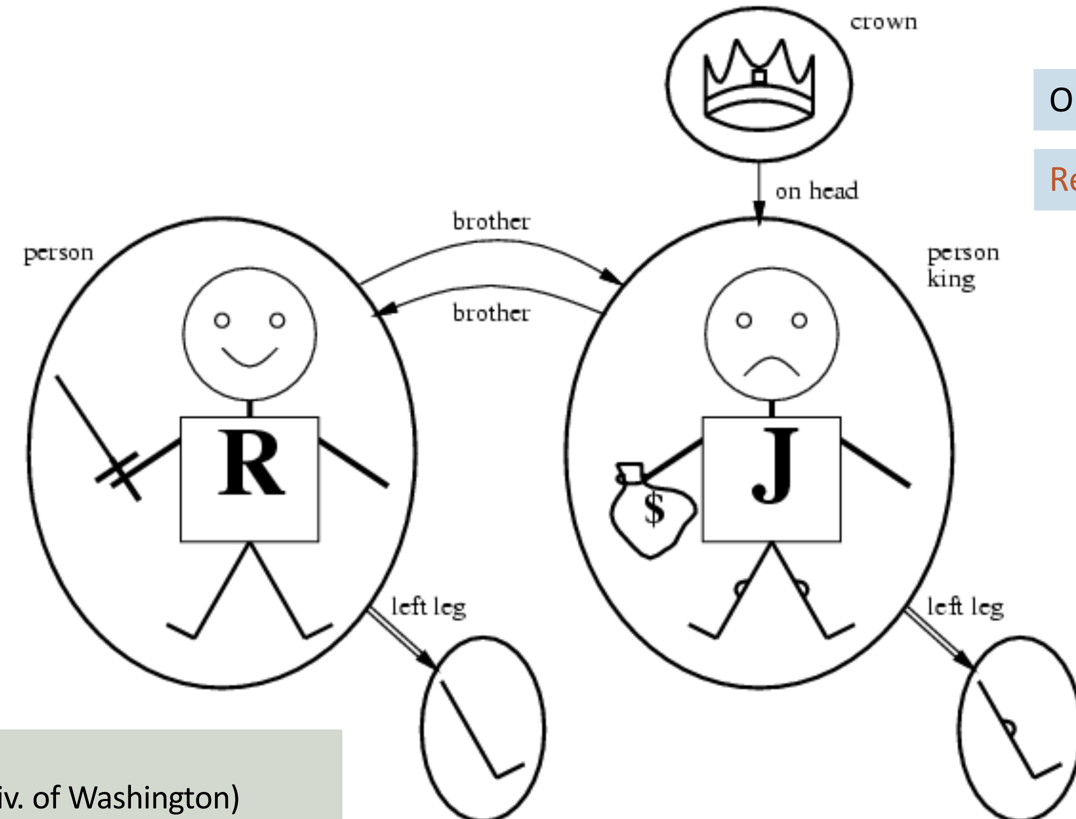
Connective → ⇒ | A | V | ⇔
Quantifier → V | 3
Constant → *A* | *X* | *John* | · · ·
Variable → a | *x* | *s* | • • •
Predicate → *Before* | *HasColor* | *Raining* | · · ·
Function → *Mother* | *LeftLegOf* | · · ·

Source: Russell & Norvig, AI: A Modern Approach

FOPL Semantics – Models and Interpretations

Richard, John
Constants

Leg(p,l)
Functions



On(x,y) King(p)
Relations

Adapted from:
a) Dan Weld's AI course (CSE 573, Univ. of Washington)
b) Russell & Norvig, AI: A Modern Approach

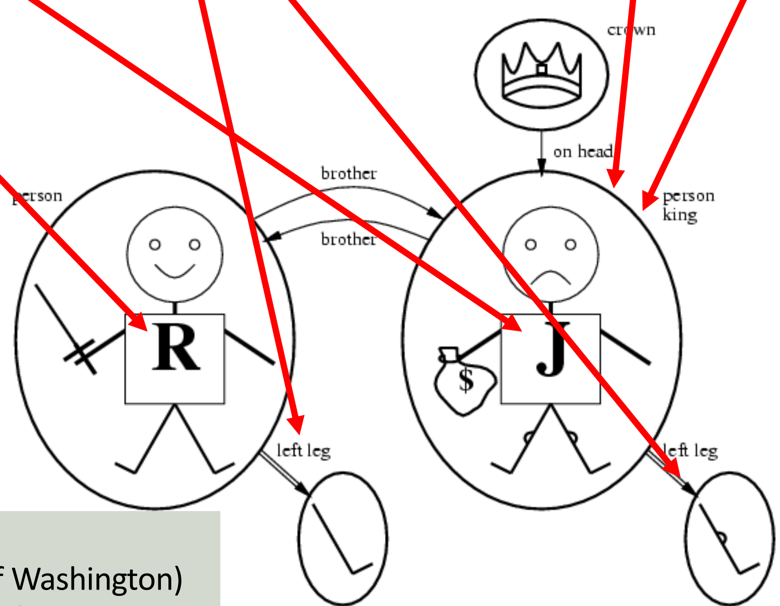
Interpretations - Mappings from Syntactic tokens → Model elements

Richard John
Constants

Leg(p,l)
Functions

On(x,y) King(p)
Relations

Depiction of one possible interpretation



Adapted from:
a) Dan Weld's AI course (CSE 573, Univ. of Washington)
b) Russell & Norvig, AI: A Modern Approach

Satisfiability, Validity, & Entailment

- S is **valid** if it is true in all interpretations
- S is **satisfiable** if it is true in some interpretations
- S is **unsatisfiable** if it is false for all interpretations
- S1 **entails** S2 if forall interpretations where S1 is true, S2 is also true

Source: Dan Weld's AI course (CSE 573, Univ. of Washington

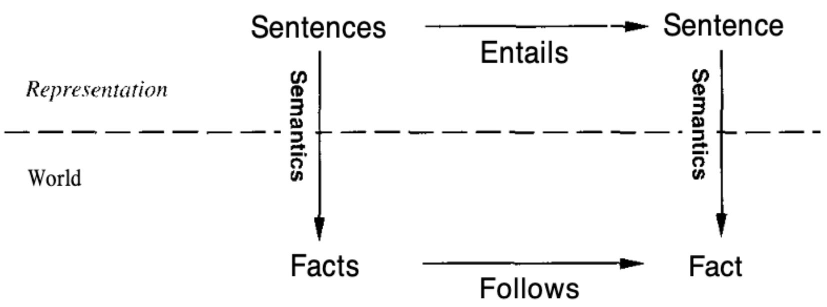
Comparing - Propositional Logic and FOPL

Ontology	Facts (P, Q)	Objects, Properties, Relations
Syntax	Atomic sentences Connectives	Variables & quantification Sentences have structure: terms father-of(mother-of(X)))
Semantics	Truth Tables	Interpretations *(Much more complicated)*
Inference Algorithm	DPLL, GSAT Fast in practice	Unification Forward, Backward chaining Prolog, theorem proving
Complexity	NP-Complete	Semi-decidable

Source: Dan Weld's AI course (CSE 573, Univ. of Washington

Formal Logic

- Properties of Logic System
  - **Soundness**: if it produces only true statements
  - **Completeness**: if it produces all true statements
  - **Consistency**: if it does not produce a sentence and its negation



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

Credits:
- Russell & Norvig, AI - A Modern Approach
- Deepak Khemani - A First Course in AI

Example: Course Selection

Example Situation – Course Selection

- A person wants to pass an academic program in two majors: A and B
- There are three subjects available: A, B and C, each with three levels (*1, *2, *3). There are thus 9 courses: A1, A2, A3, B1, B2, B3, C1, C2, C3
- To graduate, at least one course at beginner (*1) level is needed in major(s) of choice(s), and two courses at intermediate levels (*2) are needed
- Answer questions
  - Q1: How many minimum courses does the person have to take ?
  - Q2: Can a person graduate in 2 majors studying 3 courses only?
  - …

Representation – Propositional Example

- Domain Description: “There are three subjects: A, B and C, each with three levels (*1, *2, *3).”
- Representation
  - has_studied_courseA1: yes – student has taken course; no – student has not taken
  - has_studied_courseA2
  - has_studied_courseA3
  - has_studied_courseB1
  - has_studied_courseB2
  - has_studied_courseB3
  - has_studied_courseC1
  - has_studied_courseC2
  - has_studied_courseC3

LowerThan_Course_A1_CourseA2
LowerThan_Course_A2_CourseA3
LowerThan_Course_B1_CourseB2
LowerThan_Course_B2_CourseB3
LowerThan_Course_C1_CourseC2
LowerThan_Course_AC_CourseC3

- Previous statements set did not capture hierarchy between levels; new sentences would not have followed the reality in the world. Need more statements – LowerThan as shown.

Representation – FOPL Example

- Domain Description: “There are three subjects: A, B and C, each with three levels (*1, *2, *3).”
- Representation
  - has_studied (?x , ?y)
    - ?x: course name // A, B, C
    - ?y: course level // 1, 2, 3
  - lower_than _level(?x, ?y)
    - ? x: 1, 2
    - ?y: 2, 3

Revisiting Formal Representations: Ontologies

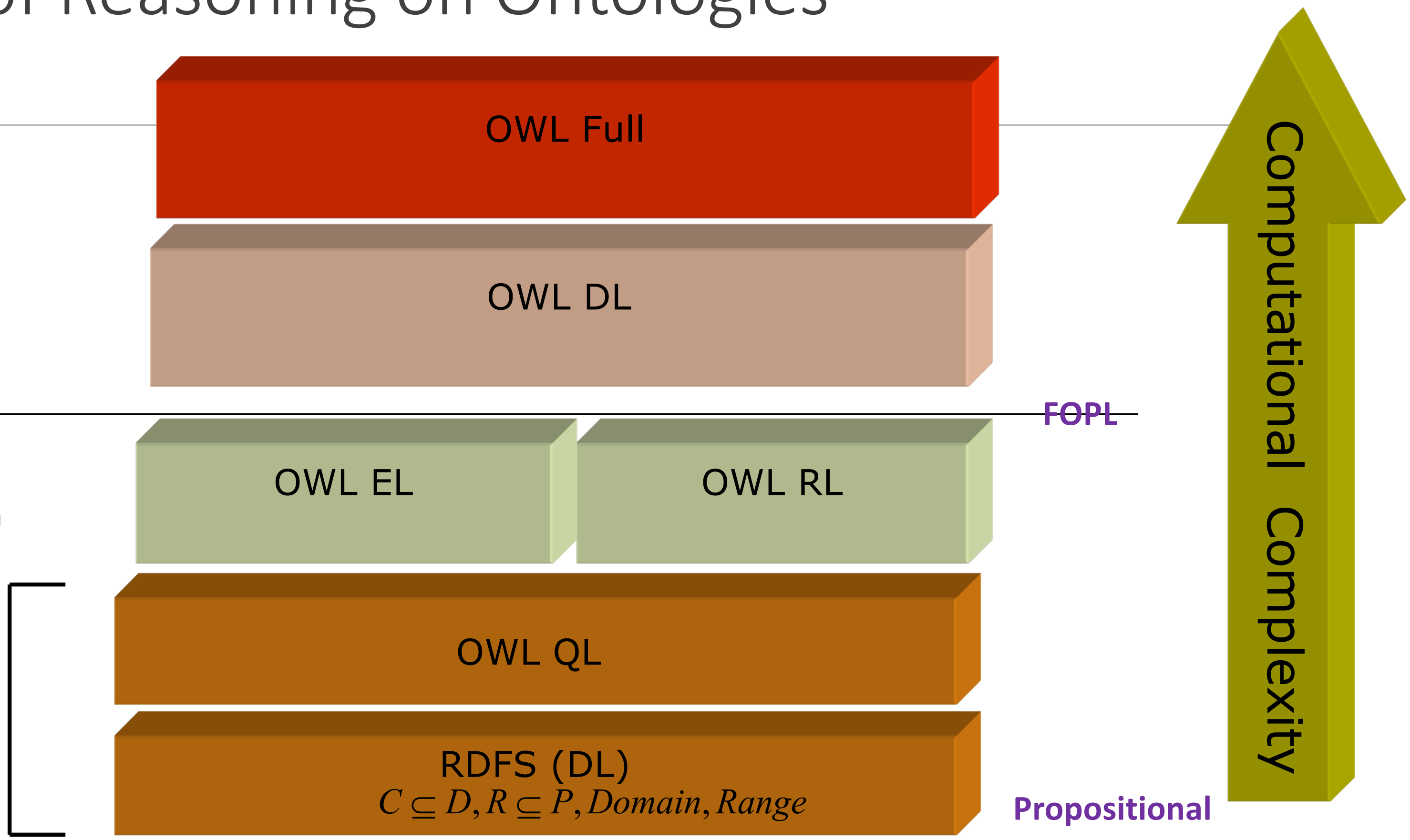
Challenge of Reasoning on Ontologies

- Undecidable

- Data Complexity open (NP-hard)
- Optimized tableau-based reasoners

- PTime-complete Data Complexity
- Efficient implementation on top of rule engine

- LogSpace Data Complexity
- Efficient evaluation on top of RDBMS



Lecture 13: Summary

- We talked about
  - Knowledge-based agents
  - Logic (Propositional)
  - Inferencing (Propositional)
  - Project B

Lecture 14:
Search, Search - Uninformed

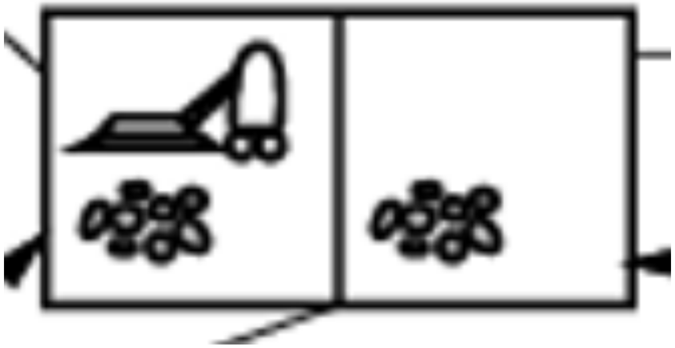
Lecture 14: Outline

We will discuss

- Problems and representation: vacuum, sliding tile, N-queens
- Search – uninformed methods
- Analyzing search performance
  - Breadth-first search
  - Uniform-cost search
  - Depth-first search
  - Depth-limited search
  - Iterative deepening search
  - Bidirectional search

Example: Vacuum World

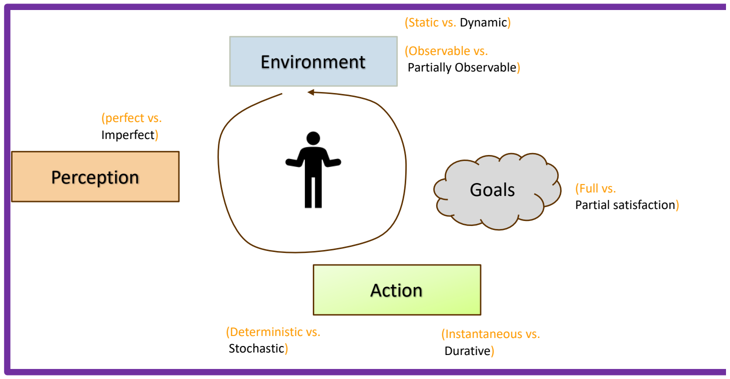
- Situation
  - Two rooms
  - One robot
  - Dirt can be in any room
- Goal
  - Clean the rooms
- Actions
  - Move left, move right, clean



Adapted from:
1. Russell & Norvig, AI: A Modern Approach
2. Bart Selman's CS 4700 Course

Goal-directed Problem Solving Agents

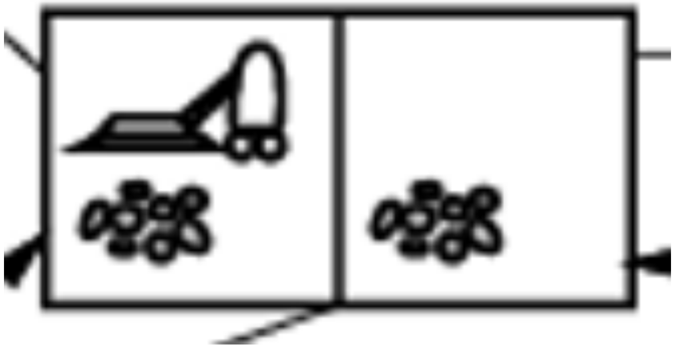
1. Goal Formulation: Have one or more (desirable) world states
2. Problem formulation: What actions and states to consider given goals and an initial state
3. Search for solution: Given the problem, search for a solution - a sequence of actions to achieve the goal starting from the initial state
4. Execution: agent can execute actions in the solution



Adapted from:
1. Russell & Norvig, AI: A Modern Approach
2. Bart Selman's CS 4700 Course

Modeling and Abstraction Consideration

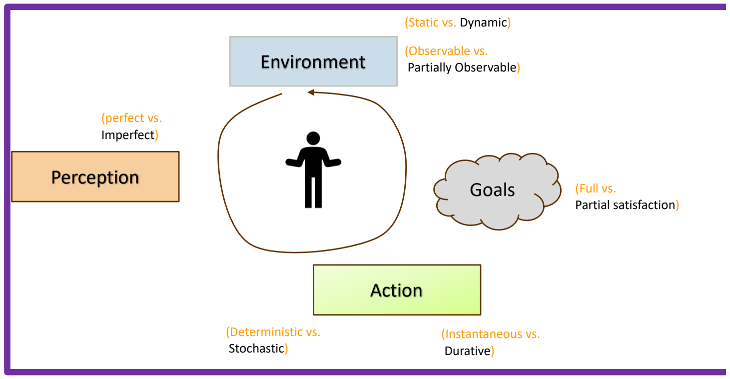
- Model: an abstract representation of the problem
  - “All models are wrong, but some are useful”
- What to capture, what to avoid
  - Only the necessary details needed to solve the problem
- In the example, we can avoid
  - For concepts
    - Size of rooms or robot
    - Quantity of dirt
  - For actions
    - Time taken to clean
    - Charging/ recharging time
    - Doing nothing – staying at the same place?



- Concepts
  - Two rooms
  - One robot
  - Dirt can be in any room
- Goal
  - Clean the rooms
- Actions
  - Move left, move right, clean

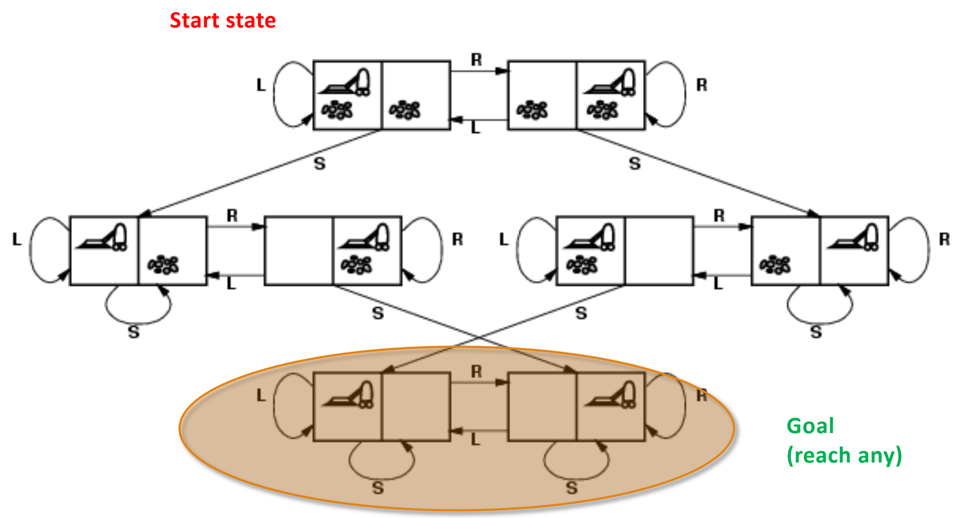
Open Loop v/s Closed Loop Systems

- Open loop
  - Systems **without** feedback
  - Assuming the world will not change, after a solution is found, one can simply execute it one action at a time
  - Usually make ***closed-world assumption*** – world cannot change other than by the system itself
- Closed loop
  - Systems **with** feedback
  - If the world keeps changing, after a solution is found, one cannot ignore perception when executing actions
  - The solution has to be relooked whenever an action is being executed. New solutions may have to be found at each step again.
  - Usually make ***open-world assumption*** – world can change due to other potential actors



Formulating a Problem

States • Initial state • Goal state	8 possible world states *(2room x 2dirt location x 2clean?)* • Any • No dirt at all locations
Actions • Transition model • Action cost	Left, Right, Suck • Action transition (edges) • 1



Adapted from:
1. Russell & Norvig, AI: A Modern Approach
2. Bart Selman's CS 4700 Course

Type of Problems

Standardized Problems

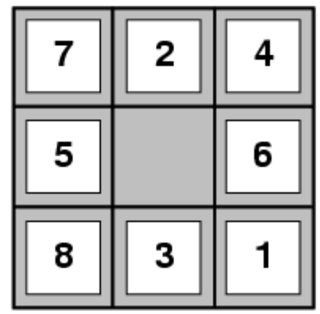
- Grid world
- Sliding tile
- Sokoban
- Chess
- …

Real-World Problems

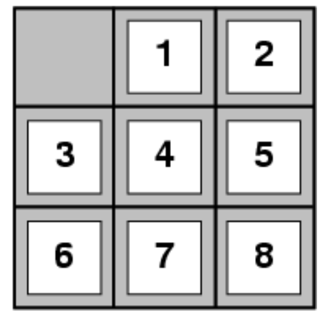
- Route finding
- Robotic / space craft navigation
- Protein design: find a sequence of amino acids that will fold into a 3D protein structure
- Dialog generation: how to give an effective answer that a person can understand
- …

Exercise: Sliding 8-tile Puzzle

States • Initial state • Goal state	
Actions • Transition model • Action cost	



Start State

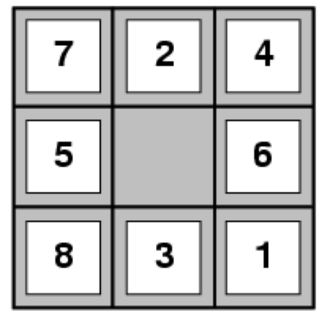


Goal State

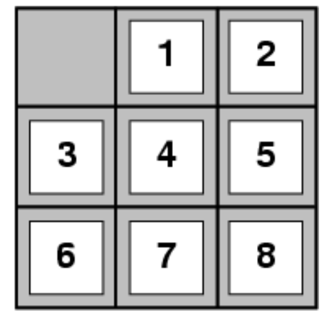
Adapted from:
Russell & Norvig, AI: A Modern Approach

Exercise: Sliding 8-tile Puzzle

States • Initial state • Goal state	Location of tiles • Any (given) • All numbers sorted, Empty tile in corner (given)
Actions • Transition model • Action cost	move blank left, right, up, down • Blank transition (edges) • 1



Start State



Goal State

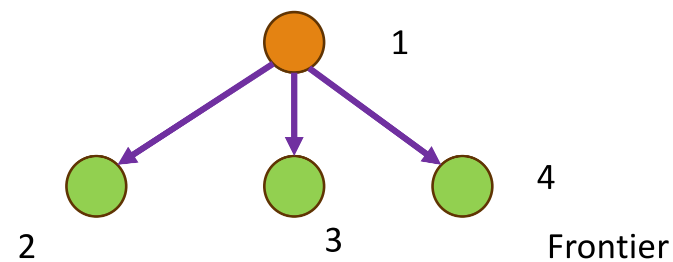
Adapted from:
1. Russell & Norvig, AI: A Modern Approach
2. Bart Selman's CS 4700 Course

Search

Search Basics

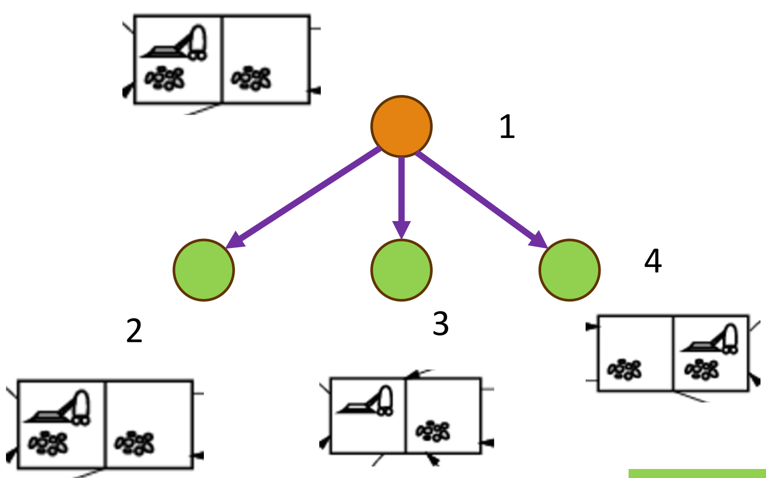
- input: a problem with **states** and actions
- output: solution(s) or flag for failure

- Concepts:
  - **Node**: corresponds to a <u>*state of the problem*</u> *(may or may not be the state of the world)*
  - **Edges**: transition between states
  - **Expand**: consider actions in the state (ACTIONS) and transition model. Generate new nodes corresponding to resulting states (RESULT)
  - **Explore**: check when a node meets goal condition

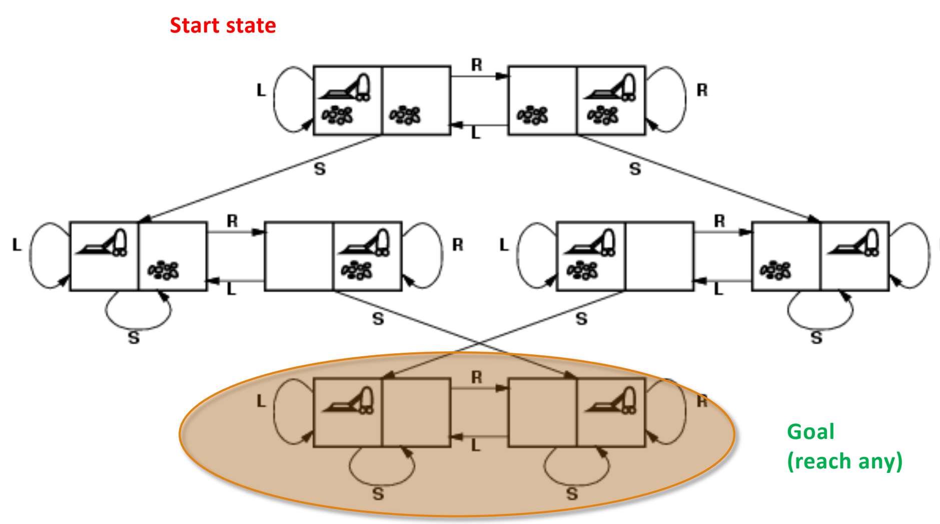


Node 1 has been **Reached**, Nodes {2,3 4} constitute Node 1's **Frontier**

Formulating a Problem



Search States



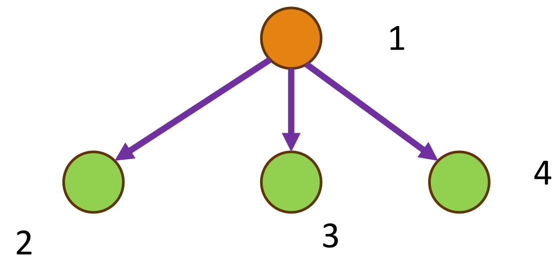
World States

Adapted from:
1. Russell & Norvig, AI: A Modern Approach
2. Bart Selman's CS 4700 Course

Tree-search Algorithms

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

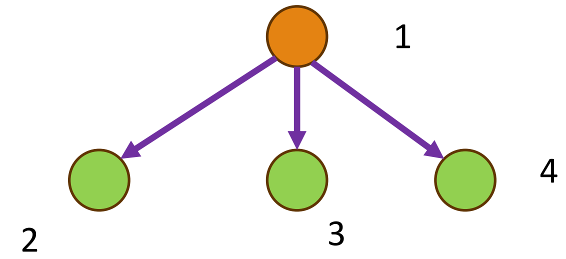
function TREE-SEARCH(*problem, strategy*) **returns** a solution, or failure
  initialize the search tree using the initial state of *problem*
  loop do
    if there are no candidates for expansion **then return** failure
    choose a leaf node for expansion according to *strategy*
    if the node contains a goal state **then return** the corresponding solution
    else expand the node and add the resulting nodes to the search tree



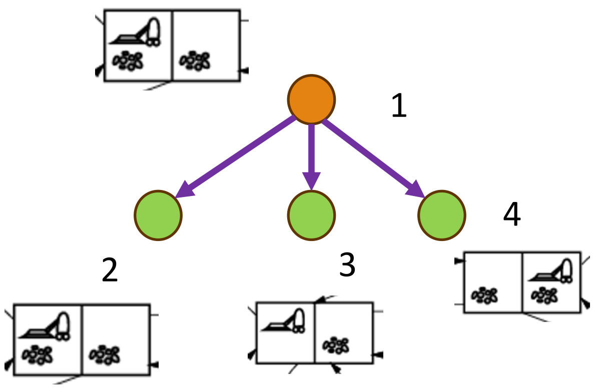
Adapted from:
1. Russell & Norvig, AI: A Modern Approach
2. Bart Selman's CS 4700 Course

Implementing a Tree-Search Algorithm

- Data structure
  - node.STATE: the state to which the node corresponds
  - node.PARENT: the node in the tree that generated this node
  - node.ACTION: the action that was applied to the parent to generate this node
  - node.PATH-COST: the total cost of the path from the initial state to this node
- Queue to store frontier
  - Is-EMPTY(frontier): true/ false depending on whether frontier is empty
  - POP(frontier): remove the top node from the frontier and return it
  - TOP(frontier): returns the top node from the frontier but does not remove it
  - ADD(node, frontier): insert node into its proper place in the queue
- Queue:
  - priority queue – removes the node with minimum cost according to some evaluation function
  - FIFO queue – first in, first output. Used in breadth first search
  - LIFO queue - last in, first output. Used in depth first search



Best-First Search



function BEST-FIRST-SEARCH(*problem*, *f*) **returns** a solution node or *failure*
  node ← NODE(STATE=*problem*.INITIAL)
  frontier ← a priority queue ordered by *f*, with *node* as an element
  reached ← a lookup table, with one entry with key *problem*.INITIAL and value *node*
  while not IS-EMPTY(*frontier*) **do**
    node ← POP(*frontier*)
    if *problem*.IS-GOAL(*node*.STATE) **then return** *node*
    for each *child* **in** EXPAND(*problem*, *node*) **do**
      s ← *child*.STATE
      if *s* is not in *reached* **or** *child*.PATH-COST < *reached*[*s*].PATH-COST **then**
        reached[*s*] ← *child*
        add *child* to *frontier*
  return *failure*

function EXPAND(*problem*, *node*) **yields** nodes
  s ← *node*.STATE
  for each *action* **in** *problem*.ACTIONS(*s*) **do**
    s′ ← *problem*.RESULT(*s*, *action*)
    cost ← *node*.PATH-COST + *problem*.ACTION-COST(*s*, *action*, *s′*)
    yield NODE(STATE=*s′*, PARENT=*node*, ACTION=*action*, PATH-COST=*cost*)

Source: Russell & Norvig, AI: A Modern Approach

Examples of Search Strategies

- Uninformed
  - Depth first
  - Breadth first
- Informed (Heuristic)
  - Greedy best first search
  - A* search

More on Search Strategies

- A search strategy is defined by picking the **order of node expansion.**
- 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 ∞)

Adapted from:
1. Russell & Norvig, AI: A Modern Approach
2. Bart Selman's CS 4700 Course

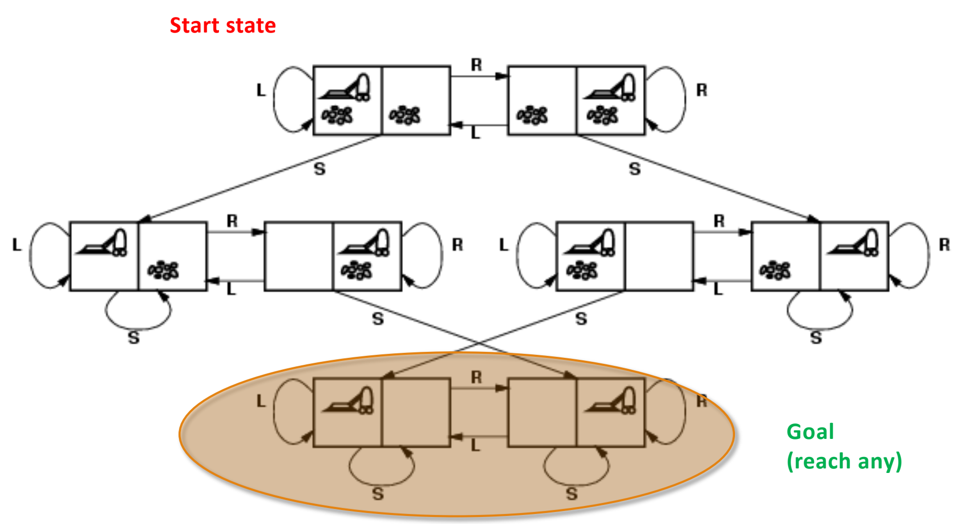
Exercise and Code

- Search Methods
  - From Book: AI – A Modern Approach, https://github.com/aimacode/aima-python/blob/master/search.ipynb

Source: Russell & Norvig, AI: A Modern Approach

Example: Vacuum World

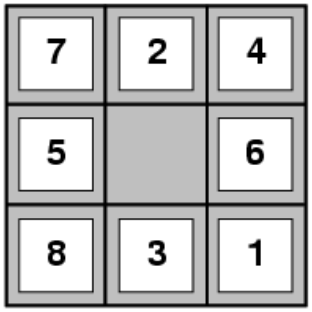
States • Initial state • Goal state	8 possible world states *(2room x 2dirt location x 2clean?)* • Any • No dirt at all locations
Actions • Transition model • Action cost	Left, Right, Suck • Action transition (edges) • 1



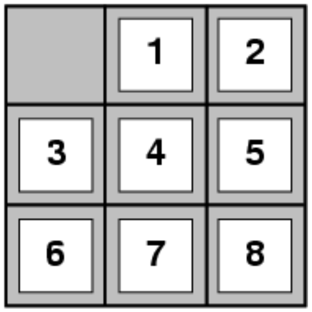
Adapted from:
1. Russell & Norvig, AI: A Modern Approach
2. Bart Selman's CS 4700 Course

Example: Sliding 8-tile Puzzle

States • Initial state • Goal state	Location of tiles • Any (given) • All numbers sorted, Empty tile in corner (given)
Actions • Transition model • Action cost	move blank left, right, up, down • Blank transition (edges) • 1



Start State

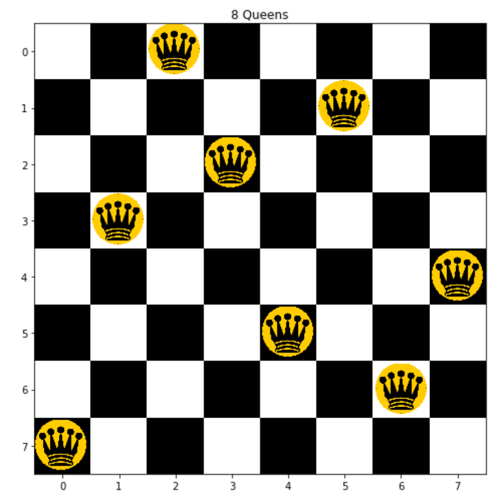


Goal State

Adapted from:
1. Russell & Norvig, AI: A Modern Approach
2. Bart Selman's CS 4700 Course

Exercise: N-Queen Puzzle

States • Initial state • Goal state	
Actions • Transition model • Action cost	



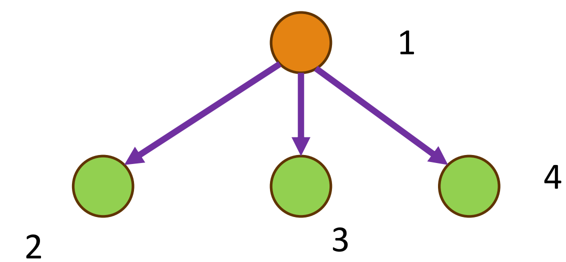
Adapted from:
Russell & Norvig, AI: A Modern Approach

Illustration: N-Queens Solving

- Solving: https://www.geeksforgeeks.org/dsa/n-queen-problem-backtracking-3/
- Visualization: https://github.com/Karthik-Nayak98/N-queens-visualiser

Tree-search Algorithms

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




```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
  initialize the search tree using the initial state of problem
  loop do
     if there are no candidates for expansion then return failure
     choose a leaf node for expansion according to strategy
     if the node contains a goal state then return the corresponding solution
     else expand the node and add the resulting nodes to the search tree
```

Adapted from:
1. Russell & Norvig, AI: A Modern Approach
2. Bart Selman's CS 4700 Course

Uninformed Search Strategies

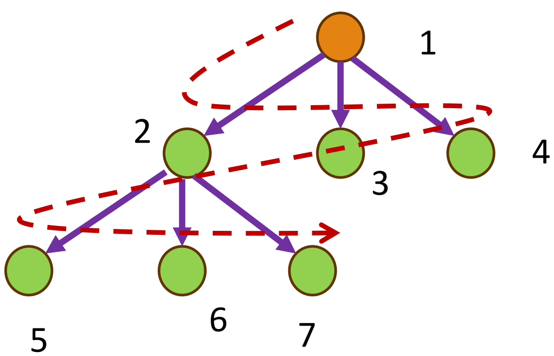
Search strategies use only the information available in the problem definition. They do not use a measure of distance to goal (*uninformed*).

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

Consideration: type of queue used for the fringe of the search tree (collection of tree nodes that have been generated but not yet expanded)

Adapted from:
1. Russell & Norvig, AI: A Modern Approach
2. Bart Selman's CS 4700 Course

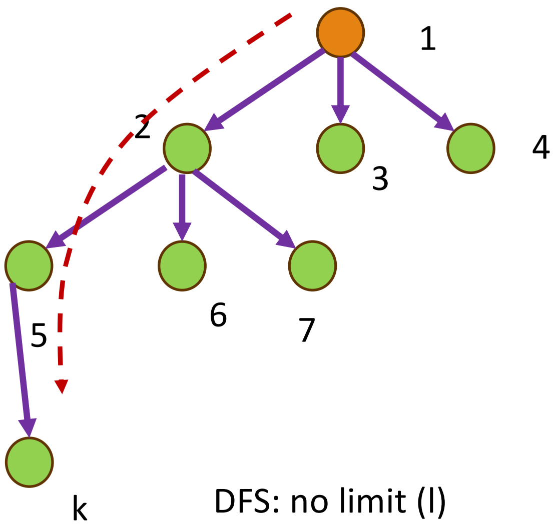
Breadth First Search (BFS)




```
function BREADTH-FIRST-SEARCH(problem) returns a solution node or failure
  node ← NODE(problem.INITIAL)
  if problem.IS-GOAL(node.STATE) then return node
  frontier ← a FIFO queue, with node as an element
  reached ← {problem.INITIAL}
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if problem.IS-GOAL(s) then return child
      if s is not in reached then
        add s to reached
        add child to frontier
  return failure
```

Adapted from: Russell & Norvig, AI: A Modern Approach

Depth First Search (DFS) and Depth Limited Search (DLS)

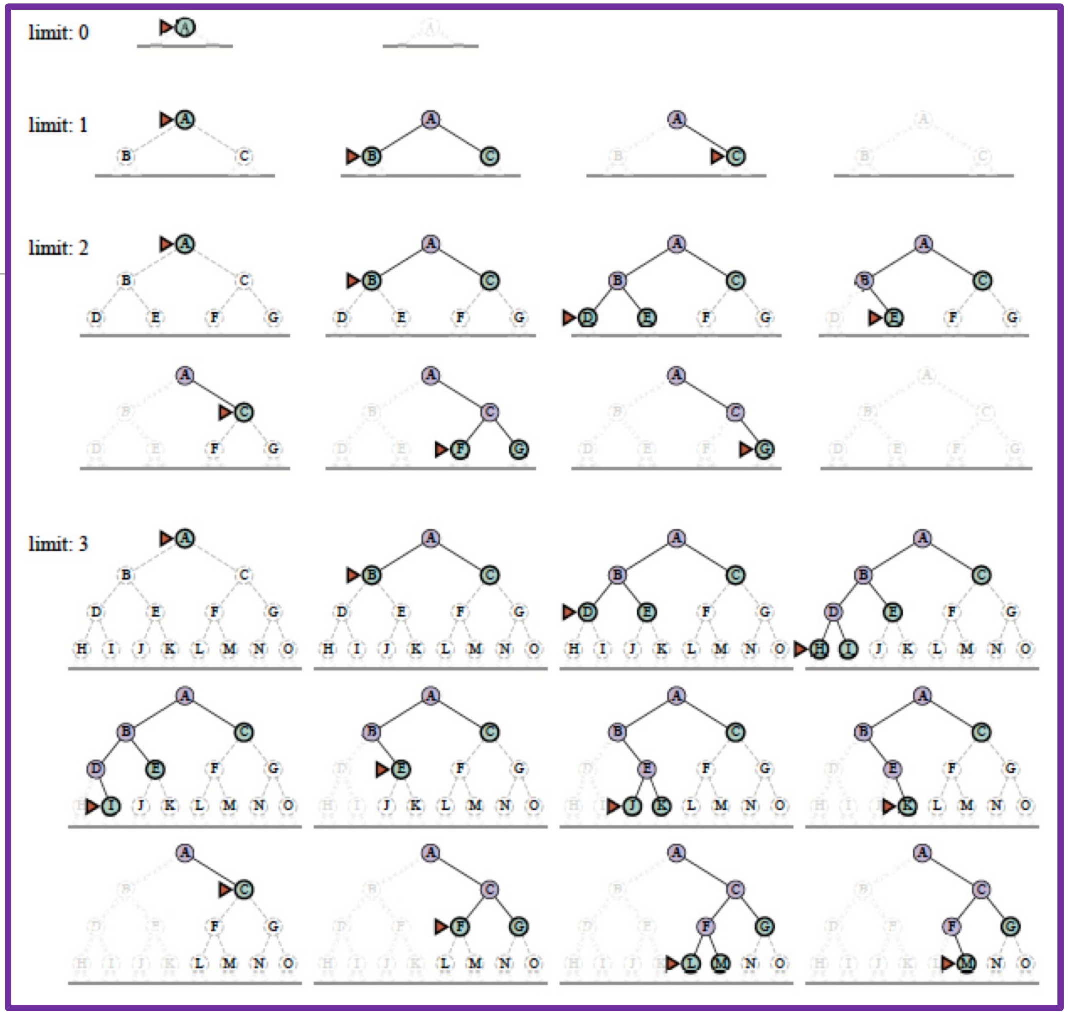



```
function DEPTH-LIMITED-SEARCH(problem, ℓ) returns a node or failure or cutoff
  frontier ← a LIFO queue (stack) with NODE(problem.INITIAL) as an element
  result ← failure
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    if DEPTH(node) > ℓ then
      result ← cutoff
    else if not IS-CYCLE(node) do
      for each child in EXPAND(problem, node) do
        add child to frontier
  return result
```

DFS: no limit (l)
Cutoff: when result is cutoff due to l
(result may be there if l increased)

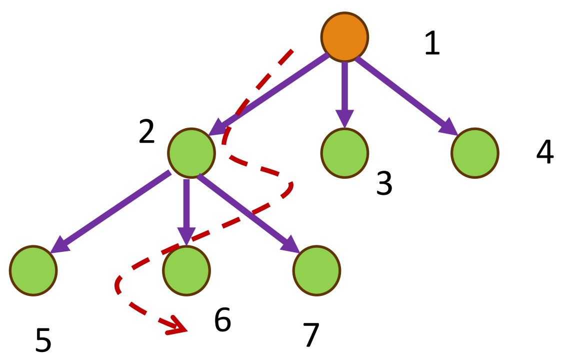
Adapted from: Russell & Norvig, AI: A Modern Approach

Illustration: DLS



Adapted from: Russell & Norvig, AI: A Modern Approach

Best-First Search

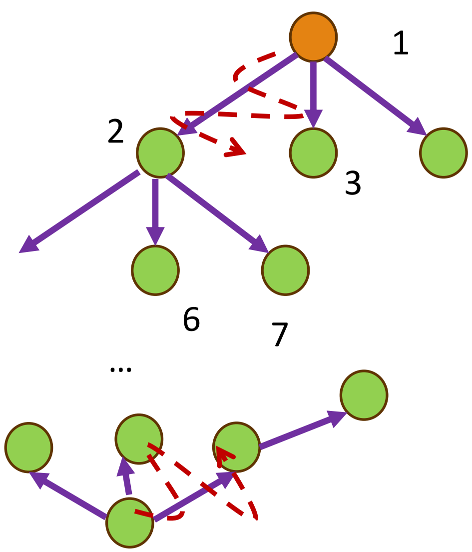



```
function BEST-FIRST-SEARCH(problem, f) returns a solution node or failure
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by f, with node as an element
  reached ← a lookup table, with one entry with key problem.INITIAL and value node
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
        reached[s] ← child
        add child to frontier
  return failure

function EXPAND(problem, node) yields nodes
  s ← node.STATE
  for each action in problem.ACTIONS(s) do
    s′ ← problem.RESULT(s, action)
    cost ← node.PATH-COST + problem.ACTION-COST(s, action, s′)
    yield NODE(STATE=s′, PARENT=node, ACTION=action, PATH-COST=cost)
```

Source: Russell & Norvig, AI: A Modern Approach

Bi-Directional Search



Q: Why do this?
Q: When do this?

```
function BIBF-SEARCH(problem_F, f_F, problem_B, f_B) returns a solution node, or failure
  node_F ← NODE(problem_F.INITIAL)                              // Node for a start state
  node_B ← NODE(problem_B.INITIAL)                              // Node for a goal state
  frontier_F ← a priority queue ordered by f_F, with node_F as an element
  frontier_B ← a priority queue ordered by f_B, with node_B as an element
  reached_F ← a lookup table, with one key node_F.STATE and value node_F
  reached_B ← a lookup table, with one key node_B.STATE and value node_B
  solution ← failure
  while not TERMINATED(solution, frontier_F, frontier_B) do
    if f_F(TOP(frontier_F)) < f_B(TOP(frontier_B)) then
      solution ← PROCEED(F, problem_F frontier_F, reached_F, reached_B, solution)
    else solution ← PROCEED(B, problem_B, frontier_B, reached_B, reached_F, solution)
  return solution

function PROCEED(dir, problem, frontier, reached, reached_2, solution) returns a solution
        // Expand node on frontier; check against the other frontier in reached_2.
        // The variable "dir" is the direction: either F for forward or B for backward.
  node ← POP(frontier)
  for each child in EXPAND(problem, node) do
    s ← child.STATE
    if s not in reached or PATH-COST(child) < PATH-COST(reached[s]) then
      reached[s] ← child
      add child to frontier
      if s is in reached_2 then
        solution_2 ← JOIN-NODES(dir, child, reached_2[s]))
        if PATH-COST(solution_2) < PATH-COST(solution) then
          solution ← solution_2
  return solution
```

Figure 3.14 Bidirectional best-first search keeps two frontiers and two tables of reached states. When a path in one frontier reaches a state that was also reached in the other half of the search, the two paths are joined (by the function JOIN-NODES) to form a solution. The first solution we get is not guaranteed to be the best; the function TERMINATED determines when to stop looking for new solutions.

Adapted from: Russell & Norvig, AI: A Modern Approach

Analyzing Search Performance

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes[1]	Yes[1,2]	No	No	Yes[1]	Yes[1,4]
Optimal cost?	Yes[3]	Yes	No	No	Yes[3]	Yes[3,4]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$

Figure 3.15 Evaluation of search algorithms. b is the branching factor; m is the maximum depth of the search tree; d is the depth of the shallowest solution, or is m when there is no solution; ℓ is the depth limit. Superscript caveats are as follows: [1] complete if b is finite, and the state space either has a solution or is finite. [2] complete if all action costs are $\geq \epsilon > 0$; [3] cost-optimal if action costs are all identical; [4] if both directions are breadth-first or uniform-cost.

Adapted from: Russell & Norvig, AI: A Modern Approach

Coding Example

- N-Queens – code notebook
  - https://github.com/biplav-s/course-ai-tai-f23/blob/main/sample-code/Class6-To-Class10-search.md

Lecture 14: Concluding Comments

We discussed

- Problems: vacuum, sliding tile, N-queens
- Search – uninformed
- Analyzing search performance

Week 7: Concluding Comments

We talked about

- First-order logic
- Search based solving
- Examples
- Uninformed search

- Week 1: Introduction, Aim: Chatbot / Intelligence Agent
- Weeks 2-3: Data: Formats, Representation and the Trust Problem
- Week 3: Machine Learning – Supervised (Classification)
- Week 4: Machine Learning - Unsupervised (Clustering) –
- Topic 5: Learning neural network, deep learning, Adversarial attacks
- Week 6: Large Language Models – Representation and Usage issues
- Weeks 7-8: Search, Heuristics - Decision Making
- Week 9: Constraints, Optimization – Decision Making
- Topic 10: Markov Decision Processes, Hidden Markov models - Decision making
- Topic 11-12: Planning, Reinforcement Learning – Sequential decision making
- Week 13: Trustworthy Decision Making: Explanation, AI testing
- Week 14: AI for Real World: Tools, Emerging Standards and Laws; Safe AI/ Chatbots

Projects B: Sep 30 – Nov 20 (7 weeks; 400 points)

- End date: **Thursday, Nov 20**
  - Remember to update spreadsheet on data/ time when finished **(Column I)**

- Choices
  - Given by instructor
  - Defined by student using project-b template; reviewed and approved by instructor

Upcoming Evaluation Milestones

- Projects B: Sep 30 – Nov 20
- Quiz 2: Oct 7
- Quiz 3: Nov 11
- Paper presentation (grad students only) : Nov 18
- Finals: Dec 11

About Week 8 – Lectures 15

Week 8 – Lecture 15

- Lecture 15: Quiz 2
- Fall Break

- Week 1: Introduction, Aim: Chatbot / Intelligence Agent
- Weeks 2: Data: Formats, Representation, ML Basics
- Week 3: Machine Learning – Supervised (Classification)
- Week 4: Machine Learning - Unsupervised (Clustering) –
- Topic 5: Learning neural network, deep learning, Adversarial attacks
- Week 6: Large Language Models – Representation and Usage issues
- Weeks 7-8: Search, Heuristics - Decision Making
- Week 9: Constraints, Optimization – Decision Making
- Topic 10: Markov Decision Processes, Hidden Markov models - Decision making
- Topic 11-12: Planning, Reinforcement Learning – Sequential decision making
- Week 13: Trustworthy Decision Making: Explanation, AI testing
- Week 14: AI for Real World: Tools, Emerging Standards and Laws; Safe AI/ Chatbots

Note: exact schedule changes slightly to accommodate for exams and holidays.