# Introduction to Artificial Intelligence

Revision

# Final Exam

- Venue: TBA
- Date: **23** November (**Tuesday**)
- Time: 5:00 – 7:00 pm

# Format

- Open book
- Six questions, emphasizing material covered after the midterm
  - Yes, **all** material in the course will be covered on the exam
- Bring a (non-programmable) calculator with you

# Outline

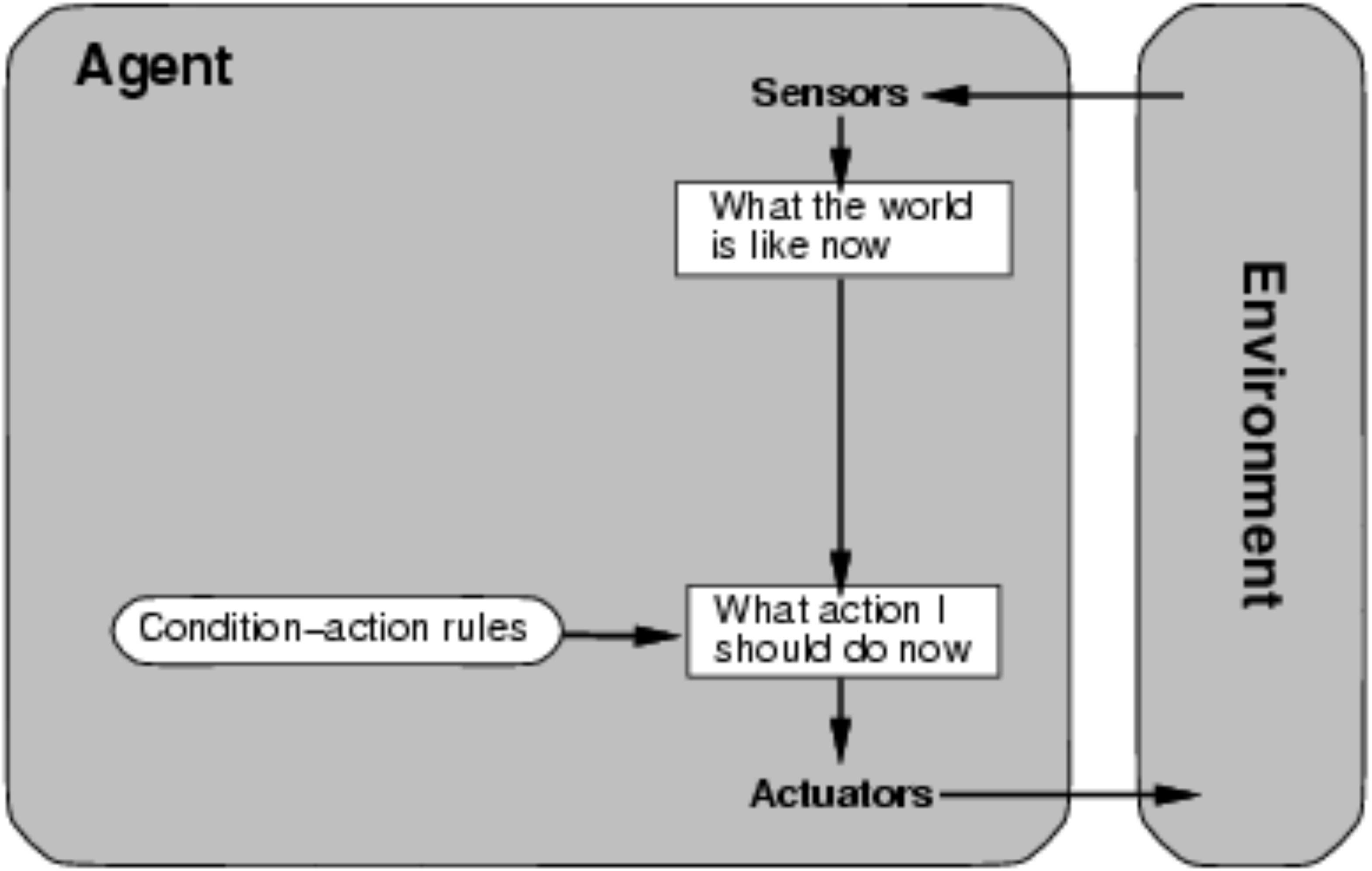
- Agents
- Search
  - Uninformed Search
  - Informed Search
- Adversarial Search
- Constraint Satisfaction
- Knowledge-Based Agents
- Uncertainty and Learning
- Natural Language Processing

# Agent types

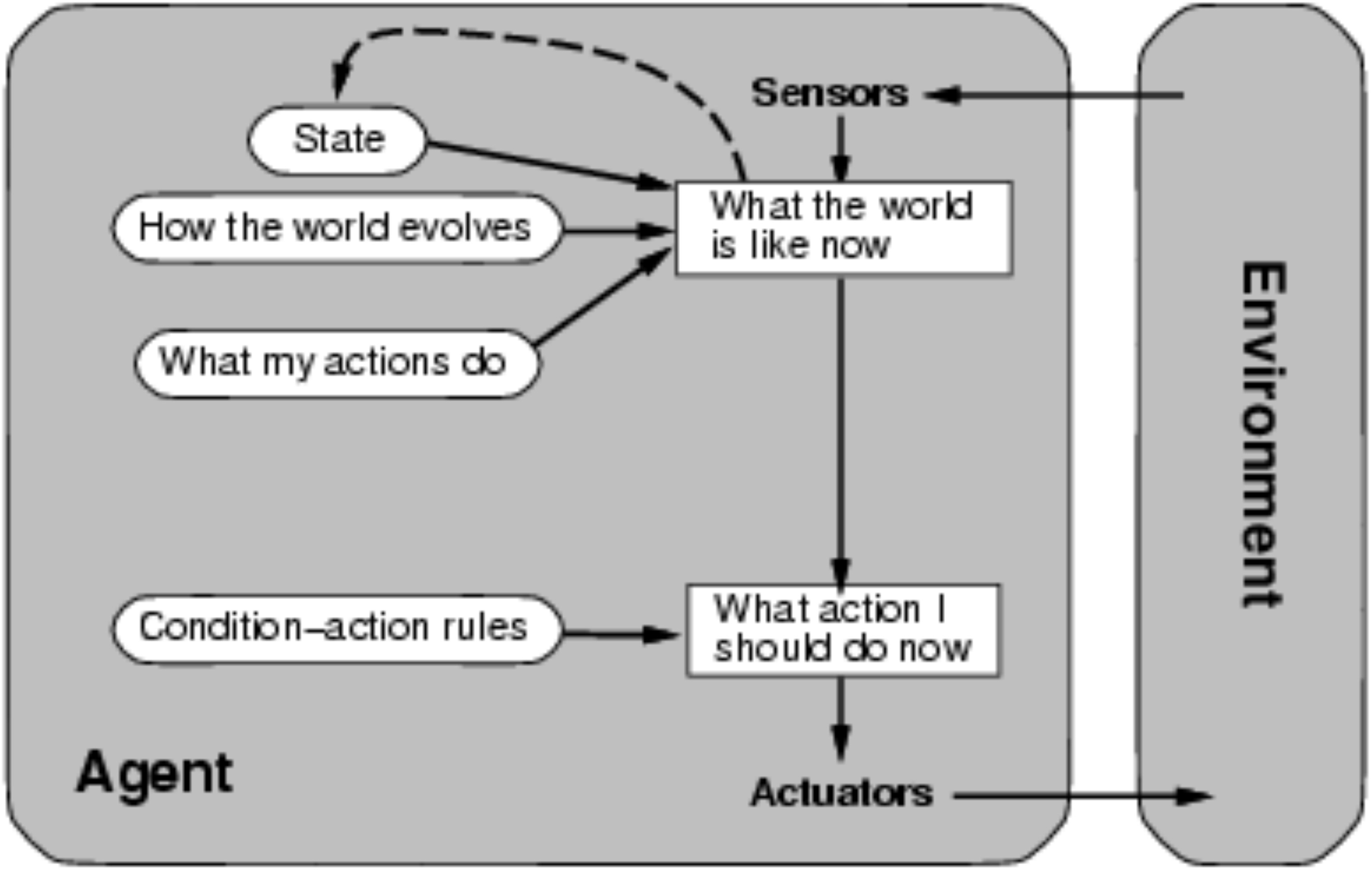
Four basic types in order of increasing generality:

- Simple reflex agents
- Model-based reflex agents
- Goal-based agents
- Utility-based agents

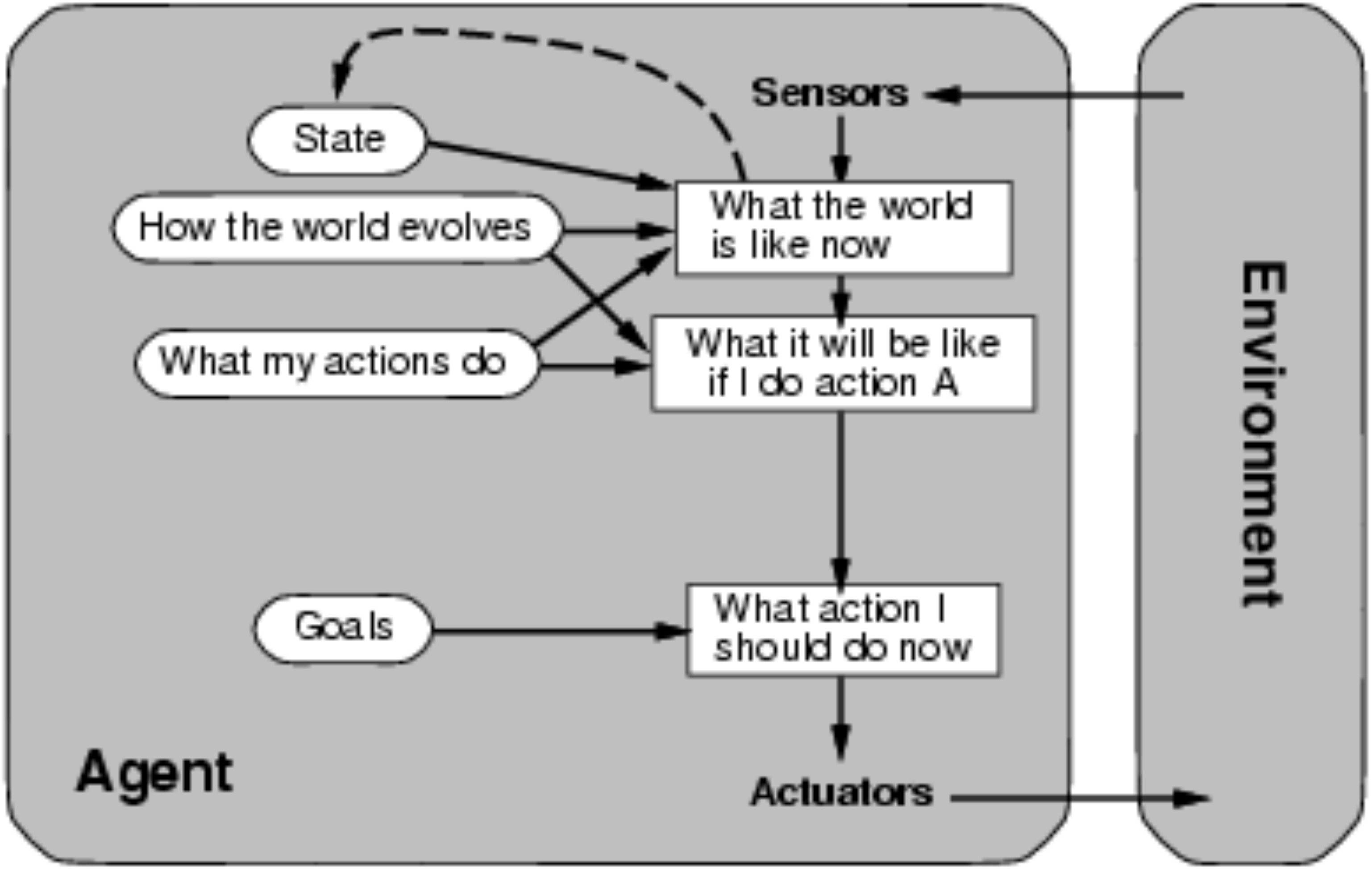
# Simple reflex agents



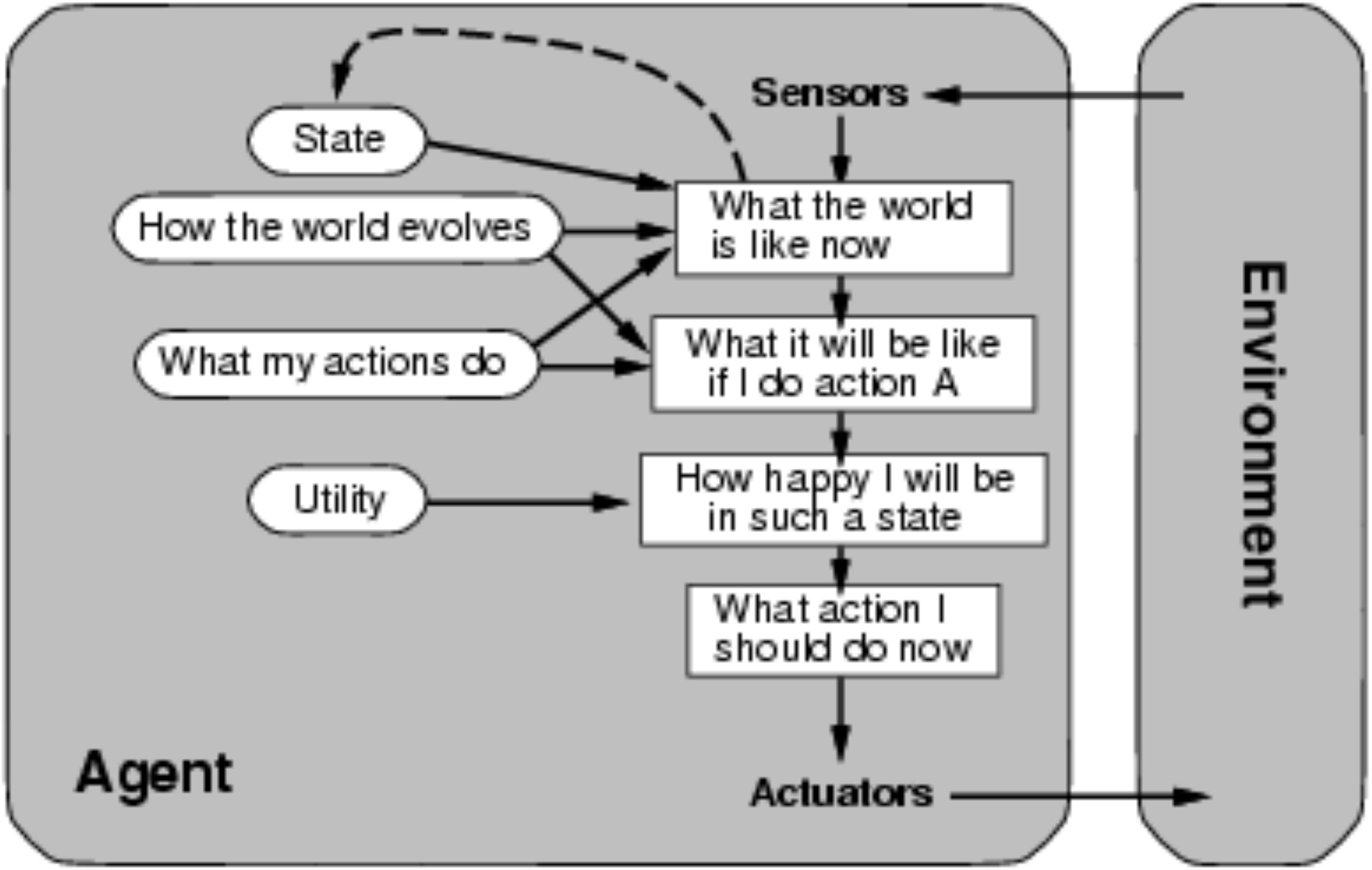
# Model-based reflex agents



# Goal-based agents



# Utility-based agents

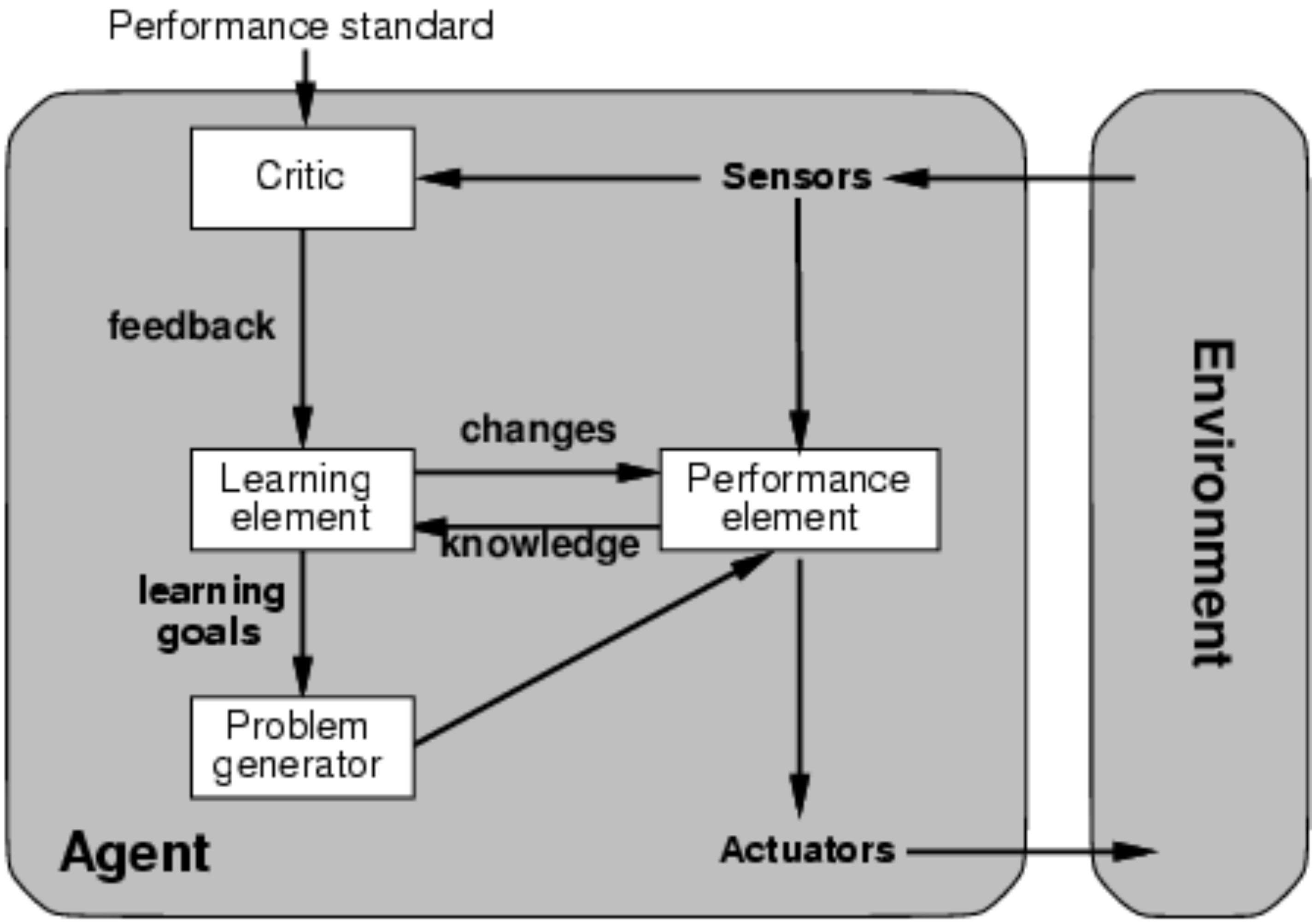


# Creating agents

Where does the intelligence come from?

- Coded by the designers
  - Knowledge representation – predicate and first order logic
- Learned by the machine
  - Machine learning – expose naïve agent to examples to learn useful actions

# Learning agents



# Searching for solutions

In most agent architectures, deciding what action to take involves considering alternatives

- Searching is judged on optimality, completeness and complexity
- Do I have a way of gauging how close I am to a goal?
  - No: Uninformed Search
  - Yes: Informed Search

# Uninformed search

- Formulate the problem, search and then execute actions
- Apply Tree-Search
- For environments that are
  - Deterministic
  - Fully observable
  - Static

# Tree search algorithm

- Basic idea:
  - offline, simulated exploration of state space by generating successors of already-explored 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
```

# Summary of algorithms

- Breadth-First – FIFO order
- Uniform-Cost – in order of cost
- Depth-First – LIFO order
- Depth-Limited – DFS to a maximum depth
- Iterative Deepening – Iterative DLS

- Bidirectional – also search from goal towards origin

| Criterion | Breadth-First | Uniform Cost | Depth First | Depth Limited | Iterative Deepening | Bidirectional |
|---|---|---|---|---|---|---|
| Complete? | Yes | Yes | No | No | Yes | Yes |
| Time | $O(b^{d+1})$ | $O(b^{\lceil C^*/e \rceil})$ | $O(b^m)$ | $O(b^l)$ | $O(b^d)$ | $O(b^{d/2})$ |
| Space | $O(b^{d+1})$ | $O(b^{\lceil C^*/e \rceil})$ | $O(bm)$ | $O(bl)$ | $O(bd)$ | $O(b^{d/2})$ |
| Optimal? | Yes | Yes | No | No | Yes | Yes |

# Repeated states: Graph-Search

```
function GRAPH-SEARCH(problem, fringe) returns a solution, or failure
    closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
        if STATE[node] is not in closed then
            add STATE[node] to closed
            fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

# Informed search

- Heuristic function *h(n)* = estimated cost of the cheapest path from n to goal.

- Greedy Best First Search
  - Minimizing estimated cost to goal
- A* Search
  - Minimizing total cost

# Properties of heuristic functions

- Admissible: never overestimates cost
- Consistent: estimated cost from node $n+1$ is $\geq$ than cost from node $n$ + step cost.

- A* using Tree-Search is optimal if the heuristic used is admissible.
  - Graph-Search needs a consistent heuristic. Why?

# Local search

- Good for solutions where the path to the solution doesn’t matter
  - Often work on a complete state
  - May not search systematically
  - Often require very little memory

- Correlated to online search
  - Have only access to the local state

# Local search algorithms

- Hill climbing search – choose best successor
- Beam search – take the best *k* successor
- Simulated annealing – allow backward moves during beginning of search
- Genetic algorithm – breed *k* successors using crossover and mutation

# Searching in specialized scenarios

- Properties of the problem often allow us to formulate
  - Better heuristics
  - Better search strategy and pruning
- Adversarial search
  - Working against an opponent
- Constraint satisfaction problem
  - Assigning values to variables
  - Path to solution doesn't matter
  - View this as an incremental search

# Adversarial Search

- Turn-taking, two-player, zero-sum games
- Minimax algorithm:
  - One turn = two plys
  - Max nodes: agent’s move, maximize utility
  - Min nodes: opponent’s move, minimize utility
  - Alpha-Beta pruning: rid unnecessary computation.

# Constraint Satisfaction

- Discrete or continuous solutions
  - Discretize and limit possible values
- Modeled as a constraint graph
- As the path to the solution doesn't matter, *local search* can be useful.

# Techniques in CSPs

- Basic: backtracking search
  - DFS for CSP
  - A leaf node (at depth $v$) is a solution
- Speed ups
  - Choosing variables
    - Minimum remaining values
    - Most constrained variable / degree
  - Choosing values
    - Least constraining value

# Pruning CSP search space

Before expanding node, can prune the search space

- Forward checking
  - Pruning values from remaining variables
- Arc consistency
  - Propagating stronger levels of consistency
  - E.g., AC-3 (applicable before searching and during search)
- Balancing arc consistency with actual searching

# Propositional and First Order Logic

- Propositional Logic
  - Facts are true or false
- First Order Logic
  - Relationships and properties of objects
  - More expressive and succinct
    - Quantifiers, functions
    - Equality operator
  - Can convert back to prop logic to do

# Inference in logic

- Given a KB, what can be inferred?
  - Query- or goal-driven
    - Backward chaining, model checking (e.g. DPLL), resolution
  - Deducing new facts
    - Forward chaining
      - Efficiency: track # of literals of premise using a count or Rete networks

# Inference in logic

- Chaining
  - Requires Definite Clauses or Horn Clauses
  - Uses Modus Ponens for sound reasoning
  - Forward or Backward
- Resolution
  - Requires Conjunctive Normal Form
  - Uses Resolution for sound reasoning
  - Proof by Contradiction

# Inference in FOL

- Don't have to propositionalize
  - Could lead to infinite sentences when functions occur
- Use unification instead
  - Standardizing apart
  - Dropping quantifiers
    - Skolem constants and functions
  - Handling equality
- Inference is semidecidable
  - Can say yes to entailed sentences, but non-entailed sentences will never terminate

# Connection to knowledge-based agents

- CSP can be formulated as logic problems and vice versa
- CSP search as model checking

| Model checking (DPLL) | CSP Search |
|---|---|
| Pure Symbol | Least constraining value |
| Unit Clause | Most constrained value |
| Early Termination | |
| | Minimum remaining values |

- Local search: WalkSAT with min-conflict heuristic

# Inference and CSPs

- Solving a CSP via inference
  - Handles special constraints (e.g., AllDiff)
  - Can learn new constraints not expressed by KB designer
- Solving inference via CSP
  - Whether a query is true under all possible constraints (satisfiable)
- Melding the two: Constraint Logic Programming (CLP)

# Uncertainty

- Leads us to use probabilistic agents
  - Only one of many possible methods!
- Modeled in terms of random variables
  - Again, we examined only the discrete case
- Answer questions based on full joint distribution

# Inference by enumeration

Interested in the posterior joint distribution of *query variables* given specific values for *evidence variables*

- Summing over *hidden variables*
- Cons: Exponential complexity

- Look for absolute and conditional independence to reduce complexity

# Bayesian networks

- One way to model dependencies
- Variable's probability only depends on its parents
- Use product rule and conditional dependence to calculate joint probabilities
- Easiest to structure causally
  - From root causes forward
  - Leads to easier modeling and lower complexity

# Learning

- Inductive learning – based on past examples
- Learn a function *h()* that approximates real function *f(x)* on examples *x*
- Balance complexity of hypothesis with fidelity to the examples
  - Minimize $\alpha\, E(h,D) + (1-\alpha)\, C(h)$

# Learning Algorithms

Many out there but the basics are:

- K nearest neighbors
  - Instance-based
  - Ignores global information
- Naïve Bayes
  - Strong independence assumption
  - Scales well due to assumptions
  - Needs normalization when dealing with unseen feature values
- Decision Trees
  - Easy to understand its hypothesis
  - Decides feature based on information gain

# Training and testing

- Judge induced h()'s quality by using a *test set*
- Training and test set must be separate; otherwise *peeking* occurs
- Modeling noise or specifics of the training data can lead to *overfitting*
  - Use pruning to remove parts of the hypothesis that aren't justifiable

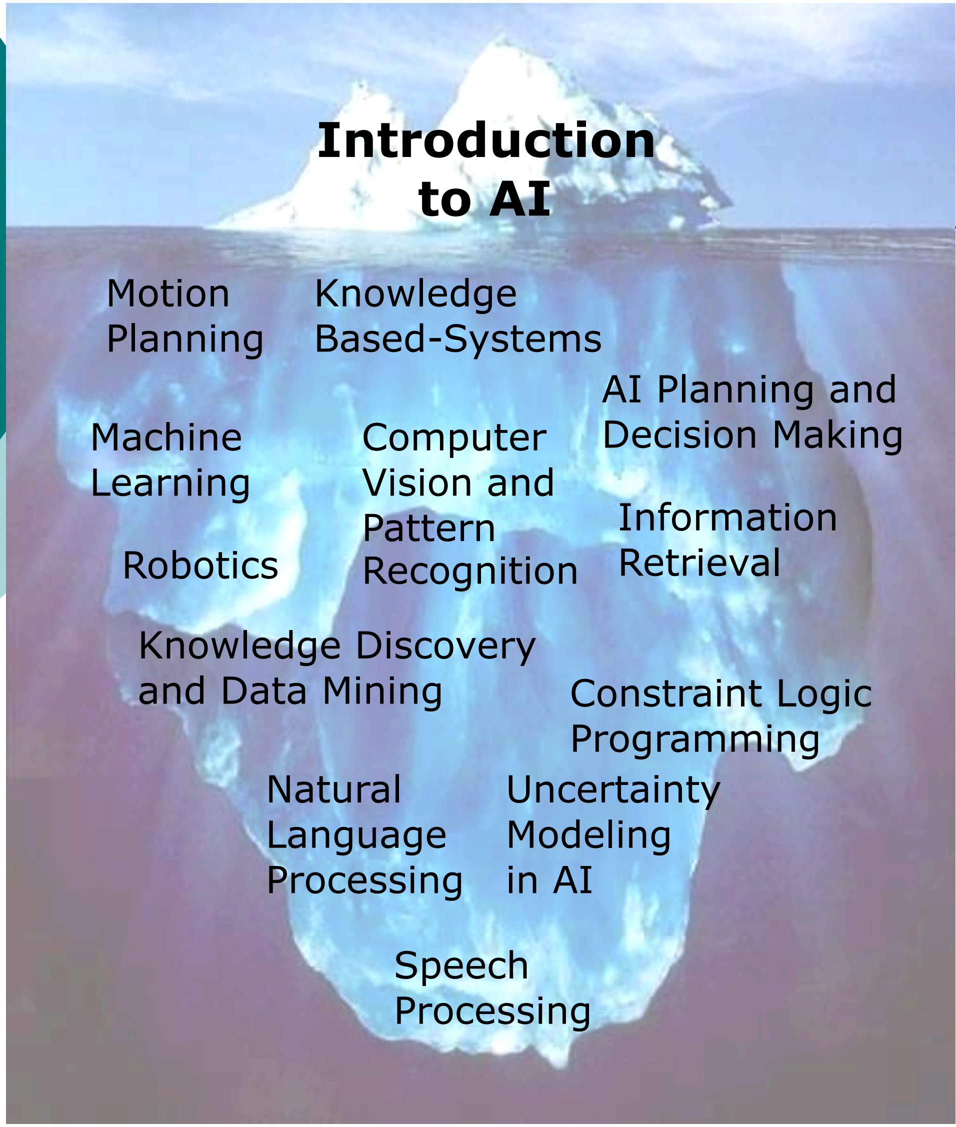
# Natural Language Processing

- Like many fields of advanced AI
  - Inverse problems: Understanding vs. Generation / Synthesis
  - How to wreck a nice beach: Ambiguity – the key problem
- Levels in processing
  - Syntactic, Discourse, Semantic, Pragmatic

# Natural Language Processing

- Syntactic Parsing
  - Bottom Up, Top Down, CYK Chart Parsing
- Augmented Grammars
  - Long distance agreement: Case and number
  - Use variables to capture parameters
  - Unification as a viable method
- Current NLP uses statistical techniques

# Where to go from here?




- Just the tip of the iceberg
- Many advanced topics
  - Introduced only a few
  - Textbook can help in exploration of AI

# That's it

- Thanks for your attention over the semester
- See you at the exam!