Artificial Intelligence Programming

Propositional Logic

Cindi Thompson

Department of Computer Science

University of San Francisco

Introduction

- So far, we've talked about search, which is a means of considering alternative possibilities.
  - The way in which problem states were represented was typically pretty straightforward.
- The other aspect of many AI problems involves representing possible states.
- Our choice of representation influences:
  - The problems our agent is able to solve.
  - The sorts of environments an agent can deal with.
  - The complexity of search
  - The sophistication of our agent.

Example

- Suppose we wanted to develop a program to keep track of students
  - Make sure they're on track to graduate
  - Ensure that their financial aid is handles properly
  - Make sure they've met all requirements for classes
  - etc
- How might we represent all of this information?

Example, continued

- Arrays of variables
- Classes and subclasses
- Relational DB schema
- These all have pros and cons as implementational choices, but may not be the easiest way to construct a model.
- We might want to build our model in such a way that facts can be *inferred*.
- A logical representation can help us do this.
  - Might still be stored internally as a database.

Example, continued

- Suppose that I want to say that Joe is either a freshman or a sophomore.
- Students must take 110 before taking 112. Joe has taken 112. Has he taken 110?
- All students are either male or female. Joe is not female. What is Joe's gender?
- All classes meet on Wednesdays, unless they are lab classes or graduate classes. 110 is a lab class. Does it meet on Wednesday?

Knowledge Representation

- Choices we'll look at include:
  - Logic-based approaches
    - Propositional logic
    - First-order logic
    - Ontologies
  - Logic is a flexible, well-understood, powerful, versatile way to represent knowledge.
  - Often fits with the way human experts describe their world
  - Facts are either true or false
  - Has a hard time dealing with (some sorts of) uncertainty.

Knowledge Representation

- Choices we'll look at include:
  - Probabilistic approaches
    - Bayesian reasoning
    - Bayesian networks
    - Decision theory
  - Probabilistic reasoning works well in domains with uncertainty.
  - Inference can be more complex
  - Requires more prior knowledge to use effectively.
  - Representational power typically more limited.

Declarative vs. Procedural

- Knowledge representation entails a shift from a procedural way of thinking about agents to a declarative way.
- Procedural: Behavior is encoded directly in agent program. Focus is on algorithms.
- Declarative: Agent knowledge is represented as sentences. Focus is on data.
  - Goal: Separate knowledge about the problem from the algorithm used to select actions.

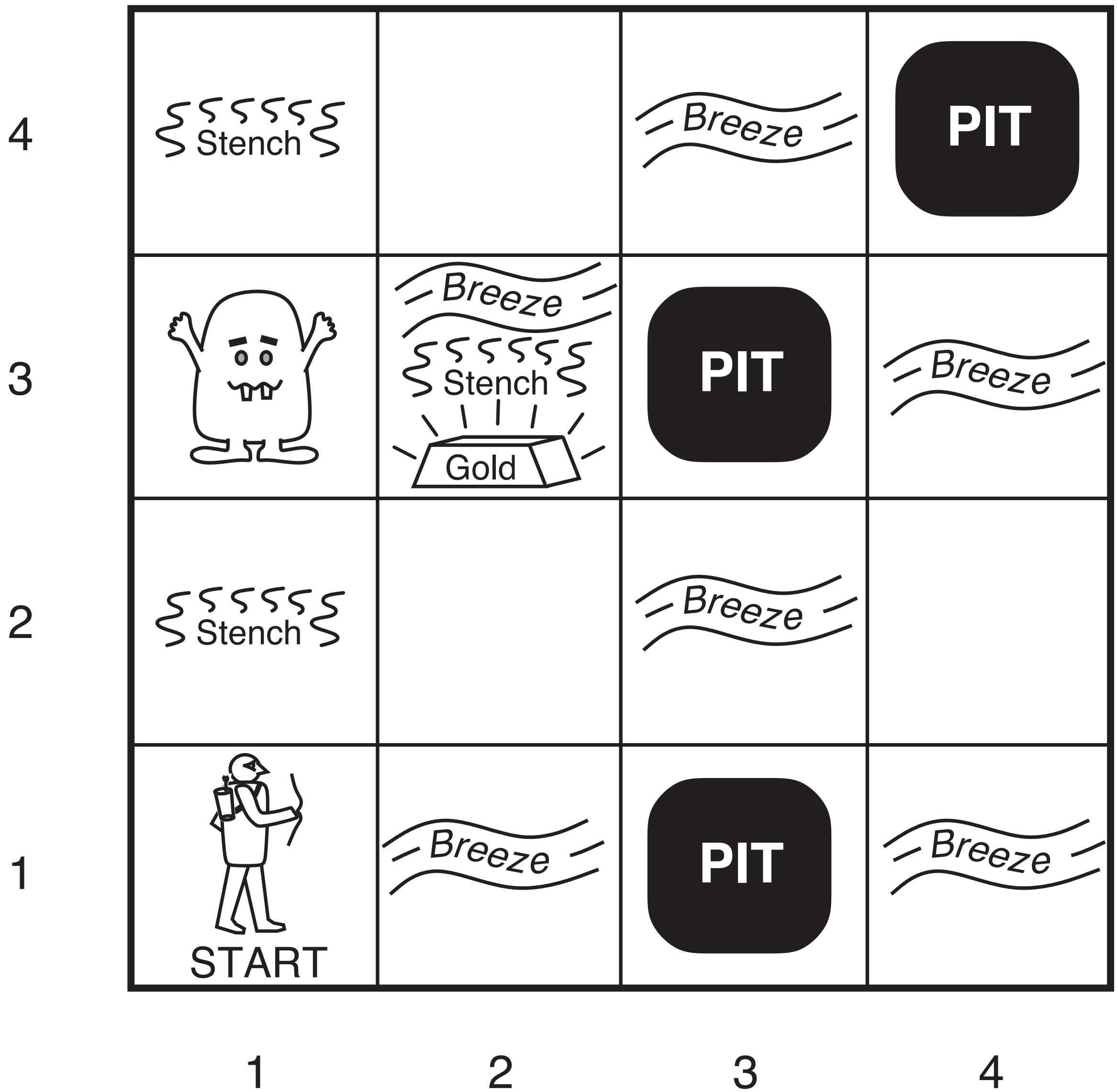
Declarative vs. Procedural

- Agents maintain a knowledge base that allows them to reason about a problem.
- Knowledge is represented as facts and relations
- Inference is typically performed automatically.
- This is sometimes called programming at the knowledge level.
- Specify facts known by an agent, along with goals.
- Programming focus is on encoding knowledge

Wumpus World

- R & N use the Wumpus World as an example domain.
- Environment: 4x4 grid of rooms.
  - Gold in one room, wumpus in another
  - Pits in some rooms
- Actions: Move forward, turn left, turn right, shoot arrow, grab gold.
- Sensors: Perceive stench, perceive breeze, perceive gold, sense wall, hear wumpus death.
- Goal: maximize performance, which means finding gold quickly without encountering the wumpus or falling into a pit.

Wumpus World



Knowledge base

- A knowledge base is composed of *sentences* that assert facts about the world.
  - What's the difference between a knowledge base and a database?
  - In principle, expressiveness and usage.
  - In practice, a knowledge base might be implemented using a database.
- Sentences describe:
  - Objects of interest in the world (wumpueses, gold, pits, rooms, agent)
  - Relationships between objects (agent is holding arrow)

Syntax and Semantics

- Syntax: Defines whether a sentence is properly constructed.
  - In arithmetic, $x + 2 = 5$ is syntactically correct, whereas $x+ = 3$ is not.
  - In a Python program, $timeElapsed = 3$ is syntactically correct, while $3 = timeElapsed$ is not.
- Semantics: Defines when a sentence is true or false.
  - The semantics of $x + 2 = 5$ are that this sentence is true in worlds where $x = 3$ and false otherwise.
  - Logical sentences must be true or false; no “degree of truth”.

Models

- Model: A model is an assignment of values to a subset of the variables of interest in our problem.
  - A model for the Vacuum cleaner world might indicate where the vacuum is, and which rooms are clean.
  - In the Wumpus World, a model would indicate the location of the pits, gold, agent, arrow, and wumpus.
- A model provides an agent with a *possible world*; one guess at how things might be.
- We'll often be interested in finding models that make a sentence true or false, or all the models that could be true for a given set of sentences.
- Models are similar to states.

Logic

- Entailment: Entailment is the idea that one sentence follows logically from another.
    - Written as: $a \models b$
    - Technically, this says: for all models where a is true, b is also true.
    - (think if-then)
    - $(a + 2 = 5) \models (a = 3)$
- Note that entailment is a property of a set of sentences, and not an instruction to an agent.

Inference

- A knowledge base plus a model allows us to perform inference.
  - For a given set of sentences, plus some assignment of values to variables, what can we conclude?
- Entailment tells us that *it is possible* to derive a sentence.
- Inference tells us *how* it is derived.
- An algorithm that only derives entailed sentences is said to be *sound*.
  - Doesn't make mistakes or conclude incorrect sentences.

Inference

- An inference algorithm that can derive all entailed sentences is *complete*.
  - If a sentence is entailed, a complete algorithm will eventually infer it.
  - If entailed sentences are goals, this is the same definition of complete we used for search.
  - That means we can think of inference as search, and use the algorithms we've already learned about.

Propositional Logic

- Propositional logic is a very simple logic.
  - Nice for examples
  - Computationally feasible.
  - Limited in representational power.
- Terms (R & N call these atomic sentences) consist of a single symbol that has a truth value.
  - $Room_{1,0}Clean$, $VacuumIn_{0,0}$

Propositional Logic

- a complex sentence is a set of terms conjoined with $\vee$, $\neg$, $\wedge$, $\Rightarrow$, $\Leftrightarrow$.
  - $Room_{1,0}Clean \wedge (Room_{0,0}Clean \vee Room_{0,0}Dirty)$
  - $Breeze_{1,1} \Rightarrow (Pit_{1,2} \vee Pit_{2,1})$

Propositional Logic

- Notice that propositional logic does not have any way to deal with classes of objects.
  - We can't concisely say "For any room, if there is a breeze, then there is a pit in the next room."
  - To say "At least one room is dirty" requires us to list all possibilities.
    - We don't have functions or predicates.
  - There's a computational tradeoff involved; if we're careful about how we use propositions, we can do fast (polynomial-time) inference.
  - But, we're limited in what our agent can reason about.
  - Propositional logic is the logic underlying hardware design (Boolean logic)

More on predicates

- Often, people will replace atomic terms with simple predicates.
  - Replace $Room_{0,1}Clean$ with $Clean(Room_{0,1})$.
  - As it is, this is fine.
  - What we're missing is a way to talk about all the rooms that are clean without explicitly enumerating them.
    - We don't have *variables* or *quantifiers*
  - To do that, we need *first-order logic*

Notation

- $A \wedge B$ - AND. sentence is true if both A and B are true.
- $A \vee B$ - OR. Sentence is true if either A or B (or both) are true.
- $\neg A$ - NOT. Sentence is true if A is false.
- $A \Rightarrow B$ - Implies. Sentence is true if A is false or B is true.
- $A \Leftrightarrow B$ - Equivalence. Sentence is true if A and B have the same truth value.

Propositional Logic - implication

- Implication is a particularly useful logical construct.
- The *sentence* $A \Rightarrow B$ is true if:
  - A is true and B is true.
  - A is false.
- Example: If it is raining right now, then it is cloudy right now.
- $A \Rightarrow B$ is equivalent to $\neg A \vee B$.
- Implication will allow us to perform inference.

Still more definitions

- Logical equivalence: Two sentences are logically equivalent if they are true for the same set of models.
  - $P \wedge Q$ is logically equivalent to $\neg(\neg P \vee \neg Q)$
- Validity (tautology): A sentence is valid if it is true for all models.
  - $A \vee \neg A$
- Contradiction: A sentence that is false in all models.
  - $A \wedge \neg A$

Still more definitions

- Satisfiability: A sentence is satisfiable if it is true for some model.
  - $Room_{0,0}Clean \vee Room_{0,1}Clean$ is true in some worlds.
  - Often our problem will be to find a model that makes a sentence true (or false).
  - A model that satisfies all the sentences we're interested in will be the goal or solution to our search.

Logical reasoning

- Logical reasoning proceeds by using existing sentences in an agent's KB to *deduce* new sentences.
- Deduction is guarateed to produce true sentences, assuming a sound mechanism is used.
- Rules of inference.
  - Modus Ponens
    - $A, A \Rightarrow B$, conclude B
  - And-Elimination
    - $A \wedge B$, conclude A.
  - Or-introduction
    - A, conclude $A \vee B$

Logical Reasoning

- Rules of inference.
  - Contraposition: $A \Rightarrow B$ can be rewritten as $\neg B \Rightarrow \neg A$
  - Double negative: $\neg(\neg A) = A$
  - Distribution
    - $A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C)$
    - $A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C)$
  - DeMorgan's theorem
    - $A \vee B$, rewrite as $\neg(\neg A \wedge \neg B)$
    - or $A \wedge B \Leftrightarrow \neg(\neg A \vee \neg B)$

Inference as Search

- We can then use good old search to perform inference and determine whether a sentence is entailed by a knowledge base.
- Basic idea: Begin with statements in our KB.
- Actions are applications of implication.
  - For example, say we know 1) $A \Rightarrow B$, 2) $B \Rightarrow C$, and 3) A.
  - One possible action is to apply Modus Ponens to 1 and 3 to conclude B.
  - We can then apply Modus Ponens again to conclude C.

Inference as Search

Options

- Breadth-first: what are all the possible conclusions from the original KB
- Depth-first: take one inference, then use it to make further inferences, and so on
- or somewhere in-between.

Successor function defines all applicable rules for a given knowledge base.

The result of this search is called a *proof*.

Example

- Begin with:
  - There is no pit in (1,1): $R_1 : \neg P_{1,1}$
  - A square has a breeze iff there is a pit in the neighboring square
  - $R_2 : B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
  - $R_3 : B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$ (and so on for all other squares)
- Assume the agent visits (1,1) and senses no breeze, but does sense a breeze in (2,1). Add:
  - $R_4 : \neg B_{1,1}$
  - $R_5 : B_{2,1}$

Example

- We can use biconditional elimination to rewrite R_2 as:
  - $R_6 : (B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$
- And-elimination on R_6 produces
  $R_7 : ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$
- Contraposition on R_7 gives us:
  $R_8 : \neg B_{1,1} \Rightarrow \neg(P_{1,2} \vee P_{2,1})$
- Modus Ponens with R_8 and R_4 produces
  $R_9 : \neg(P_{1,2} \vee P_{2,1})$
- DeMorgan's then gives us $R_{10} : \neg P_{1,2} \wedge \neg P_{2,1}$
- Our agent can conclude that there is no pit in (0,0), (1,2), or (2,1). It is not sure about (2,2)

Resolution

- The preceding rules are sound, but not necessarily complete.
- Also, search can be inefficient: there might be many operators that can be applied in a particular state.
- Luckily, there is a complete rule for inference (when coupled with a complete search algorithm) that uses a single operator.
- This is called *resolution*.
    - $A \vee B$ and $\neg A \vee C$ allows us to conclude $B \vee C$.
    - A is either true or not true. If A is true, then C must be true.
    - if A is false, then B must be true.
    - This can be generalized to clauses of any length.

Conjunctive Normal Form

- Resolution works with disjunctions.
- This means that our knowledge base needs to be in this form.
- Conjunctive Normal Form is a conjunction of clauses that are disjunctions.
- $(A \vee B \vee C) \wedge (D \vee E \vee F) \wedge (G \vee H \vee I) \wedge \ldots$
- Every propositional logic sentence can be converted to CNF.

Conjunctive Normal Form Recipe

1. Eliminate equivalence
   - $A \Leftrightarrow B$ becomes $(A \Rightarrow B) \wedge (B \Rightarrow A)$
2. Eliminate implication
   - $A \Rightarrow B$ becomes $\neg A \vee B$
3. Move $\neg$ inwards using double negation and DeMorgan's
   - $\neg(\neg A)$ becomes A
   - $\neg(A \wedge B)$ becomes $(\neg A \vee \neg B)$
4. Distribute nested clauses
   - $(A \vee (B \wedge C))$ becomes $(A \vee B) \wedge (A \vee C)$

Example

- $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
- Eliminating equivalence produces:
  - $(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$
- Removing implication gives us:
  - $(\neg B_{1,1} \vee (P_{1,2} \vee P_{2,1})) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1})$
- We then use DeMorgan's rule to move negation inwards:
  - $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$
- Finally, we distribute OR over AND:
  - $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$

Example

- Now we have clauses that can be plugged into a resolution theorem prover. (can break ANDs into separate sentences)
- They're less readable by a human, but more computationally useful.

Proof By Refutation

- Once your KB is in CNF, you can do resolution by refutation.
  - In math, this is called proof by contradiction
- Basic idea: we want to show that sentence A is true.
- Insert $\neg A$ into the KB and try to derive a contradiction.

Example

- Prove that there is not a pit in (1,2). $\neg P_{1,2}$
- Relevant Facts:
  - $R_{2a} : (\neg B_{1,1} \vee P_{1,2} \vee P_{2,1})$
  - $R_{2b} : (\neg P_{1,2} \vee B_{1,1})$
  - $R_{2c} : (\neg P_{2,1} \vee B_{1,1})$
  - $R_4 : \neg B_{1,1}$
- Insert $R_n : P_{1,2}$ into the KB

Example, continued

- Resolve R_n with R_{2b} to get: $R_6 : B_{1,1}$
- We already have a contradiction, since $R_4 : \neg B_{1,1}$
- Therefore, the sentence we inserted into the KB must be false.
- Most proofs take more than one step to get to a contradiction ...

From English to Logic

- If it rains, Joe brings his umbrella
- If Joe has an umbrella, he doesn't get wet
- If it doesn't rain, Joe doesn't get wet

Proofs

- Either Heather attended the meeting or she was not invited
- If the boss wanted Heather at the meeting, then she was invited
- Heather did not attend the meeting
- If the boss did not want Heather there, and the boss did not invite her, then she is going to be fired

Prove Heather is going to be fired.

Problem: Complexity

- Standard resolution-based theorem proving (and propositional inference in general) is exponentially hard.
- However, if we're willing to restrict ourselves a bit, the problem becomes (computationally) easy.

Solution: Horn clauses

- A *Horn* clause is a disjunction with at most one positive literal.
  - $\neg A \vee \neg B \vee \neg C \vee D$
  - $\neg A \vee \neg B$
- These can be rewritten as implications with one consequent.
  - $A \wedge B \wedge C \Rightarrow D$
  - $A \wedge B \Rightarrow False$
- Horn clauses are the basis of logic programming (sometimes called rule-based programming)

How to use a KB:

Forward Chaining

- Starts with a KB and continually applies Modus Ponens to derive all possible facts.
- This is sometimes called data-driven reasoning
- Start with domain knowledge and see what that knowledge tells you.
- This is very useful for discovering new facts or rules
- Less helpful for proving a specific sentence true or false
  - Search is not directed towards a goal

How to use a KB:

Backward Chaining

- starts with the goal and “works backward” to the start.
- Example: If we want to show that A is entailed, find a sentence whose consequent is A.
- Then try to prove that sentence’s antecendents.
- This is sometimes called query-driven reasoning.
- More effective at proving a particular query, since search is focused on a goal.
- Less likely to discover new and unknown information.
- Means-ends analysis is a similar sort of reasoning.
- Prolog uses backward chaining.

Strengths of Propositional Logic

- Declarative - knowledge can be separated from inference.
- Can handle partial information
- Can compose more complex sentences out of simpler ones.
- Sound and complete inference mechanisms (efficient for Horn clauses)

Weaknesses of Propositional logic

- Exponential increase in number of literals
- No way to describe relations between objects
- No way to quantify over objects.
- First-order logic is a mechanism for dealing with these problems.
- As always, there will be tradeoffs.
  - There's no free lunch!

Applications

- Propositional logic can work nicely in bounded domains
  - All objects of interest can be enumerated.
- Fast algorithms exist for solving SAT problems via model checking.
  - Search all models to find one that satisfies a sentence.
- Can be used for some scheduling and planning problems
  - Often, we'll use a predicate-ish notation as syntactic sugar.