First-Order Logic & Inference
AI Class 19 (Ch. 8.1–8.3, 9)

Material from Dr. Marie desJardins. Some material adopted from notes by Andreas Geyer-Schulz and Chuck Dyer.

Bookkeeping & Today

• HW5 out by 11:59pm
• Project designs Thursday
  • Not trivial to grade!
• Today:
  • A couple of midterm questions
  • Reasoning with logic-based agents
  • Logical inference
  • Model checking

<table>
<thead>
<tr>
<th>Score</th>
<th>Students</th>
</tr>
</thead>
<tbody>
<tr>
<td>A &gt; 70</td>
<td>15</td>
</tr>
<tr>
<td>B &gt; 60</td>
<td>12</td>
</tr>
<tr>
<td>C &gt; 50</td>
<td>2</td>
</tr>
<tr>
<td>D ≤ 50</td>
<td></td>
</tr>
</tbody>
</table>

Eliding +/- ...

These are very very approximate.
Midterm: State Spaces

• Describe the state space for the following puzzle:
  • You are given a grid of 5 color bars (rows). Each bar contains exactly 5 colors, with no duplications.
  • The colors on each bar can be changed by rotating the bar to the right, one step at a time.
  • The goal is to rotate each bar until every column contains every color, and no column contains the same color more than once.
  • This is an initial state – so, it’s a member of the state space.
  • What other states can you generate?
  • How many total states is that?

Midterm: State Spaces

• What is the difference between Nash Equilibrium and Pareto Optimality?
  • Nash equilibrium is when no player in a game can increase their payoff by unilaterally changing their actions.
  • Social or individual good?
  • Can you have >1 Nash equilibrium?
  • Pareto optimal is when it is not possible to make any player better off in the game without hurting another player.
  • Social or individual good?
Logical Agents for Wumpus World

Three (non-exclusive) agent architectures:

- **Reflex** agents
  - Have rules that classify situations, specifying how to react to each possible situation

- **Model-based** agents
  - Construct an internal model of their world

- **Goal-based** agents
  - Form goals and try to achieve them
A Typical Wumpus World

- The agent always starts in the field [1,1].
- The task of the agent is to find the gold, return to the field [1,1] and climb out of the cave.

A Simple Reflex Agent

- Rules to map percepts into observations:
  \( \forall b,g,u,c,t \ Percept([\text{Stench}, b, g, u, c], t) \rightarrow \text{Stench}(t) \)
  \( \forall s,g,u,c,t \ Percept([\text{Breeze}, s, g, u, c], t) \rightarrow \text{Breeze}(t) \)
  \( \forall s,b,u,c,t \ Percept([\text{Glitter}, s, b, Glitter, u, c], t) \rightarrow \text{AtGold}(t) \)

- Rules to select an action given observations:
  \( \forall t \ \text{AtGold}(t) \rightarrow \text{Action(Grab, t)} \)
A Simple Reflex Agent

• Some difficulties:

• Climb?
  • There is no percept that indicates the agent should climb out – position and holding gold are not part of the percept sequence

• Loops?
  • The percept will be repeated when you return to a square, which should cause the same response (unless we maintain some internal model of the world)

Representing Change

• Representing change in the world in logic can be tricky.

• One way is just to change the KB
  • Add and delete sentences from the KB to reflect changes
  • How do we remember the past, or reason about changes?

• **Situation calculus** is another way

• A **situation** is a snapshot of the world at some instant in time

• When the agent performs an action A in situation S1, the result is a new situation S2.
A situation is:
- A snapshot of the world
- At an interval of time
- During which nothing changes

Every true or false statement is made wrt. a situation
- Add situation variables to every predicate.
- \( at(Agent,1,1) \) becomes \( at(Agent,1,1,s0) \):
  \( at(Agent,1,1) \) is true in situation (i.e., state) \( s0 \).
Situation Calculus

- Alternatively, add a special 2nd-order predicate, \texttt{holds(f,s)} , that means “f is true in situation s.” E.g., \texttt{holds(at(Agent,1,1),s0)}
- Or: add a new function, \texttt{result(a,s)} , that maps a situation s into a new situation as a result of performing action a. For example, \texttt{result(forward, s)} is a function that returns the successor state (situation) to s
- Example: The action \texttt{agent-walks-to-location-y} could be represented by

\[(\forall x)(\forall y)(\forall s) \; (\text{at(Agent,x,s)} \land \neg \text{onbox(s)}) \rightarrow \text{at(Agent,y,result(walk(y),s))}\]

Deducing Hidden Properties

- From the perceptual information we obtain in situations, we can infer properties of locations

\[\forall l,s \; \text{at(Agent,l,s)} \land \text{Breeze(s)} \rightarrow \text{Breezy(l)}\]
\[\forall l,s \; \text{at(Agent,l,s)} \land \text{Stench(s)} \rightarrow \text{Smelly(l)}\]

- Neither Breezy nor Smelly need situation arguments because pits and Wumpuses do not move around
Deducing Hidden Properties II

• We need to write some rules that relate various aspects of a single world state (as opposed to across states)

• There are two main kinds of such rules:
  • Causal rules reflect assumed direction of causality:
    \[(\forall l1,l2,s) \text{At}(\text{Wumpus},l1,s) \land \text{Adjacent}(l1,l2) \rightarrow \text{Smelly}(l2)\]
    \[(\forall l1,l2,s) \text{At}(\text{Pit},l1,s) \land \text{Adjacent}(l1,l2) \rightarrow \text{Breezy}(l2)\]

• Systems that reason with causal rules are called model-based reasoning systems
Deducing Hidden Properties II

• We need to write some rules that relate various aspects of a single world state (as opposed to across states)

• There are two main kinds of such rules:
  • **Diagnostic rules** infer the presence of **hidden properties** directly from the percept-derived information. We have already seen two:
    \[
    \forall l,s \left[ \text{At(Agent,l,s)} \land \text{Breeze(s)} \rightarrow \text{Breezy(l)} \right]
    \]
    \[
    \forall l,s \left[ \text{At(Agent,l,s)} \land \text{Stench(s)} \rightarrow \text{Smelly(l)} \right]
    \]

Frames: A Data Structure

• A **frame** divides knowledge into **substructures** by representing “stereotypical situations.”

• Situations can be visual scenes, structures of physical objects,

• Useful for representing commonsense knowledge.
Representing Change: The Frame Problem

• **Frame axioms**: If property x doesn’t change as a result of applying action a in state s, then it stays the same.
  - On \((x, z, s) \land \text{Clear}(x, s) \rightarrow\)
    - On \((x, \text{table}, \text{Result}(\text{Move}(x, \text{table}), s)) \land\)
    - \neg \text{On}(x, z, \text{Result}(\text{Move}(x, \text{table}), s))
  - On \((y, z, s) \land y \neq x \rightarrow\) On \((y, z, \text{Result}(\text{Move}(x, \text{table}), s))
  - The proliferation of frame axioms becomes very cumbersome in complex domains

The Frame Problem II

• **Successor-state axiom**: General statement that characterizes every way in which a particular predicate can become true:
  - Either it can be made true, or it can already be true and not be changed:
    - On \((x, \text{table}, \text{Result}(a,s)) \leftrightarrow\)
      - [On \((x, z, s) \land \text{Clear}(x, s) \land a = \text{Move}(x, \text{table})]\) \lor
      - [On \((x, \text{table}, s) \land a \neq \text{Move}(x, z)\]
  - In complex worlds with longer chains of action, even these are too cumbersome
  - Planning systems use special-purpose inference to reason about the expected state of the world at any point in time during a multi-step plan
Qualification Problem

• Qualification problem:
  • How can you possibly characterize every single effect of an action, or every single exception that might occur?
  • When I put my bread into the toaster, and push the button, it will become toasted after two minutes, unless…
    • The toaster is broken, or…
    • The power is out, or…
    • I blow a fuse, or…
    • A neutron bomb explodes nearby and fries all electrical components, or…
    • A meteor strikes the earth, and the world we know it ceases to exist, or…

Ramification Problem

• How do you describe every effect of every action?
  • When I put my bread into the toaster, and push the button, the bread will become toasted after two minutes, and…
    • The crumbs that fall off the bread onto the bottom of the toaster over tray will also become toasted, and…
    • Some of the aforementioned crumbs will become burnt, and…
    • The outside molecules of the bread will become “toasted,” and…
    • The inside molecules of the bread will remain more “breadlike,” and…
    • The toasting process will release a small amount of humidity into the air because of evaporation, and…
    • The heating elements will become a tiny fraction more likely to burn out the next time I use the toaster, and…
    • The electricity meter in the house will move up slightly, and…
Knowledge Engineering!

• Modeling the “right” conditions and the “right” effects at the “right” level of abstraction is very difficult

• Knowledge engineering (creating and maintaining knowledge bases for intelligent reasoning) is a field

• Many researchers hope that automated knowledge acquisition and machine learning tools can fill the gap:
  * Our intelligent systems should be able to learn about the conditions and effects, just like we do.
  * Our intelligent systems should be able to learn when to pay attention to, or reason about, certain aspects of processes, depending on the context.

Preferences Among Actions

• A problem with the Wumpus world knowledge base that we have built so far is that it is difficult to decide which action is best among a number of possibilities.

• For example, to decide between a forward and a grab, axioms describing when it is OK to move to a square would have to mention glitter.

• This is not modular!

• We can solve this problem by separating facts about actions from facts about goals. This way our agent can be reprogrammed just by asking it to achieve different goals.
Preferences Among Actions

• The first step is to describe the desirability of actions independent of each other.

• In doing this we will use a simple scale: actions can be Great, Good, Medium, Risky, or Deadly.

• Obviously, the agent should always do the best action it can find:
  \((\forall a, s) \text{ Great}(a, s) \rightarrow \text{Action}(a, s)\)
  \((\forall a, s) \text{ Good}(a, s) \land \neg (\exists b) \text{ Great}(b, s) \rightarrow \text{Action}(a, s)\)
  \((\forall a, s) \text{ Medium}(a, s) \land (\neg (\exists b) \text{ Great}(b, s) \lor \text{ Good}(b, s)) \rightarrow \text{Action}(a, s)\)
  ...

Preferences Among Actions

• We use this action quality scale in the following way.

• Until it finds the gold, the basic strategy for our agent is:
  • Great actions include picking up the gold when found and climbing out of the cave with the gold.
  • Good actions include moving to a square that’s OK and hasn't been visited yet.
  • Medium actions include moving to a square that is OK and has already been visited.
  • Risky actions include moving to a square that is not known to be deadly or OK.
  • Deadly actions are moving into a square that is known to have a pit or a Wumpus.
Goal-Based Agents

- Once the gold is found, it is necessary to change strategies. So now we need a new set of action values.

- We could encode this as a rule:
  - $(\forall s) \text{Holding(Gold,s)} \rightarrow \text{GoalLocation([1,1]),s}$

- We must now decide how the agent will work out a sequence of actions to accomplish the goal.

- Three possible approaches are:
  - Inference: good versus wasteful solutions
  - Search: make a problem with operators and set of states
  - Planning: coming soon!

Logical Inference

Chapter 9
Model Checking

• Given KB, does sentence S hold?
  
  **Quick review: What’s a KB? What’s a sentence?**

• Basically **generate and test:**
  - Generate all the possible models
  - Consider the models M in which KB is TRUE
  - If ∀M S, then S is **provably true**
  - If ∀M ¬S, then S is **provably false**
  - Otherwise (∃M1 S ∧ ∃M2 ¬S): S is **satisfiable** but neither provably true or provably false

Efficient Model Checking

• Davis-Putnam algorithm (DPLL): Generate-and-test model checking with:
  - Early termination (short-circuiting of disjunction and conjunction)
  - Pure symbol heuristic: Any symbol that only appears negated or unnegated must be FALSE/TRUE respectively.
    - Can “conditionalize” based on instantiations already produced
  - Unit clause heuristic: Any symbol that appears in a clause by itself can immediately be set to TRUE or FALSE

• WALKSAT: Local search for satisfiability:
  - Pick a symbol to flip (toggle TRUE/FALSE), either using min-conflicts or choosing randomly
  - …or you can use any local or global search algorithm!
Reminder: Inference Rules for FOL

- Inference rules for **propositional logic** apply to **FOL**
  - Modus Ponens, And-Introduction, And-Elimination, …

- New (sound) inference rules for use with quantifiers:
  - Universal elimination
  - Existential introduction
  - Existential elimination
  - Generalized Modus Ponens (GMP)

Automating FOL Inference with Generalized Modus Ponens
Automated Inference for FOL

- Automated inference using FOL is harder than PL
  - Variables can take on an infinite number of possible values
    - From their domains, anyway
    - This is a reason to do careful KR!
  - So, potentially infinite ways to apply Universal Elimination

- **Gödel’s Completeness Theorem** says that FOL entailment is only semidecidable*
  - If a sentence is **true** given a set of axioms, can prove it
  - If the sentence is **false**, then there is no guarantee that a procedure will ever determine this
  - **Inference may never halt**

*The “halting problem”

Generalized Modus Ponens (GMP)

- Apply modus ponens reasoning to generalized rules
- Combines And-Introduction, Universal-Elimination, and Modus Ponens
  - From \( P(c) \) and \( Q(c) \) and \( (\forall x)(P(x) \land Q(x)) \rightarrow R(x) \) derive \( R(c) \)
- General case: **Given**
  - atomic sentences \( P_1, P_2, ..., P_N \)
  - implication sentence \( (Q_1 \land Q_2 \land ... \land Q_N) \rightarrow R \)
    - \( Q_1, ..., Q_N \) and \( R \) are atomic sentences
  - substitution \( \text{subst}(\theta, P_i) = \text{subst}(\theta, Q_i) \) for \( i=1,...,N \)
  - Derive new sentence: \( \text{subst}(\theta, R) \)
Generalized Modus Ponens (GMP)

- **Derive new sentence:** subst(θ, R)

- **Substitutions**
  - subst(θ, α) denotes the result of applying a set of substitutions, defined by θ, to the sentence α
  - A substitution list θ = {v₁/t₁, v₂/t₂, ..., vᵣ/tᵣ} means to replace all occurrences of variable symbol vᵢ by term tᵢ
  - Substitutions are made in left-to-right order in the list
  - subst({x/IceCream, y/Ziggy}, eats(y,x)) = eats(Ziggy, IceCream)

Horn Clauses

- A Horn clause is a sentence of the form:
  
  $$(\forall x) P_1(x) \land P_2(x) \land ... \land P_n(x) \rightarrow Q(x)$$

  where:
  - there are 0 or more $P_i$s and 0 or 1 $Q$s
  - the $P_i$s and $Q$ are positive (non-negated) literals

  - Equivalently: $P_1(x) \lor P_2(x) \ldots \lor P_n(x)$ where the $P_i$ are all atomic and at most one of them is positive

  - Prolog is based on Horn clauses

  - Horn clauses represent a subset of the set of sentences representable in FOL
Horn Clauses II

• Special cases
  • $P_1 \land P_2 \land \ldots \land P_n \rightarrow Q$
  • $P_1 \land P_2 \land \ldots \land P_n \rightarrow \text{false}$
  • $\text{true} \rightarrow Q$

• These are not Horn clauses:
  • $p(a) \lor q(a)$
  • $(P \land Q) \rightarrow (R \lor S)$

Forward Chaining

• Proofs start with the given axioms/premises in KB, deriving new sentences using GMP until the goal/query sentence is derived

• This defines a forward-chaining inference procedure because it moves “forward” from the KB to the goal [eventually]

• Inference using GMP is complete for KBs containing only Horn clauses
Forward Chaining Example

- KB:
  - allergies(X) → sneeze(X)
  - cat(Y) ∧ allergic-to-cats(X) → allergies(X)
  - cat(Felix)
  - allergic-to-cats(Lise)

- Goal:
  - sneeze(Lise)

Forward Chaining Algorithm

procedure FORWARD-CHAIN($KB$, $p$)
  if there is a sentence in $KB$ that is a renaming of $p$ then return
  Add $p$ to $KB$
  for each $(p_1 ∧ \ldots ∧ p_n ⇒ q)$ in $KB$ such that for some $i$, $\text{Unify}(p_i, p) = \emptyset$ succeeds do
    FIND-AND-INFER($KB$, $[p_1, \ldots, p_i-1, p_i\theta, p_{i+1}, \ldots, p_n], q, \emptyset$)
  end

procedure FIND-AND-INFER($KB$, premises, conclusion, $\emptyset$)
  if premises = [] then
    FORWARD-CHAIN($KB$, SUBS($\emptyset$, conclusion))
  else for each $p'$ in $KB$ such that $\text{Unify}(p', \text{SUBS}(\emptyset, \text{FIND-AND-INFER}(KB, \text{FIND-AND-INFER}(KB, \text{FIND-AND-INFER}(KB, \ldots)))) = \emptyset$ do
    FIND-AND-INFER($KB$, REST(premises), conclusion, COMPOSE(\emptyset, $\emptyset_2$))
  end
Backward Chaining

- **Backward-chaining** deduction using GMP
  - *Complete* for KBs containing *only Horn clauses*.

- **Proofs:**
  - Start with the goal query
  - Find rules with that conclusion
  - Prove each of the antecedents in the implication
  - Keep going until you reach premises!

Avoid loops
- Is new subgoal already on goal stack?
- Avoid repeated work: has subgoal already been proved true already failed?

Backward Chaining Example

- **KB:**
  - allergies(X) → sneeze(X)
  - cat(Y) ∧ allergic-to-cats(X) → allergies(X)
  - cat(Felix)
  - allergic-to-cats(Lise)

- **Goal:**
  - sneeze(Lise)
Backward Chaining Algorithm

function Backward Chaining Algorithm

function Backward Chaining Algorithm

Forward vs. Backward Chaining

- FC is data-driven
  - Automatic, unconscious processing
  - E.g., object recognition, routine decisions
  - May do lots of work that is irrelevant to the goal
- BC is goal-driven, appropriate for problem-solving
  - Where are my keys? How do I get to my next class?
  - Complexity of BC can be much less than linear in the size of the KB
Completeness of GMP

- GMP (using forward or backward chaining) is complete for KBs that contain only Horn clauses.
- It is not complete for simple KBs that contain non-Horn clauses.
- The following entail that $S(A)$ is true:
  - $(\forall x) P(x) \rightarrow Q(x)$
  - $(\forall x) \neg P(x) \rightarrow R(x)$
  - $(\forall x) Q(x) \rightarrow S(x)$
  - $(\forall x) R(x) \rightarrow S(x)$
- If we want to conclude $S(A)$, with GMP we cannot, since the second one is not a Horn clause.
- It is equivalent to $P(x) \lor R(x)$.