AI Class 21 (Ch. 10.1-10.2, 10.4.2-10.4.4)

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

Bookkeeping

- Bookkeeping
  - HW5 due 11/21 @ 11:59pm
  - Project phase 1 code due 11/28 @ 11:59pm
  - Designs today
Today

• Quick review
  • Logical agents
  • Situation calculus
  • Forward and backward chaining

• World’s fastest KR&R

• Planning
  • What is planning?
  • Approaches to planning
    • GPS / STRIPS
    • Situation calculus formalism [revisited]
    • Partial-order planning

Last Time: Agents

• Logical agents
  • Reflex: rules map directly from percepts $\rightarrow$ beliefs or percepts $\rightarrow$ actions
    $\forall b, g, u, c, t$ Percept([Stench, b, g, u, c], t) $\rightarrow$ Stench(t)
    $\forall t$ AtGold(t) $\rightarrow$ Action(Grab, t)
  • Model-based: construct a model (set of t/f beliefs about sentences) as they learn; map from models $\rightarrow$ actions
    Action(Grab, t) $\rightarrow$ HaveGold(t)
    HaveGold(t) $\rightarrow$ Action(RetraceSteps, t)
  • Goal-based: form goals, then try to accomplish them

Wumpus percepts:
[Stench, Breeze, Glitter, Bump, Scream]
Last Time: Situations

- Representing a dynamic world
  - Situations ($s_0...s_n$): the world in situation 0-n
    \[ \text{Teaching(DrM,s)} \] — today,10:10,whenNotSick, ...
  - Add ‘situation’ argument to statements
    \[ \text{AtGold(t,s)} \]
  - Or, add a ‘holds’ predicate that says ‘sentence is true in this situation’
    \[ \text{holds(At[2,1], s)} \]
  - Or, add a result(action, situation) function that takes an action and situation, and returns a new situation
    \[ \text{results(Action(goNorth), s) } \rightarrow s_1 \]

Last Time: Goal-Based Agents

- Once the gold is found, need new goals!
  - So, need a new set of actions.

- Encoded as a rule:
  \[ (\forall s) \text{Holding(Gold,s) } \rightarrow \text{GoalLocation([1,1],s)} \]

- How does the agent find a sequence of actions for goal?

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

sneeze(Lise) ← infer truth of (query)

• Forward Chaining: apply rules

  cat(Y) ∧ allergic-cats(X) → allergies(X) ∧ cat(Felix)

  →

  cat(Felix) ∧ allergic-cats(X) → allergies(X) ∧ allergic-cats(Lise)

  →

  allergies(Lise) ∧ allergies(X) → sneeze(X)

  →

  sneeze(Lise) ✓

Knowledge Base

1. Allergies lead to sneezing.
   allergies(X) → sneeze(X)

2. Cats cause allergies if allergic to cats.
   cat(Y) ∧ allergic-cats(X) → allergies(X)

3. Felix is a cat.
   cat(Felix)

4. Lise is allergic to cats.
   allergic-cats(Lise)

Last Time: Inference

sneeze(Lise) ← query

• Backward Chaining: apply rules that end with the goal

  allergies(X) → sneeze(X) + sneeze(Lise)

  new query: allergies(Lise)?

  cat(Y) ∧ allergic-cats(X) → allergies(X) + allergies(Lise)

  new query: cat(Y) ∧ allergic-cats(Lise)?

  cat(Felix) + cat(Y) ∧ allergic-cats(Lise)

  new sentence: cat(Felix) ∧ allergic-cats(Lise) ✓
Knowledge Representation and Reasoning (KR&R)

Chapters 12.1-12.2, 12.5-12.6

Introduction

• Real knowledge representation and reasoning systems: several varieties
• These differ in their intended use, expressivity, features,…
• Some major families are
  • Logic programming languages
  • Theorem provers
  • Rule-based or production systems
  • Semantic networks
  • Frame-based representation languages
  • Databases (deductive, relational, object-oriented, etc.)
  • Constraint reasoning systems
  • Description logics
  • Bayesian networks
  • Evidential reasoning
Ontologies

• Representations of general concepts
• Usually represented as a type hierarchy
  • Sort of a special case of a semantic network (wait for it...)
• “Ontological engineering” is hard!
  • How do you create an ontology for a particular application?
  • How do you maintain an ontology for changing needs?
  • How do you merge ontologies from different fields?
  • How do you map across ontologies from different fields?

Upper Ontologies

• Highest-level categories: typically these might include:
  • Measurements
  • Objects and their properties (including fluent, or changing, properties)
  • Events and temporal relationships
  • Continuous processes
  • Mental events, processes; “beliefs, desires, and intentions”
• Also useful:
  • Subtype relationships
  • PartOf relationships
  • Composite objects
Semantic Networks

- A semantic network is a representation scheme that uses a graph of labeled nodes and labeled, directed arcs to encode knowledge.
  - Usually used to represent static, taxonomic, concept dictionaries
- Typically used with a special set of procedures to perform reasoning
  - e.g., inheritance of values and relationships
- Semantic networks were very popular in the ‘60s and ‘70s but are less frequently used today.
  - Often much less expressive than other KR formalisms
- The graphical depiction associated with a semantic network is a significant reason for their popularity.

Nodes and Arcs

- Arcs define binary relationships that hold between objects denoted by the nodes.
Semantic Networks

- The ISA (is-a) or AKO (a-kind-of) relation is often used to link instances to classes, classes to superclasses.
- Some links (e.g., hasPart) are inherited along ISA paths.
- The semantics of a semantic net can be informal or very formal:
  - Often defined at the implementation level.

---

Reification

- Non-binary relationships can be represented by “turning the relationship into an object.”
- This is an example of what logicians call “reification.”
- We might want to represent the generic give event as a relation involving three things: a giver, a recipient and an object, give(john, mary, book32).
Individuals and Classes

- Many semantic networks distinguish
  - Nodes representing individuals and those representing classes
  - The “subclass” relation from the “instance-of” relation

Inference by Inheritance

- One of the main kinds of reasoning done in a semantic net is the inheritance of values along the subclass and instance links.

- Semantic networks differ in how they handle the case of inheriting multiple different values.
  - All possible values are inherited, or
  - Only the “lowest” value or values are inherited
Multiple Inheritance

- A node can have any number of superclasses that contain it, enabling a node to inherit properties from multiple “parent” nodes and their ancestors in the network.

- These rules are often used to determine inheritance in such “tangled” networks where multiple inheritance is allowed:
  - If X<A<B and both A and B have property P, then X inherits A’s property.
  - If X<A and X<B but neither A<B nor B<Z, and A and B have property P with different and inconsistent values, then X does not inherit property P at all.
Nixon Diamond

- This was the classic example circa 1980.

From Semantic Nets to Frames

- Semantic networks morphed into Frame Representation Languages in the ‘70s and ‘80s.
- A frame is a lot like the notion of an object in OOP, but has more meta-data.
- A frame has a set of slots.
- A slot represents a relation to another frame (or value).
- A slot has one or more facets.
- A facet represents some aspect of the relation.
Planning

Chapter 10.1-10.2, 10.4.2-10.4.4

Planning Problem

• Find a sequence of actions that achieves a given goal when executed from a given initial world state.

• That is, given
  • A set of operator descriptions (possible primitive actions by the agent)
  • An initial state description
  • A goal state (description or predicate)

• Compute a plan, which is
  • A sequence of operator instances,
  • Executing them in initial state \(\rightarrow\) state satisfying description of goal-state

• Goals are usually a conjunction of things to be achieved
Planning vs. Problem Solving

- Planning and problem solving methods can often solve the same sorts of problems.
- Planning is more powerful
  - Because of the representations and methods used.
- States, goals, actions decomposed into sets of sentences
  - Usually in FOL.
- Search proceeds through *plan space* rather than *state space*
  - Usually – state space planners exist.
- Subgoals can be planned independently, reducing the complexity of the planning problem.

Typical Assumptions

- **Atomic time**: Each action is indivisible.
- **No concurrent actions** allowed
  - But, actions do not need to be in order in the plan.
- **Deterministic actions**
  - The result of actions are completely known.
  - No uncertainty in results.
- Agent is the **sole cause of change** in the world.
Typical Assumptions

- **Agent is omniscient**
  - Has complete knowledge of the state of the world
  - AKA…

- **Closed world assumption:**
  - Everything known to be true about the world is in the state description
  - Anything not in the state description is false

Blocks World

The **blocks world** is a micro-world that consists of a table, a set of blocks and a robot hand.

Some domain constraints:
- Only one block can be on another block
- Any number of blocks can be on the table
- The hand can only hold one block

Typical representation:
- `ontable(a)`
- `ontable(c)`
- `on(b,a)`
- `handempty`
- `clear(b)`
- `clear(c)`
Major Approaches

- GPS / STRIPS
- Situation calculus
- Partial order planning
- Hierarchical decomposition (HTN planning)
- Planning with constraints (SATplan, Graphplan)
- Reactive planning

General Problem Solver

- The General Problem Solver (GPS) system
  - An early planner (Newell, Shaw, and Simon)
  - GPS generates actions that reduce the difference between some state and a goal state
  - GPS uses Means-Ends Analysis
    - Compare what is given or known with what is desired
    - Select a reasonable thing to do next
    - Use a table of differences to identify procedures to reduce differences
  - GPS is a state space planner: operates on state space problems specified by an initial state, some goal states, and a set of operations