

For Wednesday

- Finish chapter 9
- Homework
 - Chapter 8, exercise 6

Program 1

- Any questions?

Higher-Order Logic

- FOPC is called first-order because it allows quantifiers to range over objects (terms) but not properties, relations, or functions applied to those objects.
- Second-order logic allows quantifiers to range over predicates and functions as well:
 - $\forall x \forall y [(x=y) \Leftrightarrow (\forall p p(x) \Leftrightarrow p(y))]$
 - Says that two objects are equal if and only if they have exactly the same properties.
 - $\forall f \forall g [(f=g) \Leftrightarrow (\forall x f(x) = g(x))]$
 - Says that two functions are equal if and only if they have the same value for all possible arguments.
- Third-order would allow quantifying over predicates of predicates, etc.

Alternative Notations

- Prolog:

cat(X) :- furry(X), meows(X), has(X, claws).

good_pet(X) :- cat(X); dog(X).

- Lisp:

(forall ?x

(implies (and (furry ?x) (meows ?x) (has ?x claws))
(cat ?x)))

A Kinship Domain

$\forall m,c \text{ Mother}(c) = m \Leftrightarrow \text{Female}(m) \wedge \text{Parent}(m, c)$

$\forall w,h \text{ Husband}(h, w) \Leftrightarrow \text{Male}(h) \wedge \text{Spouse}(h,w)$

$\forall x \text{ Male}(x) \Leftrightarrow \neg \text{Female}(x)$

$\forall p,c \text{ Parent}(p, c) \Leftrightarrow \text{Child}(c, p)$

$\forall g,c \text{ Grandparent}(g, c) \Leftrightarrow \exists p \text{ Parent}(g, p) \wedge \text{Parent}(p, c)$

$\forall x,y \text{ Sibling}(x, y) \Leftrightarrow x \neq y \wedge \exists p \text{ Parent}(p, x) \wedge \text{Parent}(p, y)$

$\forall x,y \text{ Sibling}(x, y) \Leftrightarrow \text{Sibling}(y, x)$

Axioms

- Axioms are the basic predicates of a knowledge base.
- We often have to select which predicates will be our axioms.
- In defining things, we may have two conflicting goals
 - We may wish to use a small set of definitions
 - We may use “extra” definitions to achieve more efficient inference

A Wumpus Knowledge Base

- Start with two types of sentence:
 - Percepts:
 - Percept([stench, breeze, glitter, bump, scream], time)
 - Percept([Stench, None, None, None, None], 2)
 - Percept(Stench, Breeze, Glitter, None, None], 5)
 - Actions:
 - Action(action, time)
 - Action(Grab, 5)

Agent Processing

- Agent gets a percept
- Agent **tells** the knowledge base the percept
- Agent **asks** the knowledge base for an action
- Agent **tells** the knowledge base the action
- Time increases
- Agent performs the action and gets a new percept
- Agent depends on the rules that use the knowledge in the KB to select an action

Simple Reflex Agent

- Rules that map the current percept onto an action.
- Some rules can be handled that way:

`action(grab,T) :- percept([S, B, glitter, Bump, Scr],T).`

- Simplifying our rules:

`stench(T) :- percept([stench, B, G, Bu, Scr],T).`

`breezy(T) :- percept([S, breeze, G, Bu, Scr], T).`

`at_gold(T) :- percept([S, B, glitter, Bu, Scr], T).`

`action(grab, T) :- at_gold(T).`

- How well can a reflex agent work?

Situation Calculus

- A way to keep track of change
- We have a **state** or **situation** parameter to every predicate that can vary
- We also must keep track of the resulting situations for our actions
- Effect axioms
- Frame axioms
- Successor-state axioms

Frame Problem

- How do we represent what is and is not true and how things change?
- Reasoning requires keeping track of the state when it seems we should be able to ignore what does not change

Wumpus Agent's Location

- Where agent is:
 $\text{at}(\text{agent}, [1,1], s_0)$.
- Which way agent is facing:
 $\text{orientation}(\text{agent}, s_0) = 0$.
- We can now identify the square in front of the agent:
 $\text{location_toward}([X, Y], 0) = [X+1, Y]$.
 $\text{location_toward}([X, Y], 90) = [X, Y+1]$.
- We can then define adjacency:
 $\text{adjacent}(\text{Loc1}, \text{Loc2}) :- \text{Loc1} = \text{location_toward}(\text{L2}, \text{D})$.

Changing Location

at(Person, Loc, result(Act,S)) :-

(Act = forward,

Loc = location_ahead(Person, S),

\+wall(loc))

; (at(Person, Loc, S), A \= forward).

- Similar rule required for orientation that specifies how turning changes the orientation and that any other action leaves the orientation the same

Deducing Hidden Properties

breezy(Loc) :- at(agent, Loc, S), breeze(S).

Smelly(Loc) :- at(agent, Loc, S), Stench(S).

- Causal Rules

smelly(Loc2) :-

at(wumpus, Loc1, S), adjacent(Loc1, Loc2).

breezy(Loc2) :-

at(pit, Loc1, S), adjacent(Loc1, Loc2).

- Diagnostic Rules

ok(Loc2) :-

percept([none, none, G, U, C], T),

at(agent, Loc1, S), adjacent(Loc1, Loc2).

Preferences Among Actions

- We need some way to decide between the possible actions.
- We would like to do this apart from the rules that determine what actions are possible.
- We want the desirability of actions to be based on our goals.

Handling Goals

- Original goal is to find and grab the gold
- Once the gold is held, we want to find the starting square and climb out
- We have three primary methods for finding a path out
 - Inference (may be very expensive)
 - Search (need to translate problem)
 - Planning (which we'll discuss later)

Wumpus World in Practice

- Not going to use situation calculus
- Instead, just maintain the current state of the world
- Advantages?
- Disadvantages?

Inference in FOPC

- As with propositional logic, we want to be able to draw logically sound conclusions from our KB
- Soundness:
 - If we can infer A from B , B entails A .
 - If $B \vdash A$, then $B \models A$
- Complete
 - If B entails A , then we can infer A from B
 - If $B \models A$, then $B \vdash A$

Inference Methods

- Three styles of inference:
 - Forward chaining
 - Backward chaining
 - Resolution refutation
- Forward and backward chaining are sound and can be reasonably efficient but are incomplete
- Resolution is sound and complete for FOPC, but can be very inefficient

Inference Rules for Quantifiers

- The inference rules for propositional logic also work for first order logic
- However, we need some new rules to deal with quantifiers
- Let $\text{SUBST}(q, a)$ denote the result of applying a substitution or binding list q to the sentence a .

$$\text{SUBST}(\{x/\text{Tom}, y/\text{Fred}\}, \text{Uncle}(x,y)) = \text{Uncle}(\text{Tom}, \text{Fred})$$

Universal Elimination

- Formula:

$$\forall v a \vdash \text{SUBST}(\{v/g\}, a)$$

- Constraints:

– for any sentence, a , variable, v , and ground term, g

- Example:

$$\forall x \text{ Loves}(x, \text{FOPC}) \vdash \text{Loves}(\text{Califf}, \text{FOPC})$$

Existential Elimination

- Formula:

$\$v a \vdash \text{SUBST}(\{v/k\}, a)$

- Constraints:

– for any sentence, a , variable, v , and constant symbol, k , that **doesn't** occur elsewhere in the KB (**Skolem constant**)

- Example:

$\exists x (\text{Owns}(\text{Mary}, x) \wedge \text{Cat}(x)) \vdash$

$\text{Owns}(\text{Mary}, \text{MarysCat}) \wedge \text{Cat}(\text{MarysCat})$

Existential Introduction

- Formula:

$a \vdash \exists v \text{ SUBST}(\{g/v\}, a)$

- Constraints:

– for any sentence, a , variable, v , that does not occur in a , and ground term, g , that does occur in a

- Example:

$\text{Loves}(\text{Califf}, \text{FOPC}) \vdash \exists x \text{ Loves}(x, \text{FOPC})$

Sample Proof

1) $\forall x,y(\text{Parent}(x, y) \wedge \text{Male}(x) \Rightarrow \text{Father}(x,y))$

2) $\text{Parent}(\text{Tom}, \text{John})$

3) $\text{Male}(\text{Tom})$

Using Universal Elimination from 1)

4) $\forall y(\text{Parent}(\text{Tom}, y) \wedge \text{Male}(\text{Tom}) \Rightarrow \text{Father}(\text{Tom}, y))$

Using Universal Elimination from 4)

5) $\text{Parent}(\text{Tom}, \text{John}) \wedge \text{Male}(\text{Tom}) \Rightarrow \text{Father}(\text{Tom}, \text{John})$

Using And Introduction from 2) and 3)

6) $\text{Parent}(\text{Tom}, \text{John}) \wedge \text{Male}(\text{Tom})$

Using Modes Ponens from 5) and 6)

7) $\text{Father}(\text{Tom}, \text{John})$

Generalized Modus Ponens

- Combines three steps of “natural deduction” (Universal Elimination, And Introduction, Modus Ponens) into one.
- Provides direction and simplification to the proof process for standard inferences.
- Generalized Modus Ponens:

$p_1', p_2', \dots, p_n', (p_1 \wedge p_2 \wedge \dots \wedge p_n \Rightarrow q) \vdash \text{SUBST}(\theta, q)$

where θ is a substitution such that for all i

$\text{SUBST}(\theta, p_i') = \text{SUBST}(\theta, p_i)$

Example

1) $\forall x,y(\text{Parent}(x,y) \wedge \text{Male}(x) \Rightarrow \text{Father}(x,y))$

2) $\text{Parent}(\text{Tom},\text{John})$

3) $\text{Male}(\text{Tom})$

$\theta = \{x/\text{Tom}, y/\text{John}\}$

4) $\text{Father}(\text{Tom},\text{John})$

Canonical Form

- In order to use generalized Modus Ponens, all sentences in the KB must be in the form of Horn sentences:

$$\forall v_1, v_2, \dots, v_n p_1 \wedge p_2 \wedge \dots \wedge p_m \Rightarrow q$$

- Also called Horn clauses, where a clause is a disjunction of literals, because they can be rewritten as disjunctions with at most one non-negated literal.

$$\forall v_1, v_2, \dots, v_n \neg p_1 \vee \neg p_2 \vee \dots \vee \neg p_n \vee q$$

Horn Clauses

- Single positive literals (facts) are Horn clauses with no antecedent.
- Quantifiers can be dropped since all variables can be assumed to be universally quantified by default.
- Many statements can be transformed into Horn clauses, but many cannot (e.g. $P(x) \vee Q(x)$, $\neg P(x)$)

Unification

- In order to match antecedents to existing literals in the KB, we need a pattern matching routine.
- $\text{UNIFY}(p,q)$ takes two atomic sentences and returns a substitution that makes them equivalent.
- $\text{UNIFY}(p,q)=\theta$ where $\text{SUBST}(\theta,p)=\text{SUBST}(\theta,q)$
- θ is called a unifier

Unification Examples

$\text{UNIFY}(\text{Parent}(x,y), \text{Parent}(\text{Tom}, \text{John})) = \{x/\text{Tom}, y/\text{John}\}$

$\text{UNIFY}(\text{Parent}(\text{Tom},x), \text{Parent}(\text{Tom}, \text{John})) = \{x/\text{John}\}$

$\text{UNIFY}(\text{Likes}(x,y), \text{Likes}(z,\text{FOPC})) = \{x/z, y/\text{FOPC}\}$

$\text{UNIFY}(\text{Likes}(\text{Tom},y), \text{Likes}(z,\text{FOPC})) = \{z/\text{Tom}, y/\text{FOPC}\}$

$\text{UNIFY}(\text{Likes}(\text{Tom},y), \text{Likes}(y,\text{FOPC})) = \text{fail}$

$\text{UNIFY}(\text{Likes}(\text{Tom},\text{Tom}), \text{Likes}(x,x)) = \{x/\text{Tom}\}$

$\text{UNIFY}(\text{Likes}(\text{Tom},\text{Fred}), \text{Likes}(x,x)) = \text{fail}$

Same Variable

- Exact variable names used in sentences in the KB **should** not matter.
- But if **Likes(x,FOPC)** is a formula in the KB, it does not unify with **Likes(John,x)** but does unify with **Likes(John,y)**
- We can standardize one of the arguments to UNIFY to make its variables unique by renaming them.

Likes(x,FOPC) \rightarrow Likes(x_1 , FOPC)

UNIFY(Likes(John,x),Likes(x_1 ,FOPC)) = { x_1 /John, x/FOPC }

Which Unifier?

- There are many possible unifiers for some atomic sentences.
 - $\text{UNIFY}(\text{Likes}(x,y),\text{Likes}(z,\text{FOPC})) =$
 - $\{x/z, y/\text{FOPC}\}$
 - $\{x/\text{John}, z/\text{John}, y/\text{FOPC}\}$
 - $\{x/\text{Fred}, z/\text{Fred}, y/\text{FOPC}\}$
 -
- UNIFY should return the most general unifier which makes the least commitment to variable values.