

For Friday

- Read Chapter 9, sections 1-4
- Homework
 - Chapter 7, exs 8-9

Program 1

- Any questions?

Syntax for First-Order Logic

Sentence \rightarrow AtomicSentence | Sentence Connective Sentence

| Quantifier Variable Sentence | \neg Sentence | (Sentence)

AtomicSentence \rightarrow Predicate(Term, Term, ...) | Term=Term

Term \rightarrow Function(Term, Term, ...) | Constant | Variable

Connective \rightarrow \vee | \wedge | \Rightarrow | \Leftrightarrow

Quantifier \rightarrow \exists | \forall

Constant \rightarrow A | John | Car1

Variable \rightarrow x | y | z | ...

Predicate \rightarrow Brother | Owns | ...

Function \rightarrow father-of | plus | ...

Terms

- Objects are represented by terms:
 - Constants: Block1, John
 - Function symbols: father-of, successor, plus
- An n-ary function maps a tuple of n terms to another term: father-of(John), succesor(0), plus(plus(1,1),2)
- Terms are simply names for objects.
- Logical functions are not procedural as in programming languages. They do not need to be defined, and do not really return a value.
- Functions allow for the representation of an infinite number of terms.

Predicates

- Propositions are represented by a predicate applied to a tuple of terms. A predicate represents a property of or relation between terms that can be true or false:
 - Brother(John, Fred), Left-of(Square1, Square2)
 - GreaterThan(plus(1,1), plus(0,1))
- In a given interpretation, an n-ary predicate can be defined as a function from tuples of n terms to {True, False} or equivalently, a set of tuples that satisfy the predicate:
 - {<John, Fred>, <John, Tom>, <Bill, Roger>, ...}

Sentences in First-Order Logic

- An atomic sentence is simply a predicate applied to a set of terms.
 - $\text{Owns}(\text{John}, \text{Car1})$
 - $\text{Sold}(\text{John}, \text{Car1}, \text{Fred})$
- Semantics is True or False depending on the interpretation, i.e. is the predicate true of these arguments.
- The standard propositional connectives ($\vee \wedge \Rightarrow \Leftrightarrow$) can be used to construct complex sentences:
 - $\text{Owns}(\text{John}, \text{Car1}) \vee \text{Owns}(\text{Fred}, \text{Car1})$
 - $\text{Sold}(\text{John}, \text{Car1}, \text{Fred}) \Rightarrow \neg \text{Owns}(\text{John}, \text{Car1})$
- Semantics same as in propositional logic.

Quantifiers

- Allow statements about entire collections of objects
- Universal quantifier: $\forall x$
 - Asserts that a sentence is true for all values of variable x
 - $\forall x \text{ Loves}(x, \text{FOPC})$
 - $\forall x \text{ Whale}(x) \Rightarrow \text{Mammal}(x)$
 - $\forall x (\forall y \text{ Dog}(y) \Rightarrow \text{Loves}(x,y)) \Rightarrow (\forall z \text{ Cat}(z) \Rightarrow \text{Hates}(x,z))$
- Existential quantifier: \exists
 - Asserts that a sentence is true for at least one value of a variable x
 - $\exists x \text{ Loves}(x, \text{FOPC})$
 - $\exists x (\text{Cat}(x) \wedge \text{Color}(x, \text{Black}) \wedge \text{Owns}(\text{Mary}, x))$
 - $\exists x (\forall y \text{ Dog}(y) \Rightarrow \text{Loves}(x,y)) \wedge (\forall z \text{ Cat}(z) \Rightarrow \text{Hates}(x,z))$

Use of Quantifiers

- Universal quantification naturally uses implication:
 - $\forall x \text{ Whale}(x) \wedge \text{Mammal}(x)$
 - Says that everything in the universe is both a whale and a mammal.
- Existential quantification naturally uses conjunction:
 - $\exists x \text{ Owns}(\text{Mary},x) \Rightarrow \text{Cat}(x)$
 - Says either there is something in the universe that Mary does not own or there exists a cat in the universe.
 - $\forall x \text{ Owns}(\text{Mary},x) \Rightarrow \text{Cat}(x)$
 - Says all Mary owns is cats (i.e. everything Mary owns is a cat). Also true if Mary owns nothing.
 - $\forall x \text{ Cat}(x) \Rightarrow \text{Owns}(\text{Mary},x)$
 - Says that Mary owns all the cats in the universe. Also true if there are no cats in the universe.

Nesting Quantifiers

- The order of quantifiers of the same type doesn't matter:
 - $\forall x \forall y (\text{Parent}(x,y) \wedge \text{Male}(y) \Rightarrow \text{Son}(y,x))$
 - $\exists x \exists y (\text{Loves}(x,y) \wedge \text{Loves}(y,x))$
- The order of mixed quantifiers does matter:
 - $\forall x \exists y (\text{Loves}(x,y))$
 - Says everybody loves somebody, i.e. everyone has someone whom they love.
 - $\exists y \forall x (\text{Loves}(x,y))$
 - Says there is someone who is loved by everyone in the universe.
 - $\forall y \exists x (\text{Loves}(x,y))$
 - Says everyone has someone who loves them.
 - $\exists x \forall y (\text{Loves}(x,y))$
 - Says there is someone who loves everyone in the universe.

Variable Scope

- The scope of a variable is the sentence to which the quantifier syntactically applies.
- As in a block structured programming language, a variable in a logical expression refers to the closest quantifier within whose scope it appears.
 - $\exists x (\text{Cat}(x) \wedge \forall x(\text{Black}(x)))$
 - The x in $\text{Black}(x)$ is universally quantified
 - Says cats exist and everything is black
- In a well-formed formula (wff) all variables should be properly introduced:
 - $\exists x P(y)$ not well-formed
- A ground expression contains no variables.

Relations Between Quantifiers

- Universal and existential quantification are logically related to each other:
 - $\forall x \neg \text{Love}(x, \text{Saddam}) \Leftrightarrow \neg \exists x \text{ Loves}(x, \text{Saddam})$
 - $\forall x \text{ Love}(x, \text{Princess-Di}) \Leftrightarrow \neg \exists x \neg \text{Loves}(x, \text{Princess-Di})$
- General Identities
 - $\forall x \neg P \Leftrightarrow \neg \exists x P$
 - $\neg \forall x P \Leftrightarrow \exists x \neg P$
 - $\forall x P \Leftrightarrow \neg \exists x \neg P$
 - $\exists x P \Leftrightarrow \neg \forall x \neg P$
 - $\forall x P(x) \wedge Q(x) \Leftrightarrow \forall x P(x) \wedge \forall x Q(x)$
 - $\exists x P(x) \vee Q(x) \Leftrightarrow \exists x P(x) \vee \exists x Q(x)$

Equality

- Can include equality as a primitive predicate in the logic, or require it to be introduced and axiomatized as the identity relation.
- Useful in representing certain types of knowledge:
 - $\exists x \exists y (\text{Owns}(\text{Mary}, x) \wedge \text{Cat}(x) \wedge \text{Owns}(\text{Mary}, y) \wedge \text{Cat}(y) \wedge \neg(x=y))$
 - Mary owns two cats. Inequality needed to ensure x and y are distinct.
 - $\forall x \exists y \text{ married}(x, y) \wedge \forall z (\text{married}(x, z) \Rightarrow y=z)$
 - Everyone is married to exactly one person. Second conjunct is needed to guarantee there is only one unique spouse.

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)