

Languages

- How are functional and imperative languages similar?
- How are they different?
 - Has anybody noticed that all the languages we'll study are non-ambiguous languages that have some language foundation (math, logic, English)

Prolog is All About

- Given a goal, find out if it's true or not.
- Prolog uses backward chaining (goal driven search)
- Many other production systems use forward chaining (data driven search)
- In Prolog, anything that cannot be derived is false. This is known as the "closed world" assumption.

An Example

- Say I want to confirm/deny that "I am a descendant of Thomas Jefferson".
 - He was born around 250 years ago and we assume 25 yr. per generation. We also assume that people general have more children than parents (say an average of 3 children)
 - The required path back would be around 10. If we assume 2 parents for each person, then there are 2^{10} possible states to search.
 - The required path forward would be around 3^{10}

Which one?

- Goal-driven search suggested if:
 - Goal-hypothesis is given in the problem and is easily formulated. Example: a theorem prover
 - There are a large number of rules that match the facts of the problem and thus produce an increasing number of conclusions or goals.
 - Problem data are not given but must be acquired by the problem solver. Example: a medical diagnosis system where diagnostic tests are ordered to confirm/deny a particular hypothesis

The other?

- Data-driven search is suggested if:
 - All or most of the data are given in the initial problem statement. Systems that analyze data fall into this category
 - There are a large number of possible goals, but only a few ways to use the facts. Example: DENDRAL, an expert system that finds the molecular structure of organic compounds based on their formula, mass spectrographic data, and knowledge of chemistry
 - It's difficult to form a goal or hypothesis

Logic and Prolog

- Lecture based off of material available from:
 - <http://computing.unn.ac.uk/staff/cgpb4/prologbook/node1.html>

How do Production Systems Work?

- Condition-action pairs: Each pair describes a single chunk of problem-solving knowledge
- Working memory: A description of the current state of the work in a reasoning process.
- Recognize-act cycle

How might Logic Systems Work?

- Rules: Describe a single chunk of problem solving knowledge
- Facts: A description of the current state of the work in a reasoning process.
- Questions: Also called goals, they start a recognize-act cycle

The Goal

- Logical languages:
 - Allow the creation of rules and facts
 - Allow goals or questions to be asked
 - Have language support for finding solutions to the goals
 - Allow you to insert facts into your database
 - You don't have to write the code to go through all actions and definitions to figure out what action occurs

Declarative vs. Procedural Programming

- Declarative:
 - The programmer must know the relationships between different entities
- Procedural:
 - The programmer must tell the computer how to do something

Prolog

- PROgramming in LOGic
- It's pretty synonymous with the term "logical language", although there are other languages such as Goedel and LPL

Note of Caution

- Prolog is difficult to master, because it doesn't have the same structures as most other programming languages

Universal Quantification

$\forall < \text{variables} > < \text{sentence} >$

- “Everyone in Computer Science is smart”

$$\forall x \text{ In}(x, \text{ComputerScience}) \Rightarrow \text{Smart}(x)$$
- $\forall x P$ is equivalent to the conjunction of instantiations of P

$$\begin{aligned} & \text{In}(\text{Matt}, \text{ComputerScience}) \Rightarrow \text{Smart}(\text{Matt}) \\ & \wedge \text{In}(\text{Dan}, \text{ComputerScience}) \Rightarrow \text{Smart}(\text{Dan}) \\ & \wedge \text{In}(\text{Tim}, \text{ComputerScience}) \Rightarrow \text{Smart}(\text{Tim}) \\ & \wedge \dots \end{aligned}$$

Universal Quantification Common Mistake

- Typically, \Rightarrow is the main connective with \forall .
- Common mistake: using \wedge as the main connective

$$\forall x \text{ In}(x, \text{ComputerScience}) \wedge \text{Smart}(x)$$
- Means “Everyone is in Computer Science and everyone is smart”

Existential Quantification

$\exists < \text{variables} > < \text{sentence} >$

- “Somebody in Computer Science is smart”

$$\exists x \text{ In}(x, \text{ComputerScience}) \wedge \text{Smart}(x)$$
- $\exists x P$ is equivalent to the disjunction of instantiations of P

$$\begin{aligned} & \text{In}(\text{Matt}, \text{ComputerScience}) \wedge \text{Smart}(\text{Matt}) \\ & \vee \text{In}(\text{Dan}, \text{ComputerScience}) \wedge \text{Smart}(\text{Dan}) \\ & \vee \text{In}(\text{Tim}, \text{ComputerScience}) \wedge \text{Smart}(\text{Tim}) \\ & \vee \dots \end{aligned}$$

Existential Quantification Common Mistake

- Typically, \wedge is the main connective with \exists .
- Common mistake: using \Rightarrow as the main connective

$$\exists x \text{ In}(x, \text{ComputerScience}) \Rightarrow \text{Smart}(x)$$
- Is true if there’s anyone who’s not in Computer Science!

More about Quantifiers

- $\forall x \forall y$ is the same as $\forall y \forall x$
- $\exists x \exists y$ is the same as $\exists y \exists x$
- $\exists x \forall y$ is not the same as $\forall y \exists x$
- There is a person who loves everyone in the world.

$$\exists x \forall y \text{ Loves}(x, y)$$
- Everyone in the world is loved by at least one person

$$\forall x \exists y \text{ Loves}(x, y)$$

Quantifier Duality

- Each can be expressed using the other

$$\begin{aligned} \forall x \text{ Eats}(x, \text{Mushrooms}) & \quad \neg \exists x \neg \text{Eats}(x, \text{Mushrooms}) \\ \exists x \text{ Eats}(x, \text{Vegetables}) & \quad \neg \forall x \neg \text{Eats}(x, \text{Vegetables}) \end{aligned}$$

Equality

- $term_1 = term_2$ is true under a given interpretation if and only if $term_1$ and $term_2$ refer to the same object
 - $1=2$ and $\forall x * (Sqrt(x), Sqrt(x)) = x$ are satisfiable
 - $2=2$ is valid
 - Definition of (full) *Sibling* in terms of *Parent*
 $\forall x, y \text{ Sibling}(x, y) \Leftrightarrow [-(x = y) \wedge \exists m, f \text{ } -(m = f) \wedge \text{Parent}(m, x) \wedge \text{Parent}(f, x) \wedge \text{Parent}(m, y) \wedge \text{Parent}(f, y)]$

A Couple of Prolog Basics

- Starting Prolog: pl
- You probably want to type your facts/rules in a file and then load the file:
 - [filename].
 - Note: filename is really filename.pl
- Stopping the Prolog interpreter:
 - halt.
- Help on a topic: help(topic).
- Edit a file test.pl: edit(test).
- List child to screen: listing(child).
- Trace descendant: trace(descendant).
- Switch tracing off: trace(descendant, -all).

English into First Order Logic

- Can you do these?
 - Every dog hates cats.
 - All purple mushrooms are tasty.
 - You can fool some of the people all of the time.
 - You can fool all of the people some of the time.
- [answers discussed in class]

How Do We Display Them in Prolog?

- Based on First order predicate logic
 - Uses a restricted version of clausal form called Horn clause form
 - $q_1 \wedge q_2 \wedge \dots \wedge q_n \rightarrow q_0$ where each q_i and q_0 is an **atomic sentence** and all variables are universally quantified.

Multiple Clauses

- A clause is basically a prolog sentence (and ends with a period)
 - Example: eats(snoopy, birdFood).
- How do we represent?: Snoopy eats bird food and wood.

Answer

- FOPL:
 - $\text{Eats}(\text{Snoopy}, \text{BirdFood}) \wedge \text{Eats}(\text{Snoopy}, \text{Wood})$
- Prolog:
 - eats(snoopy, birdFood),eats(snoopy, wood).

What about?

- The square root of 25 is 5 or it is going to rain.
- Perl, Java, and eLisp are all languages.

Rules

- If a dog is wet and dirty, then he is smelly
- FOPL:
 - $\forall x \text{Wet}(x) \wedge \text{Dirty}(x) \wedge \text{Dog}(x) \rightarrow \text{Smelly}(x)$
- Prolog:
 - $\text{smelly}(X) \text{ :- wet}(X), \text{dirty}(X), \text{dog}(X).$
- Only one goal is allowed at the head of the rule (before the :-)

Goals and Subgoals

- $\text{smelly}(X) \text{ :- wet}(X), \text{dirty}(X), \text{dog}(X).$
 - In prolog $\text{smelly}(X)$ is a goal and $\text{wet}(X)/\text{dirty}(X)/\text{dog}(X)$ are subgoals
 - Why?

Represent These Statements

- In predicate logic and Prolog
 - All animals eat jelly beans.
 - Everyone love's Frodo.
 - Snoopy likes to eat Jessica's furniture.

Translate

- Two people live in the same house if they have the same address.
- Two people are siblings if they have the same parents.
- Someone is happy if they're healthy, wealthy, or wise.
- A dog is happy if he's healthy, wealthy, or wise.

– Note that predicates are called "functors"

Proof Methods

- Model checking
 - Truth table enumerating
 - Heuristic search in model space
- Application of inference rules
 - Legitimate generation of new sentences from old
 - Proof: a sequence of inference rule applications. We can use inference rules as operators in a standard search algorithm.

Some Basic Prolog

- Facts
 - likes(joe, eLisp).
 - likes(john, eLisp).
 - likes(joe, mary).
 - likes(mary, books).
 - likes(mary, frankenstein).
 - likes(john, frankenstein).
- Questions
 - ?- likes(joe, eLisp).
 - Yes
 - ?- likes(mary, joe).
 - No
 - ?- likes(mary, frankenstein).
 - Yes
 - ?- dislikes(mary, joe).
 - No

Multiple Matches

- Facts
 - likes(joe, eLisp).
 - likes(john, eLisp).
 - likes(joe, mary).
 - likes(mary, books).
 - likes(mary, frankenstein).
 - likes(john, frankenstein).
- ?- likes(joe, X).
- If you type “;” prolog will search for more matches
- Prolog searches the facts from top to bottom

How Do We Ask?

- Is there anything that John and Mary both like?
- likes(john,X),likes(mary,X).
- In this case, Prolog will backtrack to find the right answer.

Rules

- How do we write?
 - Jack likes anyone who likes books.
 - likes(jack,X),likes(X,books).
 - likes(jack,likes(X,books)).
 - likes(jack, X) :- likes(X, books).
 - Jack likes anyone who likes books and fish.
 - Happy people are people who like themselves.
 - happy(Person) :- likes(Person,Person).

Recursion

- descendant(X,Y) :- child(X,Y).
- descendant(X,Y) :- child(X,Z), descendant(Z,Y).
- Avoid circular definitions:
 - child(X,Y) :- parent(Y,X).
 - parent(X,Y) :- child(Y,X).

Logically Correct, But...

- descendant(X,Y) :- child(X,Y).
- descendant(X,Y) :- descendant(Z,Y), child(X,Z).
- The above statement is logically correct, but may cause Prolog to go into an infinite loop, because of Prolog's left-to-right, depth-first order of evaluation.

Predicates and Operators

- =, \= <, <=, >, >=
- Arithmetic ops: +, -, *, etc.
- not

Prolog Variables

- Variables only refer to the same entity if they're inside the same clause.
- Note: logical variables are a bit different from other variables
 - X=1; X=2; will result in 2 in Java
 - X=1,X=2 won't work since the value X can only be one value

Prolog Functions

- Defining factorial:
 - You use the word "is" to do assignment
 - Example: A is 1*1.
 - The base case will be a separate rule from the regular case and will be stated separately.
 - Rather than a regular return type you normally have the return be a variable argument to the function