

Languages

How are functional and imperative languages similar?

How are they different?

Mathematics is all about constants

$x = x+1$ is false for all values of x

Values of "variables" never change

Prolog is All About

Given a goal, find out if it's true or not.

Prolog uses backward chaining (goal driven search)

In Prolog, anything that can not 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 $2^{10} = 1024$ possible states to search.

The required path forward would be around $3^{10} = 59049$

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 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 and the CLP(D) set of languages

Note of Caution

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

A Review of Logic

Taken from : Russell & Norvig's AI book

Language

What exists?

What states of knowledge?

Propositional logic

Facts

True/False/Unknown

First-order logic

Facts, objects, relations

True/False/Unknown

Temporal logic

Facts, objects, relations, times

True/False/Unknown

Probability theory

Facts

Degree of belief 0..1

Fuzzy logic

Degree of truth

Degree of belief 0..1

Propositional Logic Semantics

Each model specifies true/false for each proposition symbol

Rules for evaluation truth with respect to a model m :

$\neg S$ is true iff S is false

$S1 \wedge S2$ is true iff $S1$ is true AND $S2$ is true

$S1 \vee S2$ is true iff $S1$ is true OR $S2$ is true

$S1 \Rightarrow S2$

is true iff $S1$ is false OR $S2$ is true

is false iff $S1$ is true AND $S2$ is false

$S1 \Leftrightarrow S2$ is true iff $S1 \Rightarrow S2$ is true AND $S2 \Rightarrow S1$ is true

What We Would Like to Do

If p stands for “all functional languages are icky” and p is true then we would like to be able to prove that “eLisp is icky”.

This is where first order predicate logic (FOPL) comes in handy.

Predicate Logic Syntax

Constants: eLisp, 2, RIT

Predicates: Sister, >, Icky, ...

Functions: Sqrt, Eat, ...

Variables: x, y, a, b

Connectives: $\wedge \vee \neg \Rightarrow \Leftrightarrow$

Equality: =

Quantifiers: $\forall \exists$

Atomic Sentences

Atomic sentences:

- predicate(term 1 ,...,term n)
- term 1 =term 2

Term:

- function(term 1 ,...,term n)
- or constant
- or variable

Examples

Daughter(Lisa, Homer)

Representing in Prolog

Thinks(Justin, Icky(eLisp))

thinks(justin, icky(elisp)).

$\forall x$ smelly(x)

smelly(X).

Variables are in upper case

The variable `_` is a new anonymous variable

Values are in lower case

Sentence ends with a “.”

Both justin and eLisp are constants and X is a variable.

Predicates may not be variables.

Constants (atoms)

A constant is an atom or a number

A number is an integer or a real number

The rules for an atom in Prolog are quite complicated:

Quoted item: ‘anything but the single quote char’

Word: lower case letter followed by any letter, digit or `_`

Symbol: any number of {+, -, *, /, \, ^, <, >, =, ', ~, :, ., ?, @, #, \$, &}

Special item: any of {[], {}, ;, !, %}

So the following are all atoms:

likes_chocolate, fooX23, ++*++, ::=:, 'What Ho!'

Truth

Sentences are true with respect to a model and an interpretation

Models contain objects and relations among them

Interpretation specifies values referred to for

Constant symbols

Predicate symbols

Function symbols

An atomic sentence $\text{predicate}(\text{term}_1, \dots, \text{term}_n)$ is true iff the objects referred to by $\text{term}_1, \dots, \text{term}_n$ are in the relation referred to by predicate

Complex Sentences

Complex sentences are made from atomic sentences using connectives

$\neg S$

$S_1 \wedge S_2$

$S_1 \vee S_2$

$S_1 \Rightarrow S_2$

$S_1 \Leftrightarrow S_2$

Examples:

$\text{Sibling}(\text{Pico}, \text{HAL}) \Rightarrow \text{Sibling}(\text{HAL}, \text{Pico})$

$>(42,3) \vee \leq(42,3)$

$>(42,3) \wedge \neg >(42,3)$

Universal Quantification

\forall <variables> <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

$\text{In}(\text{Matt}, \text{ComputerScience}) \Rightarrow \text{Smart}(\text{Matt})$

$\wedge \text{In}(\text{Dan}, \text{ComputerScience}) \Rightarrow \text{Smart}(\text{Dan})$

$\wedge \text{In}(\text{Tim}, \text{ComputerScience}) \Rightarrow \text{Smart}(\text{Tim})$

$\wedge \dots$

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 <variables> <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

$\text{In}(\text{Matt}, \text{ComputerScience}) \wedge \text{Smart}(\text{Matt})$

$\vee \text{In}(\text{Dan}, \text{ComputerScience}) \wedge \text{Smart}(\text{Dan})$

$\vee \text{In}(\text{Tim}, \text{ComputerScience}) \wedge \text{Smart}(\text{Tim})$

$\vee \dots$

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!

Also true if anyone is Smart!

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

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

Quantifier Duality

Each can be expressed using the other

$\forall x \text{ Eats}(x, \text{Mushrooms})$

$\neg \exists x \neg \text{Eats}(x, \text{Mushrooms})$

or

$\exists x \text{ Eats}(x, \text{Mushrooms})$

$\neg \forall x \neg \text{Eats}(x, \text{Mushrooms})$

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:

`Eats(Snoopy, BirdFood) ^ Eats(Snoopy, 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:

`smelly(X) :- wet(X),dirty(X),dog(X)`

Only one goal is allowed at the head of the rule (before the :-)

Goals and Subgoals

smelly(X) :- wet(X),dirty(X),dog(X)

In prolog smelly(X) is a goal and wet(X)/dirty(X)/dog(X) are subgoals

Why?

Represent These Statements

In predicate logic and Prolog

All animals eat jelly beans.

Everyone loves 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.

Unification

Prolog uses unification to match terms

The unification of two terms proceeds by matching functions and arguments

If anything fails to match the unification fails

If a variable is matched to a term then a temporary binding of the variable is made

If a variable is matched to a variable then they are temporarily set to be equal

If unification is successful it returns all of the temporary variable bindings

For sound logic it is wrong to unify something with a container of the same thing

Unification of X and $S(X)$ would lead to an effective infinite object

This is called the "occurs check" and is generally disabled in Prolog for speed

Prolog Execution

First a database of clauses is read in or generated somehow

Then queries are run against this database

A query is executed by matching all terms of the query against the database and if all terms match then the resulting variable bindings are returned

To match a term it is unified with the head of each clause in the database

If unification fails the next clause in the database is considered

If there are no more clauses then the match fails

If the unification succeeds then the terms on the RHS of the clause are (recursively) matched against the database

If all terms match then the resulting bindings are returned

If any clause fails the bindings for the term are undone and the system backtracks and tries the next possibility

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.

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

List Notation

$[]$ is the empty list

$[X]$ is the list containing X

$[X|Y]$ is the list with X the head of the list and Y the rest of the list

$[X,Y,Z]$ is a three element list

$[X,Y,Z|W]$ is a list beginning with X , Y , and Z followed by the elements in the list W

Append in Prolog

Here are the rules for appending to a list

```
app([], Z, Z) .
```

```
app([Item|X], Y, [Item|Z]) :- app(X, Y, Z) .
```

Here are some uses of append

```
swm-kayrun[154]: pl
```

```
Welcome to SWI-Prolog (Version 3.1.0)
```

```
Copyright (c) 1993-1998 University of Amsterdam. All  
rights reserved.
```

```
For help, use ?- help(Topic) . or ?- apropos(Word) .
```

```
1 ?- [append].
```

```
append compiled, 0.00 sec, 1,140 bytes.
```

```
Yes
```

```
2 ?- app([1,2], [3,4], Z) .
```

```
Z = [1, 2, 3, 4] ;
```

```
No
```

```
3 ?- app(X, Y, [1,2,3]) .
```

```
X = []
```

```
Y = [1, 2, 3] ;
```

```
X = [1]
```

```
Y = [2, 3] ;
```

```
X = [1, 2]
```

```
Y = [3] ;
```

```
X = [1, 2, 3]
```

```
Y = [] ;
```

```
No
```

```
4 ?-
```