Prolog

CS367 ARTIFICIAL INTELLIGENCE

Chapter 9

Patricia J Riddle

Outline

Logic programming and declarative programs

Introduction to Prolog

Basic operation of the Prolog interpreter

Imperative Programming

Formulate a “how to compute it” recipe, e.g.:

to compute the sum of the list, iterate through the list adding each value to an accumulator variable

```
int sum(int[] list ) {
    int result = 0;
    for(int i=0; i<list.length; ++i) {
            result += list[i];
    }
return result;
}
```

OO Programing is a type of imperative programming
(you have to say “how to compute it”)

Functional Programming

Again formulate a “how to compute it” recipe

Probably will need to do recursive decomposition

(* The sum of the empty list is zero and the sum of the list with head h and tail t is h plus the sum of the tail. *)

```
fun sum([])= 0
  | sum(h::t) = h + sum(t);
```

Logic Programming

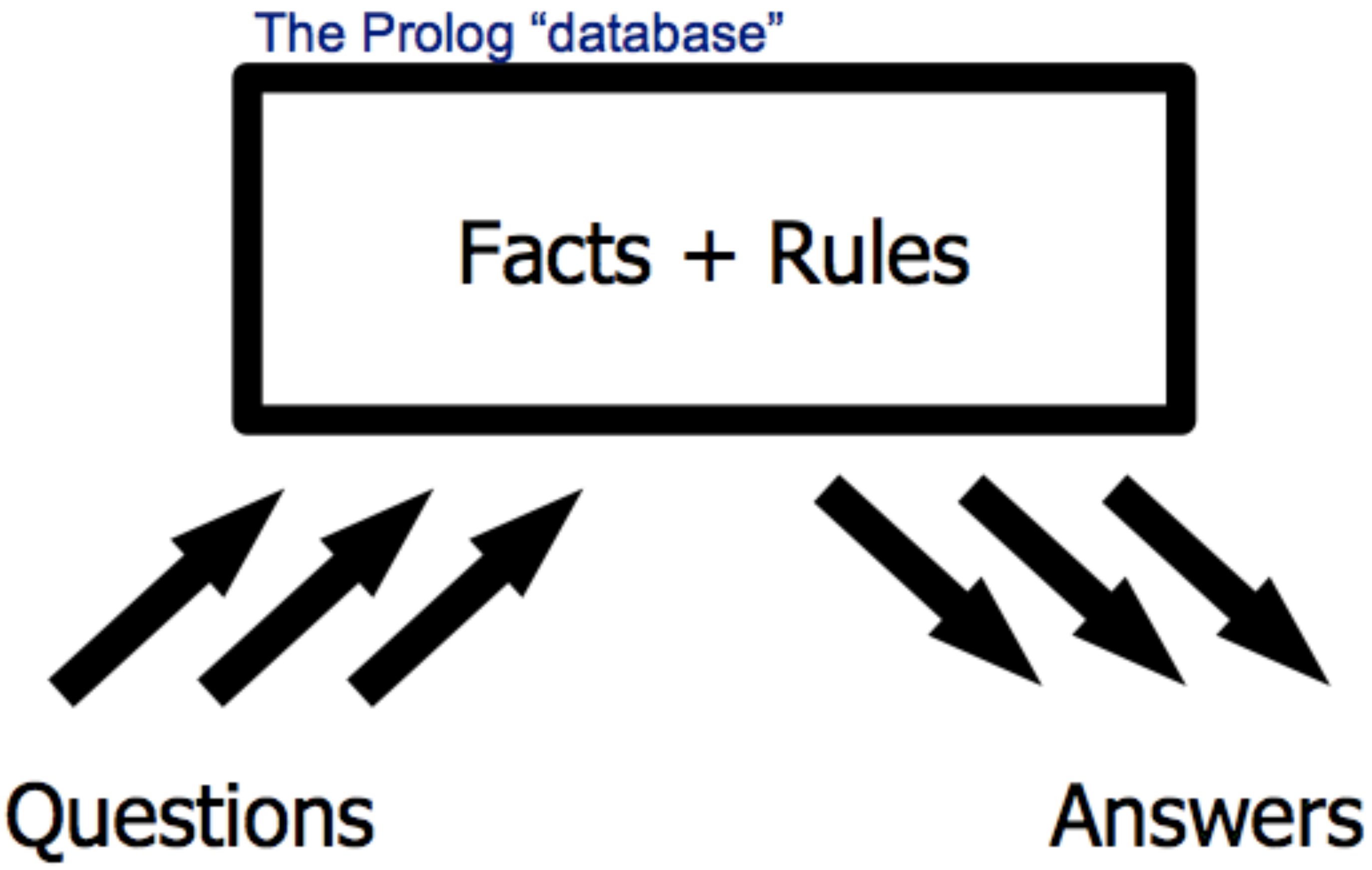
```
% the sum of the empty list is zero
sum([],0).

% the sum of the list with head H and
% tail T is N if the sum of the list T
% is M and N is M + H
sum([H|T],N) :- sum(T,M), N is M+H.
```

This is a declarative reading of a program

Not “how to compute” the result

Instead “this is true about the result”

Prolog Programs Answer Questions



The Prolog "database"

Facts + Rules

Questions

Answers

Facts

- Same predicate can take different arguments to produce distinct facts.

```
parent(abe, bob).
male(abe).
parent(ann, bob).
female(ann).
```

```
female(X).      (probably don’t want to do this!!!)
```

Variables are capitalized (or start with an “_” as in _x) and constants and predicates must begin with a small letter!!!

Rules

- a head (a single nonnegated predicate with arguments)
- a body (a set of predicates and associated arguments)

Head :- Body1, Body2

Body1 $\wedge$ Body2 $\Rightarrow$ Head

Rules

```
father(abe, bob) :- parent(abe, bob), male(abe).
mother(ann, bob) :- parent(ann, bob), female(ann).

father(X, Y) :- parent(X, Y), male(X).
mother(X, Y) :- parent(X, Y), female(X).
```

$$\forall x,y\ \text{parent}(x, y) \land \text{male}(x) \Rightarrow \text{father}(x,y)$$

$$\forall x,y\ \text{parent}(x, y) \land \text{female}(x) \Rightarrow \text{mother}(x,y)$$

- Prolog treats most variables in rules as universally quantified.

Existentially Quantified

```
brother(X, Z) :- parent(Y, X), parent(Y, Z),
male(X).
```

$\forall x,z\ \exists y\ parent(y, x) \wedge parent(y,z) \wedge male(x) \Rightarrow brother(x,z)$

- Prolog treats unbound variables in a rule's body as existentially quantified.

Recursive Rules

- The language also lets one define predicates recursively:

```
ancestor(X, Z) :- parent(X, Y), ancestor(Y, Z).
ancestor(X, Y) :- parent(X, Y).
```

- These rules specify ancestor in terms of parent and ancestor.

Queries

- A user runs a Prolog program by providing a query stated as one or more predicates with (partially) specified arguments.
- E.g., here are some queries using primitive kinship predicates:

```
?- parent(abe, bob).       … true.
?- parent(bob, abe).       … false.
?- parent(P, bob).         … P = abe.
```

The language also supports conjunctive queries:

```
?- parent(A, B), male(A), male(B).      … A = abe, B = bob;
                                          A = bob, B = dan.
```

- Prolog answers these queries by examining sets of facts and checking for consistent argument bindings.

More Queries

- Prolog queries can also refer to higher-level, defined predicates.
- E.g., here are some queries using defined kinship predicates:

```
?- father(abe, dan).                  … false.
?- brother(B, ema).                   … B = bob.
?- uncle(ann, N).                     … false.
?- grandfather(GF, GC).               … GF = abe, GC = dan.
?- ancestor(A, dan).                  … A = cat;
                                      … A = bob;
                                      … A = ann;
                                      … A = abe.
```

- These queries require more than simple lookup to answer; they depend upon multi-step reasoning.

How to enter a KB

```
2 ?- [user].
male(tom).
Warning: user://1:13:
Redefined static procedure male/1
Previously defined at /Users/prid013/Desktop/prolog:7
|: female(sally).
Warning: user://1:17:
Redefined static procedure female/1
Previously defined at /Users/prid013/Desktop/prolog:11
|:
% user://1 compiled 0.01 sec, -2 clauses
true.


3 ?-
```

- (ctrl-d to get out of user mode)

Our Knowledge Base – part 1

```
parent(abe, bob).
parent(ann, bob).
parent(bob, dan).
parent(cat, dan).
parent(ann, ema).
parent(mork, "ET").

male(abe).
male(bob).
male(dan).

female(ann).
female(cat).
```

Our Knowledge Base – part 2

```
father(X, Y) :- parent(X, Y), male(X).

mother(X, Y) :- parent(X, Y), female(X).

son(X, Y) :- parent(Y, X), male(X).

brother(X, Z) :- parent(Y, X), parent(Y, Z), male(X).

uncle(X, Z) :- brother(X, Y), parent(Y, Z), male(X).

grandfather(X, Z) :- father(X, Y), father(Y, Z).
grandfather(X, Z) :- father(X, Y), mother(Y, Z).

ancestor(X, Y) :- parent(X, Y).
ancestor(X, Z) :- parent(X, Y), ancestor(Y, Z).
```

How to load a knowledge base

```
1 ?- ['~/Desktop/prolog'].
% /Users/prid013/Desktop/prolog compiled 0.00 sec, 18 clauses
```

How to find out what is in your KB?

```
?- listing.
```

Complex Patterns with Negations

```
2 ?- parent(X, Y), not(male(X)).
X = ann,
Y = bob
X = cat,
Y = dan
X = ann,
Y = ema
X = mork,
Y = 'ET'.
```

```
3 ?- not(male(X)),parent(X,Y).
false.
```

- It is important to always have “not” after the variables are bound!!

List Structures in Prolog

```
single_list([a, b, c, d]).
three_sets([a, b, c], [d], [ ]).
more_sets([[a, b], [[c], d]]).
```

Prolog (this slide will appear again)

Appending two lists to produce a third:

```
append([],Y,Y).
append([X|L],Y,[X|Z]) :- append(L,Y,Z).
```

query: `append(A,B,[1,2]) ?`

answers:

```
A=[]     B=[1,2]

A=[1]    B=[2]

A=[1,2] B=[]
```

Reversing a List

```
reverse ([ ], X, X).
reverse ([X | Y], Z, W) :- reverse (Y, [X | Z], W).
```

Fun with Lists

```
append([1],[2],X).
append([1],X,[1,2]).

append(Z,Y,[1,2]).
append([1],X,Y).

append(X,Y,Z).

reverse(X,[],[1,2,3]).

reverse(X,Z,[1,2,3]).
reverse(X,[1,2,3],Z).
reverse([1,2,3],X,Z).

reverse(X,Y,Z).
```

Ordering Matters

```
reverse ([X | Y], Z, W) :- reverse (Y, [X | Z], W).
reverse ([ ], X, X).
```

- Is different than

```
reverse ([ ], X, X).
reverse ([X | Y], Z, W) :- reverse (Y, [X | Z], W).
```

Summary Remarks

- Prolog is a declarative language.
- You do not have to specify “How” things happen
- “not” can be a problem
- You can put variables anywhere
- Ordering matters