

Prolog I

1

Syllogisms

- “Prolog” is all about programming in logic.
 - Socrates is a man.
 - All men are mortal.
 - Therefore, Socrates is mortal.

2

Facts, rules, and queries

- Fact: Socrates is a man.
`man(socrates).`
- Rule: All men are mortal.
`mortal(X) :- man(X).`
- Query: Is Socrates mortal?
`mortal(socrates).`

3

Running Prolog I

- Create your "database" (program) in any editor
- Save it as *text only*, with a **.pl** extension
- Here's the complete "program":

```
man(socrates).  
mortal(X) :- man(X).
```

4

Running Prolog II

- Prolog is *completely interactive*.
- Begin by invoking the Prolog interpreter.
 - sicstus
- Then load your program.
 - consult('mortal.pl')
- Then, ask your question at the prompt:
 - mortal(socrates).
- Prolog responds:
 - Yes

5

On gl.umbc.edu

```
> sicstus
SICStus 3.7.1 ... Licensed to umbc.edu
| ?- consult('mortal.pl').
{consulting
 /home/faculty4/finin/cmsc/331/fall00/prolog/mortal.pl...}
{/home/faculty4/finin/cmsc/331/fall00/prolog/mortal.pl
 consulted, 0 msec 624 bytes}
yes
| ?- mortal(socrates).
yes
| ?- mortal(X).
X = socrates ?
yes
| ?-
```

6

Syntax I: Structures or Terms

- Example structures:
 - sunshine
 - man(socrates)
 - path(garden, south, sundial)
- <structure> ::=
 <name> | <name> (<arguments>)
- <arguments> ::=
 <argument> | <argument> , <arguments>

7

Syntax II: Base Clauses

- Base clauses are like simple facts.
- Example base clauses:
 - debug_on.
 - loves(john, mary).
 - loves(mary, bill).
- <base clause> ::= <structure> .

8

Syntax III: Nonbase Clauses

- Non-base clauses are like rules.
- Example non-base clauses:
 - mortal(X) :- man(X).
 - mortal(X) :- woman(X)
 - happy(X) :- healthy(X), wealthy(X), wise(X).
- <nonbase clause> ::=
 <structure> :- <structures> .
- <structures> ::=
 <structure> | <structures> , <structure>

9

Syntax IV: Predicates

- A predicate is a collection of clauses with the same *functor* and *arity*.
 loves(john, mary).
 loves(mary, bill).
 loves(chuck, X) :- female(X), rich(X).
- <predicate> ::=
 <clause> | <predicate> <clause>
- <clause> ::=
 <base clause> | <nonbase clause>

10

Syntax V: Programs

- A program is a collection of predicates.
- Predicates can be in any order.
- Predicates are used in the order in which they occur.

11

Syntax VI: Assorted details

- Variables begin with a capital letter or an underscore:
 X, Socrates, _result
- Atomic symbols do not begin with a capital letter:
 x, socrates
- Other atomic symbols must be enclosed in single quotes:
 - 'Socrates'
 - 'C:/My Documents/examples.pl'

12

Syntax VII: Assorted details

- In a quoted atom, a single quote must be quoted or backslashed: 'Can't, or won't?'
- /* Comments are like this */
- Prolog allows some infix operators, such as :- (turnstile) and , (comma). These are syntactic sugar for the functors ':' and ','.
- These are equivalent:
':-'(mortal(X), man(X)).
mortal(X) :- man(X).

13

Backtracking

- loves(chuck, X) :- female(X), rich(X).
- female(jane).
- female(mary).
- rich(mary).

Now, Suppose we ask: loves(chuck, X).

- female(X) = female(jane), X = jane.
- rich(jane) fails.
- female(X) = female(mary), X = mary.
- rich(mary) succeeds.

14

Additional answers

- female(jane).
- female(mary).
- female(susan).
- ?- female(X).
- X = jane ;
- X = mary
- Yes

15

Readings

- loves(chuck, X) :- female(X), rich(X).
- *Declarative reading:* Chuck loves X if X is female and rich.
- *Approximate procedural reading:* To find an X that Chuck loves, first find a female X, then check that X is rich.
- Declarative readings are almost always preferred.

16

Nonmonotonic logic

Prolog's facts and rules can be changed at any time.

```
assert(man(plato)).
assert((loves(chuck,X) :- female(X), rich(X))).
retract(man(plato)).
retract((loves(chuck,X) :- female(X), rich(X))).
```

17

Common problems

- Capitalization is *extremely* important!
- No space between a functor and its argument list:
`man(socrates)`, *not* `man (socrates)`.
- Don't forget the period! (But you can put it on the next line.)

18

A Simple Prolog Model

- Imagine prolog as a system which has a database composed of two components:
 - FACTS - statements about true relations which hold between particular objects in the world. For example:
parent(adam,able): adam is a parent of able
parent(eve,able): eve is a parent of able
male(adam): adam is male.
 - RULES - statements about true relations which hold between objects in the world which contain generalizations, expressed through the use of variables. For example, the rule
father(X,Y) :- parent(X,Y), male(X).
might express:
for any X and any Y, X is the father of Y if X is a parent of Y and X is male.

19

Nomenclature and Syntax

- A prolog rule is called a **clause**.
- A clause has a head, a neck and a body:
`father(X,Y) :- parent(X,Y), male(X) .`
head *neck* *body*
- the **head** is a rule's conclusion.
- The **body** is a rule's premise or condition.
- **note:**
 - read :- as IF
 - read , as AND
 - a . marks the end of input

20

Prolog Database

```
parent(adam,able)
parent(adam,cain)
male(adam)
...
```

Facts comprising the
“extensional database”

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

Rules comprising the
“intensional database”

21

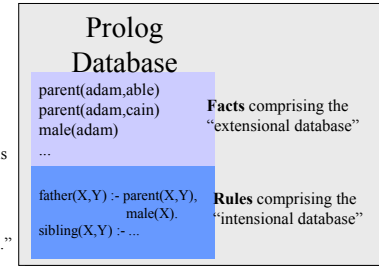
Extensional vs. Intensional

The terms *extensional* and *intensional* are borrowed from the language philosophers use for *epistemology*.

- *Extension* refers to whatever *extends*, i.e., “is quantifiable in space as well as in time”.
- *Intension* is an antonym of extension, referring to “that class of existence which may be quantifiable in time but not in space.”
- NOT *intentional* with a “t”, which has to do with “will, volition, desire, plan, ...”

For KBs and DBs we use

- *extensional* to refer to that which is explicitly represented (e.g., a fact), and
- *intensional* to refer to that which is represented abstractly, e.g., by a rule of inference.



Epistemology is “a branch of philosophy that investigates the origin, nature, methods, and limits of knowledge”

22

A Simple Prolog Session

```
| ?- assert(parent(adam,able)).      | ?- parent(X,able).
yes                                  X = adam ;
| ?- assert(parent(eve,able)).      X = eve ;
yes                                  no
| ?- assert(male(adam)).            | ?- parent(X,able) , male(X).
yes                                  X = adam ;
| ?- parent(adam,able).            no
yes
| ?- parent(adam,X).
X = able
yes
```

23

A Prolog Session

```
| ?- [user].
| female(eve).
| parent(adam,cain).
| parent(eve,cain).
| father(X,Y) :- parent(X,Y),
                 male(X).
| mother(X,Y) :- parent(X,Y),
                 female(X).
| ^Zuser consulted 356 bytes
0.0666673 sec.
yes
| ?- mother(Who,cain).
Who = eve
yes

| ?- mother(eve,Who).
Who = cain
yes
| ?- trace, mother(Who,cain).
(2) 1 Call: mother(_0,cain) ?
(3) 2 Call: parent(_0,cain) ?
(3) 2 Exit: parent(adam,cain)
(4) 2 Call: female(adam) ?
(4) 2 Fail: female(adam)
(3) 2 Back to: parent(_0,cain) ?
(3) 2 Exit: parent(eve,cain)
(5) 2 Call: female(eve) ?
(5) 2 Exit: female(eve)
(2) 1 Exit: mother(eve,cain)
Who = eve
yes
```

24

```

?- [user].
| sibling(X,Y) :-
| father(Pa,X),
| father(Pa,Y),
| mother(Ma,X),
| mother(Ma,Y),
| not(X=Y).

```

^Zuser consulted 152 bytes 0.0500008 sec.

yes

```

| ?- sibling(X,Y).

```

X = able

Y = cain ;

X = cain

Y = able ;

```

trace(sibling(X,Y))
(2) 1 Call: sibling_0_1)?
(3) 2 Call: father_65643_0)?
(4) 3 Call: parent_65643_0)?
(4) 3 Exit: parent(adam,able)
(5) 3 Call: male(adam)?
(5) 3 Exit: male(adam)
(5) 2 Exit: father(adam,able)
(6) 2 Call: father(adam_1)?
(7) 3 Call: parent(adam_1)?
(7) 3 Exit: parent(adam,able)
(8) 3 Call: male(adam)?
(8) 3 Exit: male(adam)
(8) 2 Exit: father(adam,able)
(9) 2 Call: parent_65644_able)?
(10) 3 Call: parent_65644_able)?
(10) 3 Exit: parent(adam,able)
(11) 3 Call: female(adam)?
(11) 3 Exit: female(adam)
(11) 3 Fail: female(adam)
(10) 3 Back to parent_65644_able)?
(10) 3 Exit: parent(veve,able)
(12) 3 Call: female(veve)?
(12) 3 Exit: female(veve)
(9) 2 Exit: mother(veve,able)
(13) 2 Call: mother(veve,able)?
(14) 3 Call: parent(veve,able)?
(14) 3 Exit: parent(veve,able)
(15) 3 Call: female(veve)?
(15) 3 Exit: female(veve)
(15) 3 Exit: mother(veve,able)
(16) 2 Call: not_able=able?
(17) 3 Call: able=able?
(17) 3 Exit: able=able
(16) 2 Back to: not_able=able?
(16) 2 Fail: not_able=able
(15) 3 Back to: female(veve)?
(15) 3 Exit: female(veve)
(14) 3 Back to: parent(veve,able)?
(14) 3 Exit: parent(veve,able)
(13) 2 Back to: mother(veve,able)?
(13) 2 Fail: mother(veve,able)
(12) 3 Back to: female(veve)?
(12) 3 Exit: female(veve)
(11) 3 Back to: parent_65644_able)?
(11) 3 Exit: male(adam)
(10) 3 Back to: parent_65644_able)?
(10) 3 Fail: parent_65644_able)
(9) 2 Back to: mother_65644_able)?
(9) 2 Fail: mother_65644_able)
(8) 3 Back to: male(adam)?
(8) 3 Fail: male(adam)
(7) 3 Back to: parent(adam_1)?
(7) 3 Exit: parent(adam,cain)
(6) 2 Call: male(adam)?
(6) 2 Exit: male(adam)
(5) 3 Back to: parent(adam,able)?
(5) 3 Exit: parent(adam,able)
(4) 3 Back to: parent_65643_0)?
(4) 3 Exit: parent(adam,cain)
(3) 2 Call: parent(adam,able)
(3) 2 Exit: parent(adam,able)
(2) 1 Call: parent_65644_able)?
(2) 1 Call: female(adam)?
(2) 1 Call: female(adam)
(2) 1 Back to: parent_65644_able)?
(2) 1 Exit: parent(adam,cain)
(2) 1 Exit: siblingtable(cain)
X = able
Y = cain
yes no
|%.

```

25

How to Satisfy a Goal

Here is an informal description of how Prolog satisfies a goal (like `father(adam,X)`). Suppose the goal is G:

- 1. Conjunction:** if $G = P,Q$ then first satisfy P, carry any variable bindings forward to Q, and then satisfy Q.
- 2. Disjunction:** if $G = P;Q$ then satisfy P. If that fails, then try to satisfy Q.
- 3. Negation:** if $G = \text{not}(P)$ then try to satisfy P. If this succeeds, then fail and if it fails, then succeed.
- 4. Simple goal:** if G is a simple goal, then look for a fact in the DB that unifies with G look for a rule whose conclusion unifies with G and try to satisfy its body

26

Note

- Two basic conditions are **true**, which always succeeds, and **fail**, which always fails.
- A comma (,) represents conjunction (and) and a semi-colon represents disjunction (or), as in:


```
grandParent(X,Y) :- grandFather(X,Y); grandMother(X,Y).
```
- There's no real distinction between rules and facts, which are just rules whose bodies are the trivial condition true. These are equivalent:


```
parent(adam,cain)
parent(adam,cain) :- true.
```
- Goals can be posed with any combination of variables and constants:
 - `parent(cain,able)` - Is Cain Able's parent?
 - `parent(cain,X)` - Who is a child of Cain?
 - `parent(X,cain)` - Who is Cain a child of?
 - `parent(X,Y)` - What two people have a parent/child relationship?

27

Terms

- The term is the basic data structure in Prolog.
- The term is to Prolog what the s-expression is to Lisp.
- A term is either:
 - a constant - e.g.
 - john, 13, 3.1415, +, 'a constant'
 - a variable - e.g.
 - X, Var, _, _foo
 - a compound term - e.g.
 - part(arm,body)
 - part(arm(john),body(john))

28

Compound Terms

- A compound term can be thought of as a relation between one or more terms:

part_of(finger,hand)

and is written as:

1. the relation name (called the principle functor) which must be a constant.
 2. An open parenthesis
 3. The arguments - one or more terms separated by commas.
 4. A closing parenthesis.
- The number of arguments of a compound terms is called its *arity*.

Term arity	
f	0
f(a)	1
f(a,b)	2
f(g(a),b)	2

29

The End

30