Prolog I

Syllogisms

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

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).
```


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).
```

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

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
| ?-

Syntax I: Structures

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

Syntax II: Base Clauses

- Example base clauses:
  - debug_on.
  - loves(john, mary).
  - loves(mary, bill).
- <base clause> ::= <structure> .

Syntax III: Nonbase Clauses

- Example nonbase 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>

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>

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.

Syntax VI: Assorted details

- Variables begin with a capital letter:
  X, Socrates, _result
- Atoms do not begin with a capital letter:
  x, socrates
- Other atoms must be enclosed in single quotes:
  - 'Socrates'
  - 'C:/My Documents/examples.pl'

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 ','.
- Example:
  ':-'(mortal(X), man(X)).

Backtracking

- loves(chuck, X) :- female(X), rich(X).
- female(jane).
- female(mary).
- rich(mary).
- ---------- *Suppose we ask:* loves(chuck, X).
- female(X) = female(jane), X = jane.
- rich(jane) fails.
- female(X) = female(mary), X = mary.
- rich(mary) succeeds.

Additional answers

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

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.

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))).`

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.)

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.

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

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"

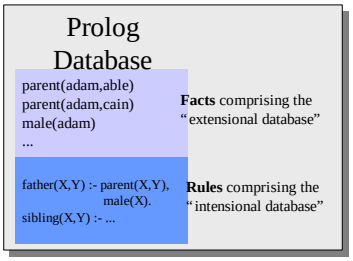
Extensional vs. Intensional

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"

A Simple Prolog Session

```
| ?- assert(parent(adam,able)).
yes
| ?- assert(parent(eve,able)).
yes
| ?- assert(male(adam)).
yes
| ?- parent(adam,able).
yes
| ?- parent(adam,X).
X = able
yes
```

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


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
```

```
| ?- [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)
(3) 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)
(6) 2  Exit: father(adam,able)
(9) 2  Call: mother(_65644,able) ?
(10) 3  Call: parent(_65644,able) ?
(10) 3  Exit: parent(adam,able)
(11) 3  Call: female(adam) ?
(11) 3  Fail: female(adam)
(10) 3  Back to: parent(_65644,able) ?
(10) 3  Exit: parent(eve,able)
(12) 3  Call: female(eve) ?
(12) 3  Exit: female(eve)
(9) 2  Exit: mother(eve,able)
(13) 2 Call: mother(eve,able) ?
(14) 3  Call: parent(eve,able) ?
(14) 3  Exit: parent(eve,able)
(15) 3  Call: female(eve) ?
(15) 3  Exit: female(eve)
(13) 2 Exit: mother(eve,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(eve) ?
(15) 3  Fail: female(eve)
(14) 3  Back to: parent(eve,able) ?
(14) 3  Fail: parent(eve,able)
(13) 2 Back to: mother(eve,able) ?
(13) 2 Fail: mother(eve,able)
(12) 3  Back to: female(eve) ?
(12) 3  Fail: female(eve)
(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)
(18) 3  Call: male(adam) ?
(18) 3  Exit: male(adam)
(6) 2  Exit: father(adam,cain)
(19) 2 Call: mother(_65644,able) ?
(20) 3  Call: parent(_65644,able) ?
(20) 3  Exit: parent(adam,able)
(21) 3  Call: female(adam) ?
(21) 3  Fail: female(adam)
(20) 3  Back to: parent(_65644,able) ?
(20) 3  Exit: parent(eve,able)
(22) 3  Call: female(eve) ?
(22) 3  Exit: female(eve)
(19) 2 Exit: mother(eve,able)
(23) 2 Call: mother(eve,cain) ?
(24) 3  Call: parent(eve,cain) ?
(24) 3  Exit: parent(eve,cain)
(25) 3  Call: female(eve) ?
(25) 3  Exit: female(eve)
(23) 2 Exit: mother(eve,cain)
(26) 2 Call: not able=cain ?
(27) 3  Call: able=cain ?
(27) 3  Fail: able=cain
(26) 2 Exit: not able=cain
(2) 1 Exit: sibling(able,cain)
X = able
Y = cain
yes no
| ?-
```


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:

- if G = P,Q then first satisfy P, carry any variable bindings forward to Q, and then satiety Q.
- if G = P;Q then satisfy P. If that fails, then try to satisfy Q.
- if G = not(P) then try to satisfy P. If this succeeds, then fail and if it fails, then succeed.
- 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

Note

- two basic conditions are true, which always succeeds, and fail, which always fails.
- A comma (,) represents conjunction (i.e. and).
- A semi-colon represents disjunction (i.e. or), as in:
  grandParent(X,Y) :- grandFather(X,Y); grandMother(X,Y).
- there is no real distinction between RULES and FACTS. A FACT is just a rule whose body is the trivial condition true. That is *parent(adam,cain)* and *parent(adam,cain) :- true.* are equivalent
- Goals can usually be posed with any of several 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?

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))

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:

  - the relation name (called the principle functor) which must be a constant.
  - An open parenthesis
  - The arguments - one or more terms separated by commas.
  - 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

The End