

## Logic Programming -- Mathematical Foundations: Predicate Calculus

- **Propositions**

- statements or queries about the state of the “universe”
- simplest form: **atomic proposition**  
form: *functor (parameters)*  
examples:     man (jake)  
                  like (bob, redheads)
- can either assert truth (“jake is a man”) or query existing knowledge base (“is jake a man?”)
- can contain variables, which can become bound  
man (x)

- **Compound Propositions**

- contain two or more atomic propositions connected by various logical operators:

| <u>Name</u> | <u>Symbol</u> | <u>Example</u> | <u>Meaning</u>       |
|-------------|---------------|----------------|----------------------|
| negation    | $\neg$        | $\neg a$       | not a                |
| conjunction | $\cap$        | $a \cap b$     | a and b              |
| disjunction | $\cup$        | $a \cup b$     | a or b               |
| equivalence | $\equiv$      | $a \equiv b$   | a is equivalent to b |
| implication | $\supset$     | $a \supset b$  | a implies b          |
|             | $\subset$     | $a \subset b$  | b implies a          |

slide 233

## Logic Programming -- Basic Principles

- Logic programming languages are *declarative* languages.
  - declarative semantics: essentially, there is a simple way to determine the meaning of each statement, and it does not depend on how the statement might be used to solve a problem
  - much simpler than imperative semantics
- Logic programming languages are *nonprocedural*.
  - Instead of specifying *how* a result is to be computed, we *describe* the desired result and let the computer system figure out how to compute it.
  - Example: Sort a list:  
 $\text{sort}(\text{old\_list}, \text{new\_list}) \subset \text{permute}(\text{old\_list}, \text{new\_list}) \cap \text{sorted}(\text{new\_list})$   
 $\text{sorted}(\text{list}) \subset \forall j \text{ such that } 1 \leq j < n, \text{list}(j) \leq \text{list}(j+1)$
- Prolog is an example of a logic programming language.

slide 234

## Predicate Calculus

- **Quantifiers** -- used to bind variables in propositions

- **universal quantifier:** "

- $\forall x.p$  -- means "for all x, P is true"

- **existential quantifier:** \$

- $\exists x.p$  -- means "there exists a value of x such that P is true"

- **Examples:**

- $\forall x.(woman(x) \supset human(x))$

- $\exists x.(mother(mary,x) \cap male(x))$

- **A canonical form for propositions -- clausal form**

$$B_1 \cup B_2 \cup \dots \cup B_n \subset A_1 \cap A_2 \dots \cap A_m$$

- **means: if all of the A's are true, at least one of the B's must be true**

- right side is the *antecedent*, left side is the *consequent*

- **Examples:**

- $likes(bob, mary) \subset likes(bob, redheads) \cap redhead(mary)$

- $father(louis, al) \cup father(louis, violet) \subset father(al, bob) \cap mother(violet, bob) \cap grandfather(louis, bob)$

- **A proposition with zero or one terms in the antecedent is called a "Horn clause". If there are no terms [e.g., "man(jake)"], it's called a "Headless Horn clause". If there's one term, it's a "Headed Horn clause".**

slide 235

## Predicate Calculus

- **Resolution** -- the process of computing inferred propositions from given propositions

- **Example:**

- if we know:

- $older(joanne, jake) \subset mother(joanne, jake)$

- $wiser(joanne, jake) \subset older(joanne, jake)$

- we can infer the proposition:

- $wiser(joanne, jake) \subset mother(joanne, jake)$

- **There are several logic rules that can be applied in resolution. In practice, the process can be quite complex.**

slide 236

## PROLOG — Name Value System

---

- **Objects are represented by either**
  - strings starting with lower case letters
  - integers
- **Identifiers of variables**
  - strings starting with an upper case character
- **Predicate names (functions)**
  - strings starting with lower case letters (like objects, but distinguishable by context)
- **Predicates**
  - <name> ( <list of arguments> )

slide 237

## Name-Value System (continued)

---

- **Types — atoms and integers**
- **Structures — lists (see later)**
- **Scope — predicate parameters and local variables (uninstantiated variables) are local to expression in which they are used; no such thing as global!**
- **State of the program does NOT include value memory**

slide 238

## Prolog -- Imperitives

---

- **Prolog maintains a database of known information about its “world”. This can be in two forms:**
  - **Fact statements -- corresponding to tailless Horn clauses, e.g.,**  
female(shelley).  
male(bill).  
father(bill, shelley).
  - **Rule statements -- corresponding to Headed Horn clauses, e.g.,**  
ancestor(mary, shelley) :- mother(mary,shelley).  
grandparent(x,z) :- parent(x,y), parent(y,z).
- **Prolog programs then can specify theorems or propositions that it wants proved or disproved, e.g.,**  
grandfather(bill, mary).
  - **System uses a theorem-proving model to see if this proposition can be inferred from the database.**  
“yes” result means it is true (according to the database)  
“no” result means either that it was proven false or that the system was simply unable to prove or disprove it

slide 239

## Imperitives Continued

---

- **Comparison Operators**
  - =, <, >, >=, <= (check for which!), \=
- **Expressions**
  - most Prologs support integer arithmetic
  - generally safest if expressions are contained in parentheses
  - check it out in your implementation
- **Assignment (local)**
  - “is” operator, infix
  - assigns right hand side value to variable on left  
X is (3+4)

slide 240

## PROLOG — Control

---

- The right hand sides of predicates are “evaluated” left to right
- On a right hand side, a false predicate causes the system to return to the last predicate to its left with a true value; a true result allows the evaluation of the right hand side to continue to the right.
- Collections of predicates are “examined” in their lexical (textual) order — top to bottom, first to last
- Recursion!
- Reference to a predicate is in effect a “function” call to the collection of predicates of that name.
- State of the program contains markers to last successful (i.e. True) instantiation in collections of facts or rules so as to support backtracking in recursion; when all markers are beyond end of all applicable predicate collections, result is “no”.

slide 241

## Prolog — Input/Output

---

- The output to a goal statement (query) can be:
  - The truth value of the resulting evaluation, or
  - The set of values that cause the goal to be true (instantiation)

- `read(X).`
- `write(Y).`

```
read(X),Y is (X+1),write(Y).
3.
4
X = 3
Y = 4 ;

no

read(X),Y=(X+1),write(Y).
6.
6+1
X = 6
Y = 6+1 ;

no
```

slide 242

## Prolog — Modularity and Abstraction

---

- facts and predicates of the same name are collected by a Prolog system to form modules — the components do not have to be textually contiguous
- collections of facts and rules may be stored in separate named files; files are “consulted” to bring them into a workspace

slide 243

## PROLOG PROGRAMS

---

- Declare FACTS about OBJECTS and their INTER-RELATIONSHIPS
- Define RULES (“CLAUSES”) that inter-relate objects
- Ask QUESTIONS about objects and their inter-relationships

slide 244

## FACTS

---

- **FACTS ARE TRUE RELATIONS ON OBJECTS**

- Michael is Cathy's father
- Chuck is Michael's and Julie's father
- David is Chuck's father
- Sam is Melody's father
- Cathy is Melody's mother
- Hazel is Michael's and Julie's mother
- Melody is Sandy's mother

- **facts need not make sense**

- The moon is made of green cheese

slide 245

## PROLOG FACTS

---

father (michael, cathy).

father (chuck, michael).

(father (chuck, julie).

father (david, chuck).

father (sam, melody).

mother (cathy, melody).

mother (hazel, michael).

mother (hazel, julie).

mother (melody, sandy).

made\_of (moon, green\_cheese).

slide 246

## RULES

---

- A person's parent is their mother or father
- A person's grand father is the father of one of their parents
- A person's grand mother is the mother one of their parents

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

(note ; means "or")

`grandfather(X, Y) :- father(X, A), parent(A, Y).`

(note , means "and")

`grandmother(X, Y) :- mother(X, A), parent(A, Y).`

slide 247

## QUESTIONS

---

Who is father of cathy ?

→ `father(X, cathy).`

Who is chuck the father of ?

→ `father(chuck, X).`

Is chuck the parent of julie ?

→ `parent(chuck, julie).`

Who is the grandmother of sandy ?

→ `grandmother(X, sandy).`

Who is the grandfather of whom ?

→ `grandfather(X, Y).`

slide 248

# PROLOG NAMES REVISITED

---

- **atoms**: Symbolic values of PROLOG
  - father ( bill, mike)
- Strings of letters, digits, and underscores starting with **lower case** letter
- **Variable**: unbound entity
  - father (X, mike)
- Strings of letters, digits, and underscores starting with **UPPER CASE** letter
- Variables are **not** bound to type by declaration

slide 249

# PROLOG FACTS & RULES

---

- **FACTS: UNCONDITIONAL ASSERTION**
  - assumed to be true
  - contains no variables  
mother(carol, jim).
  - stored in database
- **RULES: ASSERTION from which conclusions can be drawn if given conditions are true**
  - parent (X, Y) :- father(X, Y);  
                                  mother (X, Y).
  - contains variables for instantiation
  - Also stored in database

slide 250

## PROLOG INSTANTIATION

- **INSTANTIATION:** binding of a variable to value (and thus, a type)

FACTS {  
    color (apple, red).  
    color (banana, yellow).

color (X, yellow).      } question (goal)

|               |                       |
|---------------|-----------------------|
| X = apple     | color (apple, yellow) |
| instantiation | no matching pattern   |

|            |                        |
|------------|------------------------|
| X = banana | color (banana, yellow) |
|            | yes                    |

|            |  |
|------------|--|
| X = banana | results in match of goal<br>with database item |
|------------|--|

slide 251

## PROLOG UNIFICATION

- **UNIFICATION:** Process of finding instantiation of variable for which “match” is found in database of facts and rules
- Developed by Alan Robinson about 1965, but not applied until the 1970s to logic programming
- The key to Prolog

slide 252

## PROLOG EXAMPLE

FACTS

color(banana, yellow).  
color(squash, yellow).  
color(apple, green).  
color(peas, green).  
  
fruit(banana).  
fruit(apple).  
  
vegetable(squash).  
vegetable(peas).

bob eats green colored vegetables

RULE  $\left\langle \begin{array}{l} \text{eats}(\text{bob}, X) :- \\ \quad \text{color}(X, \text{green}), \text{vegetable}(X). \end{array} \right.$

bob eats X if  
X is green and  
X is a veggie

slide 253

## PROLOG EXAMPLE CONTINUED

eats(bob, X) :  
color(X, green),  
vegetable(X).

Does bob eat apples ?

eats(bob, apple).  
color(apple, green) => match  
vegetable(apple) => no

What does bob eat ?

eats(bob, X).  
color(banana, green) => no  
color(squash, green) => no  
color(apple, green) => yes  
vegetable(apple) => no  
color(peas, green) => yes  
vegetable(peas) => yes

Therefore

eats(bob, peas) true

X = peas

slide 254

## PROLOG AND/OR/NOT

---

- **DISJUNCTIVE RULES:** X if Y or Z  
parent(X, Y) :- father(X, Y);  
                  mother(X, Y).
- **CONJUNCTIVE RULES:** X if Y AND Z  
father(X, Y) :- parent(X, Y),  
                  male(X).
- **NEGATION RULES:** X if Not Y  
good(X) :- not(bad(X)).  
mother(X, Y) :- parent(X, Y),  
                  not(male(X)).

slide 255

## OLDER EXAMPLE

---

```
older(george, john).  
older(alice, george).  
older(john, mary).  
older(X, Z) :- older(X, Y), older(Y, Z).
```

Now when we ask a query that will result in TRUE, we  
get the right answer!  
older(george, mary).  
yes

But a query that is FALSE goes into an endless loop!  
older(mary, john).

Because the last element in older is the predicate that  
is repeatedly tried. The problem of “left recursion”.

**SOLUTION:**  
Remove older predicate and replace by:

```
is_older(X, Y) :- older(X, Y).  
is_older(X, Z) :- older(X, Y), is_older(Y, Z).
```

now is\_older(mary, john). produces no.

slide 256

## DON'T CARE!

---

In a predicate reference, the symbol `_` stands for an argument or parameter whose instantiated value is of no consequence.

Thus

`is-older(george,_).`

return yes. Indicating that there does exist an argument which will cause the query to be true, but the value is not returned.

slide 257

## PROLOG NOTES

---

- PROLOG is more than “LOGIC”
  - MATH
  - LIST manipulation
- LISTS are represented by [...]
  - an explicit list [a,b,c], or [A,B,C]
  - or like LISP we can identify the head and tail of a list through the use of the punctuation symbol “|” (vertical bar)  
[H|T] or [\_|T]
  - There are no explicit functions to select the head or tail (such as CAR and CDR)

slide 258

## List Functions

---

**/\*list operations\*/**

**/\*Membership\*/**

**member(H,[H|\_]).**

**member(H,[\_|T]):-member(H,T).**

**/\*Concatenation of two lists\*/**

**concat([],L,L).**

**concat([H|T],L,[H|U]):-concat(T,L,U).**

**/\*Reverse a list\*/**

**reverse([],[]).**

**reverse([H|T],L):-reverse(T,R),concat(R,[H],L).**

**/\*Equality of Lists\*/**

**equal\_lists([],[]).**

**equal\_lists([H1|T1],[H2|T2]):-H1=  
H2,equal\_lists(T1,T2).**

slide 259

## “Agatha Christie” Problem

---

**Who was the killer?**

Alice, her husband, son, daughter,  
and brother are involved in a murder.  
One of the five killed one of the other  
four.

1. A man and a woman were together in the  
at the time of the murder.
2. The victim and the killer were together  
on the beach at the time of the murder.
3. One of the children was alone at the  
time of the murder.
4. Alice and her husband were not together  
at the time of the murder.
5. The victim's twin was innocent.
6. The killer was younger than the victim.

slide 260

## Prolog Solution— Facts

---

**person(husband).  
person(son).  
person(daughter).  
person(brother).**

**child(son).  
child(daughter).**

**male(husband).  
male(son).  
male(brother).**

**female(alice).  
female(daughter).**

**twin(alice,brother).  
twin(brother,alice).  
twin(son,daughter).  
twin(daughter,son).**

slide 261

## Solution — Rules

---

**istwin(X):-twin(X,\_).**

**older(alice,son).  
older(alice,daughter).  
older(husband,son).  
older(husband,daughter).**

**inbar(M,N):-person(M),person(N),  
male(M),female(N).**

**together(S,T):-S=alice,T=husband.  
together(S,T):-T=alice,S=husband.**

**alone(P):-person(P),child(P).**

slide 262

## Solution — Goal statement

---

killed(X,Y):-person(X), person(Y),

*/\*The victim's twin was innocent.\*/*  
istwin(Y), not(twin(Y,X)),

*/\*The killer was younger than the victim.\*/*  
not(older(X,Y)),

*/\*Alice and her husband were not together  
at the time of the murder.\*/*  
not(together(X,Y)), X\=Y,

*/\*A man and a woman were together in the  
at the time of the murder.\*/*  
inbar(A,B), A\=X, B\=X, A\=Y, B\=Y,

*/\*Alice and her husband were not together  
at the time of the murder.\*/*  
not(together(A,B)),

*/\*One of the children was alone at the  
time of the murder.\*/*  
alone(C), C\=A, C\=B, C\=X, C\=Y.

slide 263

## PROLOG - Issues/Limitations

---

- “Closed World” -- the only truth is that known to the system
- Efficiency -- theorem proving can be *extremely* time consuming
- Resolution order control
  - Prolog always starts with left side of a goal, and always searches database from the top. Have some control by choice of order in the propositions and by structuring database.
  - Prolog uses backward chaining (start with goal and attempt to find sequence of propositions that leads to facts in the database). In some cases forward chaining (start with facts in the database and attempt to find a sequence of propositions that leads to the goal) can be more efficient.
  - Prolog always searches depth-first, though breadth-first can work better in some cases.
- The Negation Problem -- failure to prove is not equivalent to a logical not
  - not(not(some\_goal)) is not equivalent to some\_goal

slide 264