

# COSC 2P93 Logic Programming

- **Instructor: Brian Ross**
- **Texts:**
  - Prolog Programming for Artificial Intelligence, 3e, Ivan Bratko, Addison-Wesley 2001.
  - (recommended): Programming in Prolog ,4e, Clocksin & Mellish, Springer-Verlag, 1994.
- **Goal: An introduction logic programming and Prolog**
  - with Lisp, Prolog is one of main AI languages
- **Lots of programming questions: only way to learn a new language and programming paradigm.**

# Systems available

- **Prolog:**
  - **SGI, Windows: Sicstus Prolog 3.8.5 ( /usr/local/bin/sicstus )**
  - **Windows: LPA Prolog (5 copies)**
  - **DOS: Web page points to ‘Prolog2’, which has been used successfully in past years**
- **Note: If you find a system (free, commercial) which seems standard, feel free to use it.**
  - **however, you must port your code to Sicstus, and have it run on Sandcastle before submitting it**
  - **marker will run your code on sandcastle: if it doesn’t run, no execution mark!**

## Prolog: brief history

- **born in the mid-70's in Marseilles, France**
  - **researchers in Edinburgh then took it and made it into a comprehensive language; “Edinburgh syntax”**
- **derived from research in automatic theorem proving.**
- **Adopted by Japanese 5th Generation Computer Technology initiative**
- **Wide acceptance from AI community**
  - **Lisp is a competitor; however, Lisp is much “lower-level”**
- **Programming characteristics:**
  - **high-level: foundation in logic permits “declarative programming”**
  - **user specifies what to compute at a high level; Prolog interpreter searches for a computed solution**
  - **can write powerful applications in much less code than in C**
  - **great for: AI, fast prototypes, natural language, intelligent databases, systems programming prototyping, parallelism**
- **Implementation-wise:**
  - **Prolog is usually interpreted --> slow!**
  - **compilers exist; Prolog performance acceptable**

# Starting Prolog

(following is for Sicstus Prolog)

- First, write your programs into a text file. Helpful to use “. pro” as your Prolog code extension
- then “sicstus”
  - get the prompt: | ?-
  - Prolog interpreter is awaiting your commands
- To load your file initially:
  - ?- consult('myfile.pro'). [return]
  - note:
    1. no blank between consult and “(“
    2. the period at end of line (ends every Prolog statement)
    3. Single quotes around filename required if it uses an extension
- If you edit “myfile.pro” in another window, and want to re-load it, then type: reconsult('myfile.pro').
  - this will replace existing program code with the reloaded code
- abbreviations: ?- consult('file'). <--> ?- ['file'].  
                  ?- reconsult('file'). <--> ?- ['-file'].

# Prolog

- To exit, type: `?- halt.` (or CTRL D)
- Between loading and exiting, you run your program. Consider the following Prolog program:

```
/* my date program */  
likes(jane, sushi).  
likes(jane, salad).  
likes(jane, lobster).  
likes(mary, haggis).  
likes(steve, bbq).  
likes(steve, lobster).  
likes(steve, sushi).  
  
good_date(Guy, Gal) :- likes(Guy, Food), likes(Gal, Food).
```

- Assume this code is put in the file: `date.pro`

# Prolog

- there are two subroutines or predicates:
  - good\_date/2
  - likes/2
- predicates defined by:
  - a) name of predicate (date, likes)
  - b) number of arguments
- a predicate is composed of one or more Prolog statements or clauses
- there are 3 kinds of Prolog clauses:
  - 1. queries (questions, goals): a question for the interpreter to solve
    - » eg. ?- good\_date(steve, X).
  - 1. fact or assertion: this states an unconditional fact
    - eg. likes(jane, salad).
    - » the meaning of this: it is true that jane likes salad.

# Prolog

## – 3. rule:

- » aka procedure or predicate
- » this is like an “if-then” statement
- » a conditional truth
- » General form:  $H :- G_1, G_2, \dots, G_k.$ 
  - H is true if and only if all the  $G_i$ 's are true

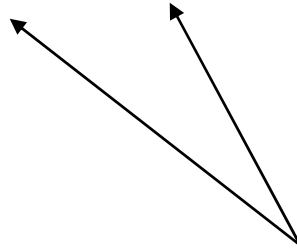
eg. `good_date(Guy, Gal) :- likes(Guy, Food), likes(Gal, Food).`

- » the meaning of this: it is a good date for a particular guy and gal if the guy likes a food, and the gal likes the same food

# Prolog

- A predicate can have a mixture of facts and rules.
- consider this fact from likes:

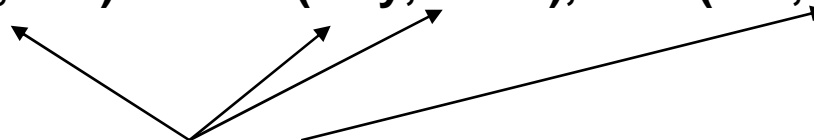
`likes(jane, salad).`



**these are constants. They represent data objects or values.**

- always begin with a lower-case letter, numeral

`good_date(Guy, Gal) :- likes(Guy, Food), likes(Gal, Food).`



**logic variables: placeholders for constants**  
- always begin with a capital letter

# Prolog

- Now to execute the program. Unlike “conventional” languages, there are many ways to execute this program...
- To interactive execute the program you issue a program call or query to the interpreter...

?- likes(jane, lobster).

yes

?- likes(jane, kraft\_dinner).

no

?- likes(jane, ferraris).

no

?- likes(steve, Food).

Food = bbq ;

% Note: typing ‘;’ tells Prolog to find next soln.

Food = lobster ;

Food = sushi;

no

# Prolog

**?- likes(Who, sushi).**

**Who = jane ;**

**Who = steve ;**

**no**

**?- good\_date(A, B).**

**A = jane, B = jane ;           % jane likes sushi**

**A = jane, B = steve ;       % jane and steve like sushi**

**A = jane, B = jane ;       % jane likes salad**

**A = jane, B = jane ;       % jane likes lobster**

**A = jane, B = steve ;      % jane and steve like lobster**

**A = mary, B = mary ;       % mary likes haggis**

**A = steve, B = steve ;     % steve likes bbq**

**A = steve, B = jane ;      % steve and mary like lobster**

**A = steve, B = steve ;     % steve likes lobster**

**A = steve, B = jane ;      % steve and jane like sushi**

**A = steve, B = steve ;     % steve likes sushi**

**no**

# Prolog

```
?- good_date(jane, steve).  
yes  
?- good_date(steve, jane).  
yes  
?- good_date(jane, jane).  
yes  
?- good_date(mary, steve).  
no  
?- likes(jane, X), likes(steve, X).  
X = sushi  
yes
```

- **Comments:**
  - exhaustive matching of answer combinations
  - matching process uses order of statements in program
  - Different variables can match to same constant (person is their own date)
  - if a solution is computed, then that same solution given as a query should result in 'yes'; vice versa for 'no'

# Prolog

- Possible enhancements:
  - 1. How to stop someone from being their own date?
  - 2. How to see the food two people like?

**% 1. smarter good\_date:**

**good\_date2(Guy, Gal) :-**

**likes(Guy, Food), likes(Gal, Food), \+ (Guy = Gal).**

**% 2. more informative good\_date:**

**good\_date3(Guy, Gal, Food) :-**

**likes(Guy, Food), likes(Gal, Food).**

- note: \+ is “not”: succeeds if the goal beside it fails
- = is ‘unification’: succeeds if terms match (more later)
- could replace 3rd goal in good\_date2 with: Guy \= Gal

# Prolog

- **Prolog interpreter:**
  - each predicate call in the body of a rule, and each call in a query, is called a *goal*.
  - the order you put goals in a query or rule is crucial for standard Prolog interpreters
    - » scheme: solve each goal in the query from left to right
  - clauses order important too
    - » scheme: test clauses in the order they reside in the predicate
  - predicate order does not matter
- hitting “ ; “ when a solution is given causes interpreter to find yet another solution: called backtracking
  - backtracking causes execution to revert back to last place a clause was successful, and move on to the next one
  - interpreter keeps track of these places in program, as well as the values given to logic variables
  - if you keep hitting “;”, all solutions possible will be given (including duplicate ones!)
- more details later

## Some comments

- How you understand the program and its queries is up to the programmer.
  - the Prolog interpreter does not understand the meaning of the predicate names, atom names, variables, ...
- The answer “no” can mean a number of things:
  - the information queried does not exist in the program, and is therefore assumed “false” (eg. likes(harvey, porridge). may not exist )
  - there is a logical error in the program, ie. the program is erroneously written
    - » eg. good\_date(Guy, Gal) :- likes(Guy, Food), likes(Food, Gal).
  - there is a spelling error:
    - » eg. likes(jane, stake).
- Likewise, a “yes” (with computed results if they exist) can mean:
  - a solution was obtained or verified (logical “true”)
  - a solution was obtained, but it is wrong, there can be logical errors in the program, spelling errors, etc.
- queries can be complex; if they are too big, it is better to write a program predicate for them (case in point: “good\_date”)

# Prolog

- **Note that we can understand each program predicate at a high level**
  - “what to” programming
  - almost like talking directly in English
  - this is called “declarative programming”
- **Can also understand program in terms of goals to solve, and the order in which to solve them**
  - “how to” programming
  - called “procedural programming”
- **Recommended to split large predicates into multiple lines**
  - makes program more readable & modifiable
- **use meaningful names for predicates, constants, variables**
- **Put all predicate clauses together in a single file**