

# Prolog Introduction

Clocksin & Mellish Ch 1 & 2

# What is a Prolog Program?

- ◇ A program consists of a database containing **one or more facts**
  - > **A fact is a relationship between a collection of objects**
  - » **dog ( fido ).**
    - > **Fido is a dog**
      - it is true that Fido is a dog
  - » **mother ( mary, joe ).**
    - > **Mary is the mother of Joe**
      - it is true that Mary is the mother of Joe
  - » **compete ( ali, leila, tennis).**
    - > **Ali and Leila compete in tennis**
      - it is true that Ali and Leila compete in tennis

## What is a Prolog Program? – 2

- ◇ Relationships can have any number of objects
- ◇ Names are usually chosen to be meaningful
  - » **Within Prolog, names are just arbitrary strings. It is people who give meaning to names.**

## What is a Prolog Program? – 3

- ◇ And a program consists of a database of **zero or more rules**
  - > **A rule is an if...then relationship of facts**
  - » **use ( umbrella ) :- weather ( raining ).**
    - > **use an umbrella if weather is raining**
  - » **use ( umbrella ) :- weather(raining) , own ( umbrella ).**
    - > **use an umbrella if weather is raining and you own an umbrella**
  - » **use ( umbrella ) :- weather ( raining ) ,  
( own ( umbrella ) ; borrow ( umbrella ) ).**
    - > **use an umbrella if weather is raining and you either own an umbrella or can borrow an umbrella**

## More on rules

- ◇ Rules have the general structure

**head :- body**

- » **Only one fact can be in the head – the consequent**
  - » **The body is a boolean combination of predicates**
  - » **Use , (and) and ; (or) and () (parenthesis) to logically organize the "condition" – the antecedent**
- ◇ Rules are written backwards to
    - » **emphasize the backward chaining for database search**
    - » **be more regular in structure, since the head is only one predicate**

# Constants

- ◇ **Constants** are names of that begin with lower case letters
  - » **ali, leila, tennis, dog, fido, mother, mary, joe, umbrella, raining, weather, own, borrow**
  - » **names of relationships are constants**

# Variables

- ◇ In place of **constants** in facts and rules one can have **variables**
  - » **variables are names that begin with upper case letters**
    - > **X, Y, Who, Whom, List, Person**  
**loves ( Everyone, barney ).**
    - > **Everyone loves barney**  
– for all values of **Everyone** it is the case that **loves(Everyone, barney)** is true.  
**noisy ( Singer ) :- valkyrie ( Singer ) ;**  
**tenor ( Singer ).**
    - > **A Singer is noisy if they are a Valkyrie or a tenor**

## Variables – 2

**dwarf ( Person ) :- brother ( Person, Other ) ,  
dwarf ( Other ).**

**> A person is a dwarf, if they the brother of other  
and the other is a dwarf**

**>> Variables can also begin with \_ (underscore)**

**\_ (anonymous variable)  
\_1 \_abc (not anonymous variable )**



# Running a Prolog Program

- ◇ Programs are stored in one or more files that are **consulted**
- ◇ On Prism to run SWI Prolog enter  
**% pl**
- ◇ The following prompt appears  
**| ?-**
- ◇ Consult the appropriate file(s) – **add to the database**  
**| ?- consult('ring.pro').**
  - > **While it is possible to enter facts and rules interactively using **consult ( user )**, it is inconvenient and error prone**
  - > **SWI-prolog does not have a **reconsult** predicate, only **consult** is used.**

## Running a Prolog Program – 2

- ◇ Make zero or more queries (next slides)
- ◇ Exit prolog

**| ?- CTRL-d /\* and for consult ( user ) on Prism \*/**

**consult(user) enables you to enter facts & rules into the database without storing them in a file. It is not an effective way to work with Prolog.**

# Queries

- ◇ A **query** in Prolog is boolean combination of predicates – like the antecedent of a rule

- > **A query is like a rule, except we leave out the consequent true**

- true :- dwarf ( alberich ).**

- > **becomes simply**

- dwarf ( alberich ).**

- ◇ Use comma (and), semicolon (or) and parenthesis to form a query expression
- ◇ Most common is to have a single predicate

## Queries – 2

- ◇ **Answer** is a **binding of the variables** that make the query **expression true** – if no variables then the **answer** is **yes**. If no such binding exists, the **answer** is no
- ◇ The database is searched to match the query (similar to the Lisp database program)
- ◇ The search
  - » **Uses backward chaining**
  - » **is depth first**
  - » **is sequential through the database from first to last**
- ◇ Try the exercise on [ring.pro](http://ring.pro)

# Structures

- ◇ **Structures** are a means of grouping a collection of other objects
  - » **Structures are also called compound terms, or complex terms**
  - » **The name of a structure is called a functor**
  - » **The items within a structure are called components**
  
- ◇ The general pattern is
  - functor ( component\_1 , component\_2 ,**
  - ...**
  - component\_n )**

## Structures – 2

- ◇ Components can also be structures – recursive definition

**if component\_1 = functor1 ( comp1, comp2 )**

**> giving**

**functor ( functor1 ( comp1, comp2 ),  
          component\_2 ,**

**...**

**component\_n )**

**> from**

**functor ( component\_1 , component\_2 ,**

**...**

**component\_n )**

## Example structures

- ◇ Books have authors and titles, so we could have  
**book ( dickens , great\_expectations )**
- ◇ People have books. In particular, Leila could have Great Expectations  
**has ( leila , book ( dickens , great\_expectations ) )**
- ◇ Facts in Prolog are structures where the predicate is the functor of a structure and the arguments of the predicate are the components of the structure

# Characters

- ◇ Prolog is based on the ASCII character set
- ◇ Characters are treated as small integers 0 .. 127
- ◇ Characters may be
  - » **printed**
  - » **read from a file or keyboard**
  - » **compared**
  - » **take part in arithmetic operations**
- ◇ Characters are distinguished as
  - » **printing – visible on the paper**
  - » **nonprinting – look like whitespace**



# Operators

◇ All predicates in Prolog are functors, even `,` `;` and `:-`

> **A rule such as**

```
dwarf ( Person ) :- brother ( Person , Other ) ,  
                    dwarf ( Other ) .
```

> **is a shorthand for**

```
:- ( dwarf ( Person ) ,  
     ( brother ( Person , Other ) , dwarf ( Other ) ) ) .
```

## Operators – 2

- ◇ Arithmetic and relational operators are also functors, thus

**a + b \* c internally is + ( a , \* ( b , c ) )**

- ◇ This is inconvenient so Prolog permits operators to be written in standard infix notation

» **You will learn later how you can define your own infix operators**

# Arithmetic

- ◇ The arithmetic operators **do not do** arithmetic. No assignments are made
  - > **It is simply pattern matching – infix operators are simply a convenience for expressing a structure**

**5 = 4 + 1. ==> no**

**4 + 1 = 4 + 1. ==> yes**

**1 + 4 = 4 + 1. ==> no**

- > **Use the operator is to do arithmetic**

**5 is 4 + 1. ==> yes**

**1 + 4 is 4 + 1. ==> no**

- ◇ Arithmetic is only done on the right!
- ◇ Right hand side is evaluated using arithmetic, then a pattern match is made with the left hand side.

## Arithmetic – 2

- ◇ Can use variables in arithmetic expressions for pattern matching

**A = 4 + 1. ==> A has the pattern "4+1"**  
– spaces removed

**A is 4 + 1. ==> A has as value the pattern 5**

> In some Prologs the latter expression simply responds yes, so try the following.

**A is 4 + 1 , A = 5. ==> A = 5 is the binding for true**

> More complex example

**B is 3 + 2 , C is B \* 5 , A is C + B**

**==> B = 5, C = 25, A = 30**

# Lists

- ◇ As in Lisp, lists are a ubiquitous structure in Prolog. The syntax changes (to protect the innocent?)
  - » **Actually () are used to delimit structure components and to provide precedence for operators, so using them for lists as well would be confusing.**
- ◇ The structure is
  - [ item-1 , item-2 , ... , item-n ]**
  - [ a , b , c ]**
  - [ a , [ b , c ], [[ [ d ] ]], e , []]**
- ◇ The empty list is **[]**

## Lists - 2

- ◇ The square bracket notation is a shorthand in place of using the functor `.`, dot

**[ a , b , c ] is really . ( a , . ( b , . ( c , [] ) ) )**

- ◇ As in Lisp, lists have a **head** (`car` / `first`) and a **tail** (`cdr` / `rest`), thus

**[ Head | Tail ]**

- ◇ But you do not have operators to extract the head and tail, all you have is pattern matching

» **We will look at example Prolog utilities on lists to demonstrate**

- ◇ Empty list has no head or tail

**[] ≠ [ \_ | \_ ]**