Logic Programming Paradigm CS315

Joie Ann M. Mac

Introduction

- Imperative programming
  - Sequence of instructions to be executed one after the other to solve a program
  - Description of problem is implicit
- Logic programming
  - Description of the problem and the method for solving it are explicitly separated from each other
  - Declarative programming

Introduction

Algorithm = Logic + Control

- Logic
  - Descriptive component
  - What to do
- Control
  - Component that finds a solution, taking the description of the problem as a point of departure
  - How it should be done

Introduction

- Problem
  - described in terms of relevant objects and relations between those objects
  - represented in the clausal normal form of **logic**

Introduction

- Advantages of splitting logic and control
  - The two components may be developed separately from each other, *i.e.*, when describing the problem need not to know how the control component operates on the resulting description
  - A logic component may be developed using a method of stepwise refinement; we have to watch over the correctness of the specification only
  - Changes to the control component affect only the efficiency of the algorithm; they do not affect the solutions produced

Features (Logic vs. Imperative)

1. Computing takes place over the domain of all terms defined over a "universal" alphabet
   - set of all words
2. Values are assigned to variables by automatically generated substitutions, called most general unifiers
   - may contain logical variables
3. The control is provided by a single

```prolog
human(socrates).
mortal(X) :- human(X).
```

```python
if socrates_is_human:
    socrates_is_mortal = True
```

Strengths and Weakness

- Strengths
  - Enormous simplicity
  - Conciseness
- Weakness
  - Restriction to one control mechanism
  - Use of single data type

Support

- Logic programming supports:
  - Declarative programming
  - Interactive programming

Support for Declarative Programming

- Procedural interpretation
  - How the computation takes place
  - Concerned with the method
  - A declarative program is viewed as a description of an algorithm that can be executed
- Declarative interpretation
  - The question what is being computed
  - Concerned with the meaning
  - A declarative program is viewed as a formula, and one can reason about its correctness without any reference to the underlying computational mechanism

Support for Interactive Programming

- User can write a single program and interact with it by means of various queries to which answers are produced

- **Prolog** is best known programming language based on the logic programming paradigm

Programming with Prolog

Introduction

- **Pro**gramming in **Log**ic
- Originally used for research in **natural language processing**
- Popular in AI community

- **NOTE**: Programming in prolog is significantly different from conventional procedural programming and requires a readjustment in the way one thinks about programming

Prolog vs. Other Languages

- Prolog is a declarative language
  - can be viewed as stating **what** is computed, independent of the method for computation
- Prolog programs are structured in terms of **relations** whereas other traditional languages are structured in terms of **functions**
  - expresses relations among entities
- Prolog programs are **nondeterministic**
  - several elements can be in a particular relation to a given element

Prolog vs. Other Languages

Example:

prolog

```
parent(john, mary).
parent(mary, ann).

ancestor(X, Y) :- parent(X, Y).
ancestor(X, Y) :- parent(X, Z), ancestor(Z, Y).
```

Facts vs Rules

- Prolog is a logic-based language
  - Programmer defines **facts**

```prolog
parent(john, mary).
```

This fact asserts that John is a parent of Mary.

  - Programmer defines **rules**

```
grandparent(X, Y) :- parent(X, Z), parent(Z, Y).
```

This rule means that "X is a grandparent of Y if X is the parent of Z, and Z is the parent of Y."

Queries

- Prolog is a logic-based language
  - Programmer can ask questions or **queries**

```prolog
?- grandparent(john, ann).
```

Prolog will respond with true or false depending on whether it can infer that John is a grandparent of Ann based on the facts and rules provided

Knowledge Representation

- Highly suitable **knowledge representation language**

```prolog
% Facts
parent(john, mary).
parent(mary, ann).

% Rules
grandparent(X, Y) :- parent(X, Z), parent(Z, Y).

% Query
?- grandparent(john, ann).
```

Introduction

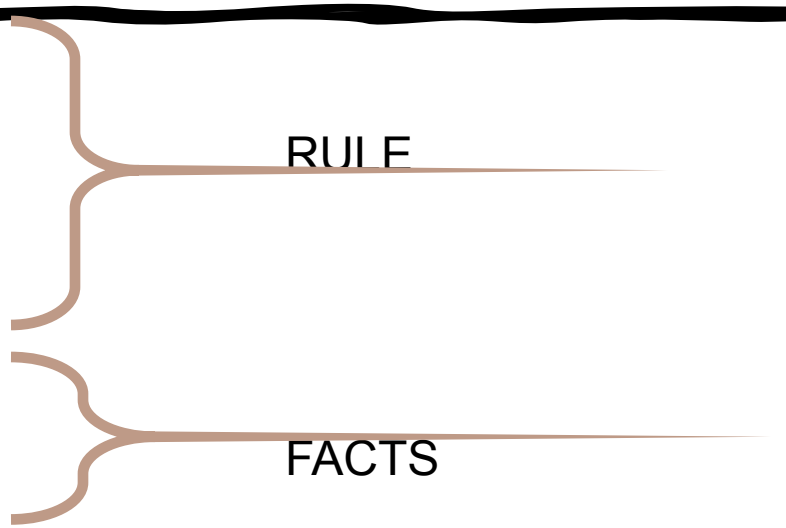
- Queries can be asked from Prolog
- Prolog will use the **facts** and **rules** in its **database** to try to construct a **proof** and it will report whether or not it was successful

Introduction

- Question
  - Will it rain?
- Rule
  - It will rain when the sky is grey and the wind is from the west
- Facts
  - The sky is grey.
  - The wind is from the west.

Introduction

```
rain :-
   windIsWesterly,
   skyIsGrey.
windIsWesterly.
skyIsGrey.
```



- Ask the query `?-rain.`
  - Prolog answers **`true`**

First-Order Logic

- First-Order Logic
  - Predicate calculus
  - Can make generalizations (Universal quantification)
- Zero-Order Logic
  - Propositional calculus
  - No means of generalizing our knowledge to groups sharing the same properties

Characteristics of Prolog

- Logic Programming
  - Prolog is a **logic based language**; the meaning of a significant fraction of the Prolog language can be completely described and understood in terms of the **Horn clause** logic of FOPL
- A Single Data Structure as the Foundation of the Language
  - Prolog offers the **term** as the basic data structure to implement any other data structure like lists, arrays, trees, records, queues, etc.

Characteristics of Prolog

- Simple Syntax
  - A Prolog program is actually a **sequence of terms** (syntactically)
- Program Data Equivalence
  - Programs and data conform to the same syntax; it is easy to interpret programs as data of other programs, and also to take data as programs
- Weak Typing
  - Types of variable in Prolog need not to be declared explicitly

Characteristics of Prolog

- Incremental Program Development
  - Prolog program can be developed and tested **incrementally**
- Extensibility
  - A Prolog system can be **extended** and **modified**. It is also possible to modify the Prolog language, and to adapt both its syntax and semantics up to the needs (e.g. object-oriented Prolog)

Tools

- SWI-Prolog
  - Download at http://www.swi-prolog.org/
- Notepad++ or any text editor

Syntax of Prolog

Basic Characters or Symbols

- Uppercase letters `A, B, …, Z`
- Lower case letters `a, b, …, z`
- Digits `0, 1, …, 9`
- Special characters `+, -, /, *, <, >, =,:,., &, ~` and `_`

Terms

- Atoms
- Numbers
- Variables
- Complex terms or Structures

Atoms

- A string of characters made up of upper-case letters, lower-case letters, digits, and the underscore character. This begins with a lower-case letter.
  - burger, big_burger, and m_money
- An arbitrary sequence of character enclosed in single quotes
  - 'Vincent', 'Five_Dollar_Shake‘
- A string of special characters
  - @= and ======> and :-

Numbers

- Used to carry our arithmetic operations

```
X is 10 * 2.    % X = 20
Y is 20 / 4.    % Y = 5.0
Z is 7 // 2.    % Z = 3  (integer division)
W is 10 mod 3.  % W = 1  (modulus)
```

- Usually integers and floating points

```prolog
X is 5 + 3.  % X = 8
```

```prolog
Y is 3.14 * 2.  % Y = 6.28
```

Variables

- A string of upper-case letters, lower-case letters, digits and under-score characters that starts either with upper-case letter or with underscore
  - X, Y, _tag, X_526, List
- The **lexical scope** of variable name is one clause
  - If a variable occurs in two clauses then it signifies two different variables
  - But within one clause the variable has same meaning

Complex Terms or Structures

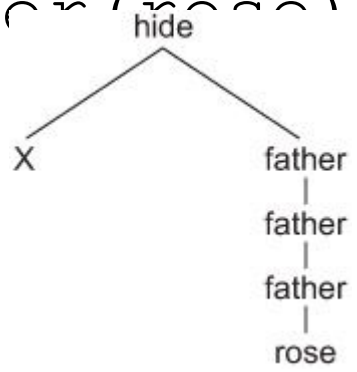
- Build out of a **functor** followed by a sequence of **arguments**
  - **Functor** is an atom and must not be a variable
  - **Arguments** are placed in parenthesis, separated by comma
    - Can be any term

```
playsGuitar(jody):
loves(john, mary):
person(name(john), age(30)):
```

Example Syntax

- `playsGuitar(jody)`
- `loves(X, Y)`
- `jealous(ram, sham)`
- `hide(X,`

  `father(father(father(rose)))`



Arity

- The number of arguments that a complex term has
  - `woman(sia)` has arity of 1
    - `woman/1`
  - `love (raghav, sia)` has arity 2
    - `love/2`
  - `love (raghav, marcellus, sia)` has arity 3
    - `love/3`

Facts

- Statements that describe **object properties** or **relations** between objects
- Assumed to be true
- Ends with **period**

```
character(priam, iliad).
character(achilles, iliad).
character(ulyses, odyssey). character(penelope,
odyssey).
```

Facts

- Examples

```
male(priam).
male(achilles).
male(ulyses).
female(helen).
female(penelope).
father(priam, hector).
mother(hecuba, hector).
```

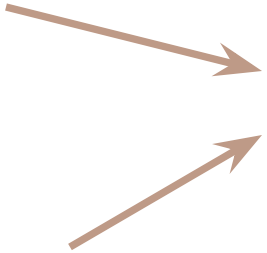
Facts

- Examples

```
king(menelaus, sparta, achaean).
king(agamemnon, argos, achaean).
king(priam, troy, trojan).
```

Queries

- A request to prove or retrieve information from the database if a fact is true
- Answers TRUE or FALSE



Goals to prove

```
?- male(ulyses).
true.
?- male(Penelope).
false.
```

Queries

- Compound queries

```
?- male(menelaus), king(menelaus, sparta,
achaean).
true.
```

- Conjunction operator is the comma (`,`) in between two structures

Logical Variables

- Begins with UPPERCASE LETTERS or an underscore (_)
- May stand for any term (constants, compound terms, and other variables
- When a query contains variables, the Prolog **resolution** algorithm searches terms in the database that **unify** with it.
- It then substitutes the variables to the matching arguments.

Logical Variables

- Consider the following facts:

```
character(ulyses, odyssey).
character(achilles, iliad).
```

- Prolog will search the database for facts where the second argument is odyssey

```
?- character(X, odyssey)..
```

- Results:

```
X = ulyses
```

Logical Variables

- Consider the following facts:

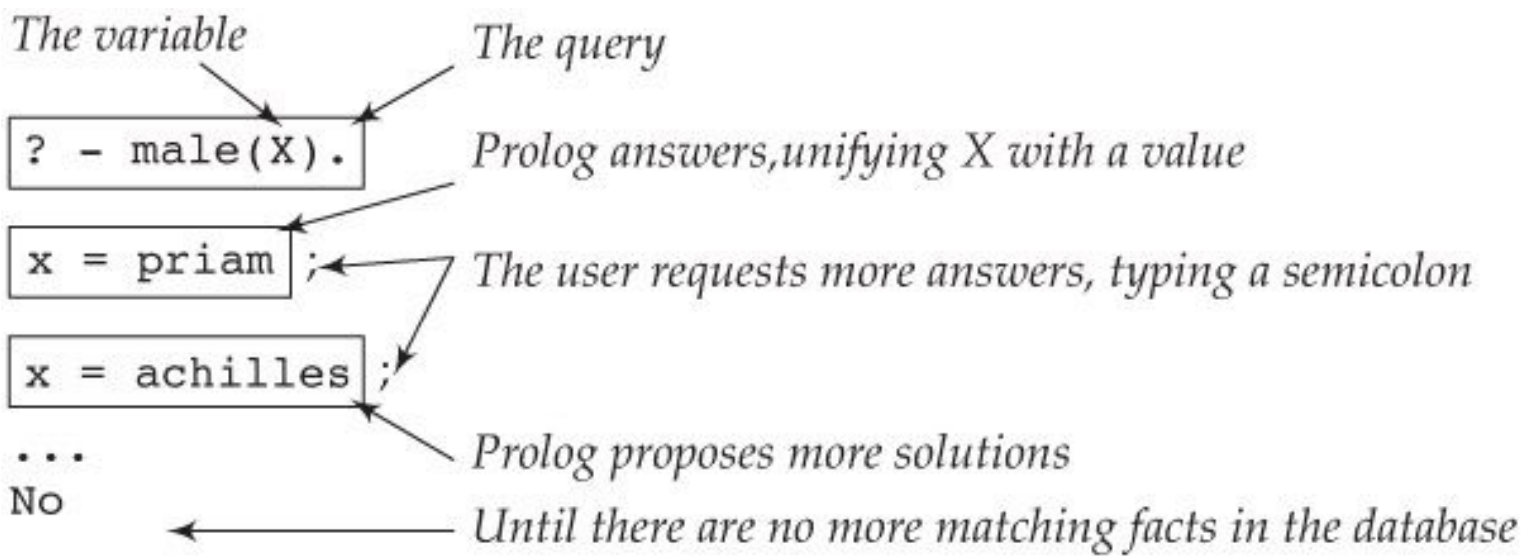
```
character(ulyses, odyssey). character(achilles,
iliad). character(penelope, odyssey).
```

- Prology Prolog will search the database for facts where the second argument is odyssey.
- It finds the facts character(ulyses, odyssey) and character(penelope, odyssey).
- It will unify X with ulyses and penelope in separate answers.
- Results:

```
?- character(X, odyssey).
X = ulyses ;
X = penelope.
```

Logical Variables

- If multiple solutions, Prolog considers the first fact to match the query
- Type semi-colon (`;`) to get the next answers until no more solution.



Shared Variables

- Goals in the conjunction query can share variables.
- This is useful to constrain arguments of different goals to have a same value.

Shared Variables

Facts:

```
parent(john, mary).

parent(john, alice).

parent(mary, charlie).

parent(alice, dave).
```

Query:

```
?- parent(X, Y), parent(Y, Z).
```

Results:

```
X = john,
Y = mary,
Z = charlie ;

X = john,
Y = alice,
Z = dave.
```

Shared Variables

Is the king of Ithaca also a father?

Fact

king(ulyses, ithaca, achaean).

father(ulyses, telmachus).

Query

?- king(X,ithaca,Y),father(X, Z).

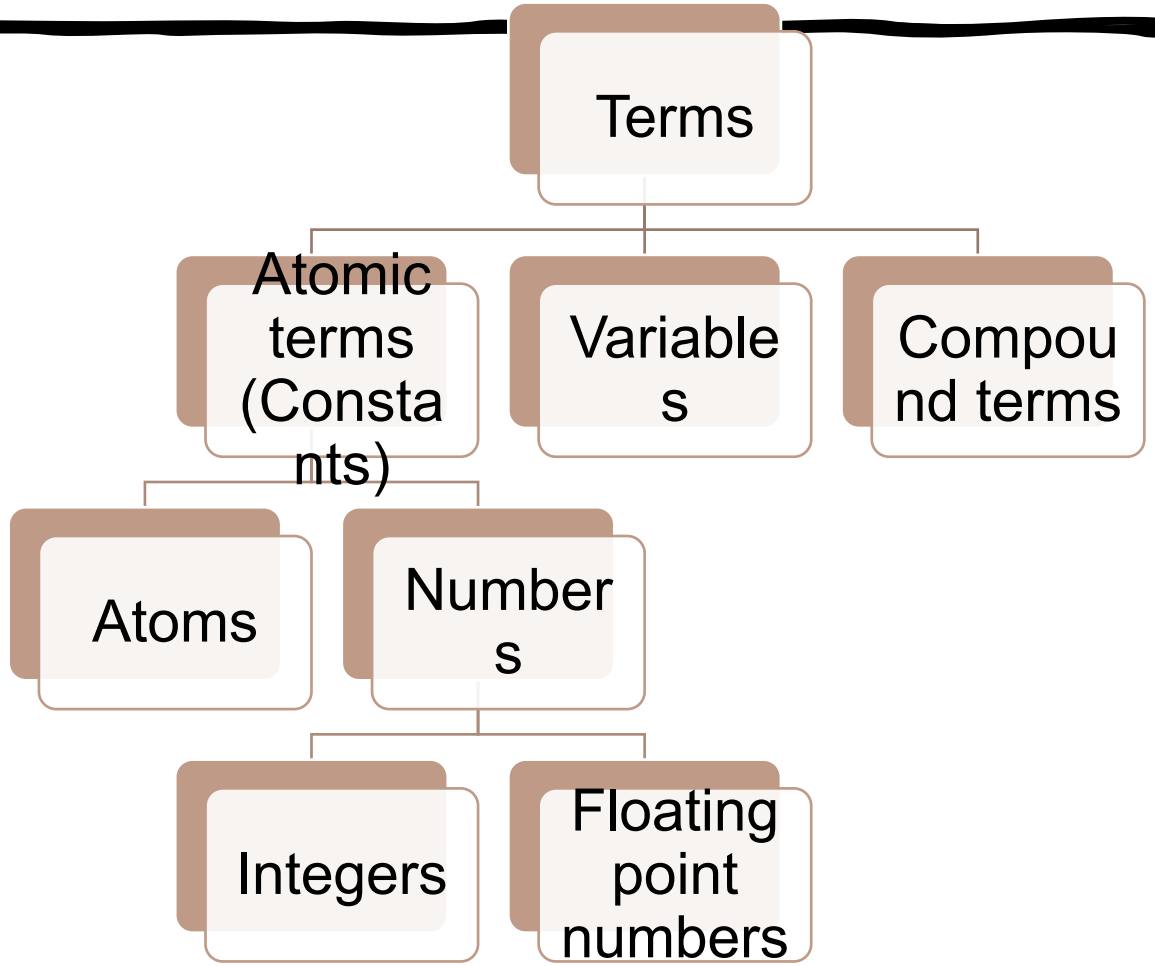
Results:
X = ulyses,
Y = achaean,
Z = telmachus.

Shared Variables

- We can indicate to Prolog that we do not need to know the values of Y and Z using **anonymous variables**.
- Replace variables with underscore (_)

```
?- king(X,ithaca,_),father(X,_).
X=ulyses.
```

Data Types


Terms
Atomic terms (Constants)
Variables
Compound terms
Atoms
Numbers
Integers
Floating point numbers

Simplified Prolog Conventions

- Atoms are sequences of letters, numbers, and/or underscores beginning with a lowercase letter, as `ulyses`, `iSLanD3`, `king_of_Ithaca`.
- Some single symbols, called solo characters are atoms: `!` `;`
- Sequences consisting entirely of specific symbols or graphic characters are atoms: `+` `-` `*` `/` `^` `<` `=` `>` `~` `:` `.` `?` `@` `#` `$` `...` `\` `'`

Simplified Prolog Conventions

- Any sequence of characters enclosed between single quotes is also an atom, as `'king of Ithaca'`. A quote within a quoted atom must be double quoted: `'I''m'`
- Numbers are either decimal integers, such as `-29,1960,` octal integers when preceded by `0o`, as `0o58`, hexadecimal integers when preceded by `0x`, as `0xF6`, or binary integers when preceded by `0b`, as `0b101`.

Simplified Prolog Conventions

- Floating-point numbers are digits with a decimal point, as `3.14`, -1.8. They may contain an exponent, as 23E-5 ($23\ 10^{-5}$) or -2.3e5 ($2.3\ 10^{-5}$).
- The ASCII numeric value of a character `x` is denoted 0'x, as 0'a (97), 0'b (98), etc.
- Variables are sequences of letters, numbers, and/or underscores beginning with an upper-case letter or the underscore character.

Simplified Prolog Conventions

- Compound terms consists of a functor, which must be an atom, followed immediately by an opening parenthesis, a sequence of terms separated by commas, and a closing parenthesis.
- Comments are of two types:

```
% Single line

/* Multiple

   lines

*/
```

Rules

- Enable to derive a new property or relation from a set of existing ones
- Used for hypotheses
- Headed horn clauses
- Right side: **antecedent** (***if*** part)
  - May be single term or conjunction
- Left side: **consequent** (***then*** part)
  - Must be single term

Rules

Facts

```
father(ulyses, telmachus).
mother(penelope, telmachus).
male(telmachus).
```

Results

```
Y = ulyses;
Y = penelope;
false.
```

Rules

```
son(X,Y):- father(Y,X), male (X).
son(X,Y):- mother(Y,X), male (X).
```

Query

```
?- son (telmachus, Y).
```

Rules

Facts

```
father(john, mike).
mother(susan, mike).
male(mike).
```

Rules

```
son(X,Y):- father(Y,X), male (X).
son(X,Y):- mother(Y,X), male (X).
```

Query

```
?- son(X, Y).
```

Results

```
X = mike,
Y = john ;

X = mike,
Y = susan.
true.
```

Rules

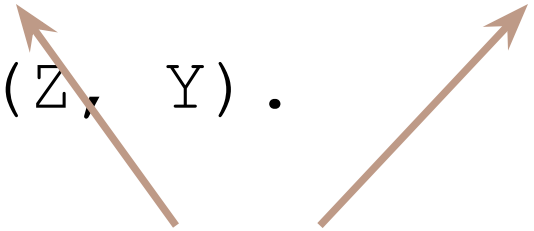
- Flexible to deduce new information from a set of facts
- Somebody is a parent if one is either a mother or a father

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

Rules

- Rules can also call other rules
- A grandparent is the parent of a parent

```
grandparent (X, Y):-
   parent (X, Z), parent (Z, Y).
```



Rules

- We can generalize the `grandparent/2` predicate and can write `ancestor/2`
- We use two rules, one of them being recursive

```
ancestor (X, Y):- parent (X, Y).
ancestor (X, Y):- parent (X, Z),  ancestor (Z, Y).
```

Rules

- Sample query about the ancestors of Hermione.

```
?- ancestor (X, hermione).
X= menelaus;
X= helen;
X= atreus;
false.
```

Clauses

- General term for **facts** and **rules**
- A predicate is defined by a set of clauses with the same principal functor and arity
- Conjunction use comma (`,`)
- Disjunction use semi-colon (`;`)
  - Scarcely used because of readability

```
P:-
Q
;
R
```

is equivalent to

```
P:- Q.
P:- R.
```

Arithmetic

- Equality (`=`)
  - Infix operator that checks equality of operands
  - May cause instantiation of some variables
- Matching (`=:=`)
  - Simply perform arithmetic operations
  - No instantiation is done
- Assignment (`is`)
  - Takes expression on the right side, evaluate it, then unifies it with the variable at the left

Arithmetic

- Examples:

```
?- 3+4 = 4+3.
false.
?- 3+4 =:= 4+3.
true.
?- X is 4 + 3.
X = 7.
```

Arithmetic

Arithmetic Operator	Meaning
+	Addition
-	Subtraction
/	Floating point Division
*	Multiplication
**	Power
//	Integer Division
Mod	Modulo, remainder of integer division

Arithmetic

Relational Operator	Meaning
X > Y	X is greater than Y
X < Y	X is less than Y
X > = Y	X is greater than or equal to Y
X = < Y	X is less than or equal to Y
X =: = Y	X & Y values are equal
X \ ± Y	X is not equal to Y

Arithmetic

- Examples:

```
?- X is 3 mod 2.
X = 1.
?- 4+3 > 3*5.
false.
```