Introduction to Logic Programming (Prolog)

Tutorial for

Programming Languages Laboratory (CS 431)

July to November -2016

Instructor: Dr. Samit Bhattacharya

Indian Institute of Technology Guwahati

Logic Programming

- There is a seeming lack of cohesion in computing approaches
  - ✓E.g. Turing machine model of computation vs relational algebra in database query
- Driving force behind logic programming
  - ✓Single formalism suffices for both logic and computation
  - ✓Logic subsumes computation
- Cohesion in LP approach
  - ✓Propositional logic → first-order logic → higher-order logic → model logic

LP-Basics

- Propositional logic (simplest of all)

  Proposition (premise): If it is raining it is cloudy
  Proposition (premise): It is raining
  Proposition (conclusion): It is cloudy

- Conclusion obtained by applying *modus ponens* (inference rules) on the premises ($P \rightarrow Q$; P is asserted to be true, so therefore Q must be true) - uses propositional calculus

LP-Basics

- First order predicate logic – allows quantification with variables (variable values can range over a *domain of discourse*)

  PL: Socrates is a man

  FOPL: X is a man

- We are concerned about the deductive system using FOPL formulae (well-formed formulae)
  - ✓ Inference using resolution refutation

LP-Basics

All mangoes are fruits
Alphonso is a mango
Therefore, Alphonso is fruit

More generally, in terms of wffs using FOPL notation

$\forall x, P(x) \Rightarrow Q(x)$
$P(a)$
Therefore, $Q(a)$

LP-Basics

- We can recast the inferencing process as follows to establish the same using resolution refutation

  $\neg P(x) \vee Q(x)$
  P(a)
  Therefore, Q(a)

- Rules
  - ✓ Find two clauses containing the same predicate, where it is negated in one clause but not in the other.
  - ✓ Perform a unification on the two predicates.
  - ✓ If any unbound variables which were bound in the unified predicates also occur in other predicates in the two clauses, replace them with their bound values (terms) there as well.
  - ✓ Discard the unified predicates, and combine the remaining ones from the two clauses into a new clause, also joined by the "V" operator.

LP-Basics

- Apply rules to the example
  - ✓Predicate P in the first clause in negated form and non-negated in the second clause; X is unbound variable and a is a bound value
  - ✓Unification on the two predicates resulting in *X substituted by a*
  - ✓Discard the unified predicate
  - ✓Apply substitution on the remaining clause (Q(X)) results in Q(a) [the conclusion] – hence proved

LP-Basics

- There are higher order logic also (need not bother about those here)
- We shall do some assignments in Prolog, which works on the idea of FOPL and resolution refutation

LP Perspectives

- There are many (overlapping) perspectives on logic programming
  - ✓ Computations as Deduction
  - ✓ Theorem Proving
  - ✓ Non-procedural Programming
  - ✓ Algorithms minus Control
  - ✓ A Very High Level Programming Language
  - ✓ A Procedural Interpretation of Declarative Specifications

Computation as Deduction

- Logic programming offers a slightly different paradigm for computation:

  COMPUTATION IS LOGICAL DEDUCTION

- It uses the language of logic to express data and programs.

  For all X and Y, X is the father of Y if

  X is a parent of Y and the gender of X is male.

Theorem Proving

- Logic Programming uses the notion of an *automatic theorem prover* as an interpreter
  - ✓ The theorem prover derives a desired solution from an initial set of axioms

- Note that the proof must be a "constructive" one so that more than a true/false answer can be obtained

  E.g. The answer to

  exists x such that x = sqrt(16)

  should be

  x = 4 or x = -4

  rather than

  true

Non-procedural Programming

- Logic Programming languages are non-procedural programming languages
- A non-procedural language is one in which one specifies WHAT needs to be computed but not HOW it is to be done
- That is, one specifies
  - ✓ the set of objects involved in the computation
  - ✓ the relationships which hold between them
  - ✓ the constraints which must hold for the problem to be solved
- And leaves it up the language interpreter or compiler to decide HOW to satisfy the constraints

Algorithms Minus Control

- Nikolas Wirth (architect of Pascal) used the following slogan as the title of a book:
  - ✓ Algorithms + Data Structures = Programs
- Bob Kowalski offers a similar one to express the central theme of logic programming:
  - ✓ Algorithms = Logic + Control
- We can view the LOGIC component as:
  - ✓ A specification of the essential logical constraints of a particular problem
- and CONTROL component as:
  - ✓ Advice to an evaluation machine (e.g. an interpreter or compiler) on how to go about satisfying the constraints)

A Very High Level Language

- A good programming language should not encumber the programmer with non-essential details.
- The development of programming languages has been toward freeing the programmer of more and more of the details...
  - ✓ ASSEMBLY LANGUAGE: symbolic encoding of data and instructions.
  - ✓ FORTRAN: allocation of variables to memory locations, register saving, etc.
  - ✓ JAVA: Platform specifics
  - ✓ ....
- Logic Programming Languages are a class of languages which attempt to free us from having to worry about many aspects of explicit control.

A Procedural Interpretation of Declarative Specifications

- One can take a logical statement like the following:
  - ✓ *For all X and Y, X is the father of Y if X is a parent of Y and*
  - ✓ *the gender of X is male.*

- Which would be expressed in an LP language as
  - ✓ *father(X,Y) :- parent(X,Y), gender(X,male).*

- And interpret it in two slightly different ways
  - ✓ **declaratively** - as a statement of the truth conditions which must be true if a father relationship holds.
  - ✓ **procedurally** - as a description of what to do to establish that a father relationship holds.

LP Languages

- Work initiated to deal with representational issues in AI (1960s and 70s)

- Planner (MIT) – earliest language (procedural paradigm)
  - ✓ Gave rise to many languages (e.g. Popler, Conniver, QLISP, Ether)

- Prolog (one of the popular languages)
  - ✓ First system by Alain Colmerauer & Philippe Roussel (1972)

- Prolog gave rise to many new languages
  - ✓ Fril, Gödel, Mercury, Oz, Visual Prolog, λProlog …

SWI-Prolog

- SWI-Prolog is a good, standard Prolog for Windows and Linux
- It's licensed under GPL, therefore free Downloadable from:

  http://www.swi-prolog.org/

Syllogisms (logical reasoning) in Prolog

Syllogism	Prolog
Socrates is a man.	man(socrates).
All men are mortal.	mortal(X) :- man(X).
Is Socrates mortal?	?- mortal(socrates).

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).
- Queries have the same form as facts

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
  - ✓ Double-clicking on your .pl file, *or*
  - ✓ Double-clicking on the Prolog application and “consulting” your file at the ?- prompt:

```
?- consult('C:\\My Programs\\adv.pl').
```

- Then, ask your question at the prompt:
  - ✓ ?- mortal(socrates).
- Prolog responds:
  - ✓ Yes

Prolog Basics

- Pure Prolog based on Horn clause (Alfred Horn, 1951)
- Execution of a Prolog program is initiated by the user's posting of a single goal, called the query
- Prolog engine tries to find a resolution refutation of the negated query (Selective Linear Definite or SLD resolution method - Robert Kowalski)

Prolog as Theorem Prover

- Prolog's "Yes" means "I can prove it" --
  Prolog's "No" means "I can't prove it"

```
?- mortal(plato).
      No
```

- This is the closed world assumption: the Prolog program knows everything it needs to know
- Prolog supplies values for variables when it can

```
?- mortal(X).
      X = socrates
```

Structure of Programs

- Programs consist of procedures
- Procedures consist of clauses
- Each clause is a fact or a rule
- Programs are executed by posing queries

Example

Predicate

Procedure for elephant

Facts

Clauses

```
elephant(george).
elephant(mary).
elephant(X) :- grey(X), mammal(X),
                hasTrunk(X).
```

Rule

Example

Queries

Replies

```
?- elephant(george).
yes
?- elephant(jane).
no
```

Clauses: Facts and Rules

‘if’
‘provided that’
‘turnstile’

Head **:-** *Body*. This is a rule.

Head. This is a fact.

Full stop at the end.

Body of a (rule) clause contains goals.

Head $\rightarrow$ likes(mary, X)

Body $\rightarrow$ human(X), honest(X).

```
likes(mary, X)   :-   human(X), honest(X).
```

Goals (comma indicates AND relation)

Interpretation of Clauses

Clauses can be given a declarative reading or a procedural reading.

Form of clause: H :- G_1, G_2, …, G_n.

Declarative reading: "That H is provable follows from goals G_1, G_2, …, G_n being provable."

Procedural reading: "To execute procedure H, the procedures called by goals G_1, G_2, …, G_n are executed first."

Example 1 : Food table shows the facts, rules, goals and their English meanings.

Facts	English meanings
food(burger).	// burger is a food
food(sandwich).	// sandwich is a food
food(pizza).	// pizza is a food
lunch(sandwich).	// sandwich is a lunch
dinner(pizza).	// pizza is a dinner

Rules	
meal(X) :- food(X).	// Every food is a meal OR Anything is a meal if it is a food

Queries / Goals	
?- food(pizza).	// Is pizza a food?
?- meal(X), lunch(X).	// Which food is meal and lunch?
?- dinner(sandwich).	// Is sandwich a dinner?

Example 2 : Student-Professor relation table shows the facts, rules, goals and their English meanings.

Facts	English meanings
studies(charlie, csc135).	// charlie studies csc135
studies(olivia, csc135).	// olivia studies csc135
studies(jack, csc131).	// jack studies csc131
studies(arthur, csc134).	// arthur studies csc134
teaches(kirke, csc135).	// kirke teaches csc135
teaches(collins, csc131).	// collins teaches csc131
teaches(collins, csc171).	// collins teaches csc171
teaches(juniper, csc134).	// juniper teaches csc134

Rules	
professor(X, Y) :- teaches(X, C), studies(Y, C).	// X is a professor of Y if X teaches C and Y studies C.

Queries / Goals	
?- studies(charlie, What).	// charlie studies what? OR What does charlie study?
?- professor(kirke, Students).	// Who are the students of professor kirke.

Syntax I: Data Types

- Called *term*
- Four types
  - Atoms
  - Numbers
  - Variables
  - Compound terms

Syntax I: Structures (compound term)

- A structure consists of a name (functor) and zero or more arguments (called *arity*).
  - Omit the parentheses if there are no arguments

- Example structures:
  - sunshine
  - man(socrates)
  - path(garden, south, sundial)

Syntax II: Clauses

- Prolog captures relations in the form of clauses
- Restricted to Horn clause (Alfred Horn, 1951)
- Two types
  - Base clause (facts)
  - Nonbase clause (rules)

Syntax II: Base Clauses

- A *base clause* is just a structure, terminated with a period.
- A base clause represents a simple fact.
- Example base clauses:
  - ✓ debug_on.
  - ✓ loves(john, mary).
  - ✓ loves(mary, bill).

Syntax III: Nonbase Clauses

- A nonbase clause is a structure, a turnstile :- (meaning "if"), and a list of structures.

- Example nonbase clauses:
  - ✓ mortal(X) :- man(X).
  - ✓ mortal(X) :- woman(X).
  - ✓ happy(X) :- healthy(X), wealthy(X), wise(X).

- The comma between structures means "and"

Syntax IV: Predicates

- A *predicate* is a collection of clauses with the same *functor* (name) and *arity* (number of arguments)

```
loves(john, mary).
loves(mary, bill).
loves(chuck, X) :- female(X), rich(X).
```

Syntax V: Programs

- A *program* is a collection of predicates.
- Predicates can be in any order.
- Clauses within a predicate are used in the order in which they occur.

Syntax VI: Variables and atoms

- Variables begin with a capital letter:
  X, Socrates, X_result
- Atoms do *not* begin with a capital letter:
  x, socrates
- Atoms containing special characters, or beginning with a capital letter, must be enclosed in single quotes:

'C:\\My Documents\\examples.pl'

Syntax VII: Strings are atoms

- In a quoted atom, a single quote must be doubled or backslashed:
  'Can''t, or won\'t?'
- Backslashes in file names must also be doubled:
  'C:\\My Documents\\examples.pl'

Common problems

- Capitalization is *meaningful*!
- No space is allowed between a functor and its argument list:
  man(socrates), *not* man (socrates).
- Double quotes indicate a list of ASCII character values, *not* a string
- Don’t forget the period! (But you can put it on the next line.)

Example

```
mother_child(trude, sally).

father_child(tom, sally).
father_child(tom, erica).
father_child(mike, tom).

sibling(X, Y) :- parent_child(Z, X), parent_child(Z, Y).

parent_child(X, Y) :- father_child(X, Y).
parent_child(X, Y) :- mother_child(X, Y).
```

```
?- sibling(sally, erica).
Yes
```

How it Works

- Initially, the only matching clause-head for sibling(sally, erica) is the first one, so proving the query is equivalent to proving the body of that clause
- Clause proved with appropriate variable bindings, i.e., the conjunction (parent_child(Z,sally), parent_child(Z,erica)).
- The next goal to be proved is the leftmost one of this conjunction, i.e., parent_child(Z, sally).
- Two clause heads match this goal. The system creates a choice-point and tries the first alternative, whose body is father_child(Z, sally).
- This goal can be proved using the fact father_child(tom, sally), so the binding Z = tom is generated, and the next goal to be proved is the second part of the above conjunction: parent_child(tom, erica).
- Again, this can be proved by the corresponding fact. Since all goals could be proved, the query succeeds.

Unification

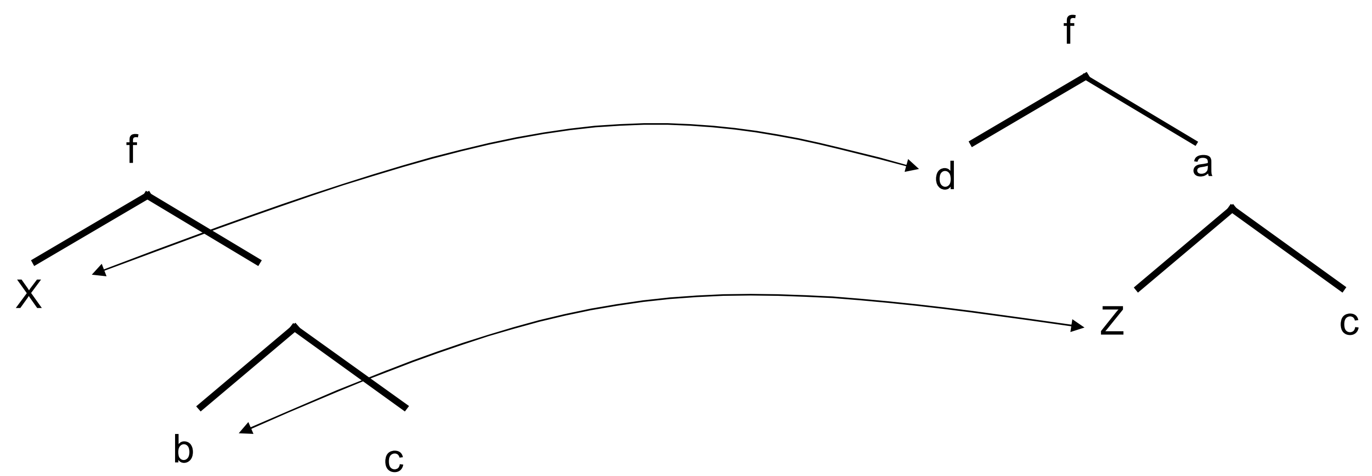
- The way Prolog matches two terms (we have two terms and we want to see if they can be made to represent the same structure) – used for reasoning

- Two *terms* unify if substitutions can be made for any variables in the terms so that the terms are made identical. If no such substitution exists, the terms do not unify

- The Unification Algorithm proceeds by recursive descent of the two terms
  - ✓ Constants unify if they are identical
  - ✓ Variables unify with any term, including other variables
  - ✓ Compound terms unify if their functors and components unify

Examples

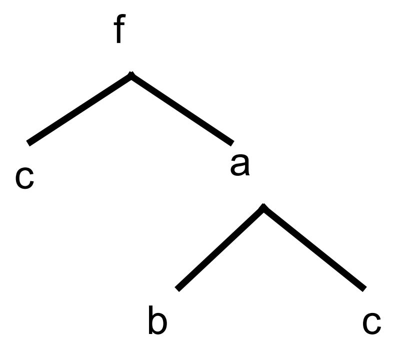
The terms f(X, a(b, c)) and f(d, a(Z, c)) unify.

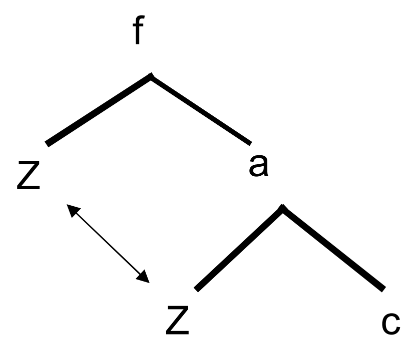


The terms are made equal if d is substituted for X, and b is substituted for Z. We also say X is instantiated to d and Z is instantiated to b, or X/d, Z/b.

Examples

The terms f(c, a(b, c)) and f(Z, a(Z, c)) do not unify.





No matter how hard you try, these two terms cannot be made identical by substituting terms for variables.

List : Useful Data Structure in Prolog

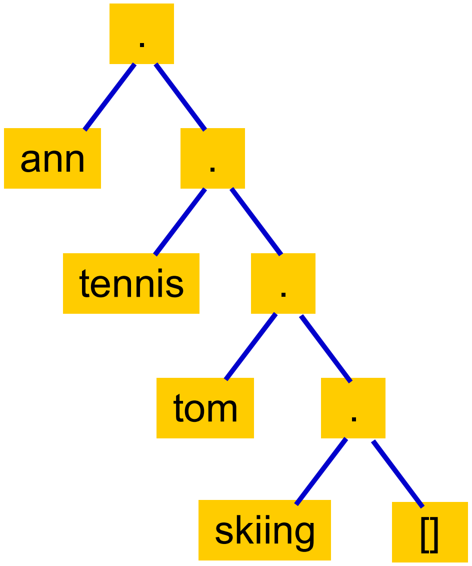
- A list is an (ordered) sequence of any number of items (special case of composite term).
- For example:
  - [ann, tennis, tom, skiing]
- A list is either empty or non-empty.
  - Empty: []
  - Non-empty:
    - The first term, called the head of the list
    - The remaining part of the list, called the tail
    - Example: [ann, tennis, tom, skiing]
      - Head: ann
      - Tail: [tennis, tom, skiing]

List : Data Structure

- In general, the head can be anything (for example: a tree or a variable); the tail has to be a list

- The head and the tail are then combined into a structure by a special functor

  .(head, Tail)



Example:

L = .(ann, .(tennis, .(tom, .(skiing, []))))).
L = [ann, tennis, tom, skiing].
are the same in Prolog.

Some operations on lists

- The most common operations on lists are
  - ✓ Checking whether some object is an element of a list, which corresponds to checking for the set membership
  - ✓ Concatenation of two lists, obtaining a third list, which may correspond to the union of sets
  - ✓ Adding a new object to a list
  - ✓ deleting some object from it

Membership

- The membership relation:

  member(X, L)

  where X is an object and L is list.
- The goal **member(X, L)** is true if X occurs in L.
- For example:

  member(b, [a, b, c]) is true

  member(b, [a, [b, c]]) is not true

  member([b, c] , [a, [b, c]]) is true

Concatenation

- The concatenation relation:

  conc(L1, L2, L3)

  here L1 and L2 are two lists, and L3 is their concatenation.

- For example:

  conc([a, b], [c, d], [a, b, c, d]) is true

  conc([a, b], [c, d], [a, b, a, c, d]) is not true

Adding an item

- To add an item to a list, it is easiest to put the new item in front of the list so that it become the new head
- If X is the new item and the list to which X is added is L then the resulting list is simply: **[X|L]**
- So we actually need no procedure for adding a new element in front of the list

Deleting an item

- Deleting an item X form a list L can be programmed as a relation:

  del(X, L, L1)

  where L1 is equal to the list L with the item X removed

- Two cases of delete relation

  (1) If X is the head of the list then the result after the deletion is the tail of the list

  (2) If X is in the tail then it is deleted from there

```
del( X, [X| Tail], Tail).
del( X, [Y| Tail], [Y|Tail1]) :- del( X, Tail, Tail1).
```

Search

• Logic Programming 'procedure' can either fail or succeed. If it succeeds, it may have computed some additional information (conveyed by instantiating variables).

- •Question: What if it fails.....? Answer: find another way to try to make it succeed.
- •Most logic programming languages use a simple, fixed search strategy to try alternatives:

-if a goal succeeds and there are more goals to achieve, then remember any untried alternatives and go on to the next goal.

– if a goal succeeds and there are no more goals to achieve, then stop with success.

– if a goal fails and there are alternate ways to solve it, then try the next one.

– if a goal fails and there are no alternate ways to solve it and there is a previous goal, then propagate failure back to the previous goal.

– if a goal fails and there are no alternate ways to solve it and no previous goal then stop with failure.

Cuts

- A cut prunes or “cuts out” and unexplored part of a Prolog search tree.
- Cuts can therefore be used to make a computation more efficient by eliminating futile searching and backtracking.
- Cuts can also be used to implement a form of negation

References

The following references may be helpful

[1] W. F. Clocksin and C. S. Mellish, *Programming in Prolog*, 5th Ed. , Springer, 2004.

[2] https://www.ida.liu.se/~ulfni53/lpp/bok/bok.pdf (e-Book)

[3] http://www.cs.ru.nl/P.Lucas/teaching/CS3510/intro-prol.pdf

END