

# **Logic Programming Paradigm**

# Logic Programming Paradigm

- **Sub Topics**

- Definition - S1-SLO1
- First-class function, Higher-order function, Pure functions, Recursion-S1-SLO2
- *Packages: Kanren, SymPy – S2-SLO 1*
- *PySWIP, PyDatalog – S2-SLO 2*
- *Other languages: Prolog, ROOP, Janus S3-SLO 1*

# Introduction

- It can be an abstract model of computation.
- Solve logical problems like puzzles, series
- Have **knowledge base** which we know before and along with the question
- you specify knowledge and how that knowledge is to be applied through a series of rules
- in logic programming, the common approach is to apply the methods of **resolution** and **unification**
- Knowledge is given to machine to produce result. Exp : **AL and ML code**

# Introductions

- The *Logical Paradigm* takes a **declarative approach** to problem-solving.
- Various **logical assertions** about a situation are made, establishing all known facts.
- Then queries are made.
- The role of the computer becomes maintaining data and logical deduction.
- Programs are written in the language of some logic..
- Execution of a logic program is a theorem proving process; that is, computation is done by logic inferences
- Prolog ( PROgramming in LOGic) is a ) is a representative logic language
- **Exp** : Prolog, ROOP, Janus

# What is a logic ?

- A logic is a language.
- It has syntax and semantics.
- More than a language, it has inference rules.
- **Syntax:**
  - The rules about how to form formulas;
  - This is usually the easy part of a logic.
- **Semantics:**
  - about the meaning carried by the formulas,
  - mainly in terms of logical consequences.
- **Inference rules**
  - Inference rules describe correct ways to derive

# Features of Logical paradigms

- Computing takes place over the domain of all terms defined over a “universal” alphabet.
- Values are assigned to variables by means of automatically generated substitutions, called most general unifiers.
- These values may contain variables, called logical variables.
- The control is provided by a single mechanism: automatic backtracking.
- **Strength** - simplicity and Conciseness
- **Weakness** - has to do with the restrictions to one control mechanism and the use of a single data type.

# History of Logic Programming (LP)

- Logic Programming has roots going back to early AI researchers like John McCarthy in the 50s & 60s
- Alain Colmerauer (France) designed Prolog as the first LP language in the early 1970s
- Bob Kowalski and colleagues in the UK evolved the language to its current form in the late 70s
- It's been widely used for many AI systems, but also for systems that need a fast, efficient and clean rule based engine
- The prolog model has also influenced the database community –
  - Exp datalog

# General Overview

- Programs written in logic languages consist of declarations that are actually statements, or propositions, in symbolic logic.
- One of the essential characteristics of logic programming languages is their semantic
- The basic concept of this semantics is that 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.



# Concepts of logic paradigm and theoretical background

- Includes both theoretical and fully implemented languages
- They all share the idea of interpreting computation as logical deduction.
- Notation Algorithm = Logic + Control
- Logic Programming uses facts and rules for solving the problem.
- That is why they are called the building blocks of Logic Programming.
- **Facts**
  - Actually, every logic program needs facts to work with so that it can achieve the given goal.
  - Facts basically are true statements about the program and data.
  - Exp : Delhi is the capital of India.
- **Rules**
  - are the constraints which allow us to make conclusions about the problem domain.
  - Basically written as logical clauses to express various facts.
  - Exp if we are building any game then all the rules must be defined.

# Parts of Logical programming

1. A series of definitions/declarations that define the problem domain
2. Statements of relevant facts
3. Statement of goals in the form of a query

# Advantages

- The system solves the problem, so the programming steps themselves are kept to a minimum;
- Proving the validity of a given program is simple.

# 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  $X, Y$ :  $X$  is the father of  $Y$  if  $X$  is a parent of  $Y$  and  $X$  is male
- Current logic programming languages use first order logic (FOL) which is often referred to as first order predicate calculus (FOPC).
- The first order refers to the constraint that we can quantify (i.e. generalize) over objects, but not over functions or relations.
- We can express "All elephants are mammals" but not
- "for every continuous function  $f$ , if  $n < m$  and  $f(n) < 0$  and  $f(m) > 0$  then there exists an  $x$  such that  $n < x < m$  and  $f(x) = 0$ "

# 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.
- 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  $x = \text{sqrt}(16)$
  - 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 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 to the language interpreter or compiler to decide how to satisfy the constraints

# A Declarative Example

- Here's a simple way to specify what has to be true if  $X$  is the smallest number in a list of numbers  $L$ 
  1.  $X$  has to be a member of the list  $L$
  2. There can be list member  $X_2$  such that  $X_2 < X$

We need to say how we determine that some  $X$  is a member of a list

1. No  $X$  is a member of the empty list
2.  $X$  is a member of list  $L$  if it is equal to  $L$ 's head
3.  $X$  is a member of list  $L$  if it is a member of  $L$ 's tail.



# Use logic to do reasoning

- Example: Given information about fatherhood and motherhood, determine grand parent relationship
- E.g. Given the information called facts
  - John is father of Lily
  - Kathy is mother of Lily
  - Lily is mother of Bill
  - Ken is father of Karen
- Who are grand parents of Bill?
- Who are grand parents of Karen?

# Example

## domains

being = symbol

## predicates

animal(being) % all animals are beings

dog(being) % all dogs are beings

die(being) % all beings die

## clauses

animal(X) :- dog(X) % all dogs are animals

dog(fido). % fido is a dog

die(X) :- animal(X) % all animals die

# Logic Operators

Name	Symbol	Example	Meaning
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
universal	$\forall X.P$		For all X, P is true
existential	$\exists X.P$		There exists a value of X such that P is true

- Equivalence means that both expressions have identical truth tables
- Implication is like an if-then statement
  - if a is true then b is true
  - note that this does not necessarily mean that if a is false that b must also be false
- Universal quantifier says that this is true no matter what x is
- Existential quantifier says that there is an X that fulfills the statement

$\forall X.(\text{woman}(X) \supset \text{human}(X))$   
 – if X is a woman, then X is a human

$\exists X.(\text{mother}(\text{mary}, X) \cap \text{male}(X))$   
 – Mary has a son (X)

# Example Statements

Consider the following knowledge:

Bob is Fred's father  $\square$  father(Bob, Fred)

Sue is Fred's mother  $\square$  mother(Sue, Fred)

Barbara is Fred's sister  $\square$  sister(Barbara, Fred)

Jerry is Bob's father  $\square$  father(Jerry, Bob)

And the following rules:

A person's father's father is the person's grandfather

A person's father or mother is that person's parent

A person's sister or brother is that person's sibling

If a person has a parent and a sibling, then the sibling has the same parent

These might be captured in first-order predicate calculus as:

$\forall x, y, z$  : if father(x, y) and father(y, z) then grandfather(x, z)

$\forall x, y$  : if father(x, y) or mother(x, y) then parent(x, y)

$\forall x, y$  : if sister(x, y) or brother(x, y) then sibling(x, y) and sibling(y, x)

$\forall x, y, z$  : if parent(x, y) and sibling(y, z) then parent(x, z)

We would rewrite these as

$\text{grandfather}(x, z) \subset \text{father}(x, y) \text{ and } \text{father}(y, z)$

$\text{parent}(x, y) \subset \text{father}(x, y)$

$\text{parent}(x, y) \subset \text{mother}(x, y)$                       etc

# Kanren

- It provides us a way to simplify the way we made code for business logic.
- It lets us express the logic in terms of rules and facts.
- The following command will help you install kanren –
  - `pip install kanren`

# Matching mathematical expressions

```
from kanren import run, var, fact
from kanren.assoccomm import eq_assoccomm as eq
from kanren.assoccomm import commutative, associative
add = 'add'
mul = 'mul'
fact(commutative, mul)
fact(commutative, add)
fact(associative, mul)
fact(associative, add)
a, b = var('a'), var('b')
Original_pattern = (mul, (add, 5, a), b)
exp1 = (mul, 2, (add, 3, 1))
exp2 = (add, 5, (mul, 8, 1))
print(run(0, (a,b), eq(original_pattern, exp1)))
print(run(0, (a,b), eq(original_pattern, exp2)))
```

- **Output**

- ((3,2))
- ()

- **Reference :**

- [https://www.tutorialspoint.com/artificial\\_intelligence\\_with\\_python/artificial\\_intelligence\\_with\\_python\\_logic\\_programming.htm](https://www.tutorialspoint.com/artificial_intelligence_with_python/artificial_intelligence_with_python_logic_programming.htm)

# SymPy Overview

## Reference:

- <https://www.geeksforgeeks.org/python-on-getting-started-with-sympy-module/>
- <https://docs.sympy.org/latest/tutorial/index.html>

# SymPy module

- [SymPy](#) is a Python library for symbolic mathematics.
- It aims to become a full-featured computer algebra system (CAS)
- While keeping the code as simple as possible in order to be comprehensible and easily extensible.
- SymPy is written entirely in Python.
- **Installing sympy module:**
  - **Pip install sympy**

## # Example

```
from sympy import * x = Symbol('x')
y = Symbol('y')
z = (x + y) + (x-y)
print("value of z is :" + str(z))
```

## Output:

value of z is :2\*x



# Find derivative, integration, limits, quadratic equation

```
# make a symbol
x = Symbol('x')
# take the derivative of sin(x)*e ^ x
ans1 = diff(sin(x)*exp(x), x)
print("derivative of sin(x)*e ^ x : ", ans1)
# Compute (e ^ x * sin(x)+ e ^ x * cos(x))dx
ans2 = integrate(exp(x)*sin(x) + exp(x)*cos(x), x)
print("indefinite integration is : ", ans2)
# Compute definite integral of sin(x ^ 2)dx
# in b / w interval of ? and ?? .
ans3 = integrate(sin(x**2), (x, -oo, oo))
print("definite integration is : ", ans3)
# Find the limit of sin(x) / x given x tends to 0
ans4 = limit(sin(x)/x, x, 0)
print("limit is : ", ans4)
# Solve quadratic equation like, example : x ^ 2-2 = 0
ans5 = solve(x**2 - 2, x)
print("roots are : ", ans5)
```

## Output :

derivative of sin(x)\*e^x : exp(x)\*sin(x) + exp(x)\*cos(x)  
indefinite integration is : exp(x)\*sin(x)  
definite integration is : sqrt(2)\*sqrt(pi)/2  
limit is : 1  
roots are : [-sqrt(2), sqrt(2)]

## SymPy Live Shell Demo

<https://docs.sympy.org/latest/tutorial/intro.html#what-is-symbolic-computation>

# Datalog Concepts

- pyDatalog is a powerful language with very few syntactic elements, mostly coming from Python :
- Variables and expressions
- Loops
- Facts
- Logic Functions and dictionaries
- Aggregate functions
- Literals and sets
- Tree, graphs and recursive algorithms
- 8-queen problem

## Reference

- <https://sites.google.com/site/pydatalog/Online-datalog-tutorial>

# PySwip Introduction

- PySwip is a Python - SWI-Prolog bridge enabling to query [SWI-Prolog](#) in your Python programs.
- It features an (incomplete) SWI-Prolog foreign language interface, a utility class that makes it easy querying with Prolog and also a Pythonic interface.
- Since PySwip uses SWI-Prolog as a shared library and ctypes to access it,
- it doesn't require compilation to be installed.
- Reference :
  - <https://pypi.org/project/pyswip/>