Open Ended Program

Aim: Study of different logic programming tools. Implement Mini Project using one of the tools.

Logic programming is a type of programming paradigm which is largely based on formal logic. Any program written in a logic programming language is a set of sentences in logical form, expressing facts and rules about some problem domain. Major logic programming language families include Prolog, Answer set programming (ASP) and Datalog. In all of these languages, rules are written in the form of *clauses*:

H: B_1, …, B_n.

and are read declaratively as logical implications: H if B_1 and … and B_n.

H is called the *head* of the rule and B_1, …, B_n is called the *body*. Facts are rules that have no body, and are written in the simplified form: H.

Several types of Logic Programming Tools are:

Prolog:

- The programming language Prolog was developed in 1972 by Alain Colmerauer.
- Prolog is a general-purpose logic programming language associated with artificial intelligence and computational linguistics.
- Prolog has its roots in first-order logic, a formal logic, and unlike many other programming languages, Prolog is intended primarily as
- a declarative programming language
- The program logic is expressed in terms of relations, represented as facts and rules. A computation is initiated by running a query over these relations
- Prolog is well-suited for specific tasks that benefit from rule-based logical queries such as searching databases, voice control systems, and filling templates.

Abductive logic programming:

- Abductive logic programming (ALP) is a high-level knowledge-
- representation framework that can be used to solve problems declaratively based on abductive reasoning.
- It extends normal logic programming by allowing some predicates to be incompletely defined, declared as abducible predicates.
- Problem solving is affected by deriving hypotheses on these abducible predicates (abductive hypotheses) as solutions of problems to be solved.
- These problems can be either observations that need to be explained (as in classical abduction) or goals to be achieved (as in normal logic programming).
- It can be used to solve problems in diagnosis, planning, natural language and machine learning. It has also been used to interpret negation as failure as a form of abductive reasoning.
- Abductive logic programs have three components, where:
    - P is a logic program of exactly the same form as in logic programming
    - A is a set of predicate names, called the abducible predicates
    - IC is a set of first-order classical formulae.

Metalogic programming:

- The objective of this study is to show how a metalogic programming language, i.e. a logic language with fully developed, built-in metalevel features, is a suitable tool for formalization and use of several forms of reasoning.
- The role of metaknowledge in expressing auxiliary inference strategies is emphasized.
- The proposed approaches are dealing with domain-specific concepts (formalization of a kind of analogical reasoning) and the other related to domain-independent sentences (expression and composing general properties of relations).
- The concept of preprocessing metaknowledge at program-consultation time for a significant improvement in efficiency is introduced and applied to the problem areas considered.

Constraint logic programming:

- Constraint logic programming is a form of constraint programming, in which logic programming is extended to include concepts from constraint satisfaction.
- A constraint logic program is a logic program that contains constraints in the body of clauses.
- An example of a clause including a constraint is clause, X+Y>0 is a constraint; A (X, Y), B(X), and programming.
- As in regular logic programming, programs are queried about the provability of a goal, which may contain constraints in addition to literals.
- A proof for a goal is composed of clauses whose bodies are satisfiable constraints and literals that can in turn be proved using other clauses. Execution is performed by an interpreter, which starts from the goal and recursively scans the clauses trying to prove the goal.
- Constraints encountered during this scan are placed in a set called constraint store. If this set is found out to be unsatisfiable, the interpreter backtracks, trying to use other clauses for proving the goal. In practice, satisfiability of the constraint store may be checked using an incomplete algorithm, which does not always detect inconsistency.

Concurrent logic programming:

- Concurrent logic programming is a variant of logic programming in which programs are sets of guarded Horn clauses of the form: $H: G_1, \ldots, G_n \mid B_1, \ldots, B_n$.
- The conjunction $G_1, \ldots, G_n$ is called the guard of the clause, and | is the commitment operator.
- Declaratively, guarded Horn clauses are read as ordinary logical implications: H if G_1 and … and G_n or B_1 and … and B_n.
- However, procedurally, when there are several clauses whose heads H match a given goal, then all of the clauses are executed in parallel, checking whether their guards G_1,
- …, G_n hold.
- If the guards of more than one clause hold, then a committed choice is made to one of the clauses, and execution proceeds with the sub goals $B_1, \ldots, B_n$ of the chosen clause.
- These sub goals can also be executed in parallel. Thus, concurrent logic programming implements a form of "don't care nondeterminism", rather than "don't know nondeterminism".

Concurrent constraint logic programming:

- Concurrent constraint logic programming combines concurrent logic programming and constraint logic programming, using constraints to control concurrency.
- A clause can contain a guard, which is a set of constraints that may block the applicability of the clause.

- When the guards of several clauses are satisfied, concurrent constraint logic programming makes a committed choice to the use of only one.
- It is a version of constraint logic programming aimed primarily at programming concurrent processes rather than (or in addition to) solving constraint satisfaction problems.
- Goals in constraint logic programming are evaluated concurrently; a concurrent process is therefore programmed as the evaluation of a goal by the interpreter.

Inductive logic programming:

- Inductive logic programming is concerned with generalizing positive and negative examples in the context of background knowledge: machine learning of logic programs.
- Recent work in this area, combining logic programming, learning and probability, has given rise to the new field of statistical relational learning and probabilistic inductive logic programming.
- Inductive logic programming (ILP) is a subfield of machine learning which uses logic programming as a uniform representation for examples, background knowledge and hypotheses. Given an encoding of the known background knowledge and a set of examples represented as a logical database of facts, an ILP system will derive a hypothesized logic program which entails all the positive and none of the negative examples.
- Schema: *positive examples* + *negative examples* + *background knowledge* ⇒ *hypothesis*. Higher-order logic programming:
- Several researchers have extended logic programming with higher-order programming features derived from higher-order logic, such as predicate variables. Such languages include the Prolog extensions HiLog and λProlog.
- Higher-order programming is a style of computer programming that uses software components, like functions, modules or objects, as values.
- It is usually instantiated with, or borrowed from, models of computation such as lambda calculus which make heavy use of higher-order functions.
- For example, in higher-order programming, one can pass functions as arguments to other functions and functions can be the return value of other functions (such as in macros or for interpreting).
- This style of programming is mostly used in functional programming, but it can also be very useful in object-oriented programming.
- A slightly different interpretation of higher-order programming in the context of object-oriented programming are higher order messages, which let messages have other messages as arguments, rather than functions.

Linear logic programming:

- Basing logic programming within linear logic has resulted in the design of logic programming languages that are considerably more expressive than those based on classical logic.
- Horn clause programs can only represent state change by the change in arguments to predicates.
- In linear logic programming, one can use the ambient linear logic to support state change.
- Some early designs of logic programming languages based on linear logic include LO [Andreoli & Pareschi, 1991], Lolli, ACl, and Forum [Miller, 1996]. Forum provides a goal-directed interpretation of all of linear logic.
- LLP is a logic programming language based on intuitionistic linear logic. LLP is a superset of Prolog and a subset of Lolli developed by Josh Hodas and Dale Miller.
- LLP system consists of: LLP to LLPAM (extended WAM) translator (written in Prolog) and

LLPAM emulator (written in C).

- LLP is distributed as free software for non-profit use. Object-oriented logic programming:
- F-logic extends logic programming with objects and the frame syntax.
- Logtalk extends the Prolog programming language with support for objects, protocols, and other OOP concepts. Highly portable, it supports most standard-compliant Prolog systems as backend compilers.
- Logtalk is an object-oriented logic programming language that extends and leverages the Prolog language with a feature set suitable for programming in the large.
    - It provides support for encapsulation and data hiding, separation of concerns and enhanced code reuse.
- Logtalk uses standard Prolog syntax with the addition of a few operators and directives.
- The Logtalk language implementation is distributed under an open source license and can run using a Prolog implementation (compliant with official and de facto standards) as the back-end compiler.

Transaction logic programming:

- Transaction logic is an extension of logic programming with a logical theory of state-modifying updates.
- It has both a model-theoretic semantics and a procedural one.
- An implementation of a subset of Transaction logic is available in the Flora-2 system.
- An extension of predicate logic, called Transaction Logic, is proposed, which accounts in a clean and declarative fashion for the phenomenon of state changes in logic programs and databases.
- Transaction Logic has a natural model theory and a sound and complete proof theory, but unlike many other logics, it allows users to program transactions.
- The semantics leads naturally to features whose amalgamation in a single logic has proved elusive in the past.
- These features include both hypothetical and committed updates, dynamic constraints on transaction execution, nondeterminism, and bulk updates.
- Finally, Transaction Logic holds promise as a logical model of hitherto non-logical phenomena, including so-called procedural knowledge in AI, and the behavior of object-oriented databases, especially methods with side effects.