Comparing Prolog and Lisp

The main difference between prolog and lisp is Prolog is a computer programming language that supports logic programming paradigm while Lisp is a computer program language that supports procedural, reflective and functional paradigms.

Lisp has symbolic and numeric atoms, lists and structure, property and association lists while Prolog has Symbolic and numeric atoms, predicate, lists and list structure. But they have some differences in syntax. In data transisting Lisp has parameter passing by value, function returns, binding of local and global variables and Prolog has variable binding manipulation through unification. Prolog has Variable scope scoping confined to a single rule or fact and Lisp has lexical or dynamic scoping of free variables, bound variables, local variables using Let blocks.

Token is the smallest element of a program that is meaningful to the compiler. Tokens are usually seperated by whitespace which can be “;” in lisp and “%” in prolog.

Lisp is an expression oriented language. Unlike most other languages, no distinction is made between “expressions” and “statements”; all code and data are written as expressions. When an expression is evaluated, it produces a value (in Common Lisp, possibly multiple values), which can then be embedded into other expressions. Each value can be any data type.

The use of parentheses is Lisp's most immediately obvious difference from other programming language families. As a result, students have long given Lisp nicknames such as Lost In Stupid Parentheses, or Lots of Irritating Superfluous Parentheses. However, the S-expression syntax is also responsible for much of Lisp's power: the syntax is simple and consistent, which facilitates manipulation by computer. However, the syntax of Lisp is not limited to traditional parentheses notation. It can be extended to include alternative notations. For example, XMLisp is a Common Lisp extension that employs the metaobject protocol to integrate S-expressions with the Extensible Markup Language . Relying on expressions gives language great flexibility. Because Lisp functions When written as lists, they can be processed just like data. This makes programs easy to write manipulating

other programs. Many Lisp dialects exploit this feature using macro systems, which enables extension of the language almost without limit.

In Prolog, the program logic is expressed in terms of relations and a calculation is started. It runs a query on these relationships. Relationships and queries are created using single data from Prolog kind, term. Relationships are defined by sentences.Given a query, the Prolog engine attempts to find a resulation refutation of the negated query. If the negated query can be refuted, for example, an instantiation for all free variables is found that makes the union of clauses and the singleton set consisting of the negated query false, it follows that the original query, with the found instantiation applied, is a logical consequence of the program. This makes Prolog (and other logic programming languages) particularly useful for database, symbolic mathematics, and language parsing applications. Because Prolog allows impure predicates, checking the truth value of certain special predicates may have some deliberate side effect, such as printing a value to the screen. Because of this, the programmer is permitted to use some amount of conventional imperative programming when the logical paradigm is inconvenient. It has a purely logical subset, called "pure Prolog", as well as a number of extralogical features.

An atom, in Prolog, means a single data item. It can be in different types. A string atom, like 'This is a string' or a symbol, like likes, john, and pizza, in likes(john, pizza). Atoms of this type must start with a lower case letter. They can include digits (after the initial lower-case letter) and the underscore character (_). the empty list []. This is a strange one: other lists are not atoms. If you think of an atom as something that is not divisible into parts (the original meaning of the word atom, though subverted by subatomic physics) then [] being an atom is less surprsing, since it is certainly not divisible into parts.strings of special characters, like <---> When using atoms of this type, some care is needed to avoid using strings of special characters with a predefined meaning, like the neck symbol :-, the cut symbol !, and various arithmetic and comparison operators. The available special characters for constructing this class of atom are: +, -, *, /, <, >, =, :, ., &, _, and ~. Numbers, in Prolog, are not considered to be atoms

The original LISP had two basic data types: atoms and lists. A list was a finite ordered sequence of elements, where each element is either an atom or a list, and an atom was a number or a symbol. A symbol was essentially a uniquely named element written in source code as an alphanumeric string and used as a variable name or data element in symbolic processing. For example, the list (FOO (BAR 1) 2) contains three elements: the symbol FOO, the list (BAR 1), and the number

The essential difference between atoms and lists was that atoms were constant and unique. Two atoms that appeared in different places in source code but were written in exactly the same way represented the same object, whereas each list was a separate object that could be changed independently of other lists and could be distinguished from other lists by comparison operators.

As more data types were introduced in later Lisp dialects, and programming styles evolved, the concept of an atom lost importance. Many dialects still retained the predicate atom for legacy compatibility, defining it true for any object which is not a cons.

A Lisp list is written with its elements separated by whitespace, and surrounded by parentheses. For instance, (1 2 foo) is a list whose elements are the three atoms 1, 2, and foo. These values are implicitly typed like they are respectively two integers and a Lisp-specific data type called a "symbol", and do not have to be declared as such. The empty list () is also represented as the special atom nil. This is the only entity in Lisp which is both an atom and a list. Expressions are written as lists, using prefix notation. The first element in the list is the name of a function, the name of a macro, a lambda expression or the name of a "special operator" (see below). The remainder of the list are the arguments. For example, the function list returns its arguments as a list, so the expression (list 1 2 (quote foo)) evaluates to the list (1 2 foo). The "quote" before the foo in the preceding example is a "special operator" which returns its argument without evaluating it. Any unquoted expressions are recursively evaluated before the enclosing expression is evaluated. For example, (list 1 2 (list 3 4)) evaluates to the list (1 2 (3 4)). Note that the third argument is a list; lists can be nested.

A list is a simple data structure commonly used in non-numerical programming. List consists of any number of items, for example, red, green, blue, white, dark. It will be represented as, [red, green, blue, white, dark]. Item list will be enclosed in square brackets.

A list may or may not be empty. In the first case, the list is simply written as a Prolog atom . In the second case, the list consists of two things the first item, called the head of the list and the remaining part of the list, called the tail.

Lisp is a computer programming language with a long history and a distinctive, fully parenthesized prefix notation. The programmer writes all program code in s-expressions or parenthesized lists. Furthermore, a function call or syntactic form can be written as a list with the function or operator's name first.

Lisp is an old, high-level programming language. The main objective of using Lisp is to represent mathematical notations for computer programs. Some popular Lisp dialects are Clojure, CommonLisp and Scheme. Moreover, it also helps to develop Artificial Intelligence (AI) applications.

Prolog is a logic programming language associated with Artificial Intelligence and computational linguistics. It has its roots in formal logic, which, unlike many other programming languages, is first-order logic. Prolog is mainly a declarative programming language. It is possible to express the program logic as a set of relations, facts and rules. Therefore, a computation can be initiated by running a query over these relations.

Prolog was one of the first logic programming languages. It helps various tasks such as theorem proving, expert systems, term rewriting, type systems, natural language processing and automated planning. It also helps to create GUIs, administrative and networked applications. Furthermore, Prolog is suitable for rule-based logical queries like voice control systems and searching databases.

Lisp is the second-oldest high-level programming language after. In contrast, Prolog is a logic programming language associated with artificial intelligence and computational linguistics. Lisp

supports functional, procedural, reflective and meta paradigms while Prolog supports logical programming paradigm.

Prolog is a programming language, and like for any programming language, there are many different ways to implement it. For example, we can compile a Prolog program to abstract or concrete machine code, or we can interpret a Prolog program with various techniques. The most popular implementation technique for Prolog is compiling it to abstract machine code. This is similar to the most common implementation methods of other languages like Java and Lisp. The Warren Abstract Machine is among the most well-known target machines for Prolog.

The time efficiency of any Prolog program depends a lot on the compiler or interpreter you are using, and available Prolog implementations differ significantly regarding performance. Some Prolog implementors have made heroic efforts to achieve peak performance in a large number of benchmarks.

An algorithm have the computational complexity. This is often referred to in terms of the order of magnitude of the function f(n) describing the number of computation steps required to process an input of size n , written as O(f(n)) . O(f(n)) tells how the cost of executing the algorithm grows with the size of the data set to which it is applied, as the size becomes large. Conventionally, n is used to denote the size of the data set, and the function O(f(n)) tells how computation time varies with n as n approaches infinity. In general, the function O(f(n)) omits any constant multiplier or lower-order terms. Many simple computations are O(n) , or ``order n''; these take a fixed amount of time per data item. More complex computations are O(n * log(n)) , ``order n-log-n''; sorting a set of n items, for example, takes at least n-log-n time. Computations that are O(n^2) , O(n^3) , and so forth, are called polynomial time algorithms. Some of the worst problems are those that are called NP complete; the best algorithms known for these take exponential time. Unfortunately, A.I. involves many such algorithms; many search procedures, for example, are NP complete. While it may take exponential time to find a solution (say, a proof for Fermat's Last Theorem), it may take only polynomial time to check a solution (verify that a given proof is correct). In brief, Lisp and Prolog are popular programming languages that help to develop AI-based applications. The main difference between Lisp and Prolog is that Lisp is a computer program language that supports functional, procedural, reflective

and meta paradigms while Prolog is a computer programming language that supports logic programming paradigm.

Although, Prolog and Lisp are two of the most popular AI programming languages, they have various differences. Lisp is a functional language while Prolog is a logic programming and declarative language. Lisp is very flexible due to its fast prototyping and macro features, so it actually allows extending the language to suit the problem at hand. In the areas of AI, graphics and user interfaces, Lisp has been used extensively because of this rapid prototyping ability. However, due to its inbuilt logic programming abilities, Prolog is ideal for AI problems with symbolic reasoning, database and language parsing applications. Choice of one over the other completely depends on the type of AI problem that need to be solved.