ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 10, October 2016

Prolog and its Evolution, Description and Evaluation

James Vivian Chinenye, Ebiesuwa Seun, Adegbenjo. A.A, Adeyeye. J.A, Kehinde. D.O,

Grace Mensah-Agyei

Department of Computer Science, Babcock University, Ilisan-Remo, Ogun State, Nigeria

Department of Computer Science, Babcock University, Ilisan-Remo, Ogun State, Nigeria

Department of Computer Science, Babcock University, Ilisan-Remo, Ogun State, Nigeria

Department of Nutrition and Dietetics, Babcock University, Ilisan-Remo, Ogun State, Nigeria

Department of Basic Sciences, Babcock University, Ilisan-Remo, Ogun State, Nigeria

Department of Microbiology, Babcock University, Ilisan-Remo, Ogun State, Nigeria

ABSTRACT: The name Prolog was chosen by Philippe Roussel as an abbreviation for programmation en logique (French for programming in logic). It was created around 1972 by Alain Colmerauer with Philippe Roussel, based on Robert Kowalski's procedural interpretation of Horn clauses. It was motivated in part by the desire to reconcile the use of logic as a declarative knowledge representation language with the procedural representation of knowledge that was popular in North America in the late 1960s and early 1970s. According to Robert Kowalski, the first Prolog system was developed in 1972 by Colmerauer and Phillipe Roussel. The first implementations of Prolog were interpreters. However, David H. D. Warren created the Warren Abstract Machine, an early and influential Prolog compiler which came to define the "Edinburgh Prolog" dialect which served as the basis for the syntax of most modern implementations.
European AI researchers favored Prolog while Americans favored Lisp, reportedly causing many nationalistic debates on the merits of the languages. Much of the modern development of Prolog came from the impetus of the Fifth Generation Computer Systems project (FGCS), which developed a variant of Prolog named Kernel Language for its first operating system.
This study discusses the rules, design, syntax and queries of PROLOG and includes an evaluation of the language following from the earlier discussions in the study.

KEYWORDS: Prolog; Logic; Syntax; Compiler; Kernel

I. INTRODUCTION

Prolog, which stands for PROgramming in LOGic, is the most widely available language in the LOGIC PROGRAMMING PARADIGM using the mathematical notions of relations and logical inference. Prolog is a declarative language rather than procedural, meaning that rather than describing how to compute a solution, a program consists of a data base of facts and logical relationships (rules) that describes the relationships which hold for the given application. Rather than running a program to obtain a solution, the user asks a question. When asked a question, the run time system searches through the data base of facts and rules to determine (by logical deduction) the answer. Often there will be more than one way to deduce the answer or there will be more than one solution, in such cases the run time system may be asked to backtrack and find other solutions. Prolog is a weakly typed language with dynamic type checking and static scope rules. Prolog is typically used in artificial intelligence applications such as natural language interfaces, automated reasoning systems and expert systems. Expert systems usually consist of a data base of facts and rules and an inference engine, the run time system of Prolog provides much of the services of an inference engine. Pure Prolog was originally restricted to the use of a resolution theorem prover with Horn clauses of the form:

A. HORN FORMULAS
The formulas we get when translating all have the same structure:
$A1 \wedge A2 \wedge \cdots \wedge An \rightarrow B$
Such a formula can be rewritten as follows:
$A1 \wedge A2 \wedge \cdots \wedge An \rightarrow B \equiv \neg(A1 \wedge A2 \wedge \cdots \wedge An) \vee B \equiv \neg A1 \vee \neg A2 \vee \cdots \vee \neg An \vee B$
Hence, formulas obtained from translating Prolog clauses can always be rewritten as disjunctions of literals with at most one positive literal. Such formulas are known as Horn formulas

B. DESCRIPTION
In Prolog, program logic is expressed in terms of relations, and a computation is initiated by running a query over these relations. Relations and queries are constructed using Prolog's single data type, the term. Relations are defined by clauses. Given a query, the Prolog engine attempts to find a resolution refutation of the negated query. If the negated query can be refuted, i.e., 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 extra logical features.

C. DESIGN
Propositional logic provides the foundation for programming in Prolog. A proposition is formed using the following rules:

- true and false are propositions
- variables p, q, r,… etc. that take on values true or false are propositions
- Boolean operators ∧∨ ¬ ⇒and = used to form more complex propositions

Predicate logic expressions include all propositions and also include variables in the domain. A predicate is a proposition in which some of the Boolean variables are replaced by:

- Boolean-valued functions
- Quantified expressions

Here are some examples of Boolean-valued functions:
prime(n) - true if n is prime
president-of-nigeria(x) - true if x is the president of the Nigeria
A predicate combines these kinds of functions using operators of the propositional calculus and the universal quantifier, ∀(which reads “for all”), and the existential quantifier, ∃(which reads “there exists”).
For example:
∀x (speaks(x,Russian)) - true if everyone on the planet speaks Russian, false otherwise
Note that this is not true for ONLY Russian speakers, this applies to everyone
∃x (speaks(x,Russian)) - true if at least one person on the planet speaks Russian
∀x∃y (speaks(x,y)) – true if for all people x, there exists a language y such that x speaks y; false otherwise
∀x (¬literate(x) ⇒(¬writes(x) ∧ ¬∃y(reads(x,y) ∧book(y)))) - true if every illiterate person x does not write and does not read a book y
A tautology is a proposition that is true for all possible values of their variables. A simple example is: q ∨ ¬q. If q is true the entire expression is true. If q is false the expression is still true. Predicates that are true for some assignment of values to their variables are called satisfiable. Those that are true for all possible assignments of values to their variables are called valid. A prolog program is essentially an implementation of predicate logic.

D. PROLOG SYNTAX
Prolog is based on facts, rules, queries, constants, and variables. Facts and rules make up the database while queries drive the search process. Facts, rules, and queries are made up of constants and variables. All prolog statements end in a period.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 10, October 2016

Facts

A fact is a proposition and begin with a lowercase alphabetic character and ends in a period. Here are some sample facts:

rainy.

This says that rainy is true.

superhero(mymother).

This says that mymother is a superhero. Note the lowercase. This distinction is important, because we'll use an initial uppercase letter to indicate a variable.

eats(mymother, pizza).

This says that mymother eats pizza.

Each fact that we enter describes the logical "world" that comprises the database of knowledge we can then reason over.

Rules

A rule is an implication like forward chaining in logic. In a rule, we can use boolean operators to connect different facts. The symbols used in prolog are as follows:

Predicate Calculus	Prolog	
∧	,	(comma)
∨	;	(semicolon)
←	:-	"if", note direction is left, not →
¬	not	

Here are some examples:

humid :- hot, wet. Same as: hot∧wet →humid

pleasant :- not(humid) ; cool. Same as: ¬humid∨cool →pleasant

likes(funmi, coconutchips) :- not(likes(vivian, coconutchips)). Same as: Funmi likes coconutchips if Vivian does not like coconutchips.

Soon we'll extend this using variables instead of just coconutchips.

Variables

Variables are denoted in prolog by identifiers that start with an uppercase letter. For example:

likes(debby, Food) :- not(likes(vivian, Food)).

This says that Debby likes any food that Vivian does not like. Note that this is quite different from:

likes(funmi, food) :- not(likes(vivian, food)).

The second statement is an atom named food, while the first is a variable that can represent any number of possible values.

Consider the following rules and facts:

```
likes(ola,Food) :-
        contains_cheese(Food),
        contains_meat(Food).
likes(ola,Food) :-
        greasy(Food).

likes(ola,chips).
contains_cheese(macaroni).
contains_cheese(lasagna).
contains_meat(lasagna).
greasy(french_fries).
```

In processing these rules, Prolog will unify the right hand side of the rule with any atoms that match the predicate. For the first rule, Food could be either macaroni or lasagna since both fit the criteria of contains_cheese. But then we AND this with contains_meat which leaves only lasagna. From these facts we can conclude that Ola likes chips, lasagna (cheese + meat), and french fries (greasy).

Queries
To query the database we can use prolog in an interpretive manner. Queries are generally made at the ?- prompt and consist of a predicate. For example, given the above data:

```
?- contains_meat(salad).
No
?- contains_meat(lasagna).
Yes
?- likes(ola,chips).
Yes
?- likes(ola, lasagna)
Yes
?- likes(ola, macaroni)
No
```

We can also make queries that include variables in them. Prolog will instantiate the variables with any valid values, searching its database in left to right depth-first order to find out if the query is a logical consequence of the specifications. Whenever Prolog finds a match, the user is prompted with the variables that satisfy the expression.

EXAMPLE
Here is a query that finds all foods that contain cheese:

```
?- contains_cheese(X).
X = macaroni ;
X = lasagna ;
No
```

Since I hit “;” each time to not accept the matches, Prolog exhausts the possible foods with cheese and returns no. If I type “y” instead Prolog will return yes:

```
?- contains_cheese(X).
X = macaroni
Yes
```

We could query the database to find all the foods that Ola likes to eat:

```
 ?- likes(ola, X).
X = lasagna ;
X = french_fries ;
X = chips ;
No
```

We could also query the database to find all the people that like to eat lasagna:

```
?- likes(X, lasagna).
X = ola ;
No.
```

Right now nobody in the database likes macaroni so we get the following:

```
?- likes(X, macaroni).
No
```

II. EVALUATION

The language was first conceived by a group around Alain Colmerauer in Marseille, France, in the early 1970s and the first Prolog system was developed in 1972 by Colmerauer with Philippe Roussel.
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 declarative: 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 was one of the first logic programming languages, and remains the most popular among such languages today, with several free and commercial implementations available.

The language has been used for theorem proving, expert systems, as well as its original intended field of use, natural language processing.
Modern Prolog environments support creating graphical user interfaces, as well as administrative and networked applications.
Prolog is well-suited for specific tasks that benefit from rule-based logical queries such as searching databases, voice control systems, and filling templates.

III. CONCLUSION

In 1987 two German developers Jan Wielemaken and Anjo Anjewierden released the SWIPROLOG, the name SWI is derived from "Sociaal-Wetenscappelijke Informatica" which means "Social Science Informatics". The latest prolog version is the SWIPROLOG 7.2.3.0.
The PROLOG language has been used over the years for theorem proving, expert systems, as well as its original intended field of use, natural language processing and has proven to be a more robust language than many other symbolic languages.

BIOGRAPHY

1. Bratko (2001). Prolog Programming For Artificial Intelligence, 3rd Ed. Addison-Wesley, HarlowCh2 Prolog Presentation, Part II Programming In Prolog, 2
2. Horn, A (1951). "On Sentences Which Are True Of Direct Unions Of Algebras", Journal Od Symbolic Logic, 16, 14-21
3. Lucas, P., University Of Abeerdeen, Pg 3
4. Priss, U (2010). Edinburgh Napier University, "Mathematics For Software Engineering"
5. Xindong W (2015). University Of Vermont, "Logic Programming and Prolog".