ISSN No:-2456-2165

Logic Programming From the Point of View of Type Theory and Predicate Linear Logic

Liberios Vokorokos
Department ofComputers and Informatics
TechnicalUniversityof Košice
Košice, Slovakia

Zuzana Bilanová
Department ofComputers and Informatics
TechnicalUniversityof Košice
Košice, Slovakia

Zuzana Dankovičová
Department ofComputers and Informatics
TechnicalUniversityof Košice
Košice, Slovakia

Abstract:- In this article we will deal with the analysis of several logic programming languages and their interpreters, including their theoretical origins. Logic programming is based on the first order predicate logic, as is the case with Prolog, with which we explain the basic principles of such an approach. Subsequently, we will describe non-traditional logical systems in logic programming - programming with higher-order logic represented by the λProlog programming language and source-oriented logic programming represented by the Vorvan programming language.

Keywords:- *linear logic; logical programming; first order predicate logic; theory of types.*

I. INTRODUCTION

The initial ideas that created the area of logic programming were introduced in 1960, when the logical programming language Planner was created. The real boom of this area came after 1965, with J. A. Robinson publications, which were the basis for the first implementations of today's most widely used logical programming language - Prolog. For the paradigm of logic programming [1], the following facts are characteristic:

- the main idea is to use a program to draw the consequences, introducing a declarative way of describing the problem solved,
- as a programming language, the first order predicate logicis mostly used,
- the basic principle is the procedural interpretation of the first order predicate logic, where the essence is to interpret implications as procedural declarations,
- when creating a logic program, a scheme of system axiom (or set of formulas) is defined. It specifies the class of tasks that the programmer solves. Also, target orders are formulated for a specific task which need to be solved,
- in logic programming, the calculation is by proving the question (target command) as the logical consequence of the set of axioms that make up the body of the program.

Logical commands and rules are logged in the logical programming languages in the form of the Horn Clauses[2]. The clause is a disjunction of literals. Literals are atomic formulas, they may be positive, e.g. p resp. negative, e.g.$\neg p$. The Horn clause may contain at most one positive literal.

Example of the Horn clause:

$$\neg p \vee \neg q \vee \ldots \vee \neg t \vee u \quad (1)$$

The conjunction of Horn's clauses is called the Horn's Formula. Horn formulas are usually enrolled as implications:

$$p \wedge q \wedge \ldots \wedge t \rightarrow u \quad (2)$$

The given hypothesis is demonstrated in the logical paradigm by the resolution method. Logic programming languages are most often used as theorem-provers, model-generators andproblem-solvers. They are used for symbolic and non-numeric computations, programming of expert systems, artificial intelligence, etc.

II. PROLOG

Prolog (from fr.PROgramation aLOGic - Logic Programming) [3]is the universal and most widely used logic programming language designed to program symbolic computations. Its basis is first order predicate logic. Prolog's programs are formulated through Horn's clauses.

A. First order predicate logic

First order predicate logic [4]is richer than propositional logic, which uses only a small part of the formal language - symbols with truth value. Predicate logic extends propositional logicby quantifiers and predicates (properties, relationships of individuals).

First order predicate logic can be described by following grammar in BNF:

$$\alpha, \beta ::= P(t, \ldots, t) \mid \neg\alpha \mid \alpha \wedge \beta \mid \alpha \vee \beta \mid \alpha \rightarrow \beta \mid (\forall x)\alpha \mid (\exists x)\alpha \quad (3)$$

$$t ::= x \mid c \mid f(t, \ldots, t) \quad (4)$$

The meaning of the symbols of first order predicate logic is as follows:

- formulasα, β,
- predicate$P(t, \ldots, t)$represents the application of the predicate symbol to terms$t$,
- logical connectives ¬ - negation, ∨ - disjunction, ∧ - conjunction, → - implication,
- quantifiers∀ - universal, ∃ - existential,
- x is a variable, c is a constant,
- $f(t, \ldots, t)$is an application of a functional symbol on the terms.

ISSN No:-2456-2165

B. Basic Principles

Basic features of Prolog:

- declarative programming language - program logic is declared using terms and relations, representing facts and rules that describe properties and relationships between objects,
- non-procedural programming language - the problem solving procedure is uninteresting for the programmer, the procedural part of the calculation is purely in the direction of Prolog,
- conversational programming language - Prolog answers user questions.

The Prolog programming language can also be said to be interpretive, interactive, based on recursion (through a single node recursively searches a linear calculus tree), even on variable substitution, its data and program are stored in one file in the same clauses.

The Prolog program is the ultimate non-empty set of Horn clauses. The clauses consist of the head and the body, and if Prolog deduces that the body is true, the head is true.

There are 3 kinds of clauses:

- *Facts*

✓ have onlya head, simple clauses, constant assertions,

✓ e.g. parent(claudius,nero).

- *Rules*

✓ have a head and a body, complex clauses, with implications allow to deduce new facts, contain variables, express the reality that one fact can depend on other groups of facts

✓ e.g. siblings(X,Y) :- parent(X,R),

parent(Y,R), X \= Y.

- *Commands*

✓ have onlya body, they are done immediately after consulting the knowledge base, starting with the ":-" symbol.

The system answers users' questions based on the information contained in the database of clauses (also known as the knowledge database) and its internal rules. Depending on whether the target clause is fulfilled in the Prolog program, Prolog answers YES or NO. It can also list the values of the arguments for which the target formula is fulfilled.

C. Use of Prolog

A simple example of the use of Prolog language is a mathematical game called the Tower of Hanoi (Fig 1).

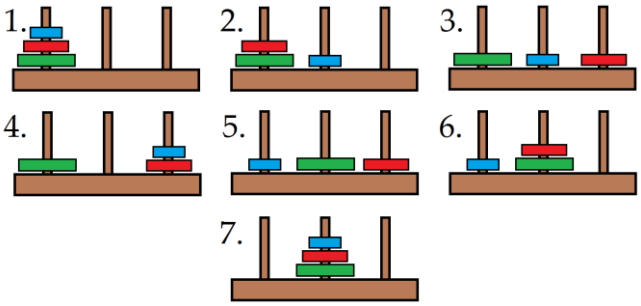

Fig 1:- Principle of Tower of Hanoi

In The Tower of Hanoi, a player has 3 rods to choose from,on the first one there are disks sorted from the biggest to thesmallest one which are supposed to be moved to the middlerod. In one turn player can move only one upper disk. Larger disk always has to be on top of the smaller one. In the Prolog programming language, this task can be solved by the following program:

```
move(1,X,Y,_) :-
write('Move top disk from '),
write(X),
write(' to '),
write(Y),
nl.

move(N,X,Y,Z):-
   N>1,
   M is N-1,
move(M,X,Z,Y),
move(1,X,Y,_),
move(M,Z,Y,X).

hanoi (P) :- move(P,left,middle,right).
```

This source code has been written to the a.pl file, which represents the Prolog knowledge database. The program itself was run in BinProlog interpreter. Fig. 2 shows the output of the Hanoi tower algorithm in BinProlog for the question of hanoi (3).

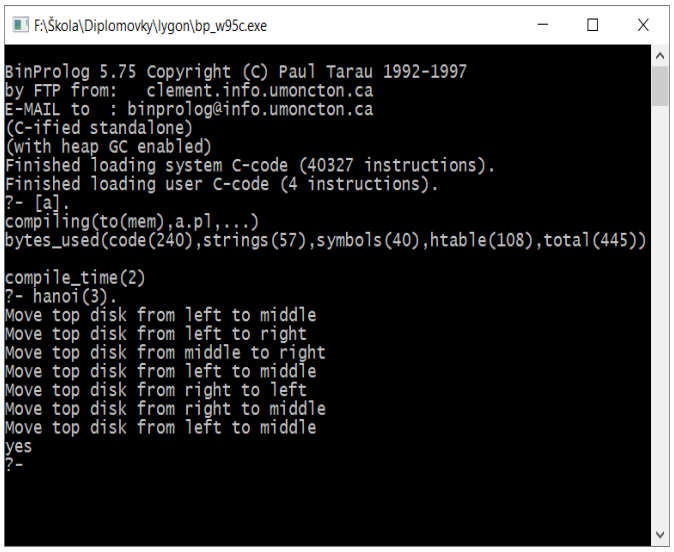

Fig 2:- Tower of Hanoi in BinProlog interpreter

III. ΛPROLOG

In the 1980s, Dale Miller and GopalanNadathur[5]created a new, wider theoretical basis for logic programming languages.They createdλProlog, which is programming language based on Prolog. λProlog extended Prolog by high-order functions, λ-terms, intuitionistic fragment of Church's simple theory of types, higher-order unification, polymorphic types, and provides mechanisms for defining modules and secure abstract data types. It also extends the classical first-order theory of Horn clauses to the intuitionistic higher-order theory of hereditary Harrop formulas.

λProlog is an interesting, higher-order logic programming language that provides effective ways of realizing computations over formal objects such as mathematical expressions, logical formulas, or proofs. Thanks to perspective of this language there was created an abstract machine named Teyjus.

D. Intuitionistic logic

In terms of intuitionism:

- constructivist approach – construction and constructive proof are key concepts for intuitionism, derived from the intuitionist understanding of logical symbols,
- the statement is true if it is possible to construct its proof,
- the rejection of the principle of proof by contradiction and the excluded middle principle,
- intuitionisticsemantics can be given in terms of Kripke semantics.

Formulas of intuitional logic have the same syntax as in predicate logic. The only differences are:

- implicationγ→ δcannot be expressed through disjunction and negation¬ γ∨ δ;
- any logic connective can be expressed by the other, all four: ∧, ∨, →, ¬are necessary.

E. λ-calculus

Until the end of the 19th century, mathematics was based on naive theory of sets. Russell's paradox, formulated at the beginning of the 20th century, proves that set theory is not consistent but contains contradictions. This was the reason for creating the λ -calculus, which in 1928 formulated the Alozo Church. The previous principle "everything is a set" has been replaced by a principle that we can paraphrase as "everythingis a function".

Simply typed λ-calculus[6] is the simplest typed functional programming language.Itssyntax can be described by following grammar in BNF:

$$t ::= x|\lambda x{:}T.t|ft \quad (5)$$

$$T ::= a|\, T \to T \quad (6)$$

The meaning of the symbols of λ-calculus is as follows:

- metavariable$t$indicates λ-term,
- variable x binds the occurrences of x in t,
- term tt is an application, which applies the function f to a term t,
- $\lambda x{:}T.t$ je λ-abstraction that defines a function with a parameter$x$type$T$and body of function$t$,
- set of terms T,
- base type a - a set of values, e.g., integers, strings,
- function type T → T where arrow → is a type constructor.

F. Higher-order hereditary Harrop formulas

The Harropformulas were introduced in 1956 and named after Ronald Harrop. The concept of these formulas is used in various fields of mathematics and logic programming.The language of higher-order hereditary Harrop formulas is determined by:

- *G*-formulas - function as goals or queries in a logic programming setting,
- *D*-formulas - function as program clauses or definition clauses in this context.

Higher-order hereditary Harrop formulas can be described by following grammar in BNF:

$$G ::= A|G \wedge G|G \vee G|(\forall x)G|\ (\exists x)G|D \supset G \quad (7)$$

$$D ::= A_r|G \supset A_r|D \wedge D|(\forall x)D \quad (8)$$

The meaning of the symbols of higher-order hereditary Harrop formulas is as follows:

- Adenote atomic formulas, $A_r$denote rigid atomic formulas,
- *D*-formulas are called program clauses and *G*-formulas are called goal formulas.

G. Use of λProlog

To use the λProlog language in practice, we will use an abstract machine named Teyjus. Teyjus uses four subsystems: tjcc compiler, loader tjlink, abstract machine emulator tjsim and user interface. The system also allows the use of a compiler on the one hand and an emulator on the other hand as separate components.

Programming in λProlog is structured trough the modules. In Teyjus, there are four files associated with a module. If the name of the module is <name> then these files are essential:

- <name>.mod - the source code,
- <name>.sig - the signature which identifies names of sorts, type constructors, constants and operators.,
- <name>.lpo - the bytecode file generated by the compiler for the module,
- <name>.lp - linked and executable version of the bytecode file produced by the linker.

In the λProlog programming language, the Tower of Hanoi can be solved by the following program, which consists of signature and module part:

```
sighanoi.

type hanint -> string -> string ->string -> o.

module hanoi.
```

han 1 A B _ :- print "Move disk from " , print A , print " to " , print B , print "\n".

han I A B C :- J is I - 1 , han J A C B , han 1 A B C , han J C B A.

Fig. 3 shows the output of the Hanoi tower algorithm in Teyjus for the question of han 3 “left” “middle” “right”.

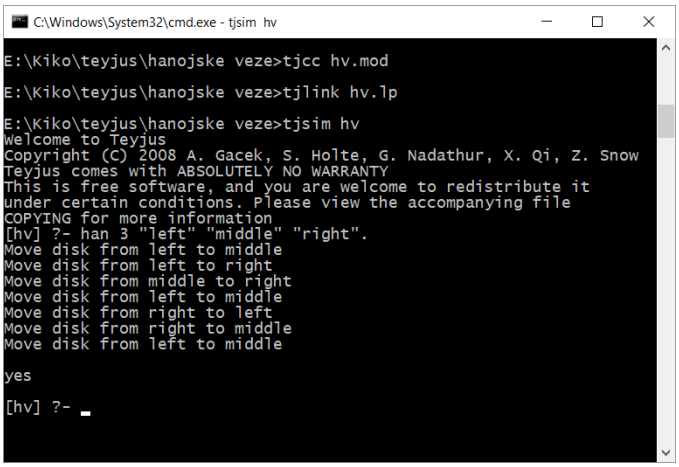

Fig 3:- TowerofHanoi in Teyjusabstract machine

IV. VORVAN

Vorvan programming language was developed on the Department of Computers and Informatics, Technical University of Košice, Slovakia by KostiantynBilan in 2016.Vorvan hasa modern graphic interpreter of resource-oriented nature based on predicate linear logic.

H. Predicate linear logic

Predicate linear logic [7] was introduced in 1987 by J.Y. Girard. This logic can be seen as anextension of first order predicate logic and intuitionistic logic. Predicate linear logic can be applied to processesin the real world, placing emphasis on resource consumption.

Predicate linear logic has the following characteristics:

- works with resources,
- the formula of this logic is an action or source,
- describes real processes in the world, use in functionalprogramming,
- introduces new linear logic connectives,
- does not contain the rule of weakness and contraction.

Predicate linear logiccan be described by following grammar in BNF[8]:

$$\varphi ::= a_n \mid 1 \mid 0 \mid \mathrm{T} \mid \bot \mid P(t,\dots,t) \mid \varphi \otimes \psi \mid \varphi \& \psi \mid \varphi \oplus \psi \mid \varphi \wp \psi \mid \varphi \multimap \psi \mid \varphi^{\bot} \mid !\varphi \mid ?\varphi \mid (\forall x)\varphi \mid (\exists x)\varphi \quad (9)$$

$$t ::= x \mid c \mid f(t,\dots,t) \quad (10)$$

The meaning of the symbols of predicate linear logicis as follows:

- φ, ψ are resource-orientedformulas,
- a_n are elementary formulas,
- 1, 0 arelogical constants,
- $\otimes, \&, \oplus, \wp$ are logical connections,
- $\multimap$ is linear implication
- $(.)^{\bot}$ is unary logicnegation,
- !, ? are modal operators,
- as in first order predicate logic, x is a variable, c is a constant and $f(t,\dots,t)$ is an application of a functional symbol on the terms.

I. Use of Vorvan

To demonstrate the use of the Vorvan programming language [9] in practice, we will utilize interpreter Vorvan, which is called the same as his programming language.Vorvan uses three subsystems as shown in Fig. 4:

- compiler (Vorvan Compiler - VC) able to locate syntactic and semantic errors,
- text editor (User Interface-UI),Vorvandevelopment environment,
- interpreter (VorvanCoMmanD - VCMD), an interactive tool inwhich the user can try out his source codes.

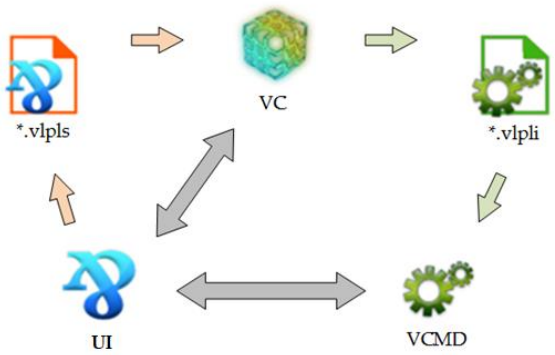

Fig 4:- The communication process of the interpreter Vorvan

Each of these components works separately. From the user interface can be directly compiled and interpreted the sourcecode stored in a *.vlpls files. The compiler creates from filesof type *.vlpls another files which type is *.vlpli. Next *.vlplifiles are processed in the interpreter VCMD.

In the λProlog programming language, the Tower of Hanoi can be solved by the following program:

```
han(1,A,B,C) <-
print("Move disk from " + !A + " to " + !B + "\n").

han(A,B,C,D) <-
```

(!A - 1 -> E) * ((han(A - 1, B, D, C) * han(1, B, C, D)) * han(E,D, C, B)).

Vorvan (unlike Teyjus and BinProlog) is a simple and modern development environment that is intuitive for the programmer and speeds up his work. Fig. 5 depicts its text editor.

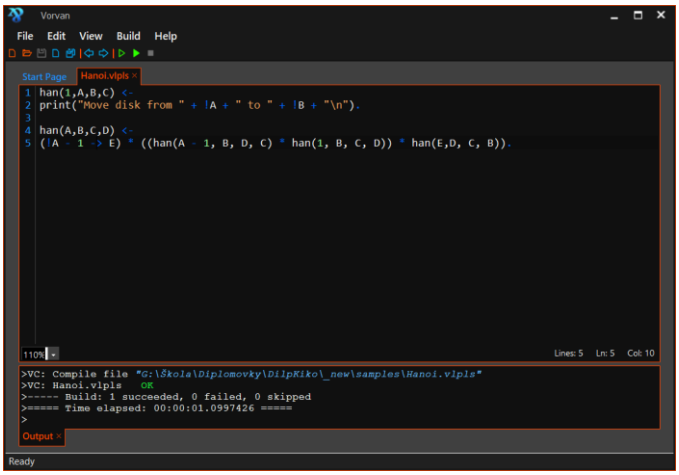

Fig 5:- User interface of Vorvan

Fig. 6 shows the output of the Hanoi tower algorithm in Teyjus for the question of han(3, left, middle, right).

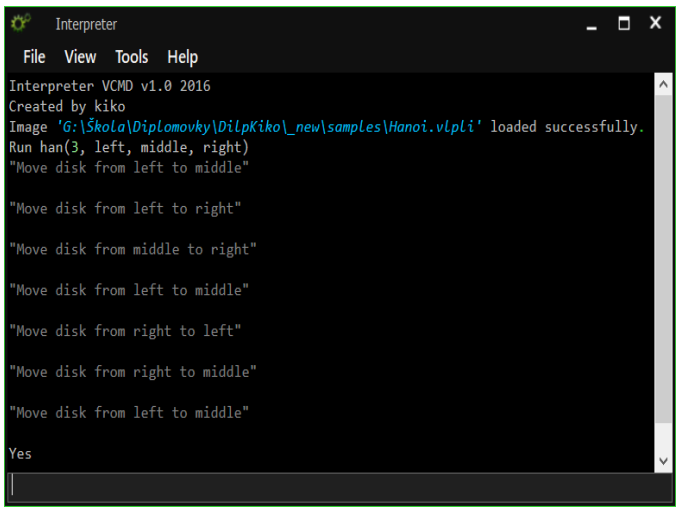

Fig 6:- TowerofHanoi in Vorvanabstract machine

The Vorvan programming language and its implementation are currently being developed under the guidance of the authors of this article.

V. CONCLUSION

On traditional example of the Tower of Hanoi we have demonstrated the differences between interpreters of the first order predicate logic,the intuitionistic logic and the predicate linear logic. Each of the described interpreters - BinProlog, Teyjus, Vorvan is suitable for solving other types of problems. At the same time, there is a common theoretical ground between BinProlog and Teyjus, as well as between BinPrologandVorvan. However, there is no link between higher-order logic programming language and resource-oriented logic programming.

Our research deals with the development of new logical programming languageVorvan, which is designed to solve problems with the ultimate number of resources. From the point of view of future research, we find it interesting to extend its functionality so it can work with λ-terms like λProlog. Linking the functionality of these interpreters would give us a tool with extraordinary expressive power able to solve a wide range of problems.

VI. ACKNOWLEDGMENT

This work was supported by Faculty of Electrical Engineering and Informatics, Technical University of Košice under contract No. FEI-2017-43 „Handwriting analysis focused on disgraphy“ and Faculty of Electrical Engineering and Informatics, Technical University of Košice under contract No. FEI-2017-47 „Design and development of verifiable BDI architecture for IDS using component and virtual reality systems“. This support is very gratefully acknowledged.

REFERENCES

[1] John W. Lloyd,“Foundations of logic programming,” Springer Science & Business Media, 2012, pp. 212.

[2] S. Ceri, G. Gottlob, L. Tanca, “Logic programming and databases,”Springer-Verlag Berlin Heidelberg, 1990, pp. 284.

[3] I. Bratko, “Prolog programming for artificial intelligence (4th edition) (International Computer Science Series), ” Pearson Education Canada, 2011, pp. 696.

[4] A. Pettorossi, M. Proietti, “First order predicate calculus and logic programming, ”Aracne, 2016, pp. 152.

[5] D. Miller, G. Nadathur, “Programming with higher-order logic,” Cambridge University Press, 2012.

[6] G. E. Revesz, “Lambda-calculus, combinators and functional programming,” Cambridge University Press, 2009.

[7] E. Demeterová, “Logical programming in informatics,” in: Electrical Engineeringand Informatics 4 : Proceedings of the Faculty of Electrical Engineering and Informatics of the Technical University of Košice, 2013, pp. 867-872.

[8] L. Vokorokos, Z. Bilanová, D. Mihályi, “Linear logic operators in transparent intensional logic, ” In: Informatics 2017. - Danvers : IEEE, 2017 S. 420-424.

[9] K. Bilan, D. Mihályi, “Logic programming paradigm language interpreter based on resource-oriented logical system principles,“ in: Electrical Engineering and Informatics 7 : Proceedings of the Faculty of Electrical Engineering and Informatics of the Technical University of Košice, 2016, pp. 291-296.