A Prolog-like Language for the Internet

Ulisses Ferreira

***Abstract*— This paper presents a three-valued logic programming language which permits definitions of clauses under closed-world assumption or without it, due to the presence of a constant (referred to as *uu*) at the language level. A third truth value is used to provide only one negation, defined here as abstract negation, while Extended Logic Programs adopt two kinds of negation. I present an operational semantics for both propositional and the predicate forms, including variables. The language can be seen as an adaptation of Prolog capable of capturing lack of information. In particular, the language can be viewed as as an appropriate compromise solution between logic and a global structure such as the Internet. Little work has been done combining logic with such a platform.**

Keywords: partial, deduction, declarative, AI, programming languages, distributed, system, Internet

I. Introduction

Since Prolog was designed, several programming languages, techniques and paradigms have been proposed and developed. Much research work has been done combining logic programming with other paradigms. A few examples include the languages λProlog[29] and LIFE[1], which combine logic with functional programming. However, in spite of the great importance of such work, no logic programming language provides representation of lack of information together with only one negation. The proposed language (Globallog) has two relevant attributes, namely, the $abstract\ negation$ and the ability to distinguish a refutable clause from the one which is not proven from a list of clauses. The need for such a distinction has become increasingly relevant for Internet programming, because connections sometimes fail and programs must be robust enough to continue running. Thus, in a distributed knowledge-based system over the Internet, a remote query that is not able to prove some predicate results in *unknown* instead of *false*.

Many important theoretical contributions[15], [16] have been made to logic programming by researchers, including well-founded semantics [17], [19] and stable models[31], also, the work of Przymusinsky and Gelfond[21], Ginsberg[22] among others. A more recent proposal is [5], which is a semantics for Prolog programs based on a 4-valued logic. Theories on negation have been proposed since the close-world assumption (CWA)[32]. [4] presents a survey on this subject. In $general$ logic programs (GLP), negation as failure (NAF)[8] is used to infer negative information. In fact, NAF is a concept which depends on the CWA[1]. A GLP clause has the form

$$L_0 \leftarrow \mathcal{L}_1, \ldots, \mathcal{L}_m. \qquad (1)$$

[1]That is why the proposed negation is called "abstract negation" in the sense that its use does not depend on whether the world is assumed to be closed or not.

where $\leftarrow$ is an implication symbol (for instance the :– symbol in Prolog), $\mathcal{L}_i$, for $1 \leq i \leq m$, is a literal possibly with the **not** symbol, i.e. the negation as failure operator, while $\mathcal{L}_0$ does not accept negation, although there has been some recent work on this constraint[24].

A newer approach called Extended Logic Programming (ELP) provides two forms of negation: NAF and explicit negation[7], [11], [26]. As well as ELP, Gelfond and Lifschitz[20] present a proposal which is based on the stable model semantics and answer sets. The authors define an extended logic program as being a sequence of clauses of the form

$$L_0 \leftarrow L_1,\ \ldots,\ L_m,\ not\ L_{m+1},\ \ldots,\ not\ L_n \qquad (2)$$

where $0 \leq m \leq n$ and each L_i is a literal, i.e. either a predicate or the explicit negation ($\neg$) of a predicate, and not is the negation as failure operator. Thus, in ELP, the CWA for a predicate p can be represented by $\neg p(X) \leftarrow not\ p(X).$, which, although flexible, makes use of two kinds of negation.

Contributions have been done by Pereira[2], [30], Kowalski and others[33], also in ELP. There are other new approaches, such as [36] based on Well-Founded Semantics. However, Globallog is outside the scope of ELP.

ELP was not conceived to allow programming under the choice of open- or closed-world assumptions. If the programmer uses NAF in a predicate p, it means that he or she is assuming a closed world. If he or she uses the explicit negation instead, it means that he or she is assuming an open world for *that* use of p. However, it is not clear how to use *positive* predicates choosing either assumptions, which is exactly what Globallog allows programmers to do. In other words, ELP is more closely related to the negations than the assumptions.

Like ELP, there exist three possible results: *false*, *true* and *unknown*. While what is stated as $false$ is simply regarded as $false$, what is not represented in the program (or what is unable to be proven) is simply regarded, by default, as $unknown$. The distinction between *false* and *unknown* allows the inference machine (also called $system$ here), not only the application program, to recognize the *need* for learning. For example, if a query from a predicate p results in $unknown$, the system could ask the user whether he or she would like to insert some definition in order to improve the knowledge base.

There are some situations, however, in which CWA with NAF are much more appropriate[3], [10]. For example, considering in Prolog the clauses for defining a member of a list, one (the programmer) can write the following rules:

$member(X, [X|\_]).$
$member(X, [\_|Z]) :- member(X, Z).$

Without the CWA, the programmer would have to explicitly