FUNCTIONAL EXTENSIONS TO PROLOG: ARE THEY NEEDED?1

Laurent Fribourg2
L.I.E.N.S., 45 rue d’Ulm - 75230 Paris cedex 05 - France

Introduction

Prolog [6] is a logic language which, besides logical variables, manipulates relation and data constructor symbols. In the last few years, the enrichment of logic languages with function symbols and functional paradigms of computation has been extensively investigated (for a survey, see e.g. [2]). The main paradigm of functional computation is reduction, which applies when an expression matches the left-hand side of a program statement, and consists in replacing the expression by the corresponding right-hand side. The advantages of Functional Programming (FP) over Logic Programming (LP), in particular Prolog, are discussed in [2], and can be summarized as:

1. **Notation.** The functional formalism is more readable than the relational one.
2. **Control.**
   1. **Backtracking.** Since reduction replaces an expression by an equivalent one, it is backtracking-free (assuming the search strategy to be depth-first). It is also able to handle negative knowledge, and precipitate backtracking when reducing an expression to a form such as "true=false".
   2. **Evaluation strategy.** Owing to the nesting of functional subterms, a functional program *a priori* contains more control information than the corresponding logic program. In particular, an optimal strategy (lazy evaluation) can be used in order to avoid unnecessary computation and to handle infinite structures.
3. **Higher-order features and Types.**

In this note we focus on point (2), and argue that control mechanisms similar to those of functional programs can be reproduced in the framework of Prolog programs without integrating functions.

1. Logic programming with functions: a bit of history

In order to enrich Logic Programming with functions, a natural approach is to consider the union of a set of Horn clauses $H$ with a set $E$ of equations as a program. In such a

---


2 This work has been partially supported by Laboratoires de Marcoussis and ESPRIT project 432.
framework, the essential property to be satisfied by the system of inference rules of the interpreter is completeness for first-order logic with equality.

The naïve solution consists in limiting the inference rules of the interpreter to resolution and adding the equality axioms to the program. Unfortunately, this leads to a combinatorial explosion of the size of the search space. To remedy this defect, Plotkin has proposed instead to replace the standard unification involved in resolution by semantic unification modulo the theory defined by the equational system $E$ [24]. Semantic unification of two terms modulo $E$ corresponds to finding instantiations (solutions) such that the instantiated terms be equal modulo $E$. In [19] this approach is provided with a semantic foundation. The major drawback of this approach is that the process of semantic unification may itself loop forever and is essentially undecidable [16]. Another approach consists of amalgamating the LP paradigm of resolution with the FP paradigm of reduction. An early experiment in this trend was FUNLOG [28]. In FUNLOG the equational system $E$ is made up of a confluent and terminating rewrite system, i.e. a functional program made of left-to-right oriented equations (rewrite system) such that reduction applied repeatedly always yields one irreducible form (termination property) and only one (confluence property). The FUNLOG interpreter applies resolution on predicates after having reduced their arguments as much as possible via the rewrite rules. Although FUNLOG combines several nice features of both programming styles (such as the logical variable of LP and the lazy evaluation of FP), the inference rules are not complete for first-order logic with equality.

In the field of automated theorem proving, a method, named narrowing, was especially conceived to ensure completeness for first-order logic with equality (in the case of a confluent and terminating system $E$) [27]. Narrowing is the natural extension of reduction to incorporate unification. It consists in applying the minimal substitution to an expression in order to make it reducible, and then to reduce it. The minimal substitution is found by unifying the expression with the left-hand side of a rewrite rule. A step of narrowing can also be followed by a sequence of reductions leading to an irreducible form (normalized narrowing). The idea of integrating LP with FP by adding (normalized) narrowing to resolution was first proposed in [14] (EQLOG). Narrowing can be used by itself without resolution (except for resolution with the reflexive axiom $X=X$) in order to semantically unify two terms modulo the confluent and terminating rewrite system $E$. Fay [9] and Hullot [17] have proved the completeness of narrowing to that purpose (in the sense that all the normalized solutions are enumerated). Hullot’s proof is based on the parallel that can be established between the sequence of reductions applied to an instance of a term and a sequence of narrowings applicable to that term. This result is the counterpart of the "lifting lemma" in the theory of resolution [26]. Such results have been extended to conditional rewrite systems [21][18][13], and to non-terminating systems [32][13]. It thus becomes possible to consider the set $H$ of Horn clauses as a special case of