Statement of Purpose

Mathematical proofs often come out the result of the combination of two complementary techniques: computation and deduction. Computation involves the symbolic manipulation of expressions, which are transformed according to some specified equivalences. Deduction is the process of deriving logical consequences from a set of initial postulates, through the application of some fixed rules of inference.

Unfortunately in the past many applications of logic to computer science failed to realize the importance of a balanced interaction of both these processes, and instead showed a strong bias toward the one or the other. A case in point is *Logic Programming* (LP), a discipline born to provide a logical underpinning to programming languages design. Perhaps the most successful byproduct of LP is the Prolog language and its derivatives, which base their operational model on depth-first proof search over a restricted class of first-order formulae (e.g. Horn clauses, Hereditary Harrop formulae). Applied to real-life problems, which often required algebraic data structures such as numbers or strings, the purely deduction-based approach used by Prolog soon exhibited its many limitations, and required, in early implementations, the introduction of extra-logical constructs. The use of these constructs make the analysis of the behavior and properties of Prolog programs much harder, nullifying most of the advantages promised by the LP approach.

A more satisfactory solution to this situation was offered at the beginning of this decade by a set of programming techniques, grouped under the name of *Constraint Logic Programming* (CLP), which finally recognized the importance of introducing algebraic computation as part of the operational semantic of Prolog. From a theoretical viewpoint, CLP lays its foundation on the studying of Term Rewriting and equational reasoning. In practice, CLP is implemented by employing very efficient decision procedures to solve frequently-occurring classes of equational problems; problems that fall outside the scope of these procedures are kept as dormant constraints until they can be furtherly simplified.

Parallel to CLP, the 90s have seen other exciting developments of logic programming, and, among these *Higher-Order Logic Programming* (HO-LP). The seminal work of Martin Löf demonstrated how Constructive Type Theory can be effectively used as foundation of mathematics. This inspired the development of HO-CLP languages such as Lambda-Prolog and Elf. These languages offer several advantages over their first-order counterparts. The use of a type system makes the task of isolating meaningless expressions easier, and therefore leads to earlier detection of many common programming mistakes. Lambda-abstraction provides a convenient way to represent the notion of bound variable, that is used pervasively in mathematics. Built-in β-reduction offers a computational component in an otherwise purely deductive programming paradigm. Finally,

the expressiveness offered by their sophisticated type structure allow one to formalize other deductive systems within these languages, and therefore be used as *Logical Frameworks*.

Unfortunately HO-LP languages, borrowing much of their operational semantic model from Prolog, inherit most of its limitations too. In these past few years, very exciting applications of these languages have surfaced (such as, for example, Necula's proof-carrying code technique for security in a mobile code environment) but they were hampered by similar problems to those mentioned before for Prolog. It is my thesis that the way to address these must go through the study of Higher-Order Term Rewriting, as the basis for the development of CLP-techniques for HO-LP languages.

The theoretical work done in the course of my Ph.D. studies can be divided in two parts.

In the first one, I analyzed an extension to LF where types can also be converted modulo an equational theory. As the LF calculus is very sensitive to changes in the type conversion relation used, all the confluence properties of β-reduction and (restricted) η-expansion, and existence of normal forms needed to be re-examined. I showed that my extension is conservative, and all the usual properties continue to hold, albeit in a weaker form.

In the second part, I introduced a notion of rewriting for this setting. Since in a dependently typed calculus terms are allowed to appear inside types, a naive definition, extending the one given by Nipkow for simply-typed calculi, is inapplicable, as it may lead to situations where rewriting invalidates the typed of an expression. Hence, I turned my attention to the study of special preorders, called dependence relations, that allow us to extract type information that is implicit in a LF signature, and use it to restrict rewriting to well-behaved instances.

Dependence relations turned out also to be useful when studying confluence of rewriting, which was shown to be, as usual, reducible to checking some special rewriting configurations known as *critical pairs*. Together with a general Critical Pair Criterion, I proved a specialized version, where fewer critical pairs need to be checked thanks to the type information conveyed by these preorders.

There are at least two directions toward which this research could evolve. The first is further application of CLP-techniques to Lambda-Prolog and Elf. It would be interesting to see to what novel techniques can be defined to make best use of the properties of these languages, or to what extent the existing one lift to this higher-order setting.

Another interesting direction of research is further study of Higher-Order Term Rewriting. Many important classes of Term Rewriting System are not covered by the results presented in my Ph.D. dissertation, including rewriting modulo associative and associative-commutative operators, and conditional rewriting.

The latter seems of particular importance, in light of Prolog extensions such as Frühwirth's Constraint Handling Rules, that allow Prolog programmers to define their own constraint solvers by means of writing conditional rewrite rules.