Higher Order Unification 30 Years Later
(Extended Abstract)

Gérard Huet

INRIA-Rocquencourt
BP 105, F-78153 Le Chesnay Cedex
Gerard.Huet@inria.fr

Abstract. The talk will present a survey of higher order unification, covering an outline of its historical development, a summary of its applications to three fields: automated theorem proving, and more generally engineering of proof assistants, programming environments and software engineering, and finally computational linguistics. It concludes by a presentation of open problems, and a few prospective remarks on promising future directions. This presentation assumes as background the survey by Gilles Dowek in the Handbook of automated theorem proving [28].

1 Theory

1.1 Early History

The problem of automating higher order logic was first posed by J. A. Robinson in [100,101]. Peter Andrews formulated a version of the resolution principle for accommodating Church’s Simple Theory of Types [16,4], and as early as 1964 Jim Guard’s team at Applied Logic Corporation had been independently investigating higher order logic versions of unification [42,41], but the complexity of the problem appeared when higher order unification was shown to be undecidable in 1972, independently by G. Huet [46] and C. Lucchesi [61], a result to be sharpened by W. Goldfarb [39]. Jensen and Pietrzykowky [89,50], at University of Waterloo, gave a complete recursive enumeration of unifiers, but the process was over-generative in an untractable fashion, since computational quagmires happened where at every step of the process all previous solutions were subsumed by the next one. Huet proposed a more tractable enumeration of pre-unifiers for better conditioned problems, where rigidity of one of the problem terms led to sequential behavior. By delaying ill-conditioned problems in the form of constraints, it became possible to implement a complete version of the resolution principle for Church’s Type Theory [49]. An extensive theory of unification in various algebraic settings, building on early work of G. Plotkin [90] was developed by G. Huet in his Thèse d’Etat [47], while P. Andrews and his collaborators at CMU went on implementing constrained resolution in the proof assistant TPS [5].

Thus the original motivation for higher-order unification was its use as a fundamental algorithm in automated theorem-provers. We shall see below that
the design and implementation of proof assistants is still its main application area, not just in the setting of classical logic with versions of the resolution principle, but in more general meta-theoretic frameworks, where it is the work horse of pattern-matching and more generally proof search. For instance, extensions of the term-rewriting paradigm of equational logic, building on higher-order unification, were proposed by Nipkow, Jouannaud and Okada, Breazu-Tannen, Blanqui, and many others \cite{76,10}, while C. Benzmueller investigated the addition of equality paramodulation to higher-order resolution \cite{8,9}.

1.2 \(\lambda\) Prolog

It was soon realized that resolution was better behaved in the restricted context of Horn sentences, and Colmerauer and Kowalski took advantage of this quality in the framework of first-order logic with the design of PROLOG, which spurred the whole new field of Logic Programming. The generalization of this promising technology to higher-order logic was successfully accomplished by D. Miller with his \(\lambda\)-PROLOG system \cite{65,73}. G. Nadathur further investigated the \(\lambda\)-PROLOG methodology and its efficient implementation \cite{71,72,70,73,74}. In the Mali team in Rennes, Yves Bekkers and colleagues implemented a \(\lambda\)-PROLOG machine with specific dynamic memory management \cite{7,14}. The most impressive application of the \(\lambda\)-PROLOG formalism was accomplished by F. Pfenning in his ELF system, a logic programming environment for Edinburgh’s Logical Framework \cite{85,88}.

The theory of higher-order unification in more powerful higher order frameworks, encompassing dependent types, polymorphism, or inductive types, was investigated by C. Elliott \cite{31,32}, G. Pym \cite{96}, G. Dowek \cite{20,26}, F. Pfenning \cite{86,87} etc. in a series of publications in the 80’s and 90’s. We refer the interested reader to the extensive presentation in G. Dowek’s survey \cite{28}.

1.3 Matching

An important special case of unification (i.e. solving equations in term structures) is when one of the terms is closed. Unification in this special case is pattern matching, a specially important algorithm in symbolic computation. Huet’s semi-decision algorithm turned into an algorithm generating a finite set of solutions in the case of second-order terms \cite{47,48,22}. Furthermore, it was possible to combine pattern-matching for linear terms with first-order unification, as Miller demonstrated with his higher-order patterns \cite{66}. However, the general problem in the full higher-order hierarchy remained open for a long time. G. Dowek showed that the third-order problem was decidable \cite{27}, while extensions in various other directions led to undecidability \cite{21,24}. Padovani showed that the fourth-order case was decidable \cite{80,81,82}. Wolfram gave an algorithm which always terminate \cite{115}, but whose completeness is dubious. More recently, Loader announced the undecidability of higher-order matching in \cite{59,60}, but the