Type Theory and Recursion
Extended Abstract

G. D. Plotkin
Department of Computer Science
Laboratory for Foundations of Computer Science
University of Edinburgh, The King's Buildings
Edinburgh EH9 3JZ, SCOTLAND

At first sight, type theory and recursion are compatible: there are many models of the typed lambda calculus with a recursion operator at all types. However the situation changes as soon as one considers sums. By a theorem of Huwig and Poigné, any cartesian closed category with binary sums and such a general recursion operator is trivial. Domain theory provides the category of cpos and continuous functions. It is cartesian closed and has a general recursion operator — the least fixed-point operator. It (necessarily) does not have binary sums, but the closely associated category of cpos and strict continuous functions does.

We propose an analysis in terms of intuitionistic linear logic, or, better, intuitionistic linear type theory. This is compatible with a general recursion operator for the intuitionistic functions. The category of cpos and strict continuous functions provides a very simple model of the combination of the type theory with such a general recursion operator. The continuous functions appear as the intuitionistic, not necessarily linear, ones. There are many other interesting models available arising from other notions of computation or variant categories of domains. As the cpo model allows contraction, relevant type theory is more appropriate for it than linear type theory. However other models do not permit contraction; an example is the category of complete semilattices and completely additive functions. This is also the category of algebras of the Hoare powerdomain which assigns to every cpo the collection of all non-empty Scott-closed subsets.

Another concern of semantics has been polymorphism. Various models of Girard's System F and related calculi have been developed. Of particular interest are those where polymorphism is treated parametrically (uniformly). Various notions of parametricity arise, such as the relational one of Reynolds or the dinatural one of Bainbridge, Freyd, Scedrov and Scott. With Reynolds' relational parametricity, one has derivable in System F finite products and sums, second-order existential quantification and initial and final algebras (of definable covariant functors). It is both interesting — and necessary for the development of the semantics of programming languages — to consider the interaction of polymorphism and recursion. However as the parametric models have categorical sums, a now familiar difficulty arises.

We consider instead a second-order intuitionistic linear type theory whose primitive type constructions are linear and intuitionistic function types and second-order quantification. In the presence of a suitably modified form of Reynold's parametricity the usual operators of propositional linear logic can be derived by formulae very similar to the Prawitz-Scott formulae familiar from the intuitionistic case; categorically, one obtains a model in Seely's sense. One also obtains initial and final algebras for definable covariant functors over the category of linear maps. As before, one can consistently add a general recursion operator. One can then show compactness, in Freyd's sense, and obtain canonical solutions to arbitrary recursive type equations of the form $X \cong T(X, X)$ (where now T is a bifunctor contravariant in its first argument and covariant in its second).

It is convenient to work in a suitable second-order logic rather than consider directly the various possible categorical structures. The logic is formed from the equational formulae of an appropriate type theory by adding sufficient logical apparatus to enable the formulation of a schema for relational parametricity. Within the logic various induction and co-induction principles can be defined for initial and final algebras; corresponding means of reasoning are available for the other derivable type constructs. The logic can be considered as a logic of programs in the same spirit as the Logic of Computable Functions, originated by Scott and further developed by Milner *et al.*