Arity Polymorphism and Dependent Types (abstract)

Eugenio Moggi

DISI, Univ. of Genova, Italy

In polytypic programming one would like to define combinators, such as *map*, which operate on datatype constructors of any *arity*. However, it is rather problematic to device a type system that can express this form of polymorphism. In a calculus like Functorial ML (of [JBM98]), instead of having one combinator *map*, one has a family of combinators map_n parametrized over natural numbers, while in a system like PolyP (of [JJ97]) there are restriction on the arity of datatype constructors. However, the most natural typing for the *map* combinator would be

$$\forall n\colon N.\forall F\colon Functor(n).\forall X,Y\colon Type^n.(\forall i\colon n.X_i \to Y_i) \to F(X) \to F(Y)$$

where n ranges over arities (say natural numbers), F over functors of arity n, X and Y over n-tuples of types, and i over *indexes* for arity n (e.g. the predecessors of n). Semantically, one wonders what is meaning of $\forall n\colon N$ and $\forall i\colon n$. In the setting of realizability models there is a standard interpretation for $\forall$, so the question becomes how should one interpret the *types/kinds* N and n?

Syntactically, the issue is: can we device a type system that allows such types, and what should terms look like? We propose a type system with a limited form of dependent types and inductive types, which provides a *reasonable* solution.