In recent years, proof assistants have reached an impressive level of maturity. They have led to the certification of complex programs such as compilers and operating systems. Yet, using a proof assistant requires highly specialised skills, and it remains very different from standard programming. To bridge this gap, we aim at designing an ML-style programming language with support for proofs of programs, combining in a single tool the flexibility of ML and the fine specification features of a proof assistant.

We thus define and study a call-by-value language whose type system extends higher-order logic with an equality type over untyped programs, a dependent function type, classical logic and subtyping. The combination of call-by-value evaluation, dependent functions and classical logic is known to raise consistency issues. To ensure the correctness of the system (logical consistency and runtime safety), we design a theoretical framework based on Krivine's classical realizability. The construction of the model relies on an essential property linking the different levels of interpretation of types in a novel way.

We finally demonstrate the expressive power of our system using our prototype implementation, by proving properties of standard programs like the map function on lists or insertion sort.

Semantics and Implementation of an Extension of ML for Proving Programs

Rodolphe Lepigre

Semantics and Implementation of
an Extension of ML for Proving Programs

Rodolphe Lepigre