Some steps toward program extraction in a type-theoretical interpretation of IFP

Ulrich Berger[1], Sewon Park[2], Holger Thies[2], and Hideki Tsuiki[2]

[1]Swansea University, United Kingdom
[2]Kyoto University, Japan

IFP (Intuitionistic Fixed Point Logic) [BPT20, BT21] extends intuitionistic first-order logic with strictly positive inductive and coinductive definitions. Sorts are classical and computational content can be attached by defining predicates over the sorts with intuitionistic disjunctions and inductive or coinductive definitions. For example, we can introduce an axiomatic sort for the real numbers and assume any classically valid disjunction-free sentence. Over the sort, predicates can be coinductively defined allowing to reason over various infinite representations of real numbers and computations over them.

IFP's domain-theoretic realizability interpretation allows partial realizers. Its program extraction, which yields possibly nonterminating programs, hence is well-suited for extracting inherently partial representations such as infinite Gray code [Tsu02].

However, the term language of IFP does not allow the construction of functions. To define a new function, the function needs to be axiomatized by extending the term language with the function symbol and the set of axioms with valid sentences describing the function.

As an alternative approach, we present an extension of IFP with lambda calculus by embedding it in a dependent type theory. Hence, not only new function terms but also new sorts, such as (classical) function spaces, can be defined following type-theoretical constructions. We further extend the underlying (proof-erased) untyped lambda calculus's reduction rules [Let02] to reflect IFP's realizability interpretation.

To this end, we implement a shallow embedding of IFP in the Coq proof assistant. We use Coq's `Prop` as a universe of classical types where we introduce axiomatic sorts such as $\mathsf{R} : \texttt{Prop}$ for the set of real numbers. As in the original IFP, classically valid sentences can be introduced in `Prop` including the law of trichotomy. R-valued functions can be defined as a lambda term using the term constructions of the type theory; e.g., as R is in `Prop`, the law of trichotomy can be used to create even some discontinuous R-valued functions.

IFP's inductive and coinductive definitions are provided by axiomatic constants μ and ν of appropriate types. Corresponding proof rules for induction, closure, coinduction, and coclosure are axiomatized as constants Ind, Cl, CoInd, and CoCl.

Using Coq's type checking engine for IFP formal proofs, we rely on MetaCoq, a Coq's meta-programming plugin [SAB+20], for extracting programs from IFP proofs and simulating the extracted programs. More precisely, using MetaCoq's Erasure plugin, from a Coq term t we can obtain a term of untyped lambda calculus $\lambda_{\Box}$, where the noncomputational `Prop` parts and type expressions in t's construction get erased by $\Box$ [SBF+19]. In our case, using this "quoting" and erasing yields $\lambda_{\mathsf{IFP}} := \lambda_{\Box,\mathsf{Ind},\mathsf{Cl},\mathsf{CoInd},\mathsf{CoCl}}$, an untyped lambda calculus with additional constructs $\Box$, Ind, Cl, CoInd, and CoCl.

We define reduction for λ_{IFP} which extends naturally the reduction of untyped lambda calculus with the intended computational meanings of the four additional IFP constructs.

As λ_{IFP} itself is an inductive data in MetaCoq, we implement the reduction as a function in MetaCoq such that we can use Coq's reduction engine to compute the reduction of λ_{IFP}. Thus, our Coq implementation allows the users to make an extended IFP proof, extract the computational content of it, and simulate the computation, all in Coq.

We demonstrate the expressiveness of our axiomatization by implementing some of the standard examples of IFP such as the translation of the signed-digit representation to infinite Gray code [BT21, Section 5].

The implementation can be found in `https://github.com/holgerthies/ifp-coq`.

References

[BPT20] Ulrich Berger, Olga Petrovska, and Hideki Tsuiki. Prawf: An interactive proof system for program extraction. In *Conference on Computability in Europe*, pages 137–148. Springer, 2020.

[BT21] Ulrich Berger and Hideki Tsuiki. Intuitionistic fixed point logic. *Annals of Pure and Applied Logic*, 172(3):102903, 2021.

[Let02] Pierre Letouzey. A new extraction for Coq. In *International Workshop on Types for Proofs and Programs*, pages 200–219. Springer, 2002.

[SAB+20] Matthieu Sozeau, Abhishek Anand, Simon Boulier, Cyril Cohen, Yannick Forster, Fabian Kunze, Gregory Malecha, Nicolas Tabareau, and Théo Winterhalter. The metacoq project. *Journal of automated reasoning*, 64(5):947–999, 2020.

[SBF+19] Matthieu Sozeau, Simon Boulier, Yannick Forster, Nicolas Tabareau, and Théo Winterhalter. Coq coq correct! verification of type checking and erasure for coq, in coq. *Proceedings of the ACM on Programming Languages*, 4(POPL):1–28, 2019.

[Tsu02] Hideki Tsuiki. Real number computation through Gray code embedding. *Theoretical Computer Science*, 284(2):467–485, 2002.