IFP style proofs in the Coq proof assistant

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

[1]KAIST, Republic of Korea
[2]Kyoto University, Japan

IFP (Intuitionistic Fixed Point Logic) is an extension of intuitionistic first-order logic by strictly positive inductive and coinductive definitions. Program extraction is based on a uniform realizability interpretation which ignores realizers of non-computational statements (Harrop formulas) (see [BT21] for details). An interactive proof system based on IFP [BPT20] has recently been implemented.

The well-known Coq proof assistant, on the other hand, is based on constructive dependent type theory. Coq also provides an extraction mechanism that automatically translates proofs to functional programs [Let02] but different from IFP, a user has to explicitly distinguish between computationally relevant and computationally irrelevant statements by assigning different sorts to the respective terms.

With IFP, it is possible to extract programs from abstract proofs based on axioms, and it is particularly well suited for the extraction of certified programs for exact real number computation (see e.g. [Ber09]). With Coq, it is also possible to write an abstract proof based on axioms, but they are treated as computationally irrelevant statements and are not subject to program extraction. However, Coq is already equipped with a rich theory of mathematics and a well-developed software environment, which facilitates simple and elegant proofs. It is therefore desirable to establish a connection between the formal systems IFP and Coq, study how IFP-style proofs can be performed in Coq, and investigate the possibility of extraction from such proofs.

As a first step towards this goal we provide an automatic translation from an IFP instance to Coq types. An instance of IFP is a language $\mathcal{L}$ consisting of sorts, terms and predicate constants and a set of axioms $\mathcal{A}$. For each language $\mathcal{L}$ and axiom set $\mathcal{A}$ of $\mathcal{L}$ of IFP, we define the set of Coq axioms as IFP-Coq$(\mathcal{L}, \mathcal{A})$ by the steps (i-v).

(i) For each sort ι in $\mathcal{L}$, define ι as a term constant (axiom) of Prop.

(ii) For each constant c of sort ι, define c as a term constant (axiom) of type ι.

(iii) For each function symbol f of arity $\iota_1 \times \cdots \times \iota_n \to \iota$, define f as a term constant (axiom) of type $\iota_1 \to \cdots \to \iota_n \to \iota$.

(iv) For each predicate symbol P of arity $(\iota_1, \cdots \iota_n)$, define P as a term constant (axiom) of type $\iota_1 \to \cdots \to \iota_n \to \texttt{Prop}$.

(v) For each operator symbol Q of arity $(\iota_1, \cdots \iota_n)$, define Q as a term constant (axiom) of type $(\iota_1 \to \cdots \to \iota_n \to \texttt{Prop}) \to (\iota_1 \to \cdots \to \iota_n \to \texttt{Prop})$.

To add the axioms from an IFP instance we further need to translate from IFP expressions to Coq types. We describe a recursive procedure for this translation. The procedure is mostly straight forward with the exception of inductive and coinductive definitions. In IFP, the expressions $\mu(\Phi)$ and $\nu(\Phi)$ can be used to inductively and coinductively define predicates for any strictly positive operator Φ. IFP contains the following proof rules for induction and coinduction:

$$\frac{}{\Phi(\mu(\Phi)) \subseteq \mu(\Phi)}\ \mathrm{CL}(\Phi) \qquad \frac{\Phi(P) \subseteq P}{\mu(\Phi) \subseteq P}\ \mathrm{IND}(\Phi, P)$$

$$\frac{}{\nu(\Phi) \subseteq \Phi(\nu(\Phi))}\ \mathrm{COCL}(\Phi) \qquad \frac{P \subseteq \Phi(P)}{P \subseteq \nu(\Phi)}\ \mathrm{COIND}(\Phi, P)$$

We translate each inductive definition $\mu(\Phi)$ to a new inductive type in Coq and generate the corresponding induction principle. This principle then has to be proven in Coq. In the case where Φ does not contain any further inductive or coinductive defintition, the proof is straight-forward and we have written a Coq tactic to automatically prove such statements. We apply the same technique to coinductive definitions.

The resulting environment can be used to prove IFP statements inside the Coq proof assistant. We could successfully convert all the IFP-proofs from [BT21] to formal proofs in Coq, except for those in RIFP.

Currently, our implementation only maps to non-computational types and thus does not provide any mechanism to generate programs from Coq proofs. However, in a later step we plan to add such functionality by implementing a realizability interpretation similar to that of IFP for IFP-Coq proof terms.

References

[Ber09] Ulrich Berger. From coinductive proofs to exact real arithmetic. In *International Workshop on Computer Science Logic*, pages 132–146. Springer, 2009.

[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.