Toward an interpretation of Intutionistic Fixed Point Logic in Coq

Holger Thies
Graduate School of Human and Environmental Science, Kyoto University
Joint work with Sewon Park and Hideki Tsuiki

November 22

The 17th Theorem Proving and Provers meeting
Kitami Institute of Technology

This work was supported by JSPS KAKENHI Grant Number 20K19744.

Overview

- IFP (Intuitionistic Fixed Point Logic) is a proof system that has been developed by U. Berger and collaborators (M. Seisenberger, O. Petrovska, H. Tsuiki) since 2009.

Overview

- IFP (Intuitionistic Fixed Point Logic) is a proof system that has been developed by U. Berger and collaborators (M. Seisenberger, O. Petrovska, H. Tsuiki) since 2009.
- IFP is an extension of first-order logic by by strictly positive inductive and coinductive definitions.

Overview

- IFP (Intuitionistic Fixed Point Logic) is a proof system that has been developed by U. Berger and collaborators (M. Seisenberger, O. Petrovska, H. Tsuiki) since 2009.
- IFP is an extension of first-order logic by by strictly positive inductive and coinductive definitions.
- IFP is in particular used for program extraction of exact real computation programs.

Overview

- IFP (Intuitionistic Fixed Point Logic) is a proof system that has been developed by U. Berger and collaborators (M. Seisenberger, O. Petrovska, H. Tsuiki) since 2009.
- IFP is an extension of first-order logic by by strictly positive inductive and coinductive definitions.
- IFP is in particular used for program extraction of exact real computation programs.
- The goal of this work is to do IFP style proofs in the Coq proof assistant and possibly to translate between the two proof assistants.

Intuitionistic Fixed Point Logic

IFP is a schema for a proof system over different abstract mathematical spaces.
An instance of IFP consists of a language $\mathcal{L}$ and a set of axioms $\mathcal{A}$.

Intuitionistic Fixed Point Logic

IFP is a schema for a proof system over different abstract mathematical spaces.
An instance of IFP consists of a language $\mathcal{L}$ and a set of axioms $\mathcal{A}$.
The Language $\mathcal{L}$ consists of

1. Sorts $\iota, \iota_1, \ldots$ as names for abstract mathematical spaces

Intuitionistic Fixed Point Logic

IFP is a schema for a proof system over different abstract mathematical spaces.
An instance of IFP consists of a language $\mathcal{L}$ and a set of axioms $\mathcal{A}$.
The Language $\mathcal{L}$ consists of

1. Sorts $\iota, \iota_1, \ldots$ as names for abstract mathematical spaces
2. Terms s, t consisting of variables, constants and function symbols

Intuitionistic Fixed Point Logic

IFP is a schema for a proof system over different abstract mathematical spaces.
An instance of IFP consists of a language $\mathcal{L}$ and a set of axioms $\mathcal{A}$.
The Language $\mathcal{L}$ consists of

1. Sorts $\iota, \iota_1, \ldots$ as names for abstract mathematical spaces
2. Terms s, t consisting of variables, constants and function symbols
3. Predicate constants of fixed arity

Intuitionistic Fixed Point Logic

IFP is a schema for a proof system over different abstract mathematical spaces.
An instance of IFP consists of a language $\mathcal{L}$ and a set of axioms $\mathcal{A}$.
The Language $\mathcal{L}$ consists of

1. Sorts $\iota, \iota_1, \dots$ as names for abstract mathematical spaces
2. Terms s, t consisting of variables, constants and function symbols
3. Predicate constants of fixed arity

Intuitionistic Fixed Point Logic

IFP is a schema for a proof system over different abstract mathematical spaces.
An instance of IFP consists of a language $\mathcal{L}$ and a set of axioms $\mathcal{A}$.
The Language $\mathcal{L}$ consists of

1. Sorts $\iota, \iota_1, \ldots$ as names for abstract mathematical spaces
2. Terms s, t consisting of variables, constants and function symbols
3. Predicate constants of fixed arity

Relative to $\mathcal{L}$ we define

- Formulas: $s = t$, $P(t)$, $A \wedge B$, $A \vee B$, $A \to B$, $\forall x\, A$, $\exists x\, A$.

Intuitionistic Fixed Point Logic

IFP is a schema for a proof system over different abstract mathematical spaces.
An instance of IFP consists of a language $\mathcal{L}$ and a set of axioms $\mathcal{A}$.
The Language $\mathcal{L}$ consists of

1. Sorts $\iota, \iota_1, \ldots$ as names for abstract mathematical spaces
2. Terms s, t consisting of variables, constants and function symbols
3. Predicate constants of fixed arity

Relative to $\mathcal{L}$ we define

- Formulas: $s = t$, $P(t)$, $A \wedge B$, $A \vee B$, $A \to B$, $\forall x\, A$, $\exists x\, A$.
- Predicates: Variables, constants, $\lambda x\, A$, $\mu(\Phi)$, $\nu(\Phi)$.

Intuitionistic Fixed Point Logic

IFP is a schema for a proof system over different abstract mathematical spaces.
An instance of IFP consists of a language $\mathcal{L}$ and a set of axioms $\mathcal{A}$.
The Language $\mathcal{L}$ consists of

1. Sorts $\iota, \iota_1, \ldots$ as names for abstract mathematical spaces
2. Terms s, t consisting of variables, constants and function symbols
3. Predicate constants of fixed arity

Relative to $\mathcal{L}$ we define

- Formulas: $s = t$, $P(t)$, $A \wedge B$, $A \vee B$, $A \to B$, $\forall x\, A$, $\exists x\, A$.
- Predicates: Variables, constants, $\lambda x\, A$, $\mu(\Phi)$, $\nu(\Phi)$.
- Operators: $\lambda X\, P$ (P strictly positive in X)

Axioms

A formula is called <u>non-computational</u> if it is disjunction-free and contains no free predicate variables.

Axioms

A formula is called non-computational if it is disjunction-free and contains no free predicate variables.
Axioms in IFP are closed non-computational (disjunction-free) formulas.
Example:

$$\forall x \, \forall y \, \neg(x < y) \to y \leq x$$

is OK but

$$\forall x \, \forall y \; x < y \lor y \leq x$$

is not.

Realizability and Program Extraction

- IFP's primary goal is program extraction

Realizability and Program Extraction

- IFP's primary goal is program extraction
- Careful distinction between computational and non-computational (Harrop) formulas

Realizability and Program Extraction

- IFP's primary goal is program extraction
- Careful distinction between computational and non-computational (Harrop) formulas
- Harrop expressions are realized by **Nil**

Realizability and Program Extraction

- IFP's primary goal is program extraction
- Careful distinction between computational and non-computational (Harrop) formulas
- Harrop expressions are realized by **Nil**
- Uniform interpretation of quantifiers:

$$a \mathrel{R} \Diamond x\, A = \Diamond x\; (a \mathrel{R} A) \text{ for } \Diamond \in \{\exists, \forall\}$$

Realizability and Program Extraction

- IFP's primary goal is program extraction
- Careful distinction between computational and non-computational (Harrop) formulas
- Harrop expressions are realized by **Nil**
- Uniform interpretation of quantifiers:

$$a \ R \ \Diamond x \, A = \Diamond x \ (a \ R \ A) \text{ for } \Diamond \in \{\exists, \forall\}$$

- Implemented in the Prawf proof assistant (U. Berger, O. Petrovska, H. Tsuiki)

Induction rules

IFP contains the following rules for strictly positive induction and coinduction:

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

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

$A \subseteq B$ is short for $\forall x\, A\, x \to B\, x$.

Example: The language of real numbers

Language:

- Sorts: one sort *R*
- Constants: $-1, 0, 1$
- Functions: $+, -, *, /, \ldots$
- Predicate constants: $<, \leq$

Axioms:

- Disjunction-free formulation of axioms of real closed field etc.

Example: Natural numbers

We can define natural numbers inductively by

$$N(x) = \mu(\lambda X \lambda x\,(x = 0 \vee X(x-1)))$$

Induction and Closure rules:

$$\frac{}{\forall x\,((x = 0 \vee N(x-1)) \to N(x))}\ \mathrm{CL}(N)$$

$$\frac{\forall x\,(x = 0 \vee P(x-1)) \to P(x)}{\forall x\, N(x) \to P(x)}\ \mathrm{IND}(N, P)$$

IFP and Coq

IFP

- Intuitionistic first-order logic
- Can add (nc) axioms
- Program Extraction
- Well-suited for proofs over abstract mathematical spaces (real numbers,...)
- Partiality

Coq

- Constructive Type Theory
- Can add axioms
- Program Extraction, Computation
- General purpose, many libraries

IFP-Coq

For each language $\mathcal{L}$ and set of axioms $\mathcal{A}$, we define a set of Coq axioms by

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

IFP-Coq

For each language $\mathcal{L}$ and set of axioms $\mathcal{A}$, we define a set of Coq axioms by

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

IFP-Coq

For each language $\mathcal{L}$ and set of axioms $\mathcal{A}$, we define a set of Coq axioms by

1. For each sort ι in $\mathcal{L}$, define ι as a term constant (axiom) of Prop.
2. For each constant c of sort ι, define c as a term constant (axiom) of type ι.
3. 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 $\iota_1 \to \cdots \to \iota_n \to \iota$.

IFP-Coq

For each language $\mathcal{L}$ and set of axioms $\mathcal{A}$, we define a set of Coq axioms by

1. For each sort ι in $\mathcal{L}$, define ι as a term constant (axiom) of Prop.
2. For each constant c of sort ι, define c as a term constant (axiom) of type ι.
3. 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 $\iota_1 \to \cdots \to \iota_n \to \iota$.
4. For each predicate symbol P of arity $(\iota_1, \cdots \iota_n)$, define P as a term constant (axiom) of type $\iota_1 \to \cdots \to \iota_n \to \texttt{Set}$.

IFP-Coq

For each language $\mathcal{L}$ and set of axioms $\mathcal{A}$, we define a set of Coq axioms by

1. For each sort ι in $\mathcal{L}$, define ι as a term constant (axiom) of Prop.
2. For each constant c of sort ι, define c as a term constant (axiom) of type ι.
3. 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 $\iota_1 \to \cdots \to \iota_n \to \iota$.
4. For each predicate symbol P of arity $(\iota_1, \cdots \iota_n)$, define P as a term constant (axiom) of type $\iota_1 \to \cdots \to \iota_n \to \texttt{Set}$.
5. For each operator symbol Q of arity $(\iota_1, \cdots \iota_n)$, define Q as a term constant (axiom) of type $(\iota_1 \to \cdots \to \iota_n \to \texttt{Set}) \to (\iota_1 \to \cdots \to \iota_n \to \texttt{Set})$.

Translating Formulas i

1. $H(c : \iota) = \ \vdash c : \iota$,
2. $H(f : \iota_1 \times \cdots \times \iota_n \to \iota) = \ \vdash f : \iota_1 \to \cdots \to \iota_n \to \iota$,
3. $H(x : \iota) = x : \iota \vdash x : \iota$,
4. $H(C : \text{ predicate}(\iota_1, \ldots, \iota_d)) = \ \vdash C : \iota_1 \to \cdots \to \iota_n \to \texttt{Set}$,
5. $H(X : \text{ predicate}(\iota_1, \ldots, \iota_d)) = X : \iota_1 \to \cdots \to \iota_d \to \texttt{Prop} \vdash X : \iota_1 \to \cdots \to \iota_d \to \texttt{Set}$,
6. $H(f(t_1, \cdots, t_n) : \iota) = \Gamma \vdash f\ t_1' \ \cdots \ t_n' : \iota$ when $H(t_i : \iota_i) = \Gamma_i \vdash t_i' : \iota_i$ and $\Gamma = \bigcup_i \Gamma_i$,
7. $H(t_1 = t_2) = \Gamma \vdash t_1' = t_2' : \texttt{Prop}$ when $H(t_1) = \Gamma_1 \vdash t_1' : \iota$, $H(t_2) = \Gamma_2 \vdash t_2' : \iota$, and $\Gamma = \Gamma_1 \cup \Gamma_2$,
8. $H(P \vee Q) = \Gamma \vdash P' \vee Q'$ when $H(P) = \Gamma_1 \vdash P' : \texttt{Prop}$, $H(Q) = \Gamma_2 \vdash Q' : \texttt{Prop}$, and $\Gamma = \Gamma_1 \cup \Gamma_2$,

Translating Formulas ii

9. $H(P \wedge Q) = \Gamma \vdash P' \wedge Q'$ when $H(P) = \Gamma_1 \vdash P' : \texttt{Prop}$, $H(Q) = \Gamma_2 \vdash Q' : \texttt{Prop}$, and $\Gamma = \Gamma_1 \cup \Gamma_2$,
10. $H(P \to Q) = \Gamma \vdash P' \to Q'$ when $H(P) = \Gamma_1 \vdash P' : \texttt{Prop}$, $H(Q) = \Gamma_2 \vdash Q' : \texttt{Prop}$, and $\Gamma = \Gamma_1 \cup \Gamma_2$,
11. $H(\exists x\ P) = \Gamma \setminus (x : \iota) \vdash \exists (x : \iota).\ P'$ when $H(x) = x : \iota$ and $H(P) = \Gamma \vdash P' : \texttt{Prop}$,
12. $H(\forall x\ P) = \Gamma \setminus (x : \iota) \vdash \forall (x : \iota).\ P'$ when $H(x) = x : \iota$ and $H(P) = \Gamma \vdash P' : \texttt{Prop}$,
13. $H(\lambda x\ P) = \Gamma \setminus (x : \iota) \vdash \lambda (x : \iota).\ P'$ when $H(x) = x : \iota$ and $H(P) = \Gamma \vdash P' : \texttt{Prop}$,

Translating Formulas iii

14. $H(\lambda X\ P) = \Gamma \setminus (X : (\iota_1 \to \cdots \to \iota_n \to \texttt{Prop})) \vdash \lambda(X : (\iota_1 \cdots \to \iota_n \to \texttt{Prop})).\ P'$ when
    $H(X) = X : \iota_1 \to \cdots \to \iota_n \to \texttt{Prop}$ and
    $H(P) = \Gamma \vdash P' : \texttt{Prop}$,

Inductive Types

For each expression $\mu(\Phi)$ in IFP we define an inductive type in Coq:

```
Inductive MPhi : (ι -> Prop) :=
    MPhic: forall x,  (Phi Mphi x) -> Mphi x.
```

where Phi is the translation of Φ.

Inductive Types

For each expression $\mu(\Phi)$ in IFP we define an inductive type in Coq:

```
Inductive MPhi : (ι -> Prop) :=
    MPhic: forall x,  (Phi Mphi x) -> Mphi x.
```

where Phi is the translation of Φ.

The right induction principle will in general not be generated by Coq but

```
Lemma ind_Phi: forall (P : ι -> Prop),
    forall x, ((Phi P x) -> P x) -> Mphi x -> P x.
```

can be proven.

Example

We formalized the real number examples from

Ulrich Berger, Hideki Tsuiki: **Intuitionistic Fixed Point Logic.**
Annals of Pure and Applied Logic 172.3 (2021): 102903.

Example

We formalized the real number examples from

Ulrich Berger, Hideki Tsuiki: **Intuitionistic Fixed Point Logic.** Annals of Pure and Applied Logic 172.3 (2021): 102903.

Recall the definition of natural numbers in IFP:

$$N(x) = \mu(\lambda X \lambda x\, (x = 0 \vee X(x-1)))$$

$\Rightarrow$ Demo in Coq.

Signed Digit Representation

The signed-digit representation for a real number $x \in [-1, 1]$ can be defined by $\nu(\Phi_{\mathrm{SD}})$ with

$$\Phi_{\mathrm{SD}} := \lambda X \lambda x \, \exists d \in \mathrm{SD} \; |2x - d| \leq 1 \wedge X(2x - d)$$

where

$$SD = \{-1, 0, 1\}$$

Mixed Induction/Coinduction

Nested inductive/coinductive definitions are used in IFP e.g. to define uniformly continuous functions. However, Coq does not accept the coinductive proof because it is not formally guarded.

Interpreting IFP in Type Theory

We embed IFP in Coq using only a simple subset of Coq's dependent type theory without general induction and coinduction. We only use the type constructions

- $0, 1$,
- N,
- $A + B$,
- $A \times B$,
- Π, Σ,
- $=$,
- The existence of a type `Set`, which is a universe of small types.

Formalizing IFP Syntax

IFP language $\mathcal{L}$ consists of

- a finite set of sorts $S = \{\iota_1, \cdots, \iota_n\}$,
- a finite set of (arity attached) constants $C = \{c_1 : \iota'_1, \cdots, c_n : \iota'_n\}$ where $\iota'_i \in S$,
- a finite set of (arity attached) operators $O = \{o_1 : \iota'_{11} \times \cdots \times \iota_{1n_1} \to \iota'_1, \cdots, o_n : \iota'_{n1} \times \cdots \times \iota_{nn_n} \to \iota'_d\}$ where $\iota'_i, \iota'_{jk} \in S$, and
- a finite set of (airty attached) relations $R = \{r_1 : (\iota'_{11}, \cdots, \iota_{1n_1}), \cdots, o_n : (\iota'_{n1}, \cdots, \iota_{nn_n})\}$.

Formalizing IFP Syntax

The term language consists of (i) v variables, (ii) constants, and (iii) operations.

$$\begin{array}{rcll} t & ::= & x & \text{variable} \\ & \mid & c & c \in C \;\; \text{constants} \\ & \mid & o(t_1, \cdots, t_n) & o \in O \;\; \text{operations} \end{array}$$

Formalizing IFP Syntax

The logical language is defined with the simultaneously defined formulae, predicates, and operators.

Formalizing IFP Syntax

Formulae:

$$
\begin{array}{rcll}
A, B & ::= & t_1 = t_2 & \text{equality} \\
& \mid & A \vee B & \text{disjunction} \\
& \mid & A \wedge B & \text{conjunction} \\
& \mid & A \to B & \text{implication} \\
& \mid & \exists x : \iota.\ A & \text{existence} \\
& \mid & \forall x : \iota.\ A & \text{universal} \\
& \mid & P(t_1, \cdots, t_n) & \text{application}
\end{array}
$$

Formalizing IFP Syntax

Predicates:

$$
\begin{array}{rcll}
P, Q & ::= & X & \text{predicate variable} \\
& \mid & r & r \in R \;\; \text{relational constants} \\
& \mid & \lambda(x_1 : \iota_1, \cdots, x_d : \iota_d).\, A & \text{abstraction} \\
& \mid & \mu(\Phi) & \text{least fixed points} \\
& \mid & \nu(\Phi) & \text{greatest fixed points}
\end{array}
$$

Operators:

$$
\Phi \;::=\; \lambda(X : (\iota_1, \cdots, \iota_n)).\, P \qquad \text{abstraction}
$$

Set Theoretical Semantics of IFP

We start with the basic semantics of the language $\mathcal{L}$:

- For each $\iota \in S$, its semantics is a set $[\![\iota]\!]_{\mathsf{Set}} \in \mathsf{Set}$.
- For each $(c : \iota) \in C$, its semantics is a point $[\![c]\!]_{\mathsf{Set}} \in [\![\iota]\!]_{\mathsf{Set}}$.
- For each $(o : \iota_1 \times \cdots \times \iota_n \to \iota) \in O$, its semantics is a morphism $[\![o]\!]_{\mathsf{Set}} : [\![\iota_1]\!]_{\mathsf{Set}} \times \cdots \times [\![\iota_n]\!]_{\mathsf{Set}} \to [\![\iota]\!]_{\mathsf{Set}}$.
- For each $(r : (\iota_1, \cdots, \iota_n)) \in R$, its semantics is a relation $[\![r]\!]_{\mathsf{Set}} : [\![\iota_1]\!]_{\mathsf{Set}} \times \cdots \times [\![\iota_n]\!]_{\mathsf{Set}} \to \{t, f\}$.

Set Theoretic Semantics of IFP

- For a well-formed formula $\Gamma; \Delta \vdash A$, we define its semantics to be a family of sets $[\![\Gamma; \Delta \vdash A]\!]_{\mathsf{Set}} : [\![\Gamma]\!]_{\mathsf{Set}} \times [\![\Delta]\!]_{\mathsf{Set}} \to \mathsf{Set}$.
- For a well-formed predicate $\Gamma; \Delta \vdash A : (\iota_1, \cdots, \iota_n)$, we define its semantics to be a function $[\![\Gamma; \Delta \vdash P : (\iota_1, \cdots, \iota_n)]\!]_{\mathsf{Set}} : [\![\Gamma]\!]_{\mathsf{Set}} \times [\![\Delta]\!]_{\mathsf{Set}} \to [\![\iota_1]\!]_{\mathsf{Set}} \times \cdots \times [\![\iota_n]\!]_{\mathsf{Set}} \to \{t, f\}$.
- For a well-formed operator $\Gamma; \Delta \vdash_{op} \Phi : (\iota_1, \cdots, \iota_n)$, we define its semantics to be a function $[\![\Gamma; \Delta \vdash P : (\iota_1, \cdots, \iota_n)]\!]_{\mathsf{Set}} : [\![\Gamma]\!]_{\mathsf{Set}} \times [\![\Delta]\!]_{\mathsf{Set}} \to [\![\iota_1]\!]_{\mathsf{Set}} \times \cdots \times [\![\iota_n]\!]_{\mathsf{Set}} \to \{t, f\}$.

Set theoretical semantics of IFP

For any sequence of families $c : \mathsf{N} \to A \to \mathtt{Set}$, we define the operators

$\sqcup(c) :\equiv \lambda(x : A).\ \Sigma(n : \mathsf{N}).\ c\ n\ x \quad \sqcap(c) :\equiv \lambda(x : A).\ \Pi(n : \mathsf{N}).\ c\ n\ x$

$\sqcup$ and $\sqcap$ are countable join and meet, respectively.

For any function $f : (A \to \mathtt{Set}) \to (A \to \mathtt{Set})$, we define:

$$\mu f := \sqcup f^n \bot$$

$$\nu f := \sqcap f^n \top$$

Future work

- Realizability Interpretation of IFP
- Complete Formalization
- (Partial) translation from Coq proofs to IFP proofs
- Extension to CFP