On the constructive content of proofs in abstract analysis

Ulrich Berger
Swansea University

j.w.w.

Hideki Tsuiki
Kyoto University

Proof and translation: Glivenko's theorem 90 years after

CLMPST, Prague, August 9, 2019

From Glivenko to Kleene

From a pragmatic point of view the interest in intuitionistic logic lies not so much in its higher 'trustability' but rather in the fact that its proofs contain computational information which proofs of classical logic lack.

From Glivenko to Kleene

From a pragmatic point of view the interest in intuitionistic logic lies not so much in its higher 'trustability' but rather in the fact that its proofs contain computational information which proofs of classical logic lack.

The translations by Glivenko, Gödel, Kolmogorov, . . . show that classical logic covers only the computationally uninformative fragment of intuitionistic logic

From Glivenko to Kleene

From a pragmatic point of view the interest in intuitionistic logic lies not so much in its higher 'trustability' but rather in the fact that its proofs contain computational information which proofs of classical logic lack.

The translations by Glivenko, Gödel, Kolmogorov, ... show that classical logic covers only the computationally uninformative fragment of intuitionistic logic

Kleene's realizability interpretation makes explicit the computational content of the parts that classical logic misses.

Computational content in the 'real' mathematical world

When trying to apply the ideas of Glivenko, Gödel and Kleene to 'real' mathematics one faces two interesting challenges:

Computational content in the 'real' mathematical world

When trying to apply the ideas of Glivenko, Gödel and Kleene to 'real' mathematics one faces two interesting challenges:

- Besides logic, mathematics uses strong axioms which, on the surface, are not constructive. Examples are choice principles and (suitable versions) of Zorn's Lemma.

  The translations of these principles require new forms of computation, for example, bar recursion and related recursions.

  This challenge will be briefly discussed in the next talk.

Computational content in the 'real' mathematical world

When trying to apply the ideas of Glivenko, Gödel and Kleene to 'real' mathematics one faces two interesting challenges:

- Besides logic, mathematics uses strong axioms which, on the surface, are not constructive. Examples are choice principles and (suitable versions) of Zorn's Lemma.

  The translations of these principles require new forms of computation, for example, bar recursion and related recursions.

  This challenge will be briefly discussed in the next talk.

- Another obstacle is that classical mathematicians like to work with abstract structures without committing to concrete representations.

  This talk will show that computational content may still be extracted, using the theory of real numbers as a running example.

The fundamental idea of program extraction

The fundamental idea of program extraction

A *proof* is a construction, represented by a text or a finite tree, that convinces us that a formula is *true*.

The fundamental idea of program extraction

A *proof* is a construction, represented by a text or a finite tree, that convinces us that a formula is *true*.

But often, a formula can also be understood as a *computational problem*.

The fundamental idea of program extraction

A *proof* is a construction, represented by a text or a finite tree, that convinces us that a formula is *true*.

But often, a formula can also be understood as a *computational problem*.

For example, the formula stating that there are infinitely many prime numbers,

$$\forall x\, \exists y\, (y > x \land \mathbf{Prime}(y))$$

can be understood as the problem of computing for every natural number x a prime number y that is greater than x.

The fundamental idea of program extraction

A *proof* is a construction, represented by a text or a finite tree, that convinces us that a formula is *true*.

But often, a formula can also be understood as a *computational problem*.

For example, the formula stating that there are infinitely many prime numbers,

$$\forall x\, \exists y\, (y > x \land \mathbf{Prime}(y))$$

can be understood as the problem of computing for every natural number x a prime number y that is greater than x.

Program extraction is based on the observation that a proof not only represents an argument why a formula is true but also contains a *program* that solves the computational problem it expresses.

Goals

Goals

Extract useful and fully verified programs.

Goals

Extract useful and fully verified programs.

Discover the logical and mathematical principles corresponding to programming paradigms:

logic	functional programming
induction	recursion
?	concurrency
?	memory management
?	lazyness

...

Minlog

`http://www.mathematik.uni-muenchen.de/~logik/minlog/`

Minlog is an interactive proof system that supports program extraction form proofs.

Most of the applications of program extraction presented in this talk have been carried out in Minlog.

Minlog is under active development at the Universities of Munich (lead), Kyoto and Swansea.

Overview

- Program extraction via realizability
- Intuitionistic fixed point logic (IFP)
- Realizability interpretation of IFP
- Brouwer's Thesis and Wellfounded Induction
- Archimedian Induction
- Application: From signed digits to inifinit Gray code
- Further applications

Realizability

Realizability attaches meaning to the Curry-Howard correspondence (in a similar way as Tarskian semantics attaches meaning to predicate logic).

Realizability

Realizability attaches meaning to the Curry-Howard correspondence (in a similar way as Tarskian semantics attaches meaning to predicate logic).

Intuitively:

If $M : A$ (that is, M codes an intuitionistic ND proof of A), then M solves the problem A according to the BHK-interpretation.

Realizability

Realizability attaches meaning to the Curry-Howard correspondence (in a similar way as Tarskian semantics attaches meaning to predicate logic).

Intuitively:

If $M : A$ (that is, M codes an intuitionistic ND proof of A), then M solves the problem A according to the BHK-interpretation.

This intuition is made precise in Kleene's realizability interpretation of **HA** by numbers ('number realizability', 1945).

Kleene's number realizability

For every closed formula A and every natural number e one defines what it means for e to *realize* A, $e\,\mathbf{r}\,A$.

$$
\begin{array}{lcl}
e\,\mathbf{r}\,A & \equiv & A \quad (A \text{ atomic}) \\
e\,\mathbf{r}\,(A \wedge B) & \equiv & e = \mathbf{P}(a,b) \wedge a\,\mathbf{r}\,A \wedge b\,\mathbf{r}\,B \\
e\,\mathbf{r}\,(A \to B) & \equiv & \forall a\,(a\,\mathbf{r}\,A \to \{e\}(a)\,\mathbf{r}\,B) \\
e\,\mathbf{r}\,(A \vee B) & \equiv & (e = \mathbf{P}(0,a) \wedge a\,\mathbf{r}\,A) \vee (e = \mathbf{P}(1,b) \wedge b\,\mathbf{r}\,B) \\
e\,\mathbf{r}\,(\forall x\,A(x)) & \equiv & \forall n\,(\{e\}(n)\,\mathbf{r}\,A(n)) \\
e\,\mathbf{r}\,(\exists x\,A(x)) & \equiv & e = \mathbf{P}(n,a) \wedge a\,\mathbf{r}\,A(n)
\end{array}
$$

where

$\mathbf{P} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ is some computable bijection, and

$\{e\}(a)\,\mathbf{r}\,B$ means that the partial recursive function (or Turing machine) with code e when applied to a terminates with some number $b \in \mathbb{N}$ such that $b\,\mathbf{r}\,B$.

Soundness Theorem

If **HA** $\vdash A$, then **HA** $\vdash e\,\mathbf{r}\,A$ for some e.

Soundness Theorem

If $\mathbf{HA} \vdash A$, then $\mathbf{HA} \vdash e \,\mathbf{r}\, A$ for some e.

Remarks:

1. The proof of the Soundness Theorem proceeds by induction on the given derivation of $\mathbf{HA} \vdash A$.
2. For the logical rules the extracted realizer e is essentially a code of the lambda-term provided by the Curry-Howard correspondence.
3. For the induction axiom the extracted realizer codes a primitive recursion (iterator).

Program extraction for **HA**

Assume $\mathbf{HA} \vdash \forall x \, \exists y \, A(x, y)$ where $A(x, y)$ is atomic.

Program extraction for **HA**

Assume **HA** $\vdash \forall x \, \exists y \, A(x, y)$ where $A(x, y)$ is atomic.

Then **HA** $\vdash e \, \mathbf{r} \, (\forall x \, \exists y \, A(x, y))$, for some e, by Soundness.

Program extraction for **HA**

Assume **HA** $\vdash \forall x\, \exists y\, A(x, y)$ where $A(x, y)$ is atomic.

Then **HA** $\vdash e\, \mathbf{r}\, (\forall x\, \exists y\, A(x, y))$, for some e, by Soundness.

This means **HA** $\vdash \forall n\, A(n, \mathbf{proj_1}(\{e\}(n)))$, that is, the function $f(n) \stackrel{\text{Def}}{=} \mathbf{proj_1}(\{e\}(n))$ solves the computational problem expressed by the formula $\forall x\, \exists y\, A(x, y)$.

Program extraction for **HA**

Assume $\mathbf{HA} \vdash \forall x\, \exists y\, A(x,y)$ where $A(x,y)$ is atomic.

Then $\mathbf{HA} \vdash e\, \mathbf{r}\, (\forall x\, \exists y\, A(x,y))$, for some e, by Soundness.

This means $\mathbf{HA} \vdash \forall n\, A(n, \mathbf{proj_1}(\{e\}(n)))$, that is, the function $f(n) \stackrel{\mathrm{Def}}{=} \mathbf{proj_1}(\{e\}(n))$ solves the computational problem expressed by the formula $\forall x\, \exists y\, A(x,y)$.

We generalize and improve program extraction by

- permitting abstract structures (instead of only natural numbers),

Program extraction for **HA**

Assume $\mathbf{HA} \vdash \forall x\, \exists y\, A(x,y)$ where $A(x,y)$ is atomic.

Then $\mathbf{HA} \vdash e\,\mathbf{r}\,(\forall x\, \exists y\, A(x,y))$, for some e, by Soundness.

This means $\mathbf{HA} \vdash \forall n\, A(n, \mathbf{proj_1}(\{e\}(n)))$, that is, the function $f(n) \stackrel{\mathrm{Def}}{=} \mathbf{proj_1}(\{e\}(n))$ solves the computational problem expressed by the formula $\forall x\, \exists y\, A(x,y)$.

We generalize and improve program extraction by

- permitting abstract structures (instead of only natural numbers),
- adding stronger axioms (instead of only induction on natural numbers),

Program extraction for **HA**

Assume $\mathbf{HA} \vdash \forall x \, \exists y \, A(x, y)$ where $A(x, y)$ is atomic.

Then $\mathbf{HA} \vdash e \, \mathbf{r} \, (\forall x \, \exists y \, A(x, y))$, for some e, by Soundness.

This means $\mathbf{HA} \vdash \forall n \, A(n, \mathbf{proj_1}(\{e\}(n)))$, that is, the function $f(n) \stackrel{\mathrm{Def}}{=} \mathbf{proj_1}(\{e\}(n))$ solves the computational problem expressed by the formula $\forall x \, \exists y \, A(x, y)$.

We generalize and improve program extraction by

- permitting abstract structures (instead of only natural numbers),
- adding stronger axioms (instead of only induction on natural numbers),
- permitting limited classical logic and choice principles,

Program extraction for **HA**

Assume $\mathbf{HA} \vdash \forall x\, \exists y\, A(x,y)$ where $A(x,y)$ is atomic.

Then $\mathbf{HA} \vdash e\, \mathbf{r}\, (\forall x\, \exists y\, A(x,y))$, for some e, by Soundness.

This means $\mathbf{HA} \vdash \forall n\, A(n, \mathbf{proj_1}(\{e\}(n)))$, that is, the function $f(n) \stackrel{\mathrm{Def}}{=} \mathbf{proj_1}(\{e\}(n))$ solves the computational problem expressed by the formula $\forall x\, \exists y\, A(x,y)$.

We generalize and improve program extraction by

- permitting abstract structures (instead of only natural numbers),
- adding stronger axioms (instead of only induction on natural numbers),
- permitting limited classical logic and choice principles,
- extracting programs in a realistic programming language (instead of codes e),

Program extraction for **HA**

Assume $\mathbf{HA} \vdash \forall x\, \exists y\, A(x, y)$ where $A(x, y)$ is atomic.

Then $\mathbf{HA} \vdash e\,\mathbf{r}\,(\forall x\, \exists y\, A(x, y))$, for some e, by Soundness.

This means $\mathbf{HA} \vdash \forall n\, A(n, \mathbf{proj_1}(\{e\}(n)))$, that is, the function $f(n) \stackrel{\mathrm{Def}}{=} \mathbf{proj_1}(\{e\}(n))$ solves the computational problem expressed by the formula $\forall x\, \exists y\, A(x, y)$.

We generalize and improve program extraction by

- permitting abstract structures (instead of only natural numbers),
- adding stronger axioms (instead of only induction on natural numbers),
- permitting limited classical logic and choice principles,
- extracting programs in a realistic programming language (instead of codes e),
- extracting simpler programs.

Including abstract mathematics

Including abstract mathematics

Kleene realizability is chained to concrete computational structures since in the clauses for quantifiers the elements of the structure are

Including abstract mathematics

Kleene realizability is chained to concrete computational structures since in the clauses for quantifiers the elements of the structure are

- used as inputs of programs:

$$e\,\mathbf{r}\,(\forall x\,A(x)) \quad\equiv\quad \forall n\,(\{e\}(n)\,\mathbf{r}\,A(n))$$

Including abstract mathematics

Kleene realizability is chained to concrete computational structures since in the clauses for quantifiers the elements of the structure are

- used as inputs of programs:

$$e\,\mathbf{r}\,(\forall x\,A(x)) \quad \equiv \quad \forall n\,(\{e\}(n)\,\mathbf{r}\,A(n))$$

- and returned as outputs of programs:

$$e\,\mathbf{r}\,(\exists x\,A(x)) \quad \equiv \quad e = \mathbf{P}(n,a) \wedge a\,\mathbf{r}\,A(n)$$

Including abstract mathematics

Kleene realizability is chained to concrete computational structures since in the clauses for quantifiers the elements of the structure are

- used as inputs of programs:

$$e\,\mathbf{r}\,(\forall x\, A(x)) \quad \equiv \quad \forall n\,(\{e\}(n)\,\mathbf{r}\,A(n))$$

- and returned as outputs of programs:

$$e\,\mathbf{r}\,(\exists x\, A(x)) \quad \equiv \quad e = \mathbf{P}(n, a) \wedge a\,\mathbf{r}\,A(n)$$

Abstract structures can be included by interpreting quantifiers uniformly:

$$a\,\mathbf{r}\,\forall x\, A(x) \quad \equiv \quad \forall x\; a\,\mathbf{r}\,A(x)$$

$$a\,\mathbf{r}\,\exists x\, A(x) \quad \equiv \quad \exists x\; a\,\mathbf{r}\,A(x)$$

Including abstract mathematics

Kleene realizability is chained to concrete computational structures since in the clauses for quantifiers the elements of the structure are

- used as inputs of programs:

$$e\,\mathbf{r}\,(\forall x\, A(x)) \quad \equiv \quad \forall n\, (\{e\}(n)\,\mathbf{r}\, A(n))$$

- and returned as outputs of programs:

$$e\,\mathbf{r}\,(\exists x\, A(x)) \quad \equiv \quad e = \mathbf{P}(n, a) \wedge a\,\mathbf{r}\, A(n)$$

Abstract structures can be included by interpreting quantifiers uniformly:

$$a\,\mathbf{r}\,\forall x\, A(x) \quad \equiv \quad \forall x\, a\,\mathbf{r}\, A(x)$$
$$a\,\mathbf{r}\,\exists x\, A(x) \quad \equiv \quad \exists x\, a\,\mathbf{r}\, A(x)$$

This uniform interpretation of quantifiers is also used for interpreting second-order arithmetic and set theory.

Kleene's interpretation of quantifiers can be recovered by relativization: $\forall x\, (x \in \mathbb{N} \rightarrow A(x))$, $\exists x\, (x \in \mathbb{N} \wedge A(x))$.

Induction

Recall induction on natural numbers:

$$\frac{P(0) \qquad \forall x\,(P(x) \to P(x+1))}{\forall x \in \mathbb{N}\, P(x)}$$

Induction

Recall induction on natural numbers:

$$\frac{P(0) \qquad \forall x \, (P(x) \to P(x+1))}{\forall x \in \mathbb{N} \, P(x)}$$

Assume "$n \, \mathbf{r} \, \mathbb{N}(x)$" is defined as "$n$ is the unary representation of $x \in \mathbb{N}$".

Induction

Recall induction on natural numbers:

$$\frac{P(0) \qquad \forall x\,(P(x) \to P(x+1))}{\forall x \in \mathbb{N}\, P(x)}$$

Assume "$n\,\mathbf{r}\,\mathbb{N}(x)$" is defined as "$n$ is the unary representation of $x \in \mathbb{N}$".

Then induction is realized as follows:

$$\frac{a\,\mathbf{r}\,P(0) \qquad f\,\mathbf{r}\,(\forall x\,(P(x) \to P(x+1)))}{\mathbf{It}(a, f)\,\mathbf{r}\,(\forall x \in \mathbb{N}\, P(x))}$$

Induction

Recall induction on natural numbers:

$$\frac{P(0) \qquad \forall x\,(P(x) \to P(x+1))}{\forall x \in \mathbb{N}\, P(x)}$$

Assume "$n\,\mathbf{r}\,\mathbb{N}(x)$" is defined as "$n$ is the unary representation of $x \in \mathbb{N}$".
Then induction is realized as follows:

$$\frac{a\,\mathbf{r}\,P(0) \qquad f\,\mathbf{r}\,(\forall x\,(P(x) \to P(x+1)))}{\mathbf{It}(a, f)\,\mathbf{r}\,(\forall x \in \mathbb{N}\, P(x))}$$

where

- $a : \tau(P) \quad (\tau(P) =$ type of realizers of $P)$,
- $f : \tau(P) \to \tau(P)$

Induction

Recall induction on natural numbers:

$$\frac{P(0) \qquad \forall x\,(P(x) \to P(x+1))}{\forall x \in \mathbb{N}\, P(x)}$$

Assume "$n\,\mathbf{r}\,\mathbb{N}(x)$" is defined as "$n$ is the unary representation of $x \in \mathbb{N}$".
Then induction is realized as follows:

$$\frac{a\,\mathbf{r}\,P(0) \qquad f\,\mathbf{r}\,(\forall x\,(P(x) \to P(x+1)))}{\mathbf{It}(a,f)\,\mathbf{r}\,(\forall x \in \mathbb{N}\, P(x))}$$

where

- $a : \tau(P) \quad (\tau(P) = \text{type of realizers of } P)$,
- $f : \tau(P) \to \tau(P)$

and $\mathbf{It}(a,f) : \mathbb{N} \to \tau(P)$ is defined recursively by

$$\mathbf{It}(a,f)(0) = a$$

$$\mathbf{It}(a,f)(n+1) = f(\mathbf{It}(a,f)(n))$$

Other forms of induction

Other forms of induction

Induction on natural numbers is a special case of a more general form of induction which also includes, for example:

Other forms of induction

Induction on natural numbers is a special case of a more general form of induction which also includes, for example:

Induction on lists, trees, . . .

$$\frac{P([]) \qquad \forall x \in A\, \forall l\, (P(l) \to P(x : l))}{\forall x \in \mathbf{List}(A)\, P(x)}$$

Other forms of induction

Induction on natural numbers is a special case of a more general form of induction which also includes, for example:

Induction on lists, trees, . . .

$$\frac{P([]) \qquad \forall x \in A \, \forall l \, (P(l) \to P(x : l))}{\forall x \in \mathbf{List}(A) \, P(x)}$$

Induction on ordinals (or any wellfounded relation $<$)

$$\frac{\forall x \, ((\forall y < x \, P(y)) \to P(x))}{\forall x < \alpha \, P(x)}$$

Other forms of induction

Induction on natural numbers is a special case of a more general form of induction which also includes, for example:

Induction on lists, trees, ...

$$\frac{P([]) \qquad \forall x \in A\, \forall l\, (P(l) \to P(x : l))}{\forall x \in \textbf{List}(A)\, P(x)}$$

Induction on ordinals (or any wellfounded relation $<$)

$$\frac{\forall x\, ((\forall y < x\, P(y)) \to P(x))}{\forall x < \alpha\, P(x)}$$

Bar induction

...

A unifying approach: Monotone induction

A unifying approach: Monotone induction

Let U be a set and $\mathcal{P}(U)$ the powerset of U.

A unifying approach: Monotone induction

Let U be a set and $\mathcal{P}(U)$ the powerset of U.

An operator $\Phi : \mathcal{P}(X) \to \mathcal{P}(X)$ is *monotone* if for all $X, Y \in \mathcal{P}(U)$

$$X \subseteq Y \to \Phi(X) \subseteq \Phi(Y)$$

A unifying approach: Monotone induction

Let U be a set and $\mathcal{P}(U)$ the powerset of U.

An operator $\Phi : \mathcal{P}(X) \to \mathcal{P}(X)$ is *monotone* if for all $X, Y \in \mathcal{P}(U)$

$$X \subseteq Y \to \Phi(X) \subseteq \Phi(Y)$$

Every monotone operator $\Phi : \mathcal{P}(X) \to \mathcal{P}(X)$ has a *least fixed point*, $\mu(\Phi) \in \mathcal{P}(U)$, which can be defined by

$$\mu(\Phi) \stackrel{\mathrm{Def}}{=} \bigcap \{X \in \mathcal{P}(U) \mid \Phi(X) \subseteq X\}$$

A unifying approach: Monotone induction

Let U be a set and $\mathcal{P}(U)$ the powerset of U.

An operator $\Phi : \mathcal{P}(X) \to \mathcal{P}(X)$ is *monotone* if for all $X, Y \in \mathcal{P}(U)$

$$X \subseteq Y \to \Phi(X) \subseteq \Phi(Y)$$

Every monotone operator $\Phi : \mathcal{P}(X) \to \mathcal{P}(X)$ has a *least fixed point*, $\mu(\Phi) \in \mathcal{P}(U)$, which can be defined by

$$\mu(\Phi) \stackrel{\text{Def}}{=} \bigcap\{X \in \mathcal{P}(U) \mid \Phi(X) \subseteq X\}$$

but also by

$$\mu(\Phi) \stackrel{\text{Def}}{=} \bigcup\{\Phi^{\alpha}(\emptyset) \mid \alpha \in \textbf{Ordinals}\}$$

Closure and induction

One can show that indeed $\mu(\Phi)$ is a fixed point of Φ, that is,

$$\Phi(\mu(\Phi)) = \mu(\Phi),$$

and it is the least element of the set $\{X \in \mathcal{P}(U) \mid \Phi(X) \subseteq X\}$.

Therefore the following rules hold:

$$\frac{}{\Phi(\mu(\Phi)) \subseteq \mu(\Phi)}\ \textbf{Cl} \qquad \frac{\Phi(X) \subseteq X}{\mu(\Phi) \subseteq X}\ \textbf{Ind}$$

Similarly for coinduction:

$$\frac{}{\nu(\Phi) \subseteq \Phi(\nu(\Phi))}\ \textbf{CoCl} \qquad \frac{X \subseteq \Phi(X)}{X \subseteq \nu(\Phi)}\ \textbf{Coind}$$

No guardedness condition.

Intuitionistic Fixed Point logic (IFP)

- Intuitionistic first-order logic with equality.
- Constants, function symbols and atomic predicates (not necessarily decidable), depending on applications.
- Free predicate variables $X, Y, \ldots$.
- Inductive and coinductive definitions as least and largest fixed points of monotone predicate transformers.
  Monotonicity is enforced by strict positivity.
- Axioms consisting of *non-computational (nc)*, that is, disjunction-free, formulas that are (classically) true. The choice of axiom depends on applications.

Programs

Programs are type free lambda terms with constructors, pattern matching and recursion:

$$
\begin{array}{rcl}
\mathit{Programs} \ni M, N & ::= & a, b \text{ variables} \\
& | & \mathbf{Nil} \mid \mathbf{L}(M) \mid \mathbf{R}(M) \mid \mathbf{P}(M, N) \\
& | & \mathbf{case}\, M\, \mathbf{of}\, \{Cl_1; \ldots; Cl_n\} \\
& | & \lambda a.\, M \\
& | & M\, N \\
& | & \mathbf{rec}\, M \\
\mathit{Clauses} \ni Cl & ::= & C(a_1, \ldots, a_n) \to M \quad (C \in \mathbf{Nil}, \mathbf{L}, \mathbf{R}, \mathbf{P})
\end{array}
$$

Programs are interpreted lazily in the Scott domain D defined by the recursive domain equation

$$D = (\mathbf{Nil} + \mathbf{L}(D) + \mathbf{R}(D) + \mathbf{P}(D \times D) + \mathbf{F}(D \to D))_\bot$$

and have an adequate lazy operational semantics.

Assigning them recursive types we get a fragment of Haskell.

Realizability for non-Harrop formulas

A formula is Harrop if it contains no disjunction or free predicate variables at a strictly positive position.

$\mathbf{H}(A)$ is realizability by **Nil** for Harrop formulas (next slide).

$$
\begin{aligned}
a\,\mathbf{r}\,A &= (a = \mathbf{Nil} \land \mathbf{H}(A)) && (A \text{ Harrop})\\
a\,\mathbf{r}\,P(\vec{t}) &= \mathbf{R}(P)(\vec{t}, a) && (P \text{ non-H.})\\
c\,\mathbf{r}\,(A \land B) &= \exists a, b\,(c = \mathbf{P}(a, b) \land a\,\mathbf{r}\,A \land b\,\mathbf{r}\,B) && (A, B \text{ non-H.})\\
a\,\mathbf{r}\,(A \land B) &= a\,\mathbf{r}\,A \land \mathbf{H}(B) && (B \text{ Harrop, } A \text{ non-H.})\\
b\,\mathbf{r}\,(A \land B) &= \mathbf{H}(A) \land b\,\mathbf{r}\,B && (A \text{ Harrop, } B \text{ non-H.})\\
c\,\mathbf{r}\,(A \lor B) &= \exists a\,(c = \mathbf{L}(a) \land a\,\mathbf{r}\,A) \lor \exists b\,(c = \mathbf{R}(b) \land b\,\mathbf{r}\,B)\\
c\,\mathbf{r}\,(A \to B) &= \forall a\,(a\,\mathbf{r}\,A \to (c\,a)\,\mathbf{r}\,B) && (A, B \text{ non-H.})\\
b\,\mathbf{r}\,(A \to B) &= \mathbf{H}(A) \to b\,\mathbf{r}\,B && (A \text{ Harrop, } B \text{ non-H.})\\
a\,\mathbf{r}\,\Diamond x\,A &= \Diamond x\,(a\,\mathbf{r}\,A) && (\Diamond \in \{\forall, \exists\}, A \text{ non-H.})
\end{aligned}
$$

Realizability for non-Harrop predicates

To every predicate variable X is assigned a predicate variable $\tilde{X}$ with an extra argument for realizers.

$\mathbf{R}(P)$ means $\lambda(\vec{x}, a) \,.\, a \,\mathbf{r}\, P(\vec{x})$.

$$\begin{aligned}
\mathbf{R}(X) &= \tilde{X} \\
\mathbf{R}(\lambda\vec{x}\, A) &= \lambda(\vec{x}, a)\,(a\,\mathbf{r}\, A) && (A \text{ non-H.}) \\
\mathbf{R}(\Box(\Phi)) &= \Box(\mathbf{R}(\Phi)) && (\Box \in \{\mu, \nu\},\ \Phi \text{ non-H.}) \\
\mathbf{R}(\lambda X\, P) &= \lambda\tilde{X}\,\mathbf{R}(P) && (P \text{ non-H.})
\end{aligned}$$

Realizability for Harrop formulas and predicates

$\mathbf{r}\,A \stackrel{\mathrm{Def}}{=} \exists a\,.\, a\,\mathbf{r}\,A.$

$$
\begin{aligned}
\mathbf{H}(P(\vec{t})) &= \mathbf{H}(P)(\vec{t}) && (P \text{ Harrop})\\
\mathbf{H}(A \wedge B) &= \mathbf{H}(A) \wedge \mathbf{H}(B) && (A, B \text{ Harrop})\\
\mathbf{H}(A \to B) &= \mathbf{r}A \to \mathbf{H}(B) && (B \text{ Harrop})\\
\mathbf{H}(\Diamond x\, A) &= \Diamond x\, \mathbf{H}(A) && (\Diamond \in \{\forall, \exists\},\ A \text{ Harrop})
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{H}(P) &= P && (P \text{ a predicate constant})\\
\mathbf{H}(\lambda \vec{x}\, A) &= \lambda \vec{x}\, \mathbf{H}(A) && (A \text{ Harrop})\\
\mathbf{H}(\Box(\Phi)) &= \Box(\mathbf{H}(\Phi)) && (\Box \in \{\mu, \nu\},\ \Phi \text{ Harrop})\\
\mathbf{H}(\lambda Y\, P) &= \lambda Y\, \mathbf{H}_Y(P) && (P\ \ Y\text{-Harrop})
\end{aligned}
$$

Soundness for IFP

Let RIFP be the extension of IFP by a sort for realizers and axioms describing the equational theory of programs.

Soundness for IFP

Let RIFP be the extension of IFP by a sort for realizers and axioms describing the equational theory of programs.

Theorem

If $\Gamma, \Delta \vdash_{\mathrm{IFP}} A$, where Γ are nc- and Δ Harrop-formulas, then $\Gamma, \mathbf{H}(\Delta) \vdash_{\mathrm{RIFP}} M \,\mathbf{r}\, A$ for some program M.

Realizers of induction and coinduction:

$$\frac{s \,\mathbf{r}\, (\Phi(P) \subseteq P)}{\mathbf{rec}\,(\lambda f\,.\,s \circ \mathbf{map}\, f)\,\mathbf{r}\,(\mu(\Phi) \subseteq P)}\ \mathbf{Ind}$$

$$\frac{s \,\mathbf{r}\, (P \subseteq \Phi(P))}{\mathbf{rec}\,(\lambda f\,.\,\mathbf{map}\, f \circ s)\,\mathbf{r}\,(P \subseteq \nu(\Phi))}\ \mathbf{Coind}$$

No guarded recursion.

Example: Real and natural numbers

- Variables $x, y, \ldots$ are intended to range over abstract real numbers
- Constants and function symbols: $0, 1, +, -, *, /, |\cdot|, \ldots$.
- Atomic predicates: $<, \leq, \ldots$.
- Nc axioms: $\forall x \,.\, x + 0 = x$, $\ldots$.

Example: Real and natural numbers

- Variables $x, y, \dots$ are intended to range over abstract real numbers
- Constants and function symbols: $0, 1, +, -, *, /, |\cdot|, \dots$.
- Atomic predicates: $<, \leq, \dots$.
- Nc axioms: $\forall x \,.\, x + 0 = x$, $\dots$.
- Inductive predicate defining the natural numbers as a subset of the reals numbers: $\mathbb{N} \stackrel{\mathrm{Def}}{=} \mu\,\Phi$, where
  $\Phi = \lambda X\, \lambda x \,.\, x = 0 \lor X(x-1)$.
  We write this more intuitively as $\mathbb{N}(x) \stackrel{\mu}{=} x = 0 \lor \mathbb{N}(x-1)$.

Example: Real and natural numbers

- Variables $x, y, \ldots$ are intended to range over abstract real numbers
- Constants and function symbols: $0, 1, +, -, *, /, |\cdot|, \ldots$.
- Atomic predicates: $<, \leq, \ldots$.
- Nc axioms: $\forall x \,.\, x + 0 = x, \ \ldots$.
- Inductive predicate defining the natural numbers as a subset of the reals numbers: $\mathbb{N} \stackrel{\mathrm{Def}}{=} \mu\,\Phi$, where $\Phi = \lambda X\,\lambda x \,.\, x = 0 \vee X(x-1)$.
  We write this more intuitively as $\mathbb{N}(x) \stackrel{\mu}{=} x = 0 \vee \mathbb{N}(x-1)$.
- Coinductive predicate defining those real numbers that can be approximated by dyadic rationals: $\mathbf{A} \stackrel{\mathrm{Def}}{=} \nu\,\Psi$, where $\Psi = \lambda X\,\lambda x \,.\, \exists n \in \mathbb{N}\, |x - n| \leq 1 \wedge X(2x)$.
  Intuitive notation $\mathbf{A}(x) \stackrel{\nu}{=} \exists n \in \mathbb{N}\, |x - n| \leq 1 \wedge \mathbf{A}(2x)$.

Example: Real and natural numbers

- Variables $x, y, \ldots$ are intended to range over abstract real numbers
- Constants and function symbols: $0, 1, +, -, *, /, |\cdot|, \ldots$.
- Atomic predicates: $<, \leq, \ldots$.
- Nc axioms: $\forall x \,.\, x + 0 = x$, ...
- Inductive predicate defining the natural numbers as a subset of the reals numbers: $\mathbb{N} \stackrel{\mathrm{Def}}{=} \mu\,\Phi$, where
  $\Phi = \lambda X\, \lambda x \,.\, x = 0 \vee X(x-1)$.
  We write this more intuitively as $\mathbb{N}(x) \stackrel{\mu}{=} x = 0 \vee \mathbb{N}(x-1)$.
- Coinductive predicate defining those real numbers that can be approximated by dyadic rationals: $\mathbf{A} \stackrel{\mathrm{Def}}{=} \nu\,\Psi$, where
  $\Psi = \lambda X\, \lambda x \,.\, \exists n \in \mathbb{N}\, |x - n| \leq 1 \wedge X(2x)$.
  Intuitive notation $\mathbf{A}(x) \stackrel{\nu}{=} \exists n \in \mathbb{N}\, |x - n| \leq 1 \wedge \mathbf{A}(2x)$.

One can prove $\mathbf{A}(x) \leftrightarrow \forall k \in \mathbb{N}\, \exists q \in \mathbb{Q}\, |x - q| \leq 2^{-k}$ where $\mathbb{Q}$ is the set of the rational numbers, defined as usual.

Accessible induction

The *accessible part* of a binary relation $\prec$ is defined inductively by

$$\mathbf{Acc}_{\prec}(x) \stackrel{\mu}{=} \forall y \prec x\, \mathbf{Acc}_{\prec}(y)$$

that is, $\mathbf{Acc}_{\prec} = \mu(\Phi)$ where $\Phi \stackrel{\mathrm{Def}}{=} \lambda X\, \lambda x\, \forall y \prec x\, X(y)$.

P is *progressive* if $\Phi(P) \subseteq P$, that is, $\mathbf{Prog}_{\prec}(P)$ holds where

$$\mathbf{Prog}_{\prec}(P) \stackrel{\mathrm{Def}}{=} \forall x\, (\forall y \prec x P(y) \to P(x))\,.$$

Accessible induction, is an instance of the rule of s.p. induction:

$$\frac{\mathbf{Prog}_{\prec}(P)}{\mathbf{Acc}_{\prec} \subseteq P}\ \mathbf{AccI}_{\prec}(P)$$

Realizing accessible induction

Assume P is non-Harrop and $\prec$ is Harrop (the most common case).

$$\frac{s \,\mathbf{r}\, \mathbf{Prog}_{\prec}(P)}{(\mathbf{rec}\, s)\,\mathbf{r}\,(\mathbf{Acc}_{\prec} \subseteq P)}\ \mathbf{WfI}_{\prec}(P)$$

Brouwer's Thesis and Wellfounded induction

Elements beginning an infinite descending sequence can be characterized coinductively by

$$\mathbf{Path}_{\prec}(x) \stackrel{\nu}{=} \exists y \prec x \, \mathbf{Path}_{\prec}(y)$$

$\neg\mathbf{Path}_{\prec}(x)$ and $\mathbf{Acc}_{\prec}(x)$ are equivalent and both are Harrop formulas (provided $\prec$ is disjunction-free).
Therefore we can postulate the axiom

$$\mathbf{BT}_{\prec} \qquad \forall x \, (\neg\mathbf{Path}_{\prec}(x) \to \mathbf{Acc}_{\prec}(x))$$

which can be viewed as an abstract version of *Brouwer's Thesis* (stating that barred sequences of natural numbers are inductively barred). $\mathbf{BT}_{\prec}$ implies *Wellfounded Induction*:

$$\frac{\mathbf{Prog}_{\prec}(P)}{\neg\mathbf{Path}_{\prec} \subseteq P} \; \mathbf{WfI}_{\prec}(P)$$

Wellfounded induction has the same realizer as accessible induction.

The Archimedean property

The Archimedean property of real numbers can be expressed by stating that there are no infinite numbers:

$$\mathrm{AP} \qquad \forall x \, \neg \infty(x)$$

where infinite numbers are characterized coinductively:

$$\infty(x) \;\stackrel{\nu}{=}\; x \geq 0 \wedge \infty(x-1)\,.$$

Lemma

$\forall x \, (\infty(x) \leftrightarrow \forall y \in \mathbb{N} \; y \leq x)$.

Proof

$\forall y \in \mathbb{N} \, \forall x \, (\infty(x) \to y \leq x)$, by induction.

$\forall x \, ((\forall y \in \mathbb{N} \; y \leq x) \to \infty(x))$, by coinduction.

Archimedean Induction

Setting $y \prec x \stackrel{\mathrm{Def}}{=} x \geq 0 \wedge y = x - 1$, clearly $\infty(x) \leftrightarrow \mathbf{Path}_{\prec}(x)$. Therefore, by the Archimedean property, $\mathbf{Path}_{\prec}$ is empty, and hence, by wellfounded induction,

$$\frac{\forall x\, ((x \geq 0 \to P(x-1)) \to P(x))}{\forall x\, P(x)}\ \mathrm{AI}(P)$$

We call this *Archimedean Induction*.

Equivalent (more useful) form (q is any fixed positive rational):

$$\frac{\forall x \in B \setminus \{0\}\, (P(x) \vee (|x| \leq q \wedge B(2x) \wedge (P(2x) \to P(x))))}{\forall x \in B \setminus \{0\}\, P(x)}\ \mathbf{AIB}_q(B, P)$$

Application: From signed digits to infinite Gray code

Coinductive characterizations of reals that have

- a signed digit representation
  $\mathbf{C}(x) \overset{\nu}{=} \exists d \in \{-1, 0, 1\} \, (|x - d/2| \leq 1/2 \wedge \mathbf{C}(2x - d))$,
- an infinite Gray code
  $\mathbf{G}(x) \overset{\nu}{=} (-1 \leq x \leq 1) \wedge (x \neq 0 \to x \leq 0 \vee x \geq 0) \wedge \mathbf{G}(1 - 2|x|)$.

Realizers of $\mathbf{C}(x)$ are total streams of signed digits.

Realizers of $\mathbf{G}(x)$ are streams of binary digits (L,R) that may be undefined at one point.

Both are admissible representations of the reals but infinite Gray code is in addition *unique*.

Using Archimedean induction one can show $\mathbf{C} \subseteq \mathbf{G}$ and extract a conversion between the two representations.

Extracted program (**C** ⊆ **G**)

```
stog :: SDrep -> InfGrayCode
stog p = case head p of {
  -1 -> L : stog (tail p) ;
   1 -> R : nh (nall (tail p)) ;
   0 -> let { q = stog (tail p) }
        in head q : R : nh (tail q)
 }

nall (L : q) = R : neg q
nall (R : q) = L : neg q

nh (L : q) = R : q
nh (R : q) = L : q
```

Extracted program of the converse inclusion ($\mathbf{G} \subseteq \mathbf{C}$)

```
stog :: InfGrayCode -> SDrep
stog q = case head q of {
  L:q' -> (-1) : gtos p' ;
  R:q' -> 1 : gtos (nh q') ;
  c:R:q'' -> 0 : gtos (c : nh q'')
 }
```

This program can be extracted as well - but not in IFP!

Why?

Programs and rules for concurrency

- Add a new formula construct $\mathbf{S}_2(A)$ which admits 2 concurrent processes as realizers ...

Programs and rules for concurrency

- Add a new formula construct $\mathbf{S}_2(A)$ which admits 2 concurrent processes as realizers ...
- ...and add a new program constructor $\mathbf{Amb}(a_1, a_2)$ for the concurrent execution of the processes a_i (motivated by McCarthy's Amb).

Programs and rules for concurrency

- Add a new formula construct $\mathbf{S}_2(A)$ which admits 2 concurrent processes as realizers ...
- ... and add a new program constructor $\mathbf{Amb}(a_1, a_2)$ for the concurrent execution of the processes a_i (motivated by McCarthy's Amb).
- $\mathbf{Amb}(a_1, a_2)$ realizes $\mathbf{S}_2(A)$ iff at least one a_i is defined and all defined a_i realize A.

Overview of further applications of program extraction

Overview of further applications of program extraction

- Discrete structures
  - Quotient and remainder on natural numbers.
  - Dijkstra's algorithm (1997, Benl, Schwichtenberg): *Reachable nodes in a weighted graph*
  - Warshall Algorithm (2001, Schwichtenberg, Seisenberger, B): *Transitive closure of a relation*

Overview of further applications of program extraction

- Discrete structures
  - Quotient and remainder on natural numbers.
  - Dijkstra's algorithm (1997, Benl, Schwichtenberg):
    Reachable nodes in a weighted graph
  - Warshall Algorithm (2001, Schwichtenberg, Seisenberger, B):
    Transitive closure of a relation
- Programs from classical proofs
  - GCD (1995, B, Schwichtenberg):
    Uses the Friedman/Dragalin A-translation
  - Dickson's Lemma (2001, Schwichtenberg, Seisenberger, B):
    F/D A-translation in infinite combinatorics
  - Higman's Lemma (2008, Seisenberger):
    Uses F/D A-translation and classical countable choice
  - Fibonacci numbers from a classical proofs (2002, Buchholz, Schwichtenberg, B):
    Uses F/D A-translation to obtain fast program

- Lambda calculus:
  - Extraction of normalization-by-evaluation (NbE) (2006, Berghofer, Letouzey, Schwichtenberg, B): *Extraction of NbE from Tait's proof of strong normalization for the typed lambda calculus (in Isabelle, Coq, Minlog)*

- Lambda calculus:
  - Extraction of normalization-by-evaluation (NbE) (2006, Berghofer, Letouzey, Schwichtenberg, B):
    Extraction of NbE from Tait's proof of strong normalization for the typed lambda calculus (in Isabelle, Coq, Minlog)

- Real numbers
  - Cauchy sequences vs signed digit representation (SD):
    Function vs stream representation, arithmetic operations.
  - Integration w.r.t. SD (2011, B):
    Real functions are given by trees realizing a nested coinductive/inductive definition

- Lambda calculus:
  - Extraction of normalization-by-evaluation (NbE) (2006, Berghofer, Letouzey, Schwichtenberg, B):
    Extraction of NbE from Tait's proof of strong normalization for the typed lambda calculus (in Isabelle, Coq, Minlog)
- Real numbers
  - Cauchy sequences vs signed digit representation (SD):
    Function vs stream representation, arithmetic operations.
  - Integration w.r.t. SD (2011, B):
    Real functions are given by trees realizing a nested coinductive/inductive definition
- Lists
  - List reversal
    Uses F/D A-translation to extract linear program from naive proof
  - In-place Quicksort (2014, Seisenberger, Woods, B):
    Extracts an 'imperative' program

- Satisfiabilty testing
  - Extraction of a SAT-solver from completeness proof for DPLL (2015, B, Forsberg, Lawrence, Seisenberger)

- Satisfiabilty testing
  - Extraction of a SAT-solver from completeness proof for DPLL (2015, B, Forsberg, Lawrence, Seisenberger)

- Parsing
  - Extraction of monadic parser combinators and left-recursion elimination (Jones, Seisenberger, B)

- Satisfiabilty testing
  - Extraction of a SAT-solver from completeness proof for DPLL (2015, B, Forsberg, Lawrence, Seisenberger)
- Parsing
  - Extraction of monadic parser combinators and left-recursion elimination (Jones, Seisenberger, B)
- Extensions: Extraction of
  - concurrent programs (Miyamoto, Petrovska, Schwichtenberg, Spreen, Takayama, Tsuiki, B)
  - imperative programs with explicit memory management from Separation Logic (Reus, B)
  - modulus of uniform continuity from Fan Theorem (B)

Concluding remarks

- The Curry-Howard correspondence and program extraction are usually associated with constructive type theory (CTT), which is implemented, e.g., in Coq and Agda.

Concluding remarks

- The Curry-Howard correspondence and program extraction are usually associated with constructive type theory (CTT), which is implemented, e.g., in Coq and Agda.

- CTT rejects the classical notions of 'structure' and 'truth' and *identifies* proofs with programs.

Concluding remarks

- The Curry-Howard correspondence and program extraction are usually associated with constructive type theory (CTT), which is implemented, e.g., in Coq and Agda.
- CTT rejects the classical notions of 'structure' and 'truth' and *identifies* proofs with programs.
- The agenda of CTT (in particular its homotopic version) is foundational: CTT proposes a new kind of mathematics.

Concluding remarks

- The Curry-Howard correspondence and program extraction are usually associated with constructive type theory (CTT), which is implemented, e.g., in Coq and Agda.
- CTT rejects the classical notions of 'structure' and 'truth' and *identifies* proofs with programs.
- The agenda of CTT (in particular its homotopic version) is foundational: CTT proposes a new kind of mathematics.
- In contrast, program extraction is rooted in first-order logic with a classical Tarskian semantics.

Concluding remarks

- The Curry-Howard correspondence and program extraction are usually associated with constructive type theory (CTT), which is implemented, e.g., in Coq and Agda.
- CTT rejects the classical notions of 'structure' and 'truth' and *identifies* proofs with programs.
- The agenda of CTT (in particular its homotopic version) is foundational: CTT proposes a new kind of mathematics.
- In contrast, program extraction is rooted in first-order logic with a classical Tarskian semantics.
- Program extraction is a technique to obtain provably correct programs from proofs in 'ordinary' mathematics.

Some references

A S Troelstra, D van Dalen, Constructivism in Mathematics, Vol. I, N-H, 1988.

D van Dalen, Logic and Structure, 3rd edition, Springer, 1994.

B, K Miyamoto, H Schwichtenberg, M Seisenberger, Minlog - A Tool for Program Extraction for Supporting Algebra and Coalgebra, LNCS 6859, 2011.

B, From coinductive proofs to exact real arithmetic: theory and applications, Logical Methods in Comput. Sci. 7, 2011, H Schwichtenberg, S S Wainer, Proofs and Computations, Cambridge University Press, 2012.

H Tsuiki. Real Number Computation through Gray Code Embedding. Theor. Comput. Sci. 284, 2002.

B, A Lawrence, F Nordvall, M Seisenberger. Extracting verified decision procedures: DPLL and Resolution. Logical Methods in Computer Science 11, 2015.

B, O Petrovska. Optimized program extraction for induction and coinduction CiE 2018, LNCS 10936, 2018.

Extracting the fan functional

Given: A continuous functional $F : (\mathbb{N} \to \mathbb{B}) \to \mathbb{N} \quad (\mathbb{B} = \{0, 1\})$

Extracting the fan functional

Given: A continuous functional $F : (\mathbb{N} \to \mathbb{B}) \to \mathbb{N} \quad (\mathbb{B} = \{0, 1\})$

Since $\mathbb{N} \to \mathbb{B}$ is compact, F is uniformly continuous (fan theorem).

Extracting the fan functional

Given: A continuous functional $F : (\mathbb{N} \to \mathbb{B}) \to \mathbb{N} \quad (\mathbb{B} = \{0, 1\})$

Since $\mathbb{N} \to \mathbb{B}$ is compact, F is uniformly continuous (fan theorem).

Wanted: The modulus of uniform continuity of F.

Extracting the fan functional

Given: A continuous functional $F : (\mathbb{N} \to \mathbb{B}) \to \mathbb{N} \quad (\mathbb{B} = \{0, 1\})$

Since $\mathbb{N} \to \mathbb{B}$ is compact, F is uniformly continuous (fan theorem).

Wanted: The modulus of uniform continuity of F.

That is, the least n such that for all $\alpha, \beta : \mathbb{N} \to \mathbb{B}$,

$$\text{if } \alpha(k) = \beta(k) \text{ for all } k < n, \text{ then } F(\alpha) = F(\beta).$$

Extracting the fan functional

Given: A continuous functional $F : (\mathbb{N} \to \mathbb{B}) \to \mathbb{N}$ $\quad (\mathbb{B} = \{0, 1\})$

Since $\mathbb{N} \to \mathbb{B}$ is compact, F is uniformly continuous (fan theorem).

Wanted: The modulus of uniform continuity of F.

That is, the least n such that for all $\alpha, \beta : \mathbb{N} \to \mathbb{B}$,

$$\text{if } \alpha(k) = \beta(k) \text{ for all } k < n, \text{ then } F(\alpha) = F(\beta).$$

The function $F \mapsto n$ is called *fan functional*.

Extracting the fan functional

Given: A continuous functional $F : (\mathbb{N} \to \mathbb{B}) \to \mathbb{N} \quad (\mathbb{B} = \{0, 1\})$

Since $\mathbb{N} \to \mathbb{B}$ is compact, F is uniformly continuous (fan theorem).

Wanted: The modulus of uniform continuity of F.

That is, the least n such that for all $\alpha, \beta : \mathbb{N} \to \mathbb{B}$,

$$\text{if } \alpha(k) = \beta(k) \text{ for all } k < n, \text{ then } F(\alpha) = F(\beta).$$

The function $F \mapsto n$ is called *fan functional*.

We show that a program computing the fan functional can be extracted from a proof that F is uniformly continuous.

Extracting the fan functional

Given: A continuous functional $F : (\mathbb{N} \to \mathbb{B}) \to \mathbb{N} \quad (\mathbb{B} = \{0, 1\})$

Since $\mathbb{N} \to \mathbb{B}$ is compact, F is uniformly continuous (fan theorem).

Wanted: The modulus of uniform continuity of F.

That is, the least n such that for all $\alpha, \beta : \mathbb{N} \to \mathbb{B}$,

$$\text{if } \alpha(k) = \beta(k) \text{ for all } k < n, \text{ then } F(\alpha) = F(\beta).$$

The function $F \mapsto n$ is called *fan functional*.

We show that a program computing the fan functional can be extracted from a proof that F is uniformly continuous.

The proof takes place in an extension of IFP by a 'bang operator'.

Is the fan functional really computable?

Computing the fan functional seems an impossible task since we have:

Theorem

It is impossible to compute from a continuous functional $F : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ a modulus of (pointwise) continuity.

The extracted program

Declarations:

```
type N = Int          -- 0,1,2,...
type B = Int          -- 0,1
type B1 = N -> B      -- Cantor space
type B2 = B1 -> N

(***) :: [B] -> B1 -> B1
s *** alpha = \n-> if n < length s
                   then s !! n
                   else alpha (n - length s)
```

The extracted program

```
minarg, maxarg :: B2 -> [B] -> B1

-- minarg f s = some alpha s.t. f (s *** alpha) is minimal

minarg f s = let {
                   s0 = s ++ [0] ; s1 = s ++ [1] ;
                   alpha0 = minarg f s0 ;
                   alpha1 = minarg f s1
                  }
             in if f (s0 *** alpha0) <= f (s1 *** alpha1)
                then [0] *** alpha0
                else [1] *** alpha1

maxarg f s = ...
```

Fan functional

```
-- testing constancy
isconst :: B2 -> [B] -> Bool
isconst f s =
   f (s *** (minarg f s)) == f (s *** (maxarg f s))

fan :: B2 -> N
fan f = aux []

  where

--  aux :: [B] -> N
    aux s = if isconst f s
            then 0
            else 1 + max (aux (s++[0])) (aux (s++[1]))
```

Bang!

If A is a formula, then $!A$ is a Harrop formula with
$a \mathbf{r} !A \stackrel{\text{Def}}{=} a = \textbf{Nil} \wedge \forall a\, (a \mathbf{r} A)$.

Bang!

If A is a formula, then $!A$ is a Harrop formula with
$a \mathbf{r} \, !A \stackrel{\text{Def}}{=} a = \mathbf{Nil} \wedge \forall a \, (a \, \mathbf{r} \, A)$.

For example, $\mathbf{Nil} \, \mathbf{r} \, !(\bot \to A)$ since, $a \, \mathbf{r} \, (\bot \to A) \equiv \bot \to a \, \mathbf{r} \, A$.

Bang!

If A is a formula, then $!A$ is a Harrop formula with $a\,\mathbf{r}\,!A \stackrel{\mathrm{Def}}{=} a = \mathbf{Nil} \wedge \forall a\,(a\,\mathbf{r}\,A)$.

For example, $\mathbf{Nil}\,\mathbf{r}\,!(\bot \to A)$ since, $a\,\mathbf{r}\,(\bot \to A) \equiv \bot \to a\,\mathbf{r}\,A$.

But $!(0 = 0 \vee 0 = 1)$ is not realizable.

Bang!

If A is a formula, then $!A$ is a Harrop formula with
$a\,\mathbf{r}\,!A \overset{\mathrm{Def}}{=} a = \mathbf{Nil} \wedge \forall a\,(a\,\mathbf{r}\,A)$.

For example, $\mathbf{Nil}\,\mathbf{r}\,!(\bot \to A)$ since, $a\,\mathbf{r}\,(\bot \to A) \equiv \bot \to a\,\mathbf{r}\,A$.

But $!(0 = 0 \vee 0 = 1)$ is not realizable.

Intuitively, $!A$ expresses that A is true (realizable) for trivial reasons.

Bang!

If A is a formula, then $!A$ is a Harrop formula with
$a \, \mathbf{r} \, !A \stackrel{\mathrm{Def}}{=} a = \mathbf{Nil} \land \forall a \, (a \, \mathbf{r} \, A)$.

For example, $\mathbf{Nil} \, \mathbf{r} \, !(\bot \to A)$ since, $a \, \mathbf{r} \, (\bot \to A) \equiv \bot \to a \, \mathbf{r} \, A$.

But $!(0 = 0 \lor 0 = 1)$ is not realizable.

Intuitively, $!A$ expresses that A is true (realizable) for trivial reasons.

A realizable version of the law of excluded middle:

$$\frac{\neg A \to B \qquad A \to !B}{B} \; !\mathbf{LEM}$$

Bang!

If A is a formula, then $!A$ is a Harrop formula with
$a\,\mathbf{r}\,!A \stackrel{\text{Def}}{=} a = \mathbf{Nil} \land \forall a\,(a\,\mathbf{r}\,A)$.

For example, $\mathbf{Nil}\,\mathbf{r}\,!(\bot \to A)$ since, $a\,\mathbf{r}\,(\bot \to A) \equiv \bot \to a\,\mathbf{r}\,A$.

But $!(0 = 0 \lor 0 = 1)$ is not realizable.

Intuitively, $!A$ expresses that A is true (realizable) for trivial reasons.

A realizable version of the law of excluded middle:

$$\frac{\neg A \to B \qquad A \to\, !B}{B}\ !\mathbf{LEM}$$

Realizing !LEM:

Assume $a\,\mathbf{r}\,(\neg A \to B)$ and $\mathbf{Nil}\,\mathbf{r}\,(A \to\, !B)$, that is,
$\neg\exists c\, c\,\mathbf{r}\,A \to a\,\mathbf{r}\,B$ and $\exists c\, c\,\mathbf{r}\,A \to \forall b\, b\,\mathbf{r}\,B$.

Using the (classical) law of excluded middle, we conclude $a\,\mathbf{r}\,B$.