IFP - A Logic for Program Extraction[1]

Ulrich Berger
Swansea University

BCTCS

Durham, April 15-17, 2019

[1] available at `www.cs.swan.ac.uk/∼csulrich/slides.html`

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*.

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*.

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

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

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

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*.

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

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

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

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

- Logic and constructivism
- Program extraction
- Example: Extracting the fan functional
- Concluding remarks

Logic and constructivism

- Predicate logic
- Peano Arithmetic
- Constructive proofs
- The Curry-Howard Correspondence

Predicate logic (a.k.a. first-order logic, FOL)

Gottlob Frege (1848 - 1925)

Predicate logic was introduced by Frege in his *Begriffsschrift*.

The language of predicate logic

The language of predicate logic

Example: “Every positive number has a positive square root”

The language of predicate logic

Example: "Every positive number has a positive square root"

$$\forall x\,(x > 0 \rightarrow \exists y\,(y > 0 \wedge x = y * y))$$

The language of predicate logic

Example: "Every positive number has a positive square root"

$$\forall x\,(x > 0 \rightarrow \exists y\,(y > 0 \wedge x = y * y))$$

The *language*, $\mathcal{L} = (\mathcal{C}, \mathcal{F}, \mathcal{P})$, for this formula consists of

Constants: $\mathcal{C} = \{0\}$

Function symbols: $\mathcal{F} = \{*\}$

Predicate symbols: $\mathcal{P} = \{>\}$

The language of predicate logic

Example: "Every positive number has a positive square root"

$$\forall x\,(x > 0 \rightarrow \exists y\,(y > 0 \wedge x = y * y))$$

The *language*, $\mathcal{L} = (\mathcal{C}, \mathcal{F}, \mathcal{P})$, for this formula consists of

Constants: $\mathcal{C} = \{0\}$

Function symbols: $\mathcal{F} = \{*\}$

Predicate symbols: $\mathcal{P} = \{>\}$

The elements of $\mathcal{L}$ are also called *non-logical symbols*. The choice of $\mathcal{L}$ may vary depending on the intended application.

The language of predicate logic

Example: "Every positive number has a positive square root"

$$\forall x\,(x > 0 \to \exists y\,(y > 0 \wedge x = y * y))$$

The *language*, $\mathcal{L} = (\mathcal{C}, \mathcal{F}, \mathcal{P})$, for this formula consists of

Constants: $\mathcal{C} = \{0\}$

Function symbols: $\mathcal{F} = \{*\}$

Predicate symbols: $\mathcal{P} = \{>\}$

The elements of $\mathcal{L}$ are also called *non-logical symbols*. The choice of $\mathcal{L}$ may vary depending on the intended application.

The other symbols occurring in a formula of predicate logic are application independent and are called *logical symbols*:

Variables: $x, y, \ldots$

Logical constants: $\top$ ("true"), '$\bot$" (false)

Logical connectives: $\wedge$ ("and"), $\vee$ ("or"), $\to$ ("implies")

Quantifiers: $\forall$ ("for all"), $\exists$ ("exists")

Equality: $=$

Negation can be defined as $\neg A \stackrel{\text{Def}}{=} A \to \bot$

The semantics of predicate logic

Alfred Tarski (1901-1983)

Tarski was the first to systematically study the notion of truth for formulas in predicate logic.

Models

A *model* (or *structure*) $\mathcal{M}$ for a language $\mathcal{L} = (\mathcal{C}, \mathcal{F}, \mathcal{P})$ consists of:

- a nonempty set M, called the *carrier set of* $\mathcal{M}$
- an interpretation in M of
  - the constants in $\mathcal{C}$,
  - the function symbols in $\mathcal{F}$,
  - the predicate symbols in $\mathcal{P}$.

In a given model $\mathcal{M}$, any $\mathcal{L}$-formula is either true or false.

Proofs

A *proof system* is a collection of rules to derive *logically valid* formulas, that is, formulas that hold in all models.

There are many different proof systems. A popular one, due to Gentzen, is called *Natural Deduction* since its rules are close to natural human reasoning.

Gerhard Gentzen (1909 - 1945)

Natural Deduction (version with explicit assumptions)

Assumption rule $\dfrac{}{\Gamma, A \vdash A}$ use

	Introduction rules	Elimination rules
$\wedge$	$\dfrac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B} \wedge^+$	$\dfrac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \wedge_{\mathsf{l}}^- \qquad \dfrac{\Gamma \vdash A \wedge B}{\Gamma \vdash B} \wedge_{\mathsf{r}}^-$
$\rightarrow$	$\dfrac{\Gamma, A \vdash B}{\Gamma \vdash A \rightarrow B} \rightarrow^+$	$\dfrac{\Gamma \vdash A \rightarrow B \qquad \Gamma \vdash A}{\Gamma \vdash B} \rightarrow^-$
$\vee$	$\dfrac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \vee_{\mathsf{l}}^+ \qquad \dfrac{\Gamma \vdash B}{\Gamma \vdash A \vee B} \vee_{\mathsf{r}}^+$	$\dfrac{\Gamma \vdash A \vee B \qquad \Gamma \vdash A \rightarrow C \qquad \Gamma \vdash B \rightarrow C}{\Gamma \vdash C} \vee^-$
$\bot$		$\dfrac{\Gamma \vdash \bot}{\Gamma \vdash A}$ efq $\qquad \dfrac{\Gamma \vdash \neg\neg A}{\Gamma \vdash A}$ raa
$\forall$	$\dfrac{\Gamma \vdash A(x)}{\Gamma \vdash \forall x\, A(x)} \forall^+$ (x not free in Γ)	$\dfrac{\Gamma \vdash \forall x\, A(x)}{\Gamma \vdash A(t)} \forall^-$
$\exists$	$\dfrac{\Gamma \vdash A(t)}{\Gamma \vdash \exists x\, A(x)} \exists^+$	$\dfrac{\Gamma \vdash \exists x\, A(x) \quad \Gamma \vdash \forall x\, (A(x) \rightarrow C)}{\Gamma \vdash C} \exists^-$ (x not free in Γ, C)

Equality rules

	Introduction rule	Elimination rule
$=$	$\dfrac{}{\Gamma \vdash t = t}$	$\dfrac{\Gamma \vdash A(s) \qquad \Gamma \vdash s = t}{\Gamma \vdash A(t)}$

Symmetry and transitivity of equality can be derived.

Short notation for proofs

Short notation for proofs

Instead of $\Gamma \vdash A$ we write A

Short notation for proofs

Instead of $\Gamma \vdash A$ we write A

Changes and uses of the antecedent Γ are indicated by labelled assumptions $u : A$:

Original	Short notation
$\dfrac{\Gamma, A \vdash B}{\Gamma \vdash A \to B} \to^+$	$\dfrac{B}{A \to B} \to^+ u : A$
$\dfrac{}{\Gamma, A \vdash A}$ use	$u : A$

Examples

$$\dfrac{\dfrac{u : A \wedge B \to C \qquad \dfrac{v : A \qquad w : B}{A \wedge B}\wedge^{+}}{C}\to^{-}}{\dfrac{\dfrac{B \to C}{A \to (B \to C)}\to^{+} v : A}{(A \wedge B \to C) \to (A \to (B \to C))}\to^{+} u : A \wedge B \to C}\to^{+} w : B$$

Examples

$$\dfrac{\dfrac{u : A \wedge B \to C \qquad \dfrac{v : A \qquad w : B}{A \wedge B}\,\wedge^{+}}{C}\,\to^{-}}{\dfrac{\dfrac{B \to C}{A \to (B \to C)}\,\to^{+} v : A}{(A \wedge B \to C) \to (A \to (B \to C))}\,\to^{+} w : B}\,\to^{+} u : A \wedge B \to C$$

$$\dfrac{u : A \vee B \qquad \dfrac{\dfrac{\dfrac{v : A}{B \vee A}\,\vee^{+}_{\mathsf{r}}}{A \to B \vee A}\,\to^{+} v : A \qquad \dfrac{\dfrac{w : B}{B \vee A}\,\vee^{+}_{\mathsf{l}}}{B \to B \vee A}\,\to^{+} w : B}{B \vee A}\,\vee^{-}}{A \vee B \to B \vee A}\,\to^{+} u : A \vee B$$

Completeness

In 1929 Kurt Gödel proved that there is a *sound and complete* proof calculus for first-order logic (equivalent to natural deduction):

Completeness Theorem

A formula in first-order logic is logically valid if and only it is provable.

$$\models A \quad \Leftrightarrow \quad \vdash A$$

Kurt Gödel (1906-1978)

Peano Arithmetic

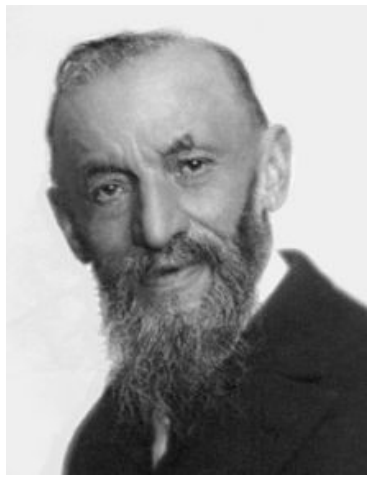
In order to prove statements that are true in the structure $\mathcal{N}$ of natural numbers, Peano introduced the following axioms:

Peano 1 $\forall x\,(x + 1 \neq 0)$

Peano 2 $\forall x, y\,(x + 1 = y + 1 \rightarrow x = y)$

Peano 3 (Induction) For every formula $A(x)$:

$$A(0) \wedge \forall x(A(x) \rightarrow A(x + 1)) \rightarrow \forall x\, A(x)$$



Giuseppe Peano (1858 - 1932)

The set of theorems provable from the Peano Axioms is called *Peano Arithmetic (***PA***)*.

Classical logic

Classical logic

Predicate logic, with Tarskian semantics and the complete proof calculus, is often called *classical logic* because it is the most traditional and widely used logic.

Classical logic

Predicate logic, with Tarskian semantics and the complete proof calculus, is often called *classical logic* because it is the most traditional and widely used logic.

In classical logic the *Law of Excluded Middle* is valid (and hence provable):

$$A \vee \neg A$$

Intuitionistic logic

Intuitionistic logic

The constructive understanding of a proof of a disjunction $A \vee B$ includes an effective procedure that determines which of A or B is true.

Intuitionistic logic

The constructive understanding of a proof of a disjunction $A \vee B$ includes an effective procedure that determines which of A or B is true.

Therefore, the classically valid law of excluded middle, $A \vee \neg A$, is rejected by constructivists since there is no effective procedure the decides, for any formula A, whether A or $\neg A$ holds.

Intuitionistic logic

The constructive understanding of a proof of a disjunction $A \vee B$ includes an effective procedure that determines which of A or B is true.

Therefore, the classically valid law of excluded middle, $A \vee \neg A$, is rejected by constructivists since there is no effective procedure the decides, for any formula A, whether A or $\neg A$ holds.

A constructive alternative to classical logic is *intuitionistic logic* which is obtained from classical logic by removing the principle of proof by contradiction ($\neg\neg A \to A$, that is, raa) from natural deduction.

Intuitionistic logic

The constructive understanding of a proof of a disjunction $A \vee B$ includes an effective procedure that determines which of A or B is true.

Therefore, the classically valid law of excluded middle, $A \vee \neg A$, is rejected by constructivists since there is no effective procedure the decides, for any formula A, whether A or $\neg A$ holds.

A constructive alternative to classical logic is *intuitionistic logic* which is obtained from classical logic by removing the principle of proof by contradiction ($\neg\neg A \to A$, that is, raa) from natural deduction.

We write $\Gamma \vdash_i A$ if A is provable from Γ in intuitionistic logic.

Disjunction and Existence Theorem for intuitionistic logic

Disjunction Theorem for Intuitionistic logic

If $\vdash_i A \vee B$, then $\vdash_i A$ or $\vdash_i B$.

Existence Theorem for Intuitionistic logic

From an intuitionistic proof of a formula of the form $\exists x\, A(x)$ one can extract a term t such that $A(t)$ is provable.

Disjunction and Existence Theorem for intuitionistic logic

Disjunction Theorem for Intuitionistic logic

If $\vdash_i A \vee B$, then $\vdash_i A$ or $\vdash_i B$.

Existence Theorem for Intuitionistic logic

From an intuitionistic proof of a formula of the form $\exists x\, A(x)$ one can extract a term t such that $A(t)$ is provable.

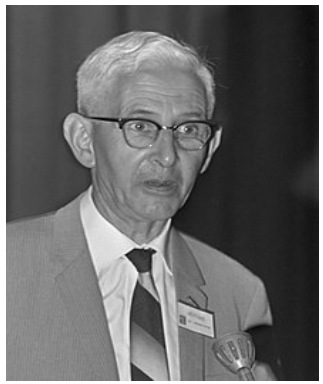
Corresponding theorems for classical logic do *not* hold. However, we have

Herbrand's Theorem

From a classical proof of a formula of the form $\exists x\, A(x)$, A quantifier free, one can extract finitely many terms $t_1, \ldots t_n$ such that $A(t_1) \vee \ldots \vee A(t_n)$ is (classically) provable.

Heyting Arithmetic

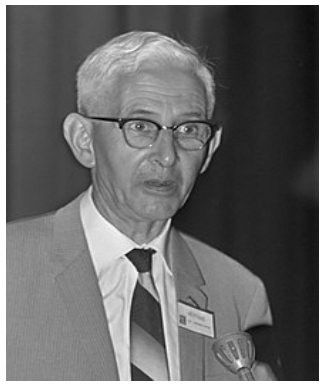
Peano Arithmetic with intuitionistic logic is called *Heyting Arithmetic*, **HA**.



Arendt Heyting (1898 - 1980)

Heyting Arithmetic

Peano Arithmetic with intuitionistic logic is called *Heyting Arithmetic*, **HA**.



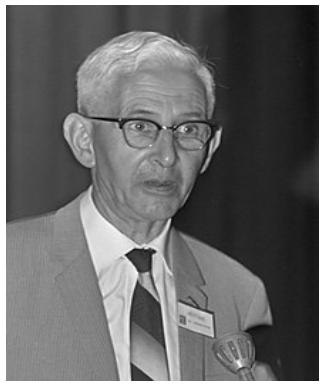
Arendt Heyting (1898 - 1980)

In **HA**:

- The Disjunction and Existence Theorems continue to hold.

Heyting Arithmetic

Peano Arithmetic with intuitionistic logic is called *Heyting Arithmetic*, **HA**.



Arendt Heyting (1898 - 1980)

In **HA**:

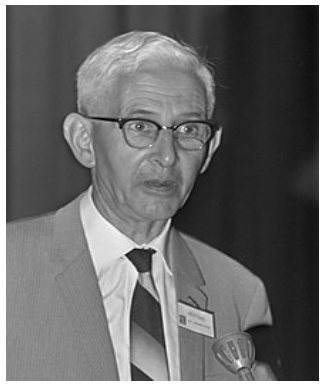
- The Disjunction and Existence Theorems continue to hold.
- The (universally generalized) law of excluded middle

$$\forall \vec{x} \, (A(\vec{x}) \vee \neg A(\vec{x}))$$

is provable for all quantifier free formulas $A(\vec{x})$.

Heyting Arithmetic

Peano Arithmetic with intuitionistic logic is called *Heyting Arithmetic*, **HA**.



Arendt Heyting (1898 - 1980)

In **HA**:

- The Disjunction and Existence Theorems continue to hold.
- The (universally generalized) law of excluded middle

$$\forall \vec{x}\,(A(\vec{x}) \vee \neg A(\vec{x}))$$

  is provable for all quantifier free formulas $A(\vec{x})$.
- More generally, **HA** and **PA** prove the same Π^0_2 formulas, that is, formulas of the form $\forall \vec{x}\,\exists \vec{y}\,A(\vec{x},\vec{y})$, $A(\vec{x},\vec{y})$ quantifier free (Parsons).

Semantics of Intuitionistic logic

Intuitionistic logic is incomplete w.r.t. Tarskian semantics, since the law of excluded middle is not provable.

However, there are other styles of semantics for which intuitionistic logic is complete and which better bring to light its constructive nature.

An informal semantics with that property is due to Brouwer, Heyting, and Kolmogorov.

Luitzen Egbertus Jan Brouwer
(1881 - 1966)

Andrey Nikolaevich Kolmogorov
(1903 - 1987)

The BHK interpretation

According to the BHK interpretation a formula expresses a *computational problem* which is defined by a description of how to solve it:

A solution to $A \wedge B$ is a pair (a, b) such that

a solves A and b solves B.

A solution to $A \vee B$ is

either $(0, a)$ where a solves A

or $(1, b)$ where b solves B.

A solution to $A \to B$ is a construction that transforms

any solution of A to a solution of B.

The lambda calculus

In the BHK interpretation it is left open what a "construction" is.

Church's *lambda calculus* provides a good notion of construction:

The lambda calculus consists of

- *lambda terms* generated by the rules

x	Variables
$\lambda x\,.\,M$	lambda-abstraction
$M\,N$	Application

- *beta-reduction*

$$(\lambda x\,.\,M)N \quad \to_\beta \quad M[N/x]$$

$M[N/x]$ denotes substitution of the term N for x in the term M.

One usually writes $M\,N\,K$ for $(M\,N)\,K$.

The Curry-Howard correspondence

The *Curry-Howard correspondence* is the observation that intuitionistic natural deduction proofs are in a natural correspondence with the typed lambda calculus or the *typed combinator calculus*.

Since typed lambda terms are the core of functional programming languages such as ML and Haskell (named after Haskell B Curry) one can also say that intuitionistic proofs correspond to programs.

Haskell B Curry (1900-1982)

Intuitionistic ND proofs vs typed lambda calculus

$$\frac{A \qquad B}{A \wedge B} \qquad \frac{M : A \qquad N : B}{(M, N) : A \times B}$$

$$\frac{A \wedge B}{A} \qquad \frac{A \wedge B}{B} \qquad \frac{M : A \times B}{\pi_0(M) : A} \qquad \frac{M : A \times B}{\pi_1(M) : B}$$

$$\frac{B}{A \to B} \to^+ u : A \qquad \frac{M : B}{\lambda x\, M : A \to B}$$

$$\frac{A \to B \qquad A}{B} \qquad \frac{M : A \to B \qquad N : B}{M\, N : B}$$

$$\frac{A}{A \vee B} \qquad \frac{B}{A \vee B} \qquad \frac{M : A}{(0, M) : A \vee B} \qquad \frac{M : B}{(1, M) : A \vee B}$$

$$\frac{A \vee B \qquad A \to C \qquad B \to C}{C} \qquad \frac{M : A \vee B \qquad N : A \to C \qquad K : B \to C}{\mathbf{case}(M, N, K) : C}$$

Program Extraction

- Realizability
- Strictly positive induction
- Intuitionistic Fixed Point Logic (IFP)
- Overview of applications of program extraction

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 ('numerical realizability', 1945).

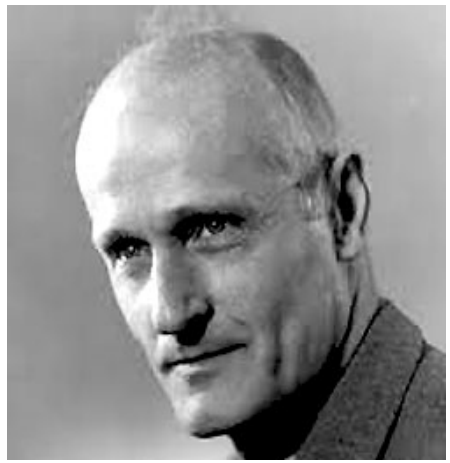
Realizability

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 ('numerical realizability', 1945).



Stephen Kleene (1909 - 1994)

Kleene's numerical 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 $\mathbf{HA} \vdash A$, then $e\,\mathbf{r}\,A$ for some e.

Soundness Theorem

If $\mathbf{HA} \vdash A$, then $e\,\mathbf{r}\,A$ 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 corresponding Curry-Howard lambda-term.
3. For the induction axiom the extracted realizer codes a primitive recursion (iterator).
4. In a formalized version of realizability the correctness of the extracted realizer can again be proven in **HA**, in other words:

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

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

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 $f(n) \stackrel{\text{Def}}{=} \mathbf{proj}_1(\{e\}(n))$ 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 $f(n) \stackrel{\text{Def}}{=} \mathbf{proj}_1(\{e\}(n))$ 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 $f(n) \stackrel{\text{Def}}{=} \mathbf{proj}_1(\{e\}(n))$ 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 $f(n) \stackrel{\text{Def}}{=} \mathbf{proj}_1(\{e\}(n))$ 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 $f(n) \stackrel{\text{Def}}{=} \mathbf{proj}_1(\{e\}(n))$ 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 $f(n) \stackrel{\text{Def}}{=} \mathbf{proj}_1(\{e\}(n))$ 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.

Embracing abstract mathematics

Embracing abstract mathematics

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

Embracing 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:

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

Embracing 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:

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

Embracing 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:

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

The chains are broken by interpreting quantifiers uniformly:

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

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

Embracing 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:

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

The chains are broken 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.

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

- $a : \tau(P) \quad (\tau(P) =$ 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, ...

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

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

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

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

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

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 (exercise) that indeed $\mu(\Phi)$ is a fixed point of Φ, that is,

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

Closure and induction

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

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

Moreover, $\mu(\Phi)$ is the least element of

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

Closure and induction

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

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

Moreover, $\mu(\Phi)$ is the least element of

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

which means that the following rules hold:

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

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.

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.

- Axioms consisting of *non-computational (nc)*, that is, disjunction-free, formulas that are (classically) true. The choice of axiom depends on applications.

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 $\Gamma \vdash_{\mathrm{IFP}} A$, where Γ consists of nc-axioms,
then $\Gamma \vdash_{\mathrm{RIFP}} M\,\mathbf{r}\,A$ for some program M.

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 $x, y, \ldots$ 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: $\mathbb{N} \stackrel{\text{Def}}{=} \mu\, \Phi$, where $\Phi = \lambda X\, \lambda x \,.\, x = 0 \vee X(x - 1)$.
  We write this more intuitively as $\mathbb{N}(x) \stackrel{\mu}{=} x = 0 \vee \mathbb{N}(x - 1)$.

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: $\mathbb{N} \stackrel{\text{Def}}{=} \mu\, \Phi$, 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: $\mathbf{A} \stackrel{\text{Def}}{=} \nu\, \Psi$, 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: $\mathbb{N} \stackrel{\text{Def}}{=} \mu\, \Phi$, 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: $\mathbf{A} \stackrel{\text{Def}}{=} \nu\, \Psi$, 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.

Overview of applications of program extraction

Overview of 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 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)

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 $a\,\mathbf{r}\,!A \stackrel{\text{Def}}{=} a = \mathbf{Nil} \wedge \forall a\,(a\,\mathbf{r}\,A)$.

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 $a\,\mathbf{r}\,!A \stackrel{\text{Def}}{=} a = \mathbf{Nil} \wedge \forall a\,(a\,\mathbf{r}\,A)$.

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

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

Bang!

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

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

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 $a\,\mathbf{r}\,!A \stackrel{\text{Def}}{=} a = \mathbf{Nil} \wedge \forall a\,(a\,\mathbf{r}\,A)$.

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

But $!(0 = 0 \vee 0 = 1)$ 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 $a\,\mathbf{r}\,!A \stackrel{\text{Def}}{=} a = \mathbf{Nil} \wedge \forall a\,(a\,\mathbf{r}\,A)$.

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.

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 $a\,\mathbf{r}\,B$.

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.