Program Extraction From Proofs[1]

Ulrich Berger
Swansea University

Autumn School "Proof and Computation"

Fischbachau, September 16-22, 2018

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

Aims of the course

The aim of this course to show that **logic** is a natural bridge between **mathematics** and **computation**.

We will study how valid reasoning in abstract mathematics leads to provably correct algorithms and hence certified computer programs

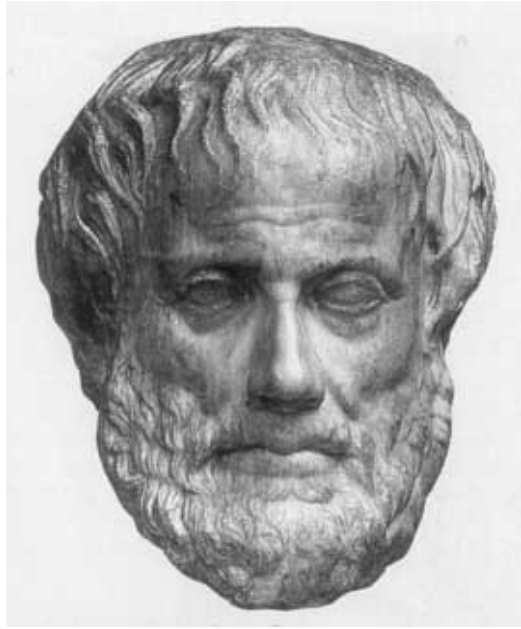
Plan of the course

Lecture 1: Logic and constructivism

Lecture 2: Proofs as programs

Lecture 3: Program extraction

Aristotle (384 - 322 BC)



Syllogisms (a form of logical deduction):

$$\frac{\text{All men are mortal} \qquad \text{Aristotle is a man}}{\text{Aristotle is mortal}}$$

Important branches of logic today

(1) Philosophical Logic

(2) Mathematical Logic

(3) Logic in Computer Science

Our course takes place mainly in (3) but we will heavily use tools from (2) (in particular the sub-field of proof theory) and motivation from (1).

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.

Minlog

`http://www.mathematik.uni-muenchen.de/$\sim$logik/minlog/`

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

The case studies on program extraction presented in this course have been carried out in Minlog.

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

Background Reading

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

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

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

[4] B, From coinductive proofs to exact real arithmetic: theory and applications, Logical Methods in Comput. Sci. 7(1), 2011, `http://www.lmcs-online.org/ojs/viewarticle.php?id=704&layout=abstract`

[5] H Schwichtenberg, S S Wainer, Proofs and Computations, Cambridge University Press, 2012.

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.

Truth, Validity, Logical Consequence

$\mathcal{M} \models A$ (formula A is *true* in model $\mathcal{M}$, or $\mathcal{M}$ *satisfies* A)

A is *logically valid* ($\models A$) $\stackrel{\text{Def}}{=}$ for *all* models $\mathcal{M}$, $\mathcal{M} \models A$.

(A is true in *all* models)

A is a *logical consequence* of a set of formulas Γ ($\Gamma \models A$) $\stackrel{\text{Def}}{=}$ for *all* models $\mathcal{M}$, if $\mathcal{M} \models \Gamma$, then $\mathcal{M} \models A$.

(A is true in *all* models of Γ, or Γ logically implies A)

Where $\mathcal{M} \models \Gamma$ means $\mathcal{M} \models B$ for all $B \in \Gamma$.

Exercises

Let $\mathcal{N}$ be the structure of natural numbers $\{0, 1, 2, \ldots\}$ with the usual constants 0 and 1 and the operations of addition and multiplication.

Let $A \stackrel{\text{Def}}{=} \exists z \, \forall x \forall y \, (x + y = z \rightarrow x = z)$

Exercises

Let $\mathcal{N}$ be the structure of natural numbers $\{0, 1, 2, \ldots\}$ with the usual constants 0 and 1 and the operations of addition and multiplication.

Let $A \stackrel{\text{Def}}{=} \exists z \, \forall x \forall y \, (x + y = z \rightarrow x = z)$

1. Is A true in $\mathcal{N}$?

Exercises

Let $\mathcal{N}$ be the structure of natural numbers $\{0, 1, 2, \ldots\}$ with the usual constants 0 and 1 and the operations of addition and multiplication.

Let $A \stackrel{\text{Def}}{=} \exists z \, \forall x \forall y \, (x + y = z \rightarrow x = z)$

1. Is A true in $\mathcal{N}$?

2. Is A logically valid?

Exercises

Let $\mathcal{N}$ be the structure of natural numbers $\{0, 1, 2, \ldots\}$ with the usual constants 0 and 1 and the operations of addition and multiplication.

Let $A \stackrel{\text{Def}}{=} \exists z \, \forall x \forall y \, (x + y = z \rightarrow x = z)$

1. Is A true in $\mathcal{N}$?

2. Is A logically valid?

Let $A' \stackrel{\text{Def}}{=} \exists z \, \forall x \, (\exists y \, (x + y = z) \rightarrow x = z)$

3. Is the formula $A \rightarrow A'$ logically valid?
(That is, does $\{A'\} \models A$ hold?)

Proofs

A *proof system* is a collection of rules to derive logically valid formulas.

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

Gerhard Gentzen (1909 - 1945)

Natural Deduction

Assumption rule u:A (assumptions are cancelled by $\to^+ u : A$)		
	Introduction rules	Elimination rules
$\wedge$	$\dfrac{A \quad B}{A \wedge B}\ \wedge^+$	$\dfrac{A \wedge B}{A}\ \wedge^-_{\mathsf{l}} \qquad \dfrac{A \wedge B}{B}\ \wedge^-_{\mathsf{r}}$
$\to$	$\dfrac{B}{A \to B}\ \to^+ u : A$	$\dfrac{A \to B \quad A}{B}\ \to^-$
$\vee$	$\dfrac{A}{A \vee B}\ \vee^+_{\mathsf{l}} \qquad \dfrac{B}{A \vee B}\ \vee^+_{\mathsf{r}}$	$\dfrac{A \vee B \quad A \to C \quad B \to C}{C}\ \vee^-$
$\bot$		$\dfrac{\bot}{A}\ \mathsf{efq} \qquad \dfrac{\neg\neg A}{A}\ \mathsf{raa}$

Quantifier rules

	Introduction rules	Elimination rules
$\forall$	$\dfrac{A(x)}{\forall x\, A(x)}\ \forall^+ \qquad (*)$	$\dfrac{\forall x\, A(x)}{A(t)}\ \forall^-$
$\exists$	$\dfrac{A(t)}{\exists x\, A(x)}\ \exists^+$	$\dfrac{\exists x\, A(x) \qquad \forall x\, (A(x) \to C)}{C}\ \exists^- \quad (**)$

Variable conditions:

(*) x must not occur free in any free (that is, uncancelled) assumption.

(**) x must not occur free in C.

Adding these rules to natural deduction yields a complete proof system.

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 (both versions)

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

	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 from these rules.

Example

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

Example

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

Exercise

Prove the *Law of Excluded Middle*: $A \vee \neg A$.

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.

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

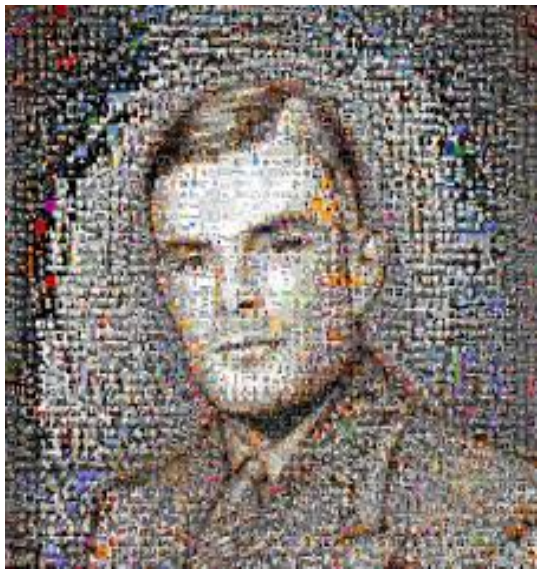
Kurt Gödel (1906-1978)

Undecidability

Church and Turing independently proved that it is undecidable whether a formula in predicate logic is valid, thus answering Hilbert's "*Entscheidungsproblem*" negatively.

David Hilbert
(1862 - 1943)

Alonzo Church
(1903 - 1995)



Alan Turing[(1)]
(1912 - 1954)

By Gödel's Completeness theorem there is at list an automatic procedure that generates, one by one, all valid formulas (since proofs can be automatically generated).

(1) picture computer generated by Arnold Beckmann

Peano Arithmetic

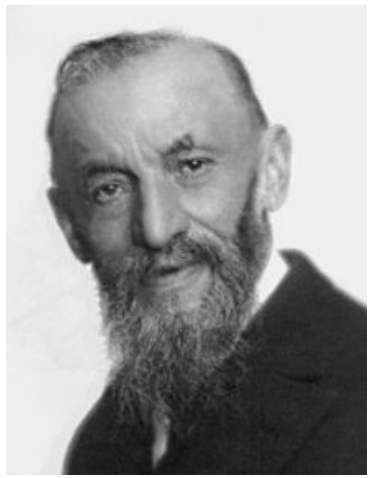
In order to prove statements that are true in the structure $\mathcal{N}$ of natural numbers, Peano introduced the following axioms:

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***)*.

Complexity of **PA** vs. truth in $\mathcal{N}$

Complexity of **PA** vs. truth in $\mathcal{N}$

All theorems of **PA** are true in $\mathcal{N}$ but not conversely.

Complexity of **PA** vs. truth in $\mathcal{N}$

All theorems of **PA** are true in $\mathcal{N}$ but not conversely.

By definition, the theorems of **PA** are effectively enumerable.

Complexity of **PA** vs. truth in $\mathcal{N}$

All theorems of **PA** are true in $\mathcal{N}$ but not conversely.

By definition, the theorems of **PA** are effectively enumerable.

In contrast, the set $\mathrm{Th}_{\mathcal{N}}$ of formulas that are true in $\mathcal{N}$ is not effectively enumerable.

Complexity of **PA** vs. truth in $\mathcal{N}$

All theorems of **PA** are true in $\mathcal{N}$ but not conversely.

By definition, the theorems of **PA** are effectively enumerable.

In contrast, the set $\mathrm{Th}_{\mathcal{N}}$ of formulas that are true in $\mathcal{N}$ is not effectively enumerable.

By Tarski's Undefinability Theorem $\mathrm{Th}_{\mathcal{N}}$ is not even definable by an arithmetic formula (Tarski).

Complexity of **PA** vs. truth in $\mathcal{N}$

All theorems of **PA** are true in $\mathcal{N}$ but not conversely.

By definition, the theorems of **PA** are effectively enumerable.

In contrast, the set $\mathrm{Th}_{\mathcal{N}}$ of formulas that are true in $\mathcal{N}$ is not effectively enumerable.

By Tarski's Undefinability Theorem $\mathrm{Th}_{\mathcal{N}}$ is not even definable by an arithmetic formula (Tarski).

Therefore, there is a huge gap between **PA** and $\mathrm{Th}_{\mathcal{N}}$.

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, as we have seen, 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, as we have seen, 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 ($\neg\neg A \rightarrow A$, 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, as we have seen, 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.

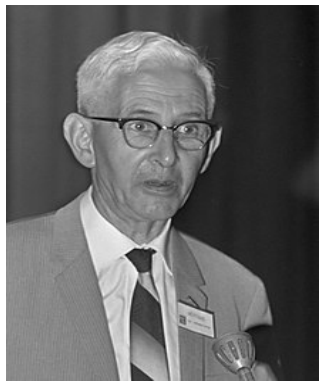
Corresponding theorems for classical logic do *not* hold. However, we have

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

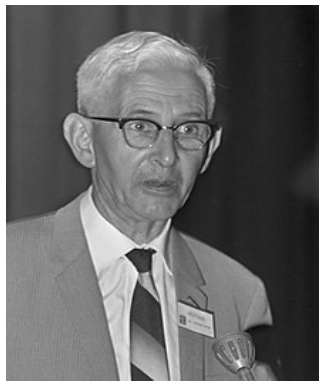
Peano Arithmetic with intuitionistic logic is called *Heyting Arithmetic*, **HA**.



Arendt Heyting (1898 - 1980)

Heyting Arithmetic

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



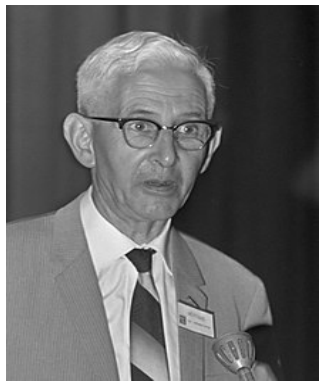
Arendt Heyting (1898 - 1980)

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**:

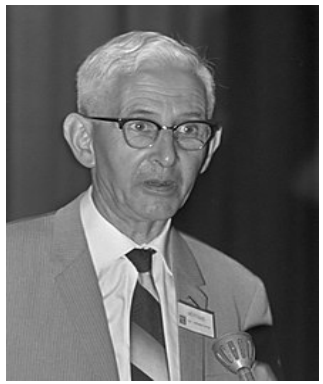
- The Disjunction and Existence Theorems continue to hold.
- The (universally generalized) law of excluded middle

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

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.

We study an interpretation 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 $(M\,N)\,K$.

Lambda calculus with types

Types are like propositional formulas with $\to$ as the only connective.

Let $\Gamma = x_1 : A_1, \ldots x_n, : A_n$ be a *context*, that is a type assignment to variables.

We define inductively the relation $\Gamma \vdash M : A$
(M has type A in context Γ).

$$\Gamma, x : A \vdash x : A$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x\, M : A \to B} \qquad \frac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash M\, N : B}$$

β-reduction and -equality

Theorem

β-reduction is *strongly normalizing*, that is, every reduction sequence $M_1 \to_\beta M_2 \to_\beta M_3, \ldots$ terminates.

Theorem

β-reduction is *confluent*, that is, if $M \to_\beta^* N_1$ and $M \to_\beta^* N_2$, then there exists a term N such that $N_1 \to_\beta^* N$ and $N_2 \to_\beta^* N$.

Theorem

The relation of β-equality, defined by

$$M =_\beta N \quad :\Leftrightarrow \quad \exists K \, (M \to_\beta^* K \wedge N \to_\beta^* K)$$

is decidable.

Extension to products and sums

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

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

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash (0, M) : A + B} \qquad \frac{\Gamma \vdash M : B}{\Gamma \vdash (1, M) : A + B}$$

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

$$\begin{aligned}
\pi_0(M, N) &\to_\beta M \\
\pi_1(M, N) &\to_\beta N \\
\mathbf{case}((0, M), N, K) &\to_\beta N\, M \\
\mathbf{case}((1, M), N, K) &\to_\beta K\, M
\end{aligned}$$

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

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

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

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

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

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

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

Proofs as Programs

- Realizability
- Strictly positive induction
- Realizability of induction
- Coinduction
- Classical axioms

Realizability

Realizability attaches meaning to the Curry-Howard correspondence.

Realizability

Realizability attaches meaning to the Curry-Howard correspondence.

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.

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).

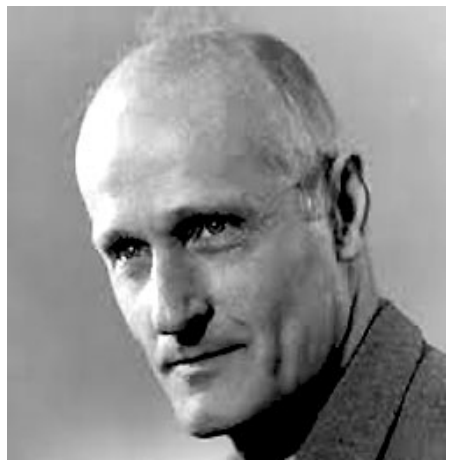
Realizability

Realizability attaches meaning to the Curry-Howard correspondence.

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, $e\,\mathbf{r}\,A$.

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

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:

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

Variants of realizability

- Modified realizability (Kreisel): Realizers are total functionals of finite types.
- Realizability with truth (**q**-realizability): Can be used to prove disjunction and existence property.
- Extensions to finite type arithmetic ($\mathbf{HA}^\omega$) and second-order arithmetic ($\mathbf{HAS}^\omega$).
- Partial Combinatory Algebra (PCA) as space of (potential) realizers (instead of natural numbers).
- Realizability for set theory (McCarty, Rathjen)
- Realizability topos (Hyland)
- Realizability with assemblies (Bauer)
- Realizability for decorated formulas (Schwichtenberg)

...

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 will try to generalize/improve this method of program extraction to

- 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 will try to generalize/improve this method of program extraction to

- abstract structures (instead of only natural numbers)
- 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 will try to generalize/improve this method of program extraction to

- abstract structures (instead of only natural numbers)
- stronger axioms (instead of only induction on natural numbers)
- 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 will try to generalize/improve this method of program extraction to

- abstract structures (instead of only natural numbers)
- stronger axioms (instead of only induction on natural numbers)
- limited classical logic and choice principles
- 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 will try to generalize/improve this method of program extraction to

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

In addition to uniform quantifiers, Minlog permits uniform versions of propositional connectives (decorated formulas).

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.

In addition to uniform quantifiers, Minlog permits uniform versions of propositional connectives (decorated formulas).

Kleene's interpretation of quantifiers can be recovered by relativization.

Simplifyng realizers

Simplifyng realizers

If A is disjunction free, then it has no computational content, that is, the potential realizers of A are trivial and $e\,\mathbf{r}\,A \equiv A$

Simplifyng realizers

If A is disjunction free, then it has no computational content, that is, the potential realizers of A are trivial and $e\,\mathbf{r}\,A \equiv A$

Therefore, in that case the realizability of $A \wedge B$ and $A \to B$ can be simplified:

Simplifyng realizers

If A is disjunction free, then it has no computational content, that is, the potential realizers of A are trivial and $e\,\mathbf{r}\,A \equiv A$

Therefore, in that case the realizability of $A \wedge B$ and $A \to B$ can be simplified:

Instead of

$$\begin{aligned} e\,\mathbf{r}\,(A \wedge B) &\equiv e = \mathbf{P}(a,b) \wedge a\,\mathbf{r}\,A \wedge b\,\mathbf{r}\,B \\ f\,\mathbf{r}\,(A \to B) &\equiv \forall a\,(a\,\mathbf{r}\,A \to f(a)\,\mathbf{r}\,B) \end{aligned}$$

Simplifyng realizers

If A is disjunction free, then it has no computational content, that is, the potential realizers of A are trivial and $e\,\mathbf{r}\,A \equiv A$

Therefore, in that case the realizability of $A \wedge B$ and $A \to B$ can be simplified:

Instead of

$$\begin{aligned} e\,\mathbf{r}\,(A \wedge B) &\equiv e = \mathbf{P}(a,b) \wedge a\,\mathbf{r}\,A \wedge b\,\mathbf{r}\,B \\ f\,\mathbf{r}\,(A \to B) &\equiv \forall a\,(a\,\mathbf{r}\,A \to f(a)\,\mathbf{r}\,B) \end{aligned}$$

we define

$$\begin{aligned} b\,\mathbf{r}\,(A \wedge B) &\equiv A \wedge b\,\mathbf{r}\,B \\ b\,\mathbf{r}\,(A \to B) &\equiv A \to b\,\mathbf{r}\,B \end{aligned}$$

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:

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

Formalizing induction

We add to the language of predicate logic *predicate variables* $X, Y, \ldots$.

Atomic formulas can now also be of the form $X(\vec{t})$.

If $A(X, \vec{x})$ is a formula where $\vec{x}$ is a tuple of individual variables and the predicate variable X occurs only *strictly positively*, that is, not on the left hand side of an implications, then we can form the predicate $\lambda\vec{x}\, A(X, \vec{x})$ and the monotone operator

$$\Phi \stackrel{\text{Def}}{=} \lambda X \lambda\vec{x}\, A(X, \vec{x})$$

and formally introduce the predicate $\mu(\Phi)$ with the intention that it denotes the least fixed point of Φ.

We express this intention by adding the derivation rules

$$\frac{}{\Phi(\mu(\Phi)) \subseteq \mu(\Phi)}\ \textbf{CI} \qquad \frac{\Phi(P) \subseteq P}{\mu(\Phi) \subseteq P}\ \textbf{Ind}$$

For every predicate P definable in the language.

Example: Natural numbers

Consider a disjunction-free first-order axiomatization of real closed fields (with the real numbers as intended model).

In particular we have the constants, functions and relation symbols $0, 1, +, *, |\cdot|, <, \ldots$ with the expected axioms.

Example: Natural numbers

Consider a disjunction-free first-order axiomatization of real closed fields (with the real numbers as intended model).

In particular we have the constants, functions and relation symbols $0, 1, +, *, |\cdot|, <, \ldots$ with the expected axioms.

We define the natural numbers as the least predicate that contains 0 and is closed under the $(+1)$ operation:

$$\mathbb{N} \stackrel{\text{Def}}{=} \mu(\Phi)$$

where $\Phi = \lambda X\, \lambda x\, (x = 0 \vee \exists y(X(y) \wedge x = y + 1))$

Example: Natural numbers

Consider a disjunction-free first-order axiomatization of real closed fields (with the real numbers as intended model).

In particular we have the constants, functions and relation symbols $0, 1, +, *, |\cdot|, <, \ldots$ with the expected axioms.

We define the natural numbers as the least predicate that contains 0 and is closed under the $(+1)$ operation:

$$\mathbb{N} \stackrel{\text{Def}}{=} \mu(\Phi)$$

where $\Phi = \lambda X\, \lambda x\, (x = 0 \vee \exists y (X(y) \wedge x = y + 1))$

Equivalently, $\Phi = \lambda X\, \lambda x\, (x = 0 \vee X(x - 1))$.

Example: Natural numbers

Consider a disjunction-free first-order axiomatization of real closed fields (with the real numbers as intended model).

In particular we have the constants, functions and relation symbols $0, 1, +, *, |\cdot|, <, \ldots$ with the expected axioms.

We define the natural numbers as the least predicate that contains 0 and is closed under the $(+1)$ operation:

$$\mathbb{N} \stackrel{\text{Def}}{=} \mu(\Phi)$$

where $\Phi = \lambda X\, \lambda x\, (x = 0 \vee \exists y(X(y) \wedge x = y + 1))$

Equivalently, $\Phi = \lambda X\, \lambda x\, (x = 0 \vee X(x - 1))$.

We use the following short notation for the definition of $\mathbb{N}$:

$$\mathbb{N}(x) \stackrel{\mu}{=} x = 0 \vee \mathbb{N}(x - 1)$$

Induction on $\mathbb{N}$

Recall that $\mathbb{N} = \mu(\Phi)$ where $\Phi = \lambda X\, \lambda x\, (x = 0 \vee X(x-1))$.

Let's work out the meaning of the rules

$$\frac{}{\Phi(\mathbb{N}) \subseteq \mathbb{N}}\ \textbf{CI} \qquad \frac{\Phi(P) \subseteq P}{\mathbb{N} \subseteq P}\ \textbf{Ind}$$

$\Phi(\mathbb{N}) \subseteq \mathbb{N}$ means $\forall x\, (\mathbb{N}(x) \to x = 0 \vee \mathbb{N}(x-1))$ which is equivalent to

$$\mathbb{N}(0) \wedge \forall x\, (\mathbb{N}(x) \to \mathbb{N}(x+1))$$

$\Phi(P) \subseteq P$ means $\forall x\, (x = 0 \vee P(x-1) \to P(x))$ which is equivalent to

$$P(0) \wedge \forall x\, (P(x) \to P(x+1))$$

Therefore, the induction rule is nothing but the familiar

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

Realizability of inductive definitions

$$\mathbf{R}(\mu(\Phi)) \stackrel{\text{Def}}{=} \mu(\mathbf{R}(\Phi))$$

where for $\Phi = \lambda X \lambda \vec{x}\, A(X, \vec{x})$ we introduce a new predicate variable $\tilde{X}$ with one argument place for realizers and set

$$\begin{aligned} a\,\mathbf{r}\,X(\vec{t}) &\stackrel{\text{Def}}{=} \tilde{X}(\vec{t}, a) \\ \mathbf{R}(\Phi) &\stackrel{\text{Def}}{=} \lambda \tilde{X} \lambda(\vec{x}, a)\, a\,\mathbf{r}\,A(X, \vec{x}) \end{aligned}$$

Realizability of inductive definitions

$$\mathbf{R}(\mu(\Phi)) \stackrel{\text{Def}}{=} \mu(\mathbf{R}(\Phi))$$

where for $\Phi = \lambda X \lambda \vec{x}\, A(X, \vec{x})$ we introduce a new predicate variable $\tilde{X}$ with one argument place for realizers and set

$$\begin{aligned} a\,\mathbf{r}\,X(\vec{t}) &\stackrel{\text{Def}}{=} \tilde{X}(\vec{t}, a) \\ \mathbf{R}(\Phi) &\stackrel{\text{Def}}{=} \lambda \tilde{X} \lambda(\vec{x}, a)\, a\,\mathbf{r}\, A(X, \vec{x}) \end{aligned}$$

For $\mathbb{N}$ this works out as

$$a\,\mathbf{r}\,\mathbb{N}(x) \stackrel{\mu}{=} (a = (0, \mathbf{Nil}) \wedge x = 0) \vee (a = (1, b) \wedge b\,\mathbf{r}\,\mathbb{N}(x - 1))$$

If we define numerals as $\underline{0} = (0, \mathbf{Nil})$, $\underline{n+1} = (1, \underline{n})$, we see that $a\,\mathbf{r}\,x$ iff $x \in \mathbb{N}$ and a is the numeral representing x.

Realizability of closure and induction

$$\Phi = \lambda X \lambda \vec{x}\, A(X, \vec{x})$$

Realizability of closure and induction

$\Phi = \lambda X \lambda \vec{x}\, A(X, \vec{x})$

It is easy to see that $(\lambda a\, a)\, \mathbf{r}\, (\Phi(\mu(\Phi)) \subseteq \mu(\Phi))$.

Realizability of closure and induction

$\Phi = \lambda X \lambda \vec{x}\, A(X, \vec{x})$

It is easy to see that $(\lambda a\, a)\, \mathbf{r}\, (\Phi(\mu(\Phi)) \subseteq \mu(\Phi))$.

$$\frac{s\, \mathbf{r}\, (\Phi(P) \subseteq P)}{g\, \mathbf{r}\, (\mu(\Phi) \subseteq P)}$$

where g is defined recursively as

$$g = s \circ (\mathbf{map}\, g) \qquad (\circ \text{ means composition})$$

and **map** is a term such that $\mathbf{map}\, \mathbf{r}\, ((X \subseteq Y) \to \Phi(X) \subseteq \Phi(Y))$.

map can be constructed by induction on A using the fact that X may only occur strictly positive in A.

Realizability of induction on natural numbers

$g = s \circ (\mathbf{map}\, g)$, that is, $g\, a = s\, (\mathbf{map}\, g\, a)$

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

$\mathbf{map}\, s\, (0, \mathbf{Nil}) = (0, \mathbf{Nil})$, $\mathbf{map}\, s\, (1, a) = (1, s\, a)$.

$a\, \mathbf{r}\, P(0)$, $f\, \mathbf{r}\, (\forall x\, (P(x) \to P(x + 1))$.

$s\, (0, \mathbf{Nil}) = a$, $s\, (1, a) = f\, a$.

Therefore,

$$\begin{aligned} g\, (0, \mathbf{Nil}) &= s\, (0, \mathbf{Nil}) = a \\ g\, (1, a) &= s\, (1, g\, a) = f\, (g\, a) \end{aligned}$$

That is, $g = \mathbf{It}(a, f)$.

Coinduction

Instead of *least* fixed points, $\mu(\Phi)$, one can also consider *greatest* fixed points, $\nu(\Phi)$.

$$\frac{}{\nu(\Phi) \to \Phi(\nu(\Phi))}\ \textbf{Cocl} \qquad \frac{P \subseteq \Phi(P)}{P \subseteq \nu(\Phi)}\ \textbf{Coind}$$

$$\frac{s\,\mathbf{r}\,(P \subseteq \Phi(P))}{g\,\mathbf{r}\,(P \subseteq \mu(\Phi))}$$

where g is defined recursively as

$$g = (\mathbf{map}\, g) \circ s$$

Soundness for IFP

We call our system of intuitionistic predicate logic with inductive and coinductive definitions IFP (Intuitionistic Fixed Point Logic).

Soundness for IFP

We call our system of intuitionistic predicate logic with inductive and coinductive definitions IFP (Intuitionistic Fixed Point Logic).

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

Soundness for IFP

We call our system of intuitionistic predicate logic with inductive and coinductive definitions IFP (Intuitionistic Fixed Point Logic).

RIFP is 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 the 'extra axioms' Γ are disjunction free, then $\Gamma \vdash_{\mathrm{RIFP}} M\,\mathbf{r}\,A$ for some program M.

Example: Addition on $\mathbb{N}$

Example: Addition on $\mathbb{N}$

Theorem $\mathbb{N}(x) \wedge \mathbb{N}(y) \rightarrow \mathbb{N}(x + y)$

Example: Addition on $\mathbb{N}$

Theorem $\mathbb{N}(x) \wedge \mathbb{N}(y) \to \mathbb{N}(x + y)$

The extracted program implements addition on natural numbers in unary representation.

Example: Addition on $\mathbb{N}$

Theorem $\mathbb{N}(x) \land \mathbb{N}(y) \to \mathbb{N}(x + y)$

The extracted program implements addition on natural numbers in unary representation.

This implementation satisfies associativity and commutativity, since $+$ has these properties (by the axioms) and elements of $\mathbb{N}$ and their unary representations are in a one-to-one correspondence.

Example: Cauchy reals

Let $\mathbb{Q}(x)$ mean that x is a rational number (definable using $\mathbb{N}$).

$$\mathbf{A}(x) \stackrel{\nu}{=} \exists q \in \mathbb{Q}\, |x - q| \leq 1 \wedge \mathbf{A}(2x)$$

Then $a \,\mathbf{r}\, \mathbf{A}(x)$ iff a is an infinite stream of rational numbers,

$$a = (q_0, (q_1, (q_2 \ldots$$

such that $|x - q_n| \leq 2^n$ for all n.

Example: Continuous real functions

Let $\mathbb{I} \stackrel{\text{Def}}{=} [-1,1]$, $\mathrm{SD} \stackrel{\text{Def}}{=} \{-1,0,1\}$.

For $i \in \mathrm{SD}$ define $\mathrm{av}_i : \mathbb{I}^{\mathbb{I}}$ by $\mathrm{av}_i(x) = (x+i)/2$.

Let F, G range over subsets of $\mathbb{I}^{\mathbb{I}}$. Define $\mathbf{C}_1 \stackrel{\text{Def}}{=}$

$$\nu(\lambda F\, \mu(\lambda G\, \{g \in \mathbb{I}^{\mathbb{I}} \mid (\exists i \in \mathrm{SD}\, \exists f \in \mathbb{I}^{\mathbb{I}}\, (g = \mathrm{av}_i \circ f \wedge F(f))) \vee \forall j \in \mathrm{SD}\, G(g \circ \mathrm{av}_j)\}$$

Theorem $\mathbf{C}_1(f)$ iff f is uniformly continuous.

The extracted program translates between the usual constructive representation of uniformly continuous functions (encoded in a suitable definition of the predicate "f is uniformly continuous") and a tree representation that is completely free of any function component (similar to the stream representation of reals).

Proof theory of inductive/coinductive definitions

Inductive definitions add considerable strength to a theory.

Our full system, with classical logic, has the proof-theoretic strength of Π^1_2-comprehension as shown by Möllerfeld (2003)

Tupailo (2004) showed that the corresponding intuitionistic system has the same strength.

Restricting to unnested (but iterated) inductive definitions one obtains the system $\mathrm{ID}^{<\omega}$ (Buchholz, Feferman, Pohlers, Sieg, 1981).

Classical axioms

Classical axioms

According to the Soundness Theorem for IFP one may admit classical disjunction-free axioms if one is satisfied that the correctness of the extracted program depends on the truth of these axioms.

Classical axioms

According to the Soundness Theorem for IFP one may admit classical disjunction-free axioms if one is satisfied that the correctness of the extracted program depends on the truth of these axioms.

Hence we may for example use stability of equality

$$\forall x, y\,(\neg\neg x = y \to x = y)$$

for real numbers

Classical axioms

According to the Soundness Theorem for IFP one may admit classical disjunction-free axioms if one is satisfied that the correctness of the extracted program depends on the truth of these axioms.

Hence we may for example use stability of equality

$$\forall x, y\,(\neg\neg x = y \to x = y)$$

for real numbers

Very useful are *Skolem axioms* (Avigad 2002) that introduce for every disjunction-free formula $A(\vec{x}, y)$ a new function symbol f and the axiom

$$\forall \vec{x} \forall y\,(A(\vec{x}, y) \to A(\vec{x}, f(\vec{x}))$$

Classical axioms

According to the Soundness Theorem for IFP one may admit classical disjunction-free axioms if one is satisfied that the correctness of the extracted program depends on the truth of these axioms.

Hence we may for example use stability of equality

$$\forall x, y\,(\neg\neg x = y \to x = y)$$

for real numbers

Very useful are *Skolem axioms* (Avigad 2002) that introduce for every disjunction-free formula $A(\vec{x}, y)$ a new function symbol f and the axiom

$$\forall \vec{x} \forall y\,(A(\vec{x}, y) \to A(\vec{x}, f(\vec{x}))$$

For example, division can be introduced by the Skolem axiom

$$\forall x, z \forall y\,(x * y = z \to x * (z/x) = z)$$

(the expected formula, $\forall x, z, y(x \neq 0 \to (x * y = z \leftrightarrow y = z/x))$, follows).

Summary of today's lecture

We introduced realizability for IFP, which is intuitionistic predicate logic with inductive and coinductive definitions.

Abstract mathematics is included by realizing quantifiers uniformly.

Induction and coinduction are completely dual and are realized by general recursion.

Classically true disjuction free axioms can be added.

Program extraction

This lecture presents case studies in program extraction

It is given jointly by members of the Minlog group:

Nils Köpp

Fredrik Nordvall-Forsberg

Helmut Schwichtenberg

Franziskus Wiesnet

B

Overview of existing case studies in program extraction

Overview of existing case studies in program extraction

- Discrete structures
  - Quotient and remainder on natural numbers.
    Presentation by B
  - 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 existing case studies in program extraction

- Discrete structures
  - Quotient and remainder on natural numbers.
    Presentation by B
  - 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

Overview ctd.

- 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)

Overview ctd.

- 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):
    Cauchy sequences are functions.
    SD representations are streams defined by coinduction.
  - Arithmetic operations on reals w.r.t. SD
    Presentation by Franziskus Wiesnet
  - Integration w.r.t. SD (2011, B):
    Real functions are given by trees realizing a nested coinductive/inductive definition

Overview ctd.

- 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

Overview ctd.

- 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)
    Presentation by Fredrik Forsberg

Overview ctd.

- 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)
    Presentation by Fredrik Forsberg

- Ongoing: Extraction of
  - monadic parsers (Jones, Seisenberger, B)
  - concurrent programs (Miyamoto, Petrovska, Schwichtenberg, Spreen, Takayama, Tsuiki, B)
  - truly imperative programs (Reus, B)
  - modulus of uniform continuity from Fan Theorem (B)