JAIST Spring School 2012
Formal Reasoning: Theory and Application
Kanazawa, 5-9 March

Program Extraction

Ulrich Berger
Swansea

Overview

I Extracting concrete programs from abstract mathematics

II Program extraction in analysis

III Projects

IV Programs from classical proofs

I Extracting concrete programs from abstract mathematics

Some reasons for formalizing mathematics

1. Verify mathematics
2. Solve mathematical problems
3. Provide a new foundation for mathematics
4. Verify programs
5. Extract verified programs

Some reasons for formalizing mathematics

1. Verify mathematics
2. Solve mathematical problems
3. Provide a new foundation for mathematics
4. Verify programs
5. Extract verified programs

1-4 require big systems.

Some reasons for formalizing mathematics

1. Verify mathematics
2. Solve mathematical problems
3. Provide a new foundation for mathematics
4. Verify programs
5. Extract verified programs

1-4 require big systems.

For 5 one can get away with a surprisingly light-weight system.

Why formalization for program extraction is easy

- We can import a lot of trusted mathematics.
- Only those parts of a proof that carry computationally relevant information need to be formalized. These parts are often rather small.
- Data structures and programs are not part of the proof system - they are generated by the extraction process.
- Extracted programs don't need to be read, understood, or maintained. Therefore, they can be rather low level. It is enough to understand proofs.

The Curry-Howard Correspondence

Disjunction

Proof of formula	Term and its type
$\dfrac{A}{A \vee B}\ \vee^{+}_{\mathrm{L}} \qquad \dfrac{B}{A \vee B}\ \vee^{+}_{\mathrm{R}}$	$\dfrac{M : A}{\mathrm{L}(M) : A + B} \qquad \dfrac{M : B}{\mathrm{R}(M) : A + B}$
$\dfrac{A \vee B \quad A \vdash C \quad B \vdash C}{C}\ \vee^{-}$	$\dfrac{K : A + B \quad a : A \vdash N_1 : C \quad b : B \vdash N_2 : C}{\mathrm{case}\, K\, \mathrm{of}\{\, \mathrm{L}(a) \to N_1 \,;\, \mathrm{R}(b) \to N_2 \,\} : C}$

Computation: $\mathrm{case}\, \mathrm{L}(M)\, \mathrm{of}\{\, \mathrm{L}(a) \to N_1 \,;\, \mathrm{R}(b) \to N_2 \,\} = N_1[M/a]$

$\mathrm{case}\, \mathrm{R}(M)\, \mathrm{of}\{\, \mathrm{L}(a) \to N_1 \,;\, \mathrm{R}(b) \to N_2 \,\} = N_2[M/b]$

Curry-Howard ctd.

Implication

$\dfrac{A \vdash B}{A \to B} \to^{+}$	$\dfrac{a : A \vdash M : B}{\lambda a.M : A \to B}$
$\dfrac{A \to B \qquad A}{B} \to^{-}$	$\dfrac{M : A \to B \qquad N : A}{MN : B}$

Computation: $(\lambda a.M)N = M[N/a]$

Curry-Howard Isomorphism?

For minimal propositional logic the Curry-Howard correspondence is an isomorphism:

- Programs (i.e. typed λ-terms) can be understood as term notations for proofs.
- Computation corresponds to proof normalization or cut-elimination.

Curry-Howard Isomorphism?

For minimal propositional logic the Curry-Howard correspondence is an isomorphism:

- Programs (i.e. typed λ-terms) can be understood as term notations for proofs.
- Computation corresponds to proof normalization or cut-elimination.

- What about full (classical) predicate logic?
- What about "real" programs?

Example: Real numbers $(\mathbb{R}, 0, 1, +, *, <)$

Axioms

$$
\begin{aligned}
& x + 0 = x \\
& x + y = y + x \\
& \ldots \\
& x \neq 0 \to \exists y\,(x * y = 1) \\
& x > 0 \to \exists y\,(x = y * y) \\
& \ldots
\end{aligned}
$$

Real numbers are viewed as an abstract structure described by axioms.

For the purpose of program extraction any "true" disjunction-free first-order formula in the language $0, 1, +, *, <$ is allowed as axiom.

Natural numbers $\mathbb{N} = \{0, 1, 2, \ldots\} \subseteq \mathbb{R}$

The natural numbers are inductively defined by

$$\mathbb{N} \stackrel{\mu}{=} \{x \mid x = 0 \vee \exists y\, (y \in \mathbb{N} \wedge x = y + 1)\}$$

This means that $\mathbb{N}$ is the least subset of $\mathbb{R}$ satisfying the equation.

The datatype corresponding to $\mathbb{N}$ is defined recursively:

$$\mathrm{Nat} = \mathrm{N}(1 + \mathrm{Nat})$$

where 1 is the unit type containing only one element, say Nil, and N is a constructor.

Proof rules and computations for natural numbers

Closure and Induction

$\dfrac{t = 0 \vee \exists y\,(y \in \mathbb{N} \wedge t = y + 1)}{t \in \mathbb{N}}$ Clos	$\dfrac{M : 1 + \mathrm{Nat}}{\mathrm{N}M : \mathrm{Nat}}$
$\dfrac{x = 0 \vee \exists y\,(A(y) \wedge x = y + 1) \vdash A(x)}{x \in \mathbb{N} \vdash A(x)}$ Ind	$\dfrac{b : 1 + A \vdash M : A}{n : \mathrm{Nat} \vdash \mathrm{It}[\lambda b.M, n] : A}$

Computation: $\mathrm{Zero} :\equiv \mathrm{N}(\mathrm{L}(\mathrm{Nil}))$, $\mathrm{Succ}(n) :\equiv \mathrm{N}(\mathrm{R}(n))$.

$$\begin{aligned} \mathrm{It}[\lambda b.M, \mathrm{Zero}] &= M[\mathrm{L}(\mathrm{Nil})/b] \\ \mathrm{It}[\lambda b.M, \mathrm{Succ}(n) &= M[\mathrm{R}(\mathrm{It}[\lambda b.M, n])/b] \end{aligned}$$

Extracting natural numbers from proofs

Theorem 0. $0 \in \mathbb{N}$.

Proof.

$$\dfrac{\dfrac{\dfrac{}{0 = 0}\,\text{Axiom}}{0 = 0 \vee \exists y\,(y \in \mathbb{N} \wedge 0 = y + 1)}\,\vee^{+}_{\mathrm{L}}}{0 \in \mathbb{N}}\,\text{Clos}$$

Extracted Program: $\mathrm{N}(\mathrm{L}(\mathrm{Nil})) \equiv \mathrm{Zero}$.

Theorem 1. $x \in \mathbb{N} \vdash x + 1 \in \mathbb{N}$.

Proof.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{x \in \mathbb{N} \qquad \dfrac{}{x+1 = x+1}\,\text{Axiom}}{x \in \mathbb{N} \wedge x+1 = x+1}\,\wedge^{+}}{\exists y\,(y \in \mathbb{N} \wedge x+1 = y+1)}\,\exists^{+}}{x+1 = 0 \vee \exists y\,(y \in \mathbb{N} \wedge x+1 = y+1)}\,\vee^{+}_{\mathrm{R}}}{x+1 \in \mathbb{N}}\,\text{Clos}$$

Extracted Program: $a : \mathrm{Nat} \vdash \mathrm{N}(\mathrm{R}(a))$ i.e. $a : \mathrm{Nat} \vdash \mathrm{Succ}(a)$.

Theorem 2. $2 \in \mathbb{N}$

Proof.

$$\dfrac{\dfrac{\dfrac{}{0 \in \mathbb{N}}\ \text{Theorem 0}}{1 \in \mathbb{N}}\ \text{Theorem 1}}{2 \in \mathbb{N}}\ \text{Theorem 1}$$

Extracted Program: $\mathrm{Succ}(\mathrm{Succ}(\mathit{Zero}))$.

Induction on natural numbers

Theorem 3.

$$\frac{A(0) \qquad A(x) \vdash A(x+1)}{x \in \mathbb{N} \vdash A(x)}\ \text{Ind-N},\ x$$

Proof.

$$\dfrac{\dfrac{\dfrac{A(0)}{z = 0 \vdash A(z)}\ \text{Eq} \qquad \dfrac{A(x) \vdash A(x+1)}{\exists x\,(A(x) \wedge z = x+1) \vdash A(z)}\ \exists^{-}}{z = 0 \vee \exists x\,(A(x) \wedge z = x+1) \vdash A(z)}\ \vee^{-}}{x \in \mathbb{N} \vdash A(x)}\ \text{Ind}$$

Recursion on unary notation

Extracted Program:

$$\frac{M_0 : A \qquad a : A \vdash M : A}{n : \mathrm{Nat} \vdash \mathrm{ItN}[a_0, \lambda a.M, n] : A}$$

where

$$\mathrm{ItN}[M_0, \lambda a.M, n] :\equiv \mathrm{It}[\lambda b.\mathrm{case}\, b\, \mathrm{of}\{\, \mathrm{L} \to M_0\, ;\, \mathrm{R}(a) \to M_1\, \}, n].$$

Computation:

$$\begin{aligned} \mathrm{ItN}[M_0, \lambda a.M_1, \mathrm{Zero}] &= M_0 \\ \mathrm{ItN}[M_0, \lambda a.M_1, \mathrm{Succ}(n)] &= M_1[\mathrm{ItN}[M_0, \lambda a.M_1, n]/a] \end{aligned}$$

Theorem 4. $x \in \mathbb{N}, y \in \mathbb{N} \vdash x + y \in \mathbb{N}$.

Proof.

$$\dfrac{\dfrac{x \in \mathbb{N}}{x + 0 \in \mathbb{N}}\ \text{Ax,Eq} \qquad \dfrac{}{x + y \in \mathbb{N} \vdash x + y + 1 \in \mathbb{N}}\ \text{Theorem 1}}{y \in \mathbb{N} \vdash x + y \in \mathbb{N}}\ \text{Ind-N, } y$$

Extracted Program:

$$a : \mathrm{Nat}, b : \mathrm{Nat} \vdash \mathrm{ItN}[a, \mathrm{Succ}, b]$$

Theory: Formulas

$P(\vec{t})$	P predicate constant, $\vec{t}$ first-order terms, (examples $=$, $<$, $\bot$)
$X(\vec{t})$	X predicate variable (example $\mathbb{N}$; we wrote $x \in \mathbb{N}$ for $\mathbb{N}(x)$)
$A \circ B$	$\circ \in \{\wedge, \vee, \to\}$
$\exists x\, A, \forall x\, A$	
$\mathrm{let}\ X \stackrel{*}{=} \mathcal{P}\ \mathrm{in}\ B$	$* \in \{\mu, \nu\}$, $\mathcal{P} \equiv \{\vec{x} \mid A\}$, A positive in X (inductive and coinductive definitions)

$\mathcal{P} \equiv \{\vec{x} \mid A\}$ is called a comprehension term.
We will use the notation $\mathcal{P}(\vec{t})$ for $A[\vec{t}/\vec{x}]$.

Proofs

- Natural deduction rules of intuitionistic first-order logic.
- Arbitrary (true) axioms and rules such that the formulas involved ar *non-computational (nc)*, i.e. contain neither disjunctions nor predicate variables (for example the rules for equality).
- The let-rule:

$$\frac{X \overset{\alpha}{=} \mathcal{P} \vdash B}{\text{let } X \overset{*}{=} \mathcal{P} \text{ in } B}\ \text{let}$$

- Induction and coinduction (next slides).

Induction and coinduction

$$\frac{X \stackrel{\mu}{=} \mathcal{P} \vdash \mathcal{P}(\vec{t})}{X \stackrel{\mu}{=} \mathcal{P} \vdash X(\vec{t})}\ \text{Closure}$$

$$\frac{X \stackrel{\mu}{=} \mathcal{P}, \mathcal{P}[\mathcal{Q}/X](\vec{x}) \vdash \mathcal{Q}(\vec{x})}{X \stackrel{\mu}{=} \mathcal{P}, X(\vec{x}) \vdash \mathcal{Q}(\vec{x})}\ \text{Induction}$$

$$\frac{X \stackrel{\nu}{=} \mathcal{P} \vdash X(\vec{t})}{X \stackrel{\nu}{=} \mathcal{P} \vdash \mathcal{P}(\vec{t})}\ \text{Coclosure}$$

$$\frac{X \stackrel{\nu}{=} \mathcal{P}, \mathcal{Q}(\vec{x}) \vdash \mathcal{P}[\mathcal{Q}/X](\vec{x})}{X \stackrel{\nu}{=} \mathcal{P}, \mathcal{Q}(\vec{x}) \vdash X(\vec{x})}\ \text{Coinduction}$$

Types

1	unit type
α	type variables (example Nat)
$\rho \circ \sigma$	$\circ \in \{\times, +, \to\}$
$\mathrm{let}\ \alpha = \rho\ \mathrm{in}\ \sigma$	where free occurrences of α in ρ and σ become bound (recursive types)

The type of a formula

We assign to every predicate symbol X a new type variable α_X.
First some easy special cases:

$$\begin{array}{lcll}
\tau(A) & \equiv & 1 & \text{if } A \text{ is nc} \\
\tau(A \wedge B) & \equiv & \tau(B) & \text{if } A \text{ is nc, but } B \text{ isn't} \\
\tau(A \wedge B) & \equiv & \tau(A) & \text{if } B \text{ is nc, but } A \text{ isn't} \\
\tau(A \to B) & \equiv & \tau(B) & \text{if } A \text{ is nc, but } B \text{ isn't}
\end{array}$$

Now we assume that none of the shown formulas is nc:

$$\begin{array}{lcl}
\tau(X(\vec{t})) & \equiv & \alpha_X \\
\tau(A \wedge B) & \equiv & \tau(A) \times \tau(B) \\
\tau(A \vee B) & \equiv & \tau(A) + \tau(B) \\
\tau(A \to B) & \equiv & \tau(A) \to \tau(B) \\
\tau(\forall x\, A) & \equiv & \tau(A) \\
\tau(\exists x\, A) & \equiv & \tau(A) \\
\tau(\text{let } X \stackrel{*}{=} \{\vec{x} \mid A\} \text{ in } B)) & \equiv & \text{let } \alpha_X = \tau(A) \text{ in } \tau(B)
\end{array}$$

Programs

a, b, c	variables
$C(M_1, \ldots, M_n)$	constructor terms, $C \in \{1, \mathrm{L}, \mathrm{R}, \mathrm{Pair}, \mathrm{In}\}$
$\mathrm{case}\, M\, \mathrm{of} \{ C_1(\vec{a_1}) \to R_1 \,;\, \ldots \,;\, C_n(\vec{a_n}) \to R_n \}$	case analysis/ pattern matching
$\lambda a.M$	lambda-abstraction
$M\, N$	application
$\mathrm{rec}\, a\, .\, M$	recursion

Semantics

The calculus can be axiomatized by

$$\begin{aligned} \mathrm{case}\, C_i(\vec{K})\, \mathrm{of}\{C_1(\vec{x_1}) \to R_1\,;\,\ldots\} &= R_i[\vec{K}/\vec{a_i}] \\ (\lambda a.M)N &= M[N/a] \\ \mathrm{rec}\, a\,.\,M &= M[\mathrm{rec}\, a\,.\,M/a] \end{aligned}$$

and given an operational semantics that is adequate w.r.t. a domain-theoretic denotational semantics:

Adequacy Theorem [Seisenberger, B] If $[\![M]\!] = d$, where d is domain element built from constructors only, then $M \Longrightarrow d$.

Note that programs and their semantics are type free.

Typing (selection of rules)

$$\frac{M : \rho \qquad N : \sigma}{\mathrm{Pair}(M, N) : \rho \times \sigma}$$

$$\frac{M : \rho \times \sigma \qquad x_1 : \rho, x_2 : \sigma \vdash N : \tau}{\mathrm{case}\, M\, \mathrm{of}\{\mathrm{Pair}(x_1, x_2) \to N\} : \tau}$$

$$\frac{M : \rho}{\mathrm{L}(M) : \rho + \sigma} \qquad \frac{M : \sigma}{\mathrm{R}(M) : \rho + \sigma}$$

$$\frac{M : \rho + \sigma \qquad x_1 : \rho \vdash M_1 : \tau \qquad x_2 : \sigma \vdash M_2 : \tau}{\mathrm{case}\, M\, \mathrm{of}\{\mathrm{L}(x_1) \to M_1\,; \mathrm{R}(x_2) \to M_2\} : \tau}$$

Typing (ctd.)

$$\frac{\alpha = \rho \vdash M : \sigma}{M : \text{let } \alpha = \rho \text{ in } \sigma}$$

$$\frac{\alpha = \rho \vdash M : \rho}{\alpha = \rho \vdash \text{In}_\rho(M) : \alpha}$$

$$\frac{\alpha = \rho \vdash M : \alpha \qquad \alpha = \rho, x : \rho \vdash N : \sigma}{\alpha = \rho \vdash \text{case } M \text{ of} \{\text{In}_\rho(x) \to N\} : \tau}$$

Program extraction

- We ignore the first-order part, i.p. quantifier rules.
- The extracted program is determined completely by the propositional rules and is extracted as expected.

Correctness

- The extracted programs represent intuitively the computational content of proofs.
- What does this mean exactly?
- How can we prove that ignoring the first-order part doesn't compromise correctness?

Type-theoretic answer (Coq): One has a reduction procedure for proofs and shows that each reduction step of a program can be traced by one or more reduction steps of the proofs.

Logical answer (Kleene, Kreisel, Gödel): One defines when a program "realizes" a formula (by recursion on formulas) and show that every extracted program realizes the proven formula (by recursion on terms).

Realizability

We assign to every predicate variable X a new predicate variable $\tilde{X}$ which has one extra argument place for realizers.

$$
\begin{array}{lcll}
a\,\mathbf{r}\,A & \equiv & A \wedge M = \mathrm{Nil} & \text{if } A \text{ is nc} \\
a\,\mathbf{r}\,(A \wedge B) & \equiv & A \wedge (a\,\mathbf{r}\,B) & \text{if } A \text{ is nc, but } B \text{ isn't} \\
a\,\mathbf{r}\,(A \wedge B) & \equiv & (a\,\mathbf{r}\,A) \wedge B & \text{if } B \text{ is nc, but } A \text{ isn't} \\
a\,\mathbf{r}\,(A \to B) & \equiv & A \to (a\,\mathbf{r}\,B) & \text{if } A \text{ is nc, but } B \text{ isn't}
\end{array}
$$

Now we assume that none of the shown formulas is nc:

$$
\begin{array}{lcl}
a\,\mathbf{r}\,X(\vec{t}) & \equiv & \tilde{X}(a,\vec{t}) \\
c\,\mathbf{r}\,(A \wedge B) & \equiv & \exists a,b\,(c = \mathrm{Pair}(a,b) \wedge a\,\mathbf{r}\,A \wedge b\,\mathbf{r}\,B) \\
c\,\mathbf{r}\,(A \vee B) & \equiv & \exists a\,(c = \mathrm{L}(a) \wedge a\,\mathbf{r}\,A) \vee \exists b\,(c = \mathrm{R}(b) \wedge b\,\mathbf{r}\,B) \\
c\,\mathbf{r}\,(A \to B) & \equiv & \exists f\,(c = \mathrm{abst}(f) \wedge \forall a\,(a\,\mathbf{r}\,A \to f(a)\,\mathbf{r}\,B)) \\
a\,\mathbf{r}\,\forall x\,A & \equiv & \forall x\,(a\,\mathbf{r}\,A) \\
a\,\mathbf{r}\,\exists x\,A & \equiv & \exists x\,(a\,\mathbf{r}\,A)
\end{array}
$$

$$
b\,\mathbf{r}\,(\mathrm{let}\ X \stackrel{*}{=} \{\vec{x} \mid A\}\ \mathrm{in}\ B) \equiv \mathrm{let}\ \tilde{X} \stackrel{*}{=} \{(a,\vec{x}) \mid a\,\mathbf{r}\,A\}\ \mathrm{in}\ b\,\mathbf{r}\,B
$$

Soundness

Soundness Theorem.

From a proof of a formula A using assumptions $B_1, \ldots, B_n$, one can extract a program term M, possibly containing variables $b_1, \ldots, b_n$, such that $M\,\mathbf{r}\,(A)$ is provable from the assumptions $b_1\,\mathbf{r}\,B_1, \ldots, \mathbf{r}(b_n)B_n$.

Furthermore one can prove $M\colon \tau(A)$, where $\tau(A)$ is the type naturally assigned to the formula A.

The proof yields in fact more: To every sub-program M' of M a formula A' is assigned and a proof of $M'\,\mathbf{r}\,A'$.

This means that one has specifications and correctness proofs of all sub-programs.

Proof of the Soundness Theorem

The proof is (as usual) by induction on proofs and in general rather straightforward, except for induction and coinduction.

The first soundness proof for coinductive definitions appears to be due to Makoto Tatsuta:

> M. Tatsuta, Realizability of Monotone Coinductive Definitions and Its Application to Program Synthesis. Proc. MPC, LNCS 1422, 338–364, 1998

Realizing natural numbers

One can easily see that for every natural number $x \in \mathbb{N}$:

$$a \mathbf{r} x \;\Leftrightarrow\; a \text{ is the unary notation of } x$$

Hence for our extracted program

$$a : \mathrm{Nat}, b : \mathrm{Nat} \vdash \mathrm{ItN}[a, \mathrm{Succ}, b]$$

the Soundness Theorem says:

If a, b are the unary notations of $x, y \in \mathbb{N}$,

then $\mathrm{ItN}[a, \mathrm{Succ}, b]$ is the unary notation of $x + y$.

.

II Program extraction in analysis

Approximating real numbers by rationals

We are looking for properties of real numbers which give us realizers that can be used ad data structures for exact real number computation:

$$\mathrm{Cauchy} :\equiv \{x \mid \forall n \in \mathbb{N}\, \exists p \in \mathbb{Q}\, (|x - p| \leq 2^{-n})\}$$

where $\mathbb{Q} :\equiv \{p \mid \exists k, l, m \in \mathbb{N}\, (m > 0 \wedge p = (k - l)/m)\}$

$$\tau(\mathbb{Q}) \quad =$$

Approximating real numbers by rationals

We are looking for properties of real numbers which give us realizers that can be used ad data structures for exact real number computation:

$$\mathrm{Cauchy} :\equiv \{x \mid \forall n \in \mathbb{N}\, \exists p \in \mathbb{Q}\, (|x - p| \leq 2^{-n})\}$$

where $\mathbb{Q} :\equiv \{p \mid \exists k, l, m \in \mathbb{N}\, (m > 0 \wedge p = (k - l)/m)\}$

$$\begin{aligned} \tau(\mathbb{Q}) &= \mathrm{Nat}^3 \\ \tau(\mathrm{Cauchy}) &= \end{aligned}$$

Approximating real numbers by rationals

We are looking for properties of real numbers which give us realizers that can be used ad data structures for exact real number computation:

$$\mathrm{Cauchy} :\equiv \{x \mid \forall n \in \mathbb{N}\, \exists p \in \mathbb{Q}\, (|x - p| \leq 2^{-n})\}$$

where $\mathbb{Q} :\equiv \{p \mid \exists k, l, m \in \mathbb{N}\, (m > 0 \wedge p = (k - l)/m)\}$

$$\begin{aligned} \tau(\mathbb{Q}) &= \mathrm{Nat}^3 \\ \tau(\mathrm{Cauchy}) &= \mathrm{Nat} \to \mathrm{Nat}^3 \end{aligned}$$

A realizer of the formula $\mathrm{Cauchy}(x)$ is an infinite sequence of (representations of) rational numbers converging to x at an exponential rate.

Signed digit streams

Set

$$\begin{aligned} \mathbb{I} &:= [-1,1] = \{x \mid -1 \leq x \leq 1\} \\ \mathrm{SD} &:= \{-1,0,1\} = \{x \mid x = -1 \vee x = 0 \vee x = 1\} \\ \mathrm{av}_d(x) &:= (x+d)/2 \end{aligned}$$

$$\mathrm{C}_0 \overset{\nu}{=} \{x \mid x \in \mathbb{I} \wedge \exists d \in \mathrm{SD}\, \exists y \in \mathrm{C}_0\, (x = \mathrm{av}_d(y))\}$$

$$\tau(\mathbb{I}) =$$

Signed digit streams

Set

$$\begin{aligned} \mathbb{I} &:= [-1,1] = \{x \mid -1 \leq x \leq 1\} \\ \mathrm{SD} &:= \{-1,0,1\} = \{x \mid x = -1 \vee x = 0 \vee x = 1\} \\ \mathrm{av}_d(x) &:= (x+d)/2 \end{aligned}$$

$$\mathrm{C}_0 \overset{\nu}{=} \{x \mid x \in \mathbb{I} \wedge \exists d \in \mathrm{SD}\, \exists y \in \mathrm{C}_0\, (x = \mathrm{av}_d(y))\}$$

$$\begin{aligned} \tau(\mathbb{I}) &= 1 \\ \tau(\mathrm{SD}) &= \end{aligned}$$

Signed digit streams

Set

$$\begin{aligned} \mathbb{I} &:= [-1,1] = \{x \mid -1 \leq x \leq 1\} \\ \mathrm{SD} &:= \{-1,0,1\} = \{x \mid x = -1 \vee x = 0 \vee x = 1\} \\ \mathrm{av}_d(x) &:= (x+d)/2 \end{aligned}$$

$$\mathrm{C}_0 \overset{\nu}{=} \{x \mid x \in \mathbb{I} \wedge \exists d \in \mathrm{SD}\, \exists y \in \mathrm{C}_0\, (x = \mathrm{av}_d(y))\}$$

$$\begin{aligned} \tau(\mathbb{I}) &= 1 \\ \tau(\mathrm{SD}) &= 1+1+1 \approx \mathrm{SD} \\ \tau(\mathrm{C}_0) &= \end{aligned}$$

Signed digit streams

Set

$$\begin{aligned} \mathbb{I} &:= [-1,1] = \{x \mid -1 \leq x \leq 1\} \\ \mathrm{SD} &:= \{-1,0,1\} = \{x \mid x = -1 \vee x = 0 \vee x = 1\} \\ \mathrm{av}_d(x) &:= (x+d)/2 \end{aligned}$$

$$\mathrm{C}_0 \stackrel{\nu}{=} \{x \mid x \in \mathbb{I} \wedge \exists d \in \mathrm{SD}\, \exists y \in \mathrm{C}_0\, (x = \mathrm{av}_d(y))\}$$

$$\begin{aligned} \tau(\mathbb{I}) &= 1 \\ \tau(\mathrm{SD}) &= 1+1+1 \approx \mathrm{SD} \\ \tau(\mathrm{C}_0) &= \text{let } \mathrm{SDS} = \mathrm{SD} \times \mathrm{SDS} \text{ in } \mathrm{SDS} \end{aligned}$$

Hence SDS is the type of infinite streams of signed digits.

Signed digits vs. Cauchy sequences

One easily sees

$$d \,\mathbf{r}\, \mathrm{Cauchy}(x) \;\Leftrightarrow\; x = \sum_{0}^{\infty} d_i * 2^{i+1}$$

I.e. d is a *signed digit representation* of x.

Theorem 5

$\mathrm{C}_0 = \mathrm{Cauchy}$.

Signed digits vs. Cauchy sequences

One easily sees

$$d \,\mathbf{r}\, \mathrm{Cauchy}(x) \;\Leftrightarrow\; x = \sum_{0}^{\infty} d_i * 2^{i+1}$$

I.e. d is a *signed digit representation* of x.

Theorem 5

$\mathrm{C}_0 = \mathrm{Cauchy}$.

From a proof of Theorem 5 one extracts programs translating between the signed-digit- and the Cauchy-representation.

Extracting exact real arithmetic

Theorem 6 If $x, y \in \mathrm{C}_0$ then $\frac{x+y}{2} \in \mathrm{C}_0$.

Extracting exact real arithmetic

Theorem 6 If $x, y \in \mathrm{C}_0$ then $\frac{x+y}{2} \in \mathrm{C}_0$.

Theorem 7 If $x, y \in \mathrm{C}_0$ then $xy \in \mathrm{C}_0$.

Extracting exact real arithmetic

Theorem 6 If $x, y \in \mathrm{C}_0$ then $\frac{x+y}{2} \in \mathrm{C}_0$.

Theorem 7 If $x, y \in \mathrm{C}_0$ then $xy \in \mathrm{C}_0$.

From these theorems one extracts implementations of addition and multiplication w.r.t. the signed digit representation.

Extracting exact real arithmetic

Theorem 6 If $x, y \in \mathrm{C}_0$ then $\frac{x+y}{2} \in \mathrm{C}_0$.

Theorem 7 If $x, y \in \mathrm{C}_0$ then $xy \in \mathrm{C}_0$.

From these theorems one extracts implementations of addition and multiplication w.r.t. the signed digit representation.

Similar implementations were studied by Edalat, Potts, Heckmann, Escardo, Ciaffaglione, Gianantonio, e.t.c.

The difference is that we *extract* the programs
– together with their correctness proofs.

Approaching real functions coinductively

We are looking for descriptions of real functions as stream processors.
The idea is as follows: For reals $x \in \mathbb{I}$ and signed digit streams $d : \mathrm{SDS}$ we have

$$\begin{aligned} d \,\mathbf{r}\, \mathrm{C}_0(x) \quad &\Leftrightarrow \quad x = \sum_0^\infty d_i * 2^{i+1} \\ &\Leftrightarrow \quad x = \mathrm{av}_{d_0} \circ \mathrm{av}_{d_1} \circ \mathrm{av}_{d_2} \circ \ldots \end{aligned}$$

Hence for $f : \mathbb{I} \to \mathbb{I}$ and x as above

$$\begin{aligned} f(x) &= f \circ \mathrm{av}_{d_0} \circ \mathrm{av}_{d_1} \circ \mathrm{av}_{d_2} \circ \ldots \\ &= (f \circ \mathrm{av}_{d_0}) \circ \mathrm{av}_{d_1} \circ \mathrm{av}_{d_2} \circ \ldots \end{aligned}$$

which means we *read* digit d_0.

Approaching real functions coinductively

But, if we are lucky, we can as well do, setting $\mathrm{va}_e(x) := 2x - d$ (the inverse of av_d),

$$\begin{aligned} f(x) &= f \circ \mathrm{av}_{d_0} \circ \mathrm{av}_{d_1} \circ \mathrm{av}_{d_2} \circ \dots \\ &= \mathrm{av}_e \circ (\mathrm{va}_e \circ f) \circ \mathrm{av}_{d_0} \circ \mathrm{av}_{d_1} \circ \mathrm{av}_{d_2} \circ \dots \end{aligned}$$

This means we are *writing* the digit e. However, this is only possible if the function $\mathrm{va}_e \circ f$ maps $\mathbb{I}$ to $\mathbb{I}$. The latter is the case if and only if $f[\mathbb{I}] \subseteq \mathbb{I}_e := [e/2 - 1/2, e/2 + 1/2]$ which is not necessarily the case.

However, if f is continuous, this will be eventually the case after finitely many reading steps (because reading makes the function flatter).

Coinductive/Inductive real functions

Hence we define $\mathrm{C}_1 \subseteq \mathbb{I}^{\mathbb{I}}$ by a nested inductive/coinductive definition as follows:

$$\mathrm{C}_1 \overset{\nu}{=} \text{let } J \overset{\mu}{=} \{f : \mathbb{I} \to \mathbb{I} \mid (\exists e \in \mathrm{SD}\, \mathrm{va}_e \in \mathrm{C}_1) \vee (\forall d \in \mathrm{SD}\, f \circ \mathrm{av}_d \in J)\} \text{ in } J$$

Memo tries for continuous functions

Theorem 8 h is uniformly continuous iff $h \in \mathrm{C}_1$.

Memo tries for continuous functions

Theorem 8 h is uniformly continuous iff $h \in \mathrm{C}_1$.

From the proof of this theorem one extracts programs translating between realisers of "f is continuous" (where continuity has to be defined in a constructively meaningful way) and realisers of "$f \in \mathrm{C}_1$".

Memo tries for continuous functions

Theorem 8 h is uniformly continuous iff $h \in \mathrm{C}_1$.

From the proof of this theorem one extracts programs translating between realisers of "f is continuous" (where continuity has to be defined in a constructively meaningful way) and realisers of "$f \in \mathrm{C}_1$".

What is a realiser of "$f \in \mathrm{C}_1$"?

Memo tries for continuous functions

Theorem 8 *h* is uniformly continuous iff $h \in \mathrm{C}_1$.

From the proof of this theorem one extracts programs translating between realisers of "*f* is continuous" (where continuity has to be defined in a constructively meaningful way) and realisers of "$f \in \mathrm{C}_1$".

What is a realiser of "$f \in \mathrm{C}_1$"?

It is a finitely branching non-wellfounded tree describing when *f* emits and absorbs digits. I.p. it is a *data structure*, not a function.

Memo tries for continuous functions

Theorem 8 *h* is uniformly continuous iff $h \in \mathrm{C}_1$.

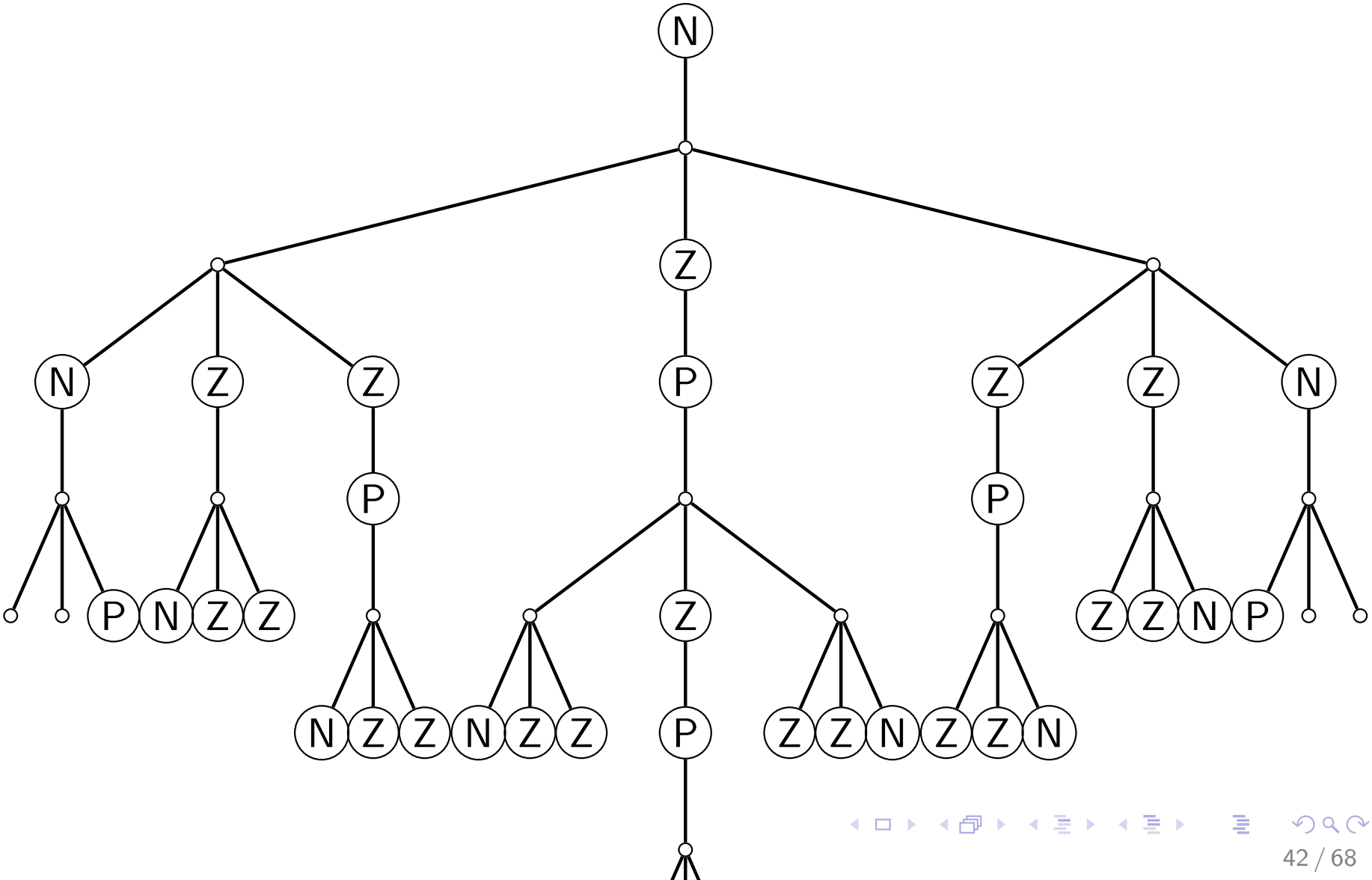
From the proof of this theorem one extracts programs translating between realisers of "*f* is continuous" (where continuity has to be defined in a constructively meaningful way) and realisers of "$f \in \mathrm{C}_1$".

What is a realiser of "$f \in \mathrm{C}_1$"?

It is a finitely branching non-wellfounded tree describing when *f* emits and absorbs digits. I.p. it is a *data structure*, not a function.

Similar trees have been studied by P. Hancock, D. Pattinson, N. Ghani.
P. Hancock, D. Pattinson, N. Ghani. Representations of Stream Processors Using Nested Fixed Points, LMCS 5, 2009.

Tree of the logistic map, $f_a(x) = a(1 - x^2) - 1$, with $a = 2/3$



Extracting memoized exact real arithmetic

Extracting memoized exact real arithmetic

The definition of $\mathrm{C}_1 \subseteq \mathbb{I}^{\mathbb{I}}$ can be generalised to $\mathrm{C}_n \subseteq \mathbb{I}^{(\mathbb{I}^n)}$.

Extracting memoized exact real arithmetic

The definition of $\mathrm{C}_1 \subseteq \mathbb{I}^{\mathbb{I}}$ can be generalised to $\mathrm{C}_n \subseteq \mathbb{I}^{(\mathbb{I}^n)}$.

Theorem 9 The average function lies in C_2.

Extracting memoized exact real arithmetic

The definition of $\mathrm{C}_1 \subseteq \mathbb{I}^{\mathbb{I}}$ can be generalised to $\mathrm{C}_n \subseteq \mathbb{I}^{(\mathbb{I}^n)}$.

Theorem 9 The average function lies in C_2.

Theorem 10 Multiplication lies in C_2.

Extracting memoized exact real arithmetic

The definition of $\mathrm{C}_1 \subseteq \mathbb{I}^{\mathbb{I}}$ can be generalised to $\mathrm{C}_n \subseteq \mathbb{I}^{(\mathbb{I}^n)}$.

Theorem 9 The average function lies in C_2.

Theorem 10 Multiplication lies in C_2.

From Theorems 5,6 one extracts implementations of addition and multiplication as memo-tries (relation to work by Hinze and Altenkirch?)

Extracting memoized exact real arithmetic

The definition of $\mathrm{C}_1 \subseteq \mathbb{I}^{\mathbb{I}}$ can be generalised to $\mathrm{C}_n \subseteq \mathbb{I}^{(\mathbb{I}^n)}$.

Theorem 9 The average function lies in C_2.

Theorem 10 Multiplication lies in C_2.

From Theorems 5,6 one extracts implementations of addition and multiplication as memo-tries (relation to work by Hinze and Altenkirch?)

Experiments show considerable speed-up when sampling "hard" functions (e.g. high iterations of the logistic map) on a very fine grid.

Extracting memoized exact real arithmetic

The definition of $\mathrm{C}_1 \subseteq \mathbb{I}^{\mathbb{I}}$ can be generalised to $\mathrm{C}_n \subseteq \mathbb{I}^{(\mathbb{I}^n)}$.

Theorem 9 The average function lies in C_2.

Theorem 10 Multiplication lies in C_2.

From Theorems 5,6 one extracts implementations of addition and multiplication as memo-tries (relation to work by Hinze and Altenkirch?)

Experiments show considerable speed-up when sampling "hard" functions (e.g. high iterations of the logistic map) on a very fine grid.

Problems in Coq with the nested inductive/coinductive definition of C_1:

Extracting memoized exact real arithmetic

The definition of $\mathrm{C}_1 \subseteq \mathbb{I}^{\mathbb{I}}$ can be generalised to $\mathrm{C}_n \subseteq \mathbb{I}^{(\mathbb{I}^n)}$.

Theorem 9 The average function lies in C_2.

Theorem 10 Multiplication lies in C_2.

From Theorems 5,6 one extracts implementations of addition and multiplication as memo-tries (relation to work by Hinze and Altenkirch?)

Experiments show considerable speed-up when sampling "hard" functions (e.g. high iterations of the logistic map) on a very fine grid.

Problems in Coq with the nested inductive/coinductive definition of C_1:

Coq accepts the definition, but rejects the attempted coinductive proofs as not formally guarded.

Integration

Let $\int f$ denote the definite integral $\int_{-1}^{1} f(x)\mathrm{d}x$.

Integration

Let $\int f$ denote the definite integral $\int_{-1}^{1} f(x)\mathrm{d}x$.

We assume the following "axioms" about $\int f$:

Integration

Let $\int f$ denote the definite integral $\int_{-1}^{1} f(x)\mathrm{d}x$.

We assume the following "axioms" about $\int f$:

(a) $\int f = \frac{1}{2}\int(\mathrm{va}_d \circ f) + d$ where $\mathrm{va}_d(x) := 2x - d$.

(b) $\int f = \frac{1}{2}(\int(f \circ \mathrm{av}_{-1}) + \int(f \circ \mathrm{av}_1))$.

Integration

Let $\int f$ denote the definite integral $\int_{-1}^{1} f(x)\mathrm{d}x$.

We assume the following "axioms" about $\int f$:

(a) $\int f = \frac{1}{2}\int(\mathrm{va}_d \circ f) + d$ where $\mathrm{va}_d(x) := 2x - d$.

(b) $\int f = \frac{1}{2}(\int(f \circ \mathrm{av}_{-1}) + \int(f \circ \mathrm{av}_1))$.

Theorem 7 If $f \in \mathrm{C}_1$, then $\forall n \in \mathbb{N}\, \exists q \in \mathbb{Q} \,|\, \int f - q| \leq 2^{-n}$.

Integration

Let $\int f$ denote the definite integral $\int_{-1}^{1} f(x)\mathrm{d}x$.

We assume the following "axioms" about $\int f$:

(a) $\int f = \frac{1}{2}\int(\mathrm{va}_d \circ f) + d$ where $\mathrm{va}_d(x) := 2x - d$.

(b) $\int f = \frac{1}{2}(\int(f \circ \mathrm{av}_{-1}) + \int(f \circ \mathrm{av}_1))$.

Theorem 7 If $f \in \mathrm{C}_1$, then $\forall n \in \mathbb{N}\, \exists q \in \mathbb{Q}\, |\int f - q| \leq 2^{-n}$.

The extracted program has some similarity with A. Simpson's, but is more efficient because the functions to be integrated are represented differently.

III Projects

Proof system for program extraction

All the exisiting proof systems are rather general purpose than specialized to program extraction.

We are developing a small system exclusively dedicated to program extraction in the form sketched previously.

The question whether to use typed or untyped relizers has been partly answered: It doesn't matter, both have, essentially, the same semantics.

> B., T. Hou. Typed vs Untyped Realizability. Submitted to MFPS 2012.

Minlog implementation of program extraction in analysis

Several results in constructive analysis, based on a Cauchy representation of reals, were implemented in Minlog and programs have been extracted, e.g. Intermediate Value Theorem, Inverse Function Theorem.

> H. Schwichtenberg. Realizability interpretation of proofs in constructive analysis, TOCS 43, 2008.

Recently, K. Miyamoto and H. Schwichtenberg have started to implement the coinductive approach to analysis in Minlog:

> K. Miyamoto and H. Schwichtenberg. Program extraction in exact real arithmetic. Submitted.

Extracting normalization by evaluation

Tait's strong normalization proof for the simply typed lambda-calculus has been formalized was shown to have the normalization-by-evaluation method as computational content:

> B. Program extraction from normalization proofs. LNCS 664, 1993.

This was implemented in Coq, Isabelle and Minlog:

> B., S. Berghofer, P. Letouzey, H. Schwichtenberg. Program extraction from normalization proofs. Studia Logica 82, 2006.

Extracting programs from transfinite induction

Gentzen showed that transfinite induction below ϵ_0 can be proven in Peano Arithmetic (in fact Heyting Arithmetic).

Formalizing his proof one obtains higher type Gödel primitive recursive programs for functions originally defined by transifinite recursion.

The extracted higher type implementations (originally due to H. Schwichtenberg) turned out to be significantly more efficient than the one using transfinite recursion.

> B. Program extraction from Gentzen's proof of transfinite induction up to ϵ_0. LNCS 2183, 2001.

Extracting in-place Quicksort

We have extracted in-place Quicksort from a proof that every list can be sorted. The extracted program can be directly viewed as imperative code.

> B., M. Seisenberger, G. Woods. A Case Study in Imperative Program Extraction: In-Place Quicksort. Submitted to ITP 2012.

Extracting a DPLL SAT-solver

We have proven in Minlog the following theorem:

> Every CNF has a DPLL refutation or a model

The extracted program is improved drastically in size and efficiency by the use of uniform quantifiers.

> B., A. Lawrence, M. Seisenberger. Extracting a DPLL Algorithm. Submitted to MFPS 2012.

IV Programs from classical proofs

The general problem

Assume

$$\Gamma \quad \vdash_{\mathrm{c}} \quad \exists x\, A_0(x)$$

where $A_0(x)$ is atomic.

We wish to extract a term t such that

$$\Delta \quad \vdash_{\mathrm{i}} \quad A_0(t)$$

for some "true" assumptions Δ.

In many cases the combined Gödel/Gentzen/Friedman translation works.

The Gödel/Gentzen/Friedman translation

$B^A :\equiv B^g[A/\bot]$ i.e.

$$\begin{array}{rcl}
\bot^A & \equiv & A \\
B^A & \equiv & (B \to A) \to A \quad (B \text{ atomic }, B \neq \bot) \\
(\exists y\, B)^A & \equiv & ((\exists y\, B^A) \to A) \to A \\
(B \vee C)^A & \equiv & ((B^A \vee C^A) \to A) \to A \\
(B \circ C)^A & \equiv & B^A \circ C^A \quad (\circ \in \{\wedge, \to\}) \\
(\forall y\, B)^A & \equiv & \forall y\, (B^A)
\end{array}$$

One has in general:

$$\text{If } \Gamma \vdash_c C, \text{ then } \Gamma^A \vdash_i C^A$$

Program extraction in classical arithmetic

$A :\equiv \exists x\, A_0(x)$.

$$\begin{array}{rcl} \mathrm{PA} & \vdash_{\mathrm{c}} & A \\ \mathrm{PA}^A & \vdash_{\mathrm{i}} & A^A \\ \mathrm{PA} & \vdash_{\mathrm{i}} & A \end{array}$$

Since $\mathrm{PA} \vdash_{\mathrm{i}} PA^A$ and $\vdash_{\mathrm{i}} A^A \to A$.

Now apply program extraction from intuitionistic proofs.

Why does $\mathrm{PA} \vdash_{\mathrm{i}} \mathrm{PA}^A$

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

$$B(0)^A \wedge \forall x\,(B(x)^A \to B(x+1)^A) \to \forall x\, B(x)^A$$

The translation of an induction axiom is again an induction axiom because the induction scheme doesn't mention atomic formulas, $\vee$, or $\exists$ explicitely.

But: The logical complexity of the translated axiom is higher!

Parsons showed that for Σ^0_2-induction the complexity can be kept.

C. Parsons. On *n*-quantifier induction. JSL 37, 1972.

Wolves in the sheep skin: Countable and Dependent Choice

$\mathrm{AC} \quad \forall n\, \exists x\, B(n,x) \to \exists f\, \forall n\, B(n, fn)$

$\mathrm{DC}\ \forall n\, \forall x\, \exists y\, A(n,x,y) \to \exists f\, \forall n\, A(n, fn, f(n+1))$

AC and DC are intuitionistically weak (conservative over Heyting Arithmetic), but classically strong (they yield full classical analysis).

Gödel/Gentzen/Friedman translation of AC:

$$\forall n\, ((\exists x\, B(n,x)^A \to A) \to A) \to ((\exists f\, \forall n\, B(n, fn)) \to A) \to A$$

AC^A and DC^A are classically true, but not provable in arithmetic.

Alternative: Open Induction

Let $U(\alpha)$ be an *open* property of infinite sequences $\alpha\colon \mathbb{N} \to \rho$, i.e. $U(\alpha)$ depends on a finite initial segment of α only.

Let $<$ be a *decidable wellfounded* binary relation on ρ.

Define the *lexicographic extension*, $<_{\mathsf{lex}}$, on $\mathbb{N} \to \rho$ by

$$\beta <_{\mathsf{lex}} \alpha :\equiv \exists n\, (\overline{\beta} n = \overline{\alpha} n \wedge \beta n < \alpha n)$$

Open induction (OI)

$$\forall \alpha\, (\forall \beta <_{\mathsf{lex}} \alpha\, U(\beta) \to U(\alpha)) \to \forall \alpha\, U(\alpha)$$

Open induction was first formulated by Raoult.

> J-C Raoult. Proving open properties by induction. Information processing letters 29, 1988.

Induction on a non-welfounded relation!

The wellfoundedness of $<$ does not imply the wellfoundedness of $<_{\mathsf{lex}}$:

$$1111111\ldots >_{\mathsf{lex}} 0111111\ldots >_{\mathsf{lex}} 0011111\ldots\ldots$$

Update induction

An important special case of open induction is *update induction*

Let α, β be *partial sequences* (in ρ) with *decidable domain*, i.e $\alpha, \beta \colon \mathrm{Nat} \to 1 + \rho$.

β is an *update* of α ($\beta <_{\mathsf{up}} \alpha$) if β is the same as α except that at one argument α is undefined, but β is defined.

Update induction (UI)

$$\forall \alpha \, (\forall \beta <_{\mathsf{up}} \alpha \, U(\beta) \to U(\alpha)) \to \forall \alpha \, U(\alpha)$$

where U ranges over open predicates.

Proposition

DC, OI and UI are classically (i.e. provably in PA^ω) equivalent.

A generalized notion of 'open'

In order to for open induction to be closed under negative translation we need to define 'open' (intuitionistically) slightly more general than usual:

A predicate $U(\alpha)$ is *open*, if it is of the form

$$U(\alpha) \equiv \forall n\, C(\overline{\alpha} n) \to \exists n\, B(\overline{\alpha} n)$$

where C is arbitrary and B is a Σ-*formula*, that is, B is $\to \forall$ free and all predicates in B have no function arguments.

Equivalences

Proposition

Open (Update) induction is closed under the GGF-translation.

Corollary

If $\mathrm{UI}(\mathrm{OI})(\mathrm{DC}) \vdash_{\mathrm{c}} \exists x\, B(x)$, then $\mathrm{UI} \vdash_{\mathrm{i}} \exists x\, B(x)$ for every Σ-formula $B(x)$.

(B as above)

Update recursion (UR)

$$\mathsf{R}^{\mathsf{u}} f \alpha =_\tau f\alpha(n, x \mapsto \mathrm{if}(n \notin \mathrm{dom}\alpha\, \mathsf{R}^{\mathsf{u}} f \alpha^x_n, 0^\tau))$$

where τ is $\to$ free.

Theorem $\mathrm{Cont} + \mathrm{UI} + \mathrm{UR} \vdash_{\mathrm{i}} \Phi\, \mathbf{mr}\, \mathrm{UI}$

for some term Φ explicitely defined from UR.

Where Cont states *continuity* for functionals:

$$\forall F, \alpha\, \exists n \forall \beta\, (\overline{\alpha} n =_\rho \overline{\beta} n \to F\alpha =_{\mathrm{Nat}} F\beta)$$

(holds in all constructively meaningful models)

Normalization

Theorem

Every closed update recursive term M of a $\rightarrow$ free type reduces to a numeral (canonical term).

Proof

1. Interpret terms in the Scott-Ershov model $\hat{\mathcal{C}}$ of partial continuous functionals and show that all terms have a total value (the totality of R^{u} is proved by update induction).
2. Use Plotkin's adequacy result (TCS 5).

Program extraction with classical choice

Theorem

From a derivation $\mathrm{DC} \vdash_{\mathrm{c}} \forall x^{\rho}\, \exists y^{\tau}\, A(x, y)$ where $A(x, y)$ is a Σ-formula and τ is a data type one can extract a closed update recursive term $\Phi^{\rho \to \tau}$ such that for each closed term r^{ρ} the term Φr reduces to a numeral $\underline{n}$ such that $A(r, \underline{n})$ holds in the total continuous (Kleene-Kreisel) functionals $\mathcal{C}$.

Other computational interpretations of classical countable choice

Spector Dialectica interpretation of $\mathrm{AC}^{\neg\neg}$ and $\mathrm{DC}^{\neg\neg}$ using Bar recursion in finite types.

Berardi/Bezem/Coquand Special realizability interpretation of $\mathrm{AC}^{\neg\neg}$ and $\mathrm{DC}^{\neg\neg}$ using a kind of 'update recursion'.

Oliva/B Modified realizability interpretation of $(\mathrm{AC}^{\neg\neg})_A$ and $(\mathrm{DC}^{\neg\neg})_A$ using modified bar recursion.

Escardo/Oliva Products of selection functions

Krivine Machine oriented interpretation ("classical realizability").

The End

Thanks for your attention!