Autumn school "Proof and Computation"

Herrsching, 14-20 September 2025

Program extraction in higher-order logic

Ulrich Berger

Swansea University

Lecture 2

Examples:

- Continuous functions, integration
- Martin Hofmann's breadth-first search
- Non-monotone induction

Axiomatic mathematics in CST

Mathematical constants, functions and predicates,
such as 0, $+$, $*$, $<$, can be treated as free variables in CST
(members of Γ).

Mathematical axioms can be treated as free assumptions
(members of Δ).

Axiomatic mathematics in CST

Mathematical constants, functions and predicates,
such as 0, $+$, $*$, $<$, can be treated as free variables in CST
(members of Γ).

Mathematical axioms can be treated as free assumptions
(members of Δ).

The realizability interpretation can be optimized such that
disjunction-free formulas are interpreted by themselves and need no realizers.

In this way, garbage in extracted programs can be largely avoided.

Minlog and Rocq have such optimizations built-in.

How the optimizations work in CST will be explained in Lecture 3.

Natural numbers

In the following, we assume one sort ι for individuals and the language and disjunction-free axioms of ordered fields.

Natural numbers

In the following, we assume one sort ι for individuals and the language and disjunction-free axioms of ordered fields.

$$\mathbb{N} := \mu_{\iota \to o}\, \Phi$$

where

$$\Phi := \lambda X : \iota \to o \,.\, \lambda x : \iota \,.\, x =_\iota 0 \vee X(x - 1)$$

Natural numbers

In the following, we assume one sort ι for individuals and the language and disjunction-free axioms of ordered fields.

$$\begin{aligned} \mathbb{N} &:= \mu_{\iota \to o}\, \Phi \\ \text{where} & \\ \Phi &:= \lambda X : \iota \to o \,.\, \lambda x : \iota \,.\, x =_\iota 0 \vee X(x-1) \end{aligned}$$

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

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

Natural numbers

In the following, we assume one sort ι for individuals and the language and disjunction-free axioms of ordered fields.

$$\begin{aligned} \mathbb{N} &:= \mu_{\iota \to o}\, \Phi \\ \text{where} & \\ \Phi &:= \lambda X : \iota \to o \,.\, \lambda x : \iota \,.\, x =_\iota 0 \vee X(x-1) \end{aligned}$$

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

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

In 'minimal' CST:

$$\begin{aligned} \Phi := \lambda X : \iota \to o \,.\, \lambda x : \iota \,.\, \forall A : o \,. \\ ((\forall P : \iota \to o \,.\, P\, x \to P\, 0) \to A) \to \\ (X(x-1) \to A) \to A \end{aligned}$$

Induction on natural numbers

Since the operator Φ is monotone, induction on $\mathbb{N}$ is available:

$$\frac{\Phi\, P \subseteq_{\iota\to o} P}{\mathbb{N} \subseteq_{\iota\to o} P}\, \mathrm{MInd}_{\iota\to o}(\mathbb{N})$$

This is equivalent to

$$\frac{P\,0 \qquad \forall x : \iota\,.\, P(x) \supset P(x+1)}{\forall x : \iota\,.\, \mathbb{N}(x) \supset P(x)}\, \mathrm{MInd}_{\iota\to o}(\mathbb{N})$$

since, by the field axioms, $\forall x : \iota\,.\, (x = 0 \vee P(x-1)) \supset P(x)$ is equivalent to $P(0) \wedge \forall x : \iota\,.\, P(x) \supset P(x+1)$.

Induction on natural numbers

Since the operator Φ is monotone, induction on $\mathbb{N}$ is available:

$$\frac{\Phi\, P \subseteq_{\iota\to o} P}{\mathbb{N} \subseteq_{\iota\to o} P}\, \mathrm{MInd}_{\iota\to o}(\mathbb{N})$$

This is equivalent to

$$\frac{P\,0 \qquad \forall x : \iota\,.\, P(x) \supset P(x+1)}{\forall x : \iota\,.\, \mathbb{N}(x) \supset P(x)}\, \mathrm{MInd}_{\iota\to o}(\mathbb{N})$$

since, by the field axioms, $\forall x : \iota\,.\, (x = 0 \vee P(x-1)) \supset P(x)$ is equivalent to $P(0) \wedge \forall x : \iota\,.\, P(x) \supset P(x+1)$.

For example, one can prove that $\mathbb{N}$ is closed under addition.

$$\forall x, y : \iota\,.\, \mathbb{N}(x) \supset \mathbb{N}(y) \supset \mathbb{N}(x+y)$$

(Assuming $\mathbb{N}(x)$, show $\mathbb{N} \subseteq_{\iota\to o} \lambda y : \iota\,.\, \mathbb{N}(x+y)$ by induction.)

Program extraction with natural numbers

Recall that $\mathbf{R}(\mu_\rho) = \mu_{\mathbf{R}(\rho)}$. Therefore,

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

This means a realizes $\mathbb{N}(x)$ iff a is a *unary* representation of x.

Program extraction with natural numbers

Recall that $\mathbf{R}(\mu_\rho) = \mu_{\mathbf{R}(\rho)}$. Therefore,

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

This means a realizes $\mathbb{N}(x)$ iff a is a *unary* representation of x.

Since $\mathcal{T}(\mu_\rho(\Phi)) = \mathbf{fix}(\mathcal{T}(\Phi))$,

$$\mathcal{T}(\mathbb{N}) = \mathbf{fix}(\lambda\alpha : * \,.\, \mathbf{1} \oplus \alpha) =: \mathbf{nat}$$

which is the type of unary natural numbers.

Program extraction with natural numbers

Recall that $\mathbf{R}(\mu_\rho) = \mu_{\mathbf{R}(\rho)}$. Therefore,

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

This means a realizes $\mathbb{N}(x)$ iff a is a *unary* representation of x.

Since $\mathcal{T}(\mu_\rho(\Phi)) = \mathbf{fix}(\mathcal{T}(\Phi))$,

$$\mathcal{T}(\mathbb{N}) = \mathbf{fix}(\lambda\alpha : *\,.\,\mathbf{1} \oplus \alpha) =: \mathbf{nat}$$

which is the type of unary natural numbers.

Exercise. Modify the definition of $\mathbb{N}$ so that *binary* representations are obtained.

Program extraction with natural numbers

Recall that $\mathbf{R}(\mu_\rho) = \mu_{\mathbf{R}(\rho)}$. Therefore,

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

This means a realizes $\mathbb{N}(x)$ iff a is a *unary* representation of x.

Since $\mathcal{T}(\mu_\rho(\Phi)) = \mathbf{fix}(\mathcal{T}(\Phi))$,

$$\mathcal{T}(\mathbb{N}) = \mathbf{fix}(\lambda\alpha : *\,.\, \mathbf{1} \oplus \alpha) =: \mathbf{nat}$$

which is the type of unary natural numbers.

Exercise. Modify the definition of $\mathbb{N}$ so that *binary* representations are obtained.

The extracted program of the inductive proof that $\mathbb{N}$ is closed under addition is the usual recursive definition of addition on unary numbers.

Signed digit representation of reals in $[-1,1]$

$$
\begin{array}{rcl}
\mathbb{I} & := & [-1,1] = \lambda x : o\,.\; -1 \le x \wedge x \le 1 \\
\mathrm{SD} & := & \{-1,0,1\} = \lambda x : \iota\,.\, x = -1 \vee x = 0 \vee x = 1 \\
\mathbf{av}_d & := & \lambda x : \iota\,.\,(x+d)/2 \quad (d = -1,0,1) \\
\mathrm{C}_0(x) & \stackrel{\nu}{=} & x \in \mathbb{I} \wedge \exists d \in \mathrm{SD}\, \exists y\,.\, x = \mathbf{av}_d(y) \wedge \mathrm{C}_0(y)
\end{array}
$$

$a\,\mathbf{r}\,\mathrm{C}_0(x)$ iff a is an infinite stream of signed digits

$$a = d_0 : d_1 : d_2 : \ldots$$

such that $x = \sum d_n * 2^{-n}$, i.e. $d_0 : d_1 : d_2 : \ldots$ is a signed digit representation of x.

One can prove by coinduction, for example, that C_0 is closed under multiplication and extract from the proof a corecursive program multiplying signed digit representations.

Continuous functions on $[-1, 1]$

If reals in $\mathbb{I}$ are represented as digit streams (with digits $-1, 0, 1$), then a continuous function $f : \mathbb{I} \to \mathbb{I}$ can be viewed as a stream transformer

$$\begin{array}{c} a_0 : a_1 : a_2 : \ldots \\ f \Downarrow \\ b_0 : b_1 : \ldots \end{array}$$

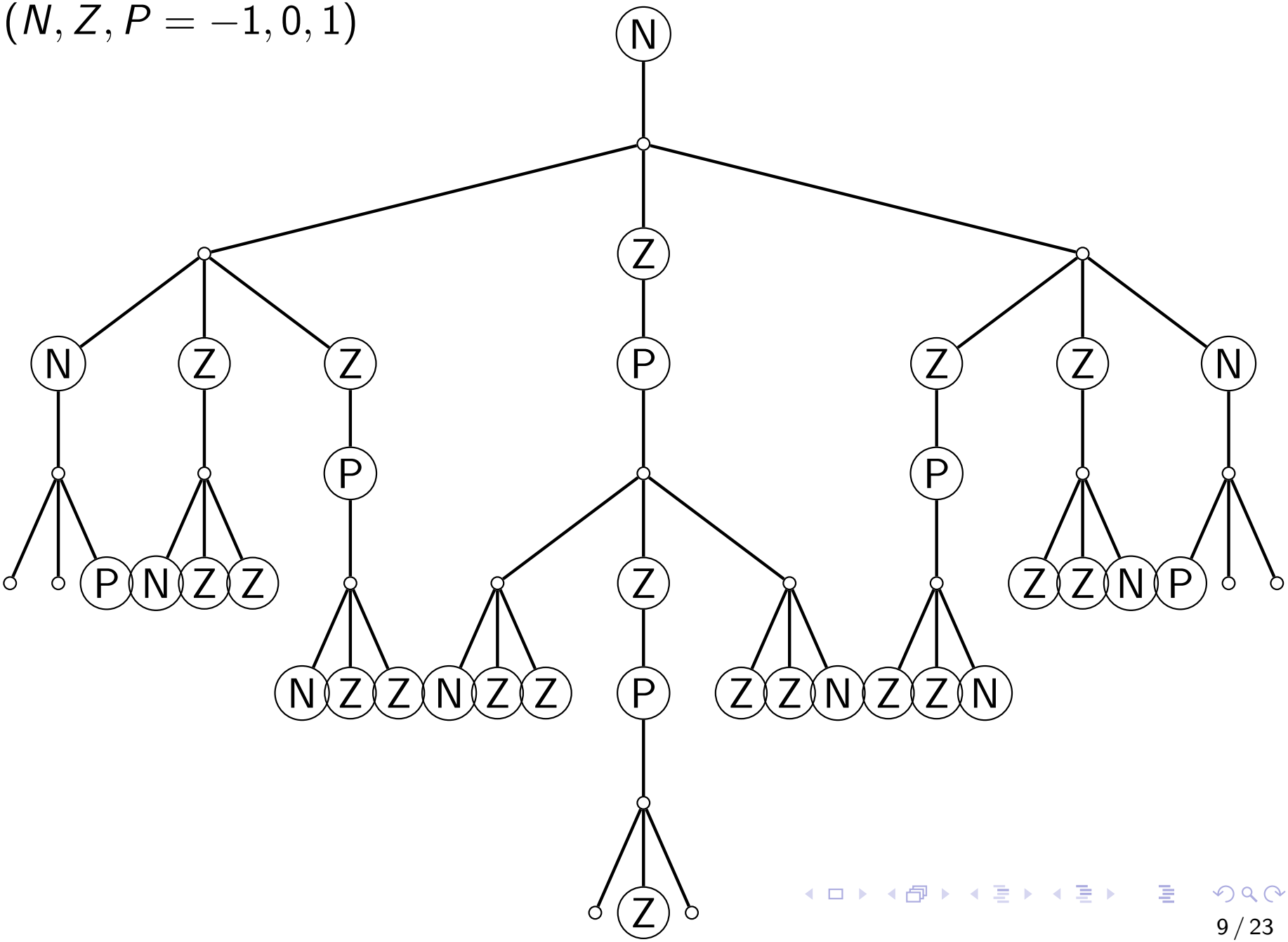
Assume f has read a_0, a_1, a_2 and produced b_0, b_1 so far. Then:

> Either produce a further output, b_2,
> or else read the next input.

Hence, a program for f can be viewed as a tree where each node is either a *writing* node which is labelled by an output digit and has one child, or a *reading* node with three children, one for each possible input digit.
On each path infinitely many writing nodes must occur.

A part of the tree of $f(x) = \frac{2}{3}(1 - x^2) - 1$

$(N, Z, P = -1, 0, 1)$



Extracting tree representations of continuous functions

Trees representing continuous functions can be extracted from proofs of membership of the following predicate:

$$\begin{aligned} \mathrm{C}_1 \quad := \quad & \nu\lambda X : (\iota \to \iota) \to o \,.\, \mu\lambda Y : (\iota \to \iota) \to o \,.\, \lambda f : \iota \to \iota \,. \\ & \exists b : \iota \, \exists g : \iota \to \iota \,. \\ & (f = \mathbf{av}_b \circ g \wedge X(g)) \vee \forall a \in \mathrm{SD} \,.\, Y(f \circ \mathbf{av}_a) \end{aligned}$$

Such a nesting of ν and μ is possible in Minlog, but not (to my knowledge) in other systems supporting program extraction.

A similar nested inductive/coinductive definition was studied by Ghani, Hancock, and Pattinson in a type theoretic context,

Integration

We add $\int$ as a formal symbol to the language (intended to denote integration over $\mathbb{I}$) and axiomatize it by the equations

$$\int(f \circ \mathbf{av}_d) = \tfrac{1}{2}\int f + d$$
$$\int f = \tfrac{1}{2}(\int(f \circ \mathbf{av}_{-1}) + \int(f \circ \mathbf{av}_1))$$

Theorem

If $\mathrm{C}_1(f)$, *then* $\int f$ *can be approximated by rationals, i.e.*

$$\forall n \in \mathbb{N}\, \exists q \in \mathbb{Q}\,.\, |\int f - q| < 2^{-n}$$

The extracted program computes for every continuous function $f : \mathbb{I} \to \mathbb{I}$ *a rational Cauchy sequence converging to* $\int f$.

Martin Hofmann's breadth-first search algorithm

```
data Tree = Leaf Int | Node Tree Int Tree

data Rou = Over | Next ((Rou -> [Int]) -> [Int])

unfold :: Rou -> (Rou -> [Int]) -> [Int]
unfold Over k = k Over
unfold (Next f) k = f k

br :: Tree -> Rou -> Rou
br (Leaf n) c = Next (\ k -> n : unfold c k)
br (Node t0 n t1) c =
     Next(\ k -> n : unfold c (k . br t0 . br t1))

extract :: Rou -> [Int]
extract Over = []
extract (Next f) = f extract

bfMH :: Tree -> [Int]
bfMH t = extract(br t Over)
```

The goal is to show that $\mathrm{bfMH}\ t$ terminates on all finite trees and yields the list of nodes in breadth-first order.

Verification strategy

We extract the algorithm from a proof that $\mathrm{bfMH}\, t$ yields the correct result for every finite tree t.

The extracted program will not use Haskell types or functions and the correctness does not rely on the correctness of the Haskell compiler.

Stating the correctness of Hofmann's algorithm

Naive breadth-first search

```
niv :: Tree -> [[Int]]
niv (Leaf n) = [[n]]
niv (Node t0 n t1) = [n] : zipWith (++) (niv t0) (niv t1)

bfspec :: Tree -> [Int]
bfspec t = concat (niv t)
```

Relevant inductive subsets of Int, [Int], Tree

$$
\begin{aligned}
\mathbb{N} &\stackrel{\mu}{=} \lambda n : \mathrm{Int} \,.\, n = 0 \vee \mathbb{N}(n-1) \\
\mathrm{List} &\stackrel{\mu}{=} \lambda l : [\mathrm{Int}] \,.\, l = [] \vee \exists n : \mathrm{Int}, l' : [\mathrm{Int}] \,.\, (l = n : l' \wedge \mathrm{N}(n) \wedge \mathrm{List}(l')) \\
\mathrm{T} &\stackrel{\mu}{=} \lambda t : \mathrm{Tree} \,. \\
&\qquad \exists n : \mathrm{Int} \,.\, (t = \mathrm{Leaf}\ n \wedge \mathrm{N}(n)) \vee \\
&\qquad \exists\, t_0, t_1 : \mathrm{Tree}, n : \mathrm{Int} \,.\, (t = \mathrm{Node}\ t_0\ n\ t_1 \wedge \mathrm{T}(t_0) \wedge N(n) \wedge T(t_1))
\end{aligned}
$$

Theorem

$\forall t : \mathrm{Tree} \,.\, \mathrm{T}(t) \supset \mathrm{List}(\mathrm{bfMH}\ t) \wedge \mathrm{bfMH}\ t = \mathrm{bfspec}\ t.$

Formalization

The types $\mathrm{Int}, [\mathrm{Int}], \mathrm{Tree}, \mathrm{Rou}$ are treated as base types.

Functions are modelled as constants (or free variables) of the appropriate types.

The defining equations are treated as axioms (or free assumptions).

$$\forall t : \mathrm{Tree}\,.\,\mathrm{T}(t) \supset \mathrm{List}(\mathrm{bfMH}\ t) \wedge \mathrm{bfMH}\ t = \mathrm{bfspec}\ t$$

Proof. We write zip for zipWith $(++)$.

$$\begin{aligned}
\text{isextractor} \quad := \quad & \lambda R : \mathrm{Rou} \times [[\mathrm{Int}]] \to o, \vec{l'} : [[\mathrm{Int}]], k : \mathrm{Rou} \to [\mathrm{Int}]\,. \\
& \forall c : \mathrm{Rou}, \vec{l} : [[\mathrm{Int}]]\,. \\
& R(c, \vec{l}) \supset \mathrm{List}(k\ c) \wedge k\ c = \mathrm{concat}(\mathrm{zip}\ \vec{l'}\ \vec{l})
\end{aligned}$$

$$\begin{aligned}
\mathrm{rep} \quad \stackrel{\mu}{=} \quad & \lambda c : \mathrm{Rou}, \vec{l_0} : [[\mathrm{Int}]]\,. \\
& (c = \mathrm{Over} \wedge \vec{l_0} = []) \vee \\
& \exists f : (\mathrm{Rou} \to [\mathrm{Int}]) \to [\mathrm{Int}], l : [\mathrm{Int}], \vec{l} : [[\mathrm{Int}]]\,. \\
& c = \mathrm{Next}(f) \wedge \vec{l_0} = l : \vec{l} \wedge \\
& \forall k : \mathrm{Rou} \to [\mathrm{Int}], \vec{l'} : [[\mathrm{Int}]]\,. \\
& \mathrm{isextractor}(\mathrm{rep}, \vec{l'}, k) \supset \\
& \mathrm{List}(f\ k) \wedge f\ k = l +\!\!+ \mathrm{concat}(\mathrm{zip}\ \vec{l'}\ \vec{l})
\end{aligned}$$

Lemma 1 $\mathrm{isextractor}(\mathrm{rep}, [], \mathrm{extract})$, i.e.

$$\forall c : \mathrm{Rou}, \vec{l} : [[\mathrm{Int}]]\,.\,\mathrm{rep}(c, \vec{l}) \supset \mathrm{List}(\mathrm{extract}\ c) \wedge \mathrm{extract}\ c = \mathrm{concat}\ \vec{l}$$

Proof of Lemma 1: Set

$R_0 := \lambda c : \mathrm{Rou}, \vec{l} : [[\mathrm{Int}]].\mathrm{List}(\mathrm{extract}\ c) \wedge \mathrm{extract}\ c = \mathrm{concat}\ ls$. Then our goal is $\mathrm{rep} \subseteq R_0$ which can be proven by monotone induction.

$\forall t : \mathrm{Tree} \,.\, \mathrm{T}(t) \supset \mathrm{List}(\mathrm{bfMH}\ t) \wedge \mathrm{bfMH}\ t = \mathrm{bfspec}\ t$

We already proved

Lemma 1. $\mathrm{isextractor}(\mathrm{rep}, [], \mathrm{extract})$.

Lemma 2.

$$\forall t : \mathrm{Tree} \,.\, \mathrm{T}(t) \supset \forall c : \mathrm{Rou}, \vec{l} : [[\mathrm{Int}]] \,.\, \mathrm{rep}(c, \vec{l}) \supset \mathrm{rep}(\mathrm{br}\ t\ c, \mathrm{zip}(\mathrm{niv}\ t)\vec{l})$$

Proof of Lemma 1: Induction on $\mathrm{T}(t)$.

Proof of the Theorem: Recall:

$$\mathrm{bfMH}\ t = \mathrm{extract}(\mathrm{br}\ t\ \mathit{Over})$$

$$\mathrm{bfspec}\ t = \mathrm{concat}(\mathrm{niv}\ t)$$

Assume $\mathrm{T}(t)$.
Since $\mathrm{rep}(\mathrm{Over}, [])$ holds, we have, by Lemma 2, $\mathrm{rep}(\mathrm{br}\ t\ \mathrm{Over}, \mathrm{niv}\ t)$.
Since $\mathrm{isextractor}(\mathrm{rep}, [], \mathrm{extract})$, by Lemma 1, it follows, by the definition of isextractor, $\mathrm{List}(\mathrm{extract}(\mathit{br}\ t\ \mathrm{Over}))$ and $\mathrm{extract}(\mathrm{br}\ t\ \mathrm{Over}) = \mathrm{concat}(\mathrm{niv} t)$, i.e. $\mathrm{bfMH}\ t = \mathrm{bfspec}\ t$.

Types of predicates and lemmas

$$
\begin{aligned}
\mathcal{T}(\mathbb{N}) &= \mathbf{nat} := \mathbf{fix}(\lambda\alpha : * \,.\, \mathbf{1} \oplus \alpha) \\
\mathcal{T}(\mathrm{List}) &= \mathbf{list} := \mathbf{fix}(\lambda\alpha : * \,.\, \mathbf{1} \oplus \mathbf{nat} \otimes \alpha) \\
\mathcal{T}(\mathrm{T}) &= \mathbf{tree} := \mathbf{fix}(\lambda\alpha : * \,.\, \mathbf{nat} \oplus (\alpha \otimes \mathbf{nat} \otimes \alpha)) \\
\mathcal{T}(\mathrm{isextractor}) &= \lambda\alpha : * \,.\, \alpha \Rightarrow \mathbf{list} \\
\mathcal{T}(\mathrm{rep}) &= \mathbf{fix}(\lambda\alpha : * \,.\, \mathbf{1} \oplus (\mathcal{T}(\mathrm{isextractor})\,\alpha \Rightarrow \mathbf{list})) \\
&= \mathbf{fix}(\lambda\alpha : * \,.\, \mathbf{1} \oplus ((\alpha \Rightarrow \mathbf{list}) \Rightarrow \mathbf{list})) \\
&\triangleq \mathrm{Rou} \\
\mathcal{T}(\mathrm{Lemma1}) &= \mathcal{T}(\mathrm{isextractor})\,\mathcal{T}(\mathrm{rep}) \\
&= \mathcal{T}(\mathrm{rep}) \Rightarrow \mathbf{list} \\
&\triangleq \mathrm{Rou} \to [\mathrm{Int}] \\
\mathcal{T}(\mathrm{Lemma2}) &= \mathbf{tree} \Rightarrow \mathcal{T}(\mathrm{rep}) \Rightarrow \mathcal{T}(\mathrm{rep}) \\
&\triangleq \mathrm{Tree} \to \mathrm{Rou} \to \mathrm{Rou} \\
\mathcal{T}(\mathrm{Theorem}) &= \mathbf{tree} \Rightarrow \mathbf{list} \\
&\triangleq \mathrm{Tree} \to [\mathrm{Int}]
\end{aligned}
$$

Extracting Hofmann's algorithm

$$\begin{aligned}
\mathbf{ep}(\text{Lemma1}) &: \mathcal{T}(\text{rep}) \Rightarrow \mathbf{list} \\
\mathbf{ep}(\text{Lemma1}) &\stackrel{\wedge}{=} \text{extract} \\
\\
\mathbf{ep}(\text{Lemma2}) &: \mathbf{tree} \Rightarrow \mathcal{T}(\text{rep}) \Rightarrow \mathcal{T}(\text{rep}) \\
\mathbf{ep}(\text{Lemma2}) &\stackrel{\wedge}{=} \text{br} \\
\\
\mathbf{ep}(\text{Theorem}) &: \mathbf{tree} \Rightarrow \mathbf{list} \\
\mathbf{ep}(\text{Theorem}) &\stackrel{\wedge}{=} \text{bfMH}
\end{aligned}$$

Question: Which lemma corresponds to unfold?

Non-monotone induction

Given: $\Phi : \mathcal{P}(U) \to \mathcal{P}(U)$ (not necessarily monotone)

Define $\Phi_\alpha \subseteq U$ ($\alpha \in \mathsf{On}$) by transfinite recursion:

$$\Phi^\alpha = \bigcup_{\beta<\alpha} \Phi(\Phi^\beta)$$

$$\mathcal{I}(\Phi) := \bigcup_{\alpha\in\mathsf{On}} \Phi^\alpha$$

$\mathcal{I}(\Phi)$ is a subset of U, called the set defined by *non-monotone induction* from Φ.

Non-monotone induction

Given: $\Phi : \mathcal{P}(U) \to \mathcal{P}(U)$ (not necessarily monotone)

Define $\Phi_\alpha \subseteq U$ ($\alpha \in \mathsf{On}$) by transfinite recursion:

$$\Phi^\alpha = \bigcup_{\beta<\alpha} \Phi(\Phi^\beta)$$

$$\mathcal{I}(\Phi) := \bigcup_{\alpha\in\mathsf{On}} \Phi^\alpha$$

$\mathcal{I}(\Phi)$ is a subset of U, called the set defined by *non-monotone induction* from Φ.

Since the sequence $(\Phi^\alpha)_{\alpha\in\mathsf{On}}$ is monotone, there exist a least ordinal $|\Phi|$ such that $\mathcal{I}(\Phi) = \Phi^{|\Phi|}$.

$|\Phi|$ is called the *closure ordinal* of Φ.

Non-monotone induction

Given: $\Phi : \mathcal{P}(U) \to \mathcal{P}(U)$ (not necessarily monotone)

Define $\Phi_\alpha \subseteq U$ ($\alpha \in \mathsf{On}$) by transfinite recursion:

$$\Phi^\alpha = \bigcup_{\beta<\alpha} \Phi(\Phi^\beta)$$

$$\mathcal{I}(\Phi) := \bigcup_{\alpha\in\mathsf{On}} \Phi^\alpha$$

$\mathcal{I}(\Phi)$ is a subset of U, called the set defined by *non-monotone induction* from Φ.

Since the sequence $(\Phi^\alpha)_{\alpha\in\mathsf{On}}$ is monotone, there exist a least ordinal $|\Phi|$ such that $\mathcal{I}(\Phi) = \Phi^{|\Phi|}$.

$|\Phi|$ is called the *closure ordinal* of Φ.

Non-monotone induction was studied intensively in the 1970s and later (e.g. by Aczel, Richter, Gandy, Moschovakis, Pohlers, Jäger, and many others).

Non-monotone induction in CST, axiomatically

In CST, $\mathcal{P}(U) := U \to o$ where U is some type.

New base types and constants:

On	base type
$<: \mathsf{On} \to \mathsf{On} \to o$	
$\mathrm{iter} : (\mathcal{P}(U) \to \mathcal{P}(U)) \to \mathsf{On} \to \mathcal{P}(U)$	$\mathrm{iter}\,\Phi\,\alpha$ represents Φ^α
$\mathcal{I} : (\mathcal{P}(U) \to \mathcal{P}(U)) \to \mathcal{P}(U)$	

Axioms:

$$\forall \Phi : \mathcal{P}(U) \to \mathcal{P}(U), \alpha : \mathsf{On}, x : U\,.\ \mathrm{iter}\ \Phi\ \alpha\ x\ \supset\subset\ \exists \beta < \alpha\,.\,\Phi\ (\mathrm{iter}\ \Phi\ \beta)\ x$$

$$\forall \Phi : \mathcal{P}(U) \to \mathcal{P}(U), x : U\,.\,\mathcal{I}\ \Phi\ x\ \supset\subset\ \exists \alpha : \mathsf{On}\,.\,\mathrm{iter}\ \Phi\ \alpha\ x$$

Axioms for $(\mathsf{On}, <)$

Non-monotone induction in CST, axiomatically

In CST, $\mathcal{P}(U) := U \to o$ where U is some type.

New base types and constants:

On	base type
$<: \mathsf{On} \to \mathsf{On} \to o$	
$\mathrm{iter} : (\mathcal{P}(U) \to \mathcal{P}(U)) \to \mathsf{On} \to \mathcal{P}(U)$	$\mathrm{iter}\,\Phi\,\alpha$ represents Φ^α
$\mathcal{I} : (\mathcal{P}(U) \to \mathcal{P}(U)) \to \mathcal{P}(U)$	

Axioms:

$$\forall \Phi : \mathcal{P}(U) \to \mathcal{P}(U), \alpha : \mathsf{On}, x : U\,.\ \mathrm{iter}\ \Phi\ \alpha\ x\ \supset\subset\ \exists \beta < \alpha\,.\,\Phi\ (\mathrm{iter}\ \Phi\ \beta)\ x$$

$$\forall \Phi : \mathcal{P}(U) \to \mathcal{P}(U), x : U\,.\,\mathcal{I}\ \Phi\ x\ \supset\subset\ \exists \alpha : \mathsf{On}\,.\,\mathrm{iter}\ \Phi\ \alpha\ x$$

Axioms for $(\mathsf{On}, <)$

Can $\mathcal{I}$ be defined in plain CST, without reference to ordinals?

Non-monotone induction in CST, axiomatically

In CST, $\mathcal{P}(U) := U \to o$ where U is some type.

New base types and constants:

On	base type
$<: \mathsf{On} \to \mathsf{On} \to o$	
$\mathrm{iter} : (\mathcal{P}(U) \to \mathcal{P}(U)) \to \mathsf{On} \to \mathcal{P}(U)$	$\mathrm{iter}\,\Phi\,\alpha$ represents Φ^α
$\mathcal{I} : (\mathcal{P}(U) \to \mathcal{P}(U)) \to \mathcal{P}(U)$	

Axioms:

$$\forall \Phi : \mathcal{P}(U) \to \mathcal{P}(U), \alpha : \mathsf{On}, x : U\,.\ \mathrm{iter}\ \Phi\ \alpha\ x\ \supset\subset\ \exists \beta < \alpha\,.\,\Phi\ (\mathrm{iter}\ \Phi\ \beta)\ x$$

$$\forall \Phi : \mathcal{P}(U) \to \mathcal{P}(U), x : U\,.\,\mathcal{I}\ \Phi\ x\ \supset\subset\ \exists \alpha : \mathsf{On}\,.\,\mathrm{iter}\ \Phi\ \alpha\ x$$

Axioms for $(\mathsf{On}, <)$

Can $\mathcal{I}$ be defined in plain CST, without reference to ordinals?

Wolfram Pohlers, 2008: "The definition of the fixed-point of a non-monotone operator needs ordinals, and is therefore much harder to express in second order number theory."

Defining non-monotone induction using higher types

Given: $\Phi : \mathcal{P}(U) \to \mathcal{P}(U)$.

Set $\mathcal{P}^2(U) := \mathcal{P}(\mathcal{P}(U)) = (U \to o) \to o$.

$$\begin{array}{rcl} \tilde{\Phi} & : & \mathcal{P}^2(U) \to \mathcal{P}^2(U) \\ \tilde{\Phi} & := & \lambda S : \mathcal{P}^2(U) \,.\, \lambda X : \mathcal{P}(U) \,. \\ & & \quad (\exists C \in \mathsf{chain}(S) \,.\, X \approx_{\mathcal{P}(U)} \bigcup C) \vee \\ & & \quad (\exists Y \in S \,.\, X \approx_{\mathcal{P}(U)} Y \cup \Phi\, Y) \end{array}$$

where '$C \in \mathsf{chain}(S)$' stands for

$$C \subseteq_{\mathcal{P}^2(U)} S \wedge \forall X, Y \in S \,.\, X \subseteq_{\mathcal{P}(U)} Y \vee Y \subseteq_{\mathcal{P}(U)} X.$$

Informally, $\tilde{\Phi}\, S = \{\bigcup C \mid C \in \mathsf{chain}(S)\} \cup \{Y \cup \Phi\, Y \mid Y \in S\}$.

Theorem

$\tilde{\Phi}$ *is monotone and* $\mathcal{I}(\Phi) = \bigcup \mu_{\mathcal{P}^2(U)}\, \tilde{\Phi}$.

Summary of Lecture 2

- Examples of

  nested,
  non-strictly positive,
  and higher-type

  induction.

Summary of Lecture 2

- Examples of

  nested,
  non-strictly positive,
  and higher-type

  induction.
- Extracting tree representations of continuous functions.

Summary of Lecture 2

- Examples of

  nested,
  non-strictly positive,
  and higher-type

  induction.
- Extracting tree representations of continuous functions.
- Extracting integration from an equational specification.

Summary of Lecture 2

- Examples of

  nested,
  non-strictly positive,
  and higher-type

  induction.
- Extracting tree representations of continuous functions.
- Extracting integration from an equational specification.
- Extracting Hofmann's breadth-first search algorithm, dand explaining the use of non-strictly positive data types.

Summary of Lecture 2

- Examples of

  nested,
  non-strictly positive,
  and higher-type

  induction.
- Extracting tree representations of continuous functions.
- Extracting integration from an equational specification.
- Extracting Hofmann's breadth-first search algorithm, dand explaining the use of non-strictly positive data types.
- Defining non-monotone induction without ordinals.