Bar recursive extensions of Gödel's system T

Thomas Powell

Institut des Hautes Études Scientifiques

PLUME Seminar, ENS Lyon

9 January 2014

Outline

1. Introduction
   - System T
   - Extensions of T
2. Two classes of bar recursive functionals
   - Explicit bar recursion
   - Implicit bar recursion
3. The open recursive functionals
   - The BBC functional
   - Open recursion
4. Concluding remarks
   - Summary
   - Programs from proofs

Gödel's system T

First conceived in 1941: an early type theory. Now standard calculus of primitive recursion in all finite types. In a nutshell:

- Types $:= \mathbb{B} \mid \mathbb{N} \mid \rho \times \tau \mid \rho^* \mid \rho \to \tau$
- Basic constants and axioms i.e. 0, S, combinators for λ-abstraction...
- Primitive recursion for each type:

$$\mathrm{R}_\rho(n) \stackrel{\rho}{=} \begin{cases} y & \text{if } n = 0 \\ z_{n-1}(\mathrm{R}_\rho(n-1)) & \text{otherwise} \end{cases}$$

Stronger than ordinary primitive recursion! $\mathrm{R}_{\mathbb{N}\to\mathbb{N}}$ defines Ackermann function.

Higher-type equality treated as fully extensional

Computational interpretation of subsystems of mathematics

Functional interpretation of Peano arithmetic (Gödel 1958)

$$\left.\begin{array}{rcl} \text{Classical logic} & \mapsto & \lambda\text{-calculus} \\ \text{Induction} & \mapsto & \text{primitive recursion } \mathsf{R}_\rho \end{array}\right\} \text{System T}$$

Computational interpretation of subsystems of mathematics

Functional interpretation of Peano arithmetic (Gödel 1958)

$$\left.\begin{array}{rcl} \text{Classical logic} & \mapsto & \lambda\text{-calculus} \\ \text{Induction} & \mapsto & \text{primitive recursion } \mathsf{R}_\rho \end{array}\right\} \text{System T}$$

Functional interpretation of classical analysis (Spector 1962)

$$\text{Countable choice} \;\mapsto\; \text{Spector's bar recursion} \;\Big\}\; \text{Extension of T}$$

Bar recursion

Generalisation of primitive recursion to well-founded trees T:

$$\mathrm{B}_{\rho,\tau}(s^{\rho^*}) := \begin{cases} Y_s & \text{if } s \text{ is a leaf of } T \\ Z_s(\lambda x \,.\, \mathrm{B}_{\rho,\tau}(s * x)) & \text{otherwise} \end{cases}$$

Much stronger than even Gödel primitive recursion!

Bar recursion

Generalisation of primitive recursion to well-founded trees T:

$$\mathrm{B}_{\rho,\tau}(s^{\rho^*}) := \begin{cases} Y_s & \text{if } s \text{ is a leaf of } T \\ Z_s(\lambda x \,.\, \mathrm{B}_{\rho,\tau}(s * x)) & \text{otherwise} \end{cases}$$

Much stronger than even Gödel primitive recursion!

Fact (Escardó/Oliva/Powell 2011)

Gödel primitive recursion equivalent to finite form of $\mathrm{B}_{\rho,\tau}$ with branches of fixed length:

$$\mathrm{B}^{\mathrm{fin}}_{\rho,\tau}(s^{\rho^*}) := \begin{cases} Y_s & \text{if } |s| \geq n \\ Z_s(\lambda x \,.\, \mathrm{B}^{\mathrm{fin}}_{\rho,\tau}(s * x)) & \text{otherwise} \end{cases}$$

Computational interpretation of subsystems of mathematics

Functional interpretation of Peano arithmetic (Gödel 1958)

$$\left.\begin{array}{rcl} \text{Classical logic} & \mapsto & \lambda\text{-calculus} \\ \text{Induction} & \mapsto & \text{primitive recursion } \mathsf{R}_\rho \end{array}\right\} \text{System T}$$

Computational interpretation of subsystems of mathematics

Functional interpretation of Peano arithmetic (Gödel 1958)

$$\left.\begin{array}{rcl} \text{Classical logic} & \mapsto & \lambda\text{-calculus} \\ \text{Induction} & \mapsto & \text{primitive recursion } \mathsf{R}_\rho \end{array}\right\} \text{System T}$$

Functional interpretation of classical analysis (Spector 1962)

$$\left.\text{Dependent choice} \quad \mapsto \quad \text{Spector's bar recursion}\right\} \text{Extension of T}$$

Computational interpretation of subsystems of mathematics

Functional interpretation of Peano arithmetic (Gödel 1958)

$$\left.\begin{array}{rcl} \text{Classical logic} & \mapsto & \lambda\text{-calculus} \\ \text{Induction} & \mapsto & \text{primitive recursion } \mathsf{R}_\rho \end{array}\right\} \text{System T}$$

Functional interpretation of classical analysis (Spector 1962)

$$\left.\text{Dependent choice} \quad \mapsto \quad \text{Spector's bar recursion}\right\} \text{Extension of T}$$

Key point

This is a general framework that encompasses a variety of computational interpretations of analysis.

Computational interpretation of subsystems of mathematics

Realizability interpretation of Peano arithmetic

$$\left.\begin{array}{rcl} \text{Classical logic} & \mapsto & \lambda\text{-calculus} \\ \text{Induction} & \mapsto & \text{primitive recursion } \mathrm{R}_\rho \end{array}\right\} \text{System T}$$

Realizability interpretation of classical analysis (Berardi et al. 1998)

$$\left.\text{Countable choice} \mapsto \text{BBC functional}\right\} \text{Extension of T}$$

Key point

This is a general framework that encompasses a variety of computational interpretations of analysis.

Computational interpretation of subsystems of mathematics

Realizability interpretation of Peano arithmetic

$$\left.\begin{array}{rcl} \text{Classical logic} & \mapsto & \lambda\text{-calculus} \\ \text{Induction} & \mapsto & \text{primitive recursion } \mathsf{R}_\rho \end{array}\right\} \text{System T}$$

Realizability interpretation of classical analysis (Berger/Oliva 2005)

$$\text{Dependent choice} \quad \mapsto \quad \text{modified bar recursion} \; \} \; \text{Extension of T}$$

Key point

This is a general framework that encompasses a variety of computational interpretations of analysis.

Computational interpretation of subsystems of mathematics

Realizability interpretation of Peano arithmetic

$$\left.\begin{array}{rcl} \text{Classical logic} & \mapsto & \lambda\text{-calculus} \\ \text{Induction} & \mapsto & \text{primitive recursion } \mathsf{R}_\rho \end{array}\right\} \text{System T}$$

Realizability interpretation of classical analysis (Berger 2004)

$$\left.\text{Open induction} \quad \mapsto \quad \text{open recursion}\right\} \text{Extension of T}$$

Key point

This is a general framework that encompasses a variety of computational interpretations of analysis.

Quick aside: Extensions of system T in other contexts

Higher-type computability theory (Gandy/Hyland 1977). In the type structure of continuous functionals, the bar-recursive functional Γ defined by

$$\Gamma(s^{\mathbb{N}^*}) \stackrel{\mathbb{N}}{=} Z(s * 0 * \lambda n \, . \, \Gamma(s * (n+1)))$$

has a recursive associate but is not Kleene computable, even in the FAN functional (more on this later!).

Quick aside: Extensions of system T in other contexts

Higher-type computability theory (Gandy/Hyland 1977). In the type structure of continuous functionals, the bar-recursive functional Γ defined by

$$\Gamma(s^{\mathbb{N}^*}) \stackrel{\mathbb{N}}{=} Z(s * 0 * \lambda n\,.\,\Gamma(s * (n+1)))$$

has a recursive associate but is not Kleene computable, even in the FAN functional (more on this later!).

Game theory (Escardo/Oliva 2010). The product of selection functions ips defined by

$$\mathrm{ips}(s^{\rho^*}) \stackrel{\mathbb{N}}{=} s \mathbin{@} \lambda n\,.\,\varepsilon_n(\lambda x\,.\,q(\mathrm{ips}(t_n * x)))$$

for $t_n = \langle \mathrm{ips}(s)_0, \ldots, \mathrm{ips}(s)_{n-1} \rangle$ computes optimal strategies in a class of unbounded sequential games.

Summary

We have a large collection of recursively defined extensions of the primitive recursive functions, including (but not confined to)

- Spector's bar recursion
- Gandy-Hyland Γ functional
- Modified bar recursion
- Symmetric BBC functional
- Open and update recursion
- Products of selection functions

These are important in proof theory, computability theory, game theory etc.

Key question

How do these forms of recursion relate to one another? In particular, **which ones are primitive recursively equivalent**?

Key question

How do these forms of recursion relate to one another? In particular, **which ones are primitive recursively equivalent**?

Why do we care about this?

- Establishing relationship between relevant extensions of system T gives us an insight into how programs extracted from proofs compare.

Key question

How do these forms of recursion relate to one another? In particular, **which ones are primitive recursively equivalent**?

Why do we care about this?

- Establishing relationship between relevant extensions of system T gives us an insight into how programs extracted from proofs compare.
- Extensions of T are important objects in mathematical logic, and it's always good to know things about important classes of objects.

Key question

How do these forms of recursion relate to one another? In particular, **which ones are primitive recursively equivalent**?

Why do we care about this?

- Establishing relationship between relevant extensions of system T gives us an insight into how programs extracted from proofs compare.
- Extensions of T are important objects in mathematical logic, and it's always good to know things about important classes of objects.
- An elegant mathematical problem...

Outline

Spector's general form of bar recursion

Has defining axiom (for arbitrary types ρ, τ):

$$\mathrm{GBR}^{\phi,r,\varphi}_{\rho,\tau}(s^{\rho^*}) \stackrel{\tau}{=} \begin{cases} r(s) & \text{if } \varphi(s * \mathbf{0}) < |s| \\ \phi_s(\lambda x^\rho \,.\, \mathrm{GBR}(s * x)) & \text{otherwise} \end{cases}$$

where $\varphi \colon \rho^{\mathbb{N}} \to \mathbb{N}$. Recursion over an *explicitly* defined tree: s a leaf iff $\varphi(s * \mathbf{0}) < |s|$.

Spector's general form of bar recursion

Has defining axiom (for arbitrary types ρ, τ):

$$\mathrm{GBR}^{\phi,r,\varphi}_{\rho,\tau}(s^{\rho^*}) \stackrel{\tau}{=} \begin{cases} r(s) & \text{if } \varphi(s * \mathbf{0}) < |s| \\ \phi_s(\lambda x^\rho \,.\, \mathrm{GBR}(s * x)) & \text{otherwise} \end{cases}$$

where $\varphi\colon \rho^{\mathbb{N}} \to \mathbb{N}$. Recursion over an *explicitly* defined tree: s a leaf iff $\varphi(s * \mathbf{0}) < |s|$.

In any model of GBR, this tree must be well-founded i.e. any infinite sequence $\alpha\colon \rho^{\mathbb{N}}$ must satisfy

$$\exists N[\varphi([\alpha](N) * 0) < N].$$

Clearly not the case in full set-theoretic model: so in particular GBR not primitive recursive.

Models of GBR

Bar recursion typically requires some kind of continuity axiom*:

$$\mathsf{Cont} \;:\; \forall \varphi^{\rho^{\mathbb{N}} \to \mathbb{N}}, \alpha^{\rho^{\mathbb{N}}} \exists N \forall \beta ([\alpha](N) = [\beta](N) \to \varphi(\alpha) = \varphi(\beta))$$

i.e. functionals of type $\rho^{\mathbb{N}} \to \mathbb{N}$ only require a finite amount of information.

* But not always: strongly majorizable functionals a model of GBR (Bezem 1985)

Models of GBR

Bar recursion typically requires some kind of continuity axiom*:

$$\mathrm{Cont}\ :\ \forall\varphi^{\rho^{\mathbb{N}}\to\mathbb{N}},\alpha^{\rho^{\mathbb{N}}}\exists N\forall\beta([\alpha](N)=[\beta](N)\to\varphi(\alpha)=\varphi(\beta))$$

i.e. functionals of type $\rho^{\mathbb{N}}\to\mathbb{N}$ only require a finite amount of information.

$$\mathrm{GBR}(\langle\rangle)\text{ n.d.}\Rightarrow\mathrm{GBR}(\langle x_0\rangle)\text{ n.d.}\Rightarrow\mathrm{GBR}(\langle x_0,x_1\rangle)\text{ n.d.}\Rightarrow\dots$$

$$\text{dependent choice}\ \Rightarrow\exists\alpha\forall n\mathrm{GBR}([\alpha](n))\text{ n.d.}$$

But for $N'=\max\{N,\varphi(\alpha)+1\}$ have $\varphi([\alpha](N'))=\varphi(\alpha)<N'$

$$\text{therefore }\mathrm{GBR}([\alpha](N'))=r([\alpha](N'))\text{ defined (contradiction!).}$$

* But not always: strongly majorizable functionals a model of GBR (Bezem 1985)

Explicit product of selection functions EPS/Spector's weak bar recursion

Has defining axiom

$$\mathrm{EPS}^{\varepsilon,q,\varphi}_{\rho}(s) \overset{\rho^{\mathbb{N}}}{=} \begin{cases} \mathbf{0} & \text{if } \varphi(s * \mathbf{0}) < |s| \\ a_s * \mathrm{EPS}(s * a_s) & \text{otherwise} \end{cases}$$

where $a_s = \varepsilon_s(\lambda x \,.\, q(s * x * \mathrm{EPS}(s * x)))$.

Explicit product of selection functions EPS/Spector's weak bar recursion

Has defining axiom

$$\mathrm{EPS}_{\rho}^{\varepsilon,q,\varphi}(s) \stackrel{\rho^{\mathbb{N}}}{=} \begin{cases} \mathbf{0} & \text{if } \varphi(s * \mathbf{0}) < |s| \\ a_s * \mathrm{EPS}(s * a_s) & \text{otherwise} \end{cases}$$

where $a_s = \varepsilon_s(\lambda x\ .\ q(s * x * \mathrm{EPS}(s * x)))$.

- Form of Spector's bar recursion most commonly encountered in proof theory.
- A special case of $\mathrm{GBR}_{\rho,\tau}$ for $\tau = \rho^{\mathbb{N}}$.
- Sufficient to solve Dialectica interpretation of double negation shift, so $\mathrm{PA} + \mathrm{AC} \mapsto \mathrm{T} + \mathrm{EPS}$.
- Also naturally computes optimal strategies in a class of unbounded sequential games (Oliva/Escardo 2010).

Tidying up explicit bar recursion

Until recently, most of the (many!) known variants of Spector's bar recursion were known to be equivalent to either GBR or EPS (thanks mainly to Luckhardt 1973, Bezem 1988, Escardó/Oliva 2010).

Tidying up explicit bar recursion

Until recently, most of the (many!) known variants of Spector's bar recursion were known to be equivalent to either GBR or EPS (thanks mainly to Luckhardt 1973, Bezem 1988, Escardó/Oliva 2010).

Theorem (Oliva/P. 2012)

EPS *primitive recursively defines* GBR, *and so* T + *'weak' bar recursion is actually as strong as* T + *general bar recursion*

Tidying up explicit bar recursion

Until recently, most of the (many!) known variants of Spector's bar recursion were known to be equivalent to either GBR or EPS (thanks mainly to Luckhardt 1973, Bezem 1988, Escardó/Oliva 2010).

Theorem (Oliva/P. 2012)

EPS *primitive recursively defines* GBR, *and so* T + *'weak' bar recursion is actually as strong as* T + *general bar recursion*

Corollary. Essentially all known extensions of system T based on a variant of Spector's bar recursion are equivalent.

Pause for a moment...

Definition

F primitive recursively defines G over Δ if for each ρ there exists a closed term $t \in \mathsf{T}$ and type ρ' such that $t(\mathsf{F}_{\rho'})$ satisfies the defining axiom of G_ρ, provably in $\mathsf{T} + \Delta$.

Pause for a moment...

Definition

F primitive recursively defines G over Δ if for each ρ there exists a closed term $t \in \mathrm{T}$ and type ρ' such that $t(\mathrm{F}_{\rho'})$ satisfies the defining axiom of G_ρ, provably in $\mathrm{T} + \Delta$.

e.g. GBR defines EPS:

$$\mathrm{GBR}^{\phi,r,\varphi}_{\rho,\rho^{\mathbb{N}}}(s) = \begin{cases} r(s) & \text{if } \varphi(s * \mathbf{0}) < |s| \\ \phi_s(\lambda x^\rho \, . \, \mathrm{GBR}(s * x)) & \text{otherwise.} \end{cases}$$

$\Rightarrow$ (system T)

$$t(\mathrm{GBR})^{\varepsilon,q,\varphi}_{\rho}(s) = \begin{cases} \mathbf{0} & \text{if } \varphi(s * \mathbf{0}) < |s| \\ a_s * t(\mathrm{GBR})(s * a_s) & \text{otherwise} \end{cases}$$

Warning: Kohlenbach's bar recursion

Not all explicit forms of bar recursion are equivalent. In his thesis (1990) Kohlenbach considers the following, novel variant of bar recursion:

$$\mathrm{KBR}^{\phi,r,\varphi}_{\rho,\tau}(s) \overset{\tau}{=} \begin{cases} r(s) & \text{if } \varphi(s * \mathbf{0}) = \varphi(s * \mathbf{1}) \\ \phi_s(\lambda x \,.\, \mathrm{KBR}(s * x)) & \text{otherwise} \end{cases}$$

KBR defines GBR, but does not exist in the majorizable functionals, and therefore is not conversely definable from GBR (else we'd contradict Bezem 1985).

Warning: Kohlenbach's bar recursion

Not all explicit forms of bar recursion are equivalent. In his thesis (1990) Kohlenbach considers the following, novel variant of bar recursion:

$$\mathrm{KBR}^{\phi,r,\varphi}_{\rho,\tau}(s) \stackrel{\tau}{=} \begin{cases} r(s) & \text{if } \varphi(s * \mathbf{0}) = \varphi(s * \mathbf{1}) \\ \phi_s(\lambda x\,.\,\mathrm{KBR}(s * x)) & \text{otherwise} \end{cases}$$

KBR defines GBR, but does not exist in the majorizable functionals, and therefore is not conversely definable from GBR (else we'd contradict Bezem 1985).

Moral: Primitive recursive definability is a subtle property, and slight variants in the defining equations can lead to fundamentally different extensions of system T.

Extensions of T

$$
\begin{tikzcd}
\boxed{\text{Spector bar recursion (GBR, EPS etc.)}} \arrow[d, "\mathcal{S}^\omega \not\models \mathrm{GBR}"] & \boxed{\text{KBR}} \arrow[l, "\mathcal{M}^\omega \not\models \mathrm{KBR}"'] \\
\boxed{\text{Gödel primitive recursion}} &
\end{tikzcd}
$$

Implicit product of selection functions IPS

Has defining axiom

$$\mathrm{IPS}^{\varepsilon,q}_{\rho}(s) \stackrel{\rho^{\mathbb{N}}}{=} s \,@\, \mathrm{IPS}(s * a_s)$$

where @ denotes overwrite and $a_s = \varepsilon_s(\lambda x \,.\, q(\mathrm{IPS}(s * x)))$, and $q \colon \rho^{\mathbb{N}} \to \mathbb{N}$. Can equivalently define via course-of-values recursion as

$$\mathrm{IPS}^{\varepsilon,q}_{\rho}(s) = s \,@\, \lambda n \,.\, \varepsilon_{t_n}(\lambda x \,.\, q(\mathrm{IPS}(t_n * x))).$$

where $t_n = [\mathrm{IPS}(s)](n)$.

Implicit product of selection functions IPS

Has defining axiom

$$\mathrm{IPS}^{\varepsilon,q}_{\rho}(s) \stackrel{\rho^{\mathbb{N}}}{=} s \mathbin{@} \mathrm{IPS}(s * a_s)$$

where $@$ denotes overwrite and $a_s = \varepsilon_s(\lambda x \,.\, q(\mathrm{IPS}(s * x)))$, and $q \colon \rho^{\mathbb{N}} \to \mathbb{N}$. Can equivalently define via course-of-values recursion as

$$\mathrm{IPS}^{\varepsilon,q}_{\rho}(s) = s \mathbin{@} \lambda n \,.\, \varepsilon_{t_n}(\lambda x \,.\, q(\mathrm{IPS}(t_n * x))).$$

where $t_n = [\mathrm{IPS}(s)](n)$.

- Equivalent to modified bar recursion MBR (Ber/Oli 2005), which is in turn based on realizer of (Berardi et al. 1998).
- Solves modified realizability interpretation of double negation shift.

Models of IPS

Unlike GBR, apparently no stopping condition. Because codomain of q has type $\mathbb{N}$, recursion over a tree *implicitly* well-founded by continuity of q.

$$\mathrm{IPS}(\langle\rangle) \text{ n.d.} \Rightarrow \mathrm{IPS}(\langle x_0\rangle) \text{ n.d.} \Rightarrow \mathrm{IPS}(\langle x_0, x_1\rangle) \text{ n.d.} \Rightarrow \ldots$$

$$\text{dependent choice } \Rightarrow \exists\alpha\forall n \mathrm{IPS}([\alpha](n)) \text{ n.d.}$$

Models of IPS

Unlike GBR, apparently no stopping condition. Because codomain of q has type $\mathbb{N}$, recursion over a tree *implicitly* well-founded by continuity of q.

$$\mathrm{IPS}(\langle\rangle) \text{ n.d.} \Rightarrow \mathrm{IPS}(\langle x_0\rangle) \text{ n.d.} \Rightarrow \mathrm{IPS}(\langle x_0, x_1\rangle) \text{ n.d.} \Rightarrow \ldots$$

$$\text{dependent choice } \Rightarrow \exists\alpha\forall n \mathrm{IPS}([\alpha](n)) \text{ n.d.}$$

Let N be point of continuity of q on α. Then

$$\begin{aligned}\mathrm{IPS}([\alpha](N)) &= [\alpha](N) \mathbin{@} \varepsilon_{t_n}(\lambda x\,.\,q(\mathrm{IPS}(t_n * x)))\\ &= [\alpha](N) \mathbin{@} \varepsilon_{t_n}(\lambda x\,.\,q([\alpha](N) \mathbin{@} \mathrm{IPS}(t_n * x)))\\ &= [\alpha](N) \mathbin{@} \varepsilon_{t_n}(\lambda x\,.\,q(\alpha))\end{aligned}$$

which is well-defined.

Implicit is stronger than explicit

Theorem (Berger/Oliva 2005)

IPS *(or equivalently* MBR*) defines* GBR, *but neither* GBR *(nor* KBR*) define* IPS, *over* **any** *theory* Δ *validated by* $\mathcal{C}^{\omega}$.

Implicit is stronger than explicit

Theorem (Berger/Oliva 2005)

IPS *(or equivalently* MBR*) defines* GBR, *but neither* GBR *(nor* KBR*) define* IPS, *over* **any** *theory* Δ *validated by* $\mathcal{C}^\omega$*.*

Proof. (Sketch!)

GBR is S1-S9 computable in $\mathcal{C}^\omega$, but IPS (of lowest type) defines the Gandy/Hyland Γ-functional which is not S1-S9 computable in $\mathcal{C}^\omega$ (even with FAN functional as an oracle). Since computable functionals are closed under primitive recursion, result follows. $\square$

Implicit is stronger than explicit

Theorem (Berger/Oliva 2005)

IPS *(or equivalently* MBR*) defines* GBR, *but neither* GBR *(nor* KBR*) define* IPS, *over* **any** *theory* Δ *validated by* $\mathcal{C}^\omega$.

Proof. (Sketch!)

GBR is S1-S9 computable in $\mathcal{C}^\omega$, but IPS (of lowest type) defines the Gandy/Hyland Γ-functional which is not S1-S9 computable in $\mathcal{C}^\omega$ (even with FAN functional as an oracle). Since computable functionals are closed under primitive recursion, result follows. □

Can use Kleene recursion theorem in S1-S9, so why is GBR computable but not IPS?

- Can define IPS and GBR as fixpoints, but must prove totality.
- Cannot prove totality of IPS because S8 rule requires objects to be total before returning a total value.

Extensions of T

$$
\begin{tikzcd}
\boxed{\text{Implicit bar recursion (IPS, MBR etc.)}} \arrow[d, "\text{S1-S9}\not\models\text{MBR}"] & \\
\boxed{\text{Spector bar recursion (GBR, EPS etc.)}} \arrow[d, "\mathcal{S}^\omega\not\models\text{GBR}"] & \boxed{\text{KBR}} \arrow[l, "\mathcal{M}^\omega\not\models\text{KBR}"'] \\
\boxed{\text{Gödel primitive recursion}} &
\end{tikzcd}
$$

Outline

The Berardi-Bezem-Coquand (BBC) functional

Has defining axiom

$$\mathrm{BBC}_{\rho}^{\varepsilon,q}(u) \stackrel{\mathbb{N}}{=} q(u \mathbin{@} \lambda n\,.\,\varepsilon_n(\lambda x\,.\,\mathrm{BBC}(u \oplus (n,x))))$$

where $u\colon \mathbb{N} \to \rho_{\perp}$ is a partial function and $u \oplus (n,x)$ extension of u with value x at point n.

The Berardi-Bezem-Coquand (BBC) functional

Has defining axiom

$$\mathrm{BBC}^{\varepsilon,q}_{\rho}(u) \stackrel{\mathbb{N}}{=} q(u \,@\, \lambda n \,.\, \varepsilon_n(\lambda x \,.\, \mathrm{BBC}(u \oplus (n,x))))$$

where $u \colon \mathbb{N} \to \rho_{\perp}$ is a partial function and $u \oplus (n,x)$ extension of u with value x at point n.

Compare with ‘simple’ version of IPS given by

$$\mathrm{ips}^{\varepsilon,q}_{\rho}(s) \stackrel{\mathbb{N}}{=} q(s \,@\, \lambda n \,.\, \varepsilon_n(\lambda x \,.\, \mathrm{ips}(t_n * x)))$$

where $t_n = [\mathrm{ips}](n)$.

The Berardi-Bezem-Coquand (BBC) functional

Has defining axiom

$$\mathrm{BBC}_\rho^{\varepsilon,q}(u) \overset{\mathbb{N}}{=} q(u \mathbin{@} \lambda n \,.\, \varepsilon_n(\lambda x \,.\, \mathrm{BBC}(u \oplus (n,x))))$$

where $u\colon \mathbb{N} \to \rho_\perp$ is a partial function and $u \oplus (n,x)$ extension of u with value x at point n.

Compare with 'simple' version of IPS given by

$$\mathrm{ips}_\rho^{\varepsilon,q}(s) \overset{\mathbb{N}}{=} q(s \mathbin{@} \lambda n \,.\, \varepsilon_n(\lambda x \,.\, \mathrm{ips}(t_n * x)))$$

where $t_n = [\mathrm{ips}](n)$.

- ips computes sequentially, BBC computes symmetrically.
- Proposed in (Berardi et al. 1998) as a more direct, efficient computational interpretation of countable choice than Spector's bar recursion.

A closer look at BBC...

Take a finite form where $q(\alpha) = q(\alpha_0, \alpha_1)$.

$$\mathsf{BBC}(u) = q(u \mathbin{@} \lambda n\,.\,\varepsilon_n(\lambda x\,.\,\mathsf{BBC}(u \oplus (n, x))))$$

A closer look at BBC...

Take a finite form where $q(\alpha) = q(\alpha_0, \alpha_1)$.

$$\mathrm{BBC}(u) = q(u \mathbin{@} \lambda n \,.\, \varepsilon_n(\lambda x \,.\, \mathrm{BBC}(u \oplus (n, x))))$$

$\mathrm{BBC}(\emptyset) = q(a, b)$ where

$$a = \varepsilon_0(\lambda x \,.\, \mathrm{BBC}((0, x)))$$

A closer look at BBC...

Take a finite form where $q(\alpha) = q(\alpha_0, \alpha_1)$.

$$\mathrm{BBC}(u) = q(u \mathbin{@} \lambda n \,.\, \varepsilon_n(\lambda x \,.\, \mathrm{BBC}(u \oplus (n, x))))$$

$\mathrm{BBC}(\emptyset) = q(a, b)$ where

$$a = \varepsilon_0(\lambda x \,.\, q(x, \varepsilon_1(\lambda y \,.\, \mathrm{BBC}((0, x) \oplus (1, y)))))$$

A closer look at BBC...

Take a finite form where $q(\alpha) = q(\alpha_0, \alpha_1)$.

$\mathsf{BBC}(u) = q(u \mathbin{@} \lambda n \,.\, \varepsilon_n(\lambda x \,.\, \mathsf{BBC}(u \oplus (n, x))))$

$\mathsf{BBC}(\emptyset) = q(a, b)$ where

$$a = \varepsilon_0(\lambda x \,.\, q(x, \varepsilon_1(\lambda y \,.\, q(x, y))))$$

A closer look at BBC...

Take a finite form where $q(\alpha) = q(\alpha_0, \alpha_1)$.

$$\mathsf{BBC}(u) = q(u \,@\, \lambda n \,.\, \varepsilon_n(\lambda x \,.\, \mathsf{BBC}(u \oplus (n, x))))$$

$\mathsf{BBC}(\emptyset) = q(a, b)$ where

$$a = \varepsilon_0(\lambda x \,.\, q(x, \varepsilon_1(\lambda y \,.\, q(x, y))))$$
$$b = \varepsilon_1(\lambda y \,.\, \mathsf{BBC}((1, y)))$$

A closer look at BBC...

Take a finite form where $q(\alpha) = q(\alpha_0, \alpha_1)$.

$$\mathsf{BBC}(u) = q(u \mathbin{@} \lambda n \,.\, \varepsilon_n(\lambda x \,.\, \mathsf{BBC}(u \oplus (n, x))))$$

$\mathsf{BBC}(\emptyset) = q(a, b)$ where

$$\begin{aligned} a &= \varepsilon_0(\lambda x \,.\, q(x, \varepsilon_1(\lambda y \,.\, q(x, y)))) \\ b &= \varepsilon_1(\lambda y \,.\, q(\varepsilon_0(\lambda x \,.\, \mathsf{BBC}((1, y) \oplus (0, x))), y)) \end{aligned}$$

A closer look at BBC...

Take a finite form where $q(\alpha) = q(\alpha_0, \alpha_1)$.

$$\mathrm{BBC}(u) = q(u \mathbin{@} \lambda n \,.\, \varepsilon_n(\lambda x \,.\, \mathrm{BBC}(u \oplus (n, x))))$$

$\mathrm{BBC}(\emptyset) = q(a, b)$ where

$$\begin{aligned} a &= \varepsilon_0(\lambda x \,.\, q(x, \varepsilon_1(\lambda y \,.\, q(x, y)))) \\ b &= \varepsilon_1(\lambda y \,.\, q(\varepsilon_0(\lambda x \,.\, q(x, y)), y)) \end{aligned}$$

A closer look at BBC...

Take a finite form where $q(\alpha) = q(\alpha_0, \alpha_1)$.

$$\mathrm{BBC}(u) = q(u \mathbin{@} \lambda n\,.\,\varepsilon_n(\lambda x\,.\,\mathrm{BBC}(u \oplus (n, x))))$$

$\mathrm{BBC}(\emptyset) = q(a, b)$ where

$$\begin{aligned} a &= \varepsilon_0(\lambda x\,.\,q(x, \varepsilon_1(\lambda y\,.\,q(x, y)))) \\ b &= \varepsilon_1(\lambda y\,.\,q(\varepsilon_0(\lambda x\,.\,q(x, y)), y)) \end{aligned}$$

$$\mathrm{ips}(s) = q(s \mathbin{@} \lambda n\,.\,\varepsilon_n(\lambda x\,.\,\mathrm{ips}(t_n * x)))$$

A closer look at BBC...

Take a finite form where $q(\alpha) = q(\alpha_0, \alpha_1)$.

$\mathrm{BBC}(u) = q(u \mathbin{@} \lambda n \,.\, \varepsilon_n(\lambda x \,.\, \mathrm{BBC}(u \oplus (n, x))))$

$\mathrm{BBC}(\emptyset) = q(a, b)$ where

$$a = \varepsilon_0(\lambda x \,.\, q(x, \varepsilon_1(\lambda y \,.\, q(x, y))))$$
$$b = \varepsilon_1(\lambda y \,.\, q(\varepsilon_0(\lambda x \,.\, q(x, y)), y))$$

$\mathrm{ips}(s) = q(s \mathbin{@} \lambda n \,.\, \varepsilon_n(\lambda x \,.\, \mathrm{ips}(t_n * x)))$

$\mathrm{ips}(\langle\rangle) = q(a, b)$ where

$$a = \varepsilon_0(\lambda x \,.\, \mathrm{ips}(\langle x \rangle))$$

A closer look at BBC...

Take a finite form where $q(\alpha) = q(\alpha_0, \alpha_1)$.

$$\mathrm{BBC}(u) = q(u \mathbin{@} \lambda n \,.\, \varepsilon_n(\lambda x \,.\, \mathrm{BBC}(u \oplus (n, x))))$$

$\mathrm{BBC}(\emptyset) = q(a, b)$ where

$$a = \varepsilon_0(\lambda x \,.\, q(x, \varepsilon_1(\lambda y \,.\, q(x, y))))$$
$$b = \varepsilon_1(\lambda y \,.\, q(\varepsilon_0(\lambda x \,.\, q(x, y)), y))$$

$$\mathrm{ips}(s) = q(s \mathbin{@} \lambda n \,.\, \varepsilon_n(\lambda x \,.\, \mathrm{ips}(t_n * x)))$$

$\mathrm{ips}(\langle\rangle) = q(a, b)$ where

$$a = \varepsilon_0(\lambda x \,.\, q(x, \varepsilon_1(\lambda y \,.\, \mathrm{ips}(\langle x, y\rangle))))$$

A closer look at BBC...

Take a finite form where $q(\alpha) = q(\alpha_0, \alpha_1)$.

$\mathrm{BBC}(u) = q(u \,@\, \lambda n \,.\, \varepsilon_n(\lambda x \,.\, \mathrm{BBC}(u \oplus (n, x))))$

$\mathrm{BBC}(\emptyset) = q(a, b)$ where

$$a = \varepsilon_0(\lambda x \,.\, q(x, \varepsilon_1(\lambda y \,.\, q(x, y))))$$
$$b = \varepsilon_1(\lambda y \,.\, q(\varepsilon_0(\lambda x \,.\, q(x, y)), y))$$

$\mathrm{ips}(s) = q(s \,@\, \lambda n \,.\, \varepsilon_n(\lambda x \,.\, \mathrm{ips}(t_n * x)))$

$\mathrm{ips}(\langle\rangle) = q(a, b)$ where

$$a = \varepsilon_0(\lambda x \,.\, q(x, \varepsilon_1(\lambda y \,.\, q(x, y))))$$

A closer look at BBC...

Take a finite form where $q(\alpha) = q(\alpha_0, \alpha_1)$.

$\mathrm{BBC}(u) = q(u \mathbin{@} \lambda n \,.\, \varepsilon_n(\lambda x \,.\, \mathrm{BBC}(u \oplus (n, x))))$

$\mathrm{BBC}(\emptyset) = q(a, b)$ where

$$a = \varepsilon_0(\lambda x \,.\, q(x, \varepsilon_1(\lambda y \,.\, q(x, y))))$$
$$b = \varepsilon_1(\lambda y \,.\, q(\varepsilon_0(\lambda x \,.\, q(x, y)), y))$$

$\mathrm{ips}(s) = q(s \mathbin{@} \lambda n \,.\, \varepsilon_n(\lambda x \,.\, \mathrm{ips}(t_n * x)))$

$\mathrm{ips}(\langle\rangle) = q(a, b)$ where

$$a = \varepsilon_0(\lambda x \,.\, q(x, \varepsilon_1(\lambda y \,.\, q(x, y))))$$
$$b = \varepsilon_1(\lambda y \,.\, \mathrm{ips}(\langle a, y\rangle))$$

A closer look at BBC...

Take a finite form where $q(\alpha) = q(\alpha_0, \alpha_1)$.

$\mathrm{BBC}(u) = q(u \mathbin{@} \lambda n \,.\, \varepsilon_n(\lambda x \,.\, \mathrm{BBC}(u \oplus (n, x))))$

$\mathrm{BBC}(\emptyset) = q(a, b)$ where

$$\begin{aligned} a &= \varepsilon_0(\lambda x \,.\, q(x, \varepsilon_1(\lambda y \,.\, q(x, y)))) \\ b &= \varepsilon_1(\lambda y \,.\, q(\varepsilon_0(\lambda x \,.\, q(x, y)), y)) \end{aligned}$$

$\mathrm{ips}(s) = q(s \mathbin{@} \lambda n \,.\, \varepsilon_n(\lambda x \,.\, \mathrm{ips}(t_n * x)))$

$\mathrm{ips}(\langle\rangle) = q(a, b)$ where

$$\begin{aligned} a &= \varepsilon_0(\lambda x \,.\, q(x, \varepsilon_1(\lambda y \,.\, q(x, y)))) \\ b &= \varepsilon_1(\lambda y \,.\, q(a, y)) \end{aligned}$$

The difficulty understanding BBC

BBC functional a beautiful mathematical object: gives an elegant, symmetric computational interpretation to the axiom of choice.

The difficulty understanding BBC

BBC functional a beautiful mathematical object: gives an elegant, symmetric computational interpretation to the axiom of choice.

- But is it more efficient? Each entry has a seperate tree of recursive calls.

The difficulty understanding BBC

BBC functional a beautiful mathematical object: gives an elegant, symmetric computational interpretation to the axiom of choice.

- But is it more efficient? Each entry has a seperate tree of recursive calls.
- How do we give a transparent proof of totality/correctness of BBC in a standard domain-theoretic framework? (Original paper gives BBC non-standard intentional properties)

The difficulty understanding BBC

BBC functional a beautiful mathematical object: gives an elegant, symmetric computational interpretation to the axiom of choice.

- But is it more efficient? Each entry has a seperate tree of recursive calls.
- How do we give a transparent proof of totality/correctness of BBC in a standard domain-theoretic framework? (Original paper gives BBC non-standard intentional properties)
- Seemingly no obvious game semantics in sense of Escardo/Oliva like bar recursion.

The difficulty understanding BBC

BBC functional a beautiful mathematical object: gives an elegant, symmetric computational interpretation to the axiom of choice.

- But is it more efficient? Each entry has a seperate tree of recursive calls.
- How do we give a transparent proof of totality/correctness of BBC in a standard domain-theoretic framework? (Original paper gives BBC non-standard intentional properties)
- Seemingly no obvious game semantics in sense of Escardo/Oliva like bar recursion.
- Not immediate how corresponding extension of T relates to those based on bar recursion.

BBC is stronger than IPS

Theorem (P. 2013)

BBC primitive recursively defines IPS *over* Cont + QF-BI.

BBC is stronger than IPS

Theorem (P. 2013)

BBC primitive recursively defines IPS *over* Cont + QF-BI.

Basic idea. Use $\mathrm{BBC}_{\rho_\perp^{\mathbb{N}}}$, which computes an infinite matrix and allows us to store information about recursive calls, eliminating independence. Shift to next column whenever making a recursive call.

$$\mathrm{BBC}(\emptyset) = q \begin{pmatrix} \vdots & \vdots & \\ \alpha_{0,1} & \alpha_{1,1} & \cdots \\ \underbrace{\alpha_{0,0}}_{\mathrm{IPS}(\langle\rangle)} & \alpha_{1,0} & \cdots \end{pmatrix}$$

Will not go into details! Key point is that BBC somehow 'contains' implicit bar recursion

Extensions of T

$$
\begin{tikzcd}
\boxed{\text{BBC functional}} \arrow[d] & \\
\boxed{\text{Implicit bar recursion (IPS, MBR etc.)}} \arrow[d, "\text{S1-S9} \not\models \text{MBR}"] & \\
\boxed{\text{Spector bar recursion (GBR, EPS etc.)}} \arrow[d, "\mathcal{S}^\omega \not\models \text{GBR}"] & \boxed{\text{KBR}} \arrow[l, "\mathcal{M}^\omega \not\models \text{KBR}"'] \\
\boxed{\text{Gödel primitive recursion}} &
\end{tikzcd}
$$

Open induction: A way of reasoning about BBC

Suppose that $<$ is a decidable, well-founded relation on ρ. Have well-founded induction over $<$ for arbitrary A

$$\mathsf{TI}_{(\rho,<)} \ : \ \forall x^{\rho}(\forall y < x A(y) \to A(x)) \to \forall x A(x).$$

Open induction: A way of reasoning about BBC

Suppose that $<$ is a decidable, well-founded relation on ρ. Have well-founded induction over $<$ for arbitrary A

$$\mathsf{TI}_{(\rho,<)} \ : \ \forall x^\rho(\forall y < xA(y) \to A(x)) \to \forall x A(x).$$

Can lift $<$ to a lexicographic ordering $<_{\mathrm{lex}}$ on sequences $\rho^{\mathbb{N}}$, where $\alpha <_{\mathrm{lex}} \beta :\equiv \exists n([\alpha](n) = [\beta](n) \wedge \alpha(n) < \beta(n))$. Open induction is induction over $<_{\mathrm{lex}}$:

$$\mathsf{OI}_{(\rho,<)} \ : \ \forall \alpha^{\rho^{\mathbb{N}}}(\forall \beta <_{\mathrm{lex}} \alpha O(\beta) \to O(\alpha)) \to \forall \alpha O(\alpha).$$

However, since $<_{\mathrm{lex}}$ is neither decidable nor well-founded, must restrict O to being an *open formula*.

Open induction: A way of reasoning about BBC

Suppose that $<$ is a decidable, well-founded relation on ρ. Have well-founded induction over $<$ for arbitrary A

$$\mathsf{TI}_{(\rho,<)} : \forall x^\rho(\forall y < xA(y) \to A(x)) \to \forall xA(x).$$

Can lift $<$ to a lexicographic ordering $<_{\mathrm{lex}}$ on sequences $\rho^{\mathbb{N}}$, where $\alpha <_{\mathrm{lex}} \beta :\equiv \exists n([\alpha](n) = [\beta](n) \wedge \alpha(n) < \beta(n))$. Open induction is induction over $<_{\mathrm{lex}}$:

$$\mathsf{OI}_{(\rho,<)} : \forall \alpha^{\rho^{\mathbb{N}}}(\forall \beta <_{\mathrm{lex}} \alpha O(\beta) \to O(\alpha)) \to \forall \alpha O(\alpha).$$

However, since $<_{\mathrm{lex}}$ is neither decidable nor well-founded, must restrict O to being an *open formula*.

$O(\alpha)$ is (classically) open if it is of the form $\forall nC([\alpha](n)) \to \exists nB([\alpha](n))$.

Open induction via the minimal bad sequence argument

Suppose that $\exists\alpha\neg O(\alpha)$ (α a 'bad' sequence). By dependent choice construct a minimal sequence β using the rule

Given $\langle\beta(0),\ldots,\beta(n-1)\rangle$, let $\beta(n)$ be such that $\langle\beta(0),\ldots,\beta(n)\rangle$ extends to a bad sequence, but $\langle\beta(0),\ldots,\beta(n-1),y\rangle$ does not for any $y<\beta(n)$.

Open induction via the minimal bad sequence argument

Suppose that $\exists\alpha\neg O(\alpha)$ (α a 'bad' sequence). By dependent choice construct a minimal sequence β using the rule

> Given $\langle\beta(0),\ldots,\beta(n-1)\rangle$, let $\beta(n)$ be such that $\langle\beta(0),\ldots,\beta(n)\rangle$ extends to a bad sequence, but $\langle\beta(0),\ldots,\beta(n-1),y\rangle$ does not for any $y<\beta(n)$.

Then $\neg O(\beta)$ holds since $\neg O(\beta)\leftrightarrow\forall n(C([\beta](n)\wedge\neg B([\beta](n)))$.

Open induction via the minimal bad sequence argument

Suppose that $\exists\alpha\neg O(\alpha)$ (α a 'bad' sequence). By dependent choice construct a minimal sequence β using the rule

> Given $\langle\beta(0),\ldots,\beta(n-1)\rangle$, let $\beta(n)$ be such that $\langle\beta(0),\ldots,\beta(n)\rangle$ extends to a bad sequence, but $\langle\beta(0),\ldots,\beta(n-1),y\rangle$ does not for any $y<\beta(n)$.

Then $\neg O(\beta)$ holds since $\neg O(\beta)\leftrightarrow\forall n(C([\beta](n)\wedge\neg B([\beta](n)))$.

But by definition β is minimal with respect to $<_{\mathrm{lex}}$, and so $\forall\gamma{<_{\mathrm{lex}}}\beta O(\gamma)\rightarrow O(\beta)$ is false.

Open induction via the minimal bad sequence argument

Suppose that $\exists\alpha\neg O(\alpha)$ (α a 'bad' sequence). By dependent choice construct a minimal sequence β using the rule

Given $\langle\beta(0),\ldots,\beta(n-1)\rangle$, let $\beta(n)$ be such that $\langle\beta(0),\ldots,\beta(n)\rangle$ extends to a bad sequence, but $\langle\beta(0),\ldots,\beta(n-1),y\rangle$ does not for any $y<\beta(n)$.

Then $\neg O(\beta)$ holds since $\neg O(\beta)\leftrightarrow\forall n(C([\beta](n)\wedge\neg B([\beta](n)))$.

But by definition β is minimal with respect to $<_{\mathrm{lex}}$, and so $\forall\gamma{<_{\mathrm{lex}}}\beta O(\gamma)\rightarrow O(\beta)$ is false.

Open induction has received a lot of attention in constructive mathematics as the contrapositive of MBS. Implicitly lies behind Kruskal's theorem and Higman's lemma. These are used to prove e.g. termination of rewrite systems.

Open recursion (Berger 2004)

For well-founded relations $<$ on ρ can define a corresponding recursor by:

$$\mathrm{R}_{(\rho,<),\sigma}(x^{\rho}) \stackrel{\sigma}{=} f_x(\lambda y \,.\, \mathrm{R}_{<}(y) \text{ if } y < x).$$

Provably well-founded for arbitrary types σ by $\mathrm{TI}_{<}$.

Open recursion (Berger 2004)

For well-founded relations $<$ on ρ can define a corresponding recursor by:

$$\mathsf{R}_{(\rho,<),\sigma}(x^\rho) \stackrel{\sigma}{=} f_x(\lambda y \,.\, \mathsf{R}_<(y) \text{ if } y < x).$$

Provably well-founded for arbitrary types σ by $\mathsf{TI}_<$.

For the lexicographic ordering $<_{\mathsf{lex}}$, can define open recursor by

$$\mathsf{OR}_{(\rho,<)}(\alpha) \stackrel{\mathbb{N}}{=} F_\alpha(\lambda n, y^\rho, \beta \,.\, \mathsf{OR}([\alpha](n) * y \,@\, \beta) \text{ if } y < \alpha(n))$$

By forcing $\sigma = \mathbb{N}$ the formula

$$O(\alpha) :\equiv [\alpha \text{ total} \to \mathsf{OR}(\alpha) \text{ total}]$$

is open by Cont, therefore $\mathsf{OR}_{(\rho,<)}$ provably well-founded by $\mathsf{OI}_{(\rho,<)}$.

Computational interpretation of open induction

Theorem (Berger 2004)

- *Open recursion solves the modified realizability interpretation of open induction.*
- *Open induction proves the double negation shift over intuitionistic logic, and the resulting open recursive realizer for countable choice is the* BBC *functional.*

Computational interpretation of open induction

Theorem (Berger 2004)

- *Open recursion solves the modified realizability interpretation of open induction.*
- *Open induction proves the double negation shift over intuitionistic logic, and the resulting open recursive realizer for countable choice is the* BBC *functional.*

The result: an elegant proof of totality and correctness of BBC in the continuous functionals, and a much deeper understanding of its recursive structure - BBC is an ‘open recursive’ functional...

BBC is a simple instance of OR

For x, $y\colon \rho_\perp$, let $x < y$ iff x defined and y undefined. Then

$$u \oplus (n, x) <_{\mathrm{lex}} u \text{ whenever } u(n) \text{ undefined.}$$

BBC is a simple instance of OR

For $x, y \colon \rho_\perp$, let $x < y$ iff x defined and y undefined. Then

$$u \oplus (n, x) <_{\text{lex}} u \text{ whenever } u(n) \text{ undefined.}$$

Let the parameter F for $\mathrm{OR}_{(\rho_\perp, <)}$ given by

$$F_u(P) := q(u \mathbin{@} \lambda n \,.\, \varepsilon_n(\lambda x \,.\, Pnxu))$$

Then can define $\mathrm{BBC}^{\varepsilon, q}(u) = \mathrm{OR}^F(u)$,

BBC is a simple instance of OR

For x, $y\colon \rho_\perp$, let $x < y$ iff x defined and y undefined. Then

$$u \oplus (n, x) <_{\mathrm{lex}} u \text{ whenever } u(n) \text{ undefined.}$$

Let the parameter F for $\mathrm{OR}_{(\rho_\perp, <)}$ given by

$$F_u(P) := q(u \mathbin{@} \lambda n \,.\, \varepsilon_n(\lambda x \,.\, Pnxu))$$

Then can define $\mathrm{BBC}^{\varepsilon,q}(u) = \mathrm{OR}^F(u)$,since

$$\mathrm{BBC}(u) = F_u(\lambda n, y, v \,.\, \mathrm{OR}([u](n) * y \mathbin{@} v) \text{ if } y < u(n))$$

BBC is a simple instance of OR

For $x, y \colon \rho_\perp$, let $x < y$ iff x defined and y undefined. Then

$$u \oplus (n, x) <_{\mathrm{lex}} u \text{ whenever } u(n) \text{ undefined.}$$

Let the parameter F for $\mathrm{OR}_{(\rho_\perp, <)}$ given by

$$F_u(P) := q(u \,@\, \lambda n \,.\, \varepsilon_n(\lambda x \,.\, Pnxu))$$

Then can define $\mathrm{BBC}^{\varepsilon, q}(u) = \mathrm{OR}^F(u)$,since

$$\begin{aligned} \mathrm{BBC}(u) &= F_u(\lambda n, y, v \,.\, \mathrm{OR}([u](n) * y \,@\, v) \text{ if } y < u(n)) \\ &= q(u \,@\, \lambda n \,.\, \varepsilon_n(\lambda x \,.\, \mathrm{OR}([u](n) * x \,@\, u) \text{ if } u(n) \text{ undefined})) \end{aligned}$$

BBC is a simple instance of OR

For $x, y\colon \rho_\perp$, let $x < y$ iff x defined and y undefined. Then

$$u \oplus (n, x) <_{\mathrm{lex}} u \text{ whenever } u(n) \text{ undefined.}$$

Let the parameter F for $\mathrm{OR}_{(\rho_\perp,<)}$ given by

$$F_u(P) := q(u \mathbin{@} \lambda n \,.\, \varepsilon_n(\lambda x \,.\, Pnxu))$$

Then can define $\mathrm{BBC}^{\varepsilon,q}(u) = \mathrm{OR}^F(u)$,since

$$\begin{aligned}
\mathrm{BBC}(u) &= F_u(\lambda n, y, v \,.\, \mathrm{OR}([u](n) * y \mathbin{@} v) \text{ if } y < u(n)) \\
&= q(u \mathbin{@} \lambda n \,.\, \varepsilon_n(\lambda x \,.\, \mathrm{OR}([u](n) * x \mathbin{@} u) \text{ if } u(n) \text{ undefined})) \\
&= q(u \mathbin{@} \lambda n \,.\, \varepsilon_n(\lambda x \,.\, \mathrm{OR}(u \oplus (n, x))))
\end{aligned}$$

BBC is a simple instance of OR

For $x, y \colon \rho_\perp$, let $x < y$ iff x defined and y undefined. Then

$$u \oplus (n, x) <_{\text{lex}} u \text{ whenever } u(n) \text{ undefined.}$$

Let the parameter F for $\mathrm{OR}_{(\rho_\perp, <)}$ given by

$$F_u(P) := q(u \mathbin{@} \lambda n \,.\, \varepsilon_n(\lambda x \,.\, Pnxu))$$

Then can define $\mathrm{BBC}^{\varepsilon, q}(u) = \mathrm{OR}^F(u)$,since

$$\begin{aligned}
\mathrm{BBC}(u) &= F_u(\lambda n, y, v \,.\, \mathrm{OR}([u](n) * y \mathbin{@} v) \text{ if } y < u(n)) \\
&= q(u \mathbin{@} \lambda n \,.\, \varepsilon_n(\lambda x \,.\, \mathrm{OR}([u](n) * x \mathbin{@} u) \text{ if } u(n) \text{ undefined})) \\
&= q(u \mathbin{@} \lambda n \,.\, \varepsilon_n(\lambda x \,.\, \mathrm{OR}(u \oplus (n, x)))) \\
&= q(u \mathbin{@} \lambda n \,.\, \varepsilon_n(\lambda x \,.\, \mathrm{BBC}(u \oplus (n, x))))
\end{aligned}$$

How strong is open recursion?

Even BBC is just a very basic instance of the schema of open recursion. But open recursion has clear connection to proof theory and is easier to reason about.

How strong is open recursion?

Even BBC is just a very basic instance of the schema of open recursion. But open recursion has clear connection to proof theory and is easier to reason about.

Theorem (P. 2013)

IPS *primitive recursively defines* $\mathrm{OR}_{(\rho,<)}$ *over* $\mathrm{Cont}+\mathrm{QF\text{-}BI}$ *whenever* $\mathrm{R}_{(\rho,<)}$ *is definable in system* T*. In particular,* IPS *defines (and is therefore equivalent to)* BBC*.*

How strong is open recursion?

Even BBC is just a very basic instance of the schema of open recursion. But open recursion has clear connection to proof theory and is easier to reason about.

Theorem (P. 2013)

IPS *primitive recursively defines* $\mathrm{OR}_{(\rho,<)}$ *over* Cont + QF-BI *whenever* $\mathrm{R}_{(\rho,<)}$ *is definable in system* T. *In particular,* IPS *defines (and is therefore equivalent to)* BBC.

Basic idea. A computational form of the minimal bad sequence argument. Idea taken from bar recursive realizer for Higman's lemma extracted in (P. 2012).

Again, proof quite intricate so won't go into details!

Extensions of T

$$
\begin{tikzcd}
\boxed{\text{BBC functional, open recursion}} \arrow[d, <->, dotted] & \\
\boxed{\text{Implicit bar recursion (IPS, MBR etc.)}} \arrow[d, "\text{S1-S9}\not\models\text{MBR}"] & \\
\boxed{\text{Spector bar recursion (GBR, EPS etc.)}} \arrow[d, "\mathcal{S}^{\omega}\not\models\text{GBR}"] & \boxed{\text{KBR}} \arrow[l, "\mathcal{M}^{\omega}\not\models\text{KBR}"'] \\
\boxed{\text{Gödel primitive recursion}} &
\end{tikzcd}
$$

Outline

Summary

Combined work of several authors has resulted in:

- The classification of most of the familiar bar-recursive extensions of system T according to primitive recursive definability.
- In each case have explicit, *instance-wise* constructions, verified semi-intuitionistically in $\text{E-HA}^{\omega} + \text{Cont} + \text{QF-BI}$ or weaker theories.

Summary

Combined work of several authors has resulted in:

- The classification of most of the familiar bar-recursive extensions of system T according to primitive recursive definability.
- In each case have explicit, *instance-wise* constructions, verified semi-intuitionistically in $\text{E-HA}^{\omega} + \text{Cont} + \text{QF-BI}$ or weaker theories.

My contribution includes:

- The equivalence of (implicit) bar recursion and open recursion and BBC functional.
- New proofs of totality of open recursion and BBC, along with confirmation of fact that these are stronger than Spector's bar recursion and non-computable in continuous functionals.

Open questions

- Have said nothing about relative strength of extensions at level of types.
- How to specific instances of recursion used to interpret axiom of choice compare in terms of computational complexity?
- Can we give a meaningful semantic comparison of bar recursion and BBC?
- Can we construct new, interesting extensions of system T that do not belong in any of the current classes?

Bar recursive interpretations of choice

Our results confirm that like MBR, both open recursion and BBC are stronger than Spector's bar recursion.

- In this sense Spector's bar recursive interpretation of choice remains optimal (essentially due to strength of Dialectica interpretation over realizability).
- Does not requre Cont reason about it or validate interpretation of choice (hence all results hold in majorizable functionals too)

Bar recursive interpretations of choice

Our results confirm that like MBR, both open recursion and BBC are stronger than Spector's bar recursion.

- In this sense Spector's bar recursive interpretation of choice remains optimal (essentially due to strength of Dialectica interpretation over realizability).
- Does not requre Cont reason about it or validate interpretation of choice (hence all results hold in majorizable functionals too)

But Spector's bar recursion fairly arbitrary form of recursion designed to extend Gödel's consistency proof, not extract programs from proofs: BBC and open recursion both devised partly to address this and improve the semantics of extracted programs.

Alternatives to Spector's bar recursion

Can we devise new forms of recursion that are more amenable to extracting programs from proofs in mathematical analysis, but still computable and weaker than implicit bar recursion?

Alternatives to Spector's bar recursion

Can we devise new forms of recursion that are more amenable to extracting programs from proofs in mathematical analysis, but still computable and weaker than implicit bar recursion?
E.g. explicit form of open recursion:

$$\Phi(\alpha^{\rho^{\mathbb{N}}}) \stackrel{\sigma}{=} F_{[u](N)*\mathbf{0}}(\lambda n < N, y, \beta \,.\, \Phi([u](n) * y @ \beta) \text{ if } y < \alpha(n))$$

where N least satisfying

$$\varphi_{[u](N)*\mathbf{0}}(\lambda n < N, y, \beta \,.\, \Phi([u](n) * y @ \beta) \text{ if } y < \alpha(n)) < N$$

Alternatives to Spector's bar recursion

Can we devise new forms of recursion that are more amenable to extracting programs from proofs in mathematical analysis, but still computable and weaker than implicit bar recursion?
E.g. explicit form of open recursion:

$$\Phi(\alpha^{\rho^{\mathbb{N}}}) \overset{\sigma}{=} F_{[u](N)*\mathbf{0}}(\lambda n < N, y, \beta \,.\, \Phi([u](n) * y \mathbin{@} \beta) \text{ if } y < \alpha(n))$$

where N least satisfying

$$\varphi_{[u](N)*\mathbf{0}}(\lambda n < N, y, \beta \,.\, \Phi([u](n) * y \mathbin{@} \beta) \text{ if } y < \alpha(n)) < N$$

Conjecture. Φ primitive recursively equivalent to Spector's bar recursion.

Could we use this to extract efficient and readable programs from minimal-bad-sequence proofs of Higman's lemma and Kruskal's theorem? New quantitative results?

Relevant papers

S. Berardi, M. Bezem and T. Coquand. On the computational content of the axiom of choice. *Journal of Symbolic Logic, 63(2):600-622, 1998.*

U. Berger. A computational interpretation of open induction. *Proc. IEEE Symposium on Logic in Computer Science, 326-334, 2004*

U. Berger and P. Oliva. Modified bar recursion. *Mathematical Structures in Theoretical Computer Science 16(2):163-183, 2006*

M. Escardo and P. Oliva. Bar recursion and products of selection functions. *Submitted for review*

P. Oliva and T. Powell On Spector's bar recursion. *Mathematical Logic Quarterly, 58:356-365, 2012*

T. Powell On Bar Recursive Interpretation of Analysis. *PhD thesis, Queen Mary University of London, 2013*

T. Powell The equivalence of bar recursion and open recursion. *Submitted for review*