SEPARATING NOTIONS OF HIGHER-TYPE POLYNOMIAL-TIME

Robert Irwin — *Syracuse University*
Bruce Kapron — *University of Victoria*
Jim Royer — *Syracuse University*

FUNCTIONALS & EFFICIENCY

Question: Suppose you have a programming language with a combinator:

$$\mathrm{C}\colon (\mathrm{N} \to \mathrm{N}) \times (\mathrm{N} \to \mathrm{N}) \to \mathrm{N}.$$
$$[\![\mathrm{C\,f\,g}]\!] \;=\; [\![\mathrm{f}]\!] \circ [\![\mathrm{g}]\!].$$

How would you say that an implementation of C is efficient?

What does NOT work: You could say simply

$$[\![\mathrm{f}]\!], [\![\mathrm{g}]\!] \in \mathrm{Ptime} \implies [\![\mathrm{C\,f\,g}]\!] \in \mathrm{Ptime}.$$

But this leaves open nasty possibilities such as

$$[\![\mathrm{f}]\!], [\![\mathrm{g}]\!] \in \mathrm{Exptime}\ \&\ [\![\mathrm{C\,f\,g}]\!] \notin \mathrm{PrimRec}.$$

What we would like: To say something along the lines of

$$\text{the complexity of } \mathrm{C\,f\,g} \;\leq\; \mathscr{X}_{\mathrm{C}}(\text{the complexity of } \mathrm{f}, \text{the complexity of } \mathrm{g})$$

for some sort of $\mathscr{X}_{\mathrm{C}}$. That is, we want to lift complexity to higher types.

HIGHER-TYPE ANALOGUES OF PTIME

- **In higher-type computational complexity, one of the first landmarks to identify is a higher-type analogue of polynomial time. Why?**

- **At type-level 2, the type-2 Basic Feasible Functionals** (Mehlhorn 1974, Cook-Urquhart 1989) **seem a reasonable choice. (More on this later.)**

- **Above type-level 2 (in the simple types), there are alternative notions:**
  - **The Basic Feasible Functionals** (Cook-Urquhart 1989)
  - **The Class** D (Seth 1995)
  - **. . .**

- **Seth conjectured** $\mathrm{BFF} \neq \mathrm{D}$**.**

- **In this paper, we**
  - **clarify the nature of** D **(& lots of other things), and**
  - **confirm Seth's conjecture.**

CHARACTERIZATIONS OF THESE CLASSES

- **The Type-2 BFFs = the functionals computable through:**
  - **A Type-2 fragment of PV^ω** (Cook-Urquhart 1989)
  - **Type-2 Bounded Typed Loop Programs** (Cook-Kapron 1989)
  - **A reasonable type-2 machine model** (Kapron-Cook 1991)
  - **Type-2 Inflationary Tiered Loop Programs** (Irwin-Kapron-Royer 1998)
  - **...**
- **The full BFFs = the functionals computable through:**
  - PV^ω (Cook-Urquhart 1989)
  - **Bounded Typed Loop Programs** (Cook-Kapron 1989)
  - **A poorly understood machine model** (Seth 1995)
  - **Inflationary Tiered Loop Programs** (this paper)
  - **...**
- **The Class D = the total functionals computable through:**
  - **a very poorly understood machine model** (Seth 1995)

A BIT OF EXPLICIT COMPUTATIONAL COMPLEXITY

The conventional ingredients for a complexity class

a machine/ cost model	+	a size measure for inputs	+	a family of bounding functions	$\longrightarrow$	$\mathcal{C}$

Some type-1 examples:

Det. TMs	+	length of a string	+	polynomials	$\longrightarrow$	Ptime
Det. SMMs	+	# nodes in a graph	+	polynomials	$\longrightarrow$	Ptime
Nondet. TMs	+	length of a string	+	polynomials	$\longrightarrow$	NP
Det. TMs	+	length of a string	+	$_2$polys	$\longrightarrow$	Exptime

A type-2 example:

OTMs/ans. length cost	+	lengths of strings & type-1 functions	+	2nd-order polynomials	$\longrightarrow$	$\mathbf{BFF}_2$

EXPLICIT COMPLEXITY AT TYPE-LEVEL 2

OTMs/Answer length cost

Oracle Turing Machines

An oracle query costs the length of the answer.

Every other twitch of the machine costs 1.

Length of a type-1 function

For $\mathrm{f}\colon \mathbf{N} \to \mathbf{N}$**, define** $|\mathrm{f}| = \lambda \mathrm{n}.\max\{\, |\mathrm{f}(\mathrm{x})| : |\mathrm{x}| \leq \mathrm{n} \,\}$**.**

2nd-order polynomials

$\mathrm{P} ::= \langle$**num. const**$\rangle \mid \langle$**type-0 var**$\rangle \mid \mathrm{P} + \mathrm{P} \mid \mathrm{P} \cdot \mathrm{P} \mid \langle$**type-1 var**$\rangle(\mathrm{P})$

E.g., $|\mathrm{f}|(|\mathrm{f}|(2 \cdot |\mathrm{y}|) + 6)$

The Type-2 BFFs

The things computable on OTMs in (2nd-order) poly time in the lengths of the (type-level 1 and 2) inputs (Kapron-Cook 1991)

EXPLICIT COMPLEXITY ABOVE TYPE-LEVEL 2

SETH'S MACHINE CHARACTERIZATIONS OF BFF AND D

- These are only partial analogues of the Kapron-Cook characterization. E.g., there is no explicit
  - notion of the length of a higher-type function,
  - notion of polynomials over lengths, or
  - mention of the new ingredient that pops up at type-level 3 — the underlying domain of computation. (More on this shortly)
- There is a lot of cleverness and insight in these characterizations.

WE WORK AT COMPLETING SETH'S PICTURE

- We introduce higher-type notions of length and polynomials.
- Rather than start with machines, we use typed programming formalisms.
- These type-systems are based on h.t. polynomials over h.t. lengths.
- These formalisms are used to treat h.t. machines. (Not In This Talk!!!)

STEP 1: A REFERENCE P.L. FOR THE BFFS

Bounded Typed Loop Programs (BTLPs)

- A simple imperative P.L.
- Simple types over $\mathbb{N}$
- $[\![\mathbb{N}]\!]$ = nat. numbers $\equiv \{0,1\}^*$
- Our BTLP =
  C-K's BTLP
  $-$ global refs to type-$\mathbb{N}$ vars
  $+$ λ-exps in proc calls
- No recursive procedures
- We avoid dealing with h.t. recursion in this paper.

- Iteration via Loop construct:

  Loop x with w do S Endloop

  means
  - S is repeated |x|-many times
  - Each "y := E" in S is equiv to
    If |E| $\leq$ |w|
    then y := E
    else y := 0
  - No assignments to x or w in S

A BTLP PROCEDURE FOR $(F,x) \mapsto \sum_{i<|x|} F(\lambda z.i)$

```
Procedure SumUp (F: (N → N) → N, x: N)
   var i, maxi, sum;

   maxi := 0;     i := 0;
   Loop x with x do     (* Find the i for which F(λz: N.i) is maximal *)
      If F(λz: N.maxi)< F(λz: N.i) then maxi := i; Endif;       i := i+1;
   Endloop;

   sum := 0;     i := 0;
   Loop x with (|x|+1)·(|F(λz: N.maxi)|+1) do   (* Compute the sum *)
      sum := sum + F(λz: N.i);       i := i+1;
   Endloop;

   Return sum
End
```

STEP 2: HIGHER TYPE LENGTH

A FIRST ATTEMPT

For $F\colon (N \to N) \to N$, lets try:

$$|F|(g) \;=\; \max\{\, |F(f)| \;:\; |f| \leq g \,\},$$

where $F \in \mathrm{Full}_{(N\to N)\to N}$, $g \in \mathrm{TMon}_{\omega\to\omega}$.

A FIRST PROBLEM

If F is unbounded on $\{\, f \;:\; f \text{ is 0-1 valued} \,\}$, then $|F|(\lambda n.1) = \infty$.

OPTION 1

Restrict F to $\mathrm{TCont}_{(N\to N)\to N}$. Then König's Lemma saves us (for the moment).

OPTION 2

Restrict the $\max$ in the definition to better reflect what happens in BTLP proc. calls.

We follow up on option 1 here.

CONVENTIONS

- ω = tallies $\equiv 0^*$.
- $x \in N$. $n \in \omega$. $f\colon N \to N$.
- $\mathrm{len}(x) =$ len. of dyadic rep. of x
- $|x| = 0^{\mathrm{len}(x)}$
- $|f|\colon \omega \to \omega$ and $|f|(n) = \max\{\, |f(x)| \;:\; |x| \leq n \,\}$.
- Full_σ = all type-σ fncts
- TCont_σ = total, cont. type-σ fncts
- TMon_σ = hered. total, mono. incr. type-σ fncts

LENGTH AT TYPE-LEVEL 3

A SECOND ATTEMPT

For $\Psi\colon \mathrm{TCont}_{((\mathrm{N}\to\mathrm{N})\to\mathrm{N})\to\mathrm{N}}$ lets try:

$$|\Psi|(\mathrm{G}) \quad = \quad \max(\{\, |\Psi(\mathrm{F})| \mid |\mathrm{F}| \leq \mathrm{G} \,\}),$$

where $\mathrm{F}\colon \mathrm{TCont}_{(\mathrm{N}\to\mathrm{N})\to\mathrm{N}}$ and $\mathrm{G}\colon \mathrm{TMon}_{(\omega\to\omega)\to\omega}$.

A SECOND PROBLEM

There are Ψ that are unbounded on $\{\, \mathrm{F} \mid \mathrm{F}$ is 0-1 valued $\}$, and hence have $|\Psi|(\lambda \mathrm{g}.1) = \infty$.

A (SORT OF) SOLUTION

Further restrict Ψ so that it is bounded on sets of the form

$$\{\, \mathrm{F} \mid |\mathrm{F}| \leq \mathrm{G} \,\}$$

for $\mathrm{G} \in \mathrm{TMon}_{(\omega\to\omega)\to\omega}$

. . . and so on up the type-hierarchy.

BOUNDED LENGTH AND BOUNDED CONTINUITY

What will have to pass for DEFINITIONS

- σ – simple type over N
- $|\sigma|_\mathrm{b}$ – corresp. simple type over ω
- $|\mathrm{x}|_\mathrm{b} = |\mathrm{x}|$, for $\mathrm{x} \in \mathrm{N}$.
  $\mathrm{BCont}_\mathrm{N} = \mathrm{N}$.
  $\mathrm{TLen}_\omega = \omega$.
- For $\sigma = \sigma_0 \times \cdots \times \sigma_\mathrm{n} \to \mathrm{N}$,
  $\mathrm{f} \in \mathrm{TCont}_\sigma$, $\ell_\mathrm{i} \in \mathrm{TLen}_{|\sigma_\mathrm{i}|_\mathrm{b}}$ &
  $\mathrm{x_i}\colon \mathrm{BCont}_{\sigma_\mathrm{i}}$, define:

  $|\mathrm{f}|_\mathrm{b}(\vec{\ell}) = \sup\{\, |\mathrm{f}(\vec{\mathrm{x}})| \;:\; |\mathrm{x_i}|_\mathrm{b} \le \ell_\mathrm{i} \,\}$.

  $\mathrm{BCont}_\sigma =$ the f's with $|\mathrm{f}|_\mathrm{b}$ total.

  $\mathrm{TLen}_{|\sigma|_\mathrm{b}} =$ the total $|\mathrm{f}|_\mathrm{b}$'s.

- For σ at type-levels 1 or 2,
  $\mathrm{BCont}_\sigma = \mathrm{TCont}_\sigma$ and
  $\mathrm{TLen}_{|\sigma|_\mathrm{b}} = \mathrm{TMon}_{|\sigma|_\mathrm{b}}$.
- For σ above type-level 2,
  $\mathrm{BCont}_\sigma \subset \mathrm{TCont}_\sigma$ and
  $\mathrm{TLen}_{|\sigma|_\mathrm{b}}$ has noncont. elms.
- In general we have

  $$|\mathrm{x_0}(\mathrm{x_1},\ldots,\mathrm{x_k})|_\mathrm{b} \le |\mathrm{x_0}|_\mathrm{b}(|\mathrm{x_1}|_\mathrm{b},\ldots,|\mathrm{x_k}|_\mathrm{b}),$$

  but not equality.

STEP 3: HIGHER-TYPE POLYNOMIALS

Example Consider the bounded continuous Ψ given by:

$$\Psi(\mathrm{F},\mathrm{x}) \;=\; \textstyle\sum_{\mathrm{i}<|\mathrm{x}|} \mathrm{F}(\lambda \mathrm{z}\,.\,\mathrm{i}).$$

We can show that

$$|\Psi(\mathrm{F},\mathrm{x})| \;\leq\; \underline{(|\mathrm{x}|+1)\cdot|\mathrm{F}|_{\mathrm{b}}\,(\lambda \mathrm{n}\,.\,|\mathrm{x}|)}.$$

We want the RHS to be a "polynomial bound" in $|\mathrm{x}|$ and $|\mathrm{F}|_{\mathrm{b}}$.

HIGHER TYPE POLYNOMIALS (HTPs)

Syntax

Simply typed λ-calculus + base type ω + **plus** and **times**

Semantics

$[\![-]\!]_{\mathrm{TLen}}$: syntax $\longrightarrow \mathrm{TLen}$ . . . the obvious thing

Depth

$\mathrm{depth}(\mathrm{t})$ = max depth of nesting of applications in t's n.f.

MORE ON THE DEPTH OF HTPs

Example

$$\mathrm{depth}\Big(\,(k+1)\cdot G(\lambda n.m)\,\Big) = 1.$$

$$\mathrm{depth}\Big(\,G(\lambda n.G(\lambda m.g(m+n)))\,\Big) = 3.$$

LEMMA **Suppose that**

- $t,\ t_1,\ldots,t_n$ **are HTP-terms &** $s = t[x_1,\ldots,x_n \mapsto t_1,\ldots,t_n]$
- $a = \mathrm{depth}(t)$ **and** $b = \max\{\mathrm{depth}(t_i) \mid i = 1,\ldots n\}$
- $\ell =$ **the max type-level of the types assigned to the** t_i**'s.**

Then, $\mathrm{depth}(s) \leq e_\ell(a,b)$**, where:**

$$e_0(a,b) = a+b. \qquad e_{\ell+1}(0,b) = b.$$
$$e_{\ell+1}(a+1,b) \;=\; \max(\,1+e_{\ell+1}(a,b),\, e_\ell(b,e_{\ell+1}(a,b))).$$

The e_ℓ's climb the Grzegorczyk hierarchy.

STEP 4: INFLATIONARY TYPED LOOP PROGRAMS

ITLP is an imperative P.L. similar to BTLP except

- types have a complexity component
- more dynamic iteration construct
- ...

ITLP Types

An ITLP type is a pair (σ, d), where

- σ is a simple type over N
  — the shape
- d is an element of ω
  — the depth

- We often write
  - (N, d) as $\mathrm{N_d}$
  - $(\sigma_0 \times \cdots \times \sigma_n \to \mathrm{N}, \mathrm{d})$ as $\sigma_0 \times \cdots \times \sigma_n \to_\mathrm{d} \mathrm{N}$
- $\mathrm{X}\colon (\sigma, \mathrm{d}) \approx$ X is σ object with a depth d HTP bound
- $\mathrm{X}\colon (\sigma, 0) \approx \mathrm{X}$ is a parameter

SOME ITLP TYPING RULES

The Abstraction Rule

$$\frac{v_0\colon(\sigma_0, 0) \quad \cdots \quad v_n\colon(\sigma_n, 0) \quad v(v'_0, \ldots, v'_m)\colon \mathrm{N}_i}{\lambda v_0, \ldots, v_n \,.\, v(v'_0, \ldots, v'_m)\colon \sigma_0 \times \cdots \times \sigma_n \to_i \mathrm{N}}$$

(We'll have $\mathrm{i} > 0$ in all the actual uses of the above rule.)

Procedure Application Rule

$$\frac{v\colon \sigma_0 \times \cdots \times \sigma_n \to_{i+1} \mathrm{N} \quad a_0\colon(\sigma_0, d_0) \ \cdots \ a_n\colon(\sigma_n, d_n)}{v(a_0, \ldots, a_n)\colon \mathrm{N}_{e_\ell(i+1,d)}}$$

where $\mathrm{d} = \max_{\mathrm{i}<\mathrm{n}} \mathrm{d_i}$ and $\ell = \max_{\mathrm{i}<\mathrm{n}}$ type-level(σ_i).

Parameter Application Rule

$$\frac{v\colon \sigma_0 \times \cdots \times \sigma_n \to_0 \mathrm{N} \quad a_0\colon(\sigma_0, d_0) \ \cdots \ a_n\colon(\sigma_n, d_n)}{v(a_0, \ldots, a_n)\colon \mathrm{N}_{d+1}}$$

where $\mathrm{d} = \max_{\mathrm{i}<\mathrm{n}} \mathrm{d_i}$.

MORE ON ITLP

Type-Level 0 Downward Coercion

ITLP includes a primitive down with the intended semantics

$$\mathrm{down}(x, y) = \begin{cases} y, & \text{if } |y| \leq |x| \\ 0, & \text{if } |y| > |x| \end{cases}$$

and typing rule

$$\frac{v_0 : N_i \quad v_1 : N_j}{\mathrm{down}(v_0, v_1) : N_i} \quad \begin{pmatrix} \text{where} \\ i < j \end{pmatrix}$$

We'll see an important variant of this later.

Semantics

We have a Full- and a BCont-based semantics

$$[\![-]\!]_{\mathrm{Full}} : \text{syntax} \longrightarrow \mathbb{D}^{\mathrm{Full}}$$
$$[\![-]\!]_{\mathrm{BCont}} : \text{syntax} \longrightarrow \mathbb{D}^{\mathrm{BCont}}$$

Elms of $\mathbb{D}^X$ are of the form (x, d)

$x \in X_\sigma$, for some σ, and $d \in \omega$.

Also

$$(x_0, d_0)\left((x_1, d_1), \ldots, (x_n, d_n)\right) = (x_0(x_1, \ldots, x_n), d),$$

where d is determined per the ITLP typing rules.

SOME THEOREMS, FINALLY!!

THEOREM (Poly Boundedness)

Suppose P is a closed type-(τ, d) ITLP procedure. Then there is a type-$|\tau|_{\mathrm{b}}$, depth-d HTP p such that

$$|[\![\mathrm{P}]\!]_{\mathrm{BCont}}|_{\mathrm{b}} \leq [\![\mathrm{p}]\!]_{\mathrm{TLen}}.$$

THEOREM (BTLP $\longrightarrow$ ITLP)

Any BTLP procedure can be translated to an "equivalent" ITLP procedure, wrt the Full-semantics,

COROLLARY

(a) BTLP is poly-bounded on BCont args.

(b) BTLP has a sensible BCont-based semantics.

DEFINITION

BCBFF = the BTLP-computable fcnls over BCont.

STEP 5: FLATTENING ITLP

ITLP$^\flat$ = ITLP − the Procedure Application Rule (with the e_ℓ's) +

The ITLP$^\flat$ Procedure Application Rule

$$\frac{\mathsf{v}\colon \sigma_0 \times \cdots \times \sigma_n \to_{i+1} \mathrm{N} \quad \mathrm{a}_0\colon (\sigma_0, \mathrm{d}_0) \quad \cdots \quad \mathrm{a}_n\colon (\sigma_n, \mathrm{d}_n)}{\mathsf{v}(\mathrm{a}_0, \ldots, \mathrm{a}_n)\colon \mathrm{N}_{i+d+1}}$$

where $\mathrm{d} = \max_{j \le n} \mathrm{d}_j$ and
where, for each $j = 0, \ldots, n$, if σ_j is an arrow type, then $\mathrm{d}_j = 0$.

ITLP$^\flat$ is reasonably close to Seth's machine-model for the BFFs.

THEOREM (ITLP ↔ ITLP$^\flat$ ↔ BTLP)
ITLP, ITLP$^\flat$, and BTLP are inter-translatable.
But some directions are more pleasant than others

COROLLARY **ITLP, ITLP$^\flat$, and BTLP each:**
(a) determine BFF wrt to their Full-based semantics.
(b) determine BCBFF wrt to their BCont-based semantics.

A SIDE STEP: ADDING H.T. DOWNWARD COERCIONS

- Suppose we extend ITLP$^\flat$ to allow terms such as

  $(*)$ $\quad \lambda$y . down(bnd(y),g(x,y))

  where, say,

  g: $\mathrm{N} \times \mathrm{N} \to_1 \mathrm{N}$, bnd: $\mathrm{N} \to_1 \mathrm{N}$, x: N_3.

- How do we type $(*)$?
  - Suppose down has the type $\mathrm{N} \times \mathrm{N} \to_0 \mathrm{N}$.
  - Then ITLP$^\flat$'s rules assign $\mathrm{N} \to_6 \mathrm{N}$.
- But, we know that

$$\left| [\![\lambda \mathrm{y} . \mathrm{down}(\mathrm{bnd}(\mathrm{y}),\mathrm{g}(\mathrm{x},\mathrm{y}))]\!]_{\mathrm{BCont}} \right|_{\mathrm{b}} \leq \left| [\![\mathrm{bnd}]\!]_{\mathrm{BCont}} \right|_{\mathrm{b}}$$

  and this holds independent of the value of x!

- So, why not assign $(*)$ the same depth as bnd?
- ITLP$^\sharp$ is such an extension with such typing rules.
  So what?

MORE ON ITLP$^\sharp$— A KEY EXAMPLE

```
Procedure H(F: (N → N) →0 N): N3
  Procedure bnd(y: N0): N1
    Return 0^1
  End
  Procedure g(x: N0, y: N0): N1
    If x=y then Return 1 else Return 0
  End (* Note: λy.g(x,y) = C{x}. *)
  var w: N3,  x: N3;
  x := 0^2;
  For w := 0^1 to x do
    x := F(λy.down(bnd(y), g(x,y)));
  Endfor;
  Return x
End
```

Suppose $\mathcal{F}_0$: $\mathrm{Full}_{(\mathrm{N}\to\mathrm{N})\to\mathrm{N}}$ is $\ni$

$$\mathcal{F}_0(\mathrm{f}) = \begin{cases} 0^{|\mathrm{x}|+1}, & \text{if } \mathrm{f} = \mathrm{C}_{\{\mathrm{x}\}}; \\ 0^0, & \text{if } \forall \mathrm{x}[\mathrm{f} \neq \mathrm{C}_{\{\mathrm{x}\}}]; \end{cases}$$

Then $\mathsf{H}(\mathcal{F}_0)$ is undefined.

However

THM (Poly Boundedness)
Suppose P is a closed type-(τ, d) ITLP$^\sharp$ procedure. Then there is a type-$|\tau|_\mathrm{b}$, depth-d HTP p such that

$$|[\![\mathrm{P}]\!]_{\mathrm{BCont}}|_\mathrm{b} \leq [\![\mathrm{p}]\!]_{\mathrm{TLen}}.$$

What is going on?

THE BFF VERSUS D SEPARATIONS

DEFINITIONS

- D^- is the class of total ITLP$^\sharp$-computable functionals over Full.
- D is the class of total functionals by Seth's D-machines over Full.
- BCD^- is the class of ITLP$^\sharp$-computable functionals over BCont.
- BCD is the class of functionals by Seth's D-machines over BCont.

THEOREM

- $\mathrm{BCBFF} \subsetneq \mathrm{BCD}^-$, with a witness at type-level 3.
- $\mathrm{BFF} \subsetneq \mathrm{D}^-$, with a witness at type-level 3.

- $\mathrm{BCD}^- \subseteq \mathrm{BCD}$ and $\mathrm{D}^- \subseteq \mathrm{D}$.
  These containments are likely strict.

CONCLUSIONS

When computing over Full

- It seems unlikely that D is recursively presentable.
- Hence, BFF seems the more natural class.

When computing over BCont

- BCBFF seems too small.
- Hence, BCD seems the more natural class.

- But, is either Full or BCont a reasonable thing to compute over?
  Can we make this question more precise?
- Are there natural notions of feasibility for other computational domains?
- While both the complexity theory and semantics above are fairly humble, they mesh!