
THE WHY AND HOW OF HIGHER-TYPE COMPLEXITY THEORY

Jim Royer
Syracuse University

For references, see:
<http://www.cis.syr.edu/~royer/bib2.html>

AN EXAMPLE TYPE-2 ALGORITHM

The Setting

- ▶ We have a **student** (our algorithm) and a **teacher** (an oracle).
- ▶ The teacher knows a certain regular language L_* .
- ▶ The student's job is to infer a DFA for L_* .
- ▶ The student can ask two sorts of questions:
 - **Membership queries:** " $x \in L_*$?", x a string
To which the teacher **should** reply:
 - ★ YES, if $x \in L_*$
 - ★ NO, if $x \notin L_*$
 - **Equivalence queries:** " $L(M) = L_*$?", M a DFA
To which the teacher **should** reply:
 - ★ YES, if $L(M) = L_*$ (**SUCCESS!**)
 - ★ z , if $L(M) \neq L_*$, z an arb. $\in L(M) \Delta L_*$
- ▶ A teacher that always behaves as it should is called **dutiful**.

ANGLUIN'S ALGORITHM

Theorem (Angluin 1987)

There is an algorithm \mathcal{A} that

- ▶ given a dutiful teacher T , \mathcal{A} learns T 's language, and
- ▶ throughout \mathcal{A} 's execution, its runtime is bounded by

$$p(\max\{|z| \mid z \text{ is one of } T\text{'s counterexamples}\})$$

where p is some polynomial indep. of T .

Proof Use Myhill-Nerode.

Aside:

Theorem (Angluin 1988)

Both membership and equivalence queries are necessary for the above sort of polytime learnability of DFAs.

REMARKS ON \mathcal{A}

- ▶ It is algorithmically interesting
- ▶ It runs in poly-time in its observed inputs.
- ▶ If the teacher always gives the smallest possible counterexamples, then the algorithm runs in time poly in the number of states of the minimal DFA for L_* .
- ▶ Bad things can happen if the teacher is not dutiful. E.g., the teacher takes $L_* = \{ a^n b^n \mid n \geq 1 \}$.
- ▶ So, on the domain of all oracles, \mathcal{A} is not total.

We want to develop a complexity theory for higher-type algorithms such as \mathcal{A} .

We are still at the beginnings of such a development.

SOME OTHER EXAMPLES

- ▶ effective versions of $(f, a, b) \mapsto \int_a^b f(x)dx$
- ▶ $\text{Compose}(f, g) = f \circ g$
 $\text{Apply}(f, x) = f(x)$
 $\text{Map}(f, [x_1, \dots, x_n]) = [f(x_1), \dots, f(x_n)]$
...
- ▶ cryptographic constructions
complexity bounded reductions (e.g., \leq_T^p)
oracle constructions (type-level $2\frac{1}{2}$)
...
- ▶ computation by interacting agents, i.e., games

WHY WORRY ABOUT THE COMPLEXITY OF HIGHER-TYPE COMPUTATION?

Formal methods folk wisdom wrt **correctness**

If you cannot **in principle** reason about
the correctness of systems,
Then you are blind to many properties of your systems.
E.g., Can you even say what you want your
systems to do?

The same goes for **efficiency**.

A FIRST STEP: HIGHER-TYPE ANALOGUES OF PTIME

Why **this** as a first step?

- ▶ Force of habit.
- ▶ We need landmarks to ground our work.
- ▶ Most examples are too simple or too complex.
- ▶ So we need to proceed by analogy.
- ▶ In ordinary complexity theory, **P** and **PF** are useful and flexible reference classes.

On the other hand . . .

“General Type-2 Complexity Classes”

by Chung-Chih Li

Ph.D. Thesis, Syracuse University

Available this Fall (cross fingers)

BUT PROCEED BY WHICH ANALOGY?

Two standard ways to define a subrecursive class

Synthetic/Implicit/P.L.-Based/...

Given a restrictive P.L. or function algebra F .

Then the class = the F -computable functions.

Examples

Cobham's characterization of PF .

... see lots of other talks at this workshop

Analytic/Explicit/Machine-Based/...

Given

(a) a general machine or P.L.

with an associated cost model

(b) a way of measuring the size of an input, and

(c) a family of bounding functions,

Then the class = the things computable through that model under those resource bounds.

Examples

PF = the functions computable on a TM (with the usual cost model) in time polynomial in the length of the input.

A CAUTIONARY EXAMPLE

AN INFORMAL PRINCIPLE

A **feasible** functional applied to **feasible** arguments should yield a **feasible** result.

We have to be careful with glib statements like the above ... especially in regard to uniformity considerations.

EXAMPLE

The algorithm \mathcal{A} seems quite feasible.

But, one can construct a poly-time $f: \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N} \ni \lambda x. \mathcal{A}(\lambda y. f(x, y))$ is **not** polynomial-time computable.

E.g., $\lambda y. f(x, y)$ is a dutiful teacher that has

$$L_* = \{ 0^n 1^n \mid 1 \leq n \leq 2^{2^x} \}.$$

(\approx Banach-Mazur Functionals)

MACHINES FOR TYPE-2

Seth's polynomial-time oracle Turing machines (POTMs)

- ▶ These are useful, but lead to problematic territory. E.g., one can program \mathcal{A} on a POTM.

The OTMs with the uniform cost model

- ▶ All operations (including queries) cost 1.
- ▶ An OTM may read only part of a query response. This makes working with them a little tricky. (laziness)

The OTMs with the answer-length cost model

- ▶ An OTM query, $f(x) = ?$, costs $\max(1, |f(x)|)$.
- ▶ All all operations cost 1.
- ▶ Intuitively, these are forced to read query responses. This makes them closer to eager-evaluation PLs.

Below we use the OTMs with the ans-length cost model.

TYPE-1 LENGTH

Conventions.

- ▶ $\omega = \text{tallies} \equiv 0^*$.
- ▶ $\text{len}(x) = \text{len. of dyadic rep. of } x$
- ▶ $|x| = 0^{\text{len}(x)}$

QUESTION:

What should the length/size of an $f: \mathbb{N} \rightarrow \mathbb{N}$ be?

OPTION 1: Make it a number.

Problem: Then $\text{Apply} = \lambda f, x. f(x)$ fails to be type-2 polynomial-time computable.

OPTION 2: Make it a function, say $|f|: \omega \rightarrow \omega$

Requirement A. $|f|$ should be monotone nondecreasing.

Why? Standard in complexity theoretic bounds.

Requirement B. $|f|(|x|)$ should be $\geq |f(x)|$.

Why? $\max(|f(x)|, 1) = \text{the cost of } "f(x) = ?."$

DEFINITION. $|f| = \lambda n. \max\{ |f(x)| \mid |x| \leq n \}$.

TYPE-1 LENGTH, CONTINUED

OPTION 3: If f is supposed to represent some other object, then the notion of the size/length of f should reflect this.

Example. The size of an input to \mathcal{A} should involve the number of states in a minimal DFA for L_* .

This makes excellent sense,
but very much depends on context.

Convention: Unless we state otherwise, type-2 things will be of type $(\mathbb{N} \rightarrow \mathbb{N}) \times \mathbb{N} \rightarrow \mathbb{N}$.

SECOND-ORDER POLYNOMIALS

Example. Let $\text{Sum} = \lambda f, x. \sum_{i \leq |x|} f(i)$.

One can show that, for all $f: \mathbb{N} \rightarrow \mathbb{N}$ and all $x \in \mathbb{N}$,

$$|\text{Sum}(f, x)| \leq (|x| + 1) \cdot (|f|(|x|) + 1).$$

We want the RHS to be a “poly bound” in $|x|$ and $|f|$.

DEFINITION. (Syntax)

A **second-order polynomial** over type-0 vars y_0, \dots, y_m and type-1 vars g_0, \dots, g_n is an expression of the form

$$a \mid y_i \mid Q_1 + Q_2 \mid Q_1 + Q_2 \mid g_j(Q_1)$$

where $a \in \omega$ and Q_1 and Q_2 are 2nd order polys.

DEFINITION. (Semantics)

The **value** of a 2nd order polynomial on $f_0, \dots, f_m: \omega \rightarrow \omega$ and $x_0, \dots, x_m: \omega$ is ... **the obvious thing**.

PUTTING THE PIECES TOGETHER

DEFINITION. (Kapron-Cook) $F: (\mathbb{N} \rightarrow \mathbb{N}) \times \mathbb{N} \rightarrow \mathbb{N}$ is a **basic polynomial-time functional** iff

there is a OTM M and a 2nd order poly $Q \ni$
for all inputs (f, x) ,

(a) M outputs $F(f, x)$, and

(b) M runs within time $Q(|f|, |x|)$.

Puzzle: But $(f, x) \mapsto |f|(|x|)$ is not basic poly-time.

EXAMPLE. $\text{Sum} = \lambda f, x. \sum_{i \leq |x|} f(i)$ is computed by an OTM with the following “program.”

Input (f, x) ;

sum := 0;

For $i := 0, \dots, |x| - 1$ do

 sum := sum + f(i);

Output sum;

When the sketch is filled in, the OTM runs within time

$q(|x| + 1) \cdot (|f|(|x|) + 1)$,

where q is an ordinary polynomial.

MORE ON SECOND-ORDER POLYNOMIALS

DEFINITION

The **depth** of a 2nd-order polynomial Q is the max depth of nesting of applications in Q .

Examples.

$$\text{depth} \left(\mathbf{g}(y + 23) \right) = 1.$$

$$\text{depth} \left(\mathbf{g}_0 \left(\left(\mathbf{g}_0(2 \cdot y \cdot \mathbf{g}_1(y^2)) \right) + 6 \right)^3 \right) = 3.$$

DEFINITION

$$Q_{k,0}(\mathbf{g}, n) = k \cdot (n + 1)^k.$$

$$Q_{k,d+1}(\mathbf{g}, n) = k \cdot (\mathbf{g}(Q_{k,d}(\mathbf{g}, n)) + 1)^k.$$

LEMMA (Cofinality of the $Q_{k,d}$'s)

For each 2nd order polynomial Q (over a type-1 variable and a type-0 variable) there are k and $d \ni$

for all \mathbf{g} and n ,

$$Q(\mathbf{g}, n) \leq Q_{k,d}(\mathbf{g}, n).$$

POLYNOMIALLY CLOCKED OTMS

Suppose M is an OTM and $k, d \in \omega$.

We define an OTM $M_{k,d}$ that on input (f, x) :

- ▶ Runs M 's program step-by-step
- ▶ As it does this, it keeps updating

$cost$ = the current cost of M computation

$$b[0] = k \cdot (|x| + 1)^k.$$

$$b[i + 1] = \max\{ k \cdot (|a| + 1)^k \mid a \in A_i \}, \text{ where}$$

$$A_i = \left\{ \begin{array}{l} a = x \text{ or } a \text{ is the answer to a query} \\ \text{“}f(z) = ?\text{” with } |z| \leq b[i] \text{ that} \\ \text{was made at or before this point} \\ \text{in the } M\text{-computation} \end{array} \right\}.$$

where $i < d$.

- ▶ If $cost > b[d]$ after a set of updates,
then $M_{k,d}$ halts with output 0.
- ▶ If the M -computation halts with output \mathcal{A} ,
then $M_{d,k}$ also halts with output \mathcal{A} .

BACK TO THE PUZZLE

THEOREM (Seth, but implicit in Kapron-Cook)

(a) Each $M_{k,d}$ has a depth- d 2^{nd} order run time bound.

(b) If M has $Q_{k,d}$ as a run time bound,

then $M_{k,d}$ computes the same functional as M .

▶ So even though $(f, x) \mapsto |f|(|x|)$ is not basic ptime, using lower approxs. to $Q(|f|, |x|)$, as in $M_{k,d}$, suffices.

▶ Like the $M_{k,d}$'s, Angluin's \mathcal{A} is polynomial-time in what it has seen thus far.

▶ But \mathcal{A} fails to determine a basic poly time functional since there is no (finite depth) 2^{nd} order poly $Q \ni$

$$|\mathcal{A}(f)| \leq Q(|f|), \text{ for all } f$$

(even if we restrict to f such that $\mathcal{A}(f) \downarrow$).

CONNECTING TO THE OTHER ANALOGY

BFF_2

= the type-2 PV^ω computable functionals, where
 $PV^\omega \approx$ the simply-typed λ calculus + \mathcal{R}
(Cook-Urquhart)

= the type-2 BTLP computable functionals, where
BTLP = Bounded Typed Loop Programs
(Cook-Kapron)

⋮

= the basic polynomial-time functionals, where
these are the fntls computed by the $M_{k,d}$'s
(Kapron-Cook)

There are other notions of type-2 poly-time (e.g., BC),
but we move on to different territory ...

A PERILOUS TRANSIT

BEYOND TYPE-LEVEL 2

- ▶ It is a famously tricky territory.
 - New distinctions arise
 - Familiar notions turn strange
 - Depending on the semantics, h.t. parameters are no longer black boxes
- ▶ The examples are even rarer and they may not fit into neat categories.
 - E.g., oracle constructions
- ▶ The semantics and complexity have to be carefully meshed
 - We'll see examples below

Given these problems, why bother?

Because they are really good problems!

MACHINE-BASED H.T. COMPLEXITY

Anil Seth's Machine Models

D-machines

D = the (total) D-machine computable functionals over the full type-hierarchy

E-machines

E = the E-machine computable functionals over the full type-hierarchy

BFF = PV^ω computable functionals (CU)
= BTLP computable functionals (CK)
⋮
= **E** (Seth)

Seth conjectured **D** \neq **E**. (More on this later)

MY DIFFICULTIES WITH SETH'S MODELS

Seth's models and results are full of cleverness & insights, but:

- ▶ They are only partial analogues of the Kapron-Cook models.
 - no explicit higher-type analogue of length
 - no explicit higher-type analogue of polynomial
- ▶ They are tied to the full type-hierarchy.
 - Computability and complexity over the full type-hierarchy is problematic.
 - In particular, it is probably the wrong setting for **D**.
- ▶ It is very hard to see what **D** is about.

These complaints may reveal more
about my faults than Seth's.

- ▶ Below Seth's models are redeveloped using IKR tools.

SOME CONVENTIONS

Recall

- ▶ $\omega = \text{tallies} \equiv 0^*$.
- ▶ $\text{len}(x) = \text{length of dyadic representation of } x$
- ▶ $|x| = 0^{\text{len}(x)}$
- ▶ $|f|: \omega \rightarrow \omega$ and $|f|(n) = \max\{|f(x)| : |x| \leq n\}$.

We introduce

- ▶ $\text{Full}_\sigma = \text{all type-}\sigma \text{ fncts}$
- ▶ $\text{TCont}_\sigma = \text{the total, continuous type-}\sigma \text{ functionals}$
- ▶ $\text{TMon}_\sigma = \text{the hereditary total, monotone-increasing type-}\sigma \text{ functionals}$
(We use pointwise ordering on functions.)

Note: $|f| \in \text{TMon}_{\omega \rightarrow \omega}$

We'll try to work within TCont
... and end up working well within TCont

HIGHER TYPE LENGTH, A 1st ATTEMPT

- ▶ For $F: \mathbf{TCont}_{(\mathbf{N} \rightarrow \mathbf{N}) \rightarrow \mathbf{N}}$, lets try:

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

where $g \in \mathbf{TMon}_{\omega \rightarrow \omega}$.

- ▶ **Observation.**

Since $\{ f \mid |f| \leq g \}$ is compact in the Scott topology and since F is continuous, it follows that $|F|$ is total.

In fact $|F| \in \mathbf{TMon}_{(\omega \rightarrow \omega) \rightarrow \omega}$.

So far, so good.

- ▶ For $\Psi: \mathbf{TCont}_{((\mathbf{N} \rightarrow \mathbf{N}) \rightarrow \mathbf{N}) \rightarrow \mathbf{N}}$ lets try:

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

where $F: \mathbf{TCont}_{(\mathbf{N} \rightarrow \mathbf{N}) \rightarrow \mathbf{N}}$ and $G: \mathbf{TMon}_{(\omega \rightarrow \omega) \rightarrow \omega}$.

- ▶ **BUT**, $\{ F \mid |F| \leq G \}$ is not compact and there are Ψ that are unbounded on such sets.

Therefore, $|\Psi|$ is not total in general.

HIGHER TYPE LENGTH, A 2nd ATTEMPT

To Fix:

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

one can try to

1. Change $|F|$ so that $\{ F \mid |F| \leq G \}$ is compact.
2. Further restrict Ψ so that it is bounded on such sets.

We follow the second option here.

▶ σ : simple type over \mathbb{N} ▶ $|\sigma|_b$: corr. type over ω

▶ $|x|_b = |x|$. ▶ $\text{BCont}_{\mathbb{N}} = \mathbb{N}$. ▶ $\text{TLen}_{\omega} = \omega$.

▶ For $\sigma = \sigma_0 \times \cdots \times \sigma_n \rightarrow \mathbb{N}$, $f \in \text{TCont}_{\sigma}$,
 $\ell_i \in \text{TLen}_{|\sigma_i|_b}$ & $x_i \in \text{BCont}_{\sigma_i}$,

define:

$$|f|_b(\vec{\ell}) = \sup\{ |f(\vec{x})| \mid |x_i|_b \leq \ell_i \}.$$

BCont_{σ} = the f 's with $|f|_b$ total.

$\text{TLen}_{|\sigma|_b}$ = the total $|f|_b$'s.

BOUNDED CONTINUOUS FUNCTIONALS

- ▶ The BCont_σ 's are called the **bounded continuous functionals**.
- ▶ $|\cdot|_b$ is called the **bounded length**.
- ▶ For σ at type-levels 1 or 2,
 $\text{BCont}_\sigma = \text{TCont}_\sigma$ and $\text{TLen}_{|\sigma|_b} = \text{TMon}_{|\sigma|_b}$.
- ▶ For σ above type-level 2,
 $\text{BCont}_\sigma \subset \text{TCont}_\sigma$ and $\text{TLen}_{|\sigma|_b}$ has noncont. elms.
- ▶ In general we have
$$|x_0(x_1, \dots, x_k)|_b \leq |x_0|_b (|x_1|_b, \dots, |x_k|_b),$$
but not equality.

HIGHER TYPE POLYNOMIALS

EXAMPLE

Consider the bounded continuous Ψ given by:

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

We can show that

$$|\Psi(F, x)| \leq (|x| + 1) \cdot |F|_b (\lambda n . |x|).$$

We want the RHS to be a “poly bound” in $|x|$ and $|F|_b$.

DEFINITION: The Higher-Type Polynomials

Syntax

Simply typed λ -calculus

+ base type ω

+ plus and times

Semantics

$\llbracket - \rrbracket_{\text{TLen}}: \text{syntax} \rightarrow \text{TLen} \dots$ the obvious thing

Depth


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

MACHINES FOR HIGHER-TYPES

Recall the conventional formula for a complexity class

- \mathcal{C} = a **general** machine/cost model
- + a size measure for inputs ✓
 - + a family of bounding functions ✓

So, we have to specify the machine/cost model.

- ▶ We extend the type-2 OTM/answer-length model
- ▶ OTMs 
- ▶ The first question we have to answer is:
What should a higher-type query look like?

HIGHER-TYPE QUERIES

- ▶ In a P.L. setting a higher-type query corresponds to $\Psi(F, f, x)$ and $\Psi(\lambda g, y. \Psi(F, g, y), f, x)$ and the like.
- ▶ For our machines, we follow Kleene and use indices — from some “reasonable” indexing of h.t. machines
- ▶ So corresponding to $\Psi(F, f, x)$, we would have a query (i_F, i_f, x) to an oracle for Ψ .
- ▶ Note that $\Psi(\lambda g, y. \Psi(F, g, y), f, x)$ presents some challenges. (P.L. concerns start to show up here.)
- ▶ Note that both “ $\Psi(F, f, x)$ ” and “ (i_F, i_f, x) ” are offerings of syntax to gain the good will of the gods.
- ▶ However, the indices approach begs fewer (or at least different) questions.

H.T. QUERIES: THE FIRST DIFFICULTY

Let $\Phi: ((\mathbb{N} \rightarrow \mathbb{N}) \times \mathbb{N} \rightarrow \{0, 1\}) \times \mathbb{N} \rightarrow \{0, 1\}$ be \ni
 $\Phi(F, x) = F(\lambda y. 2^x, x)$.

To “feasibly” compute Φ :

On input F and x

Construct j_x , an index for $\lambda y. 2^x$ (* Cheap *)

Submit (j_x, x) to the oracle for F

Output the 0–1 result of this query

What is the problem?

Take $C \in (\text{DTIME}(2^n) - \text{P})$. So, $\chi_C \notin \text{PF}$.

Let $g_C: \mathbb{N}^2 \rightarrow \{0, 1\}$ be \ni $g_C \in \text{PF}$ &
 $g_C(2^x, x) = \chi_C(x)$, for all x .

Let $G: (\mathbb{N} \rightarrow \mathbb{N}) \times \mathbb{N} \rightarrow \{0, 1\}$ be \ni
 $G(f, x) = g_C(f(0), x)$.

Clearly G is basic feasible.

Hence, if Φ is “feasible,” then so is $\lambda x. \Phi(G, x) = \chi_C$.

But $\chi_C \notin \text{PF}!!!!!!$ (Kapron)

H.T. QUERIES: FEASIBILITY

Higher type query indices must name feasible functions.

- ▶ Our general model of computation can't be so general after all — **at least wrt h.t. queries.**
- ▶ So, before our first machine makes its first query, we have to have a notion of feasibility in place.
- ▶ Beyond type-level 2, the three ingredients
machine/cost + length + bounding functions
lose independence.
- ▶ We have to be careful about uniformity:
 $\lambda y. 2^x$ is $O(|y|)$ time computable.
- ▶ Seth's Solution: **Uniform polynomial bounds**
 - **Roughly**, for a given machine on arguments \vec{x} , and for each type τ
there is a “polynomial” Q_τ over
the lengths of the x 's (& maybe other params)
bounding the **size** of type- τ functions.

H.T. QUERIES: LEVELS

- ▶ Seth also ramifies machines/indices by **levels**
- ▶ An machine of level ℓ ($\in \omega$)
may use only query-indices of levels $< \ell$.
- ▶ \approx a prohibition on recursive calls.
- ▶ We have a pretty example justifying this restriction
...but not today.
- ▶ So, are we done yet?
Almost.

THE SIZE OF QUERIES

Recall the type-2 clocking scheme

- ▶ $M_{k,d}$ runs M 's program step-by-step and as it does this, it keeps updating

$cost$ = the current cost of M computation

$$b[0] = k \cdot (|x| + 1)^k.$$

$$b[i + 1] = \max\{ k \cdot (|a| + 1)^k \mid a \in A_i \}, \text{ where}$$

$$A_i = \left\{ \begin{array}{l} a = x \text{ or } a \text{ is the answer to a query} \\ \text{“}f(\vec{z}) = ?\text{” with } |z_1|, \dots, |z_c| \leq \\ b[i] \text{ that was made at or before this} \\ \text{point in the } M\text{-computation} \end{array} \right\}.$$

where $i < d$.

- ▶ If ever $cost > b[d]$, then $M_{k,d}$ halts with output 0.
- ▶ If the M -computation halts, then so does $M_{d,k}$ with the same output.

We want to do something similar for our h.t. machines.
But how should we bound query indices?

SIZES OF H.T. QUERIES

$b[i + 1] = \max\{ k \cdot (|a| + 1)^k \mid a \in A_i \}$, where

$$A_i = \left\{ \begin{array}{l} a = x \text{ or } a \text{ is the answer to a query} \\ \text{“} f(\vec{z}) = ? \text{” with } \dots, |z_j|, \dots \leq \\ b[i] \text{ that was made at or before this} \\ \text{point in the M-computation} \end{array} \right\}.$$

- ▶ Query indices are just names for h.t. functionals, so why bound the size of these names?
- ▶ These query indices are **already bounded** in the sense of **the uniform poly bounds & levels** restrictions.
- ▶ So, in the h.t. version of A_i , let us just include the type- \mathbb{N} queries in the $\dots, |z_j|, \dots$ list.
 - This leads directly to Seth’s **D-machines**.
 - \dots and clocked machines that run forever.
 - D-machines **are** total, if args are from **BCont**.
 - **Why?** The **already bounded** analogy works in **BCont**.
 - But the functions computed are still rather wild.

SIZES OF H.T. QUERIES, CONTINUED

$b[i + 1] = \max\{ k \cdot (|a| + 1)^k \mid a \in A_i \}$, where

$$A_i = \left\{ \begin{array}{l} a = x \text{ or } a \text{ is the answer to a query} \\ \text{“}f(\vec{z}) = ?\text{” with } \dots, |z_j|, \dots \leq \\ b[i] \text{ that was made at or before this} \\ \text{point in the M-computation} \end{array} \right\}.$$

- ▶ So, in the h.t. version of A_i , let us just include **all** queries in the $\dots, |z_j|, \dots$ list.
- ▶ This leads directly to Seth’s **E-machines**.
- ▶ E-machines compute exactly
 - **BFF**, under the **Full**-based semantics. (Seth)
 - **BCBFF**, under the **BCont**-based semantics. (IKR)
($\text{BCBFF} \subset \text{BCont}$)
- ▶ It also turns out that
 - $\text{BFF} \subsetneq \text{D}$ (IKR)
 - $\text{BCBFF} \subsetneq \text{BCD}$ (IKR)

TWO DISTINCT ANALOGIES ()

- ▶ Just like the three ingredients

machine/cost + length + bounding functions

the two analogies

Synthetic/Implicit/P.L.-Based/...

Analytic/Explicit/Machine-Based/...

begin to lose independence beyond type-level 2.

- ▶ So, to make progress here we need to blend the tools and insights from the

Logic/P.L./Formal Methods/...

Complexity/Combinatoric/Algorithmic/...

communities.

- ▶ This is not such a bad thing

... because from a broader perspective

they may not be (uniformly) so distinct either.

For refs: <http://www.cis.syr.edu/~royer/bib2.html>