Lecture 6
Theory of Real Approximation

Chee Yap
Courant Institute of Mathematical Sciences
New York University

Overview

What is the computational foundation of EGC? It is really a theory of real computation. We will introduce the basic elements of such a theory. We prove a transfer theorem that locates the central problem that must be solved in exact real computation.

- 0. Review
- I. Basics of Real Approximation
- II. Numerical Computational Model
- III. Transfer theorem

0. REVIEW

I. TOWARDS A THEORY OF REAL COMPUTATION

Dilemma of Real Computation

- Standard Complexity Theory
  - Turing machines, countable domain
  - Does not work for uncountable domain!
  - Whiteboard Aside: Describe simple Turing machines
- Smale:
  - "There is not even a formal definition of algorithm in Numerical Analysis." [BCSS, p.23]
  - "Towards resolving the problem [conflict between continuous and discrete] we are led to .. allow real numbers as inputs" [BCSS, p.23]

Dilemma of Real Computation

- Standard Complexity Theory
  - Turing machines, countable domain
  - Does not work for uncountable domain!
  - Whiteboard Aside: Describe simple Turing machines
- Smale:
  - "There is not even a formal definition of algorithm in Numerical Analysis." [BCSS, p.23]
  - "Towards resolving the problem [conflict between continuous and discrete] we are led to .. allow real numbers as inputs" [BCSS, p.23]

Dilemma of Real Computation

- Standard Complexity Theory
  - Turing machines, countable domain
  - Does not work for uncountable domain!
  - Whiteboard Aside: Describe simple Turing machines
- Smale:
  - "There is not even a formal definition of algorithm in Numerical Analysis." [BCSS, p.23]
  - "Towards resolving the problem [conflict between continuous and discrete] we are led to .. allow real numbers as inputs" [BCSS, p.23]

Dilemma of Real Computation

- Standard Complexity Theory
  - Turing machines, countable domain
  - Does not work for uncountable domain!
  - Whiteboard Aside: Describe simple Turing machines
- Smale:
  - "There is not even a formal definition of algorithm in Numerical Analysis." [BCSS, p.23]
  - "Towards resolving the problem [conflict between continuous and discrete] we are led to .. allow real numbers as inputs" [BCSS, p.23]

Two Approaches to Real Computation

- Algebraic Approach (Smale, et al)
  - ∗ Real numbers are directly represented as atomic objects, and can be compared without error
  - ∗ Algebraic operators can be carried out without error
  - ∗ Whiteboard Aside: Straightline model augmented with loops and access to infinite array

- Analytic Approach (Weihrauch, etc)
  - ∗ Real numbers are represented by Cauchy sequences
  - ∗ Whiteboard Aside: Extend Turing machines to input and output infinite sequences

Two Approaches to Real Computation

- Algebraic Approach (Smale, et al)
  * Real numbers are directly represented as atomic objects, and can be compared without error
  * Algebraic operators can be carried out without error
  * Whiteboard Aside: Straightline model augmented with loops and access to infinite array

- Analytic Approach (Weihrauch, etc)
  * Real numbers are represented by Cauchy sequences
  * Whiteboard Aside: Extend Turing machines to input and output infinite sequences

Two Approaches to Real Computation

- Algebraic Approach (Smale, et al)
  - * Real numbers are directly represented as atomic objects, and can be compared without error
  - * Algebraic operators can be carried out without error
  - * Whiteboard Aside: Straightline model augmented with loops and access to infinite array

- Analytic Approach (Weihrauch, etc)
  - * Real numbers are represented by Cauchy sequences
  - * Whiteboard Aside: Extend Turing machines to input and output infinite sequences

Two Approaches to Real Computation

- Algebraic Approach (Smale, et al)
  - ∗ Real numbers are directly represented as atomic objects, and can be compared without error
  - ∗ Algebraic operators can be carried out without error
  - ∗ Whiteboard Aside: Straightline model augmented with loops and access to infinite array

- Analytic Approach (Weihrauch, etc)
  - ∗ Real numbers are represented by Cauchy sequences
  - ∗ Whiteboard Aside: Extend Turing machines to input and output infinite sequences

Two Approaches to Real Computation

- Algebraic Approach (Smale, et al)
  * Real numbers are directly represented as atomic objects, and can be compared without error
  * Algebraic operators can be carried out without error
  * Whiteboard Aside: Straightline model augmented with loops and access to infinite array

- Analytic Approach (Weihrauch, etc)
  * Real numbers are represented by Cauchy sequences
  * Whiteboard Aside: Extend Turing machines to input and output infinite sequences

- Criticisms (see [Weihrauch] or [Traub])
  - ∗ Real numbers are arbitrarily complex What about the analytic approach?

- Problems from our viewpoint:
  - ∗ Zero Problem is trivial in Algebraic Approach
  - ∗ Zero Problem is undecidable in Analytic Approach

- Criticisms (see [Weihrauch] or [Traub])
  - Real numbers are arbitrarily complex

What about the analytic approach?

- Problems from our viewpoint:
  - Zero Problem is trivial in Algebraic Approach
  - Zero Problem is undecidable in Analytic Approach

- Criticisms (see [Weihrauch] or [Traub])
  - Real numbers are arbitrarily complex

What about the analytic approach?

- Problems from our viewpoint:
  - Zero Problem is trivial in Algebraic Approach
  - Zero Problem is undecidable in Analytic Approach

How We Solve Numerical Problems??

- E.g., Solving PDE model, Numerical Optimization Problem, etc
- STEP A:
  - Design an ideal Algorithm A
  - Assume certain operations such as $\pm, \times, \exp()$
- STEP B:
  - Implements Algorithm A as a Numerical Program B
  - Accounts for numerical representation, errors, etc

How We Solve Numerical Problems??

- E.g., Solving PDE model, Numerical Optimization Problem, etc
- STEP A:
  - $*$ Design an ideal Algorithm A
  - $*$ Assume certain operations such as $\pm, \times, \exp()$
- STEP B:
  - $*$ Implements Algorithm A as a Numerical Program B
  - $*$ Accounts for numerical representation, errors, etc

How We Solve Numerical Problems??

- E.g., Solving PDE model, Numerical Optimization Problem, etc

- STEP A:
  - Design an ideal Algorithm A
  - Assume certain operations such as $\pm, \times, \exp()$

- STEP B:
  - Implements Algorithm A as a Numerical Program B
  - Accounts for numerical representation, errors, etc

How We Solve Numerical Problems??

- E.g., Solving PDE model, Numerical Optimization Problem, etc
- STEP A:
  - Design an ideal Algorithm A
  - Assume certain operations such as $\pm, \times, \exp()$
- STEP B:
  - Implements Algorithm A as a Numerical Program B
  - Accounts for numerical representation, errors, etc

How We Solve Numerical Problems??

- E.g., Solving PDE model, Numerical Optimization Problem, etc
- STEP A:
  - Design an ideal Algorithm A
  - Assume certain operations such as $\pm, \times, \exp()$
- STEP B:
  - Implements Algorithm A as a Numerical Program B
  - Accounts for numerical representation, errors, etc

How We Solve Numerical Problems??

- E.g., Solving PDE model, Numerical Optimization Problem, etc

- STEP A:
  - Design an ideal Algorithm A
  - Assume certain operations such as $\pm, \times, \exp()$

- STEP B:
  - Implements Algorithm A as a Numerical Program B
  - Accounts for numerical representation, errors, etc

How We Solve Numerical Problems??

- E.g., Solving PDE model, Numerical Optimization Problem, etc
- STEP A:
  - ∗ Design an ideal Algorithm A
  - ∗ Assume certain operations such as $\pm, \times, \exp()$
- STEP B:
  - ∗ Implements Algorithm A as a Numerical Program B
  - ∗ Accounts for numerical representation, errors, etc

How We Solve Numerical Problems??

- E.g., Solving PDE model, Numerical Optimization Problem, etc
- STEP A:
  - ∗ Design an ideal Algorithm A
  - ∗ Assume certain operations such as $\pm, \times, \exp()$
- STEP B:
  - ∗ Implements Algorithm A as a Numerical Program B
  - ∗ Accounts for numerical representation, errors, etc

What is the Abstract View?

- Step A:
  - ∗ Algorithm A belongs to an Algebraic Model (e.g., BSS)
  - ∗ Basis $\Omega = \{\pm, \times, \exp(), \ldots\}$
- Step B:
  - ∗ Program B belongs to ...?
  - ∗ See below – numerical pointer machines
- Critical Questions:
  - ∗ Can Algorithm A be implemented by some Program B?
  - ∗ Wanted: a Transfer Theorem!

What is the Abstract View?

- Step A:
  - ∗ Algorithm A belongs to an Algebraic Model (e.g., BSS)
  - ∗ Basis $\Omega = \{\pm, \times, \exp(), \ldots\}$
- Step B:
  - ∗ Program B belongs to ...?
  - ∗ See below – numerical pointer machines
- Critical Questions:
  - ∗ Can Algorithm A be implemented by some Program B?
  - ∗ Wanted: a Transfer Theorem!

What is the Abstract View?

- Step A:
  - Algorithm A belongs to an Algebraic Model (e.g., BSS)
  - Basis $\Omega = \{\pm, \times, \exp(), ...\}$
- Step B:
  - Program B belongs to ...?
  - See below – numerical pointer machines
- Critical Questions:
  - Can Algorithm A be implemented by some Program B?
  - Wanted: a Transfer Theorem!

What is the Abstract View?

- Step A:
  - ∗ Algorithm A belongs to an Algebraic Model (e.g., BSS)
  - ∗ Basis $\Omega = \{\pm, \times, \exp(), \ldots\}$
- Step B:
  - ∗ Program B belongs to ...?
  - ∗ See below – numerical pointer machines
- Critical Questions:
  - ∗ Can Algorithm A be implemented by some Program B?
  - ∗ Wanted: a Transfer Theorem!

What is the Abstract View?

- Step A:
  - $*$ Algorithm A belongs to an Algebraic Model (e.g., BSS)
  - $*$ Basis $\Omega = \{\pm, \times, \exp(), \ldots\}$
- Step B:
  - $*$ Program B belongs to ...?
  - $*$ See below – numerical pointer machines
- Critical Questions:
  - $*$ Can Algorithm A be implemented by some Program B?
  - $*$ Wanted: a Transfer Theorem!

What is the Abstract View?

- Step A:
  - ∗ Algorithm A belongs to an Algebraic Model (e.g., BSS)
  - ∗ Basis $\Omega = \{\pm, \times, \exp(), ...\}$
- Step B:
  - ∗ Program B belongs to ...?
  - ∗ See below – numerical pointer machines
- Critical Questions:
  - ∗ Can Algorithm A be implemented by some Program B?
  - ∗ Wanted: a Transfer Theorem!

What is the Abstract View?

- Step A:
  - ∗ Algorithm A belongs to an Algebraic Model (e.g., BSS)
  - ∗ Basis $\Omega = \{\pm, \times, \exp(), ...\}$
- Step B:
  - ∗ Program B belongs to ...?
  - ∗ See below – numerical pointer machines
- Critical Questions:
  - ∗ Can Algorithm A be implemented by some Program B?
  - ∗ Wanted: a Transfer Theorem!

What is the Abstract View?

- Step A:
  - $*$ Algorithm A belongs to an Algebraic Model (e.g., BSS)
  - $*$ Basis $\Omega = \{\pm, \times, \exp(), ...\}$
- Step B:
  - $*$ Program B belongs to ...?
  - $*$ See below – numerical pointer machines
- Critical Questions:
  - $*$ Can Algorithm A be implemented by some Program B?
  - $*$ Wanted: a Transfer Theorem!

What is the Abstract View?

- Step A:
  - Algorithm A belongs to an Algebraic Model (e.g., BSS)
  - Basis $\Omega = \{\pm, \times, \exp(), ...\}$
- Step B:
  - Program B belongs to ...?
  - See below – numerical pointer machines
- Critical Questions:
  - Can Algorithm A be implemented by some Program B?
  - Wanted: a Transfer Theorem!

What is the Abstract View?

- Step A:
  - Algorithm A belongs to an Algebraic Model (e.g., BSS)
  - Basis $\Omega = \{\pm, \times, \exp(), ...\}$
- Step B:
  - Program B belongs to ...?
  - See below – numerical pointer machines
- Critical Questions:
  - Can Algorithm A be implemented by some Program B?
  - Wanted: a Transfer Theorem!

Representable Reals

- Representation of reals is critical starting point
  - cf. Analytic or Algebraic Approaches
- Axioms for the set $\mathbb{F}$ of representable reals
  - $\mathbb{F}$ is a countable set dense subset of $\mathbb{R}$
  - $\mathbb{F}$ is a ring extension of $\mathbb{Z}$
  - $\mathbb{F}$ can be represented efficiently
  - Comparisons and Ring operations are polynomial-time in this representation
- E.g., $\mathbb{F}$ can be taken to be $\mathbb{Q}$ or bigfloats
- PRINCIPLE: all output and input of our

Representable Reals

- Representation of reals is critical starting point
  - ∗ cf. Analytic or Algebraic Approaches
- Axioms for the set $\mathbb{F}$ of representable reals
  - ∗ $\mathbb{F}$ is a countable set dense subset of $\mathbb{R}$
  - ∗ $\mathbb{F}$ is a ring extension of $\mathbb{Z}$
  - ∗ $\mathbb{F}$ can be represented efficiently
  - ∗ Comparisons and Ring operations are polynomial-time in this representation
- E.g., $\mathbb{F}$ can be taken to be $\mathbb{Q}$ or bigfloats
- PRINCIPLE: all output and input of our

Representable Reals

- Representation of reals is critical starting point
  - cf. Analytic or Algebraic Approaches

- Axioms for the set $\mathbb{F}$ of representable reals
  - $\mathbb{F}$ is a countable set dense subset of $\mathbb{R}$
  - $\mathbb{F}$ is a ring extension of $\mathbb{Z}$
  - $\mathbb{F}$ can be represented efficiently
  - Comparisons and Ring operations are polynomial-time in this representation

- E.g., $\mathbb{F}$ can be taken to be $\mathbb{Q}$ or bigfloats

- PRINCIPLE: all output and input of our

Representable Reals

- Representation of reals is critical starting point
  - ∗ cf. Analytic or Algebraic Approaches
- Axioms for the set $\mathbb{F}$ of representable reals
  - ∗ $\mathbb{F}$ is a countable set dense subset of $\mathbb{R}$
  - ∗ $\mathbb{F}$ is a ring extension of $\mathbb{Z}$
  - ∗ $\mathbb{F}$ can be represented efficiently
  - ∗ Comparisons and Ring operations are polynomial-time in this representation
- E.g., $\mathbb{F}$ can be taken to be $\mathbb{Q}$ or bigfloats
- PRINCIPLE: all output and input of our

Representable Reals

- Representation of reals is critical starting point
  - * cf. Analytic or Algebraic Approaches
- Axioms for the set $\mathbb{F}$ of representable reals
  - * $\mathbb{F}$ is a countable set dense subset of $\mathbb{R}$
  - * $\mathbb{F}$ is a ring extension of $\mathbb{Z}$
  - * $\mathbb{F}$ can be represented efficiently
  - * Comparisons and Ring operations are polynomial-time in this representation
- E.g., $\mathbb{F}$ can be taken to be $\mathbb{Q}$ or bigfloats
- PRINCIPLE: all output and input of our

Representable Reals

- Representation of reals is critical starting point
  - $*$ cf. Analytic or Algebraic Approaches
- Axioms for the set $\mathbb{F}$ of representable reals
  - $*$ $\mathbb{F}$ is a countable set dense subset of $\mathbb{R}$
  - $*$ $\mathbb{F}$ is a ring extension of $\mathbb{Z}$
  - $*$ $\mathbb{F}$ can be represented efficiently
  - $*$ Comparisons and Ring operations are polynomial-time in this representation
- E.g., $\mathbb{F}$ can be taken to be $\mathbb{Q}$ or bigfloats
- PRINCIPLE: all output and input of our

Representable Reals

- Representation of reals is critical starting point
  - cf. Analytic or Algebraic Approaches
- Axioms for the set $\mathbb{F}$ of representable reals
  - $\mathbb{F}$ is a countable set dense subset of $\mathbb{R}$
  - $\mathbb{F}$ is a ring extension of $\mathbb{Z}$
  - $\mathbb{F}$ can be represented efficiently
  - Comparisons and Ring operations are polynomial-time in this representation
- E.g., $\mathbb{F}$ can be taken to be $\mathbb{Q}$ or bigfloats
- PRINCIPLE: all output and input of our

Representable Reals

- Representation of reals is critical starting point
  - cf. Analytic or Algebraic Approaches
- Axioms for the set $\mathbb{F}$ of representable reals
  - $\mathbb{F}$ is a countable set dense subset of $\mathbb{R}$
  - $\mathbb{F}$ is a ring extension of $\mathbb{Z}$
  - $\mathbb{F}$ can be represented efficiently
  - Comparisons and Ring operations are polynomial-time in this representation
- E.g., $\mathbb{F}$ can be taken to be $\mathbb{Q}$ or bigfloats
- PRINCIPLE: all output and input of our

computation must be representable numbers

* HENCE: We can use Turing machines for our real computations
* HENCE: We can only talk about approximating a real function f
* HENCE: we do not worry about behavior of f at non-representable inputs
* Unlike the analytic or algebraic approach, we deliberately avoid representing all real numbers!

computation must be representable numbers

* HENCE: We can use Turing machines for our real computations

* HENCE: We can only talk about approximating a real function f

* HENCE: we do not worry about behavior of f at non-representable inputs

* Unlike the analytic or algebraic approach, we deliberately avoid representing all real numbers!

Theory of Real Approximation

- NOTATION: given $f : \mathbb{R} \longrightarrow \mathbb{R}$
  - let $\mathcal{A}f$ denote any function $\mathcal{A}f : \mathbb{F} \times \mathbb{F} \longrightarrow \mathbb{F}$ such that $|\mathcal{A}f(x,p) - f(x)| \leq 2^{-p}$
  - let $\mathcal{R}f$ denote any function $\mathcal{R}f : \mathbb{F} \times \mathbb{F} \longrightarrow \mathbb{F}$ such that $|\mathcal{R}f(x,p) - f(x)| \leq 2^{-p}|f(x)|$
- DEFINE: a real function f is absolutely approximable if $\mathcal{A}f$ is computable by a Turing Machine
  - Similarly, define relatively approximable if $\mathcal{R}f$ is computable by a Turing machine
- DEFINE: $Zero(f) = \{x \in \mathbb{F} : f(x) = 0\}$

Theory of Real Approximation

- NOTATION: given $f : \mathbb{R} \longrightarrow \mathbb{R}$

  $*$ let $\mathcal{A}f$ denote any function $\mathcal{A}f : \mathbb{F} \times \mathbb{F} \longrightarrow \mathbb{F}$ such that $|\mathcal{A}f(x,p) - f(x)| \leq 2^{-p}$

  $*$ let $\mathcal{R}f$ denote any function $\mathcal{R}f : \mathbb{F} \times \mathbb{F} \longrightarrow \mathbb{F}$ such that $|\mathcal{R}f(x,p) - f(x)| \leq 2^{-p}|f(x)|$

- DEFINE: a real function f is absolutely approximable if $\mathcal{A}f$ is computable by a Turing Machine

  $*$ Similarly, define relatively approximable if $\mathcal{R}f$ is computable by a Turing machine

- DEFINE: $Zero(f) = \{x \in \mathbb{F} : f(x) = 0\}$

Theory of Real Approximation

- NOTATION: given $f : \mathbb{R} \longrightarrow \mathbb{R}$
  - let $\mathcal{A}f$ denote any function $\mathcal{A}f : \mathbb{F} \times \mathbb{F} \longrightarrow \mathbb{F}$ such that $|\mathcal{A}f(x,p) - f(x)| \leq 2^{-p}$
  - let $\mathcal{R}f$ denote any function $\mathcal{R}f : \mathbb{F} \times \mathbb{F} \longrightarrow \mathbb{F}$ such that $|\mathcal{R}f(x,p) - f(x)| \leq 2^{-p}|f(x)|$
- DEFINE: a real function f is absolutely approximable if $\mathcal{A}f$ is computable by a Turing Machine
  - Similarly, define relatively approximable if $\mathcal{R}f$ is computable by a Turing machine
- DEFINE: $Zero(f) = \{x \in \mathbb{F} : f(x) = 0\}$

Theory of Real Approximation

- NOTATION: given $f : \mathbb{R} \longrightarrow \mathbb{R}$

  * let $\mathcal{A}f$ denote **any** function $\mathcal{A}f : \mathbb{F} \times \mathbb{F} \longrightarrow \mathbb{F}$ such that $|\mathcal{A}f(x,p) - f(x)| \leq 2^{-p}$

  * let $\mathcal{R}f$ denote **any** function $\mathcal{R}f : \mathbb{F} \times \mathbb{F} \longrightarrow \mathbb{F}$ such that $|\mathcal{R}f(x,p) - f(x)| \leq 2^{-p}|f(x)|$

- DEFINE: a real function f is absolutely approximable if $\mathcal{A}f$ is computable by a Turing Machine

  * Similarly, define **relatively approximable** if $\mathcal{R}f$ is computable by a Turing machine

- DEFINE: $Zero(f) = \{x \in \mathbb{F} : f(x) = 0\}$

Theory of Real Approximation

- NOTATION: given $f : \mathbb{R} \longrightarrow \mathbb{R}$
  - $*$ let $\mathcal{A}f$ denote **any** function $\mathcal{A}f : \mathbb{F} \times \mathbb{F} \longrightarrow \mathbb{F}$ such that $|\mathcal{A}f(x,p) - f(x)| \leq 2^{-p}$
  - $*$ let $\mathcal{R}f$ denote **any** function $\mathcal{R}f : \mathbb{F} \times \mathbb{F} \longrightarrow \mathbb{F}$ such that $|\mathcal{R}f(x,p) - f(x)| \leq 2^{-p}|f(x)|$
- DEFINE: a real function f is absolutely approximable if $\mathcal{A}f$ is computable by a Turing Machine
  - $*$ Similarly, define **relatively approximable** if $\mathcal{R}f$ is computable by a Turing machine
- DEFINE: $Zero(f) = \{x \in \mathbb{F} : f(x) = 0\}$

∗ The Zero Problem for f is to decide the set $Zero(f)$

- Computation of partial functions

∗ We assume that the Turing machine detect undefined inputs

* The Zero Problem for f is to decide the set $Zero(f)$

- Computation of partial functions
  * We assume that the Turing machine detect undefined inputs

* The **Zero Problem** for f is to decide the set $Zero(f)$

- Computation of partial functions
  * We assume that the Turing machine detect undefined inputs

Basic Properties

- THEOREM A:

  * f is relatively approximable iff f is absolutely approximable and $Zero(f)$ is decidable.

- THEOREM B:

  * There is a function f_0 that is absolutely approximable in polynomial time, but f_0 is not relatively approximable.

- THEOREM C [with C.O'Dunlaing]:

  * There exist functions g_0, h_0 that are relatively approximable in polynomial time, but $g_0 \circ h_0$ is not absolutely approximable.

Basic Properties

- THEOREM A:

  * f is relatively approximable iff f is absolutely approximable and $Zero(f)$ is decidable.

- THEOREM B:

  * There is a function f_0 that is absolutely approximable in polynomial time, but f_0 is not relatively approximable.

- THEOREM C [with C.O'Dunlaing]:

  * There exist functions g_0, h_0 that are relatively approximable in polynomial time, but $g_0 \circ h_0$ is not absolutely approximable.

Basic Properties

- THEOREM A:

  $*$ f is relatively approximable iff f is absolutely approximable and $Zero(f)$ is decidable.

- THEOREM B:

  $*$ There is a function f_0 that is absolutely approximable in polynomial time, but f_0 is not relatively approximable.

- THEOREM C [with C.O'Dunlaing]:

  $*$ There exist functions g_0, h_0 that are relatively approximable in polynomial time, but $g_0 \circ h_0$ is not absolutely approximable.

Basic Properties

- THEOREM A:

  $*$ f is relatively approximable iff f is absolutely approximable and $Zero(f)$ is decidable.

- THEOREM B:

  $*$ There is a function f_0 that is absolutely approximable in polynomial time, but f_0 is not relatively approximable.

- THEOREM C [with C.O'Dunlaing]:

  $*$ There exist functions g_0, h_0 that are relatively approximable in polynomial time, but $g_0 \circ h_0$ is not absolutely approximable.

- Whiteboard Aside: Do Proofs.

- Whiteboard Aside: Do Proofs.

Proofs

- THEOREM A:

  $*$ Let f be relatively approximable. Then $x \in Zero(f)$ iff $\mathcal{R}f(x,1) = 0$. Also, $\mathcal{A}f(x,p)$ can be computed by computing $y = \mathcal{R}f(x,1)$, $z = \lceil \lg y \rceil$ and finally set $\mathcal{A}f(x,p) \leftarrow \mathcal{R}f(x, z+p+1)$.

  $*$ Let $\mathcal{A}f$ be computable and $Zero(f)$ decidable. To compute $\mathcal{R}f(x,p)$, we output 0 iff $x \in Zero(f)$. Otherwise we compute $\mathcal{A}f(x,i)$ in the ith step, stopping when $\mathcal{A}f(x,i) \geq 2^{-i+1}$. This implies $|f(x)| \geq 2^{i}$. We then set $\mathcal{R}f(x,p) \leftarrow \mathcal{A}f(x, i+p)$. The correctness follows from $|f(x)| \geq 2^{-i}$ and hence $|\mathcal{A}f(x,i+p) - f(x)| \leq 2^{-i-p} \leq |f(x)|2^{-p}$.

- THEOREM B:

  $*$ Let $t(n)$ be the number of steps that the nth Turing machine M_n takes, on input n. So $t(n) = \infty$ if when $M_n(n)$ does not halt

  $*$ DEFINE $f_0(n) = 1/t(n)$ where $1/\infty = 0$. NOTE that $Zero(f_0)$ is the diagonal set in recursive function theory, usually denoted K.

$*$ CLAIM: f_0 is absolutely approximable

$*$ Proof: on input n, p, check that $n \in \mathbb{N}$ and then simulate $M_n(n)$ for $\lceil p \rceil$ steps. If $M_n(n)$ halt in $k \le \lceil p \rceil$ steps, we output $1/k$ (with absolute error at most 2^{-p}). Else we output 0.

$*$ CLAIM: f_0 is not relatively approximable

$*$ Proof: if it is, then $Zero(f_0) = K$ would be decidable. Contradiction

- LEMMA:

  $*$ If a function $f : \mathbb{R} \to \mathbb{R}$ is never 0, then then $\mathcal{A}f$ is computable iff $\mathcal{R}f$ is computable

  $*$ Proof: One direction is immediate from Theorem A. In the other direction, suppose $\mathcal{A}f$ is computable. Then we can compute $\mathcal{R}f(x, p)$ using $\mathcal{A}f$ as in theorem A, because we know $f(x) \neq 0$.

- THEOREM C:

  $*$ Define g_0 and h_0 via $g_0(x) = \mathtt{sign}(x - 1)$ and $h_0(x) = 1 + f_0(x)$ where f_0 is from proof of Theorem A.

  $*$ The function $g_0(x)$ is relatively approximable

  $*$ The function h_0 is relatively approximable, by above

LEMMA

* But $g_0 \circ h_0(x) = \mathtt{sign}(f_0(x))$ is not absolutely approximable:
* If it were absolutely approximable by some function F, then we can decide K: if $x \in K$ iff $\mathcal{A}F(x, 2) \leq 1/2$

Transfer Theorem

- THEOREM D: The following are equivalent:
  - $*$ (I) Val_Ω is relatively approximable over Ω
  - $*$ (II) For all problems F, if F is Ω-computable (ideal model!) then F is relative Ω-approximable (implementation model!).
- Thus Val_Ω is "universal" (or "complete").
  - $*$ Our computational scientist ought to choose his set Ω carefully
- Rest of talk is to formalize this theorem!

Transfer Theorem

- THEOREM D: The following are equivalent:
    - (I) Val_Ω is relatively approximable over Ω
    - (II) For all problems F, if F is Ω-computable (ideal model!) then F is relative Ω-approximable (implementation model!).
- Thus Val_Ω is "universal" (or "complete").
    - Our computational scientist ought to choose his set Ω carefully
- Rest of talk is to formalize this theorem!

Transfer Theorem

- THEOREM D: The following are equivalent:
  - (I) Val_Ω is relatively approximable over Ω
  - (II) For all problems F, if F is Ω-computable (ideal model!) then F is relative Ω-approximable (implementation model!).
- Thus Val_Ω is "universal" (or "complete").
  - Our computational scientist ought to choose his set Ω carefully
- Rest of talk is to formalize this theorem!

Transfer Theorem

- THEOREM D: The following are equivalent:

  * (I) Val_Ω is relatively approximable over Ω

  * (II) For all problems F, if F is Ω-computable (ideal model!) then F is relative Ω-approximable (implementation model!).

- Thus Val_Ω is "universal" (or "complete").

  * Our computational scientist ought to choose his set Ω carefully

- Rest of talk is to formalize this theorem!

Transfer Theorem

- THEOREM D: The following are equivalent:
  - (I) Val_Ω is relatively approximable over Ω
  - (II) For all problems F, if F is Ω-computable (ideal model!) then F is relative Ω-approximable (implementation model!).
- Thus Val_Ω is "universal" (or "complete").
  - Our computational scientist ought to choose his set Ω carefully
- Rest of talk is to formalize this theorem!

Transfer Theorem

- THEOREM D: The following are equivalent:
  - (I) Val_Ω is relatively approximable over Ω
  - (II) For all problems F, if F is Ω-computable (ideal model!) then F is relative Ω-approximable (implementation model!).
- Thus Val_Ω is "universal" (or "complete").
  - Our computational scientist ought to choose his set Ω carefully
- Rest of talk is to formalize this theorem!

Transfer Theorem

- THEOREM D: The following are equivalent:
  - (I) Val_Ω is relatively approximable over Ω
  - (II) For all problems F, if F is Ω-computable (ideal model!) then F is relative Ω-approximable (implementation model!).
- Thus Val_Ω is "universal" (or "complete").
  - Our computational scientist ought to choose his set Ω carefully
- Rest of talk is to formalize this theorem!

Pointer Machine

- Schönhage's storage modification machine (1978)
- Fix a finite set Δ of "colors"
- A Δ-graph $G = (V, E)$ is a finite digraph of out-degree $|\Delta|$, where each the edges out of each node has a unique color. One node is the origin.
- So any word $w \in \Delta^*$ identifies a unique node $[w]_G$ of G. Call edges of G a "pointer"
- Pointer Assignment: $w \leftarrow w'$

  $*$ This transforms G to G' by making at most one pointer modification so that $[w]_{G'} = [w']_G$

Pointer Machine

- Schönhage's storage modification machine (1978)
- Fix a finite set Δ of "colors"
- A Δ-graph $G = (V, E)$ is a finite digraph of out-degree $|\Delta|$, where each the edges out of each node has a unique color. One node is the origin.
- So any word $w \in \Delta^*$ identifies a unique node $[w]_G$ of G. Call edges of G a "pointer"
- Pointer Assignment: $w \leftarrow w'$

  $*$ This transforms G to G' by making at most one pointer modification so that $[w]_{G'} = [w']_G$

Pointer Machine

- Schönhage's storage modification machine (1978)
- Fix a finite set Δ of "colors"
- A Δ-graph $G = (V, E)$ is a finite digraph of out-degree $|\Delta|$, where each the edges out of each node has a unique color. One node is the origin.
- So any word $w \in \Delta^*$ identifies a unique node $[w]_G$ of G. Call edges of G a "pointer"
- Pointer Assignment: $w \leftarrow w'$

  $*$ This transforms G to G' by making at most one pointer modification so that $[w]_{G'} = [w']_G$

Pointer Machine

- Schönhage's storage modification machine (1978)
- Fix a finite set Δ of "colors"
- A Δ-graph $G = (V, E)$ is a finite digraph of out-degree $|\Delta|$, where each the edges out of each node has a unique color. One node is the origin.
- So any word $w \in \Delta^*$ identifies a unique node $[w]_G$ of G. Call edges of G a "pointer"
- Pointer Assignment: $w \leftarrow w'$

  $*$ This transforms G to G' by making at most one pointer modification so that $[w]_{G'} = [w']_G$

Pointer Machine

- Schönhage's storage modification machine (1978)
- Fix a finite set Δ of "colors"
- A Δ-graph $G = (V, E)$ is a finite digraph of out-degree $|\Delta|$, where each the edges out of each node has a unique color. One node is the origin.
- So any word $w \in \Delta^*$ identifies a unique node $[w]_G$ of G. Call edges of G a "pointer"
- Pointer Assignment: $w \leftarrow w'$
  - $*$ This transforms G to G' by making at most one pointer modification so that $[w]_{G'} = [w']_G$

Pointer Machine

- Schönhage's storage modification machine (1978)
- Fix a finite set Δ of "colors"
- A Δ-graph $G = (V, E)$ is a finite digraph of out-degree $|\Delta|$, where each the edges out of each node has a unique color. One node is the origin.
- So any word $w \in \Delta^*$ identifies a unique node $[w]_G$ of G. Call edges of G a "pointer"
- Pointer Assignment: $w \leftarrow w'$

  $*$ This transforms G to G' by making at most one pointer modification so that $[w]_{G'} = [w']_G$

- A pointer machine M is specified by a sequence of instructions of the form
  - Assignment: $w \leftarrow w'$
  - Test: IF $(w \equiv w')$ GOTO(L) where L is a label
  - Termination: HALT

- Clearly, a pointer machine can simulate each step of a multitape Turing machine in $O(1)$ steps
  - Need to encode the contents of Turing machine tape cell

- Input/Output: all are conventions
  - What does a pointer machine compute? Let $\mathcal{G}_\Delta$ be set of Δ-graphs

- A pointer machine M is specified by a sequence of instructions of the form
  - $*$ Assignment: $w \leftarrow w'$
  - $*$ Test: IF $(w \equiv w')$ GOTO(L) where L is a label
  - $*$ Termination: HALT

- Clearly, a pointer machine can simulate each step of a multitape Turing machine in $O(1)$ steps
  - $*$ Need to encode the contents of Turing machine tape cell

- Input/Output: all are conventions
  - $*$ What does a pointer machine compute? Let $\mathcal{G}_\Delta$ be set of Δ-graphs

- A pointer machine M is specified by a sequence of instructions of the form
  - Assignment: $w \leftarrow w'$
  - Test: IF $(w \equiv w')$ GOTO(L) where L is a label
  - Termination: HALT

- Clearly, a pointer machine can simulate each step of a multitape Turing machine in $O(1)$ steps
  - Need to encode the contents of Turing machine tape cell

- Input/Output: all are conventions
  - What does a pointer machine compute? Let $\mathcal{G}_\Delta$ be set of Δ-graphs

  * It computes $f : \mathcal{G}_\Delta \longrightarrow \mathcal{G}_\Delta$ (partial)

- Discussion: pointer machines are more robust than Turing machines
  * Cf: evaluation problem, bigfloat number truncation

* It computes $f : \mathcal{G}_\triangle \rightarrow \mathcal{G}_\triangle$ (partial)

- Discussion: pointer machines are more robust than Turing machines
  * Cf: evaluation problem, bigfloat number truncation

* It computes $f : \mathcal{G}_\Delta \longrightarrow \mathcal{G}_\Delta$ (partial)

- Discussion: pointer machines are more robust than Turing machines
  * Cf: evaluation problem, bigfloat number truncation

Algebraic Pointer Machine

- Let Ω be a set of real operators
- Let a real Δ-graph be a Δ-graph where each node u stores a real number $Val(u)$
- Algebraic assignment instruction:
  - $w := \omega(w_1, \ldots, w_n)$ where $\omega \in \Omega$ is an n-ary operator
- Numerical comparison instruction:
  - IF $(w = w')$ GOTO(L) where L is a label
- Let $\mathcal{G}_\Delta(\mathbb{R})$ be the set of real Δ graphs
  - Then an Ω-pointer machine computes a function f :

Algebraic Pointer Machine

- Let Ω be a set of real operators
- Let a real Δ-graph be a Δ-graph where each node u stores a real number $Val(u)$
- Algebraic assignment instruction:
  - $w := \omega(w_1, \ldots, w_n)$ where $\omega \in \Omega$ is an n-ary operator
- Numerical comparison instruction:
  - IF $(w = w')$ GOTO(L) where L is a label
- Let $\mathcal{G}_\Delta(\mathbb{R})$ be the set of real Δ graphs
  - Then an Ω-pointer machine computes a function f :

Algebraic Pointer Machine

- Let Ω be a set of real operators
- Let a real Δ-graph be a Δ-graph where each node u stores a real number $Val(u)$
- Algebraic assignment instruction:
  - $w := \omega(w_1, \ldots, w_n)$ where $\omega \in \Omega$ is an n-ary operator
- Numerical comparison instruction:
  - IF $(w = w')$ GOTO(L) where L is a label
- Let $\mathcal{G}_\Delta(\mathbb{R})$ be the set of real Δ graphs
  - Then an Ω-pointer machine computes a function f :

Algebraic Pointer Machine

- Let Ω be a set of real operators

- Let a real Δ-graph be a Δ-graph where each node u stores a real number $Val(u)$

- Algebraic assignment instruction:
  - $w := \omega(w_1, \ldots, w_n)$ where $\omega \in \Omega$ is an n-ary operator

- Numerical comparison instruction:
  - IF $(w = w')$ GOTO(L) where L is a label

- Let $\mathcal{G}_\Delta(\mathbb{R})$ be the set of real Δ graphs
  - Then an Ω-pointer machine computes a function f :

Algebraic Pointer Machine

- Let Ω be a set of real operators
- Let a real Δ-graph be a Δ-graph where each node u stores a real number $Val(u)$
- Algebraic assignment instruction:
  * $w := \omega(w_1, \ldots, w_n)$ where $\omega \in \Omega$ is an n-ary operator
- Numerical comparison instruction:
  * IF $(w = w')$ GOTO(L) where L is a label
- Let $\mathcal{G}_\Delta(\mathbb{R})$ be the set of real Δ graphs
  * Then an Ω-pointer machine computes a function f :

$\mathcal{G}_\Delta(\mathbb{R}) \longrightarrow \mathcal{G}_\Delta(\mathbb{R})$

$*$ DEFINITION: we say f is Ω-computable if there is an Ω-pointer machine that computes it.

- These are what Knuth calls “semi-numerical problems” Why a numeric model of computation? Turing machines are twoo unstructured

$\mathcal{G}_\Delta(\mathbb{R}) \longrightarrow \mathcal{G}_\Delta(\mathbb{R})$

$*$ DEFINITION: we say f is Ω-computable if there is an Ω-pointer machine that computes it.

- These are what Knuth calls “semi-numerical problems” Why a numeric model of computation? Turing machines are twoo unstructured

$\mathcal{G}_\Delta(\mathbb{R}) \to \mathcal{G}_\Delta(\mathbb{R})$

$*$ DEFINITION: we say f is Ω-computable if there is an Ω-pointer machine that computes it.

- These are what Knuth calls "semi-numerical problems" Why a numeric model of computation? Turing machines are twoo unstructured

Numerical Pointer Machine

- Let a numeric Δ-graph be a Δ-graph where each node u stores a $Val(u) \in \mathbb{F}$

- Replace each $\omega \in \Omega$ be a relative approximation $\widetilde{\omega}$ taking an extra precision parameter

- Numeric assignment instruction:
  - $w := \widetilde{\omega}(w_1, \ldots, w_n, p)$ where $\widetilde{\omega}$ is an relative approximation of ω

- Let $\mathcal{G}_\Delta(\mathbb{F})$ be the set of numeric Δ graphs
  - Then an Ω-pointer machine computes a function $\widetilde{f}$:

Numerical Pointer Machine

- Let a numeric Δ-graph be a Δ-graph where each node u stores a $Val(u) \in \mathbb{F}$

- Replace each $\omega \in \Omega$ be a relative approximation $\widetilde{\omega}$ taking an extra precision parameter

- Numeric assignment instruction:
  - $w := \widetilde{\omega}(w_1, \ldots, w_n, p)$ where $\widetilde{\omega}$ is an relative approximation of ω

- Let $\mathcal{G}_\Delta(\mathbb{F})$ be the set of numeric Δ graphs
  - Then an Ω-pointer machine computes a function $\widetilde{f}$:

Numerical Pointer Machine

- Let a numeric Δ-graph be a Δ-graph where each node u stores a $Val(u) \in \mathbb{F}$

- Replace each $\omega \in \Omega$ be a relative approximation $\widetilde{\omega}$ taking an extra precision parameter

- Numeric assignment instruction:
  - $w := \widetilde{\omega}(w_1, \ldots, w_n, p)$ where $\widetilde{\omega}$ is an relative approximation of ω

- Let $\mathcal{G}_\Delta(\mathbb{F})$ be the set of numeric Δ graphs
  - Then an Ω-pointer machine computes a function $\widetilde{f}$:

Numerical Pointer Machine

- Let a numeric Δ-graph be a Δ-graph where each node u stores a $Val(u) \in \mathbb{F}$

- Replace each $\omega \in \Omega$ be a relative approximation $\widetilde{\omega}$ taking an extra precision parameter

- Numeric assignment instruction:
  - $w := \widetilde{\omega}(w_1, \ldots, w_n, p)$ where $\widetilde{\omega}$ is an relative approximation of ω

- Let $\mathcal{G}_\Delta(\mathbb{F})$ be the set of numeric Δ graphs
  - Then an Ω-pointer machine computes a function $\widetilde{f}$:

$\mathcal{G}_\Delta(\mathbb{F}) \times \mathbb{F} \to \mathcal{G}_\Delta(\mathbb{F})$

* We say $\widetilde{f}$ is numeric Ω-computable

- We say $\widetilde{f}$ is an absolute/relative approximation of $f : \mathcal{G}_\Delta(\mathbb{R}) \to \mathcal{G}_\Delta(\mathbb{R})$

  * if the value at each node of $\widetilde{f}(G, p)$ are p-bit absolute/relative approximations of the corresponding values of $f(G)$

  * DEFINITION: we say f is Ω-approximable if If $\widetilde{f}$ is numeric Ω-computable NOTE: This corresponds to EGC

$\mathcal{G}_\Delta(\mathbb{F}) \times \mathbb{F} \longrightarrow \mathcal{G}_\Delta(\mathbb{F})$

* We say $\widetilde{f}$ is numeric Ω-computable

- We say $\widetilde{f}$ is an absolute/relative approximation of $f : \mathcal{G}_\Delta(\mathbb{R}) \longrightarrow \mathcal{G}_\Delta(\mathbb{R})$

  * if the value at each node of $\widetilde{f}(G, p)$ are p-bit absolute/relative approximations of the corresponding values of $f(G)$

  * DEFINITION: we say f is Ω-approximable if If $\widetilde{f}$ is numeric Ω-computable NOTE: This corresponds to EGC

$\mathcal{G}_\Delta(\mathbb{F}) \times \mathbb{F} \longrightarrow \mathcal{G}_\Delta(\mathbb{F})$

* We say $\widetilde{f}$ is numeric Ω-computable

- We say $\widetilde{f}$ is an absolute/relative approximation of $f : \mathcal{G}_\Delta(\mathbb{R}) \longrightarrow \mathcal{G}_\Delta(\mathbb{R})$
  * if the value at each node of $\widetilde{f}(G, p)$ are p-bit absolute/relative approximations of the corresponding values of $f(G)$
  * DEFINITION: we say f is Ω-approximable if If $\widetilde{f}$ is numeric Ω-computable NOTE: This corresponds to EGC

Proof of Transfer Theorem

- One direction is easy: suppose Val_Ω is not relatively Ω-approximable

  * Then not every Ω-computable functions are relatively Ω-approximable. This is because Val_Ω is Ω-computable.

- Conversely, suppose Val_Ω is relatively Ω-approximable

  * Suppose f is a Ω-computable by some Ω-machine M. We just simulate M by a numeric Ω-machine in a step by step fashion. Whenever a branch step is taken, we call the relative approximation function for Val_Ω

Proof of Transfer Theorem

- One direction is easy: suppose Val_Ω is not relatively Ω-approximable

    $*$ Then not every Ω-computable functions are relatively Ω-approximable. This is because Val_Ω is Ω-computable.

- Conversely, suppose Val_Ω is relatively Ω-approximable

    $*$ Suppose f is a Ω-computable by some Ω-machine M. We just simulate M by a numeric Ω-machine in a step by step fashion. Whenever a branch step is taken, we call the relative approximation function for Val_Ω

Proof of Transfer Theorem

- One direction is easy: suppose Val_Ω is not relatively Ω-approximable

  $*$ Then not every Ω-computable functions are relatively Ω-approximable. This is because Val_Ω is Ω-computable.

- Conversely, suppose Val_Ω is relatively Ω-approximable

  $*$ Suppose f is a Ω-computable by some Ω-machine M. We just simulate M by a numeric Ω-machine in a step by step fashion. Whenever a branch step is taken, we call the relative approximation function for Val_Ω

Conclusions

- Our theory of real approximation
  - ∗ Conforms to practice, and to the usual assumptions of theoretical algorithms
- Complexity theory of real approximation
  - ∗ Let PF be the class PF of polynomial-time approximable functions
  - ∗ It is not closed under composition!
  - ∗ Need continuity conditions (e.g., Lipschitz functions)

Conclusions

- Our theory of real approximation
  - ∗ Conforms to practice, and to the usual assumptions of theoretical algorithms

- Complexity theory of real approximation
  - ∗ Let PF be the class PF of polynomial-time approximable functions
  - ∗ It is not closed under composition!
  - ∗ Need continuity conditions (e.g., Lipschitz functions)

Conclusions

- Our theory of real approximation

  * Conforms to practice, and to the usual assumptions of theoretical algorithms

- Complexity theory of real approximation

  * Let PF be the class PF of polynomial-time approximable functions

  * It is not closed under composition!

  * Need continuity conditions (e.g., Lipschitz functions)

REFERENCE

- "On Guaranteed Accuracy Computation",
  - C. Yap, in Geometric Computation, (eds. F. Chen & D. Wang), World Scientific Pub. Co. (2004)

"A rapacious monster lurks within every computer, and it dines exclusively on accurate digits."
– B.D. McCullough (2000)

THE END