A Lambda Calculus on Real Numbers

Yang Yue

Department of Mathematics
National University of Singapore

Sept 21, 2016

Acknowledgement

This talk is based on joint results with

- Keng Meng Ng (Nanyang Technological University, Singapore)
- Nazanin Tavana (Amirkabir University of Technology, Iran)
- Duccio Pianigiani and Andrea Sorbi (University of Siena, Italy)
- Jiangjie Qiu (Renmin University, China).

Outline

Main Motivation

- Q: What is an algorithm?
- A: This was answered by Gödel, Turing, Church and others in 1930s.
- Q: What is an algorithm on real numbers? Or on domains other than ω?
- A: This was answered by many people, for example, TTE model (originated from Turing), Blum-Shub-Smale (BSS) on real numbers; Kleene and others on higher types; infinite Turing machines (Hampkins and others).

Main Motivation

- Q: What is an algorithm?
- A: This was answered by Gödel, Turing, Church and others in 1930s.
- Q: What is an algorithm on real numbers? Or on domains other than ω?
- A: This was answered by many people, for example, TTE model (originated from Turing), Blum-Shub-Smale (BSS) on real numbers; Kleene and others on higher types; infinite Turing machines (Hampkins and others).

Main Motivation

- Q: What is an algorithm?
- A: This was answered by Gödel, Turing, Church and others in 1930s.
- Q: What is an algorithm on real numbers? Or on domains other than ω?
- A: This was answered by many people, for example, TTE model (originated from Turing), Blum-Shub-Smale (BSS) on real numbers; Kleene and others on higher types; infinite Turing machines (Hampkins and others).

Main Motivation

- Q: What is an algorithm?
- A: This was answered by Gödel, Turing, Church and others in 1930s.
- Q: What is an algorithm on real numbers? Or on domains other than ω?
- A: This was answered by many people, for example, TTE model (originated from Turing), Blum-Shub-Smale (BSS) on real numbers; Kleene and others on higher types; infinite Turing machines (Hampkins and others).

Differences

- On ω, different formulations give rise to the same notion of computability; furthermore, it fits the intuition of working mathematicians.
- On other domains, there are competing notions of computability, based on different intuitions. For example, TTE and BSS.
- The main difficulty: Objects are actual infinite, but algorithms must be "finitary". The key is how to balance the two.

Differences

- On ω, different formulations give rise to the same notion of computability; furthermore, it fits the intuition of working mathematicians.
- On other domains, there are competing notions of computability, based on different intuitions. For example, TTE and BSS.
- The main difficulty: Objects are actual infinite, but algorithms must be "finitary". The key is how to balance the two.

Differences

- On ω, different formulations give rise to the same notion of computability; furthermore, it fits the intuition of working mathematicians.
- On other domains, there are competing notions of computability, based on different intuitions. For example, TTE and BSS.
- The main difficulty: Objects are actual infinite, but algorithms must be “finitary”. The key is how to balance the two.

In this Talk

- Review two earlier formalizations of computability on domains beyond natural numbers.
- Introduce the third formalization using λ-calculus.
- In this talk, we only look at Baire space $\mathcal{N} = \omega^\omega$. But it also works on $\mathbb{R}$ with some major effort.
- (We show that these formalizations are equivalent.)

In this Talk

- Review two earlier formalizations of computability on domains beyond natural numbers.
- Introduce the third formalization using λ-calculus.
- In this talk, we only look at Baire space $\mathcal{N} = \omega^\omega$. But it also works on $\mathbb{R}$ with some major effort.
- (We show that these formalizations are equivalent.)

In this Talk

- Review two earlier formalizations of computability on domains beyond natural numbers.
- Introduce the third formalization using λ-calculus.
- In this talk, we only look at Baire space $\mathcal{N} = \omega^\omega$. But it also works on $\mathbb{R}$ with some major effort.
- (We show that these formalizations are equivalent.)

In this Talk

- Review two earlier formalizations of computability on domains beyond natural numbers.
- Introduce the third formalization using λ-calculus.
- In this talk, we only look at Baire space $\mathcal{N} = \omega^\omega$. But it also works on $\mathbb{R}$ with some major effort.
- (We show that these formalizations are equivalent.)

Computability on Baire Space

- Baire space $\mathcal{N} = \omega^\omega$, whose elements are referred as *type-one* objects.
- We also need natural numbers (*type-zero* objects) for our organization.
- We consider functions from $\mathbb{N}^m \times \mathcal{N}^n \to \mathbb{N}$ and $\mathbb{N}^m \times \mathcal{N}^n \to \mathcal{N}$, from mixed types to mixed types.
- To make explanation easier, we refer to $\mathbb{N}$ (the type-zero objects) "blue" and $\mathcal{N}$ (the type-one objects) "red".

Computability on Baire Space

- Baire space $\mathcal{N} = \omega^\omega$, whose elements are referred as *type-one* objects.
- We also need natural numbers (*type-zero* objects) for our organization.
- We consider functions from $\mathbb{N}^m \times \mathcal{N}^n \to \mathbb{N}$ and $\mathbb{N}^m \times \mathcal{N}^n \to \mathcal{N}$, from mixed types to mixed types.
- To make explanation easier, we refer to $\mathbb{N}$ (the type-zero objects) "blue" and $\mathcal{N}$ (the type-one objects) "red".

Computability on Baire Space

- Baire space $\mathcal{N} = \omega^\omega$, whose elements are referred as *type-one* objects.
- We also need natural numbers (*type-zero* objects) for our organization.
- We consider functions from $\mathbb{N}^m \times \mathcal{N}^n \to \mathbb{N}$ and $\mathbb{N}^m \times \mathcal{N}^n \to \mathcal{N}$, from mixed types to mixed types.
- To make explanation easier, we refer to $\mathbb{N}$ (the type-zero objects) "blue" and $\mathcal{N}$ (the type-one objects) "red".

Computability on Baire Space

- Baire space $\mathcal{N} = \omega^\omega$, whose elements are referred as *type-one* objects.
- We also need natural numbers (*type-zero* objects) for our organization.
- We consider functions from $\mathbb{N}^m \times \mathcal{N}^n \to \mathbb{N}$ and $\mathbb{N}^m \times \mathcal{N}^n \to \mathcal{N}$, from mixed types to mixed types.
- To make explanation easier, we refer to $\mathbb{N}$ (the type-zero objects) "blue" and $\mathcal{N}$ (the type-one objects) "red".

First Formalization: Using Function Schemes

Definition

The class of *partial recursive functions over* $\mathcal{N}$ is the smallest class $\mathcal{C}$ s.t.

(1) $\mathcal{C}$ contains the following basic functions:
   - (a) Zero function $Z : \mathbb{N} \to \mathbb{N}$;
   - (b) successor function $S : \mathbb{N} \to \mathbb{N}$; and
   - (c) the projection functions;
   - (d) all TTE computable functions (later); and
   - (e) the characteristic function χ of $\{0_{\mathcal{N}}\}$ from $\mathcal{N}$ to $\mathbb{N}$.

(2) $\mathcal{C}$ is closed under
   - (a) composition (provided the types match);
   - (b) primitive recursion (w.r.t. natural number variable); and
   - (c) μ-operator (w.r.t. natural number variable).

First Formalization: Using Function Schemes

Definition

The class of *partial recursive functions over* $\mathcal{N}$ is the smallest class $\mathcal{C}$ s.t.

(1) $\mathcal{C}$ contains the following basic functions:

- (a) Zero function $Z : \mathbb{N} \to \mathbb{N}$;
- (b) successor function $S : \mathbb{N} \to \mathbb{N}$; and
- (c) the projection functions;
- (d) all TTE computable functions (later); and
- (e) the characteristic function χ of $\{0_{\mathcal{N}}\}$ from $\mathcal{N}$ to $\mathbb{N}$.

(2) $\mathcal{C}$ is closed under

- (a) composition (provided the types match);
- (b) primitive recursion (w.r.t. natural number variable); and
- (c) μ-operator (w.r.t. natural number variable).

TTE-computable functions

Given $f : \omega^{<\omega} \to \omega^{<\omega}$ and $x \in \mathcal{N}$, we say that f is *monotone along x*, if

- $f(x \upharpoonright n)\downarrow$ for infinitely many n,
- for every $n < m$, if $f(x \upharpoonright n)\downarrow$ and $f(x \upharpoonright m)\downarrow$ then $f(x \upharpoonright n) \subseteq f(x \upharpoonright m)$, and
- $\lim_{n\to\infty} |f(x \upharpoonright n)| = \infty$.

The function $F : \mathcal{N} \to \mathcal{N}$ *induced by f* is defined to be

$$F(x) = \begin{cases} \sup\{f(\sigma) : \sigma \subset x\}, & \text{if } f \text{ is monotone along } x, \\ \uparrow, & \text{otherwise.} \end{cases}$$

Definition

We say that $F : \mathcal{N} \to \mathcal{N}$ is TTE-computable if it is induced by a partial recursive function $f : \omega^{<\omega} \to \omega^{<\omega}$.

TTE-computable functions

Given $f : \omega^{<\omega} \to \omega^{<\omega}$ and $x \in \mathcal{N}$, we say that f is *monotone along* x, if

- $f(x \upharpoonright n)\downarrow$ for infinitely many n,
- for every $n < m$, if $f(x \upharpoonright n)\downarrow$ and $f(x \upharpoonright m)\downarrow$ then $f(x \upharpoonright n) \subseteq f(x \upharpoonright m)$, and
- $\lim_{n\to\infty} |f(x \upharpoonright n)| = \infty$.

The function $F : \mathcal{N} \to \mathcal{N}$ *induced by* f is defined to be

$$F(x) = \begin{cases} \sup\{f(\sigma) : \sigma \subset x\}, & \text{if } f \text{ is monotone along } x, \\ \uparrow, & \text{otherwise.} \end{cases}$$

Definition

We say that $F : \mathcal{N} \to \mathcal{N}$ is TTE-computable if it is induced by a partial recursive function $f : \omega^{<\omega} \to \omega^{<\omega}$.

TTE-computable functions

Given $f : \omega^{<\omega} \to \omega^{<\omega}$ and $x \in \mathcal{N}$, we say that f is *monotone along* x, if

- $f(x \upharpoonright n)\downarrow$ for infinitely many n,
- for every $n < m$, if $f(x \upharpoonright n)\downarrow$ and $f(x \upharpoonright m)\downarrow$ then $f(x \upharpoonright n) \subseteq f(x \upharpoonright m)$, and
- $\lim_{n\to\infty} |f(x \upharpoonright n)| = \infty$.

The function $F : \mathcal{N} \to \mathcal{N}$ *induced by* f is defined to be

$$F(x) = \begin{cases} \sup\{f(\sigma) : \sigma \subset x\}, & \text{if } f \text{ is monotone along } x, \\ \uparrow, & \text{otherwise.} \end{cases}$$

Definition

We say that $F : \mathcal{N} \to \mathcal{N}$ is TTE-computable if it is induced by a partial recursive function $f : \omega^{<\omega} \to \omega^{<\omega}$.

Remarks

- Q: Why TTE-computable functions are “effective”?
- A: Because we have an effective procedure f which uniformly compute $F(x)$ up to any given precision; anything “shorter than” x will be computable in the standard sense.
- We just take the natural step to pass the closure point (using continuity).
- Justification for χ: “if... then... else" is essential to any algorithm, thus we must know some atomic properties of the objects.
- We have another justification in terms of the machines.

Remarks

- Q: Why TTE-computable functions are "effective"?
- A: Because we have an effective procedure f which uniformly compute $F(x)$ up to any given precision; anything "shorter than" x will be computable in the standard sense.
- We just take the natural step to pass the closure point (using continuity).
- Justification for χ: "if... then... else" is essential to any algorithm, thus we must know some atomic properties of the objects.
- We have another justification in terms of the machines.

Remarks

- Q: Why TTE-computable functions are "effective"?
- A: Because we have an effective procedure f which uniformly compute $F(x)$ up to any given precision; anything "shorter than" x will be computable in the standard sense.
- We just take the natural step to pass the closure point (using continuity).
- Justification for χ: "if... then... else" is essential to any algorithm, thus we must know some atomic properties of the objects.
- We have another justification in terms of the machines.

Remarks

- Q: Why TTE-computable functions are "effective"?
- A: Because we have an effective procedure f which uniformly compute $F(x)$ up to any given precision; anything "shorter than" x will be computable in the standard sense.
- We just take the natural step to pass the closure point (using continuity).
- Justification for χ: "if... then... else" is essential to any algorithm, thus we must know some atomic properties of the objects.
- We have another justification in terms of the machines.

Remarks

- Q: Why TTE-computable functions are “effective”?
- A: Because we have an effective procedure f which uniformly compute $F(x)$ up to any given precision; anything “shorter than” x will be computable in the standard sense.
- We just take the natural step to pass the closure point (using continuity).
- Justification for χ: “if... then... else" is essential to any algorithm, thus we must know some atomic properties of the objects.
- We have another justification in terms of the machines.

Second Formalization: Using Machines

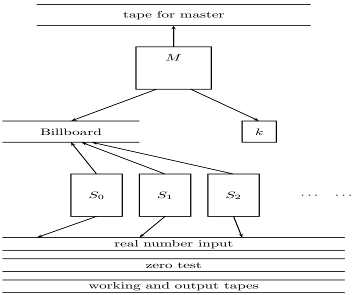

FIGURE 1. A Master-Slave machine

Transitions of Master-Slave Machines

- M has a finite set Q for its states. (We ignore the slaves, as they are the same universal TM.)
- Its program is a finite set of quadruples. (We assume single tape for convenience.)
- The quadruples are of the following three types:
  (1) Standard ones $qaa'q'$ or $qaDq'$ where $D \in \{L, R\}$.
  (2) Slave action command $qaSq'$: Slaves execute the instructions on the billboard; when every slave machine halts, the state of the master becomes q'.
  (3) Zero-test command $E01q$: To detect if a sequence is zero sequence, change the boolean bit accordingly, and change the master state from E to q. (Note that symbols 0 and 1 are unimportant.)

Transitions of Master-Slave Machines

- M has a finite set Q for its states. (We ignore the slaves, as they are the same universal TM.)
- Its program is a finite set of quadruples. (We assume single tape for convenience.)
- The quadruples are of the following three types:
  (1) Standard ones $qaa'q'$ or $qaDq'$ where $D \in \{L, R\}$.
  (2) Slave action command $qaSq'$: Slaves execute the instructions on the billboard; when every slave machine halts, the state of the master becomes q'.
  (3) Zero-test command $E01q$: To detect if a sequence is zero sequence, change the boolean bit accordingly, and change the master state from E to q. (Note that symbols 0 and 1 are unimportant.)

Transitions of Master-Slave Machines

- M has a finite set Q for its states. (We ignore the slaves, as they are the same universal TM.)
- Its program is a finite set of quadruples. (We assume single tape for convenience.)
- The quadruples are of the following three types:
  (1) Standard ones $qaa'q'$ or $qaDq'$ where $D \in \{L, R\}$.
  (2) Slave action command $qaSq'$: Slaves execute the instructions on the billboard; when every slave machine halts, the state of the master becomes q'.
  (3) Zero-test command $E01q$: To detect if a sequence is zero sequence, change the boolean bit accordingly, and change the master state from E to q. (Note that symbols 0 and 1 are unimportant.)

Transitions of Master-Slave Machines

- M has a finite set Q for its states. (We ignore the slaves, as they are the same universal TM.)
- Its program is a finite set of quadruples. (We assume single tape for convenience.)
- The quadruples are of the following three types:
  (1) Standard ones $qaa'q'$ or $qaDq'$ where $D \in \{L, R\}$.
  (2) Slave action command $qaSq'$: Slaves execute the instructions on the billboard; when every slave machine halts, the state of the master becomes q'.
  (3) Zero-test command $E01q$: To detect if a sequence is zero sequence, change the boolean bit accordingly, and change the master state from E to q. (Note that symbols 0 and 1 are unimportant.)

Transitions of Master-Slave Machines

- M has a finite set Q for its states. (We ignore the slaves, as they are the same universal TM.)
- Its program is a finite set of quadruples. (We assume single tape for convenience.)
- The quadruples are of the following three types:
  (1) Standard ones $qaa'q'$ or $qaDq'$ where $D \in \{L, R\}$.
  (2) Slave action command $qaSq'$: Slaves execute the instructions on the billboard; when every slave machine halts, the state of the master becomes q'.
  (3) Zero-test command $E01q$: To detect if a sequence is zero sequence, change the boolean bit accordingly, and change the master state from E to q. (Note that symbols 0 and 1 are unimportant.)

Fine Tuning

Lemma

A TTE-computable function $F : \mathcal{N} \to \mathcal{N}$ (say induced by f) is induced by some partial recursive function g which is non-decreasing and whose domain is downward closed. Furthermore, there is a total recursive function $k : \mathbb{N} \to \mathbb{N}$ which transfer an index e of f to an index of g.

Definition

We call a master-slave machine M *fine tuned* (for $\mathcal{N}$) if the following convention is added: When the master writes the code e on the billboard, the i-th slave just computes $\varphi_{k(e)}(x \restriction i)$.

Fine Tuning

Lemma

A TTE-computable function $F : \mathcal{N} \to \mathcal{N}$ (say induced by f) is induced by some partial recursive function g which is non-decreasing and whose domain is downward closed. Furthermore, there is a total recursive function $k : \mathbb{N} \to \mathbb{N}$ which transfer an index e of f to an index of g.

Definition

We call a master-slave machine M *fine tuned* (for $\mathcal{N}$) if the following convention is added: When the master writes the code e on the billboard, the i-th slave just computes $\varphi_{k(e)}(x \upharpoonright i)$.

The Definition

Definition

We say that a partial function f is MS-computable if there is a (fine-tuned) master-slave machine M such that

$$f(n; x) = \begin{cases} y, & \text{if } M \text{ on input } (n; x) \text{ halts} \\ & \text{and the output is } y; \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

The Third Formulation: Applied λ-Calculus

Definition

(T1) (a) Variables $x_i : i \in \omega$ are λ-terms.

(b) Constants **a** for $a \in \mathcal{N}$ are λ-terms. We also have $\perp$ for the divergency.

(c) We have a special constant ϕ which is a λ-term.

(T2) (application) If M and N are λ-terms, then MN is a λ-term.

(T3) (abstraction) If M is a λ-term and x is a variable, then $\lambda x.M$ is a λ-term.

We say that a λ-term is *pure* if it does not contain the constant symbols **a**.

The Third Formulation: Applied λ-Calculus

Definition

(T1) (a) Variables $x_i : i \in \omega$ are λ-terms.
(b) Constants $\mathbf{a}$ for $a \in \mathcal{N}$ are λ-terms. We also have $\perp$ for the divergency.
(c) We have a special constant ϕ which is a λ-term.

(T2) (application) If M and N are λ-terms, then MN is a λ-term.

(T3) (abstraction) If M is a λ-term and x is a variable, then $\lambda x.M$ is a λ-term.

We say that a λ-term is *pure* if it does not contain the constant symbols $\mathbf{a}$.

The Third Formulation: Applied λ-Calculus

Definition

(T1) (a) Variables $x_i : i \in \omega$ are λ-terms.
(b) Constants **a** for $a \in \mathcal{N}$ are λ-terms. We also have $\perp$ for the divergency.
(c) We have a special constant ϕ which is a λ-term.

(T2) (application) If M and N are λ-terms, then MN is a λ-term.

(T3) (abstraction) If M is a λ-term and x is a variable, then $\lambda x.M$ is a λ-term.

We say that a λ-term is *pure* if it does not contain the constant symbols **a**.

Reduction Rules

Recall: There are *numerals* $\ulcorner n \urcorner$ which are λ-terms to code the natural numbers n.

Definition

(A1) (β conversion) $(\lambda x.M)N \to M[x := N]$.

(A2) (δ-rules) For the sake of readability, we write ϕ as E (for the characteristic function of $\{0_{\mathcal{N}}\}$) and $\underbrace{\phi \ldots \phi}_{e+2}$ as Φ_e.

(1) $\Phi_e \mathbf{a} \to \mathbf{b}$, if the e-th TTE function on input a is defined and the output is b; otherwise $\Phi_e \mathbf{a} \to \bot$.

(2) $E\mathbf{0} \to \ulcorner 0 \urcorner$ and $E\mathbf{a} \to \ulcorner 1 \urcorner$ for $a \in \mathcal{N}$ and $a \neq 0_{\mathcal{N}}$.

Reduction Rules

Recall: There are *numerals* $\ulcorner n \urcorner$ which are λ-terms to code the natural numbers n.

Definition

(A1) (β conversion) $(\lambda x.M)N \to M[x := N]$.

(A2) (δ-rules) For the sake of readability, we write ϕ as E (for the characteristic function of $\{0_{\mathcal{N}}\}$) and $\underbrace{\phi \dots \phi}_{e+2}$ as Φ_e.

(1) $\Phi_e \mathbf{a} \to \mathbf{b}$, if the e-th TTE function on input a is defined and the output is b; otherwise $\Phi_e \mathbf{a} \to \bot$.

(2) $E\mathbf{0} \to \ulcorner 0 \urcorner$ and $E\mathbf{a} \to \ulcorner 1 \urcorner$ for $a \in \mathcal{N}$ and $a \neq 0_{\mathcal{N}}$.

Solvable and Unsolvable

Theorem (Church-Rosser)

If $M \to^* M_1$ and $M \to^* M_2$ then there is a λ-term N such that $M_1 \to^* N$ and $M_2 \to^* N$.

Thus if a λ-term reduces to a normal form it is unique.

Definition

We say that a λ-term M is *solvable* if there are λ-terms $N_1, \ldots, N_k$ such that either

$$MN_1 \ldots N_k \to^* \mathbf{I}$$

or

$$(E(MN_1 \ldots N_i))N_{i+1} \ldots N_k \to^* \mathbf{I},$$

where $\mathbf{I}$ is $\lambda x.x$. We say that M is *unsolvable* if M is not solvable.

Solvable and Unsolvable

Theorem (Church-Rosser)

If $M \to^* M_1$ and $M \to^* M_2$ then there is a λ-term N such that $M_1 \to^* N$ and $M_2 \to^* N$.

Thus if a λ-term reduces to a normal form it is unique.

Definition

We say that a λ-term M is *solvable* if there are λ-terms $N_1, \ldots, N_k$ such that either

$$MN_1 \ldots N_k \to^* \mathbf{I}$$

or

$$(E(MN_1 \ldots N_i))N_{i+1} \ldots N_k \to^* \mathbf{I},$$

where $\mathbf{I}$ is $\lambda x.x$. We say that M is *unsolvable* if M is not solvable.

The Third Formalization

Definition

We say that f is λ-definable if there exists a pure λ-term F such that for all $\vec{z} \in \mathbb{N}^m \times \mathcal{N}^n$,

$$\begin{array}{rcl} f(\vec{z}) = y & \text{implies} & F\ulcorner\vec{z}\urcorner \to^* \ulcorner y\urcorner \\ f(\vec{z})\uparrow & \text{implies} & F\ulcorner\vec{z}\urcorner \text{ is unsolable.} \end{array}$$

In this case, we say that f is λ-defined by F.

Equivalence Theorem for Computation over $\mathcal{N}$

Theorem

Over $\mathcal{N}$, f is partial recursive iff f is MS-computable iff f is λ-definable.

We show that $\{$ par rec$\} \subseteq \{\lambda$-def$\} \subseteq \{$ MS-comp$\} \subseteq \{$ par rec$\}$.

Equivalence Theorem for Computation over $\mathcal{N}$

Theorem

Over $\mathcal{N}$, f is partial recursive iff f is MS-computable iff f is λ-definable.

We show that $\{$ par rec$\} \subseteq \{\lambda$-def$\} \subseteq \{$ MS-comp$\} \subseteq \{$ par rec$\}$.

A Normal Form Theorem for Baire Space Computation

Theorem

There are primitive recursive over $\mathcal{N}$ predicate $T(e, x, z)$ and function $U(z; x)$ such that for all partial recursive function f over $\mathcal{N}$, there is an m, such that,

$$f(x) = U(\mu z T(m, x, z); x).$$

Remarks on relations with BSS and TTE

- Starting from BSS models (assuming no real parameters), add exponential functions as a primitive
- Adding all TTE-computable functions
- Furthermore, the TTE-computable functions must be coded uniformly internally (This requires natural numbers available).
- Then we have MS-computable functions, in this sense, it is the minimal model containing BSS and TTE.

Remarks on relations with BSS and TTE

- Starting from BSS models (assuming no real parameters), add exponential functions as a primitive
- Adding all TTE-computable functions
- Furthermore, the TTE-computable functions must be coded uniformly internally (This requires natural numbers available).
- Then we have MS-computable functions, in this sense, it is the minimal model containing BSS and TTE.

Remarks on relations with BSS and TTE

- Starting from BSS models (assuming no real parameters), add exponential functions as a primitive
- Adding all TTE-computable functions
- Furthermore, the TTE-computable functions must be coded uniformly internally (This requires natural numbers available).
- Then we have MS-computable functions, in this sense, it is the minimal model containing BSS and TTE.

Remarks on relations with BSS and TTE

- Starting from BSS models (assuming no real parameters), add exponential functions as a primitive
- Adding all TTE-computable functions
- Furthermore, the TTE-computable functions must be coded uniformly internally (This requires natural numbers available).
- Then we have MS-computable functions, in this sense, it is the minimal model containing BSS and TTE.

Final Remarks

- Classical Church Thesis (on $\mathbb{N}$): Intuitively computable is Turing computable.
- Q: Can we have something similar for Baire space?
- Can algorithms be just the blue part? To be more precise, fix an effectively indexed family of functions $\mathcal{F}$, define partial recursive functions with $\mathcal{F}$ as primitives; Master-slave machines with slaves computing $\mathcal{F}$; λ-calculus with external rules induced by $\mathcal{F}$. Can we always show they are equivalent?
- Can we lift them even further?

Final Remarks

- Classical Church Thesis (on $\mathbb{N}$): Intuitively computable is Turing computable.
- Q: Can we have something similar for Baire space?
- Can algorithms be just the blue part? To be more precise, fix an effectively indexed family of functions $\mathcal{F}$, define partial recursive functions with $\mathcal{F}$ as primitives; Master-slave machines with slaves computing $\mathcal{F}$; λ-calculus with external rules induced by $\mathcal{F}$. Can we always show they are equivalent?
- Can we lift them even further?

Final Remarks

- Classical Church Thesis (on $\mathbb{N}$): Intuitively computable is Turing computable.
- Q: Can we have something similar for Baire space?
- Can algorithms be just the blue part? To be more precise, fix an effectively indexed family of functions $\mathcal{F}$, define partial recursive functions with $\mathcal{F}$ as primitives; Master-slave machines with slaves computing $\mathcal{F}$; λ-calculus with external rules induced by $\mathcal{F}$. Can we always show they are equivalent?
- Can we lift them even further?

Final Remarks

- Classical Church Thesis (on $\mathbb{N}$): Intuitively computable is Turing computable.
- Q: Can we have something similar for Baire space?
- Can algorithms be just the blue part? To be more precise, fix an effectively indexed family of functions $\mathcal{F}$, define partial recursive functions with $\mathcal{F}$ as primitives; Master-slave machines with slaves computing $\mathcal{F}$; λ-calculus with external rules induced by $\mathcal{F}$. Can we always show they are equivalent?
- Can we lift them even further?