WhileCC-approximability and Acceptability of Elementary Functions

Fateme Ghasemi   Dr. Jeffery Zucker
ghases5@mcmaster.ca   zucker@mcmaster.ca

September 2025

McMaster University

Computability of Functions on $\mathbb{R}$

For total functions on $\mathbb{R}$, the following models of computation are equivalent for all functions that are effectively locally uniformly continuous [Tucker and Zucker, 2005]:

- GL-computability,testtesttesttesttest
- tracking computability,
- multipolynomial approximability, and
- **WhileCC**-approximability.

What about **partial** functions? $1/x$, $\sqrt[n]{x}$, ...

For partial functions on $\mathbb{R}$, Fu and Zucker [2014] generalize effectively locally uniform continuity to **acceptability** to get an equivalence.

Problem

How general is this class of acceptable functions?

Useful First Step Towards Solution

Show that the elementary functions satisfy the acceptability conditions.

Computability of Functions on $\mathbb{R}$

For total functions on $\mathbb{R}$, the following models of computation are equivalent for all functions that are effectively locally uniformly continuous [Tucker and Zucker, 2005]:

- GL-computability,testtesttesttesttest
- tracking computability,
- multipolynomial approximability, and
- **WhileCC**-approximability.

What about **partial** functions? $1/x$, $\sqrt[n]{x}$, ...

For partial functions on $\mathbb{R}$, Fu and Zucker [2014] generalize effectively locally uniform continuity to **acceptability** to get an equivalence.

Problem

How general is this class of acceptable functions?

Useful First Step Towards Solution

Show that the elementary functions satisfy the acceptability conditions.

Computability of Functions on $\mathbb{R}$

For total functions on $\mathbb{R}$, the following models of computation are equivalent for all functions that are effectively locally uniformly continuous [Tucker and Zucker, 2005]:

- GL-computability,testtesttesttest
- tracking computability,
- multipolynomial approximability, and
- **WhileCC**-approximability.

What about **partial** functions? $1/x$, $\sqrt[n]{x}$, ...

For partial functions on $\mathbb{R}$, Fu and Zucker [2014] generalize effectively locally uniform continuity to **acceptability** to get an equivalence.

Problem

How general is this class of acceptable functions?

Useful First Step Towards Solution

Show that the elementary functions satisfy the acceptability conditions.

Computability of Functions on $\mathbb{R}$

For total functions on $\mathbb{R}$, the following models of computation are equivalent for all functions that are effectively locally uniformly continuous [Tucker and Zucker, 2005]:

- GL-computability,testtesttesttesttest
- tracking computability,
- multipolynomial approximability, and
- **WhileCC**-approximability.

What about **partial** functions? $1/x$, $\sqrt[n]{x}$, ...

For partial functions on $\mathbb{R}$, Fu and Zucker [2014] generalize effectively locally uniform continuity to **acceptability** to get an equivalence.

Problem

How general is this class of acceptable functions?

Useful First Step Towards Solution

Show that the elementary functions satisfy the acceptability conditions.

Computability of Functions on $\mathbb{R}$

For total functions on $\mathbb{R}$, the following models of computation are equivalent for all functions that are effectively locally uniformly continuous [Tucker and Zucker, 2005]:

- GL-computability,testtesttesttesttest
- tracking computability,
- multipolynomial approximability, and
- **WhileCC**-approximability.

What about **partial** functions? $1/x$, $\sqrt[n]{x}$, ...

For partial functions on $\mathbb{R}$, Fu and Zucker [2014] generalize effectively locally uniform continuity to **acceptability** to get an equivalence.

Problem

How general is this class of acceptable functions?

Useful First Step Towards Solution

Show that the elementary functions satisfy the acceptability conditions.

Computability of Functions on $\mathbb{R}$

For total functions on $\mathbb{R}$, the following models of computation are equivalent for all functions that are effectively locally uniformly continuous [Tucker and Zucker, 2005]:

- GL-computability,testtesttesttesttest
- tracking computability,
- multipolynomial approximability, and
- **WhileCC**-approximability.

What about **partial** functions? $1/x$, $\sqrt[n]{x}$, ...

For partial functions on $\mathbb{R}$, Fu and Zucker [2014] generalize effectively locally uniform continuity to **acceptability** to get an equivalence.

Problem

How general is this class of acceptable functions?

Useful First Step Towards Solution

Show that the elementary functions satisfy the acceptability conditions.

Computability of Functions on $\mathbb{R}$

For total functions on $\mathbb{R}$, the following models of computation are equivalent for all functions that are effectively locally uniformly continuous [Tucker and Zucker, 2005]:

- GL-computability,testtesttesttesttest
- tracking computability,
- multipolynomial approximability, and
- **WhileCC**-approximability.

What about **partial** functions? $1/x$, $\sqrt[n]{x}$, ...

For partial functions on $\mathbb{R}$, Fu and Zucker [2014] generalize effectively locally uniform continuity to **acceptability** to get an equivalence.

Problem

How general is this class of acceptable functions?

Useful First Step Towards Solution

Show that the elementary functions satisfy the acceptability conditions.

Computability of Functions on $\mathbb{R}$

For total functions on $\mathbb{R}$, the following models of computation are equivalent for all functions that are effectively locally uniformly continuous [Tucker and Zucker, 2005]:

- GL-computability,testtesttesttesttest
- tracking computability,
- multipolynomial approximability, and
- **WhileCC**-approximability.

What about **partial** functions? $1/x$, $\sqrt[n]{x}$, ...

For partial functions on $\mathbb{R}$, Fu and Zucker [2014] generalize effectively locally uniform continuity to **acceptability** to get an equivalence.

Problem

How general is this class of acceptable functions?

Useful First Step Towards Solution

Show that the elementary functions satisfy the acceptability conditions.

Background – Acceptability

Definition (Acceptability)

A function $f : \mathbb{R} \to \mathbb{R}$ is **acceptable** if there exists a sequence X where:

1. X is an **effective open exhaustion** for $\mathbf{dom}(f)$, and
2. f is **effectively locally uniformly continuous w.r.t. X**.

Background – Acceptability

Definition (Acceptability)

A function $f : \mathbb{R} \to \mathbb{R}$ is **acceptable** if there exists a sequence X where:

1. X is an **effective open exhaustion** for $\mathbf{dom}(f)$, and
2. f is **effectively locally uniformly continuous w.r.t.** X.

Definition (Acceptability)

A function $f : \mathbb{R} \to \mathbb{R}$ is **acceptable** if there exists a sequence X where:

1. X is an **effective open exhaustion** for $\mathbf{dom}(f)$, and
2. f is **effectively locally uniformly continuous w.r.t.** X.

Background – Acceptability

Definition (Acceptability)

A function $f : \mathbb{R} \to \mathbb{R}$ is **acceptable** if there exists a sequence X where:

1. X is an **effective open exhaustion** for $\mathbf{dom}(f)$, and
2. f is **effectively locally uniformly continuous w.r.t. X**.

Background – Effective Open Exhaustions

Definition ([Fu and Zucker, 2014])

A sequence $(U_1, U_2, \ldots)$ of open sets is called an **effective open exhaustion** for an open $U \subseteq \mathbb{R}$ if

1. $U = \bigcup_{l=0}^{\infty} U_l$, and
2. for each $l \in \mathbb{N}$, U_l is a finite union of non-empty open finite intervals $I_1^l, I_2^l, \ldots, I_{k_l}^l$ whose closures are pairwise disjoint, and
3. for each $l \in \mathbb{N}$, $\overline{U_l} = \bigcup_{i=1}^{k_l} \overline{I_i^l} \subseteq U_{l+1}$.
4. for all l, the components I_i^l that are intervals building up the stage U_l, are *rational* and *ordered* i.e., $I_i^l = (a_i^l, b_i^l)$ for some $a_i^l, b_i^l \in \mathbb{Q}$ where $b_i^l < a_{i+1}^l$ for $i = 1, \ldots, k_l - 1$, and
5. the map $l \mapsto (a_1^l, b_1^l, \ldots, a_{k_l}^l, b_{k_l}^l)$ which delivers the sequence of stages $U_l = I_1^l \cup \ldots \cup I_{k_l}^l$ is recursive.

Example

The sequence of open sets $(-1,1), (-2,2), \ldots, (-k,k), \ldots$ is the standard effective open exhaustion for $\mathbb{R}$.

Background – Effective Open Exhaustions

Definition ([Fu and Zucker, 2014])

A sequence $(U_1, U_2, \ldots)$ of open sets is called an **effective open exhaustion** for an open $U \subseteq \mathbb{R}$ if

1. $U = \bigcup_{l=0}^{\infty} U_l$, and
2. for each $l \in \mathbb{N}$, U_l is a finite union of non-empty open finite intervals $I_1^l, I_2^l, ..., I_{k_l}^l$ whose closures are pairwise disjoint, and
3. for each $l \in \mathbb{N}$, $\overline{U_l} = \bigcup_{i=1}^{k_l} \overline{I_i^l} \subseteq U_{l+1}$.
4. for all l, the components I_i^l that are intervals building up the stage U_l, are *rational* and *ordered* i.e., $I_i^l = (a_i^l, b_i^l)$ for some $a_i^l, b_i^l \in \mathbb{Q}$ where $b_i^l < a_{i+1}^l$ for $i = 1, ..., k_l - 1$, and
5. the map $l \mapsto (a_1^l, b_1^l, ..., a_{k_l}^l, b_{k_l}^l)$ which delivers the sequence of stages $U_l = I_1^l \cup ... \cup I_{k_l}^l$ is recursive.

Example

The sequence of open sets $(-1, 1), (-2, 2), \ldots, (-k, k), \ldots$ is the standard effective open exhaustion for $\mathbb{R}$.

Background – Effective Open Exhaustions

Definition ([Fu and Zucker, 2014])

A sequence $(U_1, U_2, \ldots)$ of open sets is called an **effective open exhaustion** for an open $U \subseteq \mathbb{R}$ if

1. $U = \bigcup_{l=0}^{\infty} U_l$, and
2. for each $l \in \mathbb{N}$, U_l is a finite union of non-empty open finite intervals $I_1^l, I_2^l, ..., I_{k_l}^l$ whose closures are pairwise disjoint, and
3. for each $l \in \mathbb{N}$, $\overline{U_l} = \bigcup_{i=1}^{k_l} \overline{I_i^l} \subseteq U_{l+1}$.
4. for all l, the components I_i^l that are intervals building up the stage U_l, are *rational* and *ordered* i.e., $I_i^l = (a_i^l, b_i^l)$ for some $a_i^l, b_i^l \in \mathbb{Q}$ where $b_i^l < a_{i+1}^l$ for $i = 1, ..., k_l - 1$, and
5. the map $l \mapsto (a_1^l, b_1^l, ..., a_{k_l}^l, b_{k_l}^l)$ which delivers the sequence of stages $U_l = I_1^l \cup ... \cup I_{k_l}^l$ is recursive.

Example

The sequence of open sets $(-1,1), (-2,2), \ldots, (-k,k), \ldots$ is the standard effective open exhaustion for $\mathbb{R}$.

Background – Effective Open Exhaustions

Definition ([Fu and Zucker, 2014])

A sequence $(U_1, U_2, \ldots)$ of open sets is called an **effective open exhaustion** for an open $U \subseteq \mathbb{R}$ if

1. $U = \bigcup_{l=0}^{\infty} U_l$, and
2. for each $l \in \mathbb{N}$, U_l is a finite union of non-empty open finite intervals $I_1^l, I_2^l, \ldots, I_{k_l}^l$ whose closures are pairwise disjoint, and
3. for each $l \in \mathbb{N}$, $\overline{U_l} = \bigcup_{i=1}^{k_l} \overline{I_i^l} \subseteq U_{l+1}$.
4. for all l, the components I_i^l that are intervals building up the stage U_l, are *rational* and *ordered* i.e., $I_i^l = (a_i^l, b_i^l)$ for some $a_i^l, b_i^l \in \mathbb{Q}$ where $b_i^l < a_{i+1}^l$ for $i = 1, \ldots, k_l - 1$, and
5. the map $l \mapsto (a_1^l, b_1^l, \ldots, a_{k_l}^l, b_{k_l}^l)$ which delivers the sequence of stages $U_l = I_1^l \cup \ldots \cup I_{k_l}^l$ is recursive.

Example

The sequence of open sets $(-1, 1), (-2, 2), \ldots, (-k, k), \ldots$ is the standard effective open exhaustion for $\mathbb{R}$.

Background – Effective Open Exhaustions

Definition ([Fu and Zucker, 2014])

A sequence $(U_1, U_2, \ldots)$ of open sets is called an **effective open exhaustion** for an open $U \subseteq \mathbb{R}$ if

1. $U = \bigcup_{l=0}^{\infty} U_l$, and
2. for each $l \in \mathbb{N}$, U_l is a finite union of non-empty open finite intervals $I_1^l, I_2^l, \ldots, I_{k_l}^l$ whose closures are pairwise disjoint, and
3. for each $l \in \mathbb{N}$, $\overline{U_l} = \bigcup_{i=1}^{k_l} \overline{I_i^l} \subseteq U_{l+1}$.
4. for all l, the components I_i^l that are intervals building up the stage U_l, are *rational* and *ordered* i.e., $I_i^l = (a_i^l, b_i^l)$ for some $a_i^l, b_i^l \in \mathbb{Q}$ where $b_i^l < a_{i+1}^l$ for $i = 1, \ldots, k_l - 1$, and
5. the map $l \mapsto (a_1^l, b_1^l, \ldots, a_{k_l}^l, b_{k_l}^l)$ which delivers the sequence of stages $U_l = I_1^l \cup \ldots \cup I_{k_l}^l$ is recursive.

Example

The sequence of open sets $(-1, 1), (-2, 2), \ldots, (-k, k), \ldots$ is the standard effective open exhaustion for $\mathbb{R}$.

Background – Effective Open Exhaustions

Definition ([Fu and Zucker, 2014])

A sequence $(U_1, U_2, \ldots)$ of open sets is called an **effective open exhaustion** for an open $U \subseteq \mathbb{R}$ if

1. $U = \bigcup_{l=0}^{\infty} U_l$, and
2. for each $l \in \mathbb{N}$, U_l is a finite union of non-empty open finite intervals $I_1^l, I_2^l, \ldots, I_{k_l}^l$ whose closures are pairwise disjoint, and
3. for each $l \in \mathbb{N}$, $\overline{U_l} = \bigcup_{i=1}^{k_l} \overline{I_i^l} \subseteq U_{l+1}$.
4. for all l, the components I_i^l that are intervals building up the stage U_l, are *rational* and *ordered* i.e., $I_i^l = (a_i^l, b_i^l)$ for some $a_i^l, b_i^l \in \mathbb{Q}$ where $b_i^l < a_{i+1}^l$ for $i = 1, \ldots, k_l - 1$, and
5. the map $l \mapsto (a_1^l, b_1^l, \ldots, a_{k_l}^l, b_{k_l}^l)$ which delivers the sequence of stages $U_l = I_1^l \cup \ldots \cup I_{k_l}^l$ is recursive.

Example

The sequence of open sets $(-1, 1), (-2, 2), \ldots, (-k, k), \ldots$ is the standard effective open exhaustion for $\mathbb{R}$.

Background – Effective Open Exhaustions

Definition ([Fu and Zucker, 2014])

A sequence $(U_1, U_2, \ldots)$ of open sets is called an **effective open exhaustion** for an open $U \subseteq \mathbb{R}$ if

1. $U = \bigcup_{l=0}^{\infty} U_l$, and
2. for each $l \in \mathbb{N}$, U_l is a finite union of non-empty open finite intervals $I_1^l, I_2^l, \ldots, I_{k_l}^l$ whose closures are pairwise disjoint, and
3. for each $l \in \mathbb{N}$, $\overline{U_l} = \bigcup_{i=1}^{k_l} \overline{I_i^l} \subseteq U_{l+1}$.
4. for all l, the components I_i^l that are intervals building up the stage U_l, are *rational* and *ordered* i.e., $I_i^l = (a_i^l, b_i^l)$ for some $a_i^l, b_i^l \in \mathbb{Q}$ where $b_i^l < a_{i+1}^l$ for $i = 1, \ldots, k_l - 1$, and
5. the map $l \mapsto (a_1^l, b_1^l, \ldots, a_{k_l}^l, b_{k_l}^l)$ which delivers the sequence of stages $U_l = I_1^l \cup \ldots \cup I_{k_l}^l$ is recursive.

Example

The sequence of open sets $(-1,1), (-2,2), \ldots, (-k,k), \ldots$ is the standard effective open exhaustion for $\mathbb{R}$.

Background – Effective Open Exhaustions

Definition ([Fu and Zucker, 2014])

A sequence $(U_1, U_2, \ldots)$ of open sets is called an **effective open exhaustion** for an open $U \subseteq \mathbb{R}$ if

1. $U = \bigcup_{l=0}^{\infty} U_l$, and
2. for each $l \in \mathbb{N}$, U_l is a finite union of non-empty open finite intervals $I_1^l, I_2^l, \ldots, I_{k_l}^l$ whose closures are pairwise disjoint, and
3. for each $l \in \mathbb{N}$, $\overline{U_l} = \bigcup_{i=1}^{k_l} \overline{I_i^l} \subseteq U_{l+1}$.
4. for all l, the components I_i^l that are intervals building up the stage U_l, are *rational* and *ordered* i.e., $I_i^l = (a_i^l, b_i^l)$ for some $a_i^l, b_i^l \in \mathbb{Q}$ where $b_i^l < a_{i+1}^l$ for $i = 1, \ldots, k_l - 1$, and
5. the map $l \mapsto (a_1^l, b_1^l, \ldots, a_{k_l}^l, b_{k_l}^l)$ which delivers the sequence of stages $U_l = I_1^l \cup \ldots \cup I_{k_l}^l$ is recursive.

Example

The sequence of open sets $(-1, 1), (-2, 2), \ldots, (-k, k), \ldots$ is the standard effective open exhaustion for $\mathbb{R}$.

Background – Effective Open Exhaustions

Definition ([Fu and Zucker, 2014])

A sequence $(U_1, U_2, \ldots)$ of open sets is called an **effective open exhaustion** for an open $U \subseteq \mathbb{R}$ if

1. $U = \bigcup_{l=0}^{\infty} U_l$, and
2. for each $l \in \mathbb{N}$, U_l is a finite union of non-empty open finite intervals $I_1^l, I_2^l, ..., I_{k_l}^l$ whose closures are pairwise disjoint, and
3. for each $l \in \mathbb{N}$, $\overline{U_l} = \bigcup_{i=1}^{k_l} \overline{I_i^l} \subseteq U_{l+1}$.
4. for all l, the components I_i^l that are intervals building up the stage U_l, are *rational* and *ordered* i.e., $I_i^l = (a_i^l, b_i^l)$ for some $a_i^l, b_i^l \in \mathbb{Q}$ where $b_i^l < a_{i+1}^l$ for $i = 1, ..., k_l - 1$, and
5. the map $l \mapsto (a_1^l, b_1^l, ..., a_{k_l}^l, b_{k_l}^l)$ which delivers the sequence of stages $U_l = I_1^l \cup ... \cup I_{k_l}^l$ is recursive.

Example

The sequence of open sets $(-1,1), (-2,2), \ldots, (-k,k), \ldots$ is the standard effective open exhaustion for $\mathbb{R}$.

Background – Effective Local Uniform Continuity

Definition ([Fu and Zucker, 2014])

A function f on U is **effectively locally uniformly continuous** **w.r.t. an effective open exhaustion** $(U_n)_{n\in\mathbb{N}}$ of U, if there is a recursive function $M : \mathbb{N}^2 \twoheadrightarrow \mathbb{N}$ such that for all $k, l \in \mathbb{N}$ and all $x, y \in U_l$:

$$|x - y| < 2^{-M(k,l)} \implies |f(x) - f(y)| < 2^{-k}$$

Background – Effective Local Uniform Continuity

Definition ([Fu and Zucker, 2014])

A function f on U is **effectively locally uniformly continuous** **w.r.t. an effective open exhaustion** $(U_n)_{n\in\mathbb{N}}$ of U, if there is a recursive function $M : \mathbb{N}^2 \twoheadrightarrow \mathbb{N}$ such that for all $k, l \in \mathbb{N}$ and all $x, y \in U_l$:

$$|x - y| < 2^{-M(k,l)} \implies |f(x) - f(y)| < 2^{-k}$$

Background – Elementary Functions

Definition ([Tenenbaum and Pollard, 1985])

The **elementary functions** on $\mathbb{R}$ are partial functions defined by expressions built up from

- computable reals, and
- the variable x,

by applying (repeatedly) the basic operations below on elementary functions f, g:

- $(f+g)(x) = f(x) + g(x)$
- $(f \cdot g)(x) = f(x)g(x)$
- $\mathrm{div}_f(x) = \frac{1}{f(x)}$ where $\frac{1}{0} = \uparrow$
- $\mathrm{root}_{n,f}(x) = \sqrt[n]{f(x)}$ where $0 < n \in \mathbb{N}$
- $\ln_f(x) = \ln(f(x))$
- $\exp_f(x) = e^{f(x)}$
- $\sin_f(x) = \sin(f(x))$
- $\arcsin_f(x) = \arcsin(f(x))$

Problem

The domains of elementary functions are not all open!

Solution: Modifications

- We define $\sqrt[n]{x} = 0$ for $x < 0$ when n is even.
- We extend the definition of $\arcsin(x)$ to be $\frac{\pi}{2}$ for $x > 1$ and to be $-\frac{\pi}{2}$ for $x < -1$.

Background – Elementary Functions

Definition ([Tenenbaum and Pollard, 1985])

The **elementary functions** on $\mathbb{R}$ are partial functions defined by expressions built up from

- computable reals, and
- the variable x,

by applying (repeatedly) the basic operations below on elementary functions f, g:

- $(f+g)(x) = f(x) + g(x)$
- $(f \cdot g)(x) = f(x)g(x)$
- $\mathrm{div}_f(x) = \frac{1}{f(x)}$ where $\frac{1}{0} = \uparrow$
- $\mathrm{root}_{n,f}(x) = \sqrt[n]{f(x)}$ where $0 < n \in \mathbb{N}$
- $\ln_f(x) = \ln(f(x))$
- $\exp_f(x) = e^{f(x)}$
- $\sin_f(x) = \sin(f(x))$
- $\arcsin_f(x) = \arcsin(f(x))$

Problem

The domains of elementary functions are not all open!

Solution: Modifications

- We define $\sqrt[n]{x} = 0$ for $x < 0$ when n is even.
- We extend the definition of $\arcsin(x)$ to be $\frac{\pi}{2}$ for $x > 1$ and to be $-\frac{\pi}{2}$ for $x < -1$.

Background – Elementary Functions

Definition ([Tenenbaum and Pollard, 1985])

The **elementary functions** on $\mathbb{R}$ are partial functions defined by expressions built up from

- computable reals, and
- the variable x,

by applying (repeatedly) the basic operations below on elementary functions f, g:

- $(f + g)(x) = f(x) + g(x)$
- $(f \cdot g)(x) = f(x)g(x)$
- $\mathrm{div}_f(x) = \frac{1}{f(x)}$ where $\frac{1}{0} = \uparrow$
- $\mathrm{root}_{n,f}(x) = \sqrt[n]{f(x)}$ where $0 < n \in \mathbb{N}$
- $\ln_f(x) = \ln(f(x))$
- $\exp_f(x) = e^{f(x)}$
- $\sin_f(x) = \sin(f(x))$
- $\arcsin_f(x) = \arcsin(f(x))$

Problem

The domains of elementary functions are not all open!

Solution: Modifications

- We define $\sqrt[n]{x} = 0$ for $x < 0$ when n is even.
- We extend the definition of $\arcsin(x)$ to be $\frac{\pi}{2}$ for $x > 1$ and to be $-\frac{\pi}{2}$ for $x < -1$.

Background – Elementary Functions

Definition ([Tenenbaum and Pollard, 1985])

The **elementary functions** on $\mathbb{R}$ are partial functions defined by expressions built up from

- computable reals, and
- the variable x,

by applying (repeatedly) the basic operations below on elementary functions f, g:

- $(f+g)(x) = f(x) + g(x)$
- $(f \cdot g)(x) = f(x)g(x)$
- $\mathrm{div}_f(x) = \frac{1}{f(x)}$ where $\frac{1}{0} = \uparrow$
- $\mathrm{root}_{n,f}(x) = \sqrt[n]{f(x)}$ where $0 < n \in \mathbb{N}$
- $\ln_f(x) = \ln(f(x))$
- $\exp_f(x) = e^{f(x)}$
- $\sin_f(x) = \sin(f(x))$
- $\arcsin_f(x) = \arcsin(f(x))$

Problem

The domains of elementary functions are not all open!

Solution: Modifications

- We define $\sqrt[n]{x} = 0$ for $x < 0$ when n is even.
- We extend the definition of $\arcsin(x)$ to be $\frac{\pi}{2}$ for $x > 1$ and to be $-\frac{\pi}{2}$ for $x < -1$.

Background – Elementary Functions

Definition ([Tenenbaum and Pollard, 1985])

The **elementary functions** on $\mathbb{R}$ are partial functions defined by expressions built up from

- computable reals, and
- the variable x,

by applying (repeatedly) the basic operations below on elementary functions f, g:

- $(f+g)(x) = f(x) + g(x)$
- $(f \cdot g)(x) = f(x)g(x)$
- $\mathrm{div}_f(x) = \frac{1}{f(x)}$ where $\frac{1}{0} = \uparrow$
- $\mathrm{root}_{n,f}(x) = \sqrt[n]{f(x)}$ where $0 < n \in \mathbb{N}$
- $\ln_f(x) = \ln(f(x))$
- $\exp_f(x) = e^{f(x)}$
- $\sin_f(x) = \sin(f(x))$
- $\arcsin_f(x) = \arcsin(f(x))$

Problem

The domains of elementary functions are not all open!

Solution: Modifications

- We define $\sqrt[n]{x} = 0$ for $x < 0$ when n is even.
- We extend the definition of $\arcsin(x)$ to be $\frac{\pi}{2}$ for $x > 1$ and to be $-\frac{\pi}{2}$ for $x < -1$.

Background – Elementary Functions

Definition ([Tenenbaum and Pollard, 1985])

The **elementary functions** on $\mathbb{R}$ are partial functions defined by expressions built up from

- computable reals, and
- the variable x,

by applying (repeatedly) the basic operations below on elementary functions f, g:

- $(f+g)(x) = f(x) + g(x)$
- $(f \cdot g)(x) = f(x)g(x)$
- $\mathrm{div}_f(x) = \frac{1}{f(x)}$ where $\frac{1}{0} = \uparrow$
- $\mathrm{root}_{n,f}(x) = \sqrt[n]{f(x)}$ where $0 < n \in \mathbb{N}$
- $\ln_f(x) = \ln(f(x))$
- $\exp_f(x) = e^{f(x)}$
- $\sin_f(x) = \sin(f(x))$
- $\arcsin_f(x) = \arcsin(f(x))$

Problem

The domains of elementary functions are not all open!

Solution: Modifications

- We define $\sqrt[n]{x} = 0$ for $x < 0$ when n is even.
- We extend the definition of $\arcsin(x)$ to be $\frac{\pi}{2}$ for $x > 1$ and to be $-\frac{\pi}{2}$ for $x < -1$.

Background – Elementary Functions

Definition ([Tenenbaum and Pollard, 1985])

The **elementary functions** on $\mathbb{R}$ are partial functions defined by expressions built up from

- computable reals, and
- the variable x,

by applying (repeatedly) the basic operations below on elementary functions f, g:

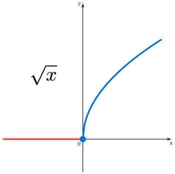
- $(f+g)(x) = f(x) + g(x)$
- $(f \cdot g)(x) = f(x)g(x)$
- $\mathrm{div}_f(x) = \frac{1}{f(x)}$ where $\frac{1}{0} = \uparrow$
- $\mathrm{root}_{n,f}(x) = \sqrt[n]{f(x)}$ where $0 < n \in \mathbb{N}$
- $\ln_f(x) = \ln(f(x))$
- $\exp_f(x) = e^{f(x)}$
- $\sin_f(x) = \sin(f(x))$
- $\arcsin_f(x) = \arcsin(f(x))$

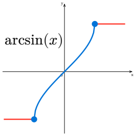
Problem

The domains of elementary functions are not all open!

Solution: Modifications

- We define $\sqrt[n]{x} = 0$ for $x < 0$ when n is even.
- We extend the definition of $\arcsin(x)$ to be $\frac{\pi}{2}$ for $x > 1$ and to be $-\frac{\pi}{2}$ for $x < -1$.





Background – Elementary Functions

Definition ([Tenenbaum and Pollard, 1985])

The **elementary functions** on $\mathbb{R}$ are partial functions defined by expressions built up from

- computable reals, and
- the variable x,

by applying (repeatedly) the basic operations below on elementary functions f, g:

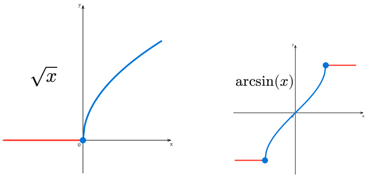
- $(f+g)(x) = f(x) + g(x)$
- $(f \cdot g)(x) = f(x)g(x)$
- $\mathrm{div}_f(x) = \frac{1}{f(x)}$ where $\frac{1}{0} = \uparrow$
- $\mathrm{root}_{n,f}(x) = \sqrt[n]{f(x)}$ where $0 < n \in \mathbb{N}$
- $\ln_f(x) = \ln(f(x))$
- $\exp_f(x) = e^{f(x)}$
- $\sin_f(x) = \sin(f(x))$
- $\arcsin_f(x) = \arcsin(f(x))$

Problem

The domains of elementary functions are not all open!

Solution: Modifications

- We define $\sqrt[n]{x} = 0$ for $x < 0$ when n is even.
- We extend the definition of $\arcsin(x)$ to be $\frac{\pi}{2}$ for $x > 1$ and to be $-\frac{\pi}{2}$ for $x < -1$.



Background – Elementary Functions

Definition ([Tenenbaum and Pollard, 1985])

The **elementary functions** on $\mathbb{R}$ are partial functions defined by expressions built up from

- computable reals, and
- the variable x,

by applying (repeatedly) the basic operations below on elementary functions f, g:

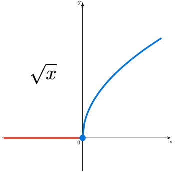
- $(f+g)(x) = f(x) + g(x)$
- $(f \cdot g)(x) = f(x)g(x)$
- $\mathrm{div}_f(x) = \frac{1}{f(x)}$ where $\frac{1}{0} = \uparrow$
- $\mathrm{root}_{n,f}(x) = \sqrt[n]{f(x)}$ where $0 < n \in \mathbb{N}$
- $\ln_f(x) = \ln(f(x))$
- $\exp_f(x) = e^{f(x)}$
- $\sin_f(x) = \sin(f(x))$
- $\arcsin_f(x) = \arcsin(f(x))$

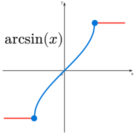
Problem

The domains of elementary functions are not all open!

Solution: Modifications

- We define $\sqrt[n]{x} = 0$ for $x < 0$ when n is even.
- We extend the definition of $\arcsin(x)$ to be $\frac{\pi}{2}$ for $x > 1$ and to be $-\frac{\pi}{2}$ for $x < -1$.





Contributions

Recall: Equivalence Theorem, [Fu and Zucker, 2014]

For any acceptable function $f : \mathbb{R} \to \mathbb{R}$ and any effective open exhaustion X for $\mathbf{dom}(f)$, the following are equivalent:

- f is an α-computable function.
- f is **WhileCC**-approximable.
- f is GL-computable w.r.t. X.
- f is effectively locally uniformly multipolynomially approximable w.r.t. X.

Theorem 1 (WhileCC-approximability Theorem)

All elementary functions are **WhileCC**-approximable.

Theorem 2 (Acceptability Theorem)

All elementary functions are acceptable.

Contributions

Recall: Equivalence Theorem, [Fu and Zucker, 2014]
For any acceptable function $f : \mathbb{R} \to \mathbb{R}$ and any effective open exhaustion X for $\mathbf{dom}(f)$, the following are equivalent:

- f is an α-computable function.
- f is **WhileCC**-approximable.
- f is GL-computable w.r.t. X.
- f is effectively locally uniformly multipolynomially approximable w.r.t. X.

Theorem 1 (WhileCC-approximability Theorem)

All elementary functions are **WhileCC**-approximable.

Theorem 2 (Acceptability Theorem)

All elementary functions are acceptable.

Contributions

Recall: Equivalence Theorem, [Fu and Zucker, 2014]

For any acceptable function $f : \mathbb{R} \to \mathbb{R}$ and any effective open exhaustion X for $\mathbf{dom}(f)$, the following are equivalent:

- f is an α-computable function.
- f is **WhileCC**-approximable.
- f is GL-computable w.r.t. X.
- f is effectively locally uniformly multipolynomially approximable w.r.t. X.

Theorem 1 (WhileCC-approximability Theorem)

All elementary functions are **WhileCC**-approximable.

Theorem 2 (Acceptability Theorem)

All elementary functions are acceptable.

Contributions

Recall: Equivalence Theorem, [Fu and Zucker, 2014]
For any acceptable function $f : \mathbb{R} \to \mathbb{R}$ and any effective open exhaustion X for $\mathbf{dom}(f)$, the following are equivalent:

- f is an α-computable function.
- f is **WhileCC**-approximable.
- f is GL-computable w.r.t. X.
- f is effectively locally uniformly multipolynomially approximable w.r.t. X.

Theorem 1 (WhileCC-approximability Theorem)

All elementary functions are **WhileCC**-approximable.

Theorem 2 (Acceptability Theorem)

All elementary functions are acceptable.

Contributions

Recall: Equivalence Theorem, [Fu and Zucker, 2014]

For any acceptable function $f : \mathbb{R} \to \mathbb{R}$ and any effective open exhaustion X for $\mathbf{dom}(f)$, the following are equivalent:

- f is an α-computable function.
- f is **WhileCC**-approximable.
- f is GL-computable w.r.t. X.
- f is effectively locally uniformly multipolynomially approximable w.r.t. X.

Theorem 1 (WhileCC-approximability Theorem)

All elementary functions are **WhileCC**-approximable.

Theorem 2.1 (Acceptability Theorem: Part 1)

The domain of any elementary function has an effective open exhaustion.

Theorem 2.2 (Acceptability Theorem: Part 2)

Any elementary function is effectively locally uniformly continuous w.r.t. an effective open exhaustion for its domain.

Contributions

Recall: Equivalence Theorem, [Fu and Zucker, 2014]

For any acceptable function $f : \mathbb{R} \to \mathbb{R}$ and any effective open exhaustion X for $\mathbf{dom}(f)$, the following are equivalent:

- f is an α-computable function.
- f is **WhileCC**-approximable.
- f is GL-computable w.r.t. X.
- f is effectively locally uniformly multipolynomially approximable w.r.t. X.

Theorem 1 (WhileCC-approximability Theorem)

All elementary functions are **WhileCC**-approximable.

Theorem 2.1 (Acceptability Theorem: Part 1)

The domain of any elementary function has an effective open exhaustion.

Theorem 2.2 (Acceptability Theorem: Part 2)

Any elementary function is effectively locally uniformly continuous w.r.t. an effective open exhaustion for its domain.

Result 1

Theorem 1 (WhileCC-approximability Theorem)

All elementary functions are **WhileCC**-approximable.

This is the easiest part, yet occupies about 30 pages of my master's thesis ... ☺

Result 1

Theorem 1 (**WhileCC**-approximability Theorem)

All elementary functions are **WhileCC**-approximable.

This is the easiest part, yet occupies about 30 pages of my master's thesis . . . ☺

Result 1

Theorem 1 (WhileCC-approximability Theorem)

All elementary functions are **WhileCC**-approximable.

This is the easiest part, yet occupies about 30 pages of my master's thesis . . . ☺

Result 1 - Background - **WhileCC** Programming Language

Syntax

- Terms

  $t^s ::= x^s \mid F(t_1^{s_1}, \ldots, t_m^{s_m})$

- Statements

  $S ::=$

  skip | div | $\bar{x} := \bar{t}$ | $S_1 \; S_2$

  | if b then S_1 else S_2 fi

  | while b do S_0 od

  | $n :=$ choose $(z : \text{nat}) : P(z, \bar{t})$

- Procedures

  $P ::=$ proc D begin S end

Algebra $\mathcal{R}$

$0_\mathsf{R}, 1_\mathsf{R}, -1_\mathsf{R} : \ \twoheadrightarrow \mathbb{R}$
$+_\mathsf{R}, \times_\mathsf{R} : \mathbb{R} \times \mathbb{R} \twoheadrightarrow \mathbb{R}$
$+_\mathsf{N}, \times_\mathsf{N} : \mathbb{N} \times \mathbb{N} \twoheadrightarrow \mathbb{N}$
$\mathsf{inv}_\mathsf{R} : \mathbb{R} \rightarrow \mathbb{R}$
$0_\mathsf{N} : \ \twoheadrightarrow \mathbb{N}$
$\mathsf{suc}_\mathsf{N} : \mathbb{N} \twoheadrightarrow \mathbb{N}$
$\mathsf{tt}, \mathsf{ff} : \ \twoheadrightarrow \mathbb{B}$
$\mathsf{and}, \mathsf{or} : \mathbb{B} \times \mathbb{B} \twoheadrightarrow \mathbb{B}$
$\mathsf{not} : \mathbb{B} \twoheadrightarrow \mathbb{B}$
$=_\mathsf{N}, <_\mathsf{N} : \mathbb{N} \times \mathbb{N} \twoheadrightarrow \mathbb{B}$
$=_\mathsf{real}, <_\mathsf{R} : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{B}$

Semantics

$$\mathsf{inv}_\mathsf{R}(x) = \begin{cases} 1/x & \text{if } x \neq 0 \\ \uparrow & \text{otherwise} \end{cases}$$

$$=_\mathsf{real}(x, y) = \begin{cases} \mathsf{ff} & \text{if } x \neq y \\ \uparrow & \text{otherwise} \end{cases}$$

$$<_\mathsf{R}(x, y) = \begin{cases} \mathsf{tt} & \text{if } x < y \\ \mathsf{ff} & \text{if } x > y \\ \uparrow & \text{if } x = y \end{cases}$$

The Semantics of a program P is denoted by a many-valued function $P^{\mathcal{R}}$.

Result 1 - Background - **WhileCC** Programming Language

Syntax

- Terms

  $t^s ::= x^s \mid F(t_1^{s_1}, \ldots, t_m^{s_m})$
- Statements

  $S ::=$

  skip | div | $\bar{x} := \bar{t}$ | $S_1\ S_2$

  | if b then S_1 else S_2 fi

  | while b do S_0 od

  | $n :=$ choose $(z : \text{nat}) : P(z, \bar{t})$
- Procedures

  $P ::=$ proc D begin S end

Algebra $\mathcal{R}$

$0_R, 1_R, -1_R : \ \twoheadrightarrow \mathbb{R}$
$+_R, \times_R : \mathbb{R} \times \mathbb{R} \twoheadrightarrow \mathbb{R}$
$+_N, \times_N : \mathbb{N} \times \mathbb{N} \twoheadrightarrow \mathbb{N}$
$\text{inv}_R : \mathbb{R} \to \mathbb{R}$
$0_N : \ \twoheadrightarrow \mathbb{N}$
$\text{suc}_N : \mathbb{N} \twoheadrightarrow \mathbb{N}$
$\text{tt}, \text{ff} : \ \twoheadrightarrow \mathbb{B}$
$\text{and}, \text{or} : \mathbb{B} \times \mathbb{B} \twoheadrightarrow \mathbb{B}$
$\text{not} : \mathbb{B} \twoheadrightarrow \mathbb{B}$
$=_N, <_N : \mathbb{N} \times \mathbb{N} \twoheadrightarrow \mathbb{B}$
$=_{\text{real}}, <_R : \mathbb{R} \times \mathbb{R} \to \mathbb{B}$

Semantics

$$\text{inv}_R(x) = \begin{cases} 1/x & \text{if } x \neq 0 \\ \uparrow & \text{otherwise} \end{cases}$$

$$=_{\text{real}}(x, y) = \begin{cases} \text{ff} & \text{if } x \neq y \\ \uparrow & \text{otherwise} \end{cases}$$

$$<_R(x, y) = \begin{cases} \text{tt} & \text{if } x < y \\ \text{ff} & \text{if } x > y \\ \uparrow & \text{if } x = y \end{cases}$$

The Semantics of a program P is denoted by a many-valued function $P^{\mathcal{R}}$.

Result 1 - Background - **WhileCC** Programming Language

Syntax

- Terms

$$t^s ::= x^s \mid F(t_1^{s_1}, \ldots, t_m^{s_m})$$

- Statements

$S ::=$

skip | div | $\bar{x} := \bar{t}$ | $S_1\ S_2$

| if b then S_1 else S_2 fi

| while b do S_0 od

| $n :=$ choose $(z : \text{nat}) : P(z, \bar{t})$

- Procedures

$P ::=$ proc D begin S end

Algebra $\mathcal{R}$

$0_R, 1_R, -1_R : \ \to \mathbb{R}$
$+_R, \times_R : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$
$+_N, \times_N : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$
$\text{inv}_R : \mathbb{R} \to \mathbb{R}$
$0_N : \ \to \mathbb{N}$
$\text{suc}_N : \mathbb{N} \to \mathbb{N}$
tt, ff : $\ \to \mathbb{B}$
and, or : $\mathbb{B} \times \mathbb{B} \to \mathbb{B}$
not : $\mathbb{B} \to \mathbb{B}$
$=_N, <_N : \mathbb{N} \times \mathbb{N} \to \mathbb{B}$
$=_{\text{real}}, <_R : \mathbb{R} \times \mathbb{R} \to \mathbb{B}$

Semantics

$$\text{inv}_R(x) = \begin{cases} 1/x & \text{if } x \neq 0 \\ \uparrow & \text{otherwise} \end{cases}$$

$$=_{\text{real}}(x, y) = \begin{cases} \text{ff} & \text{if } x \neq y \\ \uparrow & \text{otherwise} \end{cases}$$

$$<_R(x, y) = \begin{cases} \text{tt} & \text{if } x < y \\ \text{ff} & \text{if } x > y \\ \uparrow & \text{if } x = y \end{cases}$$

The Semantics of a program P is denoted by a many-valued function $P^{\mathcal{R}}$.

Syntax

- Terms

  $t^s ::= x^s \mid F(t_1^{s_1}, \ldots, t_m^{s_m})$
- Statements

  $S ::=$

  skip | div | $\bar{x} := \bar{t}$ | $S_1\ S_2$

  | if b then S_1 else S_2 fi

  | while b do S_0 od

  | $n :=$ choose $(z : \text{nat}) : P(z, \bar{t})$
- Procedures

  $P ::=$ proc D begin S end

Algebra $\mathcal{R}$

$0_R, 1_R, -1_R : \twoheadrightarrow \mathbb{R}$
$+_R, \times_R : \mathbb{R} \times \mathbb{R} \twoheadrightarrow \mathbb{R}$
$+_N, \times_N : \mathbb{N} \times \mathbb{N} \twoheadrightarrow \mathbb{N}$
$\text{inv}_R : \mathbb{R} \rightarrow \mathbb{R}$
$0_N : \twoheadrightarrow \mathbb{N}$
$\text{suc}_N : \mathbb{N} \twoheadrightarrow \mathbb{N}$
tt, ff : $\twoheadrightarrow \mathbb{B}$
and, or : $\mathbb{B} \times \mathbb{B} \twoheadrightarrow \mathbb{B}$
not : $\mathbb{B} \twoheadrightarrow \mathbb{B}$
$=_N, <_N : \mathbb{N} \times \mathbb{N} \twoheadrightarrow \mathbb{B}$
$=_{\text{real}}, <_R : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{B}$

Semantics

$$\text{inv}_R(x) = \begin{cases} 1/x & \text{if } x \neq 0 \\ \uparrow & \text{otherwise} \end{cases}$$

$$=_{\text{real}}(x, y) = \begin{cases} \text{ff} & \text{if } x \neq y \\ \uparrow & \text{otherwise} \end{cases}$$

$$<_R(x, y) = \begin{cases} \text{tt} & \text{if } x < y \\ \text{ff} & \text{if } x > y \\ \uparrow & \text{if } x = y \end{cases}$$

The Semantics of a program P is denoted by a many-valued function $P^{\mathcal{R}}$.

Result 1 - Background - **WhileCC** Programming Language

Syntax

- Terms

  $t^s ::= x^s \mid F(t_1^{s_1}, \ldots, t_m^{s_m})$

- Statements

  $S ::=$

  skip | div | $\bar{x} := \bar{t}$ | S_1 S_2

  | if b then S_1 else S_2 fi

  | while b do S_0 od

  | $n :=$ choose $(z : \text{nat}) : P(z, \bar{t})$

- Procedures

  $P ::=$ proc D begin S end

Algebra $\mathcal{R}$

$0_\mathsf{R},\ 1_\mathsf{R},\ -1_\mathsf{R} :\ \twoheadrightarrow \mathbb{R}$
$+_\mathsf{R},\ \times_\mathsf{R} : \mathbb{R} \times \mathbb{R} \twoheadrightarrow \mathbb{R}$
$+_\mathsf{N},\ \times_\mathsf{N} : \mathbb{N} \times \mathbb{N} \twoheadrightarrow \mathbb{N}$
$\mathsf{inv}_\mathsf{R} : \mathbb{R} \rightarrow \mathbb{R}$
$0_\mathsf{N} :\ \twoheadrightarrow \mathbb{N}$
$\mathsf{suc}_\mathsf{N} : \mathbb{N} \twoheadrightarrow \mathbb{N}$
$\mathsf{tt},\ \mathsf{ff} :\ \twoheadrightarrow \mathbb{B}$
$\mathsf{and},\ \mathsf{or} : \mathbb{B} \times \mathbb{B} \twoheadrightarrow \mathbb{B}$
$\mathsf{not} : \mathbb{B} \twoheadrightarrow \mathbb{B}$
$=_\mathsf{N},\ <_\mathsf{N} : \mathbb{N} \times \mathbb{N} \twoheadrightarrow \mathbb{B}$
$=_\mathsf{real},\ <_\mathsf{R} : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{B}$

Semantics

$$\mathsf{inv}_\mathsf{R}(x) = \begin{cases} 1/x & \text{if } x \neq 0 \\ \uparrow & \text{otherwise} \end{cases}$$

$$=_\mathsf{real}(x, y) = \begin{cases} \mathsf{ff} & \text{if } x \neq y \\ \uparrow & \text{otherwise} \end{cases}$$

$$<_\mathsf{R}(x, y) = \begin{cases} \mathsf{tt} & \text{if } x < y \\ \mathsf{ff} & \text{if } x > y \\ \uparrow & \text{if } x = y \end{cases}$$

The Semantics of a program P is denoted by a many-valued function $P^{\mathcal{R}}$.

Result 1 - Background - **WhileCC** Programming Language

Syntax

- Terms

  $t^s ::= x^s \mid F(t_1^{s_1}, \ldots, t_m^{s_m})$
- Statements

  $S ::=$

  skip $\mid$ div $\mid \bar{x} := \bar{t} \mid S_1\ S_2$

  $\mid$ if b then S_1 else S_2 fi

  $\mid$ while b do S_0 od

  $\mid n :=$ choose $(z : \text{nat}) : P(z, \bar{t})$
- Procedures

  $P ::=$ proc D begin S end

Algebra $\mathcal{R}$

$0_\mathsf{R}, 1_\mathsf{R}, -1_\mathsf{R} : \ \twoheadrightarrow \mathbb{R}$
$+_\mathsf{R}, \times_\mathsf{R} : \mathbb{R} \times \mathbb{R} \twoheadrightarrow \mathbb{R}$
$+_\mathsf{N}, \times_\mathsf{N} : \mathbb{N} \times \mathbb{N} \twoheadrightarrow \mathbb{N}$
$\mathsf{inv}_\mathsf{R} : \mathbb{R} \to \mathbb{R}$
$0_\mathsf{N} : \ \twoheadrightarrow \mathbb{N}$
$\mathsf{suc}_\mathsf{N} : \mathbb{N} \twoheadrightarrow \mathbb{N}$
$\mathsf{tt}, \mathsf{ff} : \ \twoheadrightarrow \mathbb{B}$
$\mathsf{and}, \mathsf{or} : \mathbb{B} \times \mathbb{B} \twoheadrightarrow \mathbb{B}$
$\mathsf{not} : \mathbb{B} \twoheadrightarrow \mathbb{B}$
$=_\mathsf{N}, <_\mathsf{N} : \mathbb{N} \times \mathbb{N} \twoheadrightarrow \mathbb{B}$
$=_\mathsf{real}, <_\mathsf{R} : \mathbb{R} \times \mathbb{R} \to \mathbb{B}$

Semantics

$$\mathsf{inv}_\mathsf{R}(x) = \begin{cases} 1/x & \text{if } x \neq 0 \\ \uparrow & \text{otherwise} \end{cases}$$

$$=_\mathsf{real}(x, y) = \begin{cases} \mathsf{ff} & \text{if } x \neq y \\ \uparrow & \text{otherwise} \end{cases}$$

$$<_\mathsf{R}(x, y) = \begin{cases} \mathsf{tt} & \text{if } x < y \\ \mathsf{ff} & \text{if } x > y \\ \uparrow & \text{if } x = y \end{cases}$$

The Semantics of a program P is denoted by a many-valued function $P^{\mathcal{R}}$.

Result 1 - Background - **WhileCC** Programming Language

Syntax

- Terms

  $t^s ::= x^s \mid F(t_1^{s_1}, \ldots, t_m^{s_m})$

- Statements

  $S ::=$

  skip | div | $\bar{x} := \bar{t}$ | $S_1\ S_2$

  | if b then S_1 else S_2 fi

  | while b do S_0 od

  | $n :=$ choose $(z : \text{nat}) : P(z, \bar{t})$

- Procedures

  $P ::=$ proc D begin S end

Algebra $\mathcal{R}$

$0_\mathsf{R}, 1_\mathsf{R}, -1_\mathsf{R} : \ \twoheadrightarrow \mathbb{R}$

$+_\mathsf{R}, \times_\mathsf{R} : \mathbb{R} \times \mathbb{R} \twoheadrightarrow \mathbb{R}$

$+_\mathsf{N}, \times_\mathsf{N} : \mathbb{N} \times \mathbb{N} \twoheadrightarrow \mathbb{N}$

$\mathsf{inv}_\mathsf{R} : \mathbb{R} \ \to \ \mathbb{R}$

$0_\mathsf{N} : \ \twoheadrightarrow \mathbb{N}$

$\mathsf{suc}_\mathsf{N} : \mathbb{N} \twoheadrightarrow \mathbb{N}$

$\mathsf{tt}, \mathsf{ff} : \ \twoheadrightarrow \mathbb{B}$

$\mathsf{and}, \mathsf{or} : \mathbb{B} \times \mathbb{B} \twoheadrightarrow \mathbb{B}$

$\mathsf{not} : \mathbb{B} \twoheadrightarrow \mathbb{B}$

$=_\mathsf{N}, <_\mathsf{N} : \mathbb{N} \times \mathbb{N} \twoheadrightarrow \mathbb{B}$

$=_\mathsf{real}, <_\mathsf{R} : \mathbb{R} \times \mathbb{R} \ \to \ \mathbb{B}$

Semantics

$$\mathsf{inv}_\mathsf{R}(x) = \begin{cases} 1/x & \text{if } x \neq 0 \\ \uparrow & \text{otherwise} \end{cases}$$

$$=_\mathsf{real}(x, y) = \begin{cases} \mathsf{ff} & \text{if } x \neq y \\ \uparrow & \text{otherwise} \end{cases}$$

$$<_\mathsf{R}(x, y) = \begin{cases} \mathsf{tt} & \text{if } x < y \\ \mathsf{ff} & \text{if } x > y \\ \uparrow & \text{if } x = y \end{cases}$$

The Semantics of a program P is denoted by a many-valued function $P^\mathcal{R}$.

Result 1 - Background

Definition (WhileCC-approximability, [Fu and Zucker, 2014])

A **WhileCC**-procedure P of type real $\times$ nat $\to$ real on $\mathcal{R}$ is said to *approximate* a function $f : \mathbb{R} \to \mathbb{R}$ iff for all $n \in \mathbb{N}$ and all $x \in \mathbb{R}$:

- $x \in \mathbf{dom}(f) \implies \emptyset \neq P^{\mathcal{R}}(x, n) \subseteq \mathbf{Nbd}(f(x), 2^{-n})$, and
- $x \notin \mathbf{dom}(f) \implies P^{\mathcal{R}}(x) = \emptyset$

where $\mathbf{Nbd}(y, r)$ has the standard definition of neighborhood on $\mathbb{R}$ i.e.,

$$\mathbf{Nbd}(y, r) = \{z \in \mathbb{R} \mid |y - z| < r\}.$$

Goal

Construct **WhileCC**-procedures approximating elementary functions by induction

Challenge: Using comparison operators introduces undefinedness.

How do we **WhileCC**-approximate "piecewise" functions?

Result 1 - Background

Definition (**WhileCC**-approximability, [Fu and Zucker, 2014])

A **WhileCC**-procedure P of type real $\times$ nat $\to$ real on $\mathcal{R}$ is said to *approximate* a function $f : \mathbb{R} \to \mathbb{R}$ iff for all $n \in \mathbb{N}$ and all $x \in \mathbb{R}$:

- $x \in \mathbf{dom}(f) \implies \emptyset \neq P^{\mathcal{R}}(x, n) \subseteq \mathbf{Nbd}(f(x), 2^{-n})$, and
- $x \notin \mathbf{dom}(f) \implies P^{\mathcal{R}}(x) = \emptyset$

where $\mathbf{Nbd}(y, r)$ has the standard definition of neighborhood on $\mathbb{R}$ i.e.,

$$\mathbf{Nbd}(y, r) = \{z \in \mathbb{R} \mid |y - z| < r\}.$$

Goal

Construct **WhileCC**-procedures approximating elementary functions by induction

Challenge: Using comparison operators introduces undefinedness.

How do we **WhileCC**-approximate "piecewise" functions?

Result 1 - Background

Definition (**WhileCC**-approximability, [Fu and Zucker, 2014])

A **WhileCC**-procedure P of type real $\times$ nat $\to$ real on $\mathcal{R}$ is said to *approximate* a function $f : \mathbb{R} \to \mathbb{R}$ iff for all $n \in \mathbb{N}$ and all $x \in \mathbb{R}$:

- $x \in \mathbf{dom}(f) \implies \emptyset \neq P^{\mathcal{R}}(x, n) \subseteq \mathbf{Nbd}(f(x), 2^{-n})$, and
- $x \notin \mathbf{dom}(f) \implies P^{\mathcal{R}}(x) = \emptyset$

where $\mathbf{Nbd}(y, r)$ has the standard definition of neighborhood on $\mathbb{R}$ i.e.,

$$\mathbf{Nbd}(y, r) = \{z \in \mathbb{R} \mid |y - z| < r\}.$$

Goal

Construct **WhileCC**-procedures approximating elementary functions by induction

Challenge: Using comparison operators introduces undefinedness.

How do we **WhileCC**-approximate "piecewise" functions?

Result 1 - Background

Definition (**WhileCC**-approximability, [Fu and Zucker, 2014])

A **WhileCC**-procedure P of type real $\times$ nat $\to$ real on $\mathcal{R}$ is said to *approximate* a function $f : \mathbb{R} \to \mathbb{R}$ iff for all $n \in \mathbb{N}$ and all $x \in \mathbb{R}$:

- $x \in \mathbf{dom}(f) \implies \emptyset \neq P^{\mathcal{R}}(x, n) \subseteq \mathbf{Nbd}(f(x), 2^{-n})$, and
- $x \notin \mathbf{dom}(f) \implies P^{\mathcal{R}}(x) = \emptyset$

where $\mathbf{Nbd}(y, r)$ has the standard definition of neighborhood on $\mathbb{R}$ i.e.,

$$\mathbf{Nbd}(y, r) = \{z \in \mathbb{R} \mid |y - z| < r\}.$$

Goal

Construct **WhileCC**-procedures approximating elementary functions by induction

Challenge: Using comparison operators introduces undefinedness.

How do we **WhileCC**-approximate "piecewise" functions?

Result 1 - Background

Definition (WhileCC-approximability, [Fu and Zucker, 2014])

A **WhileCC**-procedure P of type real $\times$ nat $\to$ real on $\mathcal{R}$ is said to *approximate* a function $f : \mathbb{R} \to \mathbb{R}$ iff for all $n \in \mathbb{N}$ and all $x \in \mathbb{R}$:

- $x \in \mathbf{dom}(f) \implies \emptyset \neq P^{\mathcal{R}}(x, n) \subseteq \mathbf{Nbd}(f(x), 2^{-n})$, and
- $x \notin \mathbf{dom}(f) \implies P^{\mathcal{R}}(x) = \emptyset$

where $\mathbf{Nbd}(y, r)$ has the standard definition of neighborhood on $\mathbb{R}$ i.e.,

$$\mathbf{Nbd}(y, r) = \{z \in \mathbb{R} \mid |y - z| < r\}.$$

Goal

Construct **WhileCC**-procedures approximating elementary functions by induction

Challenge: Using comparison operators introduces undefinedness.

How do we **WhileCC**-approximate "piecewise" functions?

Result 1 - Background

Definition (**WhileCC**-approximability, [Fu and Zucker, 2014])

A **WhileCC**-procedure P of type real $\times$ nat $\to$ real on $\mathcal{R}$ is said to *approximate* a function $f : \mathbb{R} \to \mathbb{R}$ iff for all $n \in \mathbb{N}$ and all $x \in \mathbb{R}$:

- $x \in \mathbf{dom}(f) \implies \emptyset \neq P^{\mathcal{R}}(x, n) \subseteq \mathbf{Nbd}(f(x), 2^{-n})$, and
- $x \notin \mathbf{dom}(f) \implies P^{\mathcal{R}}(x) = \emptyset$

where $\mathbf{Nbd}(y, r)$ has the standard definition of neighborhood on $\mathbb{R}$ i.e.,

$$\mathbf{Nbd}(y, r) = \{z \in \mathbb{R} \mid |y - z| < r\}.$$

Goal

Construct **WhileCC**-procedures approximating elementary functions by induction

Challenge: Using comparison operators introduces undefinedness.

How do we **WhileCC**-approximate "piecewise" functions?

Result 1 - Background

Definition (WhileCC-approximability, [Fu and Zucker, 2014])

A **WhileCC**-procedure P of type real $\times$ nat $\to$ real on $\mathcal{R}$ is said to *approximate* a function $f : \mathbb{R} \to \mathbb{R}$ iff for all $n \in \mathbb{N}$ and all $x \in \mathbb{R}$:

- $x \in \mathbf{dom}(f) \implies \emptyset \neq P^{\mathcal{R}}(x, n) \subseteq \mathbf{Nbd}(f(x), 2^{-n})$, and
- $x \notin \mathbf{dom}(f) \implies P^{\mathcal{R}}(x) = \emptyset$

where $\mathbf{Nbd}(y, r)$ has the standard definition of neighborhood on $\mathbb{R}$ i.e.,

$$\mathbf{Nbd}(y, r) = \{z \in \mathbb{R} \mid |y - z| < r\}.$$

Goal

Construct **WhileCC**-procedures approximating elementary functions by induction

Challenge: Using comparison operators introduces undefinedness.

How do we **WhileCC**-approximate "piecewise" functions?

Result 1 - Challenges

Problem

When defining piecewise functions, comparison makes a hole!

Example: Even Root - First Attempt

proc
  in x : real c : nat
begin
  if $x <_{\mathsf{R}} 0$ then
    return 0
  else
    return Root(x, c)
  fi
end

Result 1 - Challenges

Problem

When defining piecewise functions, comparison makes a hole!

Example: Even Root - First Attempt

```
proc
   in  x : real c : nat
begin
   if x <_R 0 then
      return 0
   else
      return Root(x, c)
   fi
end
```

Problem

When defining piecewise functions, comparison makes a hole!

Example: Even Root - First Attempt

```
proc
   in  x : real c : nat
begin
   if x <_R 0 then
      return 0
   else
      return Root(x, c)
   fi
end
```

Result 1 - Challenges

Problem

When defining piecewise functions, comparison makes a hole!

Example: Even Root - First Attempt

proc
  in x : real c : nat
begin
  if $x <_{\mathsf{R}} 0$ then
    return 0
  else
    return Root(x, c)
  fi
end

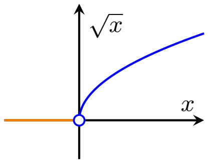
Result 1 - Challenges

Problem

When defining piecewise functions, comparison makes a hole!

Example: Even Root - First Attempt

$$
\begin{array}{l}
\textsf{proc} \\
\quad \textsf{in} \;\; x : \text{real } c : \text{nat} \\
\textsf{begin} \\
\quad \textsf{if } x <_{\mathsf{R}} 0 \textsf{ then} \\
\qquad \textsf{return } 0 \\
\quad \textsf{else} \\
\qquad \textsf{return } \text{Root}(x, c) \\
\quad \textsf{fi} \\
\textsf{end}
\end{array}
$$



Result 1 - Challenges

Problem

When defining piecewise functions, comparison makes a hole!

Solution

- Find an overlapping interval where both pieces are defined
- Use the nondeterminism of "choose"

```
proc
  in  x : real c : nat
  aux chosenVal : nat l : real
begin
  l := choose (q : real) : isCloseEnough(q, c, n)

  chosenVal := choose (k : nat) : proc
                                     in k : nat x : real
                                  begin
                                     if k =N 1 then
                                        return 0 < x
                                     else if k =N 2 then
                                        return x < l
                                     else
                                        return ff
                                     fi
                                  end
  if chosenVal =N 1 then
     return Root(x, c)
  else if chosenVal =N 2 then
     return 0
  fi
end
```

Result 1 - Challenges

Problem

When defining piecewise functions, comparison makes a hole!

Solution

- Find an overlapping interval where both pieces are defined
- Use the nondeterminism of "choose"

```
proc
  in  x : real c : nat
  aux chosenVal : nat l : real
begin
  l := choose (q : real) : isCloseEnough(q, c, n)

  chosenVal := choose (k : nat) : proc
                                    in k : nat x : real
                                  begin
                                    if k =_N 1 then
                                      return 0 < x
                                    else if k =_N 2 then
                                      return x < l
                                    else
                                      return ff
                                    fi
                                  end
  if chosenVal =_N 1 then
    return Root(x, c)
  else if chosenVal =_N 2 then
    return 0
  fi
end
```

Result 1 - Challenges

Problem

When defining piecewise functions, comparison makes a hole!

Solution

- Find an overlapping interval where both pieces are defined
- Use the nondeterminism of "choose"

```
proc
  in   x : real c : nat
  aux chosenVal : nat l : real
begin
  l := choose (q : real) : isCloseEnough(q, c, n)

  chosenVal := choose (k : nat) : proc
                                    in k : nat x : real
                                  begin
                                    if k =N 1 then
                                      return 0 < x
                                    else if k =N 2 then
                                      return x < l
                                    else
                                      return ff
                                    fi
                                  end
  if chosenVal =N 1 then
    return Root(x, c)
  else if chosenVal =N 2 then
    return 0
  fi
end
```

Result 1 - Challenges

Problem

When defining piecewise functions, comparison makes a hole!

Solution

- Find an overlapping interval where both pieces are defined
- Use the nondeterminism of "choose"

```
proc
  in  x : real c : nat
  aux chosenVal : nat l : real
begin
  l := choose (q : real) : isCloseEnough(q, c, n)

  chosenVal := choose (k : nat) : proc
                                    in k : nat x : real
                                  begin
                                    if k =N 1 then
                                      return 0 < x
                                    else if k =N 2 then
                                      return x < l
                                    else
                                      return ff
                                    fi
                                  end
  if chosenVal =N 1 then
    return Root(x, c)
  else if chosenVal =N 2 then
    return 0
  fi
end
```

Result 1 - Challenges

Problem

When defining piecewise functions, comparison makes a hole!

Solution

- Find an overlapping interval where both pieces are defined
- Use the nondeterminism of "choose"

```
proc
  in  x : real c : nat
  aux chosenVal : nat l : real
begin
  l := choose (q : real) : isCloseEnough(q, c, n)

  chosenVal := choose (k : nat) : proc
                                    in k : nat x : real
                                  begin
                                    if k =N 1 then
                                      return 0 < x
                                    else if k =N 2 then
                                      return x < l
                                    else
                                      return ff
                                    fi
                                  end
  if chosenVal =N 1 then
    return Root(x, c)
  else if chosenVal =N 2 then
    return 0
  fi
end
```

Result 1 - Challenges

Problem

When defining piecewise functions, comparison makes a hole!

Solution

- Find an overlapping interval where both pieces are defined
- Use the nondeterminism of "choose"

```
proc
  in  x : real c : nat
  aux chosenVal : nat l : real
begin
  l := choose (q : real) : isCloseEnough(q, c, n)

  chosenVal := choose (k : nat) : proc
                                     in k : nat x : real
                                   begin
                                     if k =N 1 then
                                       return 0 < x
                                     else if k =N 2 then
                                       return x < l
                                     else
                                       return ff
                                     fi
                                   end
  if chosenVal =N 1 then
    return Root(x, c)
  else if chosenVal =N 2 then
    return 0
  fi
end
```

Result 1 - Challenges

Problem

When defining piecewise functions, comparison makes a hole!

Solution

- Find an overlapping interval where both pieces are defined
- Use the nondeterminism of "choose"

```
proc
  in  x : real c : nat
  aux chosenVal : nat l : real
begin
  l := choose (q : real) : isCloseEnough(q, c, n)

  chosenVal := choose (k : nat) : proc
                                    in k : nat x : real
                                  begin
                                    if k =_N 1 then
                                      return 0 < x
                                    else if k =_N 2 then
                                      return x < l
                                    else
                                      return ff
                                    fi
                                  end
  if chosenVal =_N 1 then
    return Root(x, c)
  else if chosenVal =_N 2 then
    return 0
  fi
end
```

Result 1 - Challenges

Problem

When defining piecewise functions, comparison makes a hole!

Solution

- Find an overlapping interval where both pieces are defined
- Use the nondeterminism of "choose"

```
proc
  in  x : real c : nat
  aux chosenVal : nat l : real
begin
  l := choose (q : real) : isCloseEnough(q, c, n)

  chosenVal := choose (k : nat) : proc
                                    in k : nat x : real
                                  begin
                                    if k =_N 1 then
                                      return 0 < x
                                    else if k =_N 2 then
                                      return x < l
                                    else
                                      return ff
                                    fi
                                  end
  if chosenVal =_N 1 then
    return Root(x, c)
  else if chosenVal =_N 2 then
    return 0
  fi
end
```

Result 1 - Challenges

Problem

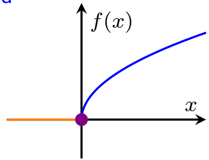
When defining piecewise functions, comparison makes a hole!

Solution

- Find an overlapping interval where both pieces are defined
- Use the nondeterminism of "choose"

```
proc
  in  x : real c : nat
  aux chosenVal : nat l : real
begin
  l := choose (q : real) : isCloseEnough(q, c, n)

  chosenVal := choose (k : nat) : proc
                                    in k : nat x : real
                                  begin
                                    if k =_N 1 then
                                      return 0 < x
                                    else if k =_N 2 then
                                      return x < l
                                    else
                                      return ff
                                    fi
                                  end
  if chosenVal =_N 1 then
    return Root(x, c)
  else if chosenVal =_N 2 then
    return 0
  fi
end
```



Result 2

Theorem 2.1 (Acceptability Theorem: Part 1)

Let $f : \mathbb{R} \to \mathbb{R}$ be an elementary function. Then, $\mathbf{dom}(f)$ has an effective open exhaustion.

First attempt – Strengthening:

Elementary function constructions preserve the property that the domain has an effective open exhaustion.

- Base cases ✓(e.g. $\sin(x)$)
- Addition and multiplication ✓(e.g. $(f+g)(x)$)
- Composition case has a counterexample:

$$f(x) = id|_{(-1,1)} \text{ and } g(x) = \begin{cases} 0 & \text{if } -1 \leq x \leq 1 \\ 1 & \text{otherwise} \end{cases}$$

$\mathbf{dom}(f)$ has an effective open exhaustion ✓
$\mathbf{dom}(g)$ has an effective open exhaustion ✓
$\mathbf{dom}(f \circ g) = [-1, 1]$ has no open exhaustion ✗

Strengthened to proving exhaustion reflection property

For any open set U with an effective open exhaustion, $f^{-1}(U)$ has an effective open exhaustion.

Proof: By induction

- Base cases ✓
- Composition ✓
- Addition and multiplication ✗

Adding decomposition of $+$ and $\cdot$

$(f+g)(x) = f(x) + g(x)$ is composed of

- $Add(x, y) = x + y$,
- $Add(x, y) = x + y$,
- $(f \times g)(x, y) = (f(x), g(y))$,
- $Diag(x) = (x, x)$

Result 2 - Challenges

Theorem 2.1 (Acceptability Theorem: Part 1)

Let $f : \mathbb{R} \to \mathbb{R}$ be an elementary function. Then, $\mathbf{dom}(f)$ has an effective open exhaustion.

First attempt – Strengthening:

Elementary function constructions preserve the property that the domain has an effective open exhaustion.

- Base cases ✓ (e.g. $\sin(x)$)
- Addition and multiplication ✓ (e.g. $(f+g)(x)$)
- Composition case has a counterexample:

$$f(x) = id|_{(-1,1)} \text{ and } g(x) = \begin{cases} 0 & \text{if } -1 \leq x \leq 1 \\ 1 & \text{otherwise} \end{cases}$$

$\mathbf{dom}(f)$ has an effective open exhaustion ✓
$\mathbf{dom}(g)$ has an effective open exhaustion ✓
$\mathbf{dom}(f \circ g) = [-1, 1]$ has no open exhaustion ✗

Strengthened to proving exhaustion reflection property

For any open set U with an effective open exhaustion, $f^{-1}(U)$ has an effective open exhaustion.

Proof: By induction

- Base cases ✓
- Composition ✓
- Addition and multiplication ✗

Adding decomposition of $+$ and $\cdot$

$(f+g)(x) = f(x) + g(x)$ is composed of

- $Add(x, y) = x + y$,
- $Add(x, y) = x + y$,
- $(f \times g)(x, y) = (f(x), g(y))$,
- $Diag(x) = (x, x)$

Result 2 - Challenges

Theorem 2.1 (Acceptability Theorem: Part 1)

Let $f : \mathbb{R} \to \mathbb{R}$ be an elementary function. Then, $\mathbf{dom}(f)$ has an effective open exhaustion.

First attempt – Strengthening:

Elementary function constructions preserve the property that the domain has an effective open exhaustion.

- Base cases ✓(e.g. $\sin(x)$)
- Addition and multiplication ✓(e.g. $(f+g)(x)$)
- Composition case has a counterexample:

$$f(x) = id|_{(-1,1)} \text{ and } g(x) = \begin{cases} 0 & \text{if } -1 \leq x \leq 1 \\ 1 & \text{otherwise} \end{cases}$$

$\mathbf{dom}(f)$ has an effective open exhaustion ✓
$\mathbf{dom}(g)$ has an effective open exhaustion ✓
$\mathbf{dom}(f \circ g) = [-1, 1]$ has no open exhaustion ✗

Strengthened to proving exhaustion reflection property

For any open set U with an effective open exhaustion, $f^{-1}(U)$ has an effective open exhaustion.

Proof: By induction

- Base cases ✓
- Composition ✓
- Addition and multiplication ✗

Adding decomposition of $+$ and $\cdot$

$(f+g)(x) = f(x) + g(x)$ is composed of

- $Add(x, y) = x + y$,
- $Add(x, y) = x + y$,
- $(f \times g)(x, y) = (f(x), g(y))$,
- $Diag(x) = (x, x)$

Result 2 - Challenges

Theorem 2.1 (Acceptability Theorem: Part 1)

Let $f : \mathbb{R} \to \mathbb{R}$ be an elementary function. Then, $\mathbf{dom}(f)$ has an effective open exhaustion.

First attempt – Strengthening:

Elementary function constructions preserve the property that the domain has an effective open exhaustion.

- Base cases ✓(e.g. $\sin(x)$)
- Addition and multiplication ✓(e.g. $(f+g)(x)$)
- Composition case has a counterexample:

$$f(x) = id|_{(-1,1)} \text{ and } g(x) = \begin{cases} 0 & \text{if } -1 \leq x \leq 1 \\ 1 & \text{otherwise} \end{cases}$$

$\mathbf{dom}(f)$ has an effective open exhaustion ✓
$\mathbf{dom}(g)$ has an effective open exhaustion ✓
$\mathbf{dom}(f \circ g) = [-1, 1]$ has no open exhaustion ✗

Strengthened to proving exhaustion reflection property

For any open set U with an effective open exhaustion, $f^{-1}(U)$ has an effective open exhaustion.

Proof: By induction

- Base cases ✓
- Composition ✓
- Addition and multiplication ✗

Adding decomposition of $+$ and $\cdot$

$(f+g)(x) = f(x) + g(x)$ is composed of

- $Add(x, y) = x + y$,
- $Add(x, y) = x + y$,
- $(f \times g)(x, y) = (f(x), g(y))$,
- $Diag(x) = (x, x)$

Result 2 - Challenges

Theorem 2.1 (Acceptability Theorem: Part 1)

Let $f : \mathbb{R} \to \mathbb{R}$ be an elementary function. Then, $\mathbf{dom}(f)$ has an effective open exhaustion.

First attempt – Strengthening:

Elementary function constructions preserve the property that the domain has an effective open exhaustion.

- Base cases ✓(e.g. $\sin(x)$)
- Addition and multiplication ✓(e.g. $(f+g)(x)$)
- Composition case has a counterexample:

$$f(x) = id|_{(-1,1)} \text{ and } g(x) = \begin{cases} 0 & \text{if } -1 \leq x \leq 1 \\ 1 & \text{otherwise} \end{cases}$$

$\mathbf{dom}(f)$ has an effective open exhaustion ✓
$\mathbf{dom}(g)$ has an effective open exhaustion ✓
$\mathbf{dom}(f \circ g) = [-1, 1]$ has no open exhaustion ✗

Strengthened to proving exhaustion reflection property

For any open set U with an effective open exhaustion, $f^{-1}(U)$ has an effective open exhaustion.

Proof: By induction

- Base cases ✓
- Composition ✓
- Addition and multiplication ✗

Adding decomposition of $+$ and $\cdot$

$(f+g)(x) = f(x) + g(x)$ is composed of

- $Add(x, y) = x + y$,
- $Add(x, y) = x + y$,
- $(f \times g)(x, y) = (f(x), g(y))$,
- $Diag(x) = (x, x)$

Result 2 - Challenges

Theorem 2.1 (Acceptability Theorem: Part 1)

Let $f : \mathbb{R} \to \mathbb{R}$ be an elementary function. Then, $\mathbf{dom}(f)$ has an effective open exhaustion.

First attempt – Strengthening:

Elementary function constructions preserve the property that the domain has an effective open exhaustion.

- Base cases ✓(e.g. $\sin(x)$)
- Addition and multiplication ✓(e.g. $(f+g)(x)$)
- Composition case has a counterexample:

$$f(x) = id|_{(-1,1)} \text{ and } g(x) = \begin{cases} 0 & \text{if } -1 \leq x \leq 1 \\ 1 & \text{otherwise} \end{cases}$$

$\mathbf{dom}(f)$ has an effective open exhaustion ✓
$\mathbf{dom}(g)$ has an effective open exhaustion ✓
$\mathbf{dom}(f \circ g) = [-1,1]$ has no open exhaustion ✗

Strengthened to proving exhaustion reflection property

For any open set U with an effective open exhaustion, $f^{-1}(U)$ has an effective open exhaustion.

Proof: By induction

- Base cases ✓
- Composition ✓
- Addition and multiplication ✗

Adding decomposition of $+$ and $\cdot$

$(f+g)(x) = f(x) + g(x)$ is composed of

- $Add(x,y) = x + y$,
- $Add(x,y) = x + y$,
- $(f \times g)(x,y) = (f(x), g(y))$,
- $Diag(x) = (x,x)$

Result 2 - Challenges

Theorem 2.1 (Acceptability Theorem: Part 1)

Let $f : \mathbb{R} \to \mathbb{R}$ be an elementary function. Then, $\mathbf{dom}(f)$ has an effective open exhaustion.

First attempt – Strengthening:

Elementary function constructions preserve the property that the domain has an effective open exhaustion.

- Base cases ✓(e.g. $\sin(x)$)
- Addition and multiplication ✓(e.g. $(f+g)(x)$)
- Composition case has a counterexample:

$$f(x) = id|_{(-1,1)} \text{ and } g(x) = \begin{cases} 0 & \text{if } -1 \leq x \leq 1 \\ 1 & \text{otherwise} \end{cases}$$

$\mathbf{dom}(f)$ has an effective open exhaustion ✓
$\mathbf{dom}(g)$ has an effective open exhaustion ✓
$\mathbf{dom}(f \circ g) = [-1, 1]$ has no open exhaustion ✗

Strengthened to proving exhaustion reflection property

For any open set U with an effective open exhaustion, $f^{-1}(U)$ has an effective open exhaustion.

Proof: By induction

- Base cases ✓
- Composition ✓
- Addition and multiplication ✗

Adding decomposition of + and ·

$(f+g)(x) = f(x) + g(x)$ is composed of

- $Add(x, y) = x + y$,
- $Add(x, y) = x + y$,
- $(f \times g)(x, y) = (f(x), g(y))$,
- $Diag(x) = (x, x)$

Result 2 - Challenges

Theorem 2.1 (Acceptability Theorem: Part 1)

Let $f : \mathbb{R} \to \mathbb{R}$ be an elementary function. Then, $\mathbf{dom}(f)$ has an effective open exhaustion.

First attempt – Strengthening:

Elementary function constructions preserve the property that the domain has an effective open exhaustion.

- Base cases ✓(e.g. $\sin(x)$)
- Addition and multiplication ✓(e.g. $(f+g)(x)$)
- Composition case has a counterexample:

$$f(x) = id|_{(-1,1)} \text{ and } g(x) = \begin{cases} 0 & \text{if } -1 \leq x \leq 1 \\ 1 & \text{otherwise} \end{cases}$$

$\mathbf{dom}(f)$ has an effective open exhaustion ✓

$\mathbf{dom}(g)$ has an effective open exhaustion ✓

$\mathbf{dom}(f \circ g) = [-1, 1]$ has no open exhaustion ✗

Strengthened to proving exhaustion reflection property

For any open set U with an effective open exhaustion, $f^{-1}(U)$ has an effective open exhaustion.

Proof: By induction

- Base cases ✓
- Composition ✓
- Addition and multiplication ✗

Adding decomposition of $+$ and $\cdot$

$(f+g)(x) = f(x) + g(x)$ is composed of

- $Add(x, y) = x + y$,
- $Add(x, y) = x + y$,
- $(f \times g)(x, y) = (f(x), g(y))$,
- $Diag(x) = (x, x)$

Result 2 - Challenges

Theorem 2.1 (Acceptability Theorem: Part 1)

Let $f : \mathbb{R} \to \mathbb{R}$ be an elementary function. Then, $\mathbf{dom}(f)$ has an effective open exhaustion.

First attempt – Strengthening:

Elementary function constructions preserve the property that the domain has an effective open exhaustion.

- Base cases ✓(e.g. $\sin(x)$)
- Addition and multiplication ✓(e.g. $(f+g)(x)$)
- Composition case has a counterexample:

$$f(x) = id|_{(-1,1)} \text{ and } g(x) = \begin{cases} 0 & \text{if } -1 \leq x \leq 1 \\ 1 & \text{otherwise} \end{cases}$$

$\mathbf{dom}(f)$ has an effective open exhaustion ✓
$\mathbf{dom}(g)$ has an effective open exhaustion ✓
$\mathbf{dom}(f \circ g) = [-1, 1]$ has no open exhaustion ✗

Strengthened to proving exhaustion reflection property

For any open set U with an effective open exhaustion, $f^{-1}(U)$ has an effective open exhaustion.

Proof: By induction

- Base cases ✓
- Composition ✓
- Addition and multiplication ✗

Adding decomposition of $+$ and $\cdot$

$(f+g)(x) = f(x) + g(x)$ is composed of

- $Add(x, y) = x + y$,
- $Add(x, y) = x + y$,
- $(f \times g)(x, y) = (f(x), g(y))$,
- $Diag(x) = (x, x)$

Result 2 - Challenges

Theorem 2.1 (Acceptability Theorem: Part 1)

Let $f : \mathbb{R} \to \mathbb{R}$ be an elementary function. Then, $\mathbf{dom}(f)$ has an effective open exhaustion.

First attempt – Strengthening:

Elementary function constructions preserve the property that the domain has an effective open exhaustion.

- Base cases ✓(e.g. $\sin(x)$)
- Addition and multiplication ✓(e.g. $(f+g)(x)$)
- Composition case has a counterexample:

$$f(x) = id|_{(-1,1)} \text{ and } g(x) = \begin{cases} 0 & \text{if } -1 \leq x \leq 1 \\ 1 & \text{otherwise} \end{cases}$$

$\mathbf{dom}(f)$ has an effective open exhaustion ✓
$\mathbf{dom}(g)$ has an effective open exhaustion ✓
$\mathbf{dom}(f \circ g) = [-1,1]$ has no open exhaustion ✗

Strengthened to proving exhaustion reflection property

For any open set U with an effective open exhaustion, $f^{-1}(U)$ has an effective open exhaustion.

Proof: By induction

- Base cases ✓
- Composition ✓
- Addition and multiplication ✗

Adding decomposition of $+$ and $\cdot$

$(f+g)(x) = f(x) + g(x)$ is composed of

- $Add(x,y) = x+y$,
- $Add(x,y) = x+y$,
- $(f \times g)(x,y) = (f(x), g(y))$,
- $Diag(x) = (x,x)$

Result 2 - Challenges

Theorem 2.1 (Acceptability Theorem: Part 1)

Let $f : \mathbb{R} \to \mathbb{R}$ be an elementary function. Then, $\mathbf{dom}(f)$ has an effective open exhaustion.

First attempt – Strengthening:

Elementary function constructions preserve the property that the domain has an effective open exhaustion.

- Base cases ✓(e.g. $\sin(x)$)
- Addition and multiplication ✓(e.g. $(f+g)(x)$)
- Composition case has a counterexample:

$$f(x) = id|_{(-1,1)} \text{ and } g(x) = \begin{cases} 0 & \text{if } -1 \leq x \leq 1 \\ 1 & \text{otherwise} \end{cases}$$

$\mathbf{dom}(f)$ has an effective open exhaustion ✓
$\mathbf{dom}(g)$ has an effective open exhaustion ✓
$\mathbf{dom}(f \circ g) = [-1, 1]$ has no open exhaustion ✗

Strengthened to proving exhaustion reflection property

For any open set U with an effective open exhaustion, $f^{-1}(U)$ has an effective open exhaustion.

Proof: By induction

- Base cases ✓
- Composition ✓
- Addition and multiplication ✗

Adding decomposition of $+$ and $\cdot$

$(f+g)(x) = f(x) + g(x)$ is composed of

- $Add(x, y) = x + y$,
- $Add(x, y) = x + y$,
- $(f \times g)(x, y) = (f(x), g(y))$,
- $Diag(x) = (x, x)$

Result 2 - Challenges

Theorem 2.1 (Acceptability Theorem: Part 1)

Let $f : \mathbb{R} \to \mathbb{R}$ be an elementary function. Then, $\mathbf{dom}(f)$ has an effective open exhaustion.

First attempt – Strengthening:

Elementary function constructions preserve the property that the domain has an effective open exhaustion.

- Base cases ✓(e.g. $\sin(x)$)
- Addition and multiplication ✓(e.g. $(f+g)(x)$)
- Composition case has a counterexample:

$$f(x) = id|_{(-1,1)} \text{ and } g(x) = \begin{cases} 0 & \text{if } -1 \leq x \leq 1 \\ 1 & \text{otherwise} \end{cases}$$

$\mathbf{dom}(f)$ has an effective open exhaustion ✓
$\mathbf{dom}(g)$ has an effective open exhaustion ✓
$\mathbf{dom}(f \circ g) = [-1, 1]$ has no open exhaustion ✗

Strengthened to proving exhaustion reflection property

For any open set U with an effective open exhaustion, $f^{-1}(U)$ has an effective open exhaustion.

Proof: By induction

- Base cases ✓
- Composition ✓
- Addition and multiplication ✗

Adding decomposition of $+$ and $\cdot$

$(f+g)(x) = f(x) + g(x)$ is composed of

- $Add(x, y) = x + y$,
- $Add(x, y) = x + y$,
- $(f \times g)(x, y) = (f(x), g(y))$,
- $Diag(x) = (x, x)$

Result 2 - Challenges

Theorem 2.1 (Acceptability Theorem: Part 1)

Let $f : \mathbb{R} \to \mathbb{R}$ be an elementary function. Then, $\mathbf{dom}(f)$ has an effective open exhaustion.

First attempt – Strengthening:

Elementary function constructions preserve the property that the domain has an effective open exhaustion.

- Base cases ✓ (e.g. $\sin(x)$)
- Addition and multiplication ✓ (e.g. $(f+g)(x)$)
- Composition case has a counterexample:

$$f(x) = id|_{(-1,1)} \text{ and } g(x) = \begin{cases} 0 & \text{if } -1 \leq x \leq 1 \\ 1 & \text{otherwise} \end{cases}$$

$\mathbf{dom}(f)$ has an effective open exhaustion ✓
$\mathbf{dom}(g)$ has an effective open exhaustion ✓
$\mathbf{dom}(f \circ g) = [-1, 1]$ has no open exhaustion ✗

Strengthened to proving exhaustion reflection property

For any open set U with an effective open exhaustion, $f^{-1}(U)$ has an effective open exhaustion.

Proof: By induction

- Base cases ✓
- Composition ✓
- Addition and multiplication ✗

Adding decomposition of $+$ and $\cdot$

$(f+g)(x) = f(x) + g(x)$ is composed of

- $Add(x, y) = x + y,$
- $Add(x, y) = x + y,$
- $(f \times g)(x, y) = (f(x), g(y)),$
- $Diag(x) = (x, x)$

Result 2 - Challenges

Theorem 2.1 (Acceptability Theorem: Part 1)

Let $f : \mathbb{R} \to \mathbb{R}$ be an elementary function. Then, $\mathbf{dom}(f)$ has an effective open exhaustion.

First attempt – Strengthening:

Elementary function constructions preserve the property that the domain has an effective open exhaustion.

- Base cases ✓ (e.g. $\sin(x)$)
- Addition and multiplication ✓ (e.g. $(f+g)(x)$)
- Composition case has a counterexample:

$$f(x) = id|_{(-1,1)} \text{ and } g(x) = \begin{cases} 0 & \text{if } -1 \leq x \leq 1 \\ 1 & \text{otherwise} \end{cases}$$

$\mathbf{dom}(f)$ has an effective open exhaustion ✓
$\mathbf{dom}(g)$ has an effective open exhaustion ✓
$\mathbf{dom}(f \circ g) = [-1, 1]$ has no open exhaustion ✗

Strengthened to proving exhaustion reflection property

For any open set U with an effective open exhaustion, $f^{-1}(U)$ has an effective open exhaustion.

Proof: By induction

- Base cases ✓
- Composition ✓
- Addition and multiplication ✗

Adding decomposition of $+$ and $\cdot$

$(f+g)(x) = f(x) + g(x)$ is composed of

- $Add(x, y) = x + y$,
- $Add(x, y) = x + y$,
- $(f \times g)(x, y) = (f(x), g(y))$,
- $Diag(x) = (x, x)$

Result 2 - Challenges

Theorem 2.1 (Acceptability Theorem: Part 1)

Let $f : \mathbb{R} \to \mathbb{R}$ be an elementary function. Then, $\mathbf{dom}(f)$ has an effective open exhaustion.

First attempt – Strengthening:

Elementary function constructions preserve the property that the domain has an effective open exhaustion.

- Base cases ✓ (e.g. $\sin(x)$)
- Addition and multiplication ✓ (e.g. $(f+g)(x)$)
- Composition case has a counterexample:

$$f(x) = id|_{(-1,1)} \text{ and } g(x) = \begin{cases} 0 & \text{if } -1 \leq x \leq 1 \\ 1 & \text{otherwise} \end{cases}$$

$\mathbf{dom}(f)$ has an effective open exhaustion ✓
$\mathbf{dom}(g)$ has an effective open exhaustion ✓
$\mathbf{dom}(f \circ g) = [-1, 1]$ has no open exhaustion ✗

Strengthened to proving exhaustion reflection property

For any open set U with an effective open exhaustion, $f^{-1}(U)$ has an effective open exhaustion.

Proof: By induction

- Base cases ✓
- Composition ✓
- Addition and multiplication ✗

Adding decomposition of $+$ and $\cdot$

$(f+g)(x) = f(x) + g(x)$ is composed of

- $Add(x, y) = x + y$,
- $Add(x, y) = x + y$,
- $(f \times g)(x, y) = (f(x), g(y))$,
- $Diag(x) = (x, x)$

Result 2 - Challenges

Theorem 2.1 (Acceptability Theorem: Part 1)

Let $f : \mathbb{R} \to \mathbb{R}$ be an elementary function. Then, $\mathbf{dom}(f)$ has an effective open exhaustion.

First attempt – Strengthening:

Elementary function constructions preserve the property that the domain has an effective open exhaustion.

- Base cases ✓(e.g. $\sin(x)$)
- Addition and multiplication ✓(e.g. $(f+g)(x)$)
- Composition case has a counterexample:

$$f(x) = id|_{(-1,1)} \text{ and } g(x) = \begin{cases} 0 & \text{if } -1 \leq x \leq 1 \\ 1 & \text{otherwise} \end{cases}$$

$\mathbf{dom}(f)$ has an effective open exhaustion ✓
$\mathbf{dom}(g)$ has an effective open exhaustion ✓
$\mathbf{dom}(f \circ g) = [-1, 1]$ has no open exhaustion ✗

Strengthened to proving exhaustion reflection property

For any open set U with an effective open exhaustion, $f^{-1}(U)$ has an effective open exhaustion.

Proof: By induction

- Base cases ✓
- Composition ✓
- Addition and multiplication ✗

Adding decomposition of $+$ and $\cdot$

$(f+g)(x) = f(x) + g(x)$ is composed of

- $Add(x, y) = x + y$,
- $Add(x, y) = x + y$,
- $(f \times g)(x, y) = (f(x), g(y))$,
- $Diag(x) = (x, x)$

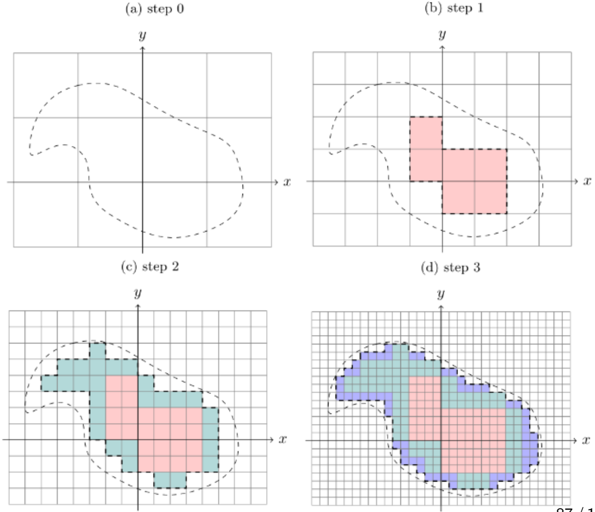
Interesting results - Acceptability Theorem

Theorem (Reducing Exhaustion-reflection to to a Decision Procedure)

A function $f : \mathbb{R}^n \to \mathbb{R}^m$ is exhaustion-reflecting, if we can decide for any m-cube Q_m whether an arbitrary rational closed n-cube is completely contained in $f^{-1}(Q_m)$.

- This is very useful for proving the exhaustion-reflection property for the addition and multiplication case.

The process of building an effective open exhaustion



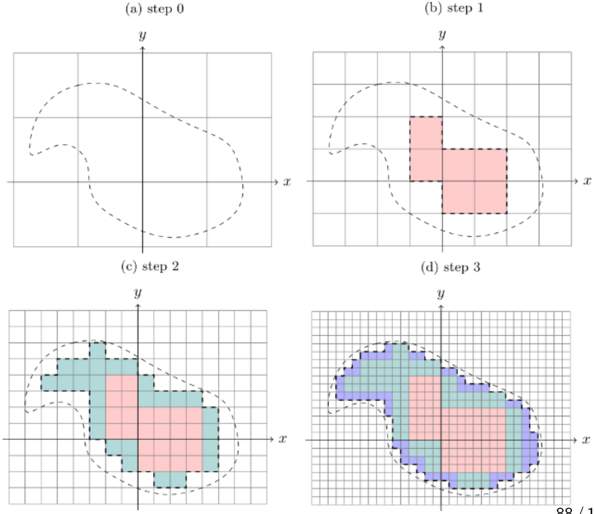
Interesting results - Acceptability Theorem

Theorem (Reducing Exhaustion-reflection to to a Decision Procedure)

A function $f : \mathbb{R}^n \to \mathbb{R}^m$ is exhaustion-reflecting, if we can decide for any m-cube Q_m whether an arbitrary rational closed n-cube is completely contained in $f^{-1}(Q_m)$.

- This is very useful for proving the exhaustion-reflection property for the addition and multiplication case.

The process of building an effective open exhaustion



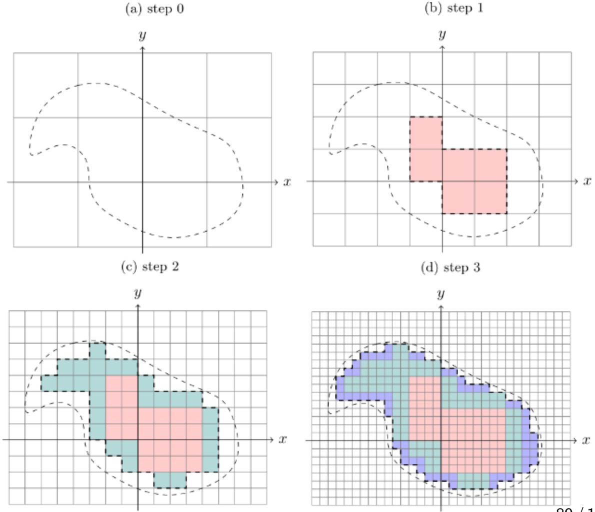
Interesting results - Acceptability Theorem

Theorem (Reducing Exhaustion-reflection to to a Decision Procedure)

A function $f : \mathbb{R}^n \to \mathbb{R}^m$ is exhaustion-reflecting, if we can decide for any m-cube Q_m whether an arbitrary rational closed n-cube is completely contained in $f^{-1}(Q_m)$.

- This is very useful for proving the exhaustion-reflection property for the addition and multiplication case.

The process of building an effective open exhaustion



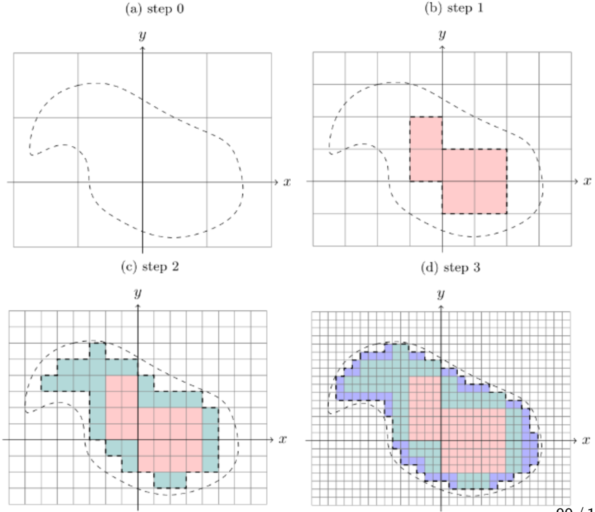
Interesting results - Acceptability Theorem

Theorem (Reducing Exhaustion-reflection to to a Decision Procedure)

A function $f : \mathbb{R}^n \to \mathbb{R}^m$ is exhaustion-reflecting, if we can decide for any m-cube Q_m whether an arbitrary rational closed n-cube is completely contained in $f^{-1}(Q_m)$.

- This is very useful for proving the exhaustion-reflection property for the addition and multiplication case.

The process of building an effective open exhaustion



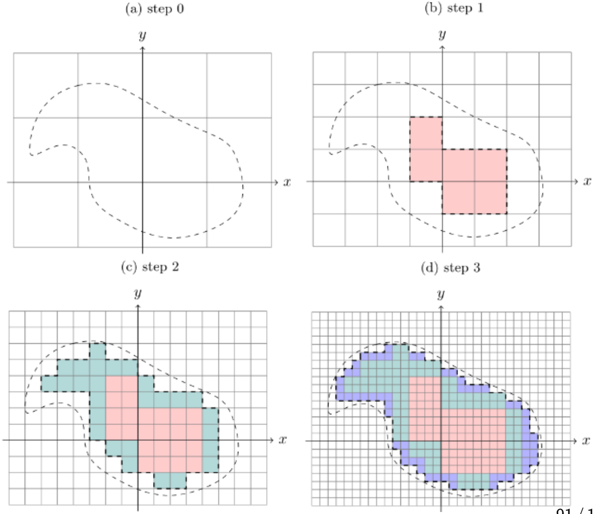
Interesting results - Acceptability Theorem

Theorem (Reducing Exhaustion-reflection to to a Decision Procedure)

A function $f : \mathbb{R}^n \to \mathbb{R}^m$ is exhaustion-reflecting, if we can decide for any m-cube Q_m whether an arbitrary rational closed n-cube is completely contained in $f^{-1}(Q_m)$.

- This is very useful for proving the exhaustion-reflection property for the addition and multiplication case.

The process of building an effective open exhaustion



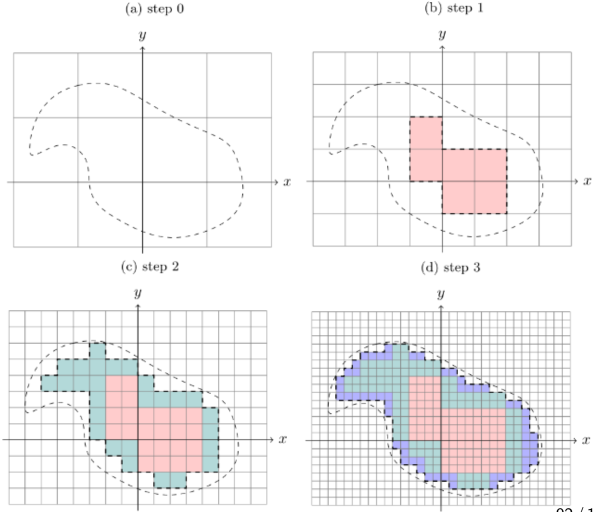
Interesting results - Acceptability Theorem

Theorem (Reducing Exhaustion-reflection to to a Decision Procedure)

A function $f : \mathbb{R}^n \to \mathbb{R}^m$ is exhaustion-reflecting, if we can decide for any m-cube Q_m whether an arbitrary rational closed n-cube is completely contained in $f^{-1}(Q_m)$.

- This is very useful for proving the exhaustion-reflection property for the addition and multiplication case.

The process of building an effective open exhaustion



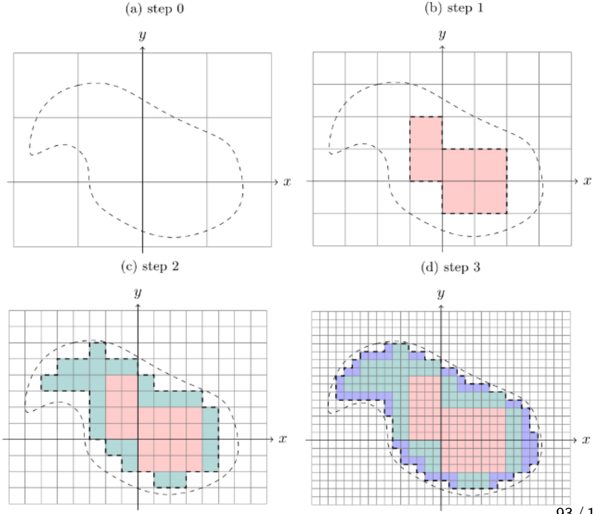
Interesting results - Acceptability Theorem

Theorem (Reducing Exhaustion-reflection to to a Decision Procedure)

A function $f : \mathbb{R}^n \to \mathbb{R}^m$ is exhaustion-reflecting, if we can decide for any m-cube Q_m whether an arbitrary rational closed n-cube is completely contained in $f^{-1}(Q_m)$.

- This is very useful for proving the exhaustion-reflection property for the addition and multiplication case.

The process of building an effective open exhaustion



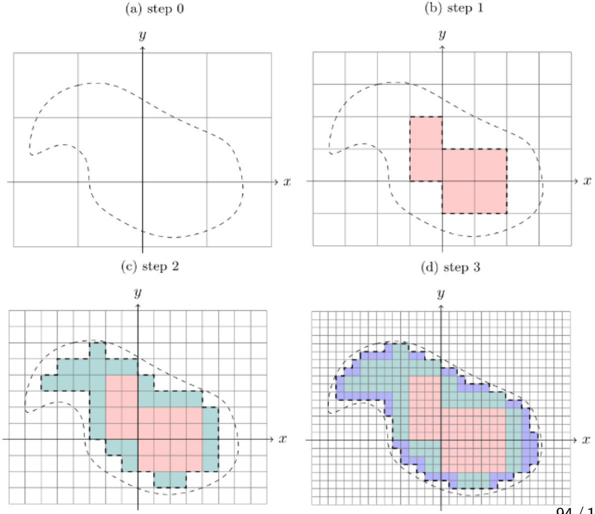
Interesting results - Acceptability Theorem

Theorem (Reducing Exhaustion-reflection to to a Decision Procedure)

A function $f : \mathbb{R}^n \to \mathbb{R}^m$ is exhaustion-reflecting, if we can decide for any m-cube Q_m whether an arbitrary rational closed n-cube is completely contained in $f^{-1}(Q_m)$.

- This is very useful for proving the exhaustion-reflection property for the addition and multiplication case.

The process of building an effective open exhaustion



Challenges

Theorem 2.2 (Acceptability Theorem: Part 2)

Any elementary function is effectively locally uniformly continuous **w.r.t. an effective open exhaustion** for its domain.

(We prove that this is) equivalent to proving:

Any elementary function has a local continuity witness.

Definition (Local continuity witness)

Let $f : \mathbb{R} \to \mathbb{R}$. A recursive function $N : \mathbb{Q} \times \mathbb{Q} \times \mathbb{N} \to \mathbb{N}$ is called a **local continuity witness** for f iff for any $a, b \in \mathbb{Q}$ with $[a, b] \subseteq \mathbf{dom}(f)$ and $k \in \mathbb{N}$, we have

$$\forall x, y \in (a, b) \quad |x - y| < 2^{-N(a,b,k)} \implies |f(x) - f(y)| < 2^{-k}.$$

Proof by induction

- Base cases: Using **WhileCC**-approximability theorem ✓
- Addition, Multiplication, and Composition: Using **WhileCC**-approximability theorem ✓

Challenges

Theorem 2.2 (Acceptability Theorem: Part 2)

Any elementary function is effectively locally uniformly continuous **w.r.t. an effective open exhaustion** for its domain.

(We prove that this is) equivalent to proving:

Any elementary function has a local continuity witness.

Definition (Local continuity witness)

Let $f : \mathbb{R} \to \mathbb{R}$. A recursive function $N : \mathbb{Q} \times \mathbb{Q} \times \mathbb{N} \to \mathbb{N}$ is called a **local continuity witness** for f iff for any $a, b \in \mathbb{Q}$ with $[a, b] \subseteq \mathbf{dom}(f)$ and $k \in \mathbb{N}$, we have

$$\forall x, y \in (a, b) \quad |x - y| < 2^{-N(a,b,k)} \implies |f(x) - f(y)| < 2^{-k}.$$

Proof by induction

- Base cases: Using **WhileCC**-approximability theorem ✓
- Addition, Multiplication, and Composition: Using **WhileCC**-approximability theorem ✓

Challenges

Theorem 2.2 (Acceptability Theorem: Part 2)

Any elementary function is effectively locally uniformly continuous **w.r.t. an effective open exhaustion** for its domain.

(We prove that this is) equivalent to proving:

Any elementary function has a local continuity witness.

Definition (Local continuity witness)

Let $f : \mathbb{R} \to \mathbb{R}$. A recursive function $N : \mathbb{Q} \times \mathbb{Q} \times \mathbb{N} \to \mathbb{N}$ is called a **local continuity witness** for f iff for any $a, b \in \mathbb{Q}$ with $[a, b] \subseteq \mathbf{dom}(f)$ and $k \in \mathbb{N}$, we have

$$\forall x, y \in (a, b) \quad |x - y| < 2^{-N(a,b,k)} \implies |f(x) - f(y)| < 2^{-k}.$$

Proof by induction

- Base cases: Using **WhileCC**-approximability theorem ✓
- Addition, Multiplication, and Composition: Using **WhileCC**-approximability theorem ✓

Challenges

Theorem 2.2 (Acceptability Theorem: Part 2)

Any elementary function is effectively locally uniformly continuous **w.r.t. an effective open exhaustion** for its domain.

(We prove that this is) equivalent to proving:

Any elementary function has a local continuity witness.

Definition (Local continuity witness)

Let $f : \mathbb{R} \to \mathbb{R}$. A recursive function $N : \mathbb{Q} \times \mathbb{Q} \times \mathbb{N} \to \mathbb{N}$ is called a **local continuity witness** for f iff for any $a, b \in \mathbb{Q}$ with $[a, b] \subseteq \mathbf{dom}(f)$ and $k \in \mathbb{N}$, we have

$$\forall x, y \in (a, b) \quad |x - y| < 2^{-N(a,b,k)} \implies |f(x) - f(y)| < 2^{-k}.$$

Proof by induction

- Base cases: Using **WhileCC**-approximability theorem ✓
- Addition, Multiplication, and Composition: Using **WhileCC**-approximability theorem ✓

Challenges

Theorem 2.2 (Acceptability Theorem: Part 2)

Any elementary function is effectively locally uniformly continuous **w.r.t. an effective open exhaustion** for its domain.

(We prove that this is) equivalent to proving:

Any elementary function has a local continuity witness.

Definition (Local continuity witness)

Let $f : \mathbb{R} \to \mathbb{R}$. A recursive function $N : \mathbb{Q} \times \mathbb{Q} \times \mathbb{N} \to \mathbb{N}$ is called a **local continuity witness** for f iff for any $a, b \in \mathbb{Q}$ with $[a, b] \subseteq \mathbf{dom}(f)$ and $k \in \mathbb{N}$, we have

$$\forall x, y \in (a, b) \quad |x - y| < 2^{-N(a,b,k)} \implies |f(x) - f(y)| < 2^{-k}.$$

Proof by induction

- Base cases: Using **WhileCC**-approximability theorem ✓
- Addition, Multiplication, and Composition: Using **WhileCC**-approximability theorem ✓

Summary

We proved that:

- all elementary functions are **WhileCC**-approximable.
- all elementary functions are acceptable and hence computable in the other three models as well.

We presented an **alternative characterization** of acceptable functions using the **local continuity witness** concept. We also found a few useful tricks along the way for implementing approximations of piecewise functions.

Summary

We proved that:

- all elementary functions are **WhileCC**-approximable.
- all elementary functions are acceptable and hence computable in the other three models as well.

We presented an **alternative characterization** of acceptable functions using the **local continuity witness** concept. We also found a few useful tricks along the way for implementing approximations of piecewise functions.

Summary

We proved that:

- all elementary functions are **WhileCC**-approximable.
- all elementary functions are acceptable and hence computable in the other three models as well.

We presented an **alternative characterization** of acceptable functions using the **local continuity witness** concept. We also found a few useful tricks along the way for implementing approximations of piecewise functions.

Summary

We proved that:

- all elementary functions are **WhileCC**-approximable.
- all elementary functions are acceptable and hence computable in the other three models as well.

We presented an **alternative characterization** of acceptable functions using the **local continuity witness** concept. We also found a few useful tricks along the way for implementing approximations of piecewise functions.

Summary

We proved that:

- all elementary functions are **WhileCC**-approximable.
- all elementary functions are acceptable and hence computable in the other three models as well.

We presented an **alternative characterization** of acceptable functions using the **local continuity witness** concept. We also found a few useful tricks along the way for implementing approximations of piecewise functions.

Future Work

Questions left unanswered:

- Are non-unary elementary functions acceptable?
  A generalization of acceptability in arbitrary metric spaces is given by Tucker and Zucker [2004].
- Can we extend the equivalence theorem in Fu and Zucker [2014] to acceptable partial functions of type $\mathbb{R}^m \to \mathbb{R}$?
- What functions are **WhileCC**-approximable but not While*-approximable [Tucker and Zucker, 1999]?

Conjecture:

- All partial unary **WhileCC**-approximable functions are acceptable.
  - **If the conjecture holds**, are *non-unary* **WhileCC**-approximable functions acceptable?
  - **If not**, what is a model of computation that characterizes exactly the class of acceptable functions?

Currently working on:

- Formalizing the concept of **WhileCC**-approximability and the aforementioned proofs in Lean.

Future Work

Questions left unanswered:

- Are non-unary elementary functions acceptable?
  A generalization of acceptability in arbitrary metric spaces is given by Tucker and Zucker [2004].
- Can we extend the equivalence theorem in Fu and Zucker [2014] to acceptable partial functions of type $\mathbb{R}^m \to \mathbb{R}$?
- What functions are **WhileCC**-approximable but not While*-approximable [Tucker and Zucker, 1999]?

Conjecture:

- All partial unary **WhileCC**-approximable functions are acceptable.
  - **If the conjecture holds**, are *non-unary* **WhileCC**-approximable functions acceptable?
  - **If not**, what is a model of computation that characterizes exactly the class of acceptable functions?

Currently working on:

- Formalizing the concept of **WhileCC**-approximability and the aforementioned proofs in Lean.

Future Work

Questions left unanswered:

- Are non-unary elementary functions acceptable?
  A generalization of acceptability in arbitrary metric spaces is given by Tucker and Zucker [2004].
- Can we extend the equivalence theorem in Fu and Zucker [2014] to acceptable partial functions of type $\mathbb{R}^m \to \mathbb{R}$?
- What functions are **WhileCC**-approximable but not While*-approximable [Tucker and Zucker, 1999]?

Conjecture:

- All partial unary **WhileCC**-approximable functions are acceptable.
  - **If the conjecture holds**, are *non-unary* **WhileCC**-approximable functions acceptable?
  - **If not**, what is a model of computation that characterizes exactly the class of acceptable functions?

Currently working on:

- Formalizing the concept of **WhileCC**-approximability and the aforementioned proofs in Lean.

Future Work

Questions left unanswered:

- Are non-unary elementary functions acceptable?
  A generalization of acceptability in arbitrary metric spaces is given by Tucker and Zucker [2004].
- Can we extend the equivalence theorem in Fu and Zucker [2014] to acceptable partial functions of type $\mathbb{R}^m \to \mathbb{R}$?
- What functions are **WhileCC**-approximable but not While*-approximable [Tucker and Zucker, 1999]?

Conjecture:

- All partial unary **WhileCC**-approximable functions are acceptable.
  - **If the conjecture holds**, are *non-unary* **WhileCC**-approximable functions acceptable?
  - **If not**, what is a model of computation that characterizes exactly the class of acceptable functions?

Currently working on:

- Formalizing the concept of **WhileCC**-approximability and the aforementioned proofs in Lean.

Future Work

Questions left unanswered:

- Are non-unary elementary functions acceptable?
  A generalization of acceptability in arbitrary metric spaces is given by Tucker and Zucker [2004].
- Can we extend the equivalence theorem in Fu and Zucker [2014] to acceptable partial functions of type $\mathbb{R}^m \to \mathbb{R}$?
- What functions are **WhileCC**-approximable but not While*-approximable [Tucker and Zucker, 1999]?

Conjecture:

- All partial unary **WhileCC**-approximable functions are acceptable.
  - **If the conjecture holds**, are *non-unary* **WhileCC**-approximable functions acceptable?
  - **If not**, what is a model of computation that characterizes exactly the class of acceptable functions?

Currently working on:

- Formalizing the concept of **WhileCC**-approximability and the aforementioned proofs in Lean.

Future Work

Questions left unanswered:

- Are non-unary elementary functions acceptable?
  A generalization of acceptability in arbitrary metric spaces is given by Tucker and Zucker [2004].
- Can we extend the equivalence theorem in Fu and Zucker [2014] to acceptable partial functions of type $\mathbb{R}^m \to \mathbb{R}$?
- What functions are **WhileCC**-approximable but not While*-approximable [Tucker and Zucker, 1999]?

Conjecture:

- All partial unary **WhileCC**-approximable functions are acceptable.
  - **If the conjecture holds**, are *non-unary* **WhileCC**-approximable functions acceptable?
  - **If not**, what is a model of computation that characterizes exactly the class of acceptable functions?

Currently working on:

- Formalizing the concept of **WhileCC**-approximability and the aforementioned proofs in Lean.

Future Work

Questions left unanswered:

- Are non-unary elementary functions acceptable?
  A generalization of acceptability in arbitrary metric spaces is given by Tucker and Zucker [2004].
- Can we extend the equivalence theorem in Fu and Zucker [2014] to acceptable partial functions of type $\mathbb{R}^m \to \mathbb{R}$?
- What functions are **WhileCC**-approximable but not While*-approximable [Tucker and Zucker, 1999]?

Conjecture:

- All partial unary **WhileCC**-approximable functions are acceptable.
  - **If the conjecture holds**, are *non-unary* **WhileCC**-approximable functions acceptable?
  - **If not**, what is a model of computation that characterizes exactly the class of acceptable functions?

Currently working on:

- Formalizing the concept of **WhileCC**-approximability and the aforementioned proofs in Lean.

Future Work

Questions left unanswered:

- Are non-unary elementary functions acceptable?
  A generalization of acceptability in arbitrary metric spaces is given by Tucker and Zucker [2004].
- Can we extend the equivalence theorem in Fu and Zucker [2014] to acceptable partial functions of type $\mathbb{R}^m \to \mathbb{R}$?
- What functions are **WhileCC**-approximable but not While*-approximable [Tucker and Zucker, 1999]?

Conjecture:

- All partial unary **WhileCC**-approximable functions are acceptable.
  - **If the conjecture holds**, are *non-unary* **WhileCC**-approximable functions acceptable?
  - **If not**, what is a model of computation that characterizes exactly the class of acceptable functions?

Currently working on:

- Formalizing the concept of **WhileCC**-approximability and the aforementioned proofs in Lean.

Future Work

Questions left unanswered:

- Are non-unary elementary functions acceptable?
  A generalization of acceptability in arbitrary metric spaces is given by Tucker and Zucker [2004].
- Can we extend the equivalence theorem in Fu and Zucker [2014] to acceptable partial functions of type $\mathbb{R}^m \to \mathbb{R}$?
- What functions are **WhileCC**-approximable but not While*-approximable [Tucker and Zucker, 1999]?

Conjecture:

- All partial unary **WhileCC**-approximable functions are acceptable.
  - **If the conjecture holds**, are *non-unary* **WhileCC**-approximable functions acceptable?
  - **If not**, what is a model of computation that characterizes exactly the class of acceptable functions?

Currently working on:

- Formalizing the concept of **WhileCC**-approximability and the aforementioned proofs in Lean.

Future Work

Questions left unanswered:

- Are non-unary elementary functions acceptable?
  A generalization of acceptability in arbitrary metric spaces is given by Tucker and Zucker [2004].
- Can we extend the equivalence theorem in Fu and Zucker [2014] to acceptable partial functions of type $\mathbb{R}^m \to \mathbb{R}$?
- What functions are **WhileCC**-approximable but not While*-approximable [Tucker and Zucker, 1999]?

Conjecture:

- All partial unary **WhileCC**-approximable functions are acceptable.
  - **If the conjecture holds**, are *non-unary* **WhileCC**-approximable functions acceptable?
  - **If not**, what is a model of computation that characterizes exactly the class of acceptable functions?

Currently working on:

- Formalizing the concept of **WhileCC**-approximability and the aforementioned proofs in Lean.

Future Work

Questions left unanswered:

- Are non-unary elementary functions acceptable?
  A generalization of acceptability in arbitrary metric spaces is given by Tucker and Zucker [2004].
- Can we extend the equivalence theorem in Fu and Zucker [2014] to acceptable partial functions of type $\mathbb{R}^m \to \mathbb{R}$?
- What functions are **WhileCC**-approximable but not While*-approximable [Tucker and Zucker, 1999]?

Conjecture:

- All partial unary **WhileCC**-approximable functions are acceptable.
  - **If the conjecture holds**, are *non-unary* **WhileCC**-approximable functions acceptable?
  - **If not**, what is a model of computation that characterizes exactly the class of acceptable functions?

Currently working on:

- Formalizing the concept of **WhileCC**-approximability and the aforementioned proofs in Lean.

References

Ming Quan Fu and Jeffery Zucker. Models of computation for partial functions on the reals. *Journal of Logical and Algebraic Methods in Programming*, 84(2):218–237, 11 2014. ISSN 2352-2208. doi:10.1016/j.jlamp.2014.11.001.

M. Tenenbaum and H. Pollard. *Ordinary Differential Equations: An Elementary Textbook for Students of Mathematics, Engineering, and the Sciences*. Dover Books on Mathematics. Dover Publications, 1985. ISBN 9780486649405. URL `https://books.google.ca/books?id=iU4zDAAAQBAJ`.

John V. Tucker and Jeffery I. Zucker. Abstract versus concrete computation on metric partial algebras. *ACM Trans. Comput. Log.*, 5(4):611–668, 2004. doi:10.1145/1024922.1024924.

J.V. Tucker and J.I. Zucker. Computation by 'While' programs on topological partial algebras. *Theoretical Computer Science*, 219(1):379–420, 1999. ISSN 0304-3975. doi:10.1016/S0304-3975(98)00297-7.

J.V. Tucker and J.I. Zucker. Computable total functions on metric algebras, universal algebraic specifications and dynamical systems. *The Journal of Logic and Algebraic Programming*, 62(1): 71–108, 2005. ISSN 1567-8326. doi:10.1016/j.jlap.2003.10.001.

A HUGE Thank you!