

WhileCC-approximability and Acceptability of Elementary Functions

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

September 2025



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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Background – Acceptability

Definition (**Acceptability**)

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

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

Background – Acceptability

Definition (**Acceptability**)

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

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

Background – Acceptability

Definition (**Acceptability**)

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

- 1 X is an **effective open exhaustion** for $\text{dom}(f)$, 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, \dots) 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 $I_1^l, I_2^l, \dots, I_{k_l}^l$ 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 $i = 1, \dots, k_l - 1$, and
- 5 the map $l \mapsto (a_1^l, b_1^l, \dots, a_{k_l}^l, b_{k_l}^l)$ which delivers the sequence of stages $U_l = I_1^l \cup \dots \cup I_{k_l}^l$ is recursive.

Example

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

Background – Effective Open Exhaustions

Definition ([Fu and Zucker, 2014])

A sequence (U_1, U_2, \dots) 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 $I_1^l, I_2^l, \dots, I_{k_l}^l$ 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 $i = 1, \dots, k_l - 1$, and
- 5 the map $l \mapsto (a_1^l, b_1^l, \dots, a_{k_l}^l, b_{k_l}^l)$ which delivers the sequence of stages $U_l = I_1^l \cup \dots \cup I_{k_l}^l$ is recursive.

Example

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

Background – Effective Open Exhaustions

Definition ([Fu and Zucker, 2014])

A sequence (U_1, U_2, \dots) 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 $I_1^l, I_2^l, \dots, I_{k_l}^l$ 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 $i = 1, \dots, k_l - 1$, and
- 5 the map $l \mapsto (a_1^l, b_1^l, \dots, a_{k_l}^l, b_{k_l}^l)$ which delivers the sequence of stages $U_l = I_1^l \cup \dots \cup I_{k_l}^l$ is recursive.

Example

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

Background – Effective Open Exhaustions

Definition ([Fu and Zucker, 2014])

A sequence (U_1, U_2, \dots) 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 $I_1^l, I_2^l, \dots, I_{k_l}^l$ 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 $i = 1, \dots, k_l - 1$, and
- 5 the map $l \mapsto (a_1^l, b_1^l, \dots, a_{k_l}^l, b_{k_l}^l)$ which delivers the sequence of stages $U_l = I_1^l \cup \dots \cup I_{k_l}^l$ is recursive.

Example

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

Background – Effective Open Exhaustions

Definition ([Fu and Zucker, 2014])

A sequence (U_1, U_2, \dots) 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 $I_1^l, I_2^l, \dots, I_{k_l}^l$ 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 $i = 1, \dots, k_l - 1$, and
- 5 the map $l \mapsto (a_1^l, b_1^l, \dots, a_{k_l}^l, b_{k_l}^l)$ which delivers the sequence of stages $U_l = I_1^l \cup \dots \cup I_{k_l}^l$ is recursive.

Example

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

Background – Effective Open Exhaustions

Definition ([Fu and Zucker, 2014])

A sequence (U_1, U_2, \dots) 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 $I_1^l, I_2^l, \dots, I_{k_l}^l$ 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 $i = 1, \dots, k_l - 1$, and
- 5 the map $l \mapsto (a_1^l, b_1^l, \dots, a_{k_l}^l, b_{k_l}^l)$ which delivers the sequence of stages $U_l = I_1^l \cup \dots \cup I_{k_l}^l$ is recursive.

Example

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

Background – Effective Open Exhaustions

Definition ([Fu and Zucker, 2014])

A sequence (U_1, U_2, \dots) 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 $I_1^l, I_2^l, \dots, I_{k_l}^l$ 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 $i = 1, \dots, k_l - 1$, and
- 5 the map $l \mapsto (a_1^l, b_1^l, \dots, a_{k_l}^l, b_{k_l}^l)$ which delivers the sequence of stages $U_l = I_1^l \cup \dots \cup I_{k_l}^l$ is recursive.

Example

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

Background – Effective Open Exhaustions

Definition ([Fu and Zucker, 2014])

A sequence (U_1, U_2, \dots) 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 $I_1^l, I_2^l, \dots, I_{k_l}^l$ 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 $i = 1, \dots, k_l - 1$, and
- 5 the map $l \mapsto (a_1^l, b_1^l, \dots, a_{k_l}^l, b_{k_l}^l)$ which delivers the sequence of stages $U_l = I_1^l \cup \dots \cup I_{k_l}^l$ is recursive.

Example

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

Background – Effective Open Exhaustions

Definition ([Fu and Zucker, 2014])

A sequence (U_1, U_2, \dots) 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 $I_1^l, I_2^l, \dots, I_{k_l}^l$ 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 $i = 1, \dots, k_l - 1$, and
- 5 the map $l \mapsto (a_1^l, b_1^l, \dots, a_{k_l}^l, b_{k_l}^l)$ which delivers the sequence of stages $U_l = I_1^l \cup \dots \cup I_{k_l}^l$ is recursive.

Example

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

Background – Effective Open Exhaustions

Definition ([Fu and Zucker, 2014])

A sequence (U_1, U_2, \dots) 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 $I_1^l, I_2^l, \dots, I_{k_l}^l$ 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 $i = 1, \dots, k_l - 1$, and
- 5 the map $l \mapsto (a_1^l, b_1^l, \dots, a_{k_l}^l, b_{k_l}^l)$ which delivers the sequence of stages $U_l = I_1^l \cup \dots \cup I_{k_l}^l$ is recursive.

Example

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

Background – Effective Open Exhaustions

Definition ([Fu and Zucker, 2014])

A sequence (U_1, U_2, \dots) 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 $I_1^l, I_2^l, \dots, I_{k_l}^l$ 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 $i = 1, \dots, k_l - 1$, and
- 5 the map $l \mapsto (a_1^l, b_1^l, \dots, a_{k_l}^l, b_{k_l}^l)$ which delivers the sequence of stages $U_l = I_1^l \cup \dots \cup I_{k_l}^l$ is recursive.

Example

The sequence of open sets $(-1, 1), (-2, 2), \dots, (-k, k), \dots$ 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 $M : \mathbb{N}^2 \rightarrow \mathbb{N}$ 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 $M : \mathbb{N}^2 \rightarrow \mathbb{N}$ 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 :

- $(f + g)(x) = f(x) + g(x)$
- $(f \cdot g)(x) = f(x)g(x)$
- $\text{div}_f(x) = \frac{1}{f(x)}$ where $\frac{1}{0} = \uparrow$
- $\text{root}_{n,f}(x) = \sqrt[n]{f(x)}$ 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 :

- $(f + g)(x) = f(x) + g(x)$
- $(f \cdot g)(x) = f(x)g(x)$
- $\text{div}_f(x) = \frac{1}{f(x)}$ where $\frac{1}{0} = \uparrow$
- $\text{root}_{n,f}(x) = \sqrt[n]{f(x)}$ 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 :

- $(f + g)(x) = f(x) + g(x)$
- $(f \cdot g)(x) = f(x)g(x)$
- $\text{div}_f(x) = \frac{1}{f(x)}$ where $\frac{1}{0} = \uparrow$
- $\text{root}_{n,f}(x) = \sqrt[n]{f(x)}$ 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 :

- $(f + g)(x) = f(x) + g(x)$
- $(f \cdot g)(x) = f(x)g(x)$
- $\text{div}_f(x) = \frac{1}{f(x)}$ where $\frac{1}{0} = \uparrow$
- $\text{root}_{n,f}(x) = \sqrt[n]{f(x)}$ 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 :

- $(f + g)(x) = f(x) + g(x)$
- $(f \cdot g)(x) = f(x)g(x)$
- $\text{div}_f(x) = \frac{1}{f(x)}$ where $\frac{1}{0} = \uparrow$
- $\text{root}_{n,f}(x) = \sqrt[n]{f(x)}$ 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 :

- $(f + g)(x) = f(x) + g(x)$
- $(f \cdot g)(x) = f(x)g(x)$
- $\text{div}_f(x) = \frac{1}{f(x)}$ where $\frac{1}{0} = \uparrow$
- $\text{root}_{n,f}(x) = \sqrt[n]{f(x)}$ 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 :

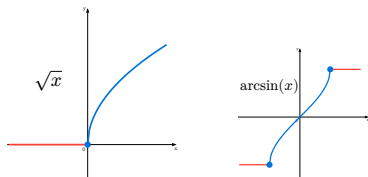
- $(f + g)(x) = f(x) + g(x)$
- $(f \cdot g)(x) = f(x)g(x)$
- $\text{div}_f(x) = \frac{1}{f(x)}$ where $\frac{1}{0} = \uparrow$
- $\text{root}_{n,f}(x) = \sqrt[n]{f(x)}$ where $0 < n \in \mathbb{N}$
- $\ln_f(x) = \ln(f(x))$
- $\exp_f(x) = e^{f(x)}$
- $\sin_f(x) = \sin(f(x))$
- $\arcsin_f(x) = \arcsin(f(x))$

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 :

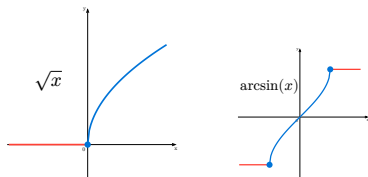
- $(f + g)(x) = f(x) + g(x)$
- $(f \cdot g)(x) = f(x)g(x)$
- $\text{div}_f(x) = \frac{1}{f(x)}$ where $\frac{1}{0} = \uparrow$
- $\text{root}_{n,f}(x) = \sqrt[n]{f(x)}$ where $0 < n \in \mathbb{N}$
- $\ln_f(x) = \ln(f(x))$
- $\exp_f(x) = e^{f(x)}$
- $\sin_f(x) = \sin(f(x))$
- $\arcsin_f(x) = \arcsin(f(x))$

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 :

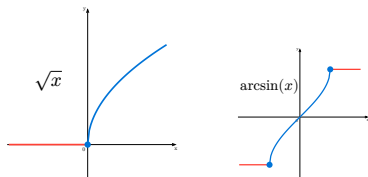
- $(f + g)(x) = f(x) + g(x)$
- $(f \cdot g)(x) = f(x)g(x)$
- $\text{div}_f(x) = \frac{1}{f(x)}$ where $\frac{1}{0} = \uparrow$
- $\text{root}_{n,f}(x) = \sqrt[n]{f(x)}$ where $0 < n \in \mathbb{N}$
- $\ln_f(x) = \ln(f(x))$
- $\exp_f(x) = e^{f(x)}$
- $\sin_f(x) = \sin(f(x))$
- $\arcsin_f(x) = \arcsin(f(x))$

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** $f : \mathbb{R} \rightarrow \mathbb{R}$ and any effective open exhaustion X for $\text{dom}(f)$, 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** $f : \mathbb{R} \rightarrow \mathbb{R}$ and any effective open exhaustion X for $\text{dom}(f)$, 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** $f : \mathbb{R} \rightarrow \mathbb{R}$ and any effective open exhaustion X for $\text{dom}(f)$, 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** $f : \mathbb{R} \rightarrow \mathbb{R}$ and any effective open exhaustion X for $\text{dom}(f)$, 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 $f : \mathbb{R} \rightarrow \mathbb{R}$ and any effective open exhaustion X for $\text{dom}(f)$, 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 $f : \mathbb{R} \rightarrow \mathbb{R}$ and any effective open exhaustion X for $\text{dom}(f)$, 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

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

Statements

$S ::=$

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

| if b then S_1 else S_2 fi

| while b do S_0 od

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

Procedures

$P ::= \text{proc } D \text{ begin } S \text{ end}$

Algebra \mathcal{R}

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

Semantics

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

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

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

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

Statement	Possible Values for n
$n := \text{choose} (k : \text{nat}) : k < 0$	$\{\uparrow\}$
$n := \text{choose} (k : \text{nat}) : \text{toReal}(k) = 0$	$\{\uparrow\}$
$n := \text{choose} (k : \text{nat}) : k < k + 1$	$\{0, 1, 2, \dots\}$
$n := \text{choose} (k : \text{nat}) : k > 2 \text{ and } k < 4$	$\{3\}$

Result 1 - Background - WhileCC Programming Language

Syntax

Terms

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

Statements

$S ::=$

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

| if b then S_1 else S_2 fi

| while b do S_0 od

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

Procedures

$P ::= \text{proc } D \text{ begin } S \text{ end}$

Algebra \mathcal{R}

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

Semantics

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

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

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

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

Statement	Possible Values for n
$n := \text{choose } (k : \text{nat}) : k < 0$	$\{\uparrow\}$
$n := \text{choose } (k : \text{nat}) : \text{toReal}(k) = 0$	$\{\uparrow\}$
$n := \text{choose } (k : \text{nat}) : k < k + 1$	$\{0, 1, 2, \dots\}$
$n := \text{choose } (k : \text{nat}) : k > 2 \text{ and } k < 4$	$\{3\}$

Result 1 - Background - WhileCC Programming Language

Syntax

• Terms

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

• 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

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

• Procedures

$P ::= \text{proc } D \text{ begin } S \text{ end}$

Algebra \mathcal{R}

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

Semantics

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

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

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

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

Statement	Possible Values for n
$n := \text{choose}$ ($k : \text{nat}$) : $k < 0$	$\{\uparrow\}$
$n := \text{choose}$ ($k : \text{nat}$) : $\text{toReal}(k) = 0$	$\{\uparrow\}$
$n := \text{choose}$ ($k : \text{nat}$) : $k < k + 1$	$\{0, 1, 2, \dots\}$
$n := \text{choose}$ ($k : \text{nat}$) : $k > 2$ and $k < 4$	$\{3\}$

Result 1 - Background - WhileCC Programming Language

Syntax

Terms

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

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

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

Procedures

$P ::= \text{proc } D \text{ begin } S \text{ end}$

Algebra \mathcal{R}

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

Semantics

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

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

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

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

Statement	Possible Values for n
$n := \text{choose}$ ($k : \text{nat}$) : $k < 0$	$\{\uparrow\}$
$n := \text{choose}$ ($k : \text{nat}$) : $\text{toReal}(k) = 0$	$\{\uparrow\}$
$n := \text{choose}$ ($k : \text{nat}$) : $k < k + 1$	$\{0, 1, 2, \dots\}$
$n := \text{choose}$ ($k : \text{nat}$) : $k > 2$ and $k < 4$	$\{3\}$

Result 1 - Background - WhileCC Programming Language

Syntax

Terms

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

Statements

$S ::=$

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

| if b then S_1 else S_2 fi

| while b do S_0 od

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

Procedures

$P ::= \text{proc } D \text{ begin } S \text{ end}$

Algebra \mathcal{R}

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

Semantics

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

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

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

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

Statement	Possible Values for n
$n := \text{choose}$ ($k : \text{nat}$) : $k < 0$	$\{\uparrow\}$
$n := \text{choose}$ ($k : \text{nat}$) : $\text{toReal}(k) = 0$	$\{\uparrow\}$
$n := \text{choose}$ ($k : \text{nat}$) : $k < k + 1$	$\{0, 1, 2, \dots\}$
$n := \text{choose}$ ($k : \text{nat}$) : $k > 2$ and $k < 4$	$\{3\}$

Result 1 - Background - WhileCC Programming Language

Syntax

• Terms

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

• 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

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

• Procedures

$P ::= \text{proc } D \text{ begin } S \text{ end}$

Algebra \mathcal{R}

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

Semantics

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

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

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

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

Statement	Possible Values for n
$n := \text{choose}$ ($k : \text{nat}$) : $k < 0$	$\{\uparrow\}$
$n := \text{choose}$ ($k : \text{nat}$) : $\text{toReal}(k) = 0$	$\{\uparrow\}$
$n := \text{choose}$ ($k : \text{nat}$) : $k < k + 1$	$\{0, 1, 2, \dots\}$
$n := \text{choose}$ ($k : \text{nat}$) : $k > 2$ and $k < 4$	$\{3\}$

Result 1 - Background - WhileCC Programming Language

Syntax

Terms

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

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

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

Procedures

$P ::= \text{proc } D \text{ begin } S \text{ end}$

Algebra \mathcal{R}

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

Semantics

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

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

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

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

Statement	Possible Values for n
$n := \text{choose } (k : \text{nat}) : k < 0$	$\{\uparrow\}$
$n := \text{choose } (k : \text{nat}) : \text{toReal}(k) = 0$	$\{\uparrow\}$
$n := \text{choose } (k : \text{nat}) : k < k + 1$	$\{0, 1, 2, \dots\}$
$n := \text{choose } (k : \text{nat}) : k > 2 \text{ and } k < 4$	$\{3\}$

Result 1 - Background - WhileCC Programming Language

Syntax

Terms

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

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

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

Procedures

$P ::= \text{proc } D \text{ begin } S \text{ end}$

Algebra \mathcal{R}

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

Semantics

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

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

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

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

Statement	Possible Values for n
$n := \text{choose } (k : \text{nat}) : k < 0$	$\{\uparrow\}$
$n := \text{choose } (k : \text{nat}) : \text{toReal}(k) = 0$	$\{\uparrow\}$
$n := \text{choose } (k : \text{nat}) : k < k + 1$	$\{0, 1, 2, \dots\}$
$n := \text{choose } (k : \text{nat}) : k > 2 \text{ and } k < 4$	$\{3\}$

Result 1 - Background - WhileCC Programming Language

Syntax

Terms

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

Statements

$S ::=$

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

| if b then S_1 else S_2 fi

| while b do S_0 od

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

Procedures

$P ::= \text{proc } D \text{ begin } S \text{ end}$

Algebra \mathcal{R}

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

Semantics

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

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

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

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

Statement	Possible Values for n
$n := \text{choose} (k : \text{nat}) : k < 0$	$\{\uparrow\}$
$n := \text{choose} (k : \text{nat}) : \text{toReal}(k) = 0$	$\{\uparrow\}$
$n := \text{choose} (k : \text{nat}) : k < k + 1$	$\{0, 1, 2, \dots\}$
$n := \text{choose} (k : \text{nat}) : k > 2 \text{ and } k < 4$	$\{3\}$

Result 1 - Background - WhileCC Programming Language

Syntax

Terms

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

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

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

Procedures

$P ::= \text{proc } D \text{ begin } S \text{ end}$

Algebra \mathcal{R}

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

Semantics

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

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

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

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

Statement	Possible Values for n
$n := \text{choose } (k : \text{nat}) : k < 0$	$\{\uparrow\}$
$n := \text{choose } (k : \text{nat}) : \text{toReal}(k) = 0$	$\{\uparrow\}$
$n := \text{choose } (k : \text{nat}) : k < k + 1$	$\{0, 1, 2, \dots\}$
$n := \text{choose } (k : \text{nat}) : k > 2 \text{ and } k < 4$	$\{3\}$

Result 1 - Background - WhileCC Programming Language

Syntax

• Terms

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

• 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

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

• Procedures

$P ::= \text{proc } D \text{ begin } S \text{ end}$

Algebra \mathcal{R}

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

Semantics

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

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

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

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

Statement	Possible Values for n
$n := \text{choose}$ ($k : \text{nat}$) : $k < 0$	$\{\uparrow\}$
$n := \text{choose}$ ($k : \text{nat}$) : $\text{toReal}(k) = 0$	$\{\uparrow\}$
$n := \text{choose}$ ($k : \text{nat}$) : $k < k + 1$	$\{0, 1, 2, \dots\}$
$n := \text{choose}$ ($k : \text{nat}$) : $k > 2$ and $k < 4$	$\{3\}$

Result 1 - Background

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

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

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

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

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

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 $\text{real} \times \text{nat} \rightarrow \text{real}$ on \mathcal{R} is said to *approximate* a function $f : \mathbb{R} \rightarrow \mathbb{R}$ iff for all $n \in \mathbb{N}$ and all $x \in \mathbb{R}$:

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

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

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

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 $\text{real} \times \text{nat} \rightarrow \text{real}$ on \mathcal{R} is said to *approximate* a function $f : \mathbb{R} \rightarrow \mathbb{R}$ iff for all $n \in \mathbb{N}$ and all $x \in \mathbb{R}$:

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

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

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

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 $\text{real} \times \text{nat} \rightarrow \text{real}$ on \mathcal{R} is said to *approximate* a function $f : \mathbb{R} \rightarrow \mathbb{R}$ iff for all $n \in \mathbb{N}$ and all $x \in \mathbb{R}$:

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

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

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

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 $\text{real} \times \text{nat} \rightarrow \text{real}$ on \mathcal{R} is said to *approximate* a function $f : \mathbb{R} \rightarrow \mathbb{R}$ iff for all $n \in \mathbb{N}$ and all $x \in \mathbb{R}$:

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

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

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

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 $\text{real} \times \text{nat} \rightarrow \text{real}$ on \mathcal{R} is said to *approximate* a function $f : \mathbb{R} \rightarrow \mathbb{R}$ iff for all $n \in \mathbb{N}$ and all $x \in \mathbb{R}$:

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

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

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

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 $\text{real} \times \text{nat} \rightarrow \text{real}$ on \mathcal{R} is said to *approximate* a function $f : \mathbb{R} \rightarrow \mathbb{R}$ iff for all $n \in \mathbb{N}$ and all $x \in \mathbb{R}$:

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

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

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

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 $\text{real} \times \text{nat} \rightarrow \text{real}$ on \mathcal{R} is said to *approximate* a function $f : \mathbb{R} \rightarrow \mathbb{R}$ iff for all $n \in \mathbb{N}$ and all $x \in \mathbb{R}$:

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

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

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

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 $\text{real} \times \text{nat} \rightarrow \text{real}$ on \mathcal{R} is said to *approximate* a function $f : \mathbb{R} \rightarrow \mathbb{R}$ iff for all $n \in \mathbb{N}$ and all $x \in \mathbb{R}$:

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

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

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

Goal

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

Challenge: Using comparison operators introduces undefinedness.

How do we **WhileCC**-approximate “piecewise” functions?

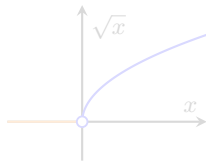
Result 1 - Challenges

Problem

When defining piecewise functions, comparison makes a hole!

Example: Even Root - First Attempt

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



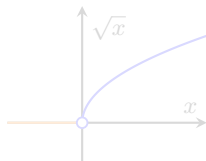
Result 1 - Challenges

Problem

When defining piecewise functions, comparison makes a hole!

Example: Even Root - First Attempt

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



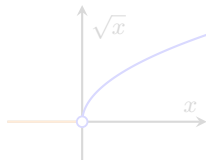
Result 1 - Challenges

Problem

When defining piecewise functions, comparison makes a hole!

Example: Even Root - First Attempt

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



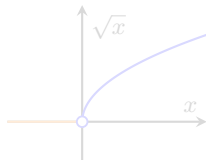
Result 1 - Challenges

Problem

When defining piecewise functions, comparison makes a hole!

Example: Even Root - First Attempt

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



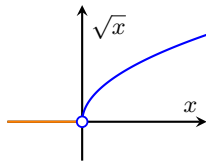
Result 1 - Challenges

Problem

When defining piecewise functions, comparison makes a hole!

Example: Even Root - First Attempt

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



Result 1 - Challenges

Problem

When defining piecewise functions, comparison makes a hole!

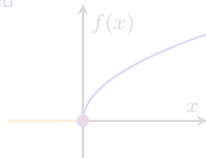
Solution

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

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

```
 $\text{chosenVal} := \text{choose } (k : \text{nat}) : \text{proc}$ 
  in  $k : \text{nat } x : \text{real}$ 
begin
  if  $k =_{\text{N}} 1$  then
    return  $0 < x$ 
  else if  $k =_{\text{N}} 2$  then
    return  $x < l$ 
  else
    return ff
  fi
end
```

```
if  $\text{chosenVal} =_{\text{N}} 1$  then
  return  $\text{Root}(x, c)$ 
else if  $\text{chosenVal} =_{\text{N}} 2$  then
  return 0
fi
end
```



Result 1 - Challenges

Problem

When defining piecewise functions, comparison makes a hole!

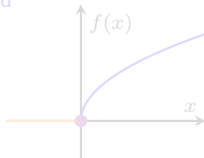
Solution

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

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

```
 $\text{chosenVal} := \text{choose } (k : \text{nat}) : \text{proc}$ 
  in  $k : \text{nat } x : \text{real}$ 
begin
  if  $k =_{\text{N}} 1$  then
    return  $0 < x$ 
  else if  $k =_{\text{N}} 2$  then
    return  $x < l$ 
  else
    return ff
  fi
end
```

```
if  $\text{chosenVal} =_{\text{N}} 1$  then
  return  $\text{Root}(x, c)$ 
else if  $\text{chosenVal} =_{\text{N}} 2$  then
  return 0
fi
end
```



Result 1 - Challenges

Problem

When defining piecewise functions, comparison makes a hole!

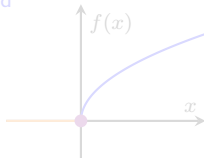
Solution

- 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!

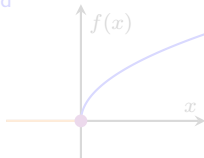
Solution

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

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

```
 $\text{chosenVal} := \text{choose } (k : \text{nat}) : \text{proc}$ 
  in  $k : \text{nat } x : \text{real}$ 
begin
  if  $k =_{\mathbb{N}} 1$  then
    return  $0 < x$ 
  else if  $k =_{\mathbb{N}} 2$  then
    return  $x < l$ 
  else
    return ff
  fi
end
```

```
if  $\text{chosenVal} =_{\mathbb{N}} 1$  then
  return  $\text{Root}(x, c)$ 
else if  $\text{chosenVal} =_{\mathbb{N}} 2$  then
  return 0
fi
end
```



Result 1 - Challenges

Problem

When defining piecewise functions, comparison makes a hole!

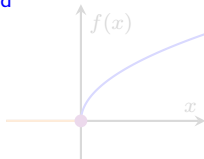
Solution

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

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

   $\text{chosenVal} := \text{choose } (k : \text{nat}) : \text{proc}$ 
    in  $k : \text{nat } x : \text{real}$ 
    begin
      if  $k =_{\text{N}} 1$  then
        return  $0 < x$ 
      else if  $k =_{\text{N}} 2$  then
        return  $x < l$ 
      else
        return ff
      fi
    end

  if  $\text{chosenVal} =_{\text{N}} 1$  then
    return  $\text{Root}(x, c)$ 
  else if  $\text{chosenVal} =_{\text{N}} 2$  then
    return 0
  fi
end
```



Result 1 - Challenges

Problem

When defining piecewise functions, comparison makes a hole!

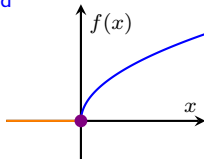
Solution

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

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

```
   $\text{chosenVal} := \text{choose } (k : \text{nat}) : \text{proc}$ 
    in  $k : \text{nat } x : \text{real}$ 
  begin
    if  $k =_{\text{N}} 1$  then
      return  $0 < x$ 
    else if  $k =_{\text{N}} 2$  then
      return  $x < l$ 
    else
      return ff
    fi
  end
```

```
  if  $\text{chosenVal} =_{\text{N}} 1$  then
    return  $\text{Root}(x, c)$ 
  else if  $\text{chosenVal} =_{\text{N}} 2$  then
    return 0
  fi
end
```



Result 2

Theorem 2.1 (Acceptability Theorem: Part 1)

Let $f : \mathbb{R} \rightarrow \mathbb{R}$ be an elementary function. Then, $\text{dom}(f)$ 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:

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

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

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

$\text{dom}(f \circ g) = [-1, 1]$ 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

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

Result 2 - Challenges

Theorem 2.1 (Acceptability Theorem: Part 1)

Let $f : \mathbb{R} \rightarrow \mathbb{R}$ be an elementary function. Then, $\text{dom}(f)$ 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:

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

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

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

$\text{dom}(f \circ g) = [-1, 1]$ 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

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

Result 2 - Challenges

Theorem 2.1 (Acceptability Theorem: Part 1)

Let $f : \mathbb{R} \rightarrow \mathbb{R}$ be an elementary function. Then, $\text{dom}(f)$ 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:

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

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

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

$\text{dom}(f \circ g) = [-1, 1]$ 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

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

Result 2 - Challenges

Theorem 2.1 (Acceptability Theorem: Part 1)

Let $f : \mathbb{R} \rightarrow \mathbb{R}$ be an elementary function. Then, $\text{dom}(f)$ 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:

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

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

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

$\text{dom}(f \circ g) = [-1, 1]$ 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

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

Result 2 - Challenges

Theorem 2.1 (Acceptability Theorem: Part 1)

Let $f : \mathbb{R} \rightarrow \mathbb{R}$ be an elementary function. Then, $\text{dom}(f)$ 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:

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

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

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

$\text{dom}(f \circ g) = [-1, 1]$ 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

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

Result 2 - Challenges

Theorem 2.1 (Acceptability Theorem: Part 1)

Let $f : \mathbb{R} \rightarrow \mathbb{R}$ be an elementary function. Then, $\text{dom}(f)$ 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:

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

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

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

$\text{dom}(f \circ g) = [-1, 1]$ 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

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

Result 2 - Challenges

Theorem 2.1 (Acceptability Theorem: Part 1)

Let $f : \mathbb{R} \rightarrow \mathbb{R}$ be an elementary function. Then, $\text{dom}(f)$ 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:

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

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

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

$\text{dom}(f \circ g) = [-1, 1]$ 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

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

Result 2 - Challenges

Theorem 2.1 (Acceptability Theorem: Part 1)

Let $f : \mathbb{R} \rightarrow \mathbb{R}$ be an elementary function. Then, $\text{dom}(f)$ 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:

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

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

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

$\text{dom}(f \circ g) = [-1, 1]$ 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

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

Result 2 - Challenges

Theorem 2.1 (Acceptability Theorem: Part 1)

Let $f : \mathbb{R} \rightarrow \mathbb{R}$ be an elementary function. Then, $\text{dom}(f)$ 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:

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

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

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

$\text{dom}(f \circ g) = [-1, 1]$ 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

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

Result 2 - Challenges

Theorem 2.1 (Acceptability Theorem: Part 1)

Let $f : \mathbb{R} \rightarrow \mathbb{R}$ be an elementary function. Then, $\text{dom}(f)$ 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:

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

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

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

$\text{dom}(f \circ g) = [-1, 1]$ 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

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

Result 2 - Challenges

Theorem 2.1 (Acceptability Theorem: Part 1)

Let $f : \mathbb{R} \rightarrow \mathbb{R}$ be an elementary function. Then, $\text{dom}(f)$ 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:

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

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

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

$\text{dom}(f \circ g) = [-1, 1]$ 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

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

Result 2 - Challenges

Theorem 2.1 (Acceptability Theorem: Part 1)

Let $f : \mathbb{R} \rightarrow \mathbb{R}$ be an elementary function. Then, $\text{dom}(f)$ 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:

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

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

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

$\text{dom}(f \circ g) = [-1, 1]$ 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

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

Result 2 - Challenges

Theorem 2.1 (Acceptability Theorem: Part 1)

Let $f : \mathbb{R} \rightarrow \mathbb{R}$ be an elementary function. Then, $\text{dom}(f)$ 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:

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

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

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

$\text{dom}(f \circ g) = [-1, 1]$ 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

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

Result 2 - Challenges

Theorem 2.1 (Acceptability Theorem: Part 1)

Let $f : \mathbb{R} \rightarrow \mathbb{R}$ be an elementary function. Then, $\text{dom}(f)$ 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:

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

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

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

$\text{dom}(f \circ g) = [-1, 1]$ 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

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

Result 2 - Challenges

Theorem 2.1 (Acceptability Theorem: Part 1)

Let $f : \mathbb{R} \rightarrow \mathbb{R}$ be an elementary function. Then, $\text{dom}(f)$ 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:

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

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

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

$\text{dom}(f \circ g) = [-1, 1]$ 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

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

Result 2 - Challenges

Theorem 2.1 (Acceptability Theorem: Part 1)

Let $f : \mathbb{R} \rightarrow \mathbb{R}$ be an elementary function. Then, $\text{dom}(f)$ 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:

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

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

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

$\text{dom}(f \circ g) = [-1, 1]$ 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

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

Result 2.1 - Takeaways

Theorem (Reducing Exhaustion-reflection to a Decision Procedure)

A function $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ 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.

Building an effective open exhaustion

Result 2.1 - Takeaways

Theorem (Reducing Exhaustion-reflection to a Decision Procedure)

A function $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ 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.

Building an effective open exhaustion

Result 2.1 - Takeaways

Theorem (Reducing Exhaustion-reflection to a Decision Procedure)

A function $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ 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.

Building an effective open exhaustion

Result 2.1 - Takeaways

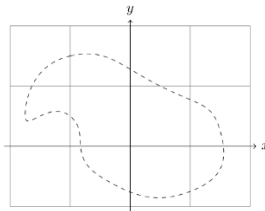
Theorem (Reducing Exhaustion-reflection to a Decision Procedure)

A function $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ 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.

Building an effective open exhaustion

(a) step 0



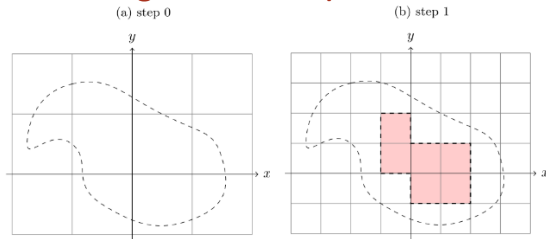
Result 2.1 - Takeaways

Theorem (Reducing Exhaustion-reflection to a Decision Procedure)

A function $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ 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.

Building an effective open exhaustion



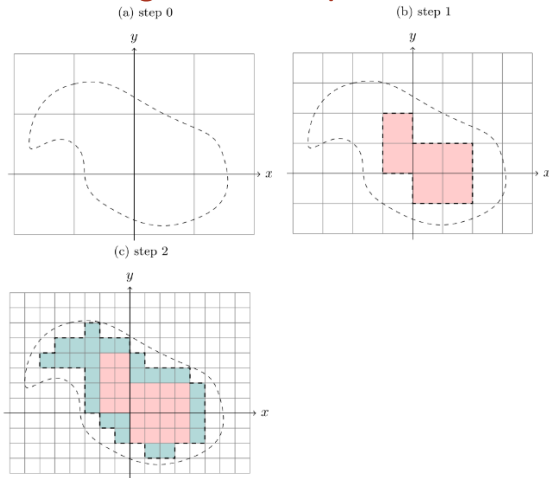
Result 2.1 - Takeaways

Theorem (Reducing Exhaustion-reflection to a Decision Procedure)

A function $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ 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.

Building an effective open exhaustion



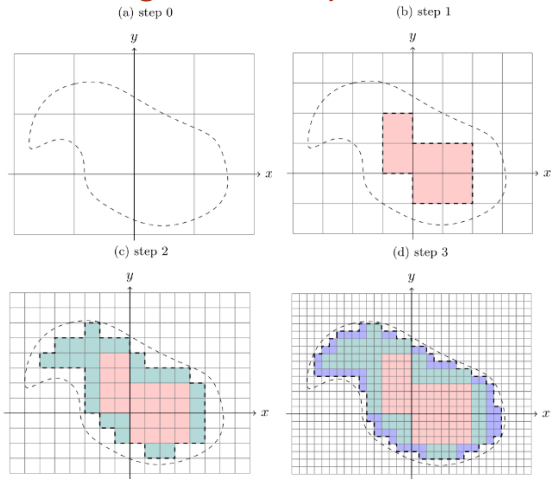
Result 2.1 - Takeaways

Theorem (Reducing Exhaustion-reflection to a Decision Procedure)

A function $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ 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.

Building an effective open exhaustion



Result 2

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

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

Theorem

Let $f : \mathbb{R} \rightarrow \mathbb{R}$ be **WhileCC**-approximable and monotone on its domain. Then f has a local continuity witness.

The notion of effective local uniform continuity is independent of effective open exhaustion.

Result 2

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

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

Theorem

Let $f : \mathbb{R} \rightarrow \mathbb{R}$ be **WhileCC**-approximable and monotone on its domain. Then f has a local continuity witness.

The notion of effective local uniform continuity is independent of effective open exhaustion.

Result 2

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

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

Theorem

Let $f : \mathbb{R} \rightarrow \mathbb{R}$ be **WhileCC**-approximable and monotone on its domain. Then f has a local continuity witness.

The notion of effective local uniform continuity is independent of effective open exhaustion.

Result 2

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

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

Theorem

Let $f : \mathbb{R} \rightarrow \mathbb{R}$ be **WhileCC**-approximable and monotone on its domain. Then f has a local continuity witness.

The notion of effective local uniform continuity is independent of effective open exhaustion.

Result 2

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

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

Theorem

Let $f : \mathbb{R} \rightarrow \mathbb{R}$ be **WhileCC**-approximable and monotone on its domain. Then f has a local continuity witness.

The notion of effective local uniform continuity is independent of effective open exhaustion.

Result 2

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

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

Theorem

Let $f : \mathbb{R} \rightarrow \mathbb{R}$ be **WhileCC**-approximable and monotone on its domain. Then f has a local continuity witness.

The notion of effective local uniform continuity is independent of effective open exhaustion.

Summary

We proved that:

- all elementary functions are **WhileCC**-approximable.
- all elementary functions are acceptable.

We also

- presented an **alternative characterization** of acceptable functions using the **local continuity witness** concept, and
- 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.

We also

- presented an **alternative characterization** of acceptable functions using the **local continuity witness** concept, and
- 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.

We also

- presented an **alternative characterization** of acceptable functions using the **local continuity witness** concept, and
- 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.

We also

- presented an **alternative characterization** of acceptable functions using the **local continuity witness** concept, and
- 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.

We also

- presented an **alternative characterization** of acceptable functions using the **local continuity witness** concept, and
- 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 $\mathbb{R}^m \rightarrow \mathbb{R}$?
- 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 $\mathbb{R}^m \rightarrow \mathbb{R}$?
- 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 $\mathbb{R}^m \rightarrow \mathbb{R}$?
- 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 $\mathbb{R}^m \rightarrow \mathbb{R}$?
- 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 $\mathbb{R}^m \rightarrow \mathbb{R}$?
- 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 $\mathbb{R}^m \rightarrow \mathbb{R}$?
- 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 $\mathbb{R}^m \rightarrow \mathbb{R}$?
- 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 $\mathbb{R}^m \rightarrow \mathbb{R}$?
- 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 $\mathbb{R}^m \rightarrow \mathbb{R}$?
- 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 $\mathbb{R}^m \rightarrow \mathbb{R}$?
- 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 $\mathbb{R}^m \rightarrow \mathbb{R}$?
- 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

- M. Q. Fu and J. 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](https://doi.org/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=iU4zDAAQBAJ>.
- J. Tucker and J. 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](https://doi.org/10.1016/S0304-3975(98)00297-7).
- J. Tucker and J. 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](https://doi.org/10.1016/j.jlap.2003.10.001).
- J. V. Tucker and J. I. Zucker. Abstract versus concrete computation on metric partial algebras. *ACM Trans. Comput. Log.*, 5(4):611–668, 2004. doi:[10.1145/1024922.1024924](https://doi.org/10.1145/1024922.1024924).

A HUGE Thank you!