Computing in Cantor's Paradise With λ_{ZFC}

or

How to Make an Infinitary λ-Calculus in κ Easy Steps

(and Why)

FLOPS 2012

Neil Toronto and Jay McCarthy

PLT @ Brigham Young University

Living in Cantor's Paradise

David Hilbert

"No one shall expel us from the Paradise that Cantor has created."

Living in Cantor's Paradise

David Hilbert

"No one shall expel us from the Paradise that Cantor has created."

Cantor's Paradise (set theory) is characterized by mind-boggling orders of ever-increasing infinities...

Living in Cantor’s Paradise

David Hilbert

“No one shall expel us from the Paradise that Cantor has created.”

Cantor’s Paradise (set theory) is characterized by mind-boggling orders of ever-increasing infinities...

... but mathematicians still want to use computers to answer questions!

Living in Cantor's Paradise

David Hilbert

"No one shall expel us from the Paradise that Cantor has created."

Cantor's Paradise (set theory) is characterized by mind-boggling orders of ever-increasing infinities...

... but mathematicians still want to use computers to answer questions!

- Simple example: $\pi = 16 \tan^{-1}\left(\frac{1}{5}\right) - 4 \tan^{-1}\left(\frac{1}{239}\right)$

Options for Domain Specific Languages

- Option 1: Write theorems and proofs in a proof assistant, extract programs
  - Problem: Re-proving theorems takes a long time!
  - Hurd 2002: Dissertation half comprised of convincing HOL of theorems from early-1900s measure theory
  - O’Connor 2008: Five months to convince Coq of his own theorems (which had detailed, published proofs)

Options for Domain Specific Languages

- Option 1: Write theorems and proofs in a proof assistant, extract programs
    - Problem: Re-proving theorems takes a long time!
    - Hurd 2002: Dissertation half comprised of convincing HOL of theorems from early-1900s measure theory
    - O'Connor 2008: Five months to convince Coq of his own theorems (which had detailed, published proofs)
- Option 2: Write semantics in contemporary math
    - Problem: Higher-order *anything* is difficult
    - Problem: No connection to implementation

Options for Domain Specific Languages

- Option 1: Write theorems and proofs in a proof assistant, extract programs
  - Problem: Re-proving theorems takes a long time!
  - Hurd 2002: Dissertation half comprised of convincing HOL of theorems from early-1900s measure theory
  - O'Connor 2008: Five months to convince Coq of his own theorems (which had detailed, published proofs)
- Option 2: Write semantics in contemporary math
  - Problem: Higher-order *anything* is difficult
  - Problem: No connection to implementation
- ∃n. Option n is a middle ground?

Example: Beautiful Differentiation

- Elliot 2010: Derives an automatic differentiation implementation from this exact specification:

$$d\ f\ x \ := \ \lim_{h \to 0} \frac{f\ (x+h) - f\ x}{h}$$

Example: Beautiful Differentiation

- Elliot 2010: Derives an automatic differentiation implementation from this exact specification:

$$d\ f\ x \ := \ \lim_{h \to 0} \frac{f\ (x + h) - f\ x}{h}$$

- Unimplementable because of $\lim_{h \to 0}$

Example: Beautiful Differentiation

- Elliot 2010: Derives an automatic differentiation implementation from this exact specification:

$$d\ f\ x\ := \lim_{h \to 0} \frac{f\ (x+h) - f\ x}{h}$$

- Unimplementable because of $\lim\limits_{h \to 0}$

- How Elliot does it:

1. Reformulates differentiation in terms of *toD* to hide use of *d*

2. Uses differentiation theorems and **`Functor`** and **`Applicative`** instance definitions to derive *d*-free functions

3. Implements using floating-point to approximate reals

Beautiful Derivations

$$
\begin{array}{lll}
& toD\ (sin\ u) & \\
\equiv & liftA_2\ D\ (sin\ u)\ (d\ (sin\ u)) & \text{definition of } toD \\
\equiv & liftA_2\ D\ (sin\ u)\ (d\ u * cos\ u) & d\ (sin\ u) = d\ u * cos\ u \\
\equiv & \lambda x \rightarrow D\ ((sin\ u)\ x)\ ((d\ u * cos\ u)\ x) & \text{definition of } liftA_2 \\
\equiv & \ldots & \\
\equiv & sin\ (toD\ u) &
\end{array}
$$

Beautiful Derivations

$$
\begin{array}{lll}
& toD\ (sin\ u) & \\
\equiv & liftA_2\ D\ (sin\ u)\ (d\ (sin\ u)) & \text{definition of } toD \\
\equiv & liftA_2\ D\ (sin\ u)\ (d\ u * cos\ u) & d\ (sin\ u) = d\ u * cos\ u \\
\equiv & \lambda x \rightarrow D\ ((sin\ u)\ x)\ ((d\ u * cos\ u)\ x) & \text{definition of } liftA_2 \\
\equiv & \ldots & \\
\equiv & sin\ (toD\ u) &
\end{array}
$$

- Problem: To elegantly derive the implementation, the derivations have to be done in a language that doesn't exist!

Apologies

"We have no implementation of *d*, so this definition of *toD* will **serve as a specification**, not an implementation."

"This **definition is not executable**, however, since *d* is not."

"Every remaining use of *d* is applied to a function whose derivative is known, so we can replace each use.... Now we have **an executable implementation again**."

"Again, this definition can be refactored, followed by replacing the **non-effective [unimplementable] applications** of *d* with known derivatives."

Example: From Bayesian Notation to Pure Racket

- Toronto and McCarthy 2010: Derive implementation of Bayesian modeling language from an exact specification using

$$\begin{aligned} \mathsf{sum}\ \mathsf{f}\ \mathsf{A} &:= \sum_{\omega \in \mathsf{A}} \mathsf{f}\ \omega \\ \mathsf{preimage}\ \mathsf{A}\ \mathsf{f}\ \mathsf{B} &:= \{\mathsf{x} \in \mathsf{A} \mid \mathsf{f}\ \mathsf{x} \in \mathsf{B}\} \end{aligned}$$

Example: From Bayesian Notation to Pure Racket

- Toronto and McCarthy 2010: Derive implementation of Bayesian modeling language from an exact specification using

$$\begin{aligned} \mathsf{sum\ f\ A} &:= \sum_{\omega \in \mathsf{A}} \mathsf{f}\ \omega \\ \mathsf{preimage\ A\ f\ B} &:= \{\mathsf{x} \in \mathsf{A} \,|\, \mathsf{f\ x} \in \mathsf{B}\} \end{aligned}$$

- Unimplementable because A may be a countable set

Example: From Bayesian Notation to Pure Racket

- Toronto and McCarthy 2010: Derive implementation of Bayesian modeling language from an exact specification using

$$\mathsf{sum}\ \mathsf{f}\ \mathsf{A} := \sum_{\omega \in \mathsf{A}} \mathsf{f}\ \omega$$

$$\mathsf{preimage}\ \mathsf{A}\ \mathsf{f}\ \mathsf{B} := \{\mathsf{x} \in \mathsf{A} \,|\, \mathsf{f}\ \mathsf{x} \in \mathsf{B}\}$$

- Unimplementable because A may be a countable set
- How we did it:

1. Assume a very powerful lambda calculus ($\boldsymbol{\lambda}_{\mathrm{ZFC}}$) exists
2. Define exact meaning of Bayesian notation in this language
3. Derive implementable approximation, prove convergence

Apologies

"[Sketch of $\boldsymbol{\lambda}_{\mathrm{ZFC}}$ features.] **We intend $\boldsymbol{\lambda}_{\mathrm{ZFC}}$** to be contemporary mathematics with well-defined lambdas, or a practical lambda calculus with infinite sets."

Apologies

"[Sketch of $\boldsymbol{\lambda}_{\text{ZFC}}$ features.] **We intend $\boldsymbol{\lambda}_{\text{ZFC}}$** to be contemporary mathematics with well-defined lambdas, or a practical lambda calculus with infinite sets."

Our Vision:

λ calculus

Apologies

“[Sketch of $\boldsymbol{\lambda}_{ZFC}$ features.] **We intend $\boldsymbol{\lambda}_{ZFC}$** to be contemporary mathematics with well-defined lambdas, or a practical lambda calculus with infinite sets.”

Our Vision:

λ calculus **Infinite Sets and Set Operations**

Apologies

“[Sketch of $\boldsymbol{\lambda}_{ZFC}$ features.] **We intend $\boldsymbol{\lambda}_{ZFC}$** to be contemporary mathematics with well-defined lambdas, or a practical lambda calculus with infinite sets.”

Our Vision:

+

=

λ calculus

Infinite Sets and Set Operations

λ_{ZFC}

Apologies

"[Sketch of $\boldsymbol{\lambda}_{ZFC}$ features.] **We intend $\boldsymbol{\lambda}_{ZFC}$** to be contemporary mathematics with well-defined lambdas, or a practical lambda calculus with infinite sets."

Our Vision:

+

=

λ calculus **Infinite Sets and Set Operations** $\boldsymbol{\lambda}_{ZFC}$

- "Computing in Cantor's Paradise With $\boldsymbol{\lambda}_{ZFC}$" realizes this vision

Lambda-ZFC Requirements

- Must be similar to implementation language
  - Higher-order functions and lambdas
  - Computable sublanguage
  - Call-by-value reduction semantics

Lambda-ZFC Requirements

- Must be similar to implementation language
  - Higher-order functions and lambdas
  - Computable sublanguage
  - Call-by-value reduction semantics
- Must have infinite sets
  - Operations expressive enough to do measure theory
  - Apply well-known theorems with only trivial translation

Lambda-ZFC Requirements

- Must be similar to implementation language
  - Higher-order functions and lambdas
  - Computable sublanguage
  - Call-by-value reduction semantics
- Must have infinite sets
  - Operations expressive enough to do measure theory
  - Apply well-known theorems with only trivial translation
- Should treat values uniformly (all values first-class)
  - Specifically allow lambdas in sets: $\{\lambda x.\, x, \lambda x.\, x + x, ...\}$
  - For minimalism: want to use $\langle a, b \rangle := \{\{a\}, \{a, b\}\}$

Building Lambda-ZFC- From the ZFC Axioms (1)

Axiom	$\boldsymbol{\lambda}^{-}_{\mathrm{ZFC}}$
Empty set. There is a set $\varnothing$ with no elements.	$\varnothing$ (value symbol)

Building Lambda-ZFC- From the ZFC Axioms (1)

Axiom	$\boldsymbol{\lambda}^{-}_{\mathrm{ZFC}}$
Empty set. There is a set $\varnothing$ with no elements.	$\varnothing$ (value symbol)
Powerset. The subsets of a set A comprise a set $\mathcal{P}(A)$.	$\mathsf{P}\ e$

Building Lambda-ZFC- From the ZFC Axioms (1)

Axiom	$\boldsymbol{\lambda}^{-}_{\mathrm{ZFC}}$
Empty set. There is a set $\varnothing$ with no elements.	$\varnothing$ (value symbol)
Powerset. The subsets of a set A comprise a set $\mathcal{P}(A)$.	$\mathsf{P}\ e$
Union. The union of a set of sets is a set.	

Building Lambda-ZFC- From the ZFC Axioms (1)

Axiom	$\boldsymbol{\lambda}^{-}_{\mathrm{ZFC}}$
Empty set. There is a set $\varnothing$ with no elements.	$\varnothing$ (value symbol)
Powerset. The subsets of a set A comprise a set $\mathcal{P}(A)$.	P e
Union. The union of a set of sets is a set.	
Example: $\bigcup\{\{x, y\}, \{y, z\}\} = \{x, y, z\}$	U e

Building Lambda-ZFC- From the ZFC Axioms (1)

Axiom	$\boldsymbol{\lambda}^{-}_{\mathrm{ZFC}}$
Empty set. There is a set $\varnothing$ with no elements.	$\varnothing$ (value symbol)
Powerset. The subsets of a set A comprise a set $\mathcal{P}(A)$.	$\mathsf{P}\ e$
Union. The union of a set of sets is a set.	
Example: $\bigcup\{\{x, y\}, \{y, z\}\} = \{x, y, z\}$	$\mathsf{U}\ e$
Example: $\bigcup\{A\} = A$	$\mathsf{take}\ e$

Building Lambda-ZFC- From the ZFC Axioms (1)

Axiom	$\boldsymbol{\lambda}^{-}_{\mathrm{ZFC}}$
Empty set. There is a set $\varnothing$ with no elements.	$\varnothing$ (value symbol)
Powerset. The subsets of a set A comprise a set $\mathcal{P}(A)$.	$\mathsf{P}\ e$
Union. The union of a set of sets is a set.	
Example: $\bigcup\{\{x,y\},\{y,z\}\} = \{x,y,z\}$	$\mathsf{U}\ e$
Example: $\bigcup\{A\} = A$	$\mathsf{take}\ e$
Cardinality. Every set A has a unique cardinality $\lvert A\rvert$.	$\mathsf{card}\ e$

Building Lambda-ZFC- From the ZFC Axioms (2)

Axiom $\boldsymbol{\lambda}^{-}_{\mathrm{ZFC}}$

Replacement. The image of a set under a binary predicate is a set.

Building Lambda-ZFC- From the ZFC Axioms (2)

Axiom	$\boldsymbol{\lambda}^{-}_{\mathrm{ZFC}}$
Replacement. The image of a set under a binary predicate is a set.	
Ex.: let $R(n, m) \iff m = n + 1$; then $\mathbb{N}^{+} = \{m \mid n \in \mathbb{N} \wedge R(n, m)\}$	

Building Lambda-ZFC- From the ZFC Axioms (2)

Axiom	$\boldsymbol{\lambda}^{-}_{\text{ZFC}}$
Replacement. The image of a set under a binary predicate is a set.	
Ex.: let $R(n,m) \iff m = n+1$; then $\mathbb{N}^+ = \{m \mid n \in \mathbb{N} \wedge R(n,m)\}$	image $e_f\ e_A$ Ex.: image $(\lambda\,\mathsf{n}.\,\mathsf{n}+1)\ \mathbb{N}$

Building Lambda-ZFC- From the ZFC Axioms (2)

Axiom	$\boldsymbol{\lambda}^{-}_{\mathrm{ZFC}}$
Replacement. The image of a set under a binary predicate is a set.	
Ex.: let $R(n,m) \iff m = n+1$; then $\mathbb{N}^{+} = \{m \mid n \in \mathbb{N} \wedge R(n,m)\}$	image $e_f\ e_A$ Ex.: image $(\lambda\, \mathsf{n}.\, \mathsf{n} + 1)\ \mathbb{N}$
Ex.: let $R(x,y) \iff Q(y)$; then the unique set y such that $Q(y)$ is $\bigcup\{y \mid x \in \mathcal{P}(\varnothing) \wedge R(x,y)\}$	

Building Lambda-ZFC- From the ZFC Axioms (2)

Axiom	$\boldsymbol{\lambda}^{-}_{\mathrm{ZFC}}$
Replacement. The image of a set under a binary predicate is a set.	
Ex.: let $R(n, m) \iff m = n + 1$; then $\mathbb{N}^{+} = \{m \mid n \in \mathbb{N} \wedge R(n, m)\}$	image $e_f\ e_A$ Ex.: image $(\lambda\,\mathsf{n}.\,\mathsf{n}+1)\ \mathbb{N}$
Ex.: let $R(x, y) \iff Q(y)$; then the unique set y such that $Q(y)$ is $\bigcup\{y \mid x \in \mathcal{P}(\varnothing) \wedge R(x, y)\}$	*none* (hasn't been a problem)

Building Lambda-ZFC- From the ZFC Axioms (2)

Axiom	$\boldsymbol{\lambda}^{-}_{\text{ZFC}}$
Replacement. The image of a set under a binary predicate is a set.	
Ex.: let $R(n,m) \iff m = n+1$; then $\mathbb{N}^{+} = \{m \mid n \in \mathbb{N} \land R(n,m)\}$	image $e_f\ e_A$ Ex.: image $(\lambda\, \mathsf{n}.\, \mathsf{n}+1)\ \mathbb{N}$
Ex.: let $R(x,y) \iff Q(y)$; then the unique set y such that $Q(y)$ is $\bigcup\{y \mid x \in \mathcal{P}(\varnothing) \land R(x,y)\}$	*none* (hasn't been a problem)
Infinity. The language ω of $n ::= 0 \mid n+1$ is a set, where $0 = \varnothing$ and $n+1 = n \cup \{n\}$.	

Building Lambda-ZFC- From the ZFC Axioms (2)

Axiom	$\boldsymbol{\lambda}^{-}_{\mathrm{ZFC}}$
Replacement. The image of a set under a binary predicate is a set.	
Ex.: let $R(n, m) \iff m = n + 1$; then $\mathbb{N}^+ = \{m \mid n \in \mathbb{N} \wedge R(n, m)\}$	image $e_f\ e_A$ Ex.: image $(\lambda\, \mathsf{n}.\, \mathsf{n} + 1)\ \mathbb{N}$
Ex.: let $R(x, y) \iff Q(y)$; then the unique set y such that $Q(y)$ is $\bigcup\{y \mid x \in \mathcal{P}(\varnothing) \wedge R(x, y)\}$	*none* (hasn't been a problem)
Infinity. The language ω of $n ::= 0 \mid n + 1$ is a set, where $0 = \varnothing$ and $n + 1 = n \cup \{n\}$.	ω (value symbol)

Grammar of Finite Terms

- Final $\boldsymbol{\lambda}^{-}_{\mathrm{ZFC}}$ grammar:

$$e \;::=\; x \mid v \mid e\ e \mid \mathsf{if}\ e\ e\ e \mid$$

Grammar of Finite Terms

- Final $\boldsymbol{\lambda}_{\mathrm{ZFC}}^{-}$ grammar:

$$\begin{aligned} e \; ::= \; & x \mid v \mid e\; e \mid \mathsf{if}\; e\; e\; e \mid \\ & e \in e \mid \mathsf{U}\; e \mid \mathsf{take}\; e \mid \mathsf{P}\; e \mid \mathsf{image}\; e\; e \mid \mathsf{card}\; e \end{aligned}$$

Grammar of Finite Terms

- Final $\boldsymbol{\lambda}^{-}_{\mathrm{ZFC}}$ grammar:

$$
\begin{aligned}
e \;&::=\; x \mid v \mid e\ e \mid \mathsf{if}\ e\ e\ e \mid \\
&\qquad e \in e \mid \mathsf{U}\ e \mid \mathsf{take}\ e \mid \mathsf{P}\ e \mid \mathsf{image}\ e\ e \mid \mathsf{card}\ e \\
v \;&::=\; \mathsf{false} \mid \mathsf{true} \mid \lambda .\, e \mid \varnothing \mid \omega
\end{aligned}
$$

Grammar of Finite Terms

- Final $\boldsymbol{\lambda}_{\text{ZFC}}^{-}$ grammar:

$$
\begin{aligned}
e &::= x \mid v \mid e\ e \mid \mathsf{if}\ e\ e\ e \mid \\
&\qquad e \in e \mid \mathsf{U}\ e \mid \mathsf{take}\ e \mid \mathsf{P}\ e \mid \mathsf{image}\ e\ e \mid \mathsf{card}\ e \\
v &::= \mathsf{false} \mid \mathsf{true} \mid \lambda.\, e \mid \varnothing \mid \omega \\
x &::= 0 \mid 1 \mid 2 \mid \cdots
\end{aligned}
$$

Grammar of Finite Terms

- Final $\boldsymbol{\lambda}_{\mathrm{ZFC}}^{-}$ grammar:

$$
\begin{aligned}
e \;&::=\; x \mid v \mid e\ e \mid \mathsf{if}\ e\ e\ e \mid \\
&\quad\;\; e \in e \mid \mathsf{U}\ e \mid \mathsf{take}\ e \mid \mathsf{P}\ e \mid \mathsf{image}\ e\ e \mid \mathsf{card}\ e \\
v \;&::=\; \mathsf{false} \mid \mathsf{true} \mid \lambda.\, e \mid \varnothing \mid \omega \\
x \;&::=\; 0 \mid 1 \mid 2 \mid \cdots
\end{aligned}
$$

- Computable sublanguage: Remove ω

Grammar of Finite Terms

- Final $\boldsymbol{\lambda}_{\mathrm{ZFC}}^{-}$ grammar:

$$
\begin{aligned}
e &::= x \mid v \mid e\ e \mid \mathsf{if}\ e\ e\ e \mid \\
&\quad\ e \in e \mid \mathsf{U}\ e \mid \mathsf{take}\ e \mid \mathsf{P}\ e \mid \mathsf{image}\ e\ e \mid \mathsf{card}\ e \\
v &::= \mathsf{false} \mid \mathsf{true} \mid \lambda.\, e \mid \varnothing \mid \omega \\
x &::= 0 \mid 1 \mid 2 \mid \cdots
\end{aligned}
$$

- Computable sublanguage: Remove ω
- Problem: What should $\mathsf{P}\ \varnothing$ reduce to? $\{\varnothing\}$ isn't a value...

An Easy (???) Solution

Solution: Include all the sets as values!

An Easy (???) Solution

Solution: Include all the sets as values!

- Problem 1: Set theory (the metalanguage) is single-sorted
- Solution 1: Recursively encode expressions as sets using tags (i.e. SICP-style records)
  - $\langle t_\lambda, e \rangle$ (equivalently $\{\{t_\lambda\}, \{t_\lambda, e\}\}$) is a lambda with body e
  - $\langle t_{\text{set}}, A \rangle$ is a set containing the members of A (e.g. the encoding of $\{\varnothing\}$ is $\langle t_{\text{set}}, \{\langle t_{\text{set}}, \varnothing \rangle\}\rangle$)

An Easy (???) Solution

Solution: Include all the sets as values!

- Problem 1: Set theory (the metalanguage) is single-sorted
- Solution 1: Recursively encode expressions as sets using tags (i.e. SICP-style records)
  - $\langle t_\lambda, e\rangle$ (equivalently $\{\{t_\lambda\}, \{t_\lambda, e\}\}$) is a lambda with body e
  - $\langle t_{\mathrm{set}}, A\rangle$ is a set containing the members of A (e.g. the encoding of $\{\varnothing\}$ is $\langle t_{\mathrm{set}}, \{\langle t_{\mathrm{set}}, \varnothing\rangle\}\rangle$)
- Problem 2: "All the sets" is not a set
- Solution 2: Find a set that acts enough like "all the sets"

All the Sets (1)

- Curious fact: unfolding $\mathcal{P}$ generates the **hereditarily finite** sets

$$\begin{aligned} \mathcal{V}(0) &= \varnothing \\ \mathcal{V}(n+1) &= \mathcal{P}(\mathcal{V}(n)) \qquad \text{successor ordinal } n+1 \end{aligned}$$

All the Sets (1)

- Curious fact: unfolding $\mathcal{P}$ generates the **hereditarily finite** sets

$$\begin{aligned} \mathcal{V}(0) &= \varnothing \\ \mathcal{V}(n+1) &= \mathcal{P}(\mathcal{V}(n)) \qquad \text{successor ordinal } n+1 \\ \mathcal{V}(\omega) &= \bigcup_{n \in \omega} \mathcal{V}(n) \end{aligned}$$

All the Sets (1)

- Curious fact: unfolding $\mathcal{P}$ generates the **hereditarily finite** sets

$$\begin{aligned} \mathcal{V}(0) &= \varnothing \\ \mathcal{V}(n+1) &= \mathcal{P}(\mathcal{V}(n)) \qquad \text{successor ordinal } n+1 \\ \mathcal{V}(\omega) &= \bigcup_{n\in\omega} \mathcal{V}(n) \end{aligned}$$

- Curious fact: ω is also a number: the **first countable ordinal**

$$\begin{aligned} 0 &= \varnothing & \omega &= \{0, 1, 2, ...\} \\ 1 &= \{0\} & \omega+1 &= \{0, 1, 2, ..., \omega\} \\ 2 &= \{0, 1\} & \omega+2 &= \{0, 1, 2, ..., \omega, \omega+1\} \\ 3 &= \{0, 1, 2\} & \omega+\omega &= \{0, 1, 2, ..., \omega, \omega+1, \omega+2, ...\} \end{aligned}$$

All the Sets (1)

- Curious fact: unfolding $\mathcal{P}$ generates the **hereditarily finite** sets

$$\begin{aligned} \mathcal{V}(0) &= \varnothing \\ \mathcal{V}(n+1) &= \mathcal{P}(\mathcal{V}(n)) \qquad \text{successor ordinal } n+1 \\ \mathcal{V}(\omega) &= \bigcup_{n \in \omega} \mathcal{V}(n) \end{aligned}$$

- Curious fact: ω is also a number: the **first countable ordinal**

$$\begin{aligned} 0 &= \varnothing & \omega &= \{0, 1, 2, \ldots\} \\ 1 &= \{0\} & \omega + 1 &= \{0, 1, 2, \ldots, \omega\} \\ 2 &= \{0, 1\} & \omega + 2 &= \{0, 1, 2, \ldots, \omega, \omega + 1\} \\ 3 &= \{0, 1, 2\} & \omega + \omega &= \{0, 1, 2, \ldots, \omega, \omega + 1, \omega + 2, \ldots\} \end{aligned}$$

- Can define more **limit ordinals** $\omega \cdot \omega$, ω^{ω}, $\omega^{\omega^{\cdot^{\cdot^{\cdot}}}}$

All the Sets (2)

- Curious fact: unfolding $\mathcal{P}$ actually generates all the sets

$$
\begin{aligned}
\mathcal{V}(0) &= \varnothing \\
\mathcal{V}(\alpha + 1) &= \mathcal{P}(\mathcal{V}(\alpha)) && \text{successor ordinal } \alpha + 1 \\
\mathcal{V}(\beta) &= \bigcup_{\alpha \in \beta} \mathcal{V}(\alpha) && \text{limit ordinal } \beta
\end{aligned}
$$

  - Every set first appears in some $\mathcal{V}(\alpha)$; e.g. $\mathbb{R} \in \mathcal{V}(\omega + 11)$

All the Sets (2)

- Curious fact: unfolding $\mathcal{P}$ actually generates all the sets

$$\begin{aligned}
\mathcal{V}(0) &= \varnothing \\
\mathcal{V}(\alpha+1) &= \mathcal{P}(\mathcal{V}(\alpha)) && \text{successor ordinal } \alpha+1 \\
\mathcal{V}(\beta) &= \bigcup_{\alpha\in\beta} \mathcal{V}(\alpha) && \text{limit ordinal } \beta
\end{aligned}$$

  - Every set first appears in some $\mathcal{V}(\alpha)$; e.g. $\mathbb{R} \in \mathcal{V}(\omega+11)$
- Can we stop unfolding at some ordinal and have a set of sets that is closed under set primitives?
  - Yes: $\mathcal{V}(\omega)$ is closed under set primitives; it's called a **Grothendieck universe**

All the Sets (2)

- Curious fact: unfolding $\mathcal{P}$ actually generates all the sets

$$\begin{aligned}
\mathcal{V}(0) &= \varnothing \\
\mathcal{V}(\alpha+1) &= \mathcal{P}(\mathcal{V}(\alpha)) && \text{successor ordinal } \alpha+1 \\
\mathcal{V}(\beta) &= \bigcup_{\alpha\in\beta} \mathcal{V}(\alpha) && \text{limit ordinal } \beta
\end{aligned}$$

  - Every set first appears in some $\mathcal{V}(\alpha)$; e.g. $\mathbb{R} \in \mathcal{V}(\omega + 11)$
- Can we stop unfolding at some ordinal and have a set of sets that is closed under set primitives?
  - Yes: $\mathcal{V}(\omega)$ is closed under set primitives; it's called a **Grothendieck universe**
- Is there a Grothendieck universe that contains ω? Undecidable.

Inaccessible Cardinal Axiom

Axiom (inaccessible cardinal). There exists an ordinal κ such that $\mathcal{V}(\kappa)$ contains ω and is closed under $\mathcal{P}$, $\bigcup$, and replacement.

- Call sets in $\mathcal{V}(\kappa)$ **hereditarily accessible** sets

Inaccessible Cardinal Axiom

Axiom (inaccessible cardinal). There exists an ordinal κ such that $\mathcal{V}(\kappa)$ contains ω and is closed under $\mathcal{P}$, $\bigcup$, and replacement.

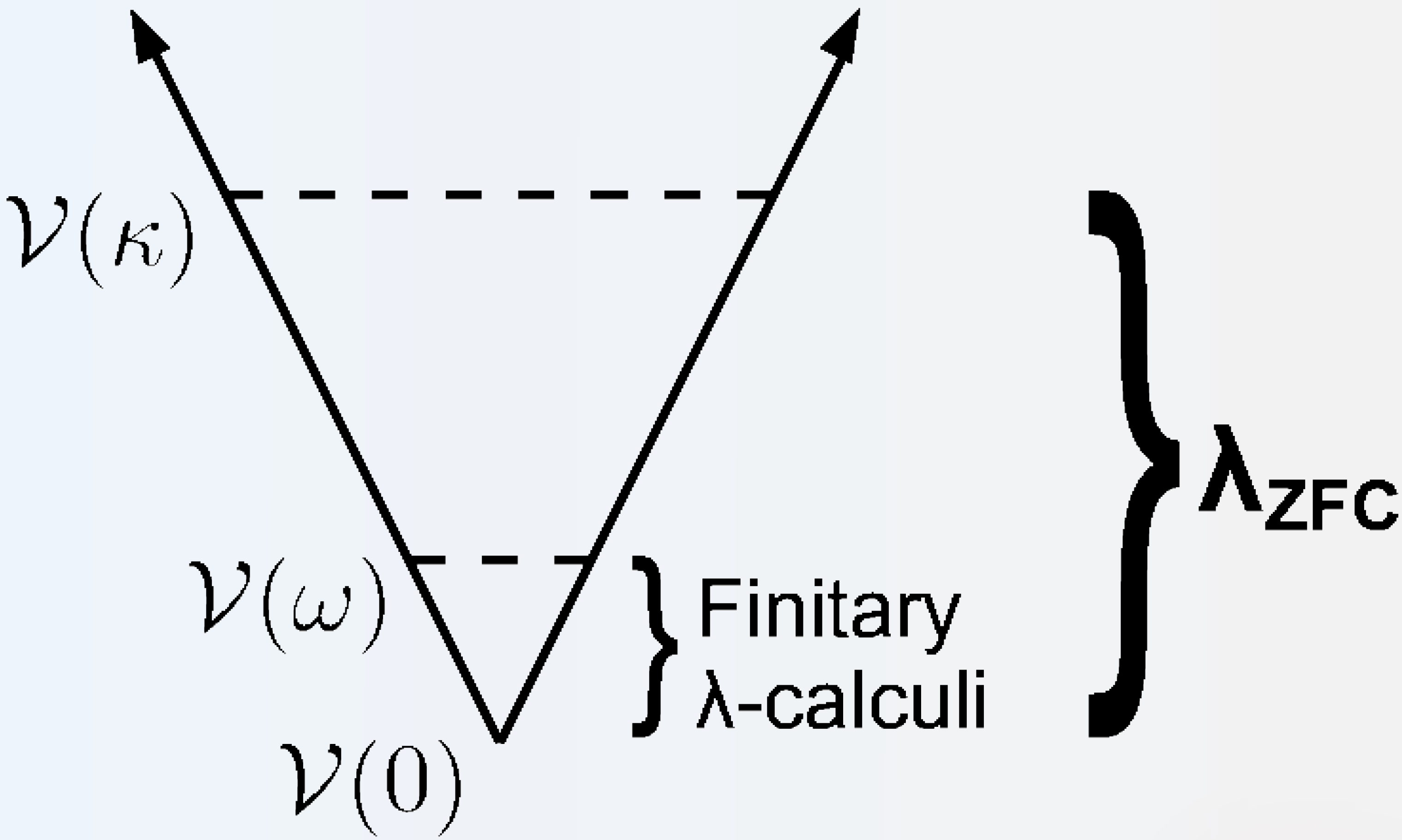
- Call sets in $\mathcal{V}(\kappa)$ **hereditarily accessible** sets
- Uncontroversial extension to ZFC, relatively mild (c.f. HOL, Coq)

Inaccessible Cardinal Axiom

Axiom (inaccessible cardinal). There exists an ordinal κ such that $\mathcal{V}(\kappa)$ contains ω and is closed under $\mathcal{P}$, $\bigcup$, and replacement.

- Call sets in $\mathcal{V}(\kappa)$ **hereditarily accessible** sets
- Uncontroversial extension to ZFC, relatively mild (c.f. HOL, Coq)
- No corresponding $\boldsymbol{\lambda}^{-}_{\mathrm{ZFC}}$ or $\boldsymbol{\lambda}_{\mathrm{ZFC}}$ expression

The Hierarchy of Sets



An Infinite Set Rule For Finite Grammars

New BNF rule: $\{y^{*\alpha}\}$ means "sets of y with less than α elements"

An Infinite Set Rule For Finite Grammars

New BNF rule: $\{y^{*\alpha}\}$ means "sets of y with less than α elements"

- Example: $h ::= \{h^{*\omega}\}$
  - Equivalent to $h ::= \{\} \mid \{h\} \mid \{h, h\} \mid \{h, h, h\} \mid \cdots$
  - Language is $\mathcal{V}(\omega)$ (the hereditarily finite sets)

An Infinite Set Rule For Finite Grammars

New BNF rule: $\{y^{*\alpha}\}$ means "sets of y with less than α elements"

- Example: $h ::= \{h^{*\omega}\}$
  - Equivalent to $h ::= \{\} \mid \{h\} \mid \{h, h\} \mid \{h, h, h\} \mid \cdots$
  - Language is $\mathcal{V}(\omega)$ (the hereditarily finite sets)
- Example: $a ::= \{a^{*\kappa}\}$
  - Language is $\mathcal{V}(\kappa)$ (the hereditarily accessible sets)

An Infinite Set Rule For Finite Grammars

New BNF rule: $\{y^{*\alpha}\}$ means "sets of y with less than α elements"

- Example: $h ::= \{h^{*\omega}\}$
  - Equivalent to $h ::= \{\} \mid \{h\} \mid \{h, h\} \mid \{h, h, h\} \mid \cdots$
  - Language is $\mathcal{V}(\omega)$ (the hereditarily finite sets)
- Example: $a ::= \{a^{*\kappa}\}$
  - Language is $\mathcal{V}(\kappa)$ (the hereditarily accessible sets)
- Example: $v ::= \langle t_{\text{set}}, \{v^{*\kappa}\}\rangle$
  - Language is every set in $\mathcal{V}(\kappa)$, recursively tagged

Finite Grammar of Infinite Terms

- Final $\boldsymbol{\lambda}_{\mathrm{ZFC}}$ grammar:

$$
\begin{aligned}
e \;&::=\; n \mid v \mid \langle t_{\mathrm{app}}, e, e\rangle \mid \langle t_{\mathrm{if}}, e, e, e\rangle \mid \langle t_{\in}, e, e\rangle \mid \langle t_{\cup}, e\rangle \mid \\
&\quad\;\; \langle t_{\mathrm{take}}, e\rangle \mid \langle t_{\mathcal{P}}, e\rangle \mid \langle t_{\mathrm{image}}, e, e\rangle \mid \langle t_{\mathrm{card}}, e\rangle \mid \langle t_{\mathrm{set}}, \{e^{*\kappa}\}\rangle \\
v \;&::=\; \langle t_{\mathrm{atom}}, t_{\mathrm{false}}\rangle \mid \langle t_{\mathrm{atom}}, t_{\mathrm{true}}\rangle \mid \langle t_{\lambda}, e\rangle \mid \langle t_{\mathrm{set}}, \{v^{*\kappa}\}\rangle \\
n \;&::=\; \langle t_{\mathrm{var}}, 0\rangle \mid \langle t_{\mathrm{var}}, 1\rangle \mid \langle t_{\mathrm{var}}, 2\rangle \mid \cdots
\end{aligned}
$$

Finite Grammar of Infinite Terms

- Final $\boldsymbol{\lambda}_{\text{ZFC}}$ grammar:

$$
\begin{aligned}
e \;\; &::= \;\; n \mid v \mid \langle t_{\text{app}}, e, e \rangle \mid \langle t_{\text{if}}, e, e, e \rangle \mid \langle t_{\in}, e, e \rangle \mid \langle t_{\cup}, e \rangle \mid \\
&\quad\;\; \langle t_{\text{take}}, e \rangle \mid \langle t_{\mathcal{P}}, e \rangle \mid \langle t_{\text{image}}, e, e \rangle \mid \langle t_{\text{card}}, e \rangle \mid \langle t_{\text{set}}, \{e^{*\kappa}\} \rangle \\
\boxed{v \;\; ::=} \;\; & \langle t_{\text{atom}}, t_{\text{false}} \rangle \mid \langle t_{\text{atom}}, t_{\text{true}} \rangle \mid \langle t_{\lambda}, e \rangle \mid \boxed{\langle t_{\text{set}}, \{v^{*\kappa}\} \rangle} \\
n \;\; &::= \;\; \langle t_{\text{var}}, 0 \rangle \mid \langle t_{\text{var}}, 1 \rangle \mid \langle t_{\text{var}}, 2 \rangle \mid \cdots
\end{aligned}
$$

- Contains encodings of all the sets in $\mathcal{V}(\kappa)$

Finite Grammar of Infinite Terms

- Final $\boldsymbol{\lambda}_{\text{ZFC}}$ grammar:

$$\begin{aligned}
e &::= n \mid v \mid \langle t_{\text{app}}, e, e\rangle \mid \langle t_{\text{if}}, e, e, e\rangle \mid \langle t_{\in}, e, e\rangle \mid \langle t_{\cup}, e\rangle \mid \\
&\qquad \langle t_{\text{take}}, e\rangle \mid \langle t_{\mathcal{P}}, e\rangle \mid \langle t_{\text{image}}, e, e\rangle \mid \langle t_{\text{card}}, e\rangle \mid \langle t_{\text{set}}, \{e^{*\kappa}\}\rangle \\
v &::= \langle t_{\text{atom}}, t_{\text{false}}\rangle \mid \langle t_{\text{atom}}, t_{\text{true}}\rangle \mid \langle t_{\lambda}, e\rangle \mid \langle t_{\text{set}}, \{v^{*\kappa}\}\rangle \\
n &::= \langle t_{\text{var}}, 0\rangle \mid \langle t_{\text{var}}, 1\rangle \mid \langle t_{\text{var}}, 2\rangle \mid \cdots
\end{aligned}$$

- Contains encodings of all the sets in $\mathcal{V}(\kappa)$
- Computable sublanguage: replace $*\kappa$ with $*\omega$

Finite Grammar of Infinite Terms

- Final $\boldsymbol{\lambda}_{\text{ZFC}}$ grammar:

$$
\begin{array}{rcl}
e & ::= & n \mid v \mid \langle t_{\text{app}}, e, e \rangle \mid \langle t_{\text{if}}, e, e, e \rangle \mid \langle t_{\in}, e, e \rangle \mid \langle t_{\cup}, e \rangle \mid \\
 & & \langle t_{\text{take}}, e \rangle \mid \langle t_{\mathcal{P}}, e \rangle \mid \langle t_{\text{image}}, e, e \rangle \mid \langle t_{\text{card}}, e \rangle \mid \langle t_{\text{set}}, \{e^{*\kappa}\} \rangle \\
v & ::= & \langle t_{\text{atom}}, t_{\text{false}} \rangle \mid \langle t_{\text{atom}}, t_{\text{true}} \rangle \mid \langle t_{\lambda}, e \rangle \mid \langle t_{\text{set}}, \{v^{*\kappa}\} \rangle \\
n & ::= & \langle t_{\text{var}}, 0 \rangle \mid \langle t_{\text{var}}, 1 \rangle \mid \langle t_{\text{var}}, 2 \rangle \mid \cdots
\end{array}
$$

- Contains encodings of all the sets in $\mathcal{V}(\kappa)$
- Computable sublanguage: replace $*\kappa$ with $*\omega$
- Ugly! Write in $\boldsymbol{\lambda}^{-}_{\text{ZFC}}$ with heaps of syntactic sugar; assume transformation to $\boldsymbol{\lambda}_{\text{ZFC}}$ before reduction

Lambda-ZFC's Reduction Semantics

Defines a κ-sized, big-step reduction relation '$\Downarrow$':

$$\frac{}{v \Downarrow v}\ (\text{val}) \qquad \frac{e_f \Downarrow \langle t_\lambda, e_y\rangle \quad e_x \Downarrow v_x \quad e_y[0\backslash v_x] \Downarrow v_y}{\langle t_{\text{app}}, e_f, e_x\rangle \Downarrow v_y}\ (\text{ap}) \qquad \frac{e_c \Downarrow a_{\text{true}} \quad e_t \Downarrow v_t}{\langle t_{\text{if}}, e_c, e_t, e_f\rangle \Downarrow v_t} \quad \frac{e_c \Downarrow a_{\text{false}} \quad e_f \Downarrow v_f}{\langle t_{\text{if}}, e_c, e_t, e_f\rangle \Downarrow v_f}\ (\text{if})$$

$$\frac{e_A \Downarrow v_A \quad V_{\text{set}}(v_A) \quad e_x \Downarrow v_x \quad v_x \in snd(v_A)}{\langle t_\in, e_x, e_A\rangle \Downarrow a_{\text{true}}} \quad \frac{e_A \Downarrow v_A \quad V_{\text{set}}(v_A) \quad e_x \Downarrow v_x \quad v_x \notin snd(v_A)}{\langle t_\in, e_x, e_A\rangle \Downarrow a_{\text{false}}}\ (\text{in})$$

$$\frac{e_A \Downarrow v_A \quad V_{\text{set}}(v_A) \quad \forall v_x \in snd(v_A).\, V_{\text{set}}(v_x)}{\langle t_\cup, e_A\rangle \Downarrow \widehat{\bigcup}(v_A)}\ (\text{union}) \qquad \frac{e_A \Downarrow v_A \quad V_{\text{set}}(v_A)}{\langle t_{\mathcal{P}}, e_A\rangle \Downarrow \widehat{\mathcal{P}}(v_A)}\ (\text{pow})$$

$$\frac{e_A \Downarrow v_A \quad V_{\text{set}}(v_A) \quad e_f \Downarrow \langle t_\lambda, e_y\rangle \quad \widehat{I}(\langle t_\lambda, e_y\rangle, v_A) \Downarrow v_y}{\langle t_{\text{image}}, e_f, e_A\rangle \Downarrow v_y}\ (\text{image}) \qquad \frac{e_A \Downarrow v_A \quad V_{\text{set}}(v_A)}{\langle t_{\text{card}}, e_A\rangle \Downarrow \widehat{C}(v_A)}\ (\text{card})$$

$$\frac{E_{\text{set}}(e_A) \quad \forall e_x \in snd(e_A).\, \exists v_x.\, e_x \Downarrow v_x}{e_A \Downarrow \langle t_{\text{set}}, \{v_x \mid e_x \in snd(e_A) \wedge e_x \Downarrow v_x\}\rangle}\ (\text{set}) \qquad \frac{e_A \Downarrow \langle t_{\text{set}}, \{v_x\}\rangle}{\langle t_{\text{take}}, e_A\rangle \Downarrow v_x}\ (\text{take})$$

Lambda-ZFC's Reduction Semantics

Defines a κ-sized, big-step reduction relation '$\Downarrow$':

$$\frac{}{v \Downarrow v}\ (\text{val}) \quad \frac{e_f \Downarrow \langle t_\lambda, e_y\rangle \quad e_x \Downarrow v_x \quad e_y[0\backslash v_x] \Downarrow v_y}{\langle t_{\text{app}}, e_f, e_x\rangle \Downarrow v_y}\ (\text{ap}) \quad \frac{e_c \Downarrow a_{\text{true}} \quad e_t \Downarrow v_t}{\langle t_{\text{if}}, e_c, e_t, e_f\rangle \Downarrow v_t} \quad \frac{e_c \Downarrow a_{\text{false}} \quad e_f \Downarrow v_f}{\langle t_{\text{if}}, e_c, e_t, e_f\rangle \Downarrow v_f}\ (\text{if})$$

$$\frac{e_A \Downarrow v_A \quad V_{\text{set}}(v_A) \quad e_x \Downarrow v_x \quad v_x \in snd(v_A)}{\langle t_\in, e_x, e_A\rangle \Downarrow a_{\text{true}}} \quad \frac{e_A \Downarrow v_A \quad V_{\text{set}}(v_A) \quad e_x \Downarrow v_x \quad v_x \notin snd(v_A)}{\langle t_\in, e_x, e_A\rangle \Downarrow a_{\text{false}}}\ (\text{in})$$

$$\frac{e_A \Downarrow v_A \quad V_{\text{set}}(v_A) \quad \forall v_x \in snd(v_A).\, V_{\text{set}}(v_x)}{\langle t_\cup, e_A\rangle \Downarrow \widehat{\bigcup}(v_A)}\ (\text{union}) \qquad \frac{e_A \Downarrow v_A \quad V_{\text{set}}(v_A)}{\langle t_{\mathcal{P}}, e_A\rangle \Downarrow \widehat{\mathcal{P}}(v_A)}\ (\text{pow})$$

$$\frac{e_A \Downarrow v_A \quad V_{\text{set}}(v_A) \quad e_f \Downarrow \langle t_\lambda, e_y\rangle \quad \widehat{I}(\langle t_\lambda, e_y\rangle, v_A) \Downarrow v_y}{\langle t_{\text{image}}, e_f, e_A\rangle \Downarrow v_y}\ (\text{image}) \qquad \frac{e_A \Downarrow v_A \quad V_{\text{set}}(v_A)}{\langle t_{\text{card}}, e_A\rangle \Downarrow \widehat{C}(v_A)}\ (\text{card})$$

$$\frac{E_{\text{set}}(e_A) \quad \forall e_x \in snd(e_A).\, \exists v_x.\, e_x \Downarrow v_x}{e_A \Downarrow \langle t_{\text{set}}, \{v_x \mid e_x \in snd(e_A) \wedge e_x \Downarrow v_x\}\rangle}\ (\text{set}) \qquad \frac{e_A \Downarrow \langle t_{\text{set}}, \{v_x\}\rangle}{\langle t_{\text{take}}, e_A\rangle \Downarrow v_x}\ (\text{take})$$

Theorem. **λ**$_{\text{ZFC}}$'s set values and $\langle t_\in, \cdot, \cdot\rangle$ are a model of ZFC-κ.

(i.e. theorems that don't depend on κ are true of **λ**$_{\text{ZFC}}$'s sets)

Expressive Enough?

- Implementable in $\boldsymbol{\lambda}_{\mathrm{ZFC}}$:
  - Bounded logic, pairs, naturals, integers, rationals, arithmetic

Expressive Enough?

- Implementable in $\boldsymbol{\lambda}_{\mathrm{ZFC}}$:
  - Bounded logic, pairs, naturals, integers, rationals, arithmetic
  - Reals and real limits (in the paper)

Expressive Enough?

- Implementable in $\boldsymbol{\lambda}_{\mathrm{ZFC}}$:
  - Bounded logic, pairs, naturals, integers, rationals, arithmetic
  - Reals and real limits (in the paper)
  - Set-theoretic `unfold`, general closure operators, metric universes, limits on metric spaces

Expressive Enough?

- Implementable in $\boldsymbol{\lambda}_{ZFC}$:
  - Bounded logic, pairs, naturals, integers, rationals, arithmetic
  - Reals and real limits (in the paper)
  - Set-theoretic `unfold`, general closure operators, metric universes, limits on metric spaces
  - Measure theory: Borel σ-algebras, arbitrary products of σ-algebras, product measures, Lebesgue measure, Lebesgue integration, conditional probability measures

Example Application: Limit Monad

- Values: language $\mathbb{U}$ of $u ::= \mathbb{R} \mid \omega \to u$

Example Application: Limit Monad

- Values: language $\mathbb{U}$ of $u ::= \mathbb{R} \mid \omega \to u$
- Computations: $\omega \to \mathbb{U}$ (converging sequences of values)

Example Application: Limit Monad

- Values: language $\mathbb{U}$ of $u ::= \mathbb{R} \mid \omega \to u$
- Computations: $\omega \to \mathbb{U}$ (converging sequences of values)
- Run function: $\mathsf{limit} : (\omega \to \mathbb{U}) \to \mathbb{U}$

Example Application: Limit Monad

- Values: language $\mathbb{U}$ of $u ::= \mathbb{R} \mid \omega \to u$
- Computations: $\omega \to \mathbb{U}$ (converging sequences of values)
- Run function: $\mathsf{limit} : (\omega \to \mathbb{U}) \to \mathbb{U}$

Example: $\mathsf{sums} : (\omega \to \mathbb{R}) \to (\omega \to \mathbb{R})$

$$\begin{aligned} \mathsf{sums\ xs} &:= \lambda \mathsf{n.\,if\ (n = 0)\ (xs\ 0)\ ((xs\ n) + (sums\ xs\ (n-1)))} \\ \sum_{\mathsf{n} \in \omega} \mathsf{e} &:\equiv \mathsf{limit\ (sums\ } \lambda \mathsf{n.\,e)} \end{aligned}$$

Example Application: Limit Monad

- Values: language $\mathbb{U}$ of $u ::= \mathbb{R} \mid \omega \to u$
- Computations: $\omega \to \mathbb{U}$ (converging sequences of values)
- Run function: $\mathsf{limit} : (\omega \to \mathbb{U}) \to \mathbb{U}$

Example: $\mathsf{sums} : (\omega \to \mathbb{R}) \to (\omega \to \mathbb{R})$

$$\begin{aligned} \mathsf{sums}\ \mathsf{xs} \ &:= \ \lambda \mathsf{n}.\, \mathsf{if}\ (\mathsf{n} = 0)\ (\mathsf{xs}\ 0)\ ((\mathsf{xs}\ \mathsf{n}) + (\mathsf{sums}\ \mathsf{xs}\ (\mathsf{n} - 1))) \\ \sum_{\mathsf{n} \in \omega} \mathsf{e} \ &:\equiv \ \mathsf{limit}\ (\mathsf{sums}\ \lambda \mathsf{n}.\, \mathsf{e}) \end{aligned}$$

Example: $\mathsf{exp\text{-}seq} : \mathbb{R} \to (\omega \to \mathbb{R})$

$$\begin{aligned} \mathsf{exp\text{-}seq}\ \mathsf{x} \ &:= \ \mathsf{sums}\ \lambda \mathsf{n}.\, \mathsf{x}^{\mathsf{n}} / \mathsf{n}! \\ \mathsf{exp}\ \mathsf{x} \ &:= \ \mathsf{limit}\ (\mathsf{exp\text{-}seq}\ \mathsf{x}) \end{aligned}$$

Uncomputable Limit Monad

- Defining functions:

$$
\begin{aligned}
\mathsf{return}\ \mathsf{x} &:= \lambda \mathsf{n}.\, \mathsf{x} \\
\mathsf{bind}\ \mathsf{xs}\ \mathsf{f} &:= \mathsf{f}\ (\mathsf{limit}\ \mathsf{xs})
\end{aligned}
$$

Uncomputable Limit Monad

- Defining functions:

$$\begin{aligned} \mathsf{return}\ \mathsf{x} &:= \lambda \mathsf{n}.\, \mathsf{x} \\ \mathsf{bind}\ \mathsf{xs}\ \mathsf{f} &:= \mathsf{f}\ (\mathsf{limit}\ \mathsf{xs}) \end{aligned}$$

- Example: $\mathsf{exp}_{\mathsf{lim}} : (\omega \to \mathbb{R}) \to (\omega \to \mathbb{R})$

$$\mathsf{exp}_{\mathsf{lim}}\ \mathsf{xs} := \mathsf{bind}\ \mathsf{xs}\ \mathsf{exp\text{-}seq}$$

Uncomputable Limit Monad

- Defining functions:

$$\begin{aligned} \mathsf{return}\ \mathsf{x} &:= \lambda \mathsf{n}.\, \mathsf{x} \\ \mathsf{bind}\ \mathsf{xs}\ \mathsf{f} &:= \mathsf{f}\ (\mathsf{limit}\ \mathsf{xs}) \end{aligned}$$

- Example: $\mathsf{exp}_{\mathsf{lim}} : (\omega \to \mathbb{R}) \to (\omega \to \mathbb{R})$

$$\mathsf{exp}_{\mathsf{lim}}\ \mathsf{xs} := \mathsf{bind}\ \mathsf{xs}\ \mathsf{exp\text{-}seq}$$

- Wish for: limit-free, drop-in replacement for bind
    - Reality says: Sorry Bucko, only under limited conditions

Uncomputable Limit Monad

- Defining functions:

$$\begin{aligned} \mathsf{return}\ x &:= \lambda n.\, x \\ \mathsf{bind}\ xs\ f &:= f\ (\mathsf{limit}\ xs) \end{aligned}$$

- Example: $\exp_{\mathsf{lim}} : (\omega \to \mathbb{R}) \to (\omega \to \mathbb{R})$

$$\exp_{\mathsf{lim}}\ xs := \mathsf{bind}\ xs\ \mathsf{exp\text{-}seq}$$

- Wish for: limit-free, drop-in replacement for bind
  - Reality says: Sorry Bucko, only under limited conditions
- Step 1: Factor using monad identities and topological theorems

$$\begin{aligned} \mathsf{bind}\ xs\ f &= \mathsf{join}\ (\mathsf{lift}\ f\ xs) \\ \mathsf{lift}\ f\ xs &:= \mathsf{return}\ (f\ (\mathsf{limit}\ xs)) \\ \mathsf{join}\ xss &:= \lambda n.\, \mathsf{limit}\ (\mathsf{flip}\ xss\ n) \end{aligned}$$

Computable Limit Monad

- Step 2: Collapse limits using topological theorems

Identity	**Condition (per-instance)**
$\mathsf{limit}\ (\mathsf{lift}\ \mathsf{f}\ \mathsf{xs}) = \mathsf{limit}\ (\mathsf{f} \circ \mathsf{xs})$	Continuity

Computable Limit Monad

- Step 2: Collapse limits using topological theorems

Identity	Condition (per-instance)
$\mathsf{limit}\ (\mathsf{lift}\ \mathsf{f}\ \mathsf{xs}) = \mathsf{limit}\ (\mathsf{f} \circ \mathsf{xs})$	Continuity
$\mathsf{limit}\ (\mathsf{join}\ \mathsf{xss}) = \mathsf{limit}\ (\lambda \mathsf{n}.\, \mathsf{xss}\ \mathsf{n}\ \mathsf{n})$	Uniform Convergence

Computable Limit Monad

- Step 2: Collapse limits using topological theorems

Identity	Condition (per-instance)
$\mathsf{limit}\ (\mathsf{lift}\ \mathsf{f}\ \mathsf{xs}) = \mathsf{limit}\ (\mathsf{f} \circ \mathsf{xs})$	Continuity
$\mathsf{limit}\ (\mathsf{join}\ \mathsf{xss}) = \mathsf{limit}\ (\lambda \mathsf{n}.\, \mathsf{xss}\ \mathsf{n}\ \mathsf{n})$	Uniform Convergence

- Limit-free replacement for bind:

$$
\begin{aligned}
\mathsf{lift}'\ \mathsf{f}\ \mathsf{xs} &:= \mathsf{f} \circ \mathsf{xs} \\
\mathsf{join}'\ \mathsf{xss} &:= \lambda \mathsf{n}.\, \mathsf{xss}\ \mathsf{n}\ \mathsf{n} \\
\mathsf{bind}'\ \mathsf{xs}\ \mathsf{f} &:= \mathsf{join}'\ (\mathsf{lift}'\ \mathsf{f}\ \mathsf{xs})
\end{aligned}
$$

Computable Limit Monad

- Step 2: Collapse limits using topological theorems

Identity	Condition (per-instance)
$\mathsf{limit}\ (\mathsf{lift}\ \mathsf{f}\ \mathsf{xs}) = \mathsf{limit}\ (\mathsf{f} \circ \mathsf{xs})$	Continuity
$\mathsf{limit}\ (\mathsf{join}\ \mathsf{xss}) = \mathsf{limit}\ (\lambda \mathsf{n}.\ \mathsf{xss}\ \mathsf{n}\ \mathsf{n})$	Uniform Convergence

- Limit-free replacement for bind:

$$\begin{aligned} \mathsf{lift}'\ \mathsf{f}\ \mathsf{xs} &:= \mathsf{f} \circ \mathsf{xs} \\ \mathsf{join}'\ \mathsf{xss} &:= \lambda \mathsf{n}.\ \mathsf{xss}\ \mathsf{n}\ \mathsf{n} \\ \mathsf{bind}'\ \mathsf{xs}\ \mathsf{f} &:= \mathsf{join}'\ (\mathsf{lift}'\ \mathsf{f}\ \mathsf{xs}) \end{aligned}$$

- Prove conditions for use of bind in $\mathsf{exp}_{\mathsf{lim}}$; then redefine

$$\mathsf{exp}_{\mathsf{lim}}\ \mathsf{xs} := \mathsf{bind}'\ \mathsf{xs}\ \mathsf{exp\text{-}seq}$$

Limit Monad Example: Computing π

Machin's formula (1706): $\pi = 16\tan^{-1}\left(\frac{1}{5}\right) - 4\tan^{-1}\left(\frac{1}{239}\right)$

Limit Monad Example: Computing π

Machin's formula (1706): $\pi = 16\tan^{-1}\left(\frac{1}{5}\right) - 4\tan^{-1}\left(\frac{1}{239}\right)$

λ$_{\text{ZFC}}$

$$\mathsf{atan\text{-}seq}\ \mathsf{y} := \\ \quad \mathsf{sums}\ \lambda\mathsf{n}.\ \frac{-1^{\mathsf{n}} \times \mathsf{y}^{2\times\mathsf{n}+1}}{2\times\mathsf{n}+1}$$

Limit Monad Example: Computing π

Machin's formula (1706): $\pi = 16 \tan^{-1}\left(\frac{1}{5}\right) - 4 \tan^{-1}\left(\frac{1}{239}\right)$

λ_{ZFC}

$$\mathsf{atan\text{-}seq}\ \mathsf{y} := \\ \quad \mathsf{sums}\ \lambda\,\mathsf{n}.\ \frac{-1^{\mathsf{n}} \times \mathsf{y}^{2\times\mathsf{n}+1}}{2\times\mathsf{n}+1}$$

Racket

```
(define (atan-seq y)
  (sums (λ (n) (/ (* (expt -1 n)
                     (expt y (+ (* n 2) 1)))
                  (+ (* n 2) 1)))))
```

Limit Monad Example: Computing π

Machin's formula (1706): $\pi = 16 \tan^{-1}\left(\frac{1}{5}\right) - 4 \tan^{-1}\left(\frac{1}{239}\right)$

λ_{ZFC}

$$\mathsf{atan\text{-}seq}\ \mathsf{y} := \\ \quad \mathsf{sums}\ \lambda \mathsf{n}.\ \frac{-1^{\mathsf{n}} \times \mathsf{y}^{2\times\mathsf{n}+1}}{2 \times \mathsf{n} + 1}$$

$$\mathsf{atan}_{\mathsf{lim}}\ \mathsf{ys} := \\ \quad \mathsf{bind}\ \mathsf{ys}\ \mathsf{atan\text{-}seq}$$

Racket

```
(define (atan-seq y)
  (sums (λ (n) (/ (* (expt -1 n)
                     (expt y (+ (* n 2) 1)))
                  (+ (* n 2) 1)))))
```

Limit Monad Example: Computing π

Machin's formula (1706): $\pi = 16\tan^{-1}\left(\frac{1}{5}\right) - 4\tan^{-1}\left(\frac{1}{239}\right)$

λ_{ZFC}

atan-seq y :=
sums λn. $\dfrac{-1^{n} \times y^{2\times n+1}}{2\times n+1}$

atan$_{\text{lim}}$ ys :=
bind ys atan-seq

Proof

Continuity
Unif. conv.

Racket

```
(define (atan-seq y)
  (sums (λ (n) (/ (* (expt -1 n)
                     (expt y (+ (* n 2) 1)))
                  (+ (* n 2) 1)))))
```

Limit Monad Example: Computing π

Machin's formula (1706): $\pi = 16 \tan^{-1}\left(\frac{1}{5}\right) - 4 \tan^{-1}\left(\frac{1}{239}\right)$

λ_{ZFC}

$\mathsf{atan\text{-}seq}\ \mathsf{y} :=$

$$\mathsf{sums}\ \lambda\,\mathsf{n}.\ \frac{-1^{\mathsf{n}} \times \mathsf{y}^{2\times\mathsf{n}+1}}{2\times\mathsf{n}+1}$$

$\mathsf{atan}_{\mathsf{lim}}\ \mathsf{ys} :=$
$\quad\mathsf{bind}\ \mathsf{ys}\ \mathsf{atan\text{-}seq}$

$\mathsf{atan}_{\mathsf{lim}}\ \mathsf{ys} :=$
$\quad\mathsf{bind}'\ \mathsf{ys}\ \mathsf{atan\text{-}seq}$

Proof

Continuity
Unif. conv.

Racket

```
(define (atan-seq y)
  (sums (λ (n) (/ (* (expt -1 n)
                     (expt y (+ (* n 2) 1)))
                  (+ (* n 2) 1)))))
```

Limit Monad Example: Computing π

Machin's formula (1706): $\pi = 16 \tan^{-1}\left(\frac{1}{5}\right) - 4 \tan^{-1}\left(\frac{1}{239}\right)$

λ_{ZFC}	Proof	Racket
atan-seq y := sums λn. $\dfrac{-1^{n} \times y^{2\times n+1}}{2 \times n + 1}$		`(define (atan-seq y) (sums (λ (n) (/ (* (expt -1 n) (expt y (+ (* n 2) 1))) (+ (* n 2) 1)))))`
$\mathrm{atan}_{\mathrm{lim}}$ ys := bind ys atan-seq	Continuity Unif. conv.	
$\mathrm{atan}_{\mathrm{lim}}$ ys := bind′ ys atan-seq		`(define (atan-lim ys) (bind ys atan-seq))`

Limit Monad Example: Computing π

Machin's formula (1706): $\pi = 16\tan^{-1}\left(\frac{1}{5}\right) - 4\tan^{-1}\left(\frac{1}{239}\right)$

λ_{ZFC}

atan-seq y :=

$$\text{sums } \lambda n.\ \frac{-1^n \times y^{2\times n+1}}{2\times n+1}$$

Racket

```
(define (atan-seq y)
  (sums (λ (n) (/ (* (expt -1 n)
                     (expt y (+ (* n 2) 1)))
                  (+ (* n 2) 1)))))
```

λ_{ZFC}

atan_{lim} ys :=
bind ys atan-seq

Proof

Continuity
Unif. conv.

λ_{ZFC}

atan_{lim} ys :=
bind′ ys atan-seq

Racket

```
(define (atan-lim ys)
  (bind ys atan-seq))
```

λ_{ZFC}

π_{lim} :=

$$(r\ 16) \times_{\text{lim}} (\text{atan}_{\text{lim}}\ (r\ \tfrac{1}{5})) -_{\text{lim}}$$
$$(r\ 4) \times_{\text{lim}} (\text{atan}_{\text{lim}}\ (r\ \tfrac{1}{239}))$$

Limit Monad Example: Computing π

Machin's formula (1706): $\pi = 16\tan^{-1}\left(\frac{1}{5}\right) - 4\tan^{-1}\left(\frac{1}{239}\right)$

λ_{ZFC}	**Proof**	**Racket**

atan-seq y :=
 sums λn. $\dfrac{-1^{n} \times y^{2\times n+1}}{2\times n+1}$
		`(define (atan-seq y)`
		`  (sums (λ (n) (/ (* (expt -1 n)`
		`                    (expt y (+ (* n 2) 1)))`
		`                 (+ (* n 2) 1)))))`

atan_{lim} ys := bind ys atan-seq	Continuity Unif. conv.	
atan_{lim} ys := bind′ ys atan-seq		`(define (atan-lim ys)` `  (bind ys atan-seq))`
π_{lim} := $(r\ 16) \times_{\text{lim}} (\text{atan}_{\text{lim}}\ (r\ \frac{1}{5})) -_{\text{lim}}$ $(r\ 4) \times_{\text{lim}} (\text{atan}_{\text{lim}}\ (r\ \frac{1}{239}))$	*None*	

Limit Monad Example: Computing π

Machin's formula (1706): $\pi = 16 \tan^{-1}\left(\frac{1}{5}\right) - 4 \tan^{-1}\left(\frac{1}{239}\right)$

λ_{ZFC}	**Proof**	**Racket**

atan-seq y :=
 sums $\lambda n. \dfrac{-1^n \times y^{2\times n+1}}{2 \times n + 1}$
		`(define (atan-seq y)`
		`  (sums (λ (n) (/ (* (expt -1 n)`
		`                    (expt y (+ (* n 2) 1)))`
		`                 (+ (* n 2) 1)))))`

atan_{lim} ys := bind ys atan-seq	Continuity Unif. conv.	
atan_{lim} ys := bind′ ys atan-seq		`(define (atan-lim ys)` `  (bind ys atan-seq))`

π_{lim} :=
 $(\text{r}\ 16) \times_{\text{lim}} (\text{atan}_{\text{lim}}\ (\text{r}\ \frac{1}{5})) -_{\text{lim}}$
 $(\text{r}\ 4) \times_{\text{lim}} (\text{atan}_{\text{lim}}\ (\text{r}\ \frac{1}{239}))$
	None	`(define pi-lim`
		`  (-lim (*lim (r 16) (atan-lim (r 1/5)))`
		`        (*lim (r 4) (atan-lim (r 1/239)))))`

Limit Monad Example: Computing π

Machin's formula (1706): $\pi = 16 \tan^{-1}\left(\frac{1}{5}\right) - 4 \tan^{-1}\left(\frac{1}{239}\right)$

λ_{ZFC}	Proof	Racket

$\text{atan-seq } y :=$
 $\quad \text{sums } \lambda n. \dfrac{-1^n \times y^{2\times n+1}}{2\times n+1}$
		`(define (atan-seq y)`
		`  (sums (λ (n) (/ (* (expt -1 n)`
		`                    (expt y (+ (* n 2) 1)))`
		`                 (+ (* n 2) 1)))))`

$\text{atan}_{\text{lim}} \text{ ys} :=$ $\quad \text{bind ys atan-seq}$	Continuity Unif. conv.	
$\text{atan}_{\text{lim}} \text{ ys} :=$ $\quad \text{bind}' \text{ ys atan-seq}$		`(define (atan-lim ys)` `  (bind ys atan-seq))`

$\pi_{\text{lim}} :=$
 $\quad (\text{r } 16) \times_{\text{lim}} (\text{atan}_{\text{lim}} (\text{r } \frac{1}{5})) -_{\text{lim}}$
 $\quad (\text{r } 4) \times_{\text{lim}} (\text{atan}_{\text{lim}} (\text{r } \frac{1}{239}))$
	None	`(define pi-lim`
		`  (-lim (*lim (r 16) (atan-lim (r 1/5)))`
		`        (*lim (r 4) (atan-lim (r 1/239)))))`

```
> (real->decimal-string (time (pi-lim 141)) 200)
cpu time: 10 real time: 18 gc time: 0
"3.14159265358979323846264338327950288419716939937510 58[...]"
```

Conclusions and Future Work

- We did awesome things
  - Defined $\boldsymbol{\lambda}_{\mathrm{ZFC}}$, which can express anything "constructive"; proved almost all theorems apply directly to $\boldsymbol{\lambda}_{\mathrm{ZFC}}$ terms
  - Defined the limit monad, defined π in it, derived an implementation, transliterated it into Racket

Conclusions and Future Work

- We did awesome things
  - Defined $\boldsymbol{\lambda}_{ZFC}$, which can express anything “constructive”; proved almost all theorems apply directly to $\boldsymbol{\lambda}_{ZFC}$ terms
  - Defined the limit monad, defined π in it, derived an implementation, transliterated it into Racket
- We will do more awesome things

Conclusions and Future Work

- We did awesome things
  - Defined $\boldsymbol{\lambda}_{\mathrm{ZFC}}$, which can express anything "constructive"; proved almost all theorems apply directly to $\boldsymbol{\lambda}_{\mathrm{ZFC}}$ terms
  - Defined the limit monad, defined π in it, derived an implementation, transliterated it into Racket
- We will do more awesome things
- Bonus Questions!
  - How much deep set theory do I need to know to use $\boldsymbol{\lambda}_{\mathrm{ZFC}}$?
  - Can proofs about $\boldsymbol{\lambda}_{\mathrm{ZFC}}$ terms be used in contemporary math?
  - How would one make a call-by-name version of $\boldsymbol{\lambda}_{\mathrm{ZFC}}$?
  - Why is $\mathbb{R} \in \mathcal{V}(\omega + 11)$?