Constructive Analysis in Nuprl

Mark Bickford

Cornell University, Computer Science

September 29, 2017

Lecture 1

Lecture 1

- Introduction to Nuprl

Lecture 1

- Introduction to Nuprl
- The Real Numbers

Lecture 1

- Introduction to Nuprl
- The Real Numbers
  - Definition

Lecture 1

- Introduction to Nuprl
- The Real Numbers
  - Definition
  - Equivalence

Lecture 1

- Introduction to Nuprl
- The Real Numbers
  - Definition
  - Equivalence
  - Acceleration

Lecture 1

- Introduction to Nuprl
- The Real Numbers
  - Definition
  - Equivalence
  - Acceleration
  - Arithmetic and Ordering

Lecture 1

- Introduction to Nuprl
- The Real Numbers
  - Definition
  - Equivalence
  - Acceleration
  - Arithmetic and Ordering
- The Continuity principle

Lecture 1

- Introduction to Nuprl
- The Real Numbers
  - Definition
  - Equivalence
  - Acceleration
  - Arithmetic and Ordering
- The Continuity principle
  - A false continuity principle

Lecture 1

- Introduction to Nuprl
- The Real Numbers
  - Definition
  - Equivalence
  - Acceleration
  - Arithmetic and Ordering
- The Continuity principle
  - A false continuity principle
  - Function, Product, Quotient, and Equality types

Lecture 1

- Introduction to Nuprl
- The Real Numbers
  - Definition
  - Equivalence
  - Acceleration
  - Arithmetic and Ordering
- The Continuity principle
  - A false continuity principle
  - Function, Product, Quotient, and Equality types
  - A true continuity principle

Lecture 2

Lecture 2

- Theorems derivable from continuity

Lecture 2

- Theorems derivable from continuity
  - Weak Markov Principle

Lecture 2

- Theorems derivable from continuity
  - Weak Markov Principle
  - Other derived continuity principles

Lecture 2

- Theorems derivable from continuity
  - Weak Markov Principle
  - Other derived continuity principles
  - Connectedness of $\mathbb{R}$

Lecture 3

Lecture 3

- Bar Induction

Lecture 3

- Bar Induction
  - Statement of BID, Realizer for BID (bar recursion)

Lecture 3

- Bar Induction
  - Statement of BID, Realizer for BID (bar recursion)
  - Fan Theorem

Lecture 3

- Bar Induction
  - Statement of BID, Realizer for BID (bar recursion)
  - Fan Theorem
  - Kleene's singular tree

Lecture 3

- Bar Induction
  - Statement of BID, Realizer for BID (bar recursion)
  - Fan Theorem
  - Kleene's singular tree
  - Soundness of Bar Induction

Lecture 3

- Bar Induction
  - Statement of BID, Realizer for BID (bar recursion)
  - Fan Theorem
  - Kleene's singular tree
  - Soundness of Bar Induction
- Brouwer's uniform continuity theorem

Lecture 3

- Bar Induction
  - Statement of BID, Realizer for BID (bar recursion)
  - Fan Theorem
  - Kleene's singular tree
  - Soundness of Bar Induction
- Brouwer's uniform continuity theorem
  - Uniform continuity from Fan

Lecture 3

- Bar Induction
  - Statement of BID, Realizer for BID (bar recursion)
  - Fan Theorem
  - Kleene's singular tree
  - Soundness of Bar Induction
- Brouwer's uniform continuity theorem
  - Uniform continuity from Fan
  - Uniform continuity for real functions

Lecture 3

- Bar Induction
  - Statement of BID, Realizer for BID (bar recursion)
  - Fan Theorem
  - Kleene's singular tree
  - Soundness of Bar Induction
- Brouwer's uniform continuity theorem
  - Uniform continuity from Fan
  - Uniform continuity for real functions
- Consequences of Brouwer's theorem

Lecture 3

- Bar Induction
  - Statement of BID, Realizer for BID (bar recursion)
  - Fan Theorem
  - Kleene's singular tree
  - Soundness of Bar Induction
- Brouwer's uniform continuity theorem
  - Uniform continuity from Fan
  - Uniform continuity for real functions
- Consequences of Brouwer's theorem
  - Simplification of formal theory

Lecture 3

- Bar Induction
  - Statement of BID, Realizer for BID (bar recursion)
  - Fan Theorem
  - Kleene's singular tree
  - Soundness of Bar Induction
- Brouwer's uniform continuity theorem
  - Uniform continuity from Fan
  - Uniform continuity for real functions
- Consequences of Brouwer's theorem
  - Simplification of formal theory
  - Two functional equations

Introduction

Introduction

- Nuprl implements *constructive type theory* (CTT) and has a *continuity principle* (CONT) and an induction rule, *bar induction for decidable bars* (BID).

Introduction

- Nuprl implements *constructive type theory* (CTT) and has a *continuity principle* (CONT) and an induction rule, *bar induction for decidable bars* (BID).
- So Nuprl can do proofs in Intuitionistic math.

Introduction

- Nuprl implements *constructive type theory* (CTT) and has a *continuity principle* (CONT) and an induction rule, *bar induction for decidable bars* (BID).
- So Nuprl can do proofs in Intuitionistic math.
- We will discuss the continuity principle and bar induction and show how these intuitionistic principles allow us to prove theorems such as Brouwer's theorem that every function $f : [a, b] \to \mathbb{R}$ is uniformly continuous.

Introduction

- Nuprl implements *constructive type theory* (CTT) and has a *continuity principle* (CONT) and an induction rule, *bar induction for decidable bars* (BID).
- So Nuprl can do proofs in Intuitionistic math.
- We will discuss the continuity principle and bar induction and show how these intuitionistic principles allow us to prove theorems such as Brouwer's theorem that every function $f : [a, b] \to \mathbb{R}$ is uniformly continuous.
- This theorem, and other theorems that follow from CONT, allow us to simplify the development of analysis in Nuprl.

Introduction

- Nuprl implements *constructive type theory* (CTT) and has a *continuity principle* (CONT) and an induction rule, *bar induction for decidable bars* (BID).
- So Nuprl can do proofs in Intuitionistic math.
- We will discuss the continuity principle and bar induction and show how these intuitionistic principles allow us to prove theorems such as Brouwer's theorem that every function $f : [a, b] \to \mathbb{R}$ is uniformly continuous.
- This theorem, and other theorems that follow from CONT, allow us to simplify the development of analysis in Nuprl.
- All the results were formally proved in Nuprl.

Introduction

- Nuprl implements *constructive type theory* (CTT) and has a *continuity principle* (CONT) and an induction rule, *bar induction for decidable bars* (BID).
- So Nuprl can do proofs in Intuitionistic math.
- We will discuss the continuity principle and bar induction and show how these intuitionistic principles allow us to prove theorems such as Brouwer's theorem that every function $f : [a, b] \to \mathbb{R}$ is uniformly continuous.
- This theorem, and other theorems that follow from CONT, allow us to simplify the development of analysis in Nuprl.
- All the results were formally proved in Nuprl.
- One theorem (strong connectedness of $\mathbb{R}$) is new.

Semantics of Nuprl

Semantics of Nuprl

- primitive collection of *terms*

Semantics of Nuprl

- primitive collection of *terms*
- computation relation on the terms, $t_1 \mapsto t_2$

Semantics of Nuprl

- primitive collection of *terms*
- computation relation on the terms, $t_1 \mapsto t_2$
- derived *computational bi-simulation relation*, $t_1 \sim t_2$

Semantics of Nuprl

- primitive collection of *terms*
- computation relation on the terms, $t_1 \mapsto t_2$
- derived *computational bi-simulation relation*, $t_1 \sim t_2$
- Types are *partial equivalence relations* (PERs) on terms

Semantics of Nuprl

- primitive collection of *terms*
- computation relation on the terms, $t_1 \mapsto t_2$
- derived *computational bi-simulation relation*, $t_1 \sim t_2$
- Types are *partial equivalence relations* (PERs) on terms
  - PER for a type T must respect $\sim$

Semantics of Nuprl

- primitive collection of *terms*
- computation relation on the terms, $t_1 \mapsto t_2$
- derived *computational bi-simulation relation*, $t_1 \sim t_2$
- Types are *partial equivalence relations* (PERs) on terms
  - PER for a type T must respect $\sim$
  - written $t_1 = t_2 \in T$

Semantics of Nuprl

- primitive collection of *terms*
- computation relation on the terms, $t_1 \mapsto t_2$
- derived *computational bi-simulation relation*, $t_1 \sim t_2$
- Types are *partial equivalence relations* (PERs) on terms
  - PER for a type T must respect $\sim$
  - written $t_1 = t_2 \in T$
  - Bishop: to define a set T we must say what its members are and say when two members represent the same element of the set.

Semantics of Nuprl

- primitive collection of *terms*
- computation relation on the terms, $t_1 \mapsto t_2$
- derived *computational bi-simulation relation*, $t_1 \sim t_2$
- Types are *partial equivalence relations* (PERs) on terms
  - PER for a type T must respect $\sim$
  - written $t_1 = t_2 \in T$
  - Bishop: to define a set T we must say what its members are and say when two members represent the same element of the set.
- Universes $\mathbb{U}_i$ closed under type constructors

Semantics of Nuprl

- primitive collection of *terms*
- computation relation on the terms, $t_1 \mapsto t_2$
- derived *computational bi-simulation relation*, $t_1 \sim t_2$
- Types are *partial equivalence relations* (PERs) on terms
  - PER for a type T must respect $\sim$
  - written $t_1 = t_2 \in T$
  - Bishop: to define a set T we must say what its members are and say when two members represent the same element of the set.
- Universes $\mathbb{U}_i$ closed under type constructors
  - $\mathbb{Z}$, Base, Atom

Semantics of Nuprl

- primitive collection of *terms*
- computation relation on the terms, $t_1 \mapsto t_2$
- derived *computational bi-simulation relation*, $t_1 \sim t_2$
- Types are *partial equivalence relations* (PERs) on terms
  - PER for a type T must respect $\sim$
  - written $t_1 = t_2 \in T$
  - Bishop: to define a set T we must say what its members are and say when two members represent the same element of the set.
- Universes $\mathbb{U}_i$ closed under type constructors
  - $\mathbb{Z}$, Base, Atom
  - $a{:}A \to B[a]$, $a{:}A \times B[a]$, $A + B$

Semantics of Nuprl

- primitive collection of *terms*
- computation relation on the terms, $t_1 \mapsto t_2$
- derived *computational bi-simulation relation*, $t_1 \sim t_2$
- Types are *partial equivalence relations* (PERs) on terms
  - PER for a type T must respect $\sim$
  - written $t_1 = t_2 \in T$
  - Bishop: to define a set T we must say what its members are and say when two members represent the same element of the set.
- Universes $\mathbb{U}_i$ closed under type constructors
  - $\mathbb{Z}$, Base, Atom
  - $a{:}A \to B[a]$, $a{:}A \times B[a]$, $A + B$
  - $\bigcap_{a:A} B[a]$, $\{a{:}A \mid B[a]\}$, $a{:}A \cap B[a]$

Semantics of Nuprl

- primitive collection of *terms*
- computation relation on the terms, $t_1 \mapsto t_2$
- derived *computational bi-simulation relation*, $t_1 \sim t_2$
- Types are *partial equivalence relations* (PERs) on terms
  - PER for a type T must respect $\sim$
  - written $t_1 = t_2 \in T$
  - Bishop: to define a set T we must say what its members are and say when two members represent the same element of the set.
- Universes $\mathbb{U}_i$ closed under type constructors
  - $\mathbb{Z}$, Base, Atom
  - $a{:}A \to B[a]$, $a{:}A \times B[a]$, $A + B$
  - $\bigcap_{a:A} B[a]$, $\{a{:}A \mid B[a]\}$, $a{:}A \cap B[a]$
  - pertype$(x, y.R(x, y))$

Semantics of Nuprl

- primitive collection of *terms*
- computation relation on the terms, $t_1 \mapsto t_2$
- derived *computational bi-simulation relation*, $t_1 \sim t_2$
- Types are *partial equivalence relations* (PERs) on terms
  - PER for a type T must respect $\sim$
  - written $t_1 = t_2 \in T$
  - Bishop: to define a set T we must say what its members are and say when two members represent the same element of the set.
- Universes $\mathbb{U}_i$ closed under type constructors
  - $\mathbb{Z}$, Base, Atom
  - $a{:}A \to B[a]$, $a{:}A \times B[a]$, $A + B$
  - $\bigcap_{a:A} B[a]$, $\{a{:}A \mid B[a]\}$, $a{:}A \cap B[a]$
  - pertype$(x, y.R(x, y)$
    - when $R(x, y)$ is a PER on Base

Semantics of Nuprl

- primitive collection of *terms*
- computation relation on the terms, $t_1 \mapsto t_2$
- derived *computational bi-simulation relation*, $t_1 \sim t_2$
- Types are *partial equivalence relations* (PERs) on terms
  - PER for a type T must respect $\sim$
  - written $t_1 = t_2 \in T$
  - Bishop: to define a set T we must say what its members are and say when two members represent the same element of the set.
- Universes $\mathbb{U}_i$ closed under type constructors
  - $\mathbb{Z}$, Base, Atom
  - $a{:}A \to B[a]$, $a{:}A \times B[a]$, $A + B$
  - $\bigcap_{a:A} B[a]$, $\{a{:}A \mid B[a]\}$, $a{:}A \cap B[a]$
  - pertype$(x, y.R(x, y)$
    - when $R(x, y)$ is a PER on Base
    - Base is the type corresponding to $\sim$

Semantics of Nuprl

- primitive collection of *terms*
- computation relation on the terms, $t_1 \mapsto t_2$
- derived *computational bi-simulation relation*, $t_1 \sim t_2$
- Types are *partial equivalence relations* (PERs) on terms
  - PER for a type T must respect $\sim$
  - written $t_1 = t_2 \in T$
  - Bishop: to define a set T we must say what its members are and say when two members represent the same element of the set.
- Universes $\mathbb{U}_i$ closed under type constructors
  - $\mathbb{Z}$, Base, Atom
  - $a{:}A \to B[a]$, $a{:}A \times B[a]$, $A + B$
  - $\bigcap_{a:A} B[a]$, $\{a{:}A \mid B[a]\}$, $a{:}A \cap B[a]$
  - pertype$(x, y.R(x, y)$
    - when $R(x, y)$ is a PER on Base
    - Base is the type corresponding to $\sim$
    - Quotient type $T//E$ is a special case

Semantics of Nuprl

- primitive collection of *terms*
- computation relation on the terms, $t_1 \mapsto t_2$
- derived *computational bi-simulation relation*, $t_1 \sim t_2$
- Types are *partial equivalence relations* (PERs) on terms
  - PER for a type T must respect $\sim$
  - written $t_1 = t_2 \in T$
  - Bishop: to define a set T we must say what its members are and say when two members represent the same element of the set.
- Universes $\mathbb{U}_i$ closed under type constructors
  - $\mathbb{Z}$, Base, Atom
  - $a{:}A \to B[a]$, $a{:}A \times B[a]$, $A + B$
  - $\bigcap_{a:A} B[a]$, $\{a{:}A \mid B[a]\}$, $a{:}A \cap B[a]$
  - pertype$(x, y.R(x, y)$
    - when $R(x, y)$ is a PER on Base
    - Base is the type corresponding to $\sim$
    - Quotient type $T//E$ is a special case
- Extensional $(x \in A \ \wedge A = B \in \mathbb{U} \Rightarrow x \in B)$

Semantics of Nuprl

- primitive collection of *terms*
- computation relation on the terms, $t_1 \mapsto t_2$
- derived *computational bi-simulation relation*, $t_1 \sim t_2$
- Types are *partial equivalence relations* (PERs) on terms
  - PER for a type T must respect $\sim$
  - written $t_1 = t_2 \in T$
  - Bishop: to define a set T we must say what its members are and say when two members represent the same element of the set.
- Universes $\mathbb{U}_i$ closed under type constructors
  - $\mathbb{Z}$, Base, Atom
  - $a{:}A \to B[a]$, $a{:}A \times B[a]$, $A + B$
  - $\bigcap_{a:A} B[a]$, $\{a{:}A \mid B[a]\}$, $a{:}A \cap B[a]$
  - pertype$(x, y.R(x, y)$
    - when $R(x, y)$ is a PER on Base
    - Base is the type corresponding to $\sim$
    - Quotient type $T//E$ is a special case
- Extensional $(x \in A \ \wedge A = B \in \mathbb{U} \Rightarrow x \in B)$
- No decidable type checking algorithm!

The real numbers

We use Bishop's definition of a regular sequence of rationals, but normalize so that *nth* rational has denominator $2n$; and then "clear denominators".

The real numbers

We use Bishop's definition of a regular sequence of rationals, but normalize so that *nth* rational has denominator $2n$; and then "clear denominators".

Definition

A sequence $x_1, x_2, x_3, \ldots$ of integers is regular if

$$\forall n, m : \mathbb{N}^+. \ |m * x_n - n * x_m| \leq 2(n + m)$$

The sequence is k-regular if

$$\forall n, m : \mathbb{N}^+. \ |m * x_n - n * x_m| \leq 2k(n + m)$$

The real numbers

We use Bishop's definition of a regular sequence of rationals, but normalize so that *nth* rational has denominator $2n$; and then "clear denominators".

Definition

A sequence $x_1, x_2, x_3, \ldots$ of integers is regular if

$$\forall n, m : \mathbb{N}^+. \ |m * x_n - n * x_m| \leq 2(n + m)$$

The sequence is k-regular if

$$\forall n, m : \mathbb{N}^+. \ |m * x_n - n * x_m| \leq 2k(n + m)$$

Definition

$$\mathbb{R} = \{x : \mathbb{N}^+ \to \mathbb{Z} \mid x \text{ is regular}\}$$

Regularity condition

Regularity condition

Lemma

Sequence x is k-regular if and only if for every $n, m \in \mathbb{N}^+$, the rational intervals $[\frac{x_n}{2kn} - \frac{1}{n}, \frac{x_n}{2kn} + \frac{1}{n}]$ and $[\frac{x_m}{2km} - \frac{1}{m}, \frac{x_m}{2km} + \frac{1}{m}]$ overlap (i.e. have a common member $q \in \mathbb{Q}$).

Regularity condition

Lemma

Sequence x is k-regular if and only if for every $n, m \in \mathbb{N}^+$, the rational intervals $[\frac{x_n}{2kn} - \frac{1}{n}, \frac{x_n}{2kn} + \frac{1}{n}]$ and $[\frac{x_m}{2km} - \frac{1}{m}, \frac{x_m}{2km} + \frac{1}{m}]$ overlap (i.e. have a common member $q \in \mathbb{Q}$).

Proof.

The intervals overlap if and only if $\frac{x_n}{2kn} + \frac{1}{n} \geq \frac{x_m}{2km} - \frac{1}{m}$ and $\frac{x_n}{2kn} - \frac{1}{n} \leq \frac{x_m}{2km} + \frac{1}{m}$. Multiplying by $2knm$ to clear the denominators, these hold if and only if $m * x_n + 2km \geq n * x_m - 2kn$ and $m * x_n - 2km \leq n * x_m + 2kn$, which is the same as $2kn + 2km \geq n * x_m - m * x_n$ and $m * x_n - n * x_m \leq 2km + 2kn$. These hold if and only if $|m * x_n - n * x_m| \leq 2k(n + m)$. □

Equivalence relation on $\mathbb{R}$

Equivalence relation on $\mathbb{R}$

Definition

Real numbers x and y are equivalent, $(x \equiv y)$, if and only if $\forall n{:}\mathbb{N}^+.\ |x_n - y_n| \leq 4$

We show next that $(x \equiv y)$ is an equivalence relation on $\mathbb{R}$.

Equivalence relation on $\mathbb{R}$

Definition

Real numbers x and y are equivalent, $(x \equiv y)$, if and only if $\forall n{:}\mathbb{N}^+.\ |x_n - y_n| \leq 4$

We show next that $(x \equiv y)$ is an equivalence relation on $\mathbb{R}$.

Definition

$\mathsf{bnddiff}(x, y) \equiv \exists B{:}\mathbb{N}.\ \forall n{:}\mathbb{N}^+.\ |x_n - y_n| \leq B$
(the pointwise difference between x and y is bounded).

Equivalence relation on $\mathbb{R}$

Definition

Real numbers x and y are equivalent, $(x \equiv y)$, if and only if $\forall n{:}\mathbb{N}^+.\ |x_n - y_n| \leq 4$

We show next that $(x \equiv y)$ is an equivalence relation on $\mathbb{R}$.

Definition

$\mathsf{bnddiff}(x, y) \equiv \exists B{:}\mathbb{N}.\ \forall n{:}\mathbb{N}^+.\ |x_n - y_n| \leq B$
(the pointwise difference between x and y is bounded).

Lemma

For all $x, y \in \mathbb{R}$, $(x \equiv y) \iff \mathsf{bnddiff}(x, y)$

Proof

If $(x \equiv y)$ then we may take $B = 4$.

Proof

If $(x \equiv y)$ then we may take $B = 4$. In the other direction, suppose $\forall n{:}\mathbb{N}^+.\ |x_n - y_n| \leq B$, then, for any $n, m \in \mathbb{N}^+$, if $5 \leq |x_n - y_n|$ then
$5m \leq m|x_n - y_n| \leq |m*x_n - n*x_m| + |n*x_m - n*y_m| + |n*y_m - m*y_n|$, by the triangle inequality.

Proof

If $(x \equiv y)$ then we may take $B = 4$. In the other direction, suppose $\forall n{:}\mathbb{N}^+.\ |x_n - y_n| \leq B$, then, for any $n, m \in \mathbb{N}^+$, if $5 \leq |x_n - y_n|$ then
$5m \leq m|x_n - y_n| \leq |m*x_n - n*x_m| + |n*x_m - n*y_m| + |n*y_m - m*y_n|$, by the triangle inequality. By regularity and our assumption, the right side is $\leq 4(n+m) + nB$. Thus, $5m \leq 4(n+m) + nB$, so, $m \leq (4+B)n$ and we obtain a contradiction when $m = 1 + (4+B)n$.

Proof

If $(x \equiv y)$ then we may take $B = 4$. In the other direction, suppose $\forall n{:}\mathbb{N}^+.\ |x_n - y_n| \leq B$, then, for any $n, m \in \mathbb{N}^+$, if $5 \leq |x_n - y_n|$ then
$5m \leq m|x_n - y_n| \leq |m*x_n - n*x_m| + |n*x_m - n*y_m| + |n*y_m - m*y_n|$, by the triangle inequality. By regularity and our assumption, the right side is $\leq 4(n+m) + nB$. Thus, $5m \leq 4(n+m) + nB$, so, $m \leq (4+B)n$ and we obtain a contradiction when
$m = 1 + (4+B)n$.
Hence, $|x_n - y_n| \leq 4$.

Proof

If $(x \equiv y)$ then we may take $B = 4$. In the other direction, suppose $\forall n{:}\mathbb{N}^+.\ |x_n - y_n| \leq B$, then, for any $n, m \in \mathbb{N}^+$, if $5 \leq |x_n - y_n|$ then
$5m \leq m|x_n - y_n| \leq |m*x_n - n*x_m| + |n*x_m - n*y_m| + |n*y_m - m*y_n|$, by the triangle inequality. By regularity and our assumption, the right side is $\leq 4(n+m) + nB$. Thus, $5m \leq 4(n+m) + nB$, so, $m \leq (4+B)n$ and we obtain a contradiction when $m = 1 + (4+B)n$.
Hence, $|x_n - y_n| \leq 4$.

Corollary

$\equiv$ *is an equivalence relation on* $\mathbb{R}$

Acceleration

We use the symbol $\div$ for integer division; it satisfies:

$$n = k(n \div k) + (n \text{ rem } k) \quad (\text{with } |n \text{ rem } k| < |k|)$$

Acceleration

We use the symbol $\div$ for integer division; it satisfies:

$$n = k(n \div k) + (n \text{ rem } k) \quad (\text{with } |n \text{ rem } k| < |k|)$$

Definition

The sequence $\text{accel}(k, x) = \lambda n.\ x_{2kn} \div 2k$ is called the k-acceleration of sequence x.

Acceleration

We use the symbol $\div$ for integer division; it satisfies:

$$n = k(n \div k) + (n \text{ rem } k) \quad (\text{with } |n \text{ rem } k| < |k|)$$

Definition

The sequence $\text{accel}(k, x) = \lambda n.\ x_{2kn} \div 2k$ is called the k-acceleration of sequence x.

Lemma

If sequence x is k-regular, then $\text{accel}(k, x)$ is regular, and $\text{bnddiff}(\text{accel}(k, x), x)$.

(See lecture notes for the proof.)

Arithmetic and Ordering

If x and y are regular sequences then $\lambda n.\ x_n + y_n$ is a 2-regular sequence. So we define

$$x + y = \mathrm{accel}(2, \lambda n.\ x_n + y_n)$$

Arithmetic and Ordering

If x and y are regular sequences then $\lambda n.\ x_n + y_n$ is a 2-regular sequence. So we define

$$x + y = \mathsf{accel}(2, \lambda n.\ x_n + y_n)$$

$$|x| \quad = \quad \lambda n.\ |x_n| \tag{1}$$

Arithmetic and Ordering

If x and y are regular sequences then $\lambda n.\ x_n + y_n$ is a 2-regular sequence. So we define

$$x + y = \mathsf{accel}(2, \lambda n.\ x_n + y_n)$$

$$|x| = \lambda n.\ |x_n| \quad (1)$$

$$\max(x, y) = \lambda n.\ \max(x_n, y_n) \quad (2)$$

Arithmetic and Ordering

If x and y are regular sequences then $\lambda n.\ x_n + y_n$ is a 2-regular sequence. So we define

$$x + y = \mathsf{accel}(2, \lambda n.\ x_n + y_n)$$

$$|x| = \lambda n.\ |x_n| \quad (1)$$

$$\max(x, y) = \lambda n.\ \max(x_n, y_n) \quad (2)$$

$$\min(x, y) = \lambda n.\ \min(x_n, y_n) \quad (3)$$

Arithmetic and Ordering

If x and y are regular sequences then $\lambda n.\ x_n + y_n$ is a 2-regular sequence. So we define

$$x + y = \mathsf{accel}(2, \lambda n.\ x_n + y_n)$$

$$\begin{aligned}
|x| &= \lambda n.\ |x_n| && (1)\\
\max(x, y) &= \lambda n.\ \max(x_n, y_n) && (2)\\
\min(x, y) &= \lambda n.\ \min(x_n, y_n) && (3)\\
-x &= \lambda n.\ -x_n && (4)
\end{aligned}$$

Arithmetic and Ordering

If x and y are regular sequences then $\lambda n.\ x_n + y_n$ is a 2-regular sequence. So we define

$$x + y = \mathsf{accel}(2, \lambda n.\ x_n + y_n)$$

$$\begin{aligned}
|x| &= \lambda n.\ |x_n| && (1)\\
\max(x, y) &= \lambda n.\ \max(x_n, y_n) && (2)\\
\min(x, y) &= \lambda n.\ \min(x_n, y_n) && (3)\\
-x &= \lambda n.\ -x_n && (4)\\
x/k &= \lambda n.\ x_{2n} \div 2k && (5)
\end{aligned}$$

Arithmetic and Ordering

If x and y are regular sequences then $\lambda n.\ x_n + y_n$ is a 2-regular sequence. So we define

$$x + y = \mathsf{accel}(2, \lambda n.\ x_n + y_n)$$

$$\begin{aligned}
|x| &= \lambda n.\ |x_n| && (1)\\
\max(x, y) &= \lambda n.\ \max(x_n, y_n) && (2)\\
\min(x, y) &= \lambda n.\ \min(x_n, y_n) && (3)\\
-x &= \lambda n.\ -x_n && (4)\\
x/k &= \lambda n.\ x_{2n} \div 2k && (5)\\
x \leq y &\Leftrightarrow \forall n{:}\mathbb{N}^+.\ x_n \leq y_n + 4 && (6)
\end{aligned}$$

Arithmetic and Ordering

If x and y are regular sequences then $\lambda n.\ x_n + y_n$ is a 2-regular sequence. So we define

$$x + y = \mathsf{accel}(2, \lambda n.\ x_n + y_n)$$

$$
\begin{aligned}
|x| &= \lambda n.\ |x_n| && (1)\\
\max(x, y) &= \lambda n.\ \max(x_n, y_n) && (2)\\
\min(x, y) &= \lambda n.\ \min(x_n, y_n) && (3)\\
-x &= \lambda n.\ -x_n && (4)\\
x/k &= \lambda n.\ x_{2n} \div 2k && (5)\\
x \le y &\Leftrightarrow \forall n{:}\mathbb{N}^+.\ x_n \le y_n + 4 && (6)\\
x < y &\Leftrightarrow \exists n{:}\mathbb{N}^+.\ x_n + 4 < y_n && (7)
\end{aligned}
$$

Arithmetic and Ordering

If x and y are regular sequences then $\lambda n.\ x_n + y_n$ is a 2-regular sequence. So we define

$$x + y = \mathsf{accel}(2, \lambda n.\ x_n + y_n)$$

$$\begin{aligned}
|x| &= \lambda n.\ |x_n| && (1)\\
\max(x, y) &= \lambda n.\ \max(x_n, y_n) && (2)\\
\min(x, y) &= \lambda n.\ \min(x_n, y_n) && (3)\\
-x &= \lambda n.\ -x_n && (4)\\
x/k &= \lambda n.\ x_{2n} \div 2k && (5)\\
x \le y &\Leftrightarrow \forall n{:}\mathbb{N}^+.\ x_n \le y_n + 4 && (6)\\
x < y &\Leftrightarrow \exists n{:}\mathbb{N}^+.\ x_n + 4 < y_n && (7)
\end{aligned}$$

Multiplication and $1/x$ (for $x\#0$) are more complicated.

More basic facts about $\mathbb{R}$

For every integer z we get a real $\lambda n.\ 2zn$ that we will also write as z. It follows from these definitions that

$$|x - (x_m/2m)| \quad \leq \quad 1/m \tag{8}$$

More basic facts about $\mathbb{R}$

For every integer z we get a real $\lambda n.\ 2zn$ that we will also write as z. It follows from these definitions that

$$|x - (x_m/2m)| \quad \leq \quad 1/m \tag{8}$$

Everything in chapter 2 of Bishop and Bridges "Constructive Analysis" has been formalized,

More basic facts about $\mathbb{R}$

For every integer z we get a real $\lambda n.\ 2zn$ that we will also write as z. It follows from these definitions that

$$|x - (x_m/2m)| \quad \leq \quad 1/m \tag{8}$$

Everything in chapter 2 of Bishop and Bridges "Constructive Analysis" has been formalized, including this important lemma:

Lemma

If $x < y$ *then* $\forall z : \mathbb{R}.\ (x < z) \vee (z < y)$

More basic facts about $\mathbb{R}$

For every integer z we get a real $\lambda n.\ 2zn$ that we will also write as z. It follows from these definitions that

$$|x - (x_m/2m)| \quad \leq \quad 1/m \tag{8}$$

Everything in chapter 2 of Bishop and Bridges "Constructive Analysis" has been formalized, including this important lemma:

Lemma

If $x < y$ *then* $\forall z{:}\mathbb{R}.\ (x < z) \vee (z < y)$

Proof.

Let n be such that $x_n + 4 < y_n$ and let $m = 12n + 1$. Then regularity of x and y implies $x_m + 8 < y_m$ (see notes). Now we must have either $x_m + 4 < z_m$ or $z_m + 4 < y_m$. □

The (false) Continuity Principle

Type $\mathbb{N}_k = \{n{:}\mathbb{N} \mid n < k\}$. Let $\mathbb{S} = \mathbb{N} \to \mathbb{N}$, and $\mathbb{S}_k = \mathbb{N}_k \to \mathbb{N}$.

The (false) Continuity Principle

Type $\mathbb{N}_k = \{n:\mathbb{N} \mid n < k\}$. Let $\mathbb{S} = \mathbb{N} \to \mathbb{N}$, and $\mathbb{S}_k = \mathbb{N}_k \to \mathbb{N}$. Brouwer's continuity principle for numbers: If F has type $\mathbb{S} \to \mathbb{N}$ and f has type $\mathbb{S}$ then $F(f)$ depends on only a finite part of f.

$$\forall F:\mathbb{S} \to \mathbb{N}.\ \forall f:\mathbb{S}.\ \exists k:\mathbb{N}.\ \forall g:\mathbb{S}.\ (g = f \in \mathbb{S}_k) \Rightarrow F(g) = F(f)$$

The (false) Continuity Principle

Type $\mathbb{N}_k = \{n:\mathbb{N} \mid n < k\}$. Let $\mathbb{S} = \mathbb{N} \to \mathbb{N}$, and $\mathbb{S}_k = \mathbb{N}_k \to \mathbb{N}$. Brouwer's continuity principle for numbers: If F has type $\mathbb{S} \to \mathbb{N}$ and f has type $\mathbb{S}$ then $F(f)$ depends on only a finite part of f.

$$\forall F:\mathbb{S} \to \mathbb{N}.\ \forall f:\mathbb{S}.\ \exists k:\mathbb{N}.\ \forall g:\mathbb{S}.\ (g = f \in \mathbb{S}_k) \Rightarrow F(g) = F(f)$$

The instance of the above continuity principle for $f = \overline{0} = \lambda i.0$ is

$$\forall F:\mathbb{S} \to \mathbb{N}.\ \exists k:\mathbb{N}.\ \forall g:\mathbb{S}.\ (g = \overline{0} \in \mathbb{S}_k) \Rightarrow F(g) = F(\overline{0})$$

The (false) Continuity Principle

Type $\mathbb{N}_k = \{n{:}\mathbb{N} \mid n < k\}$. Let $\mathbb{S} = \mathbb{N} \to \mathbb{N}$, and $\mathbb{S}_k = \mathbb{N}_k \to \mathbb{N}$. Brouwer's continuity principle for numbers: If F has type $\mathbb{S} \to \mathbb{N}$ and f has type $\mathbb{S}$ then $F(f)$ depends on only a finite part of f.

$$\forall F{:}\mathbb{S} \to \mathbb{N}.\ \forall f{:}\mathbb{S}.\ \exists k{:}\mathbb{N}.\ \forall g{:}\mathbb{S}.\ (g = f \in \mathbb{S}_k) \Rightarrow F(g) = F(f)$$

The instance of the above continuity principle for $f = \overline{0} = \lambda i.0$ is

$$\forall F{:}\mathbb{S} \to \mathbb{N}.\ \exists k{:}\mathbb{N}.\ \forall g{:}\mathbb{S}.\ (g = \overline{0} \in \mathbb{S}_k) \Rightarrow F(g) = F(\overline{0})$$

Using BHK interpretation there is a *modulus of continuity* function M of type $(\mathbb{S} \to \mathbb{N}) \to \mathbb{N}$ such that

$$\forall F{:}\mathbb{S} \to \mathbb{N}.\ \forall g{:}\mathbb{S}.\ (\forall n{:}\mathbb{N}.\ n < M(F) => g(n) = 0) \Rightarrow F(g) = F(\overline{0})$$

The (false) Continuity Principle

Type $\mathbb{N}_k = \{n{:}\mathbb{N} \mid n < k\}$. Let $\mathbb{S} = \mathbb{N} \to \mathbb{N}$, and $\mathbb{S}_k = \mathbb{N}_k \to \mathbb{N}$. Brouwer's continuity principle for numbers: If F has type $\mathbb{S} \to \mathbb{N}$ and f has type $\mathbb{S}$ then $F(f)$ depends on only a finite part of f.

$$\forall F{:}\mathbb{S} \to \mathbb{N}.\ \forall f{:}\mathbb{S}.\ \exists k{:}\mathbb{N}.\ \forall g{:}\mathbb{S}.\ (g = f \in \mathbb{S}_k) \Rightarrow F(g) = F(f)$$

The instance of the above continuity principle for $f = \overline{0} = \lambda i.0$ is

$$\forall F{:}\mathbb{S} \to \mathbb{N}.\ \exists k{:}\mathbb{N}.\ \forall g{:}\mathbb{S}.\ (g = \overline{0} \in \mathbb{S}_k) \Rightarrow F(g) = F(\overline{0})$$

Using BHK interpretation there is a *modulus of continuity* function M of type $(\mathbb{S} \to \mathbb{N}) \to \mathbb{N}$ such that

$$\forall F{:}\mathbb{S} \to \mathbb{N}.\ \forall g{:}\mathbb{S}.\ (\forall n{:}\mathbb{N}.\ n < M(F) => g(n) = 0) \Rightarrow F(g) = F(\overline{0})$$

This is false!

Escardo and Xu's proof

We have

$$\forall F : \mathbb{S} \to \mathbb{N}.\ \forall g : \mathbb{S}.\ (\forall n : \mathbb{N}.\ n < M(F) => g(n) = 0) \Rightarrow F(g) = F(\overline{0})$$

Escardo and Xu's proof

We have

$$\forall F : \mathbb{S} \to \mathbb{N}.\ \forall g : \mathbb{S}.\ (\forall n : \mathbb{N}.\ n < M(F) => g(n) = 0) \Rightarrow F(g) = F(\overline{0})$$

Let

$$J = M(\lambda f.\ 0) \qquad D(g) = M(\lambda f.\ g(f(J))$$

$$K = M(D) \qquad 0^t\overline{n} = \lambda i.\ \text{if } i < t \text{ then } 0 \text{ else } n$$

$$G(f) = \text{if } f(J) < K \text{ then } 0 \text{ else } 1 \qquad H(f) = f(J)$$

Escardo and Xu's proof

We have

$$\forall F : \mathbb{S} \to \mathbb{N}.\ \forall g : \mathbb{S}.\ (\forall n : \mathbb{N}.\ n < M(F) => g(n) = 0) \Rightarrow F(g) = F(\overline{0})$$

Let

$$J = M(\lambda f.\ 0) \qquad D(g) = M(\lambda f.\ g(f(J))$$

$$K = M(D) \qquad 0^t\overline{n} = \lambda i.\ \text{if } i < t \text{ then } 0 \text{ else } n$$

$$G(f) = \text{if } f(J) < K \text{ then } 0 \text{ else } 1 \qquad H(f) = f(J)$$

Note that $D(\overline{0}) = M(\lambda f.\ 0) = J$ and therefore, since $M(D) = K$,

$$\forall g : \mathbb{S}.\ (\forall n : \mathbb{N}.\ n < K => g(n) = 0) \Rightarrow D(g) = J$$

Escardo and Xu's proof

We have

$$\forall F : \mathbb{S} \to \mathbb{N}.\ \forall g : \mathbb{S}.\ (\forall n : \mathbb{N}.\ n < M(F) => g(n) = 0) \Rightarrow F(g) = F(\overline{0})$$

Let

$$J = M(\lambda f.\ 0) \qquad D(g) = M(\lambda f.\ g(f(J))$$

$$K = M(D) \qquad 0^t\overline{n} = \lambda i.\ \text{if } i < t \text{ then } 0 \text{ else } n$$

$$G(f) = \text{if } f(J) < K \text{ then } 0 \text{ else } 1 \qquad H(f) = f(J)$$

Note that $D(\overline{0}) = M(\lambda f.\ 0) = J$ and therefore, since $M(D) = K$,

$$\forall g : \mathbb{S}.\ (\forall n : \mathbb{N}.\ n < K => g(n) = 0) \Rightarrow D(g) = J$$

In particular,
$J = D(0^K\overline{1}) = M(\lambda f.\ 0^K\overline{1}(f(J)) = M(\lambda f.\ G(f)) = M(G)$. If $K > 0$ then $G(\overline{0}) = 0$ and $G(0^J\overline{K}) = 1$ contradicts $M(G) = J$.

Escardo and Xu's proof

We have

$$\forall F : \mathbb{S} \to \mathbb{N}.\ \forall g : \mathbb{S}.\ (\forall n : \mathbb{N}.\ n < M(F) => g(n) = 0) \Rightarrow F(g) = F(\overline{0})$$

Let

$$J = M(\lambda f.\ 0) \qquad D(g) = M(\lambda f.\ g(f(J))$$

$$K = M(D) \qquad 0^t\overline{n} = \lambda i.\ \text{if } i < t \text{ then } 0 \text{ else } n$$

$$G(f) = \text{if } f(J) < K \text{ then } 0 \text{ else } 1 \qquad H(f) = f(J)$$

Note that $D(\overline{0}) = M(\lambda f.\ 0) = J$ and therefore, since $M(D) = K$,

$$\forall g : \mathbb{S}.\ (\forall n : \mathbb{N}.\ n < K => g(n) = 0) \Rightarrow D(g) = J$$

In particular,
$J = D(0^K\overline{1}) = M(\lambda f.\ 0^K\overline{1}(f(J)) = M(\lambda f.\ G(f)) = M(G)$. If $K > 0$ then $G(\overline{0}) = 0$ and $G(0^J\overline{K}) = 1$ contradicts $M(G) = J$. If, on the other hand, $K = 0$ then $\forall g : \mathbb{S}.\ D(g) = J$ so $D(\lambda x.x) = M(\lambda f.\ (f(J)) = M(H) = J$. But $H(\overline{0}) = 0$ and $H(0^J\overline{1}) = 1$ contradicts $M(H) = J$.

Why is M impossible?

Why is M impossible?

- $D(g) = M(\lambda f.\ g(f(J))$ where $J = M(\lambda f.\ 0)$.

Why is M impossible?

- $D(g) = M(\lambda f.\ g(f(J))$ where $J = M(\lambda f.\ 0)$.
- We note that $D(\overline{0}) = M(\lambda f.\ 0) = J$.

Why is M impossible?

- $D(g) = M(\lambda f.\ g(f(J))$ where $J = M(\lambda f.\ 0)$.
- We note that $D(\overline{0}) = M(\lambda f.\ 0) = J$.
- $D(\overline{0}) = M(\lambda f.\ \overline{0}(f(J)))$

Why is M impossible?

- $D(g) = M(\lambda f.\ g(f(J))$ where $J = M(\lambda f.\ 0)$.
- We note that $D(\overline{0}) = M(\lambda f.\ 0) = J$.
- $D(\overline{0}) = M(\lambda f.\ \overline{0}(f(J)))$
- $\lambda f.\ \overline{0}(f(J))$ *extensionally equal to* $\lambda f.\ 0$ but not *intensionally equal*.

Why is M impossible?

- $D(g) = M(\lambda f.\ g(f(J))$ where $J = M(\lambda f.\ 0)$.
- We note that $D(\overline{0}) = M(\lambda f.\ 0) = J$.
- $D(\overline{0}) = M(\lambda f.\ \overline{0}(f(J)))$
- $\lambda f.\ \overline{0}(f(J))$ *extensionally equal to* $\lambda f.\ 0$ but not *intensionally equal*.
- Intensional M could have $M(\lambda f.\ 0) = 0$ but $M(\lambda f.\ \overline{0}(f(J))) > J$

Why is M impossible?

- $D(g) = M(\lambda f.\ g(f(J))$ where $J = M(\lambda f.\ 0)$.
- We note that $D(\overline{0}) = M(\lambda f.\ 0) = J$.
- $D(\overline{0}) = M(\lambda f.\ \overline{0}(f(J)))$
- $\lambda f.\ \overline{0}(f(J))$ *extensionally equal to* $\lambda f.\ 0$ but not *intensionally equal*.
- Intensional M could have $M(\lambda f.\ 0) = 0$ but $M(\lambda f.\ \overline{0}(f(J))) > J$
- Escardo and Xu show M can not be extensional.

Function types (Π type)

Function types (Π type)

- B is a *type family* over A, if $a_1 = a_2 \in A$ implies $B[a_1] = B[a_2] \in \mathbb{U}_i$.

Function types (Π type)

- B is a *type family* over A, if $a_1 = a_2 \in A$ implies $B[a_1] = B[a_2] \in \mathbb{U}_i$.
- PER for type $a : A \to B[a]$ is defined by $f = g \in a : A \to B[a]$ if and only if, for all terms a_1 and a_2, $a_1 = a_2 \in A \Rightarrow f(a_1) = g(a_2) \in B[a_1]$

Function types (Π type)

- B is a *type family* over A, if $a_1 = a_2 \in A$ implies $B[a_1] = B[a_2] \in \mathbb{U}_i$.
- PER for type $a : A \to B[a]$ is defined by $f = g \in a : A \to B[a]$ if and only if, for all terms a_1 and a_2, $a_1 = a_2 \in A \Rightarrow f(a_1) = g(a_2) \in B[a_1]$
- So function extensionality is built in to definition of $a : A \to B[a]$.

Function types (Π type)

- B is a *type family* over A, if $a_1 = a_2 \in A$ implies $B[a_1] = B[a_2] \in \mathbb{U}_i$.
- PER for type $a : A \to B[a]$ is defined by $f = g \in a : A \to B[a]$ if and only if, for all terms a_1 and a_2, $a_1 = a_2 \in A \Rightarrow f(a_1) = g(a_2) \in B[a_1]$
- So function extensionality is built in to definition of $a : A \to B[a]$.
- How can we say for every A there is a B ... without asserting the existence of an extensional function $A \to B$?

Function types (Π type)

- B is a *type family* over A, if $a_1 = a_2 \in A$ implies $B[a_1] = B[a_2] \in \mathbb{U}_i$.
- PER for type $a : A \to B[a]$ is defined by $f = g \in a : A \to B[a]$ if and only if, for all terms a_1 and a_2, $a_1 = a_2 \in A \Rightarrow f(a_1) = g(a_2) \in B[a_1]$
- So function extensionality is built in to definition of $a : A \to B[a]$.
- How can we say for every A there is a B ... without asserting the existence of an extensional function $A \to B$?
- answer: Change equality in B

Quotient type

Quotient type

- If $E(x, y)$ is an equivalence relation on a type T, then the quotient type $T//E$ is the type such that $x = y \in T//E \Leftrightarrow (x \in T \wedge y \in T \wedge E(x, y))$.

Quotient type

- If $E(x, y)$ is an equivalence relation on a type T, then the quotient type $T//E$ is the type such that $x = y \in T//E \Leftrightarrow (x \in T \wedge y \in T \wedge E(x, y))$.
- For trivial $E(x, y) = \text{True}$ then $x = y \in T//\text{True} \Leftrightarrow (x \in T \wedge y \in T)$.

Quotient type

- If $E(x, y)$ is an equivalence relation on a type T, then the quotient type $T//E$ is the type such that $x = y \in T//E \Leftrightarrow (x \in T \wedge y \in T \wedge E(x, y))$.
- For trivial $E(x, y) = \mathsf{True}$ then $x = y \in T//\mathsf{True} \Leftrightarrow (x \in T \wedge y \in T)$.
- We write $T//\mathsf{True}$ as $\lfloor T$ and call it the *half squash* of T.

Quotient type

- If $E(x, y)$ is an equivalence relation on a type T, then the quotient type $T//E$ is the type such that $x = y \in T//E \Leftrightarrow (x \in T \wedge y \in T \wedge E(x, y))$.
- For trivial $E(x, y) = \text{True}$ then $x = y \in T//\text{True} \Leftrightarrow (x \in T \wedge y \in T)$.
- We write $T//\text{True}$ as $\lfloor T$ and call it the *half squash* of T.
- Function $f \in a{:}A \rightarrow \lfloor B[a]$ is a non-extensional function.

Quotient type

- If $E(x, y)$ is an equivalence relation on a type T, then the quotient type $T//E$ is the type such that $x = y \in T//E \Leftrightarrow (x \in T \wedge y \in T \wedge E(x, y))$.
- For trivial $E(x, y) = \text{True}$ then $x = y \in T//\text{True} \Leftrightarrow (x \in T \wedge y \in T)$.
- We write $T//\text{True}$ as $\lfloor T$ and call it the *half squash* of T.
- Function $f \in a : A \rightarrow\lfloor B[a]$ is a non-extensional function.
- Brouwer's Continuity :

$$\forall F : \mathbb{S} \rightarrow \mathbb{N}.\ \forall f : \mathbb{S}.\ \lfloor \exists k : \mathbb{N}.\ \forall g : \mathbb{S}.\ (g = f \in \mathbb{S}_k) \Rightarrow F(g) = F(f)$$

Why is the continuity principle true?

Why is the continuity principle true?

- We can compute (intensionally) a modulus of continuity for F using the Nuprl computation system.

Why is the continuity principle true?

- We can compute (intensionally) a modulus of continuity for F using the Nuprl computation system.
- Let bound-domain(f, n, e) be

  $\lambda x.$ if $x < 0$ then $\perp$ else if $x < n$ then $f(n)$ else exception(e, Ax)

Why is the continuity principle true?

- We can compute (intensionally) a modulus of continuity for F using the Nuprl computation system.
- Let bound-domain(f, n, e) be

  $\lambda x.$ if $x < 0$ then $\perp$ else if $x < n$ then $f(n)$ else exception(e, Ax)

  - diverges on $x \notin \mathbb{N}$

Why is the continuity principle true?

- We can compute (intensionally) a modulus of continuity for F using the Nuprl computation system.
- Let bound-domain(f, n, e) be

  $\lambda x.$ if $x < 0$ then $\perp$ else if $x < n$ then $f(n)$ else exception(e, Ax)

  - diverges on $x \notin \mathbb{N}$
  - agrees with f on $x < n$

Why is the continuity principle true?

- We can compute (intensionally) a modulus of continuity for F using the Nuprl computation system.
- Let bound-domain(f, n, e) be

  $\lambda x.$ if $x < 0$ then $\perp$ else if $x < n$ then $f(n)$ else exception(e, Ax)

  - diverges on $x \notin \mathbb{N}$
  - agrees with f on $x < n$
  - raises exception e on $x \geq n$

Why is the continuity principle true?

- We can compute (intensionally) a modulus of continuity for F using the Nuprl computation system.
- Let bound-domain(f, n, e) be

  $\lambda x.$ if $x < 0$ then $\perp$ else if $x < n$ then $f(n)$ else exception(e, Ax)

  - diverges on $x \notin \mathbb{N}$
  - agrees with f on $x < n$
  - raises exception e on $x \geq n$
  - Ax is a canonical form of type Unit

Why is the continuity principle true?

- We can compute (intensionally) a modulus of continuity for F using the Nuprl computation system.
- Let bound-domain(f, n, e) be

  $\lambda x.$ if $x < 0$ then $\perp$ else if $x < n$ then $f(n)$ else exception(e, Ax)

  - diverges on $x \notin \mathbb{N}$
  - agrees with f on $x < n$
  - raises exception e on $x \geq n$
  - Ax is a canonical form of type Unit
- *Kleene M-function* for F is

$$M(n, f) = \nu e.F(\text{bound-domain}(f, n, e))?e : x.\mathrm{Ax}$$

Why is the continuity principle true?

- We can compute (intensionally) a modulus of continuity for F using the Nuprl computation system.
- Let bound-domain(f, n, e) be

  $\lambda x.$ if $x < 0$ then $\bot$ else if $x < n$ then $f(n)$ else exception(e, Ax)

  - diverges on $x \notin \mathbb{N}$
  - agrees with f on $x < n$
  - raises exception e on $x \geq n$
  - Ax is a canonical form of type Unit
- *Kleene M-function* for F is

$$M(n, f) = \nu e.F(\text{bound-domain}(f, n, e))?e : x.\mathrm{Ax}$$

  - The *try/catch* operator $\_?e : x.G(x)$ catches an exception with name e, binds its value to x, and continues with $G(x)$.

Why is the continuity principle true?

- We can compute (intensionally) a modulus of continuity for F using the Nuprl computation system.
- Let bound-domain(f, n, e) be

  $\lambda x.$ if $x < 0$ then $\perp$ else if $x < n$ then $f(n)$ else exception(e, Ax)

  - diverges on $x \notin \mathbb{N}$
  - agrees with f on $x < n$
  - raises exception e on $x \geq n$
  - Ax is a canonical form of type Unit
- *Kleene M-function* for F is

$$M(n, f) = \nu e.F(\text{bound-domain}(f, n, e))?e : x.\mathrm{Ax}$$

  - The *try/catch* operator $\_?e : x.G(x)$ catches an exception with name e, binds its value to x, and continues with $G(x)$.
  - ν operator chooses a *fresh* name e, so F can not catch exceptions named e.

Why is the continuity principle true?

- We can compute (intensionally) a modulus of continuity for F using the Nuprl computation system.
- Let bound-domain(f, n, e) be

  $\lambda x.$ if $x < 0$ then $\bot$ else if $x < n$ then $f(n)$ else exception(e, Ax)

  - diverges on $x \notin \mathbb{N}$
  - agrees with f on $x < n$
  - raises exception e on $x \geq n$
  - Ax is a canonical form of type Unit
- *Kleene M-function* for F is

  $$M(n, f) = \nu e.F(\text{bound-domain}(f, n, e))?e : x.\mathrm{Ax}$$

  - The *try/catch* operator $\_?e : x.G(x)$ catches an exception with name e, binds its value to x, and continues with $G(x)$.
  - ν operator chooses a *fresh* name e, so F can not catch exceptions named e.
- Iterate $M(0, f), M(1, f), M(2, f), \ldots M(k, f)$ until we get an number rather than Ax at which point we have the modulus of continuity k.

CONT :

$$\forall F : \mathbb{S} \to \mathbb{N}.\ \forall f : \mathbb{S}.\ \ \downarrow \exists k : \mathbb{N}.\ \forall g : \mathbb{S}.\ (g = f \in \mathbb{S}_k) \Rightarrow F(g) = F(f)$$

CONT :

$$\forall F : \mathbb{S} \to \mathbb{N}.\ \forall f : \mathbb{S}.\ \ \lfloor \exists k : \mathbb{N}.\ \forall g : \mathbb{S}.\ (g = f \in \mathbb{S}_k) \Rightarrow F(g) = F(f)$$

What can we prove with this?

CONT :

$$\forall F : \mathbb{S} \to \mathbb{N}.\ \forall f : \mathbb{S}.\ \ \downarrow \exists k : \mathbb{N}.\ \forall g : \mathbb{S}.\ (g = f \in \mathbb{S}_k) \Rightarrow F(g) = F(f)$$

What can we prove with this?
Come back for Lecture 2

Product (Σ) type, Set type, Squash type

Product (Σ) type, Set type, Squash type

- Dependent product type $a{:}A \times B[a]$.

Product (Σ) type, Set type, Squash type

- Dependent product type $a{:}A \times B[a]$.
- $p = q \in a{:}A \times B[a]$ iff $p \mapsto \langle a_1, b_1 \rangle$ and $q \mapsto \langle a_2, b_2 \rangle$ and $a_1 = a_2 \in A$ and $b_1 = b_2 \in B[a_1]$.

Product (Σ) type, Set type, Squash type

- Dependent product type $a{:}A \times B[a]$.
- $p = q \in a{:}A \times B[a]$ iff $p \mapsto \langle a_1, b_1 \rangle$ and $q \mapsto \langle a_2, b_2 \rangle$ and $a_1 = a_2 \in A$ and $b_1 = b_2 \in B[a_1]$.
- Set type (aka refinement type) $\{a{:}A \mid B[a]\}$

Product (Σ) type, Set type, Squash type

- Dependent product type $a{:}A \times B[a]$.
- $p = q \in a{:}A \times B[a]$ iff $p \mapsto \langle a_1, b_1 \rangle$ and $q \mapsto \langle a_2, b_2 \rangle$ and $a_1 = a_2 \in A$ and $b_1 = b_2 \in B[a_1]$.
- Set type (aka refinement type) $\{a{:}A \mid B[a]\}$
- $a \in \{a{:}A \mid B[a]\}$ iff $a = \mathsf{fst}(p)$ for some $p \in a{:}A \times B[a]$

Product (Σ) type, Set type, Squash type

- Dependent product type $a{:}A \times B[a]$.
- $p = q \in a{:}A \times B[a]$ iff $p \mapsto \langle a_1, b_1 \rangle$ and $q \mapsto \langle a_2, b_2 \rangle$ and $a_1 = a_2 \in A$ and $b_1 = b_2 \in B[a_1]$.
- Set type (aka refinement type) $\{a{:}A \mid B[a]\}$
- $a \in \{a{:}A \mid B[a]\}$ iff $a = \mathsf{fst}(p)$ for some $p \in a{:}A \times B[a]$
- Squash type $\downarrow T = \{x{:}\mathsf{Unit} \mid T\}$

Product (Σ) type, Set type, Squash type

- Dependent product type $a{:}A \times B[a]$.
- $p = q \in a{:}A \times B[a]$ iff $p \mapsto \langle a_1, b_1 \rangle$ and $q \mapsto \langle a_2, b_2 \rangle$ and $a_1 = a_2 \in A$ and $b_1 = b_2 \in B[a_1]$.
- Set type (aka refinement type) $\{a{:}A \mid B[a]\}$
- $a \in \{a{:}A \mid B[a]\}$ iff $a = \mathsf{fst}(p)$ for some $p \in a{:}A \times B[a]$
- Squash type $\downarrow T = \{x{:}\mathsf{Unit} \mid T\}$
- $T \Rightarrow (\lfloor T) \Rightarrow (\downarrow T) \Rightarrow \neg\neg T$

Stable and Squash Stable Propositions

Stable and Squash Stable Propositions

- P is *stable* if $\neg\neg P \Rightarrow P$

Stable and Squash Stable Propositions

- P is *stable* if $\neg\neg P \Rightarrow P$
- Markov's Principle (MP) says that for decidable $P(n)$, $\exists n{:}\mathbb{N}.\ P(n)$ is stable.

Stable and Squash Stable Propositions

- P is *stable* if $\neg\neg P \Rightarrow P$
- Markov's Principle (MP) says that for decidable $P(n)$, $\exists n{:}\mathbb{N}.\ P(n)$ is stable.
- Nuprl does not prove MP, but is consistent with MP.

Stable and Squash Stable Propositions

- P is *stable* if $\neg\neg P \Rightarrow P$
- Markov's Principle (MP) says that for decidable $P(n)$, $\exists n{:}\mathbb{N}.\ P(n)$ is stable.
- Nuprl does not prove MP, but is consistent with MP.
- P is *squash stable* if $\downarrow P \Rightarrow P$

Stable and Squash Stable Propositions

- P is *stable* if $\neg\neg P \Rightarrow P$
- Markov's Principle (MP) says that for decidable $P(n)$, $\exists n{:}\mathbb{N}.\ P(n)$ is stable.
- Nuprl does not prove MP, but is consistent with MP.
- P is *squash stable* if $\downarrow P \Rightarrow P$
- For decidable $P(n)$, $\exists n{:}\mathbb{N}.\ P(n)$ is squash stable, because $\mu n.\ P(n) \in \exists n{:}\mathbb{N}.\ P(n)$ if $\downarrow\exists n{:}\mathbb{N}.\ P(n)$