Inductive Construction in NuprlType Theory using Bar Induction

Mark Bickford, Robert Constable

May 12, 2014

Introduction: Two questions

What are the fundamental induction principles?

Introduction: Two questions

What are the fundamental induction principles?

What are the fundamental type constructors?

Introduction: Two questions

What are the fundamental induction principles?

What are the fundamental type constructors?

We are giving two talks on Nuprl and the type theory it implements (CTT 2014). In CTT14 we can reason about untyped computation using a version of Kleene equality. We reason about partial recursive functions using **partial types** that contain divergent terms.

Introduction: Two questions

What are the fundamental induction principles?

What are the fundamental type constructors?

We are giving two talks on Nuprl and the type theory it implements (CTT 2014). In CTT14 we can reason about untyped computation using a version of Kleene equality. We reason about partial recursive functions using **partial types** that contain divergent terms.

This talk is about why we have added **Brouwer's Bar Induction** and how it answers the first question.

Introduction: Two questions

What are the fundamental induction principles?

What are the fundamental type constructors?

We are giving two talks on Nuprl and the type theory it implements (CTT 2014). In CTT14 we can reason about untyped computation using a version of Kleene equality. We reason about partial recursive functions using **partial types** that contain divergent terms.

This talk is about why we have added **Brouwer's Bar Induction** and how it answers the first question.

The talk tomorrow proposes an answer to the second question and shows how we can define the CTT14 types, including the partial types, from a few very **basic type constructors**.

What is CTT14?

Starts with terms of an **untyped computation system**:

What is CTT14?

Starts with terms of an **untyped computation system**:

Canonical terms (values) include integers, tokens, $\lambda x.t$, $\langle t_1, t_2 \rangle$, $\mathtt{inl}(t)$, $\mathtt{inr}(t)$, and $\mathtt{Ax}$.

What is CTT14?

Starts with terms of an **untyped computation system**:

Canonical terms (values) include integers, tokens, $\lambda x.t$, $\langle t_1, t_2 \rangle$, $\texttt{inl}(t)$, $\texttt{inr}(t)$, and $\texttt{Ax}$.

Non-canonical terms include (lazy) application, $t_1 t_2$, (eager) "call-by-value", $\texttt{let}\ x := t_1\ \texttt{in}\ t_2$, and general recursion, $\mathbf{fix}(t)$, as well as "spread", "decide", arithmetic operators, and others.

Howe's version of Kleene equality

From term evaluation, Howe (1996) defined a co-inductive *approximation* relation, $t_1 \leq t_2$, on terms.

Howe's version of Kleene equality

From term evaluation, Howe (1996) defined a co-inductive *approximation* relation, $t_1 \leq t_2$, on terms.

Computational equivalence $\sim$ (a congruence) is
$a \sim b \triangleq a \leq b \;\&\; b \leq a$.

Howe's version of Kleene equality

From term evaluation, Howe (1996) defined a co-inductive *approximation* relation, $t_1 \leq t_2$, on terms.

Computational equivalence $\sim$ (a congruence) is
$a \sim b \triangleq a \leq b \ \& \ b \leq a$.

Examples:

For all terms t, $\quad \bot \leq t$.

$(\lambda x.x + 1)\ 2 \sim 3$.

$\bot \sim \textbf{fix}(\lambda x.x)$.

Howe's version of Kleene equality

From term evaluation, Howe (1996) defined a co-inductive *approximation* relation, $t_1 \leq t_2$, on terms.

Computational equivalence $\sim$ (a congruence) is $a \sim b \triangleq a \leq b \;\&\; b \leq a$.

Examples:

For all terms t, $\quad \bot \leq t$.

$(\lambda x.x + 1)\ 2 \sim 3$.

$\bot \sim \mathbf{fix}(\lambda x.x)$.

The proposition "t has a value" is defined using approx and call-by-value: $\texttt{halts}(t) \triangleq \texttt{Ax} \leq (\texttt{let}\ x := t\ \texttt{in}\ \texttt{Ax})$

Nuprl Type System

is built on top of the untyped computation system.

Allen (1987) A type is a **partial equivalence relation** on closed terms.

Nupr1 Type System

is built on top of the untyped computation system.

Allen (1987) A type is a **partial equivalence relation** on closed terms.

Equality: $a =_T b$
Dependent function: $a{:}A \rightarrow B[a]$
Dependent product: $a{:}A \times B[a]$
Disjoint union: $A + B$
Universe: $\mathbb{U}_i \quad i = 0, 1, 2, \ldots$
Subtype: $A \sqsubseteq B$
Quotient: $T//E$
Intersection: $\bigcap_{a:A} .B[a]$

More Nuprl Types

Kopylov,Nogin (2006) Image: **image**(T, f)

Subset: $\{a : A \mid B[a]\} \triangleq$ **image**$(a{:}A \times B[a], \pi_1)$

squash: $\downarrow P \triangleq \{a : \text{Unit} \mid P\}$

Union: $\bigcup_{a:A} B[a] \triangleq$ **image**$(a{:}A \times B[a], \pi_2)$

More Nuprl Types

Kopylov,Nogin (2006) Image: **image**(T, f)

Subset: $\{a : A \mid B[a]\} \triangleq \textbf{image}(a{:}A \times B[a], \pi_1)$
squash: $\downarrow P \triangleq \{a : \text{Unit} \mid P\}$

Union: $\bigcup_{a:A} B[a] \triangleq \textbf{image}(a{:}A \times B[a], \pi_2)$

Smith (1989), Crary (1998) Partial types: $\overline{T}$
contains all members of T as well as all divergent terms

More Nuprl Types

Kopylov,Nogin (2006) Image: **image**(T, f)

Subset: $\{a : A \mid B[a]\} \triangleq \textbf{image}(a{:}A \times B[a], \pi_1)$

squash: $\downarrow P \triangleq \{a : \text{Unit} \mid P\}$

Union: $\bigcup_{a:A} B[a] \triangleq \textbf{image}(a{:}A \times B[a], \pi_2)$

Smith (1989), Crary (1998) Partial types: $\overline{T}$

contains all members of T as well as all divergent terms

Allen's PER semantics (extended by Smith, Crary, et.al.) defines an inductive construction of universes closed under all of these type constructors. (Defined in Coq by V. Rahli & A. Anand, ITP 2014)

Inductive types in CTT14

Types A and B are extensionally equal, $A \equiv B$, if $A \sqsubseteq B$ & $B \sqsubseteq A$.

Inductive types in CTT14

Types A and B are extensionally equal, $A \equiv B$, if $A \sqsubseteq B$ & $B \sqsubseteq A$.

Type T is a fixedpoint of F if $T \equiv F(T)$ and is the least fixedpoint if $T \sqsubseteq A$ when A is a fixedpoint of F.

Inductive types in CTT14

Types A and B are extensionally equal, $A \equiv B$, if $A \sqsubseteq B$ & $B \sqsubseteq A$.

Type T is a fixedpoint of F if $T \equiv F(T)$ and is the least fixedpoint if $T \sqsubseteq A$ when A is a fixedpoint of F.

Equivalently, T is the least fixedpoint of F when the appropriate induction principle holds.

Inductive types in CTT14

Types A and B are extensionally equal, $A \equiv B$, if $A \sqsubseteq B$ & $B \sqsubseteq A$.

Type T is a fixedpoint of F if $T \equiv F(T)$ and is the least fixedpoint if $T \sqsubseteq A$ when A is a fixedpoint of F.

Equivalently, T is the least fixedpoint of F when the appropriate induction principle holds.

Rather than add least fixedpoints (for suitable functions F) to the universes as primitive types, **we can construct them** as subtypes of co-recursive types (which we also construct.)

Inductive types in CTT14

Types A and B are extensionally equal, $A \equiv B$, if $A \sqsubseteq B$ & $B \sqsubseteq A$.

Type T is a fixedpoint of F if $T \equiv F(T)$ and is the least fixedpoint if $T \sqsubseteq A$ when A is a fixedpoint of F.

Equivalently, T is the least fixedpoint of F when the appropriate induction principle holds.

Rather than add least fixedpoints (for suitable functions F) to the universes as primitive types, **we can construct them** as subtypes of co-recursive types (which we also construct.)

The needed **induction principle follows** from Brouwer's **Bar Induction**.

Intersection Types and Corecursive Types

All the co-recursive types we need can be constructed using intersection and induction on $\mathbb{N}$

$\mathtt{Top} \triangleq \bigcap_{a:\mathrm{Void}} .\mathrm{Void}$
This is the PER $\lambda x, y.\mathrm{True}$, so for all types T, $T \sqsubseteq \mathtt{Top}$

Intersection Types and Corecursive Types

All the co-recursive types we need can be constructed using intersection and induction on $\mathbb{N}$

$\texttt{Top} \triangleq \bigcap_{a:\text{Void}}.\text{Void}$
This is the PER $\lambda x, y.\text{True}$, so for all types T, $T \sqsubseteq \texttt{Top}$

$$\texttt{corec}(G) = \bigcap_{n:\mathbb{N}}.\textbf{fix}(\lambda P.\lambda n.\, \texttt{if}\ n = 0\ \texttt{then Top}\ \ \texttt{else}\ G\ (P\ (n-1)))\ n$$

i.e. $\bigcap_{n:\mathbb{N}}.G^n(Top)$
This is the greatest fixedpoint of G if G "preserves ω-limits".

Intersection Types and Corecursive Types

All the co-recursive types we need can be constructed using intersection and induction on $\mathbb{N}$

$\mathtt{Top} \triangleq \bigcap_{a:\mathrm{Void}}.\mathrm{Void}$
This is the PER $\lambda x, y.\mathrm{True}$, so for all types T, $T \sqsubseteq \mathtt{Top}$

$$\mathtt{corec}(G) = \bigcap_{n:\mathbb{N}}.\mathbf{fix}(\lambda P.\lambda n.\mathtt{if}\ n = 0\ \mathtt{then}\ \mathtt{Top}\ \mathtt{else}\ G\ (P\ (n-1)))\ n$$

i.e. $\bigcap_{n:\mathbb{N}}.G^n(Top)$
This is the greatest fixedpoint of G if G “preserves ω-limits”.

Aside: $\bigcap_{x:T}.P(x)$ is “uniform” all quantifier, $\forall[x{:}T].P(x)$.
We showed completeness for intuitionistic minimal logic:
$\vdash_{IML} \phi \Leftrightarrow \forall[M].M \models \phi$.

Algebraic Datatypes

For the least fixedpoint $DT \equiv F(DT)$ of an "algebraic" function F, there is a natural size function $size \in DT \to \mathbb{N}$.

Algebraic Datatypes

For the least fixedpoint $DT \equiv F(DT)$ of an "algebraic" function F, there is a natural size function $size \in DT \to \mathbb{N}$.

On $coDT = \texttt{corec}(F)$ the same function has type $coDT \to \overline{\mathbb{N}}$.

Algebraic Datatypes

For the least fixedpoint $DT \equiv F(DT)$ of an "algebraic" function F, there is a natural size function $size \in DT \to \mathbb{N}$.

On $coDT = \texttt{corec}(F)$ the same function has type $coDT \to \overline{\mathbb{N}}$.

Termination: $t \in \overline{T}, \texttt{halts}(t) \models t \in T$

Algebraic Datatypes

For the least fixedpoint $DT \equiv F(DT)$ of an "algebraic" function F, there is a natural size function $size \in DT \to \mathbb{N}$.

On $coDT = \texttt{corec}(F)$ the same function has type $coDT \to \overline{\mathbb{N}}$.

Termination: $t \in \overline{T}, \texttt{halts}(t) \models t \in T$

So we can construct DT as $\{t : coDT \mid \texttt{halts}(size(t))\}$ and get the induction on DT from induction on $size$.

Algebraic Datatypes

For the least fixedpoint $DT \equiv F(DT)$ of an "algebraic" function F, there is a natural size function $size \in DT \to \mathbb{N}$.

On $coDT = \mathtt{corec}(F)$ the same function has type $coDT \to \overline{\mathbb{N}}$.

Termination: $t \in \overline{T}, \mathtt{halts}(t) \models t \in T$

So we can construct DT as $\{t : coDT \mid \mathtt{halts}(size(t))\}$ and get the induction on DT from induction on *size*.

The definition of $list(T)$ in Nuprl is now
$\{L : colist(T) \mid \mathtt{halts}(length(L))\}$
where
$colist(T) \triangleq \mathtt{corec}(\lambda L.\text{Unit} \cup T \times L)$

W-types and parameterized families of W-types

We want to construct least fixedpoints
$W(A; a.B[a]) \equiv a{:}A \times (B[a] \rightarrow W(A; a.B[a]))$

W-types and parameterized families of W-types

We want to construct least fixedpoints
$W(A; a.B[a]) \equiv a{:}A \times (B[a] \rightarrow W(A; a.B[a])$

and, more generally, a parameterized family of W-types:

$$pW(p.A[p]; p, a.B[p, a]; p, a, b.C[p, a, b]) \equiv$$
$$\lambda p.\ a{:}A[p] \times (b{:}B[p, a] \rightarrow pW(C[p.a.b]))$$

W-types and parameterized families of W-types

We want to construct least fixedpoints
$W(A; a.B[a]) \equiv a{:}A \times (B[a] \to W(A; a.B[a]))$

and, more generally, a parameterized family of W-types:

$$pW(p.A[p]; p, a.B[p, a]; p, a, b.C[p, a, b]) \equiv \\ \quad \lambda p.\ a{:}A[p] \times (b{:}B[p, a] \to pW(C[p.a.b]))$$

We can't define a size function and use induction on $\mathbb{N}$, but we can make an "analogous" construction and get the induction principle from Bar Induction.
(For simplicity, we discuss W rather than pW.)

W-types and parameterized families of W-types

We want to construct least fixedpoints
$W(A; a.B[a]) \equiv a{:}A \times (B[a] \rightarrow W(A; a.B[a]))$

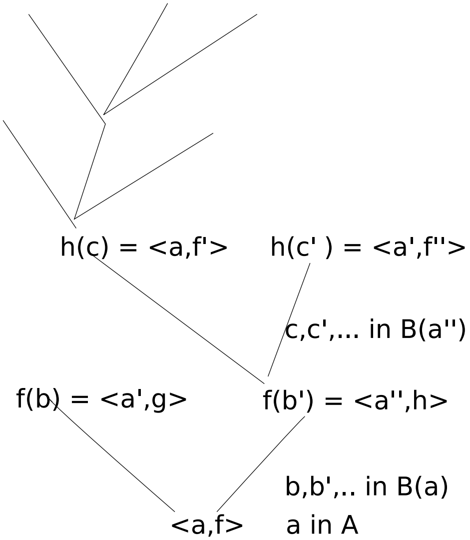
and, more generally, a parameterized family of W-types:

$$pW(p.A[p]; p, a.B[p, a]; p, a, b.C[p, a, b]) \equiv \\ \quad \lambda p.\ a{:}A[p] \times (b{:}B[p, a] \rightarrow pW(C[p.a.b]))$$

We can't define a size function and use induction on $\mathbb{N}$, but we can make an "analogous" construction and get the induction principle from Bar Induction.
(For simplicity, we discuss W rather than pW.)

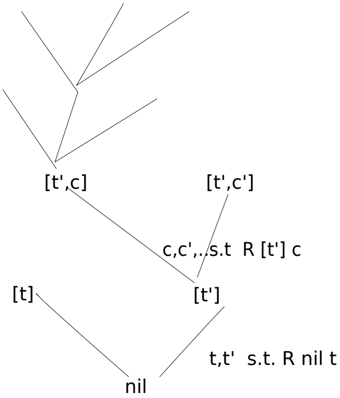
Basic idea: $W = \{w : \text{co-}W \mid \text{paths starting at } w \text{ are finite}\}$

W-type picture



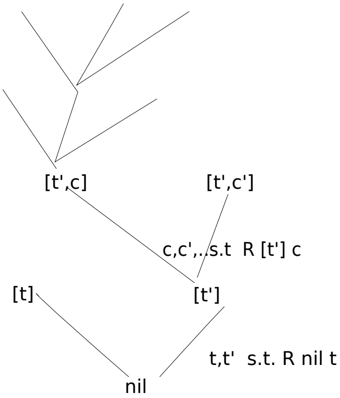
$$W(A; a.B[a]) \equiv a{:}A \times (B[a] \rightarrow W(A; a.B[a]))$$

Bar Induction in pictures



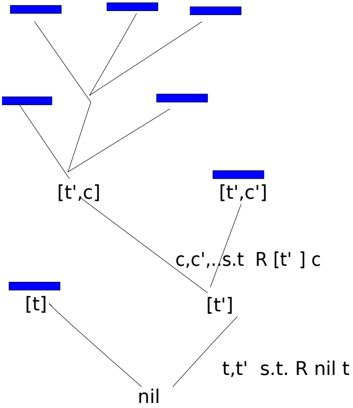
R is the spread law.

Bar Induction in pictures

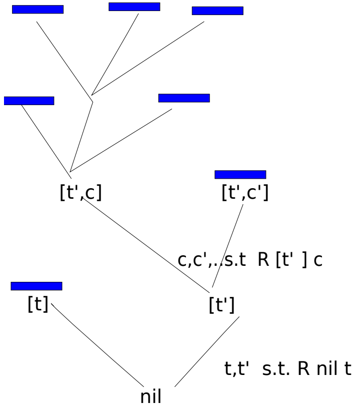


R is the spread law. **If** (1) **every path is barred**.

Bar Induction in pictures

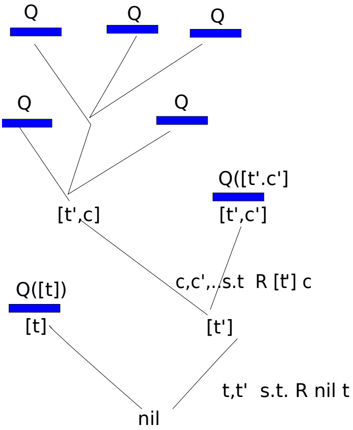


Bar Induction in pictures

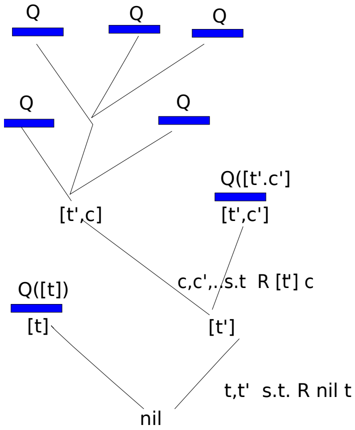


And if Base case: $B(s) \Rightarrow Q(s)$

Bar Induction in pictures

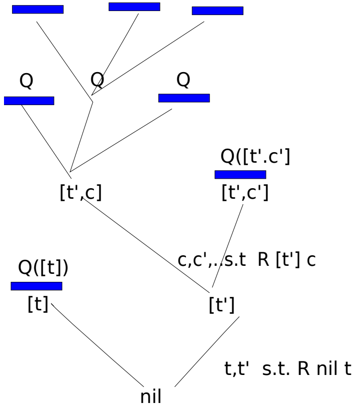


Bar Induction in pictures



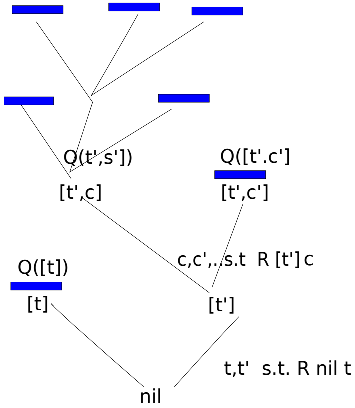
and if Induction step: $(\forall t.R(s,t) \Rightarrow Q(s \oplus t)) \Rightarrow Q(s)$

Bar Induction in pictures



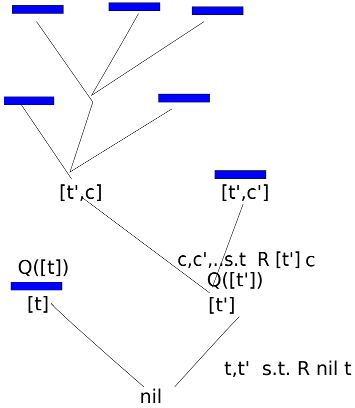
Induction step: $(\forall t.R(s,t) \Rightarrow Q(s \oplus t)) \Rightarrow Q(s)$

Bar Induction in pictures



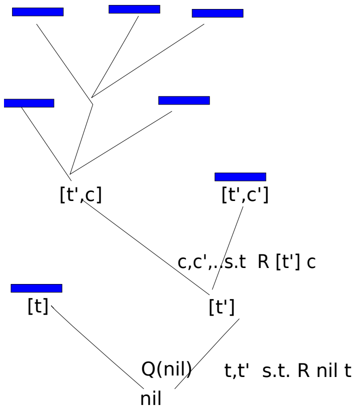
Induction step: $(\forall t.R(s,t) \Rightarrow Q(s \oplus t)) \Rightarrow Q(s)$

Bar Induction in pictures

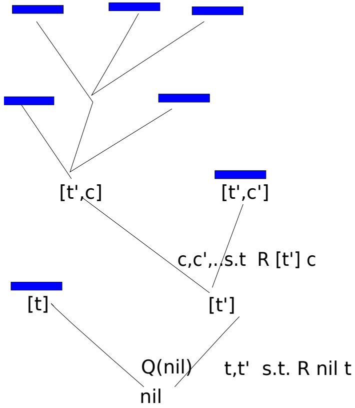


Induction step: $(\forall t. R(s,t) \Rightarrow Q(s \oplus t)) \Rightarrow Q(s)$

Bar Induction in pictures



Bar Induction in pictures



Then: Q(nil)

Bar Induction, preliminaries

Brouwer's bar induction principle, (explicated by Kleene), is about "spreads" of finite sequences (of some type T).

Bar Induction, preliminaries

Brouwer's bar induction principle, (explicated by Kleene), is about "spreads" of finite sequences (of some type T).

We use $s \in V_k(T) \triangleq \{i{:}\mathbb{N} \mid i < k\} \rightarrow T$ for a sequence s of length k, and $s \oplus_k t$ for the sequence of length $k + 1$ with t appended.

Bar Induction, preliminaries

Brouwer's bar induction principle, (explicated by Kleene), is about "spreads" of finite sequences (of some type T).

We use $s \in V_k(T) \triangleq \{i{:}\mathbb{N} \mid i < k\} \rightarrow T$ for a sequence s of length k, and $s \oplus_k t$ for the sequence of length $k+1$ with t appended.

A relation $R \in k{:}\mathbb{N} \rightarrow V_k(T) \rightarrow T \rightarrow \mathbb{P}$ is a "spread law" and s is consistent, $\mathrm{con}(R, k, s)$, if $\forall i < k.\ R(i, s, s(i))$. A function $f \in \mathbb{N} \rightarrow T$ is a *path*, $\mathrm{Path}(R, f)$, if $\forall i.\ R(i, f, f(i))$.

Bar Induction, preliminaries

Brouwer's bar induction principle, (explicated by Kleene), is about "spreads" of finite sequences (of some type T).

We use $s \in V_k(T) \triangleq \{i{:}\mathbb{N} \mid i < k\} \to T$ for a sequence s of length k, and $s \oplus_k t$ for the sequence of length $k+1$ with t appended.

A relation $R \in k{:}\mathbb{N} \to V_k(T) \to T \to \mathbb{P}$ is a "spread law" and s is consistent, $\mathrm{con}(R, k, s)$, if $\forall i < k.\ R(i, s, s(i))$. A function $f \in \mathbb{N} \to T$ is a *path*, $\mathrm{Path}(R, f)$, if $\forall i.\ R(i, f, f(i))$.

We state the bar induction rule only for expressions $Q(k, s)$ of the form $a(k, s) \in X(k, s)$ with witness $\mathtt{Ax}$.

Bar Induction, preliminaries

Brouwer's bar induction principle, (explicated by Kleene), is about "spreads" of finite sequences (of some type T).

We use $s \in V_k(T) \triangleq \{i{:}\mathbb{N} \mid i < k\} \to T$ for a sequence s of length k, and $s \oplus_k t$ for the sequence of length $k+1$ with t appended.

A relation $R \in k{:}\mathbb{N} \to V_k(T) \to T \to \mathbb{P}$ is a "spread law" and s is consistent, $\mathrm{con}(R,k,s)$, if $\forall i < k.\ R(i,s,s(i))$. A function $f \in \mathbb{N} \to T$ is a *path*, $\mathrm{Path}(R,f)$, if $\forall i.\ R(i,f,f(i))$.

We state the bar induction rule only for expressions $Q(k,s)$ of the form $a(k,s) \in X(k,s)$ with witness $\mathtt{Ax}$.

Bar Induction works "toward the root" from the hypothesis $\mathrm{ind}(R,T,Q,k,s) \triangleq \forall t{:}\{t : T \mid R(k,s,t)\}.\ Q(k+1, s \oplus t)$

Bar Induction Rule

$$\frac{\begin{array}{c} H \vdash T \in \text{Type} \qquad H,\ k{:}\mathbb{N},\ s{:}V_k(T),\ t{:}T \vdash R(k,s,t) \in \text{Type} \\ H,\ k{:}\mathbb{N},\ s{:}V_k(T),\ con(R,k,s) \vdash B(k,s)\ \vee\ \neg B(k,s) \\ H,\ f{:}\mathbb{N} \to T,\ \text{Path}(R,f) \vdash\downarrow \exists n{:}\mathbb{N}.\ B(n,f) \\ H,\ k{:}\mathbb{N},\ s{:}V_k(T),\ con(R,k,s),\ B(k,s) \vdash Q(k,s) \\ H,\ k{:}\mathbb{N},\ s{:}V_k(T), con(R,k,s),\ ind(R,T,Q,k,s) \vdash Q(k,s) \end{array}}{H \vdash Q(0,z)}$$

The first two premises prove that R is a spread law. The next two premises prove that B is a decidable bar on the spread. The fifth and sixth premises are the base and induction steps of the proof by bar induction for the term $Q(0,z)$.

The construction

Let $cW = \text{co-}W(A, a.B[a])$
For $w \in cW$, $w = \langle a, f \rangle$ where $a \in A$, $f \in B[a] \rightarrow cW$

The construction

Let $cW = \text{co-}W(A, a.B[a])$
For $w \in cW$, $w = \langle a, f \rangle$ where $a \in A$, $f \in B[a] \rightarrow cW$

A path will have steps of type
$$T_{A,B} \triangleq \langle a, f \rangle{:}cW \times (B(a) + \text{Unit})$$

The construction

Let $cW = \text{co-}W(A, a.B[a])$
For $w \in cW$, $w = \langle a, f \rangle$ where $a \in A$, $f \in B[a] \to cW$

A path will have steps of type

$$T_{A,B} \triangleq \langle a, f \rangle : cW \times (B(a) + \text{Unit})$$

The spread law $R(k, s, t)$ is defined to hold when, if the last step in s is $\langle \langle a, f \rangle, \text{inl}(b) \rangle$ then $\pi_1(t) = f(b)$.

The construction

Let $cW = \text{co-}W(A, a.B[a])$
For $w \in cW$, $w = \langle a, f \rangle$ where $a \in A$, $f \in B[a] \to cW$

A path will have steps of type
$$T_{A,B} \triangleq \langle a, f \rangle{:}cW \times (B(a) + \text{Unit})$$

The spread law $R(k, s, t)$ is defined to hold when, if the last step in s is $\langle\langle a, f \rangle, \text{inl}(b)\rangle$ then $\pi_1(t) = f(b)$.

A path $g \in \mathbb{N} \to T_{A,B}$ starts at w if $\pi_1(g(0)) = w$.

The construction

Let $cW = \text{co-}W(A, a.B[a])$
For $w \in cW$, $w = \langle a, f \rangle$ where $a \in A$, $f \in B[a] \to cW$

A path will have steps of type

$$T_{A,B} \triangleq \langle a, f \rangle{:}cW \times (B(a) + \text{Unit})$$

The spread law $R(k, s, t)$ is defined to hold when, if the last step in s is $\langle\langle a, f \rangle, \text{inl}(b)\rangle$ then $\pi_1(t) = f(b)$.

A path $g \in \mathbb{N} \to T_{A,B}$ starts at w if $\pi_1(g(0)) = w$.

The path is barred if $\downarrow \exists n : \mathbb{N}.\ \text{isr}(\pi_2(g(n)))$.

The construction

Let $cW = \text{co-}W(A, a.B[a])$
For $w \in cW$, $w = \langle a, f \rangle$ where $a \in A$, $f \in B[a] \to cW$

A path will have steps of type
$$T_{A,B} \triangleq \langle a, f \rangle{:}cW \times (B(a) + \text{Unit})$$

The spread law $R(k, s, t)$ is defined to hold when, if the last step in s is $\langle\langle a, f \rangle, \text{inl}(b)\rangle$ then $\pi_1(t) = f(b)$.

A path $g \in \mathbb{N} \to T_{A,B}$ starts at w if $\pi_1(g(0)) = w$.

The path is barred if $\downarrow \exists n : \mathbb{N}.\ \text{isr}(\pi_2(g(n)))$.

So, we define
$W \triangleq \{w : cW \mid$ every path g stating at w is barred$\}$

The result

The induction principle $\mathrm{Ind}(W, P)$ for W is

$$(\forall a{:}A.\ \forall f{:}B[a] \to W.$$
$$(\forall b{:}B[a].\ P(f(b))) \Rightarrow P(\langle a, f\rangle)) \Rightarrow (\forall w{:}W.\ P(w))$$

The result

The induction principle $\text{Ind}(W, P)$ for W is

$(\forall a{:}A.\ \forall f{:}B[a] \to W.$
$(\forall b{:}B[a].\ P(f(b))) \Rightarrow P(\langle a, f\rangle)) \Rightarrow (\forall w{:}W.\ P(w))$

We use the Bar Induction Rule to prove that
$\lambda H.\lambda w.\ \mathbf{fix}(\lambda G.\lambda w.\ \text{let } a, f = w \text{ in } H(a, f, \lambda b.G(f(b))))w$
$\in \text{Ind}(W, P)$

The result

The induction principle $\text{Ind}(W, P)$ for W is

$$(\forall a{:}A.\ \forall f{:}B[a] \rightarrow W.$$
$$(\forall b{:}B[a].\ P(f(b))) \Rightarrow P(\langle a, f \rangle)) \Rightarrow (\forall w{:}W.\ P(w))$$

We use the Bar Induction Rule to prove that
$\lambda H.\lambda w.\ \mathbf{fix}(\lambda G.\lambda w.\ \text{let } a, f = w \text{ in } H(a, f, \lambda b.G(f(b))))w$
$\in \text{Ind}(W, P)$

(suitably generalized for the more general case of the parameterized family $pW(A, B, C)$)

Primitive Inductive types not needed

In the abstract we wrote: "we *could* replace all the primitive rec-types"

Primitive Inductive types not needed

In the abstract we wrote: "we *could* replace all the primitive rec-types"

Since then, we *have constructed* all recursive types with one of these two constructions (that use a subtype of a co-recursive type).

Primitive Inductive types not needed

In the abstract we wrote: "we *could* replace all the primitive rec-types"

Since then, we *have constructed* all recursive types with one of these two constructions (that use a subtype of a co-recursive type).

We redefined the necessary tactics for induction and our code for generating algebraic datatypes.

Primitive Inductive types not needed

In the abstract we wrote: "we *could* replace all the primitive rec-types"

Since then, we *have constructed* all recursive types with one of these two constructions (that use a subtype of a co-recursive type).

We redefined the necessary tactics for induction and our code for generating algebraic datatypes.

Then we "deactivated" the rules for the primitive rec-type (Nuprl is a logical framework and interprets the rules in its library).
Everything in the library (about 15K lemmas) was rebuilt.
About two weeks work.

Primitive Inductive types not needed

In the abstract we wrote: “we *could* replace all the primitive rec-types”

Since then, we *have constructed* all recursive types with one of these two constructions (that use a subtype of a co-recursive type).

We redefined the necessary tactics for induction and our code for generating algebraic datatypes.

Then we “deactivated” the rules for the primitive rec-type (Nuprl is a logical framework and interprets the rules in its library).
Everything in the library (about 15K lemmas) was rebuilt.
About two weeks work.

So, induction on $\mathbb{N}$ and Bar Induction are the only induction principles we need.

Further Reading

S.C. Kleene and R. E. Vesley, Foundations of Inuitionistic Mathematics. 1966 (breakthrough document that inspired Martin-Lof, and others)

Stuart F. Allen, A Non-Type-Theoretic Semantics for Type-Theoretic Language. 1987

Karl Crary, Type-Theoretic Methodology for Practical Programming Languages. 1998

Scott F Smith, Partial Objects in Type Theory. 1989

Constable & Smith. Computational Foundations of Basic Recursive function Theory. 1993

Stuart F. Allen, An Abstract Semantics for Atoms in Nuprl. 2006