# Inductive-Inductive Definitions

Anton Setzer
(Joint work with Fredrik Forsberg)

Swansea University, Swansea UK

5 June 2012

# Fredrik Nordvall Forsberg

# Preliminary Remarks

- Type Theory is only the syntactic framework.
  Induction-induction and induction-recursion not necessarily bound to this framework.

# Type Theory

- Judgements:

$$\Gamma \Rightarrow \mathrm{Context}$$
$$\Gamma \Rightarrow A : \mathrm{Set} \qquad \Gamma \Rightarrow A = B : \mathrm{Set}$$
$$\Gamma \Rightarrow r : A \qquad \Gamma \Rightarrow r = s : A$$

- Some Rules:

$$\emptyset : \mathrm{Context} \qquad \frac{\Gamma \Rightarrow A : \mathrm{Set}}{\Gamma, x : A \Rightarrow \mathrm{Context}}$$

$$\frac{\Gamma, x : A \Rightarrow B : \mathrm{Set}}{\Gamma \Rightarrow (\Sigma x : A.B) : \mathrm{Set}}$$

# Simplifications

- Logical Framework:
  - Allows to form e.g.

$$A \to \mathrm{Set} : \mathrm{Type}$$
$$((x : A) \to B\ x \to \mathrm{Set}) : \mathrm{Type}$$

- With the logical framework, rules for $\Sigma$ becomes

$$\Sigma : (A : \mathrm{Set}) \to (B : A \to \mathrm{Set}) \to \mathrm{Set}$$

- That's how it occurs in theorem provers (Alf, Half, Agda, Coq, NuPrl).

# Defining Semantics using Induction-Recursion

- Formulate Semantics of Type Theory inside Type Theory.
- So we formulate in type theory a model $(\widehat{\mathrm{Set}}, [\![\ ]\!])$ of a weaker type theory.
- Done by defining
  - A set $\widehat{\mathrm{Set}}$ of codes for elements of $\mathrm{Set}$ inductively
  - a function $[\![\ ]\!] : \widehat{\mathrm{Set}} \to \mathrm{Set}$ recursively.

# Defining Semantics using Induction-Recursion

- Define inductive-recursively

$$\widehat{\mathrm{Set}} : \mathrm{Set} \qquad\qquad [\![\ \ ]\!] : \widehat{\mathrm{Set}} \to \mathrm{Set}$$

- Rule for Σ:

$$\Sigma : (A : \mathrm{Set}) \to (B : A \to \mathrm{Set}) \to \mathrm{Set}$$

is reflected into

$$\widehat{\Sigma} : (a : \widehat{\mathrm{Set}}) \to (b : [\![\, a \,]\!] \to \widehat{\mathrm{Set}}) \to \widehat{\mathrm{Set}}$$
$$[\![\, \widehat{\Sigma}\ a\ b \,]\!] = \Sigma\ [\![\, a \,]\!]\ (\lambda x.[\![\, b\ x \,]\!]) : \mathrm{Set}$$

# From Induction-Recursion to Induction-Induction

- General induction-recursion:
  - Define $A : \mathrm{Set}$ inductively,
  - while defining a function $B : A \to \mathrm{Set}$ recursively. ($\mathrm{Set}$ can be generalised to types).
- Induction-induction:
  Instead of defining $B$ recursively define $B$ inductively.
  So we define simultaneously
  - $A : \mathrm{Set}$ inductively,
  - $B : A \to \mathrm{Set}$ inductively.

# Defining Syntax using Induction-Induction

- Formulate Syntax of Type Theory inside Type Theory (Nils Danielsson)
- Define inductively simultaneously:
  - $\widehat{\mathrm{Context}} : \mathrm{Set}.$
    - $\Gamma : \widehat{\mathrm{Context}}$ represents $\Gamma \Rightarrow \mathrm{Context}.$
  - $\widehat{\mathrm{Set}} : \widehat{\mathrm{Context}} \to \mathrm{Set}.$
    - $A : \widehat{\mathrm{Set}}\ \Gamma$ represents $\Gamma \Rightarrow A : \mathrm{Set}.$
  - $\widehat{\mathrm{Term}} : (\Gamma : \widehat{\mathrm{Context}}) \to (A : \widehat{\mathrm{Set}}\ \Gamma) \to \mathrm{Set}.$
    - $r : \widehat{\mathrm{Term}}\ \Gamma\ A$ represents $\Gamma \Rightarrow r : A.$
  - $\widehat{\mathrm{SynSet}}_{=} : (\Gamma : \widehat{\mathrm{Context}}) \to (A, B : \widehat{\mathrm{Set}}\ \Gamma) \to \mathrm{Set}.$
    - $p : \widehat{\mathrm{SynSet}}_{=}\ \Gamma\ A\ B$ represents a derivation of $\Gamma \to A = B : \mathrm{Set}.$
  - etc.

# Representation of Rules

- Rule
  $$\emptyset : \mathrm{Context}$$
  represented as
  $$\widehat{\emptyset} : \widehat{\mathrm{Context}}$$
- Rule
  $$\frac{\Gamma \Rightarrow A : \mathrm{Set}}{\Gamma, x : A \Rightarrow \mathrm{Context}}$$
  represented (variable-free)
  $$\_\widehat{::}\_ : (\Gamma : \widehat{\mathrm{Context}}) \to (A : \widehat{\mathrm{Set}}\ \Gamma) \to \widehat{\mathrm{Context}}$$
  where we write $\Gamma \mathbin{\widehat{::}} A$ for $\_\widehat{::}\_\ \Gamma\ A$.

# Representation of Rules

- Rule

$$\frac{\Gamma, x : A \Rightarrow B : \mathrm{Set}}{\Gamma \Rightarrow \Sigma x : A.B : \mathrm{Set}}$$

which in full reads

$$\frac{\Gamma : \mathrm{Context} \qquad \Gamma \Rightarrow A : \mathrm{Set} \qquad \Gamma, x : A \Rightarrow B : \mathrm{Set}}{\Gamma \Rightarrow \Sigma x : A.B : \mathrm{Set}}$$

is represented as

$$\begin{aligned}\widehat{\Sigma} : & (\Gamma : \widehat{\mathrm{Context}}) \\ & \to (A : \widehat{\mathrm{Set}}\ \Gamma) \\ & \to (B : \widehat{\mathrm{Set}}\ (\Gamma \mathbin{\widehat{::}} A)) \\ & \to \widehat{\mathrm{Set}}\ \Gamma\end{aligned}$$

# Observation

- We define simultaneously
  - $\widehat{\mathrm{Context}} : \mathrm{Set}$ inductively,
  - $\widehat{\mathrm{Set}} : \widehat{\mathrm{Context}} \to \mathrm{Set}$ inductively,
  - $\widehat{\mathrm{Term}} : (\Gamma : \widehat{\mathrm{Context}}) \to \widehat{\mathrm{Set}}\ \Gamma \to \mathrm{Set}$ inductively.
  - $\cdots$
- Here restriction to only 2 levels, we define
  - $A : \mathrm{Set}$
  - $B : A \to \mathrm{Set}$

  inductive-inductively.

# Observation

- In
  - $A : \mathrm{Set}$
  - $B : A \to \mathrm{Set}$

  the constructor of $B\ x$ might refer to the constructor of $A$.
- For instance in

$$\begin{aligned}\widehat{\Sigma} : (\Gamma : \widehat{\mathrm{Context}}) \\ \to (A : \widehat{\mathrm{Set}}\ \Gamma) \\ \to (B : \widehat{\mathrm{Set}}\ (\Gamma\ \widehat{::}\ A)) \\ \to \widehat{\mathrm{Set}}\ \Gamma\end{aligned}$$

  the second argument refers to the constructor $\_\widehat{::}\_$ for $\widehat{\mathrm{Set}}$.

# Induction-Induction is not Indexed Induction

- In indexed inductive definitions
  - we have a given $I : \mathrm{Set}$
  - and define sets $A : I \to \mathrm{Set}$ inductively simultaneously.
- In induction-induction
  - the index set $A : \mathrm{Set}$ is defined simultaneously inductively with $B : A \to \mathrm{Set}$.

# Induction-Induction is not Induction-Recursion

- For a constructor

$$C\ a\ b : A$$

we have no recursive equation:

$$B\ (C\ a\ b) = \cdots$$

- In fact constructors for $A$ and constructors for $B$ are not necessarily connected.
- However constructors of $B$ might refer to constructors of $A$.
- $B : A \rightarrow \mathrm{Set}$ is defined inductively not recursively.
- Constructors of $A, B$ can refer to $B$ only strictly positively.

# Ordinal Notation System

- Typical definition:
  - The set of pre ordinals $\mathrm{T}$ is defined inductively by:
    - If $a_1, \ldots, a_k \in \mathrm{T}$ and $n_1, \ldots, n_k \in \mathbb{N} \setminus \{0\}$ then

$$\omega^{a_1} n_1 + \cdots + \omega^{a_k} n_k \in \mathrm{T}$$

  - We define $\prec$ on $\mathrm{T}$ recursively by

$$\omega^{a_1} n_1 + \cdots + \omega^{a_k} n_k \prec \omega^{b_1} m_1 + \cdots + \omega^{b_l} m_l$$

iff

$$(a_1, n_1, \ldots, a_k, n_k) \prec_{\mathrm{lex}} (b_1, m_1, \ldots, b_l, m_l)$$

  - We define $\mathrm{OT} \subseteq \mathrm{T}$ inductively:
    - If $a_1, \ldots, a_k \in \mathrm{OT}$ and $a_k \prec \cdots \prec a_1$ and $n_1, \ldots, n_k \in \mathbb{N} \setminus \{0\}$ then

$$\omega^{a_1} n_1 + \cdots + \omega^{a_k} n_k \in \mathrm{OT}$$

# Definition of $\mathrm{OT}$ Inductive-Inductively

- Define $\mathrm{OT} : \mathrm{Set}$ and $\prec : \mathrm{OT} \to \mathrm{OT} \to \mathrm{Set}$ inductive-inductively:
  - If $a_1, \ldots, a_k \in \mathrm{OT}$ and $a_k \prec \cdots \prec a_1$ and $n_1, \ldots, n_k \in \mathbb{N} \setminus \{0\}$ then

$$\omega^{a_1} n_1 + \cdots + \omega^{a_k} n_k \in \mathrm{OT}$$

  - If

$$\begin{array}{l} \omega^{a_1} n_1 + \cdots + \omega^{a_k} n_k \\ \omega^{b_1} m_1 + \cdots + \omega^{b_l} m_l \in \mathrm{OT} \end{array}$$

    and

$$(a_1, n_1, \ldots, a_k, n_k) \prec_{\mathrm{lex}} (b_1, m_1, \ldots, b_l, m_l)$$

    then

$$\omega^{a_1} n_1 + \cdots + \omega^{a_k} n_k \prec \omega^{b_1} m_1 + \cdots + \omega^{b_l} m_l$$

# Conway's Surreal Numbers

- Like Dedekind cuts, but replacing rationals by previously defined surreal numbers.
- Surreal numbers contain all ordered fields.
- Definition in set theory.
- Definition of the class of surreal numbers $\mathrm{Surreal}$ together with an ordering $\leq$:
  - If $X_L, X_R \in \mathcal{P}(\mathrm{Surreal})$ such that
  $$\neg\exists x_L \in X_L.\exists x_R \in X_R.x_R \leq x_L$$
  then $(X_L, X_R) \in \mathrm{Surreal}$
  - $X = (X_L, X_R) \leq (Y_L, Y_R) = Y$ iff
    - $\neg\exists x_L \in X_L.Y \leq x_L$
    - $\neg\exists y_R \in Y_R.y_R \leq X$

# Surreal Numbers as an Inductive-Inductive Definition

- Define simultaneously inductively

$$\begin{array}{lcl} \text{Surreal} & : & \text{Set} \\ \_\leq\_ & : & \text{Surreal} \to \text{Surreal} \to \text{Set} \\ \_\not\leq\_ & : & \text{Surreal} \to \text{Surreal} \to \text{Set} \end{array}$$

- $\mathcal{P}(\text{Surreal})$ replaced by $\Sigma a : \text{U}.\text{T}\ a \to \text{Surreal}$ for some universe $\text{U}$.
- We refer to this and $x \in X_L$ informally.

# Inductive-Inductive Definition of Surreal

- If $X_L, X_R \in \mathcal{P}(\mathrm{Surreal})$, and

$$p : \forall x_L \in X_L . \forall x_R \in X_R . x_R \not\leq x_L$$

then $(X_L, X_R)_p : \mathrm{Surreal}$.

- Assume $X = (X_L, X_R)_p, Y = (Y_L, Y_R)_q : \mathrm{Surreal}$.
Assume

$$\forall x_L \in X_L . Y \not\leq x_L$$
$$\forall y_R \in Y_R . y_R \not\leq X$$

then $X \leq Y$.

# Inductive-Inductive Definition of Surreal

- Assume $X = (X_L, X_R)_p, Y = (Y_L, Y_R)_q : \mathrm{Surreal}$.
  - If
    $$\exists x_L \in X_L . Y \leq x_L$$
    then $X \not\leq Y$.
  - If
    $$\exists y_R \in Y_R . y_R \leq X$$
    then $X \not\leq Y$.

# Inductive-Inductive Definitions in Mathematics

- Inductive-inductive definitions seem to be very frequent in mathematics.
- Usually reduced to inductive definitions by
  - first defining simultanteously inductively $Apre : \mathrm{Set}$, $Bpre : \mathrm{Set}$ by ignoring dependencies of $B$ on $A$.
  - then selecting $A \subseteq Apre$, $B \subseteq Bpre$ by selecting those elements which fulfil the correct rules.
- Seems to be a general method of reducing inductive-inductive definitions to inductive definitions (work in progress).

# Plan

- We define as for inductive-inductive definitions a closed formalisation.
- Complicated since it will define not just examples but all inductive-inductive definitions in one set of rules.

# Main Idea

- We define
  - a set
$$\mathrm{SP}^0_{\mathrm{A}} : \mathrm{Set}$$
    of codes for inductive definitions for $A$,
  - a set
$$\mathrm{SP}^0_{\mathrm{B}} : \mathrm{SP}^0_{\mathrm{A}} \to \mathrm{Set}$$
    of codes for inductive definitions for $B$.
  - the set of arguments for the constructor of $A$:
$$\mathrm{Arg}^0_{\mathrm{A}} : \mathrm{SP}^0_{\mathrm{A}} \to (X : \mathrm{Set}) \to (Y : X \to \mathrm{Set}) \to \mathrm{Set}$$

# Main Idea

- the set of arguments and indices for the constructor of $B$:

$$
\begin{array}{lll}
\mathrm{Arg}^0_{\mathrm{B}} & : & (\gamma_A : \mathrm{SP}^0_{\mathrm{A}}) \\
 & & \to (X : \mathrm{Set}) \\
 & & \to (Y : X \to \mathrm{Set}) \\
 & & \to (\mathit{intro}_A : \mathrm{Arg}^0_{\mathrm{A}}\ \gamma_A\ X\ Y \to X) \\
 & & \to (\gamma_B : \mathrm{SP}^0_{\mathrm{B}}) \\
 & & \to \mathrm{Set} \\
 & & \\
\mathrm{Index}^0_{\mathrm{B}} & : & \cdots \text{same arguments as for } \mathrm{Arg}^0_{\mathrm{B}} \cdots \\
 & & \to \mathrm{Arg}^0_{\mathrm{B}}\ \gamma_A\ X\ Y\ \mathit{intro}_A\ \gamma_B \\
 & & \to X
\end{array}
$$

# Rules for the Inductive-Inductively Defined Set

- Assume $\gamma_A : \mathrm{SP}^0_{\mathrm{A}}$, $\gamma_B : \mathrm{SP}^0_{\mathrm{B}}\ \gamma_A$.
  Let $\gamma := (\gamma_A, \gamma_B)$.
- Formation rules

$$\mathrm{A}_\gamma : \mathrm{Set} \qquad \mathrm{B}_\gamma : \mathrm{A}_\gamma \to \mathrm{Set}$$

- Introduction rule for $\mathrm{A}_\gamma$:

$$\mathrm{introA}_\gamma : \mathrm{Arg}^0_{\mathrm{A}}\ \gamma_A\ \mathrm{A}_\gamma\ \mathrm{B}_\gamma \to \mathrm{A}_\gamma$$

- Introduction rule for $\mathrm{B}_\gamma$:

$$\begin{aligned}\mathrm{introB}_\gamma : (\mathit{arg} : \mathrm{Arg}^0_{\mathrm{B}}\ \gamma_A\ \mathrm{A}_\gamma\ \mathrm{B}_\gamma\ \mathrm{intro}_\gamma\ \gamma_B) \\ \to \mathrm{B}_\gamma\ (\mathrm{Index}^0_{\mathrm{B}}\ \gamma_A\ \mathrm{A}_\gamma\ \mathrm{B}_\gamma\ \mathrm{intro}_\gamma\ \gamma_B\ \mathit{arg})\end{aligned}$$

# Definition of $\mathrm{SP}_{\mathrm{A}}$

- Instead of defining $\mathrm{SP}^0_{\mathrm{A}}$ we define a more general set

$$\mathrm{SP}_{\mathrm{A}} : (A_{ref} : \mathrm{Set}) \to \mathrm{Type}$$

  which refers to elements $A_{ref}$ of the set to be defined already referred to in inductive arguments.
- Then

$$\mathrm{SP}^0_{\mathrm{A}} := \mathrm{SP}_{\mathrm{A}}\ \emptyset$$

# Constructors for $\mathrm{SP}_{\mathrm{A}}$

- Initial case: constructor with no arguments:

$$\mathrm{nil} : \mathrm{SP}_{\mathrm{A}}\ A_{ref}$$

- One non-inductive argument of type $K$ followed by other arguments given by $\gamma$:

$$\mathrm{non}-\mathrm{ind} : (K : \mathrm{Set}) \to (\gamma : K \to \mathrm{SP}_{\mathrm{A}}\ A_{ref}) \to \mathrm{SP}_{\mathrm{A}}\ A_{ref}$$

- Inductive arguments of type $A$ indexed over a set $K$ followed by arguments (which can refer to these arguments) given by $\gamma$:

$$\mathrm{A}{-}\mathrm{ind} : (K : \mathrm{Set}) \to (\gamma : \mathrm{SP}_{\mathrm{A}}\ (A_{ref} + K)) \to \mathrm{SP}_{\mathrm{A}}\ A_{ref}$$

# Constructors for $\mathrm{SP}_{\mathrm{A}}$

- Inductive arguments of type $B$ indexed over a set $K$;
  we need to have the indices for $B$, for which we use a function
  $\mathit{index} : K \to A_{\mathit{ref}}$;
  later arguments are given by $\gamma$:

$$
\begin{aligned}
\mathrm{B{-}ind} : {} & (K : \mathrm{Set}) \\
& \to (\mathit{index} : K \to A_{\mathit{ref}}) \\
& \to (\gamma : \mathrm{SP}_{\mathrm{A}}\ A_{\mathit{ref}}) \\
& \to \mathrm{SP}_{\mathrm{A}}\ A_{\mathit{ref}}
\end{aligned}
$$

# Remaining Steps

- Define $\mathrm{Arg_A}$ recursively (straightforward).
- For defining $\mathrm{Arg_B}$ we need to define the set of terms $\mathrm{ATerm}$ of type $A$ we can form from given elements of type $A$ and the later defined constructor $intro_A$.
- Then define $\mathrm{SP_B}$ and $\mathrm{Arg_B}$, $\mathrm{Index_B}$.
- Requires some functorality problems.
- Main problems arise due to intensional equality.

# Summary

- Induction-induction is a natural way of defining the syntax of type theory inside type theory.
- Induction-induction occur naturally in mathematics.
  - Seem to be more common than induction-recursion.
  - Maybe, because they are more easily reduced to well-understood inductive definitions.
    - Usage of inductive-recursive definitions without having the concept is much more difficult.
  - Having them as first-class citizens reduces some of the complexity.

# Summary

- Examples can be formulated easily.
- Closed formalisation more complicated.

# Open Problems

- Elimination Rules (induction over an induction-inductive definitions).
  - Elimination rules for concrete examples can be written down easily.
  - An abstract general elimination rule has been defined.
  - A general concrete elimination rule complicated (due to intensional equality).
- Formulation in ordinary mathematics (first order).
- Generalisations
  - More levels.
  - More complex structures such as $B : A \to A \to \mathrm{Set}$.
  - Combination with induction-recursion.