# Inductive and Recursive Definitions in Constructive Type Theory

Peter Dybjer
Chalmers Tekniska Högskola

TYPES summer school
Göteborg
August 2005

# Some questions

What is an inductive definition of a set? What is a recursive definition of a function? Classically? Constructively?

What are the differences and similarities wrt recursive data types in functional languages?

What is the role of inductive definitions in the foundations of constructive mathematics? What is Martin-Löf type theory? What is the role of inductive definitions in Martin-Löf type theory? What other foundational systems for constructive mathematics are there? What is the role of inductive definitions for them?

What is the nature of Martin-Löf's meaning explanations? What is the syntactico-semantical approach to constructive foundations?

# More questions

What inductive definitions are constructively acceptable? The versatility of the constructive notion of an inductive definition. What are inductive families (indexed inductive definitions)? What are generalized inductive definitions? What are inductive-recursive definitions?

What recursive definitions are constructively acceptable? What are the differences and similarities wrt recursive function definitions in functional languages? What is the role of pattern matching? What is the role of well-founded recursion vs structural recursion? What is the relationship between Martin-Löf type theory and Agda? How do you program with inductive definitions?

How can you axiomatize a general theory of inductive and recursive definitions in Martin-Löf type theory with a minimum of coding?

# Plan

1. Martin-Löf type theory with one universe ($\mathbf{MLTT_U}$). Rules for natural numbers. Large elimination.

2. What is an inductive definition?

   (a) Examples
   (b) Classical definition. Rule sets. Monotone operators.

3. What is the language of constructive mathematics? What is the data? What is the role of inductive definitions?

4. Ordinary inductive definitions.

   (a) Inductive sets. Lists, binary trees, propositional formulas. General schema.
   (b) Inductive families. Family of theorems. General schema.

5. Generalized inductive definitions.

   (a) Brouwer ordinals.
   (b) Well-orderings. Hereditarily finite sets. Aczel's V and CZF.
   (c) Well-founded part of a relation. Termination of programs.

6. Induction-recursion. More about meaning explanations.

7. (Finite axiomatization of inductive and inductive-recursive definitions, if time permits)

# Terminology

| later Martin-Löf | lecture notes | Bishop | early Martin-Löf | other |
|---|---|---|---|---|
| type | sort | | category | kind |
| set | type | preset | type | |
| extensional set | set | set | | setoid, E-set |
| function | operation | operation | function | |
| extensional function | function | function | | setoid map, |

We here follow later Martin-Löf. The "lecture notes" above refer to "Type-theoretic Foundations of Constructive Mathematics" by Coquand, Dybjer, Palmgren, and Setzer.

# Original Martin-Löf type theory with one universe ($\mathbf{MLTT_U}$)

- Set formers for predicate logic: $\mathbf{0}, \mathbf{1}, +, \times, \to, \Sigma, \Pi$.
- Natural numbers $\mathrm{N}$.
- Universe of small sets $\mathrm{U}$.

All these were introduced in Martin-Löf 1972.

# More set formers

- Identity $\mathrm{I}$ (Martin-Löf 1973) - an inductive family/predicate
- Well-orderings $\mathrm{W}$ (Martin-Löf 1979) - a generalized inductive definition
- Hierarchy of universes $\mathrm{U}_0, \mathrm{U}_1, \mathrm{U}_2, \ldots$.
- Universe à la Tarski (Martin-Löf 1984) $\mathrm{U}, \mathrm{T}$ - an inductive-recursive definition

# Rules for natural numbers

Formation rule:

$$\mathrm{N} \quad : \quad \mathrm{Set}$$

Introduction rules:

$$0 \quad : \quad \mathrm{N}$$

$$\mathrm{Succ} \quad : \quad \mathrm{N} \to \mathrm{N}$$

# Elimination and equality rules for natural numbers

Elimination rule:

$$\begin{aligned}\mathrm{R} \quad : \quad & (C : \mathrm{N} \to \mathrm{Set}) \to C\ 0 \to ((x : \mathrm{N}) \to C\ x \to C\ (\mathrm{Succ}\ x)) \to \\ & (n : \mathrm{N}) \to C\ n\end{aligned}$$

Dependent elimination rule = rule for building proofs by mathematical induction = rule for typing functions from natural numbers where the target is a dependent type.

Equality rules:

$$\begin{aligned}\mathrm{R}\ C\ d\ e\ 0 &= d : C\ 0 \\ \mathrm{R}\ C\ d\ e\ (\mathrm{Succ}\ n) &= e\ n\ (\mathrm{R}\ C\ d\ e\ n) : C\ (\mathrm{Succ}\ n)\end{aligned}$$

# Primitive recursive schema

If $C : \mathrm{N} \to \mathrm{Set}, d : C\ 0, e : (x : \mathrm{N}) \to C\ x \to C\ (\mathrm{Succ}\ x)$, and

$$\begin{aligned} f\ 0 &= d \\ f\ (\mathrm{Succ}\ n) &= e\ n\ (f\ n) \end{aligned}$$

then we can define

$$f = \mathrm{R}\ C\ d\ e : (n : \mathrm{N}) \to C\ n$$

## Exercise: define some functions in $\mathbf{MLTT_U}$

1. addition, subtraction, and multiplication of natural numbers

2. the $\mathrm{half}$ function:

$$\begin{aligned} \mathrm{half}\ 0 &= 0 \\ \mathrm{half}\ (\mathrm{Succ}\ 0) &= 0 \\ \mathrm{half}\ (\mathrm{Succ}\ (\mathrm{Succ}\ n)) &= \mathrm{Succ}\ (\mathrm{half}\ n) \end{aligned}$$

3. division of natural numbers

# Equality of natural numbers

Define

$$\mathrm{eq_N} \quad : \quad \mathrm{N} \to \mathrm{N} \to \mathrm{Bool}$$

by pattern matching on constructors

$$\begin{aligned}
\mathrm{eq_N}\ 0\ 0 &= \mathrm{True} \\
\mathrm{eq_N}\ 0\ (\mathrm{Succ}\ n) &= \mathrm{False} \\
\mathrm{eq_N}\ (\mathrm{Succ}\ m)\ 0 &= \mathrm{False} \\
\mathrm{eq_N}\ (\mathrm{Succ}\ m)\ (\mathrm{Succ}\ n) &= \mathrm{eq_N}\ m\ n
\end{aligned}$$

## Exercise: define equality of natural numbers in $\mathbf{MLTT_U}$

Hint. Use the elimination rule for $\mathrm{N}$ and define it by primitive recursion of higher type (primitive recursive functional) as follows. Define

$$\mathrm{eq}_{\mathrm{N}}\, m : \mathrm{N} \to \mathrm{Bool}$$

by induction on $m : \mathrm{N}$. The base case is "to be equal to zero" and the step case is to define "to be equal to $m + 1$" in terms of "to be equal to $m$".

Note that in $\mathbf{MLTT_U}$ we define $\mathrm{Bool} = \mathbf{1} + \mathbf{1}$.

# Recursive function definitions in Agda

The Alf/Agda philosophy: we do not limit ourselves to the primitive recursive schema formalized by $\mathbb{N}$-elimination, but allow more general recursion patterns. There is a termination checker which checks that the recursive calls refer to "structurally smaller" arguments.

For example, the above definition of equality is accepted as a good definition (syntax may use case analysis) since it passes the termination checker. There is ongoing research on extending the termination checker.

# Recursive definitions of sets

Define

$$\mathrm{Vect} \quad : \quad \mathrm{Set} \to \mathrm{N} \to \mathrm{Set}$$

abbreviated $A^n = \mathrm{Vect}\ A\ n$

$$\begin{aligned} A^0 &= \mathbf{1} \\ A^{\mathrm{Succ}\ n} &= A \times A^n \end{aligned}$$

This definition is directly accepted by Agda (using case). Can we define it in $\mathbf{MLTT_U}$? Note that we cannot use $\mathrm{R}$ directly. Why?

# Large elimination

If we modify $\mathrm{R}$, so that the result type is $\mathrm{Set}$ instead of *a set* $C\ n$, then we get a *large* elimination rule

$$\mathrm{R}^{\mathrm{large}} \quad : \quad \mathrm{Set} \to (\mathrm{N} \to \mathrm{Set} \to \mathrm{Set}) \to \mathrm{N} \to \mathrm{Set}$$

Now we can define

$$A^n \quad = \quad \mathrm{R}^{\mathrm{large}}\ \mathbf{1}\ (\lambda x, X.A \times X)\ n$$

# The universe of small sets

Large elimination rules are not part of $\mathbf{MLTT_U}$. Instead we show how to use the universe $\mathrm{U}$ to approximate the effect of large elimination. We here choose the formulation à la Tarski (Aczel 1974, Martin-Löf 1984), where we have a set $\mathrm{U}$ of codes for small sets, and a decoding function $\mathrm{T}$:

$$\begin{aligned} \mathrm{U} &: \mathrm{Set} \\ \mathrm{T} &: \mathrm{U} \to \mathrm{Set} \end{aligned}$$

Remark: earlier versions of Martin-Löf type theory used universes à la Russell, where $a : \mathrm{Set}$ if $a : \mathrm{U}$.

# Inductive-recursive definition of the universe à la Tarski

We have one introduction rule for $\mathrm{U}$ and one equality rule for $\mathrm{T}$ for each small set former:

$$
\begin{array}{rcl@{\qquad}rcl}
\hat{\mathrm{N}} & : & \mathrm{U} & \mathrm{T}\,\hat{\mathrm{N}} & = & \mathrm{N} \\
\hat{\mathbf{0}} & : & \mathrm{U} & \mathrm{T}\,\hat{\mathbf{0}} & = & \mathbf{0} \\
\hat{\mathbf{1}} & : & \mathrm{U} & \mathrm{T}\,\hat{\mathbf{1}} & = & \mathbf{1} \\
(\hat{+}) & : & \mathrm{U} \to \mathrm{U} \to \mathrm{U} & \mathrm{T}\,(a \hat{+} b) & = & \mathrm{T}\,a + \mathrm{T}\,b \\
(\hat{\times}) & : & \mathrm{U} \to \mathrm{U} \to \mathrm{U} & \mathrm{T}\,(a \hat{\times} b) & = & \mathrm{T}\,a \times \mathrm{T}\,b \\
\hat{\Sigma} & : & (a : \mathrm{U}) \to (\mathrm{T}\,a \to \mathrm{U}) \to \mathrm{U} & \mathrm{T}\,(\hat{\Sigma}\,a\,b) & = & \Sigma\,(\mathrm{T}\,a)\,(\lambda x.\mathrm{T}\,(bx)) \\
 & \vdots & & & \vdots &
\end{array}
$$

Note that $\mathrm{U}$ is not a small set.

# The universe at work

Now we can define

$$A^n = \mathrm{T}\ (\mathrm{R}\ (\lambda x.\mathrm{U})\ \hat{\mathbf{1}}\ (\lambda x, X.A \hat{\times} X)\ n)$$

for $A : \mathrm{U}$. (Note that we only define $A^n$ for small $A$!)

Exercise: Define a family

$$\mathrm{Fin} : \mathrm{N} \to \mathrm{Set}$$

so that $\mathrm{Fin}\ n$ is a set with $n$ elements.

# The equality proposition

We would like to have

$$\mathrm{Eq_N} \quad : \quad \mathrm{N} \to \mathrm{N} \to \mathrm{Set}$$

so that $\mathrm{Eq_N}\ m\ n$ is inhabited iff $\mathrm{eq_N}\ m\ n = \mathrm{True}$. How is this defined in $\mathbf{MLTT_U}$? In Agda we can define directly (using case)

$$\begin{aligned}
\mathrm{Eq_N}\ 0\ 0 &= \mathbf{1} \\
\mathrm{Eq_N}\ (\mathrm{Succ}\ m)\ 0 &= \mathbf{0} \\
\mathrm{Eq_N}\ 0\ (\mathrm{Succ}\ n) &= \mathbf{0} \\
\mathrm{Eq_N}\ (\mathrm{Succ}\ m)\ (\mathrm{Succ}\ n) &= \mathrm{Eq_N}\ m\ n
\end{aligned}$$

# Exercise: some uses large elimination for truth values

Define the following functions in $\mathbf{MLTT_U}$:

1. the following function which converts a truth value to a proposition:

$$\begin{aligned} \mathrm{T}_{\mathrm{Bool}} &: \mathrm{Bool} \to \mathrm{Set} \\ \mathrm{T}_{\mathrm{Bool}}\,\mathrm{True} &= \mathbf{1} \\ \mathrm{T}_{\mathrm{Bool}}\,\mathrm{False} &= \mathbf{0} \end{aligned}$$

2. $\mathrm{Eq}_{\mathrm{N}} : \mathrm{N} \to \mathrm{N} \to \mathrm{Set}$.

# Lists and other inductive definitions

List is not a primitive set former in $\mathbf{MLTT}_{\mathrm{U}}$. Can we encode it?

Martin-Löf 1984: "We can follow the same pattern used to define natural numbers to introduce other inductively defined sets. We see here the example of lists". Exercise: write down the rules for list (formation, introduction, elimination, and equality rules).

Martin-Löf 1972: "The type $\mathrm{N}$ is just the prime example of a type introduced by an *ordinary inductive definition*. However, it seems preferable to treat this special case rather than to give a necessarily much more complicated general formulation which would include $(\Sigma \in A)B(x)$, $A + B$, $\mathrm{N}_n$ and $\mathrm{N}$ as special cases. See Martin-Löf 1971 for a general formulation of inductive definitions in the language of ordinary first order predicate logic."

# Inductive definitions – examples

- the rules for generating natural numbers by zero and successor
- the rules for generating well-formed formulas of a logic
- the axioms and inference rules generating theorems of the logic
- the productions of a context-free grammar
- the computation rules for a programming language
- the reflexive-transitive closure of a relation

# Inductive definitions and recursive datatypes

- lists generated by `Nil` and `Cons`
- binary trees generated by `EmptyTree` and `MkTree`
- algebraic types in general: parameterized, many sorted term algebras
- infinitely branching trees; Brouwer ordinals; etc.
- inductive dependent types (vectors of a certain length, trees of a certain height, balanced trees, etc)
- inductive-recursive definitions (sorted lists, freshlists, etc)

# Reflexive and nested datatypes

Note that recursive datatypes in functional languages (e g Haskell) include reflexive datatypes

```
data Lambda = Nil | Lambda (Lambda -> Lambda)
```

and nested datatypes

```
data Nest a = Nil | Cons a (Nest (a,a))
data Bush a = Nil | Cons a (Bush (Bush a))
```

Neither is accepted verbatim as an inductive definition in Martin-Löf type theory.

# What is an inductive definition in general, classically?

Two equivalent notions of inductive definition of *subset* of a set $V$ via

- rule sets on $V$
- monotone operators on subsets of $V$

See Aczel 1977: "An Introduction to Inductive Definitions" in Handbook of Mathematical Logic.

# Sets inductively generated by rule sets

A *rule* on a base set $V$ in Aczel's sense is a pair $(X, x)$ (also written $\frac{X}{x}$), such that $X \subseteq V$ and $x \in V$.

Let $\Phi$ be a set of rules on $V$. A set $Y$ is $\Phi$-*closed* if for all $\frac{X}{x} \in \Phi$

$$X \subseteq Y \supset x \in Y$$

The set *inductively generated* by $\Phi$ is defined to be the least $\Phi$-closed set

$$\mathcal{I}(\Phi) = \bigcap \{Y \subseteq V | Y \ \Phi\text{-}closed\},$$

The induction principle for $\mathcal{I}(\Phi)$ is "if $Y$ is $\Phi$-closed, then $\mathcal{I}(\Phi) \subseteq Y$". The introduction rules are "$\mathcal{I}(\Phi)$ is $\Phi$-closed, that is, if $X \subseteq \mathcal{I}(\Phi)$ then $x \in \mathcal{I}(\Phi)$".

# Example: reflexive-transitive closure of a relation

Rules for inductively generating $R^* \subseteq A \times A$ from $R \subseteq A \times A$:

$$\frac{}{xR^*x} \qquad \frac{xR^*y \qquad yR^*z}{xR^*z} \qquad \frac{xRy}{xR^*y}$$

Formal rule set (in the sense of Aczel) on $V = A \times A$:

$$\{\frac{\emptyset}{(x,x)}|x \in A\} \cup \{\frac{\{(x,y),(y,z)\}}{(x,z)}|x,y,z \in A\} \cup \{\frac{\emptyset}{(x,y)}|(x,y) \in R\}$$

# Example: inference rules for minimal logic

$$\frac{\vdash x \Rightarrow y \qquad \vdash x}{\vdash y} \qquad \frac{}{\vdash x \Rightarrow y \Rightarrow x} \qquad \frac{}{\vdash (x \Rightarrow y \Rightarrow z) \Rightarrow (x \Rightarrow y) \Rightarrow x \Rightarrow z}$$

The corresponding rule set on $V = \mathrm{Form}$ (the set of formulas)

$$\{\frac{\{x \Rightarrow y, x\}}{y} | x, y \in \mathrm{Form}\} \cup \{\frac{\emptyset}{x \Rightarrow y \Rightarrow x} | x, y \in \mathrm{Form}\} \cup$$

$$\{\frac{\emptyset}{(x \Rightarrow y \Rightarrow z) \Rightarrow (x \Rightarrow y) \Rightarrow x \Rightarrow z} | x, y, z \in \mathrm{Form}\} \cup$$

# Infinitary rules

The $\omega$-rule is

$$\frac{(\vdash x_i)_{i\in\omega}}{\vdash \bigwedge_{i\in\omega} x_i}$$

We have the rule set

$$\{\frac{\{x_i | i \in \omega\}}{\bigwedge_{i\in\omega} x_i} | x_i \in \mathrm{Form} \text{ for all } i \in \omega\}$$

We here assume that $\bigwedge_{i\in\omega} x_i \in \mathrm{Form}$ whenever $x_i \in \mathrm{Form}$ for all $i \in \omega$.

# Rules for generating natural numbers

Type-theoretic introduction rules

$$\begin{aligned} 0 &: \mathrm{N} \\ \mathrm{Succ} &: \mathrm{N} \to \mathrm{N} \end{aligned}$$

Rule set (What is $V$? $\mathrm{N}$ is given by a *fundamental* inductive definition)

$$\{\frac{\emptyset}{0}\} \cup \{\frac{\{n\}}{\mathrm{Succ}(n)} | n \in V\}$$

Monotone operator $\phi : \mathcal{P}(V) \to \mathcal{P}(V)$ which generates natural numbers:

$$\phi(X) = \{0\} \cup \{\mathrm{Succ}(n) | n \in X\} \cong 1 + X$$

# Inductively defined sets generated by monotone operators

Let $\phi : \mathcal{P}(V) \to \mathcal{P}(V)$ be monotone, that is, if $X \subseteq Y \subseteq V$, then $\phi(X) \subseteq \phi(Y) \subseteq V$. Then $\phi$ has a least prefixed point

$$\mathcal{I}(\phi) = \bigcap \{X \subseteq V | \phi(X) \subseteq X\}$$

The induction principle is "if $\phi(X) \subseteq X$ then $\mathcal{I}(\phi) \subseteq X$". The introduction rule is "$\phi(\mathcal{I}(\phi)) \subseteq \mathcal{I}(\phi)$".

Exercise. Show that inductive generation by rule sets and monotone operators are equivalent.

# Inductive definitions and constructive foundations

Classically, inductive definitions are understood as least fixed points of monotone operators (or least sets closed under a set of rules).

P. Aczel (An introduction to inductive definitions, Handbook of Mathematical Logic, 1976, pp 779 and 780.):

> An alternative approach is to take induction as a primitive notion, not needing justification in terms of other methods. ... It would be interesting to formulate a coherent conceptual framework that made induction the principal notion.

No universal principle. We may discover new stronger inductive generation principles.

# Inductive definitions and the notion of set in Martin-Löf type theory

Martin-Löf type theory is such a coherent conceptual framework.

"(1) a set A is defined by prescribing how a canonical element of A is formed as well as how two equal canonical elements of A are formed."

Per Martin-Löf (p8 in Intuitionistic Type Theory, Bibliopolis 1984)

This is the same as saying that a set is defined by its introduction rules, i e, the rules for inductively generating its members.

# Towards a language for constructive mathematics

Constructivism:

- Functions are computable
- Proofs of implications are computable functions ("methods")
- A proof of a disjunction is either a proof of left or of right disjunct
- A proof of existence gives a witness

Hence, not excluded middle, not double negation.

# What is the data?

- Kleene's partial recursive functions: natural numbers
- Turing machines: strings of characters
- Lambda calculus (untyped): lambda expressions (mix program and data)

Code natural numbers as strings of characters or as lambda expressions.

Code functions, pairs, etc as natural numbers (Gödel coding). Even coding proofs as natural numbers (Kleene realizability).

# Types of data

Natural numbers $\mathrm{N}$

Higher order functions $A \to B$ (cf Gödel's T)

Propositions. Church type theory. Cf type $\mathrm{U}$ of small types.

More types? Cf development of programming languages.

In logic. Curry-Howard: $0, 1, A + B, A \times B, \Sigma_{x:A} B, \Pi_{x:A} B$.

This yields Martin-Löf type theory 1972. (Cf also Scott 1970: Constructive validity - check. Has also version of $\mathrm{W}$-type.).

# Martin-Löf type theory and inductive definitions

- Basic set formers: $\Pi, \Sigma, +, \mathrm{I}, \mathrm{N}, \mathrm{N}_n, \mathrm{W}, \mathrm{U}_n$
- Adding new set formers with their rules when there is a need for them: lists, binary trees, the well-founded part of a relation, ....
- Exactly what is a good inductive definition? Schemata for inductive definitions, indexed inductive definitions, inductive-recursive definitions
- Generic formulation: universes for inductive definitions, indexed inductive definitions, inductive-recursive definitions

# Formulae of minimal propositional logic

Another example of an ordinary inductive definition acceptable as a primitive notion in Martin-Löf type theory:

$$\begin{array}{rcl} \mathrm{Form} & : & \mathrm{Set} \end{array}$$

$$\begin{array}{rcl} \mathrm{Atom} & : & \mathrm{N} \to \mathrm{Form} \\ \Rightarrow & : & \mathrm{Form} \to \mathrm{Form} \to \mathrm{Form} \end{array}$$

Can it be defined in $\mathbf{MLTT_U}$?

# Schema for ordinary inductive definitions of sets

*Ordinary* as opposed to *generalized* inductive definitions

*Sets* as opposed to *families* of sets.

We can introduce a new set $P$ with finitely many constructors, where each constructor has finitely many arguments, the types of which are either $P$ itself (an *inductive* argument/premise) or a set $A$ (a *side condition/non-inductive* premise). The set $A$ may depend on previous side-conditions, and may also make use of previously defined constants. It may not contain $P$.

The conclusion has type $P$.

# Parameterized ordinary inductive definitions of sets

A definition can moreover depend on *parameters* which can have arbitrary types (including the type of sets). This is a third kind of argument to a constructor. The parameters always come before the side-conditions and the inductive arguments. (Inductive arguments and side-conditions can be mixed.)

All parameters appear as the initial arguments in formation, introduction, and elimination rules for $P$.

Remark: more general schemata exist for *inductive families*, *generalized induction*, and *induction-recursion*.

# Lists as an example of a parameterized ordinary inductive definition of a set

Example, lists are given by a parameterized ordinary inductive definition of a set. The constructor

$$\mathrm{Cons} \quad : \quad (A : \mathrm{Set}) \to A \to [A] \to [A]$$

has three arguments: $A : \mathrm{Set}$ is a parameter, $a : A$ is a side-condition, $as : [A]$ is an inductive argument.

Exercise: Analyse the constructors of natural numbers, binary trees with information in the leaves, the set $\mathrm{Form}$ of formulas of minimal logic above. Analyse also $\times$ and $\Sigma$! What about $\to$ and $\Pi$?

# The inductive family of theorems

$$\begin{aligned}
\mathrm{Thm} &: \mathrm{Form} \to \mathrm{Set} \\
\\
\mathrm{K} &: (a, b : \mathrm{Form}) \to \mathrm{Thm}\ (a \Rightarrow b \Rightarrow a) \\
\mathrm{S} &: (a, b, c : \mathrm{Form}) \to \mathrm{Thm}\ ((a \Rightarrow b \Rightarrow c) \Rightarrow (a \Rightarrow b) \Rightarrow a \Rightarrow c) \\
\mathrm{Mp} &: (a, b : \mathrm{Form}) \to \mathrm{Thm}\ (a \Rightarrow b) \to \mathrm{Thm}\ a \to \mathrm{Thm}\ b
\end{aligned}$$

Exercise. By Curry-Howard, $\mathrm{Thm}$ represents an inductively defined *predicate*. Define the predicate $\mathrm{Thm}$ in $\mathbf{MLTT}_{\mathrm{U}}$ up to logical equivalence!

## Elimination rule for $\mathrm{Thm}$

$$
\begin{aligned}
\mathrm{R}_{\mathrm{Thm}} \quad : \quad & (C : (a : \mathrm{Form}) \to \mathrm{Thm}\ a \to \mathrm{Set}) \to \\
& ((a, b : \mathrm{Form}) \to C\ (a \Rightarrow b \Rightarrow a)\ (\mathrm{K}\ a\ b)) \to \\
& ((a, b, c : \mathrm{Form}) \to C\ ((a \Rightarrow b \Rightarrow c) \Rightarrow (a \Rightarrow b) \Rightarrow a \Rightarrow c)\ (\mathrm{S}\ a\ b\ c)) \to \\
& ((a, b : \mathrm{Form}) \to (p : \mathrm{Thm}\ (a \Rightarrow b)) \to (q : \mathrm{Thm}\ a) \to \\
& \qquad C\ (a \Rightarrow b)\ p \to C\ a\ q \to C\ b\ (\mathrm{Mp}\ a\ b\ p\ q)) \to \\
& (a : \mathrm{Form}) \to (p : \mathrm{Thm}\ a) \to C\ a\ p
\end{aligned}
$$

# Classical soundness of $\mathrm{Thm}$

Exercise: use the elimination rules for $\mathrm{Form}$ and $\mathrm{Thm}$ to write the follwoing two functions:

$$\begin{aligned} \mathrm{eval} &: (\mathrm{N} \to \mathrm{Bool}) \to \mathrm{Form} \to \mathrm{Bool} \\ \mathrm{sound} &: (\rho : \mathrm{N} \to \mathrm{Bool}) \to (a : \mathrm{Form}) \to \mathrm{Thm}\ a \to (\mathrm{eval}\ \rho\ a =_{\mathrm{Bool}} \mathrm{True}) \end{aligned}$$

$\mathrm{eval}$ assigns classical semantics in Bool to each formula.

$\mathrm{sound}$ is a proof that all theorems are evaluated to True under this semantics:

## Equality rules for $\mathrm{Thm}$

$$\mathrm{R}_{\mathrm{Thm}}\, C\; d\; e\; f\; (a \Rightarrow b \Rightarrow a)\; (\mathrm{K}\; a\; b) \quad = \quad d\; a\; b$$

$$\mathrm{R}_{\mathrm{Thm}}\, C\; d\; e\; f\; ((a \Rightarrow b \Rightarrow c) \Rightarrow (a \Rightarrow b) \Rightarrow a))(\mathrm{S}\; a\; b\; c) \quad = \quad e\; a\; b\; c$$

$$\mathrm{R}_{\mathrm{Thm}}\, C\; d\; e\; f\; b\; (\mathrm{Mp}\; a\; b\; p\; q) \quad = \quad f\; a\; b\; (\mathrm{R}_{\mathrm{Thm}}\, C\; d\; e\; f\; (a \Rightarrow b)\; p)(\mathrm{R}_{\mathrm{Thm}}\, C\; d\; e\; f\; a\; q)$$

# Schema for ordinary inductive definitions of families of sets

Like for *sets*, except that we have indices (cf Martin-Löf 1971):

We can introduce a new family of sets $P : I \to \mathrm{Set}$ with finitely many constructors, where each constructor has finitely many arguments, the types of which are either $P\ p$ (an inductive argument/premise) or a set $A$ (a side condition/non-inductive premise). The index $p : I$ and the set $A$ may depend on previous side-conditions, and may also make use of previously defined constants. ($A$ must not contain $P$.)

The conclusion has the type $P\ q$, where again $q : I$ may depend on previous side-conditions, and may also make use of previously defined constants.

# Inductive families in Agda

One can use `idata` in Agda for defining inductive families.

If the index $q$ (in the conclusion type $P\ q$) is a variable, then one can also use `data` in Agda.

# A generalized inductive definition: the Brouwer ordinals

$$
\begin{array}{rcl}
\mathcal{O} & : & \mathrm{Set} \\
 & & \\
0_{\mathcal{O}} & : & \mathcal{O} \\
\mathrm{Succ}_{\mathcal{O}} & : & \mathcal{O} \to \mathcal{O} \\
\mathrm{Sup}_{\mathcal{O}} & : & (\mathrm{N} \to \mathcal{O}) \to \mathcal{O}
\end{array}
$$

Note that the type of the argument of $\mathrm{Sup}_{\mathcal{O}}$ is a function type, representing the fact that it has an infinite number of (inductive) arguments. Note that $\mathcal{O}$ appears *strictly positively* in the argument type.

# Aczel rule set for Brouwer ordinals

We get set-theoretic semantics of $\mathcal{O}$ by taking the set inductively generated by the following rule set:

$$\{\frac{\emptyset}{0}\} \cup \{\frac{\{\alpha\}}{\mathrm{Succ}(\alpha)} | \alpha \in V\} \cup \{\frac{\{\beta(n) | n \in \mathrm{N}\}}{\mathrm{Sup}(\beta)} | \beta \in \mathrm{N} \to V\}$$

# Elimination rule

$$
\begin{aligned}
\mathrm{ordrec} \quad : \quad & (C : \mathcal{O} \to \mathrm{Set}) \to \\
& C\ 0_{\mathcal{O}} \to \\
& ((x : \mathcal{O}) \to C\ x \to C\ (\mathrm{Succ}_{\mathcal{O}}\ x)) \to \\
& ((f : \mathrm{N} \to \mathcal{O}) \to ((x : \mathrm{N}) \to C\ (f\ x)) \to C\ (\mathrm{Sup}_{\mathcal{O}}\ f)) \to \\
& (c : \mathcal{O}) \to \\
& C\ c
\end{aligned}
$$

Exercise: write down the equality rules.

# Some Brouwer ordinals

$$\begin{aligned}
\omega &= \mathrm{Sup}_{\mathcal{O}}\ (\lambda n.\iota_{\mathrm{N}\mathcal{O}}\ n) : \mathcal{O} \\
\iota_{\mathrm{N}\mathcal{O}} &: \mathrm{N} \to \mathcal{O} \\
\iota_{\mathrm{N}\mathcal{O}}\ 0 &= 0_{\mathcal{O}} \\
\iota_{\mathrm{N}\mathcal{O}}\ (\mathrm{Succ}\ n) &= \mathrm{Succ}_{\mathcal{O}}\ (\iota_{\mathrm{N}\mathcal{O}}\ n)
\end{aligned}$$

Why is

$$2\omega = \mathrm{Sup}_{\mathcal{O}}\ (\lambda n.\mathrm{R}\ (\lambda n.\mathcal{O})\ \omega\ (\lambda y, z.\mathrm{Succ}_{\mathcal{O}}\ z))?$$

Exercise. Do some more ordinals, eg $\omega^2, \omega^\omega, \epsilon_0$. Do ordinal addition.

# Well-orderings

$$\begin{aligned}
\mathrm{W} &: (A : \mathrm{Set}) \to (A \to \mathrm{Set}) \to \mathrm{Set} \\
\\
\mathrm{Sup} &: (A : \mathrm{Set}) \to \\
&\quad (B : A \to \mathrm{Set}) \to \\
&\quad (a : A) \to \\
&\quad (B\ a \to \mathrm{W}\ A\ B) \to \\
&\quad \mathrm{W}\ A\ B
\end{aligned}$$

Exercise: write down the elimination and equality rules.

# Schema for generalized inductive definitions

Inductive arguments of constructors in a generalized inductive definition of a set $P$ can have types

$$(x_1 : A_1) \to \ldots \to A_n \to P$$

where $P$ does not appear in $A_i$. ($A_i$ may depend on previous arguments, etc.)

It is also possible to encode all sets given by a generalized inductive definition in terms of $\mathrm{W}$ up to extensional equality.

Exercise: Find $A$ and $B$ so that $\mathrm{W}\ A\ B$ encodes $\mathrm{N}$. Similar question for the Brouwer ordinals. (See Martin-Löf 1984)

# The set of finitely branching trees

We can define the set of finitely branching trees with arbitrary finite branching degree (no information in the nodes)

$$\mathrm{V}_{\mathrm{fin}} = \mathrm{W}\ \mathrm{N}\ \mathrm{Fin}$$

# Hereditarily finite iterative sets

The elements of $\mathrm{V}_{\mathrm{fin}}$ can represent the hereditarily finite sets, i e, finite sets all of whose elements are also hereditarily finite sets. However, when comparing two hereditarily finite sets for equality, order and repetition of elements do not matter. We define extensional equality as bisimilarity:

$$\begin{aligned} \mathrm{Sup}\ n\ b =_{\mathrm{ext}} \mathrm{Sup}\ n'\ b' \quad = \quad & \forall i : \mathrm{Fin}\ n.\ \exists i' : \mathrm{Fin}\ n'.\ b\ i =_{\mathrm{ext}} b'\ i' \wedge \\ & \forall i' : \mathrm{Fin}\ n'.\ \exists i : \mathrm{Fin}\ n.\ b'\ i' =_{\mathrm{ext}} b\ i \end{aligned}$$

(Note: we have omitted the two parameter arguments of $\mathrm{Sup}$.)

Extensional membership is defined by

$$a \in_{\mathrm{ext}} \mathrm{Sup}\ n\ b \quad = \quad \exists i : \mathrm{Fin}\ n. a =_{\mathrm{ext}} b\ i$$

# Operations on hereditarily finite sets

Exercise: Define the empty hereditarily finite set. Define union and intersection, and power set of a hereditarily finite set! Define the finite ordinals.

# Aczel's constructive cumulative hierarchy $\mathrm{V}$

$\mathrm{V}_{\mathrm{fin}}$ only contains hereditarily finite iterative sets. In a similar way we can define Aczel's set $\mathrm{V}$ of iterative sets by

$$\mathrm{V} = \mathrm{W}\ \mathrm{U}\ \mathrm{T}$$

The branching can now be indexed by an arbitrary (possibly infinite) small set $\mathrm{T}\ a$. The definitions of extensional equality and extensional membership are analogous to those for $\mathrm{V}_{\mathrm{fin}}$.

Aczel gives axioms for a constructive version CZF of ZF set theory, where the axioms hold for $\mathrm{V}$ with extensional equality and extensional membership.

# Exercise: constructions in $\mathrm{V}$

Check that the subset relation, the operations of union and intersection, and the finite ordinals are defined in the same way as in $\mathrm{V}_{\mathrm{fin}}$.

Construct the first infinite ordinal $\omega : \mathrm{V}$!

What happens if we try to define the powerset of an arbitrary element in $\mathrm{V}$?

# Constructive foundations

Predicative constructive systems:

**Type theory.** Martin-Löf type theory

**Lambda calculus (untyped).** Aczel's first order theory of combinators (logical theory of constructions etc.). Use intuitionistic predicate logic and inductive predicates on domain of lambda expressions. Cf Feferman's explicit mathematics.

**Set theory.** Aczel's Constructive ZF - use axioms for $\mathrm{V}$

**Category theory.** Moerdijk - Palmgren's predicative topos - axioms for the category of setoids in Martin-Löf type theory

# The well-founded part of a relation

Given a set $A$ and a binary relation $(>)$ on $A$ an element $x$ is in the well-founded part of $(>)$ if there is no infinite descending chain $x > x_1 > x_2 > \cdots$.

An alternative definition is by a generalized inductive definition: $x$ is in the well-founded part of $(>)$ provided all elements $x'$ which are "smaller" $(x > x')$ are in the well-founded part. In particular each "smallest" element is in the well-founded part.

$$\mathrm{Wfp} \quad : \quad A \to \mathrm{Set}$$

$$\mathrm{Sup} \quad : \quad (x : A) \to ((x' : A) \to (x > x') \to \mathrm{Wfp}\ x') \to \mathrm{Wfp}\ x$$

# Exercise: correspondence between the two definitions of well-foundedness

Prove that the inductive definition implies the no-infinite descending chain definition in Martin-Löf type theory!

$\mathrm{Wfp}$ on the previous page was defined for a *fixed* set $A$ and a *fixed* relation $(>)$. Rewrite the definiton so that $A$ and $(>)$ become parameters!

# Using the well-founded part to encode general recursive definitions

Encode general recursive function

```
f : A -> B
```

by

$$f' : (x : A) \to \mathrm{Wfp}\ A\ (>_{\mathtt{f}})\ x \to B$$

where $(>_{\mathtt{f}})$ is the recursive call relation: $x >_{\mathtt{f}} x'$ whenever the computation of `f` $x$ will generate a call `f` $x'$.

# Encoding division

For example, the partial recursive division function

```
div m n = if (n < m) then 0 else (div m (n - m))
```

has the recursive call relation $(>_{\texttt{div}\ m})$, where

$$n >_{\texttt{div}\ m} p \quad = \quad n =_{\mathrm{N}} m + p$$

What is $\mathrm{Wfp}\ (>_{\texttt{div}\ m})$? What happens if $m =_{\mathrm{N}} 0$?

# Inductive-recursive definitions

Recall the inductive-recursive definition of the universe á la Tarski. We only display one constructor to show the inductive-recursive nature of the definition:

$$\begin{array}{rcl} \mathrm{U} & : & \mathrm{Set} \\ \mathrm{T} & : & \mathrm{U} \to \mathrm{Set} \end{array}$$

$$\begin{array}{rcl} \hat{\Sigma} & : & (a : \mathrm{U}) \to (\mathrm{T}a \to \mathrm{U}) \to \mathrm{U} \\ \mathrm{T}\ (\hat{\Sigma}\ a\ b) & = & \Sigma x : \mathrm{T}\ a.\mathrm{T}\ (b\ x) \end{array}$$

Why is such a strange definition constructively valid? Use Martin-Löf style meaning explanations!

# Inductive-recursive definition of ordered lists

$$
\begin{aligned}
\text{OrdList} &: \text{Set} \\
\text{lb} &: \text{N} \to \text{OrdList} \to \text{Bool}
\end{aligned}
$$

$$
\begin{aligned}
\text{Nil} &: \text{OrdList} \\
\text{Cons} &: (x : \text{N}) \to (xsp : \text{OrdList}) \to \text{T}\ (\text{lb}\ x\ xsp) \to \text{OrdList}
\end{aligned}
$$

$$
\begin{aligned}
\text{lb}\ x\ \text{Nil} &= \text{True} \\
\text{lb}\ x\ (\text{Cons}\ y\ xsp\ q) &= x \leq y
\end{aligned}
$$

# Set-theoretic semantics of the universe à la Tarski

Rule set

$$\{\frac{\{(a,A)\}\cup\{(b(x),B(x))|x\in A\}}{(\hat{\Sigma}(a,b),\Sigma_{x:A}B(x))}|a,A\in V,b,B\in A\to V\}\cup\cdots$$

Exercise: give similar set-theoretic semantics to the inductive-recursive definition of sorted lists with the lower bound function!

# Some references

- P. Aczel, An introduction to inductive definitions, chapter C.7 in the Handbook of Mathematical Logic, North-Holland 1977.

- Inductive and inductive-recursive definitions in Martin-Löf type theory:

  `http://www.cs.chalmers.se/~peterd/papers/inductive.html`

- The calculus of inductive constructions:

  `http://pauillac.inria.fr/cdrom/www/coq/doc/node.0.3.html`