Generic Programs, Generic Proofs, and Dependent Types

Peter Dybjer
Chalmers University of Technology
Göteborg, Sweden

Cambridge
24 October 2003

A new result!

Theorem.

$$x = x$$

Should reflexivity of equality be an axiom?

A. M. Turing (1936):

I shall also suppose that the number of symbols which may be printed is finite. If we were to allow an infinity of symbols, then there would be symbols differing to an arbitrary small extent. The effect of this restriction of the number of symbols is not very serious. It is always possible to use sequences of symbols in the place of single symbols. . . . The difference from our point of view between the single and compound symbols is that the compound symbols, if they are too lengthy, cannot be observed at a glance. This is in accordance with experience. We cannot tell at a glance whether 9999999999999999 and 999999999999999 are the same.

Reflexivity of generic equality

Theorem.

$$(\Sigma : \mathrm{Sig}) \to (x : \mathrm{T}_\Sigma) \to x =_\Sigma x$$

where

$$(=_\Sigma) : \mathrm{T}_\Sigma \to \mathrm{T}_\Sigma \to \mathrm{Bool}$$

is a generic equality test,

$$\mathrm{Sig} : \mathrm{Set}$$

is the set of signatures, ie codes for a suitable class of datatypes (cf ML's *equality types*), and

$$\mathrm{T}_\Sigma : \mathrm{Set}$$

is (the carrier of) the algebra of Σ-terms, ie an initial Σ-algebra.

Generic map

Another motivating example of generic programming. We have

$$\mathrm{map} : (X \to Y) \to [X] \to [Y]$$

An analogous function for binary trees:

$$\mathrm{mapBT} : (X \to Y) \to \mathrm{BT}\,X \to \mathrm{BT}\,Y$$

In general

$$\mathrm{map}_\Sigma : (X \to Y) \to \mathrm{T}_\Sigma\,X \to \mathrm{T}_\Sigma\,Y$$

where $\Sigma : \mathrm{Sig}$ is a code for a “regular” parameterized datatype

$$\mathrm{T}_\Sigma : \mathrm{Set} \to \mathrm{Set}$$

Generic programs and dependent types

Generic programs are written by induction on codes for datatypes (signatures for term algebras). Generic functional programming languages (eg PolyP, Generic Haskell) extend Haskell with special facilities for such definitions.

With dependent types we have natural “internal” types for generic programs, such as

$$(=) : (\Sigma : \mathrm{Sig}) \to \mathrm{T}_\Sigma \to \mathrm{T}_\Sigma \to \mathrm{Bool}$$

In dependent type theory you can also do generic proofs (Pfeiffer and Rueß 1999).

Signatures and universes

We here make use of a universe, that is a set of codes

$$\mathrm{Sig}$$

and a decoding function (family of sets)

$$\mathrm{T} : \mathrm{Sig} \to \mathrm{Set}$$

which to each code (signature) $\Sigma : \mathrm{Sig}$ assigns the datatype (set, term algebra) T_Σ it denotes. (Usually, universes are denoted by the letter U in dependent type theory, but here we prefer the term signature since we borrow terminology from universal algebra.)

Why generic programs and generic proofs?

Important for libraries of programs!

Libraries are important for proof assistants!

Generic proofs of generic programs help limit the size of libraries.

Generic dependent type theory

The usual formulations of Martin-Löf Type Theory or the Calculus of Inductive Constructions do not allow definitions on the code of a “datatype” (= inductive definition).

Martin-Löf type theory and inductive definitions - a brief history

1970-79 Basic set formers: $\Pi, \Sigma, +, \mathrm{I}, \mathrm{N}, \mathrm{N}_n, \mathrm{W}, \mathrm{U}_n$

1979-85 Adding new set formers with their rules when there is a need for them: lists, binary trees, the well-founded part of a relation, ...

1985-91 Exactly what is a good inductive definition? Schemata for inductive definitions, indexed inductive definitions, inductive-recursive definitions

1999- Generic formulation: universes for inductive definitions, indexed inductive definitions, inductive-recursive definitions

References

- A finite axiomatization of inductive-recursive definitions (with Anton Setzer). Pages 129 - 146 in Proceedings of TLCA 1999, LNCS 1581.

- Indexed induction-recursion (with Anton Setzer). Pages 93-113 in Proof Theory in Computer Science International Seminar, Dagstuhl Castle, Germany, October 7-12, 2001, LNCS 2183.

- Induction-recursion and initial algebras (with Anton Setzer), 2003. Annals of Pure and Applied Logic, in press.

- Universes for generic programs and proofs in dependent type theory (with Marcin Benke and Patrik Jansson), 2003. To appear in Nordic Journal of Computing.

Plan

1. Generic dependent type theory
2. Generic programming - generic equality
3. Generic proofs - reflexivity of generic equality
4. More general classes of datatypes (if time permits)

Generic dependent type theory

We work in Martin-Löf type theory with rules for one-sorted term algebras (axiomatizing “initial algbra semantics”):

- Logical framework. Rules for $(x : A) \to B$, $(x : A) \times B$, 0, 1, 2, Set.
- Rules for arities and signatures. For simplicity only one-sorted signatures, but much more powerful classes of datatypes can be represented.
- Auxiliary constructions for setting up “initial algebra semantics”
- Generic formation, introduction, elimination, and equality rules for T_Σ.

Notation

Dependent function types are written

$$(x : A) \to B$$

This is the type of functions f mapping $a : A$ to $f\,a : B[x := a]$. $B : A \to \mathrm{Set}$ is an A-indexed family of sets.

Dependent product types are written

$$(x : A) \times B$$

This is the type of pairs (a, b) such that $a : A$ and $b : B[x := a]$.

Recall: rules for natural numbers

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

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

Generic rules for T_Σ

Formation rule

$$\mathrm{T}_\Sigma : \mathrm{Set}$$

Introduction rule

$$\mathrm{Intro}_\Sigma : ? \to \mathrm{T}_\Sigma$$

Elimination rule

$$\mathrm{rec}_\Sigma : (C : \mathrm{T}_\Sigma \to \mathrm{Set}) \to ? \;\to (x : \mathrm{T}_\Sigma) \to C\,x$$

Equality rule

$$\mathrm{rec}_\Sigma\, C\, d\, (\mathrm{Intro}_\Sigma\, y) = d\, y \;\cdots ?\; \cdots$$

One-sorted term algebras

A one-sorted signature is a finite list of natural numbers, representing the arities of the operations of the signature. Examples:

Σ	T_Σ
$[\,]$	empty set
$[0,1]$	natural numbers
$[0,0]$	truth values
$[0,1,1]$	lists of truth values
$[0,2]$	binary trees without information in the nodes

We could consider more general class Sig of signatures, eg giving rise to the ML-notion of equality class. Even more, capturing all kinds of inductive definitions in Martin-Löf type theory (indexed, inductive-recursive, ...).

Signature = universe for one-sorted term algebras

We introduce the type of signatures

$$\mathrm{Sig\ Type}$$

and the decoding function

$$\mathrm{T} : \mathrm{Sig} \to \mathrm{Set}$$

which maps a signature Σ to (the carrier of) its term algebra $\mathrm{T}_\Sigma : \mathrm{Set}$.

The initial Σ-algebra diagram

$$\begin{tikzcd}
\mathrm{F}_\Sigma \mathrm{T}_\Sigma \arrow[r, "\mathrm{Intro}_\Sigma"] \arrow[d, "\mathrm{F}_\Sigma(\mathrm{iter}_\Sigma\, C\, d)"'] & \mathrm{T}_\Sigma \arrow[d, "\mathrm{iter}_\Sigma\, C\, d"] \\
\mathrm{F}_\Sigma\, C \arrow[r, "d"'] & C
\end{tikzcd}$$

Pattern functor:

$$\mathrm{F}_{[n_1,\ldots,n_m]}\, X \quad = \quad X^{n_1} + \cdots + X^{n_m}$$

A diagram for T_Σ-elimination

$$\begin{tikzcd}
 & \mathrm{F}_\Sigma \mathrm{T}_\Sigma \arrow[r, "\mathrm{Intro}_\Sigma"] \arrow[dl, "\mathrm{F}_\Sigma\langle \mathrm{id}, \mathrm{rec}_\Sigma\, C\, d\rangle"'] \arrow[d, "{\langle \mathrm{id}, \mathrm{F}^{\mathrm{map}}_\Sigma\, \mathrm{T}_\Sigma\, C\, (\mathrm{rec}_\Sigma\, C\, d)\rangle}"] & \mathrm{T}_\Sigma \arrow[d, "{\langle \mathrm{id}, \mathrm{rec}_\Sigma\, C\, d\rangle}"] \\
\mathrm{F}_\Sigma((x : \mathrm{T}_\Sigma) \times C\, x) \arrow[r, "\cong_{\Sigma, \mathrm{T}_\Sigma, C}"'] & (y : \mathrm{F}_\Sigma\, \mathrm{T}_\Sigma) \times \mathrm{F}^{\mathrm{IH}}_\Sigma \mathrm{T}_\Sigma\, C\, y \arrow[r, "e"'] & (x : \mathrm{T}_\Sigma) \times C\, x
\end{tikzcd}$$

$$e\,(y, z) \quad = \quad (\mathrm{Intro}_\Sigma\, y, d\, y\, z)$$

Auxiliary constructions

$$\mathrm{F}^{\mathrm{IH}}_{\Sigma} : (X : \mathrm{Set}) \to (X \to \mathrm{Set}) \to \mathrm{F}_{\Sigma}\, X \to \mathrm{Set}$$
$$\mathrm{F}^{\mathrm{IH}}_{[n_1,\ldots,n_m]}\, X\, C\, (\mathrm{In}_i\, (x_1, \ldots, x_{n_i})) = C\, x_1 \times \cdots \times C\, x_{n_i}$$

and

$$\begin{aligned} \mathrm{F}^{\mathrm{map}}_{\Sigma} \quad : \quad & (X : \mathrm{Set}) \to (C : X \to \mathrm{Set}) \\ & \to ((x : X) \to C\, x) \to (y : \mathrm{F}_{\Sigma}\, X) \to \mathrm{F}^{\mathrm{IH}}_{\Sigma}\, X\, C\, y \end{aligned}$$

$$\mathrm{F}^{\mathrm{map}}_{[n_1,\ldots,n_m]}\, X\, C\, h\, (\mathrm{In}_i\, (x_1, \ldots, x_{n_i})) = (h\, x_1, \ldots, h\, x_{n_i})$$

Generic rules for T_Σ

$$\begin{aligned}
\mathrm{T}_\Sigma &: \mathrm{Set} \\
\mathrm{Intro}_\Sigma &: \mathrm{F}_\Sigma \mathrm{T}_\Sigma \to \mathrm{T}_\Sigma \\
\mathrm{rec}_\Sigma &: (C : \mathrm{T}_\Sigma \to \mathrm{Set}) \to ((y : \mathrm{F}_\Sigma\, \mathrm{T}_\Sigma) \to \mathrm{F}^{\mathrm{IH}}_\Sigma\, \mathrm{T}_\Sigma\, C\, y \to C\, (\mathrm{Intro}_\Sigma\, y)) \\
&\quad \to (x : \mathrm{T}_\Sigma) \to C\, x
\end{aligned}$$

Equality rule

$$\mathrm{rec}_\Sigma\, C\, d\, (\mathrm{Intro}_\Sigma\, y) = d\, y\, (\mathrm{F}^{\mathrm{map}}_\Sigma\, C\, (\mathrm{rec}_\Sigma\, C\, d)\, y)$$

Large elimination

We may add a large version of this elimination too, where C can be an arbitrary family of types, that is,

$$C[x] \text{ Type} \quad (x : \mathrm{T}_\Sigma)$$

not just a family of sets.

Martin-Löf type theory with one-sorted term algebras

Start with logical framework (including at least dependent function and product types, and $\mathbf{0}, \mathbf{1}, \mathbf{2}$). Add

arities formation, introduction, (and elimination and equality) rules for N

signatures formation, introduction (and elimination and equality) rules for lists of natural numbers

pattern functors defining rules for object part of F_Σ, and the auxiliary $\mathrm{F}_\Sigma^{\mathrm{IH}}$ and $\mathrm{F}_\Sigma^{\mathrm{map}}$

term algebras formation, introduction, elimination, and equality rules for T_Σ with constants Intro_Σ and rec_Σ.

Generic programming

A generic size function.

A special case of the initial algebra diagram. Let $\Sigma = [n_1, \ldots, n_m]$.

$$\begin{tikzcd}
\mathrm{T}_\Sigma^{n_1} + \cdots + \mathrm{T}_\Sigma^{n_m} \arrow[r, "\mathrm{Intro}_\Sigma"] \arrow[d, "\mathrm{size}_\Sigma^{n_1} + \cdots + \mathrm{size}_\Sigma^{n_m}"'] & \mathrm{T}_\Sigma \arrow[d, "\mathrm{size}_\Sigma"] \\
\mathrm{N}^{n_1} + \cdots + \mathrm{N}^{n_m} \arrow[r, "\mathrm{sizestep}_\Sigma"'] & \mathrm{N}
\end{tikzcd}$$

Generic recursion step

The recursion step is defined by induction on the signature:

$$\begin{aligned}\mathrm{sizestep}_{n::\Sigma}\ (\mathrm{Inl}\, xs) &= 1 + \mathrm{sum}_n\, xs \\ \mathrm{sizestep}_{n::\Sigma}\ (\mathrm{Inr}\, y) &= \mathrm{sizestep}_{\Sigma}\ y\end{aligned}$$

where

$$\mathrm{sum} : (n : \mathrm{N}) \to \mathrm{N}^n \to \mathrm{N}$$

is a function summing the elements of a vector of natural numbers.

Generic equality

$$\begin{tikzcd}
\mathrm{T}_\Sigma^{n_1} + \cdots + \mathrm{T}_\Sigma^{n_m} \arrow[r, "\mathrm{Intro}_\Sigma"] \arrow[d, "(=_\Sigma)^{n_1} + \cdots + (=_\Sigma)^{n_m}"'] & \mathrm{T}_\Sigma \arrow[d, "(=_\Sigma)"] \\
(\mathrm{T}_\Sigma \to \mathrm{Bool})^{n_1} + \cdots + (\mathrm{T}_\Sigma \to \mathrm{Bool})^{n_m} \arrow[r, "\mathrm{eqstep}_\Sigma"'] & \mathrm{T}_\Sigma \to \mathrm{Bool}
\end{tikzcd}$$

Generic recursion step

$$\mathrm{eqstep}_\Sigma\,(\mathrm{In}_i\,(f_1,\ldots,f_{n_i}))\,(\mathrm{Intro}_\Sigma\,(\mathrm{In}_i\,(y_1,\ldots,y_{n_i}))) = f_1\,y_1 \wedge \cdots \wedge f_{n_i} y_{n_i}$$

and for $i \neq j$

$$\mathrm{eqstep}_\Sigma\,(\mathrm{In}_i\,(f_1,\ldots,f_{n_i}))\,(\mathrm{Intro}_\Sigma\,(\mathrm{In}_j\,(y_1,\ldots,y_{n_j}))) = \mathrm{False}$$

can be defined by induction on the signature.

Code for the generic recursion step

$$
\begin{array}{lcl}
(=_{\Sigma}) & : & \mathrm{T}_{\Sigma} \to \mathrm{T}_{\Sigma} \to \mathrm{Bool} \\
x =_{\Sigma} x' & = & \mathrm{iter}_{\Sigma}\ \mathrm{eqstep}_{\Sigma}\, x\, x' \\
\\
\mathrm{eqstep}_{\Sigma} & : & \mathrm{F}_{\Sigma}\,(\mathrm{T}_{\Sigma} \to \mathrm{Bool}) \to \mathrm{T}_{\Sigma} \to \mathrm{Bool} \\
\mathrm{eqstep}_{\Sigma}\, y\, x & = & \mathrm{recogAll}_{\Sigma}\, \mathrm{T}_{\Sigma}\, y\, (\mathrm{out}_{\Sigma}\, x) \\
\\
\mathrm{out}_{\Sigma} & : & \mathrm{T}_{\Sigma} \to \mathrm{F}_{\Sigma}\mathrm{T}_{\Sigma} \\
\mathrm{out}_{\Sigma}\, x & = & \mathrm{rec}_{\Sigma}\, (\lambda x.\mathrm{F}_{\Sigma}\mathrm{T}_{\Sigma})\, (\lambda y z. y)
\end{array}
$$

More code for the generic recursion step

$$\mathrm{recogAll}_{\Sigma} : (X : \mathrm{Set}) \to \mathrm{F}_{\Sigma}\,(X \to \mathrm{Bool}) \to \mathrm{F}_{\Sigma}\,X \to \mathrm{Bool}$$

$$\begin{array}{lllllll}
\mathrm{recogAll}_{n::\Sigma} & X & (\mathrm{Inl}\,fs) & (\mathrm{Inl}\,xs) & = & \mathrm{andArgs}_n\,X\,fs\,xs \\
\mathrm{recogAll}_{n::\Sigma} & X & (\mathrm{Inr}\,x) & (\mathrm{Inr}\,y) & = & \mathrm{recogAll}_{\Sigma}\,X\,x\,y \\
\mathrm{recogAll}_{n::\Sigma} & X & (\mathrm{Inl}\,fs) & (\mathrm{Inr}\,y) & = & \mathrm{False} \\
\mathrm{recogAll}_{n::\Sigma} & X & (\mathrm{Inr}\,x) & (\mathrm{Inl}\,xs) & = & \mathrm{False}
\end{array}$$

$$\mathrm{andArgs}_n : (X : \mathrm{Set}) \to (X \to \mathrm{Bool})^n \to X^n \to \mathrm{Bool}$$

$$\begin{array}{lllllll}
\mathrm{andArgs}_0 & X & () & () & = & \mathrm{True} \\
\mathrm{andArgs}_{m+1} & X & (f, fs) & (x, xs) & = & f\,x \wedge \mathrm{andArgs}_m\,X\,fs\,xs
\end{array}$$

Reflexivity of generic equality

Instantiate the generic elimination rule (structural recursion on T_Σ)!

$$\begin{tikzcd}
\mathrm{F}_\Sigma \mathrm{T}_\Sigma \arrow[r, "\mathrm{Intro}_\Sigma"] \arrow[d, "\langle \mathrm{id}, \mathrm{F}^{\mathrm{map}}_\Sigma\, \mathrm{T}_\Sigma\, C\, \mathrm{ref}_\Sigma \rangle"] & \mathrm{T}_\Sigma \arrow[d, "\langle \mathrm{id}, \mathrm{ref}_\Sigma \rangle"'] \\
(y : \mathrm{F}_\Sigma\, \mathrm{T}_\Sigma) \times \mathrm{F}^{\mathrm{IH}}_\Sigma\, \mathrm{T}_\Sigma\, C\, y \arrow[r, "\lambda(y, z).(\mathrm{Intro}_\Sigma\, y, \mathrm{refstep}_\Sigma\, y\, z)"'] & (x : \mathrm{T}_\Sigma) \times |x =_\Sigma x|
\end{tikzcd}$$

$$\begin{tikzcd}
\mathrm{F}_\Sigma \mathrm{T}_\Sigma \arrow[r, "\mathrm{Intro}_\Sigma"] \arrow[d, "\langle \mathrm{id}, \mathrm{F}^{\mathrm{map}}_\Sigma\, \mathrm{T}_\Sigma\, C\, \mathrm{ref}_\Sigma \rangle"] & \mathrm{T}_\Sigma \arrow[d, "\langle \mathrm{id}, \mathrm{ref}_\Sigma \rangle"'] \\
(y : \mathrm{F}_\Sigma\, \mathrm{T}_\Sigma) \times \mathrm{F}^{\mathrm{IH}}_\Sigma\, \mathrm{T}_\Sigma\, C\, y \arrow[r, "\lambda(y,z).(\mathrm{Intro}_\Sigma\, y, \mathrm{refstep}_\Sigma\, y\, z)"'] & (x : \mathrm{T}_\Sigma) \times C\, x
\end{tikzcd}$$

$$\begin{aligned}
C\, x &= |x =_\Sigma x| \\
\mathrm{F}^{\mathrm{IH}}_\Sigma\, \mathrm{T}_\Sigma\, C\, (\mathrm{In}_i\, (x_1, \ldots, x_{n_i})) &= |x_1 =_\Sigma x_1| \times \ldots \times |x_{n_i} =_\Sigma x_{n_i}| \\
\mathrm{F}^{\mathrm{map}}_\Sigma\, \mathrm{T}_\Sigma\, C\, \mathrm{ref}_\Sigma\, (\mathrm{In}_i\, (x_1, \ldots, x_{n_i})) &= (\mathrm{ref}_\Sigma\, x_1, \ldots, \mathrm{ref}_\Sigma\, x_{n_i})
\end{aligned}$$

Converting from a truth value to a set

Truth values, sets, and propositions:

$$|-| \quad : \quad \mathrm{Bool} \to \mathrm{Set}$$

$$\begin{aligned} |\mathrm{False}| &= \mathbf{0} \\ |\mathrm{True}| &= \mathbf{1} \end{aligned}$$

$$\begin{aligned} \bot &= \mathbf{0} \\ \top &= \mathbf{1} \end{aligned}$$

How to write $\mathrm{refstep}_\Sigma$

$$\mathrm{refstep}_\Sigma \quad : \quad (y : \mathrm{F}_\Sigma\,\mathrm{T}_\Sigma) \to \mathrm{F}^{\mathrm{IH}}_\Sigma\,\mathrm{T}_\Sigma\,C\,y \to C\,(\mathrm{Intro}_\Sigma\,y)$$

Hence, informally

$$\begin{aligned} & \mathrm{refstep}_\Sigma\,(\mathrm{In}_i\,(x_1, \ldots, x_{n_i})) \\ : \quad & |x_1 =_\Sigma x_1| \times \cdots \times |x_{n_i} =_\Sigma x_{n_i}| \to |\mathrm{Intro}_\Sigma\,(\mathrm{In}_i\,(x_1, \ldots, x_{n_i})) =_\Sigma \mathrm{Intro}_\Sigma\,(\mathrm{In}_i\,(x_1, \ldots \end{aligned}$$

Simplify the result type:

$$|x_1 =_\Sigma x_1| \times \cdots \times |x_{n_i} =_\Sigma x_{n_i}| \to |(x_1 =_\Sigma x_1) \wedge \cdots \wedge (x_{n_i} =_\Sigma x_{n_i})|$$

Two abbreviations

$$\begin{aligned}
\mathrm{Rel} &: \mathrm{Set} \to \mathrm{Set} \\
\mathrm{Rel}\ & X = X \to X \to \mathrm{Bool} \\
\mathrm{lref} &: (X : \mathrm{Set}) \to \mathrm{Rel}\, X \to X \to \mathrm{Set} \\
\mathrm{lref}\ & X\, r\, x = |r\, x\, x|
\end{aligned}$$

where the first argument to lref will be hidden for brevity.

Code for generic reflexivity

$$\begin{array}{lcl} \mathrm{ref}_\Sigma & : & (x : \mathrm{T}_\Sigma) \to |x \;=_\Sigma\; x| \\ \mathrm{ref}_\Sigma & = & \mathrm{rec}_\Sigma\,(\mathrm{lref}\,(=_\Sigma))\,(\mathrm{refmatch}_\Sigma\,\mathrm{T}_\Sigma\,(=_\Sigma)) \end{array}$$

ie $\mathrm{refstep}_\Sigma$ is a special case of $\mathrm{refmatch}$ defined by:

$$\begin{array}{lcl} \mathrm{refmatch}_\Sigma & : & (X : \mathrm{Set}) \to (r : \mathrm{Rel}\,X) \to \\ & & (y : \mathrm{F}_\Sigma\,X) \to \mathrm{F}^{\mathrm{IH}}_\Sigma\,X\,(\mathrm{lref}\,r)\,y \to |\mathrm{recogAll}_\Sigma\,X\,(\mathrm{F}^1_\Sigma\,X\,\mathrm{Bool}\,r\,y)\,y| \end{array}$$

$$\begin{array}{llllcl} \mathrm{refmatch}_{n::\Sigma} & X & r & (\mathrm{Inl}\,xs) & = & \mathrm{refargs}_n\,X\,r\,xs \\ \mathrm{refmatch}_{n::\Sigma} & X & r & (\mathrm{Inr}\,y) & = & \mathrm{refmatch}_\Sigma\,X\,r\,y \end{array}$$

More code for generic reflexivity

$\mathrm{refargs}$ handles the arguments to the constructor:

$$
\begin{array}{lll}
\mathrm{refargs}_n & : & (X : \mathrm{Set}) \to (r : \mathrm{Rel}\, X) \to \\
 & & (xs : X^n) \to (\mathrm{lref}\, r)^n\, xs \to |\mathrm{andArgs}_n\, X\, (r^n\, xs)\, xs|
\end{array}
$$

$$
\begin{array}{llllll}
\mathrm{refargs}_0 & X & r & () & () & = & () \\
\mathrm{refargs}_{m+1} & X & r & (x, xs) & (ih, ihs) & = & ih\, (\mathrm{refargs}_m\, X\, r\, xs\, ihs)
\end{array}
$$

This concludes the proof ...

This is the first, or perhaps more accurately the second ..., generic proof.

Is this a tour de force?

- Equality is not the simplest generic function. Reflexivity of equality is not the simplest generic proof! Nevertheless, the proof is not so long. But it uses complex dependent types - comes from general framework.
- $(=) : \mathrm{N} \to \mathrm{N} \to \mathrm{Bool}$ written with primitive recursion on higher type is superficially a bit more complex than written by 4 recursion equations.
- Would it be simpler to prove reflexivity in a framework with general recursion? YOU set up this logic for generic general recursive programs.
- Type-checking dependent types sometimes difficult - types may be *extensionally* but not *definitionally (computationally)* equal.

Generalizing the notion of a signature

iterated inductive definitions

$$\mathrm{Cons} : \mathrm{N} \to \mathrm{ListN} \to \mathrm{ListN}$$

the first argument is a side-condition.

generalized inductive definitions

$$\mathrm{Sup} : (\mathrm{N} \to \mathcal{O}) \to \mathcal{O}$$

constructors with dependent types

$$\mathrm{Sup} : (x : A) \to (B\,x \to \mathrm{W}) \to \mathrm{W}$$

Signatures for generalized inductive definitions

$$\begin{array}{rcl} \epsilon & : & \mathrm{Sig} \\ \sigma & : & (A : \mathrm{Set}) \to (A \to \mathrm{Sig}) \to \mathrm{Sig} \\ \rho & : & \mathrm{Set} \to \mathrm{Sig} \to \mathrm{Sig} \end{array}$$

$$\begin{array}{rcl} \mathrm{F}_{\epsilon}\, X & = & \mathbf{1} \\ \mathrm{F}_{\sigma\, A\, \Sigma}\, X & = & (x : A) \times \mathrm{F}_{\Sigma\, x}\, X \\ \mathrm{F}_{\rho\, A\, \Sigma}\, X & = & (A \to X) \times \mathrm{F}_{\Sigma}\, X \end{array}$$

Parameterized inductive definitions

parameterized inductive definitions

$$\mathrm{Cons} : (A : \mathrm{Set}) \to A \to \mathrm{List}\, A \to \mathrm{List}\, A$$

The first argument is a parameter - it is the same for all constructors.

We need universes for parameterized inductive definitions because much generic programming is about such definitions (generic map, zip, etc).

If we just want to formalize parameterized inductive definitions in Martin-Löf type theory, parameterization is taken care of by the logical framework.