Codata

Thorsten Altenkirch

University of Nottingham

Haskell: data = codata ?

Haskell: data = codata ?

- $$\textbf{data List} = \text{Nil} \mid \text{Cons } \textbf{Nat List}$$

Haskell: data = codata ?

- $$\textbf{data List} = \mathrm{Nil} \mid \mathrm{Cons}\ \textbf{Nat List}$$

- $$\begin{aligned} &even \in \textbf{List} \to \textbf{Bool} \\ &even\ \mathrm{Nil} &&= \mathrm{True} \\ &even\ (\mathrm{Cons}\ a\ as) &&= \neg\ (even\ as) \end{aligned}$$

Haskell: data = codata ?

- $$\textbf{data List} = \text{Nil} \mid \text{Cons } \textbf{Nat List}$$

- $$\begin{aligned} &\mathit{even} \in \textbf{List} \to \textbf{Bool} \\ &\mathit{even}\ \text{Nil} &&= \text{True} \\ &\mathit{even}\ (\text{Cons}\ a\ as) &&= \neg\,(\mathit{even}\ as) \end{aligned}$$

- $$\begin{aligned} &\mathit{from} \in \textbf{Nat} \to \textbf{List} \\ &\mathit{from}\ n = \text{Cons}\ n\ (\mathit{from}\ (n+1)) \end{aligned}$$

Haskell: data = codata ?

- $$\textbf{data List} = \text{Nil} \mid \text{Cons } \textbf{Nat List}$$

- $$\begin{aligned} &\mathit{even} \in \textbf{List} \to \textbf{Bool} \\ &\mathit{even}\ \text{Nil} &&= \text{True} \\ &\mathit{even}\ (\text{Cons}\ a\ as) &&= \neg\ (\mathit{even}\ as) \end{aligned}$$

- $$\begin{aligned} &\mathit{from} \in \textbf{Nat} \to \textbf{List} \\ &\mathit{from}\ n = \text{Cons}\ n\ (\mathit{from}\ (n+1)) \end{aligned}$$

- $\mathit{even}\ (\mathit{from}\ 0)$ diverges!

Type Theory: data $\neq$ codata

Type Theory: data $\neq$ codata

$$\textbf{data List} = \mathrm{Nil} \mid \mathrm{Cons}\ \textbf{Nat List}$$

$$\textbf{codata List}^{\infty} = \mathrm{Nil}^{\infty} \mid \mathrm{Cons}^{\infty}\ \textbf{Nat List}^{\infty}$$

Type Theory: data $\neq$ codata

- $$\textbf{data List} = \text{Nil} \mid \text{Cons}\ \textbf{Nat List}$$
  $$\textbf{codata List}^{\infty} = \text{Nil}^{\infty} \mid \text{Cons}^{\infty}\ \textbf{Nat List}^{\infty}$$

- $$\begin{aligned} &\mathit{even} \in \textbf{List} \to \textbf{Bool} \\ &\mathit{even}\ \text{Nil} && = \text{True} \\ &\mathit{even}\ (\text{Cons}\ a\ as) && = \neg\ (\mathit{even}\ as) \end{aligned}$$

Type Theory: data $\neq$ codata

- $$\textbf{data List} = \text{Nil} \mid \text{Cons } \textbf{Nat List}$$
  $$\textbf{codata List}^\infty = \text{Nil}^\infty \mid \text{Cons}^\infty\ \textbf{Nat List}^\infty$$

- $$\mathit{even} \in \textbf{List} \to \textbf{Bool}$$
  $$\mathit{even}\ \text{Nil} = \text{True}$$
  $$\mathit{even}\ (\text{Cons}\ a\ \mathit{as}) = \neg\ (\mathit{even}\ \mathit{as})$$

- $$\mathit{from} \in \textbf{Nat} \to \textbf{List}^\infty$$
  $$\mathit{from}\ n = \text{Cons}^\infty\ n\ (\mathit{from}\ (n+1))$$

Type Theory: data $\neq$ codata

- $$\begin{aligned}&\textbf{data List} = \text{Nil} \mid \text{Cons}\ \textbf{Nat List}\\ &\textbf{codata List}^\infty = \text{Nil}^\infty \mid \text{Cons}^\infty\ \textbf{Nat List}^\infty\end{aligned}$$
- $$\begin{aligned}&\mathit{even} \in \textbf{List} \to \textbf{Bool}\\ &\mathit{even}\ \text{Nil} = \text{True}\\ &\mathit{even}\ (\text{Cons}\ a\ as) = \neg\,(\mathit{even}\ as)\end{aligned}$$
- $$\begin{aligned}&\mathit{from} \in \textbf{Nat} \to \textbf{List}^\infty\\ &\mathit{from}\ n = \text{Cons}^\infty\ n\ (\mathit{from}\ (n+1))\end{aligned}$$
- $\mathit{even}\ (\mathit{from}\ 0)$ doesn't typecheck.

codata in Type Theory

codata in Type Theory

- Thierry Coquand
  Infinite Objects in Type Theory
  TYPES 93

codata in Type Theory

- Thierry Coquand
  Infinite Objects in Type Theory
  TYPES 93
- Eduardo Giminez
  Coinductive Types in COQ
  93 – 95
  see Coq'Art, pp.347 – 376

Codata ?

Codata ?

- Codata seems more exotic then data.

Codata ?

- Codata seems more exotic then data.
- Categorically codata (terminal coalgebras) is a dual of data (initial algebras)

Codata ?

- Codata seems more exotic then data.
- Categorically codata (terminal coalgebras) is a dual of data (initial algebras)
- Proposal: a conceptual duality based on contracts

Codata ?

- Codata seems more exotic then data.
- Categorically codata (terminal coalgebras) is a dual of data (initial algebras)
- Proposal: a conceptual duality based on contracts
- which justifies *Observational Type Theory* reflecting this symmetry.

Data – the producer contract

Data – the producer contract

The producer of **data** promises that he/she will construct data only using the agreed constructors.

Data – the producer contract

The producer of **data** promises that he/she will construct data only using the agreed constructors.

Consequences:

Data – the producer contract

The producer of **data** promises that he/she will construct data only using the agreed constructors.

Consequences:

- pattern matching

Data – the producer contract

The producer of **data** promises that he/she will construct data only using the agreed constructors.

Consequences:

- pattern matching
- structural recursion

Data – the producer contract

The producer of **data** promises that he/she will construct data only using the agreed constructors.

Consequences:

- pattern matching
- structural recursion
- induction as structural recursion on proofs

Codata – the consumer contract

Codata – the consumer contract

The consumer of **codata** promises that he/she will only analyze codata using the patterns induced by the agreed constructors.

Codata – the consumer contract

The consumer of **codata** promises that he/she will only analyze codata using the patterns induced by the agreed constructors.

Consequences:

Codata – the consumer contract

The consumer of **codata** promises that he/she will only analyze codata using the patterns induced by the agreed constructors.

Consequences:

- constructors

Codata – the consumer contract

The consumer of **codata** promises that he/she will only analyze codata using the patterns induced by the agreed constructors.

Consequences:

- constructors
- guarded corecursion

Codata – the consumer contract

The consumer of **codata** promises that he/she will only analyze codata using the patterns induced by the agreed constructors.

Consequences:

- constructors
- guarded corecursion
- coinduction as guarded recursion on proofs

A simple proposition

A simple proposition

$$\begin{aligned}
&mapS \in \mathbf{List}^{\infty} \to \mathbf{List}^{\infty}\\
&mapS\ \mathrm{Nil}^{\infty} = \mathrm{Nil}^{\infty}\\
&mapS\ \mathrm{Cons}^{\infty}\ n\ \vec{n} = \mathrm{Cons}^{\infty}\ (n+1)\ (mapS\ \vec{n})
\end{aligned}$$

A simple proposition

$$\begin{aligned} \mathit{mapS} &\in \mathbf{List}^{\infty} \to \mathbf{List}^{\infty} \\ \mathit{mapS}\ \mathrm{Nil}^{\infty} &= \mathrm{Nil}^{\infty} \\ \mathit{mapS}\ \mathrm{Cons}^{\infty}\ n\ \vec{n} &= \mathrm{Cons}^{\infty}\ (n+1)\ (\mathit{mapS}\ \vec{n}) \end{aligned}$$

let $$\frac{n \in \mathbf{Nat}}{\mathit{lem}\ n \in \mathit{mapS}\ (\mathit{from}\ n) = \mathit{from}\ (n+1)}$$

A simple proposition

$$\begin{aligned}
mapS &\in \mathbf{List}^\infty \to \mathbf{List}^\infty \\
mapS\ \mathrm{Nil}^\infty &= \mathrm{Nil}^\infty \\
mapS\ \mathrm{Cons}^\infty\ n\ \vec{n} &= \mathrm{Cons}^\infty\ (n+1)\ (mapS\ \vec{n})
\end{aligned}$$

let $$\frac{n \in \mathbf{Nat}}{lem\ n \in mapS\ (from\ n) = from\ (n+1)}$$

- Let's have a closer look at $=$.

Equality for List

Equality for List

data $$\frac{\vec{m}, \vec{n} \in \mathbf{List}}{\vec{m} = \vec{n} \in \mathbf{Prop}}$$ **where**

Equality for List

data $\dfrac{\vec{m}, \vec{n} \in \mathbf{List}}{\vec{m} = \vec{n} \in \mathbf{Prop}}$ **where**

$$\dfrac{}{\mathrm{EqNil} \in \mathrm{Nil} = \mathrm{Nil}}$$

Equality for List

data $\dfrac{\vec{m}, \vec{n} \in \mathbf{List}}{\vec{m} = \vec{n} \in \mathbf{Prop}}$ **where**

$$\frac{}{\mathrm{EqNil} \in \mathrm{Nil} = \mathrm{Nil}}$$

$$\frac{p \in m = n \quad \vec{p} \in \vec{m} = \vec{n}}{\mathrm{EqCons}\ p\ \vec{p} \in \mathrm{Cons}\ m\ \vec{m} = \mathrm{Cons}\ n\ \vec{n}}$$

Properties of $=$

Properties of $=$

let $$\frac{\vec{n} \in \mathbf{List}}{\mathit{refl}\ \vec{n} \in \vec{n} = \vec{n}}$$

Properties of $=$

let $$\frac{\vec{n} \in \mathbf{List}}{\mathit{refl}\ \vec{n} \in \vec{n} = \vec{n}}$$

$$\begin{aligned} \mathit{refl}\ \mathrm{Nil} &= \mathrm{EqNil} \\ \mathit{refl}\ (\mathrm{Cons}\ n\ \vec{n}) &= \mathrm{EqCons}\ (\mathit{refl}\ n)\ (\mathit{refl}\ \vec{n}) \end{aligned}$$

Properties of $=$

let $$\frac{\vec{n} \in \mathbf{List}}{\mathit{refl}\ \vec{n} \in \vec{n} = \vec{n}}$$

$$\begin{aligned} \mathit{refl}\ \mathrm{Nil} &= \mathrm{EqNil} \\ \mathit{refl}\ (\mathrm{Cons}\ n\ \vec{n}) &= \mathrm{EqCons}\ (\mathit{refl}\ n)\ (\mathit{refl}\ \vec{n}) \end{aligned}$$

let $$\frac{\vec{p} \in \vec{m} = \vec{n} \quad \vec{q} \in \vec{n} = \vec{o}}{\mathit{trans}\ \vec{p}\ \vec{q} \in \vec{m} = \vec{o}}$$

Properties of $=$

let $$\frac{\vec{n} \in \mathbf{List}}{\mathit{refl}\ \vec{n} \in \vec{n} = \vec{n}}$$

$$\begin{aligned} \mathit{refl}\ \mathrm{Nil} &= \mathrm{EqNil} \\ \mathit{refl}\ (\mathrm{Cons}\ n\ \vec{n}) &= \mathrm{EqCons}\ (\mathit{refl}\ n)\ (\mathit{refl}\ \vec{n}) \end{aligned}$$

let $$\frac{\vec{p} \in \vec{m} = \vec{n} \quad \vec{q} \in \vec{n} = \vec{o}}{\mathit{trans}\ \vec{p}\ \vec{q} \in \vec{m} = \vec{o}}$$

$$\begin{aligned} &\mathit{trans}\ \mathrm{EqNil} \qquad\qquad \mathrm{EqNil} = \mathrm{EqNil} \\ &\mathit{trans}\ (\mathrm{EqCons}\ p\ \vec{p})\ (\mathrm{EqCons}\ q\ \vec{p}) \\ &\qquad = \mathrm{EqCons}\ (\mathit{trans}\ p\ q)\ (\mathit{trans}\ \vec{p}\ \vec{q}) \end{aligned}$$

Equality for List$^\infty$

Equality for $\mathbf{List}^\infty$

codata $\dfrac{\vec{m}, \vec{n} \in \mathbf{List}^\infty}{\vec{m} = \vec{n} \in \mathbf{Prop}}$ **where**

Equality for List$^\infty$

codata $\dfrac{\vec{m}, \vec{n} \in \mathbf{List}^\infty}{\vec{m} = \vec{n} \in \mathbf{Prop}}$ **where**

$$\overline{\mathrm{EqNil}^\infty \in \mathrm{Nil}^\infty = \mathrm{Nil}^\infty}$$

Equality for $\mathbf{List}^\infty$

codata $\dfrac{\vec{m}, \vec{n} \in \mathbf{List}^\infty}{\vec{m} = \vec{n} \in \mathbf{Prop}}$ **where**

$$\frac{}{\mathrm{EqNil}^\infty \in \mathrm{Nil}^\infty = \mathrm{Nil}^\infty}$$

$$\frac{p \in m = n \quad \vec{p} \in \vec{m} = \vec{n}}{\mathrm{EqCons}^\infty\ p\ \vec{p} \in \mathrm{Cons}^\infty\ m\ \vec{m} = \mathrm{Cons}^\infty\ n\ \vec{n}}$$

Properties of $=$

Properties of $=$

let $$\frac{\vec{n} \in \mathbf{List}^{\infty}}{\mathit{refl}\ \vec{n} \in \vec{n} = \vec{n}}$$

Properties of $=$

let $$\frac{\vec{n} \in \mathbf{List}^{\infty}}{\mathit{refl}\ \vec{n} \in \vec{n} = \vec{n}}$$

$$\begin{aligned} \mathit{refl}\ \mathrm{Nil}^{\infty} &= \mathrm{EqNil}^{\infty} \\ \mathit{refl}\ (\mathrm{Cons}^{\infty}\ n\ \vec{n}) &= \mathrm{EqCons}^{\infty}\ (\mathit{refl}\ n)\ (\mathit{refl}\ \vec{n}) \end{aligned}$$

Properties of $=$

let $$\frac{\vec{n} \in \textbf{List}^{\infty}}{\mathit{refl}\ \vec{n} \in \vec{n} = \vec{n}}$$

$$\begin{aligned} \mathit{refl}\ \mathrm{Nil}^{\infty} &= \mathrm{EqNil}^{\infty} \\ \mathit{refl}\ (\mathrm{Cons}^{\infty}\ n\ \vec{n}) &= \mathrm{EqCons}^{\infty}\ (\mathit{refl}\ n)\ (\mathit{refl}\ \vec{n}) \end{aligned}$$

let $$\frac{\vec{p} \in \vec{m} = \vec{n} \quad \vec{q} \in \vec{n} = \vec{o}}{\mathit{trans}\ \vec{p}\ \vec{q} \in \vec{m} = \vec{o}}$$

Properties of $=$

let $$\frac{\vec{n} \in \mathbf{List}^{\infty}}{\mathit{refl}\ \vec{n} \in \vec{n} = \vec{n}}$$

$$\begin{aligned} \mathit{refl}\ \mathrm{Nil}^{\infty} &= \mathrm{EqNil}^{\infty} \\ \mathit{refl}\ (\mathrm{Cons}^{\infty}\ n\ \vec{n}) &= \mathrm{EqCons}^{\infty}\ (\mathit{refl}\ n)\ (\mathit{refl}\ \vec{n}) \end{aligned}$$

let $$\frac{\vec{p} \in \vec{m} = \vec{n} \quad \vec{q} \in \vec{n} = \vec{o}}{\mathit{trans}\ \vec{p}\ \vec{q} \in \vec{m} = \vec{o}}$$

$$\begin{aligned} &\mathit{trans}\ \mathrm{EqNil}^{\infty} \qquad \mathrm{EqNil}^{\infty} = \mathrm{EqNil}^{\infty} \\ &\mathit{trans}\ (\mathrm{EqCons}^{\infty}\ p\ \vec{p})\ (\mathrm{EqCons}^{\infty}\ q\ \vec{q}) \\ &\qquad = \mathrm{EqCons}^{\infty}\ (\mathit{trans}\ p\ q)\ (\mathit{trans}\ \vec{p}\ \vec{q}) \end{aligned}$$

A simple proof

let $$\frac{n \in \mathbf{Nat}}{lem\ n \in mapS\ (from\ n) = from\ (n+1)}$$

A simple proof

let $$\frac{n \in \mathbf{Nat}}{\mathit{lem}\ n \in \mathit{mapS}\ (\mathit{from}\ n) = \mathit{from}\ (n+1)}$$

$$\mathit{lem}\ n = \mathrm{EqCons}^{\infty}\ (n+1)\ (\mathit{lem}\ (n+1))$$

A simple proof

let $$\frac{n \in \mathbf{Nat}}{lem\ n \in mapS\ (from\ n) = from\ (n+1)}$$

$$lem\ n = \mathrm{EqCons}^{\infty}\ (n+1)\ (lem\ (n+1))$$

- Coinductive reasoning can be easy.

A simple proof

let $$\frac{n \in \mathbf{Nat}}{\mathit{lem}\ n \in \mathit{mapS}\ (\mathit{from}\ n) = \mathit{from}\ (n+1)}$$

$$\mathit{lem}\ n = \mathrm{EqCons}^{\infty}\ (n+1)\ (\mathit{lem}\ (n+1))$$

- Coinductive reasoning can be easy.
- Guarded coinduction is guarded corecursion on proofs.

A simple proof

let $$\frac{n \in \mathbf{Nat}}{lem\ n \in mapS\ (from\ n) = from\ (n+1)}$$

$$lem\ n = \mathrm{EqCons}^{\infty}\ (n+1)\ (lem\ (n+1))$$

- Coinductive reasoning can be easy.
- Guarded coinduction is guarded corecursion on proofs.
- There is no need to construct bisimulations.

The mirror

The mirror

data	codata

The mirror

data	codata
inductive	

The mirror

data	codata
inductive	coinductive

The mirror

data	codata
inductive	coinductive
finite objects	

The mirror

data	codata
inductive	coinductive
finite objects	infinite objects

The mirror

data	codata
inductive	coinductive
finite objects	infinite objects
structural recursion	

The mirror

data	codata
inductive	coinductive
finite objects	infinite objects
structural recursion	guarded corecursion

The mirror

data	codata
inductive	coinductive
finite objects	infinite objects
structural recursion	guarded corecursion
structural induction	

The mirror

data	codata
inductive	coinductive
finite objects	infinite objects
structural recursion	guarded corecursion
structural induction	guarded coinduction

The mirror

data	codata
inductive	coinductive
finite objects	infinite objects
structural recursion	guarded corecursion
structural induction	guarded coinduction

- Where do functions live?

The mirror

data	codata
inductive	coinductive
finite objects	infinite objects
structural recursion	guarded corecursion
structural induction	guarded coinduction

- Where do functions live?
- Functions are codata.

The mirror

data	codata
inductive	coinductive
finite objects	infinite objects
structural recursion	guarded corecursion
structural induction	guarded coinduction

- Where do functions live?
- Functions are codata.
- Consumer contract:
  You may only apply a function.

Leibniz ?

let $$\frac{P \in \mathbf{Nat} \to \mathbf{Type} \quad \vec{q} \in \vec{m} = \vec{n} \quad \vec{m} \in \mathbf{List} \quad p \in P\ \vec{m}}{\mathit{leibniz}\ P\ \vec{p}\ p \in P\ \vec{n}}$$

Leibniz ?

let $$\frac{P \in \mathbf{Nat} \to \mathbf{Type} \quad \vec{q} \in \vec{m} = \vec{n} \quad \vec{m} \in \mathbf{List} \quad p \in P\ \vec{m}}{\mathit{leibniz}\ P\ \vec{p}\ p \in P\ \vec{n}}$$

$$\begin{array}{lll} \mathit{leibniz}\ P\ \mathrm{EqNil} & \mathrm{Nil} & p = p \end{array}$$

$$\mathit{leibniz}\ P\ (\mathrm{EqCons}\ q\ \vec{q})\ (\mathrm{Cons}\ m\ \vec{m})\ p =$$

$$\quad \mathit{leibniz}\ (\lambda n \to P\ (\mathrm{Cons}\ n\ \vec{m}))\ m\ q$$

$$\quad (\mathit{leibniz}\ (\lambda \vec{n} \to P\ (\mathrm{Cons}\ m\ \vec{n}))\ \vec{m}\ \vec{q}\ p)$$

Leibniz ?

let $\dfrac{P \in \mathbf{Nat} \to \mathbf{Type} \quad \vec{q} \in \vec{m} = \vec{n} \quad \vec{m} \in \mathbf{List} \quad p \in P\ \vec{m}}{\mathit{leibniz}\ P\ \vec{p}\ p \in P\ \vec{n}}$

$$\mathit{leibniz}\ P\ \mathrm{EqNil} \qquad \mathrm{Nil} \qquad p = p$$

$$\mathit{leibniz}\ P\ (\mathrm{EqCons}\ q\ \vec{q})\ (\mathrm{Cons}\ m\ \vec{m})\ p =$$

$$\quad \mathit{leibniz}\ (\lambda n \to P\ (\mathrm{Cons}\ n\ \vec{m}))\ m\ q$$

$$\quad (\mathit{leibniz}\ (\lambda \vec{n} \to P\ (\mathrm{Cons}\ m\ \vec{n}))\ \vec{m}\ \vec{q}\ p)$$

- $\mathit{leibniz}$ doesn't dualize to $\mathbf{List}^{\infty}$.

Observational Type Theory

Observational Type Theory

- We can implement $leibniz$ by internalizing the setoid model – see my LICS 99 paper *Extensional Type Theory, intensionally.*

Observational Type Theory

- We can implement $leibniz$ by internalizing the setoid model – see my LICS 99 paper *Extensional Type Theory, intensionally.*
- Using this construction we implement both consumer and producer contracts without giving up decidability.

Observational Type Theory

- We can implement $leibniz$ by internalizing the setoid model – see my LICS 99 paper *Extensional Type Theory, intensionally.*
- Using this construction we implement both consumer and producer contracts without giving up decidability.
- This is based on a translation of Observational Type Theory into intensional Type Theory + a proof irrelevant universe of propositions.

Observational Type Theory

- We can implement $leibniz$ by internalizing the setoid model – see my LICS 99 paper *Extensional Type Theory, intensionally.*
- Using this construction we implement both consumer and producer contracts without giving up decidability.
- This is based on a translation of Observational Type Theory into intensional Type Theory + a proof irrelevant universe of propositions.
- Alternative: any two hypothetical proofs of False are convertible.

A short history of Type Theory

A short history of Type Theory

Anarchy

A short history of Type Theory

Anarchy

No contracts, not even producer contracts.

A short history of Type Theory

Anarchy

No contracts, not even producer contracts.
Instead of $\Pi\ n \in \mathbf{Nat}$: ... we write
$\Pi\ n \in \mathbf{Nat}.(Ind\ n) \to \ldots$.

A short history of Type Theory

Anarchy

No contracts, not even producer contracts.

Instead of $\Pi\ n \in \mathbf{Nat}$: ... we write

$\Pi\ n \in \mathbf{Nat}.(Ind\ n) \to \ldots.$

Impredicative encodings of data

A short history of Type Theory

Wild West

A short history of Type Theory

Wild West

Producer contracts but no consumer contracts.

A short history of Type Theory

Wild West

Producer contracts but no consumer contracts.
We can quantify over $\mathbf{Nat}$

A short history of Type Theory

Wild West

Producer contracts but no consumer contracts.

We can quantify over $\mathbf{Nat}$

We have to verify again and again that a consumer of codata respects equality.

A short history of Type Theory

Wild West

Producer contracts but no consumer contracts.

We can quantify over $\mathbf{Nat}$

We have to verify again and again that a consumer of codata respects equality.

Intensional Type Theory

A short history of Type Theory

Rule of law

A short history of Type Theory

Rule of law

Producer and consumer contracts.

A short history of Type Theory

Rule of law

Producer and consumer contracts.

We can quantify over $\mathbf{Nat}$

A short history of Type Theory

Rule of law

Producer and consumer contracts.
We can quantify over Nat
We know that any consumer of codata respects equality.

A short history of Type Theory

Rule of law

Producer and consumer contracts.
We can quantify over $\mathbf{Nat}$
We know that any consumer of codata respects equality.
Observational Type Theory

Observational Epigram

Observational Epigram

- The goal of our recently funded EPSRC project *Decidable Type Theory with Observational Equality* is to implement a Type Theory with observational equality (Observational Epigram)

Observational Epigram

- The goal of our recently funded EPSRC project *Decidable Type Theory with Observational Equality* is to implement a Type Theory with observational equality (Observational Epigram)
- We want to improve on my LICS 99 paper by adding the conversion equality
  $\mathit{leibniz} \ldots \mathit{refl}\ x \equiv x$

Observational Epigram

- The goal of our recently funded EPSRC project *Decidable Type Theory with Observational Equality* is to implement a Type Theory with observational equality (Observational Epigram)
- We want to improve on my LICS 99 paper by adding the conversion equality
  $leibniz \ldots refl\ x \equiv x$
- And hence strictly extend intensional Type Theory.

Observational Epigram

- The goal of our recently funded EPSRC project *Decidable Type Theory with Observational Equality* is to implement a Type Theory with observational equality (Observational Epigram)
- We want to improve on my LICS 99 paper by adding the conversion equality $\mathit{leibniz} \ldots \mathit{refl}\ x \equiv x$
- And hence strictly extend intensional Type Theory.
- We also want to realize another extension of the mirror:

Observational Epigram

- The goal of our recently funded EPSRC project *Decidable Type Theory with Observational Equality* is to implement a Type Theory with observational equality (Observational Epigram)
- We want to improve on my LICS 99 paper by adding the conversion equality $leibniz \ldots refl\ x \equiv x$
- And hence strictly extend intensional Type Theory.
- We also want to realize another extension of the mirror:

data	codata
subset types	quotient types