Observational Equality, now!

joint work with Conor McBride and Wouter Swierstra

supported by EPSRC grant EP/C512022
Observational Equality for Dependently Typed Programming

Thorsten Altenkirch

School of Computer Science
University of Nottingham

October 5, 2007

What is happening with Epigram 2?

- Observational Equality is implemented as part of the core of Epigram 2.
- Thanks to Conor McBride, Nicolas Oury, Wouter Swierstra, Peter Morris and James Chapman.
- **Today:** How to steal (most of) observational equality for existing systems using generic programming.
- Verification of metatheoretic properties by translation.

Dependently typed programming (DTP)

- Languages:

  phase-insensitive:

  Cayenne, Epigram, Agda, ...

  phase-sensitive:

  DML, Ωmega, Haskell with GADTs, ...

- Equality:

$$\mathsf{Vec} : \mathsf{Nat} \to \mathsf{Set} \to \mathsf{Set}$$

$$as : \mathsf{Vec}\,(x + y)\,A$$

  how to obtain

$$??? : \mathsf{Vec}\,(y + x)\,A$$

  using that $x + y = y + x$.

Extensional vs. Intensional ?

- ETT Extensional Type Theory
- ITT Intensional Type Theory
- OTT Observational Type Theory

	ETT	ITT	OTT
defn vs. prop. eq	$=$	$\neq$	$\neq$
decidable typechecking	-	+	+
open normalisation	-	+	+
obs. equality	+	-	+

Equality basics

- Equality type (propositional equality)

$$\frac{\vdash A : \mathrm{Set} \quad a, b : A}{\vdash a =_A b : \mathrm{Prop}}$$

- Introduction:

$$\frac{\vdash a : A}{\vdash \mathrm{refl}_A\, a : a =_A a}$$

- Definitional equality, e.g. $0 + x \equiv x$.
- Conversion rule

$$\frac{\vdash s : S \quad \vdash S \equiv T}{\vdash s : T}$$

- Embedding:

if $\vdash a \equiv b : A$ then $\vdash a =_A a \equiv a =_A b : \mathrm{Prop}$
and therefore $\vdash \mathrm{refl}_A\, a : a =_A b$

Using equality in ETT

- Equality reflection

$$\frac{\vdash\ q : a =_A b}{\vdash\ a \equiv b : A}$$

- q has disappeared $\implies$ $\equiv$ undecidable.
- Extensionality law is provable:

$$\text{if}\ \ \forall x.\, f\ x = g\ x\ \ \text{then}\ \ (\lambda x.\, f\ x) = (\lambda x.\, g\ x)\ \ \text{so}\ \ f = g$$

Using equality in ITT

- Equality elimination

$$\frac{\vdash\ q : a =_A b \quad \vdash\ T : A \to \mathrm{Set} \quad \vdash\ t : T\ a}{\mathrm{subst}_{A;a;b}\ q\ T\ t : T\ b}$$

with the associated computational rule

$$\vdash\ \mathrm{subst}_{A;a;a}\ (\mathrm{refl}_A\ a)\ T\ t \equiv t : T\ a$$

- More bureaucratic (every coercion has to be marked).
- Extensionality is not provable, e.g. we can show

$$\mathrm{plus0} : \forall x.\ 0{+}x = x{+}0$$

but there is no closed proof of:

$$\lambda x.\ 0{+}x = \lambda x.\ x{+}0$$

Extensionality as an axiom?

- Why don't we just add an axiom?

$$\frac{q : \forall x.\, f\, x = g\, x}{\mathrm{ext}\, q : f = g}$$

- We loose canonicity! E.g.

$$\mathrm{subst}\, (\mathrm{ext}\, \mathrm{plus0})\, (\lambda\_.\, \mathsf{Nat})\, 0 : \mathsf{Nat}$$

cannot be reduced to a numeral.

A brief history of equality

Hofmann(PhD 95) : Setoid model to define extensional equality
no large elims.

Hofmann(Types 95) : Conservativity of equality reflection
but we loose canonicity.

A.(LICS 99) : Setoid model with proof-irrelevant proposition
not conservative over ITT.

McBride (PhD 99) Heterogenous equality
also called *John Major equality*

Oury(TPHOL 05) : Equality reflection for CoC
extending Hofmann's approach.

- Equality between sets (computed!) and coercions:

$$\frac{S, T : \mathrm{Set}}{S = T : \mathrm{Prop}} \qquad \frac{Q : S = T \quad s : S}{s\,[Q{:}S{=}T\rangle : T}$$

- Heterogenous equality (computed) between values:

$$\frac{s : S \quad t : T}{(s : S) = (t : T) : \mathrm{Prop}}$$

- Why heterogenous? Dependent functions preserve equality:

$$\forall x, y.\, (x : A) = (y : A) \rightarrow (f\ x : B[x]) = (f\ y : B[y])$$

- Coherence

$$\frac{Q : S = T \quad s : S}{\{s \parallel Q{:}S{=}T\} : (s : S) = (s\,[Q{:}S{=}T\rangle : T)}$$

also requires heterogenous equality!

A simple Core Type Theory

$$
\begin{array}{lll}
\text{set} & \mathbf{S} ::= & \mathbf{G} \mid \mathbf{B}\,\mathbf{X}:\mathbf{S}.\,\mathbf{S} \mid \mathsf{If}\ \mathbf{T}\ \mathsf{Then}\ \mathbf{S}\ \mathsf{Else}\ \mathbf{S} \\
\text{ground} & \mathbf{G} ::= & 0 \mid 1 \mid 2 \\
\text{binder} & \mathbf{B} ::= & \Pi \mid \Sigma \mid \mathsf{W} \\
\text{term} & \mathbf{T} ::= & \langle\rangle \mid \mathtt{tt} \mid \mathtt{ff} \mid \lambda \mathbf{X}:\mathbf{S}.\,\mathbf{T} \mid \langle \mathbf{T},\mathbf{T}\rangle_{\Sigma \mathbf{X}:\mathbf{S}.\,\mathbf{S}} \mid \mathbf{T} \triangleleft_{\mathsf{W}\mathbf{X}:\mathbf{S}.\,\mathbf{S}} \mathbf{T} \\
& \mid & \mathbf{T}!\mathbf{S} \mid \mathsf{if}\ \mathbf{T}/\mathbf{X}.\mathbf{S}\ \mathsf{then}\ \mathbf{T}\ \mathsf{else}\ \mathbf{T} \\
& \mid & \mathbf{T}\,\mathbf{T} \mid \mathsf{fst}\ \mathbf{T} \mid \mathsf{snd}\ \mathbf{T} \mid \mathsf{rec}\ \mathbf{T}/\mathbf{X}.\mathbf{S}\ \mathsf{with}\ \mathbf{T}
\end{array}
$$

Typing rules (see paper), e.g.

$$
\frac{\Gamma \vdash s : S \quad \Gamma \vdash f : T[s] \to \mathsf{W} x : S.\, T}{\Gamma \vdash s \triangleleft_{\mathsf{W} x : S.\, T} f : \mathsf{W} x : S.\, T}
$$

Encoding of datatypes

- Disjoint union:

$$\begin{aligned} S+T &\mapsto \Sigma b{:}2.\ \mathsf{If}\ b\ \mathsf{Then}\ S\ \mathsf{Else}\ T \\ \mathrm{inl}\ s &\mapsto \langle \mathtt{tt}, s\rangle \\ \mathrm{inr}\ t &\mapsto \langle \mathtt{ff}, t\rangle \end{aligned}$$

- Natural numbers:

$$\begin{aligned} \mathrm{Tr}\ b &\mapsto \mathsf{If}\ b\ \mathsf{Then}\ 1\ \mathsf{Else}\ 0 \\ \mathrm{Nat} &\mapsto \mathsf{W} b{:}2.\ \mathrm{Tr}\ b \\ \mathrm{zero} &\mapsto \mathtt{ff} \triangleleft \lambda z.\ z!\mathrm{Nat} \\ \mathrm{suc}\ n &\mapsto \mathtt{tt} \triangleleft \lambda\_.\ n \end{aligned}$$

- Primitive recursion:

$$\begin{aligned} \mathrm{plus} \mapsto\ & \lambda x\ y.\ \mathsf{rec}\ x\ \mathsf{with} \\ & \lambda b.\ \mathsf{if}\ b\ \mathsf{then}\ \lambda f\ h.\ \mathrm{suc}\ (h\ \langle\rangle)\ \mathsf{else}\ \lambda f\ h.\ y \end{aligned}$$

A problem: induction / dependent recursion

We would like:

$$\mathrm{ind}_P : P[\mathrm{zero}] \to (\Pi n{:}\mathrm{Nat}.\ P[n] \to P[\mathrm{suc}\ n]) \to \Pi n{:}\mathrm{Nat}.\ P[n]$$

but the obvious program doesn't type check:

$$\mathrm{ind}_P \mapsto \lambda pz\ ps\ n.\ \mathsf{rec}\ n\ \mathsf{with}\ \lambda b.\ \mathsf{if}\ b\ \mathsf{then}\ \lambda f\ h.\ ps\,(f\,\langle\rangle)\,(h\,\langle\rangle)\ \mathsf{else}\ \lambda f\ h.\ pz$$

Too many possible implementations of zero such as:

$$\mathrm{zero}' \mapsto \mathsf{ff} \triangleleft \lambda z.\ \mathrm{suc}\ (\mathrm{suc}\ \mathrm{zero})$$

Encoding the core theory in Agda 2

data Empty : Set **where**

record Unit : Set **where**

data Bool : Set **where**
  tt : Bool
  ff : Bool

record $\Sigma\ (S : \mathrm{Set})(T : S \to \mathrm{Set})\ :\ \mathrm{Set}$ **where**
  $\mathrm{fst} : S$
  $\mathrm{snd} : T\ \mathrm{fst}$

data $\mathrm{W}\ (S : \mathrm{Set})(T : S \to \mathrm{Set})\ :\ \mathrm{Set}$ **where**
  $\_\triangleleft\_ : (x : S) \to (T\ x \to \mathrm{W}\ S\ T) \to \mathrm{W}\ S\ T$

An inductive-recursive universe

mutual

data ‘set’ : Set **where**

‘0’, ‘1’, ‘2’ : ‘set’

‘Π’, ‘Σ’, ‘W’ : $(S : \text{‘set’}) \rightarrow (\llbracket S \rrbracket \rightarrow \text{‘set’}) \rightarrow \text{‘set’}$

$$
\begin{aligned}
&\llbracket \_ \rrbracket : \text{‘set’} \rightarrow \text{Set} \\
&\llbracket \text{‘0’} \rrbracket = \text{Empty} \\
&\llbracket \text{‘1’} \rrbracket = \text{Unit} \\
&\llbracket \text{‘2’} \rrbracket = \text{Bool} \\
&\llbracket \text{‘}\Pi\text{’}\ S\ T \rrbracket = (x : \llbracket S \rrbracket) \rightarrow \llbracket T\ x \rrbracket \\
&\llbracket \text{‘}\Sigma\text{’}\ S\ T \rrbracket = \Sigma\ \llbracket S \rrbracket\ (\lambda x \mapsto \llbracket T\ x \rrbracket) \\
&\llbracket \text{‘W’}\ S\ T \rrbracket = \text{W}\ \llbracket S \rrbracket\ (\lambda x \mapsto \llbracket T\ x \rrbracket)
\end{aligned}
$$

A propositional fragment

$$\mathbf{P} ::= \ \bot \mid \top \mid \mathbf{P} \wedge \mathbf{P} \mid \forall \mathbf{X} : \mathbf{S}.\ \mathbf{P}$$

mutual
  data 'prop' : Set **where**
    '⊥','⊤' : 'prop'
    '∧' : 'prop' → 'prop' → 'prop'
    '∀' : (S : 'set') → ($[\![S]\!]$ → 'prop') → 'prop'
  $\lceil \_ \rceil$: 'prop' → 'set'
  ...

Equality of types

$$\frac{\Gamma \vdash S\ \textbf{set} \quad \Gamma \vdash T\ \textbf{set}}{\Gamma \vdash S = T\ \textbf{prop}} \qquad \frac{\Gamma \vdash Q : \lceil S = T \rceil \quad \Gamma \vdash s : S}{\Gamma \vdash s\,[Q{:}S{=}T\rangle : T}$$

- We are going to define $S = T$ by recursion over S, T.
- and then $s\,[Q{:}S{=}T\rangle$ by inspecting s and Q.

The easy cases

$$0 = 0 \mapsto \top$$
$$1 = 1 \mapsto \top$$
$$2 = 2 \mapsto \top$$

$$z\,[Q\colon 0{=}0\rangle \mapsto z$$
$$u\,[Q\colon 1{=}1\rangle \mapsto u$$
$$b\,[Q\colon 2{=}2\rangle \mapsto b$$

The not so easy cases...

$$
\begin{array}{rcll}
(\Pi x_0 : S_0.\ T_0) = (\Pi x_1 : S_1.\ T_1) & \mapsto & ? & \\
(\Sigma x_0 : S_0.\ T_0) = (\Sigma x_1 : S_1.\ T_1) & \mapsto & ? & \\
(\mathsf{W} x_0 : S_0.\ T_0) = (\mathsf{W} x_1 : S_1.\ T_1) & \mapsto & ? & \\
S = T & \mapsto & \bot & \text{for other canonical sets}
\end{array}
$$

$$
\begin{array}{rcll}
f_0\ [Q : \Pi x_0 : S_0.\ T_0 = \Pi x_1 : S_1.\ T_1\ \rangle & \mapsto & ? & \\
p_0\ [Q : \Sigma x_0 : S_0.\ T_0 = \Sigma x_1 : S_1.\ T_1\ \rangle & \mapsto & ? & \\
(s_0 \lhd f_0)\ [Q : \mathsf{W} x_0 : S_0.\ T_0 = \mathsf{W} x_1 : S_1.\ T_1\ \rangle & \mapsto & ? & \\
x\ [Q : S = T\ \rangle & \mapsto & Q!T & \text{otherwise}
\end{array}
$$

Σ-types

$$(\Sigma x_0 : S_0.\ T_0) = (\Sigma x_1 : S_1.\ T_1) \mapsto S_0 = S_1 \wedge \forall x_0 : S_0.\ \forall x_1 : S_1.\ (x_0 : S_0) = (x_1 : S_1) \Rightarrow T_0[x_0] = T_1[x_1]$$

$$\begin{array}{l} \ldots ; \langle \mathrm{Q}_S , \mathrm{Q}_T \rangle : (\Sigma x_0 : S_0.\ T_0) = (\Sigma x_1 : S_1.\ T_1); \\ \vdash \langle \mathrm{s}_0 , \mathrm{t}_0 \rangle\ [\langle \mathrm{Q}_S , \mathrm{Q}_T \rangle\rangle \mapsto \textbf{let} \\ \qquad \mathrm{s}_1 \mapsto \mathrm{s}_0\ [\mathrm{Q}_S\rangle : S_1 \\ \qquad \mathrm{R} \mapsto \mathrm{Q}_T\ \mathrm{s}_0\ \mathrm{s}_1\ \{\mathrm{s}_0 \parallel \mathrm{Q}_S\} : \lceil T_0[\mathrm{s}_0] = T_1[\mathrm{s}_1] \rceil \\ \qquad \mathrm{t}_1 \mapsto \mathrm{t}_0\ [\mathrm{R}\rangle : T_1[\mathrm{s}_1] \\ \quad \textbf{in}\ \langle \mathrm{s}_1 , \mathrm{t}_1 \rangle : \Sigma x_1 : S_1.\ T_1 \end{array}$$

Π-types

$$
\begin{array}{l}
(\Pi x_0 : S_0.\ T_0) = (\Pi x_1 : S_1.\ T_1) \ \mapsto \\
\quad S_1 = S_0 \wedge \\
\quad \forall x_1 : S_1.\ \forall x_0 : S_0.\ (x_1 : S_1) = (x_0 : S_0) \Rightarrow T_0[x_0] = T_1[x_1]
\end{array}
$$

$$
\begin{array}{l}
\quad \ldots ; \langle \mathrm{Q}_S, \mathrm{Q}_T \rangle : (\Pi x_0 : S_0.\ T_0) = (\Pi x_1 : S_1.\ T_1); \\
\vdash\ f_0\ [\langle \mathrm{Q}_S, \mathrm{Q}_T \rangle\rangle \mapsto \lambda s_1.\ \mathbf{let} \\
\qquad\qquad \mathrm{s}_0 \mapsto \mathrm{s}_1\ [\mathrm{Q}_S\rangle : S_0 \\
\qquad\qquad \mathrm{t}_0 \mapsto f_0\ \mathrm{s}_0 : T_0[\mathrm{s}_0] \\
\qquad\qquad \mathrm{R} \mapsto \mathrm{Q}_T\ s_1\ \mathrm{s}_0\ \{s_1 \parallel \mathrm{Q}_S\} : \lceil T_0[\mathrm{s}_0] = T_1[s_1] \rceil \\
\qquad\qquad \mathrm{t}_1 \mapsto \mathrm{t}_0\ [\mathrm{R}\rangle : T_1[s_1] \\
\qquad \mathbf{in}\ \mathrm{t}_1
\end{array}
$$

W-types

See paper.

Value equality

$$\frac{\Gamma \vdash s : S \quad \Gamma \vdash t : T}{\Gamma \vdash (s : S) = (t : T)\ \mathbf{prop}}$$

$$\frac{\Gamma \vdash Q : \lceil S = T \rceil \quad \Gamma \vdash s : S}{\Gamma \vdash \{s \parallel Q{:}S{=}T\} : \lceil (s : S) = (s\,[Q{:}S{=}T\rangle : T) \rceil}$$

- We define $(s : S) = (t : T)$ by inspecting s, t.
- We are not going to define $\{s \parallel Q{:}S{=}T\}$ even though we could.

The easy cases

$$
\begin{aligned}
(z_0 : \mathbf{0}) = (z_1 : \mathbf{0}) &\mapsto \top \\
(u_0 : \mathbf{1}) = (u_1 : \mathbf{1}) &\mapsto \top \\
(\mathtt{tt} : \mathbf{2}) = (\mathtt{tt} : \mathbf{2}) &\mapsto \top \\
(\mathtt{tt} : \mathbf{2}) = (\mathtt{ff} : \mathbf{2}) &\mapsto \bot \\
(\mathtt{ff} : \mathbf{2}) = (\mathtt{tt} : \mathbf{2}) &\mapsto \bot \\
(\mathtt{ff} : \mathbf{2}) = (\mathtt{ff} : \mathbf{2}) &\mapsto \top
\end{aligned}
$$

Equality of functions

$$
\begin{array}{l}
(f_0 : \Pi x_0 : S_0.\ T_0) = (f_1 : \Pi x_1 : S_1.\ T_1) \mapsto \\
\quad \forall x_0 : S_0.\ \forall x_1 : S_1.\ (x_0 : S_0) = (x_1 : S_1) \Rightarrow \\
\quad\quad (f_0\ x_0 : T_0[x_0]) = (f_1\ x_1 : T_1[x_1])
\end{array}
$$

Equality of pairs

$$
\begin{aligned}
&(p_0 : \Sigma x_0 : S_0.\ T_0) = (p_1 : \Sigma x_1 : S_1.\ T_1) \mapsto \\
&\quad (\mathsf{fst}\ p_0 : S_0) = (\mathsf{fst}\ p_1 : S_1) \wedge \\
&\quad (\mathsf{snd}\ p_0 : T_0[\mathsf{fst}\ p_0]) = (\mathsf{snd}\ p_1 : T_1[\mathsf{fst}\ p_1])
\end{aligned}
$$

Strong Normalisation

Lemma (Strong Normalisation)

OTT is strongly normalising.

SKETCH OF PROOF SKETCH

Model the universe construction in a known strongly normalizing Type Theory (e.g. CIC).

Is there something missing?

- We haven't added equations for coherence:

$$\frac{\Gamma \vdash Q : \lceil S = T \rceil \quad \Gamma \vdash s : S}{\Gamma \vdash \{s \parallel Q{:}S{=}T\} : \lceil (s : S) = (s\,[Q{:}S{=}T\rangle : T) \rceil}$$

- We haven't defined reflexivity:

$$\frac{\Gamma \vdash s : S}{\Gamma \vdash \overline{s{:}S} : \lceil (s : S) = (s : S) \rceil}$$

- We haven't defined respectfulness:

$$\frac{\Gamma \vdash S \textbf{ set} \quad \Gamma; x{:}S \vdash T \textbf{ set}}{\Gamma \vdash \mathrm{R}x{:}S.\,T : \lceil \forall y{:}S.\,\forall z{:}S.\,(y : S) = (z : S) \Rightarrow T[y] = T[z] \rceil}$$

- And indeed, we are not going to add equations for any of those constants!

What about canonicity ?

- We have introduced constants without equations!
- We could actually define coherence $\{s \parallel Q{:}S = T\}$.
- But not reflexivity ($\overline{s : S}$) or respect ($\mathrm{R}x : S$)
  because they have to be shown by induction on terms, not types.
- Are we back at square 1?
  We could have just added extensionality?

Canonicity from consistency

Lemma (Canonicity from Consistency)

Suppose OTT is consistent, i.e. that there is no s such that $\mathcal{E} \vdash s : 0$*. Then, for all normal S and s,*

- *if* $\mathcal{E} \vdash S\ \mathbf{set}$ *then S is canonical;*
- *if* $\mathcal{E} \vdash s : S$ *then either s is canonical, or s is a proof.*

Consistency from the Extensional Theory

Theorem (Consistency)

There is no s such that $\mathcal{E} \vdash s : 0$.

SKETCHY PROOF SKETCH : Model OTT in ETT.

Corollary (Canonicity)

If $\mathcal{E} \vdash S\ \mathbf{set}$ *then S is canonical.*
If $\mathcal{E} \vdash s : S$ *then s is either canonical or a proof.*

Induction for natural numbers

$$\mathrm{ind}_P : P[\mathrm{zero}] \to (\Pi n{:}\mathrm{Nat}.\ P[n] \to P[\mathrm{suc}\ n]) \to \Pi n{:}\mathrm{Nat}.\ P[n]$$

$$\begin{aligned}\mathrm{ind}_P \mapsto\ & \lambda pz\ ps\ n.\ \mathsf{rec}\ n\ \mathsf{with} \\ & \quad \lambda b.\ \mathsf{if}\ b\ \mathsf{then}\ \lambda f\ h.\ ps\ (f\ \langle\rangle)\ (h\ \langle\rangle) \\ & \qquad\qquad [?{:}P[\mathrm{suc}\ (f\ \langle\rangle)] = P[\mathbf{tt} \lhd f]\ \rangle \\ & \qquad \mathsf{else}\ \lambda f\ h.\ pz\ [?{:}P[\mathrm{zero}] = P[\mathbf{ff} \lhd f]\ \rangle\end{aligned}$$

- See paper on how to fill the ?s.

Conservativity over ITT?

- Definitional laws like

$$\mathrm{ind}_P\ pz\ ps\ \mathrm{zero} \mapsto pz$$

  do not hold *definitionally*!
- Instead we have:

$$\mathrm{ind}_P\ pz\ ps\ \mathrm{zero} \mapsto pz\ [\cdots : P[\mathrm{zero}] = P[\mathrm{zero}]\ \rangle$$

- Note that the coercion coerces definitionally equal types!
- We solve this problem by defining a quotation operation on normal forms, which eliminates unnecessary coercions.
- You have to modify definitional equality to do this. (not **now**!)

Summary

- We introduce OTT:
  an intensional Type Theory
  with extensional propositional equality.
- Can be implemented within existing ITT
  using a universe construction.
- We show via the embedding that OTT is normalizing,
  definitional equality and type checking are decidable
- Canonicity holds for non-propositional types
  this follows from the consistency of the extensional theory.
- OTT's definitional equality is conservative over ITT
  this requires a modified definitional equality.

Missing pieces

- Carry out the details of the encoding in CIC.
- Definitionally redundant constructors?
- Show that ETT is a conservative extension of OTT.
- Coinductive data.
- Quotient types.
- Do we need the consistency of ETT?