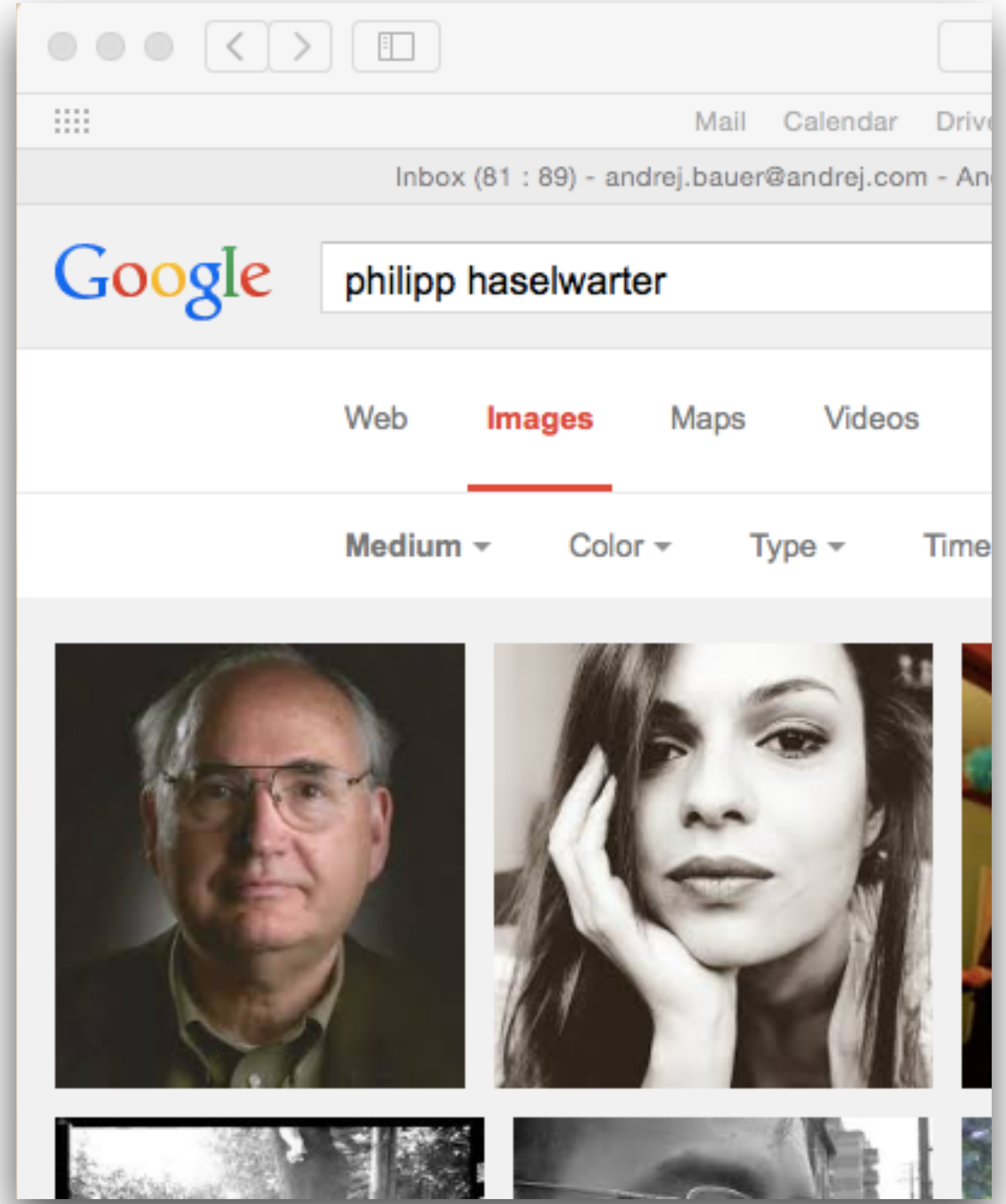
The troublesome reflection rule

Andrej Bauer
University of Ljubljana

TYPES 2015
Tallinn, May 18–21, 2015

Matija Pretnar

Chris Stone



Philipp Haselwarter?

Talk outline

- Equality reflection – bad & good
- Current development
- Future possibilities

$$\Gamma \vdash A \ : \ \mathsf{Type}$$

$$\Gamma \vdash e \ : \ A$$

$$\Gamma \vdash e_1 \equiv_A e_2$$

$$\Gamma \vdash A \equiv_{\mathsf{Type}} B$$

$$\Gamma \vdash \mathsf{Type} \quad : \quad \mathsf{Type}$$

$$\Gamma \vdash e \; : \; A$$

$$\Gamma \vdash e_1 \equiv_A e_2$$

Dependent product

$$\frac{\Gamma, x{:}A \vdash B \ : \ \mathsf{Type}}{\Gamma \vdash \prod_{x:A} B \ : \ \mathsf{Type}} \qquad\qquad \frac{\Gamma, x{:}A \vdash e \ : \ B}{\Gamma \vdash (\lambda x{:}A \,.\, e) \ : \ \prod_{x:A} B}$$

$$\frac{\Gamma \vdash e_1 \ : \ \prod_{x:A} B \qquad \Gamma \vdash e_2 \ : \ A}{\Gamma \vdash e_1\, e_2 \ : \ B[e_2/x]}$$

$$(\lambda x{:}A \,.\, e_1)\, e_2 \equiv_{B[e_2/x]} e_1[e_2/x]$$

$$(\lambda x{:}A \,.\, e\, x) \equiv_{\prod_{x:A} B} e$$

Equality

$$\frac{\Gamma \vdash A \ : \ \mathsf{Type} \qquad \Gamma \vdash a \ : \ A \qquad \Gamma \vdash b \ : \ A}{\Gamma \vdash \mathsf{Eq}_A(a, b) \ : \ \mathsf{Type}}$$

$$\frac{\Gamma \vdash a \ : \ A}{\Gamma \vdash \mathsf{refl}_A(a) \ : \ \mathsf{Eq}_A(a, a)} \qquad \frac{\Gamma \vdash p \ : \ \mathsf{Eq}_A(a, b)}{\Gamma \vdash a \equiv_A b}$$

$$p \equiv_{\mathsf{Eq}_A(a,b)} \mathsf{refl}_A(a)$$

J eliminator

$$\frac{\begin{array}{c}\Gamma \vdash a, b \;:\; A \\ \Gamma \vdash p \;:\; \mathsf{Eq}_A(a, b) \\ \Gamma, x{:}A \vdash c \;:\; C(x, x, \mathsf{refl}(x))\end{array}}{\Gamma \vdash \mathsf{J}(\ldots) \;:\; C(a, b, p)}$$

just use c[a/x] for J(..)

$$
\begin{aligned}
A &: \mathsf{Type}, \\
a &: A, \\
B &: \mathsf{Type}, \\
b &: B, \\
p &: \mathsf{Eq}_{\mathsf{Type}}(A, B), \\
q &: \mathsf{Eq}_{A}(a, b)
\end{aligned}
$$

cannot strengthen

cannot exchange

$$p : \mathsf{Eq}_{\mathsf{Type}}(\mathsf{nat} \longrightarrow \mathsf{nat}, \mathsf{nat} \longrightarrow \mathsf{bool})$$

$$\vdash$$

$$0 : \mathsf{bool}$$

$$
\begin{aligned}
\mathsf{nat} &: \mathsf{Type} \\
\mathsf{Z} &: \mathsf{nat} \\
\mathsf{S} &: \mathsf{nat} \longrightarrow \mathsf{nat} \\
\mathsf{ind} &: \textstyle\prod_{P:\mathsf{nat}\longrightarrow\mathsf{Type}} P\mathsf{Z} \longrightarrow \left(\prod_{n:\mathsf{nat}} P\,n \longrightarrow P\,(\mathsf{S}\,n)\right) \longrightarrow \prod_{m:\mathsf{nat}} P\,m
\end{aligned}
$$

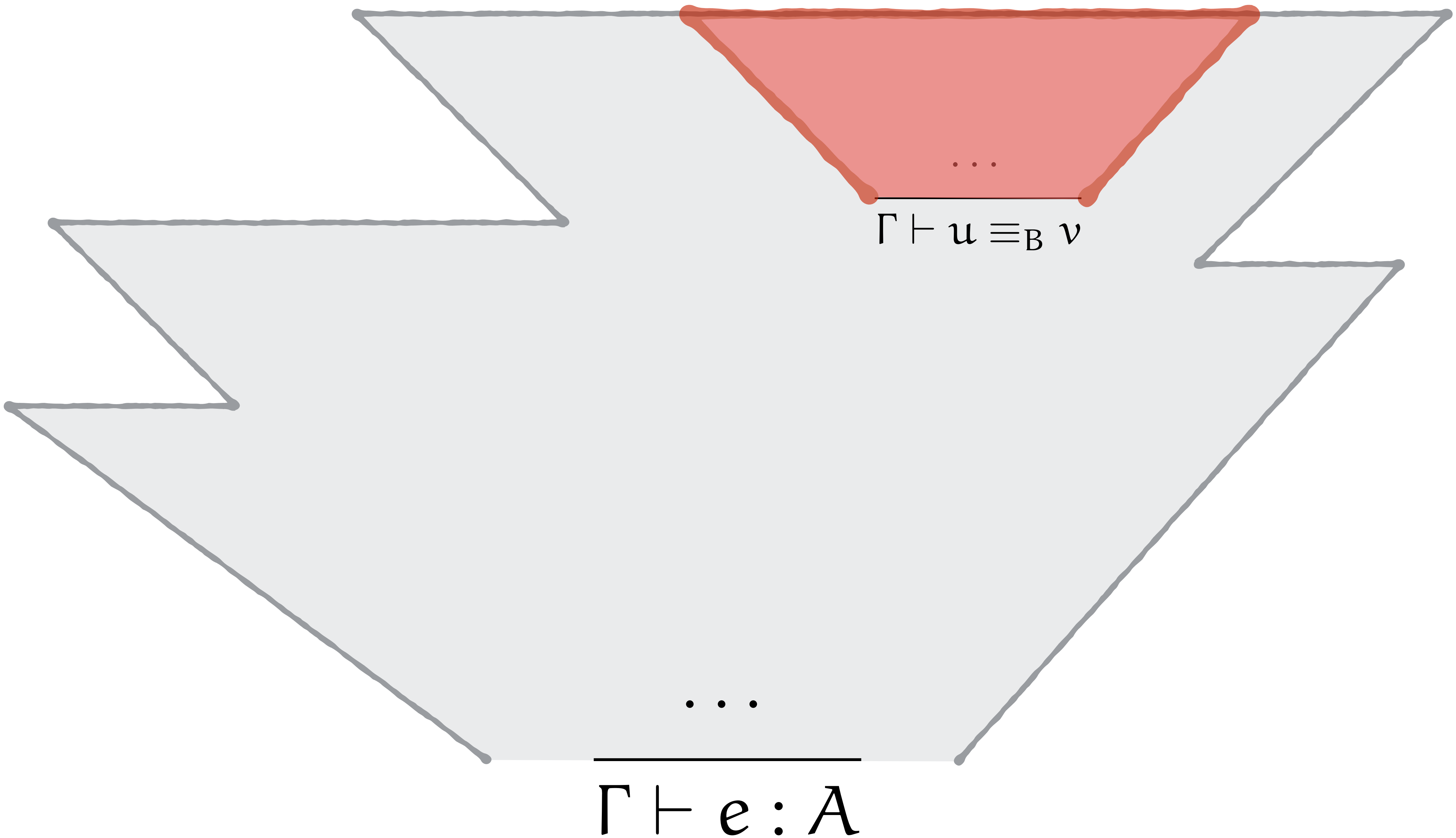
$$
\begin{aligned}
\beta_{\mathsf{Z}} :\ & \textstyle\prod_{P:\mathsf{nat}\longrightarrow\mathsf{Type}} \prod_{x:P\mathsf{Z}} \prod_{f:(\Pi_{n:\mathsf{nat}} P\,n \longrightarrow P\,(\mathsf{S}\,n))} \\
& \qquad \mathsf{Eq}_{P\mathsf{Z}}(\mathsf{ind}\,P\,x\,f\,\mathsf{Z}, x) \\
\beta_{\mathsf{S}} :\ & \textstyle\prod_{P:\mathsf{nat}\longrightarrow\mathsf{Type}} \prod_{x:P\mathsf{Z}} \prod_{f:(\Pi_{n:\mathsf{nat}} P\,n \longrightarrow P\,(\mathsf{S}\,n))} \prod_{n:\mathsf{nat}} \\
& \qquad \mathsf{Eq}_{P\mathsf{Z}}(\mathsf{ind}\,P\,x\,f\,(\mathsf{S}\,n), f\,n\,(\mathsf{ind}\,P\,x\,f\,n))
\end{aligned}
$$

Extensional type theories

- Nuprl: based on a PER model over an untyped calculus with strong normalization

- HOL: simply typed and classical

- We would like a dependently typed system which is not bound to a single model



. . .
$\Gamma \vdash u \equiv_B v$
. . .
$\Gamma \vdash e : A$

Type theory as a programming language

- **Input:** a program that derives a judgment
- **Evaluation:** construction of the derivation
- **Equality checking:** computational effect
- **Output:** the derived judgment

Input computations

`x`	variable
`Type`	universe
$\prod x{:}c_1.c_2$	product
$\lambda x{:}c_1.c_2$	abstraction
$c_1\ c_2$	application
$\mathtt{Eq}(c_1,c_2)$	equality type
`refl(c)`	reflexivity
$c_1::c_2$	ascription
$\mathtt{hint}\ c_1\ \mathtt{in}\ c_2$	general hint
$\mathtt{beta}\ c_1\ \mathtt{in}\ c_2$	β-hint
$\mathtt{eta}\ c_1\ \mathtt{in}\ c_2$	η-hint

handlers directing equality checks through reflection

Output terms & types

x	variable
$\mathtt{Type}$	universe
$\prod x{:}A.B$	product
$\lambda x{:}A.(e{:}B)$	abstraction
$e_1 \; ^{@(x:A.B)} \; e_2$	application
$\mathtt{Eq}_A(e_1, e_2)$	equality type
$\mathtt{refl}_A(e)$	reflexivity

Output terms & types

x	variable
$\mathtt{Type}$	universe
$\prod x{:}A.B$	product
$\lambda x{:}A.(e{:}B)$	abstraction
$e_1 \ ^{@(x:A.B)}\ e_2$	application
$\mathrm{Eq}_A(e_1,e_2)$	equality type
$\mathrm{refl}_A(e)$	reflexivity

β-rule

$$\frac{\Gamma \vdash A_1 \equiv A_2 \qquad\qquad \Gamma, x{:}A_1 \vdash B_1 \equiv B_2}{\Gamma \vdash ((\lambda x{:}A_1.e_1{:}B_1)\ ^{@(x:A_2.B_2)}\ e_2) \equiv e_1[e_2/x]}$$

$$(\lambda x{:}nat.x{:}nat)\ ^{@(nat\to bool)}\ 0 \not\equiv 0$$

Operational semantics

$$\Gamma;\mathscr{E} \vdash \mathtt{c} \rightsquigarrow (\mathtt{e},\mathtt{A})$$

"In context Γ using hints $\mathscr{E}$
computation c evaluates to (e,A)"

Soundness:

If $\Gamma;\mathscr{E} \vdash \mathtt{c} \rightsquigarrow (\mathtt{e},\mathtt{A})$ then $\Gamma \vdash \mathtt{e}:\mathtt{A}$.

$$\frac{(x : A) \in \Gamma}{\Gamma ; \mathscr{E} \vdash x \rightsquigarrow (x, A)}$$

$$\frac{}{\Gamma ; \mathscr{E} \vdash \mathrm{Type} \rightsquigarrow (\mathrm{Type}, \mathrm{Type})}$$

$$\frac{\begin{array}{l} \Gamma ; \mathscr{E} \vdash c_1 \rightsquigarrow (A, T_1) \\ \Gamma ; \mathscr{E} \vdash T_1 \equiv_{\mathrm{Type}} \mathrm{Type} \\ \Gamma , x : A ; \mathscr{E} \vdash c_1 \rightsquigarrow (B, T_2) \\ \Gamma , x : A ; \mathscr{E} \vdash T_2 \equiv_{\mathrm{Type}} \mathrm{Type} \end{array}}{\Gamma ; \mathscr{E} \vdash \prod x : c_1 . c_2 \rightsquigarrow \prod x : A . B}$$

“normalization”

$$\frac{\begin{array}{l}\Gamma;\mathscr{E} \vdash c_1 \rightsquigarrow (e_1, A_1) \\ \Gamma;\mathscr{E} \vdash A_1 \mapsto^{\mathrm{whnf}} \Pi x{:}C.D \\ \Gamma;\mathscr{E} \vdash c_2 \rightsquigarrow (e_2, A_2) \\ \Gamma;\mathscr{E} \vdash C \equiv_{\mathrm{Type}} A_2\end{array}}{\Gamma;\mathscr{E} \vdash c_1\ c_2 \rightsquigarrow (e_1\ {}^{@(x:A.B)}\ e_2,\ D[e_2/x])}$$

$$\frac{\begin{array}{l}\Gamma;\mathcal{E} \vdash c_1 \rightsquigarrow (e_1,A_1) \\ \Gamma;\mathcal{E} \vdash c_2 \rightsquigarrow (e_2,A_2) \\ \Gamma;\mathcal{E} \vdash A_2 \equiv_{\mathrm{Type}} \mathrm{Type} \\ \Gamma;\mathcal{E} \vdash A_1 \equiv_{\mathrm{Type}} e_2\end{array}}{\Gamma;\mathcal{E} \vdash c_1{::}c_2 \rightsquigarrow (e_1,e_2)}$$

Equality hints

$$\mathcal{E} \;=\; \mathcal{E}_{\equiv},\ \mathcal{E}_{\beta},\ \mathcal{E}_{\eta}$$

general hints

β-hints

η-hints

A hint is a universally quantified equation:

$$\prod x_1 : A_1 \ldots x_n : A_n .\, \mathrm{Eq}_B(e_1, e_2)$$

β-hints

```
pair_fst:
  ∏A,B:Type. ∏x:A. ∏y:B.
    Eq_A(fst A B (pair A B x y),x)
```

η-hints

```
pair_eta:
  ∏A,B:Type. ∏u,v:A×B.
    Eq_A(fst A B u, fst A B v) →
    Eq_B(snd A B u, snd A B v) →
    Eq_A×B(u,v)
```

$$\frac{\begin{array}{l}\Gamma;\mathscr{E}_{\equiv},\mathscr{E}_{\beta},\mathscr{E}_{\eta} \vdash \mathtt{c}_1 \rightsquigarrow (\mathtt{e}_1,\mathtt{A}_1) \\ \Gamma;\mathscr{E}_{\equiv},\mathscr{E}_{\beta},\mathscr{E}_{\eta} \vdash \mathtt{A}_1 \mapsto \prod \mathtt{x}_1{:}\mathtt{A}_1 \ldots \mathtt{x}_n{:}\mathtt{A}_n.\mathtt{Eq}_{\mathtt{B}}(\mathtt{e}_1,\mathtt{e}_2) \\ \Gamma;\mathscr{E}_{\equiv},\mathscr{E}_{\beta} \cup \{\prod \mathtt{x}_1{:}\mathtt{A}_1 \ldots \mathtt{x}_n{:}\mathtt{A}_n.\mathtt{Eq}_{\mathtt{B}}(\mathtt{e}_1,\mathtt{e}_2)\},\mathscr{E}_{\eta} \vdash \mathtt{c}_2 \rightsquigarrow (\mathtt{e}_2,\mathtt{A}_2)\end{array}}{\Gamma;\mathscr{E}_{\equiv},\mathscr{E}_{\beta},\mathscr{E}_{\eta} \vdash \mathtt{beta}\ \mathtt{c}_1\ \mathtt{in}\ \mathtt{c}_2 \rightsquigarrow (\mathtt{e}_2,\mathtt{A}_2)}$$

Checking $e_1 \equiv_A e_2$

1. Decompose $e_1 \equiv_A e_2$ into subgoals that have smaller types, e.g.

   η-rules

   $$e_1 \equiv_{A \times B} e_2$$

   reduces to

   $$\texttt{fst}\ e_1 \equiv_A \texttt{fst}\ e_2 \quad \text{and} \quad \texttt{snd}\ e_1 \equiv_B \texttt{snd}\ e_2$$

2. When the type cannot be decomposed further, check that e_1 and e_2 are equal by normalization.

   β-rules

β-hints as definitions

```
a : A
a_def : Eq(a,e)

beta a_def in …
```

β-hints as definitions

```
a : A
a_def1 : Eq(a,e₁)
a_def2 : Eq(a,e₂)

beta a_def1 in …
beta a_def2 in …
```

What else?

- Voevodsky’s Homotopy Type System
- Enrich the input language with other computational effects and handlers
- Implement standard proof assistant techniques (implicit arguments, proof search, type clases, …) in the language

This material is based upon work supported by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF under Award No. FA9550-14-1-0096.

Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the Air Force Office of Scientific Research, Air Force Materiel Command, USAF.