Normalization by evaluation for $\lambda^{\to 2}$

Thorsten Altenkirch — University of Nottingham

Tarmo Uustalu — Tallinn Technical University

Motivation

Motivation

- Implementations of typed λ-calculi to support **type-directed construction of certified, correct programs**.

Motivation

- Implementations of typed λ-calculi to support **type-directed construction of certified, correct programs**.
- **Normalisation of evaluation** (NbE) used in the actual implementation of recent tools such as **Epigram**.

Motivation

- Implementations of typed λ-calculi to support **type-directed construction of certified, correct programs**.
- **Normalisation of evaluation** (NbE) used in the actual implementation of recent tools such as **Epigram**.
- Offers efficent implementations **and** straightforward correctness arguments.

More motivation

- Goal: make equality more **extensional**. From $=_\beta$ to $=_{\beta\eta}$.

More motivation

- Goal: make equality more **extensional**. From $=_\beta$ to $=_{\beta\eta}$.
- Study simple calculi first - here $\lambda^{\to 2}$ = simple types ($\lambda^{\to}$) + booleans (2).

More motivation

- Goal: make equality more **extensional**. From $=_\beta$ to $=_{\beta\eta}$.
- Study simple calculi first - here $\lambda^{\to 2}$ = simple types ($\lambda^{\to}$) + booleans (2).
- Discuss extensions to more interesting systems.

More motivation

- Goal: make equality more **extensional**. From $=_\beta$ to $=_{\beta\eta}$.
- Study simple calculi first - here $\lambda^{\to 2}$ = simple types ($\lambda^{\to}$) + booleans (2).
- Discuss extensions to more interesting systems.
- Use type-theoretic methodology (on paper).

More motivation

- Goal: make equality more **extensional**. From $=_\beta$ to $=_{\beta\eta}$.
- Study simple calculi first - here $\lambda^{\to 2}$ = simple types ($\lambda^{\to}$) + booleans (2).
- Discuss extensions to more interesting systems.
- Use type-theoretic methodology (on paper).
- Here: Haskell as a poor man's type theory.

More motivation

- Goal: make equality more **extensional**. From $=_\beta$ to $=_{\beta\eta}$.
- Study simple calculi first - here $\lambda^{\to 2}$ = simple types ($\lambda^{\to}$) + booleans (2).
- Discuss extensions to more interesting systems.
- Use type-theoretic methodology (on paper).
- Here: Haskell as a poor man's type theory.
- In future: implementation within epigram.

The simplest typed λ-calculus?

The simplest typed λ-calculus?

- $\lambda^{\rightarrow}$ needs type-variables

The simplest typed λ-calculus?

- $\lambda^{\rightarrow}$ needs type-variables
  not as simple as it looks!

The simplest typed λ-calculus?

- $\lambda^{\to}$ needs type-variables
  not as simple as it looks!
- $\lambda^{\to 0}$,$\lambda^{\to 1}$ without type-variables

The simplest typed λ-calculus?

- $\lambda^{\rightarrow}$ needs type-variables
  not as simple as it looks!
- $\lambda^{\rightarrow 0}$,$\lambda^{\rightarrow 1}$ without type-variables
  are equationally inconsistent.

The simplest typed λ-calculus?

- $\lambda^{\rightarrow}$ needs type-variables
  not as simple as it looks!
- $\lambda^{\rightarrow 0}$,$\lambda^{\rightarrow 1}$ without type-variables
  are equationally inconsistent.
- $\lambda^{\rightarrow 2}$ without type-variables

The simplest typed λ-calculus?

- $\lambda^{\rightarrow}$ needs type-variables
  not as simple as it looks!
- $\lambda^{\rightarrow 0}$,$\lambda^{\rightarrow 1}$ without type-variables
  are equationally inconsistent.
- $\lambda^{\rightarrow 2}$ without type-variables
  the simplest (interesting) typed λ-calculus!

$\lambda^{\rightarrow 2}$ in a nutshell

$\lambda^{\to 2}$ in a nutshell

$$\frac{}{\mathtt{True}, \mathtt{False} : \mathtt{Bool}} \qquad \frac{t : \mathtt{Bool} \quad u_0, u_1 : \sigma}{\mathtt{If}\, t\, u_0\, u_1 : \sigma}$$

$\lambda^{\rightarrow 2}$ in a nutshell

$$\frac{}{\texttt{True}, \texttt{False} : \texttt{Bool}} \qquad \frac{t : \texttt{Bool} \quad u_0, u_1 : \sigma}{\texttt{If}\, t\, u_0\, u_1 : \sigma}$$

$$\begin{array}{llll}
\texttt{If}\,\texttt{True}\, u_0\, u_1 & =_{\beta\eta} & u_0 & (\beta) \\
\texttt{If}\,\texttt{False}\, u_0\, u_1 & =_{\beta\eta} & u_1 & (\beta) \\
\texttt{If}\, t\,\texttt{True}\,\texttt{False} & =_{\beta\eta} & t & (\eta) \\
f\,(\texttt{If}\, t\, u_0\, u_1) & =_{\beta\eta} & \texttt{If}\, t\,(f\, u_0)\,(f\, u_1) & (\xi)
\end{array}$$

$\lambda^{\to 2}$ in a nutshell

$$\frac{}{\texttt{True}, \texttt{False} : \texttt{Bool}} \qquad \frac{t : \texttt{Bool} \quad u_0, u_1 : \sigma}{\texttt{If}\, t\, u_0\, u_1 : \sigma}$$

$$\begin{array}{lcll}
\texttt{If}\,\texttt{True}\,u_0\,u_1 & =_{\beta\eta} & u_0 & (\beta) \\
\texttt{If}\,\texttt{False}\,u_0\,u_1 & =_{\beta\eta} & u_1 & (\beta) \\
\texttt{If}\,t\,\texttt{True}\,\texttt{False} & =_{\beta\eta} & t & (\eta) \\
f\,(\texttt{If}\,t\,u_0\,u_1) & =_{\beta\eta} & \texttt{If}\,t\,(f\,u_0)\,(f\,u_1) & (\xi)
\end{array}$$

Categorically:

$$\mathrm{Hom}(\Gamma \times \mathrm{Bool}, \sigma) \simeq \mathrm{Hom}(\Gamma, \sigma) \times \mathrm{Hom}(\Gamma, \sigma)$$

Example

$$\begin{aligned}
\texttt{once} &= \lambda f^{\texttt{Bool}\to\texttt{Bool}} \lambda x^{\texttt{Bool}} f\, x \\
\texttt{twice} &= \lambda f^{\texttt{Bool}\to\texttt{Bool}} \lambda x^{\texttt{Bool}} f\, (f\, x) \\
\texttt{thrice} &= \lambda f^{\texttt{Bool}\to\texttt{Bool}} \lambda x^{\texttt{Bool}} f\, (f\, (f\, x))
\end{aligned}$$

$$\texttt{once}, \texttt{twice}, \texttt{thrice} : (\texttt{Bool} \to \texttt{Bool}) \to (\texttt{Bool} \to \texttt{Bool})$$

Example

$$\begin{aligned}\texttt{once} &= \lambda f^{\texttt{Bool}\to\texttt{Bool}} \lambda x^{\texttt{Bool}} f\ x \\ \texttt{twice} &= \lambda f^{\texttt{Bool}\to\texttt{Bool}} \lambda x^{\texttt{Bool}} f\ (f\ x) \\ \texttt{thrice} &= \lambda f^{\texttt{Bool}\to\texttt{Bool}} \lambda x^{\texttt{Bool}} f\ (f\ (f\ x))\end{aligned}$$

$\texttt{once}, \texttt{twice}, \texttt{thrice} : (\texttt{Bool} \to \texttt{Bool}) \to (\texttt{Bool} \to \texttt{Bool})$

$\texttt{once} \neq_{\beta\eta} \texttt{twice}$

Example

$$
\begin{aligned}
\texttt{once} &= \lambda f^{\texttt{Bool}\to\texttt{Bool}}\lambda x^{\texttt{Bool}} f\, x \\
\texttt{twice} &= \lambda f^{\texttt{Bool}\to\texttt{Bool}}\lambda x^{\texttt{Bool}} f\,(f\, x) \\
\texttt{thrice} &= \lambda f^{\texttt{Bool}\to\texttt{Bool}}\lambda x^{\texttt{Bool}} f\,(f\,(f\, x))
\end{aligned}
$$

$\texttt{once}, \texttt{twice}, \texttt{thrice} : (\texttt{Bool} \to \texttt{Bool}) \to (\texttt{Bool} \to \texttt{Bool})$

$\texttt{once} \neq_{\beta\eta} \texttt{twice}$

$\texttt{once} =_{\beta\eta} \texttt{thrice}$

Why ?

Why ?

$$
\begin{aligned}
f\,(f\,(f\,\mathtt{True})) &=_{\beta\eta} \mathtt{If}\,(f\,\mathtt{True})\,(f\,(f\,\mathtt{True}))\,(f\,(f\,\mathtt{False}))\\
&=_{\beta\eta} \mathtt{If}\,(f\,\mathtt{True})\,\mathtt{True}\,(f\,(f\,\mathtt{False}))\\
&=_{\beta\eta} \mathtt{If}\,(f\,\mathtt{True})\,\mathtt{True}\,(\mathtt{If}\,(f\,\mathtt{False})\,(f\,\mathtt{True})\,(f\,\mathtt{False}))\\
&=_{\beta\eta} \mathtt{If}\,(f\,\mathtt{True})\,\mathtt{True}\,(\mathtt{If}\,(f\,\mathtt{False})\,\mathtt{False}\,\mathtt{False})\\
&=_{\beta\eta} \mathtt{If}\,(f\,\mathtt{True})\,\mathtt{True}\,\mathtt{False}\\
&=_{\beta\eta} f\,\mathtt{True}
\end{aligned}
$$

Symmetrically, we can show that $f\,(f\,(f\,\mathtt{False})) =_{\beta\eta} f\,\mathtt{False}$, and hence

$$
\begin{aligned}
&\mathtt{thrice}\\
&= \lambda f^{\mathtt{Bool}\to\mathtt{Bool}}\lambda x^{\mathtt{Bool}} f\,(f\,(f\,x))\\
&=_{\beta\eta} \lambda f^{\mathtt{Bool}\to\mathtt{Bool}}\lambda x^{\mathtt{Bool}} \mathtt{If}\,x\,(f\,(f\,(f\,\mathtt{True})))\,(f\,(f\,(f\,\mathtt{False})))\\
&=_{\beta\eta} \lambda f^{\mathtt{Bool}\to\mathtt{Bool}}\lambda x^{\mathtt{Bool}} \mathtt{If}\,x\,(f\,\mathtt{True})\,(f\,\mathtt{False})\\
&=_{\beta\eta} \lambda f^{\mathtt{Bool}\to\mathtt{Bool}}\lambda x^{\mathtt{Bool}} f\,x\\
&= \mathtt{once}
\end{aligned}
$$

A simpler proof ?

A simpler proof ?

$$\mathrm{Bool} = \{\mathrm{true}, \mathrm{false}\}$$

A simpler proof ?

$$\mathrm{Bool} = \{\mathrm{true}, \mathrm{false}\}$$

$$\mathrm{Bool} \to \mathrm{Bool} = \\ \{x \mapsto \mathrm{true}, x \mapsto x, x \mapsto \neg x, x \mapsto \mathrm{false}\}$$

A simpler proof ?

$$\mathrm{Bool} = \{\mathrm{true}, \mathrm{false}\}$$

$$\begin{aligned}&\mathrm{Bool} \to \mathrm{Bool} = \\ &\quad \{x \mapsto \mathrm{true}, x \mapsto x, x \mapsto \neg x, x \mapsto \mathrm{false}\}\end{aligned}$$

$$\forall f \in \mathrm{Bool} \to \mathrm{Bool}. f^3 = f$$

A simpler proof ?

$$\mathrm{Bool} = \{\mathrm{true}, \mathrm{false}\}$$

$$\begin{aligned}&\mathrm{Bool} \to \mathrm{Bool} = \\ &\quad \{x \mapsto \mathrm{true}, x \mapsto x, x \mapsto \neg x, x \mapsto \mathrm{false}\}\end{aligned}$$

$$\forall f \in \mathrm{Bool} \to \mathrm{Bool}. f^3 = f$$

Why does this hold for $=_{\beta\eta}$?

A simpler proof ?

$$\mathrm{Bool} = \{\mathrm{true}, \mathrm{false}\}$$

$$\begin{aligned}&\mathrm{Bool} \to \mathrm{Bool} = \\ &\quad \{x \mapsto \mathrm{true}, x \mapsto x, x \mapsto \neg x, x \mapsto \mathrm{false}\}\end{aligned}$$

$$\forall f \in \mathrm{Bool} \to \mathrm{Bool}. f^3 = f$$

Why does this hold for $=_{\beta\eta}$?
Corollary of our NbE construction.

Another *corollary*: normalisation

```
Main> once
Lam (Bool :-> Bool) "f" (Lam Bool "x" (App (Var "f") (Var "x")))
Main> :t nf
nf :: Ty -> Tm -> Tm
Main> :t nf'
nf' :: Tm -> Maybe (Ty,Tm)
Main> nf' once
Just ((Bool :-> Bool) :-> (Bool :-> Bool),Lam (Bool :-> Bool) "x"
(If (App (Var "x") TTrue) (If (App (Var "x") TFalse) (Lam Bool "x"
 TTrue) (Lam Bool "x" (Var "x"))) (If (App (Var "x") TFalse) (Lam
Bool "x" (If (Var "x") TFalse TTrue)) (Lam Bool "x" TFalse))))
Main> nf' thrice
Just ((Bool :-> Bool) :-> (Bool :-> Bool),Lam (Bool :-> Bool) "x"
 (If (App (Var "x") TTrue) (If (App (Var "x") TFalse) (Lam Bool "x"
 TTrue) (Lam Bool "x" (Var "x"))) (If (App (Var "x") TFalse) (Lam
 Bool "x" (If (Var "x") TFalse TTrue)) (Lam Bool "x" TFalse))))
```

NbE : the basic idea

NbE : the basic idea

1. Define a semantic interpretation

$$\frac{t : \sigma}{[\![t]\!] \in [\![\sigma]\!]} \qquad \frac{t =_{\beta\eta} u}{[\![t]\!] = [\![u]\!]}$$

NbE : the basic idea

1. Define a semantic interpretation

$$\frac{t : \sigma}{[\![t]\!] \in [\![\sigma]\!]} \qquad \frac{t =_{\beta\eta} u}{[\![t]\!] = [\![u]\!]}$$

2. Invert evaluation, i.e. define

$$\mathsf{quote}^{\sigma} \in [\![\sigma]\!] \to \mathsf{Tm}\,\sigma \qquad \mathsf{quote}^{\sigma}\,[\![t]\!] =_{\beta\eta} t$$

NbE : the basic idea

1. Define a semantic interpretation

$$\frac{t : \sigma}{[\![t]\!] \in [\![\sigma]\!]} \qquad \frac{t =_{\beta\eta} u}{[\![t]\!] = [\![u]\!]}$$

2. Invert evaluation, i.e. define

$$\mathsf{quote}^{\sigma} \in [\![\sigma]\!] \to \mathsf{Tm}\,\sigma \qquad \mathsf{quote}^{\sigma}\,[\![t]\!] =_{\beta\eta} t$$

3. Now define

$$\mathsf{nf}^{\sigma}\,t = \mathsf{quote}^{\sigma}\,[\![t]\!]$$

nf

$$\frac{}{\mathsf{nf}\ t =_{\beta\eta} t} \qquad \frac{t =_{\beta\eta} u}{\mathsf{nf}\ t = \mathsf{nf}\ u}$$

nf

$$\frac{}{\mathsf{nf}\ t =_{\beta\eta} t} \qquad \frac{t =_{\beta\eta} u}{\mathsf{nf}\ t = \mathsf{nf}\ u}$$

nf is effective because our development takes place in a constructive set theory (ala Martin-Löf).

nf

$$\frac{}{\mathsf{nf}\, t =_{\beta\eta} t} \qquad \frac{t =_{\beta\eta} u}{\mathsf{nf}\, t = \mathsf{nf}\, u}$$

nf is effective because our development takes place in a constructive set theory (ala Martin-Löf).
The effectiveness of nf is witnessed by an implementation in Haskell.

Implementation in Haskell

Haskell-types can only approximate the intended types, e.g.

$$\mathsf{nf} \in \Pi_{\sigma \in \mathsf{Ty}} \mathsf{Tm}\,\sigma \to \mathsf{Tm}\,\sigma$$

is implemented as

```
nf :: Ty -> Tm -> Tm
```

The semantics

$$
\begin{aligned}
[\![\texttt{Bool}]\!] &= \mathrm{Bool} \\
&= \{\mathrm{true}, \mathrm{false}\} \\
[\![\sigma \to \tau]\!] &= [\![\sigma]\!] \to [\![\tau]\!]
\end{aligned}
$$

Implementation in Haskell

```
data El = STrue | SFalse | SLam Ty (El -> El)
```

Decision trees

Decision trees

- We use decision trees to enumerate types.

$$\frac{\sigma \in \mathsf{Ty}}{\mathsf{Tree}\,\sigma \in \star}\ \text{where}\ \frac{x \in [\![\sigma]\!]}{\mathsf{Val}\,x \in \mathsf{Tree}\,\sigma} \quad \frac{l, r \in \mathsf{Tree}\,\sigma}{\mathsf{Choice}\,l\,r \in \mathsf{Tree}\,\sigma}$$

Decision trees

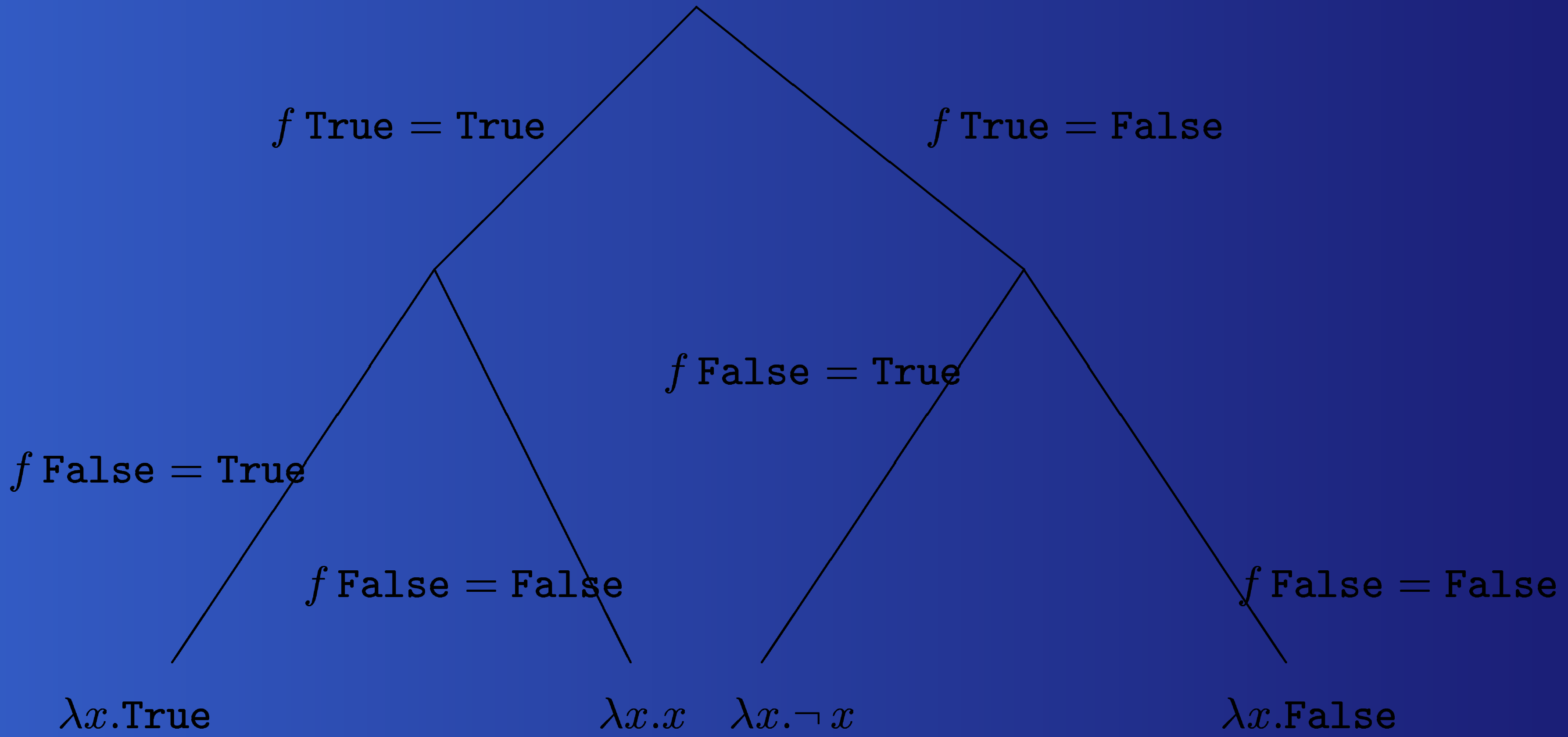
- We use decision trees to enumerate types.

$$\frac{\sigma \in \mathsf{Ty}}{\mathsf{Tree}\,\sigma \in \star}\ \text{where}\ \frac{x \in [\![\sigma]\!]}{\mathsf{Val}\,x \in \mathsf{Tree}\,\sigma} \quad \frac{l, r \in \mathsf{Tree}\,\sigma}{\mathsf{Choice}\,l\,r \in \mathsf{Tree}\,\sigma}$$

- We define by simultanous recursion over $\sigma \in \mathsf{Ty}$

$$\begin{array}{rcl} \mathsf{enum}\,\sigma & \in & \mathsf{Tree}\,\sigma \\ \mathsf{questions}\,\sigma & \in & [\![\sigma]\!] \to [\![\mathtt{Bool}]\!] \end{array}$$

Decision tree for $\texttt{Bool} \to \texttt{Bool}$



find

- We also implement

$$\frac{as \in [\llbracket \mathtt{Bool} \rrbracket] \qquad ts \in \mathsf{Tree}\,\llbracket\sigma\rrbracket \qquad as \diamond ts}{\mathsf{find}\, as\; ts \in \llbracket\sigma\rrbracket}$$

where $as \diamond ts$ expresses that the length of as matches the depth of ts.

Implementing quote

$$\frac{x \in [\![\sigma]\!]}{\mathsf{quote}^{\sigma} \in \mathsf{Tm}\,\sigma}$$

Implementing quote

$$\frac{x \in [\![\sigma]\!]}{\mathsf{quote}^{\sigma} \in \mathsf{Tm}\,\sigma}$$

$$\begin{aligned}
\mathsf{quote}^{\texttt{Bool}}\,\mathrm{true} &= \texttt{True} \\
\mathsf{quote}^{\texttt{Bool}}\,\mathrm{false} &= \texttt{False}
\end{aligned}$$

Implementing quote

$$\frac{x \in [\![\sigma]\!]}{\mathsf{quote}^{\sigma} \in \mathsf{Tm}\,\sigma}$$

$$\begin{aligned}
\mathsf{quote}^{\mathtt{Bool}}\,\mathrm{true} &= \mathtt{True} \\
\mathsf{quote}^{\mathtt{Bool}}\,\mathrm{false} &= \mathtt{False}
\end{aligned}$$

$$\mathsf{quote}^{\sigma \to \tau}\, f \;=$$

Implementing quote

$$\frac{x \in [\![\sigma]\!]}{\mathsf{quote}^{\sigma} \in \mathsf{Tm}\,\sigma}$$

$$\begin{aligned}
\mathsf{quote}^{\mathtt{Bool}}\,\mathrm{true} &= \mathtt{True} \\
\mathsf{quote}^{\mathtt{Bool}}\,\mathrm{false} &= \mathtt{False}
\end{aligned}$$

$$\begin{aligned}
\mathsf{quote}^{\sigma\to\tau}\,f &= \lambda\mathsf{x}^{\sigma}.\mathsf{find}_{\mathsf{syn}}\,[q\,\mathtt{x} \mid q \leftarrow \mathsf{questions}_{\mathsf{syn}}\,\sigma] \\
&\qquad (\mathsf{fmap}\,((\mathsf{quote}\,\tau).f)\,(\mathsf{enum}_{\mathsf{set}}\,\sigma))
\end{aligned}$$

Implementing quote

$$\frac{x \in [\![\sigma]\!]}{\mathsf{quote}^{\sigma} \in \mathsf{Tm}\,\sigma}$$

$$\begin{aligned}
\mathsf{quote}^{\texttt{Bool}}\ \mathrm{true} &= \texttt{True} \\
\mathsf{quote}^{\texttt{Bool}}\ \mathrm{false} &= \texttt{False}
\end{aligned}$$

$$\begin{aligned}
\mathsf{quote}^{\sigma\to\tau}\ f &= \lambda \mathsf{x}^{\sigma}.\mathsf{find}_{\mathsf{syn}}\,[q\,\mathbf{x} \mid q \leftarrow \mathsf{questions}_{\mathsf{syn}}\,\sigma] \\
&\qquad (\mathsf{fmap}\,((\mathsf{quote}\,\tau).f)\,(\mathsf{enum}_{\mathsf{set}}\,\sigma))
\end{aligned}$$

Note that we need only one bound variable!

Correctness of quote

How do we show

$$\frac{t : \sigma}{\mathsf{quote}^{\sigma}\,[\![t]\!] =_{\beta\eta} t}?$$

Logical relations

Logical relations

$$\frac{\sigma \in \mathsf{Ty}}{\mathsf{R}^{\sigma} \subseteq \mathsf{Tm}\ \sigma \times [\![\sigma]\!]_{\mathsf{set}}}$$

Logical relations

$$\frac{\sigma \in \mathsf{Ty}}{\mathsf{R}^{\sigma} \subseteq \mathsf{Tm}\ \sigma \times [\![\sigma]\!]_{\mathsf{set}}}$$

$$\begin{aligned}
t\mathsf{R}^{\mathtt{Bool}}\mathrm{true} &= t =_{\beta\eta} \mathtt{True} \\
t\mathsf{R}^{\mathtt{Bool}}\mathrm{false} &= t =_{\beta\eta} \mathtt{False} \\
t\mathsf{R}^{\sigma:\to\tau} f &= \forall u \mathsf{R}^{\sigma} d \to \mathtt{App}\ t\ u \mathsf{R}^{\tau} f\ d
\end{aligned}$$

Logical relations

$$\frac{\sigma \in \mathsf{Ty}}{\mathsf{R}^\sigma \subseteq \mathsf{Tm}\ \sigma \times [\![\sigma]\!]_{\mathsf{set}}}$$

$$\begin{aligned}
t\mathsf{R}^{\mathtt{Bool}}\mathrm{true} &= t =_{\beta\eta} \mathtt{True} \\
t\mathsf{R}^{\mathtt{Bool}}\mathrm{false} &= t =_{\beta\eta} \mathtt{False} \\
t\mathsf{R}^{\sigma:\to\tau} f &= \forall u \mathsf{R}^\sigma d \to \mathtt{App}\ t\ u \mathsf{R}^\tau f\ d
\end{aligned}$$

Fundamental theorem: $\dfrac{t : \sigma}{t R^\sigma [\![t]\!]}$

Logical relations

$$\frac{\sigma \in \mathsf{Ty}}{\mathsf{R}^\sigma \subseteq \mathsf{Tm}\,\sigma \times [\![\sigma]\!]_{\mathsf{set}}}$$

$$\begin{aligned}
t\mathsf{R}^{\mathtt{Bool}}\mathrm{true} &= t =_{\beta\eta} \mathtt{True} \\
t\mathsf{R}^{\mathtt{Bool}}\mathrm{false} &= t =_{\beta\eta} \mathtt{False} \\
t\mathsf{R}^{\sigma:\to\tau} f &= \forall u \mathsf{R}^\sigma d \to \mathtt{App}\; t\; u \mathsf{R}^\tau f\; d
\end{aligned}$$

Fundamental theorem: $\dfrac{t : \sigma}{tR^\sigma [\![t]\!]}$

Main lemma: $\dfrac{tR^\sigma x}{t =_{\beta\eta} \mathsf{quote}\, x}$

Further work

Further work

- Extend the construction to $\lambda^{\rightarrow 01+\times}$ (*almost done*).

Further work

- Extend the construction to $\lambda^{\to 01+\times}$ (*almost done*).
- Extend the construction to $\lambda^{\Pi\Sigma 012}$ (finite Type Theory)
  Useful as a hardware description language

Further work

- Extend the construction to $\lambda^{\to 01+\times}$ (*almost done*).
- Extend the construction to $\lambda^{\Pi\Sigma 012}$ (finite Type Theory)
  Useful as a hardware description language
- Use BDDs instead of Decision Trees to improve efficiency.

Further work

- Extend the construction to $\lambda^{\to 01+\times}$ (*almost done*).
- Extend the construction to $\lambda^{\Pi\Sigma 012}$ (finite Type Theory)
  Useful as a hardware description language
- Use BDDs instead of Decision Trees to improve efficiency.
- Can we extend this approach to type variables?

Related Work

- Neil Ghani
  Adjoint Rewriting
  PhD, 1995
- Thorsten Altenkirch,Peter Dybjer,
  Martin Hofmann,Phil Scott
  Normalization by evaluation for typed lambda calculus with coproducts
  LICS 2001
- Vincent Balat
  Une étude des sommes fortes : isomorphismes et formes normales
  PhD thesis, 2002