Normalization by evaluation, algebraic theories, computational effects

Danel Ahman
Hughes Hall
University of Cambridge
An impure higher-order program

```haskell
function foo f =
  let x = read() in
    let y = f (proj₀ (x , one)) in
      let z = return y in
        write zero ; return z
```
An impure higher-order program

```haskell
function foo f =
  let x = read() in
  let y = f (proj₀ (x , one)) in
  let z = return y in
  write zero ; return z
```

- Reading from a memory cell
- Writing to a memory cell
An **impure** higher-order program

```plaintext
function foo f =
    let x = read() in
    let y = f (proj₀ (x, one)) in
    let z = return y in
    write zero ; return z
```

higher-order argument

reading from a memory cell

writing to a memory cell
How to reason about these effects?
Computational effects

• Examples: global state, input/output, choice, ...
Computational effects

• Examples: global state, input/output, choice, ...

• We model them using algebraic theories $T = (\Sigma, E)$

  operations $\Sigma$ + equations $E$

*Plotkin, Power '02*
Computational effects

• Examples: global state, input/output, choice, ...

• We model them using algebraic theories $T = (\Sigma, E)$

  operations $\Sigma$
  
  read $x \; y$
  write$_{\text{zero}}$ $x$
  write$_{\text{one}}$ $x$

  + equations $E$
  
  write$_{\text{zero}}$ (write$_{\text{one}}$ $x$) $\equiv$ write$_{\text{one}}$ $x$
  write$_{\text{one}}$ (write$_{\text{zero}}$ $x$) $\equiv$ write$_{\text{zero}}$ $x$
  write$_{\text{zero}}$ (read $x \; y$) $\equiv$ write$_{\text{zero}}$ $x$
  ...

Plotkin, Power '02
How to reason about impure programs based on these algebraic effect theories?
A fine-grain call-by-value intermediate language

Levy, Power, Thielecke '03

- **Type signature**
  \[ \sigma ::= \alpha | 1 | \sigma \times \sigma | \sigma \rightarrow \sigma | ... \]

- **Value terms**
  \[
  \begin{align*}
  \Gamma, x : \sigma, \Gamma' \vdash _v x : \sigma & \quad & \Gamma \vdash _v V_1 : \sigma_1 \quad \Gamma \vdash _v V_2 : \sigma_2 \quad & \Gamma \vdash _v V : \sigma_1 \times \sigma_2 \quad & \Gamma \vdash _v \pi_i(V) : \sigma_i \\
  \Gamma \vdash _v \langle V_1, V_2 \rangle : \sigma_1 \times \sigma_2 & \quad & \Gamma \vdash _v \lambda x : \sigma.N : \sigma \rightarrow \tau \\
  \Gamma \vdash _v \lambda x : \sigma.N : \sigma \rightarrow \tau & \quad & \Gamma, x : \sigma \vdash _p N : \tau \\
  \end{align*}
  \]

- **Producer terms**
  \[
  \begin{align*}
  \Gamma \vdash _p M : \sigma \quad \Gamma, x : \sigma \vdash _p N : \tau & \quad & \Gamma \vdash _v V : \sigma \\
  \Gamma \vdash _p M \text{ to } x.N : \tau & \quad & \Gamma \vdash _p \text{ return } V : \sigma \\
  \Gamma \vdash _v V : \sigma \rightarrow \tau \quad \Gamma \vdash _v W : \sigma & \quad & \Gamma \vdash _p VW : \tau \\
  \end{align*}
  \]
Extending algebraic theories to the intermediate language

- Every operation in $\Sigma$ defines a producer term

\[
\frac{\Gamma \vdash p \ M_0 : \sigma \quad \Gamma \vdash p \ M_1 : \sigma}{\Gamma \vdash p \ \text{read}_\sigma(M_0, M_1) : \sigma}
\]

\[
\frac{\Gamma \vdash p \ M : \sigma}{\Gamma \vdash p \ \text{write}_{\text{zero}}\sigma(M) : \sigma}
\]

\[
\frac{\Gamma \vdash p \ M : \sigma}{\Gamma \vdash p \ \text{write}_{\text{one}}\sigma(M) : \sigma}
\]

- Extend the usual beta-eta equations with all the equations in $E$

\[
\frac{\Gamma, x : \sigma \vdash p \ M : \tau \quad \Gamma \vdash v \ V : \sigma}{\Gamma \vdash p \ (\lambda x : \sigma. M)V \equiv M[V/x] : \tau}
\]

\[
\frac{\Gamma \vdash v \ V \sigma \rightarrow \tau}{\Gamma \vdash v \ V \equiv \lambda x : \sigma.(Vx) : \sigma \rightarrow \tau}
\]

\[
\frac{\Gamma \vdash p \ M : \sigma}{\Gamma \vdash p \ \text{write}_{\text{zero}}\sigma \ (\text{write}_{\text{one}}\sigma \ M) \equiv \text{write}_{\text{one}}\sigma \ M : \sigma}
\]
Extending algebraic theories to the intermediate language

• Every operation in \( \Sigma \) defines a producer term

\[
\begin{align*}
\Gamma \vdash_p M_0 : \sigma & \quad \Gamma \vdash_p M_1 : \sigma \\
\Gamma \vdash_p \text{read}_\sigma(M_0, M_1) : \sigma
\end{align*}
\]

\[
\begin{align*}
\Gamma \vdash_p M : \sigma \\
\Gamma \vdash_p \text{write}_{(\text{zero})\sigma}(M) : \sigma
\end{align*}
\]

\[
\begin{align*}
\Gamma \vdash_p M : \sigma \\
\Gamma \vdash_p \text{write}_{(\text{one})\sigma}(M) : \sigma
\end{align*}
\]

• Extend the usual beta-eta equations

\[
\begin{align*}
\Gamma, x : \sigma \vdash_p M : \tau & \quad \Gamma \vdash_v V : \sigma \\
\Gamma \vdash_p (\lambda x : \sigma. M)V \equiv M[V/x] : \tau
\end{align*}
\]

\[
\begin{align*}
\Gamma \vdash_v V \sigma \rightarrow \tau & \\
\Gamma \vdash_v V \equiv \lambda x : \sigma.(V x) : \sigma \rightarrow \tau
\end{align*}
\]

with all the equations in \( E \)

\[
\begin{align*}
\Gamma \vdash_p M : \sigma \\
\Gamma \vdash_p \text{write}_{(\text{zero})\sigma}(\text{write}_{(\text{one})\sigma}(M)) \equiv \text{write}_{(\text{one})\sigma} M : \sigma
\end{align*}
\]
Extending algebraic theories to the intermediate language

- Every operation in \( \Sigma \) defines a producer term

  \[
  \frac{\Gamma \vdash_p M_0 : \sigma \quad \Gamma \vdash_p M_1 : \sigma}{\Gamma \vdash_p \text{read}_\sigma(M_0, M_1) : \sigma}
  \]

  \[
  \frac{\Gamma \vdash_p M : \sigma}{\Gamma \vdash_p \text{write}_{\text{zero}}\sigma(M) : \sigma}
  \]

  \[
  \frac{\Gamma \vdash_p M : \sigma}{\Gamma \vdash_p \text{write}_{\text{one}}\sigma(M) : \sigma}
  \]

- Extend the usual beta-eta equations

  \[
  \frac{\Gamma, x : \sigma \vdash_p M : \tau \quad \Gamma \vdash_v V : \sigma}{\Gamma \vdash_p (\lambda x : \sigma.M)V \equiv M[V/x] : \tau}
  \]

  \[
  \frac{\Gamma \vdash_v V \rightarrow \tau}{\Gamma \vdash_v V \equiv \lambda x : \sigma.(V x) : \sigma \rightarrow \tau}
  \]

  with all the equations in \( E \)

  \[
  \frac{\Gamma \vdash_p M : \sigma}{\Gamma \vdash_p \text{write}_{\text{zero}}\sigma(\text{write}_{\text{one}}\sigma M) \equiv \text{write}_{\text{one}}\sigma M : \sigma}
  \]
Is this representation of algebraic theories correct?
Theorem:
Given two terms in the algebraic effect theory,
they are provably equal in the algebraic theory
iff
they are provably equal in the extended language
Theorem:
Given two terms in the algebraic effect theory,
they are provably equal in the algebraic theory

✓

iff

✓

they are provably equal in the extended language
Theorem:
Given two terms in the algebraic effect theory,
they are provably equal in the algebraic theory
iff
they are provably equal in the extended language

✓

Tricky!
Provable equality

function foo f =
  let x = read() in
  let y = f (proj₀ (x, one)) in
  let z = return y in
  write zero ; return z

is provably equal to

function foo f =
  let x = read() in
  let z = f x in
  write zero ; return z
How to decide provable equality?
Normalization

• So we want to decide when terms are provably equal.

• We do this by computing their normal forms.

Theorem:
Given two provably equal terms in the language, they have canonical normal forms.
Normalization by evaluation

- A semantic notion of normalization
  - Berger & Schwichtenberg '91, Filinski '01, Fiore et. al. '02, Abel et. al. '07

- We define an inverse of interpretation called reification

\[
\text{nf} = \text{reify} \circ \text{interpret}
\]
Normalization by evaluation

- A semantic notion of normalization
  - Berger & Schwichtenberg '91, Filinski '01, Fiore et. al. '02, Abel et. al. '07

- We define an inverse of interpretation called reification

\[ \text{nf} = \text{reify} \circ \text{interpret} \]

**Diagram:**
- Terms → Interpreted terms → Denotational semantics → Normal forms
- Kripke logical relations

**Presheaf model with a strong residual monad**
Why a residualizing interpretation?

- We need to preserve the order of (possible) effects!

```plaintext
function foo f =
  let x = read() in
  let y = f (proj0 (x , one)) in
  let z = return y in
  write zero ; return z
```

1. the reading effect
2. possible unknown effects
3. the writing effect
Why a residualizing interpretation?

- We need to preserve the order of (possible) effects!

```haskell
function foo f =
  let x = read() in
  let y = f (proj₀ (x , one)) in
  let z = return y in
  write zero ; return z
```

1. the reading effect
2. possible unknown effects
3. the writing effect
The main normalization results
Provably equal normal forms

\[ \text{nf} = \text{reify} \circ \text{interpret} \]

**Theorem:**
Given a term \( t \) in the language,

\[ \text{nf} \, t \text{ is provably equal to } t \text{ in the language} \]
Canonical normal forms

\[ \text{nf} = \text{reify} \circ \text{interpret} \]

**Theorem:**
Given two provably equal terms \( t \) and \( u \) in the language, \( \text{nf} \, t \) and \( \text{nf} \, u \) are equivalent up to the algebraic theory (equal if \( E \) is empty)
This representation is correct!

**Theorem:**
Given two terms in the algebraic effect theory,

they are provably equal in the algebraic theory

iff

they are provably equal in the extended language
Conclusions and future work

• We have justified the correctness of extending algebraic theories to a call-by-value intermediate language

• The normalization algorithm and proofs have been rigorously formalized in Agda
  • \( \approx 6000 \) lines of formal proofs

• Future investigations
  • sum types and natural numbers
  • parametrized and second-order algebraic theories