How **not** to prove Strong Normalisation

based on joint work with James Chapman

Thorsten Altenkirch

School of Computer Science and IT
University of Nottingham

April 11, 2007

Long time ago . . .

1993 *A formalization of the strong normalization proof for System F in LEGO*
Typed Lambda Calculi and Applications (TLCA)

1993 *Constructions, Inductive Types and Strong Normalization*
PhD thesis, University of Edinburgh

1994 *Proving Strong Normalization of CC by Modifying Realizability Semantics*
Types for Proofs and Programs (TYPES), 1994

Strong Normalisation ?

- A reduction relation $\triangleright \subseteq \mathrm{Tm} \times \mathrm{Tm}$ is *strongly normalizing*, if all sequences $t_0 \triangleright t_1 \triangleright t_2 \triangleright \ldots$ are finite.
- If $\triangleright$ is strongly normalizing and confluent, then the associated equivalence relation relation $\simeq \subseteq \mathrm{Tm} \times \mathrm{Tm}$ is decidable.
- Example: β-reduction, the congruence closure of

  $$(\lambda x.t)u \triangleright t[x = u]$$

  is strongly normalizing on terms typable in the simply typed λ calculus. (Tait 1967).
- The same is true for terms typable in System F proven by Girard, 1972 using *candidates of reducibility*.
- See *Proofs and Types*, 1989 by Girard, Taylor and Lafont.

Questions

- How to deal with η-expansion?

$$t \rhd \lambda x.t\,x$$

- How to deal with stronger theories?
  E.g. strong products or coproducts?
  Dependent types . . .
- How to combine with substitution?
  E.g. λ^σ + β-reduction is not strongly normalizing
  Mellies, 1995
- Is there a better way to tell the story?
- And who would implement normalisation like this?

```
while redex left do
  replace redex by reduct
```

Normalisation API

$$\mathrm{Nf}\,\sigma \subseteq \mathrm{Tm}\,\sigma$$

$$\frac{t \in \mathrm{Tm}\,\sigma}{\mathrm{nf}\,t \in \mathrm{Nf}\,\sigma}$$

$$\frac{t \simeq u}{\mathrm{nf}\,t = \mathrm{nf}\,u} \qquad \frac{}{t \simeq \mathrm{nf}\,t}$$

Implementations

- Strong normalisation.
- Normalisation by evaluation (NbE).
  Berger and Schwichtenberg, 1991
- Big step normalisation (BSN).

The simply typed λ calculus

$$\frac{}{\mathrm{v}_0 \in \mathrm{Tm}\,\Gamma.\sigma\,\sigma} \qquad \frac{t \in \mathrm{Tm}\,\Gamma\,\sigma}{t^{+\tau} \in \mathrm{Tm}\,\Gamma.\tau\,\sigma} \qquad \frac{t \in \mathrm{Tm}\,\Gamma.\sigma\,\tau \quad u \in \mathrm{Tm}\,\Gamma\,\sigma}{t[u] \in \mathrm{Tm}\,\Gamma\,\tau}$$

$$\frac{t \in \mathrm{Tm}\,\Gamma.\sigma\,\tau}{\lambda^\sigma t \in \mathrm{Tm}\,\Gamma\,\sigma \to \tau} \qquad \frac{t \in \mathrm{Tm}\,\Gamma\,\sigma \to \tau \quad u \in \mathrm{Tm}\,\Gamma\,\sigma}{t\,u \in \mathrm{Tm}\,\Gamma\,\tau}$$

Families of congruences $\simeq_{\mathrm{w}}, \simeq_\beta, \simeq_{\beta\eta} \subseteq (\mathrm{Tm}\,\Gamma\,\sigma)^2$:

$\simeq_{\mathrm{w}}$ weak equality, closed under

$(\lambda^\sigma t)u \simeq_{\mathrm{w}} t[u] \quad (\beta)$ but not under $\dfrac{t \simeq u}{\lambda^\sigma t \simeq \lambda^\sigma u} \quad (\xi)$.

$\simeq_\beta$ closed under (β) and (ξ).

$\simeq_{\beta\eta}$ closed under $(\beta), (\xi)$ and
$\lambda^\sigma(t^{+\sigma}\,\mathrm{v}_0) \simeq_{\beta\eta} t \quad (\eta)$

Big step normalisation

- Implement an evaluator:

$$\frac{t \in \mathrm{Tm}\,\Gamma\,\sigma \qquad \vec{v} \in \mathrm{Env}\,\Delta\,\Gamma}{\mathrm{eval}\,t\,\vec{v} \in \mathrm{Val}\,\Delta\,\sigma}$$

  using an environment machine.
- We define a function

$$\frac{v \in \mathrm{Val}\,\Gamma\,\sigma}{\mathrm{quote}^{\mathrm{w}}\,v \in \mathrm{Nf}\,\Gamma\,\sigma}$$

- We show (using Tait's method) that for all $t \in \mathrm{Tm}\,\Gamma\,\sigma$
  1. $\mathrm{eval}\,t\,\vec{v}$ terminates returning v.
  2. and $\mathrm{quote}^{\mathrm{w}}\,v \simeq_{\mathrm{w}} t$

Big step normalisation ...

- To reflect $\simeq_\beta$ and $\simeq_{\beta\eta}$ we define quote^β and $\mathrm{quote}^{\beta\eta}$.
- We also show:

$$\frac{t =_w u}{\mathrm{eval}\; t\; \vec{v} = \mathrm{eval}\; u\; \vec{v}}$$

- $$\mathrm{nf}\; t = \mathrm{quote}^{\mathrm{w}}(\mathrm{eval}\; t\; \mathrm{id})$$

  where $\mathrm{id} \in \mathrm{Env}\,\Gamma\,\Gamma$ is the identity environment.

eval

$$\frac{t \in \mathrm{Tm}\,\Gamma\,\sigma \quad \vec{v} \in \mathrm{Env}\,\Delta\,\Gamma}{\mathrm{eval}\,t\,\vec{v} \in \mathrm{Val}\,\Delta\,\sigma} \qquad \frac{f \in \mathrm{Val}\,\Gamma\,(\sigma \to \tau) \quad v \in \mathrm{Val}\,\Gamma\,\sigma}{f @ v \in \mathrm{Val}\,\Gamma\,\tau}$$

$$\begin{array}{rcl}
\mathrm{eval}\,\mathrm{v}_0\,(\vec{v}, v) & = & v \\
\mathrm{eval}\,t^{+\sigma}\,(\vec{v}, v) & = & \mathrm{eval}\,t\,\vec{v} \\
\mathrm{eval}\,(\lambda^\sigma t)\,\vec{v} & = & (\lambda^\sigma t)[\vec{v}] \\
\mathrm{eval}\,(t\,u)\,\vec{v} & = & (\mathrm{eval}\,t\,\vec{v}) @ (\mathrm{eval}\,u\,\vec{v})
\end{array}$$

$$\begin{array}{rcl}
(\lambda^\sigma t[\vec{v}]) @ v & = & \mathrm{eval}\,t\,(\vec{v}, v) \\
n @ v & = & n\,v
\end{array}$$

Val, Ne

$$\frac{t \in \mathrm{Tm}\,\Gamma.\sigma\,\tau \qquad \vec{v} \in \mathrm{Env}\,\Delta\,\Gamma}{\lambda^{\sigma} t[\vec{v}] \in \mathrm{Val}\,\Delta\,(\sigma \to \tau)} \qquad \frac{n \in \mathrm{Ne}\,\Gamma\,\sigma}{n \in \mathrm{Val}\,\Gamma\,\sigma}$$

$$\frac{x \in \mathrm{Var}\,\Gamma\,\sigma}{x \in \mathrm{Ne}\,\Gamma\,\sigma} \qquad \frac{n \in \mathrm{Ne}\,\Gamma\,(\sigma \to \tau) \qquad v \in \mathrm{Val}\,\Gamma\,\sigma}{n\,v \in \mathrm{Ne}\,\Gamma\,\tau}$$

$$\frac{}{() \in \mathrm{Env}\,\Gamma\,\bullet} \qquad \frac{\vec{v} \in \mathrm{Env}\,\Gamma\,\Delta \qquad v \in \mathrm{Val}\,\Gamma\,\sigma}{(\vec{v}, v) \in \mathrm{Env}\,\Gamma\,\Delta.\sigma}$$

where $\mathrm{Var}\,\Gamma\,\sigma \subseteq \mathrm{Tm}\,\Gamma\,\sigma$
only using v_0 and $t^{+\sigma}$.

Partiality

- It is not clear, that eval and $@$ are total.
  We use ideas from Bove & Capretta.
- We use inductively defined relations:

$$\frac{t \in \mathrm{Tm}\,\Gamma\,\sigma \quad \vec{v} \in \mathrm{Env}\,\Delta\,\Gamma \quad w \in \mathrm{Val}\,\Delta\,\sigma}{\mathrm{eval}\,t\,\vec{v} \downarrow w \in \mathbf{Prop}}$$

$$\frac{f \in \mathrm{Val}\,\Gamma\,(\sigma \to \tau) \quad v \in \mathrm{Val}\,\Gamma\,\sigma \quad w \in \mathrm{Val}\,\Gamma\,\tau}{f@v \downarrow w \in \mathbf{Prop}}$$

- We write

$$\begin{aligned} \mathrm{eval}\,t\,\vec{v} \downarrow &= \exists w.\mathrm{eval}\,t\,\vec{v} \downarrow w \\ f@v \downarrow &= \exists w.f@v \downarrow w \end{aligned}$$

- We can define total versions of eval and $@$ by structural induction over $\mathrm{eval}\,t\,\vec{v} \downarrow$ and $f@v \downarrow$.

quote

$$\frac{v \in \mathrm{Val}\,\Gamma\,\sigma}{\mathrm{quote}\,v \in \mathrm{Nf}\,\Gamma\,\sigma}$$

$$\begin{aligned}
\mathrm{quote}^{\mathrm{w}}\,(\lambda^{\sigma} t[\vec{w}]) &= \lambda^{\sigma} t[\vec{w}] \\
\mathrm{quote}^{\beta}\,(\lambda^{\sigma} t[\vec{w}]) &= \lambda^{\sigma} \mathrm{quote}^{\beta}(\mathrm{nf}\; t\; \vec{v}) \\
\mathrm{quote}^{\beta\eta}_{\sigma\to\tau}\, f &= \lambda^{\sigma} \mathrm{quote}^{\beta\eta}(f^{+\sigma} @ \mathrm{v}_0)
\end{aligned}$$

Strongly computable

$$\frac{\forall v.\mathrm{SCV}^{\sigma}\ v \implies f@v \downarrow w \wedge \mathrm{quote}\ w =_{w} (\mathrm{quote}\ f)\,(\mathrm{quote}\ t)}{\mathrm{SCV}^{\sigma \to \tau}\ f}$$

$$\frac{\forall \vec{v}.\mathrm{SCV}\ \vec{v} \implies \mathrm{eval}\ t\ \vec{v} \downarrow w \wedge t[\mathrm{quote}\ \vec{v}] = \mathrm{quote}\ w \wedge \mathrm{SCV}\ w}{\mathrm{SCT}\ t}$$

Theorem $\dfrac{t \in \mathrm{Tm}\,\Gamma\,\sigma}{\mathrm{SCT}^{\sigma}\ t}$ by induction over t.

Corollary $\dfrac{t \in \mathrm{Tm}\,\Gamma\,\sigma}{\mathrm{nf}\ t \downarrow v \wedge \mathrm{quote}\ v \simeq_{\mathrm{w}} t}$

Conclusions

- Big step normalisation (BSN) is an alternative to using small step reduction and prove strong normalisation and confluence.
- We hope that BSN leads to simpler or new proofs for typed λ calculi.
- The definition of nf is similar to the ones actually used in implementations.
- It seems straightforward to implement a substitution calculus similar to $\lambda^\sigma + \beta\eta$.
- Unlike *Normalisation by evaluation* we don't need higher order functions.
- See *Tait in one big step* (joint with James Chapman, MSFP 06) for an application of BSN to combinatory logic.