Names, Equations, Relations: Practical Ways to Reason about *new*

Ian Stark

BRICS
Department of Computer Science
University of Aarhus
Denmark

April 1997

What does it mean to be *new*?

Many useful aspects of programming languages depend on ‘names’: anonymous tags taken on demand from some infinite supply.

A proper theory of these can help to analyse notions of identity, privacy, scope, pointers, interference, . . .

This talk presents a calculus of names and higher-order functions, with a logic of equations and relations for reasoning about them.

The nu-calculus

A simply-typed lambda-calculus with a type ν of *names* $n, m, \ldots$ that can be compared $(n = m)$ and created fresh $(\nu n.M)$.

There are rules for typing $s, \Gamma \vdash M : \sigma$ and evaluation $s \vdash M \Downarrow (s')V$. For example:

$$\nu n.\nu n'.(n = n') \ \Downarrow \ (n, n')\, false$$

$$\nu n.\nu n'.\lambda x.(\mathrm{if}\, x = n'\, then\, n\, else\, n')$$

This second function has to be applied at least twice to extract all the names within.

Some more expressions

The nu-calculus is call-by-value, and general β-conversion is not appropriate.

$$(\lambda x.(x = x))(\nu n.n) \Downarrow (n)\, true$$
$$(\nu n.n) = (\nu n.n) \Downarrow (n_1, n_2)\, false\ .$$

The expression $\nu n.n$ used here can be usefully abbreviated as new.

Contextual equivalence

Two expressions are *contextually equivalent* if they can be freely exchanged in any program.

$$\nu n.M \approx M \qquad \nu n.\nu n'.M \approx \nu n'.\nu n.M$$

$$(\lambda x.M)V \approx M[V/x] \qquad \nu n.(\lambda x.n) \not\approx \lambda x.(\nu n.n)$$

$$\mathit{if}\ B\ \mathit{then}\ (\nu n.M)\ \mathit{else}\ M' \ \approx\ \nu n.(\mathit{if}\ B\ \mathit{then}\ M\ \mathit{else}\ M')$$

$$\nu n.\lambda x.(x = n) \ \approx\ \lambda x.\mathit{false}$$

This last equivalence relies on the name n remaining private however the function is used.

Some other contextual equivalences

$$
\begin{aligned}
\nu n.\nu n'.\lambda f.(fn = fn') &\approx \lambda f.\mathrm{true} \\
&\not\approx \nu n.\lambda f.\nu n'.(fn = fn')\ .
\end{aligned}
$$

These are distinguished by the function

$$(\lambda F : (\nu \to o) \to o\ .\ F(\lambda x.F(\lambda y.x = y)))\ .$$

Natural numbers:

$$
\begin{aligned}
F_p \ = \ \nu n_0 \ldots \nu n_p.\lambda x.\ & \mathrm{if}\ x = n_0\ \mathrm{then}\ n_1 \\
& \mathrm{else\,if}\ x = n_1\ \mathrm{then}\ n_2 \\
& \quad \vdots \\
& \mathrm{else\,if}\ x = n_p\ \mathrm{then}\ n_0\ \mathrm{else}\ n_0\ .
\end{aligned}
$$

Problems with contextual equivalence

Because it considers all possible programs, contextual equivalence is

- ✓ the right notion for checking code transformation, replacing algorithms, checking assertions and matching specifications;
- $\times$ hard to demonstrate in any particular case.

Thus we turn to other relations that imply contextual equivalence but are simpler to demonstrate.

Operational methods

Applicative equivalence

Identifies functions if they give equivalent results at all arguments, up to ‘garbage collection’ of names.

Sufficient to reason in the presence of names, but not about the names themselves.

Logical relations

Use *spans* $R : s_1 \rightleftharpoons s_2$ between sets of names. Functions are related if they take related arguments to related results.

This is enough to reason about the private/public distinction, and in particular to prove all first-order contextual equivalences.

Problems with operational methods

- Consideration of all possible arguments.
- Needs a detailed understanding of evaluation.
- Open terms and higher-order functions require meta-level reasoning.
- Proof-theoretic complexity issues are “interesting”.

To avoid these we distil the hands-on operational methods into two systems of rules.

Equational reasoning

$$\beta_{id}\ \frac{}{s, \Gamma \vdash (\lambda x.x)M = M} \qquad \frac{}{s, \Gamma \vdash F(\nu n.M) = \nu n.(FM)}\ n \notin fn(F)$$

$$\frac{\begin{array}{ll} s, \Gamma \vdash \ M_1[n/x] = M_2[n/x] & \text{each } n \in s \\ s \oplus \{n'\}, \Gamma \vdash M_1[n'/x] = M_2[n'/x] & \text{some fresh } n' \end{array}}{s, \Gamma \oplus \{x : \nu\} \vdash M_1 = M_2}$$

$$s, \Gamma \vdash M_1 = M_2 \quad \Longrightarrow \quad s, \Gamma \vdash M_1 \approx M_2 \ .$$

- Similar in power to applicative equivalence, but easier to use.
- Works directly on open terms and at higher types.
- Provides more than just $\beta\eta$-etc. rewriting.

Relational reasoning

$$\frac{\Gamma \vdash M_1 \ (R \oplus \overleftarrow{n_1}) \ M_2}{\Gamma \vdash (\nu n_1 . M_1) \ R \ M_2} \qquad \frac{s, \Gamma \vdash M_1 = M_2 \quad \Gamma \vdash M_2 \ R \ M_3}{\Gamma \vdash M_1 \ R \ M_3}$$

$$\frac{\begin{array}{ll} \Gamma \vdash (M_1[n/x]) \ (R \oplus \widehat{n}) \ (M_2[n/x]) & \text{some fresh } n \\ \Gamma \vdash (M_1[n_1/x]) \ R \ (M_2[n_2/x]) & \text{each } (n_1, n_2) \in R \end{array}}{\Gamma \oplus \{x : \nu\} \vdash M_1 \ R \ M_2}$$

$$\Gamma \vdash M_1 \ (id_s) \ M_2 \quad \Longrightarrow \quad s, \Gamma \vdash M_1 \approx M_2 \ .$$

- Integrates fully with equational reasoning.
- Explicit handling of private *vs.* public names.
- Complete for ground types and first-order functions.

Example

To demonstrate

$$\nu n.\lambda x{:}\nu.(x = n) \;\approx\; \lambda x{:}\nu.false$$

the crucial closing steps are

$$\frac{\dfrac{x : \nu \vdash (x = n)\ (\overleftarrow{n}\,)_o\ false}{\vdash (\lambda x.(x = n))\ (\overleftarrow{n}\,)_{\nu\to o}\ (\lambda x.false)}}{\vdash (\nu n.\lambda x.(x = n))\ \emptyset_{\nu\to o}\ (\lambda x.false)}$$

The span $(\overleftarrow{n}\,) : \{n\} \rightleftharpoons \emptyset$ used here captures our intuition that the name bound to n on the left hand side is private, never revealed, and need not be matched in the right hand expression.

Results

$$
\begin{tikzcd}
\text{Applicative equivalence} \arrow[r, phantom, "\subseteq"] \arrow[d, phantom, "\rotatebox{90}{$\subseteq$}"] & \text{Logical relations} \arrow[d, phantom, "\rotatebox{90}{$\subseteq$}"] \arrow[dr, phantom, "\subseteq"] & \\
 & & \text{Contextual equivalence} \\
\text{Equational reasoning} \arrow[r, phantom, "\subseteq"] & \text{Relational reasoning} \arrow[ur, phantom, "\subseteq"] &
\end{tikzcd}
$$

Summary

Accessibility	**Scope**
Denotational	Equational
Operational	Relations on names
Rule-based	Relations on states
Mechanised	Exceptions, concurrency, . . .

These two dimensions are not a tradeoff! We can reasonably expect progress on both fronts.