# Operational Versus Denotational Methods in the Semantics of Higher Order Languages

Andrew M. Pitts
Cambridge University Computer Laboratory

Slides available at:

www.cl.cam.ac.uk/users/ap/talks/plilp98/

‘But once feasibility has been checked by an operational model, operational reasoning should be immediately abandoned; it is essential that all subsequent reasoning, calculation and design should be conducted in each case at the highest possible level of abstraction.’

C. A.R. Hoare (1996, page 182)

The advice in the second half of the above quotation—that one should always strive for the highest possible level of abstraction—is very sound. Nevertheless, I am going to try to persuade you to ignore the advice in the first half of the quotation and embrace, rather than abandon, operational reasoning. The justification for this lies in the claim that operational semantics can (sometimes—we are still learning how) be presented at a sufficient 'level of abstraction' to support quite palatable methods of reasoning, calculation and design. This is possible, in part, through the adoption of syntactic analogues of some of the notions and techniques of denotational semantics. It's this 'technology transfer' from denotational semantics to operational semantics which forms the background to this tutorial talk.

Let's begin by surveying the strengths and weaknesses of the denotational and operational approaches to PL semantics for various kinds of application . . .

## Uses of formal semantics of programming languages

| | denotational | operational |
|---|---|---|
| **understanding** | $+$ | $-$ |
| **implementing** | $-$ | $+$ |
| **proving:** | | |
| ‘pencil & paper’ | $\pm$ | $\pm$ |
| machine-assisted | $\pm$ | $+$ |

**‘Understanding’:**

Mainly because of the strong discipline of compositionality imposed by the denotational approach, a denotational model can often tell us about *the expressive power and inter-relationship between various computational features*—something that an operational semantics is much less likely to do.

A classic example of this: the Scott-Strachey denotational semantics of mutable variables that can store function closures involves a reflexive domain (one containing its own function space as a retract); such domains model the untyped lambda calculus; in the untyped lambda calculus we can write fixpoint combinators; therefore we may expect to be able to simulate fixpoint recursion using such ‘second-order’ variables (and indeed we can).

**‘Implementing’:**

That operational semantics is more useful than denotational semantics here is clear.

**‘Proving’:**

See next slide.

## Proving properties of higher order programming languages

- Higher order features (functions, procedures, processes, . . . ) severely complicate matters!
- Yesteryear: many *purely operational* properties were tackled with denotational tools, for lack of any other viable approach.
- Now: viable operational methods are emerging—some are syntactical analogues of denotational techniques, some are not.

## Machine-assisted proving

**Denotational methods:** relatively sophisticated mathematics—not amenable to fully formalised proof?

**Operational methods:** simpler mathematics, but long and tedious proofs (lots of induction clauses)—good candidate for machine-assistance.

## Case study: Milner's 'Context Lemma' for PCF

- statement
- proof via a syntactical logical relation and the 'unwinding theorem' (syntactic fixpoint induction)
- robustness of the proof method w.r.t. language extensions

---

N.B. the issue is not so much how to prove this *particular* result, but how to develop *robust* proof methods for this kind of result—ones which can cope with presence of extra PL features.

I've chosen this example because

- PCF is simple and well-known;
- Context Lemma is a non-trivial result (just!) whose proof(s) we can discuss in detail within the time available;
- the result and its proofs generalise in interesting ways to languages with more complicated features (see later).

## PCF syntax

**Types**

$$\tau ::= nat \mid \tau \to \tau$$

**Terms**

$$\begin{aligned} M ::= \mathbf{0} \mid \mathbf{succ}(M) \mid \mathbf{pred}(M) \\ \mid \mathbf{if}\ M = \mathbf{0}\ \mathbf{then}\ M\ \mathbf{else}\ M \\ \mid x \mid \lambda\, x : \tau \,.\, M \mid M\ M \mid \mathbf{fix}(M) \end{aligned}$$

where $x$ ranges over an infinite set of *variables*.

Identify expressions up to renaming of $\lambda$-bound variables.

# PCF typing relation, $\Gamma \vdash M : \tau$

- $\Gamma$ is a type environment, i.e. a finite partial function mapping variables to types

- $M$ is a term

- $\tau$ is a type.

Relation is inductively defined by a set of axioms and rules which are syntax-directed (follow structure of $M$). E.g.

$$\frac{\Gamma, x : \tau \vdash M : \tau'}{\Gamma \vdash \lambda\, x : \tau \,.\, M : \tau \to \tau'} \qquad \frac{\Gamma \vdash M : \tau \to \tau}{\Gamma \vdash \mathbf{fix}(M) : \tau}$$

[Other axioms and rules omitted.]

## PCF evaluation relation, $M \Downarrow V$

$M, V$ are closed PCF terms and $V$ is a value:

$$V \ ::= \mathbf{0} \mid \mathbf{succ}(V) \mid \lambda\, x : \tau \,.\, M$$

Evaluation relation is inductively defined by the axioms and rules that follow the structure of $M$. E.g.

CBN rule for application: $$\frac{F \Downarrow \lambda\, x : \tau \,.\, M \quad M[A/x] \Downarrow V}{F\, A \Downarrow V}$$

unfolding rule for $\mathbf{fix}$: $$\frac{F\, \mathbf{fix}(F) \Downarrow V}{\mathbf{fix}(F) \Downarrow V}$$

[Other axioms and rules omitted.]

# Contextual equivalence of PCF terms

Given PCF terms $M_1$, $M_2$, PCF type $\tau$, and type environment $\Gamma$

$$\boxed{\Gamma \vdash M_1 \cong_{\mathrm{ctx}} M_2 : \tau}$$

is defined to hold iff

- Both the typings $\Gamma \vdash M_1 : \tau$ and $\Gamma \vdash M_2 : \tau$ hold.
- For all PCF contexts $\mathcal{M}$ for which $\mathcal{M}[M_1]$ and $\mathcal{M}[M_2]$ are closed terms of type $nat$ and for all closed values $V : nat$,

$$\mathcal{M}[M_1] \Downarrow V \;\Leftrightarrow\; \mathcal{M}[M_2] \Downarrow V.$$

[N.B. the closed values of type $nat$ are the numerals, $\mathbf{succ}^n(\mathbf{0})$.]

It's easy to see directly from the definition of $\cong_{\mathrm{ctx}}$ that it is a *congruence relation*—i.e. an equivalence relation that respects the various PCF term-forming operations. However nearly all identities we might expect $\cong_{\mathrm{ctx}}$ to satisfy (arithmetic identities, beta and eta conversion, unfolding fixpoints, etc) are very hard to prove directly from its definition—because of that quantification over all contexts $\mathcal{M}[-]$ occurring in the definition.

Harder still are conditional-equational properties of $\cong_{\mathrm{ctx}}$, such as the following . . .

## A version of Milner's Context Lemma for PCF

can be stated as an extensionality property of closed function expressions:

**Function Extensionality Theorem.**

$$\vdash F_1 \cong_{\mathrm{ctx}} F_2 : \tau \to \tau'$$

*holds if (& only if)*

$$\vdash F_1\, A \cong_{\mathrm{ctx}} F_2\, A : \tau'$$

*holds for all* $A : \tau$.

## Proofs of Milner's Context Lemma (and related extensionality properties)

- Milner's original proof was syntactic, but for a limited version of PCF (PCF combinators) and, apparently, the method does not generalise well.
- Howe (I&C 124, 1996) developed a nice, robust syntactical method.
- Can get such results from adequate denotational semantics via logical relations between syntax and semantics (Mulmuley; Pitts, LJIGPL 5, 1997).

Here: an operational version of this last approach.

## Our tools

- Induction over the structure of type expressions (`TypInd`).
- Induction over the derivation of typings (`TermInd`).
  $\Gamma \vdash M : \tau$.
- Induction over the derivation of evaluations $M \Downarrow V$.
- Syntactic fixpoint induction (to be explained).

The first three are just instances of the same tool, i.e. the usual induction principle associated with any monotone inductive definition; but the fourth is of quite a different nature.

Note that since $\cong_{\mathrm{ctx}}$ is reflexive, the Function Extensionality Theorem can be restated as

$$\vdash F_1 \cong_{\mathrm{ctx}} F_2 : \tau \to \tau'$$

*holds if (& only if) for all* $A_1, A_2$

$$\vdash A_1 \cong_{\mathrm{ctx}} A_2 : \tau \;\Rightarrow\; \vdash F_1\, A_1 \cong_{\mathrm{ctx}} F_2\, A_2 : \tau'.$$

This motivates the following definition of a binary 'logical' relation $\Delta_\tau$ between closed terms of same type, defined by `TypInd` ...

# A logical relation between closed PCF terms

At type $nat$:

$$(N_1, N_2) \in \Delta_{nat} \overset{\text{def}}{\Leftrightarrow} \forall V\, (N_1 \Downarrow V \;\Leftrightarrow\; N_2 \Downarrow V)$$

At function types:

$$\begin{aligned}(F_1, F_2) \in \Delta_{\tau \to \tau'} \overset{\text{def}}{\Leftrightarrow}\ & \forall A_1, A_2 \\ & (A_1, A_2) \in \Delta_\tau \;\Rightarrow\; (F_1\, A_1, F_2\, A_2) \in \Delta_{\tau'}\end{aligned}$$

To prove Function Extensionality Theorem for $\cong_{\mathrm{ctx}}$, suffices to show for all $\tau$, $M_1$, $M_2$ that

$$\vdash M_1 \cong_{\mathrm{ctx}} M_2 : \tau \;\Leftrightarrow\; (M_1, M_2) \in \Delta_\tau .$$

---

In fact coincidence of $\cong_{\mathrm{ctx}}$ with $\Delta$ proves more about $\cong_{\mathrm{ctx}}$ than just function extensionality—e.g. it implies

$$\begin{aligned} (\lambda\, x : \tau\,.\, M)\, A \;&\cong_{\mathrm{ctx}}\; M[A/x] \\ \mathbf{fix}(M) \;&\cong_{\mathrm{ctx}}\; M\,\mathbf{fix}(M) \end{aligned}$$

and other pairs with same immediate evaluation behaviour (‘cos $\Delta$ contains Kleene equivalence—see later).

# Key lemmas about $\Delta$

1. $\Delta$ *is respected by* $\cong_{\mathrm{ctx}}$:

$$\frac{M_1 \cong_{\mathrm{ctx}} M_2 : \tau \quad (M_2, M_3) \in \Delta_\tau \quad M_3 \cong_{\mathrm{ctx}} M_4 : \tau}{(M_1, M_4) \in \Delta_\tau}$$

[Proof: straightforward application of `TypInd` and congruence properties of $\cong_{\mathrm{ctx}}$.]

2. **Fundamental Property**: $\Delta$ (extended to open terms via closing substitutions) *is compatible with the PCF term-formers* ...

## Logical relation extended to open terms

Given any $\Gamma \vdash M_1 : \tau$ and $\Gamma \vdash M_2 : \tau$,
with $\Gamma = \{x_1 : \tau_1, \ldots, x_n : \tau_n\}$ say,
define:

$$\Gamma \vdash M_1 \mathrel{\Delta} M_2 : \tau \overset{\text{def}}{\Leftrightarrow}$$
$$\forall \, (A_{11}, A_{21}) \in \Delta_{\tau_1}, \ldots, (A_{1n}, A_{2n}) \in \Delta_{\tau_n} \, .$$
$$(M_1[\vec{A}_1/\vec{x}], M_2[\vec{A}_2/\vec{x}]) \in \Delta_\tau .$$

$\cong_{\mathrm{ctx}}$ and $\Delta$ are examples of PCF (binary) open term relations

i.e. sets $\mathcal{R}$ of quadruples $(\Gamma, M_1, M_2, \tau)$ satisfying

$$\Gamma \vdash M_1 \; \mathcal{R} \; M_2 : \tau \quad \Rightarrow \quad \Gamma \vdash M_1 : \tau \text{ and } \Gamma \vdash M_1 : \tau$$

(where we write $\Gamma \vdash M_1 \; \mathcal{R} \; M_2 : \tau$ instead of $(\Gamma, M_1, M_2, \tau) \in \mathcal{R}$).

## Compatibility

An open term relation $\mathcal{R}$ is compatible if it is respected by each of the PCF term-forming operations, i.e.

$$\Gamma, x : \tau \vdash x \mathrel{\mathcal{R}} x : \tau$$

$$\frac{\Gamma, x : \tau \vdash M_1 \mathrel{\mathcal{R}} M_2 : \tau'}{\Gamma \vdash (\lambda\, x : \tau \,.\, M_1) \mathrel{\mathcal{R}} (\lambda\, x : \tau \,.\, M_2) : \tau \to \tau'}$$

$$\frac{\Gamma \vdash F_1 \mathrel{\mathcal{R}} F_2 : \tau \to \tau' \quad \Gamma \vdash A_1 \mathrel{\mathcal{R}} A_2 : \tau}{\Gamma \vdash (F_1\, A_1) \mathrel{\mathcal{R}} (F_2\, A_2) : \tau'}$$

$$\frac{\Gamma \vdash F_1 \mathrel{\mathcal{R}} F_2 : \tau \to \tau}{\Gamma \vdash \mathbf{fix}(F_1) \mathrel{\mathcal{R}} \mathbf{fix}(F_2) : \tau}$$

etc, etc.

## ‘Context-free’ characterisation of contextual equivalence

**Lemma.** $\cong_{\mathrm{ctx}}$ *is the greatest open term relation* $\mathcal{R}$ *which is compatible and*

adequate: if $\emptyset \vdash N_1 \mathrel{\mathcal{R}} N_2 : nat$, then for all $V : nat$

$$N_1 \Downarrow V \;\Leftrightarrow\; N_2 \Downarrow V\,.$$

---

A general property of $\cong_{\mathrm{ctx}}$ (i.e. not specific to PCF).

Enables development of theory of contextual equivalence without having to mention contexts explicitly—desirable in presence of multiple syntactic categories involving complicated forms of static binding.

See (Lassen, PhD Thesis, 1998) for proof and applications.

## Functional Extensionality Theorem—proof outline

(A) Prove $\Delta$ is compatible.

(B) Since $\Delta$ is adequate by definition of $\Delta_{nat}$, conclude from (A) and the 'context-free' characterisation of $\cong_{\mathrm{ctx}}$ that $\Delta \subseteq \cong_{\mathrm{ctx}}$.

(C) From (A), $\Delta$ is reflexive (*any compatible relation is reflexive*—proof by `TermInd`), so

$$
\begin{array}{ll}
M_1 \cong_{\mathrm{ctx}} M_2 : \tau & \\
\quad \Rightarrow M_1 \cong_{\mathrm{ctx}} M_2 : \tau \ \& \ (M_2, M_2) \in \Delta_\tau & \text{reflexivity} \\
\quad \Rightarrow (M_1, M_2) \in \Delta_\tau & \text{Key Lemma 1} \\
\quad \Rightarrow M_1 \cong_{\mathrm{ctx}} M_2 : \tau & \text{by (B).}
\end{array}
$$

So $\cong_{\mathrm{ctx}}$ coincides with $\Delta$, and hence is extensional.

It remains to prove (A) . . .

But first it's time for a break!

# Intermission

‘Gentlemen, it is a fact that every philosopher of eminence for the last two centuries has either been murdered, or, at the least, been very near it, insomuch that if a man calls himself a philosopher, and never had his life attempted, rest assured there is nothing in him; and against Locke’s philosophy in particular, I think it is an unanswerable objection (if we needed any) that, although he carried his throat about him in this world for seventy-two years, no man ever condescended to cut it.’

Thomas de Quincey, *Murder Considered as One of the Fine Arts*

# Causes of Death of Some Philosophers

Abelard: nun. Augustine: hippo. Barwise: bad situation. Belnap: became irrelevant. Cantor: set aside. Carnap: left the material mode. Chomsky: became past tense. Church: recursive causes. Copernicus: revolution. Darwin: unfit. Derrida: deconstructed. Descartes: stopped thinking. Devlin: fell off Clapham omnibus. Einstein: diced with God. Frege: fell under a concept. Freud: slipped. Galileo: stopped moving. Gentzen: cut turned septic. Gödel: became incomplete. Heisenberg: uncertain causes. Heraclitus: fell in the same river twice. Hilbert: informal causes. Kant: found the means to his own end. Kripke: went rigid. Kuhn: paradigm lost. Leibniz: monadnucleosis. Nietzsche: overpowered himself. Ockham: shaved beyond necessity. Pascal: the wagers of sin. Peirce: abducted. Penrose: became computable. Plato: caved in. Pythagoras: squared on the hypotenuse. Quine: semantic ascent. Russell: vicious circle. Sartre: nothing doing. Sheffer: stroke. Spinoza: substance abuse. Tarski: ‘death’. Turing: solved the halting problem. Wittgenstein: became the late Wittgenstein. Zeno: immobility.

With thanks to `www.dar.cam.ac.uk/dhm11/Deaths.html`

Back to work.

(Reprise Slide 22.)

## Proving compatibility of $\Delta$: abstraction

To show $\dfrac{\Gamma, x : \tau \vdash M_1 \mathrel{\Delta} M_2 : \tau'}{\Gamma \vdash (\lambda\, x : \tau \,.\, M_1) \mathrel{\Delta} (\lambda\, x : \tau \,.\, M_2) : \tau \to \tau'}$

suffices to show that if

$$(M_1[A_1/x], M_2[A_2/x]) \in \Delta_{\tau'} \quad \text{for all } (A_1, A_2) \in \Delta_\tau$$

then $(\lambda\, x : \tau \,.\, M_1, \lambda\, x : \tau \,.\, M_2) \in \Delta_{\tau \to \tau'}$, i.e. then

$$\begin{aligned} &(A_1, A_2) \in \Delta_\tau \;\Rightarrow \\ &\qquad ((\lambda\, x : \tau \,.\, M_1)A_1, (\lambda\, x : \tau \,.\, M_2)A_2) \in \Delta_{\tau'}. \end{aligned}$$

This is a consequence of the fact that $\Delta$ respects Kleene equivalence . . .

# Kleene equivalence

Given closed terms $M_1, M_2$ of any type $\tau$, write $\boxed{M_1 \cong_{\mathrm{kl}} M_2 : \tau}$ to mean $M_1 \Downarrow V \;\Leftrightarrow\; M_2 \Downarrow V$ holds for all closed values $V$.

Examples:

$$(\lambda\, x : \tau \,.\, M)A \cong_{\mathrm{kl}} M[A/x] : \tau'$$

$$F_1 \cong_{\mathrm{kl}} F_2 : \tau \to \tau' \;\;\&\;\; A : \tau \;\Rightarrow\; F_1\, A \cong_{\mathrm{kl}} F_2\, A : \tau'$$

The second fact suffices to show that $\Delta$ respects $\cong_{\mathrm{kl}}$ (proof by `TypInd`—cf. Key Lemma 1) and then the first suffices to prove compatibility of $\Delta$ w.r.t. abstraction.

Note that the logical relation at type $nat$ is just $\cong_{\mathrm{kl}}$.

The compatibility properties of $\Delta$ for variables, $\mathbf{0}$, $\mathbf{succ}$, $\mathbf{pred}$, and application are all simple to prove. Compatibility for conditionals requires a use of $\mathtt{TypInd}$ over the type of $\mathbf{if}\ N = \mathbf{0}\ \mathbf{then}\ M\ \mathbf{else}\ M'$; the induction step for function types uses

$$(\mathbf{if}\ N = \mathbf{0}\ \mathbf{then}\ M\ \mathbf{else}\ M')\,A \;\cong_{\mathrm{kl}}\; \mathbf{if}\ N = \mathbf{0}\ \mathbf{then}\ M\,A\ \mathbf{else}\ M'\,A$$

The remaining compatibility condition is that for $\mathbf{fix}$ . . .

## Proving compatibility of $\Delta$: fixpoints

To show

$$\frac{\Gamma \vdash F_1 \;\Delta\; F_2 : \tau \to \tau}{\Gamma \vdash \mathbf{fix}(F_1) \;\Delta\; \mathbf{fix}(F_2) : \tau}$$

suffices to show

$$(F_1, F_2) \in \Delta_{\tau \to \tau} \;\;\Rightarrow\;\; (\mathbf{fix}(F_1), \mathbf{fix}(F_2)) \in \Delta_\tau .$$

How?

Can't do it by `TypInd`—e.g. consider base case of such an induction . . .

## Proof of $(F_1, F_2) \in \Delta_{nat \to nat} \Rightarrow (\mathbf{fix}(F_1), \mathbf{fix}(F_2)) \in \Delta_{nat}$ . . . not!

---

Assume $(F_1, F_2) \in \Delta_{nat \to nat}$.

Have to show $\mathbf{fix}(F_1) \Downarrow V \;\Leftrightarrow\; \mathbf{fix}(F_2) \Downarrow V$.

But note that $\mathbf{fix}(F_i) \Downarrow V$ iff $F_i\,\mathbf{fix}(F_i) \Downarrow V$.

So need to prove $(F_1\,\mathbf{fix}(F_1), F_2\,\mathbf{fix}(F_2)) \in \Delta_{nat}$

which would follow from $(F_1, F_2) \in \Delta_{nat \to nat}$ and . . .

. . . $(\mathbf{fix}(F_1), \mathbf{fix}(F_2)) \in \Delta_{nat}$, which is what we're trying to prove.

Damn!

We can prove compatibility for $\mathbf{fix}$ via a syntactic analogue of . . .

**Scott's Fixpoint Induction.**

*Given*

*a continuous function on a domain* $f : D \to D$

*and an* *admissible* *subset* $\Delta \subseteq D$
*(i.e.* $\Delta$ *contains* $\perp$ *and is closed under lubs of chains),*

*if* $f$ *maps* $\Delta$ *into itself, then*

$$lfp(f) = \bigsqcup_{n \geq 0} f^n(\perp) \quad \text{is in } \Delta.$$

| **semantics** | $\sim$ | **syntax** |
|---|---|---|
| domain $D$ | $\sim$ | type $\tau$ |
| cts function $f : D \to D$ | $\sim$ | *any* closed term $F : \tau \to \tau$ |
| $\bot \in D$ | $\sim$ | $\Omega_\tau \stackrel{\text{def}}{=} \mathbf{fix}(\lambda\, x : \tau\, .\, x)$ |
| $f^n(\bot)$ | $\sim$ | $\begin{cases} \mathbf{fix}^{(0)}(F) & \stackrel{\text{def}}{=} \Omega_\tau \\ \mathbf{fix}^{(n+1)}(F) & \stackrel{\text{def}}{=} F\,\mathbf{fix}^{(n)}(F) \end{cases}$ |
| $\mathit{lfp}(f)$ | $\sim$ | $\mathbf{fix}(F)$ |

## Unwinding Theorems

**At type $nat$:**

$$\mathcal{M}[\mathbf{fix}(F)] \Downarrow V \;\Leftrightarrow\; \exists n \geq 0.\, \mathcal{M}[\mathbf{fix}^{(n)}(F)] \Downarrow V$$

(recall that values $V : nat$ are numerals $\mathbf{succ}^k(\mathbf{0})$).

**At any type:**

$$\mathcal{M}[\mathbf{fix}(F)]{\Downarrow} \;\Leftrightarrow\; \exists n \geq 0.\, \mathcal{M}[\mathbf{fix}^{(n)}(F)]{\Downarrow}$$

(where $M{\Downarrow}$ means $\exists V.\, M \Downarrow V$).

---

Not PCF-specific. Correlates with continuity properties in domain-theoretic denotational semantics (e.g. invalid in presence of unbounded non-determinism).

Many proofs of Unwinding Theorems in literature.

(E.g. Mason, Smith & Talcott, 1996; cf. labelling techniques à la Levy for untyped $\lambda$-calculus.)

Can use induction on derivation of evaluation, $\Downarrow$, to prove it . . .

. . . but much easier to use induction on length of transition sequences for a suitable transition relation $M \to M'$ satisfying

$$M \Downarrow V \;\Leftrightarrow\; M \to^{*} V.$$

The proof is easier still if $\longrightarrow$ defined not in the style of Plotkin's original SOS, but à la Felleisen using evaluation contexts, or better still, using stacks of evaluation frames.

(See Pitts & Stark, 1998; Lassen, PhD Thesis, 1998.)

Alas, there is not enough time to elaborate upon this.

Instead, let's use the Unwinding Theorem to complete the last step in the proof of the coincidence of $\Delta$ with $\cong_{\mathrm{ctx}}$: the compatibility property for $\mathbf{fix}$. The next slide gives the base case $\tau = nat$; slide 34 gives a syntactic admissibility property of $\Delta$ from which the general case is a simple consequence.

## Proof of

## $(F_1, F_2) \in \Delta_{nat \to nat} \Rightarrow (\mathbf{fix}(F_1), \mathbf{fix}(F_2)) \in \Delta_{nat}$

Assume $(F_1, F_2) \in \Delta_{nat \to nat}$. Then by induction on $n$

(∗) $\forall n \geq 0.\, (\mathbf{fix}^{(n)}(F_1), \mathbf{fix}^{(n)}(F_2)) \in \Delta_{nat}$.

Hence

$$
\begin{array}{rcll}
\mathbf{fix}(F_1) \Downarrow V & \Leftrightarrow & \exists n \geq 0.\, \mathbf{fix}^{(n)}(F_1) \Downarrow V & \text{Unwinding Thm} \\
& \Leftrightarrow & \exists n \geq 0.\, \mathbf{fix}^{(n)}(F_2) \Downarrow V & \text{by } (*) \\
& \Leftrightarrow & \mathbf{fix}(F_2) \Downarrow V & \text{Unwinding Thm.}
\end{array}
$$

So $(\mathbf{fix}(F_1), \mathbf{fix}(F_2)) \in \Delta_{nat}$.

## An admissibility property of the logical relation $\Delta$

**Proposition.** For each type $\tau$

1. $(\Omega_\tau, \Omega_\tau) \in \Delta_\tau$
2. for all $\tau'$, $\mathcal{M}_i[- : \tau'] : \tau$, and $\mathbf{fix}(F_i) : \tau'$ ($i = 1, 2$)

$$\frac{\forall n \geq 0.\, (\mathcal{M}_1[\mathbf{fix}^{(n)}(F_1)], \mathcal{M}_2[\mathbf{fix}^{(n)}(F_2)]) \in \Delta_\tau}{(\mathcal{M}_1[\mathbf{fix}(F_1)], \mathcal{M}_2[\mathbf{fix}(F_2)]) \in \Delta_\tau}$$

Proof: by `TypInd` for $\tau$.

1. Easy (using $(\Omega_{\tau \to \tau'}\ A) \cong_{\mathrm{kl}} \Omega_{\tau'}$ for induction step for $\tau \to \tau'$).
2. Contexts appear in order to make induction step go through; base case of the induction needs the unwinding theorem. . . .

## Extensions

Method of syntactical logical relations for proving properties of recursive programs up to $\cong_{\mathrm{ctx}}$ described here has been extended to cover

- Algol-like local state (Pitts, 1997)
- (impredicative) polymorphic datatypes (Pitts, 1998)
- ML-like local state (Pitts & Stark, 1998) [storing flat data only]
- recursive datatypes (Birkedal & Harper, 1997) [top-level, non-parameterised]

## Existence problem

For PCF, logical relation is constructed using `TypInd`. Works for PCF's *simple types*, but won't work for, e.g., mixed-variance recursive types, or non-flat state (objects).

Try to define the whole family of relations $(\Delta_\tau \mid \tau)$ simultaneously? . . .
It's specified as a fixed point of an operator on such families of relations, but operator is *non-monotone* 'cos of definition of $\Delta_{\tau\to\tau'}$. So why should there be a fixed point?

## Solving the existence problem with ‘syntactic projections’

Syntactic analogue of limit-colimit construction yields a version of Unwinding Theorem expressing identity function as limit of canonical projection functions. (See Smith, 1998.)

Permits construction of non-monotonically specified logical relations by mimicking construction of ‘minimal invariant relations’ on recursive domains of (Pitts, I&C 127, 1996): see (Birkedal & Harper, 1997). But

- Canonical projections may not be definable in the original programming language.
- It’s hard (& complicated) work! Maybe Lassen’s calculus of relations and bisimilarity up-to-context results (PhD Thesis, 1998) will provide a simpler way to many of the desired applications.

# Prospect

Examples of purely operational results which originally were tackled with denotational semantics, but which now are being proved using syntactic methods (and for more fully-featured languages):

- Semantic equivalence of programs—as discussed here.
- Correctness of translations/compilation (e.g. CPS: Birkedal & Harper, 1997).
- Type soundness results (e.g. Felleisen & Wright, 1994; see also Alves-Foss, *Formal Syntax and Semantics of Java*, SLNCS, to appear).

# References


Berry, G. and G. Boudol (1992). The chemical abstract machine. *Theoretical Computer Science 96*.

Birkedal, L. and R. Harper (1997). Relational interpretation of recursive types in an operational setting (Summary). In M. Abadi and T. Ito (Eds.), *Theoretical Aspects of Computer Software, 3rd Int. Symp.*, Volume 1281 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin.

Felleisen, M. and R. Hieb (1992). The revised report on the syntactic theories of sequential control and state. *Theoretical Computer Science 103*, 235–271.

Gordon, A. D. and A. M. Pitts (Eds.) (1998). *Higher Order Operational Techniques in Semantics*. Publications of the Newton Institute. Cambridge University Press.

Harper, R. and C. Stone (1996). A type-theoretic account of Standard ML 1996 (version 2). Technical Report CMU-CS-96-136R, Carnegie Mellon University, Pittsburgh, PA.

Hoare, C. A. R. (1996). Algebra and models. In I. Wand and R. Milner (Eds.), *Computing Tomorrow. Future research directions in computer science*, Chapter 9, pp. 158–187. Cambridge University Press.

Mason, I. A., S. F. Smith, and C. L. Talcott (1996). From operational semantics to domain theory. *Information and Computation 128*(1), 26–47.

Milner, R. (1990). Functions as processes. In *Proc. 17th Int. Coll. on Automata, Languages and Programming*, Volume 443 of *Lecture Notes in Computer Science*, pp. 167–180. Springer-Verlag, Berlin.

Pitts, A. M. (1997). Reasoning about local variables with operationally-based logical relations. In P. W. O'Hearn and R. D. Tennent (Eds.), *Algol-Like Languages*, Volume 2, Chapter 17, pp. 173–193. Birkhauser.

Pitts, A. M. (1998). Existential types:logical relations and operational equivalence. In *Proceedings ICALP'98*, *Lecture Notes in Computer Science*, Springer-Verlag, Berlin.

Pitts, A. M. and I. D. B. Stark (1998). Operational reasoning for functions with local state. In (Gordon and Pitts, 1998).

Smith, S. F. (1998). The coverage of operational semantics. In (Gordon and Pitts, 1998).

Wright, A. K. and M. Felleisen (1994). A syntactic approach to type soundness. *Information and computation 115*, 38–94.