Operational domain theory and its applications

Weng Kin, Ho
Mathematics and Mathematics Education
National Institute of Education, Singapore
wengkin.ho@nie.edu.sg

FOST 2010 Workshop
20 Jan 2010

This is where I come from ...

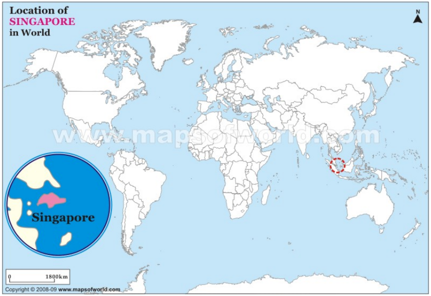


Figure: Location of Singapore in the world map

This is where I come from ...

Figure: National Institute of Education, Singapore

Outline

1. Preliminaries: The language

Outline

1. Preliminaries: The language
2. Denotational semantics

Outline

1. Preliminaries: The language
2. Denotational semantics
3. Problems with Scott model

Outline

1. Preliminaries: The language
2. Denotational semantics
3. Problems with Scott model
4. ODT

Outline

Outline

Outline

Outline

Outline

Subtopics to be covered

Subtopics to be covered

1. Difficulties with domains model and motivation for ODT

Subtopics to be covered

1. Difficulties with domains model and motivation for ODT
2. Rational chain completeness

Subtopics to be covered

1. Difficulties with domains model and motivation for ODT
2. Rational chain completeness
3. Open sets and continuity

Subtopics to be covered

1. Difficulties with domains model and motivation for ODT
2. Rational chain completeness
3. Open sets and continuity
4. ODT: relationship between operational pre-order and topology

Highlights

In today's talk, we encounter

1. the domains model of denotational semantics

Highlights

In today's talk, we encounter

1. the domains model of denotational semantics
2. the problem of Full Abstraction

Highlights

In today's talk, we encounter

1. the domains model of denotational semantics
2. the problem of Full Abstraction
3. operational tools such as bisimilarity and co-induction principle

Highlights

In today's talk, we encounter

1. the domains model of denotational semantics
2. the problem of Full Abstraction
3. operational tools such as bisimilarity and co-induction principle
4. the operational domain theory and topology

Highlights

In today's talk, we encounter

1. the domains model of denotational semantics
2. the problem of Full Abstraction
3. operational tools such as bisimilarity and co-induction principle
4. the operational domain theory and topology
5. their relationship between them

PCF

I assume the audience has some familiarity with the sequential functional language

Programming (language) for Computable Functionals

abbreviated as PCF.

PCF

PCF was brought in by Gordon Plotkin in the form of LCF as a programming language.

Figure: Gordon Plotkin

PCF

In a nutshell ...

$$\text{PCF} = \text{simply-typed } \lambda\text{-calculus} + \text{fixed-point combinator}$$

We adopt a call-by-name evaluation strategy.

PCF types

The (essential) ground type for arithmetic is

$$\mathtt{Nat}$$

with canonical values given by

$$\underline{0}, \ldots, \ \underline{n}, \ \underline{n+1}, \ldots$$

where $\underline{n+1} \equiv \mathrm{succ}(\underline{n})$.

PCF types

Salient (big-step) evaluation in Nat constitutes of

$$\frac{t \Downarrow \underline{n}}{\mathrm{succ}(t) \Downarrow \underline{n+1}}, \quad \frac{t \Downarrow \underline{n+1}}{\mathrm{pred}(t) \Downarrow \underline{n}}, \quad \frac{t \Downarrow \underline{0}}{\mathrm{pred}(t) \Downarrow \underline{0}}.$$

and a test for zero $(t == 0)$.

PCF types

Most importantly, we have this fragment of the type formation rule in BNF

$$\sigma := \mathtt{Nat} \mid \overline{\omega} \mid \Sigma \mid \sigma \times \sigma \mid \sigma \to \tau$$

PCF types

Most importantly, we have this fragment of the type formation rule in BNF

$$\sigma := \mathtt{Nat} \mid \overline{\omega} \mid \Sigma \mid \sigma \times \sigma \mid \sigma \to \tau$$

The canonical values are

$$v := \underline{n} \mid n \mid \top \mid (s, t) \mid \lambda x.t.$$

PCF types

Notice we have introduced two more useful (ornamental) types:

$$\overline{\omega},\ \Sigma$$

PCF types

Notice we have introduced two more useful (ornamental) types:

$$\overline{\omega},\ \Sigma$$

$$\overline{\omega} = 0 < 1 < 2 < \cdots < n < n+1 < \cdots < \infty$$

PCF types

Notice we have introduced two more useful (ornamental) types:

$$\overline{\omega},\ \Sigma$$

$\overline{\omega} = 0 < 1 < 2 < \cdots < n < n+1 < \cdots < \infty$

Σ = observational true $\top$, non-observable false $\bot$

PCF types

For the vertical ordinals $\overline{\omega}$, we have:

PCF types

For the vertical ordinals $\overline{\omega}$, we have:

- the canonical values are n,

PCF types

For the vertical ordinals $\overline{\omega}$, we have:

- the canonical values are n,
- the successor of n, labeled $n+1$, and

PCF types

For the vertical ordinals $\overline{\omega}$, we have:

- the canonical values are n,
- the successor of n, labeled $n + 1$, and
- the predecessor function, and

PCF types

For the vertical ordinals $\overline{\omega}$, we have:

- the canonical values are n,
- the successor of n, labeled $n+1$, and
- the predecessor function, and
- only a test if $n > 0$, but not equality to 0 – meant to be the non-terminating computation in $\overline{\omega}$.

Fixed point combinator

Fixed point combinator works like this

$$\frac{f(\mathrm{fix}(f)) \Downarrow v}{\mathrm{fix}(f) \Downarrow v}$$

and in particular

- $\infty := \mathrm{fix}(\lambda x^{\overline{\omega}}.x + 1)$, and
- $\perp_\sigma := \mathrm{fix}(\mathrm{id}_\sigma)$ which does not evaluate to anything.

Contextual pre-order and equivalence

Definition (PCF contexts)

A context $C[-]$ can be thought of as a program with a hole into which (sub)programs are substituted to form programs.

Contextual pre-order and equivalence

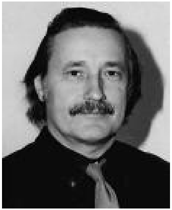
Definition (Contextual pre-order and equivalence)

The contextual pre-order $\sqsubseteq$ is defined as follows:

$$x \sqsubseteq_\sigma y \iff \forall C[-_\sigma] \in \mathrm{Ctx}_\Sigma . C[x] \Downarrow \top \implies C[y] \Downarrow \top.$$

Contextual equivalence, $=$, is the *symmetrization* of $\sqsubseteq$.

Scott-Strachey approach



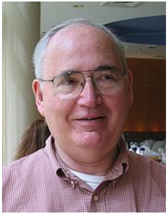


Figure: Christopher Strachey (1916–1975) and Dana Scott (1932–)

Meaning of syntax

Syntax in a programming language has its in-built operational meaning or semantics of evaluation.

Meaning of syntax

Syntax in a programming language has its in-built operational meaning or semantics of evaluation.
However, even the operational semantics is supposed to have some meaning!

Meaning of syntax

Semantics of a programmer/user

- We don't think like computers or processors.

Meaning of syntax

Semantics of a programmer/user

- We don't think like computers or processors.
- We tend to think the way that appeals to us.

Meaning of syntax

Semantics of a programmer/user

- We don't think like computers or processors.
- We tend to think the way that appeals to us.
- We want computer programs to do the things that we want them to.

Meaning of syntax

Semantics of a programmer/user

As mathematicians, we wish to think of programs as functions

Meaning of syntax

Semantics of a programmer/user

As mathematicians, we wish to think of programs as functions

$$f : \sigma \to \tau$$

where $\sigma :=$ source of data input, and
$\tau :=$ range of data output.

Meaning of syntax

This interpretation of programs fits well with the way function is perceived. It is a black box!

$$f = g \iff \forall x \in \sigma . f(x) = g(x).$$

Scott-Strachey's denotational semantics

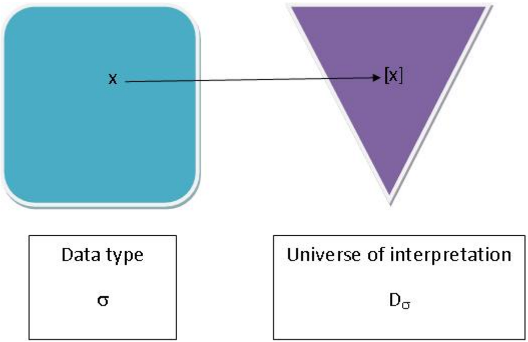


Figure: Strachey-Scott denotational semantics

Scott-Strachey's D.S.

We summarize this approach:

Scott-Strachey's D.S.

We summarize this approach:

- Denotational semantics is rather mathematical and abstract.

Scott-Strachey's D.S.

We summarize this approach:

- Denotational semantics is rather mathematical and abstract.
- Operational semantics is more concrete or closer to the computational intuitions.

Wish list for a denotational model

1. Syntax independence

The denotations of programs should not involve the syntax of the source language.

Wish list for a denotational model

2. Adequacy

All observably distinct programs have distinct denotations.

Wish list for a denotational model

3. Full abstraction

Two programs have the same denotations precisely when they are observationally equivalent.

Denotation equality implies operational equivalence

Computational adequacy easily implies *adequacy*

$$[\![x]\!] = [\![y]\!] \implies x =_\sigma y.$$

Denotation equality implies operational equivalence

Computational adequacy easily implies *adequacy*

$$[\![x]\!] = [\![y]\!] \implies x =_\sigma y.$$

The French school's translation of this results in the terminology

abstract.

Full abstraction

In case the converse holds, i.e.,

$$[\![x]\!] = [\![y]\!] \Longleftarrow x =_\sigma y,$$

we say that the model is *fully* abstract.

Lack of full abstraction

Unfortunately, the Scott model is *not fully abstract*:

$$x =_\sigma y \not\Longrightarrow [\![x]\!] = [\![y]\!].$$

Lack of full abstraction

Implication

Using the Scott model, there is **NO** hope of proving, for instance, that

$$\mathrm{portest}_1 = \mathrm{portest}_2.$$

Lack of full abstraction

Implication

Using the Scott model, there is **NO** hope of proving, for instance, that

$$\mathrm{portest}_1 = \mathrm{portest}_2.$$

The culprit behind f.a. problem: Lack of parallel-or!

A pathological example

Let $C \subseteq \sigma$ be a subset of closed type σ.
It is claimed that a PCF program

$$\texttt{forall}_C : (\sigma \to \Sigma) \to \Sigma$$

does the following operational job:

Job

$$\texttt{forall}_C(p) = \texttt{T} \iff \underbrace{\forall x \in C . p(x) = \texttt{T}}_{\text{job}}.$$

A pathological example

Naive attempt

In the domain of interpretation, it is usually easy to prove this:

$$[\![\mathtt{forall}_C(p)]\!] = \top \iff \forall s \in [\![C]\!].[\![p]\!](s) = \top.$$

A pathological example

Naive attempt

In the domain of interpretation, it is usually easy to prove this:

$$[\![\mathtt{forall}_C(p)]\!] = \top \iff \forall s \in [\![C]\!].[\![p]\!](s) = \top.$$

But why won't this idea work in general?

A pathological example

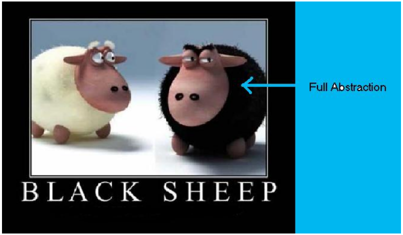


Figure: Don't always blame me!

A pathological example

Consider proving the following (problematic) implication:

$$\forall x \in C.p(x) = \mathtt{T} \implies \mathtt{forall}_C(p) = \mathtt{T}.$$

A pathological example

Any sensible proof strategy should include at least the following chain of implications:

A pathological example

Any sensible proof strategy should include at least the following chain of implications:

$$\begin{array}{ccc} \forall x \in C.p(x) = \mathtt{T} & \Longrightarrow & \mathtt{forall}_C(p) = \mathtt{T} \\ \Downarrow & & \Uparrow \\ \forall s \in [\![C]\!].[\![p]\!](s) = \top & \Longrightarrow & [\![\mathtt{forall}_C(p)]\!] = [\![\mathtt{T}]\!] \end{array}$$

A pathological example

But this will not work in general because

$$\forall x \in C.p(x) = \mathtt{T} \qquad \Longrightarrow \qquad \mathtt{forall}_C(p) = \mathtt{T}$$

A pathological example

But this will not work in general because

$$
\begin{array}{ccc}
\forall x \in C. p(x) = \mathtt{T} & \Longrightarrow & \mathtt{forall}_C(p) = \mathtt{T} \\
\Downarrow \text{ fails} & & \Uparrow \text{ Adequacy} \\
\forall s \in [\![C]\!]. [\![p]\!](s) = \top & \Longrightarrow & [\![\mathtt{forall}_C(p)]\!] = [\![\mathtt{T}]\!]
\end{array}
$$

A pathological example

The problem here is this:

A pathological example

The problem here is this:

$$\forall s \in [\![C]\!].\exists x \in C.s = [\![x]\!]$$

fails to hold in general!

Definability problem

This hiccup I term it as the

definability problem.

Existing literature also calls it an issue of *universality* of the model.

Definability problem

More stubborn problem – doesn't go away easily.

Definability problem

More stubborn problem – doesn't go away easily.

Adding Plotkin's parallel-or por, Scott model becomes fully abstract for PCF^{+}

Definability problem

More stubborn problem – doesn't go away easily.
Adding Plotkin's parallel-or por, Scott model becomes fully abstract for PCF^{+}

but still ...

not every element of the Scott model is definable in the language!

What is this?

Figure: What is this I am saying?

Dilemma: Choice between fish and bear's paw

We want the best of both worlds:

Dilemma: Choice between fish and bear's paw

We want the best of both worlds:

Desirable attributes	**Approach**	**Remarks**

Dilemma: Choice between fish and bear's paw

We want the best of both worlds:

Desirable attributes	**Approach**	**Remarks**
abstract denotation		

Dilemma: Choice between fish and bear's paw

We want the best of both worlds:

Desirable attributes	**Approach**	**Remarks**
abstract denotation	Scott's model	

Dilemma: Choice between fish and bear's paw

We want the best of both worlds:

Desirable attributes	Approach	Remarks
abstract denotation	Scott's model	Beautiful but not f.a.

Dilemma: Choice between fish and bear's paw

We want the best of both worlds:

Desirable attributes	**Approach**	**Remarks**
abstract denotation	Scott's model	Beautiful but not f.a.
free of full abstraction and definability problems		

Dilemma: Choice between fish and bear's paw

We want the best of both worlds:

Desirable attributes	Approach	Remarks
abstract denotation	Scott's model	Beautiful but not f.a.
free of full abstraction and definability problems	Combinatory logic	

Dilemma: Choice between fish and bear's paw

We want the best of both worlds:

Desirable attributes	Approach	Remarks
abstract denotation	Scott's model	Beautiful but not f.a.
free of full abstraction and definability problems	Combinatory logic	Ugly but f.a.

Dilemma: Choice between fish and bear's paw

We want the best of both worlds:

Desirable attributes	**Approach**	**Remarks**
abstract denotation	Scott's model	Beautiful but not f.a.
free of full abstraction and definability problems	Combinatory logic Games semantics	Ugly but f.a.

Dilemma: Choice between fish and bear's paw

We want the best of both worlds:

Desirable attributes	**Approach**	**Remarks**
abstract denotation	Scott's model	Beautiful but not f.a.
free of full abstraction and definability problems	Combinatory logic Games semantics	Ugly but f.a. Nice and f.a.

Dilemma: Choice between fish and bear's paw

We want the best of both worlds:

Desirable attributes	Approach	Remarks
abstract denotation	Scott's model	Beautiful but not f.a.
free of full abstraction and definability problems	Combinatory logic Games semantics	Ugly but f.a. Nice and f.a. but not extensional

Operational domain theory

One possible way is to manufacture a ‘hybrid’ method, i.e.,

Import the domain theoretic tools
directly to the operational world!

Operational domain theory

One possible way is to manufacture a ‘hybrid’ method, i.e.,

Import the domain theoretic tools
directly to the operational world!

The result is ...

An operational domain theory – ODT!

Operational domain theory

To develop a satisfactory ODT, we need a (computationally) natural

Operational domain theory

To develop a satisfactory ODT, we need a (computationally) natural

- pre-order $\sqsubseteq_\sigma$ for each type σ,

Operational domain theory

To develop a satisfactory ODT, we need a (computationally) natural

- pre-order $\sqsubseteq_\sigma$ for each type σ,
- completeness (w.r.t. $\sqsubseteq_\sigma$) holds for each type σ, and

Operational domain theory

To develop a satisfactory ODT, we need a (computationally) natural

- pre-order $\sqsubseteq_\sigma$ for each type σ,
- completeness (w.r.t. $\sqsubseteq_\sigma$) holds for each type σ, and
- notion of continuity for each functional type $\sigma \to \tau$.

Different aspects of ODT

Different aspects of ODT

1. Operational toolkit

Different aspects of ODT

1. Operational toolkit
2. Operational domain

Different aspects of ODT

1. Operational toolkit
2. Operational domain
3. Operational topology

Operational toolkit

This is inspired by existing tools developed by researchers to reason about concurrency in process calculi, e.g., π-calculus.

Operational toolkit

We expect $\sqsubseteq$ to be a typed-indexed family of relations with extensional properties like:

Proposition

$$s \sqsubseteq_\gamma s' \iff (s \Downarrow v \implies s' \Downarrow v)$$

for ground types γ*.*

Operational toolkit

and also like:

Proposition

$$f \sqsubseteq_{\sigma \to \tau} f' \iff \forall x : \sigma . f(x) \sqsubseteq_{\tau} f'(x),$$

Operational toolkit

Modeling after these extensionality properties, define for any type-indexed relation $\mathcal{R}_\sigma$ a new relation $\mathcal{R}'_\sigma$, for instance,

$$s \; \mathcal{R}'_\gamma \; s' \iff (s \Downarrow v \implies s' \Downarrow v)$$

for ground types γ,

Operational toolkit

and

$$f \; \mathcal{R}'_{\sigma\to\tau} \; f' \iff \forall x : \sigma . f(x) \; \mathcal{R}'_{\tau} \; f'(x)$$

for function types $\sigma \to \tau$.

Operational toolkit

Definition

An type-indexed family of relation $\mathcal{R}$ *is called a simulation if* $\mathcal{R} \subseteq \mathcal{R}'$*, and the similarity if it is the largest simulation such that*

$$\mathcal{R} = \mathcal{R}'.$$

Bisimulation (resp. bisimilarity) are *symmetrization* of simulation (resp. similarity).

Operational toolkit: Power tools

Theorem (Operational extensionality theorem)

Operational toolkit: Power tools

Theorem (Operational extensionality theorem)

1. *Contextual pre-order coincides with similarity.*

Operational toolkit: Power tools

Theorem (Operational extensionality theorem)

1. *Contextual pre-order coincides with similarity.*
2. *Contextual equivalence coincides with bisimilarity.*

Operational toolkit: Power tools

Theorem (Co-induction principle)

To prove $s \sqsubseteq_\sigma t$, it is enough to find a simulation $\mathcal{R}$ such that

$$s \ \mathcal{R}_\sigma \ t.$$

Likewise for contextual equivalence, find a corresponding bisimulation.

Power of operational tools

By just using these operational tools, contextual pre-order can be proven to obey:

Proposition (Inequational logic)

$$s \sqsubseteq s' \text{ and } t \sqsubseteq t' \implies s[t/x] \sqsubseteq s'[t'/x].$$

Power of operational tools

By just using these operational tools, contextual pre-order can be proven to obey:

Proposition (Extensionality rules)

$$f \sqsubseteq f' \iff \forall x. f(x) \sqsubseteq f'(x).$$

Power of operational tools

By just using these operational tools, contextual pre-order can be proven to obey:

Proposition (β-rules)

$$(\lambda x.s)t = s[t/x].$$

Power of operational tools

By just using these operational tools, contextual pre-order can be proven to obey:

Proposition (η-rules)

$$f = \lambda x.f(x).$$

Power of operational tools

Definition

Define $\sqsubseteq^{kl}$ is the indexed family of relations given by

$$s \sqsubseteq^{kl}_{\sigma} t \iff \forall v. s \Downarrow v \implies t \Downarrow v.$$

The symmetrization of this yields Kleene equivalence.

Power of operational tools

Proposition

$\sqsubseteq^{kl}$ is a simulation, and hence

$$\sqsubseteq^{kl} \subseteq \sqsubseteq$$

Power of operational tools

Corollary

For any closed term $x : \sigma$,

$$\bot_\sigma \sqsubseteq_\sigma x.$$

That's why $\bot$ is called *bottom*.

Different aspects of ODT

Different aspects of ODT

1. Operational toolkit

Different aspects of ODT

1. Operational toolkit
2. Operational domain

Different aspects of ODT

1. Operational toolkit
2. Operational domain
3. Operational topology

Failure of chain completeness

Take your favorite non-computable sequence of natural numbers

$$\{a_0,\ a_1,\ a_2, \ldots\}.$$

For each $k \in \mathbb{N}$, write a program

```
f_k :: Nat -> Nat
f_k n = if n =< k
           then a_n
           else bot
```

Failure of chain completeness

Proposition

In PCF, the contextual pre-order $\sqsubseteq$ is not chain complete.

Failure of chain completeness

Theorem (Normann 2003)

In PFC, there exists a type τ of the form $\sigma_1 \to \sigma_2 \to \sigma_3$ and a chain of τ elements x_i such that it is unbounded in PCF_Ω but is bounded in PCF^{++}.

Notes

$\mathrm{PCF}_\Omega = \mathrm{PCF} + \mathrm{Oracles}.$
$\mathrm{PCF}^{++} = \mathrm{PCF} + \mathsf{por} + \exists.$

Rational chain completeness

Recall that ...

$$\frac{f(\mathrm{fix}(f)) \Downarrow v}{\mathrm{fix}(f) \Downarrow v}$$

is the operational semantics for the fixed point combinator $\mathrm{fix}(-)$.

Rational chain completeness

Elementary rational chain of programs:

$$\perp \sqsubseteq g(\perp) \sqsubseteq g^{(2)}(\perp) \sqsubseteq g^{(3)}(\perp) \sqsubseteq \ldots$$

Rational chain completeness

Elementary rational chain of programs:

$$\perp \sqsubseteq g(\perp) \sqsubseteq g^{(2)}(\perp) \sqsubseteq g^{(3)}(\perp) \sqsubseteq \ldots$$

Wait!

$$\perp \sqsubseteq g(\perp)$$

Why?

Rational chain completeness

Elementary rational chain of programs:

$$\perp \sqsubseteq g(\perp) \sqsubseteq g^{(2)}(\perp) \sqsubseteq g^{(3)}(\perp) \sqsubseteq \dots$$

Rational chain completeness

Elementary rational chain of programs:

$$\perp \sqsubseteq g(\perp) \sqsubseteq g^{(2)}(\perp) \sqsubseteq g^{(3)}(\perp) \sqsubseteq \ldots$$

has a least upper bound given by the fixed point combinator applied to g:

$$\bigsqcup_{n \in \mathbb{N}} g^{(n)}(\perp) = \mathrm{fix}(g).$$

Rational chain completeness

Definition

Let $g : \tau \to \tau$ and $h : \tau \to \sigma$. Then a chain of the form

$$x_n = h(g^{(n)}(\perp))$$

is defined as a rational chain of programs.

This is the same as a program of type

$$x : \overline{\omega} \to \sigma$$

Rational chain completeness

Theorem (Rational chain completeness)

$$\bigsqcup_{n \in \mathbb{N}} h(g^{(n)}(\perp)) = h(\mathrm{fix}(g)).$$

Proof.

Omitted. $\square$

Central idea

To prove anything about the infinite,

Central idea

To prove anything about the infinite,

1. prove it for every finite part of it, and then

Central idea

To prove anything about the infinite,

1. prove it for every finite part of it, and then
2. take limits.

Finite elements

But what is meant by finite?

Finite elements

Definition ((Rationally) finite element)

An element $b : \sigma$ is called (rationally) finite if every rational chain x_n,

$$b \sqsubseteq_\sigma \bigsqcup x_n \implies \exists n \in \mathbb{N}. b \sqsubseteq_\sigma x_n.$$

We write $K_\sigma := \{x : \sigma \mid x \text{ is finite}\}$.

SFP

Definition (Sequence of Finite Posets: SFP)

1. *A deflation on a type σ is an element of type $\sigma \to \sigma$ which*

SFP

Definition (Sequence of Finite Posets: SFP)

1. *A deflation on a type σ is an element of type $\sigma \to \sigma$ which*
   - *is below id_σ the identity on σ, and*

SFP

Definition (Sequence of Finite Posets: SFP)

1. *A* *deflation* *on a type* σ *is an element of type* $\sigma \to \sigma$ *which*
   - *is below* id_σ *the identity on* σ, *and*
   - *has finite image modulo contextual equivalence.*

SFP

Definition (Sequence of Finite Posets: SFP)

2. *A SFP structure on a type σ is a rational chain of id_n of idempotent deflations with*

$$\bigsqcup_n \mathrm{id}_n^\sigma = \mathrm{id}_\sigma.$$

SFP

Definition (Sequence of Finite Posets: SFP)

3. *A type is SFP if it has an SFP structure.*

SFP

Theorem (SFP theorem)

Every PCF type σ is SFP.

A number of useful corollaries emerge from this theorem.

SFP

Corollary

For any closed term $x : \sigma$,

$$x = \bigsqcup \downarrow x \cap K_\sigma.$$

This tells us that the data types are operationally algebraic domains.

SFP

Corollary

The following statements are equivalent:

1. b *is finite.*
2. *For every rational chain* x_n *with* $b = \bigsqcup x_n$*, there is* $n \in \mathbb{N}$ *such that already* $b = x_n$.

SFP

So we have indeed succeeded in quantifying domain-theoretic finiteness into number-theoretic finiteness.

Corollary

b is finite iff $b = \mathrm{id}_n(b)$ holds for some finite n.

For instance, every total element of the base types `Nat`, Σ, `Bool` are finite.

SFP

Corollary

$f =_{\sigma\to\tau} g$ if and only if $f(b) =_{\tau} g(b)$ for every finite element b.

SFP

For $n \in \mathbb{N}$, define

Definition

$$x =_n y \iff \mathrm{id}_n(x) = \mathrm{id}_n(y).$$

SFP

By far, the most useful fact is this:

Corollary

$$x =_\sigma y \iff \forall n \in \mathbb{N}. x =_n y.$$

Different aspects of ODT

Different aspects of ODT

1. Operational toolkit

Different aspects of ODT

1. Operational toolkit
2. Operational domain

Different aspects of ODT

1. Operational toolkit
2. Operational domain
3. Operational topology

Domain theory

Domain theory can in fact be seen as topology of partial orders. So,

Maxim

no topology = no domain theory.

Domain theory

Many preceding works have pointed towards the use of 'topology' in computation:

Domain theory

Many preceding works have pointed towards the use of ‘topology’ in computation:

- M. Smyth: open set to express ‘an observable/affirmative predicate’.

Domain theory

Many preceding works have pointed towards the use of 'topology' in computation:

- M. Smyth: open set to express 'an observable/affirmative predicate'.
- S. Vickers: locales to express geometric logic.

Domain theory

Many preceding works have pointed towards the use of 'topology' in computation:

- M. Smyth: open set to express 'an observable/affirmative predicate'.
- S. Vickers: locales to express geometric logic.
- S. Abramsky: Stone-duality to express program logic.

Domain theory

Many preceding works have pointed towards the use of ‘topology’ in computation:

- M. Smyth: open set to express ‘an observable/affirmative predicate’.
- S. Vickers: locales to express geometric logic.
- S. Abramsky: Stone-duality to express program logic.
- Yu. Ershov: continuous maps to express computability.

Some photos



Figure: Some famous people in domains and semantics

Continuous maps

Following Ershov's ideas, one can start with a very natural definition:

Definition

A function $f : \sigma \to \tau$ is continuous if it is definable in the language.

Continuous maps

The definability of functions dictates the nature of the hierarchy of 'topologies' on types!

Continuous maps

The definability of functions dictates the nature of the hierarchy of ‘topologies’ on types!

Maxim

Functions are first-class citizens in functional programming paradigm.

Opens

Next question: What are then the open sets?

Opens

Next question: What are then the open sets?

Definition ((Operational) opens)

A subset $U \subseteq \sigma$ is (operationally) open in type σ if its characteristic function $\chi_U : \sigma \to \Sigma$ is continuous.

Note that

$$\chi_U(x) = \top \iff x \in U.$$

Opens as semi-decidable sets

Open subsets of a data type is precisely the *semi-decidable* (with respect to the language) subsets of that data type.

Opens as semi-decidable sets

Open subsets of a data type is precisely the *semi-decidable* (with respect to the language) subsets of that data type.
Clearly, conjunction of two semi-decisions is still a semi-decision.

Opens aren't really opens

Proposition

The opens of a data type do not form a topology!

Opens aren't really opens

Proposition

The opens of a data type do not form a topology!

Proof.

Otherwise, (weak) parallel-or is PCF-definable. $\square$

Opens aren't really opens

Here a weak parallel-or $\vee$ means

$$p \vee q = \top \iff p = \top \text{ or } q = \top.$$

Opens aren't really opens

So operational topology isn't a topology!

Opens aren't really opens

So operational topology isn't a topology!
But it behaves very much like one.

Specialization order

Proposition

For any $x,\ y : \sigma$,

$$x \sqsubseteq_\sigma y \iff \forall \text{ open } U. x \in U \implies y \in U.$$

The contextual pre-order coincides with the specialization order induced by operational opens.

Specialization order

Proposition

All opens are upper with respect to the contextual pre-order.

Specialization order

Proposition

All opens are upper with respect to the contextual pre-order.

Proof.

By definition! $\square$

Specialization order

Proposition

All continuous functions are monotone.

Corollary (Turing's Halting Problem)

There is no continuous function $f : \Sigma \to \Sigma$ *such that*

$$f(\bot) = \top \;\&\; f(\top) = \bot.$$

Specialization order

Proposition

All continuous functions are monotone.

Corollary (Turing's Halting Problem)

There is no continuous function $f : \Sigma \to \Sigma$ *such that*

$$f(\bot) = \top \;\&\; f(\top) = \bot.$$

Data types as d-spaces

Operational opens are not just ‘opens’. They are somewhat intrinsic to the operational semantics of the language!

Data types as d-spaces

Because the intrinsic topology on types is the Scott topology, we expect the operational opens to behave like Scott-open sets.

Analogy

directed complete $\rightsquigarrow$ rational-chain complete

Scott open $\rightsquigarrow$?

Data types as d-spaces

Proposition

Opens are operationally Scott open in the sense that they are

Data types as d-spaces

Proposition

Opens are operationally Scott open in the sense that they are

1. *upper with respect to $\sqsubseteq$, and*
2. *inaccessible by joins of rational chains.*

Data types as d-spaces

Recall that a *d-space* is a topological space in which every open set is Scott-open. These are also known as *monotone convergence spaces*.

Data types as d-spaces

Recall that a *d-space* is a topological space in which every open set is Scott-open. These are also known as *monotone convergence spaces*.

So from the above result, we have shown that every data type is an operational d-space.

Data types as algebraic domains

Theorem

The following are equivalent:

Data types as algebraic domains

Theorem

The following are equivalent:

1. *b is finite.*

Data types as algebraic domains

Theorem

The following are equivalent:

1. b *is finite.*
2. $\uparrow b$ *is open.*

Data types as algebraic domains

Corollary

Every open set is the union of open sets of the form $\uparrow b$ with b finite.

In other words, the sets $\uparrow b$ with b finite forms a basis for the operational topology.

Conclusion

In today's talk, I have

Conclusion

In today's talk, I have

1. introduced the Scott model of PCF and other functional language variants

Conclusion

In today's talk, I have

1. introduced the Scott model of PCF and other functional language variants
2. explained the Full Abstraction and Definability Problems

Conclusion

In today's talk, I have

1. introduced the Scott model of PCF and other functional language variants
2. explained the Full Abstraction and Definability Problems
3. suggested ODT as an alternative ideal semantics for PCF

Conclusion

In today's talk, I have

1. introduced the Scott model of PCF and other functional language variants
2. explained the Full Abstraction and Definability Problems
3. suggested ODT as an alternative ideal semantics for PCF
4. showed promising signs offered by the operational toolkit

Conclusion

In today's talk, I have

1. introduced the Scott model of PCF and other functional language variants
2. explained the Full Abstraction and Definability Problems
3. suggested ODT as an alternative ideal semantics for PCF
4. showed promising signs offered by the operational toolkit
5. introduced the operational domain theory and topology of a functional sequential language, and

Conclusion

In today's talk, I have

1. introduced the Scott model of PCF and other functional language variants
2. explained the Full Abstraction and Definability Problems
3. suggested ODT as an alternative ideal semantics for PCF
4. showed promising signs offered by the operational toolkit
5. introduced the operational domain theory and topology of a functional sequential language, and
6. demonstrated the interplay between these.

References

References

1. M.H. Escardó. *Synthetic topology of data types and classical spaces*. ENTCS, 87, pp. 21–156, 2004.

References

1. M.H. Escardó. *Synthetic topology of data types and classical spaces*. ENTCS, 87, pp. 21–156, 2004.
2. M.H. Escardó & W.K. Ho. *An operational domain theory and topology of sequential languages*. Information and Computation, 207(3), pp. 411–437, 2009.

References

1. M.H. Escardó. *Synthetic topology of data types and classical spaces*. ENTCS, 87, pp. 21–156, 2004.
2. M.H. Escardó & W.K. Ho. *An operational domain theory and topology of sequential languages*. Information and Computation, 207(3), pp. 411–437, 2009.
3. W.K. Ho. *Operational Domain Theory and Topology of Sequential Functional Languages*. PhD Thesis, School of Computer Science, University of Birmingham, Oct 2006.

All these can be downloaded from my webpage at `http://math.nie.edu.sg/wkho/pubtalk.htm`.