Programs, Domains, and Logic

Achim Jung

April 15, 2015

I. Recursive functions

II. Scott's Domain Theory

III. Denotational Semantics

IV. From algebraic to continuous domains

V. From continuous domains to topology

Primitive recursive functions

A scheme for defining functions from $\mathbb{N}^k$ to $\mathbb{N}$.

Basic functions:

— constant function $c_0 : \mathbb{N}^0 \to \mathbb{N}, () \mapsto 0$

— successor function $s : \mathbb{N} \to \mathbb{N}, (x) \mapsto x + 1$

— projections $\pi_i^k : \mathbb{N}^k \to \mathbb{N}, (x_1, \ldots, x_k) \mapsto x_i$

Constructions:

— composition: If $g_1, \ldots, g_k : \mathbb{N}^l \to \mathbb{N}$ and $h : \mathbb{N}^k \to \mathbb{N}$ are primitive recursive, then so is $h \circ (g_1, \ldots, g_k) : \mathbb{N}^l \to \mathbb{N}$.

— primitive recursion: If $f : \mathbb{N}^k \to \mathbb{N}$ and $g : \mathbb{N}^{k+2} \to \mathbb{N}$ are primitive recursive, then so is $h : \mathbb{N}^{k+1} \to \mathbb{N}$, given by

$$\begin{aligned} h(0, x_1, \ldots, x_k) &:= f(x_1, \ldots, x_k) \\ h(s(y), x_1, \ldots, x_k) &:= g(y, h(y, x_1, \ldots, x_k), x_1, \ldots, x_k) \end{aligned}$$

μ-recursion

To primitive recursion add:

— minimisation: If $f : \mathbb{N}^{k+1} \to \mathbb{N}$ is a μ-recursive function, then so is $\mu(f) : \mathbb{N}^k \to \mathbb{N}$, where

$$\mu(f)(x_1, \ldots, x_k) \quad := \quad \text{the smallest } z \text{ such that } f(z, x_1, \ldots, x_k) = 0 \text{ if such a } z \text{ exists and undefined otherwise}$$

Where it all started

D. S. Scott. A type theoretic alternative to ISWIM, CUCH, OWHY. Manuscript, University of Oxford, 1969

> *[...] probably the most well-known unpublished manuscript in Programming Language Theory.*
> *(Gunter 1992)*

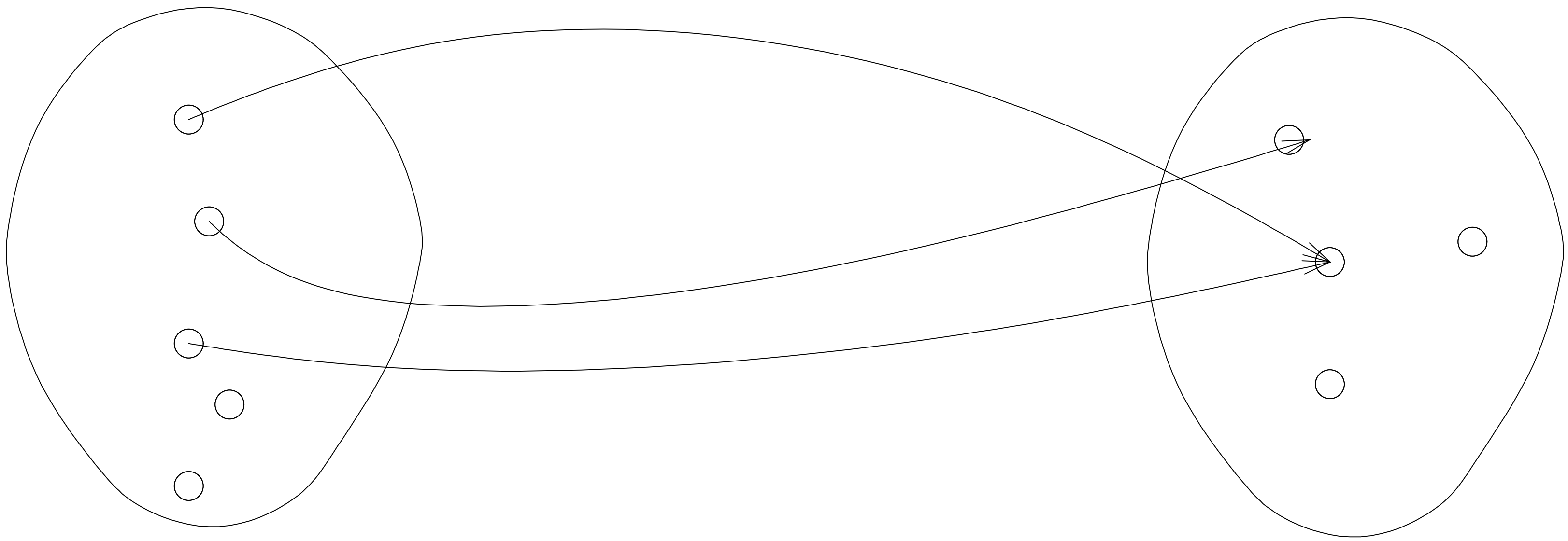
Where it all started

D. S. Scott. A type theoretic alternative to ISWIM, CUCH, OWHY. Manuscript, University of Oxford, 1969

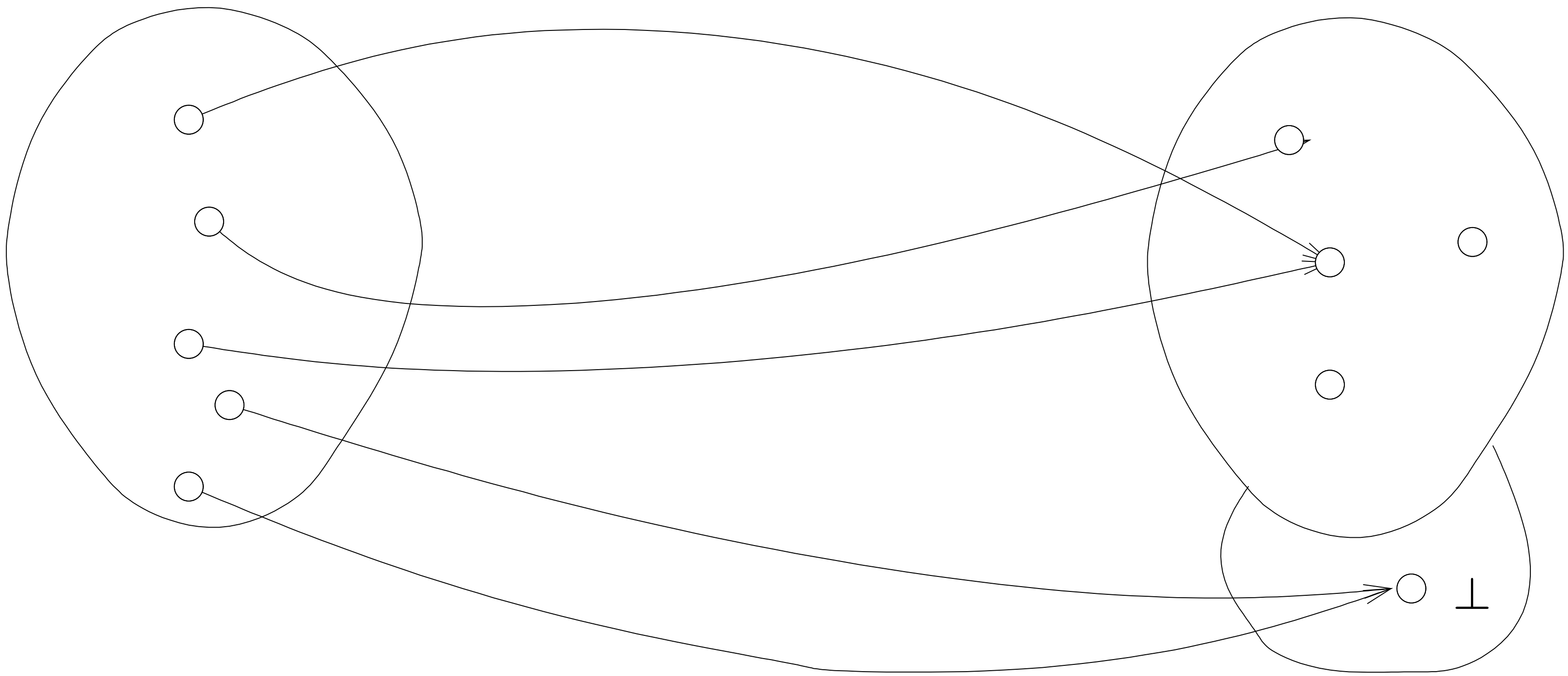
> *[...] probably the most well-known unpublished manuscript in Programming Language Theory.*
> *(Gunter 1992)*

D. S. Scott. A type-theoretical alternative to ISWIM, CUCH, OWHY. *Theoretical Computer Science*, 121:411–440, 1993

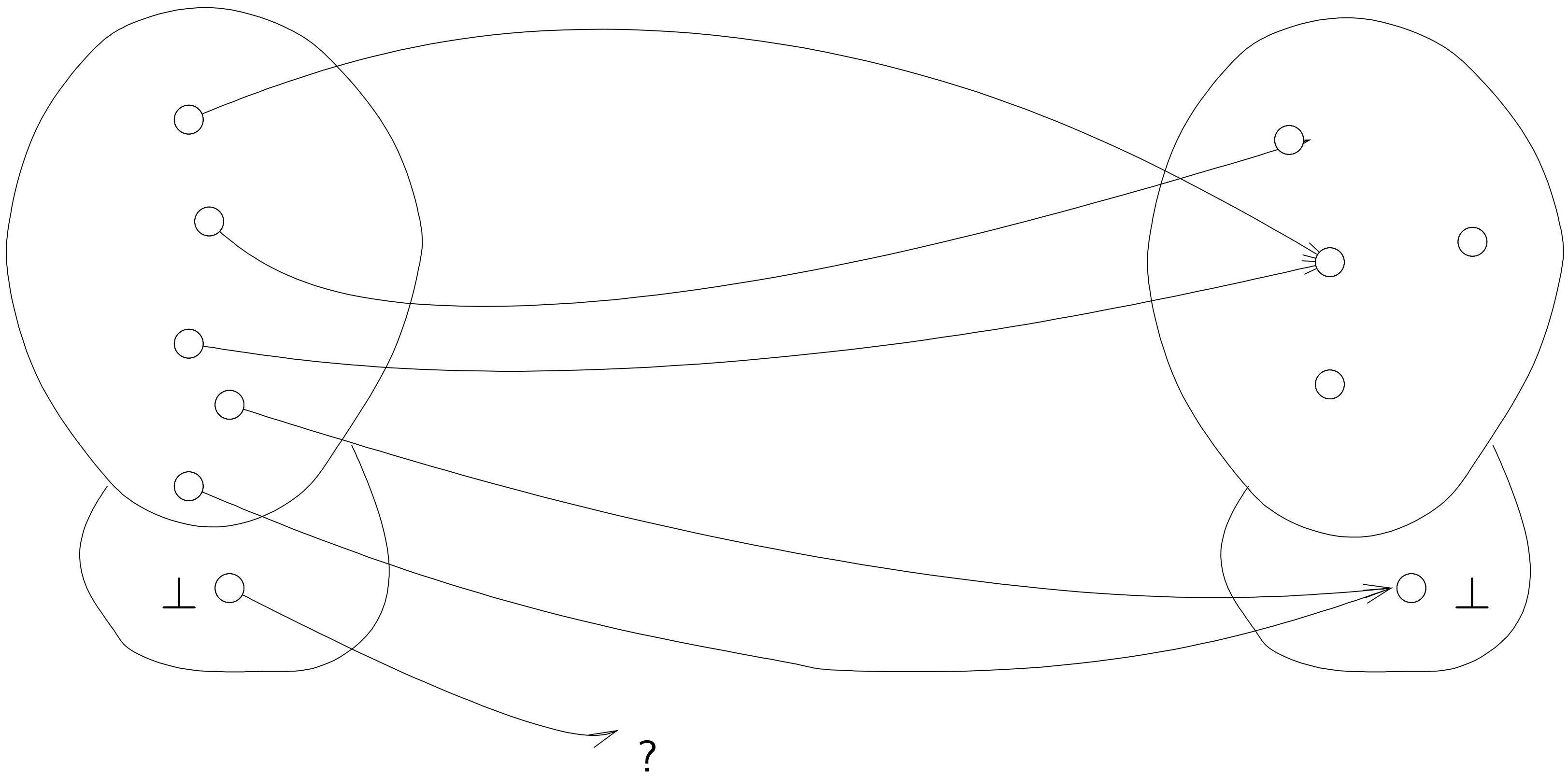
From partial functions to domains



From partial functions to domains



From partial functions to domains



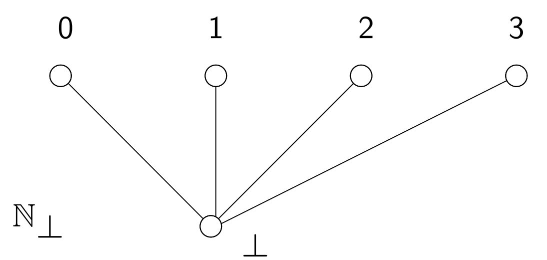
What to do with $\bot$?

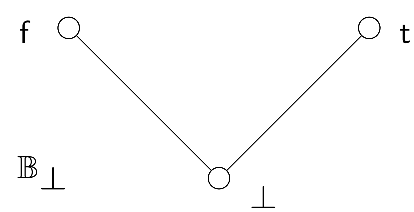
- Allow it to be mapped arbitrarily?
- Always map it to $\bot$?

What to do with $\bot$?

- Allow it to be mapped arbitrarily?
- Always map it to $\bot$?

Scott's proposal: Introduce an order relation in which $\bot$ is smaller than every other element:

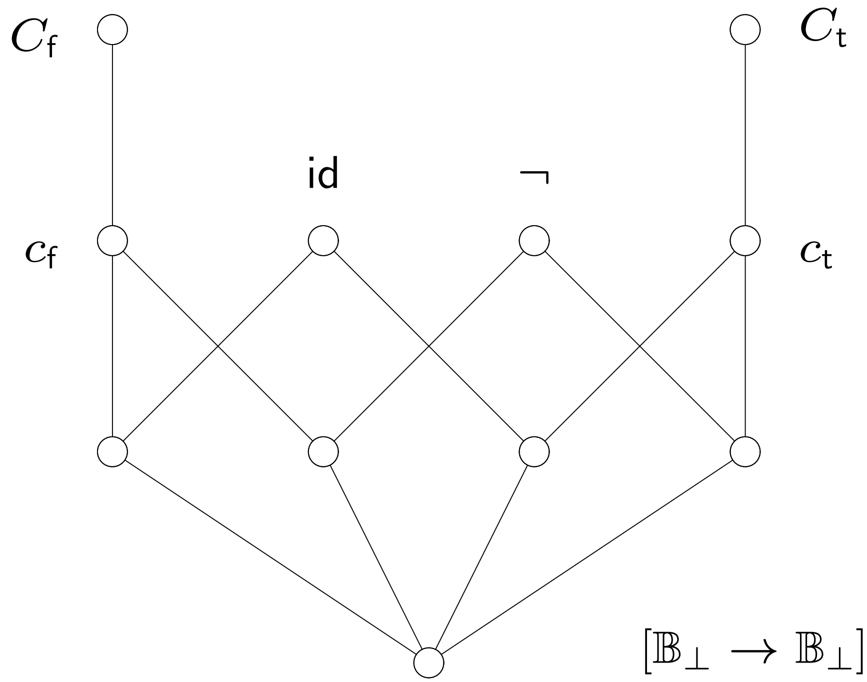




and stipulate that functions have to preserve the order.

An example function space: $[\mathbb{B}_\perp \to \mathbb{B}_\perp]$

Instead of $3^3 = 27$ many elements we get 11:



NB: This contains the $2^2 = 4$ many elements of $[\mathbb{B} \to \mathbb{B}]$, or the $3^2 = 9$ elements of $[\mathbb{B} \rightharpoonup \mathbb{B}]$.

What's the difference between c_{f} and C_{f}?

$$c_{\mathsf{f}} : \begin{array}{ccc} \mathsf{t} & \mapsto & \mathsf{f} \\ \mathsf{f} & \mapsto & \mathsf{f} \\ \bot & \mapsto & \bot \end{array} \qquad C_{\mathsf{f}} : \begin{array}{ccc} \mathsf{t} & \mapsto & \mathsf{f} \\ \mathsf{f} & \mapsto & \mathsf{f} \\ \bot & \mapsto & \mathsf{f} \end{array}$$

What's the difference between c_{f} and C_{f}?

$$c_{\mathsf{f}}: \begin{array}{ccc} \mathsf{t} & \mapsto & \mathsf{f} \\ \mathsf{f} & \mapsto & \mathsf{f} \\ \bot & \mapsto & \bot \end{array} \qquad C_{\mathsf{f}}: \begin{array}{ccc} \mathsf{t} & \mapsto & \mathsf{f} \\ \mathsf{f} & \mapsto & \mathsf{f} \\ \bot & \mapsto & \mathsf{f} \end{array}$$

```
bool c_false(bool x) {if (x == true)  return false else
                                      return false;}
```

```
bool C_false(bool x) {return false;}
```

What's the difference between c_{f} and C_{f}?

$$c_{\mathsf{f}}: \begin{array}{ccc} \mathsf{t} & \mapsto & \mathsf{f} \\ \mathsf{f} & \mapsto & \mathsf{f} \\ \bot & \mapsto & \bot \end{array} \qquad C_{\mathsf{f}}: \begin{array}{ccc} \mathsf{t} & \mapsto & \mathsf{f} \\ \mathsf{f} & \mapsto & \mathsf{f} \\ \bot & \mapsto & \mathsf{f} \end{array}$$

```
bool c_false(bool x) {if (x == true)  return false else
                                      return false;}
```

```
bool C_false(bool x) {return false;}
```

Do we care about such distinctions?

A more complicated example: $[\mathbb{N}_\perp \to \mathbb{N}_\perp]$

As before:

$[\mathbb{N}_\perp \to \mathbb{N}_\perp]$ contains all of $[\mathbb{N} \to \mathbb{N}]$ or $[\mathbb{N} \rightharpoonup \mathbb{N}]$, so - in particular - has uncountably many elements.

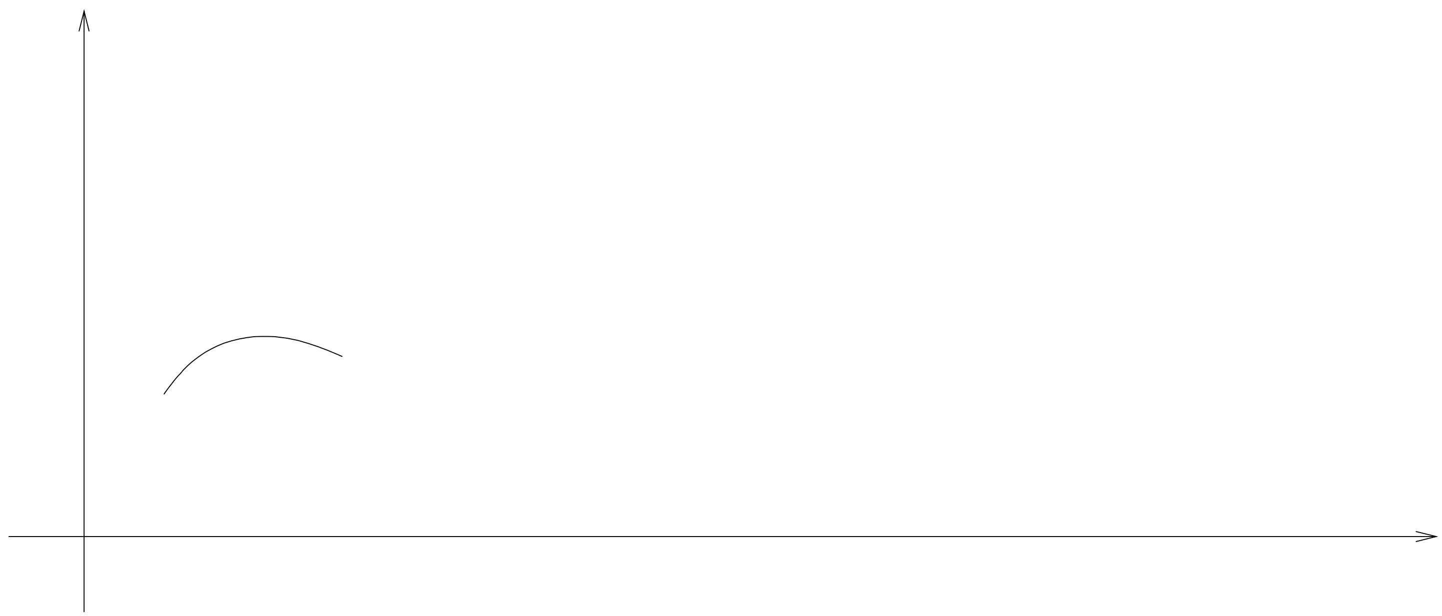
A more complicated example: $[\mathbb{N}_\perp \to \mathbb{N}_\perp]$

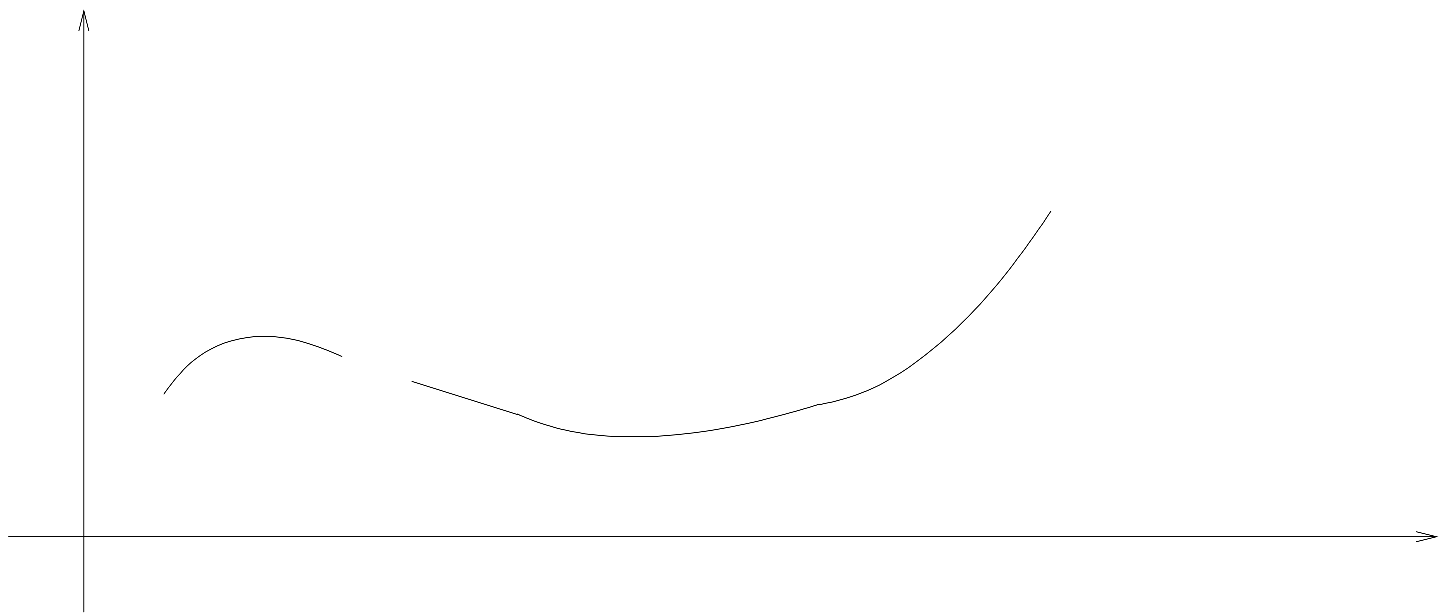
As before:

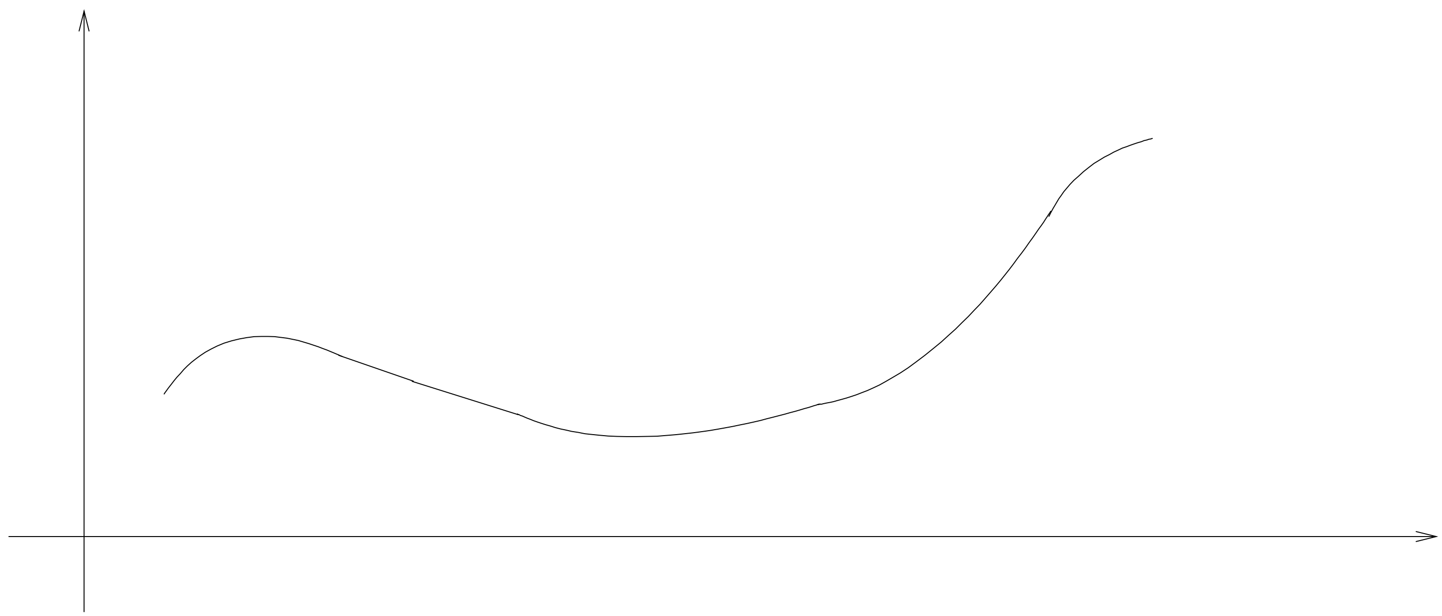
$[\mathbb{N}_\perp \to \mathbb{N}_\perp]$ contains all of $[\mathbb{N} \to \mathbb{N}]$ or $[\mathbb{N} \rightharpoonup \mathbb{N}]$, so - in particular - has uncountably many elements.

It also contains infinite ascending chains:

$$\boxed{\begin{array}{ccc} 0 & \mapsto & 0 \end{array}} \sqsubseteq \boxed{\begin{array}{ccc} 0 & \mapsto & 0 \\ 1 & \mapsto & 1 \end{array}} \sqsubseteq \boxed{\begin{array}{ccc} 0 & \mapsto & 0 \\ 1 & \mapsto & 1 \\ 2 & \mapsto & 4 \end{array}} \sqsubseteq \boxed{\begin{array}{ccc} 0 & \mapsto & 0 \\ 1 & \mapsto & 1 \\ 2 & \mapsto & 4 \\ 3 & \mapsto & 9 \end{array}} \sqsubseteq \ \ldots$$







Scott's thesis

How big is $[[\mathbb{N}_\perp \to \mathbb{N}_\perp] \to \mathbb{N}_\perp$?

Scott's thesis

How big is $[[\mathbb{N}_\perp \to \mathbb{N}_\perp] \to \mathbb{N}_\perp$?

Scott observed that a terminating computation of type
`(nat -> nat) -> nat`
can query its argument only finitely often.

Thus only a finite part of the graph of the argument is needed.

This was known to recursion theorists as the Myhill-Shepardson Theorem.

Scott's thesis

How big is $[[\mathbb{N}_\perp \to \mathbb{N}_\perp] \to \mathbb{N}_\perp$?

Scott observed that a terminating computation of type
`(nat -> nat) -> nat`
can query its argument only finitely often.

Thus only a finite part of the graph of the argument is needed.

This was known to recursion theorists as the Myhill-Shepardson Theorem.

Scott formulated and generalized this as follows:

If f is computable and if the input is of the form $\bigsqcup^\uparrow x_i$ then the output $f(x)$ can be computed as $\bigsqcup^\uparrow f(x_i)$.

Furthermore, this should hold at all types.

Domains

Definition. *A Scott domain is an ordered structure $\langle D; \sqsubseteq \rangle$ such that*

- *there is a smallest element $\bot$;*
- *sups of chains (directed sets) always exist;*
- *every element is the sup of finite elements;*
- *there are only countably many finite elements;*
- *sups of bounded sets exist.*

Morphisms are Scott-continuous functions: $f(\bigsqcup^{\uparrow}_{i \in I} x_i) = \bigsqcup^{\uparrow}_{i \in I} f(x_i)$

The category Scott

- **Scott** is closed under many constructions: lifting, product, function space ("cartesian closed")

- All functions have fixpoints: fix: $[D \to D] \to D$ such that $f(\mathrm{fix} f) = \mathrm{fix} f$

- fix is a morphism in **Scott**

- Functors **Scott** $\to$ **Scott** have fixpoints ("domain equations")

NB: It cannot be closed under sums but there is a reasonable alternative.

The category Scott

- **Scott** is closed under many constructions: lifting, product, function space ("cartesian closed")

- All functions have fixpoints: fix: $[D \to D] \to D$ such that $f(\text{fix} f) = \text{fix} f$

- fix is a morphism in **Scott**

- Functors **Scott** $\to$ **Scott** have fixpoints ("domain equations")

NB: It cannot be closed under sums but there is a reasonable alternative.

This is Scott's "type-theoretic alternative to ISWIM, CUCH, OWHY"

I. Recursive functions

II. Scott's Domain Theory

III. Denotational Semantics

IV. From algebraic to continuous domains

V. From continuous domains to topology

The idea of a pure functional language

- Choose a semantic category of *types*

The idea of a pure functional language

- Choose a semantic category of *types*
- Choose some elementary types and type constructors

The idea of a pure functional language

- Choose a semantic category of *types*
- Choose some elementary types and type constructors
- Give names to some elementary morphisms

The idea of a pure functional language

- Choose a semantic category of *types*
- Choose some elementary types and type constructors
- Give names to some elementary morphisms
- Choose (and name) mechanisms for combining morphisms

The idea of a pure functional language

- Choose a semantic category of *types*
- Choose some elementary types and type constructors
- Give names to some elementary morphisms
- Choose (and name) mechanisms for combining morphisms
- Decide on rewrite rules for the combinators

Example: μ-recursive functions

- category of types: sets and partial functions
- elementary types: $\mathbb{N}^k$
  type constructors: none
- elementary morphisms: 0, s, π_i
- combinators: composition, primitive recursion, μ-recursion
- rewrite rules: defining equations:

$$h(n, x_1, \ldots, x_k) \longrightarrow \begin{array}{l} \text{if } n = 0 \\ \text{then } f(x_1, \ldots, x_k) \\ \text{else } g(n-1, h(y, x_1, \ldots, x_k), x_1, \ldots, x_k) \end{array}$$

PCF (Programming computable functions)

G. D. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5:223–255, 1977

Category of types: **Scott**

Elementary types and type constructors:

- $\mathbb{B}_\perp$ for the booleans
- $\mathbb{N}_\perp$ for the natural numbers
- $[\cdot \to \cdot]$ as the only constructor

Elementary morphisms

- t and f in $\mathbb{B}_\perp$
- $\underline{0}$, $\underline{1}$, $\underline{2}$, $\underline{3}$, etc., in $\mathbb{N}_\perp$
- succ and pred in $\mathbb{N}_\perp \to \mathbb{N}_\perp$
- if_σ in $\mathbb{B}_\perp \to \sigma \to \sigma \to \sigma$ for every (definable) type σ
- fix_σ in $[\sigma \to \sigma] \to \sigma$ for every (definable) type σ

Combinators

Could use K and S, known from combinatory logic.

Alternatively: lambda abstraction and application.

This works because we have a cartesian closed category.

Rewrite rules

$$\text{if } \mathsf{t}\ M\ \ N \longrightarrow M \qquad\qquad \frac{C \longrightarrow C'}{\text{if } C\ M\ N \longrightarrow \text{if } C'\ M\ N}$$

$$\text{fix}\ M \longrightarrow M(\text{fix}\ M)$$

$$(\lambda x.\, M)\, N \longrightarrow M[x := N]$$

etc. etc.

Expressivity

Enough elementary types and type constructors?

Theorem. *Every Scott domain is a retract of* $\mathbb{N}_\perp \to \mathbb{N}_\perp$

Enough primitive functions and constructors?

Theorem. *Every partial recursive function from* $\mathbb{N}$ *to* $\mathbb{N}$ *is programmable in PCF.*

Enough rewrite rules?

Theorem. *If a program expression* P *denotes* $n \in \mathbb{N}$ *then* P *can be rewritten to* $\underline{n}$ *in finitely many steps ("adequacy").*

Expressivity — the bad news

Fact. *Even the finite domain* $\mathbb{B}_\perp^2 \to \mathbb{B}_\perp \cong [\mathbb{B}_\perp \to [\mathbb{B}_\perp \to \mathbb{B}_\perp]]$ *contains elements which are not denoted by a term of PCF (as already noted by Scott).*

Expressivity — the bad news

Fact. *Even the finite domain* $\mathbb{B}_\perp^2 \to \mathbb{B}_\perp \cong [\mathbb{B}_\perp \to [\mathbb{B}_\perp \to \mathbb{B}_\perp]]$ *contains elements which are not denoted by a term of PCF (as already noted by Scott).*

Fact. *Consequently, there are differences in the model which are "invisible" to the programming language.*

Expressivity — the bad news

Fact. *Even the finite domain* $\mathbb{B}_\perp^2 \to \mathbb{B}_\perp \cong [\mathbb{B}_\perp \to [\mathbb{B}_\perp \to \mathbb{B}_\perp]]$ *contains elements which are not denoted by a term of PCF (as already noted by Scott).*

Fact. *Consequently, there are differences in the model which are "invisible" to the programming language.*

This is the "Full Abstraction Problem".

Expressivity — the bad news

Fact. *Even the finite domain* $\mathbb{B}_\perp^2 \to \mathbb{B}_\perp \cong [\mathbb{B}_\perp \to [\mathbb{B}_\perp \to \mathbb{B}_\perp]]$ *contains elements which are not denoted by a term of PCF (as already noted by Scott).*

Fact. *Consequently, there are differences in the model which are "invisible" to the programming language.*

This is the "Full Abstraction Problem".

Two directions for a solution:

1) Make the language richer

2) Restrict the model

History of the "Full Abstraction Problem"

Some contributors:

Plotkin, Abramsky, Berry, Brookes, Bucciarelli, Cartwright, Curien, Ehrhard, Felleisen, Geva, Girard, Hyland, Jagadeesan, Jung, Lévy, Longley, Kahn, Malacaria, Meyer, Milner, Mulmuley, Nickau, O'Hearn, Ong, Riecke, Sazonov, Sieber, Stoughton, Streicher, Tiuryn, Winskel

One solution

A. Jung and J. Tiuryn. A new characterization of lambda definability. In M. Bezem and J. F. Groote, editors, *Typed Lambda Calculi and Applications*, volume 664 of *Lecture Notes in Computer Science*, pages 245–257. Springer Verlag, 1993

P. W. O'Hearn and J. G. Riecke. Kripke logical relations and PCF. *Information and Computation*, 120(1):107–116, 1995

but here there are infinitely many conditions imposed on the model. Can one do better?

One solution

A. Jung and J. Tiuryn. A new characterization of lambda definability. In M. Bezem and J. F. Groote, editors, *Typed Lambda Calculi and Applications*, volume 664 of *Lecture Notes in Computer Science*, pages 245–257. Springer Verlag, 1993

P. W. O'Hearn and J. G. Riecke. Kripke logical relations and PCF. *Information and Computation*, 120(1):107–116, 1995

but here there are infinitely many conditions imposed on the model. Can one do better?

Theorem. [Loader 1996] *It is not decidable which finitary elements of* **Scott** *are PCF-definable.*

Th. Streicher. *Domain-Theoretic Foundations of Functional Programming*. World Scientific, 2006. 132pp

Nonetheless, good things have happened

- Scott's original proposal ("LCF") led to proof assistants HOL and Isabelle
- PCF led to functional programming languages ML (OCaml) and Haskell
- The Full Abstraction Problem led to Linear Logic and Game Semantics
- Game semantics has led to verifiers and compilers (Ghica)

I. Recursive functions

II. Scott's Domain Theory

III. Denotational Semantics

IV. From algebraic to continuous domains

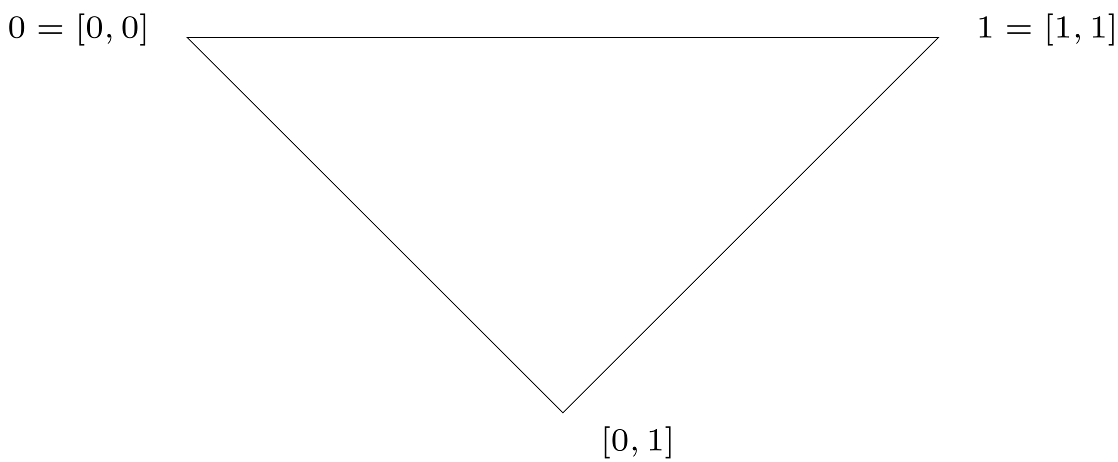
V. From continuous domains to topology

Exact real-number computation

M. H. Escardó. PCF extended with real numbers. *Theoretical Computer Science*, 162:79–115, 1996

A new domain

Neither the unit interval $[0, 1]$ nor the real line $\mathbb{R}$ are Scott domains. We replace this with the interval domain $\mathbb{I} = \{[a, b] \mid 0 \leq a \leq b \leq 1\}$, ordered by reversed inclusion.



This has all the properties of Scott domains except that there are no "finite elements" apart from $\perp = [0, 1]$.

Scott domains

Definition.

A Scott domain *is an ordered structure* $\langle D; \sqsubseteq \rangle$ *such that*

- *there is a smallest element* $\perp$*;*
- *sups of chains (directed sets) always exist;*
- *every element is the sup of* finite *elements;*
- *there are only countably many finite elements;*
- *sups of bounded sets exist.*

category **Scott**

Continuous Scott domains

Definition.

A continuous Scott domain is an ordered structure $\langle D; \sqsubseteq \rangle$ *such that*

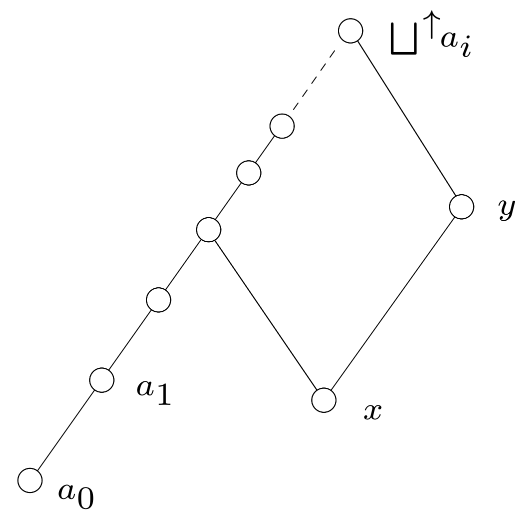
- *there is a smallest element* $\perp$*;*
- *sups of chains (directed sets) always exist;*
- *every element is the sup of relatively finite elements;*
- *there is some countable basis;*
- *sups of bounded sets exist.*

category **contScott**

The way-below relation

An element x is way-below another element y if you "can't reach y without passing x:"

$$y \sqsubseteq \bigsqcup_{i \in I}^{\uparrow} a_i \quad \Rightarrow \quad x \sqsubseteq a_i \text{ for some } i \in I$$



One writes $x \ll y$ if this is the case.

The way-below relation on $\mathbb{I}$

An interval $[a, b]$ is way-below $[a', b']$ if and only if $a < a'$ and $b' < b$, in other words, if $[a, b]$ is a neighbourhood of $[a', b']$.

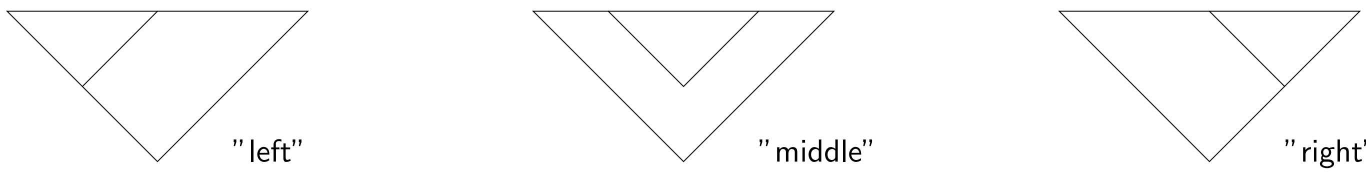
This hints at a connection between Domain Theory and classical mathematics (topology) that Dana Scott and a group of mathematicians explored in detail in the 1970s.

G. Gierz, K. H. Hofmann, K. Keimel, J. D. Lawson, M. Mislove, and D. S. Scott. *A Compendium of Continuous Lattices*. Springer Verlag, 1980

G. Gierz, K. H. Hofmann, K. Keimel, J. D. Lawson, M. Mislove, and D. S. Scott. *Continuous Lattices and Domains*, volume 93 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 2003

Escardó's RealPCF

Much is as in PCF but we now work in **contScott**, so $\mathbb{I}$ is one of the objects available to us. We take as primitive the following three re-scaling functions:



Some findings

Theorem. *RealPCF is expressive: every computable function* $\mathbb{I} \to \mathbb{I}$ *is programmable (but parallel constructs are necessary).*

Theorem. *RealPCF's rewrite rules are* adequate*.*

Probabilistic computation

Now imagine that we want the language to contain a probabilistic operator, such as

```
choose 0.5 M N
```

which evaluates as `M` or `N`, each with probability $1/2$.

There are no "maps" for this in either the category **Scott** or **contScott**.

Probabilistic computation

Now imagine that we want the language to contain a probabilistic operator, such as

```
choose 0.5 M N
```

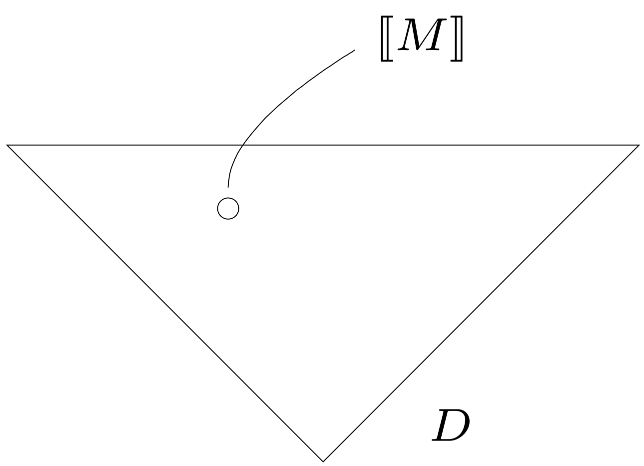
which evaluates as `M` or `N`, each with probability $1/2$.

There are no "maps" for this in either the category **Scott** or **contScott**.

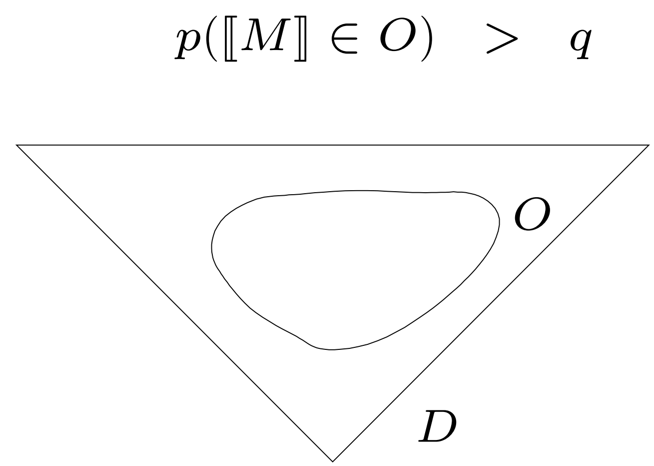
Solution. Add a monad P to the categorical structure and interpret a term of type $\sigma \to \tau$ as a morphism $P\sigma \to P\tau$.

In our case, we expect P to capture probabilistic information over a domain D, i.e., instead of speaking of a fixed value in D, we say what the probability of the value belonging to a set O is.

Probabilistic semantics

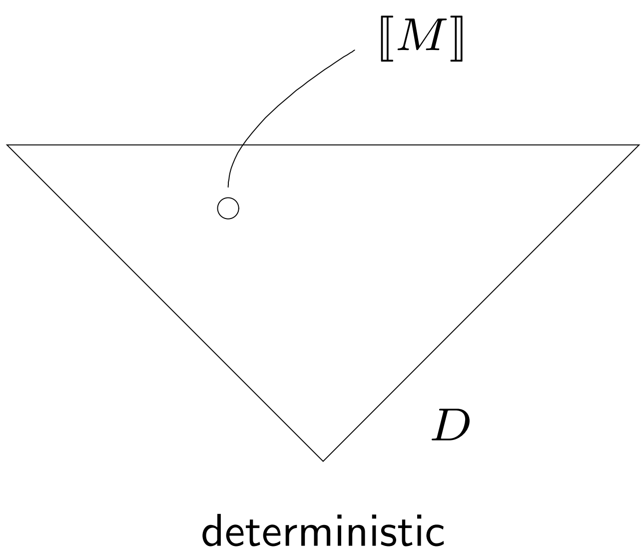


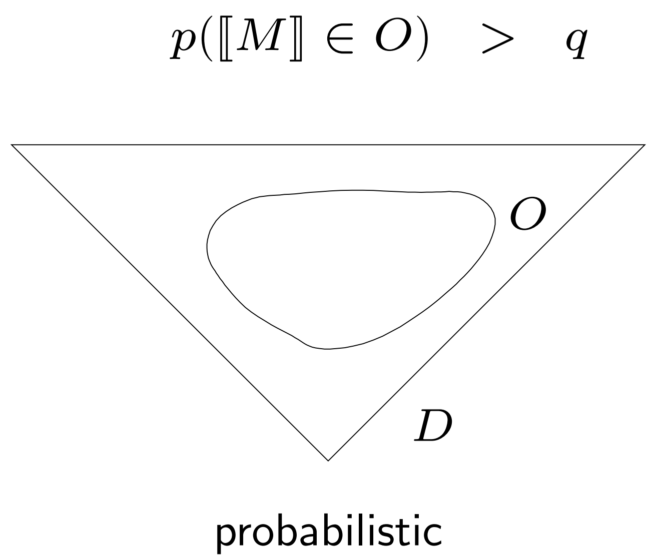
deterministic



probabilistic

Probabilistic semantics





J. Goubault-Larrecq. Full abstraction for non-deterministic and probabilistic extensions of PCF I — the angelic cases. *Journal of Logic and Algebraic Programming*, 2015. To appear

Enter the Scott topology

Definition. *A set U of a domain is Scott-open if it is upwards closed and unreachable by directed suprema:*

$$\bigsqcup_{i \in I}^{\uparrow} a_i \in U \quad \Rightarrow \quad a_i \in U \text{ for some } i \in I$$

Fact. *Topologically continuous functions between domains are precisely the Scott-continuous ones.*

Valuations

Definition. [Saheb-Djahromi 1980] *A valuation is a Scott-continuous map* ν *from the open sets of a domain* D *to the unit interval* $[0, 1]$*, satisfying*

— $\nu(\emptyset) = 0$, $\nu(D) = 1$

— $\nu(O \cup U) = \nu(O) + \nu(U) - \nu(O \cap U)$

Theorem. [Lawson, Edalat, Keimel] *For many spaces, valuations and measures are in one-to-one correspondence.*

Furthermore, there is a very satisfactory theory of integration based on valuations, as well as a Riesz theorem.

The probabilistic powerdomain

The set $\mathcal{V}D$ of valuations, ordered pointwise, is called the probabilistic powerdomain of D.

Theorem. [Jones 1990] *The probabilistic powerdomain of a continuous domain D is again a continuous domain.*

The probabilistic powerdomain

The set $\mathcal{V}D$ of valuations, ordered pointwise, is called the probabilistic powerdomain of D.

Theorem. [Jones 1990] *The probabilistic powerdomain of a continuous domain D is again a continuous domain.*

However: We do not get a Scott-domain, as bounded sets may not have a supremum.

What is a “category of domains”?

We want the semantic category to be

- cartesian closed, so we can form function spaces (and use the λ-calculus as our base language)
- approximated, so that we can link syntax and semantics
- closed under $\mathcal{V}$, so that we can model probabilistic computation

Continuous domains

Definition.

A continuous domain is an ordered structure $\langle D; \sqsubseteq \rangle$ such that

- *there is a smallest element $\perp$;*
- *sups of chains (directed sets) always exist;*
- *every element is the sup of relatively finite elements;*
- *there is some countable basis;*

Have dropped: sups of bounded sets

Cartesian closure

Fact. *The category of continuous domains is not cartesian closed.*

Theorem. [J. 1990] *The largest cartesian closed category of continuous domains is* **FS***, the category of FS-domains.*

Closure under $\mathcal{V}$

Theorem. [J. & Tix 1998] *If D is a finite tree or a reversed finite tree, then $\mathcal{V}D$ is an FS-domain.*

This is still the best available result.

A. Jung and R. Tix. The troublesome probabilistic powerdomain. In A. Edalat, A. Jung, K. Keimel, and M. Kwiatkowska, editors, *Proceedings of the Third Workshop on Computation and Approximation*, volume 13 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers B.V., 1998. 23 pages

Overview

	approximated	cart. closed	closed under $\mathcal{V}$
DCPO	X	✓	✓
CONT	✓	X	✓
FS	✓	✓	??
RB	✓	✓	??
contScott	✓	✓	X
Scott	✓	✓	X

Overview

	approximated	cart. closed	closed under $\mathcal{V}$	
DCPO	X	✓	✓	
CONT	✓	X	✓	
QRB	✓	X	✓	J G-L 2012
FS	✓	✓	??	
RB	✓	✓	??	
contScott	✓	✓	X	
Scott	✓	✓	X	

V. From continuous domains to topology

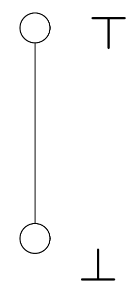
Domains and topology

We already needed the Scott topology Σ_D for the definition of the probabilistic powerdomain.

We also know that Scott-continuity is the same as topological continuity.

In fact, the order structure and the topology determine each other uniquely for continuous domains.

Furthermore, a Scott-open set on D is the same as a continuous ("characteristic") function $D \to 2$. So open sets are analogous to semi-decidable properties (albeit at all types, not just $\mathbb{N}$).



Stably compact spaces

Idea: Drop the order but keep a "nice" topological structure.

Stably compact spaces

Idea: Drop the order but keep a "nice" topological structure.

Definition. *A stably compact space is a topological space which is*

- T_0
- *compact*
- *locally compact*
- *stably compact: (finite) intersections of saturated compact sets are compact*
- *well-filtered:* $\bigcap^{\downarrow}_{i \in I} K_i \subseteq U \quad \Rightarrow \quad \exists i \in I.\, K_i \subseteq U$

They are quite nice, actually...

Fact. *Compact Hausdorff spaces are stably compact.*

Fact. *Most domains are stably compact in their Scott topology, for example, all of* **FS***.*

Theorem. [J 2004] *Stably compact spaces are closed under the probabilistic powerdomain construction (as well as many others).*

Theorem. [Lawson; Alvarez-Manilla, J, Keimel, 2004] *Every valuation on a stably compact space extends uniquely to a measure.*

They are quite nice, actually...

Fact. *Compact Hausdorff spaces are stably compact.*

Fact. *Most domains are stably compact in their Scott topology, for example, all of* **FS**.

Theorem. [J 2004] *Stably compact spaces are closed under the probabilistic powerdomain construction (as well as many others).*

Theorem. [Lawson; Alvarez-Manilla, J, Keimel, 2004] *Every valuation on a stably compact space extends uniquely to a measure.*

But...

Fact. **SCS** *is not cartesian closed.*

Approximation

How to capture topological spaces with syntax?

There may not be any canonical approximating elements...

Approximation

How to capture topological spaces with syntax?

There may not be any canonical approximating elements...

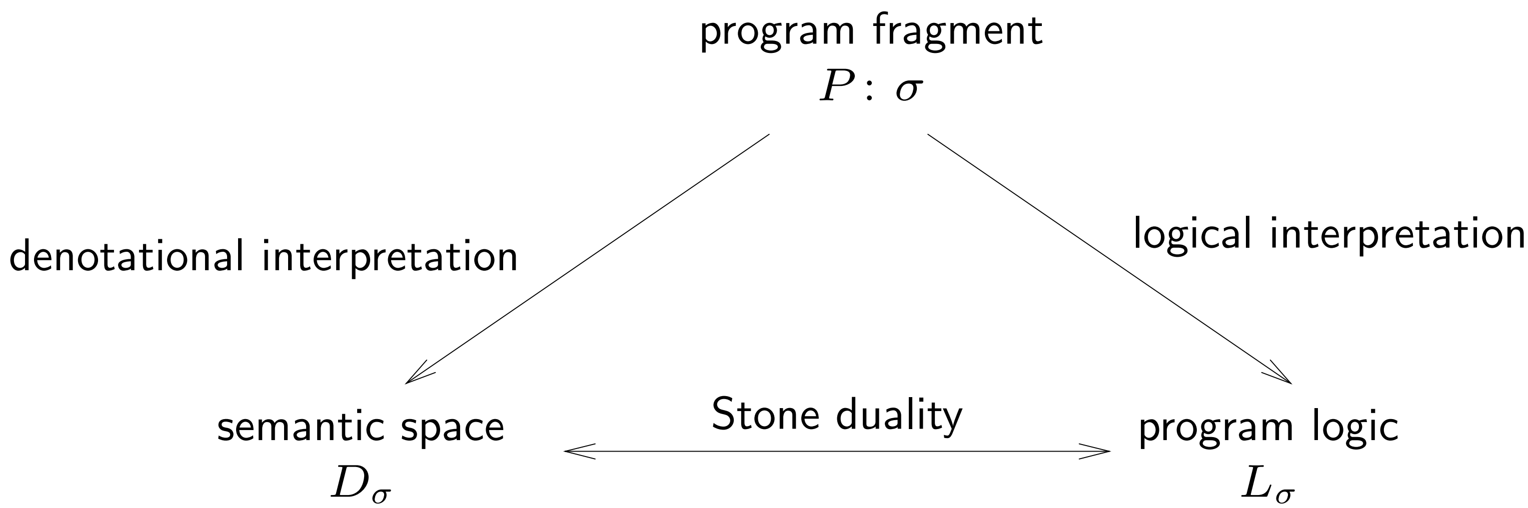
Idea: Work entirely with the lattice of open sets.

That's possible because of Stone duality which establishes an equivalence between categories of topological spaces and categories of lattices.

$$\begin{array}{rcl} \text{Stone spaces} & \cong & \text{Boolean algebras} \\ \text{spectral spaces} & \cong & \text{distributive lattices} \end{array}$$

Fact. *Scott domains are spectral spaces.*

Abramsky's Domain Theory in Logical Form



S. Abramsky. Domain theory in logical form. *Annals of Pure and Applied Logic*, 51:1–77, 1991

(LiCS Test-of-Time award 2007)

The logical reading of topology

open set	=	(observable) predicate
continuous function	=	predicate transformer
point	=	model (i.e. prime filter of formulas)
domain	=	propositional logical theory
domain construction	=	presentation of a logical theory

M. B. Smyth. Powerdomains and predicate transformers: a topological view. In J. Diaz, editor, *Automata, Languages and Programming*, volume 154 of *Lecture Notes in Computer Science*, pages 662–675. Springer Verlag, 1983

M. B. Smyth. Topology. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science, vol. 1*, pages 641–761. Clarendon Press, 1992

Example: A theory for nondeterministic choice

From a propositional logical theory $\mathcal{L}$ construct its *Smyth power theory* $\mathcal{PL}$ by

generators $\{\Box\varphi \mid \varphi \in \mathcal{L}\}$

axioms

$$\begin{aligned} \mathsf{t} &\leftrightarrow \Box\mathsf{t} \\ \Box(\varphi \wedge \psi) &\leftrightarrow \Box\varphi \wedge \Box\psi \end{aligned}$$

rule

$$\frac{\varphi \to \psi}{\Box\varphi \to \Box\psi}$$

However....

Stably compact spaces are not spectral.

Going beyond lattices

Definition. *A strong proximity lattice is a distributive lattice* $(L; \wedge, \vee, \mathsf{t}, \mathsf{f})$ *equipped with a binary relation* $\prec$ *which satisfies the "logical" axioms*

$$\begin{array}{lllll}
(\prec\!-\mathsf{t}) & x \prec \mathsf{t} & & \\
(\mathsf{f}-\!\prec) & \mathsf{f} \prec x & & \\
(\prec\!-\wedge) & x \prec y,\ x \prec y' & \iff & x \prec y \wedge y' \\
(\vee-\!\prec) & x \prec y,\ x' \prec y & \iff & x \vee x' \prec y
\end{array}$$

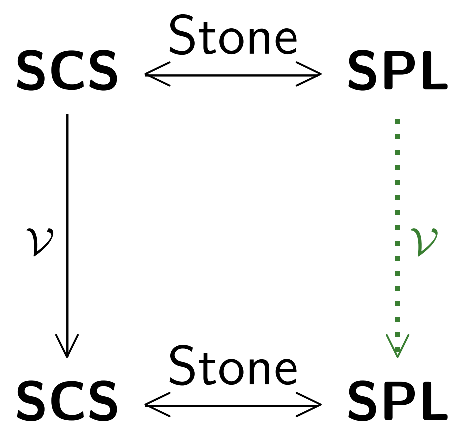
and the interpolation axioms

$$\begin{array}{llll}
(\wedge-\!\prec) & a \wedge x \prec y & \Longrightarrow & \exists a' \in X.\ a \prec a' \ \textit{and}\ a' \wedge x \prec y \\
(\prec\!-\vee) & x \prec y \vee a & \Longrightarrow & \exists a' \in X.\ a' \prec a \ \textit{and}\ x \prec y \vee a'
\end{array}$$

Stone duality for stably compact spaces

Theorem. [J & Sünderhauf 1995] *The Stone duals of strong proximity lattices are precisely the stably compact spaces.*

Transferring the probabilistic power space construction to the logic via Stone Duality



The logic of the probabilistic powerdomain

Task. From a domain logic $\mathcal{L}$ give a presentation for $\mathcal{VL}$

generators $\langle \varphi, r \rangle$ for all $\varphi \in \mathcal{L}$ and all $r \in (0,1) \cap \mathbb{Q}$
with the intended reading: probability of φ is greater than r

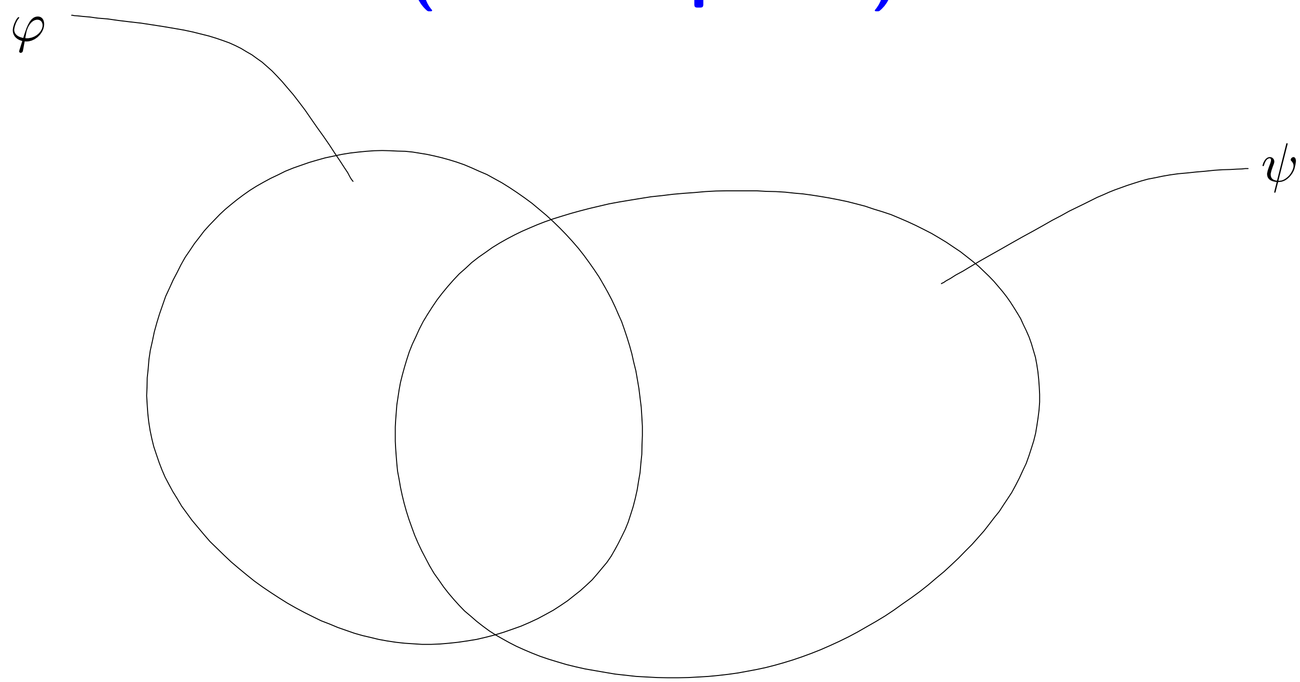
axioms and rules

$$\frac{}{\langle \mathsf{f}, p \rangle \prec \mathsf{f}}$$

$$\frac{\varphi \vee \psi \prec \rho \quad \varphi \wedge \psi \prec \sigma \quad p + q > r + s}{\langle \varphi, p \rangle \wedge \langle \psi, q \rangle \prec \langle \rho, r \rangle \vee \langle \sigma, s \rangle}$$

$$\frac{\varphi \prec \rho \wedge \sigma \quad \psi \prec \rho \vee \sigma \quad p + q > r + s}{\langle \varphi, p \rangle \wedge \langle \psi, q \rangle \prec \langle \rho, r \rangle \vee \langle \sigma, s \rangle}$$

Illustrating the soundness of the first modularity law (over-simplified)



$$\frac{p + q > r + s}{\langle \varphi, p \rangle \wedge \langle \psi, q \rangle \prec \langle \varphi \vee \psi, r \rangle \vee \langle \varphi \wedge \psi, s \rangle}$$

Theorem 1. [Heckmann 1994; J & Moshier 2002]
If $\mathcal{L}$ is a domain logic that is sound and complete for the stably compact space X, then logic $\mathcal{VL}$ is sound and complete for the probabilistic power space $\mathcal{V}X$.

Desharnais-Edalat-Panangaden Logic

$$\varphi, \psi ::= \mathsf{t} \mid \varphi \wedge \psi \mid \langle a, r \rangle \varphi$$

Say

$$s \Vdash \langle a, r \rangle \varphi$$

if the probability of the result state satisfying φ is greater than r when action a is performed in state s.

Theorem. [Desharnais, Edalat, Panangaden, 1998] *Two states of a labelled Markov process are (Larsen-Skou) bisimilar if and only if they satisfy the same DEP formulas.*

Desharnais-Edalat-Panangaden Logic

$$\varphi, \psi ::= \mathsf{t} \mid \varphi \wedge \psi \mid \langle a, r \rangle \varphi$$

Say

$$s \Vdash \langle a, r \rangle \varphi$$

if the probability of the result state satisfying φ is greater than r when action a is performed in state s.

Theorem. [Desharnais, Edalat, Panangaden, 1998] *Two states of a labelled Markov process are (Larsen-Skou) bisimilar if and only if they satisfy the same DEP formulas.*

Note that DEP logic contains formulas but no derivation system.

Probabilistic synchronization trees

Consider the labelled Markov process Proc defined by the domain equation

$$D \cong \mathcal{V} D^{\mathsf{Act}}$$

Theorem. [Desharnais, Gupta, Jagadeesan, Panangaden, 2003]
For any labelled Markov process, two states are bisimilar if and only if they are mapped to the same element under the final morphism into Proc.

Probabilistic synchronization trees

Consider the labelled Markov process Proc defined by the domain equation

$$D \cong \mathcal{V} D^{\mathsf{Act}}$$

Theorem. [Desharnais, Gupta, Jagadeesan, Panangaden, 2003] *For any labelled Markov process, two states are bisimilar if and only if they are mapped to the same element under the final morphism into* Proc*.*

Idea: Consider the domain logic $\mathcal{L}$ associated with Proc by Stone duality, which has disjunction as well as conjunction (and so is richer than DEP logic) but also comes equipped with a sound and complete derivation system.

Semantic/logical proof of the DEP Theorem

The domain logic $\mathcal{L}$ for Proc is generated by the grammar

$$\varphi, \psi ::= \mathsf{t} \mid \mathsf{f} \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \langle a, \varphi, r \rangle$$

By soundness and completeness of domain logic, two elements of D are equal if and only if they satisfy the same domain logic formulas.

Hence, all we need to do is show that disjunctions are not required to separate round prime filters.

Proof sketch

(Ignoring the set of actions)

- Let F and G be two different round prime filters, say $F \not\subseteq G$

Proof sketch

(Ignoring the set of actions)

- Let F and G be two different round prime filters, say $F \not\subseteq G$
- there exists $\varphi \in F$, $\varphi \notin G$

Proof sketch

(Ignoring the set of actions)

- Let F and G be two different round prime filters, say $F \not\subseteq G$
- there exists $\varphi \in F$, $\varphi \notin G$
- consider the structure of φ:

  it cannot be f because F is prime;

  it cannot be t because G is a filter;

  if it is of the form $\psi \wedge \psi'$ then one of ψ or ψ' must belong to $F \setminus G$ because G is a filter;

if it is of the form $\psi \vee \psi'$ then one of ψ or ψ' must belong to $F \setminus G$ because F is prime.

So, can assume that φ has the form $\langle \psi, p \rangle$.

if it is of the form $\psi \vee \psi'$ then one of ψ or ψ' must belong to $F \setminus G$ because F is prime.

So, can assume that φ has the form $\langle \psi, p \rangle$.

- Thus we have shown how to eliminate all propositional structure at the outer level. We must now show how we can eliminate "embedded" disjunctions. Since formulas in L have finite depth, it is sufficient to show how disjunctions can be "percolated up" from one level to the next higher one.

if it is of the form $\psi \vee \psi'$ then one of ψ or ψ' must belong to $F \setminus G$ because F is prime.

So, can assume that φ has the form $\langle \psi, p \rangle$.

- Thus we have shown how to eliminate all propositional structure at the outer level. We must now show how we can eliminate "embedded" disjunctions. Since formulas in L have finite depth, it is sufficient to show how disjunctions can be "percolated up" from one level to the next higher one.

- So consider the case that φ has the form $\langle \rho \vee \sigma, p \rangle$. Remember that F defines a valuation μ, and G a valuation ν. By construction, $\mu(\rho \vee \sigma) > \nu(\rho \vee \sigma)$.

By the modularity of valuations,

$$\mu(\rho \vee \sigma) = \mu(\rho) + \mu(\sigma) - \mu(\rho \wedge \sigma)$$

we have

$$\mu(\rho) + \mu(\sigma) - \mu(\rho \wedge \sigma) > \nu(\rho) + \nu(\sigma) - \nu(\rho \wedge \sigma)$$

Hence one of the following must be true:

- $\mu(\rho) > \nu(\rho)$ or $\langle \rho, r \rangle \in F \setminus G$ for some $r \in (0, 1)$
- $\mu(\sigma) > \nu(\sigma)$ or $\langle \sigma, r \rangle \in F \setminus G$ for some $r \in (0, 1)$
- $\mu(\rho \wedge \sigma) < \nu(\rho \wedge \sigma)$ or $\langle \rho \wedge \sigma, r \rangle \in G \setminus F$ for some $r \in (0, 1)$

Q.E.D.