Denoting computation

A jog from Scott Domains to Hypercoherence Spaces

Paolo Tranquilli

13/12/2006

Outline

Motivation

Why am I here? Well, mainly I wanted to know more about what I will show you, and what better way than this? So, please, bear with me, it won't take long... I hope!

Motivation

Why am I here? Well, mainly I wanted to know more about what I will show you, and what better way than this? So, please, bear with me, it won't take long... I hope!

Outline

First things first: what does operational semantic mean?

- Operational semantic is a cool and fancy name for what describes how a program written in some language runs.
- Functional programming language, λ**-calculus** at the core:
  - variables: x
  - application of a function to an argument: $M\,N$
  - definition of a function by abstraction of a variable: $\lambda x.M$
  - $+$ constants $+$ types

First things first: what does operational semantic mean?

- Operational semantic is a cool and fancy name for what describes how a program written in some language runs.
- Functional programming language, λ**-calculus** at the core:
  - variables: x
  - application of a function to an argument: $M\,N$
  - definition of a function by abstraction of a variable: $\lambda x.M$
  - + constants + types

First things first: what does operational semantic mean?

- Operational semantic is a cool and fancy name for what describes how a program written in some language runs.
- Functional programming language, λ**-calculus** at the core:
  - variables: x
  - application of a function to an argument: $M\,N$
  - definition of a function by abstraction of a variable: $\lambda x.M$
  - + constants + types

Operational Semantics Means...

- ... giving rules for the evaluation of terms.
- Interaction between abstraction and application, involving substitution:

$$\lambda x.M\ N \triangleright M[x := N]$$

- Constants have their rules also, for example:

$$\texttt{if}\ P\ Q\ R \triangleright \begin{cases} Q & \text{if } P \triangleright \texttt{true} \\ R & \text{if } P \triangleright \texttt{false} \end{cases}$$

Denotational semantic means instead...

- ... capturing (or trying to do so) the essence of a program, regardless of its evaluation.
- This is done by interpreting programs as true functions (or rather morphisms) between mathematical structures (or rather objects of a category) which interpret the types.
- This interpretation is usually denoted by $[\![\,.\,]\!]$.

Outline

Set-theoretic partial functions

- Types as **sets**: $[\![o]\!] := \{\mathrm{tt}, \mathrm{ff}\} =: \mathrm{Bool}$, $[\![\sigma \to \tau]\!] := [\![\tau]\!] \rightharpoonup [\![\sigma]\!]$. Partiality sits there for handling divergence.
- Programs are interpreted with their extensional meaning, for $[\![\mathtt{if}]\!] \in \mathrm{Bool} \rightharpoonup (\mathrm{Bool} \rightharpoonup (\mathrm{Bool} \rightharpoonup \mathrm{Bool}))$ as an example:

$$[\![\mathtt{if}]\!](\mathrm{tt})(x)(y) := x$$
$$[\![\mathtt{if}]\!](\mathrm{ff})(x)(y) := y$$

Way too many functions: the interpretations get drowned in the sea of all the set theoretical functions!

Set-theoretic partial functions

- Types as **sets**: $[\![o]\!] := \{\,\mathrm{tt}, \mathrm{ff}\,\} =: \mathrm{Bool}$, $[\![\sigma \to \tau]\!] := [\![\tau]\!] \rightharpoonup [\![\sigma]\!]$. Partiality sits there for handling divergence.
- Programs are interpreted with their extensional meaning, for $[\![\mathtt{if}]\!] \in \mathrm{Bool} \rightharpoonup (\mathrm{Bool} \rightharpoonup (\mathrm{Bool} \rightharpoonup \mathrm{Bool}))$ as an example:

$$[\![\mathtt{if}]\!](\mathrm{tt})(x)(y) := x$$
$$[\![\mathtt{if}]\!](\mathrm{ff})(x)(y) := y$$

Way too many functions: the interpretations get drowned in the sea of all the set theoretical functions!

Terms themselves

- We can formally define a category in which objects are types τ and morphisms $M : \sigma \to \tau$ are evaluated terms M of type $\sigma \to \tau$.
- Types are interpreted as themselves, terms as their evaluation.

Way too uninformative: the interpretation does not say anything more than the syntax!

Terms themselves

- We can formally define a category in which objects are types τ and morphisms $M : \sigma \to \tau$ are evaluated terms M of type $\sigma \to \tau$.
- Types are interpreted as themselves, terms as their evaluation.

Way too uninformative: the interpretation does not say anything more than the syntax!

Outline

From Operational to Denotational

Operational semantic	$\xrightarrow{[\![\,\cdot\,]\!]}$	**Denotational semantic**
types τ	$\longrightarrow$	objects $[\![\tau]\!]$
terms $M : \sigma \to \tau$	$\longrightarrow$	morphisms $[\![M]\!] : [\![\sigma]\!] \to [\![\tau]\!]$
reduction $M \triangleright N$	$\longrightarrow$	equality $[\![M]\!] = [\![N]\!]$
dynamic	$\longrightarrow$	static

From Operational to Denotational

Operational semantic	$\xrightarrow{[\![\cdot]\!]}$	**Denotational semantic**
types τ	$\longrightarrow$	objects $[\![\tau]\!]$
terms $M : \sigma \to \tau$	$\longrightarrow$	morphisms $[\![M]\!] : [\![\sigma]\!] \to [\![\tau]\!]$
reduction $M \rhd N$	$\longrightarrow$	equality $[\![M]\!] = [\![N]\!]$
dynamic	$\longrightarrow$	static

From Operational to Denotational

Operational semantic	$\xrightarrow{[\![\,\cdot\,]\!]}$	**Denotational semantic**
types τ	$\longrightarrow$	objects $[\![\tau]\!]$
terms $M : \sigma \to \tau$	$\longrightarrow$	morphisms $[\![M]\!] : [\![\sigma]\!] \to [\![\tau]\!]$
reduction $M \rhd N$	$\longrightarrow$	equality $[\![M]\!] = [\![N]\!]$
dynamic	$\longrightarrow$	static

From Operational to Denotational

Operational semantic	$\xrightarrow{[\![\,\cdot\,]\!]}$	**Denotational semantic**
types τ	$\longrightarrow$	objects $[\![\tau]\!]$
terms $M : \sigma \to \tau$	$\longrightarrow$	morphisms $[\![M]\!] : [\![\sigma]\!] \to [\![\tau]\!]$
reduction $M \triangleright N$	$\longrightarrow$	equality $[\![M]\!] = [\![N]\!]$
dynamic	$\longrightarrow$	static

From Operational to Denotational

Operational semantic	$\xrightarrow{[\![\cdot]\!]}$	**Denotational semantic**
types τ	$\longrightarrow$	objects $[\![\tau]\!]$
terms $M : \sigma \to \tau$	$\longrightarrow$	morphisms $[\![M]\!] : [\![\sigma]\!] \to [\![\tau]\!]$
reduction $M \rhd N$	$\longrightarrow$	equality $[\![M]\!] = [\![N]\!]$
dynamic	$\longrightarrow$	static

CCC

In order to interpret λ-calculus we must necessarily be able to handle function spaces *inside* the category. This amounts to using a **cartesian closed category**, ccc in short.

cartesian
: for A, B we have a *product* $A \times B$, so that we have (a natural transformation):

$$f : P \to A,\ g : P \to B \overset{\sim}{\longmapsto} \langle f, g \rangle : P \to A \times B$$

and the void product: a terminal object 1, neutral up to isomorphism for $\times$.

closed
: for A, B we have a *function space* $A \Rightarrow B$, so that we have (a natural transformation):

$$f : A \times B \to C \overset{\sim}{\longmapsto} \Lambda(f) : A \to B \Rightarrow C$$

CCC

In particular:

- *arrows* $A \to B$ of the category are in correspondance with the *points* $1 \to A \Rightarrow B$ of the object $A \Rightarrow B$.
- we have a morphism $\mathrm{ev} = \Lambda^{-1}(\mathrm{id}_{A \Rightarrow B}) : (A \Rightarrow B) \times A \to B$ which represents *internally* the application of a function to a point.

What Do We Get?

Quoting from the back of *Domains and Lambda-Calculi* by Amadio and Curien:

> *The main goals are to provide formal tools to assess the meaning of programming constructs [...] and to prove properties about programs, such as whether they terminate, or whether their result is a solution of the problem they are supposed to solve.*

Outline

1. Motivation
2. Introducing Denotational Semantics
   - What Does Denotational Semantic Mean?
   - Trivial examples
   - Basic things to know
3. Orders
   - Scott domains
   - dI-domains
4. Events
   - Event structures
   - Coherences
   - Hypercoherences
5. Conclusion

A good idea!

- Dana Scott's idea is to use a refined mathematical concept:
- Types are interpreted as **topological spaces**, with a spatial flavour to the concept of information.
- Programs are interpreted as **continuous functions**, with computation as some kind of well-behaved flow of information.

However ....

A good idea!

- Dana Scott's idea is to use a refined mathematical concept:
- Types are interpreted as **topological spaces**, with a spatial flavour to the concept of information.
- Programs are interpreted as **continuous functions**, with computation as some kind of well-behaved flow of information.

However topological spaces behave terribly with function spaces, one needs many constraints!
Anyway usually *domains* are presented as *orders*.

A Scott Domain Is...

... a partially ordered set $(D, \sqsubseteq)$.

- Points of D (states) represent amounts of information.
- $x \sqsubseteq y$ represents that y contains all the information in x.
- Supremum and infimum of X (if they exist) are noted by $\bigsqcup X$ and $\bigsqcap X$.

A Scott Domain Is...

... a poset $(D, \sqsubseteq)$.

- Points of D (states) represent amounts of information.
- $x \sqsubseteq y$ represents that y contains all the information in x.
- Supremum and infimum of X (if they exist) are noted by $\bigsqcup X$ and $\bigsqcap X$.

A Scott Domain Is...

... a directed complete poset $(D, \sqsubseteq)$.

- $X \neq \emptyset$ directed if $\forall x, y \in X. \exists z \in X. x \sqsubseteq z \,\&\, y \sqsubseteq z$.
- D has suprema for all of its directed subsets.
- Directed sets represent arbitrary approximations of possibly infinite information. Here we say D has the targets of such approximations.

A Scott Domain Is...

... a dcpo $(D, \sqsubseteq)$.

- $X \neq \emptyset$ directed if $\forall x, y \in X. \exists z \in X. x \sqsubseteq z \,\&\, y \sqsubseteq z$.
- D has suprema for all of its directed subsets.
- Directed sets represent arbitrary approximations of possibly infinite information. Here we say D has the targets of such approximations.

A Scott Domain Is...

... a bounded complete dcpo $(D, \sqsubseteq)$.

- $X \neq \emptyset$ bounded if $\exists z \in D. \forall x \in X. x \sqsubseteq z$.
- D has suprema for all of its bounded subsets.
- Bounded sets represent consistent information which they can be completed to a common state. Here we say compatible information has a unique way of being extended.
- In particular we have a bottom element $\bot = \bigsqcup \emptyset$ which represents *no* information (as for definitions this fact turns the dcpo into a cpo).

A Scott Domain Is...

... a bcpo $(D, \sqsubseteq)$.

- $X \neq \emptyset$ bounded if $\exists z \in D. \forall x \in X. x \sqsubseteq z$.
- D has suprema for all of its bounded subsets.
- Bounded sets represent consistent information which they can be completed to a common state. Here we say compatible information has a unique way of being extended.
- In particular we have a bottom element $\bot = \bigsqcup \emptyset$ which represents *no* information (as for definitions this fact turns the dcpo into a cpo).

A Scott Domain Is...

... an algebraic bcpo $(D, \sqsubseteq)$.

- The *compact* elements of D are $\mathcal{K}(D) := \{\, d \in D \mid \forall X \subseteq_{\mathrm{dir}} D.(d \sqsubseteq \bigsqcup X \Rightarrow \exists x \in D.d \sqsubseteq x) \,\}$
- Compact elements are primitive in some sense: they cannot be in an approximation $\bigsqcup X$ without already being present in some approximant.
- Algebraicity is the condition: $\forall d \in D.d = \bigsqcup \{\, k \in \mathcal{K}(D) \mid k \sqsubseteq d \,\}$.
- Here we say that compacts are really primitive: every element is approximable (representable) by them.

That's It!

A Scott Domain is an algebraic bcpo.

Not such a long definition, don't you think?

Example: Flat Domains

- S set: $S_\perp$ is $(S + \{\perp\}, \sqsubseteq)$ where $x \sqsubseteq y \iff x = \perp$. These are *flat* domains.
- They are used to represent atomic data, know all or nothing. For example

$$\mathrm{Bool}_\perp = \begin{tikzcd}[column sep=small] \mathrm{tt} \arrow[dr, dash] & & \mathrm{ff} \arrow[dl, dash] \\ & \perp & \end{tikzcd} \qquad \mathbb{N}_\perp = \begin{tikzcd}[column sep=small] 0 \arrow[d, dash] & 1 \arrow[dl, dash] & 2 \arrow[dll, dash] & \cdots \\ \perp & & & \end{tikzcd}$$

Morphisms of Scott Domains

$f : D \to E$ is continuous iff:

- it is *monotonic*: $x \sqsubseteq y \implies f(x) \sqsubseteq f(y)$;
- it preserves directed suprema: $\forall X \subseteq_{\mathrm{dir}} D. f(\bigsqcup X) = \bigsqcup f(X)$.

What we are saying is that in order to process infinite information we can stick to the approximations of it and indeed get approximations of the output.

Scott Domains Are a CCC - Product

The product is the set-theoretic one with componentwise order, the unit is $\{\bot\}$. So for example $\mathrm{Bool}_\bot \times \mathrm{Bool}_\bot$ is:

$$\begin{tikzcd}
 & & (\mathrm{tt}, \mathrm{tt}) \arrow[ddll, dash] \arrow[ddrr, dash] & & \\
 & (\mathrm{tt}, \mathrm{ff}) \arrow[dl, dash] \arrow[d, dash] & (\mathrm{ff}, \mathrm{ff}) \arrow[dl, dash] \arrow[dr, dash] & (\mathrm{ff}, \mathrm{tt}) \arrow[d, dash] \arrow[dr, dash] & \\
(\mathrm{tt}, \bot) \arrow[drr, dash] & (\bot, \mathrm{ff}) \arrow[dr, dash] & & (\mathrm{ff}, \bot) \arrow[dl, dash] & (\bot, \mathrm{tt}) \arrow[dll, dash] \\
 & & (\bot, \bot) & &
\end{tikzcd}$$

Scott Domains Are a CCC - Function Space

We have to define the order on the set of continuous functions, but the pointwise one turns out to be good.

$$f \sqsubseteq_c g \iff \forall x. f(x) \sqsubseteq g(x)$$

Ok, So What's the Problem?

We have more functions than we would like to have.
Most notably the *parallel or*:

$$\begin{aligned} \mathsf{por}(\mathrm{tt}, x) &:= \mathrm{tt} \\ \mathsf{por}(x, \mathrm{tt}) &:= \mathrm{tt} \\ \mathsf{por}(\mathrm{ff}, \mathrm{ff}) &:= \mathrm{ff} \\ \mathsf{por}(x, y) &:= \bot \text{ otherwise} \end{aligned}$$

por is Scott continuous but cannot be computed sequentially!
How can I capture sequentiality?

Ok, So What's the Problem?

We have more functions than we would like to have.
Most notably the *parallel or*:

$$\begin{aligned} \mathrm{por}(\mathrm{tt}, x) &:= \mathrm{tt} \\ \mathrm{por}(x, \mathrm{tt}) &:= \mathrm{tt} \\ \mathrm{por}(\mathrm{ff}, \mathrm{ff}) &:= \mathrm{ff} \\ \mathrm{por}(x, y) &:= \perp \text{ otherwise} \end{aligned}$$

por is Scott continuous but cannot be computed sequentially!
How can I capture sequentiality?

Outline

From Continuity to Stability

Berry's idea: $f : D \to E$ is stable iff:

- it is continuous;
- if X is bounded then $f(\bigsqcap X) = \bigsqcap f(X)$.

Given partial info about an output, there exist a minimum info read from input that produces that partial info, in particular that output info cannot come from various input sources.

por Is Not Stable

$(\mathrm{tt}, \perp)$ and $(\perp, \mathrm{tt})$ are bounded by $(\mathrm{tt}, \mathrm{tt})$. But

$$\mathsf{por}((\mathrm{tt}, \perp) \sqcup (\perp, \mathrm{tt})) = \mathsf{por}(\perp, \perp) = \perp$$

$$\mathsf{por}(\mathrm{tt}, \perp) \sqcup \mathsf{por}(\perp, \mathrm{tt}) = \mathrm{tt} \sqcup \mathrm{tt} = \mathrm{tt}$$

There is no minimum info taken from input in order to produce output tt, as there are *two* possible ways of gaining that information.

Oops...

With pointwise ordering of functions ev fails to be stable. No hope of having a CCC!
Imposing the stability of ev one gets a restriction of the ordering, the stable one:

$$f \sqsubseteq_s g \iff \forall x \sqsubseteq y . f(x) = f(y) \sqcap g(x)$$

Quite awkward if you ask me...

Oops...

With pointwise ordering of functions ev fails to be stable. No hope of having a CCC!
Imposing the stability of ev one gets a restriction of the ordering, the stable one:

$$f \sqsubseteq_s g \iff \forall x \sqsubseteq y. f(x) = f(y) \sqcap g(x)$$

Quite awkward if you ask me...

Traces

In fact a more natural definition comes from *traces*. A stable function is completely determined by

$$\mathrm{trace}(f) := \{\, (d, e) \in \mathcal{K}(D) \times \mathcal{K}(E) \mid e \sqsubseteq f(d) \text{ with } d \text{ minimal} \,\}$$

Then we remarkably have

$$f \sqsubseteq_s g \iff \mathrm{trace}(f) \subseteq \mathrm{trace}(g)$$

So $\sqsubseteq_s$ is a good notion of "less information than". Unfortunately $(D \Rightarrow_s E, \sqsubseteq_s)$ is not a Scott domain in general...

Traces

In fact a more natural definition comes from *traces*. A stable function is completely determined by

$$\mathrm{trace}(f) := \{\, (d, e) \in \mathcal{K}(D) \times \mathcal{K}(E) \mid e \sqsubseteq f(d) \text{ with } d \text{ minimal} \,\}$$

Then we remarkably have

$$f \sqsubseteq_s g \iff \mathrm{trace}(f) \subseteq \mathrm{trace}(g)$$

So $\sqsubseteq_s$ is a good notion of “less information than”. Unfortunately $(D \Rightarrow_s E, \sqsubseteq_s)$ is not a Scott domain in general...

Ladies and Gentlemen, Meet the dI-Domains

dI-domains are Scott domains in which:

d) $\sqcap$ distributes over $\sqcup$: if $\exists b \sqcup c$ then $a \sqcap (b \sqcup c) = (a \sqcap b) \sqcup (a \sqcap c)$;

I) all compacts have a finite number of states under them: compacts do not contain infinite pieces of information.

dI-domains and stable functions are a CCC with the usual product.

Hmm, it's becoming more complicated... but dI-domains are event structures!

Ladies and Gentlemen, Meet the dI-Domains

dI-domains are Scott domains in which:

d) $\sqcup$ distributes over $\sqcap$: if $\exists b \sqcup c$ then $a \sqcap (b \sqcup c) = (a \sqcap b) \sqcup (a \sqcap c)$;

I) all compacts have a finite number of states under them: compacts do not contain infinite pieces of information.

dI-domains and stable functions are a CCC with the usual product.

Hmm, it's becoming more complicated... but dI-domains are event structures!

Ladies and Gentlemen, Meet the dI-Domains

dI-domains are Scott domains in which:

d) $\sqcap$ distributes over $\sqcup$: if $\exists b \sqcup c$ then $a \sqcap (b \sqcup c) = (a \sqcap b) \sqcup (a \sqcap c)$;

I) all compacts have a finite number of states under them: compacts do not contain infinite pieces of information.

dI-domains and stable functions are a CCC with the usual product.

Hmm, it's becoming more complicated... but dI-domains are event structures!

Ladies and Gentlemen, Meet the dI-Domains

dI-domains are Scott domains in which:

d) $\sqcap$ distributes over $\sqcup$: if $\exists b \sqcup c$ then $a \sqcap (b \sqcup c) = (a \sqcap b) \sqcup (a \sqcap c)$;

I) all compacts have a finite number of states under them: compacts do not contain infinite pieces of information.

dI-domains and stable functions are a CCC with the usual product.

Hmm, it's becoming more complicated... but dI-domains are event structures!

Outline

Let's Try and Keep it Simple!

An event structure is $E = (|E|, Con, \vdash)$ where:

- $|E|$ is a set: the events.
- $\emptyset \neq Con \subseteq \mathcal{P}_{\mathrm{fin}}(E)$: consistency, s.t. $Y \subseteq X \in Con \implies Y \in Con$.
- $\vdash \subseteq Con \times E$: the enabling relation

The *states* of E, noted $D(E)$, are subsets x of E

- consistent, i.e. $\forall X \subseteq_{\mathrm{fin}} x . X \in Con$, and
- safe, i.e. each $e \in x$ has a history of enablings all inside x

Stable Event Structures

An event structure is *stable* if every event in every state has a unique enabling in that state.

$$\forall x \in D(E),\ e \in x,\ X, Y \subseteq_{\text{fin}} x.(X \vdash e \mathbin{\&} Y \vdash e \implies X = Y)$$

- $D(E)$ for E stable event structure are exactly the dI-domains.
- Traces on event structures get a nicer form:

  $$\text{trace}(f) = \{\, (x, e) \in D_{\text{fin}}(E) \times |F| \mid e \in f(x) \text{ with } x \text{ minimal} \,\}$$

Let's try to simplify more, even restricting our scope... what if all events are initial, i.e. $\vdash = \{\emptyset\} \times |E|$?

Stable Event Structures

An event structure is *stable* if every event in every state has a unique enabling in that state.

$$\forall x \in D(E),\ e \in x,\ X, Y \subseteq_{\text{fin}} x.(X \vdash e \,\&\, Y \vdash e \implies X = Y)$$

- $D(E)$ for E stable event structure are exactly the dI-domains.
- Traces on event structures get a nicer form:

  $$\text{trace}(f) = \{\, (x, e) \in D_{\text{fin}}(E) \times |F| \mid e \in f(x) \text{ with } x \text{ minimal} \,\}$$

Let's try to simplify more, even restricting our scope... what if all events are initial, i.e. $\vdash = \{\emptyset\} \times |E|\}$?

Simpler!

A qualitative event structure is $E = (|E|, Con)$ where:

- $|E|$ is a set: the events.
- $\emptyset \neq Con \subseteq \mathcal{P}_{\mathrm{fin}}(E)$: consistency, s.t. $Y \subseteq X \in Con \implies Y \in Con$.

The *states* of E, noted $D(E)$, are subsets x of E

- consistent, i.e. $\forall X \subseteq_{\mathrm{fin}} x . X \in Con$

Safeness of states and stability are for free!

$D(E)$ for E qualitative event structure are exactly the qualitative domains.

Good. Can we simplify even more? What if *Con* is generated by a binary relation?

Simpler!

A qualitative event structure is $E = (|E|, Con)$ where:

- $|E|$ is a set: the events.
- $\emptyset \neq Con \subseteq \mathcal{P}_{\text{fin}}(E)$: consistency, s.t. $Y \subseteq X \in Con \implies Y \in Con$.

The *states* of E, noted $D(E)$, are subsets x of E

- consistent, i.e. $\forall X \subseteq_{\text{fin}} x. X \in Con$

Safeness of states and stability are for free!

$D(E)$ for E qualitative event structure are exactly the qualitative domains.

Good. Can we simplify even more? What if Con is generated by a binary relation?

Outline

Simpler! Simpler!

A coherence space is $E = (|E|, \frown)$ where:

- $|E|$ is a set: the web.
- $\frown$ a binary symmetric reflexive relation: coherence.

The *states* of E, noted $D(E)$, are subsets x of E

- consistent, i.e. $\forall e, f \in x : e \frown f$

Well, quite simple: E is a reflexive undirected graph, and $D(E)$ are its cliques.

Let's take a look at the product and function spaces.

Simpler! Simpler!

A coherence space is $E = (|E|, \frown)$ where:

- $|E|$ is a set: the web.
- $\frown$ a binary symmetric reflexive relation: coherence.

The *states* of E, noted $D(E)$, are subsets x of E

- consistent, i.e. $\forall e, f \in x : e \frown f$

Well, quite simple: E is a reflexive undirected graph, and $D(E)$ are its cliques.

Let's take a look at the product and function spaces.

Products, very briefly

- $|E \times F| = |E| + |F| = \{0\} \times |E| \cup \{1\} \times |F|$;
- $(i, a) \frown (j, b)$ iff $i = j \implies a \frown b$ in the relative space.
- In fact $D(E \times F) = D(E) \times D(F)$.
- 1 is the empty web.

Function Spaces

- We have a coherence space in which states are exactly the traces of stable functions;
- $|E \Rightarrow_s F| = D_{\mathrm{fin}}(E) \times |F|$;
- $(x, f) \frown (y, g)$ iff
  $x \cup y \in D(E) \implies (f \frown g \,\&\, (f = g \implies x = y))$

Mmm, seems like two operations...

- Exponential $|!E| = D_{\mathrm{fin}}(E)$ with $x \frown y$ iff $x \cup y \in D(E)$;
- Linear arrow $|E \multimap F| = |E| \times |F|$ with $(d, f) \frown (e, g)$ iff
  $d \frown e \implies (f \frown g \,\&\, (f = g \implies d = e))$.

Function Spaces and a Glimpse of Linear Logic

- We have a coherence space in which states are exactly the traces of stable functions;
- $|E \Rightarrow_s F| = D_{\text{fin}}(E) \times |F|$;
- $(x, f) \frown (y, g)$ iff
  $x \cup y \in D(E) \implies (f \frown g \,\&\, (f = g \implies x = y))$

Mmm, seems like two operations...

- Exponential $|!E| = D_{\text{fin}}(E)$ with $x \frown y$ iff $x \cup y \in D(E)$;
- Linear arrow $|E \multimap F| = |E| \times |F|$ with $(d, f) \frown (e, g)$ iff
  $d \frown e \implies (f \frown g \,\&\, (f = g \implies d = e))$.

Function Spaces and a Glimpse of Linear Logic

- We have a coherence space in which states are exactly the traces of stable functions;
- $|E \Rightarrow_s F| = D_{\mathrm{fin}}(E) \times |F|$;
- $(x, f) \coh (y, g)$ iff
  $x \cup y \in D(E) \implies (f \coh g \,\&\, (f = g \implies x = y))$

Mmm, seems like two operations...

- Exponential $|!E| = D_{\mathrm{fin}}(E)$ with $x \coh y$ iff $x \cup y \in D(E)$;
- Linear arrow $|E \multimap F| = |E| \times |F|$ with $(d, f) \coh (e, g)$ iff
  $d \coh e \implies (f \coh g \,\&\, (f = g \implies d = e))$.

Problems again?

There is much more to coherence spaces, but there is also a problem, shared by all models with stable functions.

Problems again? Yes, the Gustave function

There is much more to coherence spaces, but there is also a problem, shared by all models with stable functions.

The Gustave function G

The Gustave Function

A vicious one, the Gustave function G is defined by:

$$\begin{aligned} G(\mathrm{tt}, \mathrm{ff}, x) &:= \mathrm{tt} \\ G(x, \mathrm{tt}, \mathrm{ff}) &:= \mathrm{tt} \\ G(\mathrm{ff}, x, \mathrm{tt}) &:= \mathrm{tt} \\ G(x, y, z) &:= \perp \text{ otherwise} \end{aligned}$$

It is stable, because $(\mathrm{tt}, \mathrm{ff}, \perp), (\perp, \mathrm{tt}, \mathrm{ff}), (\mathrm{ff}, \mathrm{tt}, \perp)$ are not compatible, so we do not check for preservation of minimum. Like por it is not sequentializable, no first input to look at.

Getting Rid of the Gustave Function

We want to regard $\{(\mathrm{tt}, \mathrm{ff}, \bot), (\bot, \mathrm{tt}, \mathrm{ff}), (\mathrm{ff}, \mathrm{tt}, \bot)\}$ as coherent, while still any two of the triples are incoherent.

We move on to a kind of consistency *not* downward consistent.

Getting Rid of the Gustave Function

We want to regard $\{(\mathrm{tt}, \mathrm{ff}, \bot), (\bot, \mathrm{tt}, \mathrm{ff}), (\mathrm{ff}, \mathrm{tt}, \bot)\}$ as coherent, while still any two of the triples are incoherent.

We move on to a kind of consistency *not* downward consistent.

dIC-Domains and Strongly Stable Functions

- dIC-domains: dI-domains D equipped with a $\mathcal{C}(D) \subseteq \mathcal{P}_{\mathrm{fin}}(D)$ with some properties.
- bounded finite sets of states are in $\mathcal{C}(D)$.
- $f : D \to E$ is strongly stable iff
  - it is continuous;
  - $\forall X \in \mathcal{C}(D). f(X) \in \mathcal{C}(E)\ \&\ f(\bigsqcap X) = \bigsqcap f(X)$.
- $A := \{(\mathrm{tt}, \mathrm{ff}, \bot), (\bot, \mathrm{tt}, \mathrm{ff}), (\mathrm{ff}, \mathrm{tt}, \bot)\} \in \mathcal{C}(\mathrm{Bool}_\bot^3)$, but

$$G(\bigsqcap A) = G(\bot, \bot, \bot) = \bot \quad \text{while} \quad \bigsqcap G(A) = \bigsqcap \{\mathrm{tt}\} = \mathrm{tt}$$

Outline

1. Motivation
2. Introducing Denotational Semantics
   - What Does Denotational Semantic Mean?
   - Trivial examples
   - Basic things to know
3. Orders
   - Scott domains
   - dI-domains
4. Events
   - Event structures
   - Coherences
   - Hypercoherences
5. Conclusion

Down again

As for coherence spaces, boil down to qualitative domains with coherence and then...

A hypercoherence space is $E = (|E|, \Gamma)$ where:

- $|E|$ is a set: the web.
- $\Gamma \subseteq \mathcal{P}_{\text{fin}}(|E|)$ is s.t. $\{e\} \in \Gamma$: hypercoherence.

The *states* of E, noted $D(E)$, are subsets x of E

- consistent, i.e. $\forall X \subseteq_{\text{fin}} x : X \in \Gamma$

Well, almost simple: E is a reflexive undirected hypergraph, and $D(E)$ are its cliques.

Down again

As for coherence spaces, boil down to qualitative domains with coherence and then...

A hypercoherence space is $E = (|E|, \Gamma)$ where:

- $|E|$ is a set: the web.
- $\Gamma \subseteq \mathcal{P}_{\mathrm{fin}}(|E|)$ is s.t. $\{e\} \in \Gamma$: hypercoherence.

The *states* of E, noted $D(E)$, are subsets x of E

- consistent, i.e. $\forall X \subseteq_{\mathrm{fin}} x : X \in \Gamma$

Well, almost simple: E is a reflexive undirected hypergraph, and $D(E)$ are its cliques.

Products, very briefly

- $|E \times F| = |E| + |F| = \{0\} \times |E| \cup \{1\} \times |F|$;
- $X \in \Gamma(E \times F)$ iff $X \cap |E| = \emptyset \implies X \cap |F| \in \Gamma(F)$ and viceversa.
- In fact $D(E \times F) = D(E) \times D(F)$.
- 1 is the empty web.

Function Spaces and a Glimpse of Linear Logic

- We have a coherence space in which states are exactly the traces of stable functions;
- $|E \Rightarrow_s F| = D_{\mathrm{fin}}(E) \times |F|$;
- $(x, f) \frown (y, g)$ iff
  $x \cup y \in D(E) \implies (f \frown g \mathbin{\&} (f = g \implies x = y))$

Mmm, seems like two operations...

- Exponential $|!E| = D_{\mathrm{fin}}(E)$ with $x \frown y$ iff $x \cup y \in D(E)$;
- Linear arrow $|E \multimap F| = |E| \times |F|$ with $(d, f) \frown (e, g)$ iff
  $d \frown e \implies (f \frown g \mathbin{\&} (f = g \implies d = e))$.

Function Spaces and a Glimpse of Linear Logic

- We have a coherence space in which states are exactly the traces of stable functions;
- $|E \Rightarrow_s F| = D_{\text{fin}}(E) \times |F|$;
- $(x, f) \frown (y, g)$ iff
  $x \cup y \in D(E) \implies (f \frown g \,\&\, (f = g \implies x = y))$

Mmm, seems like two operations...

- Exponential $|!E| = D_{\text{fin}}(E)$ with $x \frown y$ iff $x \cup y \in D(E)$;
- Linear arrow $|E \multimap F| = |E| \times |F|$ with $(d, f) \frown (e, g)$ iff
  $d \frown e \implies (f \frown g \,\&\, (f = g \implies d = e))$.

Function Spaces and Again a Glimpse of Linear Logic

- We have a hypercoherence space in which states are exactly the traces of strongly stable functions;
- $|E \Rightarrow_s sF| = D_{\mathrm{fin}}(E) \times |F|$;
- $\Gamma(X)$ iff $\forall u \subseteq^*_{\mathrm{fin}} |E|.(u \lhd \pi_1(X) \implies u \in \Gamma(E))$ implies

$$(\pi_2(X) \in \Gamma \;\&\; (\#\pi_2(X) = 1 \implies \#\pi_1(X) = 1))$$

- $u \lhd X$ means $\forall e \in u. \exists v \in X. e \in v$ and $\forall v \in X. \exists e \in u. e \in v$.

Function Spaces and Again a Glimpse of Linear Logic

- We have a hypercoherence space in which states are exactly the traces of strongly stable functions;
- $|E \Rightarrow_s sF| = D_{\mathrm{fin}}(E) \times |F|$;
- $\Gamma(X)$ iff $\forall u \subseteq^*_{\mathrm{fin}} |E|.(u \lhd \pi_1(X) \implies u \in \Gamma(E))$ implies

$$(\pi_2(X) \in \Gamma \,\&\, (\#\pi_2(X) = 1 \implies \#\pi_1(X) = 1))$$

Mmm, seems again like two operations...

- Exponential $|!E| = D_{\mathrm{fin}}(E)$ with $X \in \Gamma$ iff $\forall u \subseteq^*_{\mathrm{fin}} |E|.(u \lhd X \implies u \in \Gamma(E)$;
- Linear arrow $|E \multimap F| = |E| \times |F|$ with $X \in \Gamma$ iff

$$\pi_1(X) \in \Gamma(E) \implies (\pi_2(X) \in \Gamma(F) \,\&\, (\#\pi_2(X) = 1 \implies \#\pi_1(X) = 1)). \quad (1)$$

Victory!

If E_i, E are flat hypercoherences, a function $f : \prod D(E_i) \to D(E)$ is sequential iff it is strongly stable.

At Last!

This slides have turned out to be a kind of (maybe too) fast tutorial. There is much still to say about denotational semantic. Anyway my personal interests in them is about:

- denoting proofs. Coherence spaces are full complete for MLL. Hypercoherence and MALL? Not quite the same, but it should be worked out.
- the Gustave function arise in proof theory as particular structures which do not correspond to sequential proofs. What other parallelism between the two worlds can be done?
- is there anything more to say about the relations between (strongly) stable functions and the linear ones which give rise to a bunch of linear adjoints?

Au Revoir!

Well, I hope I will be able to speak about these things another time, and why not, maybe even with answers?