Tutorial on Semantics Part III

A Survey of More Advanced Topics

Prakash Panangaden[1]

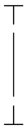
[1]School of Computer Science
McGill University
on sabbatical leave at
Department of Computer Science
Oxford University

Fields Institute, Toronto: 20th June 2011

Outline

A very special domain

- Recall, we needed a domain D such that $D \simeq [D \to D]$.
- This looks like a recursive construction but at the level of the category of domains rather than within a domain.
- First we give an ad-hoc description of how to construct this, then we give a more general theory of recursive domain equations later.
- Start with a simple domain $D_0 = \{\bot \leq \top\}$.



- The plan: inductively construct $D_{n+1} = [D_n \to D_n]$ and take the "limit."

The Details

- We need to build a “chain”

$$D_0 \longrightarrow D_1 \longrightarrow D_2 \longrightarrow \ldots D_n \longrightarrow D_{n+1} \longrightarrow \ldots$$

- but what is the arrow above?

An **embedding-projection pair** between domains D and E is a pair of functions $e : D \longrightarrow E$ and $p : E \longrightarrow D$ such that

$$p \circ e = id_D \text{ and } e \circ p \leq id_E.$$

- In fact e determines p, for continuous domains the formula is

$$p(x) = \bigvee_D \{y | e(y) \leq x\}.$$

- These e-p pairs compose and there is an obvious identity: so Scott domains and e-p pairs form a category.

More details

- Given an e-p pair $(e, p) : D \longrightarrow E$ we define a new e-p pair $(e', p') : [D \longrightarrow D] \longrightarrow [E \longrightarrow E]$ as follows:
  Let $f \in [D \longrightarrow D], g \in [E \longrightarrow E]$, then

$$e'(f) = e \circ f \circ p, \quad p'(g) = p \circ g \circ e.$$

- We start it off with the standard e-p pair $(e_0, p_0) : D \longrightarrow [D \longrightarrow D]$ given by $e_0(d) = x \mapsto d$ and $p_0(f) = f(\perp)$.
- We construct the usual inverse limit of the sequence above: D_∞; this is our goal

$$D_\infty \simeq [D_\infty \longrightarrow D_\infty].$$

- The inverse limit is all sequences $\{(x_0, x_1, \ldots, x_n, \ldots)\}$ with $x_n \in D_n$ and $p_n(x_{(n+1)} = x_n$.

T-algebras

- An *initial object* I in a category has a *unique* morphism to *every* other object.
- Given a functor $T : \mathcal{C} \longrightarrow \mathcal{C}$, a T-*algebra* is an object A and a morphism $\alpha : TA \longrightarrow A$.
- T-algebras form a category.

$$\begin{array}{ccc} TA & \xrightarrow{\alpha} & A \\ {\scriptstyle Tf}\downarrow & & \downarrow{\scriptstyle f} \\ TB & \xrightarrow[\beta]{} & B \end{array}$$

- An *initial* T-algebra; a must be unique:

$$\begin{array}{ccc} TI & \xrightarrow{\kappa} & I \\ {\scriptstyle Ta}\downarrow & & \downarrow{\scriptstyle a} \\ TA & \xrightarrow[\alpha]{} & A \end{array}$$

Lambek's Lemma

Theorem

An initial T-algebra $TI \xrightarrow{\kappa} I$ defines an isomorphism: $\exists\lambda : I \longrightarrow TI$ with $\kappa \circ \lambda = id_I$ and $\lambda \circ \kappa = id_{TI}$.

Proof

$$
\begin{array}{ccc}
TI & \xrightarrow{\kappa} & I \\
{\scriptstyle T\lambda}\downarrow & & \downarrow{\scriptstyle \lambda} \\
TTI & \xrightarrow[T\kappa]{} & TI \\
{\scriptstyle T\kappa}\downarrow & & \downarrow{\scriptstyle \kappa} \\
TI & \xrightarrow[\kappa]{} & I
\end{array}
$$

Fixed points for domain equations

- How do we solve equations like

$$L \simeq 1 + At \times L : \text{Lists of Atoms}$$

  or

$$T \simeq At \times F \text{ and } F \simeq 1 + T \times F : \text{Trees and Forests.}$$

  or even

$$D \simeq [D \longrightarrow D] \text{ ?}$$

- Lambek's lemma gives the clue: one can imitate fixed-point theory at the categorical level.
- Breakthrough idea: Use categories where the hom sets have order structure (Mitch Wand: 74,76)
- Develop systematially a theory of solving domain equations in such *order-enriched* categories: Plotkin and Smyth (79).
- Key result (roughly): one can lift results about limits and continuity from homsets to the category.

Continuity in analysis

Definition

A function $f : \mathbf{R} \longrightarrow \mathbf{R}$ is continuous at x_0 if $\forall \epsilon > 0 \exists \delta > 0$ such that $\forall x \in (x_0 - \delta, x_0 + \delta)$, $|f(x) - f(x_0)| \leq \epsilon$.

Computational version

A function f is continuous (at x_0) if whenever I prescribe a tolerance for the accuracy of the answer, there is a limit on the accuracy of the input that guarantees that the answer will be within the prescribed tolerance.

Continuity is a coomputability concept. Might computability be a topological concept?

The Scott topology

- The use of the word “continuity” suggests topology.

The Scott topology

An open set $U \subset D$ in a continuous domain has the property that if $X \subset D$ is a directed set with $\bigvee X \in U$ then $X \cup U \neq \emptyset$.

- This is just like the open sets of real analysis.
- If a sequence converges to a limit inside some open interval then the sequence itself must enter the open interval.

Observability

- Open sets are the observable properties!
- The axioms of topology make sense from the point of view of being "physically" testable properties: Smyth.
- The computable functions are continuous: to obtain a finite piece of the answer one needs only a finite piece of the input.
- In fact concepts like compactness can be given a computational meaning (Escardo): one can exhaustively search infinite sets by exploiting compactness!
- These concepts have even been fruitfully used in physics [Keye Martin, P. Comm. Math. Phys. 2006].

The Lawson topology

- With the Scott topology, compactness cannot be used effectively in domains with a least element.
- Lawson topology: take and Scott-open set U, take any finite set F and take as basis for the topology seet sets of the form $U \setminus \uparrow F$, where $\uparrow F = \{x | \exists y \in F,\ y \leq x\}$.
- The Lawson topology carries some negative information.
- For any continuous lattice the Lawson topology is compact and Hausdorff.
- For an algebraic dcpo the Lawson topology is metrizable.
- For streams with the prefix order

$$d(x,y) = 2^n,\ \text{where } x[n] \neq y[n] \text{ and } \forall m < n,\ x[m] = y[m].$$

Classical Stone duality

- Every boolean algebra is isomorphic to a boolean algebra of sets: Stone representation theorem.
- Given $\mathcal{B}$ we construct the set of *ultrafilters* (or maps into the two-element boolean algebra) ordered by inclusion.
- Much more is true: we can make the collection of ultrafilters into a topological space $\mathcal{S}$: for every $x \in \mathcal{B}$ we define U_x to be the set of ultrafilters that contain x. This gives the base for a topology.
- With this topology $\mathcal{S}$ is Hausdorff, compact and has a base of *closed and open* (clopen) sets: it is called a *Stone space*.
- If there is a BA homomorphism $h : \mathcal{B}_1 \longrightarrow \mathcal{B}_2$ we get a *continuous* map $\hat{h} : \mathcal{S}_2 \longrightarrow \mathcal{S}_1$ by composition.

The categorical picture

- Call the category of boolean algebras $\mathbf{BA}$ and the category of Stone spaces $\mathbf{Stone}$.
- Then one has functors from $\mathbf{BA}$ to $\mathbf{Stone}^{op}$.
- The composites are naturally isomorphic to the identity functors.
- As categories $\mathbf{Stone}^{op}$ and $\mathbf{BA}$ are *the same*.
- One has two views of the same structures: algebraic and topological.
- Other examples: Compact Hausdorff spaces and $C*$-algebras.
- Vector spaces and itself!
- Many, many, many more...
- Denotational semantics and axiomatic semantics.

Predicate transformers: Dijkstra

- In operational semantics: given a state (or set of states) and a transition system (which may be nondeterministic) what are the next states after the execution of a command.
- In predicate transformers: if *after* the execution of a command a property P holds what *must have been true* before? The **weakest precondition**.
- Note the backward flow in wp semantics.
- Given two continuous domains D and E, viewed as topological spaces with the open sets written $\mathcal{O}_D$ and $\mathcal{O}_E$, a **predicate transformer** is a *strict, continuous and multiplicative* map $p : \mathcal{O}_E \to \mathcal{O}_D$.
- We can (but won't) formalize what a state transformer is as well.
- Duality: The category of state transformers is equivalent to the (opposite of) the category of predicate transformers. (Smyth, Plotkin)

Duality more generally

- This can be extended to the realm of probabilistic programs and *expectation transformers*. (Kozen)
-

Logic	Probability
States s	Distributions μ
Formulas P	Random variables f
Satisfaction $s \models P$	Integration $\int f \mathrm{d}\mu$

- One can define a generalized transition system as a *co-algebra* and a (modal) logic as an algebra.
- One obtains a general Stone-type duality between co-algebras and algebras: between generalized transition systems and modal logics.
- Bonsangue, Kurz, Moss, Pattinson, Schröder, Rutten, Jacobs, Silva, Worrell, Pavlovic, Mislove, Simpson, Kupke, Bezhanishvili and Panangaden.

Locales

- One can study topological spaces in terms of their complete lattice of open sets.
- These lattices are complete and satisfy the infinite distributivity law:
$$x \wedge \bigvee S = \bigvee \{x \wedge s | s \in S\}.$$
They are called *frames*.
- The category of frames has morphisms that preserve finite meets and arbitrary joins: the spirit of topology.
- Locales are the opposite category of frames.
- Many ideas are clarified by taking the dual view and working with locales instead of spaces and points.
- This is the point-free of topology.
- It shall be very fruitful (one day) in probability theory.
- Fantastic book: Stone Spaces by Peter Johnstone.

Domain theory in logical form

- A famous paper by Abramsky with the above title spells out and implements the following programme based on the perspective of Stone duality.
- Define a metalanguage for types and terms (programs)
- Interpret types as domains and terms as elements in appropriate domains: denotational semantics.
- Give a logical interpretation of the same language: types are propositional theories, the finite elements are the propositions.
- In the logical view, terms are described by axiomatizing satisfaction. A modal logic of programs.
- The two interpretations are shown to be Stone duals.
- Ties together semantics, logic and verification.

Axiomatic domain theory

- What should a category of domains be?
- Categories of domains should have enough structure to support the solution of recursive domain equations.
- Axiomatic domain theory: require products, exponentials, sums, limits and colimits, the ability to solve recursive domain equations.
- Also require that one has Stone-type duality to give a logic of observable properties.
- Most of the emphasis in axiomatic domain theory was finding the right axioms for solving recursive domain equations: fundamental early work by Alex Simpson and Marcelo Fiore.
- It gave an axiomatic framework for studying adequacy in extensions of PCF.
- It also provided reasoning principles for recursive types.

Synthetic domain theory

- Scott: domains should be "sets" perhaps in another mathematical universe.
- Toposes are alternative mathematical universes or alternative set theories.
- From the "inside" it looks like you are doing set theory.
- From the "outside" it looks very different.
- Can one build toposes where one gets domains by just doing naive set theory inside the topos?

What are effects?

- Consider a computation that produces a result but also updates a store or produces output.
- These were called “side effects” suggesting that they happened on the side.
- Moggi initiated the systematic study of these through the theory of monads.
- This was so influential that they even became a languge mechanism in Haskell.

Plotkin and Power's theory of effects

- Describe effects not as monads but as a particular kind of formalism called a Lawvere theory which puts the emphasis on operations and equational laws for the effects.
- It is equivalent to working with monads but it is much easier to see how to combine different effects.
- The key achievement is to provide a modular way to combine semantics for different kinds of effects: update, IO, jumps, nondterminism, probability etc.

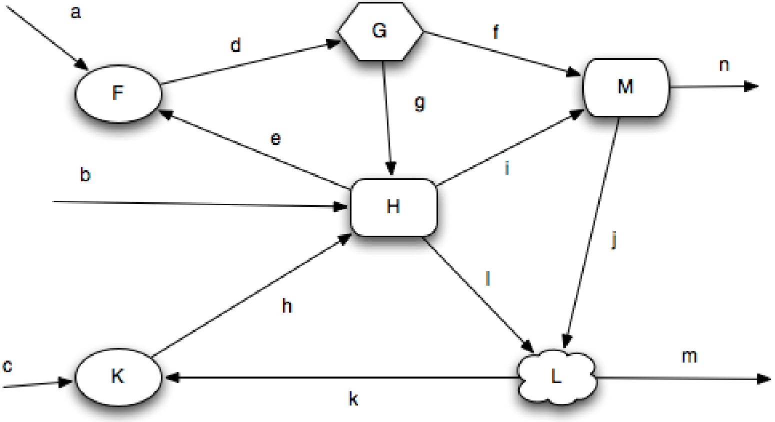
What is concurrency?

- Describe *autonomous, interacting* computational entities.
- The computational agents are not necessarily performing a single task.
- In parallel programming one wants to exploit parallelism to mask latency.
- Here one is interested in modelling that, but also distributed transactions, operating systems, communication protocols and other tasks.
- No universally-acknowledged fundamental paradigm: synchronous vs asynchronous, communication by message passing, by shared variables, by broadcast, mobile or not.
- Proliferation of formalisms, semantic models and logics.

Kahn-McQueen networks

- Network of autonomous computing agents connected by unbounded FIFO buffers as comunication channels. Channels are named, point-to-point and directional.
- Each agent runs a sequential program. Communication primitives: **read c** and **write e to c**. Read is *blocking*.

An example network



$$d = F(a,e),\ f = G_1(d),\ g = G_2(d),\ i = H_2(b,g,h).$$

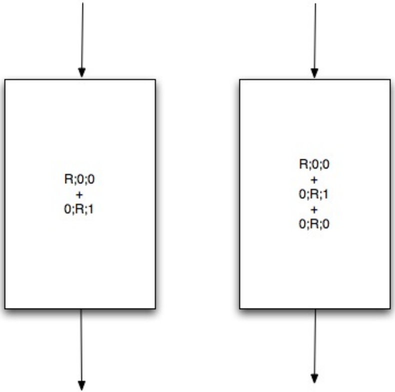
The Kahn principle

- Each agent computes a continuous function from input *streams* to output *streams*. Not functional at the token level. The network is described by a set of equations.
- The networks may have cycles, so the set of equations may be recursive.
- Operational semantics is by token pushing.
- Denotational semantics is by least fixed point theory: Kahn principle.

What happens if we introduce nondeterminism?

- The input-output relation is not a function.
- We cannot just work with relations.
- The IO relation is not compositional.

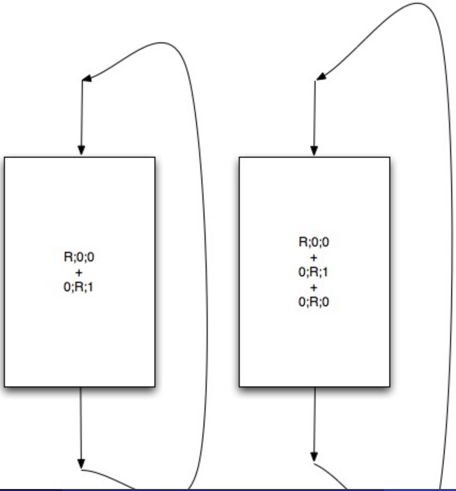
An example



ε	$\varepsilon, 0$
0	$00, 01$

An example

Now the one on the left outputs ε or 01 but the one on the right can output 00 as well as the previous two possibilities.



- There is a rich theory of expressive power of nondeterministic dataflow but this is not the place for it.
- Fully abstract semantics based on traces were developed by several workers but Bengt Jonsson deservedly gets the credit for doing it first.
- One needs new abstractions to deal with concurrent computation.
- Process calculi were started by Milner as foundational calculi for concurrent computation.
- Independently Hoare invented CSP as a concurrent computation paradigm.
- Concurrency theory needs a 30 hour tutorial!

Powerdomains

- Concurrency almost always introduces indeterminacy. Plotkin introduced the domain theoretic analogue of the powerset.
- How to order sets of elements from a domain?
- Consider three programs: P, Q, R. P outputs 1, Q may output 1 or may loop forever and R just loops forever. Are they equivalent?
- One view P and Q are the same since the set of possible results are the same. One can define an order on sets based on this intuition and obtain a domain called the Hoare or lower powerdomain.
- Another view Q and R are the same since we cannot guarantee anything about their termination behaviour. The powerdomain based on this intuition is called the Smyth or upper powerdomain.
- Finally, all three are different: this leads to the Plotkin or convex powerdomain.

The Plotkin powerdomain

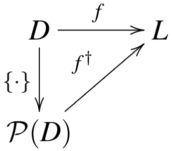
- For flat domains one can formalize the third powerdomain with the Egli-Milner order

$$A \sqsubseteq_{EM} B \text{ iff } \forall x \in A \exists y \in B,\ x \leq y \wedge \forall y \in B \exists x \in A,\ x \leq y.$$

- For non-flat domains D, one starts with all non-empty finite subsets of D and orders them by the EM order; this gives a pre-order.
- To construct the Plotkin powerdomain $\mathcal{P}(D)$ we form the ideal completion of this preorder.
- Viewed as subsets of D the elements of $\mathcal{P}(D)$ are non-empty, convex, Lawson-compact subsets of D ordered by the EM order.
- Lawson compactness captures the idea that the programs are finitely-branching.

Algebraic properties

- One can define a continuous operation $\cup : \mathcal{P}(D) \times \mathcal{P}(D) \longrightarrow \mathcal{P}(D)$ (union) which makes $\mathcal{P}(D)$ a semi-lattice and a map $\{\cdot\} : D \longrightarrow \mathcal{P}(D)$ which is the continuous analogue of the singleton embedding.
- There is a canonical way of extending any continuous map $f : D \longrightarrow L$ to $f^{\dagger} : \mathcal{P}(D) \longrightarrow L$ in such a way that the diagram below commutes



Categorical properties

- If D is a Scott domain then $\mathcal{P}(D)$ *may not be* a Scott domain.
- If one wants to combine nondeterminism with higher types one needs a CCC which is closed under the action of $\mathcal{P}(\cdot)$.
- Plotkin found the category SFP which is a CCC of algebraic domains which is closed under the action of forming the convex powerdomain.
- Smyth showed that this is the largest CCC of ω-algebraic domains.

Probabilistic systems

- Probability is important to formalize many kinds of systems.
- Much research on discrete probabilistic systems.
- In the late 1980s Claire Jones and Plotkin developed probabilistic powerdomains.
- We still do not know any CCC of continuous domains which is closed under the formation of the probabilistic powerdomain.
- In the last 1990s Blute, Desharnais, Edalat, P. introduced labelled transition systems on *continuous* state spaces: labelled Markov processes and showed some striking results about logic and bisimulation.
- Desharnais et al. constructed a *universal* LMP by solving a recursive domain equation in the category of Lawson compact continuous domains.
- Lots of hard mathematics needed to combine probability and nondeterminism; an ongoing project.