DOMAIN THEORY IN LOGICAL FORM (To appear in Proceedings of LICS '87)

Samson Abramsky

Department of Computing
Imperial College of Science and Technology

November 10, 2006

Abstract

A metalanguage for denotational semantics is given a logical interpretation: types are interpreted as propositional theories; terms (in an extended typed λ-calculus, denoting elements of types) are embedded in an endogenous program logic, which characterises their behaviour in terms of which properties they satisfy; terms (in an extension of the algebraic metalanguage of cartesian closed categories, denoting morphisms between types) are embedded in an exogenous logic, which characterises their behaviour as predicate transformers or modal operators. This interpretation is related to the standard domain-theoretic one via the machinery of Stone duality, and an exact correspondence is obtained. Some applications to logics for specific computational situations (e.g. concurrency) are mentioned.

1 Introduction

The classic Stone Representation Theorem for Boolean algebras [25] is the prototype for a wide class of "Stone-type duality theorems" [11]. The general form of these theorems is to assert an equivalence between a category of topological spaces and (the opposite of) a category of lattices and lattice-homomorphisms. In one direction, we pass from a space X to a lattice derived from the topology on X; we can think of this as a lattice of *properties* over X, since its elements are just subsets of X. In the other direction, starting from an abstract lattice A, which we can think of as a logical theory of properties or "observable quantities", we pass to a space whose points are the consistent, complete theories over A, i.e. the "point-like" bundles of properties over A.

The importance of Stone Duality for computer science is that *it provides the right framework for understanding the relationship between denotational semantics and program logic.* (As far as I know, this point was first made in [22], and also in unpublished work of Gordon Plotkin.)

Denotational semantics is always based, more or less explicitly, on a typed functional metalanguage. The types are interpreted as topological spaces (usually domains in the sense of Scott [20, 21], but sometimes metric spaces, as in [3, 16]), while the terms denote elements of or functions between these spaces. A *program logic* comprises an assertion language of formulas for expressing properties of programs, and an interface between these properties and the programs themselves. Two main types of interface can be identified [18]:

Endogenous logic In this style, formulas describe properties pertaining to the "world" of a single program. Notation:

$$P \models \phi$$

where P is a program and ϕ is a formula. Examples: temporal logic as used e.g. in [18]; Hennessy-Milner logic [8]; type inference [5].

Exogenous logic Here, programs are embedded in formulas as *modal operators.* Notation:

$$[P]\phi$$

where P is now a program denoting a function or relation. Examples: dynamic logic [7], including as special cases Hoare logic [10], since "Hoare triples" $\{\phi\}P\{\psi\}$ can be represented by

$$\phi \to [P]\psi,$$

and Dijkstra's wlp-calculus [6], since $wlp(P, \psi)$ can be represented as $[P]\psi$. (Total correctness assertions can also be catered for; see [7].)

Extensionally, formulas denote sets of points in our denotational domains, i.e. ϕ is a syntactic description of $\{x|x \text{ satisfies } \phi\}$. Then $P \models \phi$ can be interpreted as $x \in U$, where x is the point denoted by P, and U is the set denoted by ϕ. Similarly, $[M]\phi$ can be interpreted as $f^{-1}(U)$, where f is the function denoted by M (and elaborations of this when M denotes a relation or multifunction). In this way, we can give a topological interpretation of program logic.

But this is not all: duality cuts both ways. We can also use it to give a *logical interpretation of denotational semantics.* Rather than starting with the denotational domains as spaces of points, and then interpreting formulas as sets of points, we can give an axiomatic presentation of the topologies on our spaces, viewed as abstract lattices (logical theories), and then reconstruct the points from the properties they satisfy. In other words, we can present denotational semantics in axiomatic form, as a logic of programs. This has a number of attractions:

- It unifies semantics and program logic in a general and systematic setting.
- It extends the scope of program logic to the entire range of denotational semantics – higher-order functions, recursive types, powerdomains etc.
- The syntactic presentation of recursive types, powerdomains etc. makes these constructions more "visible" and easier to calculate with.
- The construction of "points", i.e. denotations of computational processes, from the properties they satisfy is very compatible with work currently being done in a mainly operational setting in concurrency [8, 26] and elsewhere [4], and offers a promising approach to unification of this work with denotational semantics.

2 Domains As Propositional Theories

The setting we take for our work in this paper is **SDom**, the category of Scott domains [20, 21], i.e. the consistently complete algebraic cpo's. (However, everything should carry over to **SFP** [17].) Such domains D can also be viewed as spaces with the Scott topology $\Omega(D)$, with sub-basic open sets of the form $\uparrow b \equiv \{d \in D | b \sqsubseteq d\}$, b a finite element of D. These spaces are *coherent* [11], meaning that they are determined by the distributive lattice of compact-open subsets $(K\Omega(D), \subseteq)$, i.e. sets of the form $\uparrow X$, where X is a finite set of finite elements of D. The advantage of the coherent case for our purposes is that we can present our logic in a finitary form. Given a distributive lattice A, its space of points is constructed as $\mathrm{Spec}\, A$, the set of prime filters over A, with topology generated by basic sets

$$U_a \equiv \{x \in \mathrm{Spec}\, A | a \in x\} \quad (a \in A).$$

The domain ordering can be recovered as the *specialisation ordering*, which in terms of this representation of $\mathrm{Spec}\, A$ is just inclusion of filters.

Some Trade Secrets of Domains

An algebraic domain is coherent (in the sense of [11]) iff it is "2/3 SFP" in the terminology of [17]. The topological significance of the "MUB axioms" [17] is that they ensure that the compact-open sets are closed under finite intersections, and hence form a distributive lattice which generates the topology. Moreover, we can recover $K(D)$, the poset of finite elements of the domain D, from $A = K\Omega(D)$ as follows: an element $a \in A$ is *prime* if $a \leq b \vee c$ implies $a \leq b$ or $a \leq c$; and a is *consistent* if $a \neq 0$. The sub-poset of consistent primes is denoted by $\mathit{CPR}(A)$. Now the consistent primes of $K\Omega(D)$ have the form $\uparrow b$, $b \in K(D)$. Thus:

$$K(D) \;\cong\; (\mathit{CPR}(K\Omega(D)))^{\mathrm{op}}.$$

Also, there are "enough" primes, in the sense that each $U \in K\Omega(D)$ can be written as a disjunction of primes:

$$\uparrow X \;=\; \bigcup\{\uparrow b | b \in X\}.$$

We begin by introducing the first part of a metalanguage for denotational semantics, the *type expressions*, with syntax

$$\sigma \quad ::= \quad \mathbf{1} \mid \sigma \times \tau \mid \sigma \rightarrow \tau \mid \sigma \oplus \tau \mid (\sigma)_\perp \mid P_u\sigma \mid P_l\sigma \mid t \mid \mathrm{rec}\, t.\sigma$$

where t ranges over type variables, and σ, τ over type expressions.

The standard way of interpreting these expressions is as objects of **SDom** (more generally as cpo's, but **SDom** is closed under all the above constructions as a subcategory of **CPO**). Thus for each type expression σ we define a domain $\mathcal{D}(\sigma) = (D(\sigma), \sqsubseteq_\sigma)$ in **SDom**; $\sigma \times \tau$ is interpreted as product, $\sigma \to \tau$ as function space, $\sigma \oplus \tau$ as coalesced sum, $(\sigma)_\perp$ as lifting, $P_u\sigma$ and $P_l\sigma$ as the upper and lower (or Smyth and Hoare) powerdomains, and $\mathrm{rec}\, t.\sigma$ as the solution of the domain equation

$$t = \sigma(t),$$

i.e. as the initial fixpoint of an endofunctor over **SDom**. Other constructions (e.g. strict function space, smash product) can be added to the list.

So far, all this is standard ([17, 23]). Now we begin our alternative approach. For each type expression σ, we shall define a propositional theory $\mathcal{L}(\sigma) = (L(\sigma),\ \leq_\sigma,\ =_\sigma)$, where:

- $L(\sigma)$ is a set of formulae
- $\leq_\sigma$, $=_\sigma$ are the relations of logical *entailment* and *equivalence* between formulae.

$\mathcal{L}(\sigma)$ is defined inductively via formation rules, axioms and inference rules in the usual way. We give *selected examples* of the definitions.

Formation Rules (selected)

$$t, f \in L(\sigma)$$

$$\frac{\phi, \psi \in L(\sigma)}{\phi \wedge \psi, \phi \vee \psi \in L(\sigma)}$$

$$\frac{\phi \in L(\sigma),\ \psi \in L(\tau)}{(\phi \times \psi) \in L(\sigma \times \tau),\ (\phi \to \psi) \in L(\sigma \to \tau)}$$

$$\frac{\phi \in L(\sigma),\ \psi \in L(\tau)}{(\phi \oplus \psi) \in L(\sigma \oplus \tau)}$$

$$\frac{\phi \in L(\sigma)}{\Box\phi \in L(P_u\sigma),\ \Diamond\phi \in L(P_l\sigma)}$$

$$\frac{\phi \in L(\sigma[\mathrm{rec}\, t.\sigma/t])}{\phi \in L(\mathrm{rec}\, t.\sigma)}$$

We should think of $(\phi \to \psi)$, $\Box\phi$ etc. as "constructors" or "generators", which build basic formulae at complex types from arbitrary formulae at simpler types. Note that no constructors are introduced for recursive types; we are taking advantage of the observation, familiar from work on information systems [14], that if we work with preorders it is easy to solve domain equations up to *identity*.

Auxiliary Predicates

Before proceeding to the axiomatisation proper, we shall define some auxiliary predicates on formulas. These will be used as side-conditions on a number of axioms and rules (e.g. $(\to - \vee - R)$ below). Thus it is important that they are recursive predicates, defined syntactically on formulae. The main predicates we define are:

- PNF(ϕ): ϕ is in *prime normal form*, defined by the conditions that disjunctions only occur in ϕ immediately under $\Box$, and that occurrences of $\oplus$ are of one of the forms $(\phi \oplus f)$, $(f \oplus \psi)$.

Then for ϕ in PNF, we define:

- CON(ϕ): ϕ is *consistent*, i.e. so that we have

  $$\mathrm{CON}(\phi) \;\Leftrightarrow\; \neg(\phi \leq f) \;\Leftrightarrow\; [\![\phi]\!] \neq \emptyset$$

  (where $[\![]\!]$ is the semantics to be introduced below).

- T(ϕ): ϕ requires *termination*, i.e. so that we have

  $$\mathrm{T}(\phi) \;\Leftrightarrow\; \neg(t \leq \phi) \;\Leftrightarrow\; \bot \notin [\![\phi]\!].$$

Of these, the idea of formal consistency, and its definition for function spaces, go back to [12], and also play a major role in [20, 21]. The other predicates, as syntactic conditions on expressions, are apparently new (and in the presence of the type constructions we are considering, specifically function space and coalesced sum, the definitions of CON and T are *mutually recursive*). We illustrate the definitions of CON and T by giving the clauses for function types and coalesced sum:

- $\mathrm{CON}(\bigwedge_{i\in I}(\phi_i \to \psi_i) \;\Leftrightarrow\; \forall J \subseteq I.$

$$\mathrm{CON}(\bigwedge_{j\in J}\phi_j) \;\Rightarrow\; \mathrm{CON}(\bigwedge_{j\in J}\psi_j)$$

- $\mathrm{T}(\phi \rightarrow \psi) \;\;\Leftrightarrow\;\; \mathrm{CON}(\phi)\ \&\ \mathrm{T}(\psi)$
- $\mathrm{CON}(\bigwedge_{i\in I}(\phi_i \oplus f) \wedge \bigwedge_{j\in J}(f \oplus \psi_j)) \;\;\Leftrightarrow$

  $\neg(\mathrm{T}(\bigwedge_{i\in I}\phi_i)\ \&\ \mathrm{T}(\bigwedge_{j\in J}\psi_j))$

  $\&\ \mathrm{CON}(\bigwedge_{i\in I}\phi_i)\ \&\ \mathrm{CON}(\bigwedge_{j\in J}\psi_j)$
- $\mathrm{T}(\phi \oplus f) \;\;\Leftrightarrow\;\; \mathrm{T}(f \oplus \phi) \;\;\Leftrightarrow\;\; \mathrm{T}(\phi)$.

Once we have defined CON and T, we can introduce a predicate CPR(ϕ), which holds if ϕ is a disjunction of formulae ϕ_i in PNF such that for some i CON(ϕ_i) holds.

Now we turn to the axiomatization. The axioms of our logic are all "polymorphic" in character, i.e. they arise from the type constructions uniformly over the types to which the constructions are applied. Thus we omit type subscripts.

The axioms fall into two main groups.

Logical Axioms (selected)

These give each $\mathcal{L}(\sigma)$ the structure of a distributive lattice.

$(\leq -ref)\quad \phi \leq \phi$

$(\leq -trans)\quad \dfrac{\phi \leq \psi,\ \psi \leq \chi}{\phi \leq \chi}$

$(= -I)\quad \dfrac{\phi \leq \psi,\ \psi \leq \phi}{\phi = \psi}$

$(\wedge - I)\quad \dfrac{\phi \leq \psi_1,\ \phi \leq \psi_2}{\phi \leq \psi_1 \wedge \psi_2}$

$(\wedge - E - L)\quad \phi \wedge \psi \leq \phi$

$(\wedge - dist)\quad \phi \wedge (\psi \vee \chi) \leq (\phi \wedge \psi) \vee (\psi \wedge \chi)$

Type-specific Axioms (selected)

These articulate each type construction, by showing how its generators interact with the logical structure.

$$(\to - \leq) \quad \frac{\phi' \leq \phi,\ \psi \leq \psi'}{(\phi \to \psi) \leq (\phi' \to \psi')}$$

$$(\to - \wedge) \quad (\phi \to \bigwedge_{i \in I} \psi_i) = \bigwedge_{i \in I} (\phi \to \psi_i)$$

$$(\to - \vee - L) \quad (\bigvee_{i \in I} \phi_i \to \psi) = \bigwedge_{i \in I} (\phi_i \to \psi)$$

$$(\to - \vee - R) \quad (\phi \to \bigvee_{i \in I} \psi_i) = \bigvee_{i \in I} (\phi \to \psi_i) \quad (CPR(\phi))$$

$$(\Box - \leq) \quad \frac{\phi \leq \psi}{\Box\phi \leq \Box\psi}$$

$$(\Box - \wedge) \quad \Box \bigwedge_{i \in I} \phi_i = \bigwedge_{i \in I} \Box\phi_i$$

$$(\Box - f) \quad \Box f = f$$

The axiom $(\Box - f)$ exemplifies the possibilities for fine-tuning in our approach. It corresponds exactly to the *omission* of the empty set from the upper powerdomain.

To make precise the sense in which this axiomatic presentation is equivalent to the usual denotational construction of domains we define, for each (closed) type expression σ, an interpretation function

$$[\![]\!]_\sigma : L(\sigma) \longrightarrow K\Omega(\mathcal{D}(\sigma))$$

by

$$[\![\phi \wedge \psi]\!]_\sigma = [\![\phi]\!]_\sigma \cap [\![\psi]\!]_\sigma$$

$$[\![t]\!]_\sigma = D(\sigma) = 1_{K\Omega(\mathcal{D}(\sigma))}$$

$$[\![(\phi \to \psi)]\!]_{\sigma \to \tau} = \{f \in D(\sigma \to \tau) | f([\![\phi]\!]_\sigma) \subseteq [\![\psi]\!]_\tau\}$$

$$[\![\Box\phi]\!]_{P_u\sigma} = \{S \in D(P_u\sigma) | S \subseteq [\![\phi]\!]_\sigma\}$$

$$[\![\Diamond\phi]\!]_{P_l\sigma} = \{S \in D(P_l\sigma) | S \cap [\![\phi]\!]_\sigma \neq \emptyset\}$$

$$\vdots$$

etc. Then for $\phi, \psi \in L(\sigma)$, we define

$$\mathcal{D}(\sigma) \models \phi \leq \psi \;\equiv\; [\![\phi]\!]_\sigma \subseteq [\![\psi]\!]_\sigma.$$

Theorem 1 (Soundness and Completeness) *For all $\phi, \psi \in L(\sigma)$:*

$$\mathcal{L}(\sigma) \vdash \phi \leq \psi \iff \mathcal{D}(\sigma) \models \phi \leq \psi.$$

Now we define

$$\mathcal{LA}(\sigma) \;\equiv\; (L(\sigma)/=_\sigma, \leq_\sigma / =_\sigma),$$

the *Lindenbaum algebra* of $\mathcal{L}(\sigma)$.

Theorem 2 (Stone Duality) *$\mathcal{LA}(\sigma)$ is the Stone dual of $\mathcal{D}(\sigma)$, i.e.*

$$\begin{array}{rl} (i) & \mathcal{D}(\sigma) \;\cong\; \mathrm{Spec}\,\mathcal{LA}(\sigma) \\ (ii) & K\Omega(\mathcal{D}(\sigma)) \;\cong\; \mathcal{LA}(\sigma). \end{array}$$

The proofs of these results are rather long, and will be presented elsewhere. They involve the following main steps, for each type construction:

- A *Normal Form* lemma, to the effect that every formula can be proved equivalent to a disjunction of consistent primes (in the syntactic sense of our auxiliary predicates). This is the syntactic counterpart of the semantic "Trade Secret" mentioned above.

- The auxiliary predicates are proved *adequate*, i.e. sound and complete with respect to the semantic interpretation.

- One then proves a *prime completeness theorem*, using the standard description of the ordering on finite elements of $\mathcal{D}(\sigma)$ to prove completeness for syntactically prime formulae. Combined with the normal form lemma, this easily yields the full Completeness Theorem. This in turn, combined with the fact that all elements of $K\Omega(\mathcal{D}(\sigma))$ are definable by formulae in $L(\sigma)$, yields the Stone Duality theorem.

- Finally, to carry these results over to recursive types, we must ensure that our syntactic type constructions on theories are functorial, and commute with the corresponding semantic constructions on domains. Fortunately, this needs to be done only with respect to embeddings induced by a partial ordering on theories, following ideas of [14].

3 Programs As Elements: Endogenous Logic

We extend our metalanguage for denotational semantics to include typed terms.

Syntax (selected examples)

We have a set of variables $Var(\sigma)$ for each σ.

$$\frac{x \in Var(\sigma), \quad M : \tau}{\lambda x.M : \sigma \to \tau}$$

$$\frac{M : \sigma \to \tau, \quad N : \sigma}{MN : \tau}$$

$$\frac{M : \sigma}{\{\!|M|\!\}_u : P_u\sigma}$$

$$\frac{M : \sigma \to P_u\tau}{M_u^\dagger : P_u\sigma \to P_u\tau}$$

$$\frac{M, N : P_u\sigma}{M \uplus_u N : P_u\sigma}$$

$$\frac{M : P_u\sigma, \quad N : P_u\tau}{M \otimes_u N : P_u(\sigma \times \tau)}$$

We write $\Lambda(\sigma)$ for the set of terms of type σ. Note that $\{\!|.|\!\}$, $\_^\dagger$ arise from the adjunction defining the powerdomain construction; $\uplus$ is the operation of the free algebras for this adjunction; while $\otimes$ is the universal map for the tensor product with respect to this operation [9].

We now introduce an endogenous program logic with assertions of the form

$$M, \Gamma \vdash \phi$$

where $M : \sigma$, $\phi \in L(\sigma)$, and $\Gamma \in \prod_\sigma \{Var(\sigma) \rightarrow L(\sigma)\}$ gives *assumptions* on the free variables of M.

Axiomatisation (Selected Examples)

$$x, \Gamma[x \mapsto \phi] \vdash \phi$$

$$\frac{M, \Gamma \vdash \phi \quad \mathcal{L} \vdash \phi \leq \psi}{M, \Gamma \vdash \psi}$$

$$\frac{M, \Gamma[x \mapsto \phi] \vdash \psi}{\lambda x.M, \Gamma \vdash (\phi \rightarrow \psi)}$$

$$\frac{M, \Gamma \vdash (\phi \rightarrow \psi) \quad N, \Gamma \vdash \phi}{MN, \Gamma \vdash \psi}$$

$$\frac{M, \Gamma \vdash \phi}{\{\!|M|\!\}_u, \Gamma \vdash \Box\phi}$$

$$\frac{M, \Gamma \vdash (\phi \rightarrow \Box\psi)}{M_u^\dagger, \Gamma \vdash (\Box\phi \rightarrow \Box\psi)}$$

$$\frac{M, \Gamma \vdash \Box\phi \quad N, \Gamma \vdash \Box\phi}{M \uplus_u N, \Gamma \vdash \Box\phi}$$

$$\frac{M, \Gamma \vdash \Box\phi \quad N, \Gamma \vdash \Box\psi}{M \otimes_u N, \Gamma \vdash \Box(\phi \times \psi)}$$

Following standard ideas, we can give a denotational semantics for this metalanguage, in the form of a map

$$[\![]\!]_\sigma : \Lambda(\sigma) \longrightarrow Env \longrightarrow \mathcal{D}(\sigma)$$

where $Env \equiv \prod_\sigma \{Var(\sigma) \rightarrow \mathcal{D}(\sigma)\}$ is the set of *environments*. We can use this to give a semantics for assertions:

$$M, \Gamma \models \phi \ \equiv \ \forall \rho \in Env.\, \rho \models \Gamma \Rightarrow [\![M]\!]_\sigma \rho \models \phi$$

where

$$\rho \models \Gamma \;\equiv\; \forall x \in Var.\, \rho x \models \Gamma x$$

and for $d \in D(\sigma)$, $\phi \in L(\sigma)$:

$$d \models \phi \;\equiv\; d \in [\![\phi]\!]_\sigma.$$

Theorem 3 *The Endogenous logic is sound and complete: for all M, Γ, ϕ:*

$$M, \Gamma \vdash \phi \iff M, \Gamma \models \phi.$$

We can state this result more sharply in terms of Stone Duality: it says that

$$\eta_\sigma(\{[\phi]_{=_\sigma} \mid M, \Gamma \vdash \phi\}) = [\![M]\!]_\sigma \rho,$$

where

$$\eta_\sigma : \mathrm{Spec}\, \mathcal{LA}(\sigma) \;\cong\; \mathcal{D}(\sigma)$$

is the component of the natural isomorphism arising from Theorem 2; i.e. that we recover the point of $\mathcal{D}(\sigma)$ given by the denotational semantics of M from the properties we can prove to hold of M in our logic.

4 Programs As Morphisms: Exogenous Logic

Now we introduce a second extension of our denotational metalanguage, which is based on the algebraic metalanguage for cartesian closed categories [19, 13], just as the language of the previous section is an extended typed λ-calculus. Terms are sorted on *morphism types* (σ, τ), with formation rules exemplified by

$$\frac{f : (\sigma \times \tau, \upsilon)}{\Lambda(f) : (\sigma, \tau \to \upsilon)}$$

$$Ap : ((\sigma \to \tau) \times \sigma, \tau).$$

We then form a *dynamic logic* $\mathcal{DL}$, with syntax given by

- $L(\sigma) \subseteq DL(\sigma)$

- $\dfrac{f : (\sigma, \tau) \ \psi \in DL(\tau)}{[f]\psi \in DL(\sigma).}$

We illustrate the axiomatisation of $\mathcal{DL}(\sigma)$ by two examples:

$$\frac{(\phi \times \psi) \ \leq \ [f]\chi}{\phi \ \leq \ [\Lambda(f)](\psi \to \chi)}$$

$$(\phi \to \psi) \times \phi \ \leq \ [Ap]\psi$$

We write the set of (closed) terms of morphism type (σ, τ) as $\Lambda(\sigma, \tau)$. Once again, using standard ideas we can define a semantic function

$$[\![ ]\!]_{\sigma,\tau} : \Lambda(\sigma, \tau) \longrightarrow \mathbf{SDom}(\mathcal{D}(\sigma), \mathcal{D}(\tau))$$

which interprets each term as a morphism in the appropriate hom-set. Now we define

$$\models \phi \ \leq_\sigma \ [f]\psi \ \ \equiv \ \ [\![\phi]\!]_\sigma \subseteq ([\![f]\!]_{\sigma,\tau})^{-1}([\![\psi]\!]_\tau).$$

A *Hoare triple* in $\mathcal{DL}(\sigma)$ is a formula $\phi \leq [f]\psi$ such that ϕ and ψ do not contain any program modalities.

Theorem 4 (Completeness For Hoare Triples) *Let $\phi \leq [f]\psi$ be a Hoare triple. Then*

$$\vdash \phi \leq [f]\psi \iff \models \phi \leq [f]\psi.$$

The restricted form of this Theorem is necessary, since we have

Theorem 5 *The validity problem for $\mathcal{DL}$ is Π^0_2-complete.*

5 Applications: The Logic Of A Domain Equation

A denotational analysis of a computational situation results in the description of a domain which provides an appropriate semantic universe for this situation. Canonically, domains are specified by type expressions in a metalanguage. We can then use our approach to "turn the handle", and generate a logic for this situation in a quite mechanical way. Two substantive case studies of this kind have been carried out, in the areas of concurrency [1] and the λ-calculus [2].

For example, in [1] we define a domain equation for synchronisation trees, and generate a logic which can be applied to the whole class of labelled transition systems. This logic subsumes Hennessy-Milner logic [8], and can be taken as a rational reconstruction of it. Furthermore, we *automatically* get a compositional proof theory for this logic, along the lines indicated above. Since one can define a denotational semantics for, e.g., SCCS [15] in our denotational metalanguage, we get a compositional proof system along the lines of those developed by Stirling and Winskel [24, 27]. Moreover, this proof system is guaranteed to be in harmony with our semantics.

References

[1] S. Abramsky, *A Domain Equation For Bisimulation*, unpublished manuscript, 1986.

[2] S. Abramsky, *The Lazy λ-Calculus*, unpublished manuscript, 1986.

[3] J. de Bakker and J. Zucker, Processes and the Denotational Semantics of Concurrency, *Information and Control* **54** (1982) 70-120.

[4] G. Berry, P.-L. Curien and J.-J. Levy, Full Abstraction For Sequential Languages: The State Of The Art, in: M. Nivat and J. Reynolds, eds., *Algebraic Semantics* (Cambridge University Press, 1985).

[5] H. B. Curry and R. Feys, *Combinatory Logic, Vol. 1* (North Holland, Amsterdam, 1958).

[6] E. W. Dijkstra, *A Discipline Of Programming* (Prentice Hall, New York, 1976).

[7] D. Harel, *First Order Dynamic Logic*, Lecture Notes in Computer Science **68** (Springer, Berlin, 1979).

[8] M. Hennessy and R. Milner, On Observing Non-determinism and Concurrency, in: J. de Bakker and J. van Leeuwen, eds., *Automata, Languages and Programming. Proceedings 1980*, Lecture Notes in Computer Science **85** (Springer, Berlin, 1980), 299-309.

[9] M. Hennessy and G. Plotkin, Full Abstraction for a simple parallel programming language, in: J. Becvar, ed., *Mathematical Foundations of Computer Science. Proceedings 1979*, Lecture Notes in Computer Science **74** (Springer, Berlin, 1979).

[10] C. A. R. Hoare, An Axiomatic Basis For Computer Programming, *Comm. of the ACM* **12** (1969) 576-580.

[11] P. Johnstone, *Stone Spaces* (Cambridge University Press, 1982).

[12] G. Kreisel, Interpretation of analysis by means of functionals of finite type, in: *Constructivity in Mathematics* (North Holland, Amsterdam, 1959).

[13] J. Lambek and P. Scott, *Introduction to Higher Order Categorical Logic* (Cambridge University Press, 1986).

[14] G. Winskel and K. G. Larsen, Using Information Systems To Solve Recursive Domain Equations Effectively, in: G. Kahn, D. B. MacQueen and G. Plotkin, eds., *Semantics of Data Types*, Lecture Notes in Computer Science **173** (Springer, Berlin, 1984).

[15] R. Milner, Calculi for Synchrony and Asynchrony, *Theoretical Computer Science* (1983) 267-310.

[16] M. Nivat, Infinite words, infinite trees, infinite computations, in: J. de Bakker, and J. van Leeuwen, eds., *Foundations of Computer Science III.2*, Mathematical Centre Tracts **109** (1979) 3-52.

[17] G. Plotkin, *Lecture Notes In Advanced Domain Theory*, University of Edinburgh, 1981.

[18] A. Pnueli, The temporal logic of programs, in: *Proceedings of the 19th Annual Symposium on Foundations of Computer Science* (IEEE, New York, 1977) 46-57.

[19] A. Poigne, On Specifications, Theories and Models with Higher Types, *Information and Control* (1986).

[20] D. Scott, *Lectures on a Mathematical Theory of Computation*, Programming Reseach Group Monograph (Oxford, 1981).

[21] D. Scott, Domains for Denotational Semantics, in: M. Nielsen and E. M. Schmidt, eds., *Automata, Languages and Programming. Proceedings 1982*, Lecture Notes in Computer Science **140** (Springer, Berlin, 1982).

[22] M. Smyth, Powerdomains and Predicate Transformers: A Topological View, in: J. Diaz, ed., *Automata, Languages and Programming. Proceedings 1983*, Lecture Notes in Computer Science **154** (Springer, Berlin, 1983) 662-675.

[23] M. Smyth and G. Plotkin, The Category-Theoretic Solution Of Recursive Domain Equations, *SIAM J. On Computing* **11, 4** (1982).

[24] C. Stirling, Modal Logics for Communicating Processes, Edinburgh University Computer Science Department Report, 1985.

[25] M. H. Stone, The Theory Of Representations For Boolean Algebras, *Trans. Amer. Math. Soc.* **40** (1936) 37-111.

[26] G. Winskel, *Events In Computation*, Ph.D. Thesis, University of Edinburgh 1980.

[27] G. Winskel, A Complete Proof System For SCCS with Modal Assertions, Cambridge University Computer Laboratory Report, 1985.