Non-Determinism in a Functional Setting

(Extended Abstract)*

C.-H. Luke Ong

Computer Laboratory, University of Cambridge,

Pembroke Street, CB2 3QG England. Email: `Luke.Ong@cl.cam.ac.uk`

and

DISCS, National University of Singapore.

Abstract

The pure untyped Lambda Calculus augmented with an (erratic) choice operator is considered as an idealised non-deterministic functional language. Both the "may" and the "must" modalities of convergence are of interest to us. Following Abramsky's work on domain theory in logical form, we identify the denotational type that captures our computational situation: $\delta = \mathbf{P}[[\delta \to \delta]_\perp]$ where $\mathbf{P}[-]$ is the Plotkin powerdomain functor. We then carry out a systematic programme which hinges on three distinct interpretations of δ, namely, process-theoretic, denotational and logical. The main theme of our programme is the complementarity of the various interpretations of δ. This work may be seen as a step towards a reapprochement between the algebraic theory of processes in Concurrency on the one hand, and the Lazy Lambda Calculus as a foundation for functional programming on the other.

1 Introduction

Our overall objective in this work is to develop a theory of non-determinism based on functions. The **operational scenario** we have in mind is the computation of pure functional programs in a non-deterministic (and possibly concurrent) setting. We combine the computational perspectives of the function paradigm with those of the process paradigm as follows:

(1) A program is a non-deterministic applicative process.

(2) The only thing observable about *a* computation of a program is the value it may yield. A value is a function from processes to processes.

(3) To compute a program is to ascertain the (or a possible) value of the process *i.e.* evaluation. Evaluation is implemented by a notion of reduction; so "computation = evaluation = reduction".

(4) An evaluation of a process may *either* diverge *or* it may converge non-deterministically to a value.

(5) Any two computations (or "runs") of the same process may or may not exhibit the same behaviour.

The canonical language which satisfies the above criteria is the **Non-Deterministic Lambda Calculus**, written $\mathbf{\Lambda}_{\mathrm{n}}$. Terms of $\mathbf{\Lambda}_{\mathrm{n}}$ are defined by the following grammar:

$$M \quad ::= \quad x \quad | \quad \lambda x.M \quad | \quad \lambda^{\mathrm{v}} x.M \quad | \quad M \cdot M \quad | \quad M + M.$$

As usual, x ranges over a denumerable collection of variables. There are two kinds of abstractions in $\mathbf{\Lambda}_{\mathrm{n}}$: call-by-name $\lambda x.M$ and call-by-value $\lambda^{\mathrm{v}} x.M$. As is usual, we often write $M \cdot N$ simply as MN. The "erratic" choice operator $+$ introduces non-determinism to the language. The collection of values (= abstractions) is denoted VAL and is ranged over by meta-variables J, K. The collection of programs (= closed

*To appear in Proc. 8th Annual IEEE Symposium on Logic in Computer Science, Montreal, June 93.

terms) is denoted $\mathbf{\Lambda}_{\mathrm{n}}^{o}$. We present the operational semantics of $\mathbf{\Lambda}_{\mathrm{n}}$ in Plotkin-style transition semantics: $M > N$ is read as "M *may* reduce in one step to N". The binary relation $>$ over $\mathbf{\Lambda}_{\mathrm{n}}^{o}$ is inductively defined by the following rules:

$$(\lambda x.P)M > P[M/x] \qquad M + N > M$$

$$(\lambda^{\mathrm{v}} x.P)K > P[K/x] \qquad M + N > N$$

$$\frac{M > N}{MP > NP} \qquad \frac{M > N}{(\lambda^{\mathrm{v}} x.P)M > (\lambda^{\mathrm{v}} x.P)N}.$$

Following standard practice in functional programming, we only consider *weak* reduction *i.e.* no reduction "under a lambda" is permitted. Note that call-by-name abstraction implements *run-time choice* whereas call-by-value abstraction implements a weak version of *call-time choice* [HA80]. We write $\gg$ as the reflexive and transitive closure of $>$ and $M \not> \stackrel{\mathrm{def}}{=} \neg[\exists N.M > N]$. It is easy to see that for any program M, $M \not>$ iff $\mathrm{VAL}(M)$. We define a *may-converge* relation $\Downarrow^{\mathrm{may}} \subseteq \mathbf{\Lambda}_{\mathrm{n}}^{o} \times \mathrm{VAL}$ and a *may-diverge* predicate $\Uparrow^{\mathrm{may}} \subseteq \mathbf{\Lambda}_{\mathrm{n}}^{o}$ as follows:

$$\begin{aligned} M \Downarrow^{\mathrm{may}} K &\stackrel{\mathrm{def}}{=} M \gg K; \\ M \Uparrow^{\mathrm{may}} &\stackrel{\mathrm{def}}{=} \exists\{\, M_i : i \in \omega \,\}.M \equiv M_0 \quad \& \\ &\qquad \forall i \in \omega.M_i > M_{i+1}. \end{aligned}$$

Starting from a fixed program M say, the set

$$\{\, N : M \gg N \,\}$$

of programs "reachable" from M by the one-step may-reduce relation clearly forms a tree rooted at M and whose terminal nodes are values. In this tree, $M \Downarrow^{\mathrm{may}} K$ holds iff there is a finite path from M which terminates at K; and $M \Uparrow^{\mathrm{may}}$ iff there is an infinite path starting from M. So $\Downarrow^{\mathrm{may}}$ and $\Uparrow^{\mathrm{may}}$ are two kinds of "observation quanta" which together provide an extensional view of the behaviour of the program M.

Related work In [San92], Sangiorgi studies equivalence between terms in the Lazy Lambda Calculus [AO92] relative to Milner's encoding into the π-calculus. de'Liguoro and Piperno [dP91] propose a version of non-deterministic lambda calculus which permits reduction "under a lambda". They focus on a Morris-style contextual preorder (relative only to the must-modality of convergence) and establish a denotational characterization using the Smyth powerdomain. See also [JP90] which considers both the may and the must modalities of convergence.

Outline In this work, we follow a systematic pattern of enquiry. Our *operational* intuition informally described in statements (1) to (5) has already been given formal expression in the shape of a canonical programming language $\mathbf{\Lambda}_{\mathrm{n}}$. We then pin down the computational situation by the denotational type $\delta \equiv \mathsf{rec}\, t.\mathbf{P}[[t \to t]_{\perp}]$ where $\mathsf{rec}\, t.\text{-}$ is the standard type recursion operator. The rest of the programme hinges on three different interpretations of δ:

(1) *Process-theoretic:* Following ideas from Process Algebra, we introduce *non-deterministic applicative transition systems* (NATS) as an abstract framework to unify syntax (= languages), semantics (= models) and axiomatics (= proof systems). We equip each NATS with an intrinsic notion of *applicative bisimulation* $\sqsubseteq^{\mathrm{B}}$ and prove that it is a precongruence for a large class of programmable NATSs which includes $\mathbf{\Lambda}_{\mathrm{n}}$ (see [Ong92a] for a proper treatment). A Morris-style contextual characterization (w.r.t. the may- and the must-modalities of convergence) is not possible for applicative bisimulation. Instead, we develop an intensional notion of *testing* and show that applicative bisimulation is characterized by the induced notion of testing preorder. [The new process-theoretic interpretation of denotational types in general will be given proper treatment elsewhere.]

(2) *Denotational*: We read δ standardly as a recursive domain equation and regard the initial solution $\mathbb{D}$ satisfying $\mathbb{D} = \mathbf{P}[[\mathbb{D} \to \mathbb{D}]_{\perp}]$ in the category of SFP domains as our canonical model. $\mathbb{D}$ is an internally fully abstract NATS (*i.e.* $\sqsubseteq^{\mathrm{B}}$ coincides with the intrinsic partial order $\sqsubseteq_{\mathbb{D}}$) and it is a model of $\mathbf{\Lambda}_{\mathrm{n}}$. The interpretation is weakly adequate but not fully abstract for $\mathbf{\Lambda}_{\mathrm{n}}$.

(3) *Logical*: We "turn the handle" of Abramsky's domain logic programme to assemble an elaborate logical system $\mathcal{L}(\delta)$ tailor-made for NATSs according to the prescription laid down by δ. The derived logic satisfaction preorder coincides precisely with testing preorder for a class of *approximable* (a weak continuity property) NATSs *e.g.* $\mathbb{D}$. We identify a class of *finitary* NATS (*e.g.* $\mathbb{D}$) in which finitary formulae have the same discriminatory power as infinitary formulae. An endogenous program logic is then introduced which is sound and complete for $\mathbb{D}$.

We conclude this work with a (finitary) full abstraction conjecture. More generally, this work illustrates a

generic approach to programming semantics (already adumbrated in [AO92, Abr91a]) which is based on a synthesis of the three dominant styles of interpretations: in principle at least, it is equally applicable to a wide range of computational situations describable by any of the following denotational types: ($\oplus$ is the standard coalesced sum)

$$\begin{array}{rcl} \sigma & ::= & \mathbf{1} \mid \sigma \times \sigma \mid \sigma \oplus \sigma \mid \sigma \to \sigma \mid \\ & & \mathbf{P}[\sigma] \mid (\sigma)_\perp \mid t \mid \mathsf{rec}\, t.\sigma. \end{array}$$

2 A Process-Theoretic Reading

We code up the five informal statements of operational criteria into a number of structural constraints which define a class of abstract objects called **non-deterministic applicative transition systems** (or NATS for short). NATS will be used in this work as an important bridge which inter-connects syntax (= language), semantics (= models) and axiomatics (= program logics) of our computational situation. Formally, a NATS is a structure $\langle A, \Downarrow^{\mathrm{may}}, \Uparrow^{\mathrm{may}} \rangle$ where A is a set of non-deterministic applicative processes,

- $\Uparrow^{\mathrm{may}}$, called the *may-diverge predicate*, is a proper subset of A,
- $\Downarrow^{\mathrm{may}} \subseteq A \times A^A$, the *may-converge relation*;

satisfying the following axiom: for any $a \in A$,

$$\text{(NATS)} \qquad \neg[\exists f.a \Downarrow^{\mathrm{may}} f] \implies a \Uparrow^{\mathrm{may}}.$$

Notation It is convenient to introduce the following *derived* predicates:

$$\begin{array}{rcl} a\Downarrow^{\mathrm{may}} & \stackrel{\mathrm{def}}{=} & \exists f.a \Downarrow^{\mathrm{may}} f; \\ a\Downarrow^{\mathrm{must}} & \stackrel{\mathrm{def}}{=} & \neg[a\Uparrow^{\mathrm{may}}]; \\ a\Uparrow^{\mathrm{must}} & \stackrel{\mathrm{def}}{=} & \neg[a\Downarrow^{\mathrm{may}}]. \end{array}$$

Clearly, a NATS A may equivalently be specified by the pair $\Downarrow^{\mathrm{may}}$ and $\Downarrow^{\mathrm{must}}$. Note that the axiom (NATS) is equivalent to: for any $a \in A$, $a\Downarrow^{\mathrm{must}} \implies a\Downarrow^{\mathrm{may}}$.

Capability Function Given a NATS A, we define an associated *capability function* from A to $\mathcal{P}_{\mathrm{ne}}(A^A \cup \{-\})$ ($\mathcal{P}_{\mathrm{ne}}(-)$ is the collection of non-empty subsets) where $-$ is a distinguished element not in A^A as follows:

$$\mathrm{cap}(a) \stackrel{\mathrm{def}}{=} \{ - : a\Uparrow^{\mathrm{may}} \} \cup \{ f \in A^A : a \Downarrow^{\mathrm{may}} f \}.$$

Because of the axiom (NATS), the function $\mathrm{cap}(-)$ is well-defined; $\mathrm{cap}(a)$ captures the entire range of operational behaviour which the process a can non-deterministically exhibit. We denote the image of the function $\mathrm{cap}(-)$ in $\mathcal{P}_{\mathrm{ne}}(A^A \cup \{-\})$ as $\mathrm{cap}(A)$. A NATS A is said to have a *functional retract* if for any a and for any f such that $a\Downarrow^{\mathrm{may}} f$, there is an $\overline{f} \in A$ such that $\mathrm{cap}(\overline{f}) = \{ f \}$. An element a of a NATS is said to have *finite* (resp. *countable*) *capability property* if $\mathrm{cap}(a)$ is a finite (resp. countable) set. A NATS has *finite capability property* if every element has finite capability property; similarly for countable capability property.

Example 2.1 The Non-Deterministic Lambda Calculus $\mathbf{\Lambda}_{\mathrm{n}}$ is a NATS with a functional retract. In $\mathbf{\Lambda}_{\mathrm{n}}$, there is a natural interpretation $(-)^*$ of values as functions from $\mathbf{\Lambda}_{\mathrm{n}}^o$ to $\mathbf{\Lambda}_{\mathrm{n}}^o$: for any value K, just define K^* as the map $N \mapsto KN$. By confusing K with K^*, we see that $\Downarrow^{\mathrm{may}} \subseteq \mathbf{\Lambda}_{\mathrm{n}}^o \times (\mathbf{\Lambda}_{\mathrm{n}}^o)^{(\mathbf{\Lambda}_{\mathrm{n}}{}^o)}$. $\mathbf{\Lambda}_{\mathrm{n}}$ does not have the finite capability property: just consider $\mathbf{\Upsilon} \equiv \mathbf{Y}F$ where

$$\begin{array}{rcl} F & \equiv & \lambda x_1 x_2.\mathbf{\Omega} + \lambda^{\mathrm{v}} y.\lambda x.y, \\ \mathbf{\Omega} & \equiv & (\lambda x.xx)(\lambda x.xx), \\ \mathbf{Y} & \equiv & \lambda f.(\lambda x.f(xx))(\lambda x.f(xx)). \end{array}$$

We can show that

$$\mathrm{cap}(\mathbf{\Upsilon}) = \{ \lambda x_1 \cdots x_n.\mathbf{\Omega} : n \geqslant 1 \} \cup \{ - \}.$$

It is easy to see that $\mathbf{\Lambda}_{\mathrm{n}}^o$ has countable capability property.

Applicative Bisimulation How should the behaviour of one process be compared with that of another? Following ideas of Park and Milner, we introduce a notion of *applicative bisimulation* for NATSs which takes into account both the *may* and the *must* modalities of convergence. Given an arbitrary NATS A, we define its applicative bisimulation (a binary relation over A) as the maximal fixpoint of a monotone function in Figure 1.

The set $\mathcal{P}(A \times A)$ of binary relations over A is a complete lattice under set inclusion. Since F is monotone, by Tarski's Theorem, it has a maximal fixpoint, given by $\bigcup\{ R \subseteq A \times A : R \subseteq F(R) \}$ which we define to be $\precsim^{\mathrm{B}}$. Note that $\precsim^{\mathrm{B}}$ is a preorder; the associated equivalence is written $\sim^{\mathrm{B}}$. As is standard, we can also describe $\precsim^{\mathrm{B}}$ more explicitly in terms of iterations of F:

Consider the function $F : \mathcal{P}(A \times A) \to \mathcal{P}(A \times A)$ defined as follows: for any $R \subseteq A \times A$

$$F(R) \stackrel{\text{def}}{=} \left\{ \begin{array}{lll} \langle a, b \rangle : & \forall f.a \Downarrow^{\text{may}} f & \Longrightarrow \quad [\exists g.b \Downarrow^{\text{may}} g \ \& \ \forall c \in A.\langle f(c), g(c) \rangle \in R] \quad \& \\ & a \Downarrow^{\text{must}} & \Longrightarrow \left\{ \begin{array}{lll} (i) & b \Downarrow^{\text{must}} & \& \\ (ii) & \forall g.b \Downarrow^{\text{may}} g & \Longrightarrow \quad \exists f.a \Downarrow^{\text{may}} f \quad \& \\ & & \qquad \forall c \in A.\langle f(c), g(c) \rangle \in R \end{array} \right. \end{array} \right\}$$

Figure 1.

---

we define $\sqsubseteq^{\text{B}}_{\alpha}$ with $\alpha \in$ ordinals by induction:

$$\begin{array}{lll} \sqsubseteq^{\text{B}}_{0} & \stackrel{\text{def}}{=} & A \times A; \\ \sqsubseteq^{\text{B}}_{\alpha+1} & \stackrel{\text{def}}{=} & F(\sqsubseteq^{\text{B}}_{\alpha}) \quad \text{if } \alpha \text{ is a successor ordinal;} \\ \sqsubseteq^{\text{B}}_{\lambda} & \stackrel{\text{def}}{=} & \cap_{\alpha<\lambda} \sqsubseteq^{\text{B}}_{\alpha} \quad \text{if } \lambda \text{ is a limit ordinal.} \end{array}$$

This sequence of relations is decreasing and bounded below by $\sqsubseteq^{\text{B}}$. For any (small) NATS, the sequence is eventually stationary *i.e.* for some λ, for all $\alpha > \lambda$, $\sqsubseteq^{\text{B}}_{\alpha} = \sqsubseteq^{\text{B}}_{\lambda}$. The least such λ is called the closure ordinal; and we have $\sqsubseteq^{\text{B}} = \sqsubseteq^{\text{B}}_{\lambda}$.

Since $\text{cap}(a)$ is in some sense a "complete" description of the (extensional) behaviour of the process a, we expect to be able to express applicative bisimulation equivalently as an appropriate relation on capabilities. To this end, we define an extensional preorder $\sqsubseteq^{\text{ext}}$ on $\text{cap}(A)$:

$$\begin{array}{lll} f \sqsubseteq^{\text{ext}} g & \stackrel{\text{def}}{=} & f = - \quad \text{or} \\ & & [g \neq - \ \& \ \forall c \in A.f(c) \sqsubseteq^{\text{B}} g(c)]. \end{array}$$

Consider the derived *Egli-Milner* preorder $\sqsubseteq_{\text{EM}}$ defined over the preordered set $\langle \text{cap}(A), \sqsubseteq^{\text{ext}} \rangle$. We have the following characterization which provides a convenient way to establish applicative bisimulation between processes:

Proposition 2.2 (Capability) *For any $a, b \in A$ a* NATS, $a \sqsubseteq^{\text{B}} b \iff \text{cap}(a) \sqsubseteq_{\text{EM}} \text{cap}(b)$. □

Example Since $\mathbf{\Lambda}_{\text{n}}$ has a functional retract, the associated applicative bisimulation is equivalent to the specification given in Figure 2. It is easy to see that for any $M \in \mathbf{\Lambda}^{o}_{\text{n}}$, $\mathbf{\Omega} \sqsubseteq^{\text{B}} M$ and $M \sqsubseteq^{\text{B}} \mathbf{YK}$.

We present (selectively) two important properties of applicative bisimulation in the NATS $\mathbf{\Lambda}^{o}_{\text{n}}$:

Theorem 2.3 *(i)* **(Precongruence)** *For $M, N \in \mathbf{\Lambda}^{o}_{\text{n}}$ and for any closed context $C[\,]$, whenever $M \sqsubseteq^{\text{B}} N$ then $C[M] \sqsubseteq^{\text{B}} C[N]$. This is an instance of a general result in [Ong92a].*

(ii) **(Laws of $\mathbf{\Lambda}_{\text{n}}$)** *Interpreting $=$ as $\sim^{\text{B}}$, the following laws are valid in $\mathbf{\Lambda}^{o}_{\text{n}}$: $+$ is idempotent, commutative and associative;*

$$(\lambda x.P)N = P[N/x] \quad (\lambda^{\text{v}} x.P)K = P[K/x]$$
$$(M + N)L = ML + NL$$
$$(\lambda^{\text{v}} x.P)(M + N) = (\lambda^{\text{v}} x.P)M + (\lambda^{\text{v}} x.P)N.$$

□

A Notion of Testing

Can applicative bisimulation in $\mathbf{\Lambda}_{\text{n}}$ be characterised by a Morris-style contextual preorder ([Bar84, p. 428])? Consider the following operational preorder:

$$\begin{array}{lll} M \sqsubseteq^{\text{cxt}} N & \stackrel{\text{def}}{=} & \forall C[\,].[C[M]\Downarrow^{\text{may}} \Longrightarrow C[N]\Downarrow^{\text{may}} \ \& \\ & & C[M]\Downarrow^{\text{must}} \Longrightarrow C[N]\Downarrow^{\text{must}}]. \end{array}$$

If we read the predicates $\Downarrow^{\text{may}}$ and $\Downarrow^{\text{must}}$ as the computational "observables" then we can think of contexts $C[\,]$ as tests which the Experimenter may perform with a view to comparing the operational behaviour of programs M and N.

In fact, $\sqsubseteq^{\text{B}}$ affords a finer distinction between terms than does $\sqsubseteq^{\text{cxt}}$ *i.e.* $\sqsubseteq^{\text{B}} \subset \sqsubseteq^{\text{cxt}}$. The inclusion is an easy consequence of the precongruence property of $\sqsubseteq^{\text{B}}$. To see the strictness of the inclusion, consider:

Example 2.4 Let

$$\begin{array}{lll} \Theta_1 & \equiv & \lambda x.\mathbf{\Omega} + \lambda xy.\mathbf{\Omega}; \\ \Theta_2 & \equiv & \lambda x.(\mathbf{\Omega} + \lambda y.\mathbf{\Omega}). \end{array}$$

We can show that $\Theta_1 \sim^{\text{cxt}} \Theta_2$ (the proof is non-trivial) but $\Theta_1 \not\sqsubseteq^{\text{B}} \Theta_2$ and $\Theta_2 \not\sqsubseteq^{\text{B}} \Theta_1$. It is instructive to see why $\Theta_1 \not\sqsubseteq^{\text{B}} \Theta_2$: whereas Θ_1 satisfies the property **(Q)** defined as

For any $M, M' \in \mathbf{\Lambda}^o_{\mathrm{n}}$, $M \sqsubseteq^{\mathrm{B}} M' \iff$

$$\left\{ \begin{array}{lll} \forall K.M \Downarrow^{\mathrm{may}} K & \Longrightarrow & \exists K'.M' \Downarrow^{\mathrm{may}} K' \ \& \ \forall L.KL \sqsubseteq^{\mathrm{B}} K'L \quad \& \\ M \Downarrow^{\mathrm{must}} & \Longrightarrow & M' \Downarrow^{\mathrm{must}} \ \& \ [\forall K'.M' \Downarrow^{\mathrm{may}} K' \Longrightarrow \exists K.M \Downarrow^{\mathrm{may}} K \ \& \ \forall L.KL \sqsubseteq^{\mathrm{B}} K'L]. \end{array} \right\}$$

Figure 2.

"it may converge to a value which when applied to any process *must* converge";

Θ_2 does not.

The moral then is this: to characterise applicative bisimulation, we need a more intensional notion of testing capable of describing *contingent* behaviour — it should allow the Experimenter to perform tests in the intermediate stages of all possible computations. In particular, the notion of testing must be elaborate enough to articulate such properties as **(Q)** above.

A Language of Testing We introduce a language of *tests* which has two sorts: π, the sort of *processes*, and κ, the sort of *capabilities*. For the rest of this work, we let A range over NATSs and σ over the sorts $\{\pi, \kappa\}$. We write $\mathrm{TESTS}_{\infty\pi}$ and $\mathrm{TESTS}_{\infty\kappa}$ for the collection of tests (ranged over by τ) of sort π and κ respectively. They are defined by the following rule induction: (I may be infinite)

$$\frac{\{\tau_i \in \mathrm{TESTS}_{\infty\sigma}\}_{i\in I}}{\bigvee_{i\in I}\tau_i, \bigwedge_{i\in I}\tau_i \in \mathrm{TESTS}_{\infty\sigma}}$$

$$\frac{a \in A \quad \tau \in \mathrm{TESTS}_{\infty\pi}}{a\tau \in \mathrm{TESTS}_{\infty\kappa}} \qquad \frac{\tau \in \mathrm{TESTS}_{\infty\kappa}}{\forall\tau, \exists\tau \in \mathrm{TESTS}_{\infty\pi}}.$$

The subscript ∞ indicates the (possible) presence of infinite conjunctions and disjunctions. We write $\mathrm{TESTS}_{\omega\pi}$ and $\mathrm{TESTS}_{\omega\kappa}$ for the respective sub-languages of $\mathrm{TESTS}_{\infty\pi}$ and $\mathrm{TESTS}_{\infty\kappa}$ obtained by restricting I to range over finite sets.

Semantics of Testing We make precise the assertion that "a process a of a NATS A passes a σ-test τ", written $\mathrm{obs}_\sigma(a, \tau)$. The predicates

$$\begin{array}{lll} \mathrm{obs}_\pi & \subseteq & A \times \mathrm{TESTS}_{\infty\pi}, \\ \mathrm{obs}_\kappa & \subseteq & A^A \times \mathrm{TESTS}_{\infty\kappa}; \end{array}$$

are defined by mutual induction as follows: for $s \in A \cup A^A$, $a \in A$ and $f \in A^A$,

$$\begin{array}{rcl} \mathrm{obs}_\sigma(s, \bigwedge_{i\in I}\tau_i) & \stackrel{\mathrm{def}}{=} & \forall i \in I.\mathrm{obs}_\sigma(s, \tau_i); \\ \mathrm{obs}_\sigma(s, \bigvee_{i\in I}\tau_i) & \stackrel{\mathrm{def}}{=} & \exists i \in I.\mathrm{obs}_\sigma(s, \tau_i); \\ \mathrm{obs}_\kappa(f, a\tau) & \stackrel{\mathrm{def}}{=} & \mathrm{obs}_\pi(f(a), \tau); \\ \mathrm{obs}_\pi(a, \forall\tau) & \stackrel{\mathrm{def}}{=} & a\Downarrow^{\mathrm{must}} \quad \& \\ & & \forall f.a \Downarrow^{\mathrm{may}} f.\mathrm{obs}_\kappa(f, \tau); \\ \mathrm{obs}_\pi(a, \exists\tau) & \stackrel{\mathrm{def}}{=} & \exists f.a \Downarrow^{\mathrm{may}} f \ \& \ \mathrm{obs}_\kappa(f, \tau). \end{array}$$

Note that $\Downarrow^{\mathrm{must}}$ is just the test $\forall\mathsf{t}$, and $\Downarrow^{\mathrm{may}}$ the test $\exists\mathsf{t}$ where $\mathsf{t} \equiv \bigwedge_{i\in\varnothing}\tau_i$ and $\mathsf{f} \equiv \bigvee_{i\in\varnothing}\tau_i$.

Remark Our test expressions are reminiscent of the denotational formulation of tests in [Abr87]. We may classify them according to the criteria of effectiveness and (finite) observability. The first group which consists of κ-tests of the shape $a\tau$, corresponds to the familiar notion of *traces*.

- *Copying* $\bigwedge_{i\in I}\tau_i, \bigvee_{i\in I}\tau_i$: To find out whether a process a satisfies the test $\bigwedge_{i\in I}\tau_i$ say, we need I copies of the process a so that for each $i \in I$, the test τ_i may be performed on a copy of a. When all subtests have been completed, we need a way of combining the outcome of each subtest in order to obtain a result for the overall test. This process of "copying" would definitely be objectionable from the point of view of effectiveness.

- *Global Testing* $\forall\tau, \exists\tau$: For a process a to satisfy a global test $\forall\tau$ say, apart from the requirement that a must converge, we need to be able to enumerate all possible evaluations of a, check that for each possible value f, the subtest τ is satisfied. Such tests go beyond the bounds of finite observability.

We define *modal depth* $\mathsf{md}(\tau)$ of a test τ:

$$\begin{array}{lcl} \mathsf{md}(\bigwedge_{i\in I}\tau_i) & = & \mathsf{md}(\bigvee_{i\in I}\tau_i) \\ & \stackrel{\text{def}}{=} & \sup\{\,\mathsf{md}(\tau_i) : i\in I\,\}; \\ \mathsf{md}(a\tau) & \stackrel{\text{def}}{=} & \mathsf{md}(\tau); \\ \mathsf{md}(\forall\tau) & = & \mathsf{md}(\exists\tau) \\ & \stackrel{\text{def}}{=} & \mathsf{md}(\tau)+1. \end{array}$$

Define further,

$$\textsc{tests}_{\infty\pi}(a) \quad \stackrel{\text{def}}{=} \quad \{\,\phi\in\textsc{tests}_{\infty\pi} : \mathrm{obs}_\pi(a,\phi)\,\};$$

similarly $\textsc{tests}_{\omega\pi}(a)$ *etc.* Given a limit ordinal λ, we define

$$\textsc{tests}^{\lambda}_{\infty\sigma} \quad \stackrel{\text{def}}{=} \quad \{\,\tau\in\textsc{tests}_{\infty\sigma} : \mathsf{md}(\tau)<\lambda\,\};$$

if λ is a successor ordinal, then we replace $<$ with $\leqslant$ in the above definition. Define $\textsc{tests}^{\lambda}_{\infty\pi}(a) \stackrel{\text{def}}{=} \textsc{tests}_{\infty\pi}(a)\cap\textsc{tests}^{\lambda}_{\infty\pi}$.

Example (Refer to Example 2.4.) For any $X\in\mathbf{\Lambda}^o_{\mathrm{n}}$, write $\mu\equiv\forall X\Downarrow^{\mathrm{may}}$ and $\nu\equiv\exists X\Downarrow^{\mathrm{must}}$ (which encodes property **(Q)**) We have

$$\begin{array}{lll} \mathrm{obs}_\pi(\Theta_1,\nu) & \text{but} & \neg\mathrm{obs}_\pi(\Theta_2,\nu); \\ \mathrm{obs}_\pi(\Theta_2,\mu) & \text{but} & \neg\mathrm{obs}_\pi(\Theta_1,\mu). \end{array}$$

For any $\phi\in\textsc{tests}_{\infty\pi}$ with $\mathsf{md}(\phi)\geqslant 1$, if $\mathrm{obs}_\pi(M,\phi)$ for some M, then $\mathrm{obs}_\pi(\mathbf{YK},\phi)$ and $\neg[\mathrm{obs}_\pi(\mathbf{\Omega},\phi)]$. It is easy to see that $\mathbf{YK}$ and $\mathbf{\Omega}$ are the largest and least term w.r.t. $\precsim^{\mathrm{B}}$ respectively.

This notion of testing does characterise applicative bisimulation (in a style reminiscent of the characterization of bisimulation by Hennessy-Milner logic). For $a,b\in A$, we define

$$\begin{array}{lcl} a\precsim^{\mathrm{T}}_{\infty,\lambda} b & \stackrel{\text{def}}{=} & \textsc{tests}^{\lambda}_{\infty\pi}(a)\subseteq\textsc{tests}^{\lambda}_{\infty\pi}(b), \\ a\precsim^{\mathrm{T}}_{\infty} b & \stackrel{\text{def}}{=} & \textsc{tests}_{\infty\pi}(a)\subseteq\textsc{tests}_{\infty\pi}(b); \end{array}$$

similarly for $a\precsim^{\mathrm{T}}_{\omega} b$.

Theorem 2.5 (Testing Characterization) *For any $a,b\in A$ a* NATS *and for any ordinal λ, we have $a\precsim^{\mathrm{B}}_{\lambda} b \iff a\precsim^{\mathrm{T}}_{\infty,\lambda} b$. Hence, $a\precsim^{\mathrm{B}} b \iff a\precsim^{\mathrm{T}}_{\infty} b$. In particular, if A has finite capability property, then a stronger result holds: $a\precsim^{\mathrm{B}} b \iff a\precsim^{\mathrm{T}}_{\omega} b$.* □

3 Denotational Intepretation

The canonical domain is the initial solution (call it $\mathbb{D}$) to the domain equation: $D=\mathbf{P}[[D\to D]_\perp]$ in the category $\mathbb{SFP}$ of SFP domains and continuous functions. Plotkin powerdomain is the right choice since we care for both the may- and the must-type properties in non-determinism (Hoare powerdomain only addresses the may-type modality while Smyth powerdomain addresses only the must-). Our reference for Plotkin powerdomain is [Plo76] with which we assume the reader is familiar. We name the isomorphism pair as follows:

$$\mathbb{D} \quad\xrightarrow{\text{fun}}\quad \mathbf{P}[[\mathbb{D}\to\mathbb{D}]_\perp] \quad\xrightarrow{\text{gr}}\quad \mathbb{D}.$$

Define a may-diverge predicate $\Uparrow^{\mathrm{may}}\subseteq\mathbb{D}$ and a may-converge relation $\Downarrow^{\mathrm{may}}\subseteq\mathbb{D}\times\mathbb{D}^{\mathbb{D}}$ as

$$\begin{array}{rcl} d\Uparrow^{\mathrm{may}} & \stackrel{\text{def}}{=} & -\in\mathrm{fun}(d), \\ d\Downarrow^{\mathrm{may}} f & \stackrel{\text{def}}{=} & \mathrm{up}_{[\mathbb{D}\to\mathbb{D}]}(f)\in\mathrm{fun}(d); \end{array}$$

where $\mathrm{up}_D : D\to D_\perp$ is the canonical embedding for any CPO D. Since

$$\neg[\exists f.d\Downarrow^{\mathrm{may}} f] \quad\text{iff}\quad \mathrm{fun}(d)=\{\,-\,\} \quad\text{iff}\quad d=-,$$

the axiom (NATS) is satisfied, and so $\langle\mathbb{D},\Uparrow^{\mathrm{may}},\Downarrow^{\mathrm{may}}\rangle$ is a NATS. There is an obvious 1-1 correspondence between $\mathrm{cap}(d)$ and $\mathrm{fun}(d)$, for any $d\in\mathbb{D}$. Since $\mathbb{D}$ is a NATS, it has an intrinsic applicative bisimulation preorder $\precsim^{\mathrm{B}}$. How does $\precsim^{\mathrm{B}}$ compare with the inherent partial order $\sqsubseteq_{\mathbb{D}}$ of the SFP domain $\mathbb{D}$? We can show:

Proposition 3.1 (Internal Full Abstraction) *For any $d,e\in\mathbb{D}$, $d\sqsubseteq_{\mathbb{D}} e \iff d\precsim^{\mathrm{B}} e$.* □

If it is right to regard $\mathbb{D}$ as the canonical model of our computational scenario (which we believe it is), then this theorem gives us confidence that both the notions of NATS as well as $\precsim^{\mathrm{B}}$ are apposite.

Plotkin Powerdomain Preliminaries (See [Plo76].) It is well-known that the Lawson topology on an algebraic CPO D is compact iff D is "2/3 SFP". For any subset $X\subseteq D$, we write the Lawson closure of X as $\mathrm{Cl}[X]$; $\mathrm{Con}(\mathrm{Cl}[X])$ is the least convex, Lawson closed subset of D containing X. We work with the following representation of $\mathbf{P}[D]$ for any "2/3 SFP" CPO D:

$$\mathbf{P}[D] \quad\stackrel{\text{def}}{=}\quad \langle\,\{\,X\subseteq D : X\neq\varnothing \;\&\; X=X^\circ\,\},\sqsubseteq_{\mathrm{EM}}\rangle$$

where $X^\circ\stackrel{\text{def}}{=}\mathrm{Con}(\mathrm{Cl}[X])$. For $X\subseteq D$, let

$$X^*\stackrel{\text{def}}{=}\uparrow X\cap\overline{X}. \qquad \text{(upper- and Scott-closed)}$$

Note that for a Scott compact subset $X \subseteq D$, $X^* = X^\circ$. We shall use the following standard (continuous) Plotkin powerdomain operation: for any $f : D_1 \times \cdots \times D_n \to D$ in $\mathbb{SFP}$, its *multilinear extension* $f^\dagger : \mathbf{P}[D_1] \times \cdots \times \mathbf{P}[D_n] \to \mathbf{P}[D]$ is defined as the function

$$\langle X_1, \cdots, X_n \rangle \quad \mapsto \quad \{ f(x_1, \cdots, x_n) : x_i \in X_i \}^\circ.$$

Succinctly, $\langle \mathbf{P}, \{\!|-|\!\}_{_}, \biguplus_{_} \rangle$ is a monad in the category $\mathbb{SFP}$. The monadic unit and multiplication are standardly defined as follows: for $X, Y \in \mathbf{P}[D]$, $\Theta \in \mathbf{P}[\mathbf{P}[D]]$,

$$\begin{aligned} \{\!|d|\!\}_D &\stackrel{\text{def}}{=} \{ d \}^\circ = \{ d \}; \\ X \uplus_D Y &\stackrel{\text{def}}{=} (X \cup Y)^\circ = \mathrm{Con}(X \cup Y); \\ \biguplus_D(\Theta) &\stackrel{\text{def}}{=} (\textstyle\bigcup \Theta)^\circ = \mathrm{Con}(\bigcup \Theta). \end{aligned}$$

$\mathbb{D}$: a Model of the Non-Deterministic Lambda Calculus First, $\mathbb{D}$ needs to be endowed with an applicative structure. We claim that $\langle \{\!|-|\!\}_{\mathbb{D}}, \Psi_{\mathbb{D}} \rangle$ defines a retract of $\mathbb{D}$ into $\mathbf{P}[\mathbb{D}]$ where

$$\Psi_{\mathbb{D}} \stackrel{\text{def}}{=} \mathrm{gr} \circ \biguplus_{[\mathbb{D} \to \mathbb{D}]_\perp} \circ \mathbf{P}[\mathrm{fun}].$$

Application is defined as: for $d, e \in \mathbb{D}$,

$$\begin{aligned} d \cdot e &\stackrel{\text{def}}{=} \Psi_{\mathbb{D}}(\mathrm{app}^\dagger \langle \mathrm{fun}(d), \{\!|e|\!\}_{\mathbb{D}} \rangle) \\ &= \Psi_{\mathbb{D}}((\{ f(e) : d \Downarrow^{\mathrm{may}} f \} \cup \{ - : d \Uparrow^{\mathrm{may}} \})^*); \end{aligned}$$

with $\mathrm{app} : [\mathbb{D} \to \mathbb{D}]_\perp \times \mathbb{D} \to \mathbb{D}$ defined as

$$\mathrm{app}(\langle k, e \rangle) \stackrel{\text{def}}{=} \begin{cases} f(e) & \text{if } k = \mathrm{up}_{[\mathbb{D} \to \mathbb{D}]}(f), \\ - & \text{else } \textit{i.e.}\ k = -. \end{cases}$$

To interpret the erratic choice operator, we define a binary operation $+$ in $\mathbb{D}$: for $d, e \in \mathbb{D}$,

$$d + e \stackrel{\text{def}}{=} \mathrm{gr}(\mathrm{fun}(d) \uplus_{[\mathbb{D} \to \mathbb{D}]_\perp} \mathrm{fun}(e)).$$

The operation $+$ is idempotent, commutative and associative. Also, application right-distributes over $+$: for $d, e, f \in \mathbb{D}$,

$$(d + e) \cdot f = (d \cdot f) + (e \cdot f).$$

We can now define $[\![-]\!]^{\mathbb{D}}_{_} : \mathbf{\Lambda}_{\mathrm{n}}(\mathbb{D}) \times \mathsf{Env} \to \mathbb{D}$, a standard compositional (and continuous) valuation of $\mathbf{\Lambda}_{\mathrm{n}}$ in $\mathbb{D}$ by structural induction as in Figure 3. Note that call-by-value abstractions denote linear functions (in the sense of [Plo76]). A structure $\langle A, \Downarrow^{\mathrm{may}}, \Uparrow^{\mathrm{may}}, [\![-]\!]^A_{_} \rangle$ is said to be a *non-deterministic lambda transition system* (or NLTS for short) if $\langle A, \Downarrow^{\mathrm{may}}, \Uparrow^{\mathrm{may}} \rangle$ is a NATS and $[\![-]\!]^A_{_}$ is a compositional valuation (as above) satisfying all the laws of $\mathbf{\Lambda}_{\mathrm{n}}$ in Theorem 2.3(ii). Note that $\mathbf{\Lambda}^o_{\mathrm{n}}$ is trivially an NLTS. We can prove:

Theorem 3.2 (Canonical Model)
$\langle \mathbb{D}, \Downarrow^{\mathrm{may}}, \Uparrow^{\mathrm{may}}, [\![-]\!]^{\mathbb{D}}_{_} \rangle$ is an NLTS. In this sense, $\mathbb{D}$ is a model of $\mathbf{\Lambda}_{\mathrm{n}}$. $\square$

4 Logical Interpretation

The key denotational type δ may be given a *logical* interpretation. In the framework of Abramsky's domain logic, a denotational type τ gives a prescription of how a proof system may be assembled specifically for the computational situation described by τ. Here, we "turn the handle" of Abramsky's programme to obtain the proof system $\mathcal{L}(\delta)$ for the type δ. Our standard reference is of course [Abr91b] and we shall assume that the reader is familiar with it.

The logic $\mathcal{L}(\delta)$ has two sorts: π for *processes* and κ for *capabilities*. We write $\mathcal{L}_{\infty\pi}$ and $\mathcal{L}_{\infty\kappa}$ (instead of $\mathcal{L}(\delta)_{\infty\pi}$ say, for simplicity) for the class of formulae of sort π and κ respectively. They are defined inductively as follows: (recall $\sigma \in \{ \pi, \kappa \}$),

$$\frac{\{ \phi_i \in \mathcal{L}_{\infty\sigma} \}_{i \in I}}{\bigvee_{i \in I} \phi_i, \bigwedge_{i \in I} \phi_i \in \mathcal{L}_{\infty\sigma}}$$

$$\frac{\phi, \psi \in \mathcal{L}_{\infty\pi}}{(\phi \to \psi)_\perp \in \mathcal{L}_{\infty\kappa}} \qquad \frac{\phi \in \mathcal{L}_{\infty\kappa}}{\Box\phi, \Diamond\psi \in \mathcal{L}_{\infty\pi}}.$$

The sub-languages $\mathcal{L}_{\omega\pi}$ and $\mathcal{L}_{\omega\kappa}$ are obtained by restricting I to range over finite sets. We abbreviate $\bigvee_{i \in \varnothing} \phi_i$ and $\bigwedge_{i \in \varnothing} \phi_i \in \mathcal{L}_{\infty\sigma}$ as f and t respectively.

The proof system $\mathcal{L}(\delta)$ axiomatises the following:

logical *entailment* relation	$\leqslant_\sigma$,
logical *equivalence* relation	$=_\sigma$,
coprimeness predicate	$\mathsf{coP}_\sigma(-)$;

for both sorts $\sigma \in \{ \pi, \kappa \}$. We write $\mathcal{L}_{\infty\sigma} \vdash \phi \leqslant_\sigma \psi$ to mean that $\phi \leqslant_\sigma \psi$ is provable; similarly for $\mathcal{L}_{\omega\sigma} \vdash \mathsf{coP}_\sigma(\phi)$ *etc.* The axioms are organised into four groups *viz.* logical, lifted function space, powerdomain and coprimeness as presented in the Appendix.

Domain Logic: an Overview The correct intuition is topological: think of the (finitary) logic $\mathcal{L}_{\omega\pi}$ as an equivalent presentation of the space $\mathbb{D}$. Since an SFP

Valuation of variables and constants is standard; for $M \in \mathbf{\Lambda}_{\mathrm{n}}$ and $\rho \in \mathsf{Env}$,

$$\begin{array}{rclrcl} [\![M \cdot N]\!]^{\mathbb{D}}_{\rho} & \stackrel{\text{def}}{=} & [\![M]\!]^{\mathbb{D}}_{\rho} \cdot [\![N]\!]^{\mathbb{D}}_{\rho}; & [\![\lambda x.M]\!]^{\mathbb{D}}_{\rho} & \stackrel{\text{def}}{=} & \mathrm{gr}(\{\!|\mathrm{up}_{[\mathbb{D}\to\mathbb{D}]}(\lambda\xi.[\![M]\!]^{\mathbb{D}}_{\rho[x\mapsto\xi]})|\!\}_{[\mathbb{D}\to\mathbb{D}]_\perp}); \\ [\![M + N]\!]^{\mathbb{D}}_{\rho} & \stackrel{\text{def}}{=} & [\![M]\!]^{\mathbb{D}}_{\rho} + [\![N]\!]^{\mathbb{D}}_{\rho}; & [\![\lambda^{\mathrm{v}} x.M]\!]^{\mathbb{D}}_{\rho} & \stackrel{\text{def}}{=} & \mathrm{gr}(\{\!|\mathrm{up}_{[\mathbb{D}\to\mathbb{D}]}(\lambda\xi.\Psi_{\mathbb{D}}(U_{M,x,\rho}(\xi)))|\!\}_{[\mathbb{D}\to\mathbb{D}]_\perp}). \end{array}$$

where

$$U_{M,x,\rho}(\xi) \stackrel{\text{def}}{=} \left\{ \left\{ \begin{array}{ll} [\![M]\!]^{\mathbb{D}}_{\rho[x\mapsto \mathrm{gr}(\{\!|u|\!\}_{[\mathbb{D}\to\mathbb{D}]_\perp})]} & \text{if } u \neq -; \\ - & \text{else.} \end{array} \right\} : u \in \mathrm{fun}(\xi) \right\}^*$$

Figure 3.

domain is *coherent* (*i.e.* its compact-opens are closed under finite intersection and hence form a distributive lattice $\langle \mathsf{K}\Omega(\mathbb{D}), \subseteq \rangle$ which generates the topology), a finitary presentation of the logic suffices. A formula ϕ in $\mathcal{L}_{\omega\pi}$ is just a syntactic representation of a compact-open subset of $\mathbb{D}$. Moreover, we can recover $\mathsf{K}(\mathbb{D})$ (the poset of compact elements of $\mathbb{D}$) from $A = \mathsf{K}\Omega(\mathbb{D})$ as follows: $a \in A$ is *coprime* if $a \leqslant b \vee c$ implies $a \leqslant b$ or $a \leqslant c$. Writing $\mathsf{coP}(A)$ as the coprimes of A, we have $\mathsf{K}(\mathbb{D}) \cong \mathsf{coP}(\mathsf{K}\Omega(\mathbb{D}))^{\mathrm{op}}$. There are "enough" coprimes, in the sense that each $U \in \mathsf{K}\Omega(\mathbb{D})$ can be written as a disjunction of coprimes. Now we define $\mathcal{LA} \stackrel{\text{def}}{=} \langle \mathcal{L}_{\omega\pi}/{=_\pi}, \leqslant_\pi/{=_\pi} \rangle$, the Lindenbaum Algebra of $\mathcal{L}_{\omega\pi}$. Given a distributive lattice A, the *spectrum* of A written $\mathsf{Spec}(A)$, is the space constructed with the set of prime filters as points, with topology generated by $U_a \stackrel{\text{def}}{=} \{ X \in \mathsf{Spec}(A) : a \in x \}$ for $a \in A$; the domain ordering can be recovered as the specialisation ordering (= inclusion of filters). Applying a key result of [Abr91b], we get:

Theorem 4.1 (Stone Duality) *$\mathbb{D}$ and $\mathcal{LA}$ are Stone duals i.e. $\mathbb{D} \cong \mathsf{Spec}(\mathcal{LA})$ and $\mathsf{K}\Omega(\mathbb{D}) \cong \mathcal{LA}$.* □

Satisfaction Semantics of the Logic Though intended for the SFP domain $\mathbb{D}$, the logic $\mathcal{L}(\delta)$ makes perfect sense in the general setting of NATS. Given any NATS A, we define two satisfaction relations:

$$\begin{array}{rcl} \models^A_\pi & \subseteq & A \times \mathcal{L}_{\infty\pi}, \\ \models^A_\kappa & \subseteq & (A^A \cup \{-\}) \times \mathcal{L}_{\infty\kappa}. \end{array}$$

For p and k ranging over the appropriate domains, we define $s \models^A_\sigma \phi$ (read "s satisfies the σ-formula ϕ") inductively as follows:

$$\begin{array}{rcl} p \models^A_\pi \Box\phi & \stackrel{\text{def}}{=} & \forall k \in \mathrm{cap}(p).k \models^A_\kappa \phi; \\ p \models^A_\pi \Diamond\phi & \stackrel{\text{def}}{=} & \exists k \in \mathrm{cap}(p).k \models^A_\kappa \phi; \\ k \models^A_\kappa (\phi \to \psi)_\perp & \stackrel{\text{def}}{=} & k \neq - \quad \& \\ & & [\forall p \in A.p \models^A_\pi \phi \implies k(p) \models^A_\pi \psi]. \end{array}$$

The cases of ϕ being a conjunction or disjunction are standard. Clearly, for any $a \in A$, $a \models^A_\sigma \mathsf{t}$ and $\neg[a \models^A_\sigma \mathsf{f}]$; writing $\lambda \equiv (\mathsf{t} \to \mathsf{t})_\perp$, we have $a\Downarrow^{\mathrm{may}}$ iff $a \models^A_\pi \Diamond\lambda$, and $a\Downarrow^{\mathrm{must}}$ iff $a \models^A_\pi \Box\lambda$. We can now spell out precisely the relationship between coprime formulae and compact elements of $\mathbb{D}$ in the following result derived from [Abr91b]:

Theorem 4.2 (Coprimeness) *For any $\phi \in \mathcal{L}_{\omega\pi}$ with $\mathcal{L}_{\omega\pi} \vdash \mathsf{coP}_\pi(\phi)$, there exists $\mathsf{K}(\phi) \in \mathsf{K}(\mathbb{D})$ such that $d \models^{\mathbb{D}}_\pi \phi \iff \mathsf{K}(\phi) \sqsubseteq_{\mathbb{D}} d$ for any $d \in \mathbb{D}$.* □

Given any NATS A, we define

$$A \models \phi \leqslant_\pi \psi \stackrel{\text{def}}{=} \forall a \in A.a \models^A_\pi \phi \implies a \models^A_\pi \psi.$$

The logic $\mathcal{L}_{\omega\pi}$ is sound and complete w.r.t. the satisfaction semantics in the canonical model $\mathbb{D}$:

Theorem 4.3 (Completeness) *For any $\phi, \psi \in \mathcal{L}_{\omega\pi}$, $\mathcal{L}_{\omega\pi} \vdash \phi \leqslant_\pi \psi \iff \mathbb{D} \models \phi \leqslant_\pi \psi$.* □

However, in general, a NATS A (*e.g.* $\mathbf{\Lambda}^o_{\mathrm{n}}$) does not have enough structure to guarantee even the soundness of $\mathcal{L}_{\omega\pi}$, let alone completeness. In the case of $\mathbf{\Lambda}^o_{\mathrm{n}}$, soundness fails because not every coprime formula (= compact element of $\mathbb{D}$) is definable in $\mathbf{\Lambda}^o_{\mathrm{n}}$, thus invalidating, say rule (fs-5) in the Appendix.

Endogenous Program Logic We let ? range over maps from variables to $\mathcal{L}_{\omega\pi}$. For any $M \in \mathbf{\Lambda}_{\mathrm{n}}$ we define $? \vdash M : \phi$ (read "M satisfies the formula ϕ under the assumption ?") inductively in Figure 4.

$$\frac{\{\,? \vdash M : \phi_i\,\}_{i\in I}}{? \vdash M : \bigwedge_{i\in I}\phi_i} \qquad \frac{\{\,?\,[x \mapsto \phi_i] \vdash M : \psi\,\}_{i\in I}}{?\,[x \mapsto \bigvee_{i\in I}\phi_i] \vdash M : \psi} \qquad \frac{?\,[x \mapsto \phi] \vdash M : \psi}{? \vdash \lambda x.M : \Box(\phi \to \psi)_\perp}$$

$$\frac{\forall x.?\,(x) \leqslant_\pi \Delta(x) \quad \Delta \vdash M : \phi \quad \mathcal{L}_{\omega\pi} \vdash \phi \leqslant_\pi \psi}{? \vdash M : \psi} \qquad \frac{? \vdash M : \Diamond\phi}{? \vdash M + N : \Diamond\phi} \qquad \frac{? \vdash N : \Diamond\phi}{? \vdash M + N : \Diamond\phi}$$

$$\frac{? \vdash M : \Box\phi \qquad ? \vdash N : \Box\phi}{? \vdash M + N : \Box\phi} \qquad \frac{?\,[x \mapsto \Box\phi] \vdash M : \Box\psi}{? \vdash \lambda^{\mathrm{v}} x.M : \Box(\Box\phi \to \Box\psi)_\perp} \qquad \frac{?\,[x \mapsto \Diamond\phi] \vdash M : \Diamond\psi}{? \vdash \lambda^{\mathrm{v}} x.M : \Box(\Diamond\phi \to \Diamond\psi)_\perp}$$

$$\frac{? \vdash M : \Box\bigvee_{i\in I}(\gamma_i \to \phi_i)_\perp \quad ? \vdash N : \bigwedge_{i\in I}\gamma_i}{? \vdash M \cdot N : \Box\bigvee_{i\in I}\phi_i} \qquad \frac{? \vdash M : \Diamond(\gamma \to \Diamond\phi)_\perp \quad ? \vdash N : \gamma}{? \vdash M \cdot N : \Diamond\phi}$$

Figure 4.

Semantics of Program Logic in NLTS Let A be an NLTS. Define, for $M \in \mathbf{\Lambda}_{\mathrm{n}}^{o}$ and $\rho \in \mathsf{Env}(A)$,

$$\rho \vDash^A M : \phi \quad \stackrel{\text{def}}{=} \quad [\![M]\!]_\rho^A \vDash_\pi^A \phi.$$

In the presence of assumptions $?$ on the variables,

$$? \vDash^A M : \phi \quad \stackrel{\text{def}}{=} \quad \forall \rho.\rho \vDash^A ? \implies \rho \vDash^A M : \phi$$

where $\rho \vDash^A ? \stackrel{\text{def}}{=} \forall x \in \mathsf{Var}.\rho(x) \vDash_\pi^A ?\,(x)$. Given an NLTS A we say that A has *program logic completeness* just in case for any $M \in \mathbf{\Lambda}_{\mathrm{n}}$ and any $\phi \in \mathcal{L}_{\omega\pi}$, $? \vdash M : \phi \iff ? \vDash^A M : \phi$. We can show that $\mathbb{D}$ is program logic complete:

Theorem 4.4 (Completeness) *For any $\phi \in \mathcal{L}_{\omega\pi}$, for any $M \in \mathbf{\Lambda}_{\mathrm{n}}$, $? \vdash M : \phi \iff ? \vDash^{\mathbb{D}} M : \phi$.* $\Box$

5 Correspondence Results

Logic satisfaction induces a preorder on a NATS in exactly the same way as testing preorder. What is the relationship between the two induced preorders? We write: for $s \in A$ or $A^A \cup \{-\}$,

$$\mathcal{L}_{\infty\sigma}(s) \quad \stackrel{\text{def}}{=} \quad \{\,\phi \in \mathcal{L}_{\infty\sigma} : s \vDash_\sigma^A \phi\,\},$$

similarly for $\mathcal{L}_{\omega\sigma}(s)$; further, write $a \sqsubseteq_\infty^{\mathcal{L}} b$ for $\mathcal{L}_{\infty\pi}(a) \subseteq \mathcal{L}_{\infty\pi}(b)$ and similarly for $a \sqsubseteq_\omega^{\mathcal{L}} b$. In the case of $\mathbb{D}$, by appealing to the Testing Characterization, Internal Full Abstraction and Stone Duality Theorems, we can prove:

Theorem 5.1 (Correspondence) *For $d, e \in \mathbb{D}$, $d \sqsubseteq_\infty^{\mathrm{T}} e \iff d \sqsubseteq^{\mathrm{B}} e \iff d \sqsubseteq_{\mathbb{D}} e \iff d \sqsubseteq_\omega^{\mathcal{L}} e$.* $\Box$

The situation is understandably much less clear-cut for an arbitrary NATS. But testing and satisfaction preorders do coincide for a reasonable sub-class of NATSs. To establish this, we first define $\mathbf{N}\mathcal{L}_{\infty\pi}$, the class of *normal forms* for the infinitary logic $\mathcal{L}_{\infty\pi}$ as follows:

$$\frac{\{\,\delta_i \in \mathbf{N}\mathcal{L}_{\infty\pi}\,\}_{i\in I} \quad I \neq \varnothing}{\Box\bigvee_{i\in I}(\phi_i \to \delta_i)_\perp, \Diamond\bigwedge_{i\in I}(\phi_i \to \delta_i)_\perp \in \mathbf{N}\mathcal{L}_{\infty\pi}};$$

$$\frac{\{\,\delta_i \in \mathbf{N}\mathcal{L}_{\infty\pi}\,\}_{i\in I}}{\bigvee_{i\in I}\delta_i, \bigwedge_{i\in I}\delta_i \in \mathbf{N}\mathcal{L}_{\infty\pi}}.$$

Lemma 5.2 (Normal Form) *For any $\phi \in \mathcal{L}_{\infty\pi}$, there is a $\psi \in \mathbf{N}\mathcal{L}_{\infty\pi}$ such that $\mathcal{L}_{\infty\pi} \vdash \phi =_\pi \psi$. The axioms used are: ($\wedge$-dist), ($\vee$-dist), ($\Box$-$\wedge$) and ($\Diamond$-$\vee$) which are all sound for any NATS.* $\Box$

We identify a class of NATS satisfying a kind of weak continuity property known as *approximability*. Let A be a NATS. A map $\lceil - \rceil$ from A to $\mathcal{L}_{\infty\pi}$ is called a *logical assignment* if $a \vDash_\pi^A \lceil a \rceil$ for each $a \in A$. Given any $\tau \in \mathrm{TESTS}_{\infty\pi}$ and any logical assignment $\lceil - \rceil$, we define an associated formula $(\tau)^{\mathcal{L}} \in \mathcal{L}_{\infty\pi}$ inductively as follows: (recall $\lambda \equiv (\mathsf{t} \to \mathsf{t})_\perp$)

$$\begin{aligned}
(a\tau)^{\mathcal{L}} &\stackrel{\text{def}}{=} (\lceil a \rceil \to (\tau)^{\mathcal{L}})_\perp; \\
(\forall\tau)^{\mathcal{L}} &\stackrel{\text{def}}{=} \Box(\tau)^{\mathcal{L}} \wedge \Box\lambda; \\
(\exists\tau)^{\mathcal{L}} &\stackrel{\text{def}}{=} \Diamond(\tau)^{\mathcal{L}} \wedge \Diamond\lambda;
\end{aligned}$$

$(\bigwedge_{i\in I}\tau_i)^{\mathcal{L}} \stackrel{\text{def}}{=} \bigwedge_{i\in I}(\tau_i)^{\mathcal{L}}$, similarly for $(\bigvee_{i\in I}\tau_i)^{\mathcal{L}}$. We call $(\tau)^{\mathcal{L}}$ the *logical encoding* of τ with respect to $\lceil - \rceil$. A NATS A is said to be *approximable* if there is a logical assignment $\lceil - \rceil$ such that for each $a \in A$ and each $\tau \in \mathrm{TESTS}_{\infty\pi}$, whenever $\mathrm{obs}_\pi(a, \tau)$ then $a \vDash_\pi^A (\tau)^{\mathcal{L}}$,

where $(-)^{\mathcal{L}}$ is defined w.r.t. $\lceil - \rceil$. Next, we define a translation $(-)^{\mathrm{T}}$ from $\mathbf{N}\mathcal{L}_{\infty\pi}$ to $\mathrm{TESTS}_{\infty\pi}$ inductively as follows: (note $I \neq \varnothing$ by def. of $\mathbf{N}\mathcal{L}_{\infty\pi}$)

$$\begin{array}{rcl}
(\bigwedge_{i\in I}\phi_i)^{\mathrm{T}} & \stackrel{\mathrm{def}}{=} & \bigwedge_{i\in I}(\phi_i)^{\mathrm{T}}; \\
(\bigvee_{i\in I}\phi_i)^{\mathrm{T}} & \stackrel{\mathrm{def}}{=} & \bigvee_{i\in I}(\phi_i)^{\mathrm{T}}; \\
(\Box\bigvee_{i\in I}(\phi_i\to\psi_i)_\perp)^{\mathrm{T}} & \stackrel{\mathrm{def}}{=} & \forall\bigvee_{i\in I}\bigwedge_{a:a\models\phi_i} a(\psi_i)^{\mathrm{T}}; \\
(\Diamond\bigwedge_{i\in I}(\phi_i\to\psi_i)_\perp)^{\mathrm{T}} & \stackrel{\mathrm{def}}{=} & \exists\bigwedge_{i\in I}\bigwedge_{a:a\models\phi_i} a(\psi_i)^{\mathrm{T}}.
\end{array}$$

Lemma 5.3 *(i) For any $\phi \in \mathbf{N}\mathcal{L}_{\infty\pi}$ and $a \in A$, $a \models^A_\pi \phi \iff \mathrm{obs}_\pi(a,(\phi)^{\mathrm{T}})$.*

(ii) For any $a \in A$ an approximable NATS *and any $\tau \in \mathrm{TESTS}_{\infty\pi}$, $\mathrm{obs}_\pi(a,\tau) \iff a \models^A_\pi (\tau)^{\mathcal{L}}$.* □

Theorem 5.4 (Logic Characterization) *For $a, b \in A$ an approximable* NATS*, $a \sqsubseteq^{\mathrm{T}}_\infty b \iff a \sqsubseteq^{\mathcal{L}}_\infty b$.*

What is the relationship between $\sqsubseteq^{\mathcal{L}}_\omega$ and $\sqsubseteq^{\mathcal{L}}_\infty$? We introduce the axioms of *bounded non-determinism* (**BN**), *finite approximability* (**FA**) and *Compact Graph* (**CG**) in Figure 5. (**BN**) is a statement of compactness and (**FA**) reflects the fact that only closed sets are taken as elements of a powerdomain construction (both due to Abramsky). (**CG**) is a property of continuous functions between Lawson compact spaces (since $\mathbb{D}$ is "2/3 SFP"). A NATS A is said to be *finitary* just in case the axioms of (**BN**), (**FA**) and (**CG**) are valid in A. We claim that $\mathbb{D}$ is finitary but not $\mathbf{\Lambda}^o_{\mathrm{n}}$ — (**FA**) is not valid, just consider $\boldsymbol{\Upsilon}$ in Example 2.1. More generally, we can show that:

Theorem 5.5 (Finitary Adequacy) *For $a, b \in A$, a finitary* NATS*, $a \sqsubseteq^{\mathcal{L}}_\infty b \iff a \sqsubseteq^{\mathcal{L}}_\omega b$.* □

Adequacy and Non-Full Abstraction

How much (or accurate) do the denotational and logical interpretations of $\mathbf{\Lambda}_{\mathrm{n}}$ in $\mathbb{D}$ tell us about the actual operational behaviour (*e.g.* convergence property, $\sqsubseteq^{\mathrm{B}}$) of programs of $\mathbf{\Lambda}_{\mathrm{n}}$? For any NLTS A, we say that it is *weakly adequate* for $\mathbf{\Lambda}_{\mathrm{n}}$ if for any $M \in \mathbf{\Lambda}^o_{\mathrm{n}}$,

$$\begin{array}{lcl}
M\Downarrow^{\mathrm{may}} & \iff & [\![M]\!]^A\Downarrow^{\mathrm{may}} \quad \& \\
M\Downarrow^{\mathrm{must}} & \iff & [\![M]\!]^A\Downarrow^{\mathrm{must}}.
\end{array}$$

We say that the NLTS A is *fully abstract* for $\mathbf{\Lambda}^o_{\mathrm{n}}$ if for any $M, N \in \mathbf{\Lambda}^o_{\mathrm{n}}$, $M \sqsubseteq^{\mathrm{B}} N$ iff $[\![M]\!]^A \sqsubseteq^{\mathrm{B}} [\![N]\!]^A$. In the case of A being $\mathbb{D}$, full abstraction is the property: for any $M, N \in \mathbf{\Lambda}^o_{\mathrm{n}}$,

$$M \sqsubseteq^{\mathrm{B}} N \iff [\![M]\!]^{\mathbb{D}} \sqsubseteq_{\mathbb{D}} [\![N]\!]^{\mathbb{D}}.$$

This definition is reasonable since $\sqsubseteq^{\mathrm{B}}$ (rather than a contextual preorder like $\sqsubseteq^{\mathrm{cxt}}$ say) offers arguably the finest distinction between terms which is computationally meaningful.

Theorem 5.6 *The* NLTS $\mathbb{D}$ *is weakly adequate but not fully abstract for $\mathbf{\Lambda}^o_{\mathrm{n}}$.* □

To see why full abstraction fails, consider $\boldsymbol{\Upsilon}$ as defined in Example 2.4. We claim:

$$\begin{array}{rcl}
\boldsymbol{\Upsilon} + \mathbf{YK} & \not\sqsubseteq^{\mathrm{B}} & \boldsymbol{\Upsilon}; \\
[\![\boldsymbol{\Upsilon} + \mathbf{YK}]\!]^{\mathbb{D}} & = & [\![\boldsymbol{\Upsilon}]\!]^{\mathbb{D}};
\end{array}$$

though we do have $\boldsymbol{\Upsilon} + \mathbf{YK} \sqsubseteq^{\mathrm{B}}_\omega \boldsymbol{\Upsilon}$. This mismatch essentially arises from the fact that $\mathbb{D}$ is a finitary NATS whereas $\mathbf{\Lambda}^o_{\mathrm{n}}$ is not. But full abstraction also fails for another reason: we say that *(deterministic) parallel may-converge testing* is definable in a NATS A if there is some $\mathbf{p} \in A$ such that for any $a, b \in A$ both $\mathbf{p}$ and $\mathbf{p}a$ converge deterministically, further ($\mathbf{I}$ is the identity)

$$\begin{array}{lcl}
[a\Downarrow^{\mathrm{may}} \text{ or } b\Downarrow^{\mathrm{may}}] & \Longrightarrow & \mathbf{p}ab \sim^{\mathrm{B}} \mathbf{I} \quad \& \\
[a\Uparrow^{\mathrm{must}} \ \& \ b\Uparrow^{\mathrm{must}}] & \Longrightarrow & \mathbf{p}ab \sim^{\mathrm{B}} \mathbf{\Omega}.
\end{array}$$

Note that such a $\mathbf{p}$ is definable in $\mathbb{D}$ but not in $\mathbf{\Lambda}^o_{\mathrm{n}}$ (in which not even *deterministic* may-converge testing is definable despite the presence of call-by-value abstraction).

Elsewhere [Ong92b], we have investigated properties of a concurrent extension of $\mathbf{\Lambda}_{\mathrm{n}}$ obtained by augmenting it with an appropriate concurrent binary operator $\parallel$ (*cf.* [Bou92]). The associated applicative bisimulation is a precongruence [Ong92a]. The operator $\parallel$ is carefully chosen so that

- it introduces parallelism to $\mathbf{\Lambda}^o_{\mathrm{n}}$ satisfying:

$$\begin{array}{lcl}
[a\Downarrow^{\mathrm{may}} \text{ or } b\Downarrow^{\mathrm{may}}] & \iff & a \parallel b\Downarrow^{\mathrm{may}}, \\
[a\Downarrow^{\mathrm{must}} \text{ or } b\Downarrow^{\mathrm{must}}] & \iff & a \parallel b\Downarrow^{\mathrm{must}};
\end{array}$$

- semantically, $\parallel$ is in some sense, the best continuous approximation to McCarthy's ambiguous operator amb [McC63] (which is not even $\sqsubseteq_{\mathrm{EM}}$-monotonic, let alone continuous). Operational properties of amb in the setting of the Lazy Lambda Calculus have been studied in [HM92].

An Approach to Full Abstraction How should the language $\mathbf{\Lambda}_{\mathrm{n}}$ be modified for which $\mathbb{D}$ becomes fully abstract — bearing in mind the two factors which account for the failure? To circumvent the finitary

Writing $\mathrm{Fin}(I)$ as the set of finite subsets of I,

$$\textbf{(BN)}\ \Box \bigvee_{i\in I} \phi_i \leqslant_\pi \bigvee_{P\in \mathrm{Fin}(I)} \Box \bigvee_{p\in P} \phi_p \quad (\phi_i \in \mathcal{L}_{\omega\kappa}) \qquad \textbf{(FA)}\ \bigwedge_{P\in \mathrm{Fin}(I)} \Diamond \bigwedge_{p\in P} \phi_p \leqslant_\pi \Diamond \bigwedge_{i\in I} \phi_i \quad (\phi_i \in \mathcal{L}_{\omega\kappa})$$

$$\textbf{(CG)}\quad (\bigwedge_{i\in I} \phi_i \to \bigvee_{j\in J} \psi_j)_\perp \quad \leqslant_\kappa \quad \bigvee_{P\in \mathrm{Fin}(I), Q\in \mathrm{Fin}(J)} (\bigwedge_{p\in P} \phi_p \to \bigvee_{q\in Q} \psi_q)_\perp \quad (\mathcal{L}_{\omega\pi} \vdash \mathsf{coP}_\pi(\phi_i), \mathsf{coP}_\pi(\psi_j))$$

Figure 5.

model / infinitary language mismatch, we consider a *finitary* notion of applicative bisimulation, $\precsim^{\mathrm{fin}}$ such that $\boldsymbol{\Upsilon} + \mathbf{YK} \precsim^{\mathrm{fin}} \boldsymbol{\Upsilon}$, say. We say that an NLTS A is *coprime expressive* if for any $\phi \in \mathsf{coP}_\pi$, there is some $a_\phi \in A$ such that $a_\phi \models^A_\pi \psi$ iff $\mathcal{L}_{\omega\pi} \vdash \phi \leqslant_\pi \psi$ for any $\psi \in \mathsf{coP}_\pi$. By Stone Duality Theorem, this property is equivalent to the internal definability of all compact elements of $\mathbb{D}$ in A.

Conjecture Let $\mathbf{C}$ be a set of constants with interpretation in $\mathbb{D}$. Suppose $\mathbb{D}$ is weakly adequate for $\mathbf{\Lambda}_{\mathrm{n}}(\mathbf{C})$ in which both the (deterministic) may-converge testing and must-converge testing are definable. Finitary full abstraction of $\mathbb{D}$ for $\mathbf{\Lambda}_{\mathrm{n}}(\mathbf{C})$ (i.e. for any $M, N \in \mathbf{\Lambda}_{\mathrm{n}}(\mathbf{C})^o$, $M \precsim^{\mathrm{fin}} N$ iff $[\![M]\!]^{\mathbb{D}} \sqsubseteq_{\mathbb{D}} [\![N]\!]^{\mathbb{D}}$) is equivalent to the validity of any of the following properties in $\mathbf{\Lambda}_{\mathrm{n}}(\mathbf{C})$:

(i) definability of deterministic parallel may converge and must converge testing,

(ii) coprime expressiveness,

(iii) program logic completeness. $\Box$

Acknowledgements This research was supported by a fellowship of Trinity College, Cambridge, England, and also by NUS grant RP 900632.

References

[Abr87] S. Abramsky. Observation equivalence as a testing equivalence. *Theoretical Comp. Sc.*, 225–241, 1987.

[Abr91a] S. Abramsky. A domain equation for bisimulation. *Information and Computation*, 92:161–218, 1991.

[Abr91b] S. Abramsky. Domain theory in logical form. *Annals of Pure and Applied Logic*, 51:1–77, 1991.

[AO92] S. Abramsky and C.-H. L. Ong. *Full Abstraction in the Lazy Lambda Calculus.* Technical Report 259, Cambridge University Computer Laboratory, 1992. 105 pp. To appear in *Info. and Comp.* in 1993.

[Bar84] H. Barendregt. *The Lambda Calculus.* North-Holland, revised edition, 1984.

[Bou92] G. Boudol. Lambda calculi for (strict) parallel functions. *Info. and Comp.*, 1992. To appear.

[dP91] U. de'Liguoro and A. Piperno. Must preorder in non-deterministic untyped lambda calculus. In *Proc. CAAP*, pages 203– 220, Springer, 1991.

[HA80] M. C. B. Hennessy and E. A. Ashcroft. A mathematical semantics for a non-deterministic typed lambda calculus. *Theoretical Computer Science*, 11:227–245, 1980.

[HM92] J. Hughes and A. Moran. A semantics for locally bottom-avoiding choice. In *Proc. Glasgow Func. Prog. Workshop '92*, 1992.

[JP90] R. Jagadeesan and P. Panangaden. A domain-theoretic model for a higher-order process calculus. In *Proc. ICALP*, Springer, 1990. LNCS Vol. 443.

[McC63] J. McCarthy. A basis for a mathematical theory of computation. In P. Braffort and D. Hirschberg, editors, *Computer Programming and Formal Systems*, pages 33–70, North-Holland, 1963.

[Ong92a] C.-H. L. Ong. The concurrent lambda calculus 1: a general pre-congruence theorem for applicative bisimulation. 1992. Unpublished Manuscript, 27pp.

[Ong92b] C.-H. L. Ong. Functions, concurrency and non-determinism. 1992. Unpublished Manuscript, 70pp.

[Plo76] G. D. Plotkin. A powerdomain construction. *SIAM J. Computing*, 5(3):452–487, 1976.

[San92] D. Sangiorgi. The lazy lambda calculus in a concurrency scenario. In *Proc. IEEE LICS*, 1992.

A Proof System for $\mathcal{L}(\delta)$

Logical Axioms The following axioms give $\mathcal{L}_{\infty\pi}$ and $\mathcal{L}_{\infty\kappa}$ the structure of a *completely distributive lattice.*

$$(\leqslant_\sigma\text{-ref}) \quad \phi \leqslant_\sigma \phi \qquad\qquad (\leqslant_\sigma\text{-trans}) \quad \frac{\phi \leqslant_\sigma \psi \quad \psi \leqslant_\sigma \chi}{\phi \leqslant_\sigma \chi}$$

$$(=_\sigma\text{-I}) \quad \frac{\phi \leqslant_\sigma \psi \quad \psi \leqslant_\sigma \phi}{\phi =_\sigma \psi} \qquad\qquad (=_\sigma\text{-E}) \quad \frac{\phi =_\sigma \psi}{\phi \leqslant_\sigma \psi \quad \psi \leqslant_\sigma \phi}$$

$$(\wedge\text{-I}) \quad \frac{\{\, \phi \leqslant_\sigma \psi_i \,\}_{i\in I}}{\phi \leqslant_\sigma \bigwedge_{i\in I} \psi_i} \qquad\qquad (\wedge\text{-E}) \quad \bigwedge_{i\in I} \phi_i \leqslant_\sigma \phi_j \quad (j \in I)$$

$$(\vee\text{-I}) \quad \frac{\{\, \phi_i \leqslant_\sigma \psi \,\}_{i\in I}}{\bigvee_{i\in I} \phi_i \leqslant_\sigma \psi} \qquad\qquad (\vee\text{-E}) \quad \phi_j \leqslant_\sigma \bigvee_{i\in I} \phi_i \quad (j \in I)$$

$$(\wedge\text{-dist}) \quad \bigwedge_{i\in I} \bigvee_{j\in J_i} \phi_{i,j} =_\sigma \bigvee_{f\in\prod_{i\in I} J_i} \bigwedge_{i\in I} \phi_{i,f(i)} \qquad\qquad (\vee\text{-dist}) \quad \bigvee_{i\in I} \bigwedge_{j\in J_i} \phi_{i,j} =_\sigma \bigwedge_{f\in\prod_{i\in I} J_i} \bigvee_{i\in I} \phi_{i,f(i)}$$

Lifted Function Space Axioms These axioms are an amalgation of those for function and lifted types in [Abr91b].

$$\text{(fs-1)} \quad (\phi \to \bigwedge_{i\in I} \psi_i)_\perp =_\kappa \bigwedge_{i\in I} (\phi \to \psi_i)_\perp \quad (I \neq \varnothing)$$

$$\text{(fs-2)} \quad (\bigvee_{i\in I} \phi_i \to \psi)_\perp =_\kappa \bigwedge_{i\in I} (\phi_i \to \psi)_\perp \quad (I \neq \varnothing)$$

$$\text{(fs-3)} \quad (\phi \to \mathsf{t})_\perp \leqslant_\kappa (\mathsf{t} \to \mathsf{t})_\perp \qquad\qquad \text{(fs-4)} \quad (\mathsf{f} \to \psi)_\perp =_\kappa (\mathsf{t} \to \mathsf{t})_\perp$$

$$\text{(fs-5)} \quad \frac{}{(\phi \to \bigvee_{i\in I} \psi_i)_\perp =_\kappa \bigvee_{i\in I} (\phi \to \psi_i)_\perp} \quad \mathsf{coP}_\pi(\phi)$$

$$\text{(fs-6)} \quad \frac{\phi' \leqslant_\pi \phi \quad \psi \leqslant_\pi \psi'}{(\phi \to \psi)_\perp \leqslant_\kappa (\phi' \to \psi')_\perp}$$

Powerdomain Axioms

$$(\Box\text{-}\wedge) \quad \Box \textstyle\bigwedge_{i\in I} \phi_i =_\pi \bigwedge_{i\in I} \Box\phi_i \qquad (\Box\text{-}\leqslant) \quad \frac{\phi \leqslant_\kappa \psi}{\Box\phi \leqslant_\pi \Box\psi} \qquad (\Box\text{-f}) \quad \Box\mathsf{f} =_\pi \mathsf{f}$$

$$(\Diamond\text{-}\vee) \quad \Diamond \textstyle\bigvee_{i\in I} \phi_i =_\pi \bigvee_{i\in I} \Diamond\phi_i \qquad (\Diamond\text{-}\leqslant) \quad \frac{\phi \leqslant_\kappa \psi}{\Diamond\phi \leqslant_\pi \Diamond\psi}$$

$$(\Box\text{-}\wedge) \quad \Box\phi \wedge \Diamond\psi \leqslant_\pi \Diamond(\phi \wedge \psi) \qquad (\Box\text{-}\vee) \quad \Box(\phi \vee \psi) \leqslant_\pi \Box\phi \vee \Diamond\psi$$

Coprimeness Axioms

$$\mathsf{coP}_\sigma(\mathsf{t}) \qquad \frac{\mathsf{coP}_\sigma(\phi) \qquad \phi =_\sigma \psi}{\mathsf{coP}_\sigma(\psi)} \qquad \frac{\{\, \mathsf{coP}_\kappa(\phi_i) \,\}_{i\in I} \quad I \neq \varnothing}{\mathsf{coP}_\pi(\Box \bigvee_{i\in I} \phi_i \wedge \bigwedge_{i\in I} \Diamond\phi_i)}$$

$$\frac{\{\, \mathsf{coP}_\pi(\phi_i), \mathsf{coP}_\pi(\psi_i) \,\}_{i\in I} \qquad \forall J \subseteq I.\exists K \subseteq I.[\bigwedge_{j\in J} \phi_j =_\pi \bigvee_{k\in K} \phi_k] \qquad [\forall j \in J.\forall k \in K.\psi_k \leqslant_\pi \psi_j]}{\mathsf{coP}_\kappa(\bigwedge_{i\in I} (\phi_i \to \psi_i)_\perp)}$$