Full Abstractness for a Functional/Concurrent Language With Higher-Order Value-Passing

Takis Hartonas Matthew Hennessy*
{hartonas, matthewh}@cogs.susx.ac.uk
COGS
University of Sussex at Brighton
Falmer, Brighton BN1 9QH, UK

Abstract

We study an applied typed call-by-value λ-calculus which in addition to the usual types for higher-order functions contains an extra type called proc, for processes; the constructors for terms of this type are similar to those found in standard process calculi such as CCS.

We first give an operational semantics for this language in terms of a labelled transition system which is then used to give a behavioural preorder based on contexts; the expression **N** dominates **M** if in every appropriate context if **M** can produce a boolean value then so can **N**.

Based on standard domain constructors we define a model, a prime algebraic lattice, which is *fully abstract* with respect to this behaviour preorder; expressions are related in the model if and only if they are related behaviourally.

The proof method uses concepts which are of independent interest. It involves characterising the domain using

Processes communicate by exchanging values along a common channel and since the reception of values here is implemented, at least partially, using function application, it is therefore natural to interpret the sequential part of the language in a *call-by-value* fashion. Thus in function application, $(\lambda X.\mathbf{M})\mathbf{N}$ and in the output of values, $\alpha![\mathbf{N}]\mathbf{M}$, the expression $\mathbf{N}$ is first reduced to a *value*. For basic types values are predetermined while for functional types a natural choice is to take λ-abstractions. For the type proc there is no obvious choice for the set of values. For this reason there is one complication in the type system. A subset of types, called the *transmittable types*, is defined by limiting functional types to be of the form $\sigma \to \tau$, where σ is either a base type or, recursively, a transmittable type. This precludes application expressions $\mathbf{MN}$, where $\mathbf{N}$ is a process, and, since channels must have associated with them transmittable types, $\alpha![\mathbf{N}]\mathbf{M}$. Thus the transmission of processes is not allowed, which is natural. But abstractions over processes, values of type unit $\to$ proc, which are often called *scripts*, may be freely exchanged. In short the core Facile, described in [3] is an elegant and powerful combination of λ-calculus and CCS.

We will study a language, which for convenience we call *mini-Facile*, very similar to core Facile. The major omission is the local scoping mechanism for channel names. For reasons of definability we will also introduce two extra operators; both of these are very natural but they do not appear in the language studied in [3]. In Section 3 we give an operational semantics to this language based on labelled transition systems. In the definition there are three kinds of judgements between closed expressions:

- $\mathbf{M} \longrightarrow \mathbf{N}$.
  This generalises the call-by-value reduction relation of the λ-calculus. It incorporates β-reduction, the application of an abstraction to a value, and for expressions of type proc it represents communication, the exchange of values between processes.

- $\mathbf{P} \xrightarrow{\alpha?} \lambda X.\mathbf{Q}$ where both $\mathbf{P}, \mathbf{Q}$ are of type process.
  This represents the ability of the process $\mathbf{P}$ to input a value from the channel α; when received it will be fed as an argument to the abstraction $\lambda X.\mathbf{Q}$

- $\mathbf{P} \xrightarrow{\alpha!} [\mathbf{v}]\mathbf{Q}$ where both $\mathbf{P}$, $\mathbf{Q}$ are processes.
  This states that the process $\mathbf{P}$ is capable of outputing the value $\mathbf{v}$ along the channel α. If this output is accepted the residual $\mathbf{Q}$ is activated.

Based on this operational semantics we define a contextual preorder over arbitrary expressions in the spirit of [18, 2, 5]. The idea is to start with a basic set of *observation predicates* $\mathcal{O}$. Then we say two expressions of the same type are related, $\mathbf{M} \sqsubseteq_{\mathcal{O}} \mathbf{N}$, if for every appropriate context $C[\ ]$ whenever $C[\mathbf{M}]$ satisfies an observation so does $C[\mathbf{N}]$. We examine two possible sets of predicates. The first is the ability to produce a boolean value while the second is the ability of a process to produce a value on a channel.

are chosen while values of functional type $\sigma_1 \to \sigma_2$ are taken to be $[\mathbf{D}_\sigma$

expression, ζ is a formula representing a property and Γ is a set of assumptions about the free variables possibly occuring in $\mathbf{M}$. In fact, the distinction between values and computations is reflected in the logical language; the language is sorted on each type σ of transmittable expressions, introducing a distinction between properties of values and properties of computations, alongside the properties for processes.

lower Egli-Milner order ($F \leq_\ell$ iff $\forall u \in F \exists v \in \quad u \leq_p v$). Idl(Fin($P$)) is the ideal completion of P (recall that an *ideal* is a lower directed subset); we denote this construction by $\mathcal{P}_l(P)$

LEMMA **2.1** Let $(P, \leq_p)$ be any partial order with bottom. Then $\mathcal{P}_l(P)$ is a prime algebraic lattice with the embedding of P as primes and the embedding of Fin(P) as its compact elements. ∎

A particular instance of this construction is the powerdomain construction. Recall that a powerdomain functor $\mathcal{P}\_$ is a *free* functor delivering, for each dcpo $\mathbf{D}$, the free semilattice dcpo $\mathcal{P}\_(\mathbf{D})$ satisfying a pre-determined set of equations with respect to the semilattice operation $\uplus$ of *formal union*. As usual, $\{\!|\cdot|\!\} : \mathbf{D} \to \mathcal{P}\_(\mathbf{D})$ denotes the *formal singleton* (insertion of generators) continuous map. We use $\mathcal{P}_H(\mathbf{D})$ to denote the lower (Hoare) powerdomain of $\mathbf{D}$.

THEOREM **2.2** If $\mathbf{D}$ is an algebraic dcpo, then $\mathbf{D} \cong \mathit{Idl}(\mathcal{K}(\mathbf{D}))$ and $\mathcal{P}_H(\mathbf{D}) \cong$Idl(Fin($\mathcal{K}(\mathbf{D})$)).

PROOF: See for example [1], vol. 3. ∎

A continuous function over domains f is *linear* provided $f(d \vee d') = fd \vee fd'$. If f has more than one argument, (for example $f : \mathbf{D} \times \mathbf{D}' \to \mathbf{E}$) then we say it is *multilinear* if it is linear in each of its arguments. We will often use the following fact.

THEOREM **2.3** Let $\mathbf{D}$, $\mathbf{E}$ be domains and $f : \mathbf{D} \to \mathbf{E}$ a function. Then f is continuous iff it is determined by its effect on the compact elements of $\mathbf{D}$ and linear iff it is determined by its effect on the prime elements of $\mathbf{D}$. ∎

A similar result holds for functions with more than one argument. By means of a tensor product construction, which we will use in our domain equation, multilinear functions can be dispensed with, in favor of linear functions.

PROPOSITION **2.4** Let $\mathcal{C}$ be the category of domains (prime algebraic lattices) with linear morphisms. If $\langle \mathbf{D} \times \mathbf{E}, \uparrow \rangle$ is the category of (bi)linear maps with domain $\mathbf{D} \times \mathbf{E}$, then there exists a linear morphism $\mathbf{lin}_{D,E}$ that is initial in $\langle \mathbf{D} \times \mathbf{E}, \uparrow \rangle$.

PROOF: The codomain of $\mathbf{lin}_{D,E}$ will be written as $\mathbf{D} \otimes \mathbf{E}$ (the tensor product of $\mathbf{D}$ and $\mathbf{E}$). Initialityfact.

LEMMA **2.5** For each pair $\mathbf{D}$, $\mathbf{E}$ of prime algebraic lattices, there is an initial left-strict bilinear morphism on $\mathbf{D} \times \mathbf{E}$. ∎

The codomain of this initial map will be denoted by $\mathbf{D}^s \otimes \mathbf{E}$. Because of universality of the tensor product $\otimes$ of Proposition 2.4 there exists a linear morphism $\mathbf{left}_\perp : \mathbf{D} \otimes \mathbf{E} \to \mathbf{D}^s \otimes \mathbf{E}$ such that $\mathbf{left}_\perp(\perp \otimes d) = \perp \in \mathbf{D}^s \otimes \mathbf{E}$. The initial morphism from the cartesian product into $\mathbf{D}^s \otimes \mathbf{E}$ must then be the composition $\mathbf{left}_\perp \circ \mathbf{lin}_{D,E}$. Since we will only have use for the left-strict version of the tensor product we hereafter use $\mathbf{A} \otimes \mathbf{B}$ to denote the *left-strict* tensor product of $\mathbf{A}$ and $\mathbf{B}$.

Our domain equation associates a domain $\mathbf{D}_\sigma$ of *values* to each transmittable type σ while the domains of computations of type σ take the form $T(\mathbf{D}_\sigma)$ where T is the composite monad $T = L\mathcal{P}_H(\ )$ obtained from the Hoare powerdomain and the lifting monad. The unit η of the monad is the map $\eta = \{\!|(\ )_\perp|\!\}$ and the multiplication μ is the obvious map $\mu : T^2 \to T$, dropping the outermost pair of formal set-braces and outermost lifting. For an algebraic dcpo $\mathbf{A}$, $T(\mathbf{A})$ will always be a prime algebraic lattice, with join operation supplied by the formal union map $\uplus$. Thus values of type $\sigma' \to \sigma$ will be interpreted in the space $[\mathbf{D}_{\sigma'} \to T$

Type Inference System The type system for the programming language is given by the grammar

$$\begin{array}{rcl} \sigma_{_G} \in \mathsf{GType} & ::= & \mathsf{unit} \mid \mathsf{bool} \mid \mathsf{int} \\ \sigma \in \mathsf{VType} & ::= & \sigma_{_G} \mid \sigma \to \tau \\ \tau \in \mathsf{Type} & ::= & \sigma \mid \mathsf{p} \end{array}$$

There is a special type for processes, proc, and a separate set of types for *transmittable values*, objects which can be sent and received by processes. These are either *base types* or abstractions over transmittable value types. Thus, for example, functions may be exchanged between processes. We do not allow terms of type proc to be exchanged in this manner, but, since unit $\to$ proc is a transmittable value type, objects which may be construed as *delayed* processes may be exchanged.

The language, which for convenience we call *mini-Facile*, is given by the grammar below, where X ranges over a set Var of variables and α is a communication channel name from a set $\mathcal{N}$; we assume these channel names have a unique transmittable value-type associated to them and write $\alpha \in \mathcal{N}^{\sigma}$ or sometimes treat σ as a function and write $\sigma(\alpha)$ for the value-type of α.

$$\begin{array}{rcl} \ell \in \mathsf{Lit(eral)} & ::= & \mathbf{tt} \mid \mathbf{ff} \mid \mathbf{n}\,(n \in \mathrm{N}) \mid () \\ \mathbf{v} \in \mathsf{Val(ue)} & ::= & \ell \mid \mu X \lambda Y.\mathbf{M} \mid \\ \mathbf{M} \in \mathsf{Exp(ression)} & ::= & \mathbf{v} \mid \mathbf{M} = \mathbf{N} \mid X \mid \mathbf{M}\mathbf{M} \mid \\ & & \mathsf{if}\ \mathbf{M}\ \mathsf{then}\ \mathbf{M}\ \mathsf{else}\ \mathbf{M} \mid \mathbf{M} \oplus \mathbf{M} \mid \mathbf{M} + \mathbf{M} \\ & & \mathbf{nil} \mid \mathbf{M}_{_{\mathcal{A}}}|_{_{\mathcal{B}}}\mathbf{M} \mid \mathsf{res}_{\alpha}\mathbf{M} \mid \\ & & \alpha?\mathbf{M} \mid \alpha![\mathbf{M}]\mathbf{M} \end{array}$$

The language may be viewed as an applied call-by-value λ-calculus, with a recursion operator. We abbreviate $\mu X \lambda Y.\mathbf{M}$ as $\lambda Y.\mathbf{M}$ when

Table 1: Type Inference System

$\triangleright \mathbf{tt} : \mathsf{bool}$ $\quad$ $\triangleright \mathbf{ff} : \mathsf{bool}$ $\quad$ $\triangleright \mathbf{n} : \mathsf{int}$

$\triangleright () : \mathsf{unit}$ $\quad$ $\triangleright \mathbf{nil} : \mathsf{proc}$ $\quad$ $X : \sigma \triangleright X : \sigma$

$$\frac{H \triangleright \mathbf{M} : \tau}{H, X : \sigma \triangleright \mathbf{M} : \tau} \ (W) \qquad \frac{H, X : \sigma \to \tau, Y : \sigma \triangleright \mathbf{M} : \tau}{H \triangleright \mu X \lambda Y . \mathbf{M} : \sigma \to \tau} \ (\mu)$$

$$\frac{H \triangleright \mathbf{M} : \sigma_{_G} \quad H \triangleright \mathbf{N} : \sigma_{_G}}{H \triangleright \mathbf{M} = \mathbf{N} : \mathsf{bool}} \ (eq)$$

$$\frac{H \triangleright \mathbf{M} : \sigma \to \tau \quad H \triangleright \mathbf{N} : \sigma}{H \triangleright \mathbf{M}\mathbf{N} : \tau} \ (App)$$

$$\frac{H \triangleright \mathbf{M} : \tau \quad H \triangleright \mathbf{N} : \tau}{H \triangleright \mathbf{M} \oplus \mathbf{N} : \tau} \ (IC) \qquad \frac{H \triangleright \mathbf{P} : \mathsf{proc} \quad H \triangleright \mathbf{Q} : \mathsf{proc}}{H \triangleright \mathbf{P} + \mathbf{Q} : \mathsf{proc}} \ (EC)$$

$$\frac{H \triangleright \mathbf{M} : \mathsf{proc} \quad H \triangleright \mathbf{N} : \mathsf{proc}}{H \triangleright \mathbf{M}_{|_{\mathcal{A}}|_{\mathcal{B}}} \mathbf{N} : \mathsf{proc}} \ (P) \qquad \frac{H \triangleright \mathbf{B} : \mathsf{bool} \quad H \triangleright \mathbf{M} : \tau \quad H \triangleright \mathbf{N} : \tau}{H \triangleright \mathsf{if}\ \mathbf{B}\ \mathsf{then}\ \mathbf{M}\ \mathsf{else}\ \mathbf{N} : \tau} \ (Cond)$$

$$\frac{H \triangleright \mathbf{M} : \sigma \quad H \triangleright \mathbf{N} : \mathsf{proc}}{H \triangleright \alpha ![\mathbf{M}]\mathbf{N} : \mathsf{proc}} \ (\alpha \in \mathcal{N}^\sigma) \ (\alpha!)$$

$$\frac{H \triangleright \mathbf{M} : \sigma \to \mathsf{proc}}{H \triangleright \alpha ?\mathbf{M} : \mathsf{proc}} \ (\alpha \in \mathcal{N}^\sigma) \ (\alpha?) \qquad H \triangleright \mathbf{M} : \mathsf{proc}$$

The two relatively non-standard operators we use are

- the parameterised form of communication $\mathbf{P}\,{}_{\mathcal{A}}|_{\mathcal{B}}\,\mathbf{Q}$, which restricts possible moves of the construct $\mathbf{P}\,{}_{\mathcal{A}}|_{\mathcal{B}}\,\mathbf{Q}$ to those in $\mathcal{A}$ when the action is due to $\mathbf{P}$, or in $\mathcal{B}$ when the action is due to $\mathbf{Q}$, while allowing unrestricted communication between $\mathbf{P}$ and $\mathbf{Q}$. This operator is imported from [12] where it was originally introduced.

- the result function: The expression $\mathsf{res}_{\alpha}(\mathbf{P})$ has the same type as the channel α; it allows the process P to compute until a value v can be produced on α and this value is then returned as the value of the expression. The effect of this operator is to recycle back into the functional fragment of the language values that have been output by processes. It is similar in spirit to the special action $\xrightarrow{\checkmark_v}$ of (Ferreira et al. [8]).

We use both of these operators in our definability theorem, which states that all compact elements in the denotational model are definable in the language.

The result function can be used to implement the internal choice operator between arbitrary expressions of n

Table 2: Operational Semantics

AXIOMS

$\mathbf{M} \oplus \mathbf{N} \longrightarrow \mathbf{M}, \quad \mathbf{M} \oplus \mathbf{N} \longrightarrow \mathbf{N}$

$\alpha?(\mu X\lambda Y.\mathbf{P}) \xrightarrow{\alpha?} \mu X\lambda Y.\mathbf{P}$

$\alpha![\mathbf{v}]\mathbf{P} \xrightarrow{\alpha!} [\mathbf{v}]\mathbf{P}$

$(\mu X\lambda Y.\mathbf{M})\mathbf{v} \longrightarrow \mathbf{M}[(\mu X\lambda Y.\mathbf{M})/X][\mathbf{v}/Y]$

$\mathsf{if}\ \mathtt{tt}\ \mathsf{then}\ \mathbf{M}\ \mathsf{else}\ \mathbf{N} \longrightarrow \mathbf{M}$

$\mathsf{if}\ \mathtt{ff}\ \mathsf{then}\ \mathbf{M}\ \mathsf{else}\ \mathbf{N} \longrightarrow \mathbf{N}$

$\ell = \ell \longrightarrow \mathtt{tt}$

RULES

$$\frac{\mathbf{P} \longrightarrow \mathbf{P}'}{\mathbf{P}+\mathbf{Q} \longrightarrow \mathbf{P}'+\mathbf{Q}} \qquad \frac{\mathbf{Q} \longrightarrow \mathbf{Q}'}{\mathbf{P}+\mathbf{Q} \longrightarrow \mathbf{P}+\mathbf{Q}'} \qquad \frac{\mathbf{P} \xrightarrow{\alpha} \mathbf{M}}{\mathbf{P}+\mathbf{Q} \xrightarrow{\alpha} \mathbf{M}}$$

$$\frac{\mathbf{Q} \xrightarrow{\alpha} \mathbf{M}}{\mathbf{P}+\mathbf{Q} \xrightarrow{\alpha} \mathbf{M}} \qquad \frac{\mathbf{P} \longrightarrow \mathbf{P}'}{\mathbf{P}_{\mathcal{A}}|_{\mathcal{B}}\mathbf{Q} \longrightarrow \mathbf{P}'_{\mathcal{A}}|_{\mathcal{B}}\mathbf{Q}} \qquad \frac{\mathbf{Q} \longrightarrow \mathbf{Q}'}{\mathbf{P}_{\mathcal{A}}|_{\mathcal{B}}\mathbf{Q} \longrightarrow \mathbf{P}_{\mathcal{A}}|_{\mathcal{B}}\mathbf{Q}'}$$

$$\frac{\mathbf{M} \longrightarrow \mathbf{M}' \quad \mathbf{N} \longrightarrow \mathbf{N}'}{\mathbf{M}=\mathbf{N} \longrightarrow \mathbf{M}'=\mathbf{N}'} \qquad \frac{\mathbf{M} \longrightarrow \mathbf{M}'}{\mathbf{M}\mathbf{N} \longrightarrow \mathbf{M}'\mathbf{N}} \qquad \frac{\mathbf{M} \longrightarrow \mathbf{M}'}{\mathrm{v}\mathbf{M} \longrightarrow \mathrm{v}\mathbf{M}'}$$

$$\frac{\mathbf{M} \longrightarrow \mathbf{N}}{\alpha?\mathbf{M} \longrightarrow \alpha?\mathbf{N}}\ \text{И}$$

Definition **3.1 (May-Testing Preorder)**
The may-testing preorder $\sqsubseteq_{\approx}$

4 Denotational Semantics

We determine the model, the collection of value domains, via a domain equation with "coefficients" in the category of bounded-complete algebraic dcpo's, to be solved in the types proc and $\sigma \to \tau$ in the subcategory of prime algebraic lattices. The coefficients are the constants of the equation, the ground value domains for integers, booleans and for the unit type which we choose to be the discrete cpo's $\mathbb{IN}$, $\mathbb{IB}$ and $\mathbf{U}$. The constructors used in the statement of the equation are the continuous function space, cartesian product, the Hoare powerdomain and lifting monads $\mathcal{P}_H(\cdot)$ and L and the tensor product functor discussed in Section 2.

The model is determined as the initial solution to the domain equation given in Figure 1.

Figure 1: A Domain Equation

$$
\begin{array}{lll}
[\![\mathsf{int}]\!] = & \mathbf{D}_{\mathsf{int}} & = \mathbb{N} \\
[\![\mathsf{bool}]\!] = & \mathbf{D}_{\mathsf{bool}} & = \mathbb{B} \\
[\![\mathsf{unit}]\!] = & \mathbf{D}_{\mathsf{unit}} & = \mathrm{U} \\
[\![\sigma \to \sigma']\!] = & \mathbf{D}_{\sigma \to \sigma'} & = [\mathbf{D}_\sigma \to T(\mathbf{D}_{\sigma'})] \\
[\![\sigma \to \mathsf{proc}]\!] = & \mathbf{D}_{\sigma \to \mathsf{proc}} & = [\mathbf{D}_\sigma \to L(\mathbf{D}_{\mathsf{proc}})] \\
 & \mathbf{C}^{in} &
\end{array}
$$

existence of the appropriate embedding maps $\theta^\alpha : F^\alpha(\mathbf{D}) \to \mathbf{E}$. Then the map $\theta : F(\mathbf{D}) \to \mathbf{E}$ defined by $\theta = \prod_{\alpha \in \mathcal{N}} \theta^\alpha : F(\mathbf{D}) \to \mathbf{E}$ can be verified to be such that $\theta \circ F(\rho_m) = \xi_m$ for each $m \in \omega$. This shows that F is continuous.

If T is the composite monad $T(X) = \mathcal{P}_{_H}(X)_\perp \cong \mathcal{P}_{_H}(X_\perp)$, then T is a continuous functor sending a bc algebraic dcpo to a prime algebraic lattice whose primes are of the form $\perp, \{\!|c|\!\}_\perp$ (or, equivalently, $\{\!|\perp|\!\}, \{\!|c_\perp|\!\}$) with c compact.

If $\mathbf{D}_\alpha$, with α in a countable index set $\mathcal{N}$, are prime algebraic lattices and we set $F^\alpha(X) = T(\mathbf{D}_\alpha) \otimes^{s} X$ and $\quad^\alpha(X) = [\mathbf{D}_\alpha \to X_\perp]$ then each of F^α, $\quad^\alpha$ is continuous and the category of prime algebraic lattices is closed under F^α and $\quad^\alpha$. Hence by the previous discussion it is also closed under the construction $H = \prod_{\alpha \in \mathcal{N}} (F^\alpha \times \quad^\alpha)$ and H is a continuous functor, by the argument given above, since both F^α and $\quad^\alpha$ are continuous functors.

Existence of an initial

In the next definition we introduce a notation for describing the prime elements of the domains in a way that supports later proofs by induction on primes.

In the ground value domains compact and prime elements coincide and in fact every element is a compact-prime since the order is discrete. We now define sets $A^\tau_{\mathcal{KP}}, A^\tau_{\mathcal{K}}$ by induction on the type τ.

DEFINITION **4.3** The sets $A^\tau_{\mathcal{KP}}, A^\tau_{\mathcal{K}}$ are the least sets such that

1. $A^{\mathsf{int}}_{\mathcal{KP}} = \mathbb{N} = A^{\mathsf{int}}_{\mathcal{K}}$ and similarly for the types $\mathsf{bool}, \mathsf{unit}$
2. For any type τ other than the ground types $\mathsf{int}, \mathsf{bool}, \mathsf{unit}$, the sets $A^\tau_{\mathcal{K}} \supseteq A^\tau_{\mathcal{KP}}$ consist of finite formal joins $c = p_1 \vee \cdots \vee p_s$ with $s \geq 1$ and $p_i \in A^\tau_{\mathcal{KP}}$
3. $A^{\sigma\to\sigma'}_{\mathcal{KP}}$ consists of a bottom element $\bot$ and formal step functions $c \to k$ where $c \in A^\sigma_{\mathcal{K}}$ and $k \in A^{\sigma'}_{\mathcal{K}}$
4. $A^{\sigma\to\mathsf{proc}}_{\mathcal{KP}}$ consists of a bottom element $\bot$ and formal step functions $c \to \pi$ where $c \in A^\sigma_{\mathcal{K}}$ and $\pi \in A^{\mathsf{proc}}_{\mathcal{KP}}$
5. $A^{\mathsf{proc}}_{\mathcal{KP}}$ consists of formal elements of the form
   - $\bot_{\mathsf{proc}}$
   - $\alpha_{out}(c \otimes \pi)$ where $c \in A^{\sigma(\alpha)}_{\mathcal{K}}$ and $\pi \in A^{\mathsf{proc}}_{\mathcal{KP}}$
   - $\alpha_{in}(c \to \pi)$ where $c \in A^{\sigma(\alpha)}_{\mathcal{K}}$ and $\pi \in A^{\mathsf{proc}}_{\mathcal{KP}}$. ∎

Next we define an ordering of the sets $A^\tau_{\mathcal{K}}$. Note that in the representation of prime elements given above $c \to \bot$ is to be distinguished from the bottom element of the function space since it really represents the element $c \to \{\!|\bot_\bot|\!\}$. Similarly, in $\alpha_{out}(\bot \otimes \pi)$ the left-strict product should not reduce the concretion $\bot \otimes \pi$ to a bottom element since this element really represents the prime $\alpha_{out}(\{\!|\bot_\bot|\!\} \otimes \pi)$. Similarly for $\alpha_{in}(c \to \pi)$. This explains why we introduced a fresh $\bot$ element for each of the higher types and for proc.

DEFINITION **4.4** The relations $\leq_\tau$ are the least reflexive, transitive and antisymmetric (i.e. partial order) relations on $A^\tau_{\mathcal{K}}$ such that

1. $\leq_{\mathsf{int}}, \leq_{\mathsf{bool}}$ and $\leq_{\mathsf{unit}}$ are the identity relations
2. they satisfy axioms/rules such that all mentioned formal joins become least upper bound operators in the relevant ordering and all mentioned $\bot$ elements become the least elements in the ordering
3. $c \to k \leq_{\sigma\to\tau} c' \to k'$ iff $c' \leq_\sigma c$ and $k \leq_\tau k'$
4. $\alpha_{out}(c \otimes \pi) \leq_{\mathsf{proc}} \alpha_{out}(c' \otimes \pi')$ provided $c \leq_{\sigma(\alpha)} c'$ and $\pi \leq_{\mathsf{proc}} \pi'$
5. $c \to \pi \leq_{\sigma\to\mathsf{proc}} c' \to \pi'$ iff $c' \leq_\sigma c$ and $\pi \leq_{\mathsf{proc}} \pi'$
6. $\alpha_{in}(c \to \pi) \leq \alpha_{in}(c' \to \pi')$ iff $c \to \pi \leq_{\sigma(\alpha)\to\mathsf{proc}} c' \to \pi'$

Since algebraic dcpo's are characterized by their bases of compact elements we have the following

THEOREM **4.5** Let $\mathbf{D}'_\tau = Idl(A^\tau_\mathcal{K})$ be defined as the ideal completions of the partial orders $(A^\tau_\mathcal{K}, \leq_\tau)$. Then $\mathbf{D}'_\tau \cong \mathbf{D}_\tau$.

PROOF: The proof follows by calculating the actual primes of the domains $\mathbf{D}_\tau$ from the bilimit construction of the initial solution to the domain equation. For example, the primes of the function space $[\mathbf{D}_\sigma \to T(\mathbf{D}_{\sigma'})]$ are the step functions $c \to P$ with $c \in \mathcal{K}(\mathbf{D}_\sigma)$ and P a prime in $T(\mathbf{D}_{\sigma'})$, i.e. an element of the form $\{\!|\bot|\!\}$ or $\{\!|k_\bot|\!\}$ with $k \in \mathcal{K}(\mathbf{D}_{\sigma'})$. Similarly, the actual primes in the domain of processes are infinite sequences of pairs with $(\bot, \bot)$ everywhere, in the case of the bottom element, or except for at most one position, in the case of nontrivial primes, where a prime of one of the forms $(\{\!|c_\bot|\!\} \otimes \pi, \bot)$ or $(\bot, c \to \pi_\bot)$ occurs. ∎

Interpretation: Let $\mathbf{Exp}^\mathcal{H}_\tau$ be the set of terms in context $H \rhd \mathbf{M} : \tau$. The interpretation function, presented in Table 3, is a partial map (defined on $H \rhd \mathbf{M} : \tau$ provided it is derivable in the type system) and it follows the standard pattern. For convenience, we define an interpretation function $\mathcal{V}[\![\cdot]\!]$ assigning to a *value* expression $\mathbf{v}$ a continuous function into the appropriate value domain $\mathbf{D}_\sigma$. Thus $[\![H \rhd \mathbf{v} : \sigma]\!]$ is always obtained by composition with the unit η of the monad T: $[\![H \rhd \mathbf{v} : \sigma]\!] = \eta \circ \mathcal{V}[\![H \rhd \mathbf{v} : \sigma]\!]$.

Table 3: Interpretation Function

$\mathcal{V}[\![H \rhd \mathbf{n} : \mathsf{int}]\!]$	=	$\lambda x \in [\![H]\!].\, n$ (Similarly for other litterals)
$\mathcal{V}[\![H \rhd \mu X \lambda Y.\mathbf{M} : \sigma \to \tau]\!]$	=	$\mathrm{Y} \circ \mathbf{curry}(\mathbf{curry}([\![H, X : \sigma \to \tau, Y : \sigma \rhd \mathbf{M} : \tau]\!]))$
$[\![H \rhd \mathbf{v} : \sigma]\!]$	=	$\eta \circ \mathcal{V}[\![H \rhd \mathbf{v} : \sigma]\!]$
$[\![H \rhd \mathbf{FM} : \tau]\!]$	=	$\mathbf{apply}^T([\![H \rhd \mathbf{F} : \sigma \to \tau]\!],\ [\![H \rhd \mathbf{M} : \sigma]\!])$
$[\![H \rhd \mathbf{M} = \mathbf{N} : \mathsf{bool}]\!]$	=	$\mathrm{EQ} \circ ([\![H \rhd \mathbf{M} : \sigma_G]\!],\ [\![H \rhd \mathbf{N} : \sigma_G]\!])$
$[\![H \rhd \mathsf{if}\ \mathbf{B}\ \mathsf{then}\ \mathbf{M}\ \mathsf{else}\ \mathbf{N} : \tau]\!]$	=	$\mathrm{COND}_\tau \circ ([\![H \rhd \mathbf{B} : \mathsf{bool}]\!],\ [\![H \rhd \mathbf{M} : \tau]\!],\ [\![H \rhd \mathbf{N} : \tau]\!]$
$[\![H \rhd \mathbf{M} \oplus \mathbf{N} : \tau]\!]$	=	$[\![H \rhd \mathbf{M} : \tau]\!] \vee [\![H \rhd \mathbf{N} : \tau]\!]$
$[\![H \rhd \mathbf{P} \oplus \mathbf{Q} : \mathsf{proc}]\!]$	=	$[\![H \rhd \mathbf{P} : \mathsf{proc}]\!] \vee [\![H \rhd \mathbf{Q} : \mathsf{proc}]\!]$
$[\![H \rhd \mathbf{P} + \mathbf{Q} : \mathsf{proc}]\!]$	=	$[\![H \rhd \mathbf{P} : \mathsf{proc}]\!] \vee [\![H \rhd \mathbf{Q} : \mathsf{proc}]\!]$
$[\![H \rhd \mathbf{nil} : \mathsf{proc}]\!]$	=	$\lambda x.\, \bot$
$[\![H \rhd \mathsf{res}_\alpha(\mathbf{P}) : \sigma(\alpha)]\!]$	=	$\mathsf{res}_\alpha \circ ([\![H \rhd \mathbf{P} : \mathsf{proc}]\!]$
$[\![H \rhd \mathbf{P}_\mathcal{A}\|_\mathcal{B}\mathbf{Q} : \mathsf{proc}]\!]$	=	$\mathrm{PAR}_{\mathcal{A},\mathcal{B}} \circ ([\![H \rhd \mathbf{P} : \mathsf{proc}]\!],\ [\![H \rhd \mathbf{Q} : \mathsf{proc}]\!])$
$[\![H \rhd \alpha![\mathrm{M}]\mathbf{Q} : \mathsf{proc}]\!]$	=	$\alpha_{out} \circ ([\![H \rhd \mathbf{M} : \sigma(\alpha)]\!] \otimes [\![H \rhd \mathbf{Q} : \mathsf{proc}]\!])$
$[\![H \rhd \alpha?\mathbf{N} : \mathsf{proc}]\!]$	=	$\alpha^T_{in} \circ ([\![H \rhd \mathbf{N} : \sigma(\alpha) \to \mathsf{proc}]\!])$

The map $\mathbf{apply}^T$ is defined as $\mathbf{ext}^{(2)}(\mathbf{apply})$ where $\mathbf{ext}^{(2)}_{A,B,C} : (\mathbf{A} \times \mathbf{B} \to T\mathbf{C}) \longrightarrow (T\mathbf{A} \times T\mathbf{B} \to T\mathbf{C})$ is the standard extension map of the monad T. Essentially the same definition

can be given for the case of functions into proc and we use $\mathbf{apply}^T$ for both cases, as discussed in Section 2. EQ and COND_τ are the natural equality and conditional morphisms. The map $\mathsf{res}_\alpha : \mathbf{D}_{\mathsf{proc}} \longrightarrow T(\mathbf{D}_{\sigma(\alpha)})$ is defined on primes (and then linearly extended to all elements) in the obvious way: $\mathsf{res}_\alpha(\pi) = \{\!|c_\perp|\!\}$ if $\pi = \alpha_{out}(c \otimes \pi')$ and $\perp$ otherwise.

The interpretation $\mathrm{PAR}_{\mathcal{A},\mathcal{B}}$ of the parallel operator is somewhat more complex. Its definition on primes is given in Table 4 where $\pi_{\mathcal{A}}|_{\mathcal{B}}\pi'$ is the join of elements of the form listed in the third column and where each such element appears in the join provided the related condition in column four holds. Table 4 has been compiled after a similar table in Hennessy [12]. The choice operators $\oplus$ and $+$ are interpreted as joins of functions, where the join $f \vee g$ is determined by the join operation in the common codomain of f and g. The maps α_{out} and α_{in}, previously defined, are used to interpret input and output on α. For input processes we use the natural linear strict extension of the map $\alpha_{in} : [\mathbf{D}_{\sigma(\alpha)} \to L(\mathbf{D}_{\mathsf{proc}})] \longrightarrow \mathbf{D}_{\mathsf{proc}}$ to a map $\alpha_{in}^T : T(\mathbf{D}_{\sigma(\alpha)} \to L(\mathbf{D}$

PROPOSITION **4.8**

1. (**Call-by-value β-Reduction**)

One direction of this result is straightforward, as it simply says that the operational semantics is reflected correctly in the model. However to prove the converse we need a typed modal language $\mathcal{L}$ of properties ζ of program terms. This is the subject of Section 5. The program logic we develop is of independent interest and we aim to prove that it is complete in its natural operational semantics, detailed in Section 5.

From the adequacy result one direction of the full-abstraction follows without much difficulty. For the converse we need a definability result, essentially saying that all compact elements in the domains $\mathbf{D}_\tau$ are denotable by some closed expression from *mini-Facile*. This is the subject of Section 6. Using definability we can also derive completeness of the program logic in the operational semantics. The proof of full-abstraction is then given in Section 7.

5 Program Logic

Let $\mathcal{T}, \mathcal{V}, \mathcal{P}$ be the sets of closed expressions, value expressions and process expressions. The operational semantics determines a many-sorted transition system $\langle \mathcal{T}, \mathcal{V}, \mathcal{P}, \Downarrow, \stackrel{\alpha!}{\Longrightarrow}, \stackrel{\alpha?}{\Longrightarrow} \rangle$ for which we seek to provide a natural logical language of properties and associated proof systems.

In short, the logical language is generated by the grammar below on the signature of logical operators $\{\&, \sqcap, \rightarrow, \Diamond, \langle\alpha!\rangle[\,], \langle\alpha?\rangle\}$ where only $\Diamond$ and $\langle\alpha?\rangle$ are unary while every other connective is binary:

$$\begin{array}{rcl} S \in \mathbf{AtFmla} & := & S_n\ (n \in N) \mid S_{\mathbf{tt}} \mid S_{\mathbf{ff}} \mid S_{()} \mid \omega_{\mathsf{proc}} \mid \omega^L_\sigma \mid \omega^T_\sigma \mid \\ \zeta \in \mathbf{Fmla} & := & S \mid \zeta\&\zeta \mid \zeta \sqcap \zeta \mid \zeta \rightarrow \zeta \mid \Diamond\zeta \mid \langle\alpha!\rangle[\zeta]\zeta \mid \langle\alpha?\rangle\zeta \end{array}$$

We will sort the language separating properties of values, for each trype $\sigma \in \mathsf{VType}$, from such of computations and processes. For mnemonic reasons we use $A, B, C, \ldots$ for properties of values, $\varphi, \psi, \chi, \ldots$ for properties of processes and $\zeta, \eta, \vartheta, \ldots$ for properties of arbitrary terms.

REMARK **5.1** Nontrivial properties of computations of some type σ are always either of the form $\Diamond A$ or conjunctions $\zeta \sqcap \eta$, where at least one of ζ, η is a nontrivial property. Distinct conjunction operations for properties of values as opposed to such of computations are needed. This becomes clear when attempting to find a unique conjunction rule for the program logic that will be sound in the denotational semantics. The join in the domain of values is to be logically distinguished from the formal union operation in the domain of computations. Thus different connectives and associated rules are needed so that we can have the logical analogues of situations like $\{\!|c \vee k|\!\} \leq d$ and $\{\!|c, k|\!\} = \{\!|c|\!\} \uplus \{\!|k|\!\} \leq d$. ∎

DEFINITION **5.2** The languages $\mathcal{L}_\sigma(\mathcal{V})$, $\sigma \in \mathsf{VType}, \mathcal{L}_\tau(\mathcal{T})$, $\tau \in \mathsf{Type}$ are the least sets satisfying the recursive conditions in Table 5. ∎

Two sorts of semantics relations, indexed in types, $\mathrel{\mathbb{R}}_\sigma$ and $\models_\tau$ will be defined. The relation $\mathrel{\mathbb{R}}_\sigma$ is a binary relation from values of some type σ to sentences in $\mathcal{L}_\sigma(\mathcal{V})$ while $\models_\tau$ relates arbitrary expressions of type τ to sentences in $\mathcal{L}_\tau(\mathcal{T})$, subject to the mutual recursive clauses of Table 6.

LEMMA **5.3** By definition, $\mathbf{M} \Longrightarrow \mathbf{v}$ and $\mathbf{v} \mathrel{\mathbb{R}}_\sigma A$ implies $\mathbf{M} \models_\sigma \Diamond A$. Furthermore, $\mathbf{M} \Longrightarrow \mathbf{N}$ and $\mathbf{N} \models_\tau \zeta \in \mathcal{L}_\tau(\mathcal{T})$ implies $\mathbf{M} \models_\tau \zeta$.

PROOF: By structural induction on the sentence $\zeta \in \mathcal{L}_\tau(\mathcal{T})$. ∎

Table 7: A Proof System $\mathcal{G}_E$ for Semantic Entailment

(Id)	$A \mid\!\sim_\sigma A$	(T1)	$\zeta \vdash_\sigma \omega_\sigma^T$
(T2)	$A \mid\!\sim_{\sigma' \to \sigma} B \to \omega_\sigma^T$	(T3)	$A \mid\!\sim_{\sigma \to \mathsf{proc}} \omega_\sigma^L$
(Cut1)	$\dfrac{A \mid\!\sim_\sigma B \quad B \mid\!\sim_\sigma C}{A \mid\!\sim_\sigma C}$	(Cut2)	$\dfrac{\zeta \vdash_\tau \eta \quad \eta \vdash_\tau \vartheta}{\zeta \vdash_\tau \vartheta}$
($\to$)	$\dfrac{B \mid\!\sim_\sigma A \quad \zeta \vdash_\tau \eta}{A \to \zeta \mid\!\sim_{\sigma \to \tau} B \to \eta}$	(&R)	$\dfrac{A \mid\!\sim_\sigma B \quad A \mid\!\sim_\sigma C}{A \mid\!\sim_\sigma B \& C}$
(&L1)	$\dfrac{A \mid\!\sim_\sigma C}{A \& B \mid\!\sim_\sigma C}$	(&L2)	$\dfrac{B \mid\!\sim_\sigma C}{A \& B \mid\!\sim_\sigma C}$
($\Diamond$)	$\dfrac{A \mid\!\sim_\sigma B}{\Diamond A \vdash_\sigma \Diamond B}$	(T2)	$\varphi \vdash_{\mathsf{proc}} \omega_{\mathsf{proc}}$
($\alpha!$)	$\dfrac{A \mid\!\sim_{\sigma(\alpha)} B \quad \varphi \vdash_{\mathsf{proc}} \psi}{\langle\!\langle \alpha! \rangle\!\rangle [A] \varphi \vdash_{\mathsf{proc}} \langle\!\langle \alpha! \rangle\!\rangle [B] \psi}$	($\alpha?$)	$\dfrac{B \mid\!\sim_{\sigma(\alpha)} A \quad \varphi \vdash_{\mathsf{proc}} \psi}{\langle\!\langle \alpha? \rangle\!\rangle A \to \varphi \vdash_{\sigma(\alpha) \to \mathsf{proc}} \langle\!\langle \alpha? \rangle\!\rangle B \to \psi}$
($\&_\varphi$L1)	$\dfrac{\varphi \vdash_{\mathsf{proc}} \chi}{\varphi \& \psi \vdash_{\mathsf{proc}} \chi}$	($\&_\varphi$L2)	$\dfrac{\psi \vdash_{\mathsf{proc}} \chi}{\varphi \& \psi \vdash_{\mathsf{proc}} \chi}$
($\&_\varphi$R)	$\dfrac{\varphi \vdash_{\mathsf{proc}} \psi \quad \varphi \vdash_{\mathsf{proc}} \chi}{\varphi \vdash_{\mathsf{proc}} \psi \& \chi}$	($\sqcap$R)	$\dfrac{\zeta \vdash_\sigma \eta \quad \zeta \vdash_\sigma \vartheta}{\zeta \vdash_\sigma \eta \sqcap \vartheta}$
($\sqcap$L1)	$\dfrac{\zeta \vdash_\sigma \vartheta}{\zeta \sqcap \eta \vdash_\sigma \vartheta}$	($\sqcap$L2)	$\dfrac{\eta \vdash_\sigma \vartheta}{\zeta \sqcap \eta \vdash_\sigma \vartheta}$

Axiomatizing Semantic Entailment: We may think of formulae extensionally, as the sets of terms that satisfy them. This induces notions of semantic entailment where for $A, B \in \mathcal{L}_\sigma(\mathcal{V})$ and $\zeta, \eta \in \mathcal{L}_\tau(\mathcal{T})$

- $A \mid\!\approx_\sigma B$ iff for all values $\mathbf{v} \in \mathcal{V}$, $\mathbf{v} \mid\!\approx_\sigma A$ implies $\mathbf{v} \mid\!\approx_\sigma B$
- $\zeta \models_\tau \eta$ iff for all $\mathbf{M}$, $\mathbf{M} \models_\tau \zeta$ implies $\mathbf{M} \models_\tau \eta$

Semantic entailment is axiomatized in a Gentzen-style implicational system, $\mathcal{G}_E$, in Table 7.

LEMMA **5.4** The system $\mathcal{G}_E$ is sound in the operational semantics. In other words, $A \mid\!\sim_\sigma B$ implies $A \mid\!\approx_\sigma B$, and $\zeta \vdash_\tau \eta$ implies $\zeta \models_\tau \eta$.

PROOF: By the usual induction on length of proof. ∎

To relate the logic with the model we first interpret sentences as compact elements. The interpretation maps $\mathcal{V}[\![\cdot]\!]$ and $[\![\cdot]\!]$, defined in Table 8 by mutual recursion, interpret properties of values as compact elements of the value domains $\mathbf{D}_\sigma$ and properties of computations and processes as compact elements of $T(\mathbf{D}_\sigma)$ and $\mathbf{D}_{\mathsf{proc}}$, respectively.

Table 8: Interpretation of Sentences as Elements of the Domain

- $\mathcal{V}[\![S_n]\!] = n \in \mathbb{N}$ and similarly for the other atomic sentences
- $\mathcal{V}[\![A\&B]\!] = \mathcal{V}[\![A]\!] \vee \mathcal{V}[\![B]\!] \qquad (\sigma \neq \sigma_G \in \mathsf{GType})$
- $\mathcal{V}[\![A \to \zeta]\!] = \mathcal{V}[\![A]\!] \to [\![\zeta]\!]$
- $\mathcal{V}[\![\omega^L_\sigma]\!] = \bot \in [\mathbf{D}_\sigma \to L(\mathbf{D}_{\mathsf{proc}})]$
- $[\![\omega^T_\sigma]\!] = \{\!|\bot|\!\} \in T(\mathbf{D}_\sigma)$
- $[\![\Diamond A]\!] = \{\!|\mathcal{V}[\![A]\!]_\bot|\!\}$
- $[\![\zeta \sqcap \eta]\!] = [\![\zeta]\!] \uplus [\![\eta]\!]$
- $[\![\omega_{\mathsf{proc}}]\!] = \bot \in \mathbf{D}_{\mathsf{proc}}$
- $[\![\varphi\&\psi]\!] = [\![\varphi]\!] \vee [\![\psi]\!]$
- $[\![\langle\alpha!\rangle[A]\psi]\!] = \alpha_{out}(\{\!|\mathcal{V}[\![A]\!]_\bot|\!\} \otimes [\![\psi]\!])$
- $[\![\langle\alpha?\rangle A \to \psi]\!] = \alpha_{in}(\mathcal{V}[\![A]\!] \to [\![\psi]\!]_\bot)$

Remark: In the sequel we will leave lifting implicit and write, e.g.
$\alpha_{in}(\mathcal{V}[\![A]\!] \to [\![\psi]\!])$, $\alpha_{out}(\{\!|\mathcal{V}[\![A]\!]|\!\} \otimes [\![\psi]\!])$ and $\{\!|\mathcal{V}[\![A]\!]|\!\}$

PROPOSITION **5.5**

1. The interpretation of a sentence ζ is a compact element of the respective domain, depending on the sorting and type of ζ. Conversely, for every compact element C of the domains $\mathbf{D}_\sigma, T(\mathbf{D}_\sigma)$ and $\mathbf{D}_{\mathsf{proc}}$ there is a sentence ζ of appropriate sort and type such that $C = [\![\zeta]\!]$ or $C = \mathcal{V}[\![\zeta]\!]$, as appropriate.
2. (**Completeness of the System $\mathcal{G}_E$ in the Denotational Semantics**)
   - $A \mathrel{|\!\sim}_\sigma B$ iff $\mathcal{V}[\![B]\!] \leq \mathcal{V}[\![A]\!]$ in $L(\mathbf{D}_\sigma)$
   - $\zeta \vdash_\sigma \eta$ iff $[\![\eta]\!] \leq [\![\zeta]\!]$ in $T(\mathbf{D}_\sigma)$, and
   - $\varphi \vdash_{\mathsf{proc}} \psi$ iff $[\![\psi]\!] \leq [\![\varphi]\!]$ in $\mathbf{D}_{\mathsf{proc}}$.

PROOF: For part 1, we only comment on the case of a sentence of the form $A \to \zeta \sqcap \eta$. The interpretation yields the element $\mathcal{V}[\![A]\!] \to [\![\zeta]\!] \uplus [\![\eta]\!]$ where the formal union is the join operation in $T(\mathbf{D}_{\sigma'})$. This function is identical to the compact element $(\mathcal{V}[\![A]\!] \to [\![\zeta]\!]) \vee (\mathcal{V}[\![A]\!] \to [\![\eta]\!])$ in the model, where the join $\vee$ is now taken in the function space $[\mathbf{D}_\sigma \to T(\mathbf{D}_{\sigma'})]$. Similarly for $A \to \varphi\&\psi$.

The structure of our modal language does not directly reflect the fact that the transition system is equiped with parallel operators ${}_{\mathcal{A}}|_{\mathcal{B}}$. Instead we interpret ${}_{\mathcal{A}}|_{\mathcal{B}}$ as an operator on formulae; $\varphi\,{}_{\mathcal{A}}|_{\mathcal{B}}\,\psi$ can be considered as a shorthand for a finite conjunction of formula in $\mathcal{L}_{\mathsf{proc}}(\mathcal{T})$. The components of this conjunction are defined by induction on the structure of φ and ψ,

Table 9: A Proof System $\mathcal{G}_S$ for Satisfiability: Program Logic

(Mon1) $\quad \Gamma \mathrel{|\!\sim_\sigma} \mathbf{v} : A \qquad A \mathrel{|\!\sim_\sigma} B$

For an environment ρ and an assumption Γ let $\rho \models^D \Gamma$ iff for any variable X, $\mathcal{V}[\![\Gamma(X)]\!] \leq \rho(X)$.

DEFINITION **5.10** Let $\Gamma \models^D_\tau \mathbf{M} : \zeta$ if and only if for every environment ρ, if $\rho \models^D \Gamma$, then $[\![\zeta]\!] \leq [\![\mathbf{M}]\!]_\rho$. ∎

Similarly, we may let $\Gamma \approx^D_\sigma \mathbf{v} : A$ iff for every environment ρ, if $\rho \models^D \Gamma$, then $\mathcal{V}[\![A]\!] \leq \mathcal{V}[\![\mathbf{v}]\!]$. Since the latter inequality is equivalent to $\{|\mathcal{V}[\![A]\!]|\} \leq \{|\mathcal{V}[\![\mathbf{v}]\!]|\}$, i.e. $[\![\Diamond A]\!] \leq [\![\mathbf{v}]\!]$, we may only use the relations $\models^D_\tau$.

We can show that the program logic is complete in the denotational semantics. A priori, there is no reason why completeness with respect to this technical notion of semantics would have any bearing on the completeness of the program logic in its natural, operational, semantics. Using our definability results, however, we will be in a position to show in Section 7 that the operational and denotational semantics for the program logic coincide.

THEOREM **5.11 (Soundness-Completeness of $\mathcal{G}_S$ in the Denotational Semantics)**

$$\Gamma \vdash_\tau \mathbf{M} : \zeta \text{ if and only if } \Gamma \models^D_\tau \mathbf{M} : \zeta$$

PROOF: Given an assumption Γ, define the environment ρ_Γ by $\rho(X) = [\![\Gamma(X)]\!]$. Then it is immediate that $\Gamma \models^D \mathbf{M} : \zeta$ iff $[\![\zeta]\!] \leq [\![\mathbf{M}]\!]_{\rho_\Gamma}$. So the soundness and completeness claim is equivalent to the claim $\Gamma \vdash_\tau \mathbf{M} : \zeta$ if and only if $[\![\zeta]\!] \leq [\![\mathbf{M}]\!]_{\rho_\Gamma}$, which we may alternatively use.

Soundness: The soundness part is proven by induction on length of proofs, and uses the soundness in the denotational semantics of the proof system $\mathcal{G}_E$ for semantic entailment, Proposition 5.5. It is mostly straightforward and we only do a few cases as an example.

For the monotonicity rule (Mon), if $\rho \models \Gamma$ then $\mathcal{V}[\![A]\!] \leq \mathcal{V}[\![\mathbf{v}]\!]_\rho$ and, by Proposition 5.5, $\mathcal{V}[\![B]\!] \leq \mathcal{V}[\![A]\!]$. Then $\mathcal{V}[\![B]\!] \leq \mathcal{V}[\![\mathbf{v}]\!]_\rho$.

Proof of soundness for the conjunction rules is straightforward but it illustrates the need for

Completeness: If ζ is any of ω_σ^T or ω_{proc} the claim is trivial using one of the (T) axioms. If ζ is of the form $\eta \sqcap \vartheta$ we may use the induction hypothesis on η, ϑ and then the conclusion follows by using the appropriate conjunction rule. The remaining cases are $\zeta = \Diamond A$ and $\zeta = \varphi$ of one of the form $\langle \alpha ! \rangle [A]\psi$ or $\langle \alpha ? \rangle A \to \psi$. This means that $[\![\zeta]\!]$ is a prime element of the model such that $[\![\zeta]\!] \neq \bot$

depending on the type of $\mathbf{M}$. By completeness of the program logic in the denotational semantics (Proposition 5.11) $\vdash_\tau \mathbf{M} : \zeta$ follows from $[\![\zeta]\!] \leq [\![\mathbf{M}]\!]_\rho$. By soundness (Proposition 5.9) of the program logic in the operational semantics $\mathbf{M} \models_\tau \zeta$. If τ is a basic type then ζ is of the form $\Diamond S_\ell$ and the definition of the satisfaction relation in that case implies that $\mathbf{M} \Downarrow$. If the type is $\sigma \to \tau$ then again the definition implies that $\mathbf{M} \Downarrow \mathbf{v}$ for some value $\mathbf{v}$. For the case $\tau = \mathsf{proc}$ we proceed by structural induction on $\zeta \neq \omega_{\mathsf{proc}}$. For example if $\mathbf{M} \models \langle\!\langle \alpha! \rangle\!\rangle [A]\varphi$ then $\mathbf{M} \stackrel{\alpha!}{\Longrightarrow} [\mathbf{v}]Q$ hence $\mathbf{M} \stackrel{\alpha}{\Longrightarrow}$. ∎

Adequacy is sufficient for the proof of soundness of the model. The proof of the converse requires a definability result to which we turn next.

6 Definability

We show in this section, adapting to the specific features of our language and extending the definability result of [24] for the functional fragment of our language, that every prime and compact element of the model is definable in *mini-Facile* and that the partial order on compact and primes can be captured operationally by appropriate tests. In order to increase readability throughout this section we consistently use the notation for elements of the domain we introduced in Section 4, see Definition 4.3 and Theorem 4.5. For a term $\mathbf{M}$ of functional type $\sigma \to \tau$ and an element $d \in T(\mathbf{D}_\sigma)$ or $d \in L(\mathbf{D}_{\mathsf{proc}})$, we will write $[\![\mathbf{M}]\!]_\rho(d)$ as an *abbreviation* for $\mathbf{apply}^T([\![\mathbf{M}]\!]_\rho, d)$. For simplicity of notation we write $\mathbf{P}_\alpha|_0\mathbf{Q}$ for $\mathbf{P}_{\{\alpha\}}|_\emptyset\mathbf{Q}$. We also write Ω for a typical divergent term of type σ or deadlocked term of type proc, where for example $\Omega_{\mathsf{int}} = (\mu X \lambda Y.XY)0$ (and similarly for the other ground types) and $\Omega_{\sigma\to\tau} = (\mu X \lambda Y.XY)(\lambda Z.\mathbf{M})$. It is convenient to abbreviate nested conditionals using a product term inductively defined by

$$\mathsf{if}\ (\textstyle\prod_{i=1}^{i=n+1} \mathbf{B}_i)\ \mathsf{then}\ \mathbf{M}\ \mathsf{else}\ \mathbf{N} = \mathsf{if}\ \mathbf{B}_1\ \mathsf{then}\ (\mathsf{if}\ (\textstyle\prod_{i=2}^{i=n} \mathbf{B}_i)\ \mathsf{then}\ \mathbf{M}\ \mathsf{else}\ \mathbf{N})\ \mathsf{else}\ \mathbf{N} \qquad (1)$$

where the case $n = 0$ is the usual conditional. We will abbreviate finite sums $\mathbf{M}_1 \oplus \cdots \oplus \mathbf{M}_s$ using a summation notation, $\sum_{i=1}^{i=s} \mathbf{M}_i$.

To define the prime elements of $T(\mathbf{D}_\sigma)$ it is sufficient to provide names $\mathbf{N}_\perp$ for the bottom element $\{\!|\perp|\!\}$ and $\mathbf{N}_c$ for the nontrivial primes $\{\!|c_\perp|\!\}$ where c is a compact element of the value domain $\mathbf{D}_\sigma$.

DEFINITION **6.1** If c is a compact element of $\mathbf{D}_\sigma$, for some $\sigma \in \mathsf{VType}$, then c is defined by the term $\mathbf{M}_c$ provided $\mathcal{V}[\![\mathbf{M}_c]\!]_\rho = c$. Similarly, if π is a prime of $\mathbf{D}_{\mathsf{proc}}$, then π is defined by $\mathbf{N}_\pi$ provided $[\![\mathbf{N}_\pi]\!]_\rho = \pi \in \mathbf{D}_{\mathsf{proc}}$. ∎

In Table 10 we define families of names $\mathbf{N}^\alpha_\pi, \mathbf{N}^\alpha_C, \mathbf{N}^\alpha_\perp$ for prime elements $\pi \in \mathbf{D}_{\mathsf{proc}}$ and compact elements $C \in \mathbf{D}_\sigma$ (hence primes of $T(\mathbf{D}_\sigma)$) for $\sigma \in \mathsf{VType}$, indexed by channel names $\alpha \in \mathcal{N}$ such that $[\![\mathbf{N}^\alpha_\pi]\!]_\rho = \pi$ and $[\![\mathbf{N}^\alpha_\perp]\!]_\rho = \{\!|\perp|\!\}, [\![\mathbf{N}^\alpha_c]\!]_\rho = \{\!|c|\!\}$ (we leave lifting implicit, for simplicity). To obtain names for all primes we need to be also able to capture operationally the partial order on prime and compact elements. This purpose is served by tests $\mathbf{T}^\alpha_\pi$ and $\mathbf{T}^\alpha_C$, in the sense made explicit in the statement of the Definability Theorem.

LEMMA **6.2** Let $c \in \mathcal{K}\mathbf{D}_\sigma, \pi \in \mathcal{KP}\mathbf{D}_{\mathsf{proc}}$ and $d \in T\mathbf{D}_\sigma, d' \in \mathbf{D}_{\mathsf{proc}}$. Then $[\![\mathbf{T}^\alpha_c]\!]_\rho(d) \in \{\perp, \{\mathbf{tt}\}\}$ and $[\![\mathbf{T}^\alpha_\pi]\!]_{\rho\alpha}|_0 d' \in \{\perp, \alpha_{out}(\{\mathbf{tt}\} \otimes \perp)\}$ for any channel α not occuring in c or π.

Table 10: Defining Terms for Primes

(

PROOF: The case $c \in \mathcal{K}\mathbf{D}_\sigma$ is obvious since the test $\mathbf{T}_c^\alpha$ is defined in terms of a conditional that either converges to $\mathbf{tt}$ or diverges.

Now let π be a prime in $\mathbf{D}_{\mathsf{proc}}$ and d any element of $\mathbf{D}_{\mathsf{proc}}$. Since the parallel operator on $\mathbf{D}_{\mathsf{proc}}$ is linear, by its definition, we may assume that $d = \pi'$ is a prime. We examine the cases for π and π'.

- If $\pi = \bot_{\mathsf{proc}}$, then $\mathbf{T}_\pi^\alpha = \alpha![\mathbf{tt}]\mathbf{nil}$ and then $\alpha_{out}(\{\!|\mathbf{tt}|\!\}\otimes\bot)_\alpha|_0\pi'$ can be computed using Table 4 with $\mathcal{A} = \{\alpha\}$ and $\mathcal{B} = \emptyset$. Going through the cases for π' we obtain $\alpha_{out}(\{\!|\mathbf{tt}|\!\}\otimes\bot)_\alpha|_0\bot = \alpha_{out}(\{\!|\mathbf{tt}|\!\}\otimes\bot)$ while for $\pi' \neq \bot$

2. For every compact element $c \in \mathcal{K}(\mathbf{D}_\sigma)$ and each channel $\alpha \in \mathcal{N}$

1. If α does not occur in c, $\mathbf{T}$

If ζ is a property of processes, $\zeta \in \mathcal{L}_{\mathsf{proc}}(\mathcal{T})$, then the cases ω_{proc} and $\varphi \& \psi$ are immediate. The other two cases, $\langle \alpha ! \rangle [A]\psi$ and $\langle \alpha ? \rangle A \rightarrow \psi$ are proven using the second part of the monotonicity Lemma 4.9 and induction. ∎

THEOREM **7.2**
The operational and denotational semantics for the program logic coincide. In other words

$$\Gamma \models_{\tau}^{\mathcal{O}} \mathbf{M} : \zeta \text{ if and only if } \Gamma \models_{\tau}^{D} \mathbf{M} : \zeta$$

PROOF: The direction ($\Leftarrow$) is immediate since $\Gamma \models_{\tau}^{D}$

PROOF: For (1), we first show that for *closed* process expressions $\mathbf{P}$, $\mathbf{Q}$, the hypothesis $H \rhd \mathbf{P} \sqsubseteq_{\mathcal{T}} \mathbf{Q}$ implies $[\![\mathbf{P}]\!]_\rho \leq [\![\mathbf{Q}]\!]_\rho$. To establish the latter we show that $\pi \leq [\![\mathbf{P}]\!]_\rho$ implies $\pi \leq [\![\mathbf{Q}]\!]_\rho$ for any prime π. From the Definability result we know that there are expressions $\mathbf{N}^\alpha_\pi$, such that $[\![\mathbf{N}^\alpha_\pi]\!]_\rho = \pi$; we choose an α which does not occur in $\pi, \mathbf{P}, \mathbf{Q}$. Then $[\![\alpha![\mathbf{tt}]\mathbf{nil}]\!]_\rho \leq [\![\mathbf{T}^\alpha_{\pi\,\alpha}|_0 \mathbf{N}^\alpha_\pi]\!]_\rho \leq [\![\mathbf{T}^\alpha_{\pi\,\alpha}|_0 \mathbf{P}]\!]_\rho$. It follows that $\mathbf{T}^\alpha_{\pi\,\alpha}|_0 \mathbf{P} \stackrel{\alpha!}{\Longrightarrow} [\mathbf{tt}]\Omega$, using Corollary 6.3. This remains true for α not occurring in any of $\pi, \mathbf{P}, \mathbf{Q}$. By Lemma 6.2 it follows that $\mathbf{T}^{\alpha,\pi}{}_w|_0 \mathbf{P} \stackrel{w!}{\Longrightarrow}$ and then the hypothesis implies $\mathbf{T}^{\alpha,\pi}{}_w|_0 \mathbf{Q} \stackrel{w!}{\Longrightarrow}$. Since we chose α not occurring in $\mathbf{Q}$, Lemma 6.2 implies that $\mathbf{T}^\alpha_{\pi\,\alpha}|_0 \mathbf{Q} \stackrel{\alpha!}{\Longrightarrow} [\mathbf{tt}]\mathbf{nil}$. Use Proposition 6.5 to conclude that $\pi \leq [\![\mathbf{Q}]\!]_\rho$.

We next consider the case when $\mathbf{M}$, $\mathbf{N}$ are *closed* expressions of some transmittable value type. Then $\mathbf{M} \sqsubseteq_{\mathcal{T}} \mathbf{N}$ implies $\beta![\lambda().\mathbf{M}]\mathbf{nil} \sqsubseteq_{\mathcal{T}} \beta![\lambda().\mathbf{N}]\mathbf{nil}$ and using the previous case we conclude $[\![\lambda().\mathbf{M}]\!]_\rho \leq [\![\lambda().\mathbf{N}]\!]_\rho$. It follows that $[\![\mathbf{M}]\!]_\rho \leq [\![\mathbf{N}]\!]_\rho$.

Finally consider the case when $\mathbf{M}$, $\mathbf{N}$ are arbitrary language expressions and assume that $H \rhd \mathbf{M} \sqsubseteq_{\mathcal{T}} \mathbf{N}$. It is sufficient to show that $[\![\mathbf{M}]\!]_\rho \leq [\![\mathbf{N}]\!]_\rho$ for any *compact* environment ρ, i.e. any environment such that $\rho(X)$ is a compact element for every variable X. We can use the Definability result to define a closed substitution s_ρ such that $[\![s_\rho(X)]\!]_\rho = \rho(X)$ for every variable X. From the Subsitution Lemma, Proposition 4.6, it follows that $[\![\mathbf{M}]\!]_\rho = [\![\mathbf{M}s_\rho]\!]_\rho$ and similarily for $\mathbf{N}$. Let $C[\cdot]$ be the context $\lambda X_1 \ldots \lambda X_k[\cdot]s_\rho(X_1)\ldots s_\rho(X_k)$, where $X_1 \ldots X_k$ are all the variables used in H. Then $\mathbf{M}s_\rho = C[\mathbf{M}]$. Moreover the expressions $C[\mathbf{M}]$, $C[\mathbf{N}]$ are both closed and $H \rhd \mathbf{M} \sqsubseteq_{\mathcal{T}} \mathbf{N}$ implies $H \rhd C[\mathbf{M}] \sqsubseteq_{\mathcal{T}} C[\mathbf{N}]$. Now use the previous case to conclude $[\![C[\mathbf{M}]]\!]_\rho \leq [\![C[\mathbf{N}]]\!]_\rho$.

For (2), suppose $H \rhd \mathbf{M} \sqsubseteq_{\mathcal{D}} \mathbf{N}$. We must show $\Gamma \models^{\mathcal{O}} \mathbf{M} : \zeta$ implies $\Gamma \models^{\mathcal{O}} \mathbf{N} : \zeta$. Let s be a closed substitution such that $s \models \Gamma$ and $\mathbf{M}s \models \zeta$. We must prove that $\mathbf{N}s \models \zeta$.

Let $C[\cdot]$ denote the context used in the previous Proposition, $\lambda X_1 \ldots \lambda X_k[\cdot]s(X_1)\ldots s_(X_k)$, where $X_1 \ldots X_k$ are all the variables used in H. Then $\mathbf{M}s = C[\mathbf{N}]$ and $\mathbf{N}s = C[\mathbf{N}]$. Also $H \rhd \mathbf{M} \sqsubseteq_{\mathcal{D}} \mathbf{N}$ implies $[\![C[\mathbf{M}]]\!]_\rho \leq [\![C[\mathbf{N}]]\!]_\rho$.

The completeness result for the logic $\mathcal{G}_E$, Corollary 7.4, means that $C[\mathbf{M}] \models \zeta$ implies $[\![\zeta]\!] \leq [\![C[\mathbf{M}]]\!]_\rho$ and therefore $[\![\zeta]\!] \leq [\![C[\mathbf{N}]]\!]_\rho$. Again using the logic comleteness we obtain $C[\mathbf{N}] \models \zeta$, i.e. the required $\mathbf{N}s \models \zeta$.

Finally, for (3), suppose $H \rhd \mathbf{M} \sqsubseteq_{\mathcal{L}} \mathbf{N}$ and $C[\cdot]$ is a context such that both $C[\mathbf{M}]$ and $C[\mathbf{N}]$ are closed expressions of type proc and $C[\mathbf{M}] \stackrel{\alpha!}{\Longrightarrow} [\mathbf{v}]\mathbf{Q}$ for some $\mathbf{v}, \mathbf{Q}$. We must show that $C[\mathbf{M}] \stackrel{\alpha!}{\Longrightarrow}$.

Let $p \leq [\![\mathbf{v}]\!]$ be a prime. Then $p = [\![A]\!]$ for some sentence A and it follows from the logic completeness results that $\mathbf{v} \models_\sigma A$. Then $C[\mathbf{M}] \models_{\mathsf{proc}} \langle\!\langle \alpha! \rangle\!\rangle [A]\omega$. The hypothesis implies that $C[\mathbf{N}] \models_{\mathsf{proc}} \langle\!\langle \alpha! \rangle\!\rangle [A]\omega$ and therefore $C[\mathbf{N}] \stackrel{\alpha!}{\Longrightarrow}$ as well.

The case when $C[\mathbf{M}] \stackrel{\alpha?}{\Longrightarrow}$ is similar. ∎

The Operators res_α **[The**

zero or more occurrences of the hole [ ]) such that for $\mathbf{P}$ a closed process term $\mathcal{C}[\mathbf{P}] : \mathsf{bool}$ then the hypothesis that $\mathcal{C}[\mathbf{P}] \Downarrow b \in \{\mathbf{tt}, \mathbf{ff}\}$ implies that for any closed process term $\mathbf{Q}$ there is also a reduction $\mathcal{C}[\mathbf{Q}] \Downarrow b$. The proof is by transition induction and, since there are no rules allowing an

$\lambda X.(X\,\lambda().\delta![\mathbf{ff}]\mathbf{nil}) : ((\mathsf{unit} \to \mathsf{proc}) \to$

[2] S. Abramsky, "Domain Theory in Logical Form", *Annals of Pure and Applied Logic*, vol 51, pp. 1-77, 1991.

[3] R. M. Amadio, "Translating Core Facile", Technical Report ECRC-1994-3, European Computer-Industry Research Center, Munich, 1994.

[4] H.P. Barendregt, *The λ-Calculus, its Syntax and Semantics*, North-Holland, Amsterdam, 1984.

[5] G. Boudol, "A λ-Calculus for (Strict) Parallel Functions", *Information and Computation*, vol 108, pp. 51-127, 1994.

[6] H. Barendregt, M. Coppo and M. Dezani-Ciancaglini, "A Filter Model and the Completeness of Type Assignment", *Journal of Symbolic Logic*, vol 48, pp. 931-940, 1983.

[7] A. Giacalone, P. Mishra and S. Prasad, "FACILE: A symmetric integration of Concurrent and Functional Programming", *International Journal of Parallel Programming*, vol 15, No 2, pp. 121-160, 1989.

[8] W. Fereira, M. Hennessy and A. Jeffrey, "Combining the Typed λ-Calculus with CCS", Report 2/96, University of Sussex, Computer Science, May 1996.

[9] C. Fournet and G. Gonthier, "The Reflexive CHAM and the Join-Calculus", Proc. POPL 94, 1994.

[10] A. Jeffrey, "A Fully Abstract Semantics for a Concurrent Functional Language with Monadic Types", Proc. *LICS'95*, 1995.

[11] M. Hennessy, *An Algebraic Theory of Processes*, MIT Press, Cambridge, MA, 1988.

[12] M. Hennessy, "A Fully Abstract Denotational Model for Higher-Order Processes", *Information and Computation* vol. 112, No 1, pp. 55-95, 1994.

[13] M. Hennessy, "Higher-Order Processes and their Models", *ICALP '94*.

[14] K.G. Larsen, " Proof Systems for Satisfiability in Hennessy-Milner Logic with Recursion", *Theoretical Computer Science* vol 72, 265-288, 1990.

[15] R. Milner, *Communication and Concurrency*, Prentice-Hall, Englewood Cliffs, NJ, 1989.

[16] R. Milner, J. Parrow and D. Walker, "A Calculus of Mobile Processes I, II", *Information and Computation* vol 100, pp 1-40, pp 41-77, 1992.

[17] E. Moggi, "Notions of Computation and Monads", *Information and Computation* **93**, 1991, pp. 55-92.

[18] J. Morris, "Lambda-Calculus Models of Programming Languages", Ph.D. Thesis, MIT, 1968.

[19] G. Plotkin, "A Powerdomain Construction", *SIAM Journal on Computation*, vol 5, pp 452-487, 1976.

[20] G. Plotkin, "LCF Considered as a Programming Language", *Theoretical Computer Science* vol 5, pp. 323-355, 1997.

[21] J.H. Reppy, "A Higher-Order Concurrent Language", in *Proceedings of the ACM SIGPLAN '91 PLDI, SIGPLAN Notices*, No 26, pp. 294-305, 1991.

[22] J.H. Reppy, *Higher-Order Concurrency*, Ph.D. thesis, Cornell University, 1991.

[23] J. Reppy, "CML: A Higher-Order Concurrent Language", in *Proc. ACM-SIGPLAN 91, Conf. on Programming Language Design and Implementation*, 1991.

[24] K. Sieber, "Call-by-Value and Nondeterminism", in *Typed λ-Calculi and Applications*, eds. M. Bezen and J.F.Groote, LNCS 664, Springer-Verlag 1993.

[25] I. Stark, "A Fully-Abstract Domain Model for the π-Calculus", Proc. *LICS'96*, 1996.

[26] C. Stirling, "A Proof-Theoretic Characterization of Observational Equivalence", *Theoretical Computer Science* vol 39, 27-45, 1985.

[27] C. Stirling, "Modal Logics for Communicating Systems", *Theoretical Computer Science* vol 49, 311-347, 1987.

[28] B. Thomsen, *Calculi for Higher-Order Communicating Systems*, Ph.D. thesis, Imperial College, 1990.

[29] B. Thomsen, "Plain Chocs: A Second Generation Calculus for Higher Order Processes", *Acta Informatica* vol 30, pp 1-59, 1993.

[30] G. Winskel, "A Complete Proof System for SCCS with Modal Assertions", LNCS vol 206, 392-410, 1985.