Information and Computation **145**, 64–106 (1998)
Article No. IC982723

# Full Abstractness for a Functional/Concurrent



Chrysafis Hartonas and Matthew Hennessy*

*COGS, University of Sussex at Brighton, Falmer, Brighton BN1 9QH, United Kingdom*
E-mail: hartonas@cogs.susx.ac.uk, matthewh@cogs.susx.ac.uk

We study an applied typed call-by-value $\lambda$-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 labeled transition system which is then used to give a behavioral 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 behavior preorder; expressions are related in the model if and only if they are related behaviorally.

The proof method uses concepts that are of independent interest. It involves characterizing the domain using filters of a property logic for program expressions and developing a program logic for relating program expressions with property formulae. 

## 1. INTRODUCTION

A number of new languages for describing distributed systems have emerged in recent years. These include, among others, CML [30], the $\pi$-calculus [23], the Join Calculus [14], and Facile [10]. These languages contain many interesting and complex features for the organization of distributed computation. In this paper we concentrate on one important aspect, the interaction between higher-order constructs and interprocess communication.

Our language, which we refer to as *higher-order CCS*, HOCCS, contains base types such as bool and int and exponential types $\sigma \to \tau$ for functions, and the formation rules for expressions of these types are exactly as those found in the $\lambda$-calculus [5]. There is one new type, proc, for processes. The syntax for expressions with this type is borrowed from the process algebra CCS [22] in particular

* Research supported by EPSRC grant GR/K60701.



0890-5401/98 $25.00

the value-passing version. A set of communication channel names, $\mathcal{N}$, is assumed, each with an associated type. Thus $\alpha![\mathbf{M}]\,\mathbf{N}$ is an expression of type proc if $\mathbf{N}$ is an expression of type proc and $\mathbf{M}$ has the same type as that of the channel $\alpha$. The reader familiar with process algebras will recognize it as a process which can output on the channel $\alpha$ and then proceed as its residual $\mathbf{N}$. Similarly $\alpha?\mathbf{M}$ is an expression of type proc, assuming $\mathbf{M}$ is a functional expression of type $\sigma \to \mathsf{proc}$ where $\sigma$ is the type of the channel $\alpha$, which can apply an input received from $\alpha$ to the function represented by $\mathbf{M}$. There are also constructors such as $\mathbf{M}|\mathbf{N}$ for placing *processes* in parallel.

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 $\lambda$-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 $\sigma$ 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 $\mathsf{unit} \to \mathsf{proc}$, which are often called *scripts*, may be freely exchanged.

The language we study has much in common with *core Facile*, as described in [4] but with two serious omissions. In HOCCS the computations generated by processes have very restricted impact on the computations of values; this can only be effected using the operator $\mathsf{res}_\alpha$, which when applied to a process returns a value. In *core Facile* the interaction between process and value computations is much more extensive. The second omission is the absence from HOCCS of dynamic channel generation, familiar from the $\pi$-calculus and common to languages such as Facile and CML.

In Section 3 we give an operational semantics HOCCS based on labeled transition systems. In the definition there are three kinds of judgements between closed expressions:

- $\mathbf{M} \to \mathbf{N}$. This generalizes the call-by-value reduction relation of the $\lambda$-calculus. It incorporates $\beta$-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 $\alpha$; 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 $\alpha$. 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 [25, 3, 6]. 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. Because of the constructs in our language it is straightforward to show that these two variations coincide.

In Section 4 we give a denotational model for the language. With each type we associate a domain for interpreting expressions of that type. For transmittable types $\sigma$ this has the form $T(\mathbf{D}_\sigma)$, where $\mathbf{D}_\sigma$ is a domain of *values* of that type, a bounded-complete algebraic dcpo, and $T$ is a suitable monad on the category of bounded-complete algebraic dcpo's. Intuitively $T(\mathbf{D}_\sigma)$ is a domain of *computations*, in the spirit of Moggi [24]. Following these ideas the value domains are constructed in the standard manner. For base types suitable bounded-complete algebraic dcpo's are chosen while values of functional type $\sigma_1 \to \sigma_2$ are taken to be $[\mathbf{D}_{\sigma_1} \to T(\mathbf{D}_{\sigma_2})]$, the set of morphisms (continuous functions) from the domain $\mathbf{D}_{\sigma_1}$ of values of type $\sigma_1$ to the domain $T(\mathbf{D}_{\sigma_2})$ of computations of type $\sigma_2$. The monad used in these constructions is the composition $T = \mathscr{P}_H(\cdot)_\perp$ of the Hoare powerdomain monad with the lifting monad. This means that the domains in which terms of the language are interpreted have the structure of *prime algebraic lattices.*

However, the type $\mathsf{proc}$ introduces a complication. Intuitively there are no values of this type and therefore this domain, $\mathbf{D}_{\mathsf{proc}}$ has to be considered separately. Moreover the value domains of type $\sigma \to \mathsf{proc}$ must also be considered separately; they will have the simple form $[\mathbf{D}_\sigma \to L(\mathbf{D}_{\mathsf{proc}})]$ where $L$ is the lifting monad $L(\mathbf{D}) = \mathbf{D}_\perp$.

The domain $\mathbf{D}_{\mathsf{proc}}$ is defined as the initial solution of a domain equation with coefficients in the category of algebraic dcpo's, and by construction will also be a prime algebraic lattice. The equation has the form

$$\mathbf{D}_{\mathsf{proc}} = \prod_{\alpha \in \mathcal{N}} (C(\mathbf{D}_{\sigma(\alpha)}, \mathbf{D}_{\mathsf{proc}}) \times A(\mathbf{D}_{\sigma(\alpha)}, \mathbf{D}_{\mathsf{proc}})),$$

where $C$ and $A$ are suitable constructors modeling concretions and abstractions. This equation encapsulates the view that each process is determined by its potential to input/output on each channel $\alpha$.

To model the input potential (abstractions) we will not use the space of functions from $\mathbf{D}_\sigma$ to $T(\mathbf{D}_{\mathsf{proc}})$, for reasons discussed just above but also because this space introduces distinctions that cannot be detected by our notion of observation. The terms $\lambda X.(\mathbf{P} \oplus \mathbf{Q})$ and $\lambda X.\mathbf{P} \oplus \lambda X.\mathbf{Q}$ are operationally distinguishable and are interpreted differently in the domain $T(\mathbf{D}_{\sigma \to \mathsf{proc}})$. However, this difference cannot be observed in the terms $\alpha?\lambda X.(\mathbf{P} \oplus \mathbf{Q})$ and $\alpha?(\lambda X.\mathbf{P} \oplus \lambda X.\mathbf{Q})$. To take this phenomenon into account input potential is represented as a function in a space $A(\mathbf{D}_{\sigma(\alpha)}, \mathbf{D}_{\mathsf{proc}}) = [\mathbf{D}_{\sigma(\alpha)} \to L(\mathbf{D}_{\mathsf{proc}})]$.

The output potential is represented, approximately, by a pair of objects, one from $\mathbf{D}_{\sigma(\alpha)}$ representing the value to be output and one from $\mathbf{D}_{\mathsf{proc}}$ itself, representing its

residual after this output is performed. The nondeterminism at the level of evaluation imposes that we in fact represent output along a port $\alpha$ by a pair consisting of a computation of type $\sigma(\alpha)$, hence an element of the domain $T(\mathbf{D}_{\sigma(\alpha)})$ and a residual process. However, we cannot use the standard Cartesian product to combine these two components because of the distributivity of prefixing over internal choice, $\oplus$; behaviorally $\alpha![\mathbf{M}\oplus\mathbf{N}]\,\mathbf{P}$ is identified with $\alpha![\mathbf{M}]\,\mathbf{P}\oplus\alpha![\mathbf{N}]\,\mathbf{P}$ and $\alpha![\mathbf{M}](\mathbf{P}\oplus\mathbf{Q})$ with $\alpha![\mathbf{M}]\,\mathbf{P}\oplus\alpha![\mathbf{M}]\,\mathbf{Q}$. However, using cartesian product would also identify $\alpha![\mathbf{M}\oplus\mathbf{N}]\,\mathbf{P}\oplus\mathbf{Q}$ with $\alpha![\mathbf{M}]\,\mathbf{P}\oplus\alpha![\mathbf{N}]\,\mathbf{Q}$. Instead we use a bilinear tensor product $\otimes$. By construction of the tensor product (see proof of Proposition 2.4) the first of the two equations above will be satisfied while the third will fail. This is because extending tensor notation to elements of the domain we will have

$$(a\vee b)\otimes c=(a\otimes c)\vee(b\otimes c)$$

$$a\otimes(b\vee c)=(a\otimes b)\vee(a\otimes c).$$

But

$$(a\vee b)\otimes(c\vee d)\neq(a\otimes c)\vee(b\otimes d).$$

Furthermore and because a process such as $\alpha![\mathbf{M}]\,\mathbf{K}$, where $\mathbf{M}$ cannot reduce to a value can never output on $\alpha$; communication is call-by-value. This means that to properly represent the output potential of processes a *left-strict* tensor ${}^{s}\otimes$ is required.

The major contribution of this paper is, first, the construction of a fully abstract model and, second, a complete proof system for satisfiability that can be then used to argue about properties of programs. The functional fragment of our language is essentially a version of call-by-value, nondeterministic PCF. Thus we aim at reusing the model of [32], extending it in a way that preserves the full-abstraction result for $\mathrm{PCF}_{nv}$ reported in [32]. Our proof technique for full abstraction is different in that it uses the approach advocated in [3]. In analogy with [7–9] the domains are characterized using the filters of a typed logic for program properties; similar characterizations, using different logics, appear in [3, 6, 19]. The interpretation of the language is characterized in terms of a *program logic* whose judgements take the form $\Gamma\vdash_{\tau}\mathbf{M}:\zeta$ where $\mathbf{M}$ is a language expression, $\zeta$ is a formula representing a property and $\Gamma$ 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 $\sigma$ of transmittable expressions, introducing a distinction between properties of values and properties of computations, alongside the properties for processes. The logic then uses two provability predicates $\mid\sim$ and $\vdash$, with appropriate interaction rules. Our aim is to provide a complete logical proof system that can be used to argue about operational behavior. Results of this kind have been produced for CCS and SCCS, independently of the full abstraction question [21, 34, 35, 38]. As a step in this direction we first give a denotational semantics for the logic, based on an interpretation of formulae as compact elements of the model, and prove a completeness theorem for this semantic notion. The details are in Section 5.

Completeness of the program logic in the denotational semantics together with a sound behavioral interpretation of the program logic, based on the operational semantics, is sufficient to establish the first main result of the paper, *Adequacy*:

for closed expressions **M**, the interpretation of **M** is different than $\perp$ if and only if $\mathbf{M}\Downarrow$, i.e., intuitively **M** can produce some value operationally.

From adequacy it is not too difficult to establish one half of full-abstraction; if two expressions are related denotationally they are related operationally. The converse requires a *definability* result. Our language is sufficiently expressive to define all compact elements in the domains. This theorem is the subject of Section 7. It should be pointed out that it depends on the powerful parallel operator $_{\mathscr{A}}|_{\mathscr{B}}$ introduced in [19]. The operator $_{\mathscr{A}}|_{\mathscr{B}}$ restricts certain channels to be used only for internal communication and it allows us to test for input or output activity on specified channels. It also depends on a novel operator for extracting a value from a process; the expression $\mathsf{res}_{\alpha}(\mathbf{P})$ evaluates to any value output by the process **P** on the channel $\alpha$. We use this operator in defining functions into processes. We show that full abstraction fails without this latter operator.

The full abstractness results are given in Section 7. This section also contains various completeness results for the logics which have been introduced in the course of the paper. In particular, using the definability result the program logic is shown complete for the operational semantics.

## 2. MATHEMATICAL PRELIMINARIES

To facilitate the reading of the paper we devote this section to gathering together the notation used, explaining the constructions required to model our programming language, and drawing attention to some details of significance in later developments. For basic concepts and terminology of domain theory and category theory the reader is referred to any standard reference, e.g., [2].

We interpret HOCCS in the Cartesian closed category of bounded-complete algebraic dcpo's with continuous functions as morphisms. More precisely, the model is specified by a domain equation with parameters the discrete dcpo's of integers, booleans and the one-element unit domain, to be solved for function space and for the process domain. It will follow from the form of the domain equation and given relevant closure properties of the category of prime algebraic lattices that the value domains for functional terms and the domain for processes are prime algebraic lattices.

For an algebraic dcpo **D**, $\mathscr{K}(\mathbf{D})$ and $\mathscr{KP}(\mathbf{D})$ denote, respectively, the sets of compact and compact-prime elements of **D**, where recall that a compact element $p$ is *prime* provided $p \leqslant d \vee d'$ implies $p \leqslant d$ or $p \leqslant d'$. Prime algebraic lattices can be constructed from any partial order $(P, \leqslant_p)$ with a bottom element $\perp$. Let $\mathrm{Fin}(P)$ be the set of all finite nonempty subsets of $P$ ordered by the lower Egli-Milner order ($F \leqslant_\ell G$ iff $\forall u \in F\ \exists v \in G\ u \leqslant_p v$). $\mathrm{Idl}(\mathrm{Fin}(P))$ is the ideal completion of $P$ (recall that an *ideal* is a lower directed subset); we denote this construction by $\mathscr{P}_I(P)$.

LEMMA 2.1. *Let $(P, \leqslant_p)$ be any partial order with bottom. Then $\mathscr{P}_l(P)$ 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 $\mathscr{P}\_$ is a *free* functor delivering, for each dcpo $\mathbf{D}$, the free semilattice dcpo $\mathscr{P}\_(\mathbf{D})$ satisfying a predetermined set of equations with respect to the semilattice operation $\uplus$ of *formal union*. As usual, $\{\!|\cdot|\!\}: \mathbf{D} \to \mathscr{P}\_(\mathbf{D})$ denotes the *formal singleton* (insertion of generators) continuous map. We use $\mathscr{P}_H(\mathbf{D})$ to denote the lower (Hoare) powerdomain of $\mathbf{D}$.

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

*Proof.* See for example [2], 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 $\mathscr{C}$ 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 $\mathbf{E}$). Initiality for $\mathbf{lin}_{D,E}: \mathbf{D} \times \mathbf{E} \to \mathbf{D} \otimes \mathbf{E}$ simply means that if $f: \mathbf{D} \times \mathbf{E} \to \mathbf{E}'$ is any (bi)linear morphism, then there exists a unique linear morphism $\hat{f}: \mathbf{D} \otimes \mathbf{E} \to \mathbf{E}'$ making the relevant triangle commute, i.e., such that $f = \hat{f} \circ \mathbf{lin}_{D,E}$.

The function $\mathbf{lin}_{D,E}$ is the insertion of generators map into $\mathscr{P}_\ell(P)$, where $P$ is the partial order of pairs $(p, q)$ with $p$ prime in $\mathbf{D}$ and $q$ prime in $\mathbf{E}$, ordered coordinatewise. Recall that $\mathscr{P}_\ell(P)$ abbreviates Idl(Fin($P$)). Initiality of $\mathbf{lin}$ follows from the universal property of ideal completions. ∎

Following established practice in algebra, we write $p \otimes q$ rather than $\mathbf{lin}_{D,E}(p, q)$ and then linearly extend $\otimes$ to all elements $d \in \mathbf{D}$, $e \in \mathbf{E}$ by setting $d \otimes e = \bigvee_{p \leqslant d} \bigvee_{q \leqslant e} p \otimes q$. Thus $\otimes$ is used both as a constructor on domains and as a binary operation on elements. It is immediate to see that $a \otimes b \leqslant a' \otimes b'$ iff $a \leqslant a'$ and $b \leqslant b'$. Since the construction is easily seen to be functorial, we also use the notation $f \otimes g$ for the result of applying the tensor functor to the linear maps $f$ and $g$. The definition of $f \otimes g$ is the obvious one, $(f \otimes g)(d \otimes e) = (fd) \otimes (ge)$. Symmetry and associativity of $\otimes$ can be easily shown, as they are inherited from the corresponding properties of the Cartesian product.

Because of call-by-value we are unable to use $\otimes$ directly for the interpretation of concretions; we need a left-strict version. A (bi)linear function $f: \mathbf{D} \times \mathbf{D}' \to \mathbf{E}$ is *left-strict* provided $f(\perp_D, d') = \perp_E$. By an argument similar to that in the proof of Proposition 2.4 we can show that:

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 $\sigma$ while the domains of computations of type $\sigma$ take the form $T(\mathbf{D}_\sigma)$ where $T$ is the composite monad $T = L\mathscr{P}_H(\ )$ obtained from the Hoare powerdomain and the lifting monad. The unit $\eta$ of the monad is the map $\eta = \{\!|(\ )_\perp|\!\}$ and the multiplication $\mu$ 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 $[\mathbf{D}_{\sigma'} \to T(\mathbf{D}_\sigma)]$ while computation of that type will be interpreted in $T([\mathbf{D}_{\sigma'} \to T(\mathbf{D}_\sigma)])$. The monad $T$ comes equiped with extension maps

$$\mathbf{ext}: [\mathbf{A} \to T\mathbf{B}] \to [T\mathbf{A} \to T\mathbf{B}] \qquad \mathbf{ext}^{(2)}: [\mathbf{A} \times \mathbf{B} \to T\mathbf{C}]] \to [T\mathbf{A} \times T\mathbf{B} \to T\mathbf{C}]$$

where given $f: \mathbf{A} \to T\mathbf{B}$, $\mathbf{ext}(f)$ is defined as the composition $T\mathbf{A} \xrightarrow{T(f)} T^2\mathbf{B} \xrightarrow{\mu} T\mathbf{B}$, where $\mu$ is the multiplication of the monad. The map $\mathbf{ext}^{(2)}$ is derived from $\mathbf{ext}$ using the categorical structure and its set-theoretic definition is

$$\mathbf{ext}^{(2)} := \lambda f.\mathbf{ext}(\lambda a \in \mathbf{A}.(\mathbf{ext}\ \lambda b \in \mathbf{B}.fab)$$

In other words, $\mathbf{ext}$ and $\mathbf{ext}^{(2)}$ are defined on continuous functions $f, h$ as the strict functions $\mathbf{ext}(f)$, $\mathbf{ext}^{(2)}(h)$, letting $\mathbf{ext}(f)\{\!|\perp|\!\} = \perp$ and similarly for $\mathbf{ext}^{(2)}$, then determining their action on primes by

$$\mathbf{ext}(f)\{\!|c|\!\} = fc \qquad \mathbf{ext}^{(2)}(h)(\{\!|c|\!\}, \{\!|k|\!\}) = h(c,k),$$

and finally linearly extended to all elements. Hence $\mathbf{ext}(f)(\mathscr{I})$ is the ideal generated by the set of $\mathbf{ext}(f)\{\!|c|\!\}$ with $\{\!|c|\!\} \leqslant \mathscr{I}$ and similarly for $\mathbf{ext}^{(2)}$. We set $\mathbf{apply}^T = \mathbf{ext}^{(2)}(\mathbf{apply})$ where $\mathbf{apply}$ is the standard application continuous morphism supplied by the Cartesian closed structure of bounded complete algebraic dcpo's. Since the ideal generated by the $\mathbf{ext}(f)\{\!|c|\!\}$ with $\{\!|c|\!\} \leqslant \mathscr{I}$ is simply the join in $T\mathbf{B}$ of the $\mathbf{ext}(f)\{\!|c|\!\}$ this allows us to use similar extension maps for the case of application for functions into processes, where recall that we set $\mathbf{D}_{\sigma \to \mathsf{proc}} = \mathbf{D}_\sigma \to L(\mathbf{D}_{\mathsf{proc}})$. Given the map $\mathbf{apply}_{A,LB}: [\mathbf{A} \to L(\mathbf{B})] \times \mathbf{A} \to L(\mathbf{B})$ and provided $\mathbf{B}$ is a complete lattice, as is the case for $\mathbf{D}_{\mathsf{proc}}$, the map $\mathbf{ext}^{(2)}(\mathbf{apply}): T(\mathbf{A} \to L(\mathbf{B})) \times T\mathbf{A} \to L(\mathbf{B})$ is

defined as above for primes and linearly extended to arbitrary elements using the join of **B**. We use $\mathbf{apply}^T$ for both cases discussed. Context will determine the details of application.

## 3. LANGUAGE AND OPERATIONAL SEMANTICS

In this section we describe the programming language and its operational semantics.

### *Type Inference System*

The type system for the programming language is given by the grammar below. For convenience we assume distinct base types for booleans and unit.

$$\begin{aligned} \sigma_G \in \mathsf{GType} &::= \mathsf{unit} \mid \mathsf{bool} \mid \mathsf{int} \\ \sigma \in \mathsf{VType} &::= \sigma_G \mid \sigma \to \tau \\ \tau \in \mathsf{Type} &::= \sigma \mid \mathsf{proc} \end{aligned}$$

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 $\mathsf{unit} \to \mathsf{proc}$ is a transmittable value type, objects which may be construed as *delayed* processes may be exchanged.

The language, which we call HOCCS is given by the grammar below, where $X$ ranges over a set Var of variables and $\alpha$ 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 $\sigma$ as a function and write $\sigma(\alpha)$ for the value-type of $\alpha$.

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

The language may be viewed as an applied call-by-value $\lambda$-calculus, with a recursion operator. We abbreviate $\mu X \lambda Y.\mathbf{M}$ as $\lambda Y.\mathbf{M}$ when $X$ does not occur free in $\mathbf{M}$ and we typically write $\lambda().\mathbf{M}$ for functionals of type $\mathsf{unit} \to \tau$. In addition to some literals for ground types there are a series of special constructors for process terms. These include standard prefix and choice operators, a parameterized parallel operator, and a new operator, $\mathsf{res}_\alpha$, for extracting a value from a process. Strictly speaking we might have just a single choice operator on processes as the semantics

and observational preorder we consider in this paper will make no distinction between the two. No harm is done by having both, however, in anticipation of further work on preorders such as must testing that will distinguish between the two. We also point out that the choice operator on transmittable types is definable given the operator res.

The language is typed using typing judgements of the form $H \rhd \mathbf{M} : \tau$, where $H$ is a *typing context*, i.e., a finite set of pairs $X_1 : \sigma_1, ..., X_s : \sigma_s$, where $\sigma_i \in \mathsf{VType}$. In writing $H, X : \sigma \rhd \mathbf{M} : \tau$ it is assumed that $X$ does not already occur in the typing context $H$. We let $\mathscr{H}$ be the set of all typing contexts. The type inference system is given in Table 1 and is an extension of the standard typing system for the $\lambda$-calculus.

*Operational Semantics*

In the sequel we use **P**, **Q** to range over well-typed expressions of type proc while **M**, **N** will range over well-typed expressions of arbitrary type. A term **M** is *closed* iff for any typing context $H$ and in particular for the empty context, $H \rhd \mathbf{M} : \tau$ is derivable for some type $\tau$. We then say that **M** is of type $\tau$, usually denoted by $\mathbf{M} : \tau$. A closed value expression will usually be refered to as a value.

The operational semantics for the language is given in Table 2. This is given in terms of a binary reduction relation $\rightarrow$ between well-typed *closed* expressions of the same type. On expressions of transmission value type this relation is similar to call-by-value reduction in the $\lambda$-calculus, where *values* are either literals, for ground types, or $\lambda$-abstractions for higher types, while on process terms it is similar to *internal moves*, $\xrightarrow{\tau}$, commonly used in process algebras. As usual to define this reduction relation over process terms we need two auxiliary relations $\xrightarrow{\alpha?}$, $\xrightarrow{\alpha!}$,

**TABLE 1**

**Type Inference System**

$\rhd \mathbf{tt}$: bool $\qquad \rhd \mathbf{ff}$: bool $\qquad \rhd \mathbf{n}$: int

$\rhd ()$: unit $\qquad \rhd \mathbf{il}$: proc $\qquad X : \sigma \rhd X : \sigma$

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

$$\frac{H \rhd \mathbf{M} : \sigma_G \quad H \rhd \mathbf{N} : \sigma_G}{H \rhd \mathbf{M} = \mathbf{N} : \mathsf{bool}}\ (eq)$$

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

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

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

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

$$\frac{H \rhd \mathbf{M} : \sigma \to \mathsf{proc}}{H \rhd \alpha? \mathbf{M} : \mathsf{proc}}\ (\alpha \in \mathscr{N}^{\sigma})\ (\alpha?) \qquad \frac{H \rhd \mathbf{M} : \mathsf{proc}}{H \rhd \mathsf{res}_{\alpha}(\mathbf{M}) : \sigma}\ (\alpha \in \mathscr{N}^{\sigma})\ (res)$$

**TABLE 2**

**Operational Semantics**

Axioms

$\mathbf{M} \oplus \mathbf{N} \to \mathbf{M}$, $\mathbf{M} \oplus \mathbf{N} \to \mathbf{N}$

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

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

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

if **tt** then **M** else **N** $\to$ **M**

if **ff** then **M** else **N** $\to$ **N**

$\ell = \ell \to \mathbf{tt}$

Rules

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

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

$$\frac{\mathbf{M} \to \mathbf{M}' \qquad \mathbf{N} \to \mathbf{N}'}{\mathbf{M} = \mathbf{N} \to \mathbf{M}' = \mathbf{N}'} \qquad \frac{\mathbf{M} \to \mathbf{M}'}{\mathbf{MN} \to \mathbf{M}'\mathbf{N}} \qquad \frac{\mathbf{M} \to \mathbf{M}'}{\mathbf{vM} \to \mathbf{vM}'}$$

$$\frac{\mathbf{M} \to \mathbf{N}}{\alpha?\mathbf{M} \to \alpha?\mathbf{N}} \qquad \frac{\mathbf{M} \to \mathbf{N}}{\alpha! [\mathbf{M}]\, \mathbf{Q} \to \alpha! [\mathbf{N}]\, \mathbf{Q}} \qquad \frac{\mathbf{P} \to \mathbf{Q}}{\mathsf{res}_\alpha(\mathbf{P}) \to \mathsf{res}_\alpha(\mathbf{Q})}$$

$$\frac{\mathbf{P} \xrightarrow{\alpha?} \mu X \lambda Y.\mathbf{P}'\ (\alpha \in A)}{\mathbf{P}\ {}_{\mathscr{A}}|_{\mathscr{B}}\ \mathbf{Q} \xrightarrow{\alpha?} \mu X \lambda Y.(\mathbf{P}'\ {}_{\mathscr{A}}|_{\mathscr{B}}\ \mathbf{Q})} \qquad \frac{\mathbf{Q} \xrightarrow{\alpha?} \mu X \lambda Y.\mathbf{Q}'\ (\alpha \in B)}{\mathbf{P}\ {}_{\mathscr{A}}|_{\mathscr{B}}\ \mathbf{Q} \xrightarrow{\alpha?} \mu X \lambda Y.(\mathbf{P}\ {}_{\mathscr{A}}|_{\mathscr{B}}\ \mathbf{Q}')} \qquad \frac{\mathbf{P} \xrightarrow{\alpha!} [\mathbf{v}]\, \mathbf{Q}}{\mathsf{res}_\alpha(\mathbf{P}) \to \mathbf{v}}$$

$$\frac{\mathbf{P} \xrightarrow{\alpha!} [\mathbf{v}]\, \mathbf{P}' \qquad \mathbf{Q} \xrightarrow{\alpha?} \mu X \lambda Y.\mathbf{Q}'}{\mathbf{P}\ {}_{\mathscr{A}}|_{\mathscr{B}}\ \mathbf{Q} \to \mathbf{P}'\ {}_{\mathscr{A}}|_{\mathscr{B}}\ (\mu X \lambda Y.\mathbf{Q}')\, \mathbf{v}} \qquad \frac{\mathbf{P} \xrightarrow{\alpha?} \mu X \lambda Y.\mathbf{P}' \qquad \mathbf{Q} \xrightarrow{\alpha!} [\mathbf{v}]\, \mathbf{Q}'}{\mathbf{P}\ {}_{\mathscr{A}}|_{\mathscr{B}}\ \mathbf{Q} \to (\mu X \lambda Y.\mathbf{P}')\, \mathbf{v}\ {}_{\mathscr{A}}|_{\mathscr{B}}\ \mathbf{Q}'}$$

$$\frac{\mathbf{P} \xrightarrow{\alpha!} [\mathbf{v}]\, \mathbf{P}'\ (\alpha \in A)}{\mathbf{P}\ {}_{\mathscr{A}}|_{\mathscr{B}}\ \mathbf{Q} \xrightarrow{\alpha!} [\mathbf{v}](\mathbf{P}'\ {}_{\mathscr{A}}|_{\mathscr{B}}\ \mathbf{Q})} \qquad \frac{\mathbf{Q} \xrightarrow{\alpha!} [\mathbf{u}]\, \mathbf{Q}'\ (\alpha \in B)}{\mathbf{P}\ {}_{\mathscr{A}}|_{\mathscr{B}}\ \mathbf{Q} \xrightarrow{\alpha!} [\mathbf{u}](\mathbf{P}\ {}_{\mathscr{A}}|_{\mathscr{B}}\ \mathbf{Q}')}$$

$$\frac{\mathbf{B} \to \mathbf{B}'}{\text{if } \mathbf{B} \text{ then } \mathbf{M} \text{ else } \mathbf{N} \to \text{if } \mathbf{B}' \text{ then } \mathbf{M} \text{ else } \mathbf{N}}$$

defined over well-typed closed process terms, which capture the potential for input and output over the channel $\alpha$.

Most of the rules are straightforward but note that recursion is only allowed over abstractions and the unwinding of recursive definitions occurs at the point when an abstraction is applied to a value. This accounts for the slightly complicated form of the call-by-value $\beta$-reduction rule

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

but note that this form of $\beta$-reduction will be also used to implement communication between processes:

$$\frac{\mathbf{P} \xrightarrow{\alpha!} [\mathbf{v}]\, \mathbf{P}' \qquad \mathbf{Q} \xrightarrow{\alpha?} \mu X \lambda Y.\mathbf{Q}'}{\mathbf{P}\ {}_{\mathscr{A}}|_{\mathscr{B}}\ \mathbf{Q} \to \mathbf{P}'\ {}_{\mathscr{A}}|_{\mathscr{B}}\ (\mu X \lambda Y.\mathbf{Q}')\, \mathbf{v}}.$$

However, only values may be exchanged in a communication. More specifically in the process expression $\mathbf{P}\ {}_{\mathscr{A}}|_{\mathscr{B}}\ \mathbf{Q}$ for a communication to occur between **P** and **Q** we must have that

- **P** is ready to output a value on a channel $\alpha$, i.e., $\mathbf{P} \xrightarrow{\alpha!} [\mathbf{v}]\ \mathbf{P}'$, and
- **Q** is in a form of *head normal form*, ready to receive a value on $\alpha$, $\mathbf{Q} \xrightarrow{\alpha?} \mu X \lambda Y.\mathbf{Q}'$.

So, for example, for a communication to occur in $\alpha![\mathbf{M}]\mathbf{P}\ {}_{\mathscr{A}}|_{\mathscr{B}}\ \alpha?\mathbf{Q}$, first **M** must be reduced to a value **v**, **Q** must be reduced to a $\lambda$-abstraction, and then the transmission of **v** may occur.

The two relatively nonstandard operators we use are the following:

- The parameterized form of communication $\mathbf{P}\ {}_{\mathscr{A}}|_{\mathscr{B}}\ \mathbf{Q}$, which restricts possible moves of the construct $\mathbf{P}\ {}_{\mathscr{A}}|_{\mathscr{B}}\ \mathbf{Q}$ to those in $\mathscr{A}$ when the action is due to **P**, or in $\mathscr{B}$ when the action is due to **Q**, while allowing unrestricted communication between **P** and **Q**. This operator is imported from [19] where it was originally introduced.
- The result function: The expression $\mathsf{res}_\alpha(\mathbf{P})$ has the same type as the channel $\alpha$; it allows the process *P* to compute until a value **v** can be produced on $\alpha$ 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 $\xrightarrow{\surd v}$ of (Ferreira *et al.* [13]).

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 the same type. The expression

$$(\mathsf{res}_\alpha(\alpha![\lambda().\mathbf{M}]\ \mathbf{nil} + \alpha![\lambda().\mathbf{N}]\ \mathbf{nil}))()$$

behaves in much the same way as $\mathbf{M} \oplus \mathbf{N}$, modulo some extra reductions. Here, and in the sequel, we use $\lambda Y.\mathbf{M}$ rather than $\mu.\mathbf{M}$ when $X$ does not occur free in **M** and we write $\lambda().\mathbf{M}$ when the variable is of type unit.

This nondeterminism at the level of data also means that, in any reasonable semantics, the $\eta$-rule will not be valid. One would expect the expressions **M** and $\lambda X.\mathbf{M}X$ to be behaviorally equivalent, for any functional expression **P**. However, in HOCCS these expressions in addition to being applied to arguments can also be transmitted between processes. So one can easily construct a context $C[\cdot]$ of the form

$$\alpha![\cdot]\ \mathbf{nil}\ {}_{\mathscr{A}}|_{\mathscr{B}}\ \alpha?\lambda Y.\mathbf{M}$$

which distinguishes between these expressions. For example, for $\mathbf{P} = \lambda X.1 \oplus \lambda X.2$, if **M** is

$$\text{if } Y() = 1 \text{ then (if } Y() = 2 \text{ then } \beta![0]\ \mathbf{N} \text{ else } \mathbf{nil}) \text{ else } \mathbf{nil}$$

then $C[\lambda X.\mathbf{P}X]$ can produce a value on the channel $\beta$ whereas for $C[\mathbf{P}]$ this is impossible. Note that this problem already occurs in the sub-language consisting of $\lambda$-expressions, on the assumption that we have an internal choice operator, as communication is partially implemented using $\beta$-reduction.

*Context and Testing Preorders*

The form of behavioral preorder we use is a variation on the *may* testing of [18] and the testing of Boudol [6].

We use the standard *weak* versions of the operational relations; $\stackrel{\varepsilon}{\Longrightarrow}$ is the reflexive transitive closure of silent reduction $\rightarrow$ and $\stackrel{\alpha}{\Longrightarrow}$, where $\alpha$ is $\alpha!$ or $\alpha?$, is defined by $\stackrel{\alpha}{\Longrightarrow} = \stackrel{\varepsilon}{\Longrightarrow}\stackrel{\alpha}{\longrightarrow}\stackrel{\varepsilon}{\Longrightarrow}$.

The first preorder is based on direct observations of process expressions. For a closed process expression $\mathbf{P}$ we write $\mathbf{P}\Downarrow\alpha$ if $\mathbf{P}\stackrel{\alpha}{\Longrightarrow}$; i.e., if $\mathbf{P}$ can send or receive some value on $\alpha$. This leads to an observational preorder between closed process terms: $\mathbf{P} \prec \mathbf{Q}$ if for every $\alpha$ (of the form $\alpha?$ or $\alpha!$), $\mathbf{P}\Downarrow\alpha$ implies $\mathbf{Q}\Downarrow\alpha$.

DEFINITION 3.1 (May-Testing Preorder). The may-testing preorder $\sqsubseteq_{\mathscr{T}}$ is defined by $H \rhd \mathbf{M} \sqsubseteq_{\mathscr{T}} \mathbf{N}$ if

1. $H \rhd \mathbf{M} : \tau$ and $H \rhd \mathbf{N} : \tau$ for some type $\tau$,
2. for every context $C[\cdot]$ such that both $C[\mathbf{M}]$ and $C[\mathbf{N}]$ are closed process expressions $C[\mathbf{M}] \prec C[\mathbf{N}]$.

This preorder is a minor variation on the *may testing* preorder of [18]; in the original definition only contexts of a very restricted form were allowed in the testing of processes. But it will follow from our results that this restriction is unimportant. With simple process calculi this preorder can be alternatively described in terms of inclusion of traces. In the language we consider here higher-order traces would have to be considered but it is not clear to us what an appropriate and technically useful and manageable notion of higher-order traces should be. We do not know of any relevant work in this direction.

An alternative form of preorder, more directly in the spirit of the contextual testing of Morris [25], can also be defined. This is based on observations performed on the functional expressions of the language; in fact we restrict observations to be performed on expressions of type bool. First the *possible to converge predicate* $\Downarrow$ is generalized to closed terms of transmittable value-type by letting $\mathbf{M}\Downarrow\mathbf{v}$ if $\mathbf{M}$ can evaluate to the value $\mathbf{v}$, i.e., $\mathbf{M} \stackrel{\varepsilon}{\Longrightarrow} \mathbf{v}$. For expressions $\mathbf{M}$, $\mathbf{N}$ of some transmittable type $\sigma$ define $\mathbf{M} \prec \mathbf{N}$ iff for all values $\mathbf{v}$, $\mathbf{M}\Downarrow\mathbf{v}$ implies $\mathbf{N}\Downarrow\mathbf{v}$.

DEFINITION 3.2 (Context Preorder). $H \rhd \mathbf{M} \sqsubseteq_{\mathscr{C}} \mathbf{N}$ if

1. $H \rhd \mathbf{M} : \tau$ and $H \rhd \mathbf{M} : \tau$ for some type $\tau$,
2. for every context $C[\cdot]$ such that both $C[\mathbf{M}]$ and $C[\mathbf{N}]$ are closed expressions of type bool $C[\mathbf{M}] \prec C[\mathbf{N}]$.

There are natural variations on this preorder, for example allowing observations of integer type; however, this is unimportant. We can show that these preorders coincide.

PROPOSITION 3.3. *$H \rhd \mathbf{M} \sqsubseteq_{\mathscr{C}} \mathbf{N}$ implies $H \rhd \mathbf{M} \sqsubseteq_{\mathscr{T}} \mathbf{N}$.*

*Proof.* Suppose $C[\mathbf{M}]$, $C[\mathbf{N}]$ are both closed process expressions. There are two cases.

- $C[\mathbf{M}] \overset{\alpha!}{\Longrightarrow}$. Then $(\lambda X.\mathbf{tt})\,\mathsf{res}_\alpha(C[\mathbf{M}]) \Downarrow \mathbf{tt}$. From the hypothesis $(\lambda X.\mathbf{tt})\,\mathsf{res}_\alpha(C[\mathbf{N}]) \Downarrow \mathbf{tt}$, which can only be possible if $C[\mathbf{N}] \overset{\alpha!}{\Longrightarrow}$.
- $C[\mathbf{M}] \overset{\alpha?}{\Longrightarrow}$. In this case we use the context $(\lambda X.\mathbf{tt})\mathsf{res}_\beta(C[\,\cdot\,]\,|_{\varnothing}\,|_{\{\beta\}}\,\alpha!\,[\mathbf{v}]\,\beta!\,[\mathbf{tt}]\,\mathbf{nil})$. ∎

PROPOSITION 3.4. *$H \rhd \mathbf{M} \sqsubseteq_{\mathscr{T}} \mathbf{N}$ implies $H \rhd \mathbf{M} \sqsubseteq_{\mathscr{C}} \mathbf{N}$*

*Proof.* Suppose $C[\mathbf{M}]$, $C[\mathbf{N}]$ are both closed expressions of type bool and suppose $C[\mathbf{M}] \Downarrow \mathbf{v}$ for some boolean value $\mathbf{v}$. Let $D[\,\cdot\,]$ be the context

$$(\alpha?\lambda X.\text{if } x = \mathbf{v} \text{ then } \beta!\mathbf{v} \text{ else } \mathbf{nil})\,\{\beta\}\,|_{\varnothing}\,\alpha!\,[\,C[\,\cdot\,]\,]\,\mathbf{nil}.$$

Then $D[\mathbf{M}] \overset{\beta}{\Longrightarrow}$ and by the hypothesis we have $D[\mathbf{N}] \overset{\beta}{\Longrightarrow}$. By the construction of the context $D[\,\cdot\,]$ this can only happen if $C[\mathbf{N}] \Downarrow \mathbf{v}$. ∎

## 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 the unit type which we choose to be the discrete cpo's $\mathbb{N}$, $\mathbb{B}$, 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 $\mathscr{P}_H(\,\cdot\,)$ 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 Fig. 1.

$$\begin{array}{lll}
[\![\mathsf{int}]\!] = & \mathbf{D}_{\mathsf{int}} & = \mathbb{N} \\
[\![\mathsf{bool}]\!] = & \mathbf{D}_{\mathsf{bool}} & = \mathbb{B} \\
[\![\mathsf{unit}]\!] = & \mathbf{D}_{\mathsf{unit}} & = \mathbf{U} \\
[\![\sigma \to \sigma']\!] = & \mathbf{D}_{\sigma \to \sigma'} & = [\mathbf{D}_\sigma \to T(\mathbf{D}_{\sigma'})] \\
[\![\sigma \to \mathsf{proc}]\!] = & \mathbf{D}_{\sigma \to \mathsf{proc}} & = [\mathbf{D}_\sigma \to L(\mathbf{D}_{\mathsf{proc}})] \\
 & \mathbf{C}^\alpha & = T(\mathbf{D}_{\sigma(\alpha)})^s \otimes \mathbf{D}_{\mathsf{proc}} \\
 & \mathbf{F}^\alpha & = [\mathbf{D}_{\sigma(\alpha)} \to L(\mathbf{D}_{\mathsf{proc}})] \\
[\![\mathsf{proc}]\!] = & \mathbf{D}_{\mathsf{proc}} & = \prod_{\alpha \in \mathcal{N}} (\mathbf{C}^\alpha \times \mathbf{F}^\alpha) \\
 & & \\
 & L(\mathbf{D}) & = \mathbf{D}_\perp \\
 & T(\mathbf{D}) & = L\mathcal{P}_H(\mathbf{D}) \cong \mathcal{P}_H(L\mathbf{D})
\end{array}$$

**FIG. 1.** A domain equation.

THEOREM 4.1. *The domain equation has an initial solution in the category of bounded complete algebraic dcpo's with continuous morphisms such that* $\mathbf{D}_{\mathsf{proc}}$ *and* $\mathbf{D}_{\sigma\to\tau}$ *for all* $\sigma, \tau \in \mathsf{Type}$ *are prime algebraic lattices.*

*Proof.* Fix a bounded-complete algebraic dcpo $\mathbf{A}_0$. Then for any prime algebraic lattice $\mathbf{L}$, the continuous function space $[\mathbf{A}_0 \to \mathbf{L}]$ is a prime algebraic lattice. The complete lattice structure is induced by that of $\mathbf{L}$ and the primes of the function space are the step functions $c \to p$ with $c$ compact in $\mathbf{A}_0$ and $p$ prime in $\mathbf{L}$. Recall that the step function $c \to p$ is defined by

$$c \to p(x) = \begin{cases} p & \text{if } \ c \leqslant x \\ \bot & \text{otherwise.} \end{cases}$$

The bottom element is the function $\lambda d \in \mathbf{A}_0.\, \bot$. That the functor $F_0(X) = [\mathbf{A}_0 \to X]$ is continuous is easily seen by a local continuity argument.

Note also that, if $\mathscr{N}$ is a countable (perhaps infinite) set and $(F^\alpha)_{\alpha\in\mathscr{N}}$ is an $\mathscr{N}$-indexed family of continuous functors, then the functor $F = \prod_{\alpha\in\mathscr{N}} F^\alpha$ is also continuous. Indeed, if $\langle (\mathbf{D}_n)_{n\in\omega}, (f_{mn})_{m\leqslant n\in\omega}\rangle$ is a chain of embedding/projection pairs (where the projection determined by $f_{mn}$ is $f^R_{mn}$) on pointed dcpo's and $\Delta = (\mathbf{D}, (\rho_n)_{n\in\omega})$ is its canonical universal cone, then by functoriality of $F$, $F\Delta$ is a commuting cone. Suppose $(\mathbf{E}, (\xi_m)_{m\in\omega})$ is another commuting cone over the diagram $\langle F(\mathbf{D}_n)_{n\in\omega}, F(f_{mn})_{m\leqslant n\in\omega}\rangle$. The projections $\pi^\alpha_m : F(\mathbf{D}_m) \twoheadrightarrow F^\alpha(\mathbf{D}_m)$ are associated to embeddings $j^\alpha_m : F^\alpha(\mathbf{D}_m) \to F(\mathbf{D}_m)$ sending an element to an infinite sequence that is bottom everywhere except possibly for the $\alpha$th-coordinate. This results in a commuting cone $(\mathbf{E}, (\xi^\alpha_m)_{m\in\omega})$ over the diagram $\langle F^\alpha(\mathbf{D}_m)_{m\in\omega}, F^\alpha(f_{mn})_{m\leqslant n\in\omega}\rangle$. Continuity of the $F^\alpha$ implies 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 $\theta = \prod_{\alpha\in\mathscr{N}} \theta^\alpha : F(\mathbf{D}) \to \mathbf{E}$ 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 $T(X) = \mathscr{P}_H(X)_\bot \cong \mathscr{P}_H(X_\bot)$, then $T$ is a continuous functor sending a bc algebraic dcpo to a prime algebraic lattice whose primes are of the form $\bot$, $\{\!|c_\bot|\!\}$ (or, equivalently, $\{\!|\bot|\!\}$, $\{\!|c_\bot|\!\}$) with $c$ compact.

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

Existence of an initial solution in the category of dcpo's follows by continuity of the functors, discussed above. The ground domains are bc algebraic dcpo's and this category is closed under all constructions used so that the initial solution lies in the category of bc algebraic dcpo's with continuous morphisms. In fact, we can regard this equation as parametric on the choice of bc algebraic dcpo's for the ground domains to be solved for higher types and for the process type. By the discussion above the subcategory of prime algebraic lattices is closed under the constructions involved and hence the solution in $\mathbf{D}_{\sigma\to\tau}$ and $\mathbf{D}_{\mathsf{proc}}$ yields prime algebraic lattices. ∎

*Compact and Prime Elements*

We now briefly make some observations on the prime and compact elements of our model that will be useful in the sequel.

In the continuous function space $[\mathbf{A} \to \mathbf{L}]$ where $\mathbf{A}$ is only assumed to be algebraic and $\mathbf{L}$ is a prime algebraic lattice, the prime elements are the step functions $c \to p$ where $c$ is compact in $\mathbf{A}$ and $p$ is a compact prime of $\mathbf{L}$. Hence the primes of the function spaces $[\mathbf{D}_\sigma \to T(\mathbf{D}_{\sigma'})]$, where $T$ is the monad $T(\mathbf{D}) = \mathscr{P}_H(\mathbf{D}_\perp) \cong \mathscr{P}_H(\mathbf{D})_\perp$, take the form $c \to \{\!|\perp|\!\}$, the bottom element of the space, for any compact $c$, and $c \to \{\!|k_\perp|\!\}$ with $k$ compact in $\mathbf{D}_{\sigma'}$ (perhaps $k = \perp \in \mathbf{D}_{\sigma'}$). Similarly, the primes of the continuous function space $[\mathbf{D}_\sigma \to L(\mathbf{D}_{\mathsf{proc}})]$, where $L$ is lifting, are the step functions of the form $c \to \perp$, the bottom element of the space, for any compact $c$ of $\mathbf{D}_\sigma$, and $c \to \pi_\perp$ where $\pi$ is a compact prime of $\mathbf{D}_{\mathsf{proc}}$. The primes of $\mathbf{D}_{\mathsf{proc}}$ are $\omega$-sequences with $(\perp, \perp)$ everywhere except possibly for one index $\alpha$ where a prime of the Cartesian product $\mathbf{C}^\alpha \times \mathbf{F}^\alpha$ occurs. This prime has one of the forms $(p, \perp)$ or $(\perp, q)$.

To have a better representation of the primes $\pi$ of $\mathbf{D}_{\mathsf{proc}}$ we first define the functions

$$\alpha_{out}\colon T(\mathbf{D}_{\sigma(\alpha)}) \,{}^s\!\otimes \mathbf{D}_{\mathsf{proc}} \to \mathbf{D}_{\mathsf{proc}} \qquad \alpha_{in}\colon [\mathbf{D}_{\sigma(\alpha)} \to L(\mathbf{D}_{\mathsf{proc}})] \to \mathbf{D}_{\mathsf{proc}},$$

where we let $\pi_\beta$ be the projection of $\mathbf{D}_{\mathsf{proc}}$ to the $\beta$th-coordinate:

$$\pi_\beta \alpha_{out}(d) = \begin{cases} (\perp, \perp) & \text{if} \quad \beta \neq \alpha \\ (d, \perp) & \text{otherwise} \end{cases}$$

$$\pi_\beta \alpha_{in}(f) = \begin{cases} (\perp, \perp) & \text{if} \quad \beta \neq \alpha \\ (\perp, f) & \text{otherwise.} \end{cases}$$

LEMMA 4.2. *The functions $\alpha_{in}$ and $\alpha_{out}$ are strict and linear.*

*Proof.* Immediate from the definition of $\alpha_{in}, \alpha_{out}$ and of joins in a product domain. ∎

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 $A^\tau_{\mathscr{K}\mathscr{P}}, A^\tau_{\mathscr{K}}$ by induction on the type $\tau$.

DEFINITION 4.3. The sets $A^\tau_{\mathscr{K}\mathscr{P}}, A^\tau_{\mathscr{K}}$ are the least sets such that

1. $A^{\mathsf{int}}_{\mathscr{K}\mathscr{P}} = \mathbb{N} = A^{\mathsf{int}}_{\mathscr{K}}$ and similarly for the types bool, unit
2. For any type $\tau$ other than the ground types int, bool, unit, the sets $A^\tau_{\mathscr{K}} \supseteq A^\tau_{\mathscr{K}\mathscr{P}}$ consist of finite formal joins $c = p_1 \vee \cdots \vee p_s$ with $s \geqslant 1$ and $p_i \in A^\tau_{\mathscr{K}\mathscr{P}}$
3. $A^{\sigma \to \sigma'}_{\mathscr{K}\mathscr{P}}$ consists of a fresh bottom element $\perp$ and formal step functions $c \to k$ where $c \in A^\sigma_{\mathscr{K}}$ and $k \in A^{\sigma'}_{\mathscr{K}}$
4. $A^{\sigma \to \mathsf{proc}}_{\mathscr{K}\mathscr{P}}$ consists of a fresh bottom element $\perp$ and formal step functions $c \to \pi$ where $c \in A^\sigma_{\mathscr{K}}$ and $\pi \in A^{\mathsf{proc}}_{\mathscr{K}\mathscr{P}}$

5. $A^{\mathsf{proc}}_{\mathscr{K}\mathscr{P}}$ consists of formal elements of the form

- $\perp_{\mathsf{proc}}$
- $\alpha_{out}(c \otimes \pi)$ where $c \in A^{\sigma(\alpha)}_{\mathscr{K}}$ and $\pi \in A^{\mathsf{proc}}_{\mathscr{K}\mathscr{P}}$
- $\alpha_{in}(c \to \pi)$ where $c \in A^{\sigma(\alpha)}_{\mathscr{K}}$ and $\pi \in A^{\mathsf{proc}}_{\mathscr{K}\mathscr{P}}$.

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

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

1. $\leqslant_{\mathsf{int}}, \leqslant_{\mathsf{bool}}$ and $\leqslant_{\mathsf{unit}}$ 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 $\perp$ elements become the least elements in the ordering
3. $c \to k \leqslant_{\sigma \to \tau} c' \to k'$ iff $c' \leqslant_\sigma c$ and $k \leqslant_\tau k'$
4. $\alpha_{out}(c \otimes \pi) \leqslant_{\mathsf{proc}} \alpha_{out}(c' \otimes \pi')$ provided $c \leqslant_{\sigma(\alpha)} c'$ and $\pi \leqslant_{\mathsf{proc}} \pi'$
5. $c \to \pi \leqslant_{\sigma \to \mathsf{proc}} c' \to \pi'$ iff $c' \leqslant_\sigma c$ and $\pi \leqslant_{\mathsf{proc}} \pi'$
6. $\alpha_{in}(c \to \pi) \leqslant \alpha_{in}(c' \to \pi')$ iff $c \to \pi \leqslant_{\sigma(\alpha) \to \mathsf{proc}} c' \to \pi'$

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

THEOREM 4.5. *Let* $\mathbf{D}'_\tau = Idl(A^{\tau}_{\mathscr{K}})$ be defined as the ideal completions of the partial orders $(A^{\tau}_{\mathscr{K}}, \leqslant_\tau)$. 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 $c \in \mathscr{K}(\mathbf{D}_\sigma)$ and $P$ a prime in $T(\mathbf{D}_{\sigma'})$, i.e., an element of the form $\{\!|\perp|\!\}$ or $\{\!|k_\perp|\!\}$ with $k \in \mathscr{K}(\mathbf{D}_{\sigma'})$. Similarly, the actual primes in the domain of processes are infinite sequences of pairs with $(\perp, \perp)$ 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 $(\{\!|c_\perp|\!\} \otimes \pi, \perp)$ or $(\perp, c \to \pi_\perp)$ occurs. ∎

*Interpretation*

Let $\mathbf{Exp}^{\mathscr{H}}_{\tau}$ 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 a standard compositional pattern. The domain of the interpretation function $[\![ \cdot ]\!]$ is thus the set of derivable sequents $H \rhd \mathbf{M} : \tau$, while

**TABLE 3**

**Interpretation Function**

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

its codomain is $T(\mathbf{D}_\sigma)$ or $\mathbf{P}$ accordingly as $\tau = \sigma$ or $\tau = \mathsf{proc}$. For convenience, we define an interpretation function $\mathscr{V}[\![\cdot]\!]$ defined on derivable sequents $H \rhd \mathbf{v} : \sigma$, where $\mathbf{v}$ is a value term, and whose codomain is the domain of values of type $\sigma$, i.e., $\mathbf{D}_\sigma$. Thus $[\![H \rhd \mathbf{v} : \sigma]\!]$ is always obtained by composition with the unit $\eta$ of the monad $T$: $[\![H \rhd \mathbf{v} : \sigma]\!] = \eta \circ \mathscr{V}[\![H \rhd \mathbf{v} : \sigma]\!]$.

The map $\mathbf{apply}^T$ is defined as $\mathbf{ext}^{(2)}(\mathbf{apply})$ where $\mathbf{ext}^{(2)}_{A,B,C}: (\mathbf{A} \times \mathbf{B} \to T\mathbf{C}) \to (T\mathbf{A} \times T\mathbf{B} \to T\mathbf{C})$ is the standard extension map of the monad $T$. Essentially the same definition can be given for the case of functions into $\mathsf{proc}$ and we use $\mathbf{apply}^T$ for both cases, as discussed in Section 2. EQ and $\mathrm{COND}_\tau$ are the natural equality and conditional morphisms. The map $\mathsf{res}_\alpha$: $\mathbf{D}_{\mathsf{proc}} \to T(\mathbf{D}_{\sigma(\alpha)})$ 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 map $\mathbf{curry}_{A,B,C}: Hom(A \times B, C) \to Hom(A, B \to C)$ is the standard isomorphism of Hom-sets associated to the Cartesian closed structures: $\mathbf{curry}(f)\, a = \lambda b \in B.\, f(a, b)$. We typically omit the subscripts $A, B, C$.

The interpretation $\mathrm{PAR}_{\mathscr{A},\mathscr{B}}$ of the parallel operator is somewhat more complex. Its definition on primes is given in Table 4 where $\pi \mid_{\mathscr{A}\mathscr{B}} \pi'$ 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 [19] and it is essentially an interleaved implementation of the parallel operator in the model. 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 $\alpha_{out}$ and $\alpha_{in}$, previously defined, are used to interpret input and output on $\alpha$. For input processes we use the natural linear strict extension of the map $\alpha_{in}$: $[\mathbf{D}_{\sigma(\alpha)} \to L(\mathbf{D}_{\mathsf{proc}})] \to \mathbf{D}_{\mathsf{proc}}$ to a map $\alpha_{in}^T$: $T(\mathbf{D}_{\sigma(\alpha)} \to L(\mathbf{D}_{\mathsf{proc}})) \to \mathbf{D}_{\mathsf{proc}}$, setting $\alpha_{in}^T(\{\!|\perp|\!\}) = \perp$, $\alpha_{in}^T(\{\!|c_\perp|\!\}) = \alpha_{in}(c)$.

A set-theoretic interpretation can be obtained from the interpretation in Table 3 in the standard way, in terms of maps $\rho \in \mathbf{Env}^{\mathscr{H}}$, the set of $\mathscr{H}$-environments. Specifically, environments are restricted to the maps assigning *values* to variables. They

**TABLE 4**

**The Parallel Operator**

| $\pi$ | $\pi'$ | $\pi \mathbin{\mathscr{A}\vert\mathscr{B}} \pi'$ | Condition |
|---|---|---|---|
| $\perp$ | $\perp$ | $\perp$ | |
| $\perp$ | $\alpha_{out}(c \otimes \pi_1)$ | $\alpha_{out}(c \otimes \pi'_1)$ | $\pi'_1 \leqslant \perp \mathbin{\mathscr{A}\vert\mathscr{B}} \pi_1$, $\alpha \in \mathscr{B}$ |
| $\perp$ | $\alpha_{in}(c \to \pi_1)$ | $\alpha_{in}(c \to \pi'_1)$ | $\pi'_1 \leqslant \perp \mathbin{\mathscr{A}\vert\mathscr{B}} \pi_1$, $\alpha \in \mathscr{B}$ |
| $\alpha_{in}(c \to \pi_1)$ | $\beta_{out}(k \otimes \pi_2)$ | $\alpha_{in}(c \to \pi'_1)$ | $\pi'_1 \leqslant \pi_1 \mathbin{\mathscr{A}\vert\mathscr{B}} \pi'$, $\alpha \in \mathscr{A}$ |
| | | $\beta_{out}(k \otimes \pi'_2)$ | $\pi'_2 \leqslant \pi \mathbin{\mathscr{A}\vert\mathscr{B}} \pi_2$, $\beta \in \mathscr{B}$ |
| | | any $\pi_0 \leqslant \perp \mathbin{\mathscr{A}\vert\mathscr{B}} \pi_2$ | $\alpha = \beta$ |
| | | any $\pi_0 \leqslant \pi_1 \mathbin{\mathscr{A}\vert\mathscr{B}} \pi_2$ | $\alpha = \beta$ and $k \leqslant c$ |
| $\alpha_{in}(c \to \pi_1)$ | $\beta_{in}(k \to \pi_2)$ | $\alpha_{in}(c \to \pi'_1)$ | $\pi'_1 \leqslant \pi_1 \mathbin{\mathscr{A}\vert\mathscr{B}} \pi'$, $\alpha \in \mathscr{A}$ |
| | | $\beta_{in}(k \to \pi'_2)$ | $\pi'_2 \leqslant \pi \mathbin{\mathscr{A}\vert\mathscr{B}} \pi_2$, $\beta \in \mathscr{B}$ |
| $\alpha_{out}(c \otimes \pi_1)$ | $\beta_{out}(k \otimes \pi_2)$ | $\alpha_{out}(c \otimes \pi'_1)$ | $\pi'_1 \leqslant \pi_1 \mathbin{\mathscr{A}\vert\mathscr{B}} \pi'$, $\alpha \in \mathscr{A}$ |
| | | $\beta_{out}(k \otimes \pi'_2)$ | $\pi'_2 \leqslant \pi \mathbin{\mathscr{A}\vert\mathscr{B}} \pi_2$, $\beta \in \mathscr{B}$ |

are finite maps $\rho$ such that there is a context $H \in \mathscr{H}$, $H = X_1 : \sigma_1, ..., X_s : \sigma_s$, such that $dom(\rho) = \{X_1, ..., X_s\}$ and $\rho(X_i) \in \mathbf{D}_{\sigma_i}$. We then say that $\rho$ is an $H$-environment and we write $\rho \in \mathbf{Env}^{\mathscr{H}}$ for the set of all $H$-environments with $H \in \mathscr{H}$. For reader's convenience and to facilitate later reference we list some of the clauses in the set-theoretic interpretation. We typically omit the typing context in the sequel since the interpretation function does not depend on any particular context $H$.

1. $\mathscr{V}[\![\lambda X.\mathbf{M}]\!]_\rho = \lambda v \in \mathbf{D}_\sigma.\, [\![\mathbf{M}]\!]_{\rho[v/X]} \in \mathbf{D}_{\sigma \to \tau}$
2. $\mathscr{V}[\![\mu X \lambda Y.\mathbf{M}]\!]_\rho = \mathbf{lfp}(\lambda f \in \mathbf{D}_{\sigma \to \tau}.\, \mathscr{V}[\![\lambda Y.\mathbf{M}]\!]_{\rho[f/X]})$
3. Variables $X$ assigned type $\sigma$ by a context $H$ are interpreted, given an $H$-environment $\rho$, in a similar fashion, namely $\mathscr{V}[\![X^\sigma]\!]_\rho = \rho(X) \in \mathbf{D}_\sigma$
4. $[\![\mathbf{v}]\!]_\rho = \{\!|(\mathscr{V}[\![\mathbf{v}]\!]_\rho)_\perp|\!\}$ and $[\![X^\sigma]\!]_\rho = \{\!|\rho(X)_\perp|\!\}$
5. $[\![\mathbf{M} = \mathbf{N}]\!]_\rho = \begin{cases} [\![\mathbf{tt}]\!]_\rho & \text{if } \exists \ell\, [\![\ell]\!]_\rho \leqslant [\![\mathbf{M}]\!]_\rho \cap [\![\mathbf{N}]\!]_\rho \\ \perp & \text{otherwise} \end{cases}$

The intuition in this case is that $\mathbf{M} = \mathbf{N}$ is to be interpreted as true provided that $\mathbf{M}$, $\mathbf{N}$ can converge to a common ground value.

6. $[\![\mathbf{M} \oplus \mathbf{N}]\!]_\rho = [\![\mathbf{M}]\!]_\rho \cup [\![\mathbf{N}]\!]_\rho \in T(\mathbf{D}_\sigma)$
7. $[\![\mathbf{P} \oplus \mathbf{Q}]\!]_\rho = [\![\mathbf{P}]\!]_\rho \vee [\![\mathbf{Q}]\!]_\rho = [\![\mathbf{P} + \mathbf{Q}]\!]_\rho \in \mathbf{D}_{\mathbf{proc}}$

EXAMPLE 1. The interpretation specified above identifies the three processes $\alpha?.\lambda X.(\mathbf{P} + \mathbf{Q})$, $\alpha?(\lambda X.\mathbf{P} \oplus \lambda X.\mathbf{Q})$, and $(\alpha?\lambda X.\mathbf{P}) \oplus (\alpha?\lambda X.\mathbf{Q})$.

By definition, $[\![\alpha?.\lambda X.(\mathbf{P} + \mathbf{Q})]\!]_\rho = \alpha^T_{in}(\{\!|\mathscr{V}[\![\lambda X.(\mathbf{P} + \mathbf{Q})]\!]|\!\})$ and again by definition this is the same as $\alpha_{in}(\lambda v \in \mathbf{D}_\sigma.\, [\![\mathbf{P}]\!]_{\rho[v/X]} \vee [\![\mathbf{Q}]\!]_{\rho[v/X]})$. The interpretation of the second process is $\alpha^T_{in}([\![\lambda X.\mathbf{P}]\!]_\rho \cup [\![\lambda X.\mathbf{Q}]\!]_\rho)$ which is the same as $\alpha^T_{in}(\{\!|\lambda v \in \mathbf{D}_\sigma.\, [\![\mathbf{P}]\!]_{\rho[v/X]},\ \lambda v \in \mathbf{D}_\sigma.\, [\![\mathbf{Q}]\!]_{\rho[v/X]}|\!\})$ which given the definition of $\alpha^T_{in}$ is

$\alpha_{in}(\lambda v \in \mathbf{D}_\sigma . [\![\mathbf{P}]\!]_{\rho[v/X]}) \vee \alpha_{in}(\lambda v \in \mathbf{D}_\sigma . [\![\mathbf{Q}]\!]_{\rho[v/X]})$ where the join is taken in $\mathbf{D}_{\mathbf{proc}}$. Since $\alpha_{in}$ is linear, this is identical to

$$\alpha_{in}(\lambda v \in \mathbf{D}_\sigma . [\![\mathbf{P}]\!]_{\rho[v/X]} \vee \lambda v \in \mathbf{D}_\sigma . [\![\mathbf{Q}]\!]_{\rho[v/X]}),$$

where the main join inside the parentheses is now taken in the complete lattice $[\mathbf{D}_\sigma \to L(\mathbf{D}_{\mathbf{proc}})]$. The join inside the parentheses is identical to $\lambda v \in \mathbf{D}_\sigma . ([\![\mathbf{P}]\!]_{\rho[v/X]} \vee [\![\mathbf{Q}]\!]_{\rho[v/X]})$, since join is defined pointwise on function spaces. Hence the interpretation of the first two processes is the same. Similar small calculation shows that the interpretation of the third process is the same as for the other two.

Some properties of the interpretation that will be frequently used are discussed below.

A *substitution* is a map from variables to expressions and we use $\mathbf{M}s$ to denote the result of applying the substitution $s$ to the expression $\mathbf{M}$. When $s$ is a simple substitution, such as the identity everywhere except on a variable $X$, we will continue to use the notation $\mathbf{M}[s(X)/X]$. We call $s$ a *value-substitution* provide $s(X)$ is a value for every $X$ in its domain. We call $s$ an $H$-substitution provided that if $H \rhd X''\sigma$, then $H \rhd s(X) : \sigma$. Since the interpretation of the language follows a standard compositional schema we have the following:

PROPOSITION 4.6 (Substitution Lemma). *For every environment $\rho$ and value-substitution $s$, $[\![\mathbf{M}s]\!]_\rho = [\![\mathbf{M}]\!]_{\rho[\mathscr{V}[\![s(X)]\!]_\rho/X]}$.*

For ease of reference we also list the following.

LEMMA 4.7. *The following hold*:

1. $\mathscr{V}[\![\mu X\lambda Y.\mathbf{M}]\!]_\rho = \mathscr{V}[\![\lambda Y.\mathbf{M}[{}^{\mu X\lambda Y.\mathbf{M}}/_X]]\!]_\rho$
2. $\mathbf{apply}(\mathscr{V}[\![\mu X\lambda Y.\mathbf{M}]\!]_\rho, v) = [\![\mathbf{M}[{}^{\mu X\lambda Y.\mathbf{M}}/_X]]\!]_{\rho[v/Y]}$.

*Proof.* [Sketch] For the first, from the (set-theoretic) interpretation of the fixpoint term we get $\mathscr{V}[\![\mu X\lambda Y.\mathbf{M}]\!]_\rho = \mathscr{V}[\![\lambda Y.\mathbf{M}]\!]_{\rho[\mathscr{V}[\![\mu X\lambda Y.\mathbf{M}]\!]_\rho/X]}$. Then use the Substitution Lemma.

For the second, use the first part and consult the set-theoretic interpretation of $\lambda$-abstracts. ∎

It follows from the Substitution Lemma in combination with Lemma 4.7 and from the definition of the parallel operator $\mathscr{A}|_{\mathscr{B}}$ that

PROPOSITION 4.8. 1. (**Call-by-value β-Reduction**) *For any value* $\mathbf{v}$,

$$[\![(\mu X\lambda Y.\mathbf{M})\ \mathbf{v}]\!]_\rho = [\![\mathbf{M}[\mathbf{v}/Y, \mu X\lambda Y.\mathbf{M}/X]]\!]_\rho.$$

2. (**Communication**) *For any value* $\mathbf{v}$ *and process expressions* $\mathbf{P}$, $\mathbf{Q}$

$$[\![\alpha?\mu X\lambda Y.\mathbf{Q}\ {}_{\mathscr{A}}|_{\mathscr{B}}\ \alpha![\mathbf{v}]\ \mathbf{P}]\!]_\rho \geqslant [\![(\mu X\lambda Y.\mathbf{Q})\ \mathbf{v}\ {}_{\mathscr{A}}|_{\mathscr{B}}\ \mathbf{P}]\!]_\rho.$$

*Proof.* The first follows from the observation that

$$\begin{aligned}\mathbf{apply}^T(\llbracket \mu X\lambda Y.\mathbf{M}\rrbracket_\rho, \llbracket \mathrm{v}\rrbracket_\rho) &= \mathbf{apply}^T(\{\!|\mathscr{V}\llbracket \mu X\lambda Y.\mathbf{M}\rrbracket_\rho|\!\}, \{\!|\mathscr{V}\llbracket \mathrm{v}\rrbracket_\rho|\!\}) \\ &= \mathbf{apply}(\mathscr{V}\llbracket \mu X\lambda Y.\mathbf{M}\rrbracket_\rho, \mathscr{V}\llbracket \mathrm{v}\rrbracket_\rho)\end{aligned}$$

and then by combining the Substitution Lemma with Lemma 4.7.

For the second, to show that

$$\llbracket (\mu X\lambda Y.\mathbf{Q})\ \mathrm{v}\rrbracket_\rho \ {}_{\mathscr{A}}|_{\mathscr{B}}\ \llbracket \mathbf{P}\rrbracket_\rho \leqslant \llbracket \alpha?\mu X\lambda Y.\mathbf{Q}\rrbracket_\rho \ {}_{\mathscr{A}}|_{\mathscr{B}}\ \llbracket \alpha!\ [\mathrm{v}]\ \mathbf{P}\rrbracket_\rho$$

let $k$ be a compact element $k \leqslant \mathscr{V}\llbracket \mathrm{v}\rrbracket_\rho$, $c \to \pi_1$ a prime below $\mathscr{V}\llbracket \mu X\lambda Y.\mathbf{Q}\rrbracket_\rho$ and $\pi_2$ a prime below $\llbracket \mathbf{P}\rrbracket_\rho$. Then $\alpha_{in}(c \to \pi_1) \ {}_{\mathscr{A}}|_{\mathscr{B}}\ \alpha_{out}(k \otimes \pi_2) \leqslant \llbracket \alpha?\mu X\lambda Y.\mathbf{Q}\rrbracket_\rho \ {}_{\mathscr{A}}|_{\mathscr{B}}\ \llbracket \alpha!\ [\mathrm{v}]\ \mathbf{P}\rrbracket_\rho$. Consulting Table 4 it follows that $(c \to \pi_1)(k) \ {}_{\mathscr{A}}|_{\mathscr{B}}\ \pi_2 \leqslant \alpha_{in}(c \to \pi_1) \ {}_{\mathscr{A}}|_{\mathscr{B}}\ \alpha_{out}(k \otimes \pi_2)$. ∎

We list here a property of the interpretation in relation to the operational semantics that will be used in the course of the adequacy and definability proofs.

LEMMA 4.9 (Monotonicity). *Assume all terms appearing below are closed.*

1. *For any* $\mathbf{M}$, $\mathbf{N}$, $\mathbf{M} \overset{\varepsilon}{\Longrightarrow} \mathbf{N}$ *implies* $\llbracket \mathbf{N}\rrbracket_\rho \leqslant \llbracket \mathbf{M}\rrbracket_\rho$
2. *For any process* $\mathbf{P}$
   - *If* $\mathbf{P} \overset{\alpha?}{\Longrightarrow} \mathbf{M}$, *then* $\bot < \llbracket \alpha?\mathbf{M}\rrbracket_\rho \leqslant \llbracket \mathbf{P}\rrbracket_\rho$
   - *If* $\mathbf{P} \overset{\alpha!}{\Longrightarrow} [\mathrm{v}]\ \mathbf{Q}$, *then* $\bot < \llbracket \alpha!\ [\mathrm{v}]\ \mathbf{Q}\rrbracket_\rho \leqslant \llbracket \mathbf{P}\rrbracket_\rho$.

*Proof.* Both (1) and (2) are proven by transition induction. The only interesting cases are $\beta$-reduction and communication which we have dealt with in Lemma 4.8. ∎

The denotational model induces the following preorder on expressions.

DEFINITION 4.10 (Denotational Preorder). For any terms $\mathbf{M}$, $\mathbf{N}$ let $H \rhd \mathbf{M} \sqsubseteq_{\mathscr{D}} \mathbf{N}$ iff $\llbracket H \rhd \mathbf{M} : \tau\rrbracket \leqslant \llbracket H \rhd \mathbf{N} : \tau\rrbracket$ for some type $\tau$. For closed expressions this is abbreviated to $\mathbf{M} \sqsubseteq_{\mathscr{D}} \mathbf{N}$.

Our aim is to show that the model is *fully abstract* for both of the operational preorders that we have defined (context and may-testing preorders), namely that

$$H \rhd \mathbf{M} \sqsubseteq_{\mathscr{C}} \mathbf{N} \text{ iff } \mathrm{H} \rhd \mathbf{M} \sqsubseteq_{\mathscr{T}} \mathbf{N} \text{ iff } \mathrm{H} \rhd \mathbf{M} \sqsubseteq_{\mathscr{D}} \mathbf{N}.$$

As a first step toward this result we prove an *adequacy* result:

$$\text{For closed expressions, } \mathbf{M}\Downarrow \text{ if and only if } \llbracket \mathbf{M}\rrbracket \neq \bot.$$

One direction of this result is straightforward, as it simply says that the operational semantics is reflected correctly in the model. To prove the converse we introduce a typed modal language $\mathscr{L}$ of properties $\zeta$ of program terms and a related program logic. We interpret the program logic denotationally and prove a completeness result, which intuitively states that the program logic characterizes the denotational model. Using this result we then prove the nontrivial part of the Computational Adequacy theorem: If $\llbracket \mathbf{M}\rrbracket \neq \bot$, then $\mathbf{M}\Downarrow$. The logical language, its operational

semantics and the associated proof systems are detailed in 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.

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 HOCCS. 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 $\mathscr{T}, \mathscr{V}, \mathscr{P}$ be the sets of closed expressions, value expressions, and process expressions. The operational semantics determines a many-sorted transition system $\langle \mathscr{T}, \mathscr{V}, \mathscr{P}, \Downarrow, \stackrel{\alpha!}{\Longrightarrow}, \stackrel{\alpha?}{\Longrightarrow} \rangle$ 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 $\{\&, \sqcap, \to, \Diamond, \langle\!\langle \alpha! \rangle\!\rangle [\,], \langle\!\langle \alpha? \rangle\!\rangle\}$ where only $\Diamond$ and $\langle\!\langle \alpha? \rangle\!\rangle$ are unary while every other connective is binary:

$$S \in \mathbf{AtFmla} := S_n\ (n \in N) \mid S_{\mathbf{tt}} \mid S_{\mathbf{ff}} \mid S_{()} \mid \omega_{\mathsf{proc}} \mid \omega_\sigma^L \mid \omega_\sigma^T \mid$$

$$\zeta \in \mathbf{Fmla} := S \mid \zeta \,\&\, \zeta \mid \zeta \sqcap \zeta \mid \zeta \to \zeta \mid \Diamond\zeta \mid \langle\!\langle \alpha! \rangle\!\rangle [\zeta]\, \zeta \mid \langle\!\langle \alpha? \rangle\!\rangle\, \zeta.$$

DEFINITION 5.1. The languages $\mathscr{L}_\sigma(\mathscr{V})$, $\sigma \in \mathsf{VType}$, $\mathscr{L}_\tau(\mathscr{T})$, $\tau \in \mathsf{Type}$ are the least sets satisfying the recursive conditions in Table 5.

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.

Properties (other than $\omega$) of computations of some type $\sigma$ are always either of the form $\Diamond A$ or conjunctions $\zeta \sqcap \eta$. Distinct conjunction operations for properties of values as opposed to properties 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

**TABLE 5**

**A Modal Language of Properties**

- $\mathscr{L}_{\mathsf{int}}(\mathscr{V}) = \{S_0, S_1, \ldots, S_n, \ldots\}$ and similarly for $\mathsf{bool}$, $\mathsf{unit}$
- $\omega_{\mathsf{proc}} \in \mathscr{L}_{\mathsf{proc}}$, $\omega_\sigma^L \in \mathscr{L}_{\sigma \to \mathsf{proc}}(\mathscr{V})$ and $\omega_\sigma^T \in \mathscr{L}_\sigma(\mathscr{T})$
- $A \in \mathscr{L}_\sigma(\mathscr{V})$ implies $\Diamond A \in \mathscr{L}_\sigma(\mathscr{T})$
- — $\zeta, \eta \in \mathscr{L}_\sigma(\mathscr{T})$ implies $\zeta \sqcap \eta \in \mathscr{L}_\sigma(\mathscr{T})$
  — $A, B \in \mathscr{L}_\sigma(\mathscr{V})$ implies $A \,\&\, B \in \mathscr{L}_\sigma(\mathscr{V})$, provided $\sigma \notin \mathsf{GType}$
  — $\varphi, \psi \in \mathscr{L}_{\mathsf{proc}}$ implies $\varphi \,\&\, \psi \in \mathscr{L}_{\mathsf{proc}}(\mathscr{T})$
- $A \in \mathscr{L}_\sigma(\mathscr{V})$, $\zeta \in \mathscr{L}_\tau(\mathscr{T})$ implies $A \to \zeta \in \mathscr{L}_{\sigma \to \tau}(\mathscr{V})$
- $A \in \mathscr{L}_{\sigma(\alpha)}(\mathscr{V})$, $\psi \in \mathscr{L}_{\mathsf{proc}}(\mathscr{T})$ implies $\langle\!\langle \alpha! \rangle\!\rangle [A]\, \psi \in \mathscr{L}_{\mathsf{proc}}(\mathscr{T})$
- $A \in \mathscr{L}_{\sigma(\alpha)}(\mathscr{V})$, $\psi \in \mathscr{L}_{\mathsf{proc}}(\mathscr{T})$ implies $\langle\!\langle \alpha? \rangle\!\rangle\, A \to \psi \in \mathscr{L}_{\mathsf{proc}}(\mathscr{T})$.

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 $\{\!|c \vee k|\!\} \leqslant d$ and $\{\!|c, k|\!\} = \{\!|c|\!\} \uplus \{\!|k|\!\} \leqslant d$. Note, however, that the domains for ground values do not have a lattice structure and so the recursive clause for & needs to be restricted appropriately; hence the side-condition that $\sigma \notin \mathsf{GType}$.

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

LEMMA 5.2. $\mathbf{M} \Rightarrow \mathbf{v}$ *and* $\mathbf{v} \mathrel{|\!\approx}_\sigma A$ *implies* $\mathbf{M} \models_\sigma \Diamond A$. *Furthermore*, $\mathbf{M} \Rightarrow \mathbf{N}$ *and* $\mathbf{N} \models_\tau \zeta \in \mathscr{L}_\tau(\mathscr{T})$ *implies* $\mathbf{M} \models_\tau \zeta$.

*Proof.* By structural induction on the sentence $\zeta \in \mathscr{L}_\tau(\mathscr{T})$. ▮

*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 $A, B \in \mathscr{L}_\sigma(\mathscr{V})$ and $\zeta, \eta \in \mathscr{L}_\tau(\mathscr{T})$

- $A \mathrel{|\!\approx}_\sigma B$ iff for all values $\mathbf{v} \in \mathscr{V}$, $\mathbf{v} \mathrel{|\!\approx}_\sigma A$ implies $\mathbf{v} \mathrel{|\!\approx}_\sigma B$
- $\zeta \models_\tau \eta$ 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, $\mathscr{G}_E$, in Table 7.

LEMMA 5.3. *The system* $\mathscr{G}_E$ *is sound in the operational semantics. In other words,* $A \vdash_\sigma B$ *implies* $A \mathrel{|\!\approx}_\sigma B$, *and* $\zeta \vdash_\tau \eta$ *implies* $\zeta \models_\tau \eta$.

*Proof.* By the usual induction on length of proof. ▮

**TABLE 6**

**A Two-Sorted, Typed Satisfaction Relation**

| | | | | |
|---|---|---|---|---|
| $\mathbf{v}$ | $\mathrel{|\!\approx}_{\sigma_G}$ | $S_\ell$ | iff | $\mathbf{v} = \ell$ |
| $\mathbf{v}$ | $\mathrel{|\!\approx}_\sigma$ | $A \,\&\, B$ | iff | $\mathbf{v} \mathrel{|\!\approx}_\sigma A$ and $\mathbf{v} \mathrel{|\!\approx}_\sigma B$ |
| $\mathbf{v}$ | $\mathrel{|\!\approx}_{\sigma\to\tau}$ | $A \to \zeta$ | iff | $\forall \mathbf{u} \in \mathscr{V}$ ($\mathbf{u} \mathrel{|\!\approx}_\sigma A$ implies $\mathbf{vu} \models_\tau \zeta$) |
| $\mathbf{v}$ | $\mathrel{|\!\approx}_{\sigma\to\mathsf{proc}}$ | $\omega_\sigma^L$ | | always |
| $\mathbf{M}$ | $\models_\sigma$ | $\omega_\sigma^T$ | | always |
| $\mathbf{M}$ | $\models_\sigma$ | $\Diamond A$ | iff | $\exists \mathbf{v} \in \mathscr{V}$ ($\mathbf{M} \Downarrow \mathbf{v}$ and $\mathbf{v} \mathrel{|\!\approx}_\sigma A$) |
| $\mathbf{M}$ | $\models_\sigma$ | $\zeta \sqcap \eta$ | iff | $\mathbf{M} \models_\sigma \zeta$ and $\mathbf{M} \models_\sigma \eta$ |
| $\mathbf{P}$ | $\models_{\mathsf{proc}}$ | $\omega_{\mathsf{proc}}$ | | always |
| $\mathbf{P}$ | $\models_{\mathsf{proc}}$ | $\varphi \,\&\, \psi$ | iff | $\mathbf{P} \models_{\mathsf{proc}} \varphi$ and $\mathbf{P} \models_{\mathsf{proc}} \psi$ |
| $\mathbf{P}$ | $\models_{\mathsf{proc}}$ | $\langle\!\langle \alpha! \rangle\!\rangle [A] \psi$ | iff | $\exists \mathbf{v} \in \mathscr{V}\ \exists \mathbf{Q} \in \mathscr{P}$ ($\mathbf{P} \overset{\alpha!}{\Longrightarrow} [\mathbf{v}]\,\mathbf{Q}$, $\mathbf{v} \mathrel{|\!\approx}_{\sigma(\alpha)} A$ and $\mathbf{Q} \models_{\mathsf{proc}} \psi$) |
| $\mathbf{P}$ | $\models_{\mathsf{proc}}$ | $\langle\!\langle \alpha? \rangle\!\rangle A \to \psi$ | iff | $\exists \mathbf{v} \in \mathscr{V}$ ($\mathbf{P} \overset{\alpha?}{\Longrightarrow} \mathbf{v}$ and $\mathbf{v} \mathrel{|\!\approx}_{\sigma\to\mathsf{proc}} A \to \psi$) |

**TABLE 7**

**A Proof System $\mathscr{G}_E$ for Semantic Entailment**

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

To relate the logic with the model we first interpret sentences as compact elements. The interpretation maps $\mathscr{V}[\![\cdot]\!]$ 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**

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

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

PROPOSITION 5.4. 1. *The interpretation of a sentence* $\zeta$ *is a compact element of the respective domain. Specifically,* $\mathscr{V}[\![A]\!] \in \mathscr{K}(\mathbf{D}_\sigma)$ *for* $A \in \mathscr{L}_\sigma(\mathscr{V})$, $[\![\zeta]\!] \in \mathscr{K}(\mathscr{L}_\sigma(\mathscr{T}))$ *for* $\zeta \in \mathscr{L}_\sigma(\mathscr{V}) \cup \mathscr{L}_\sigma(\mathscr{T})$ and $[\![\varphi]\!] \in \mathscr{K}(\mathbf{P})$ *for* $\varphi \in \mathscr{L}_{\mathsf{proc}}(\mathscr{T})$. *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* $\zeta$ *such that* $C = [\![\zeta]\!]$ *or* $C = \mathscr{V}[\![\zeta]\!]$, *as appropriate.*

2. (*Completeness of the System* $\mathscr{G}_E$ *in the Denotational Semantics*)

- $A \vdash_\sigma B$ *iff* $\mathscr{V}[\![B]\!] \leqslant \mathscr{V}[\![A]\!]$ *in* $L(\mathbf{D}_\sigma)$
- $\zeta \vdash_\sigma \eta$ *iff* $[\![\eta]\!] \leqslant [\![\zeta]\!]$ *in* $T(\mathbf{D}_\sigma)$, *and*
- $\varphi \vdash_{\mathsf{proc}} \psi$ *iff* $[\![\psi]\!] \leqslant [\![\varphi]\!]$ *in* $\mathbf{D}_{\mathsf{proc}}$.

*Proof.* For part 1, the proof is by induction on the structure of a sentence $\zeta$ for one direction and on the structure of a compact element $C$ for the converse. For example, if $\zeta \in \mathscr{L}_{\mathsf{proc}}(\mathscr{T})$, then each of the cases $\omega_{\mathsf{proc}}$, $\langle\!\langle \alpha! \rangle\!\rangle [A] \psi$, $\langle\!\langle \alpha? \rangle\!\rangle (A \to \psi)$ or $\varphi \,\&\, \psi$ are immediate by induction. Conversely, given, for example, a prime of the form $\alpha_{out}(c \otimes \pi)$ we may assume by induction that $c = \mathscr{V}[\![A]\!]$ and $\pi = [\![\psi]\!]$. Then $\alpha_{out}(c \otimes \pi) = [\![\langle\!\langle \alpha! \rangle\!\rangle [A] \psi]\!]$.

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

For the second part, the soundness direction is proven by induction on the length of the proof of, e.g., $\zeta \vdash_\sigma \eta$. The completeness direction is proven by structural induction. For example, suppose that $[\![\varphi]\!] \leqslant [\![\psi]\!]$. If $[\![\varphi]\!] = \bot$, i.e. $\varphi = \omega_{\mathsf{proc}}$ then the claim holds because $\psi \vdash_{\mathsf{proc}} \omega_{\mathsf{proc}}$ is an axiom. If $\psi$ is of the form $\langle\!\langle \alpha! \rangle\!\rangle [A] \vartheta$, then the ordering of prime and compact elements implies that $[\![\psi]\!]$ must be of the form $\alpha_{out}(c \otimes \pi)$, or a join involving this prime. Hence $\varphi$ must be a conjunction with a conjunct of the form $\langle\!\langle \alpha! \rangle\!\rangle [B] \chi$. In fact, given the conjunction rules in the proof system we may just consider the case where $\psi = \langle\!\langle \alpha! \rangle\!\rangle [B] \chi$. The ordering on primes implies that $\mathscr{V}[\![A]\!] \leqslant \mathscr{V}[\![B]\!]$ and $[\![\vartheta]\!] \leqslant [\![\chi]\!]$. By induction $B \vdash A$ and $\chi \vdash \vartheta$. Using the relevant rule it follows that $\langle\!\langle \alpha! \rangle\!\rangle [B] \vartheta \vdash \langle\!\langle \alpha! \rangle\!\rangle [A] \chi$. The case of input is similar. If $\psi = \psi_1 \,\&\, \psi_2$ use induction and the rule for conjunction. ∎

The structure of our modal language does not directly reflect the fact that the transition system is equiped with parallel operators $_{\mathscr{A}}|_{\mathscr{B}}$. Instead we interpret $_{\mathscr{A}}|_{\mathscr{B}}$ as an operator on formulae; $\varphi\, _{\mathscr{A}}|_{\mathscr{B}} \psi$ can be considered as a shorthand for a finite conjunction of formula in $\mathscr{L}_{\mathsf{proc}}(\mathscr{T})$. The components of this conjunction are defined by induction on the structure of $\varphi$ and $\psi$, as specified below. Whenever any of the conditions mentioned in the definition fails to hold we let the defined sentence be equal to $\omega$.

- $\omega\, _{\mathscr{A}}|_{\mathscr{B}}\, \omega = \omega$
- $\omega\, _{\mathscr{A}}|_{\mathscr{B}}\, \langle\!\langle \alpha! \rangle\!\rangle [A] \varphi = \langle\!\langle \alpha! \rangle\!\rangle [A] \psi_1 \,\&\, \cdots \,\&\, \langle\!\langle \alpha! \rangle\!\rangle [A] \psi_s$

provided $\omega \,{}_{\mathscr{A}}|_{\mathscr{B}}\, \varphi = \psi_1 \& \cdots \& \psi_s$ and $\alpha \in \mathscr{B}$

- $\omega \,{}_{\mathscr{A}}|_{\mathscr{B}}\, \langle\!\langle \alpha? \rangle\!\rangle\, A \to \varphi = \omega \,{}_{\mathscr{A}}|_{\mathscr{B}}\, \langle\!\langle \alpha? \rangle\!\rangle\, A \to \psi_1 \& \cdots \& \omega \,{}_{\mathscr{A}}|_{\mathscr{B}}\, \langle\!\langle \alpha? \rangle\!\rangle\, A \to \psi_s$

provided $\alpha \in \mathscr{B}$ and $\omega \,{}_{\mathscr{A}}|_{\mathscr{B}}\, \varphi = \psi_1 \& \cdots \& \psi_s$

The cases of the form $- \,{}_{\mathscr{A}}|_{\mathscr{B}}\, \omega$ are defined in a symmetric way.

- $\langle\!\langle \alpha? \rangle\!\rangle\, A \to \varphi \,{}_{\mathscr{A}}|_{\mathscr{B}}\, \langle\!\langle \beta? \rangle\!\rangle\, B \to \psi = \phi_1 \& \phi_2$, where

1. $\phi_1 = \langle\!\langle \alpha? \rangle\!\rangle\, A \to \varphi_1 \& \cdots \& \langle\!\langle \alpha? \rangle\!\rangle\, A \to \varphi_s$

provided $\alpha \in \mathscr{A}$ and $\varphi \,{}_{\mathscr{A}}|_{\mathscr{B}}\, \langle\!\langle \beta? \rangle\!\rangle\, B \to \psi = \varphi_1 \& \cdots \& \varphi_s$

2. $\phi_2 = \langle\!\langle \beta? \rangle\!\rangle\, B \to \psi_1 \& \cdots \& \langle\!\langle \beta? \rangle\!\rangle\, B \to \psi_r$

provided $\langle\!\langle \alpha? \rangle\!\rangle\, A \to \varphi \,{}_{\mathscr{A}}|_{\mathscr{B}}\, \psi = \psi_1 \& \cdots \& \psi_r$ and $\beta \in \mathscr{B}$.

- $\langle\!\langle \alpha! \rangle\!\rangle [A]\, \varphi \,{}_{\mathscr{A}}|_{\mathscr{B}}\, \langle\!\langle \beta! \rangle\!\rangle [B]\, \psi = \phi_1 \& \phi_2$ where

1. $\phi_1 = \langle\!\langle \alpha! \rangle\!\rangle [A]\, \varphi_1 \& \cdots \& \langle\!\langle \alpha! \rangle\!\rangle [A]\, \varphi_s$

provided $\alpha \in \mathscr{A}$ and $\varphi \,{}_{\mathscr{A}}|_{\mathscr{B}}\, \langle\!\langle B! \rangle\!\rangle\, \psi = \varphi_1 \& \cdots \& \varphi_s$

2. $\phi_2 = \langle\!\langle \beta! \rangle\!\rangle [B]\, \psi_1 \& \cdots \& \langle\!\langle \beta! \rangle\!\rangle [B]\, \psi_r$

provided $\beta \in \mathscr{B}$ and $\langle\!\langle \alpha! \rangle\!\rangle [A]\, \varphi \,{}_{\mathscr{A}}|_{\mathscr{B}}\, \psi = \psi_1 \& \cdots \& \psi_r$.

The remaining case involves a "communication" capability:

- $\langle\!\langle \alpha? \rangle\!\rangle\, A \to \varphi \,{}_{\mathscr{A}}|_{\mathscr{B}}\, \langle\!\langle \beta! \rangle\!\rangle [B]\, \psi = \phi_1 \& \cdots \& \phi_4$,

where

1. $\phi_1 = \langle\!\langle \alpha? \rangle\!\rangle\, A \to \varphi_1 \& \cdots \& \langle\!\langle \alpha? \rangle\!\rangle\, A \to \varphi_s$ if $\alpha \in \mathscr{A}$ and $\varphi \,{}_{\mathscr{A}}|_{\mathscr{B}}\, \langle\!\langle \beta! \rangle\!\rangle [B]\, \psi = \varphi_1 \& \cdots \& \varphi_s$

2. $\phi_2 = \langle\!\langle \beta! \rangle\!\rangle [B]\, \psi_1 \& \cdots \& \langle\!\langle \beta! \rangle\!\rangle [B]\, \psi_r$ if $\beta \in \mathscr{B}$ and $\langle\!\langle \alpha? \rangle\!\rangle\, A \to \varphi \,{}_{\mathscr{A}}|_{\mathscr{B}}\, \psi = \psi_1 \& \cdots \& \psi_r$

3. $\phi_3 = \omega \,{}_{\mathscr{A}}|_{\mathscr{B}}\, \psi$ provided $\alpha = \beta$

4. $\phi_4 = \varphi \,{}_{\mathscr{A}}|_{\mathscr{B}}\, \psi$ provided $\alpha = \beta$ and $A \vdash\!\sim_\sigma B$ is derivable in $\mathscr{G}_E$.

The symmetric case $\langle\!\langle \beta! \rangle\!\rangle [B]\, \psi \,{}_{\mathscr{A}}|_{\mathscr{B}}\, \langle\!\langle \alpha? \rangle\!\rangle\, A \to \varphi$ is defined in an analogous manner.

The reader will have no doubt noticed the similarity between the definition of $\varphi \,{}_{\mathscr{A}}|_{\mathscr{B}}\, \psi$ and the definition of the parallel operator on the primes of the domain $\mathbf{D}_{\mathbf{proc}}$ given in Table 4. The proof of the following lemma follows from a close comparison of the two definitions.

LEMMA 5.5. *Let* $\varphi, \psi \in \mathscr{L}_{\mathbf{proc}}(\mathscr{T})$. *Then* $[\![\varphi \,{}_{\mathscr{A}}|_{\mathscr{B}}\, \psi]\!] = [\![\varphi]\!] \,{}_{\mathscr{A}}|_{\mathscr{B}}\, [\![\psi]\!]$.

*A Proof System for Satisfiability—Program Logic*

So far, we provided a sound proof system with judgements of the form $\zeta \vdash_\tau \eta$ (resp. $A \vdash\!\sim_\sigma B$), axiomatizing the relation of semantic entailment $\zeta \models_\tau \eta$ (resp. $A \approx_\sigma B$). The latter is defined by $\mathbf{M} \models_\tau \zeta$ implies $\mathbf{M} \models_\tau \eta$, where $\mathbf{M}$ is any closed

term of type $\tau$ and similarly for $A \lessapprox_\sigma B$, restricting to closed value terms. It is useful to devise a proof system with judgments of the form $\mathbf{M} \vdash_\tau \zeta$ and $\mathbf{v} \vdash_\sigma A$ meaning, intuitively, that it is provable in the logic system that the term $\mathbf{M}$, resp. $\mathbf{v}$, has the indicated property. If, moreover, the proof system is complete then it can be used to calculate behavioral properties of complex program terms from properties of simpler terms. We will provide such a system in the sequel. In fact, we first extend satisfaction to open terms, given assumptions $\Gamma$ about properties of closed terms substituted in for the variables. Given that reduction is call-by-value we may view an *assumption* as a finite map $\Gamma$ from variables of the programming language to *value sentences* in $\mathscr{L}_\sigma(\mathscr{V})$, for $\sigma \in \mathsf{VType}$. We write $\Gamma$'s in the form $X_1 : A_1, ..., X_n : A_n$. In writing $\Gamma, X : A$ as an assumption, we presume that $X \notin dom(\Gamma)$. If $\Gamma$ is an assumption and $s$ is a closed value-substitution, then we say that $s \models \Gamma$ iff for each $X$ in the domain of $\Gamma$, $s(X) \lessapprox_\sigma \Gamma X$.

DEFINITION 5.6. $\Gamma \models^{\mathscr{O}}_\tau \mathbf{M} : \zeta$ iff for any closed value-substitution $s$, if $s \models \Gamma$ then $\mathbf{M}s \models_\tau \zeta$.

Similarly, for possibly open value-expressions, we may define $\Gamma \lessapprox^{\mathscr{O}}_\sigma \mathbf{v} : A$ iff $s \models \Gamma$ implies $\mathbf{v}s \lessapprox_\sigma A$. It is easy to see that $\Gamma \lessapprox^{\mathscr{O}}_\sigma \mathbf{v} : A$ iff $\Gamma \models^{\mathscr{O}}_\sigma \mathbf{v} : \Diamond A$ and we will typically only use the relation $\Gamma \models^{\mathscr{O}}_\tau \mathbf{M} : \zeta$.

The program logic presented in Table 9 axiomatizes this notion of satisfiability between program expressions and logical formulae. The relation of satisfiability induces a new operational preorder between language expressions, based on their ability to satisfy logical formula.

DEFINITION 5.7. $H \rhd \mathbf{M} \sqsubseteq_{\mathscr{L}} \mathbf{N}$ iff

- $H \rhd \mathbf{M} : \tau$ and $H \rhd \mathbf{M} : \tau$ for some type $\tau$
- for every assumption $\Gamma$, $\Gamma \models^{\mathscr{O}}_\tau \mathbf{M} : \zeta$ implies $\Gamma \models^{\mathscr{O}}_\tau \mathbf{N} : \zeta$.

It is worth pointing out that by the introduction and elimination rules for $\Diamond$, $\Gamma \vdash_\sigma \mathbf{v} : A$ iff $\Gamma \vdash_\sigma \mathbf{v} : \Diamond A$. We could have then opted for using only $\vdash$ but this would force us to use rules that introduce logical operators (in particular, conjunction of properties of values) underneath a modal operator.

PROPOSITION 5.8 (Soundness of $\mathscr{G}_S$ in the Operational Semantics). *The program logic is sound in the operational semantics, i.e., $\Gamma \vdash_\tau \mathbf{M} : \zeta$ implies $\Gamma \models^{\mathscr{O}}_\tau \mathbf{M} : \zeta$.*

*Proof.* The proof is by a standard induction on length of proofs which amounts to showing that the axioms are sound and if the premiss of a rule is sound so is its conclusion. All cases are straightforward. We discuss the Recursion, Application, and Result rules as examples.

For the recursion rule, suppose $s \models \Gamma$. To show that $(\mu X \lambda Y.\mathbf{M})\, s \lessapprox_{\sigma \to \tau} A \to \zeta$ let $\mathbf{v}$ be a value and assume $\mathbf{v} \lessapprox_\sigma A$. Set $s' = s[Y \mapsto \mathbf{v}]$. Since $s' \models \Gamma, Y : A$ it follows by the hypothesis that the premise of the $(\mu)$ rule is sound that $\mathbf{M}[\mu X \lambda Y.\mathbf{M}/X]\, s' \models_\tau \zeta$, i.e., $\mathbf{M}[\mu X \lambda Y.\mathbf{M}/X, \mathbf{v}/Y]\, s \models_\tau \zeta$. From the operational semantics and by the monotonicity Lemma 5.2 it follows that $(\mu X \lambda Y.\mathbf{M})\, \mathbf{v}s \models_\tau \zeta$ and hence, by definition, $(\mu X \lambda Y.\mathbf{M})\, s \lessapprox_{\sigma \to \tau} A \to \zeta$.

**TABLE 9**

**A Proof System $\mathscr{G}_S$ for Satisfiability: Program Logic**

| | | | |
|---|---|---|---|
| (Mon1) | $\dfrac{\Gamma \vdash_\sigma \mathbf{v} : A \quad A \vdash_\sigma B}{\Gamma \vdash_\sigma \mathbf{v} : B}$ | (Mon2) | $\dfrac{\Gamma \vdash_\tau \mathbf{M} : \zeta \quad \zeta \vdash_\tau \eta}{\Gamma \vdash_\tau \mathbf{M} : \eta}$ |
| (Id) | $\Gamma, X : A \vdash_\sigma X : A$ | (T1) | $\Gamma \vdash_\sigma \mathbf{M} : \omega_\sigma^T$ |
| (T2) | $\Gamma \vdash_{\sigma \to \mathsf{proc}} \mathbf{v} : \omega_\sigma^L$ | (T3) | $\Gamma \vdash_{\sigma' \to \sigma} \mathbf{v} : A \to \omega_\sigma^T$ |
| ($\mathrm{T}_\varphi$) | $\Gamma \vdash_{\mathsf{proc}} \mathbf{P} : \omega_{\mathsf{proc}}$ | ($\ell$) | $\Gamma \vdash_{\sigma_G} \ell : S_\ell \ (\ell = = \mathbf{tt}, \mathbf{ff}, (), \mathbf{n})$ |
| ($\Diamond I$) | $\dfrac{\Gamma \vdash_\sigma \mathbf{v} : A}{\Gamma \vdash_\sigma \mathbf{v} : \Diamond A}$ | ($\Diamond E$) | $\dfrac{\Gamma \vdash_\sigma \mathbf{v} : \Diamond A}{\Gamma \vdash_\sigma \mathbf{v} : A}$ |
| (B) | $\dfrac{\Gamma \vdash_{\sigma_G} \mathbf{M} : \Diamond S_\ell \quad \Gamma \vdash_{\sigma_G} \mathbf{N} : \Diamond S_\ell}{\Gamma \vdash_{\mathsf{bool}} (\mathbf{M} = \mathbf{N}) : \Diamond S_{\mathbf{tt}}}$ | ($\sqcap$) | $\dfrac{\Gamma \vdash_\sigma \mathbf{M} : \zeta \quad \Gamma \vdash_\sigma \mathbf{M} : \eta}{\Gamma \vdash_\tau \mathbf{M} : \zeta \sqcap \eta}$ |
| (&) | $\dfrac{\Gamma \vdash_\sigma \mathbf{v} : A \quad \Gamma \vdash_\sigma \mathbf{v} : B}{\Gamma \vdash_\sigma \mathbf{v} : A \,\&\, B}$ | ($\&_\varphi$) | $\dfrac{\Gamma \vdash_{\mathsf{proc}} \mathbf{P} : \varphi \quad \Gamma \vdash_{\mathsf{proc}} \mathbf{P} : \psi}{\Gamma \vdash_{\mathsf{proc}} \mathbf{P} : \varphi \,\&\, \psi}$ |
| (J1) | $\dfrac{\Gamma \vdash_\tau \mathbf{M} : \zeta}{\Gamma \vdash_\tau \mathbf{M} \oplus \mathbf{N} : \zeta}$ | (J2) | $\dfrac{\Gamma \vdash_\tau \mathbf{N} : \zeta}{\Gamma \vdash_\tau \mathbf{M} \oplus \mathbf{N} : \zeta}$ |
| (+1) | $\dfrac{\Gamma \vdash_{\mathsf{proc}} \mathbf{P} : \zeta}{\Gamma \vdash_{\mathsf{proc}} \mathbf{P} + \mathbf{Q} : \zeta}$ | (+2) | $\dfrac{\Gamma \vdash_{\mathsf{proc}} \mathbf{Q} : \zeta}{\Gamma \vdash_{\mathsf{proc}} \mathbf{P} + \mathbf{Q} : \zeta}$ |
| (App) | $\dfrac{\Gamma \vdash_{\sigma \to \tau} \mathbf{M} : \Diamond(A \to \zeta) \quad \Gamma \vdash_\sigma \mathbf{N} : \Diamond A}{\Gamma \vdash_\tau \mathbf{M}\mathbf{N} : \zeta}$ | | |
| ($\mu$) | $\dfrac{\Gamma, Y : A \vdash_\tau \mathbf{M}[^{\mu X \lambda Y . \mathbf{M}}/_X] : \zeta}{\Gamma \vdash_{\sigma \to \tau} \mu X \lambda Y . \mathbf{M} : A \to \zeta}$ | (res) | $\dfrac{\Gamma \vdash_{\mathsf{proc}} \mathbf{P} : \langle\!\langle \alpha! \rangle\!\rangle [A]\, \phi}{\Gamma \vdash_{\sigma(\alpha)} res_\alpha(\mathbf{P}) : \Diamond A}$ |
| (Cond1) | $\dfrac{\Gamma \vdash_{\mathsf{bool}} \mathbf{B} : \Diamond S_{\mathbf{tt}} \quad \Gamma \vdash_\tau \mathbf{M} : \zeta}{\Gamma \vdash_\tau \text{if } \mathbf{B} \text{ then } \mathbf{M} \text{ else } \mathbf{N} : \zeta}$ | (Cond2) | $\dfrac{\Gamma \vdash_{\mathsf{bool}} \mathbf{B} : \Diamond S_{\mathbf{ff}} \quad \Gamma \vdash_\tau \mathbf{N} : \zeta}{\Gamma \vdash_\tau \text{if } \mathbf{B} \text{ then } \mathbf{M} \text{ else } \mathbf{N} : \zeta}$ |
| ($\alpha?$) | $\dfrac{\Gamma \vdash_{\sigma \to \mathsf{proc}} \mathbf{M} : \Diamond A}{\Gamma \vdash_{\mathsf{proc}} \alpha?\mathbf{M} : \langle\!\langle \alpha? \rangle\!\rangle A}$ | ($\alpha!$) | $\dfrac{\Gamma \vdash_\sigma \mathbf{M} : \Diamond A \quad \Gamma \vdash_{\mathsf{proc}} \mathbf{P} : \varphi}{\Gamma \vdash_{\mathsf{proc}} \alpha!\,[\mathbf{M}]\,\mathbf{P} : \langle\!\langle \alpha! \rangle\!\rangle [A]\, \varphi}$ |
| (Par) | $\dfrac{\Gamma \vdash_{\mathsf{proc}} \mathbf{P} : \varphi \quad \Gamma \vdash_{\mathsf{proc}} \mathbf{Q} : \psi \quad \varphi \mathscr{A}\vert_{\mathscr{B}} \psi \vdash \chi}{\Gamma \vdash_{\mathsf{proc}} \mathbf{P} \mathscr{A}\vert_{\mathscr{B}} \mathbf{Q} : \chi}$ | | |

For the application rule (App), given $s \models \Gamma$ and given soundness of the premisses it follows that there exist values $\mathbf{v}$, $\mathbf{u}$ such that $\mathbf{M}s \Downarrow \mathbf{v}$, $\mathbf{N}s \Downarrow \mathbf{u}$ and $\mathbf{v} \mathrel{\dot{\approx}}_{\sigma \to \tau} A \to \zeta$, $\mathbf{u} \mathrel{\dot{\approx}}_\sigma A$. Then $\mathbf{MN} \Rightarrow \mathbf{vu} \models_\tau \zeta$ and so, using the monotonicity Lemma 5.1 it follows $\mathbf{MN} \models_\tau \zeta$.

Soundness of the premise of the ($\mathsf{res}_\alpha$) rule means that $\mathbf{P}s \models_{\mathsf{proc}} \langle\!\langle \alpha! \rangle\!\rangle [A]\, \psi$. This implies that there is a value $\mathbf{v}$ and a process $\mathbf{Q}$ such that $\mathbf{P}s \overset{\alpha!}{\Longrightarrow} [\mathbf{v}]\, \mathbf{Q}$ with $\mathbf{v} \mathrel{\dot{\approx}}_\sigma A$. Then $\mathsf{res}_\alpha(\mathbf{P}) \Rightarrow \mathbf{v}$ and $\mathbf{v} \mathrel{\dot{\approx}}_\sigma A$ implies that $\mathsf{res}_\alpha \models_\sigma \sigma \Diamond A$. ∎

*Completeness of the Program Logic in the Denotational Semantics*

The denotational semantics induces a relation between closed program expressions and logical formulae; we write $\models^D \mathbf{M} : \zeta$ if $[\![\zeta]\!] \leqslant [\![\mathbf{M}]\!]$. Similarly, $\mathrel{\dot{\approx}}^D \mathbf{v} : A$ iff $\mathscr{V}[\![A]\!] \leqslant \mathscr{V}[\![\mathbf{v}]\!]$. In order to relate this to the program logic we need to generalize it to arbitrary program expressions.

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

DEFINITION 5.9. Let $\Gamma \models^{D}_{\tau} \mathbf{M} : \zeta$ if and only if for every environment $\rho$, if $\rho \models^{D} \Gamma$, then $[\![\zeta]\!] \leqslant [\![\mathbf{M}]\!]_{\rho}$.

Similarly, we may let $\Gamma \approx^{D}_{\sigma} \mathbf{v} : A$ iff for every environment $\rho$, if $\rho \models^{D} \Gamma$, then $\mathscr{V}[\![A]\!] \leqslant \mathscr{V}[\![\mathbf{v}]\!]$. Since the latter inequality is equivalent to $\{\!|\mathscr{V}[\![A]\!]|\!\} \leqslant \{\!|\mathscr{V}[\![\mathbf{v}]\!]|\!\}$, i.e., $[\![\Diamond A]\!] \leqslant [\![\mathbf{v}]\!]$, 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 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 5.10 (Soundness-Completeness of $\mathscr{G}_S$ 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 $\Gamma$, define the environment $\rho_{\Gamma}$ by $\rho(X) = [\![\Gamma(X)]\!]$. Then it is immediate that $\Gamma \models^{D} \mathbf{M} : \zeta$ iff $[\![\zeta]\!] \leqslant [\![\mathbf{M}]\!]_{\rho_{\Gamma}}$. So the soundness and completeness claim is equivalent to the claim $\Gamma \vdash_{\tau} \mathbf{M} : \zeta$ if and only if $[\![\zeta]\!] \leqslant [\![\mathbf{M}]\!]_{\rho_{\Gamma}}$, 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 $\mathscr{G}_E$ for semantic entailment, Proposition 5.4. It is mostly straightforward and we only do a few cases as an example.

For the monotonicity rule (Mon), if $\rho \models \Gamma$ then $\mathscr{V}[\![A]\!] \leqslant \mathscr{V}[\![\mathbf{v}]\!]_{\rho}$ and, by Proposition 5.4, $\mathscr{V}[\![B]\!] \leqslant \mathscr{V}[\![A]\!]$. Then $\mathscr{V}[\![B]\!] \leqslant \mathscr{V}[\![\mathbf{v}]\!]_{\rho}$.

Proof of soundness for the conjunction rules is straightforward but it illustrates the need for different conjunction logical operators and justifies our introduction of two semantic maps $\mathscr{V}[\![\cdot]\!]$ and $[\![\cdot]\!]$. For the conjunction connective & for value-sentences, the hypothesis $\rho \models \Gamma$ implies that $\mathscr{V}[\![A]\!] \vee \mathscr{V}[\![B]\!] = \mathscr{V}[\![A \,\&\, B]\!] \leqslant \mathscr{V}[\![\mathbf{v}]\!]_{\rho}$. For soundness of the rule $(\sqcap)$ for conjunctive properties of computations the hypothesis $\rho \models \Gamma$ implies that $[\![\zeta]\!] \uplus [\![\eta]\!] \leqslant [\![\mathbf{M}]\!]_{\rho}$. The left-hand side of the inequality is $[\![\zeta \sqcap \eta]\!] \leqslant [\![\mathbf{M}]\!]_{\rho}$ and this shows soundness of the rule.

For the recursion rule $(\mu)$, assume $\rho \models \Gamma$ and set $\rho' := \rho[\, Y := \mathscr{V}[\![A]\!]\,]$ so that $\rho' \models \Gamma, Y : A$. Then $[\![\zeta]\!] \leqslant [\![\mathbf{M}[\mu X \lambda Y.\mathbf{M}/X]]\!]_{\rho[\mathscr{V}[\![A]\!]/Y]}$, assuming by induction soundness of the premise. The right-hand side of the inequality is equal to $\mathbf{apply}(\mathscr{V}[\![\mu X \lambda Y.\mathbf{M}]\!]_{\rho}, \mathscr{V}[\![A]\!])$, using the second part of Lemma 4.7. Then $\mathscr{V}[\![A]\!] \to [\![\zeta]\!] \leqslant \mathscr{V}[\![\mu X \lambda Y.\mathbf{M}]\!]_{\rho}$, and so

$$[\![A \to \zeta]\!] = \{\!|\mathscr{V}[\![A]\!] \to [\![\zeta]\!]|\!\} \leqslant \{\!|\mathscr{V}[\![\mu X \lambda Y.\mathbf{M}]\!]_{\rho}|\!\} = [\![\mu X \lambda Y.\mathbf{M}]\!]_{\rho}.$$

For the rule ($\alpha!$) and given $\rho \models \Gamma$ we may assume $[\![\Diamond A]\!] = \{\!|\mathscr{V}[\![A]\!]|\!\} \leqslant [\![\mathbf{M}]\!]_\rho$ and $[\![\varphi]\!] \leqslant [\![\mathbf{P}]\!]_\rho$. Then

$$[\![\langle\!\langle \alpha! \rangle\!\rangle [A]\, \varphi]\!] = \alpha_{out}(\{\!|\mathscr{V}[\![A]\!]|\!\} \otimes [\![\varphi]\!]) \leqslant \alpha_{out}([\![\mathbf{M}]\!]_\rho \otimes [\![\mathbf{P}]\!]_\rho) = [\![\alpha!\, [\mathbf{M}]\, \mathbf{P}]\!]_\rho$$

and this shows that ($\alpha!$) is sound.

For the rule ($\mathsf{res}_\alpha$) and given $\rho \models \Gamma$ we may assume that $\alpha_{out}(\{\!|\mathscr{V}[\![A]\!]|\!\} \otimes [\![\varphi]\!]) \leqslant [\![\mathbf{P}]\!]_\rho$. Applying the $\mathsf{res}$ function of the model to both sides of the inequality we obtain $\{\!|\mathscr{V}[\![A]\!]|\!\} \leqslant \mathsf{res}_\alpha([\![\mathbf{P}]\!]_\rho)$; hence, by definitions, $[\![\Diamond A]\!] \leqslant [\![\mathsf{res}_\alpha(\mathbf{P})]\!]_\rho$.

Soundness for the other rules is shown by similar arguments and we turn now to completeness, showing that $[\![\zeta]\!] \leqslant [\![\mathbf{M}]\!]_{\rho_\Gamma}$ implies $\Gamma \vdash \mathbf{M} : \zeta$ by structural induction on $\mathbf{M}$ and $\zeta$.

*Completeness*

If $\zeta$ is any of $\omega_\sigma^T$ or $\omega_{\mathsf{proc}}$ the claim is trivial using one of the (T) axioms. If $\zeta$ is of the form $\eta \sqcap \vartheta$ we may use the induction hypothesis on $\eta$, $\vartheta$ 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 $\langle\!\langle \alpha! \rangle\!\rangle [A]\, \psi$ or $\langle\!\langle \alpha? \rangle\!\rangle A \to \psi$. This means that $[\![\zeta]\!]$ is a prime element of the model such that $[\![\zeta]\!] \neq \bot$ and we consider now the cases for $\mathbf{M}$.

($\mathbf{M} \equiv \ell$) If $\mathbf{M}$ is a literal $\ell$ of some ground type $\sigma_G$ then $\zeta$ must be $\Diamond S_k$, for some literal $k$ of type $\sigma_G$. The hypothesis implies $k = \ell$ and the conclusion follows by the axiom ($\ell$).

($\mathbf{M} \equiv \mu X \lambda Y.\mathbf{N}$) Then $\zeta$ is of the form $\Diamond(A \to \eta)$ and the hypothesis implies that $\mathscr{V}[\![A \to \eta]\!] \leqslant \mathscr{V}[\![\mu X \lambda Y.\mathbf{N}]\!]_{\rho_\Gamma}$. Since $\mathscr{V}[\![A \to \eta]\!]$ is the step function $\mathscr{V}[\![A]\!] \to [\![\eta]\!]$ we obtain, by application to $\mathscr{V}[\![A]\!]$ on both sides, $[\![\eta]\!] \leqslant \mathbf{apply}(\mathscr{V}[\![\mu X \lambda Y.\mathbf{N}]\!]_{\rho_\Gamma}, \mathscr{V}[\![A]\!])$. By Lemma 4.7 we have

$$[\![\eta]\!] \leqslant \mathbf{apply}(\mathscr{V}[\![\mu X \lambda Y.\mathbf{N}]\!]_{\rho_\Gamma}, \mathscr{V}[\![A]\!]) = [\![\mathbf{N}[\mu X \lambda Y.\mathbf{N}/X]]\!]_{\rho_\Gamma[\mathscr{V}[\![A]\!]/Y]}$$

and then by induction $\Gamma, Y : A \vdash_\tau \mathbf{N}[\mu X \lambda Y.\mathbf{N}/X] : \eta$.

Using the rule ($\mu$) we obtain $\Gamma \vdash_{\sigma \to \tau} \mu X \lambda Y.\mathbf{N} : A \to \eta$. The $\Diamond$-introduction rule allows us to rewrite this as $\Gamma \vdash_{\sigma \to \tau} \mu X \lambda Y.\mathbf{N} : \Diamond(A \to \eta)$.

($\mathbf{M} \equiv (\mathbf{N} = \mathbf{K})$) From $\bot \neq [\![\zeta]\!] \leqslant [\![\mathbf{N} = \mathbf{K}]\!]_{\rho_\Gamma}$ and the definition of $[\![\mathbf{N} = \mathbf{K}]\!]_\rho$ it follows that $[\![\zeta]\!] \leqslant [\![\mathbf{N} = \mathbf{K}]\!]_{\rho_\Gamma} = \{\!|\mathbf{tt}|\!\}$; hence $\zeta$ is $\Diamond S_{\mathbf{tt}}$, and there is some literal $\ell$ such that $[\![\ell]\!]_{\rho_\Gamma} \leqslant [\![\mathbf{N}]\!]_{\rho_\Gamma} \cap [\![\mathbf{K}]\!]_{\rho_\Gamma}$. Hence $[\![\ell]\!]_{\rho_\Gamma} = \{\!|\ell|\!\} = \{\!|\mathscr{V}[\![S_\ell]\!]|\!\} = [\![\Diamond S_\ell]\!] \leqslant [\![\mathbf{N}]\!]_{\rho_\Gamma}$ and similarly $[\![\Diamond S_\ell]\!] \leqslant [\![\mathbf{N}]\!]_{\rho_\Gamma}$. By induction, $\Gamma \vdash_{\sigma_G} \mathbf{N} : \Diamond S_\ell$ and $\Gamma \vdash_{\sigma_G} \mathbf{K} : \Diamond S_\ell$. Then we may apply the rule ($B$) to conclude that $\Gamma \vdash_{\mathsf{bool}} \mathbf{N} = \mathbf{K} : \Diamond S_{\mathbf{tt}}$.

($\mathbf{M} \equiv \mathbf{FN}$) Assume $\bot \neq [\![\zeta]\!] \leqslant [\![\mathbf{FN}]\!]_{\rho_\Gamma} = \mathbf{apply}^T([\![\mathbf{F}]\!]_{\rho_\Gamma}, [\![\mathbf{N}]\!]_{\rho_\Gamma})$. By strictness of $\mathbf{apply}^T$ it follows that $[\![\mathbf{F}]\!]_{\rho_\Gamma} \neq \bot$ and $[\![\mathbf{N}]\!]_{\rho_\Gamma} \neq \bot$. By linearity of $\mathbf{apply}^T$, let $p \neq \{\!|\bot|\!\}$ be a nontrivial prime in $T(\mathbf{D}_\sigma)$ such that $p \leqslant [\![\mathbf{N}]\!]_{\rho_\Gamma}$ and $[\![\zeta]\!] \leqslant \mathbf{apply}^T([\![\mathbf{F}]\!]_{\rho_\Gamma}, p)$. The prime $p$ is the formal singleton of a join of primes of the domain $\mathbf{D}_\sigma$ and by logical definability of compact and prime elements, Proposition 5.4, $p = \{\!|\mathscr{V}[\![A]\!]|\!\} \leqslant [\![\mathbf{N}]\!]_{\rho_\Gamma}$

for some $A \in \mathscr{L}_\sigma(\mathscr{V})$. It follows from $[\![\zeta]\!] \leqslant \mathbf{apply}^T([\![\mathbf{F}]\!]_{\rho_\Gamma}, \{\!|\mathscr{V}[\![A]\!]|\!\})$ that $[\![A \to \zeta]\!] \leqslant [\![\mathbf{F}]\!]_{\rho_\Gamma}$. By induction we then obtain $\Gamma \vdash_{\sigma\to\tau} \mathbf{F} : \Diamond(A \to \zeta)$ and $\Gamma \vdash_\sigma \mathbf{N} : \Diamond A$. Using the application rule (App) we obtain $\Gamma \vdash_\tau \mathbf{FN} : \zeta$.

($\mathbf{M} \equiv$ if $\mathbf{B}$ then $\mathbf{N}$ else $\mathbf{N}'$) The argument is similar and uses the two rules (Cond) and (Cond′) for the conditional.

($\mathbf{M} \equiv \mathbf{N} \oplus \mathbf{N}'$ or $\mathbf{M} \equiv \mathbf{P} + \mathbf{Q}$) Use the rules $(J1)$, $(J2)$ and $(+1)$, $(+2)$, respectively.

($\mathbf{M} \equiv \mathbf{nil}$) Trivial.

($\mathbf{M} \equiv \mathsf{res}_\alpha(\mathbf{P})$) We now assume $\bot \neq [\![\zeta]\!] \leqslant \mathsf{res}_\alpha([\![\mathbf{P}]\!]_{\rho_\Gamma})$. Observe first that $\zeta \in \mathscr{L}_{\sigma(\alpha)}(\mathscr{T})$ is of the form $\Diamond B$ for some $B \in \mathscr{L}_{\sigma(\alpha)}(\mathscr{V})$. By linearity of $\mathsf{res}_\alpha$ let $\pi$ be a prime $\pi = \alpha_{out}(c \otimes \pi') \leqslant [\![\mathbf{P}]\!]_{\rho_\Gamma}$ such that

$$[\![\Diamond B]\!] = [\![\zeta]\!] \leqslant \{\!|c_\bot|\!\} = \mathsf{res}_\alpha(\alpha_{out}(c \otimes \pi')) \leqslant \mathsf{res}_\alpha([\![\mathbf{P}]\!]_{\rho_\Gamma}).$$

By logical definability of compact elements, Proposition 5.4, let $A \in \mathscr{L}_\sigma(\mathscr{V})$ be a property of values such that $\mathscr{V}[\![A]\!] = c$ and $\varphi \in \mathscr{L}_{\mathsf{proc}}(\mathscr{T})$ such that $\pi' = [\![\varphi]\!]$. Then from $[\![\Diamond B]\!] \leqslant [\![\Diamond A]\!]$ and Proposition 5.4 it follows $\Diamond A \vdash_{\sigma(\alpha)} \Diamond B$ and from $\alpha_{out}(\{\!|\mathscr{V}[\![A]\!]|\!\} \otimes [\![\varphi]\!]) \leqslant [\![\mathbf{P}]\!]_{\rho_\Gamma}$ it follows by induction that $\Gamma \vdash_{\mathsf{proc}} \mathbf{P} : \langle\!\langle \alpha! \rangle\!\rangle [A]\, \varphi$. By the $\mathsf{res}_\alpha$ rule, $\Gamma \vdash_{\sigma(\alpha)} \mathsf{res}_\alpha(\mathbf{P}) : \Diamond A$. Using the monotonicity rule it follows that $\Gamma \vdash_{\sigma(\alpha)} \mathsf{res}_\alpha(\mathbf{P}) : \Diamond B$.

($\mathbf{M} \equiv \alpha![\,\mathbf{N}\,]\,\mathbf{Q}$) If $\bot \neq [\![\varphi]\!]$ is a prime and $[\![\varphi]\!] \leqslant \alpha_{out}([\![\mathbf{M}]\!]_{\rho_\Gamma} \otimes [\![\mathbf{P}]\!]_{\rho_\Gamma})$ then $\varphi$ must be of the form $\langle\!\langle \alpha! \rangle\!\rangle [A]\, \psi$ so that $[\![\varphi]\!] = \alpha_{out}(\{\!|\mathscr{V}[\![A]\!]|\!\} \otimes [\![\psi]\!])$. By induction it follows that $\Gamma \vdash_{\sigma(\alpha)} \mathbf{M} : \Diamond A$ and $\Gamma \vdash_{\mathsf{proc}} \mathbf{P} : \psi$. Using the $(\alpha!)$ rule we obtain $\Gamma \vdash_{\mathsf{proc}} \alpha![\,\mathbf{M}\,]\,\mathbf{P} : \langle\!\langle \alpha! \rangle\!\rangle [A]\, \psi$.

($\mathbf{M} \equiv \alpha?\mathbf{N}$) Assuming $[\![\varphi]\!] \leqslant [\![\alpha?\mathbf{N}]\!]_{\rho_\Gamma}$ and since by case assumption $[\![\varphi]\!] \neq \bot$ and it is a prime it follows that $\varphi$ is of the form $\varphi \equiv \langle\!\langle \alpha? \rangle\!\rangle A \to \psi$. Recall that $\alpha_{in}([\![A \to \psi]\!]) = \alpha_{in}(\{\!|\mathscr{V}[\![A]\!] \to [\![\psi]\!]|\!\})$. The hypothesis then implies that $[\![\Diamond(A \to \psi)]\!] \leqslant [\![\mathbf{N}]\!]_{\rho_\Gamma}$ and therefore, by induction, $\Gamma \vdash_{\sigma\to\mathsf{proc}} \mathbf{N} : \Diamond(A \to \psi)$. By an application of the $(\alpha?)$ rule we obtain $\Gamma \vdash_{\mathsf{proc}} \alpha?\mathbf{N} : \langle\!\langle \alpha? \rangle\!\rangle A \to \psi$.

($\mathbf{M} \equiv \mathbf{P} \;{}_{\mathscr{A}}|_{\mathscr{B}}\; \mathbf{Q}$) Assume $[\![\zeta]\!] \leqslant [\![\mathbf{P} \;{}_{\mathscr{A}}|_{\mathscr{B}}\; \mathbf{Q}]\!]_{\rho_\Gamma} = [\![\mathbf{P}]\!]_{\rho_\Gamma} \;{}_{\mathscr{A}}|_{\mathscr{B}}\; [\![\mathbf{Q}]\!]_{\rho_\Gamma}$. By primeness of $[\![\zeta]\!]$ and linearity of the operator ${}_{\mathscr{A}}|_{\mathscr{B}}$ in the domain $\mathbf{D}_{\mathsf{proc}}$ let $[\![\zeta]\!] \leqslant \pi \;{}_{\mathscr{A}}|_{\mathscr{B}}\; \pi'$ for some primes $\pi \leqslant [\![\mathbf{P}]\!]_{\rho_\Gamma}$ and $\pi' \leqslant [\![\mathbf{Q}]\!]_{\rho_\Gamma}$. By Proposition 5.4, let $\pi = [\![\varphi]\!]$, $\pi' = [\![\psi]\!]$ so that $\pi \;{}_{\mathscr{A}}|_{\mathscr{B}}\; \pi' = [\![\varphi]\!] \;{}_{\mathscr{A}}|_{\mathscr{B}}\; [\![\psi]\!] = [\![\varphi \;{}_{\mathscr{A}}|_{\mathscr{B}}\; \psi]\!]$ where we also used Lemma 5.5. It now follows that $[\![\zeta]\!] \leqslant [\![\varphi \;{}_{\mathscr{A}}|_{\mathscr{B}}\; \psi]\!]$. Since $\varphi \;{}_{\mathscr{A}}|_{\mathscr{B}}\; \psi$ is a finite conjunction it follows that $[\![\varphi \;{}_{\mathscr{A}}|_{\mathscr{B}}\; \psi]\!]$ is a compact element. By completeness of the system $\mathscr{G}_E$ in the denotational semantic, Proposition 5.4, we obtain $\varphi \;{}_{\mathscr{A}}|_{\mathscr{B}}\; \psi \vdash_{\mathsf{proc}} \zeta$. By induction, $\Gamma \vdash_{\mathsf{proc}} \mathbf{P} : \varphi$ and $\Gamma \vdash_{\mathsf{proc}} \mathbf{Q} : \psi$. An application of the rule (Par) then yields $\Gamma \vdash_{\mathsf{proc}} \mathbf{P} \;{}_{\mathscr{A}}|_{\mathscr{B}}\; \mathbf{Q} : \zeta$.

($\mathbf{M} \equiv X$) If $\mathbf{M}$ is a variable $X$ and $[\![\zeta]\!] \leqslant [\![X]\!]_{\rho_\Gamma} = [\![\Gamma(X)]\!]$ then since assumptions only assign properties of values to variables the hypothesis is equivalent to $\mathscr{V}[\![A]\!] \leqslant \mathscr{V}[\![X]\!]_{\rho_\Gamma} = \rho_\Gamma(X)$ for some $A$ such that $\zeta = \Diamond A$. Since $X \in dom(\Gamma)$ we may set $\Gamma = \Delta, X : B$ for some $B$. By the identity axiom $\Delta, X : B \mathrel{|\!\sim}_\sigma X : B$. From $\mathscr{V}[\![A]\!] \leqslant \mathscr{V}[\![X]\!]_{\rho_\Gamma} = \mathscr{V}[\![B]\!]$ and Proposition 5.4 we also obtain $B \mathrel{|\!\sim}_\sigma A$ from which,

using the monotonicity rule, we obtain $\Gamma \vdash_\sigma X : A$. We may then use the $\Diamond$ rule to rewrite this in the form $\Gamma \vdash_\sigma X : \zeta$ since $\zeta \equiv \Diamond A$.

Hence the program logic is sound and complete for the denotational semantics. ∎

The completeness result for the program logic entails that the model is adequate. We end this section with a proof of the Adequacy theorem. It will then be used in the course of the definability argument in order to prove Corollary 6.3; the latter is necessary for the proof of full abstraction.

THEOREM 5.11 (Computational Adequacy). *For any closed expression* $\mathbf{M}$, $\mathbf{M}\Uparrow$ *iff* $[\![\mathbf{M}]\!]_\rho = \bot$.

*Proof.* Recall that if $\mathbf{M}$ is of some value type $\sigma$ then $\mathbf{M}\Downarrow$ means that $\mathbf{M}$ converges to some value $\mathbf{v}$, and if $\mathbf{M}$ is of type $\mathsf{proc}$ then it means that $\mathbf{M} \stackrel{\alpha}{\Longrightarrow}$ for some channel $\alpha$.

($\Leftarrow$) If $\mathbf{M}$ is a value then obvious. Otherwise by hypothesis $\mathbf{M}\Downarrow \mathbf{v}$ for some value $\mathbf{v}$. By Lemma 4.9 $\bot \neq [\![\mathbf{v}]\!]_\rho \leqslant [\![\mathbf{M}]\!]_\rho$. The case where $\mathbf{M}$ is of type process follows immediately from the second part of the same Lemma.

($\Rightarrow$) If $[\![\mathbf{M}]\!]_\rho \neq \bot$, then there is a prime $p$ such that $\bot \neq p \leqslant [\![\mathbf{M}]\!]_\rho$. By Proposition 5.4, $p = [\![\zeta]\!]$ for some sentence $\zeta$ and moreover this sentence must be different from $\omega_{\mathsf{proc}}$ or $\omega_\sigma^T$, whichever is relevant depending on the type of $\mathbf{M}$. By completeness of the program logic in the denotational semantics (Proposition 5.10) $\vdash_\tau \mathbf{M} : \zeta$ follows from $[\![\zeta]\!] \leqslant [\![\mathbf{M}]\!]_\rho$. By soundness (Proposition 5.8) of the program logic in the operational semantics $\mathbf{M} \models_\tau \zeta$. If $\tau$ is a basic type then $\zeta$ 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 (Theorem 6.4, Proposition 6.5) 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 [32] for the functional fragment of our language, that every prime and compact element of the model is definable in HOCCS 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 $\mathbf{P}\,{}_{\{\alpha\}}|_\varnothing\,\mathbf{Q}$. We also write $\Omega$ for a typical divergent term of type $\sigma$ or deadlocked term of type $\mathsf{proc}$, where for example $\Omega_{\mathsf{int}} = (\mu X \lambda Y.XY)\,0$ (and similarly for the

other ground types) and $\Omega_{\sigma\to\tau} = (\mu X\lambda Y.XY)(\lambda Z.\mathbf{M})$. Alternatively, we could have of course incorporated constants $\Omega_\tau$, $\tau \in \mathsf{Type}$, in the language and thus make the use of recursive terms for definability purposes superfluous.

It is convenient to abbreviate nested conditionals using a product term inductively defined by

$$\text{if } \left(\prod_{i=1}^{i=n+1} \mathbf{B}_i\right) \text{ then } \mathbf{M} \text{ else } \mathbf{N} = \text{if } \mathbf{B}_1 \text{ then } \left(\text{if } \left(\prod_{i=2}^{i=n} \mathbf{B}_i\right) \text{ then } \mathbf{M} \text{ else } \mathbf{N}\right) \text{ else } \mathbf{N}, \tag{1}$$

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 $\mathscr{V}[\![\mathbf{M}_c]\!]_\rho = c$. Similarly, if $\pi$ is a prime of $\mathbf{D}_{\mathbf{proc}}$, then $\pi$ is defined by $\mathbf{N}_\pi$ provided $[\![\mathbf{N}_\pi]\!]_\rho = \pi \in \mathbf{D}_{\mathbf{proc}}$.

In Table 10 we define families of names $\mathbf{N}^\alpha_\pi$, $\mathbf{N}^\alpha_C$, $\mathbf{N}^\alpha_\perp$ for prime elements $\pi \in \mathbf{D}_{\mathbf{proc}}$ and compact elements $C \in \mathbf{D}_\sigma$ (hence primes of $T(\mathbf{D}_\sigma)$) for $\sigma \in \mathsf{VType}$, indexed by channel names $\alpha \in \mathscr{N}$ 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 also be 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* $c \in \mathscr{K}\mathbf{D}_\sigma$, $\pi \in \mathscr{K}\mathscr{P}\mathbf{D}_{\mathbf{proc}}$ *and* $d \in T\mathbf{D}_\sigma$, $d' \in \mathbf{D}_{\mathbf{proc}}$. *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* $\alpha$ *not occurring in* $c$ *or* $\pi$.

*Proof.* The case $c \in \mathscr{K}\mathbf{D}_\sigma$ is obvious since the test $\mathbf{T}^\alpha_c$ is defined in terms of a conditional that either converges to $\mathbf{tt}$ or diverges.

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

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

- If $\pi = \beta_{out}(c \otimes \pi_1)$ then $[\![\mathbf{T}^\alpha_\pi]\!]_\rho = \beta^T_{in}(\{\!|\lambda v \in \mathbf{D}_{\sigma(\beta)}.[\![\text{if } \mathbf{T}^\alpha_c X \text{ then } \mathbf{T}^\alpha_{\pi_1} \text{ else } \mathbf{nil}]\!]_{\rho[v/X]}|\!\})$. By the definition of $\beta^T_{in}$ we obtain $[\![\mathbf{T}^\alpha_\pi]\!]_\rho = \beta_{in}(\lambda v \in \mathbf{D}_{\sigma(\beta)}.[\![\text{if } \mathbf{T}^\alpha_c X \text{ then } \mathbf{T}^\alpha_{\pi_1} \text{ else } \mathbf{nil}]\!]_{\rho[v/X]})$. Since both $\beta_{in}$ and the parallel operator are linear it is enough to consider $c' \to \pi'_1 \leqslant \lambda v \in \mathbf{D}_{\sigma(\beta)}.[\![\text{if } \mathbf{T}^\alpha_c X \text{ then } \mathbf{T}^\alpha_{\pi_1} \text{ else } \mathbf{nil}]\!]_{\rho[v/X]}$ and show that $\beta_{in}(c' \to \pi'_1)\ {}_\alpha|_0\ \pi'$ is either $\perp$ or $\alpha_{out}(\{\!|\mathbf{tt} \otimes \perp)$. By induction, $[\![\mathbf{T}^\alpha_c X]\!]_{\rho[c'/X]} = [\![\mathbf{T}^\alpha_c]\!]_\rho(c')$ evaluates to either $\perp$ or $\{\!|\mathbf{tt}|\!\}$ and hence the condition on $c' \to \pi'_1$ reduces to $\pi'_1 \leqslant [\![\mathbf{T}^\alpha_{\pi_1}]\!]_\rho$.

**TABLE 10**

**Defining Terms for Primes**

| | | |
|---|---|---|
| (int) | $\mathbf{N}^{\alpha}_{\perp}$ | $= \Omega$ |
| | $\mathbf{N}^{\alpha}_{n}$ | $= \mathbf{n}$ |
| | $\mathbf{T}^{\alpha}_{\perp}$ | $= \lambda X.\,\mathbf{tt}$ |
| | $\mathbf{T}^{\alpha}_{n}$ | $= \lambda X.$ if $X = \mathbf{n}$ then $\mathbf{tt}$ else $\Omega$ |
| (bool) | Similar | |
| (unit) | Similar | |
| $(\sigma \to \tau)$ | $\mathbf{N}^{\alpha}_{\perp}$ | $= \Omega$ |
| | $\mathbf{T}^{\alpha}_{\perp}$ | $= \lambda X.\,\mathbf{tt}$ |
| | $(\sigma \to \sigma')$ | Set $\mathbf{M}^{\alpha}_{c \to k} =$ if $\mathbf{T}^{\alpha}_{c} X$ then $\mathbf{N}^{\alpha}_{k}$ else $\Omega$ and where $C = \bigvee_{i=1}^{i=s} c_i \to k_i$ let |
| | $\mathbf{N}^{\alpha}_{C}$ | $= \lambda X. \sum_{i=1}^{i=s} \mathbf{M}^{\alpha}_{c_i \to k_i}$ |
| | $\mathbf{T}^{\alpha}_{C}$ | $= \lambda Y.$ if $\left( \prod_{i=1}^{i=s} \mathbf{T}^{\alpha}_{k_i}(Y \mathbf{N}^{\alpha}_{c_i}) \right)$ then $\mathbf{tt}$ else $\Omega$ |
| | $(\sigma \to \mathsf{proc})$ | Set $\mathbf{M}^{\alpha}_{c \to \pi} =$ if $\mathbf{T}^{\alpha}_{c} X$ then $\mathbf{N}^{\alpha}_{\pi}$ else $\Omega$ and where $C = \bigvee_{i=1}^{i=s} c_i \to \pi_i$ let |
| | $\mathbf{N}^{\alpha}_{C}$ | $= \lambda X. \sum_{i=1}^{i=s} \mathbf{M}^{\alpha}_{c_i \to \pi_i}$ |
| | $\mathbf{T}^{\alpha}_{C}$ | $= \lambda Y.$ if $\left( \prod_{i=1}^{i=s} res_{\alpha}(\mathbf{T}^{\alpha}_{\pi_i} \alpha|_0\, Y \mathbf{N}^{\alpha}_{c_i}) \right)$ then $\mathbf{tt}$ else $\Omega$ |
| (proc) | $\mathbf{N}^{\alpha}_{\perp}$ | $= \mathbf{nil}$ |
| | $\mathbf{N}^{\alpha}_{\beta_{\mathrm{out}}(c \otimes \pi)}$ | $= \beta! [\mathbf{N}^{\alpha}_{c}]\, \mathbf{N}^{\alpha}_{\pi}$ |
| | $\mathbf{N}^{\alpha}_{\beta_{in}(c \to \pi)}$ | $= \beta? \mathbf{N}^{\alpha}_{c \to \pi}$ |
| | $\mathbf{T}^{\alpha}_{\perp}$ | $= \alpha! [\mathbf{tt}]\, \mathbf{nil}$ |
| | $\mathbf{T}^{\alpha}_{\beta_{\mathrm{out}}(c \otimes \pi)}$ | $= \beta? \lambda X.$ if $\mathbf{T}^{\alpha}_{c} X$ then $\mathbf{T}^{\alpha}_{\pi}$ else $\mathbf{nil}$ |
| | $\mathbf{T}^{\alpha}_{\beta_{in}(c \to \pi)}$ | $= \beta! [\mathbf{N}^{\alpha}_{c}]\, \mathbf{T}^{\alpha}_{\pi}$ |

The value of $\beta_{in}(c' \to \pi'_1)\ {}_{\alpha}|_0\, \pi'$ is $\perp$ when $\pi' = \perp$ or when $\pi'$ is of the form $\delta_{in}(k \to \hat{\pi})$. Consulting Table 4 it follows that the only case where the resulting value may not be $\perp$ is the case $\beta_{in}(c' \to \pi'_1)\ {}_{\alpha}|_0\, \alpha \beta_{out}(k \otimes \hat{\pi})$ when, given that $\beta \notin \mathscr{A} = \{\alpha\}$, $\mathscr{B} = \varnothing$, the value of the parallel is the join of all primes below $\pi'_1\ {}_{\alpha}|_0\, \hat{\pi}$. Since $\pi'_1 \leqslant [\![\mathbf{T}^{\alpha}_{\pi_1}]\!]_{\rho}$ this join is below $[\![\mathbf{T}^{\alpha}_{\pi_1}]\!]_{\rho}\ {}_{\alpha}|_0\, \hat{\pi}$ which, by induction, is either $\perp$ or $\alpha_{out}(\{\!|\mathbf{tt}|\!\} \otimes \perp)$.

• If $\pi = \beta_{in}(c \to \pi_1)$ then $[\![\mathbf{T}^{\alpha}_{\pi}]\!]_{\rho} = \beta_{out}([\![\mathbf{N}^{\alpha}_{c}]\!]_{\rho} \otimes [\![\mathbf{T}^{\alpha}_{\pi_1}]\!]_{\rho})$. It is sufficient to show that if $c_0 \leqslant [\![\mathbf{N}^{\alpha}_{c}]\!]_{\rho}$ and $\pi_0 \leqslant [\![\mathbf{T}^{\alpha}_{\pi_1}]\!]_{\rho}$, then $\beta_{out}(c_0 \otimes \pi_0)\ {}_{\alpha}|_0\, \pi'$ must be either $\perp$ or $\alpha_{out}(\{\!|\mathbf{tt}|\!\} \otimes \perp)$. Again consulting Table 4 the cases where $\pi'$ is either $\perp$ or of the form $\delta_{out}(c'_0 \otimes \pi'_o)$ result in the value $\perp$ for $\beta_{out}(c_0 \otimes \pi_0)\ {}_{\alpha}|_0\, \pi'$. The only case, given that $\beta \notin \mathscr{A} = \{\alpha\}$, $\mathscr{B} = \varnothing$, where the value of the parallel may not be $\perp$ is when $\pi' = \beta_{in}(c'_0 \to \pi'_0)$. It follows from Table 4 that the value of the parallel in this case is below $[\![\mathbf{T}^{\alpha}_{\pi_1}]\!]\ {}_{\alpha}|_0\, \hat{\pi}$ given that we assume that $\pi_0 \leqslant [\![\mathbf{T}^{\alpha}_{\pi_1}]\!]_{\rho}$. By induction, $[\![\mathbf{T}^{\alpha}_{\pi_1}]\!]\ {}_{\alpha}|_0\, \hat{\pi}$ must be either $\perp$ or $\alpha_{out}(\{\!|\mathbf{tt}|\!\} \otimes \perp)$. ▮

COROLLARY 6.3. *For $\alpha$ not occurring in $c$ or $\pi$, $\mathbf{T}^{\alpha}_{c}\mathbf{M} \Downarrow \mathbf{tt}$ iff $[\![\mathbf{T}^{\alpha}_{c}\mathbf{M}]\!]_{\rho} = \{\!|\mathbf{tt}|\!\}$.*

Similarly $\mathbf{T}^{\alpha}_{\pi\ \alpha}|_0\,\mathbf{P} \stackrel{\alpha!}{\Longrightarrow} [\mathbf{tt}]\,\Omega$ (for some deadlocked process $\Omega$) iff $[\![\mathbf{T}^{\alpha}_{\pi\ \alpha}|_0\,\mathbf{P}]\!]_\rho = \alpha_{out}(\{\!|\mathbf{tt}|\!\} \otimes \perp)$ iff $\mathsf{res}_\alpha(\mathbf{T}^{\alpha}_{\pi\ \alpha}|_0\,\mathbf{P}) \Downarrow \mathbf{tt}$. If, in addition, $\alpha$ does not occur in $\mathbf{P}$, then the above are equivalent to $\mathbf{T}^{\alpha,\pi}{}_{w}|_0\,\mathbf{P} \stackrel{w!}{\Longrightarrow}$ where we define $\mathbf{T}^{\alpha,\pi} = (\alpha?\lambda X.\ \text{if } \mathbf{T}^{\alpha}_{\mathbf{tt}} X \text{ then } w![\mathbf{tt}]\,\mathbf{nil} \text{ else } \mathbf{nil})\ {}_{w}|_0\,\mathbf{T}^{\alpha}_{\pi}$.

*Proof.* By definition of the tests $\mathbf{T}^{\alpha}_{c}$, the expression $\mathbf{T}^{\alpha}_{c}\mathbf{M}$ either diverges or else it converges to $\mathbf{tt}$. By Lemma 6.2 the interpretation $[\![\mathbf{T}^{\alpha}_{c}\mathbf{M}]\!]_\rho$ is either $\perp$ or $\{\!|\mathbf{tt}|\!\}$. Then the claim $\mathbf{T}^{\alpha}_{c}\mathbf{M} \Downarrow \mathbf{tt}$ iff $[\![\mathbf{T}^{\alpha}_{c}\mathbf{M}]\!]_\rho = \{\!|\mathbf{tt}|\!\}$ follows from Adequacy, Theorem 5.11. The other case is similar. ∎

THEOREM 6.4 (Definability Theorem). 1. *For every prime* $\pi \in \mathscr{KP}(\mathbf{D}_{\mathbf{proc}})$ *and each channel* $\alpha \in \mathscr{N}$:

1. *If* $\alpha$ *does not occur in* $\pi$, *then* $\pi \leqslant \pi'$ *in* $\mathbf{D}_{\mathbf{proc}}$ *iff* $\mathsf{res}_\alpha(\mathbf{T}^{\alpha}_{\pi\ \alpha}|_0\,\mathbf{N}^{\alpha}_{\pi'}) \Downarrow \mathbf{tt}$.
2. $[\![\mathbf{N}^{\alpha}_{\pi}]\!]_\rho = \pi$.

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

1. *If* $\alpha$ *does not occur in* $c$, $\mathbf{T}^{\alpha}_{c}\mathbf{N}^{\alpha}_{c'} \Downarrow \mathbf{tt}$ *iff* $c \leqslant c'$ *in* $\mathbf{D}_\sigma$.
2. $[\![\mathbf{N}^{\alpha}_{c}]\!]_\rho = \{\!|c|\!\}$.

*Proof.* We prove both 1 and 2 by simultaneous induction, on the structure of the prime or compact element $\pi$ or $c$.

1.1. The proof is by induction on $\pi$ and within that on $\pi'$.

$(\pi = \perp)$

Since $\mathbf{T}^{\alpha}_{\perp} = \alpha![\mathbf{tt}]\,\mathbf{nil}$ it is clear that $\mathbf{T}^{\alpha}_{\perp\ \alpha}|_0\,\mathbf{P}$ can make a move to $[\mathbf{tt}]\,\mathbf{nil}$ hence $\mathsf{res}_\alpha(\mathbf{T}^{\alpha}_{\perp\ \alpha}|_0\,\mathbf{P})$ evaluates to $\mathbf{tt}$, for any $\mathbf{P}$.

$(\pi = \beta_{out}(c \otimes \pi_1))$

If $\pi'$ is not an "out"-prime, i.e., of the form $\delta_{out}(c' \otimes \pi'_1)$, then $\mathbf{T}^{\alpha}_{\perp\ \alpha}|_0\,\mathbf{N}^{\alpha}_{\pi'}$ is deadlocked (recall that we assume $\alpha$ does not occur in $\pi$). In this case of course we have $\pi \not\leqslant \pi'$.

The only interesting case is when $\pi'$ is of the form $\beta_{out}(c' \otimes \pi'_1)$. Then a communication is possible and $\mathbf{T}^{\alpha}_{\perp\ \alpha}|_0\,\mathbf{N}^{\alpha}_{\pi'}$ reduces to $(\text{if } \mathbf{T}^{\alpha}_{c}\mathbf{N}^{\alpha}_{c'} \text{ then } \mathbf{T}^{\alpha}_{\pi_1} \text{ else } \mathbf{nil})\ {}_{\alpha}|_0\,\mathbf{N}^{\alpha}_{\pi'_1}$. By induction, the boolean condition is satisfied iff $c \leqslant c'$ in which case the test reduces to $\mathbf{T}^{\alpha}_{\pi_1\ \alpha}|_0\,\mathbf{N}^{\alpha}_{\pi'_1}$. Since by induction $\mathsf{res}_\alpha(\mathbf{T}^{\alpha}_{\pi_1\ \alpha}|_0\,\mathbf{N}^{\alpha}_{\pi'_1}) \Downarrow \mathbf{tt}$ iff $\pi_1 \leqslant \pi'_1$ we may conclude that in all cases $\mathsf{res}_\alpha(\mathbf{T}^{\alpha}_{\pi\ \alpha}|_0\,\mathbf{N}^{\alpha}_{\pi'}) \Downarrow \mathbf{tt}$ iff $\pi \leqslant \pi'$.

$(\pi = \beta_{in}(c \to \pi_1))$

The only case of interest is when $\pi' = \beta_{in}(c' \to \pi'_1)$ but it can be treated as in the previous subcase.

1.2. Straightforward, using induction hypothesis.

2.1. If $C = \perp$ or an element of a ground domain then the claim is immediately seen to be true. We now separate the two cases of $C = \bigvee_{i=1}^{i=s} c_i \to k_i$ in $\mathbf{D}_{\sigma\to\sigma'}$ and $C = \bigvee_{i=1}^{i=s} c_i \to \pi_i$ in $\mathbf{D}_{\sigma\to\mathbf{proc}}$, for $s \geqslant 1$ in both cases. The first case is straightforward by unfolding definitions and using the induction hypothesis. It reduces to

examining whether for each $i = 1, ..., s$ the expression $\mathbf{T}^{\alpha}_{k_i}(\mathbf{N}^{\alpha}_{C'}\mathbf{N}^{\alpha}_{c_i})$ evaluates to $\mathbf{tt}$. Given the definition of $\mathbf{N}^{\alpha}_{C'}$, for some $C' = \bigvee_{j=1}^{j=r} c'_j \to k'_j$, this is the case exactly when for each $i$ there can be found a $j$ such that $c_j \to k_i \leqslant c'_j \to k'_j$.

Now let $C = \bigvee_{i=1}^{i=s} c_i \to \pi_i$ and $C' = \bigvee_{j=1}^{j=r} c'_j \to \pi'_j$. By the definition of $\mathbf{T}^{\alpha}_{C}$ and $\mathbf{N}^{\alpha}_{C'}$ we have

$$\mathbf{T}^{\alpha}_{C}\mathbf{N}^{\alpha}_{C'} \to \text{if } \prod_{i=1}^{i=s} \mathsf{res}_{\alpha}(\mathbf{T}^{\alpha}_{\pi_i}{}_{\alpha}|_0\, \mathbf{N}^{\alpha}_{C'}\mathbf{N}^{\alpha}_{c_i}) \text{ then } \mathbf{tt} \text{ else } \Omega.$$

For each fixed $i = 1, ..., s$ and by $\beta$-reduction we have

$$\mathbf{N}^{\alpha}_{C'}\mathbf{N}^{\alpha}_{c_i} \to \sum_{j=1}^{j=r} (\text{if } \mathbf{T}^{\alpha}_{c'_j}\mathbf{N}^{\alpha}_{c_i} \text{ then } \mathbf{N}^{\alpha}_{\pi'_j} \text{ else } \Omega)$$

By induction, for each $i$ there is $j$ and a reduction $\mathbf{N}^{\alpha}_{C'}\mathbf{N}^{\alpha}_{c_i} \Rightarrow \mathbf{N}^{\alpha}_{\pi'_j}$ iff $c'_j \leqslant c_i$. If such $j = j(i)$ can be found for each $i = 1, ..., s$ then by induction $\mathbf{T}^{\alpha}_{C}\mathbf{N}^{\alpha}_{C'} \Downarrow \mathbf{tt}$ if and only if $\pi_i \leqslant \pi'_{j(i)}$. The way step functions are ordered indicates that $\mathbf{T}^{\alpha}_{C}\mathbf{N}^{\alpha}_{C'} \Downarrow \mathbf{tt}$ iff $C \leqslant C'$.

2.2. The only case of some interest is when $\pi$ is of the form $\beta_{in}(c \to \pi_1)$. Then

$$[\![\mathbf{N}^{\alpha}_{\pi}]\!]_{\rho} = [\![\beta?\lambda X.\ \text{if } \mathbf{T}^{\alpha}_{c}X \text{ then } \mathbf{N}^{\alpha}_{\pi_1} \text{ else } \mathbf{nil}]\!]_{\rho}.$$

Using linearity of $\beta_{in}$ it follows that $[\![\mathbf{N}^{\alpha}_{\pi}]\!]_{\rho}$ is the join of the primes $\beta_{in}(c_0 \to \pi_0)$ taken over the $c_0 \to \pi_0$ satisfying, after a small calculation, $c \leqslant c_0, \pi_0 \leqslant \pi_1$. Hence the join is just $\beta_{in}(c \to \pi_1)$ which is the prime $\pi$. ∎

PROPOSITION 6.5. *Let $c, \pi$ be compact and prime elements of types $\sigma$ and* proc, *respectively, and $\alpha$ a channel name not occurring in $c$ or $\pi$. Then for closed expressions* $\mathbf{M}, \mathbf{P}$ *of the respective types*

1. $\mathbf{T}^{\alpha}_{c}\mathbf{M} \Downarrow \mathbf{tt}$ *iff* $\{\!|c|\!\} \leqslant [\![\mathbf{M}]\!]_{\rho}$
2. $\mathbf{T}^{\alpha}_{\pi}{}_{\alpha}|_0\, \mathbf{P} \stackrel{\alpha!}{\Longrightarrow} [\mathbf{tt}]\, \Omega$ *iff* $\pi \leqslant [\![\mathbf{P}]\!]_{\rho}$.

*Proof.* ($\Leftarrow$) If $\{\!|c|\!\} \leqslant [\![\mathbf{M}]\!]_{\rho}$ then, for $\alpha$ not occurring in $c$, $[\![\mathbf{N}^{\alpha}_{c}]\!]_{\rho} \leqslant [\![\mathbf{M}]\!]_{\rho}$ and hence monotonicity of application implies $[\![\mathbf{T}^{\alpha}_{c}\mathbf{N}^{\alpha}_{c}]\!]_{\rho} \leqslant [\![\mathbf{T}^{\alpha}_{c}\mathbf{M}]\!]_{\rho}$. Since $\mathbf{T}^{\alpha}_{c}\mathbf{N}^{\alpha}_{c} \Downarrow \mathbf{tt}$, by a trivial application of the Definability Lemma, it follows from Lemma 6.2 that $[\![\mathbf{T}^{\alpha}_{c}\mathbf{M}]\!]_{\rho} = \{\!|\mathbf{tt}|\!\}$ and then by Corollary 6.3 $\mathbf{T}^{\alpha}_{c}\mathbf{M} \Downarrow \mathbf{tt}$. The case for processes is similar, using again Lemma 6.2 and Corollary 6.3.

($\Rightarrow$) The hypothesis $\mathbf{T}^{\alpha}_{C}\mathbf{M} \Downarrow \mathbf{tt}$ implies that $\mathbf{M} \Downarrow \mathbf{v}$, for some value $\mathbf{v}$. We proceed by induction on the structure of $C$.

If $C = n \in \mathbf{D}_{\mathsf{int}}$ then $\mathbf{M}$ must be of type int, hence $\mathbf{v} = \mathbf{m}$. The hypothesis implies that $m = n$ and so $\{\!|n|\!\} \leqslant \{\!|m|\!\} = [\![\mathbf{v}]\!]_{\rho} \leqslant [\![\mathbf{M}]\!]_{\rho}$, where we used Lemma 4.9.

If $C = \bigvee_{i=1}^{i=s} c_i \to k_i$ is in $\mathbf{D}_{\sigma \to \sigma'}$ then $\mathbf{M}$ is of the same type and so $\mathbf{M} \Downarrow \mu X\lambda Y.\mathbf{N}$ for some $\mathbf{N}$. By inspecting the definition of the test $\mathbf{T}^{\alpha}_{C}$ and from the hypothesis it follows that for each $i \in \{1, ..., s\}$ we must have $\mathbf{T}^{\alpha}_{k_i}((\mu X\lambda Y.\mathbf{N})\, \mathbf{N}^{\alpha}_{c_i}) \Downarrow \mathbf{tt}$. By induction

$\{\!|k_i|\!\} \leqslant [\![(\mu X\lambda Y.\mathbf{N})\,\mathbf{N}^{\alpha}_{c_i}]\!]_\rho = [\![\mu X\lambda Y.\mathbf{N}]\!]_\rho\,(\{\!|c_i|\!\}) = [\![\lambda Y.\mathbf{M}[\mu X\lambda Y.\mathbf{N}/X]]\!]_\rho\,(\{\!|c_i|\!\})$. It follows from the definition of the interpretation that

$$k_i \leqslant \mathbf{apply}(\lambda v \in \mathbf{D}_\sigma.\,[\![\mathbf{M}]\!][\mu X\lambda Y.\mathbf{N}/X]]\!]_{\rho[v/Y]}, c_i)$$

from which it follows that $c_i \to k_i \leqslant \lambda v \in \mathbf{D}_\sigma.\,[\![\mathbf{N}[\mu X\lambda Y.\mathbf{N}/X]]\!]_{\rho[v/Y]}$ in $\mathbf{D}_{\sigma\to\sigma'}$, for each $i = 1, ..., s$. Thus $C \leqslant \lambda v \in \mathbf{D}_\sigma.\,[\![\mathbf{N}[\mu X\lambda Y.\mathbf{N}/X]]\!]_{\rho[v/Y]}$ and so

$$\{\!|C|\!\} \leqslant \{\!|\lambda v \in \mathbf{D}_\sigma.\,[\![\mathbf{N}[\mu X\lambda Y.\mathbf{N}/X]]\!]_{\rho[v/Y]}|\!\} = [\![\mu X\lambda Y.\mathbf{N}]\!]_\rho.$$

Since $\mathbf{M} \Downarrow \mu X\lambda Y.\mathbf{N}$ using Lemma 4.9 we may conclude that $\{\!|C|\!\} \leqslant [\![\mathbf{M}]\!]_\rho$.

Finally, if $C \in \mathbf{D}_{\sigma\to\mathbf{proc}}$, $C = \bigvee_{i=1}^{i=s} c_i \to \pi_i$, it follows from the case assumption and from the definition of the test $\mathbf{T}^{\alpha}_{C}$ that

$$\mathsf{res}_\alpha(\mathbf{T}^{\alpha}_{\pi_i\,\alpha}|_0\,(\mu X\lambda Y.\mathbf{N})\,\mathbf{N}^{\alpha}_{c_i}) \Downarrow \mathbf{tt}.$$

By induction and unpacking the definition of application is

$$\pi_i \leqslant \mathbf{apply}(\lambda v \in \mathbf{D}_\sigma.\,[\![\mathbf{N}[\mu X\lambda Y.\mathbf{N}/X]]\!]_{\rho[v/Y]}, c_i).$$

The argument can now be completed along the lines of the argument just given for the case $C \in \mathbf{D}_{\sigma\to\sigma'}$ and we may thus conclude that $\{\!|C|\!\} \leqslant [\![\mathbf{M}]\!]_\rho$.

It remains to show that $\mathbf{T}^{\alpha}_{\pi\,\alpha}|_0\,\mathbf{P} \overset{\alpha!}{\Longrightarrow} [\mathbf{tt}]\,\Omega$ implies $\pi \leqslant [\![\mathbf{P}]\!]_\rho$ which we show by analyzing the cases for $\pi$.

If $\pi = \beta_{out}(c \otimes \pi_1)$ then $\mathbf{T}^{\alpha}_{\pi} = \beta?\lambda X.\mathbf{if}\ \mathbf{T}^{\alpha}_{c} X\ \mathbf{then}\ \mathbf{T}^{\alpha}_{\pi_1}\ \mathbf{else}\ \mathbf{nil}$. The hypothesis implies that $\mathbf{P} \overset{\beta!}{\Longrightarrow} [\mathbf{v}]\,\mathbf{Q}$ for some value $\mathbf{v}$ and residual process $\mathbf{Q}$. It also implies that $\mathbf{T}^{\alpha}_{c}\mathbf{v} \Downarrow \mathbf{tt}$ and $\mathbf{T}^{\alpha}_{\pi_1\,\alpha}|_0\,\mathbf{Q} \overset{\alpha!}{\Longrightarrow} [\mathbf{tt}]\,\Omega$. By induction $\pi_1 \leqslant [\![\mathbf{Q}]\!]_\rho$ and $c \leqslant [\![\mathbf{v}]\!]_\rho$. Further, using Lemma 4.9 it follows from $\mathbf{P} \overset{\beta!}{\Longrightarrow} [\mathbf{v}]\,\mathbf{Q}$ that

$$[\![\beta!\,[\mathbf{v}]\,\mathbf{Q}]\!]_\rho = \beta_{out}([\![\mathbf{v}]\!]_\rho \otimes [\![\mathbf{Q}]\!]_\rho) \leqslant [\![\mathbf{P}]\!]_\rho.$$

Since $\pi_1 \leqslant [\![\mathbf{Q}]\!]_\rho$ and $c \leqslant [\![\mathbf{v}]\!]_\rho$ it follows that $\pi = \beta_{out}(c \otimes \pi_1) \leqslant [\![\mathbf{P}]\!]_\rho$.

Finally suppose $\pi = \beta_{in}(c \to \pi_1)$. Then $\mathbf{T}^{\alpha}_{\pi} = \beta![\,\mathbf{N}^{\alpha}_{c}]\,\mathbf{T}^{\alpha}_{\pi_1}$ and the hypothesis implies that $\mathbf{P} \overset{\beta?}{\Longrightarrow} \mu X\lambda Y.\mathbf{M}$ for some $\mathbf{M}$ and $\mathbf{T}^{\alpha}_{\pi_1\,\alpha}|_0\,(\mu X\lambda Y.\mathbf{M})\,\mathbf{N}^{\alpha}_{c} \overset{\alpha!}{\Longrightarrow} [\mathbf{tt}]\,\Omega$. By induction $\pi_1 \leqslant [\![(\mu X\lambda Y.\mathbf{M})\,\mathbf{N}^{\alpha}_{c}]\!]_\rho$ from which it follows that $c \to \pi_1 \leqslant [\![\mu X\lambda Y.\mathbf{M}]\!]_\rho$. Then, using again Lemma 4.9, we obtain $\pi = \beta_{in}(c \to \pi_1) \leqslant [\![\beta?\mu X\lambda Y.\mathbf{M}]\!]_\rho \leqslant [\![\mathbf{P}]\!]_\rho$. ∎

## 7. FULL ABSTRACTION AND COMPLETENESS RESULTS

A final ingredient in the full abstraction proof is that the logic systems $\mathscr{G}_S$, $\mathscr{G}_E$ introduced as an axiomatization for semantic entailment and satisfiability are complete in the operational semantics. We prove this in this section, using the definability result to show first that the denotational and operational semantics for the program logic coincide.

*Completeness of the Logic Proof Systems in the Operational Semantics*

LEMMA 7.1. *For closed terms* $\mathbf{M}$, $\mathbf{v}$

$$\mathbf{M} \models_\tau \zeta \Rightarrow [\![\zeta]\!] \leqslant [\![\mathbf{M}]\!]_\rho \qquad \text{and} \qquad \mathbf{v} \mathrel{\mathbb{R}}_\sigma A \Rightarrow \mathscr{V}[\![A]\!] \leqslant \mathscr{V}[\![\mathbf{v}]\!]_\rho. \tag{2}$$

*Proof.* Recall that $\rho_\Gamma(X) = [\![\Gamma(X)]\!]$ for each $X \in dom(\Gamma)$ and where $\Gamma(X) \in \mathscr{L}_\sigma(\mathscr{V})$ for some $\sigma \in \mathsf{VType}$. Hence $\rho_\Gamma(X) = [\![\Gamma(X)]\!] = \{\!|\mathscr{V}[\![\Gamma(X)]\!]|\!\}$. The proof is by induction on $\zeta, A$. If $\zeta = \omega_\sigma^T$ the claim is trivial and the case $\zeta = \eta \sqcap \vartheta$ is resolved using the induction hypothesis. Suppose now $\zeta = \Diamond A$ in which case $\tau = \sigma$ for some $\sigma$ in $\mathsf{VType}$ and $A \in \mathscr{L}_\sigma(\mathscr{V})$. By definition of the satisfaction relation it follows that $\mathbf{M} \Downarrow \mathbf{v}$ for some value $\mathbf{v}$ such that $\mathbf{v} \mathrel{\mathbb{R}}_\sigma A$. If $A$ is atomic, hence $S_\ell$ for some literal $\ell$ or $\omega_\sigma^L \in \mathscr{L}_{\sigma \to \mathsf{proc}}(\mathscr{V})$ then the claim is trivial and the case where $A$ is a conjunction is resolved using the induction hypothesis. Suppose now that $A$ is of the form $B \to \eta$. By Definability, Theorem 6.4, let $\mathbf{u} = \mathbf{N}^\alpha_{\mathscr{V}[\![B]\!]}$ so that $\mathscr{V}[\![B]\!] = \mathscr{V}[\![\mathbf{u}]\!]$. It follows by soundness of the program logic in the denotational semantics that $\mathbf{u} \mathrel{\mathbb{R}}_\sigma B$ and then, by definition, $\mathbf{v}u \models_\tau \eta$. By induction $[\![\eta]\!] \leqslant [\![\mathbf{v}\mathbf{u}]\!] = \mathbf{apply}^T([\![\mathbf{v}]\!], [\![\mathbf{u}]\!])$. By definition of the interpretation of values and by construction of the map $\mathbf{apply}^T$ this is equivalent to $[\![\eta]\!] \leqslant \mathbf{apply}(\mathscr{V}[\![\mathbf{v}]\!], \mathscr{V}[\![\mathbf{u}]\!])$ and since $\mathscr{V}[\![\mathbf{u}]\!] = \mathscr{V}[\![B]\!]$, by choice of $\mathbf{u}$, it follows that $\mathscr{V}[\![B]\!] \to [\![\eta]\!] \leqslant \mathscr{V}[\![\mathbf{v}]\!]$. This implies that $[\![\Diamond(B \to \eta)]\!] \leqslant [\![\mathbf{v}]\!]$. Hence using monotonicity, Lemma 4.9, we obtain $[\![\zeta]\!] = [\![\Diamond A]\!] \leqslant [\![\mathbf{v}]\!] \leqslant [\![\mathbf{M}]\!]$.

If $\zeta$ is a property of processes, $\zeta \in \mathscr{L}_{\mathsf{proc}}(\mathscr{T})$, then the cases $\omega_{\mathsf{proc}}$ and $\varphi \,\&\, \psi$ are immediate. The other two cases, $\langle\!\langle \alpha! \rangle\!\rangle [A] \psi$ and $\langle\!\langle \alpha? \rangle\!\rangle A \to \psi$, 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,*

$$\Gamma \models^{\mathscr{O}}_\tau \mathbf{M} : \zeta \text{ if and only if } \Gamma \models^{\mathscr{D}}_\tau \mathbf{M} : \zeta.$$

*Proof.* The direction ($\Leftarrow$) is immediate since $\Gamma \models^{\mathscr{D}}_\tau \mathbf{M} : \zeta$ is equivalent (by completeness of the program logic in the denotational semantics, Theorem 5.10) to $\Gamma \vdash_\tau \mathbf{M} : \zeta$. By soundness of the program logic in the operational semantic (Proposition 5.8) it follows that $\Gamma \models^{\mathscr{O}}_\tau \mathbf{M} : \zeta$.

Now consider the direction ($\Rightarrow$). Using Lemma 7.1 we now show that $\Gamma \models_\tau \mathbf{M} : \zeta$ implies $[\![\zeta]\!] \leqslant [\![\mathbf{M}]\!]_{\rho_\Gamma}$. By definability let $s_\Gamma$ be the value-substitution defined by $s_\Gamma(X) = \mathbf{N}^\alpha_{\mathscr{V}[\![\Gamma(X)]\!]}$ where we choose $\alpha$ not occurring in any of the $\mathscr{V}[\![\Gamma(X)]\!]$. Then $s_\Gamma \models \Gamma$ and therefore $\mathbf{M}s_\Gamma \models_\tau \zeta$. By Lemma 7.1 it follows that $[\![\zeta]\!] \leqslant [\![\mathbf{M}s_\Gamma]\!]_\rho$. By the Substitution Lemma (Proposition 4.6) and given the definition of $s_\Gamma$ and $\rho_\Gamma$ it follows that $[\![\zeta]\!] \leqslant [\![\mathbf{M}s_\Gamma]\!]_\rho = [\![\mathbf{M}]\!]_{\rho_\Gamma}$. ∎

COROLLARY 7.3 (Completeness of the Proof System $\mathscr{G}_S$ for Satisfiability).

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

*Proof.* Combine Theorem 7.2 on the equivalence of operational and denotational semantics for the program logic and Theorem 5.10 on soundness and completeness of the program logic in the denotational semantics. ▌

COROLLARY 7.4 (Completeness of the Proof System $\mathscr{G}_E$ for Semantic Entailment). *For any sentences* $\zeta, \eta$, $\zeta \vdash_\tau \eta$ *if and only if* $\zeta \models_\tau \eta$.

*Proof.* Soundness has been shown in Theorem 5.3. If $\zeta \models_\tau \eta$ and since $\mathbf{N}^\alpha_{[\![\zeta]\!]} \models_\tau \zeta$ it follows that $\mathbf{N}^\alpha_{[\![\zeta]\!]} \models_\tau \eta$. This is equivalent to $\models^{\mathscr{O}}_\tau \mathbf{N}^\alpha_{[\![\zeta]\!]} : \eta$. By Theorem 7.2 this is equivalent to $\models^{D}_\tau \mathbf{N}^\alpha_{[\![\zeta]\!]} : \eta$. By definition of this relation we obtain $[\![\eta]\!] \leqslant [\![\mathbf{N}^\alpha_{[\![\zeta]\!]}]\!]_{\rho_\Gamma} = [\![\zeta]\!]$, where $\Gamma = \varnothing$. By completeness of the system $\mathscr{G}_E$ in the denotational semantics, Proposition 5.4, this is equivalent to $\zeta \vdash_\tau \eta$. ▌

*Full Abstraction*

In this section we conclude with showing that the four preorders defined on language expressions, and the induced equivalences, coincide. Because of Propositions 3.4, 3.3 it is sufficient to establish the following:

THEOREM 7.5 (Full Abstraction).

$$H \rhd \mathbf{M} \sqsubseteq_{\mathscr{T}} \mathbf{N} \overset{(1)}{\Longrightarrow} H \rhd \mathbf{M} \sqsubseteq_{\mathscr{D}} \mathbf{N} \overset{(2)}{\Longrightarrow} H \rhd \mathbf{M} \sqsubseteq_{\mathscr{L}} \mathbf{N} \overset{(3)}{\Longrightarrow} H \rhd \mathbf{M} \sqsubseteq_{\mathscr{T}} \mathbf{N}$$

*Proof.* For (1), we first show that for *closed* process expressions $\mathbf{P}$, $\mathbf{Q}$, the hypothesis $H \rhd \mathbf{P} \sqsubseteq_{\mathscr{T}} \mathbf{Q}$ implies $[\![\mathbf{P}]\!]_\rho \leqslant [\![\mathbf{Q}]\!]_\rho$.

To establish the latter we show that $\pi \leqslant [\![\mathbf{P}]\!]_\rho$ implies $\pi \leqslant [\![\mathbf{Q}]\!]_\rho$ for any prime $\pi$. Let $\pi \leqslant [\![\mathbf{P}]\!]_\rho$ be a prime and $\mathbf{N}^\alpha_\pi$ a name for $\pi$ with $\alpha$ not occurring in any of $\pi, \mathbf{P}, \mathbf{Q}$. By Proposition 6.5, $\mathbf{T}^\alpha_{\pi\,\alpha}|_0\,\mathbf{P} \overset{\alpha!}{\Longrightarrow} [\mathbf{tt}]\,\Omega$. From the hypothesis that $\mathbf{P} \sqsubseteq_{\mathscr{T}} \mathbf{Q}$ applied to the context $\mathbf{T}^\alpha_{\pi\,\alpha}|_0\,[-]$ we obtain $\mathbf{T}^\alpha_{\pi\,\alpha}|_0\,\mathbf{Q} \overset{\alpha!}{\Longrightarrow} [\mathbf{v}]\,\mathbf{Q}'$, for some $\mathbf{v}$ and $\mathbf{Q}'$. Consulting Lemma 6.2 and by Monotonicity (Lemma 4.9) we must have $[\![\mathbf{v}]\!]_\rho \leqslant \{\!|\mathbf{tt}|\!\}$ and $[\![\mathbf{Q}]\!]' \leqslant \bot$. Hence $\mathbf{v} = \mathbf{tt}$ and by Adequacy (Theorem 5.11) $\mathbf{Q}' = \Omega$ is a divergent term. Since now $\mathbf{T}^\alpha_{\pi\,\alpha}|_0\,\mathbf{Q} \overset{\alpha!}{\Longrightarrow} [\mathbf{tt}]\,\Omega$ and application of Proposition 6.5 yields $\pi \leqslant [\![\mathbf{Q}]\!]_\rho$. We next consider the case when $\mathbf{M}, \mathbf{N}$ are *closed* expressions of some transmittable value type. Then $\mathbf{M} \sqsubseteq_{\mathscr{T}} \mathbf{N}$ implies $\beta![\lambda().\mathbf{M}]\,\mathbf{nil} \sqsubseteq_{\mathscr{T}} \beta![\lambda().\mathbf{N}]\,\mathbf{nil}$ and using the previous case we conclude $[\![\lambda().\mathbf{M}]\!]_\rho \leqslant [\![\lambda().\mathbf{N}]\!]_\rho$. It follows that $[\![\mathbf{M}]\!]_\rho \leqslant [\![\mathbf{N}]\!]_\rho$.

Finally consider the case when $\mathbf{M}, \mathbf{N}$ are arbitrary language expressions and assume that $H \rhd \mathbf{M} \sqsubseteq_{\mathscr{T}} \mathbf{N}$. For an open term $\mathbf{M}$ with the free variable $X$ (similarly for more than one free variables) we may think of $[\![\mathbf{M}]\!]_\rho$ as the continuous function $f_M = (d \mapsto [\![\mathbf{M}]\!]_{\rho[d/X]})$. By continuity $f_M$ is determined by its restriction to compact elements $d$. It is then sufficient to show that $[\![\mathbf{M}]\!]_\rho \leqslant [\![\mathbf{N}]\!]_\rho$ for any *compact* environment $\rho$, 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 $s_\rho$ such that $[\![s_\rho(X)]\!]_\rho = \rho(X)$ for every variable $X$. From the Substitution Lemma, Proposition 4.6, it follows that $[\![\mathbf{M}]\!]_\rho = [\![\mathbf{M}s_\rho]\!]_\rho$ and similarly for $\mathbf{N}$. Let $C[\,\cdot\,]$ be the context $\lambda X_1 \cdots \lambda X_k[\,\cdot\,]\, s_\rho(X_1) \cdots s_\rho(X_k)$, where $X_1 \cdots X_k$ are all the variables used in $H$. Then $[\![\mathbf{M}s_\rho]\!]_\rho = [\![C[\mathbf{M}]]\!]_\rho$. Moreover the expressions $C[\mathbf{M}]$, $C[\mathbf{N}]$ are both closed

and $H \rhd \mathbf{M} \sqsubseteq_{\mathscr{T}} \mathbf{N}$ implies $H \rhd C[\mathbf{M}] \sqsubseteq_{\mathscr{T}} C[\mathbf{N}]$. Now use the previous case to conclude $[\![C[\mathbf{M}]]\!]_\rho \leqslant [\![C[\mathbf{N}]]\!]_\rho$. Hence $[\![\mathbf{M}s_\rho]\!]_\rho \leqslant [\![\mathbf{N}s_\rho]\!]_\rho$ and so $[\![\mathbf{M}]\!]_\rho \leqslant [\![\mathbf{N}]\!]_\rho$.

For (2), suppose $H \rhd \mathbf{M} \sqsubseteq_{\mathscr{D}} \mathbf{N}$. We must show $\Gamma \models^{\mathscr{O}} \mathbf{M} : \zeta$ implies $\Gamma \models^{\mathscr{O}} \mathbf{N} : \zeta$. 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, $\lambda X_1 \cdots \lambda X_k[\cdot]\, s(X_1) \cdots s_(X_k)$, where $X_1 \cdots X_k$ are all the variables used in $H$. Then $[\![\mathbf{M}s]\!]_\rho = [\![C[\mathbf{N}]]\!]_\rho$ and $[\![\mathbf{N}s]\!]_\rho = [\![C[\mathbf{N}]]\!]_\rho$. Also $H \rhd \mathbf{M} \sqsubseteq_{\mathscr{D}} \mathbf{N}$ implies $[\![C[\mathbf{M}]]\!]_\rho \leqslant [\![C[\mathbf{N}]]\!]_\rho$.

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

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

Let $p \leqslant [\![\mathrm{v}]\!]$ be a prime. Then $p = [\![A]\!]$ for some sentence $A$ and it follows from the logic completeness results that $\mathrm{v} \models_\sigma A$. 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* $\mathsf{res}_\alpha$

The proof that the model is fully abstract has made significant use of the functionals $\mathsf{res}_\alpha$ for definability purposes. We conclude this section with a discussion on the necessity of these operators for full abstraction purposes.

THEOREM 7.6. *Let $\mathscr{L}^-$ be the fragment of* HOCCS *without the functionals* $\mathsf{res}_\alpha$. *Then full abstraction for the context preorder on $\mathscr{L}^-$ and definability fail.*

*Proof.* The proof that the may testing and context preorders coincide made use of $\mathsf{res}_\alpha$ and hence in $\mathscr{L}^-$ the two cases need to be treated separately. For the context preorder it is rather straightforward to see that full abstraction fails for the fragment $\mathscr{L}^-$. If $\mathscr{C}[\ ]$ is any context (with zero or more occurrences of the hole [ ]) such that for $\mathbf{P}$ a closed process term $\mathscr{C}[\mathbf{P}] : \mathsf{bool}$ then the hypothesis that $\mathscr{C}[\mathbf{P}] \Downarrow b \in \{\mathbf{tt}, \mathbf{ff}\}$ implies that for any closed process term $\mathbf{Q}$ there is also a reduction $\mathscr{C}[\mathbf{Q}] \Downarrow b$. The proof is by transition induction and, since there are no rules allowing an inference of a reduction step $\mathbf{M} \to \mathbf{N}$ for terms of some transmittable type $\sigma \in \mathsf{VType}$ from some reduction $\mathbf{P}_1 \to \mathbf{P}_2$ of process terms, it reduces to the case of one-step reductions $\mathscr{C}[\mathbf{P}] \to b$. Inspecting the axioms it follows that $\mathscr{C}[\mathbf{Q}] \to b$. Hence any two process terms $\mathbf{P}$, $\mathbf{Q}$ are context-equivalent while, for example, $[\![\alpha!\,[\mathbf{tt}]\,\mathbf{nil}]\!]_\rho \neq [\![\alpha!\,[\mathbf{ff}]\,\mathbf{nil}]\!]_\rho$.

We next show that definability fails for the fragment $\mathscr{L}^-$. It follows from the above argument that there can exist no functional term $\mathbf{F}$ of type $(\mathsf{unit} \to \mathsf{proc}) \to \mathsf{bool}$ in the fragment $\mathscr{L}^-$ such that for any process term $\mathbf{P}$ the following are equivalent:

1. $\mathbf{F}\lambda().\mathbf{P}\Downarrow$
2. $\mathbf{F}\lambda().\mathbf{P}\Downarrow \mathbf{tt}$
3. $\mathbf{P} \stackrel{\alpha!}{\Longrightarrow} [\mathbf{tt}]\,\mathbf{Q}$, for some process $\mathbf{Q}$.

This is because setting $\mathscr{C}[\ ] \equiv \mathbf{F}\lambda().[\ ]$ we must have $\mathscr{C}[\alpha![\mathbf{tt}]\,\mathbf{nil}]\Downarrow \mathbf{tt}$, while $\mathscr{C}[\alpha![\mathbf{ff}]\,\mathbf{nil}]\Uparrow$, contradicting the conclusion derived above that all process terms are context-equivalent in $\mathscr{L}^-$. If the model becomes unsound for the testing preorder on the fragment $\mathscr{L}^-$ then of course it is not fully abstract. We may then assume that $[\![\mathbf{M}]\!] \leqslant [\![\mathbf{N}]\!]$ implies $\mathbf{M} \sqsubseteq_{\mathscr{T}} \mathbf{N}$. Note also that restriction to the fragment $\mathscr{L}^-$ does not invalidate the following properties of the interpretation, proven by transition induction.

$$\mathbf{M} \Rightarrow \mathbf{N} \text{ implies } [\![\mathbf{N}]\!] \leqslant [\![\mathbf{M}]\!] \text{ and } \mathbf{P} \stackrel{\alpha!}{\Longrightarrow} [\mathbf{v}]\,\mathbf{Q} \text{ implies } [\![\alpha![\mathbf{v}]\,\mathbf{Q}]\!] \leqslant [\![\mathbf{P}]\!]$$

Let $\bigstar$ be the unique element of the unit domain. Then $f = (\bigstar \to \alpha_{out}(\{\!|\mathbf{tt}|\!\} \otimes \bot)) \to \{\!|\mathbf{tt}|\!\}$ is a compact element of the domain $[\mathbf{D}_{\mathsf{unit}} \to L(\mathbf{D}_{\mathsf{proc}})] \to T(\mathbf{D}_{\mathsf{bool}})$. We claim that for any $\mathbf{P}$, conditions 1–3 above would be equivalent for a name $\mathbf{F}$ for $f$, assuming one could exist in $\mathscr{L}^-$.

From $[\![\mathbf{F}]\!] = \{\!|f|\!\}$ it follows that $[\![\mathbf{F}\lambda().\mathbf{P}]\!] \in \{\bot, \{\!|\mathbf{tt}|\!\}\}$. If $\mathbf{F}\lambda().\mathbf{P}\Downarrow$ then for some value $\mathbf{v}$ we have

$$\bot < \{\!|\curlyvee [\![\mathbf{v}]\!]_\bot |\!\} = [\![\mathbf{v}]\!] \leqslant [\![\mathbf{F}\lambda().\mathbf{P}]\!] \in \{\bot, \{\!|\mathbf{tt}|\!\}\}$$

and so $\mathbf{v} = \mathbf{tt}$; i.e., $\mathbf{F}\lambda().\mathbf{P}\Downarrow$ implies $\mathbf{F}\lambda().\mathbf{P}\Downarrow \mathbf{tt}$. If $\mathbf{P} \stackrel{\alpha!}{\Longrightarrow} [\mathbf{tt}]\,\mathbf{Q}$ for some process $\mathbf{Q}$, then $[\![\alpha![\mathbf{tt}]\,\mathbf{Q}]\!] \leqslant [\![\mathbf{P}]\!]$ and hence, given that $\mathbf{F}$ names $f$ it follows that

$$\{\!|\mathbf{tt}|\!\} = [\![\mathbf{F}\lambda().\alpha![\mathbf{tt}]\,\mathbf{Q}]\!] \leqslant [\![\mathbf{F}\lambda().\mathbf{P}]\!]$$

and soundness implies $\mathbf{F}\lambda().\mathbf{P}\Downarrow \mathbf{tt}$. Finally, assuming that $\mathbf{F}\lambda().\mathbf{P}$ converges to $\mathbf{tt}$ implies that $\alpha_{out}(\{\!|\mathbf{tt}|\!\} \otimes \bot) \leqslant [\![\mathbf{P}]\!]$ and soundness again implies $\alpha![\mathbf{tt}]\,\mathbf{nil} \sqsubseteq_{\mathscr{T}} \mathbf{P}$, hence $\mathbf{P} \stackrel{\alpha!}{\Longrightarrow} [\mathbf{tt}]\,\mathbf{Q}$ for some $\mathbf{Q}$.

Hence the hypothesis that a name for $f$ exists contradicts the fact that there can be no term $\mathbf{F}$ such that for any process term $\mathbf{P}$ 1–3 above are equivalent. So either the model becomes unsound for the testing preorder (hence not fully abstract) or else definability fails. ▌

A small variation of the argument shows that any function of the form $(\bigstar \to \alpha_{out}(\{\!|\ell|\!\} \otimes \bot)) \to \{\!|b|\!\}$, $b \in \{\mathbf{tt}, \mathbf{ff}\}$, is not definable in the fragment $\mathscr{L}^-$. We conjecture, but have not rigorously shown at this moment, that failure of definability entails that the model is not complete for the testing preorder. The idea is to consider terms of the form $\mathbf{M} \equiv \lambda X.(X\lambda().\delta![\mathbf{tt}]\,\mathbf{nil})$, $\mathbf{N} \equiv \lambda X.(X\lambda().\delta![\mathbf{ff}]\,\mathbf{nil}) : ((\mathsf{unit} \to \mathsf{proc}) \to \mathsf{bool}) \to \mathsf{bool}$. Then for a process context $\mathscr{K}[\ ]$ to detect the difference between $\mathbf{M}$ and $\mathbf{N}$ it needs to be shown that there must be an evaluation step where $\mathbf{M}, \mathbf{N}$ are applied to a term $\mathbf{F}$ of type $(\mathsf{unit} \to \mathsf{proc}) \to \mathsf{bool}$. It is easy to see however that $\mathbf{MF}\Downarrow b \in \{\mathbf{tt}, \mathbf{ff}\}$ if and only if $\mathbf{NF}\Downarrow b$.

## 8. CONCLUSIONS

In this paper we have studied a language which combines sequential and concurrent control features. The functional fragment of the language resembles a call-by-value, nondeterministic version of PCF, as in [32]. This has been augmented with a new type for processes with constructors for the prefixing, choice and concurrency primitives of value-passing CCS.

The main achievement of the paper is to provide a fully-abstract denotational model, specified using a domain equation, for the standard Morris-style testing preorder of $\lambda$-calculus terms, extended to this new setting. This was accomplished by casting our model in a logical form and providing the relevant completeness and definability results. In the course of the full abstraction argument we provided a proof system for satisfiability and proved it complete in the operational semantics (using the definability result). This system is of interest on its own and independent of the full abstraction result as it can be used to compute properties of complex program terms from such of simple terms. The logical language we have introduced can be thought of as specifying a type system for the programming language. We have kept this system simple, introducing logical operators (type constructors) only to the extent necessary for our main results. However it would be interesting to make the type system more expressive, for example by adding a fixpoint operator, and in this way obtain a language for expressing nontrivial behavioral properties of processes and an associated proof system.

As we have stated in the Introduction, although HOCCS contains many features of *core Facile* [4], there are some serious omissions. Allowing the computation threads which compute values to interact more easily with process threads would increase the expressiveness of the language; we believe that our framework can accommodate this extension by using a more complicated domain equation. However, allowing dynamic name creation is much more problematic and new techniques, such as those employed in [33], are certainly required to model such language extensions.

Much of the literature on semantics for higher-order concurrent languages is based on the theory of *bisimulations*; example publications in this vein include [36, 37, 31, 13]. However, none of these propose denotational models, which is not surprising in view of the traditional difficulty which denotational techniques encounter with trying to capture bisimulation based theories. The major exception to this is presented in [1], a fully-abstract denotational model, defined using a domain equation, for *strong bisimulation equivalence* [22] over CCS. However, *strong bisimulation equivalence* [22] is of limited interest as a semantic equivalence for processes and finding reasonable denotational domains for so-called *weak bisimulation equivalences* [22], particularly those which appear necessary in the presence of higher-order constructs [13], is a serious problem. However, in [19, 15] denotational models, similar in style to ours, are presented for other higher-order languages; the first is for CHOCS [36], while the second treats a monadic version of CML. It should be emphasized that all of these models, including that presented in the present paper and even that in [6] for an extension of the $\lambda$-calculus with a parallel operator, are constructed with *may* testing in mind and none accommodate the

static scoping of channel names. Thus the major challenges are to extend the mathematical framework so that a proper scoping of channel names can be properly modeled, possibly using the ideas of [33] and to extend the property logics so as to capture behavioral preorders finer than *may* testing. Some preliminary results for *must* testing can be found in [20, 12]. We believe that whatever techniques will emerge will be equally applicable to a wide range of concurrent languages including FACILE, CHOCS, and CML.

We end with some pointers to related work on the use of type systems and filter models. The technique was originally used in [7] to obtain models of the $\lambda$-calculus and in [3] a very general theory for the construction of such models is developed. In [9, 6] suitable logics are developed for characterizing extensions of the $\lambda$-calculus with a parallel operator while the full-abstraction results already cited from [19, 15] are obtained by extending the technique to process based languages.

Received July 8, 1997; final manuscript received February 9, 1998

## REFERENCES


1. Abramsky, S. (1991), A domain equation for bisimulation, *Inform. and Comput.* **92**, 161–218.
2. Abramsky, S., Gabbay, D. M., and Maibaum, T. S. E. (Eds.) (1994), "Handbook of Logic in Computer Science," Vol. 1–6, Oxford Science Publications, Clarendon Press, Oxford.
3. Abramsky, S. (1991), Domain theory in logical form, *Ann. Pure Appl. Logic* **51**, 1–77.
4. Amadio, R. M. (1994), "Translating Core Facile," Technical Report ECRC-1994-3, European Computer-Industry Research Center, Munich.
5. Barendregt, H. P. (1984), "The $\lambda$-Calculus, its Syntax and Semantics," North-Holland, Amsterdam.
6. Boudol, G. (1994), A $\lambda$-calculus for (strict) parallel functions, *Inform. Comput.* **108**, 51–127.
7. Barendregt, H., Coppo, M., and Dezani-Ciancaglini, M. (1983), A filter model and the completeness of type assignment, *J. Symbolic Logic* **48**, 931–940.
8. Damiani, F., Dezani-Ciancaglini, M., and Giannini, P., A filter model for mobile processes, *Math. Str. Comp. Sci.*, to appear.
9. Dezani-Ciancaglini, M., de'Liguoro, U., and Piperno, A. (1993), Filter models for a parallel and non deterministic $\lambda$-calculus, *in* "Proc. Mathematical Foundations of Computer Science 1993" (A. M. Borzyszkowski and S. Sokolowski, Eds.), LNCS 711, pp. 403–412, Springer-Verlag, Berlin.
10. Giacalone, A., Mishra, P., and Prasad, S. (1989), FACILE: A symmetric integration of concurrent and functional programming, *Int. J. Parallel Program.* **15**(2), 121–160.
11. Ferreira, W., Hennessy, M., and Jeffrey, A. (1996), A theory of weak bisimulation for core CML, *in* "Proceedings of ACM SIGPLAN Int. Conf. Functional Programming," LNCS, pp. 201–212, Assoc. Comput. Mach., New York.
12. Ferreira, W., and Hennessy, M. (1995), Towards a semantic theory for CML, *in* "Proc. Mathematical Foundations of Computer Science 1995" (J. Wiedermann and P. Hájek, Eds.), LNCS 969, pp. 454–466, Springer-Verlag, Berlin.
13. Fereira, W., Hennessy, M., and Jeffrey, A. (1996), "Combining the Typed $\lambda$-Calculus with CCS," Report 2/96, University of Sussex, Computer Science.
14. Fournet, C., and Gonthier, G. (1994), The reflexive CHAM and the join-calculus, *in* "Proc. 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages," pp. 372–385, Assoc. Comput. Mach., New York.
15. Jeffrey, A. (1995), A fully abstract semantics for a concurrent functional language with monadic types, *in* "Proceedings, Tenth Annual IEEE Symposium on Logic in Computer Science," pp. 244–254, IEEE Computer Society Press.

16. Hartonas, C. (1998), Duality for modal $\mu$-logics, *Theoretical Comput. Sci.*, to appear.
17. Hartonas, C., and Hennessy, M. (1998), Full abstraction for a functional/concurrent language with higher-order value-passing (extended abstract), *in* "Proc. Computer Science Logic, CSL 97," LNCS 1414, pp. 239–354, Springer-Verlag, Berlin.
18. Hennessy, M. (1988), "An Algebraic Theory of Processes," MIT Press, Cambridge, MA.
19. Hennessy, M. (1994), A fully abstract denotational model for higher-order processes, *Inform. and Comput.* **112**(1), 55–95.
20. Hennessy, M., Higher-order processes and their models, *in* "Proceedings, 21st International Colloquium on Automata, Languages and Programming" (S. Abiteboul and E. Shamir, Eds.), LNCS 820, pp. 286–303, Springer-Verlag, Berlin.
21. Larsen, K. G. (1990), Proof systems for satisfiability in Hennessy-Milner logic with recursion, *Theoret. Comput. Sci.* **72**, 265–288.
22. Milner, R. (1989), "Communication and Concurrency," Prentice-Hall, Englewood Cliffs, NJ.
23. Milner, R., Parrow, J., and Walker, D. (1992), A calculus of mobile processes I, II, *Inform. and Comput.* **100**, 1–40, 41–77.
24. Moggi, E. (1991), Notions of computation and monads, *Inform. and Comput.* **93**, 55–92.
25. Morris, J. (1968), "Lambda-Calculus Models of Programming Languages," Ph.D. thesis, MIT.
26. Plotkin, G. (1976), A powerdomain construction, *SIAM J. Comput.* **5**, 452–487.
27. Plotkin, G. (1997), LCF considered as a programming language, *Theoret. Comput. Sci.* **5**, 323–355.
28. Reppy, J. H. (1991), A higher-order concurrent language, *in* "Proceedings of the ACM SIGPLAN '91 PLDI, SIGPLAN Notices," No. 26, pp. 294–305, Assoc. Comput. Mach., New York.
29. Reppy, J. H. (1991), "Higher-Order Concurrency," Ph.D. thesis, Cornell University.
30. Reppy, J. (1991), CML: A higher-order concurrent language, *in* "Proc. ACM-SIGPLAN '91, Conf. on Programming Language Design and Implementation," Assoc. Comput. Mach., New York.
31. Sangiorgi, D., Bisimulation in higher-order calculu, *Inform. and Comput.*, to appear. [Technical Report INRIA-Sophia Antipolis RR-2508. Revised version of the homonym paper appeared in the proc. IFIP Working Conference on Programming Concepts, Methods and Calculi (PROCOMET'94), North Holland, 1994].
32. Sieber, K. (1993), Call-by-value and nondeterminism, *in* "Typed $\lambda$-Calculi and Applications" (M. Bezen and J. F. Groote, Eds.), LNCS 664, Springer-Verlag, Berlin.
33. Stark, I. (1996), A fully-abstract domain model for the $\pi$-calculus, *in* "Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science," pp. 36–42, IEEE Computer Society Press.
34. Stirling, C. (1985), A proof-theoretic characterization of observational equivalence, *Theoret. Comput. Sci.* **39**, 27–45.
35. Stirling, C. (1987), Modal logics for communicating systems, *Theoret. Comput. Sci.* **49**, 311–347.
36. Thomsen, B. (1990), "Calculi for Higher-Order Communicating Systems," Ph.D. thesis, Imperial College.
37. Thomsen, B. (1993), Plain chocs: A second generation calculus for higher order processes, *Acta Informatica* **30**, 1–59.
38. Winskel, G. (1985), A complete proof system for SCCS with modal assertions, *in* "Proceedings, 5th Conference on Foundations of Software Technology and Theoretical Computer Science" (S. Maheshwari, Ed.), LNCS 206, pp. 392–410, Springer-Verlag, Berlin.

*Printed in Belgium*