Completeness and Logical Full Abstraction in Modal Logics for Typed Mobile Processes

Martin Berger[1], Kohei Honda[2], and Nobuko Yoshida[1]

[1] Imperial College London
[2] Queen Mary, University of London

Abstract. We study an extension of Hennessy-Milner logic for the π-calculus which gives a sound and complete characterisation of representative behavioural preorders and equivalences over typed processes. New connectives are introduced representing actual and hypothetical typed parallel composition and hiding. We study three compositional proof systems, characterising the May/Must testing preorders and bisimilarity. The proof systems are uniformly applicable to different type disciplines. Logical axioms distill proof rules for parallel composition studied by Amadio and Dam. We demonstrate the expressiveness of our logic through verification of state transfer in multiparty interactions and fully abstract embeddings of program logics for higher-order functions.

1 Introduction

Communication is becoming a foremost element of computing, from web services to sensor networks to multicore programming. The diversity of behaviour these communicating systems exhibit is staggering, including functional and stateful, sequential and concurrent, and deterministic and non-deterministic. A useful way of understanding this diversity is to classify behaviour into *types*. A compositional universe of types has fundamental merit in engineering, helping distilled understanding of the semantics of behaviour and guaranteeing basic safety such as the absence of communication errors.

The π-calculus [17] is an expressive formalism for concurrency, representing a vast array of communication behaviours with its small syntax. Starting from Milner's sorting [16], many different notions of types have been studied to classify different universes of interactions. For example, one linear type discipline turns the π-calculus into a semantic universe which exactly captures call-by-name and call-by-value higher-order sequential computation [4].

Built on the preceding studies of modal logics for the untyped π-calculus [2,8,18] and CCS [20,21], as well as on our own works on program logics [3,10,25], the present work introduces a sound and complete modal logic for typed π-calculi which is uniformly applicable to diverse type disciplines. Its adaptability comes from three logical operators, representing actual and hypothetical parallel composition and hiding. The introduction of these operators is less about sheer expressiveness than about the organisation of proof rules. Compositional reasoning is now confined to the proof rules of the logic, which precisely follow the syntactic structures of processes; whereas extracting the modal content of composition is relegated to the axioms of the assertion language.

L. Aceto et al. (Eds.): ICALP 2008, Part II, LNCS 5126, pp. 99–111, 2008.

This organisation helps us uniformly treat multiple type disciplines and their mixture in logic: different type disciplines induce different axioms for these operators, reflecting their distinct semantic effects, while keeping the identical proof rules.

Typed composition in the π-calculus often yields locally deterministic interactions, which allows us to abstract away silent actions semantically. This is often essential for reasoning about embeddings of data structures and programming languages. To capture this effect, the present study considers modal assertions and proof systems for weak typed transitions. Suggested by our study on logics for higher-order functions [10], we construct three proof systems, the first one based on the May modality, the second one on Must, and the third one which mixes these modalities. By deriving characteristic formulae, we prove completeness of these proof systems with respect to the May/Must testing preorders and bisimilarity. These results are established for the integration of three channel type disciplines widely found in the literature, *non-deterministic*, *linear* and *replicated*. These results extend to other linear and non-linear disciplines.

The combination of types and logics offers a powerful reasoning framework. We show two case studies. First we reason about a practical business protocol, using a new axiom for fixed point formulae for merging states in synchronised interactions. Second we show our logic can fully abstractly embed the total and partial program logics for call-by-value higher-order functions studied in [10]. The result extends to other program logics, offering a unifying view on logics for sequential and concurrent programs.

Related Work. Hennessy-Milner logic of the untyped π-calculus is first studied in [18] where early and late bisimilarities are characterised. Amadio and Dam [2] study model checking and proof systems of Hennessy-Milner logic of the untyped π-calculus with minimal and maximal fixed points. Dam [8] presents a proof system with ordinal-indexed fixed point formulae with a powerful discharge rule and presents specifications on Milner's encoding of data structures. Our logic is built on these works. One of the key contributions of the present work is the introduction of axioms for parallel composition based on typed synchronisation algebra, through which we can logically capture the semantics of typed processes. As far as we know, ours is the first modal logic for mobile processes which fully characterises typed semantics.

Other process logics for the untyped π-calculus include [15,23], which study efficient proof search using a freshness quantifier ∇; [6], which presents a logic for spatial properties using a hiding operator and a freshness operator; and [5], which extends Abramsky's logical characterisation of a class of CPOs to obtain a negation-less logic which corresponds to a power domain constructed by Fiore and others and which characterises a strong late bisimilarity.

The logical operators for actual and hypothetical parallel composition appeared in Stirling's early work [20,21]. Their usage in the present work originates in [3]. The operator for hypothetical composition allows rely-guarantee-based reasoning [12], whose analogue in the sequent format is studied by Simpson [19] as well as in [2,7,8]. Logical full abstraction of PCF is studied in [14] in the context of CPOs. A derivation of a program logic from a typed process logic is studied in [9]. A fully abstract embedding of a program logic in a modal process logic may not be found in the literature.

The full version of the present paper [1] lists detailed proofs and further examples.

2 Processes and Types

Processes. We use a typed π-calculus with three kinds of channel types: *linear*, *non-deterministic* and *replicated*. Linear types are based on *session types* [11,22] which allow legible description of structured communication. For simplicity, we omit the delegation primitive. The grammar of processes $(P, Q, \ldots)$ is given by:

$$\begin{array}{rl} P ::= & \mathbf{0} \mid a(k).P \mid !a(k).P \mid \overline{a}(k).P \mid k(x).P \mid \overline{k}\langle e\rangle.P \mid k \triangleleft l.P \mid k \triangleright [l_i : P_i]_{i\in I} \\ \mid & \texttt{if}\ e\ \texttt{then}\ P\ \texttt{else}\ Q \mid P|Q \mid (\nu u)P \mid (\mathbf{rec}X(\tilde{x}).P)\langle\tilde{e}\rangle \mid X\langle\tilde{e}\rangle \end{array}$$

$k, k', ..$ are *linear channels*; $a, b, c, ..$ *shared channels*; $u, u', ..$ their union; $v, w, ..$ *values*, which are constants (numbers and booleans) and channels; $x, y, ..$ variables; $X, Y, ..$ process variables; and $l, l_i, ..$ labels for branching. Expressions $(e, e', ..)$ are variables, constants, arithmetic/boolean operations (such as $e + e'$) and linear/shared channels.

The process $a(k).P$ receives a request to establish a session from $\overline{a}(k).Q$. $!a(k).P$ is the replicated version of $a(k).P$. In all of these three prefixes, k is bound in the body. $k(x).P$ receives a value from $\overline{k}\langle e\rangle.Q$ via k; and $k \triangleright [l_i : P_i]_{i\in I}$ (with I finite) waits with $\{l_i\}_{i\in I}$-labelled branches from which $k \triangleleft l.P$ selects one. $P \mid Q$ is a parallel composition and $(\nu u)P$ is a hiding. A recursive process $(\mathbf{rec}X(\tilde{x}).P)\langle\tilde{e}\rangle$ consists of a recursive definition $(\mathbf{rec}X(\tilde{x}).P)$ and actual parameters $\tilde{e}$. In $\mathbf{rec}X(\tilde{x}).P$, a process variable X and formal parameters $\tilde{x}$ are binders. $\mathsf{fn}(P)$ denotes the free channels in P. We often omit $\mathbf{0}$ and the empty vector. For example we write $\overline{k}$ for $\overline{k}\langle\rangle.\mathbf{0}$ and $\mathbf{rec}X.P$ for $(\mathbf{rec}X().P)\langle\rangle$.

The structural congruence $\equiv$ is standard [22,11], in which we include the unfolding rule for recursion: $(\mathbf{rec}X(\tilde{x}).P)\langle\tilde{e}\rangle \equiv P[\tilde{v}/\tilde{x}][\mathbf{rec}X(\tilde{x}).P/X]$ with $e_i \downarrow v_i$, where $e \downarrow v$ means e evaluates to v. The reduction rules are generated by:

$$\begin{array}{ll} a(k).P \mid \overline{a}(k).Q \longrightarrow (\nu k)(P \mid Q) & !a(k).P \mid \overline{a}(k).Q \longrightarrow !a(k).P \mid (\nu k)(P \mid Q) \\ k(x).P \mid \overline{k}\langle e\rangle.Q \longrightarrow P[v/x] \mid Q \quad (e \downarrow v) & k \triangleright [l_i : P_i]_{i\in I} \mid k \triangleleft l_j.Q \longrightarrow P_j \mid Q \quad (j \in I) \end{array}$$

with the standard if-then-else rules, closing under the evaluation contexts and structure rules. The first rule carries out *session initiation* via bound name passing. The second rule is for value passing and the third for branching.

As an example, a simple ATM process with an initial value 300 is given below.

$$\begin{array}{lll} \mathbf{rec}X(x).(a(k).(\mathbf{rec}Y(yk).k \triangleright [& \mathsf{balance} : \overline{k}\langle y\rangle.Y\langle yk\rangle, & \\ & \mathsf{deposit} : k(w).\overline{k}\langle y+w\rangle.Y\langle y+wk\rangle, & \\ & \mathsf{quit} : X\langle y\rangle]\ )\langle xk\rangle &)\langle 300\rangle \end{array}$$

This ATM first establishes a session identified by k; and offers three options, $\mathsf{balance}$, $\mathsf{deposit}$ and quit. If $\mathsf{balance}$ is selected, then it shows the balance of the account, and recurs with the same amount (y). If $\mathsf{deposit}$ is selected, then it receives a deposited amount w, and recurs with the new state $(y+w)$. If quit is chosen, it exits the loop and terminates the conversation. The actual parameter 300 indicates the initial balance.

Types and Typing. The grammar of types follows [11], augmented with replicated types, $(\tau)^!$ and $(\tau)^?$, from [4].

$$\begin{array}{rl} \alpha ::= & \texttt{nat} \mid \texttt{bool} \mid (\tau) \mid (\tau)^! \mid (\tau)^? \mid \mathbf{rec}\,t.\alpha \mid t \\ \tau ::= & \downarrow\alpha;\tau \mid \uparrow\alpha;\tau \mid \&\{l_i : \tau_i\}_{i\in I} \mid \oplus\{l_i : \tau_i\}_{i\in I} \mid \mathbf{rec}\,t.\tau \mid \mathsf{end} \mid t \mid \bot \end{array}$$

We call α a *shared type*, which consists of *non-deterministic type* (τ); *server type* $(\tau)^!$ and *client types* $(\tau)^?$ together called *replicated types*; *atomic type* `nat` and `bool`; recursive type $\mathbf{rec}\,t.\tau$; and a type variable. We take an *equi-recursive* view of types, not distinguishing between a type $\mathbf{rec}\,t.\alpha$ and its unfolding $\alpha[\mathbf{rec}\,t.\alpha/t]$. τ is a *linear type*. Type $\downarrow\alpha;\tau$ represents first inputting a value of type α, then performing the actions typed by τ; type $\uparrow\alpha;\tau$ is its dual. Type $\&\{l_i:\tau_i\}_{i\in I}$ represents waiting with n options, and behaves as τ_i if the i-th action is selected; type $\oplus\{l_i:\tau_i\}_{i\in I}$ is its dual. Type end represents inaction and is often omitted. $\perp$ indicates that no further connection is possible at a given channel. The *dual type* of α is defined by exchanging ! and ?, $\uparrow$ and $\downarrow$, and & and $\oplus$. Atomic types, (τ), end, and t are self-dual.

The partial commutative and associative operator $\odot$ [22,4], which controls a parallel composition, is defined by: (1) $\tau\odot\overline{\tau}=\perp$; (2) $\alpha\odot\alpha=\alpha$ if $\overline{\alpha}=\alpha$; and (3) $(\tau)^!\odot(\overline{\tau})^?=(\tau)^!$ and $(\tau)^?\odot(\tau)^?=(\tau)^?$. (1) says that once we compose two processes at a linear channel, the channel is no longer composable. (3) says a server should be unique, while an arbitrary number of clients can request interactions. Δ_0 and Δ_1 are *compatible*, written $\Delta_0\asymp\Delta_1$, if $\Delta_0(u)\odot\Delta_1(u)$ is defined for each $u\in\mathsf{dom}(\Delta_0)\cap\mathsf{dom}(\Delta_1)$; $\Delta_i(u)=\alpha$, then $u\in\mathsf{dom}(\Delta_j)$; and process variables are disjoint. If $\Delta_0\asymp\Delta_1$, we set $\Delta_0\odot\Delta_1=\{(\Delta_0\odot\Delta_1)(u)\mid u\in\mathsf{dom}(\Delta_0)\cap\mathsf{dom}(\Delta_1)\}\cup\Delta_0\setminus\mathsf{dom}(\Delta_1)\cup\Delta_1\setminus\mathsf{dom}(\Delta_0)$.

Typing environments $\Gamma,\Delta,\ldots$ are given by $\Gamma::=\emptyset\mid\Gamma,a:\alpha\mid\Gamma,X:\tilde{\alpha}\tilde{\tau}\mid\Gamma,k:\tau$. The typing judgement for process P is given as $\Gamma\vdash P$. The typing rules are identical with [22,11] for linear/non-deterministic types, augmented with the typing for replicated types from [4] (allowing only client typed channels to be free under a replicated prefix). We only list the following rule for parallel composition.

$$\Gamma_i\vdash P_i \text{ with } i=1,2 \text{ and } \Gamma_1\asymp\Gamma_2, \text{ then } \Gamma_1\odot\Gamma_2\vdash P_1\mid P_2$$

As an example, session channel k in ATM is typed by:

$$\tau \;=\; \mathbf{rec}\,t.\&\{\mathsf{balance}:\uparrow\mathtt{nat};t,\ \mathsf{deposit}:\downarrow\mathtt{nat};\uparrow\mathtt{nat};t,\ \mathsf{quit}:\mathsf{end}\}$$

The same session from the user's viewpoint is typed dually as $\overline{\tau}=\mathbf{rec}\,t.\oplus\{\mathsf{balance}:\downarrow\mathtt{nat};t,\ \mathsf{deposit}:\uparrow\mathtt{nat};\downarrow\mathtt{nat};t,\ \mathsf{quit}:\mathsf{end}\}$, composable with τ by $\odot$.

Bisimilarity and Testing. *Transition labels* $(\ell,\ell',..)$ are given by the grammar:

$$\ell \quad ::= \quad \tau \mid a(k) \mid \overline{a}(k) \mid kv \mid \overline{k}v \mid k(a) \mid \overline{k}(a) \mid k\triangleright l \mid k\triangleleft l$$

where k and a in (k) and (a) introduce binding. ℓ is *shared* if it has shape $a(k)$ or $\overline{a}(k)$; *linear* if it is neither shared nor τ. We write $\overline{\ell}$ for the dual of ℓ, defined by exchanging the input and output (for example $\overline{a(k)}=\overline{a}(k)$). $\overline{\tau}$ is undefined. We use the standard early transition relation augmented with $k\triangleleft l.P\xrightarrow{k\triangleleft l}P$ and $k\triangleright[l_i:P_i]_{i\in I}\xrightarrow{k\triangleright l_j}P_j\ (j\in I)$. The *typed early transition* is defined by setting $\Gamma\vdash P\xrightarrow{\ell}\Gamma\setminus\ell\vdash Q$ if $P\xrightarrow{l}Q$ and if the operation $\Gamma\setminus\ell$ is defined, where $\Gamma\setminus\ell$ is defined if ℓ conforms to Γ, in which case $\Gamma\setminus\ell$ denotes the resulting environment. For example, assuming $\Gamma=\Delta,k:\&\{l_i:\tau_i\}_{i\in I}$, if $l=l_j\ (j\in I)$ then $\Gamma\setminus k\triangleright l=\Delta,k:\tau_j$; otherwise $\Gamma\setminus k\triangleright l_j$ is undefined. We often leave Γ and Δ implicit. $\Longrightarrow$ stands for a reflexive and transitive closure of $\xrightarrow{\tau}$. We define the *early weak bisimilarity*, the *weak May preorder* and the *(divergence-insensitive) weak Must preorder* in the standard way, written $\approx$, $\sqsubseteq_{may}$ and $\sqsubseteq_{must}$, respectively.

3 Assertions

A Logical Language. Our logical language is Hennessy-Milner logic with equality, value/name passing modality and fixed point formulae [2,8], augmented with new operators. The grammar of assertions ($A, B, C, \ldots$) follows.

$$\begin{array}{rl} A ::= & e_1 = e_2 \mid A \wedge B \mid \neg A \mid \forall x^{\rho}.A \mid \langle\!\langle\, \rangle\!\rangle A \mid \langle \ell \rangle A \mid (\mu X(\tilde{x}).A)\langle \tilde{e} \rangle \mid X\langle \tilde{e} \rangle \\ & \mid \nu x^{\rho}.A \mid A \circ B \mid A \rhd B \end{array}$$

Above ℓ ranges over $a(k)$, $\overline{a}(k)$, $k\langle e \rangle$, $\overline{k}\langle e \rangle$, $k \rhd l_i$ and $k \lhd l_i$. ρ stands for either α or τ. We define $A \vee B$, $A \supset B$, $\exists x^{\rho}.A$, $[\ell]A$, $[\![\,]\!]A$, and $(\nu X(\tilde{x}).A)\langle \tilde{e} \rangle$, by dualisation.

$\langle \ell \rangle A$ says that the process has some immediate, or strong, ℓ action, satisfying A as the result. $\langle\!\langle\, \rangle\!\rangle A$ says that after some sequence of zero or more silent actions, the process will satisfy A (dually, in $[\![\,]\!]A$, after whatever zero or more silent actions, the process will satisfy A). We write $\langle\!\langle \ell \rangle\!\rangle A$ for $\langle\!\langle\, \rangle\!\rangle \langle \ell \rangle \langle\!\langle\, \rangle\!\rangle A$, saying that some weak ℓ-transition leads to A. Dually $[\![\ell]\!]A$ says that any weak ℓ-transition ends up satisfying A. The combination of strong and weak modalities is important for proof systems and axioms.

The minimal and maximal fixed points use parameters following [2,8], which are essential *for describing* state-changing *loops*, as in the ATM example. We assume that $X\langle \tilde{e} \rangle$ never occurs in A negatively (the assumption part of $\rhd$ is contravariant) [8].

$A \circ B$ (read as "*A par B*") is understood as A, B in [21]. Informally, a process $\Gamma \vdash P$ satisfies $A \circ B$ when $\Gamma \vdash P$ has the same observable behaviour as $Q|R$, together typed under Γ, such that Q satisfies A and R satisfies B. This puts typing constraints on A and B: if A and B have minimal typings Δ and Δ', we demand $\Delta \asymp \Delta'$ and $\Delta \odot \Delta' \subset \Gamma$.

$A \rhd B$ (read as "*rely A then B*") is a typed version of the consequence relation studied in [20]. A process $\Gamma \vdash P$ satisfies $A \rhd B$ if, for each appropriately typed Q satisfying A, $P|Q$ satisfies B. Again this constrains the typing of A and B: if A has the minimal typing Δ, we demand $\Gamma \asymp \Delta$ and that B is typed under $\Gamma \odot \Delta$. For example, for $\Gamma \vdash P$ with $\Gamma(k) = \overline{\tau}$ to satisfy $B \rhd C$, k can be typed as τ in B, and, if so, k is typed $\bot$ in C.

$\nu x^{\rho}.A$ is the quantifier for name hiding. A process, say P, satisfies $\nu x^{\rho}.A$ if there is a fresh name u of type ρ and P' such that $(\nu\, u)P' \approx P$ and P' satisfies A. Its logical nature differs substantially from $\exists$, as studied in [25].

We often omit type annotations for quantifiers. T denotes $1 = 1$, F its negation. The standard association of operators is assumed, e.g. $\forall x.A \wedge B \supset C$ is parsed as $((\forall x.A) \wedge B) \supset C$ ($\circ$, $\rhd$, $\nu x.A$ associate as $\wedge$, $\supset$, $\exists x.A$). We use the following notation:

Definition 1 (mixed modality). $(\!|\ell|\!)A \ = \ [\![\,]\!](\langle \ell \rangle \mathsf{T} \wedge [\ell]A)$.

The modal formula $(\!|\ell|\!)A$ (read: "surely ℓ then A") says that now or after any silent actions the process may have, it can do a strong ℓ-action, and then it satisfies A.

Examples of Assertions. We illustrate $\circ$ and $\rhd$ using a simple example.

$$P \equiv !b(k).k(x).\overline{k}\langle x+1 \rangle.\mathbf{0} \qquad\qquad Q \equiv \overline{b}(k).\overline{k}\langle 2 \rangle.k(y).\overline{h}\langle y \rangle.\mathbf{0}$$

P accepts a session request, receives a number and returns its increment: Q requests a session, sends 2 and receives and forwards the result to h. P and Q are typed under $b : (\downarrow\mathtt{nat}; \uparrow\mathtt{nat}; \mathtt{end})^{!}$ and $b : (\uparrow\mathtt{nat}; \downarrow\mathtt{nat}; \mathtt{end})^{?}, h : \uparrow\mathtt{nat}; \mathtt{end}$, respectively.

We now assert for P and Q and their composition. First for P and Q individually:

$$A \;=\; \forall x^{\mathtt{nat}}.\langle\!\langle b(k)\rangle\!\rangle\langle\!\langle kx\rangle\!\rangle\langle\!\langle \overline{k}x+1\rangle\!\rangle\mathsf{T} \quad B \;=\; \forall y^{\mathtt{nat}}.\langle\!\langle \overline{b}(k)\rangle\!\rangle\langle\!\langle \overline{k}2\rangle\!\rangle\langle\!\langle ky\rangle\!\rangle\langle\!\langle \overline{h}y\rangle\!\rangle\mathsf{T}$$

From this we assert $A\circ B$ for $P|Q$. Since $A\circ B \;\supset\; \langle\!\langle \overline{h}3\rangle\!\rangle\mathsf{T}$ (by the axioms in Section 4 later), we know $P|Q$ can emit 3 via h. From this entailment we also know Q satisfies $A \rhd \langle\!\langle \overline{h}3\rangle\!\rangle\mathsf{T}$, i.e. when composed with any behaviour satisfying A, it can emit 3 via h.

Above we only used the May modality. In fact, we can strengthen A and B using the mixed modality (cf. Definition 1) as follows.

$$A' \;=\; \forall x^{\mathtt{nat}}.\llparenthesis b(k)\rrparenthesis\llparenthesis kx\rrparenthesis\llparenthesis \overline{k}x+1\rrparenthesis\mathsf{T} \qquad\qquad B' \;=\; \forall y^{\mathtt{nat}}.\llparenthesis \overline{b}(k)\rrparenthesis\llparenthesis \overline{k}2\rrparenthesis\llparenthesis ky\rrparenthesis\llparenthesis \overline{h}y\rrparenthesis\mathsf{T}$$

We can then show that $A'\circ B'$ entails $\llparenthesis \overline{h}3\rrparenthesis\mathsf{T}$, hence $P|Q$ surely emits 3 via h. This entailment depends on the type of b: if b's type is non-deterministic, e.g. $b : (\downarrow\mathtt{nat};\uparrow\mathtt{nat};\mathtt{end})$, then this assertion can*not* be derived (as discussed in Proposition 4 later).

Next we consider a specification of the simple ATM, given as:

$$\llparenthesis a(k)\rrparenthesis\,(\nu Y(yk).\llparenthesis k\rhd\mathsf{balance}\rrparenthesis\llparenthesis \overline{k}y\rrparenthesis Y\langle yk\rangle)\langle 300k\rangle \tag{3.1}$$

The assertion says the process is ready to receive a session request via a: then it enters a loop, and, if asked to show a balance, it shows y, and recurs. The initial balance is 300. Now a user of ATM may satisfy: $\forall x.\llparenthesis \overline{a}(k)\rrparenthesis\llparenthesis k\lhd\mathsf{balance}\rrparenthesis\llparenthesis kx\rrparenthesis\llparenthesis \overline{h}x\rrparenthesis\mathsf{T}$. which, when combined with (3.1) by $\circ$, gives us $\langle\!\langle \overline{h}300\rangle\!\rangle\mathsf{T}$. In contrast to the previous example, we can*not* derive $\llparenthesis \overline{h}300\rrparenthesis\mathsf{T}$ since another user may interfere at the shared channel a before this user. This distinction will be formally underpinned in Proposition 4 later.

Semantics of Assertions. The interpretation of assertions follows [8], extended to the typed setting. We list the key points. First, a *property* (written $p,q,..$) is a set of typed processes under an identical typing which are without free value/process variables and which are closed under $\approx$. We define operators on properties as:

$$\begin{array}{llll} p|q \;=\; \bigcup_{P\in p,Q\in q}[P|Q]_{\approx} & & (\nu u)p \;=\; \bigcup_{P\in p}[(\nu u)P]_{\approx} \\ \langle\!\langle\,\rangle\!\rangle p' \;=\; \{P \mid P\Longrightarrow P'\in p'\} & & \langle\ell\rangle p' \;=\; \{P \mid P\approx P_0 \xrightarrow{\ell} P'\in p'\} \end{array}$$

A *parametrised property of type* $\tilde{\rho}$ (written $f,g,\ldots$) is a function which maps a vector of values typed $\tilde{\rho}$ to a property. An interpretation of variables ($\xi,\xi',..$) follows [8], mapping a variable to a value and an assertion variable to a parametrised property. Given $\Gamma\vdash A$ where Γ types the free channels in A, the *interpretation of* $\Gamma\vdash A$ *under* ξ, written $\langle\!\langle\Gamma\vdash A\rangle\!\rangle\xi$, or $\langle\!\langle A\rangle\!\rangle\xi$ if Γ is known from the context, is given by the standard clauses for equality, conjunction, universal quantifier, negation and assertion variable, augmented with the following clauses. For modality, we set:

$$\langle\!\langle\Gamma\vdash\langle\!\langle\,\rangle\!\rangle A\rangle\!\rangle\xi = \langle\!\langle\,\rangle\!\rangle\langle\!\langle\Gamma\vdash A\rangle\!\rangle\xi, \quad \langle\!\langle\Gamma\vdash\langle\ell\rangle A\rangle\!\rangle\xi = \langle\ell\rangle\langle\!\langle\Gamma\setminus\ell\vdash A\rangle\!\rangle\xi$$

where $\Gamma\setminus\ell$ adds a mapping w.r.t. ℓ. For $\circ$, $\rhd$ and ν we set:

$$\langle\!\langle\Gamma\vdash A\circ B\rangle\!\rangle\xi = \bigcup_{\Delta\odot\Theta=\Gamma}\langle\!\langle\Delta\vdash A\rangle\!\rangle\xi\,|\,\langle\!\langle\Theta\vdash B\rangle\!\rangle\xi \quad \langle\!\langle\Gamma\vdash\nu x^{\rho}.A\rangle\!\rangle\xi = (\nu u)\langle\!\langle\Gamma,u:\rho\vdash A\rangle\!\rangle(\xi\cdot x\mapsto u)$$
$$\langle\!\langle\Gamma\vdash A\rhd B\rangle\!\rangle\xi = \mathsf{max}\; p^{\Gamma}.\;((p|\;\langle\!\langle\Delta\vdash A\rangle\!\rangle\xi)\;\subset\;\langle\!\langle\Delta\odot\Gamma\vdash B\rangle\!\rangle\xi)$$

Above $\mathsf{max}\; p^{\Gamma}.\mathcal{P}$ denotes the maximum property (by set inclusion) typed under Γ which satisfies $\mathcal{P}$. The following clause for μ-recursion is from [8].

$$\langle\!\langle\Gamma\vdash(\mu X(\tilde{x}).A)\langle\tilde{e}\rangle\rangle\!\rangle\xi \;=\; (\mathsf{fix}\,\lambda f.\lambda\tilde{v}.(\langle\!\langle A\rangle\!\rangle(\xi\cdot X\mapsto f\cdot\tilde{x}\mapsto\tilde{v})))(\xi(\tilde{e}))$$

where fix is the least fixed point and $\xi(e)$ is the interpretation of e under ξ.

$$\frac{E \vdash P \blacktriangleright A}{E \vdash a(k).P \blacktriangleright \langle\!\langle a(k) \rangle\!\rangle A} \quad \frac{E \vdash P \blacktriangleright A}{E \vdash \overline{a}(k).P \blacktriangleright \langle\!\langle \overline{a}(k) \rangle\!\rangle A} \quad \frac{E \vdash P \blacktriangleright A}{E \vdash !a(k).P \blacktriangleright \langle\!\langle a(k) \rangle\!\rangle A} \quad \frac{E \vdash P \blacktriangleright A}{E \vdash \overline{a}(k).P \blacktriangleright \langle\!\langle \overline{a}(k) \rangle\!\rangle A}$$

Acc,Req,Ser,CReq

$$\frac{E \vdash P \blacktriangleright A}{E \vdash k(x).P \blacktriangleright \forall x.\langle\!\langle kx \rangle\!\rangle A} \quad \frac{E \vdash P \blacktriangleright A}{E \vdash \overline{k}\langle e \rangle.P \blacktriangleright \langle\!\langle \overline{k}e \rangle\!\rangle A} \quad \frac{E \vdash P_i \blacktriangleright A_i \quad i = 1,2}{E \vdash P_1 \mid P_2 \blacktriangleright A_1 \circ A_2} \quad \frac{E \vdash P \blacktriangleright A \quad x \text{ fresh}}{E \vdash (\nu u)P \blacktriangleright \nu x.A[x/u]}$$

Rcv,Send,Conc, Res

$$\frac{E \vdash P_i \blacktriangleright A_i \quad \forall i \in I}{E \vdash k \triangleright [l_i : P_i]_{i \in I} \blacktriangleright \bigwedge_{i \in I} \langle\!\langle k \triangleright l_i \rangle\!\rangle A_i} \quad \frac{E \vdash P \blacktriangleright A_j}{E \vdash k \triangleleft l_j.P \blacktriangleright \langle\!\langle k \triangleleft l_j \rangle\!\rangle A_j} \quad \frac{-}{E \vdash \mathbf{0} \blacktriangleright \mathsf{T}} \qquad \text{Bra,Sel,Inact}$$

$$\frac{-}{E, X : (\tilde{x})A \vdash X\langle \tilde{e} \rangle \blacktriangleright A[\tilde{e}/\tilde{x}]} \quad \frac{E, X : (\tilde{x})(\forall j \lneq i.A(j)) \vdash P \blacktriangleright A(i)}{E \vdash (\mathbf{rec}X.(\tilde{x}).P)\langle \tilde{e} \rangle \blacktriangleright \forall i.A(i)[\tilde{e}/\tilde{x}]} \qquad \text{Var,Rec-ind}$$

$$\frac{E \vdash P_1 \blacktriangleright e \supset A \quad E \vdash P_2 \blacktriangleright \neg e \supset A}{E \vdash \texttt{if}\ e\ \texttt{then}\ P_1\ \texttt{else}\ P_2 \blacktriangleright A} \quad \frac{E \vdash P \blacktriangleright A \quad A \supset B}{E \vdash P \blacktriangleright B} \qquad \text{If, Conseq}$$

Fig. 1. Proof System (the May Modality)

4 Proof Rules, Axioms and Completeness

Rules for the May Modality. Write $\Gamma; E \vdash P \blacktriangleright A$ for the provability judgement where Γ types P and A (except auxiliary variables in A) and E contains assignments of the form $X : (\tilde{x})A$, mapping a process variable to a parametrised formula ($\tilde{x}$ are binders). We often write $E \vdash P \blacktriangleright A$, leaving Γ implicit. We consider three systems, one for the May modality, one for Must, and one for their combination. They soundly and completely characterise the May/Must preorders and bisimilarity, respectively.

The proof rules for the May modality are given in Figure 1. There is a single rule for each typing rule, except that Conseq has no corresponding rules. The typing is not mentioned, assuming it follows the typing rules. The first eight rules are standard (Ser does not use a fixed point, which suffices due to the semantics of replication, cf. Proposition 4 (6) later). Conc and Res hide complexity of process composition under $\circ$ and ν, which is to be unfolded by the axioms for these operators.

Inact and Var are standard. In Rec-ind, we assume i, j are in some well-ordered set [10]. We make this rule applicable to fixed point operators by introducing the notation $(\mu/\nu X^{\kappa}(\tilde{x}).A)\langle \tilde{e} \rangle$ from [8], with κ ranging over ordinals. The notation stands for the standard approximant to the least fixed point, given as: $(\mu X^{0}(\tilde{x}).A)\langle \tilde{e} \rangle \equiv \mathsf{F}$, $(\mu X^{\kappa+1}(\tilde{x}).A)\langle \tilde{e} \rangle \equiv A[(\mu X^{\kappa}(\tilde{x}).A)/X][\tilde{e}/\tilde{x}]$, and $(\mu X^{\lambda}(\tilde{x}).A)\langle \tilde{e} \rangle \equiv \exists_{i \lneq \lambda}(\mu X^{i}(\tilde{x}).A)\langle \tilde{e} \rangle$ with λ a limit ordinal. Dually for ν-recursion. For example, via this notation, an inference for $(\mathbf{rec}X(k).\overline{k}1.X\langle k \rangle)\langle k \rangle$ is given as follows, setting $A(i) = \nu Y^{i}(k).\langle\!\langle \overline{k}1 \rangle\!\rangle Y\langle k \rangle$.

$$\frac{X : (k)\forall j \lneq i.A(j) \vdash \overline{k}1.X\langle k \rangle \blacktriangleright A(i)}{\vdash (\mathbf{rec}X(k).\overline{k}1.X\langle k \rangle)\langle k \rangle \blacktriangleright (\nu Y(k).\langle\!\langle \overline{k}1 \rangle\!\rangle Y\langle k \rangle)\langle k \rangle}$$

Using higher ordinals becomes necessary when we have a lexicographic ordering, as with the behaviour with nested recursions.

The conditional rule is standard. The final proof rule is the consequence rule as found in Hoare logic.

Rules for the Must and Mixed Modalities. The May proof rules ensure that a process *can* reach a certain state: in contrast, the Must rules ensure that a process *cannot* reach a certain state. We first define the abbreviation $\mathsf{noact}(\Gamma)$, which says: "no actions at $\mathsf{dom}(\Gamma)$ are possible". Let $\mathsf{noact}(k\!:\downarrow\alpha;\tau) = \forall x^{\alpha}.\llbracket kx\rrbracket\mathsf{F}$, $\mathsf{noact}(k\!:\&\{l_i : \tau_i\}) = \wedge_i\llbracket k\triangleright l_i\rrbracket\mathsf{F}$, $\mathsf{noact}(k\!:\mathtt{end}) = \mathsf{noact}(x\!:\mathtt{nat}) = \mathsf{noact}(x\!:\mathtt{bool}) = \mathsf{T}$ and similarly for outputs and shared names. Set $\mathsf{noact}(\tilde{u} : \tilde{\rho}) = \wedge_i\mathsf{noact}(u_i : \rho_i)$. We then write $\llbracket\ell,\Gamma\rrbracket A$ for $\llbracket\ \rrbracket([\ell]A \wedge \mathsf{noact}(\Gamma))$ with $\ell \neq \tau$, which says: "ℓ is the only action possible and if it ever happens then A follows". Using this predicate, the proof system for the Must modality is given by replacing $\langle\!\langle\ell\rangle\!\rangle$ in each prefix rule in Figure 1 with $\llbracket\ell,\Delta\rrbracket$, where Δ is the typing of a process minus that of ℓ; and for Inact, replacing T with $\mathsf{noact}(\Gamma)$, assuming Γ is the implicit typing. Other rules stay unchanged, except for adding:

$$\frac{E, X : (\tilde{x})A \vdash P \blacktriangleright A \quad A \text{ admissible}}{E \vdash (\mathbf{rec}\,X(\tilde{x}).P)\langle\tilde{e}\rangle \blacktriangleright A[\tilde{e}/\tilde{x}]} \qquad \mathsf{Rec\text{-}adm}$$

where admissibility is defined via syntactic unfoldings [10]. Given $R \equiv (\mathbf{rec}\,X(\tilde{x}).P)\langle\tilde{e}\rangle$, let $P^0 \equiv \mathbf{0}$ and $P^{n+1} \equiv P[(\tilde{x})P^n/X]$ (where we set $((\tilde{x})Q)\langle\tilde{e}\rangle = Q[\tilde{e}/\tilde{x}]$). Then a closed formula A is *admissible* if: (1) P^0 satisfies A; and (2) If P_i satisfies A for each $i \geq 0$, then $(\mathbf{rec}\,X(\tilde{x}).P)\langle\tilde{x}\rangle$ also satisfies A. This is extended to open formulae closing under admissible properties. In practice, we may use a tractable variant of admissibility: for example, if we restrict P to be sequential (i.e. without parallel composition), there is a simple syntactic characterisation of admissibility.

To capture both modalities in a single proof system, we strengthen the Must prefix rules through the use of the combined modality $(\!|\ell,\Delta|\!)A$, which stands for $(\!|\ell|\!)A \wedge \mathsf{noact}(\Delta)$ (cf. Definition 1). The proof system is given by replacing $\langle\!\langle\ell\rangle\!\rangle$ in each prefix rule in Figure 1 with $(\!|\ell,\Delta|\!)$, fully capturing the semantics of prefix. Other rules remain identical except for adding the following recursion rule, due to Larsen [13].

$$\frac{E, X : (\tilde{x})X'\langle\tilde{x}\rangle \vdash P \blacktriangleright A}{E \vdash (\mathbf{rec}\,X(\tilde{x}).P)\langle\tilde{e}\rangle \blacktriangleright (\mu X'(\tilde{x}).A)\langle\tilde{e}\rangle} \qquad \mathsf{Rec\text{-}mix}$$

Soundness and Relative Completeness. Let us write $\Gamma;E \vdash_{may} P \blacktriangleright A$, $\Gamma;E \vdash_{must} P \blacktriangleright A$ and $\Gamma;E \vdash_{mix} P \blacktriangleright A$, for provability in the May/Must/Mixed proof systems, respectively. We also write $\Gamma;E \models P \blacktriangleright A$ (read: $\Gamma \vdash P$ *satisfies* A *under* E), when we have $P \in \langle\!\langle\Gamma \vdash A\rangle\!\rangle(\xi \cdot \langle\!\langle E\rangle\!\rangle\xi)$ for each ξ, where $\langle\!\langle E\rangle\!\rangle\xi$ is the obvious interpretation of process variables under E and ξ. We first observe:

Theorem 2 (soundness). $\Gamma;E \vdash_{may} P \blacktriangleright A$ *implies* $\Gamma;E \models P \blacktriangleright A$*, similarly for* $\Gamma;E \vdash_{must} P \blacktriangleright A$ *and* $\Gamma;E \vdash_{mix} P \blacktriangleright A$*.*

Thus the three proof systems are all sound under the same satisfaction relation, allowing the mixed use of their proof rules in reasoning. Further each system precisely captures a distinct process semantics, as shown by the following completeness result. The proof is by syntactically deriving characteristic formulae, which also entails observational and descriptive completeness in the sense of [10].

Theorem 3 (completeness). *Let $\Gamma \vdash P$ and A be closed. Then $\models P \blacktriangleright A$ with A being an upper-closed property w.r.t.$\sqsubseteq_{may}$ (resp. a downward-closed property w.r.t. $\sqsubseteq_{must}$) implies $\vdash_{may} P \blacktriangleright A$ (resp. $\vdash_{must} P \blacktriangleright A$). Further for any A, if $\models P \blacktriangleright A$ then $\vdash_{mix} P \blacktriangleright A$.*

Basic Axioms. The operators $\circ$ and ν, used in the proof rules, do not directly describe the communication behaviour of a process: It is through the axioms of the assertion language that modal behaviours are extracted. Some of the basic axioms follow.

Proposition 4. *Below we assume well-typedness of formulae.*

1. $B \supset (A \rhd (A \circ B))$, $A \rhd (B \rhd C) \equiv (A \circ B) \rhd C$ *and* $(A \circ (A \rhd B)) \supset B$.
2. $A \circ B \equiv A \wedge B$ *if* $\mathsf{fn}(A) \cap \mathsf{fn}(B) = \emptyset$ *and all free channels are server typed.*
3. $(\llparenthesis \ell \rrparenthesis A) \circ B \equiv \llparenthesis \ell \rrparenthesis (A \circ B)$ *and* $\llparenthesis \ell \rrparenthesis A \circ \llparenthesis \overline{\ell} \rrparenthesis B \equiv \nu \mathsf{bn}(\ell).(A \circ B)$, *with ℓ linear.*
4. $\llparenthesis \ell \rrparenthesis A \circ \llparenthesis \overline{\ell} \rrparenthesis B \supset (\langle\!\langle \ell \rangle\!\rangle (A \circ \llparenthesis \overline{\ell} \rrparenthesis B) \wedge \langle\!\langle\, \rangle\!\rangle (A \circ B) \wedge \langle\!\langle \overline{\ell} \rangle\!\rangle (\llparenthesis \ell \rrparenthesis A \circ B))$
5. $\llparenthesis a(k) \rrparenthesis A \circ \llparenthesis \overline{a}(k) \rrparenthesis B \equiv \llparenthesis a(k) \rrparenthesis A \circ \nu k.(A \circ B)$, *where a has a server type.*
6. $(\nu X(\tilde{x}).A)\langle \tilde{e} \rangle \circ (\nu Y(\tilde{y}).B)\langle \tilde{g} \rangle \supset (\nu Z(\tilde{x}\tilde{y}).C[Z\langle \tilde{e}\tilde{g} \rangle]_i)\langle \tilde{e}\tilde{g} \rangle$ *where* $(A \circ B \supset C[X\langle \tilde{e} \rangle \circ Y\langle \tilde{g} \rangle]_i)$ *is valid and* $C[X\langle \tilde{e} \rangle_i \circ Y\langle \tilde{g} \rangle_i]_{i \in I}$ *denotes a formula with multiple holes indexed by I, assuming all occurrences of X and Y are thus exhausted.*

The three axioms in (1) relate $\rhd$ and $\circ$. In (2), $\mathsf{fn}(A)$ is the set of names and variables of channel types. In (3), the second axiom eliminates dual actions. In (4) the prefixing $\langle\!\langle\, \rangle\!\rangle$ cannot be removed due to state change, unlike (2). In (5), the axiom relies on a having a server type, corresponding to the replication law in [16,24]. In the Server-Client example in Section 3, if we type b with a non-deterministic type, we cannot apply this axiom, hence cannot derive $\llparenthesis \overline{h}3 \rrparenthesis \mathsf{T}$. In (6), A and B indicate well-synchronised recursive interactions, in which case we can merge their states under recursions.

Elimination of $\circ$ and ν. Through these and other axioms, we can transform formulae into those without $\circ$ and ν. We discuss a basic result for such elimination, using deterministic type disciplines from [4,24] (the typing in [4] ensures determinacy, to which [24] adds a causality constraint to ensure strong normalisation: essentially the same result holds for processes in Section 2 without non-deterministic types). We extend $\langle a(k) \rangle$ to $\langle a\tilde{b}(k) \rangle$ (dually for output) since, in [4,24], a server channel (a) carries not only a linear channel (k) but client-typed channels ($\tilde{b}$). We also replace the use of bisimilarity in Section 3 with the standard reduction-based congruence [4,24], denoted $\cong$, which adds semantic precision. In correspondence, we refine the interpretation of equality and quantification over server-typed names. Below let α be server-typed.

$$\langle\!\langle \Gamma \vdash e_1^{\alpha} = e_2^{\alpha} \rangle\!\rangle \xi = \{\Gamma \vdash P \mid P[\xi(e_1)\xi(e_2)/\xi(e_2)\xi(e_1)] \cong P\}$$

$$\langle\!\langle \Gamma\xi \vdash \forall x^{\alpha}.A \rangle\!\rangle \xi = \max p^{\Gamma\xi}.(\forall q^{u:\alpha}.p|q \subset \langle\!\langle \Gamma, x:\alpha \vdash A \rangle\!\rangle (\xi \cdot x \mapsto u))$$

The first clause says that two replicated channels are equal if the corresponding behaviours are. Together these clauses treat replicated channels as the behaviours they represent, while maintaining the standard axioms for equality and quantifiers. Their significance will become clear when we discuss logical full abstraction in Section 5. The same proof systems satisfy completeness for $\cong$ and the corresponding precongruences.

Now let us say A is $\circ$-*free* (resp. ν-*free*) if $\circ$ (resp. ν) does not occur in A. A is *approximately* $\circ$-*free* if $\circ$ occurs only in fixed point formulae whose finite unfoldings are $\circ$-free up to logical equivalence. We also say A *characterises* P when $\Gamma \models P \blacktriangleright A$ and, moreover, whenever $\Gamma \models Q \blacktriangleright A$ we have $P \cong Q$.

Theorem 5 (elimination of $\circ$ and ν under determinism). *Let P be typable by the type discipline in [4] (resp. [24]). Then there is an algorithm to find a ν-free and approximately $\circ$-free formula (resp. a ν-free and $\circ$-free formula) which characterises P.*

5 Applications

State Transfer: Synchronising Stateful Interactions. As the first reasoning example, we extend the previous ATM in Sections 2 and 3 to three-party interactions among User, ATM and Bank. Our purpose is to demonstrate how we can reason about the transfer of state induced by synchronised actions among multiple parties. ATM is extended with withdraw option, in which ATM asks Bank each time it receives a request from User, and forwards the answer to User. The π-calculus term representing this behaviour, which we call *ATM*, is given as:

$$\begin{aligned} a(k).\overline{b}(k').\mathbf{rec}\,Y.(\,k \rhd [\,&\mathsf{balance}: \overline{k'} \lhd \mathsf{balance}.k'(z).\overline{k}\langle z\rangle.Y,\\ &\mathsf{withdraw}: k(n).\overline{k'} \lhd \mathsf{withdraw}.\overline{k'}\langle n\rangle.k' \rhd [\mathsf{ok}: k \lhd \mathsf{ok}.Y, \mathsf{no}: k \lhd \mathsf{no}.Y\,],\\ &\mathsf{quit}: \overline{k'} \lhd \mathsf{quit}\,]\,) \end{aligned}$$

The new ATM no longer has its own state, dispensing with parameters in its recursion. At the same time, the state change in Bank is reflected onto ATM through interactions, so that ATM will *behave to User as if it were stateful*. In turn, User would demand the following invariance: if User withdraws money several times *within a single session*, the withdrawal of an amount n succeeds if n is within the immediately preceding balance, say z, with the resulting balance $z-n$. Below we give a specification of ATM, as seen from User, asserting this invariance. The specification $\mathsf{ATMSpec}(a,x)$, where x is an initial balance, is given as the formula $\llparenthesis a(k) \rrparenthesis \langle\!\langle\, \rangle\!\rangle (\nu Z(z).A)\langle x\rangle$ where we set A to be:

$$\llparenthesis k \rhd \mathsf{withdraw} \rrparenthesis\ \forall n.\llparenthesis kn \rrparenthesis\ (\,z \geq n \supset \llparenthesis k \lhd \mathsf{ok} \rrparenthesis Z\langle z-n\rangle\ \wedge\ z < n \supset \llparenthesis k \lhd \mathsf{no} \rrparenthesis Z\langle z\rangle\,)$$

Let $\mathsf{BankSpec}(b,x)$ be a specification for Bank given as $\llparenthesis b(k') \rrparenthesis (\nu Z(z).B)\langle x\rangle$ where $B = A[k'/k]$ with k' fresh in A. We now show:

$$\mathit{ATM} \models \mathsf{BankSpec}(b,300)\ \rhd\ \mathsf{ATMSpec}(a,300)$$

To reach this judgement, we start from a formula directly derived by the proof rules, which we call $\mathsf{ATMSpec}_0(a,b)$, defined as $\llparenthesis a(k) \rrparenthesis \llparenthesis \overline{b}(k') \rrparenthesis \nu Y.A_0$ where we set A_0 to be:

$$\llparenthesis k \rhd \mathsf{withdraw} \rrparenthesis \llparenthesis k' \lhd \mathsf{withdraw} \rrparenthesis\ \forall n.\llparenthesis kn \rrparenthesis \llparenthesis \overline{k}'n \rrparenthesis (\ \llparenthesis k' \rhd \mathsf{ok} \rrparenthesis \llparenthesis k \lhd \mathsf{ok} \rrparenthesis .Y \wedge \llparenthesis k' \rhd \mathsf{no} \rrparenthesis \llparenthesis k \lhd \mathsf{no} \rrparenthesis .Y\ )$$

It thus suffices to show $\mathsf{ATMSpec}_0(a,b) \circ \mathsf{BankSpec}(b,300) \supset \mathsf{ATMSpec}(a,300)$. We first calculate $A_0\ \circ\ B \supset A$ by compensating all dual strong linear actions by Axiom (2) in Proposition 4. This and Axiom (6) of the same proposition give us:

$$(\nu Y.A_0)\ \circ\ (\nu Z(z).B)\langle x\rangle \quad \supset \quad (\nu Z(z).A)\langle x\rangle$$

Thus we have successfully transferred Bank's state to the specification for ATM. Finally by Axiom (4) in Proposition 4 we calculate:

$$\begin{aligned} &\llparenthesis b(k')\rrparenthesis.(\nu Z(z).B)\langle 300\rangle \circ \llparenthesis a(k)\rrparenthesis\llparenthesis \overline{b}(k')\rrparenthesis(\nu Y.A_0) \\ &\qquad \supset \llparenthesis a(k)\rrparenthesis(\llparenthesis b(k')\rrparenthesis(\nu Z(z).B)\langle 300\rangle \circ \llparenthesis \overline{b}(k')\rrparenthesis(\nu Y.A_0)) \\ &\qquad \supset \llparenthesis a(k)\rrparenthesis\langle\!\langle\ \rangle\!\rangle(\nu Z(z).A)\langle 300\rangle \end{aligned}$$

Above the logical calculation of interaction at b induces $\langle\!\langle\ \rangle\!\rangle$ in the final line, indicating a shared, hence possibly nondeterministic, interaction: in contrast, all actions *within* a session have strong modality. In this way the present framework allows specifications and reasoning about the fine-grained mixture of determinism and non-determinism.

Logical Full Abstraction of PCFv. One of the notable effects of types in the π-calculus is to enhance the semantic precision of the embedding of diverse calculi and programming languages in this calculus. When a type discipline is sufficiently strong, the embedding even enjoys full abstraction [4]. In the following we demonstrate that the proposed logic inherits this feature at a logical level. We use the complete program logic for call-by-value PCF (henceforth PCFv) from [10] and the process logic under the type discipline of [4] based on the reduction-based equality $\cong$, discussed in Section 4.

We first review PCFv and its logic. PCFv-types are either atomic types (`nat` and `bool`) or arrow types ($\alpha \Rightarrow \beta$). PCFv-terms ($M, N, \ldots$) and formulae ($A, B, \ldots$) are given by the following grammar.

$$\begin{aligned} M \ &::= \ x \mid \mathtt{op}(\tilde{M}) \mid \lambda x^{\alpha}.M \mid MN \mid \mu x^{\alpha\Rightarrow\beta}.\lambda y^{\alpha}.M \mid \mathtt{if}\ M\ \mathtt{then}\ N_1\ \mathtt{else}\ N_2 \\ A \ &::= \ e_1 = e_2 \mid A \wedge B \mid \forall x^{\alpha}.A \mid \neg A \mid x \bullet y \searrow z \end{aligned}$$

In the first line (terms), $\mathtt{op}(\tilde{M})$ denotes the standard first-order operations (including constants). In the second line (formulae), $x \bullet y \searrow z$, called *evaluation formula*, specifies that a function x, when applied to an argument y, converges and results in a value z. The semantics of these formulae exactly follows [10]. The judgement $\models [A]M :_u [B]$ intuitively says that if the free variables in M satisfy A, the program M terminates and whose result, named u, satisfies B. For its formal definition, see [10].

We use Milner's encoding of call-by-value λ-calculus [16]. Below we only show primary ones.

$$\begin{aligned} &\langle\!\langle x\rangle\!\rangle_k \ = \ \overline{k}\langle x\rangle \qquad\qquad \langle\!\langle \lambda x.M\rangle\!\rangle_k \ = \ (\nu\, a)(\overline{k}\langle a\rangle | !a(xk').\langle\!\langle M\rangle\!\rangle_{k'}) \\ &\langle\!\langle MN\rangle\!\rangle_k \ = \ (\nu\, k_1)(\langle\!\langle M\rangle\!\rangle_{k_1} | k_1(m).(\nu\, k_2)(\langle\!\langle N\rangle\!\rangle_{k_2} | k_2(n).\overline{m}\langle nk\rangle)) \end{aligned}$$

The last line uses free name passing unlike [4], following [24, §6]. The embedding of types is given accordingly [4]. For formulae, the standard constructs are mapped directly: $\langle\!\langle e_1 = e_2\rangle\!\rangle \equiv e_1 = e_2$, $\langle\!\langle A \wedge B\rangle\!\rangle \equiv \langle\!\langle A\rangle\!\rangle \wedge \langle\!\langle B\rangle\!\rangle$, $\langle\!\langle \neg A\rangle\!\rangle \equiv \neg\langle\!\langle A\rangle\!\rangle$ and $\langle\!\langle \forall x^{\alpha}.A\rangle\!\rangle \equiv \forall x.\langle\!\langle A\rangle\!\rangle$. In the first map, equality of two names in the PCFv-logic denotes equality of their denotations: to embed this notion in the process logic, we need the refinement of semantics of equality in Section 4. For evaluation formulae we set:

$$\langle\!\langle x \bullet y \searrow z\rangle\!\rangle \ \equiv \ \langle\!\langle xy(k)\rangle\!\rangle \langle\!\langle \overline{k}z\rangle\!\rangle \mathsf{T},$$

which decomposes an evaluation formula to a modal formula with the May modality (which corresponds to total correctness under determinism).

Below we say a formula A of PCFv-logic with $\mathsf{fv}(A) = \{u\}$ is *upper-closed with respect to u* [10] if, whenever V named u satisfies A, and if W is greater than V in the standard observational precongruence of PCFv, then W named u also satisfies A.

Theorem 6 (logical full abstraction of PCFv). *Let V be a well-typed closed PCFv-term and A be upper-closed with respect to u and, moreover,* $\mathsf{fv}(A) = \{u\}$. *Then we have* $\models [\top]V :_u [A]$ *if and only if* $\langle\!\langle V \rangle\!\rangle_k \models \exists u.(\langle\!\langle \overline{k}x \rangle\!\rangle \top \wedge \langle\!\langle A \rangle\!\rangle [x/u])$.

The proof uses the correspondence of characteristic formulae on both sides, observing that the May preorder and the contextual preorder coincide via the encoding of terms, and that validity in upper-closed formulae is preserved and reflected via the encoding of assertions. By translating partial correctness formulae using the Must modality, we obtain logical full abstraction for the PCFv-logic for partial correctness in [10].

References

1. Full version of this paper as a DoC technical report, Imperial College London (to appear, 2008) `www.dcs.qmul.ac.uk/~kohei/processlogic`
2. Amadio, R., Dam, M.: A modal theory of types for the π-calculus. In: Jonsson, B., Parrow, J. (eds.) FTRTFT 1996. LNCS, vol. 1135, pp. 347–365. Springer, Heidelberg (1996)
3. Berger, M.: A program logic for sequential higher-order control (1): stateless case. Typescript, 36 pages (October 2007)
4. Berger, M., Honda, K., Yoshida, N.: Sequentiality and the π-calculus. In: Abramsky, S. (ed.) TLCA 2001. LNCS, vol. 2044, pp. 29–45. Springer, Heidelberg (2001)
5. Bonsangue, M., Kurz, A.: Pi-calculus in logical form. In: LICS 2007, pp. 303–312. IEEE, Los Alamitos (2007)
6. Caires, L., Cardelli, L.: A spatial logic for concurrency. I& C 186(2), 194–235 (2003)
7. Cardelli, L., Gordon, A.D.: Anytime, anywhere: Modal logics for mobile ambients. In: POPL, pp. 365–377 (2000)
8. Dam, M.: Proof systems for pi-calculus logics. In: Logic for Concurrency and Synchronisation. Trends in Logic, Studia Logica Library, pp. 145–212. Kluwer, Dordrecht (2003)
9. Honda, K.: From process logic to program logic. In: ICFP 2004, pp. 163–174. ACM, New York (2004)
10. Honda, K., Berger, M., Yoshida, N.: Descriptive and relative completeness for logics for higher-order functions. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, pp. 360–371. Springer, Heidelberg (2006)
11. Honda, K., Vasconcelos, V.T., Kubo, M.: Language Primitives and Type Disciplines for Structured Communication-based Programming. In: Hankin, C. (ed.) ESOP 1998 and ETAPS 1998. LNCS, vol. 1381, pp. 22–138. Springer, Heidelberg (1998)
12. Jones, C.B.: Specification and design of (parallel) programs. In: IFIP Congress, pp. 321–332 (1983)
13. Larsen, K.G.: Proof systems for satisfiability in Hennessy-Milner logic with recursion. Theor. Comput. Sci. 72(2&3), 265–288 (1990)
14. Longley, J., Plotkin, G.: Logical full abstraction and PCF. In: Tbilisi Symposium on Logic, Language and Information, CLSI (1998)
15. Miller, D., Tiu, A.: A proof theory for generic judgments. ACM Transactions on Computational Logic 6(4), 749–783 (2005)

16. Milner, R.: The polyadic π-calculus: A tutorial. In: Proceedings of the International Summer School on Logic Algebra of Specification, Marktoberdorf (1992)
17. Milner, R., Parrow, J., Walker, D.: A Calculus of Mobile Processes, Parts I and II. Info.& Comp. 100(1) (1992)
18. Milner, R., Parrow, J., Walker, D.: Modal logics for mobile processes. TCS 114, 149–171 (1993)
19. Simpson, A.: Sequent calculi for process verification: Hennessy-Milner logic for an arbitrary GSOS. J. Log. Algebr. Program. 60-61, 287–322 (2004)
20. Stirling, C.: A complete compositional model proof system for a subset of CCS. In: Brauer, W. (ed.) ICALP 1985. LNCS, vol. 194, pp. 475–486. Springer, Heidelberg (1985)
21. Stirling, C.: Modal logics for communicating systems. TCS 49, 311–347 (1987)
22. Takeuchi, K., Honda, K., Kubo, M.: An Interaction-based Language and its Typing System. In: Halatsis, C., Philokyprou, G., Maritsas, D., Theodoridis, S. (eds.) PARLE 1994. LNCS, vol. 817, pp. 398–413. Springer, Heidelberg (1994)
23. Tiu, A.F.: Model checking for pi-calculus using proof search. In: Abadi, M., de Alfaro, L. (eds.) CONCUR 2005. LNCS, vol. 3653, pp. 36–50. Springer, Heidelberg (2005)
24. Yoshida, N., Berger, M., Honda, K.: Strong Normalisation in the π-Calculus. Information and Computation 191, 145–202 (2004)
25. Yoshida, N., Honda, K., Berger, M.: Logical reasoning for higher-order functions with local state. In: Seidl, H. (ed.) FOSSACS 2007. LNCS, vol. 4423, pp. 361–377. Springer, Heidelberg (2007)