Control in the π-Calculus

[Extended Abstract]

Kohei Honda
Queen Mary
kohei@dcs.qmul.ac.uk

Nobuko Yoshida
Imperial College London
yoshida@doc.ic.ac.uk

Martin Berger
Queen Mary
martinb@dcs.qmul.ac.uk

1. INTRODUCTION

This paper presents a type-preserving translation from the call-by-value $\lambda\mu$-calculus ($\lambda\mu\nu$-calculus) [23] into a typed π-calculus, and shows it is fully abstract up to natural consistent contextual congruences in respective calculi. The full abstraction is proved via an inverse transformation from the typed π-terms which inhabit the $\lambda\mu\nu$-types into the $\lambda\mu\nu$-calculus [23] (the so-called definability argument), using proof techniques based on games semantics and process calculi.

While there are different notions of control which would be represented as distinct forms of typed interactions in the π-calculus, surprisingly the full-control, the $\lambda\mu$-calculus originally introduced by Parigot [24] and whose call-by-value version is later studied by Ong-Stewart [23], has a particularly simple representation as a subset of the linear π-calculus introduced in [31]. Since we already know quite a few properties about the linear π-calculus, for example the strong normalisability is instantly derived for the subcalculus for control from our result in [31]. A tight operational correspondence assisted by the definability result, as we have shown in this paper, would open a possibility to use typed π-calculi as a tool to investigate and analyse various control structures in a uniform setting, possibly integrated with other language primitives and operational structures.

In the rest of the extended abstract, we first introduce the linear π-calculus with control, present the embedding of the call-by-value $\lambda\mu$-calculus, then outlines the definability arguments. We then discuss how the definability leads to an equational full abstraction for suitably defined behavioural equivalence for the $\lambda\mu$-calculus. The extended abstract concludes with further topics and open issues on the connection between the calculi with control and the linear/affine typed π-calculi [2, 3, 11, 31].

Categories and Subject Descriptors: F.3.2 [Semantics of Programming Lanugages]:Process models

General Terms: Theory

Keywords: Types, Control, the π-Calculus, Definability, Full Abstraction

2. PROCESSES AND TYPES

2.1 Processes

The π-calculus used in this abstract is a subset of the standard asynchronous π-calculus [21, 20, 8]. The following gives the reduction rule of this calculus.

$$x(\vec{y}).P \,|\, \overline{x}\langle\vec{v}\rangle \;\longrightarrow\; P\{\vec{v}/\vec{y}\} \tag{1}$$

Here $\vec{y}$ denotes a potentially empty vector $y_1...y_n$, $|$ denotes parallel composition, $x(\vec{y}).P$ is input, and $\overline{x}\langle\vec{v}\rangle$ is asynchronous output. Operationally, this reduction represents the consumption of an asynchronous message by a receptor. The idea extends to a receptor with a replication, $!x(\vec{y}).P$:

$$!x(\vec{y}).P \,|\, \overline{x}\langle\vec{v}\rangle \;\longrightarrow\; !x(\vec{y}).P \,|\, P\{\vec{v}/\vec{y}\}, \tag{2}$$

where the replicated process remains in the configuration after reduction. The π-calculus used in this abstract is the above π-calculus but without linear input prefixes: hence we only have (2) for the main communication rule.

Types for processes prescribe usage of names. To be able to do this with precision, it is important to control dynamic sharing of names. For this purpose, it is useful to restrict name passing to *bound (private) name passing*, where only bound names are passed in interaction. This allows tighter control of sharing without losing essential expressiveness, making it easier to administer name usage in more stringent ways. The resulting calculus is sometimes called the asynchronous πI-calculus in the literature [27] and has the equivalent expressive power with the version with free name passing (for the result in the typed setting, see [31]). In the present study, the restriction to bound name passing leads to a clean inverse transformation from the π-calculus into the $\lambda\mu$-calculus.

Syntactically we restrict an output to the form $(\nu\,\vec{y})(\overline{x}\langle\vec{y}\rangle|P)$ (where names in $\vec{y}$ are pairwise distinct), which we henceforth write $\overline{x}(\vec{y})P$. For dynamics, we only have the rule corresponding to (2), which now has the following form by the restriction to the bound output.

$$!x(\vec{y}).P \,|\, \overline{x}(\vec{y})Q \;\longrightarrow\; !x(\vec{y}).P \,|\, (\nu\,\vec{y})(P\,|\,Q)$$

Note "$\overline{x}(\vec{y})Q$" indicates that $\overline{x}(\vec{y})$ is an asynchronous output exporting $\vec{y}$ which are originally local to Q. After communication, $\vec{y}$ are shared between P and Q.

The formal grammar of the calculus is defined below.

$$P \;::=\; !x(\vec{y}).P \;\mid\; \overline{x}(\vec{y})P \;\mid\; P|Q \;\mid\; (\nu\,x)P \;\mid\; \mathbf{0}$$

Here $(\nu\,x)P$ is name hiding and $\mathbf{0}$ denotes nil. We use the standard structure rules, denoted by $\equiv$. We leave the full definition of the reduction and the structure rules to Appendix A.

2.2 Types and Typing for the Control

A central idea for precisely embedding functional computation in the π-calculus is to restrict process behaviour to be a *deterministic, sequential* one. To realise this idea, the following three simple conditions are ensured by types.

1. for each name, there is a unique stateless replicated input with zero or more dual outputs
2. channels has no circular dependency
3. only one single thread (output) can run on parallel

For example, by the first condition,

$$P_1 \stackrel{\text{def}}{=} !\,b.\overline{a} \mid !\,b.\overline{c} \tag{3}$$

is untypable because b is associated to two replicators, but

$$P_2 \stackrel{\text{def}}{=} !\,b.\overline{a} \mid \overline{b} \mid !\,c.\overline{b} \tag{4}$$

is typable since, while output at b appears twice, replicated input at b appears only once. Also by the second condition,

$$P_3 \stackrel{\text{def}}{=} !\,b.\overline{a} \mid !\,a.\overline{b} \tag{5}$$

is untypable: we can easily observe if we compose message $\overline{a}$ to the above process, then the computation does not terminate. Finally by the third condition, the following two processes

$$P_4 \stackrel{\text{def}}{=} \overline{a} \mid \overline{a} \quad \text{and} \quad P_5 \stackrel{\text{def}}{=} !b.(\overline{a} \mid \overline{c}) \tag{6}$$

are both untypable since two threads (outputs) are/would be running on parallel. But

$$P_6 \stackrel{\text{def}}{=} !a.\overline{b} \mid !b.\overline{c} \mid !e.\overline{c} \mid \overline{a} \tag{7}$$

is typable though c appears twice. The resulting typed processes seem very restricted but sufficient to embed the full control fully abstractly. These three conditions are guaranteed by a simple typing system which we shall introduce in the next two paragraphs.

Types. First we introduce the syntax of *channel types*. They indicate possible usage of channels.

$$\tau \ ::= \ (\vec{\tau})^p \qquad\qquad p \ ::= \ ! \mid ?$$

$\tau, \tau', ..$ (resp. $p, p', ..$) range over types (resp. modes). ! and ? are called *server* mode and *client* mode, respectively, and they are *dual* to each other. We note that this is a subset of channel types in [31] given by taking off the linear modes. As a simple example, a type

$$(\tau_1 \tau_2)^!$$

means a channel with this type should be used as a replicator which inputs two channels typed by τ_1 and τ_2, respectively. Then the *dual of* τ is defined as the result of dualising all modes in τ_i. For example, $(\overline{\tau_1}\,\overline{\tau_2})^?$ is a dual of the above type. $\mathsf{md}(\tau)$ denotes the outermost mode of τ. To guarantee the uniqueness of the server, we introduce the partial operation $\odot$ on types generated from:

$$\tau \odot \overline{\tau} = \overline{\tau} \odot \tau = \overline{\tau} \ \text{ and } \ \tau \odot \tau = \tau \ \text{ with } \ (\mathsf{md}(\tau) = ?)$$

This operation means that a server should be unique, but an arbitrary number of clients can request interactions. Note that other composition is undefined. Hence by this law, P_1 in (3) becomes untypable. Here we also assume IO-alternation on types, i.e. τ_i in $(\vec{\tau})^!$ has ?-mode and dually for $(\vec{\tau})^?$.

$$\text{(Zero)}\ \frac{-}{\vdash_{\mathtt{I}} \mathbf{0} \triangleright \emptyset} \qquad \text{(Par)}\ \frac{\vdash_{\phi_i} P_i \triangleright A_i \quad (i=1,2) \qquad A_1 \asymp A_2 \quad \phi_1 \asymp \phi_2}{\vdash_{\phi_1 \odot \phi_2} P_1 | P_2 \triangleright A_1 \odot A_2}$$

$$\text{(Res)}\ \frac{\vdash_{\phi} P \triangleright A \qquad \mathsf{md}(A(x)) = !}{\vdash_{\phi} (\nu x)P \triangleright A/x} \qquad \text{(Weak)}\ x \notin \mathsf{fn}(A)\ \frac{\vdash_{\phi} P \triangleright A \qquad \mathsf{md}(\tau) = ?}{\vdash_{\phi} P \triangleright A, x:\tau} \qquad \text{(Weak-}io\text{)}\ \frac{\vdash_{\mathtt{I}} P \triangleright A}{\vdash_{\mathtt{O}} P \triangleright A}$$

$$\text{(In}^!\text{)}\ x \notin \mathsf{fn}(A)\ \frac{\vdash_{\mathtt{O}} P \triangleright \vec{y}:\vec{\tau},\ ?A}{\vdash_{\mathtt{I}} !x(\vec{y}).P \triangleright x:(\vec{\tau})^! \to A} \qquad \text{(Out}^?\text{)}\ \frac{\vdash_{\mathtt{I}} P \triangleright A^{\vec{y}:\vec{\tau}} \asymp x:(\vec{\tau})^?}{\vdash_{\mathtt{O}} \overline{x}(\vec{y})P \triangleright A/\vec{y} \odot x:(\vec{\tau})^?}$$

Figure 1: Typing for π^{C}

To guarantee the second condition, we introduce an *action type* ranged over $A, B, C \ldots$. The syntax is given as follows:

$$A \quad ::= \quad \emptyset \mid x:\tau \mid x:(\vec{\tau_1})^! \to y:(\vec{\tau_2})^? \mid A, B$$

Edges denotes dependency between channels and are used to prevent vicious cycles between names. We compose two processes typed by A and B when $A(a) \odot B(a)$ is defined for all $a \in \mathtt{dom}(A) \cap \mathtt{dom}(B)$, and a composition creates no circularity between names. For example, a composition of $x:\tau_1 \to y:\tau_2$ and $y:\overline{\tau_2} \to x:\overline{\tau_1}$ is undefined. We write $A \asymp B$ if $A \odot B$ is defined. By this condition, P_3 (5) become untypable.

Finally the third condition is guaranteed by attaching *IO-mode*, $\phi \in \{\mathtt{I}, \mathtt{O}\}$, to the typing judgement, which has the partial algebra:

$$\mathtt{I} \odot \mathtt{I} = \mathtt{I} \text{ and } \mathtt{I} \odot \mathtt{O} = \mathtt{O} \odot \mathtt{I} = \mathtt{O}.$$

In IO-modes, $\mathtt{O}$ indicates a unique active output: thus $\mathtt{O} \odot \mathtt{O}$ is undefined, which means that we do not want more than one active thread at the same time. Hence P_4 and P_5 in (6) are untypable. We write $\phi_1 \asymp \phi_2$ if $\phi_1 \odot \phi_2$ is defined.

Typing. The judgement takes the form of

$$\vdash_{\phi} P \triangleright A$$

which is read P has type A with mode ϕ. We present the typing system in Figure 1. The rules are obtained just by restricting the typing system in [31] to the replicated fragment of the syntax we are now using. The resulting typed calculus is called π^{C}. In the following, we briefly illustrate each typing rule.

- In (Zero), we start in $\mathtt{I}$-mode with empty type since there is no active output.
- In (Par), "$\asymp$" controls composability, ensuring that at most one thread is active in a given term (by $\phi_1 \asymp \phi_2$) and uniqueness of replicated inputs and non-circularity (by $A_1 \asymp A_2$). The resulting type is given by merging two types.
- In (Res), we do not allow ? to be restricted since this action expects its dual server always exists in the environment. $A/\vec{y}$ means the result of deleting types of $\vec{y}$ from A.
- In (Weak), we can only weaken ?-moded channel since there is possibility of no output action. Similarly for (Weak-io).

- ($\mathsf{In}^!$) ensures non-circularity at x (by $x \notin \mathsf{fn}(A)$) and no free input occurrence under input (by $?A$ which means $\mathsf{md}(A) = \{?\}$). Then it records the causality from input to free outputs.
- ($\mathsf{Out}^?$) essentially the rule composes the output prefix and the body in parallel. In the condition, $A^{\vec{y}:\vec{\tau}}$ means each $y_i : \tau_i$ appears in A. It also changes the input mode from the output one to indicate the active thread. Note that this rule does not suppress the body by prefix since output is asynchronous.

The reader can check that P_2 in (4) and P_6 in (7) are typed as:

$$\vdash_{\mathrm{O}} P_2 \triangleright b:()^! \to a:()^?, c:()^! \to a:()^?$$
$$\vdash_{\mathrm{O}} P_6 \triangleright a:()^! \to c:()^?, b:()^! \to c:()^?, e:()^! \to c:()^?$$

The subject reduction of π^{C} is an immediate consequence of that in [31]. Following [31], we define an extended notion of reduction, called *the extended reduction* $\searrow$, which we shall use extensively in the present study. $\searrow$ is given as the least compatible relation over processes, taken modulo $\equiv$, which includes:

$$\begin{array}{rcl} C[\overline{x}(\vec{y})P] \,|\, !x(\vec{y}).Q & \searrow_r & C[(\nu\,\vec{y})(P|Q)] \,|\, !x(\vec{y}).Q \\ (\nu\,x)!x(\vec{y}).Q & \searrow_g & \mathbf{0} \end{array}$$

Note that $\searrow$ calculates *under* prefixes, which is unusual in process calculi. For example, we have

$$P_2 \longrightarrow !\,b.\overline{a} \,|\, \overline{a} \,|\, !\,c.\overline{b} \searrow !\,b.\overline{a} \,|\, \overline{a} \,|\, !\,c.\overline{a}$$

Since this is the image of extended reduction in [31] onto the subcalculus, we immediately know:

PROPOSITION 2.1.

1. (Subject Reduction) *If* $\vdash_{\phi} P \triangleright A$ *and* $P \searrow Q$ *then* $\vdash_{\phi} Q \triangleright A$.
2. (CR) *If P is typable and* $P \searrow Q_i$ $(i = 1,2)$ *with* $Q_1 \not\equiv Q_2$, *we have* $Q_i \searrow^+ R$ $(i = 1,2)$ *for some R.*
3. (SN) *If P is typable then P does not have infinite $\searrow$-reductions.*

We also note that $\searrow$ together with the standard congruent rules precisely generate the weak bisimilarity $\approx$, because (again) the transition relation is the faithful image of that for the pure linear π-calculus in [31].

Contextual Congruence. The above proposition suggests non-deterministic state change (which plays a basic role in e.g. bisimilarity and testing/failure equivalence) may safely be ignored in typed equality, so that a Morris-like contextual equivalence suffices as a basic equality over processes. Let us define:

$$P \Downarrow_x \text{ iff } P \longrightarrow\!\!\!\!\to \overline{x}(\vec{y})Q \text{ for some } Q$$

We can now define a typed equality. Below, a relation over typed processes is *typed* if it relates only processes with identical action type and IO-mode. A relation $\cong \supseteq \equiv$ is a *typed congruence* when it is a typed equivalence closed under typed contexts.[1]

DEFINITION 2.2. $\cong_\pi$ *is the maximum typed congruence satisfying: if* $\vdash_{\mathrm{O}} P \cong_\pi Q \triangleright x:()^?$, *then* $P \Downarrow_x$ *iff* $Q \Downarrow_x$.

By a simple operational reasoning, we can show $\cong_\pi$ is maximally consistent [9], i.e. adding any additional equation to it leads to inconsistency.

[1] ?-actions are not considered as observables in [31, 2] since, intuitively, they do not affect the environment. But in π^{C} we take ?-actions as observables since non-existence/existence of ?-actions is the only sensible way to induce non-triviality in typed equality.

3. ENCODING

In this section we present a type-preserving embedding of the call-by-value $\lambda\mu$-calculus by Ong and Stewart [23] in π^{C}. Ong and Stewart showed various control primitives of call-by-value languages (such as call-cc in ML) can be encoded in this calculus and its extension with recursion [23].

Types $(\alpha, \beta, \ldots)$ are those of simply typed λ-calculus with the atomic type $\perp$ (we can add other atomic types with appropriate inhabitants and operations on them). We use variables $(x, y, \ldots)$ as well as control variables, or names $(a, b, \ldots)$. We use the sequent of the form $\Gamma \vdash M : \alpha; \Delta$ where Γ is a finite map from variables to types and Δ is a finite map from names to non-$\perp$-types. We note the sequent in [23] has the form $\Gamma; \Delta \vdash M : \alpha$, which is natural from a logical viewpoint. We choose the present notation because it is close to its process representation, as we shall see soon. The typing rules are given in Figure 2. In the rules, we assume newly introduced names/variables in the conclusion are always fresh. The notation $\Gamma \cdot x : \tau$ indicates x is not in the domain of Γ. $M\{z/xy\}$ denotes the result of substituting z in M for both x and y, similarly for $M\{c/ab\}$. The reduction rules for the calculus (taking λ-abstraction and variables as values) is given in Appendix B. In the rules we include the η_v-reduction, unlike [23]. Their inclusion or non-inclusion does not affect the following technical development.

The encoding of types is given by two maps, $\alpha^\bullet$ and α° for $\alpha \neq \perp$, defined by the following mutual recursion. Below let $\alpha \neq \perp$.

$$\alpha^\bullet \stackrel{\text{def}}{=} (\alpha^\circ)^? \quad (\alpha \Rightarrow \beta)^\circ \stackrel{\text{def}}{=} \begin{cases} (\overline{\alpha^\circ}\beta^\bullet)^! & (\beta \neq \perp) \\ (\overline{\alpha^\circ})^! & (\beta = \perp) \end{cases}$$
$$(\perp \Rightarrow \beta)^\circ \stackrel{\text{def}}{=} \begin{cases} (\beta^\bullet)^! & (\beta \neq \perp) \\ ()^! & (\beta = \perp) \end{cases}$$

The environments for names and variables are mapped as follows, starting from $\emptyset^\bullet \stackrel{\text{def}}{=} \emptyset$ and $\emptyset^\circ \stackrel{\text{def}}{=} \emptyset$.

$$(a:\alpha \cdot \Delta)^\bullet \stackrel{\text{def}}{=} a:\alpha^\bullet \cdot \Delta^\bullet \qquad (x:\alpha \cdot \Gamma)^\circ \stackrel{\text{def}}{=} \begin{cases} x:\overline{\alpha^\circ} \cdot \Gamma^\circ & (\alpha \neq \perp) \\ \Gamma^\circ & (\alpha = \perp) \end{cases}$$

The treatment of $\perp$ reflects its special role in classical natural deduction. The encoding of terms, which closely follows that of types, is given in Figure 3. The encoding actually takes a typed term $\Gamma \vdash M : \alpha; \Delta$ as its source, though in the rules we omit environments for brevity. Here and in later encodings, we assume newly introduced names are always fresh. u in $\langle\!\langle M : \alpha \rangle\!\rangle_u$ is called its *principal port*. We also use the following expressions.

1. $\overline{x}\langle\vec{y}\rangle^{\vec{\tau}} \stackrel{\text{def}}{=} \overline{x}(\vec{z})\Pi[z_i \to y_i]^{\overline{\tau_i}}$ with each τ_i having an output mode,
2. $P\{x(\vec{y}) = R\} \stackrel{\text{def}}{=} (\nu\,x)(P \,|\, !x(\vec{y}).R)$.

Above in (1), $[x \to y]^\tau$ is a copy-cat agent (cf. [2]), defined as:

$$[x \to x']^{(\vec{\tau})^!} \stackrel{\text{def}}{=} !x(\vec{y}).\overline{x'}(\vec{y}')\Pi[y'_i \to y_i]^{\overline{\tau_i}}$$

Note that the notation in (2) already appeared in the context of CPS calculus [7, Remark 15]. The encoding is standard except for the treatment if $\perp$-types, named terms and μ-abstraction. Intuitively, $[a]M$ jumps to a instead of to its principal port, while $\mu a.M$ redirects all jumps to a to its principal port. We observe:

PROPOSITION 3.1. (type-preservation) $\Gamma \vdash M : \alpha; \Delta$ *implies* $\vdash_{\mathrm{O}} \langle\!\langle M : \alpha \rangle\!\rangle_u \triangleright (u : \alpha \cdot \Delta)^\bullet, \Gamma^\circ$.

PROOF. By rule induction on the rules in Figure 2. The two interesting cases, ($\mathsf{C\text{-}var}$) and ($\mathsf{C\text{-}name}$), are both proved by the following claim: *If* $\vdash_{\phi} P \triangleright A$ *such that* $A(x) = A(y)$ *and, moreover,*

$\mathsf{md}(A(x)) = ?$, *then* $\vdash_\phi P\{z/xy\} \triangleright A\{z/xy\}$ *for fresh* z. The claim itself is by an easy induction on the rules in Figure 1. $\square$

Note both the type of the term and names (control variables) in the $\lambda\mu$-calculus are mapped with $(\ )^\bullet$, indicating the naturalness of the shape of the sequent $\Gamma \vdash M : \alpha;\Delta$ in the present context. From a logical viewpoint, a control variable may as well be regarded as a "negative assumption" which is waiting to become the conclusion of a deduction when the absurdity is reached [24, 29].

It is instructive to see how the (call-by-value) call-cc is translated into a process.

EXAMPLE 3.2. Let $\kappa \stackrel{\text{def}}{=} \lambda y^{(\alpha\Rightarrow\beta)\Rightarrow\alpha}.\mu a^\alpha.[a](y(\lambda x^\alpha.\mu b^\beta.[a]x))$. Then we have

$$[\![\kappa]\!]_u \approx \overline{u}(c)!c(yz).\overline{y}(ec)(!e(xw).\overline{z}\langle x\rangle^{\overline{\alpha^\circ}} \mid !c(x').\overline{z}\langle x'\rangle^{\overline{\alpha^\circ}})$$

which first signals itself at u, then, when invoked with an argument y and a return point z, asks at y with an argument e and a return point c. Then whichever is invoked, it would return with the received value to the initial return point z.

Next we establish the correspondence in dynamics. Below $\rightarrow_{\lambda\mu\mathrm{v}}$ is the reduction relation on $\lambda\mu$-terms presented in [23]. $\mathtt{size}(M)$ is the size of M, which is inductively defined as:

$$\begin{array}{lcl}
\mathtt{size}(x) & = & 1\\
\mathtt{size}(\lambda x.M) & = & \mathtt{size}(M)+1\\
\mathtt{size}(MN) & = & \mathtt{size}(M)+\mathtt{size}(N)\\
\mathtt{size}([a]M) & = & 1+\mathtt{size}(N)\\
\mathtt{size}(\mu a.M) & = & 1+\mathtt{size}(M)
\end{array}$$

PROPOSITION 3.3. *If* $M \rightarrow_{\lambda\mu\mathrm{v}} M'$ *then we have either* $\langle\!\langle M:\alpha\rangle\!\rangle \equiv \langle\!\langle M':\alpha\rangle\!\rangle$ *and* $\mathtt{size}(M) \gneq \mathtt{size}(M')$ *or* $\langle\!\langle M:\alpha\rangle\!\rangle_u \searrow^+ P$ *such that* $\langle\!\langle M':\alpha\rangle\!\rangle_u \searrow^* P$.

PROOF. See Appendix C. $\square$

COROLLARY 3.4. $\rightarrow_{\lambda\mu\mathrm{v}}$ *on* $\lambda\mu$*-terms is strongly normalising.*

4. DECODING

4.1 Canonical Normal Forms

A key observation towards definability is that we can translate back processes having the translation of $\lambda\mu$-types (which we hereafter call $\lambda\mu v$*-processes*) into the original $\lambda\mu$-terms. To study the decoding, it is convenient to introduce *canonical normal forms* (CNFs) which are essentially a subset of $\lambda\mu$-terms which precisely correspond to their process representation. First, preterms for CNFs, ranged over by $\mathrm{N},\ldots$, are given by the following grammar.

$$\begin{array}{lcl}
\mathrm{N} & ::= & \mathsf{c} \mid \lambda x^\alpha.\mathrm{N} \mid \mathtt{let}\ x = y\mathrm{U}\ \mathtt{in}\ \mathrm{N} \mid \mathtt{let}\ _ = y\mathrm{U} \mid \\
 & & [a]\mathrm{U} \mid \mu a^\alpha.\mathrm{N}\\
\mathrm{U} & ::= & \mathsf{c} \mid \lambda x^\alpha.\mathrm{N} \mid \mu a^\alpha.[a]\mathrm{U},
\end{array}$$

where c is a constant with type $\perp$ and we assume:

1. In $[a]\mathrm{N}$, N does not have form $\mu b^\beta.\mathrm{N}'$.
2. In $\mu a^\alpha.\mathrm{N}$,
   (a) if $\mathrm{N} \equiv [a]\mathrm{U}$ then $a \in \mathsf{fn}(\mathrm{U})$ and
   (b) if either $\mathrm{N} \equiv \mathtt{let}\ x = y\mathrm{U}'\ \mathtt{in}\ \mathrm{N}'$ or $\mathrm{N} \equiv \mathtt{let}\ \_ = y\mathrm{U}'$ then $a \in \mathsf{fn}(\mathrm{U}')$.
3. In $\mu a^\alpha.[a]\mathrm{U}$, the same condition as 2-(a) is assumed.

The introduction of the constant, together with the conditions on name occurrences, are given so that there is a one-to-one correspondence between CNFs and $\lambda\mu v$-processes, as will be clarified in the proof of Lemma 4.3 later. Using these preterms, the set of CNFs are generated by the typing rules in Figure 2 (except the rule for application) together with the new rules in Figure 4. Note, in ($\perp$-const) in Figure 4, c, which witnesses absurdity, is introduced only when $\perp$ is assumed in the environment (logically this says that we can say an absurd thing only when the environment is absurd: the converse is not true by the way so we are not totally excused). CNFs correspond to $\lambda\mu$-terms as follows. In the first rule we assume x is chosen arbitrarily from variables assigned to $\perp$.

$$\begin{array}{rcl}
(\Gamma\cdot x:\perp \vdash \mathsf{c}:\perp;\Delta)^* & \stackrel{\text{def}}{=} & \Gamma\cdot x:\perp \vdash x:\perp;\Delta\\
(\Gamma \vdash \mathtt{let}\ x^\beta = y\mathrm{U}\ \mathtt{in}\ \mathrm{N}:\gamma;\Delta)^* & \stackrel{\text{def}}{=} & \Gamma \vdash \mathrm{N}^*\{y\mathrm{U}^*/x\}:\gamma;\Delta\\
(\Gamma \vdash \mathtt{let}\ _ = y\mathrm{U}:\perp;\Delta)^* & \stackrel{\text{def}}{=} & \Gamma \vdash y\mathrm{U}^*:\perp;\Delta
\end{array}$$

For other forms we assume the congruent mapping. Via $(\ )^*$ we can encode CNFs to processes. CNFs can also be directly encoded into processes by the rules in Figure 5 combined with those for abstraction, naming and μ-abstraction in Figure 3. By easy calculation we obtain:

PROPOSITION 4.1. *Let* $\Gamma \vdash N:\alpha;\Delta$. *Then* $\langle\!\langle N^*\rangle\!\rangle_u \searrow^* \langle N\rangle_u \not\searrow$.

4.2 Definability

The decoding is performed on $\searrow$-normal forms, using an inductive generation of the set NF_e of $\searrow$-normal forms modulo $\equiv$ by the following rules (implicitly assuming typability):

1. $\mathbf{0} \in \mathsf{NF}_e$
2. if $P,Q \in \mathsf{NF}_e$ and P and Q do not share a common free name of different polarities, then $P|Q \in \mathsf{NF}_e$
3. $P \in \mathsf{NF}_e$ then $x(\vec{y}).P \in \mathsf{NF}_e$, $!x(\vec{y}).P \in \mathsf{NF}_e$
4. $\overline{x}(\vec{y})P \in \mathsf{NF}_e$ (where $\overline{x}(\vec{y})P$ is a prime output, i.e. the initial x is the only free active occurrence in the term)

The decoding $[P]_u^{\Gamma^\circ;\Delta^\bullet}$ translates $P \in \mathsf{NF}_e$ such that $\vdash_{\Delta^\bullet} P \triangleright \Gamma^\circ$ with $u \notin \mathtt{dom}(\Gamma)$ to a $\lambda\mu$-preterm, whose rules are given in Figure 6 ($\mathtt{dom}(\Gamma)$ denotes the pre-image of Γ). In the second rule from the last, $P_{\langle a\rangle}$ indicates P is a prime output with subject a, while $P_{\langle m/a\rangle}$ is the result of replacing the (unique) active occurrence of a in $P_{\langle a\rangle}$ with m. The map is well-defined by noting each rule decreases the size of processes (which is indirect in the last two rules). We observe, writing $\mathtt{image}(\Gamma)$ for the image of Γ:

PROPOSITION 4.2. *Let* $\perp \notin \mathtt{image}(\Gamma)$, $u \notin \mathtt{dom}(\Gamma)$ *and* $P \in \mathsf{NF}_e$. *Then* $\vdash P \triangleright \Gamma^\circ \cdot \Delta^\bullet$ *implies, with x fresh:*

1. *if* $\Delta = \Delta_0 \cdot u:\alpha$ *then* $\Gamma\cdot x:\perp \vdash [P]_u^{\Gamma^\circ;\Delta^\bullet}:\alpha;\Delta_0$ *and*
2. *if* $u \notin \mathtt{dom}(\Delta)$ *then* $\Gamma\cdot x:\perp \vdash [P]_u^{\Gamma^\circ;\Delta^\bullet}:\perp;\Delta_0$.

We can now prove:

LEMMA 4.3. *If* $\vdash P \triangleright \Gamma^\circ\cdot\Delta^\bullet \in \mathsf{NF}_e$ *s.t.* $u \notin \mathtt{dom}(\Gamma)$ *then* $\langle [P]_u^{\Gamma^\circ;\Delta^\bullet}\rangle_u \equiv P$. *Conversely, if* $\Gamma \vdash N:\alpha;\Delta$ *s.t.* $u \notin \mathtt{dom}(\Gamma)$ *then* $[\langle N\rangle_u]_u^{\Gamma^\circ;\Delta^\bullet} \equiv_\alpha N$.

PROOF. Let $\vdash P \triangleright \Gamma^\circ\cdot\Delta^\bullet \in \mathsf{NF}_e$ with $u \notin \mathtt{dom}(\Gamma)$. We show $\langle [P]_u^{\Gamma^\circ;\Delta^\bullet}\rangle_u \equiv P$ by rule induction on rules in Figure 6. All rules except the last two rules are immediate. In the following, $P \stackrel{[\,]_u}{\longmapsto} Q$ means an application of $[\,]_u$ to P results to Q. Similarly for $\stackrel{\langle\,\rangle_u}{\longmapsto}$.

$$
\text{(ID)}\ \frac{-}{\Gamma\cdot x:\alpha\vdash x:\alpha;\Delta}
\qquad
\text{(C-var)}\ \frac{\Gamma\cdot x:\alpha\cdot y:\alpha\vdash M:\alpha;\Delta}{\Gamma\cdot z:\alpha\vdash M\{z/xy\}:\alpha;\Delta}
\qquad
\text{(C-name)}\ \frac{\Gamma\vdash M:\alpha;\Delta\cdot a:\alpha\cdot b:\alpha}{\Gamma\vdash M\{c/ab\}:\alpha;\Delta\cdot c:\alpha}
$$

$$
(\Rightarrow\text{-I})\ \frac{\Gamma\cdot x:\alpha\vdash M:\beta;\Delta}{\Gamma\vdash \lambda x^{\alpha}.M:\alpha\Rightarrow\beta;\Delta}
\qquad
(\Rightarrow\text{-E})\ \frac{\Gamma\vdash M:\alpha\Rightarrow\beta;\Delta \quad \Gamma\vdash N:\alpha;\Delta}{\Gamma\vdash MN:\beta;\Delta}
\qquad
(\bot\text{-I})\ \frac{\Gamma\vdash M:\alpha;\Delta \quad \alpha\neq\bot}{\Gamma\vdash [a]M:\bot;\Delta\cdot a:\alpha}
\qquad
(\bot\text{-E})\ \frac{\Gamma\vdash M:\bot;\Delta\cdot a:\alpha}{\Gamma\vdash \mu a^{\alpha}.M:\alpha;\Delta}
$$

Figure 2: Typing Rules for $\lambda\mu$-Calculus

$$
\langle\!\langle x:\alpha\rangle\!\rangle_u \stackrel{\text{def}}{=} \begin{cases} \overline{u}\langle x\rangle^{\overline{\alpha^\circ}} & (\alpha\neq\bot)\\ \mathbf{0} & (\alpha=\bot)\end{cases}
$$

$$
\langle\!\langle \lambda x^{\alpha}.M:\alpha\Rightarrow\beta\rangle\!\rangle_u \stackrel{\text{def}}{=} \begin{cases} \overline{u}(c)!c(xz).\langle\!\langle M:\beta\rangle\!\rangle_z & (\alpha\neq\bot,\beta\neq\bot)\\ \overline{u}(c)!c(z).\langle\!\langle M:\beta\rangle\!\rangle_z & (\alpha=\bot,\beta\neq\bot)\\ \overline{u}(c)!c(x).\langle\!\langle M:\bot\rangle\!\rangle_z & (\alpha\neq\bot,\beta=\bot)\\ \overline{u}(c)!c.\langle\!\langle M:\bot\rangle\!\rangle_z & (\alpha=\bot,\beta=\bot)\end{cases}
$$

$$
\langle\!\langle MN:\beta\rangle\!\rangle_u \stackrel{\text{def}}{=} \begin{cases} \langle\!\langle M:\alpha\Rightarrow\beta\rangle\!\rangle_m\{m(c)=(\langle\!\langle N:\alpha\rangle\!\rangle_n\{n(e)=\overline{c}\langle eu\rangle^{\overline{\alpha^\circ}\overline{\beta^\circ}}\})\} & (\alpha\neq\bot,\beta\neq\bot)\\ \langle\!\langle M:\alpha\Rightarrow\beta\rangle\!\rangle_m\{m(c)=\overline{c}\langle u\rangle^{\overline{\beta^\circ}}\} & (\alpha=\bot,\beta\neq\bot)\\ \langle\!\langle M:\alpha\Rightarrow\beta\rangle\!\rangle_m\{m(c)=(\langle\!\langle N:\alpha\rangle\!\rangle_n\{n(e)=\overline{c}\langle e\rangle^{\overline{\alpha^\circ}}\})\} & (\alpha\neq\bot,\beta=\bot)\\ \langle\!\langle M:\alpha\Rightarrow\beta\rangle\!\rangle_m\{m(c)=\overline{c}\} & (\alpha=\beta=\bot)\end{cases}
$$

$$
\langle\!\langle [a]M:\bot\rangle\!\rangle_u \stackrel{\text{def}}{=} \langle\!\langle M:\alpha\rangle\!\rangle_m\{a/m\}
$$

$$
\langle\!\langle \mu a^{\alpha}.M:\alpha\rangle\!\rangle_u \stackrel{\text{def}}{=} \langle\!\langle M:\bot\rangle\!\rangle_m\{u/a\}
$$

Figure 3: Encoding of $\lambda\mu$-terms

$$
(\bot\text{-const})\ \frac{-}{\Gamma\cdot x:\bot\vdash \mathtt{c}:\bot;\Delta}
\qquad
\text{(let)}\ \frac{\Gamma\cdot x:\beta\vdash \mathrm{U}:\gamma;\Delta \quad \Gamma\vdash y\mathrm{N}:\beta;\Delta \quad (\beta\neq\bot)}{\Gamma\vdash \mathtt{let}\ x^{\beta}=y\mathrm{U}\ \mathtt{in}\ \mathrm{N}:\gamma;\Delta}
\qquad
(\text{let-}\bot)\ \frac{\Gamma\vdash y:\alpha\Rightarrow\bot;\Delta \quad \Gamma\vdash \mathrm{U}:\alpha;\Delta}{\Gamma\vdash \mathtt{let}\ \_=y\mathrm{U}:\bot;\Delta}
$$

Figure 4: Typing Rules for CNFs

$$
\begin{aligned}
\langle \mathtt{c}:\bot\rangle_u &\stackrel{\text{def}}{=} \mathbf{0}\\
\langle \mathtt{let}\ x=y\mathrm{U}\ \mathtt{in}\ \mathrm{N}:\gamma\rangle_u &\stackrel{\text{def}}{=} \begin{cases} \overline{y}(wz)(P\,|\,!z(x).\langle \mathrm{N}:\gamma\rangle_u) & (\mathrm{U}\neq\mathtt{c},\ \langle\mathrm{U}\rangle_c \stackrel{\text{def}}{=} \overline{c}(w)P)\\ \overline{y}(z)!z(x).\langle \mathrm{N}:\gamma\rangle_u & (\mathrm{U}=\mathtt{c})\end{cases}\\
\langle \mathtt{let}\ \_=y\mathrm{U}:\bot\rangle_u &\stackrel{\text{def}}{=} \begin{cases} \overline{y}(w)P & (\mathrm{U}\neq\mathtt{c},\ \langle\mathrm{U}\rangle_c \stackrel{\text{def}}{=} \overline{c}(w)P)\\ \overline{y} & (\mathrm{U}=\mathtt{c})\end{cases}
\end{aligned}
$$

Figure 5: Encoding of CNFs

$$
\begin{array}{rcll}
[\mathbf{0}]_u^{\Gamma^\circ;\Delta^\bullet} & \stackrel{\text{def}}{=} & \mathtt{c}:\bot & \\
[\overline{u}(c)!c(xz).R]_u^{\Gamma^\circ;\Delta^\bullet\cdot u:(\alpha\Rightarrow\beta)^\bullet} & \stackrel{\text{def}}{=} & \lambda x^{\alpha}.[R]_z^{\Gamma^\circ\cdot x:\overline{\alpha^\circ};\Delta^\bullet\cdot z:\beta^\bullet} & \\
[\overline{u}(c)!c(z).R]_u^{\Gamma^\circ;\Delta^\bullet\cdot u:(\bot\Rightarrow\beta)^\bullet} & \stackrel{\text{def}}{=} & \lambda x^{\bot}.[R]_z^{\Gamma^\circ;\Delta^\bullet\cdot z:\beta^\bullet} & \\
[\overline{u}(c)!c(x).R]_u^{\Gamma^\circ;\Delta^\bullet\cdot u:(\alpha\Rightarrow\bot)^\bullet} & \stackrel{\text{def}}{=} & \lambda x^{\alpha}.[R]_m^{\Gamma^\circ\cdot x:\overline{\alpha^\circ};\Delta^\bullet} & \\
[\overline{u}(c)!c.R]_u^{\Gamma^\circ;\Delta^\bullet\cdot u:(\bot\Rightarrow\bot)^\bullet} & \stackrel{\text{def}}{=} & \lambda x^{\bot}.[R]_m^{\Gamma^\circ;\Delta^\bullet} & \\
[\overline{y}(wz)(R|!z(x).Q)]_u^{\Gamma^\circ;\Delta^\bullet} & \stackrel{\text{def}}{=} & \mathtt{let}\ x^{\beta}=y[\overline{c}(w)R]_c^{\Gamma^\circ;\Delta^\bullet}\ \mathtt{in}\ [Q]_u^{(\Gamma\cdot w:\beta)^\circ;\Delta^\bullet} & (\Gamma(y)=\alpha\Rightarrow\beta) \\
[\overline{y}(z)!z(x).Q]_u^{\Gamma^\circ;\Delta^\bullet} & \stackrel{\text{def}}{=} & \mathtt{let}\ x^{\beta}=y\mathtt{c}\ \mathtt{in}\ [Q]_u^{(\Gamma\cdot w:\beta)^\circ;\Delta^\bullet} & (\Gamma(y)=\bot\Rightarrow\beta) \\
[\overline{y}(w)R]_u^{\Gamma^\circ;\Delta^\bullet} & \stackrel{\text{def}}{=} & \mathtt{let}\ \_=y[\overline{c}(w)P]_c^{\Gamma^\circ;\Delta^\bullet} & (\Gamma(y)=\alpha\Rightarrow\bot) \\
[\overline{y}]_u^{\Gamma^\circ;\Delta^\bullet} & \stackrel{\text{def}}{=} & \mathtt{let}\ \_=y\mathtt{c} & (\Gamma(y)=\bot\Rightarrow\bot) \\
[P_{\langle a\rangle}]_u^{\Gamma^\circ;\Delta^\bullet\cdot a:\alpha^\bullet} & \stackrel{\text{def}}{=} & [a][P_{\langle m/a\rangle}]_m^{\Gamma^\circ;\Delta^\bullet\cdot a:\alpha^\bullet\cdot m:\alpha^\bullet} & \\
[P]_u^{\Gamma^\circ;\Delta^\bullet\cdot u:\alpha^\bullet} & \stackrel{\text{def}}{=} & \mu u^{\alpha}.[P]_m^{\Gamma^\circ;\Delta^\bullet\cdot u:\alpha^\bullet} & (\text{if no other rules apply})
\end{array}
$$

We assume $\alpha,\beta \neq \bot$, $u \notin \mathsf{fn}(R)$ and $u \notin \mathtt{dom}(\Delta)$.

Figure 6: Decoding of $\lambda\mu$-typed processes

For the first of the last two rule, assuming $\vdash P \triangleright \Gamma^\circ \cdot \Delta^\bullet \cdot a : \alpha^\bullet$:

$$P_{\langle a\rangle} \stackrel{[]_u}{\longmapsto} [a][P_{\langle m/a\rangle}]_m \stackrel{\langle\rangle_u}{\longmapsto} \langle[P_{\langle m/a\rangle}]_m\rangle_m\{a/m\} \stackrel{\text{def}}{=} (P_{\langle m/a\rangle})\{a/m\} \equiv P$$

For the second rule, we have:

$$P \stackrel{[]_u}{\longmapsto} \mu u^{\alpha}.[P]_m \stackrel{\langle\rangle_u}{\longmapsto} \langle \mu u^{\alpha}.[P]_m\rangle_u \stackrel{\text{def}}{=} \langle[P]_m\rangle_m \equiv P,$$

as required. For the other direction, assume $\Gamma \vdash M:\alpha;\Delta$ and $u \notin \mathtt{dom}(\Gamma)$. We show $[\langle \mathrm{N}\rangle_u]_u^{\Gamma^\circ;\Delta^\bullet} \equiv_\alpha \mathrm{N}$ by induction on the rules in Figures 3 and 5. The only non-trivial cases are again named terms and μ-abstraction. For the former, assuming $\Gamma \vdash [a]\mathrm{U} : \bot;\Delta$:

$$
\begin{array}{rcl}
[a]\mathrm{U}:\bot & \stackrel{\langle\rangle_u}{\longmapsto} & \langle \mathrm{U}:\alpha\rangle_m\{a/m\} \stackrel{[]_u}{\longmapsto} [a]([\langle \mathrm{U}:\alpha\rangle_m\{a/m\}])_{\langle\{m/a\}\rangle} \\
 & \equiv_\alpha & [a]\mathrm{U}:\bot
\end{array}
$$

as required.

For μ-abstraction, we first note the side condition "if no other rules apply" in the last rule of Figure 6 means P in $[P]_u^{\Gamma^\circ;\Delta^\bullet\cdot u:\alpha^\bullet}$, satisfies one of:

(i) $P \equiv \mathbf{0}$ with $u \in \mathtt{dom}(\Delta)$;

(ii) $P_{\langle u\rangle}$ with $occ(u,P) \geq 2$ or

(iii) either $P \equiv \overline{y}(wz)(R|z(x).Q)$ or $P \equiv \overline{y}(w)R$, with $occ(u,R) = \omega$.

where $occ(x,P)$ denotes the number of free occurrences of x in P, which counts free occurrences of x in the standard way except we set $occ(x,P) = \omega$ when x occurs free under replication. Let $\Gamma \vdash \mu a^{\alpha}.\mathrm{N}' : \alpha;\Delta$. The cases $\mathrm{N}' \stackrel{\text{def}}{=} \mathtt{c}$ and $\mathrm{N}' \stackrel{\text{def}}{=} \lambda x.\mathrm{N}''$, corresponding to the conditions (i) and (ii) above, are immediate. When $\mathrm{N}' \stackrel{\text{def}}{=} [a]\mathrm{U}$ with $a \in \mathsf{fn}(\mathrm{U})$, noting $a \in \mathsf{fn}(\mathrm{U})$ implies $occ(a, \langle[a]\mathrm{U}\rangle) \geq 2$,

$$
\begin{array}{rcl}
\mu a^{\alpha}.[a]\mathrm{U}:\alpha & \stackrel{\langle\rangle_u}{\longmapsto} & \langle[a]\mathrm{U}:\bot\rangle\{u/a\} \\
 & \stackrel{[]_u}{\longmapsto} & \mu u^{\alpha}.[\langle[a]\mathrm{U}:\bot\rangle\{u/a\}]_m \\
 & \equiv_\alpha & \mu a^{\alpha}.[a]\mathrm{U}:\alpha
\end{array}
$$

as required. If $\mathrm{N}' \stackrel{\text{def}}{=} \mathtt{let}\ x = y\mathrm{U}\ \mathtt{in}\ \mathrm{N}'$ and $a \in \mathsf{fn}(\mathrm{U})$, we have, noting $a \in \mathsf{fn}(\mathrm{U})$ implies $occ(a, \mathtt{let}\ x = y\mathrm{U}\ \mathtt{in}\ \mathrm{N}) = \omega$:

$(\mu a^{\alpha}.\mathtt{let}\ x = y\mathrm{U}\ \mathtt{in}\ \mathrm{N}):\alpha$

$$
\begin{array}{rl}
\stackrel{\langle\rangle_u}{\longmapsto} & \langle(\mathtt{let}\ x = y\mathrm{U}\ \mathtt{in}\ \mathrm{N}):\bot\rangle\{u/a\} \\
\stackrel{[]_u}{\longmapsto} & \mu u^{\alpha}.[\langle(\mathtt{let}\ x = y\mathrm{U}\ \mathtt{in}\ \mathrm{N}):\bot\rangle\{u/a\}]_m \\
\equiv_\alpha & (\mu a^{\alpha}.\mathtt{let}\ x = y\mathrm{U}\ \mathtt{in}\ \mathrm{N}):\alpha
\end{array}
$$

as required. The case when $\mathrm{N}' \stackrel{\text{def}}{=} \mathtt{let}\ \_ = y\mathrm{U}$ with $a \in \mathsf{fn}(\mathrm{U})$ is the same. $\square$

Let us say $\Gamma \vdash M:\alpha;\Delta$ with $u \notin \mathtt{dom}(\Gamma)$ *defines* $\vdash P \triangleright \Gamma^\circ \cdot \Delta^\bullet \in \mathsf{NF}_e$ *at* u iff $\langle\langle M:\alpha\rangle\rangle_u \searrow^* P$. A $\lambda\mu$-term is *closed* if it contains neither free names nor free variables. We can now establish the definability.

THEOREM 4.4. *(definability) Let* $\vdash P \triangleright \Gamma^\circ \cdot \Delta^\bullet \cdot u:\alpha^\bullet \in \mathsf{NF}_e$ *such that* $\bot \notin \mathtt{image}(\Gamma)$*. Then* $\Gamma \cdot x:\bot \vdash [P]_u:\alpha\ ;\Delta$ *defines P. Further if* $\Gamma = \Delta = \emptyset$ *and* $P \not\equiv \mathbf{0}$*, then there is a closed* $\lambda\mu$*-term which defines P.*

PROOF. The first half is immediate from Proposition 4.1 and Lemma 4.3. The latter half is by inspecting by induction that the given condition implies all occurrences of control constants can be replaced by (bound) variables. $\square$

4.3 Full Abstraction

To prove the full abstraction, the first task is to define a suitable observational congruence in the $\lambda\mu\nu$-calculus. There can be different notions of observational congruences for the calculus; here we choose a large, but consistent congruence, which is motivated by the equality induced by $\cong_\pi$. It is notable that the induced congruence is closely related with (and possibly coincide with some of) the notions of equality over full controls, as studied by Laird [15, 16], Selinger [28] and others.

We first define the set of observables, which are an infinite series of closed terms of the type $\bot \Rightarrow \bot \Rightarrow \bot$. To define them, we start from the following set of terms.

$$
\begin{aligned}
W_0 &\stackrel{\text{def}}{=} \lambda z^{\perp}.\mu u^{\perp\Rightarrow\perp}.z\\
W_1 &\stackrel{\text{def}}{=} \lambda z^{\perp}.\mu u^{\perp\Rightarrow\perp}.[w]\lambda z^{\perp}.\mu u^{\perp\Rightarrow\perp}.z\\
W_2 &\stackrel{\text{def}}{=} \lambda z^{\perp}.\mu u^{\perp\Rightarrow\perp}.[w]\lambda z^{\perp}.\mu u^{\perp\Rightarrow\perp}.[w]\lambda z^{\perp}.\mu u^{\perp\Rightarrow\perp}.z\\
&\ \ \vdots\\
W_{n+1} &\stackrel{\text{def}}{=} \lambda z^{\perp}.\mu u^{\perp\Rightarrow\perp}.[w]W_n\ .
\end{aligned}
$$

Let $\gamma = \perp \Rightarrow \perp \Rightarrow \perp$. We then define:

$$Obs \stackrel{\text{def}}{=} \{W_0\} \cup \{\mu w^{\gamma}.[w]W_{n+1},\ n \in \mathbb{N}\}$$

where we take terms up to the α-equality. All terms in *Obs* are closed $\longrightarrow$-normal forms of type γ (W_0 can also be written $\mu w.[w]W_0$, but is treated separately since the latter is not a normal form). To illustrate the choice of *Obs*, we show below the π-calculus representations of $W_0, \mu w.[w]W_1, \mu w.[w]W_2, \ldots$.

$$
\begin{aligned}
P_0 &\stackrel{\text{def}}{=} \overline{w}(c)!c(u).\mathbf{0}\\
P_1 &\stackrel{\text{def}}{=} \overline{w}(c)!c(u).\overline{w}(c)!c(u).\mathbf{0}\\
P_2 &\stackrel{\text{def}}{=} \overline{w}(c)!c(u).\overline{w}(c)!c(u).\overline{w}(c)!c(u).\mathbf{0}\\
&\ \ \vdots\\
P_{n+1} &\stackrel{\text{def}}{=} \overline{w}(c)!c(u).P_n.
\end{aligned}
$$

By the standard context lemma for typed processes [25, 9, 2], we can restrict the differentiating contexts for these processes to the shape of $(\nu\, w)(R \mid [\,\cdot\,])$ such that $\vdash R \triangleright w : \overline{\gamma^{\bullet}} \to a : ()^{?}$. Using them, we can easily check $P_0 \cong_\pi P_1 \cong_\pi P_2 \cong_\pi \ldots$, i.e. all these terms are contextually equal in π^{C}. Further a process in the same type but with a different $\searrow$-normal form is immediately distinct modulo $\cong_\pi$. For example, take $Q \stackrel{\text{def}}{=} \overline{w}(c)!c(u)\overline{u}(e)!e.\mathbf{0}$. Then, setting $C[\,\cdot\,]$ to be the above mentioned context with $R \stackrel{\text{def}}{=} !w(c).\overline{c}(u)!u.\overline{a}$, we can observe $C[Q]$ outputs at a, but $C[P_i]$ does not for any i. It is notable that all terms in *Obs* are equated in the call-by-value, total and extensional version of Laird's games for control [16, 10] and in the call-by-value part of Selinger's dual universe [28].

These observations motivate the following definition. Below $C[\,\cdot\,]^{\beta}_{\Gamma;\alpha;\Delta}$ is a typed context whose hole takes a term typed as $\alpha;\Delta$ under the base Γ and which returns a closed term of type β.

DEFINITION 4.5. *We write $\Gamma \vdash M \cong_{\lambda\mu} N : \alpha;\Delta$ when, for each typed context $C[\,]^{\perp\Rightarrow\perp\Rightarrow\perp}_{\Gamma;\alpha;\Delta}$, we have:*

$$C[M] \Downarrow L \in Obs \quad \textit{iff} \quad C[N] \Downarrow L' \in Obs.$$

Note we treat all values in *Obs* as an identical observable. The following result is proved by inspecting the shape of normal forms in the $\lambda\mu\nu$-terms which are typed as $\vdash M : \gamma; w : \gamma$ and which do not own a named subterm except at w (the latter point corresponds to each P_i never outputting except at w), using induction on the length of terms.

LEMMA 4.6. *Let $\vdash L : \perp \Rightarrow \perp \Rightarrow \perp$ be a normal form such that $\langle\langle L\rangle\rangle_w \cong_\pi \overline{w}(c)!c(u).\mathbf{0}$. Then $L \in Obs$.*

Further, the same routine as in [31, Section 5] gives us, via Proposition 3.3:

PROPOSITION 4.7. (computational adequacy) *Let $M : \perp \Rightarrow \perp \Rightarrow \perp$ be closed. Then $M \Downarrow L \in Obs$ iff $\langle\langle M\rangle\rangle_u \searrow^* \langle\langle L\rangle\rangle_u$ such that $L \in Obs$.*

By the standard argument using Lemma 4.6 and Proposition 4.7, we obtain:

COROLLARY 4.8. (soundness) *$\langle\langle M\rangle\rangle_u \cong_\pi \langle\langle N\rangle\rangle_u$ implies $M \cong_{\lambda\mu} N$.*

Now suppose $M_1 \cong_{\lambda\mu} M_2$, but $\langle\langle M_1\rangle\rangle_u \not\cong_\pi \langle\langle M_2\rangle\rangle_u$. Then the latter's difference is detectable by a π-calculus typed context $C[\,\cdot\,]$ of the shape similar to the one given above, with additional abstraction for variables and names (cf. Lemma 5.1 in [31]), which can send $\langle\langle M_{1,2}\rangle\rangle_u$ to any two semantically distinct points. By Theorem 4.4, we can consider this detector as an interpretations of $\lambda\mu$-terms so that we can safely set, via Lemma 4.6, $\langle\langle C[M_1]\rangle\rangle_u \Downarrow \langle\langle L\rangle\rangle_u$ and $\langle\langle C[M_2]\rangle\rangle_u \Downarrow \langle\langle L'\rangle\rangle_u$ such that $L \in Obs$ and $L' \notin Obs$. But this means, by Proposition 4.7, $C[M_1] \Downarrow L \in Obs$ and $C[M_2] \Downarrow L' \notin Obs$, that is $M_1 \not\cong_{\lambda\mu} M_2$, contradicting our assumption. We have now reached the main result of the paper.

THEOREM 4.9. (full abstraction) *$\langle\langle M\rangle\rangle_u \cong_\pi \langle\langle N\rangle\rangle_u$ if and only if $M \cong_{\lambda\mu} N$.*

5. FURTHER NOTES

This extended abstract presents the typed π-calculus for the full control, which arises in a simplest possible way as the calculus, i.e. as a subcalculus of the linear π-calculus in [31] that only uses replicated inputs. As far as we know, the connection between the control and the π-calculus is first pointed out by Thielecke in his thesis [30]. The main contribution of the present work in this context is the use of duality-based type structure in the π-calculus, by which the embedding of control constructs in processes becomes semantically exact. Hoping the present work can serve as a starting point of a fruitful dialogue between the studies on control operators and those on theories of typed processes, we list a few further topics and related works in the following.

There are a few studies (for example [28]) on conjunction and disjunction in the $\lambda\mu$-calculus. By moving to classical logics, not only negation but also these connectives (especially disjunction) bear a new significance. The encoding can be extended to these connectives keeping the syntax of π^{C} as in Section 2 (i.e. without introducing branching and selections constructs [31]). One interesting observation is that a natural encoding of the disjunction type gives rise to an encoding of terms in processes which is directly based on the standard idiom for representing the choice in unary communication. In another vein, the call-by-name $\lambda\mu$-calculus also enjoys a concise encoding into π^{C}, regarding which we are currently working on the proof of full abstraction.

The proof techniques used for definability in Section 4 are closely related to those used in game semantics. While we cannot discuss in this abstract for the space sake, the induced interaction structure (labelled transition) corresponds to those in games studied by Laird [15]. The characterisation in terms of interaction structure of typed processes becomes particularly useful when we consider the affine version of π^{C} (whose typing rules are identical except we do not record causality), where, due to nontermination, we can no longer rely on a direct syntactic argument. We would be able to use such transition-based characterisation for verifying the fully abstract embedding of such languages as PCF_μ and its call-by-value version.

The type structures for the linear/affine π-calculi are based on duality, here arising in a simplest possible way, as mutually dual input and output modes of channel types. This duality has a direct applicability for analysis of processes and programs, as can be seen in the new flow analysis we have recently developed for typed π-calculi [12]. This duality allows a clean decomposition of typed behaviours in programming languages into name passing interaction, and is in close correspondence with polarity in Polarised Linear Logic by Laurent [18, 19]. We believe the understanding of the connection between the linear/affine π-calculi and Polarised Linear Logic can be further deepened using π^{C}. In a different context, Curien and Herbelin [5] presents a calculus for control based on Gentzen's LK, in which a strong notion of duality elucidates the distinction between the call-by-name and call-by-value evaluations in the setting of full control. The operational structure of their calculus suggests an intriguing connection between their calculus and typed name passing processes, for which π^{C} would offer a useful starting point of study. Another question immediately arising from the result of this paper would be a relationship between existing CPS transformations/inversions [6, 7, 26], on the one hand, and the encoding/decoding in Sections 3 and 4 in this paper on the other.

Finally the technical development in the present paper may suggest how the typed π-calculus can be used as a tool for analysing fine-grained computational behaviours of controls via embedding. This would allow us to position various findings on syntax and semantics of diverse notions of control (cf. [17, 4, 30]) in a broad universe of typed name passing processes, which may lead to further development and applications of these findings in both theoretical and practical settings. Another interesting topic from a different viewpoint is the study of categorical structures which can cleanly articulate π^{C} (and other linear/affine calculi), starting from the past studies on semantics of controls.

Acknowledgement. The authors thank anonymous referees for their comments and suggestions, and Pierre-Louis Curien for stimulating discussions. Kohei Honda and Martin Berger are partially supported by EPSRC grant GR/S55545. Nobuko Yoshida is partially supported by EPSRC grants GR/R33465 and GR/S55538.

6. REFERENCES

[1] Abramsky, S., Jagadeesan, R. and Malacaria, P., Full Abstraction for PCF, 1994. *Info. & Comp.* 163 (2000), 409-470.

[2] Berger, M., Honda, K. and Yoshida, N., Sequentiality and the π-Calculus, *TLCA01*, LNCS 2044, pp.29-45, Springer, 2001.

[3] Berger, M., Honda, K. and Yoshida, N., Genericity and the π-Calculus, *FoSSaCs'03*, LNCS 2620, 103–119, Springer, 2003.

[4] Berdine, J., O'Hearn, P., Reddy, U. and Thielecke, H., Linear Cotinuation Passing, *Higher-Order Symbolic Computation*, 15(2/3):181–208, Sep. 2002.

[5] Curien, P-L., Danvy, Herbelin, H., The Duality of Computation. ICFP'00, ACM, 2000.

[6] Danvy, O. and Fillinski, A., Representing control: A study of the CPS transformation, MSCS, 2(4):361–391, 1992.

[7] Führmann, C. and Thielecke, H., On the call-by-value CSP transform and its semantics, To appear in *Journal of Information and Computation*.

[8] Honda, K. and Tokoro, M. An object calculus for asynchronous communication. ECOOP'91, LNCS 512, 133–147, 1991.

[9] Honda, K. and Yoshida, N., On Reduction-Based Process Semantics. *TCS*, pp.437–486, No.151, North-Holland, 1995.

[10] Honda, K. and Yoshida, N., Game-Theoretic Analysis of Call-by-Value Computation, *TCS*, Vol. 221 (1999), North-Holland, 1999.

[11] Honda, K. and Yoshida, N., A Uniform Type Structure for Secure Information Flow, *POPL'02*, 81–92, ACM Press, 2002. The full version: a DOC technical report, Imperial College London, revised August 2003, available at: `www.doc.ic.ac.uk/~yoshdia`.

[12] Honda, K. and Yoshida, N., Noninterference through Flow Analysis, a DOC technical report, Imperial College London, revised September 2003, available at: `www.doc.ic.ac.uk/~yoshdia`.

[13] Honda, K. and Yoshida, N. Game-theoretic analysis of call-by-value computation. *TCS*, 221 (1999), 393–456.

[14] Hyland, M. and Ong, L., Full Abstraction for PCF: I, II and III. *Info. & Comp.* 163 (2000), 285-408.

[15] Laird, J., Full abstraction for functional languages with controls, LICS'97, IEEE, 1997.

[16] Laird, J. *A semantic analysis of control.* Phd thesis, University of Edinburgh, 1998.

[17] Laird, J., A game semantics for linearly used continuations, *FoSSaCs'03*, LNCS 2620, 313–327, Springer, 2003.

[18] Laurent, O., Polarized proof-nets and $\lambda\mu$-calculus, *TCS*, 290(1):161–188, Dec, 2002.

[19] Laurent, O., Polarized games, *LICS'02*, 265-274, IEEE, 2002.

[20] Milner, R., Functions as Processes, *MSCS.* 2(2):119–141, 1992,

[21] Milner, R., Parrow, J. and Walker, D., A Calculus of Mobile Processes, *Info. & Comp.* 100(1):1–77, 1992.

[22] Myers, M., JFlow: Practical mostly-static information flow control. POPL'99, 228–241, 1999.

[23] Ong, L. and Stewart, C., A Curry-Howard foundation for functional computation with control. *POPL'97*, ACM, 1997.

[24] Parigot, M., $\lambda\mu$-calculus: an algorithmic interpretation of classical natural deduction. Proc. *Logic Programming and Automated Reasoning*, LNCS 624, 190–201, Springer, 1992.

[25] Pierce, B.C. and Sangiorgi. D, Typing and subtyping for mobile processes. *LICS'93*, pp.187–215, IEEE, 1993.

[26] Sabry, A and Felleisen, M., Reasoning about Programs in Continuation-Passing Style, Lisp and Symbolic Computation 6(3-4):289–360 (1993).

[27] Sangiorgi, D. π-calculus, internal mobility, and agent-passing calculi. *TCS*, 167(2):235–271, 1996.

[28] Selinger, P., Control Category and Duality: on the Categorical Semantics of the Lambda-Mu Calculus, MSCS (2001), Vol. 11, 207–260, 2001.

[29] Stewart, C., On the formulae-as-types correspondence on classical logic, PhD Thesis, Oxford, 1999.

[30] Thielecke, H., Categorical Structure of Continuation Passing Style, PhD thesis, University of Edinburgh, 1997.

[31] Yoshida, N., Berger, M. and Honda, K., Strong normalisation in the π-Calculus, *LICS'01*, IEEE, 2001. The full version is available at www.doc.ic.ac.uk/~yoshida. To appear in *Journal of Information and Computation*.

[32] Yoshida, N., Honda, K. and Berger, M. Linearity and Bisimulation, FoSSaCs 2002, LNCS 2303, pp.417–433, Springer, 2002. A full version as a MCS technical report, 2001-48, University of Leicester, 2001, available at www.mcs.le.ac.uk/~yoshida.

APPENDIX

A. REDUCTION AND STRUCTURE RULES

The reduction and structure rules for π^{C} is defined in Figure 7.

(Structural Rules)

$$
\begin{array}{llll}
\text{(S0)}\ P \equiv Q \quad \text{if } P \equiv_\alpha Q & \text{(S1)}\ P|\mathbf{0} \equiv P & \text{(S2)}\ P|Q \equiv Q|P & \\
\text{(S3)}\ P|(Q|R) \equiv (P|Q)|R & \text{(S4)}\ (\nu x)\mathbf{0} \equiv \mathbf{0} & \text{(S5)}\ (\nu x)(\nu y)P \equiv (\nu y)(\nu x)P & \\
\text{(S6)}\ (\nu x)(P|Q) \equiv ((\nu x)P)|Q \ \ (x \notin \mathsf{fn}(Q)) & \multicolumn{3}{l}{\text{(S7)}\ \overline{x}(\vec{y})\overline{z}(\vec{w})P \equiv \overline{z}(\vec{w})\overline{x}(\vec{y})P \ \ (x,z \notin \{\vec{w}\vec{y}\})} \\
\text{(S8)}\ (\nu z)\overline{x}(\vec{y})P \equiv \overline{x}(\vec{y})(\nu z)P \ \ (z \notin \{x\vec{y}\}) & \multicolumn{3}{l}{\text{(S9)}\ \overline{x}(\vec{y})(P|Q) \equiv (\overline{x}(\vec{y})P)|Q \ \ (\{\vec{y}\} \cap \mathsf{fn}(Q) = \emptyset)}
\end{array}
$$

(Reduction)

$$
\begin{array}{llll}
(\mathsf{Com}_!) & !x(\vec{y}).P \,|\, \overline{x}(\vec{y})Q \longrightarrow !x(\vec{y}).P|(\nu \vec{y})(P|Q) & (\mathsf{Res}) & P \longrightarrow Q \implies (\nu x)P \longrightarrow (\nu x)Q \\
(\mathsf{Par}) & P \longrightarrow P' \implies P|Q \longrightarrow P'|Q & (\mathsf{Out}) & P \longrightarrow Q \implies \overline{x}(\vec{y})P \longrightarrow \overline{x}(\vec{y})Q \\
 & & (\mathsf{Cong}) & P \equiv P' \longrightarrow Q' \equiv Q \implies P \longrightarrow Q
\end{array}
$$

Figure 7: Reduction and Structural Rules

In the last four rules, we let $M \stackrel{\text{def}}{=} C[[a]L_i]_{i \in I}$ where I enumerates all a-named subterms (with possible nesting) and $\beta \neq \bot$.

$$
\begin{array}{llll}
(\beta_v) & (\lambda x.M)V \longrightarrow M\{V/x\} & (\eta_v) & \lambda x.(Vx) \longrightarrow V \\
(\mu\text{-}\beta) & \mu a.[b]M \longrightarrow M\{b/a\} & (\mu\text{-}\eta) & \mu a.[a]M \longrightarrow M \quad (\text{if } a \notin \mathsf{fn}(M)) \\
(\zeta_{\mathrm{fun}}) & (\mu a^{\alpha \Rightarrow \beta}.M)N \longrightarrow \mu b.C[[b](L_iN)]_i & (\zeta_{\mathrm{fun},\bot}) & (\mu a^{\alpha \Rightarrow \bot}.M)N \longrightarrow C[(L_iN)]_i \\
(\zeta_{\mathrm{arg}}) & V^{\alpha \Rightarrow \beta}(\mu a^{\alpha}.M) \longrightarrow \mu b.C[[b](VL_i)]_i & (\zeta_{\mathrm{arg},\bot}) & V^{\alpha \Rightarrow \bot}(\mu a^{\alpha}.M) \longrightarrow C[(VL_i)]_i
\end{array}
$$

Figure 8: Reduction Rules for the $\lambda\mu v$

B. REDUCTION FOR THE $\lambda\mu$V-CALCULUS

The reduction rules for the $\lambda\mu v$-calculus is given in Figure 8.

C. PROOF FOR PROPOSITION 3

In the following proof we often omit principal ports of $\bot$-typed terms, observing $u \notin \langle\!\langle M:\bot \rangle\!\rangle_u$ for each $\Gamma \vdash M:\bot;\Delta$. Throughout we assume newly introduced names are fresh.

(β_v-reduction), (η_v-reduction) Omitted.

(μ-reductions) Note μ-reductions strictly decrease a term size.

$$
\begin{array}{llll}
(\mu\text{-}\beta) & \langle\!\langle [b]\mu a.M:\bot \rangle\!\rangle_u & \equiv & (\langle\!\langle M:\bot \rangle\!\rangle\{u/a\})\{b/u\} \\
 & & \equiv & \langle\!\langle M:\bot \rangle\!\rangle\{b/a\} \stackrel{\text{def}}{=} \langle\!\langle M\{b/a\}:\bot \rangle\!\rangle. \\
(\mu\text{-}\eta) & \langle\!\langle \mu a.[a]M:\alpha \rangle\!\rangle_u & \equiv & (\langle\!\langle M:\alpha \rangle\!\rangle_u\{u/a\})\{a/u\} \\
 & & \stackrel{\text{def}}{=} & \langle\!\langle M:\alpha \rangle\!\rangle_u.
\end{array}
$$

(ζ-reduction) Let $M \stackrel{\text{def}}{=} C[[a]L_i]_{i \in I}$ where I enumerates all a-named subterms (with possible nesting) and let $\langle\!\langle M:\bot \rangle\!\rangle \stackrel{\text{def}}{=} C'[\langle\!\langle L_i \rangle\!\rangle_a]_i$ accordingly. Note all free occurrences of a in $\langle\!\langle M:\bot \rangle\!\rangle$ are exhaustively mentioned in this way. Given $C[[a]L_i]_i$, we write $C[[b]L_iN]_i$ (say) to indicate the result of filling each hole with a new subterm (because of possible nesting of holes, if $[..]_j$ occurs in L_i then the corresponding subterm should also be replaced), similarly we write $C'[\langle\!\langle L_i \rangle\!\rangle_a\{a(x)=R'\}]$. We show one case of (ζ_{fun}) [23]. The remaining cases are similar. Below we assume $\beta \neq \bot$ and we let either $R \stackrel{\text{def}}{=} \langle\!\langle N \rangle\!\rangle_n\{n(e)=\overline{c}\langle eu \rangle\}$ (if $\alpha \neq \bot$) or $R \stackrel{\text{def}}{=} \overline{c}\langle u \rangle$ (if $\alpha = \bot$).

Further we let $P_1 \searrow^{+}\!\swarrow P_2$ denote $P_i \searrow^{+} P'$ $(i = 1,2)$ for some P'.

$$
\begin{array}{llll}
(\boldsymbol{\zeta_{\mathrm{fun}}}) & \langle\!\langle (\mu a^{\alpha \Rightarrow \beta}.M)N \rangle\!\rangle_u & \stackrel{\text{def}}{=} & \langle\!\langle M:\bot \rangle\!\rangle\{m/a\}\{m(c)=R\} \\
 & & \equiv & \langle\!\langle M:\bot \rangle\!\rangle\{a(c)=R\} \\
 & & \equiv & C'[\langle\!\langle L_i \rangle\!\rangle_a]_i\{a(c)=R\} \\
 & & \searrow^{+}\!\swarrow & C'[\langle\!\langle L_i \rangle\!\rangle_{l_i}\{l_i(c)=R\}]_i\{u/b\} \\
 & & \stackrel{\text{def}}{=} & \langle\!\langle \mu b.C[[b](L_iN)] \rangle\!\rangle_u.
\end{array}
$$

Let $R \stackrel{\text{def}}{=} \langle\!\langle N \rangle\!\rangle_n\{n(e)=\overline{c}\langle e \rangle\}$ if $\beta \neq \bot$, or $R \stackrel{\text{def}}{=} \overline{c}$ if $\beta = \bot$.

$$
\begin{array}{llll}
(\boldsymbol{\zeta_{\mathrm{fun},\bot}}) & \langle\!\langle (\mu a^{\alpha \Rightarrow \bot}.M)N \rangle\!\rangle_u & \stackrel{\text{def}}{=} & \langle\!\langle M:\bot \rangle\!\rangle\{m/a\}\{m(c)=R\} \\
 & & \equiv & C'[\langle\!\langle L_i \rangle\!\rangle_a]_i\{a(c)=R\} \\
 & & \searrow^{+}\!\swarrow & C'[\langle\!\langle L_i \rangle\!\rangle_{l_i}\{l_i(c)=R\}]_i \\
 & & \stackrel{\text{def}}{=} & \langle\!\langle C[(L_iN)] \rangle\!\rangle_u.
\end{array}
$$

Next we consider (ζ_{arg}) rules in [23]. Let $\alpha \neq \bot$. First let $\beta \neq \bot$ and $\langle\!\langle V:\alpha \Rightarrow \beta \rangle\!\rangle_m \stackrel{\text{def}}{=} \overline{m}(c)!c(e'u').R$ and $R' \stackrel{\text{def}}{=} R\{eu/e'u'\}$. Noting $m \notin \mathsf{fn}(R)$ by V being a value, we have, with ${}^{+}\!\swarrow$ denoting the inverse of $\searrow^{+}$:

$$
\begin{array}{llll}
(\boldsymbol{\zeta_{\mathrm{arg}}}) & \langle\!\langle V(\mu a^{\alpha}.M) \rangle\!\rangle_u & \searrow^{+} & \langle\!\langle M:\bot \rangle\!\rangle\{a(e)=R'\} \\
 & & \searrow^{+}\!\swarrow & C'[\langle\!\langle L_i \rangle\!\rangle_a\{a(e)=R'\})]_i \\
 & & {}^{+}\!\swarrow & \langle\!\langle C[[b](VL_i)]_i:\bot \rangle\!\rangle\{u/b\} \\
 & & \stackrel{\text{def}}{=} & \langle\!\langle \mu b.C[[b](VL_i)] \rangle\!\rangle_u.
\end{array}
$$

Finally with $\langle\!\langle V:\alpha \Rightarrow \bot \rangle\!\rangle_m \stackrel{\text{def}}{=} \overline{m}(c)!c(e').R$ and $R' \stackrel{\text{def}}{=} R\{e/e'\}$,

$$
\begin{array}{llll}
(\boldsymbol{\zeta_{\mathrm{arg},\bot}}) & \langle\!\langle V(\mu a^{\alpha}.M) \rangle\!\rangle & \searrow^{+}\!\swarrow & C'[\langle\!\langle L_i \rangle\!\rangle_a\{a(e)=R'\})]_i \\
 & & {}^{+}\!\swarrow & \langle\!\langle C[(VL_i)]_i:\bot \rangle\!\rangle,
\end{array}
$$

as required.