2. *the converse, i.e.* $\mathcal{F}[\![P]\!] \xrightarrow{\mu'} P''$ *implies there exists* P', μ *s.t.*

   $P \xrightarrow{\mu} P'$ *and* $P'' = \mathcal{F}[\![P']\!]$, $\mu' = \mathcal{F}[\![\mu]\!]$.

Proof: Another easy induction on the structure of P, both for (1) and for (2). Again, we consider the most significant cases.

- $P = a(X).Q$.

  Then $\mathcal{F}[\![P]\!] = a(x).\mathcal{F}[\![Q]\!]$. It holds that $P \xrightarrow{a\langle Tr_m \rangle} Q\{Tr_m/X\}$ and $\mathcal{F}[\![P]\!] \xrightarrow{a\langle m \rangle} \mathcal{F}[\![Q]\!]\{m/x\}$; we have $a\langle m \rangle = \mathcal{F}[\![a\langle Tr_m \rangle]\!]$ and by Lemma 5.1.4, $\mathcal{F}[\![Q\{Tr_m/X\}]\!] = \mathcal{F}[\![Q]\!]\{m/x\}$.

- $P = P_1 \mid P_2$.

  Then $\mathcal{F}[\![P]\!] = \mathcal{F}[\![P_1]\!] \mid \mathcal{F}[\![P_2]\!]$. We consider only (1) in the case that the rule used to infer the action is com. We have

$$\frac{P_1 \xrightarrow{(\boldsymbol{\nu}\, m)\overline{a}\langle Tr_m \rangle} P_1' \quad P_2 \xrightarrow{a\langle Tr_m \rangle} P_2'}{P_1 \mid P_2 \xrightarrow{\tau} \boldsymbol{\nu}\, m\,(P_1' \mid P_2')}$$

  Using induction twice,

$$\frac{\mathcal{F}[\![P_1]\!] \xrightarrow{(\boldsymbol{\nu}\, m)\overline{a}\langle m \rangle} \mathcal{F}[\![P_1']\!] \quad \mathcal{F}[\![P_2]\!] \xrightarrow{a\langle m \rangle} \mathcal{F}[\![P_2']\!]}{\mathcal{F}[\![P_1]\!] \mid \mathcal{F}[\![P_2]\!] \xrightarrow{\tau} \boldsymbol{\nu}\, m\,(\mathcal{F}[\![P_1']\!] \mid \mathcal{F}[\![P_2']\!]) = \mathcal{F}[\![\boldsymbol{\nu}\, m\,(P_1' \mid P_2')]\!]} \qquad \square$$

The generalisation of Lemma 5.1.5 to the weak case is straightforward.

Corollary 5.1.6 (operational correspondence for $\mathcal{F}$ on weak transitions) *Let* $P \in THO\pi$ *and* μ *a basic action. Then*

1. $P \stackrel{\mu}{\Longrightarrow} P'$ *implies* $\mathcal{F}[\![P]\!] \stackrel{\mathcal{F}[\![\mu]\!]}{\Longrightarrow} \mathcal{F}[\![P']\!]$.

2. *the converse, i.e.* $\mathcal{F}[\![P]\!] \stackrel{\mu'}{\Longrightarrow} P''$ *implies there exists* P', μ *s.t.*

   $P \stackrel{\mu}{\Longrightarrow} P'$ *and* $P'' = \mathcal{F}[\![P']\!]$, $\mu' = \mathcal{F}[\![\mu]\!]$. $\square$

Correctness w.r.t. weak barbed equivalence and congruence

In the proof of the results below we exploit the characterisations of barbed equivalence in terms of weak context bisimulation ($\approxeq_{\mathrm{Ct}}$) and weak triggered bisimulation ($\approxeq_{\mathrm{Tr}}$) in HOπ (Corollary 4.7.8), and of weak early bisimulation ($\approx_{\mathrm{e}}$) in π-calculus (Theorem 3.3.3); moreover we only check the actions occurring at names whose object sort has been modified by $\mathcal{F}$. The proof for the remaining actions is a straightforward application of Lemma 5.1.5 and Corollary 5.1.6.

Lemma 5.1.7 *Suppose P and Q are $THO\pi$ processes. Then $P \approx Q$ implies $\mathcal{F}[\![P]\!] \approx \mathcal{F}[\![Q]\!]$.*

PROOF: We prove that

$$\mathcal{R} = \{(\mathcal{F}[\![P]\!], \mathcal{F}[\![Q]\!]) \,:\, P, Q \in THO\pi \text{ and } P \approxeq_{\mathrm{Tr}} Q\}$$

is a $\approx_{\mathrm{e}}$-bisimulation. Suppose $\mathcal{F}[\![P]\!] \xrightarrow{a\langle m\rangle} P''$. By Lemma 5.1.5,

$$P \xrightarrow{a\langle Tr_m\rangle} P' \quad \text{with} \quad P'' = \mathcal{F}[\![P']\!].$$

The definition of $\approxeq_{\mathrm{Tr}}$ guarantees that there exists Q' s.t. $Q \stackrel{a\langle Tr_m\rangle}{\Longrightarrow} Q' \approxeq_{\mathrm{Tr}} P'$ only for $m \notin fn(P,Q)$. Since here in general m may occur free in P or Q, we have to exploit the equality between $\approxeq_{\mathrm{Ct}}$ and $\approxeq_{\mathrm{Tr}}$ on triggered agents. From $P \approxeq_{\mathrm{Ct}} Q$ we get

$$Q \stackrel{a\langle Tr_m\rangle}{\Longrightarrow} Q' \approxeq_{\mathrm{Ct}} P'$$

Since $Q' \in THO\pi$ by Lemma 4.6.1, also $Q' \approxeq_{\mathrm{Tr}} P'$; moreover by Corollary 5.1.6,

$$\mathcal{F}[\![Q]\!] \stackrel{a\langle m\rangle}{\Longrightarrow} \mathcal{F}[\![Q']\!]$$

yielding $P'' \,\mathcal{R}\, \mathcal{F}[\![Q']\!]$.

Let us consider now outputs. By Lemma 5.1.5 every output performed by $\mathcal{F}[\![P]\!]$ is the image, under $\mathcal{F}$, of an action performed by P. By Lemma 4.6.1, every higher-order output performed by P is of the form $(\boldsymbol{\nu}\, m)\overline{a}\langle Tr_m\rangle$. Hence the corresponding output of $\mathcal{F}[\![P]\!]$ are of the form $\mathcal{F}[\![P]\!] \xrightarrow{(\boldsymbol{\nu}\, m)\overline{a}\langle m\rangle} P''$. Now, proceeding as in the input case above — but this time we do not need to employ $\approxeq_{\mathrm{Ct}}$ — we find P' with $P'' = \mathcal{F}[\![P']\!]$, and $Q' \approxeq_{\mathrm{Tr}} P'$ s.t. $\mathcal{F}[\![Q]\!] \xrightarrow{(\boldsymbol{\nu}\, m)\overline{a}\langle m\rangle} \mathcal{F}[\![Q']\!]$. $\square$

Lemma 5.1.8 *For processes $P, Q \in THO\pi$, $\mathcal{F}[\![P]\!] \approx \mathcal{F}[\![Q]\!]$ implies $P \approx Q$.*

Proof: We prove that

$$\mathcal{R} = \{(P, Q) \; : \; \mathcal{F}[\![P]\!] \approx_{\mathrm{e}} \mathcal{F}[\![Q]\!]\}$$

is a $\approxeq_{\mathrm{Tr}}$-bisimulation. Suppose $P \overset{a\langle Tr_m\rangle}{\longrightarrow} P'$, and $m \notin fn(P, Q)$. Then, by Lemma 5.1.5, $\mathcal{F}[\![P]\!] \overset{a\langle m\rangle}{\longrightarrow} \mathcal{F}[\![P']\!]$. From $\mathcal{F}[\![P]\!] \approx_{\mathrm{e}} \mathcal{F}[\![Q]\!]$, we get $\mathcal{F}[\![Q]\!] \overset{a\langle m\rangle}{\Longrightarrow} Q''$ and by Corollary 5.1.6, $Q \overset{a\langle Tr_m\rangle}{\Longrightarrow} Q'$ with $\mathcal{F}[\![Q']\!] = Q''$. Therefore, $Q' \mathcal{R} P'$.

Now the case of an output action. Using Lemma 4.6.1 the transition for P is of the form $P \overset{(\boldsymbol{\nu}\, m)\overline{a}\langle Tr_m\rangle}{\longrightarrow} P'$ and everything follows similarly to the input case. $\square$

We now show that the faithfulness of $\mathcal{F}$ extends to abstractions too. Then we shall be able to express the final results on *open* agents.

Lemma 5.1.9 *For abstractions $F, E \in THO\pi$, we have*

$$F \approx E \;\; \textit{iff} \;\; \mathcal{F}[\![F]\!] \approx \mathcal{F}[\![E]\!].$$

Proof: This assertion is easy to prove for first-order abstractions using the two previous lemmas, so we only consider higher-order abstractions. Suppose $F = (X)P$ and $E = (X)Q$ with $F \approx E$. Using the characterisation of $\approx$ in terms of $\approxeq_{\mathrm{Ct}}$, we infer that for every m, $P\{Tr_m/X\} \approx Q\{Tr_m/X\}$. Hence by Lemma 5.1.7, $\mathcal{F}[\![P\{Tr_m/X\}]\!] \approx \mathcal{F}[\![Q\{Tr_m/X\}]\!]$, from which, using Lemma 5.1.4, also $\mathcal{F}[\![P]\!]\{m/x\} \approx \mathcal{F}[\![Q]\!]\{m/x\}$. But this means $(x)\mathcal{F}[\![P]\!] \approx (x)\mathcal{F}[\![Q]\!]$ which, by definition of the encoding, is the same as $\mathcal{F}[\![(X)P]\!] \approx \mathcal{F}[\![(X)Q]\!]$.

The implication in the opposite direction can be worked out similarly, but this time using the characterisation of $\approx$ in terms of $\approxeq_{\mathrm{Tr}}$ instead of $\approxeq_{\mathrm{Ct}}$. $\square$

Theorem 5.1.10 ($\mathcal{F}$ respects $\approx$) *For agents $A_1, A_2 \in THO\pi^{\circ}$, we have*

$$A_1 \approx A_2 \; \textit{iff} \; \mathcal{F}[\![A_1]\!] \approx \mathcal{F}[\![A_2]\!]$$

Proof: Lemmas 5.1.7 and 5.1.8 prove the result for closed processes, and Lemma 5.1.9 for closed abstractions. Therefore, it remains to consider the case in which A_1 or A_2 contain free variables. The proof for this case is similar to the proof of Lemma 5.1.9 for higher-order abstractions. (Actually, it is simpler: In the implication from left to right we do not need to employ $\approxeq_{\mathrm{Ct}}$, since $\approxeq_{\mathrm{Tr}}$ is enough. The reason is that a variable which is free in A_1, A_2 becomes a free name in $\mathcal{F}[\![A_1]\!]$ and $\mathcal{F}[\![A_2]\!]$ and to check $\mathcal{F}[\![A_1]\!] \approx_{\mathrm{e}} \mathcal{F}[\![A_2]\!]$ we do not have to consider all possible instantiations of that name). $\square$

Now we can extend the faithfulness of $\mathcal{F}$ to the full congruence $\approx^{\mathrm{c}}$.

Theorem 5.1.11 ($\mathcal{F}$ respects $\approx^{\mathrm{c}}$) *For $A_1, A_2 \in THO\pi^{\mathrm{o}}$, we have*

$$A_1 \approx^{\mathrm{c}} A_2 \textit{ iff } \mathcal{F}[\![A_1]\!] \approx^{\mathrm{c}} \mathcal{F}[\![A_2]\!]$$

Proof: In view of the characterisation of $\approx^{\mathrm{c}}$ in terms of $\approxeq^{\mathrm{c}}_{\mathrm{Tr}}$ in HOπ and of $\approx^{\mathrm{c}}_{\mathrm{e}}$ in π-calculus, two HOπ or π-calculus agents are in the relation $\approx^{\mathrm{c}}$ if they are in the relation $\approx$ for all name substitutions. Therefore, for closed agents the result of this theorem follows from Theorem 5.1.10 since, for any A, the agents A and $\mathcal{F}[\![A]\!]$ have the same sets of free names. This does not hold if A is open, since the free variables of A become free names in $\mathcal{F}[\![A]\!]$. However, the result for open agents can be proved exploiting the result for closed agents. We omit the details, which are simple. $\square$

Remark 5.1.12 The reason for proving the correctness of $\mathcal{F}$ w.r.t. weak equivalences is that later we want to combine $\mathcal{F}$ with mapping $\mathcal{T}$ defined in Section 4.5.2, and $\mathcal{T}$ does not respect the strong equivalences. We conjecture however that the faithfulness of $\mathcal{F}$ holds in the strong case too. $\square$

5.2 The compilation of HOπ into π-calculus

By putting together mappings $\mathcal{T}$ and $\mathcal{F}$, we obtain a compilation $\mathcal{C}$ from higher-order agents and sortings to first-order agents and sortings. $\mathcal{C}$ is defined on sorting as $\mathcal{F}$, since $\mathcal{T}$ does not modify a sorting. The direct definition of $\mathcal{C}$ on agents is reported in Table 5–2. In the table, it is supposed that m and x are fresh names, that is, they do not occur in the source agent. We recall that a higher-order abstraction is an abstraction which takes an agent as argument, i.e. its

$$
\begin{array}{rcl}
\mathcal{C}[\![X]\!] & \stackrel{def}{=} & \begin{cases} \mathcal{C}[\![(Y)X\langle Y\rangle]\!] & \text{if } X \text{ is a higher-order abstraction} \\ \mathcal{C}[\![(a)X\langle a\rangle]\!] & \text{otherwise} \end{cases} \\
\mathcal{C}[\![\alpha.P]\!] & \stackrel{def}{=} & \begin{cases} (\overline{a}\langle m\rangle.\mathcal{C}[\![P]\!])\,\{m := \mathcal{C}[\![F]\!]\} & \text{if } \alpha = \overline{a}\langle F\rangle \\ a(x).\mathcal{C}[\![P]\!] & \text{if } \alpha = a(X) \\ \alpha.\mathcal{C}[\![P]\!] & \text{otherwise} \end{cases} \\
\mathcal{C}[\![X\langle F\rangle]\!] & \stackrel{def}{=} & (\overline{x}\langle m\rangle.\mathbf{0})\,\{m := \mathcal{C}[\![F]\!]\}
\end{array}
$$

$$
\begin{array}{rclrcl}
\mathcal{C}[\![X\langle b\rangle]\!] & \stackrel{def}{=} & \overline{x}\langle b\rangle.\mathbf{0} & \mathcal{C}[\![\,!\,P]\!] & \stackrel{def}{=} & !\,\mathcal{C}[\![P]\!] \\
\mathcal{C}[\![P \mid Q]\!] & \stackrel{def}{=} & \mathcal{C}[\![P]\!] \mid \mathcal{C}[\![Q]\!] & \mathcal{C}[\![\sum_{i\in I} P_i]\!] & \stackrel{def}{=} & \sum_{i\in I} \mathcal{C}[\![P_i]\!] \\
\mathcal{C}[\![\boldsymbol{\nu}\, a\, P]\!] & \stackrel{def}{=} & \boldsymbol{\nu}\, a\, \mathcal{C}[\![P]\!] & \mathcal{C}[\![[a=b]P]\!] & \stackrel{def}{=} & [a=b]\mathcal{C}[\![P]\!] \\
\mathcal{C}[\![(X)P]\!] & \stackrel{def}{=} & (x)\mathcal{C}[\![P]\!] & \mathcal{C}[\![(a)P]\!] & \stackrel{def}{=} & (a)\mathcal{C}[\![P]\!]
\end{array}
$$

Table 5–2: The transformation $\mathcal{C}$

sort is (S), for some S. We also recall that, from the constructions of $\mathcal{T}$ and $\mathcal{F}$, the definition of $\mathcal{C}$ depends upon the selection of the subsorting SOb of Ob, from which the name-triggers are drawn. From the results proved for $\mathcal{T}$ and $\mathcal{F}$ (Theorems 4.5.8, 5.1.10 5.1.11), we get:

Theorem 5.2.1 (correctness of $\mathcal{C}$) *For every $A_1, A_2 \in HO\pi^{\circ}$ and sorting Ob, with $A_1, A_2 : Ob$, it holds that*

1. *$\mathcal{C}[\![A_1]\!]$ is a π-calculus process, and $\mathcal{C}[\![A_1]\!] :: \mathcal{C}[\![Ob]\!]$,*
2. *$\mathcal{C}[\![A_1]\!] \approx \mathcal{C}[\![A_2]\!]$ iff $A_1 \approx A_2$,*
3. *$\mathcal{C}[\![A_1]\!] \approx^{c} \mathcal{C}[\![A_2]\!]$ iff $A_1 \approx^{c} A_2$.* □

We refer to Examples 5.1.1 and 5.1.2 to see how $\mathcal{C}$ behaves. More interesting examples will be provided in Chapter 6, when considering the encoding of λ-calculus. Note that the compilation $\mathcal{C}$ is obtained combining $\mathcal{T}$ and $\mathcal{F}$ in one *single* step, as opposite to a definition by induction in which at each step the depth of the object sorts appearing in the sorting Ob is decreased by one. The single step allows us to use the compilation $\mathcal{C}$ also in the case in which Ob is infinite and there is no limit on the depth of the sorts in Ob. Moreover, an inductive definition would have not much simplified the correctness proofs of $\mathcal{C}$.

A few considerations to emphasise the faithfulness of $\mathcal{C}$ are worthwhile. By itself Theorem 5.2.1 does not reveal anything about how closely $\mathcal{C}[\![P]\!]$ simulates P; actually, nothing prevents from obtaining the same results with a very bizarre encoding. First of all, let us show the operational correspondence existing between P and $\mathcal{C}[\![P]\!]$, as derived by composing Lemmas 4.5.10 and 5.1.5

Lemma 5.2.2 (operational correspondence between P and $\mathcal{C}[\![P]\!]$) *Suppose $m \notin fn(P, \mathcal{C}[\![P]\!])$:*

1. (a) *If* $P \xrightarrow{(\boldsymbol{\nu}\,\widetilde{b})\overline{a}\langle F\rangle} P'$, *then* $\mathcal{C}[\![P]\!] \xrightarrow{(\boldsymbol{\nu}\,m)\overline{a}\langle m\rangle}\equiv \mathcal{C}[\![\boldsymbol{\nu}\,\widetilde{b}\,(P' \mid\, !\,m \star F)]\!]$;

   (b) *if* $P \xrightarrow{a\langle Tr_m\rangle} P'$, *then* $\mathcal{C}[\![P]\!] \xrightarrow{a\langle m\rangle} \mathcal{C}[\![P']\!]$,

   (c) *if* $P \xrightarrow{\tau} P'$, *then either*

   - $\mathcal{C}[\![P]\!] \xrightarrow{\tau} \mathcal{C}[\![P']\!]$ *or*
   - $\widetilde{b}, G, F$ *exist s.t.*
     $P' \equiv \boldsymbol{\nu}\,\widetilde{b}\,G\langle F\rangle$, *and* $\mathcal{C}[\![P]\!] \xrightarrow{\tau}\equiv \mathcal{C}[\![\boldsymbol{\nu}\,\widetilde{b}\,(G\langle Tr_m\rangle\,\{m := F\})]\!]$;

   (d) *if* $P \xrightarrow{\mu} P'$, *and* μ *is a first-order input or output, then* $\mathcal{C}[\![P]\!] \xrightarrow{\mu} \mathcal{C}[\![P']\!]$;

2. *the converse of (1), i.e:*

   (a) *If* $\mathcal{C}[\![P]\!] \xrightarrow{(\boldsymbol{\nu}\,m)\overline{a}\langle m\rangle} P''$, *then* $\widetilde{b}$, F, P' *exist s.t.*
   $P \xrightarrow{(\boldsymbol{\nu}\,\widetilde{b})\overline{a}\langle F\rangle} P'$, *and* $P'' \equiv \mathcal{C}[\![\boldsymbol{\nu}\,\widetilde{b}\,(P' \mid\, !\,m \star F)]\!]$;

   (b) *if* $\mathcal{C}[\![P]\!] \xrightarrow{a\langle m\rangle} P''$ *then* P' *exists s.t.*
   $P \xrightarrow{a\langle Tr_m\rangle} P'$, *and* $P'' = \mathcal{C}[\![P']\!]$;

   (c) *if* $\mathcal{C}[\![P]\!] \xrightarrow{\tau} P''$ *then either*

   - P' *exists s.t.* $P \xrightarrow{\tau} P'$ *and* $P'' = \mathcal{C}[\![P']\!]$, *or*
   - $\widetilde{b}, G, F$ *exist s.t.*
     $P \xrightarrow{\tau}\equiv \boldsymbol{\nu}\,\widetilde{b}(G\langle F\rangle)$ *and* $P'' \equiv \mathcal{C}[\![\boldsymbol{\nu}\,\widetilde{b}\,(G\langle Tr_m\rangle\,\{m := F\})]\!]$;

(d) if $\mathcal{C}[\![P]\!] \xrightarrow{\mu} P''$ *and* μ *is a first-order input or output, then* P' *exists s.t.* $P \xrightarrow{\mu} P'$, *and* $P'' = \mathcal{C}[\![P']\!]$. $\square$

Secondly, let us point out that by definitions of $\mathcal{T}$ and $\mathcal{F}$, if P is a first-order process then it is not modified by $\mathcal{C}$, i.e.

$$\mathcal{C}[\![P]\!] = P$$

Thirdly, suppose that P is a HOπ process which can only perform first-order actions. This does not imply that P is also a π-calculus process, as *internally* P could perform communications of agents. But if some extension of the domain of $\approx_{\mathrm{e}}$ is conceded, then $\approx_{\mathrm{e}}$ can be used to compare P with π-calculus processes, and from Lemma 5.2.2 we get

$$P \approx_{\mathrm{e}} \mathcal{C}[\![P]\!]$$

Optimisations?

There are critical points in the definition of $\mathcal{C}$ on agents which is worth indicating. We doubt that non-trivial improvements are possible without losing full abstraction. The first optimisation one might be tempted of, is on the output of agent-variables, defining

$$\mathcal{C}[\![\overline{a}\langle X\rangle.Q]\!] \stackrel{def}{=} \overline{a}\langle x\rangle.\mathcal{C}[\![Q]\!] \tag{$*$}$$

This is shorter than the rule we gave; for instance, in the simplest case where X is a first-order abstraction, the definition in Table 5–2 gives:

$$\mathcal{C}[\![\overline{a}\langle X\rangle.Q]\!] \stackrel{def}{=} \left(\overline{a}\langle m\rangle.\mathcal{C}[\![Q]\!]\right)\{m := (y)\overline{x}\langle y\rangle\} \tag{$**$}$$

The intuitive justification for $(*)$ would be the following. Suppose that previous interactions have instantiated the variable X of the term $\overline{a}\langle X\rangle.Q$ with F. In the encoding π-calculus terms the simulation of these interactions causes the instantiation of the name x with a trigger, say m_F, to $\mathcal{C}[\![F]\!]$. Now, with the rule $(*)$ this same trigger m_F is then transmitted in the output along a. Instead, with the rule $(**)$ a new name-trigger m to the term $(y)\overline{m_F}\langle y\rangle$ is transmitted: This just seems to introduce a further level of indirection to the activation of $\mathcal{C}[\![F]\!]$.

But rule $(*)$ in general is *not* sound. The problem has to do with sharing. With rule $(*)$ two outputs of the same variable become at first-order outputs of name-triggers, or pointers, to the same term; this identity can be recognised and affect the successive behaviour. Consider in fact

$$\begin{aligned} P &\stackrel{def}{=} \boldsymbol{\nu}\, a \left(\overline{a}\langle F\rangle \mid a(X).\overline{b}\langle X\rangle.\overline{b}\langle X\rangle\right) \\ Q &\stackrel{def}{=} \boldsymbol{\nu}\, a \left(\overline{a}\langle F\rangle \mid a(X).\overline{b}\langle F\rangle.\overline{b}\langle F\rangle\right) \end{aligned}$$

Clearly P and Q are equivalent, since they reduce in one step to the same term. But adopting rule $(*)$ their translations are not! In fact we would have

$$\begin{aligned} \mathcal{C}[\![P]\!] &\stackrel{def}{=} \boldsymbol{\nu}\, a \left(\overline{a}\langle m\rangle \{m := F\} \mid a(x).\overline{b}\langle x\rangle.\overline{b}\langle x\rangle\right) \\ \mathcal{C}[\![Q]\!] &\stackrel{def}{=} \boldsymbol{\nu}\, a \left(\overline{a}\langle m\rangle \{m := F\} \mid a(x).\left(\overline{b}\langle m_1\rangle.\overline{b}\langle m_2\rangle \{m_2 := F\}\right) \{m_1 := F\}\right) \end{aligned}$$

which can be distinguished since after the initial interaction, $\mathcal{C}[\![Q]\!]$ can perform two outputs of private (and hence distinct) names at b, namely m_1 and m_2, whereas in $\mathcal{C}[\![P]\!]$ the two outputs at b communicate the same name.

For similar reasons, the optimisation

$$\mathcal{C}[\![X\langle Y\rangle]\!] \stackrel{def}{=} \overline{x}\langle y\rangle.\mathbf{0}$$

is not sound. For this, take the equivalent HOπ processes

$$c(X).\boldsymbol{\nu}\, a \left(a(Y).(X\langle Y\rangle \mid X\langle Y\rangle) \mid \overline{a}\langle F\rangle\right)$$

and

$$c(X).\boldsymbol{\nu}\, a \left(a(Y).(X\langle F\rangle \mid X\langle F\rangle) \mid \overline{a}\langle F\rangle\right).$$

Their π-calculus translations would be distinguished by reasoning similarly as above. There are however situations when the optimisations considered are indeed sound; we leave for future work more precise answers to this issue.

5.3 Related work: Thomsen's encoding and higher-order bisimulation

The first attempt at encoding a higher-order process calculus into π-calculus was made by Thomsen. For this, he used Plain CHOCS (PC) which, as mentioned in Section 2.3, corresponds to the subcalculus of HOπ which has the sorting $\{Name \mapsto (())\}$.

Thomsen's study acted as stimulus and basis for our work. When applied to PC, our compilation $\mathcal{C}$ coincides with Thomsen's translation and in this sense can be seen as an extension of the latter. Our analysis however strengthens and completes Thomsen's in various aspects. First, PC is a (special case of) a second-order language, whereas HOπ is of ω-order. Secondly, to establish an operational correspondence between PC processes and their π-calculus encodings, Thomsen has to make some modifications to the transition system for PC. This seems rather arbitrary and obscures the meaning of the results obtained. Thirdly, Thomsen does not succeed in proving full abstraction for his encoding, i.e. something like

$$P \overset{\sim}{\leftrightarrow}_1 Q \text{ iff } \mathcal{C}[\![P]\!] \overset{\sim}{\leftrightarrow}_2 \mathcal{C}[\![Q]\!] \tag{5.1}$$

for some natural equivalence $\overset{\sim}{\leftrightarrow}_1$ and $\overset{\sim}{\leftrightarrow}_2$. But he conjectures that this is true if $\overset{\sim}{\leftrightarrow}_1$ is the variant of higher-order bisimulation obtained using the modified transition system and $\overset{\sim}{\leftrightarrow}_2$ is the ordinary bisimulation for π-calculus as defined in [MPW92].

But we can show that the implication from right to left fails. Take the processes

$$P \overset{def}{=} \overline{a}\langle \mathbf{0}\rangle.\mathbf{0} \quad \text{and} \quad Q \overset{def}{=} (\boldsymbol{\nu}\, x)\overline{a}\langle \overline{x}.\mathbf{0}\rangle.\mathbf{0} \tag{5.2}$$

There is no way using higher-order bisimulation, to avoid the distinction between them (see Section 1.3), although one reasonably expects them to be equivalent. This is also confirmed by the result of the encoding into π-calculus, where $\mathcal{C}[\![P]\!]$ and $\mathcal{C}[\![Q]\!]$ are strongly bisimilar.

All this offers us the opportunity of a comparison between higher-order bisimulation and barbed bisimulation in higher-order process calculi. First, the precise definition of higher-order bisimulation. We present here its version for HOπ.

Definition 5.3.1 *Higher-order bisimulation, written $\sim_{\mathrm{H}}$, is the largest symmetric relation on HOπ processes s.t. $\sim_{\mathrm{H}}$ is a first-order simulation and $P \sim_{\mathrm{H}} Q$ implies:*

1. *whenever $P \xrightarrow{a\langle F\rangle} P'$, there exists Q' s.t. $Q \xrightarrow{a\langle F\rangle} Q'$ and $P' \sim_{\mathrm{H}} Q'$,*

2. *whenever* $P \overset{(\boldsymbol{\nu}\widetilde{b})\overline{a}\langle F\rangle}{\longrightarrow} P'$ *with* $\widetilde{b} \cap fn(P,Q) = \emptyset$ *, there exist* Q', E *and* $\widetilde{c}$ *with* $\widetilde{c} \cap fn(P,Q) = \emptyset$ *s.t.* $Q \overset{(\boldsymbol{\nu}\widetilde{c})\overline{a}\langle E\rangle}{\longrightarrow} Q'$ *and* $F \sim_{\mathrm{H}} E$*,* $P' \sim_{\mathrm{H}} Q'$*.*

[In clause (2), $\sim_{\mathrm{H}}$ *is defined on abstractions and open agents as expected, i.e.* $F \sim_{\mathrm{H}} E$ *if for every* G *(resp. for every name* a *if* F, E *are first-order abstractions),* $F\langle G\rangle \sim_{\mathrm{H}} E\langle G\rangle$ *(resp.* $F\langle a\rangle \sim_{\mathrm{H}} E\langle a\rangle$*).]* $\square$

There is a necessary remark to make about clause (2). If we had faithfully followed the definition of $\sim_{\mathrm{H}}$ on PC in [Tho90], we would have proposed:

(2′) whenever $P \overset{(\boldsymbol{\nu}\widetilde{b})\overline{a}\langle F\rangle}{\longrightarrow} P'$ with $\widetilde{b} \cap fn(P,Q) = \emptyset$, there exist Q', E s.t. $Q \overset{(\boldsymbol{\nu}\widetilde{b})\overline{a}\langle E\rangle}{\longrightarrow} Q'$ and $F \sim_{\mathrm{H}} E$, $P' \sim_{\mathrm{H}} Q'$.

The difference is that in (2′) the same set of names, namely $\widetilde{b}$, is exported by P and Q. Now, this might appear reasonable if, following Thomsen, we added the transition rule

$$\frac{P \overset{(\boldsymbol{\nu}\widetilde{b})\overline{a}\langle F\rangle}{\longrightarrow} P'}{P \overset{(\boldsymbol{\nu}\widetilde{b'})\overline{a}\langle F\rangle}{\longrightarrow} P'} \quad \widetilde{b} \subseteq \widetilde{b'} \text{ and } \widetilde{b'} \cap fn(P) = \emptyset \tag{5.3}$$

which allows to augment the set of names exported. Unfortunately, rule (5.3) does not seem to be sufficient. For instance, consider the following processes, in which for simplicity we have used only names which can either carry a process or the empty tuple:

$$P = \overline{a}\langle \mathbf{0}\rangle.\mathbf{0} \qquad\qquad Q = (\boldsymbol{\nu}\, c)\overline{a}\langle \boldsymbol{\nu}\, b\,(b.c)\rangle.\mathbf{0}$$

Since in both cases the object part of the name a is a deadlocked process, we would like to have $P \sim_{\mathrm{H}} Q$. However, they are *not* equivalent applying clause (2′), and even adding the rule (5.3). In fact, in every action of Q at least one restriction — namely the restriction on c — is exported. This is not true for P, since we have $P \overset{\overline{a}\langle \mathbf{0}\rangle}{\longrightarrow} \mathbf{0}$ which hence, cannot be matched by Q. Building on this example, one can also show that if (2′) is used, then the congruence of $\sim_{\mathrm{H}}$ over the restriction operator is lost. In fact, take $R \overset{def}{=} \overline{a}\langle \boldsymbol{\nu}\, b\,(b.c)\rangle.\mathbf{0}$; then $P \sim_{\mathrm{H}} R$, but

$$P \sim_{\mathrm{H}} \boldsymbol{\nu}\, c\, P \not\sim_{\mathrm{H}} \boldsymbol{\nu}\, c\, R = Q$$

A way of adjusting clause (2′) could also be to keep the rule (5.3) but then replace $Q \stackrel{(\boldsymbol{\nu}\widetilde{b})\overline{a}\langle E\rangle}{\longrightarrow} Q'$ in (2′) with $Q \stackrel{(\boldsymbol{\nu}\widetilde{b'})\overline{a}\langle E\rangle}{\longrightarrow} Q'$, for some $\widetilde{b'} \supseteq \widetilde{b}$. Then we conjecture that the induced relation would be the same as the one obtained from our Definition 5.3.1.

We now try to compare our version of weak higher-order bisimulation, denoted by $\approx_{\mathrm{H}}$ and defined in the standard way from $\sim_{\mathrm{H}}$, with barbed equivalence.

Theorem 5.3.2 *The relation $\approx_{\mathrm{H}}$ is strictly included in $\approx$.*

One can show $\approx_{\mathrm{H}} \neq \approx$ using the processes P, Q in (5.2). The hard part is to show the inclusion. The "simplest" way we could find is based on the proof that

$$\mathcal{R} = \Big\{(P,Q) \ : \ \exists \widetilde{b}, n,\ P_i \approx_{\mathrm{H}} Q_i, \text{ for } 1 \leq i \leq n, \text{ and } P = \boldsymbol{\nu}\,\widetilde{b}\left(\textstyle\prod_{i=1}^{n} \mathcal{T}[\![P_i]\!]\right),\ Q = \boldsymbol{\nu}\,\widetilde{b}\left(\textstyle\prod_{i=1}^{n} \mathcal{T}[\![Q_i]\!]\right)\Big\} \quad \subseteq \approxeq_{\mathrm{Tr}}$$

From this, if $P \approx_{\mathrm{H}} Q$ then we can infer $\mathcal{T}[\![P]\!] \approxeq_{\mathrm{Tr}} \mathcal{T}[\![Q]\!]$, from which, using the theory developed in Chapter 4, also $P \approx Q$. We cannot hope to get the first of these implications by proving that $\mathcal{T}$ respects $\approx_{\mathrm{H}}$ because this is in general false. The proof that $\mathcal{R} \subseteq \approxeq_{\mathrm{Tr}}$ is non-trivial; for this — among the others — we need a more powerful up-to technique than those described in Section 2.5.3; for instance the one employing expansions and described in the last section of [SM92] would do. We have not reported the details here, on one hand because they involve technicalities which go beyond the scope of this thesis, on the other hand because if we had available the congruence of $\approx_{\mathrm{H}}$ over parallel composition and restriction (which seem non-trivial), the inclusion $\approx_{\mathrm{H}} \subseteq \approx$ could be derived with a very simple proof. Therefore, since without these congruences $\approx_{\mathrm{H}}$ is probably almost useless, we thought more interesting to present the simpler – although incomplete – proof, to show where the need for the congruence of $\approx_{\mathrm{H}}$ arises.

We want to prove that

$$\mathcal{R}' = \{(P,Q) \ : \ P \approx_{\mathrm{H}} Q\}$$

is a $\approxeq_{\mathrm{Nr}}$-bisimulation. The interesting part is the verification of the clause for higher-order outputs. If $P \stackrel{(\boldsymbol{\nu}\widetilde{b})\overline{a}\langle F\rangle}{\longrightarrow} P'$, then we need $\widetilde{c}, Q'$, and E s.t. $Q \stackrel{(\boldsymbol{\nu}\widetilde{c})\overline{a}\langle E\rangle}{\longrightarrow} Q'$

and

$$\boldsymbol{\nu}\,\widetilde{b}\,(\,!\,m \star F \mid P'\,)\;\mathcal{R}\,'\boldsymbol{\nu}\,\widetilde{c}\,(\,!\,m \star E \mid Q'\,) \tag{5.4}$$

First, using α-conversion we can suppose $\widetilde{b} \cap fn(P, Q) = \emptyset$. Now, since $P \approx_{\mathrm{H}} Q$, we have

$$Q \overset{(\boldsymbol{\nu}\,\widetilde{c})\overline{a}\langle E\rangle}{\longrightarrow} Q',$$

where $\widetilde{c} \cap fn(P, Q) = \emptyset$, $F \approx_{\mathrm{H}} E$, $P' \approx_{\mathrm{H}} Q'$. At this point, we would like to use the congruence properties of $\approx_{\mathrm{H}}$ and infer

$$(\boldsymbol{\nu}\,\widetilde{b} \cup \widetilde{c})(\,!\,m \star F \mid P'\,) \approx_{\mathrm{H}} (\boldsymbol{\nu}\,\widetilde{b} \cup \widetilde{c})(\,!\,m \star E \mid Q'\,)$$

from here, (5.4) can be derived by canceling the appropriate restrictions. $\square$

Notice that the result above also proves that the implication from left to right in (5.1) holds.

So what can be our conclusion on the comparison between $\approx_{\mathrm{H}}$ and $\approx$? First of all, in our view the latter is more natural than the former; some distinction made by $\approx_{\mathrm{H}}$ seem difficult to justify. However, at least we can say that $\approx_{\mathrm{H}}$ is *sound*, in that the equivalences that it gives are true for $\approx$. This means that $\approx_{\mathrm{H}}$ can represent a useful tool for proving $\approx$ via *decomposition*, since in the clause for output, $\approx_{\mathrm{H}}$ allows a separate analysis of the object part and of the continuation. But to make this conceivable, it remains to prove the congruence properties of $\approx_{\mathrm{H}}$ and to check whether the quantification over all agents in clause (1) of the definition of $\approx_{\mathrm{H}}$ can be removed by employing triggers (or something similar to these) as it is for $\cong_{\mathrm{Nr}}$.

Chapter 6

An Investigation into Functions as Processes

6.1 Introduction

In [Mil90a], Milner presents the encodings of lazy λ-calculus and (a weak version of) call-by-value λ-calculus into π-calculus. In each case, he compares the equivalence induced on λ-terms, where two λ-terms are identified if their process encodings are behavioral equivalent, with Abramsky's applicative bisimulation. He proves that the former implies the latter, but that the other way round fails. It is left as an open problem to characterise this induced equivalence. It remains also to be studied which kind of λ-model – if any – can be constructed from the π-terms.

The purpose of this chapter is twofold:

- To continues the study of Milner's encodings;
- To show the usefulness of compilation $\mathcal{C}$.

Using the abstraction power of the HOπ, we give encodings both for the lazy and for the call-by-value λ-calculus which are easier to understand and to deal with than those available in the π-calculus. This is especially evident for the lazy λ-calculus which, following Boudol's terminology in [Bou89], is made a *subcalculus* of HOπ, in the sense that β-reduction becomes an *instance* of the communication law with a one-to-one correspondence between the two. By applying compilation $\mathcal{C}$, we can turn the HOπ encodings into π-calculus encodings which can then be compared with Milner's. This is interesting

- to test the canonicity of the encodings,
- to examine the efficiency of $\mathcal{C}$.

In the lazy λ-calculus the outcome of applying $\mathcal{C}$ to the HOπ encoding is precisely Milner's encoding (see the commutative diagram at page 11). Hence all results and properties proved for one of the encodings can then be extended to the other. By working in the HOπ, we are able to show that they give rise indeed to a λ-model, where a weak form of extensionality holds. The model is not fully abstract, though. To obtain full abstraction we follow two directions: In the *restrictive* approach, based on the use of barbed bisimulation, the semantic domain is cut down; in the *expansive* approach λ-calculus is enriched with constants to obtain a *direct* characterisation of the equivalence induced by the encodings.

In call-by-value the situation is less clear. In [Mil90a] Milner presents two candidates for the encoding, and it is not obvious which one should be preferred: The first one allows an easier reasoning, but the second one is more efficient. Moreover, when applied to the HOπ encoding, compilation $\mathcal{C}$ does not return either of them. Apparently, to obtain them, some code transformation has to be carried out. The study of these transformations leads to interesting conclusions. First, it suggests a correction of the order in which some actions appear in Milner's encodings. This rearrangement does not affect the operational correspondence between λ- and π-terms. However, it affects the equivalence on the encoding π-terms, in a way which makes the encoding more faithful to the encoded call-by-value discipline. Secondly, the study of the transformations reveals that β-reduction is not valid in Milner's second encoding, which reduces a lot its importance.[1] The counter-

[1] The version of [Mil90a] appeared in the *Jour. of Math. Structures* was written when the results in this chapter were already known and thus presents only the first encoding.

example is fairly sophisticated and we doubt we could have obtained it without going through HOπ.

We adopt $\approx^{c}$ (weak barbed congruence) as behavioural equivalence for the terms of our calculi. In the proofs, we shall often use the characterisations of $\approx^{c}$ in terms of weak early congruence for π-calculus (Corollary 4.7.9), and in terms of weak context congruence and weak normal congruence for HOπ (Corollary 3.3.4). However, due to the particular structure of agents encoding λ-terms, the results we present should be — to a reasonable extent — independent from the choice of the process equivalence.

A remark about the use of the full abstraction of compilation $\mathcal{C}$ and of the characterisations of $\approx^{c}$, presented in the previous chapters. These results were proved on agents which may exchange one only value in their communications. We already stressed that this does not intend to be a limitation on their applicability. The choice of unary-tuple was made to make the proof-techniques more readable — by distinguishing the requirements on higher-order and first-order objects — and the generalisation to arbitrary-tuple communications is completely smooth (see also the discussion in Section 4.7.3). In this chapter, we shall apply such results to the polyadic calculi (precisely, to processes which may perform *binary* communications).

Structure of the chapter: In Section 6.2 we recall some preliminaries on λ-calculus. In Section 6.3 we present the encodings of lazy λ-calculus into π-calculus and HOπ and we make the comparison between them via $\mathcal{C}$. In Section 6.4 we do the same for call-by-value λ-calculus. In Sections 6.5 and 6.6 we examine the lazy λ-calculus encodings from the point of view of the model theory of λ-calculus and we study the equivalence on the λ-terms induced by the behavioural equivalence adopted on the process calculi.

6.2 λ-calculus: Preliminaries on its syntax and its encoding

Syntax and notation

λ-calculus is the commonly accepted foundation for functional programming. The syntax is simple, and based on the operators of abstraction and application. An evaluation mechanism defines the operational meaning of a term built out of this syntax.

We use x and y to range over the set of λ-calculus variables. To simplify the encodings, x, y will also represent names in the π-calculus. We use L, M, N to range over the class Λ^{o} of open λ-terms, as usual given by:

$$M \quad ::= \quad x \quad | \quad \lambda x.M \quad | \quad MN$$

The definitions of free variables, closed terms, substitutions, α-conversion etc. are taken to be the standard ones (see [Bar84], [HS86]). We shall not distinguish between α-convertible terms and we write $M = N$ if M and N are α-convertible. The subclass of closed λ-terms is denoted by Λ. A closed term $C_{\lambda}[\cdot]$ with a single hole in it is a λ-*context*. We use Ω to denote the (divergent) term $\lambda x.xx(\lambda x.xx)$.

λ-calculus encodings

The idea common to all various attempts at embedding λ-calculus into a process calculus [Bou89,Bou92,Tho90,Mil90a] is to view functional application as a particular parallel combination of two agents, the function and its argument, and β reduction as a particular case of communication. The encodings we shall present follow the same idea.

A λ-term is encoded as a map from names of a certain sort s to processes, that is, as an abstraction of sort (s). To lighten the exposition, we shall omit the sortings. To derive them, it is enough to say that they are always the expected ones, and that in each case, at most three sorts are needed; one for names p, q, r, another one for x, y, the last one for v, w. Moreover, these sortings always behave well w.r.t. compilation $\mathcal{C}$, in the following sense: Whenever an HOπ encoding Ob_H is compared with a π-calculus encoding Ob_π via $\mathcal{C}$, and Ob_H, Ob_π do not include redundant sorts (i.e. sorts not used in the encoding) then it is always true that $\mathcal{C}[\![Ob_H]\!] = Ob_\pi$.

In the algebraic manipulations on HOπ processes sometimes we shall need the following simple form of the *expansion law*. Suppose that the process R can only perform visible actions at names in $\widetilde{m}$, i.e., $\{a \;:\; R \Downarrow_a\} \subseteq \widetilde{m}$; then

$$\boldsymbol{\nu}\, a, \widetilde{m}\left(\overline{a}\langle\widetilde{K}\rangle.P \mid a(\widetilde{U}).Q \mid R\right) \approx^{\mathrm{c}} \boldsymbol{\nu}\, a, \widetilde{m}\left(P \mid Q\{\widetilde{K}/\widetilde{U}\} \mid R\right)$$

Note that for $R = \mathbf{0}$, the result is proved by Lemmas 4.4.4(1) and 4.4.2, and Corollary 4.7.9. We shall use this expansion law a few times in this chapter.

We adopt the following — trivial — simplification of the encoding $\mathcal{C}$ on output processes whose continuation is the $\mathbf{0}$ process: We shall use

$$\mathcal{C}[\![\overline{a}\langle F\rangle.\mathbf{0}]\!] \stackrel{def}{=} \boldsymbol{\nu}\, m\,(\overline{a}\langle m\rangle.\,!\,m \star \mathcal{C}[\![F]\!])$$

rather than

$$\mathcal{C}[\![\overline{a}\langle F\rangle.\mathbf{0}]\!] \stackrel{def}{=} \boldsymbol{\nu}\, m\,(\overline{a}\langle m\rangle.\mathbf{0} \mid\, !\,m \star \mathcal{C}[\![F]\!])$$

6.3 The Lazy λ-Calculus Encoding

The lazy λ-calculus is proposed by Abramsky in [Abr87] as a basis for lazy functional programming languages and the evaluation mechanism is guided by what the implementation of such languages suggests; in particular reductions are forbidden within a λ-abstraction.

The reduction relation $\Longrightarrow\, \subseteq \Lambda \times \Lambda$ is defined by rules (β) and (App), the core rules for the lazy λ-calculus, plus rules *Refl* and *Trans*, describing the reflexive and transitive nature of $\Longrightarrow$.

$$(\beta)\ (\lambda x.M)N \Longrightarrow M\{N/x\} \qquad\qquad (App)\ \frac{M \Longrightarrow M'}{MN \Longrightarrow M'N}$$

$$(Refl)\ M \Longrightarrow M \qquad\qquad (Trans)\ \frac{M \Longrightarrow M' \qquad M' \Longrightarrow M''}{M \Longrightarrow M''}$$

6.3.1 Applicative Bisimulation

The lazy reduction strategy admits only reductions at the extreme left of a term and hence is deterministic. A term either converges to an abstraction or diverges. The convergence predicate $\Downarrow$ is so defined:

$$\lambda x.M \Downarrow \qquad\qquad \frac{M \Longrightarrow M' \Downarrow}{M \Downarrow}$$

Inspired by the work of Milner and Park in concurrency [Par81,Mil83], Abramsky [Abr87] introduces an operational equivalence on λ-terms called *applicative bisimulation*, built on the idea that convergence is the only observable property.[1]

Definition 6.3.1 Applicative bisimulation *is the largest symmetric relation* $\cong \subseteq \Lambda \times \Lambda$ *s.t. if* $M \cong N$ *and* $M \Longrightarrow \lambda x.M'$*, then some* N' *exists s.t.* $N \Longrightarrow \lambda x.N'$ *and for all* L*,* $M'\{L/x\} \cong N'\{L/x\}$*.* $\Box$

The relation $\cong$ is extended to open λ-terms using substitutions: if M and N are open λ-terms with $\{x_1, \ldots, x_n\}$ as free variables, then $M \cong N$ if for all closed $L_1, \ldots, L_n$, it holds that $M\{L_1/x_1, \ldots, L_n/x_n\} \cong N\{L_1/x_1, \ldots, L_n/x_n\}$. It is easy to see that $\cong$ is an equivalence. Moreover, since the reduction relation is deterministic, it holds that

Lemma 6.3.2 *If* $M \longrightarrow M'$*, then* $M \cong M'$*.* $\Box$

The relation $\cong$ has been extensively studied by Abramsky and Ong [Ong88, AO89]. They show that $\cong$ is a congruence using the characterisation in Theorem 6.3.4 below.

Definition 6.3.3 *Let* $M, N \in \Lambda$*. We write*

- $M \stackrel{\bullet}{\approx} N$ *when* $M \Downarrow$ *iff* $N \Downarrow$*.*

[1] Abramsky's original definition is in terms of *simulation* rather then bisimulation. In fact, he mainly works with preorders. He then considers the equivalence induced by such preorder: Since the reduction relation is deterministic, his definition is equivalent to our Definition 6.3.1. Moreover Abramsky uses the name applicative bisimulation for the preorder; but it seems more accurate and in line with the concurrency tradition to reserve this word for the equivalence.

- $M \approx^{c} N$ *when* $C_\lambda[M] \stackrel{\bullet}{\approx} C_\lambda[N]$ *holds for each closed* λ*-context* $C_\lambda[\cdot]$.

The symbology in Definition 6.3.3 is motivated by the fact that if we think of λ as the only port of λ-calculus and $M \Downarrow$ as meaning "M can interact at λ", then the relations introduced are the corresponding on λ-calculus of weak barbed bisimulation and congruence.

Theorem 6.3.4 (Abramsky, Ong) *On closed* λ*-term,* $\cong$ *and* $\approx^{c}$ *coincide.*

Proof: The difficult part is the inclusion $\cong \subseteq \approx^{c}$. This result is proved in [Abr87] by going through domain logic. A simpler proof has been suggested by Alan Stoughton and proceeds by induction on the structure of the contexts. See [AO89] for the details. □

Notice how close the above result resembles those in Chapters 4 and 5 about direct characterisations of barbed congruence in π-calculus and HOπ. This similitude will be in fact exploited in Section 6.6.1 to obtain certain full abstraction results.

Corollary 6.3.5 *For each* $M, N \in \Lambda^{o}$ *and* λ*-contexts* $C_\lambda[\cdot]$*,* $M \cong N$ *implies* $C_\lambda[M] \cong C_\lambda[N]$.

Proof: If M and N are closed, the result follows from Theorem 6.3.4. The result is extended to open terms proceeding by induction on the structure of $C_\lambda[\cdot]$. □

6.3.2 Encoding into π-calculus

We use $\mathcal{P}$ to denote Milner's encoding of the lazy λ-calculus into π-calculus. If $\mathcal{P}[\![M]\!]\langle p\rangle$ is a process resulting from the translation of the term M, the name p acts as a pointer to the argument sequence for M. If M reduces to a λ-abstraction, the corresponding derivative of $\mathcal{P}[\![M]\!]\langle p\rangle$ will receive along the name p a pointer to its first argument and a pointer to the rest of its argument sequence. In the application rule, the restriction on q prevents interference from other processes.

$$\begin{aligned}
\mathcal{P}[\![\lambda x.M]\!] &\stackrel{def}{=} (p)p(x,q).\mathcal{P}[\![M]\!]\langle q\rangle \\
\mathcal{P}[\![x]\!] &\stackrel{def}{=} (p)\overline{x}\langle p\rangle.\mathbf{0} \\
\mathcal{P}[\![MN]\!] &\stackrel{def}{=} (p)\boldsymbol{\nu}\, q\left(\mathcal{P}[\![M]\!]\langle q\rangle \mid (\boldsymbol{\nu}\, x)\overline{q}\langle x,p\rangle.!\,x(r).\mathcal{P}[\![N]\!]\langle r\rangle\right) \\
&\qquad x \text{ not free in } \mathcal{P}[\![N]\!]
\end{aligned}$$

It seems established that $\mathcal{P}$ is canonical, i.e. it is the "best" or "simplest" encoding of the lazy λ-calculus into π-calculus. Nevertheless, one has to think carefully about it – in particular at the encoding of application – to understand that it really does work (see [Mil90a] for examples).

6.3.3 Encoding into HOπ

The encoding into HOπ makes very transparent the idea of regarding function application as a special case of communication. For convenience, a variable x of the λ-calculus is mapped into its upper-case variable X in HOπ.

$$\begin{aligned}
\mathcal{H}[\![\lambda x.M]\!] &\stackrel{def}{=} (p)p(X,q).\mathcal{H}[\![M]\!]\langle q\rangle \\
\mathcal{H}[\![x]\!] &\stackrel{def}{=} X \\
\mathcal{H}[\![MN]\!] &\stackrel{def}{=} (p)\boldsymbol{\nu}\, q\left(\mathcal{H}[\![M]\!]\langle q\rangle \mid \overline{q}\langle \mathcal{H}[\![N]\!],p\rangle.\mathbf{0}\right)
\end{aligned}$$

The theorem below shows how the encoding behaves. There is a one-to-one correspondence between reductions in λ-terms and in their encodings; moreover, to capture validity of β reduction, we simply need an elementary use of structural congruence. Therefore, following Boudol [Bou89], we can claim that *lazy λ-calculus is a subcalculus of HOπ.*

Proposition 6.3.6 (operational correspondence) *Let $M \in \Lambda$:*

1. (a) *If $M \longrightarrow M'$, then $\mathcal{H}[\![M]\!]\langle p\rangle \longrightarrow\equiv \mathcal{H}[\![M']\!]\langle p\rangle$,*

   (b) *the converse, i.e.*

   if $\mathcal{H}[\![M]\!]\langle p\rangle \longrightarrow Q$, then $\exists M' \in \Lambda$ s.t. $Q \equiv \mathcal{H}[\![M']\!]\langle p\rangle$ and $M \longrightarrow M'$.

2. (a) *If $\mathcal{H}[\![M]\!]\langle p\rangle \xrightarrow{\mu}$ for some $\mu \neq \tau$, then M must be of the form $\lambda x.M'$,*

(b) *the converse, i.e.*

if M *is of the form* $\lambda x.M'$, *then* $\mathcal{H}[\![M]\!]\langle p\rangle \xrightarrow{\mu}$, *for some* $\mu \neq \tau$.

Proof: We only consider (1), since (2) is easy. Both (a) and (b) can be proved by induction on the structure of M.

- $M = \lambda x.M'$.

  M and $\mathcal{H}[\![M]\!]\langle p\rangle$ have no reduction.

- $M = M_1M_2$ and $M_1 = \lambda x.M_3$.

  Then $M \longrightarrow M_3\{M_2/x\}$. On the other hand,

$$\begin{array}{rcl} \mathcal{H}[\![(\lambda x.M_3)M_2]\!]\langle p\rangle & = & \\ \boldsymbol{\nu}\, q\left(q(X,r).\mathcal{H}[\![M_3]\!]\langle r\rangle \mid \overline{q}\langle \mathcal{H}[\![M_2]\!], p\rangle\right) & \longrightarrow & \\ \boldsymbol{\nu}\, q\left(\mathcal{H}[\![M_3]\!]\langle p\rangle\{\mathcal{H}[\![M_2]\!]/X\}\right) & \equiv & (q \text{ not free in the body}) \\ \mathcal{H}[\![M_3]\!]\langle p\rangle\{\mathcal{H}[\![M_2]\!]/X\} & = & (\text{because of the compositional} \\ \mathcal{H}[\![M_3\{M_2/x\}]\!]\langle p\rangle & & \text{definition of the encoding}) \end{array}$$

- $M = M_1M_2$ and M_1 is not of the form $\lambda x.M_3$.

  Immediate using the inductive hypothesis, since it must be that $M_1 \longrightarrow M_1'$ and $M' \equiv M_1'M_2$. □

6.3.4 Correspondence between the two encodings

It is easy to verify that the output of compilation $\mathcal{C}$ on the HOπ encoding is exactly Milner's encoding. Using '$\circ$' to denote function composition, we therefore have the following:

Theorem 6.3.7 $\mathcal{C} \circ \mathcal{H} = \mathcal{P}$. □

In virtue of Theorem 5.2.1, this means that we can infer for Milner's encoding all results we can prove working with the HOπ encoding. The latter is simpler to study because it yields a more direct correspondence with the encoded λ-terms; it will be accurately examined in Sections 6.5 and 6.6.

6.4 The Call-by-Value Encoding

In call-by-value λ-calculus, reductions may only occur when the argument is a value, i.e. an abstraction. The reduction relation $\Longrightarrow \subseteq \Lambda \times \Lambda$ used by Milner in [Mil90a] is described by the usual rules *Refl* and *Trans* plus the rules β_v, App_L, App_R.

$$(\beta_v)\ (\lambda x.M)\lambda y.N \Longrightarrow M\{\lambda y.N/x\}$$

$$(App_L)\ \frac{M \Longrightarrow M'}{MN \Longrightarrow M'N} \qquad\qquad (App_R)\ \frac{N \Longrightarrow N'}{MN \Longrightarrow MN'}$$

Reduction $\Longrightarrow$ is no longer deterministic. However, convergence remains deterministic and *strong*: that is if $M \Longrightarrow \lambda x.M'$, then $\lambda x.M'$ is unique and all reduction sequences from M are finite. Convergence and applicative bisimulation are defined for call-by-value in the same way as for lazy λ-calculus.

Now we shall try to repeat for call-by-value what we did in Section 6.3 for lazy λ-calculus: We propose an encoding into HOπ and then we compare it with Milner's into π-calculus through compilation $\mathcal{C}$.

With respect to lazy λ-calculus, the call-by-value encodings are slightly more involved. They also lose their apparent canonicity, which is implicitly confirmed by the fact that in his original work [Mil90a], Milner presents *two* candidates for the encoding. Basically, the problems in the translation of call-by-value are originated by the following *dichotomy* in the behaviour of a λ-abstraction. Take the term MN: Both M and N could reduce to an abstraction; but if it is M, then the abstraction is destined to perform an *input* of a value, whereas if it is N, the abstraction represents an *output* value. This causes disagreement on whether the process which encodes a λ-abstraction should first perform an input or an output.

6.4.1 Encoding into π-calculus

Milner's two candidates for the encoding in [Mil90a] differ only in the rule for the translation of a variable. In the second encoding this rule is simpler, but the first one allows easier reasoning and proofs. In both cases, p is used in $\mathcal{P}_i[\![M]\!]\langle p\rangle$ to communicate (with an output action) that M has reduced to a value. In contrast with lazy λ-calculus, the translation of an application allows the two arguments M and N to run in parallel. Then the dichotomy in the behaviour of a λ-abstraction is solved by the arbiter $App\langle p, q, r\rangle$ which imposes the correct interaction between M and N.

FIRST CANDIDATE

$$\begin{aligned}
\mathcal{P}_1[\![\lambda x.M]\!] &\stackrel{def}{=} (p)(\boldsymbol{\nu}\, y)\overline{p}\langle y\rangle.\,!\,y(w).w(x,q).\mathcal{P}_1[\![M]\!]\langle q\rangle \\
\mathcal{P}_1[\![x]\!] &\stackrel{def}{=} (p)(\boldsymbol{\nu}\, y)\overline{p}\langle y\rangle.\,!\,y(w).\overline{x}\langle w\rangle.\mathbf{0} \\
\mathcal{P}_1[\![MN]\!] &\stackrel{def}{=} (p)(\boldsymbol{\nu}\, q,r)\big(\mathcal{P}_1[\![M]\!]\langle q\rangle \mid \mathcal{P}_1[\![N]\!]\langle r\rangle \mid App\langle p,q,r\rangle\big) \\
&\quad\;\text{where } App \stackrel{def}{=} (p,q,r)q(x).(\boldsymbol{\nu}\, v)\overline{x}\langle v\rangle.r(y).\overline{v}\langle y,p\rangle.\mathbf{0}
\end{aligned}$$

SECOND CANDIDATE

Let us denote it with $\mathcal{P}_2$. The only difference with the first encoding is in the clause for the variable, which now is

$$\mathcal{P}_2[\![x]\!] \stackrel{def}{=} (p)\overline{p}\langle x\rangle.\mathbf{0}$$

6.4.2 Encoding into HOπ

The idea of the encoding into HOπ is the same as for the encoding into π-calculus, but it can be implemented more directly.

$$\begin{aligned}
\mathcal{H}[\![\lambda x.M]\!] &\stackrel{def}{=} (p)\overline{p}\Big\langle (w)w(X,q).\mathcal{H}[\![M]\!]\langle q\rangle\Big\rangle.\mathbf{0} \\
\mathcal{H}[\![x]\!] &\stackrel{def}{=} (p)\overline{p}\langle X\rangle.\mathbf{0} \\
\mathcal{H}[\![MN]\!] &\stackrel{def}{=} (p)(\boldsymbol{\nu}\, q,r)\big(\mathcal{H}[\![M]\!]\langle q\rangle \mid \mathcal{H}[\![N]\!]\langle r\rangle \mid App_H\langle p,q,r\rangle\big) \\
&\quad\;\text{where } App_H \stackrel{def}{=} (p,q,r)q(X).r(Y).\boldsymbol{\nu}\, v\,(X\langle v\rangle \mid \overline{v}\langle Y,p\rangle.\mathbf{0})
\end{aligned}$$

The format of the translation of variables resembles the one in Milner's second encoding $\mathcal{P}_2$. But the similarity is only syntactical. In section 6.4.3 we shall see, using the compilation $\mathcal{C}$, that semantically $\mathcal{H}$ agrees with $\mathcal{P}_1$ rather than $\mathcal{P}_2$.

It is enlightening to relate the HOπ encoding for call-by-value to the one for lazy λ-calculus. In the above rules for abstraction and variable, the "core" is the object part of the output at p, and has the same format of the corresponding rule for lazy λ-calculus. Now the rule for application in call-by-value should become clearer: The arbiter App_H receives along p and r the "cores" of $\mathcal{H}[\![M]\!]$ and $\mathcal{H}[\![N]\!]$ and then imposes on them the "lazy application".

The encoding into HOπ is therefore easier to understand than those in the π-calculus, although the abstraction power of HOπ does not seem to make the same difference as for lazy λ-calculus. To see how $\mathcal{H}$ works, we show the validity of β reduction.

Lemma 6.4.1 (validity of β reduction) $\mathcal{H}[\![(\lambda x.M)\lambda y.N]\!] \approx^{c} \mathcal{H}[\![M\{\lambda y.N/x\}]\!]$

PROOF: Let us abbreviate $\mathcal{H}[\![\lambda x.M]\!]\langle q\rangle$ as $\overline{q}\langle F_M\rangle$ and $\mathcal{H}[\![\lambda x.N]\!]\langle r\rangle$ as $\overline{r}\langle F_N\rangle$, where

$$F_M = (w)w(X,p).\mathcal{H}[\![M]\!]\langle p\rangle \quad \text{and} \quad F_N = (w)w(X,p).\mathcal{H}[\![N]\!]\langle p\rangle$$

By repeated use of the expansion law and some obvious simplification, we get

$$\begin{array}{rcl}
\mathcal{H}[\![(\lambda x.M)\lambda y.N]\!]\langle p\rangle & = & \\
(\boldsymbol{\nu}\, q,r)\big(\overline{q}\langle F_M\rangle \mid \overline{r}\langle F_N\rangle \mid App_H\langle p,q,r\rangle\big) & \approx^{c} & \\
\boldsymbol{\nu}\, v\,\big(F_M\langle v\rangle \mid \overline{v}\langle F_N,p\rangle\big) & = & \qquad\qquad (*)\\
\boldsymbol{\nu}\, v\,\big(v(X,p).\mathcal{H}[\![M]\!]\langle p\rangle \mid \overline{v}\langle F_N,p\rangle\big) & \approx^{c} & \\
\mathcal{H}[\![M]\!]\langle p\rangle\{F_N/X\} & = & \\
\mathcal{H}[\![M\{\lambda y.N/x\}]\!]\langle p\rangle & &
\end{array}$$

where the last equality follows from the compositional definition of the encoding. Notice that $(*)$ is an instance of the lazy application in HOπ. The validity of the latter gives the correctness of the successive reduction. $\square$

6.4.3 Correspondence among the encodings

Let us apply compilation $\mathcal{C}$ to the HOπ encoding and see what we get back.

$$\begin{array}{rcl}
\mathcal{C}[\![\mathcal{H}[\![\lambda x.M]\!]]\!] & = & (p)(\boldsymbol{\nu}\, y)\overline{p}\langle y\rangle.\,!\,y(w).w(x,q).\mathcal{C}[\![\mathcal{H}[\![M]\!]]\!]\langle q\rangle \\
\mathcal{C}[\![\mathcal{H}[\![x]\!]]\!] & = & (p)(\boldsymbol{\nu}\, y)\overline{p}\langle y\rangle.\,!\,y(w).\overline{x}\langle w\rangle.\mathbf{0} \\
\mathcal{C}[\![\mathcal{H}[\![MN]\!]]\!] & = & (p)(\boldsymbol{\nu}\, q,r)\big(\mathcal{C}[\![\mathcal{H}[\![M]\!]]\!]\big)\langle q\rangle \mid \mathcal{C}[\![\mathcal{H}[\![N]\!]]\!]\langle r\rangle \mid \\
& & \qquad q(x).r(y).\boldsymbol{\nu}\, v\,\big(\overline{x}\langle v\rangle \mid (\boldsymbol{\nu}\, m)\overline{v}\langle m,p\rangle.\,!\,m(w).\overline{y}\langle w\rangle.\mathbf{0}\big) \qquad \sim^{c} \\
& & (p)(\boldsymbol{\nu}\, q,r)\big(\mathcal{C}[\![\mathcal{H}[\![M]\!]]\!]\big)\langle q\rangle \mid \mathcal{C}[\![\mathcal{H}[\![N]\!]]\!]\langle r\rangle \mid \\
& & \qquad q(x).r(y).\boldsymbol{\nu}\, v\,\overline{x}\langle v\rangle.(\boldsymbol{\nu}\, m)\overline{v}\langle m,p\rangle.\,!\,m(w).\overline{y}\langle w\rangle.\mathbf{0}
\end{array}$$

The above use of $\sim^{c}$ (strong barbed congruence) has been derived from the algebraic law

$$\boldsymbol{\nu}\, v\, (\overline{x}\langle v\rangle \mid \overline{v}\langle\tilde{y}\rangle.P) \sim^{c} \boldsymbol{\nu}\, v\, (\overline{x}\langle v\rangle.\overline{v}\langle\tilde{y}\rangle.P)$$

(intuitively, since v is restricted, the action $\overline{v}\langle\tilde{y}\rangle$ may only fire after the action $\overline{x}\langle v\rangle$). There are two differences between the encoding $\mathcal{C}\circ\mathcal{H}$ and Milner's encodings $\mathcal{P}_1$ and $\mathcal{P}_2$:

1. The order of the actions $r(y)$ and $\boldsymbol{\nu}\, v\, \overline{x}\langle v\rangle$ in the rule for application is reversed,

2. the process $\boldsymbol{\nu}\, m\, \overline{v}\langle m,p\rangle.\,!\, m(w).\overline{y}\langle w\rangle.\mathbf{0}$ is "optimised" as $\overline{v}\langle y,p\rangle.\mathbf{0}$ in $\mathcal{P}_1$ and $\mathcal{P}_2$; further, in $\mathcal{P}_2$ a similar optimisation occurs in the rule for variable. .

Each of the above differences brings up interesting issues, which we are going to analise in the order. We look first at the point (1), and we consider its effect on $\mathcal{P}_1$ and $\mathcal{P}_2$. We call $\mathcal{P}'_i$, $i = 1,2$ the encoding obtained from $\mathcal{P}_i$ by commuting the actions $r(y)$ and $\boldsymbol{\nu}\, v\, \overline{x}\langle v\rangle$ in the application rule. The action rearrangement in question is sound if the terms M and N of the application are closed; i.e. if $M,N\in\Lambda$ we have (we underline the exchanged actions):

$$\begin{array}{rcl} \mathcal{P}_i[\![MN]\!] & \stackrel{def}{=} & (p)(\boldsymbol{\nu}\, q,r)\big(\mathcal{P}_i[\![M]\!]\langle q\rangle \mid \mathcal{P}_i[\![N]\!]\langle r\rangle \mid q(x).\underline{(\boldsymbol{\nu}\, v)\overline{x}\langle v\rangle.r(y)}.\overline{v}\langle y,p\rangle.\mathbf{0}\big) \\ & \approx^{c} & (p)(\boldsymbol{\nu}\, q,r)\big(\mathcal{P}_i[\![M]\!]\langle q\rangle \mid \mathcal{P}_i[\![N]\!]\langle r\rangle \mid q(x).\underline{r(y).(\boldsymbol{\nu}\, v)\overline{x}\langle v\rangle}.\overline{v}\langle y,p\rangle.\mathbf{0}\big) \end{array}$$

Intuitively, the equivalence holds because the commutation does not affect the order of actions which interact with the same component $\mathcal{P}_i[\![M]\!]$ or $\mathcal{P}_i[\![N]\!]$. This result has an immediate consequence, due to the fact that the λ-strategy we are encoding only allows reductions between closed terms: The operational correspondence between λ and π-terms showed by Milner for the encodings $\mathcal{P}_i$'s, extends to the $\mathcal{P}'_i$'s as well.

However, the difference between $\mathcal{P}_i$ and $\mathcal{P}'_i$ does show up when the terms M and N of the application are open. The encodings $\mathcal{P}'_i$'s appear closer than the $\mathcal{P}_i$'s to the call-by-value intuition. We justify this affirmation with an example. Consider the λ-term $\lambda x.x\Omega$: Since the call-by-value application $x\Omega$ has a divergent argument Ω, the term $x\Omega$ is supposed not to produce any visible behaviour. Consequently, we expect that a faithful encoding of call-by-value equates $\lambda x.x\Omega$ and $\lambda x.\Omega$. But this is true only for the encodings $\mathcal{P}'_i$'s, whereas it fails for the $\mathcal{P}_i$'s.

Lemma 6.4.2 *For $i = 1, 2$, we have:*

- $\mathcal{P}'_i[\![\lambda x.x\Omega]\!] \approx^{c} \mathcal{P}'_i[\![\lambda x.\Omega]\!]$,
- $\mathcal{P}_i[\![\lambda x.x\Omega]\!] \not\approx^{c} \mathcal{P}_i[\![\lambda x.\Omega]\!]$

Proof: (Sketch)It is enough to examine the encodings of $x\Omega$ and Ω. The process $\mathcal{P}_i[\![\Omega]\!]\langle p\rangle$ and $\mathcal{P}'_i[\![\Omega]\!]\langle p\rangle$ do not interact with the environment, hence

$$\mathcal{P}_i[\![\Omega]\!]\langle p\rangle \approx^{c} \mathcal{P}'_i[\![\Omega]\!]\langle p\rangle \approx^{c} \mathbf{0} \tag{6.1}$$

Similarly, it holds that $\mathcal{P}'_i[\![x\Omega]\!] \approx^{c} \mathbf{0}$. We show this for $\mathcal{P}'_1[\![x\Omega]\!]$. The transformations below are obtained applying the expansion law of Section 6.2, the law (6.1) and a few simple structural congruence rules. We have:

$$\begin{array}{rcl}
\mathcal{P}'_1[\![x\Omega]\!] & \stackrel{def}{=} & (\boldsymbol{\nu}\, q, r\,)\big(\boldsymbol{\nu}\, z\, \overline{q}\langle z\rangle.!z(w).\overline{x}\langle w\rangle \mid \mathcal{P}'_1[\![\Omega]\!](r) \mid q(z).r(y).\boldsymbol{\nu}\, v\, \overline{z}\langle v\rangle.\overline{v}\langle y, p\rangle\big) \\
& \approx^{c} & (\boldsymbol{\nu}\, r, z\,)\big(!z(w).\overline{x}\langle w\rangle \mid \mathbf{0} \mid r(y).\boldsymbol{\nu}\, v\, \overline{z}\langle v\rangle.\overline{v}\langle y, p\rangle\big) \approx^{c} \mathbf{0}
\end{array}$$

where the last equality holds because the term on the left is deadlocked and hence equal to $\mathbf{0}$. In contrast, we can show that $\mathcal{P}_1[\![x\Omega]\!]$ does exhibit a visible behaviour:

$$\begin{array}{rcl}
\mathcal{P}_1[\![x\Omega]\!] & \stackrel{def}{=} & (\boldsymbol{\nu}\, q, r\,)\big(\boldsymbol{\nu}\, z\, \overline{q}\langle z\rangle.!z(w).\overline{x}\langle w\rangle \mid \mathcal{P}_1[\![\Omega]\!](r) \mid q(z).\boldsymbol{\nu}\, v\, \overline{z}\langle v\rangle.r(y).\overline{v}\langle y, p\rangle\big) \\
& \approx^{c} & (\boldsymbol{\nu}\, r, z\,)\big(!z(w).\overline{x}\langle w\rangle \mid \mathbf{0} \mid \boldsymbol{\nu}\, v\, \overline{z}\langle v\rangle.r(y).\overline{v}\langle y, p\rangle\big) \\
& \approx^{c} & (\boldsymbol{\nu}\, r, z, v\,)\big(\overline{x}\langle v\rangle \mid !z(w).\overline{x}\langle w\rangle \mid \mathbf{0} \mid r(y).\overline{v}\langle y, p\rangle\big) \quad \xrightarrow{\boldsymbol{\nu}\, v\, \overline{x}\langle v\rangle}
\end{array}$$

Similarly,

$$\begin{array}{rcl}
\mathcal{P}_2[\![x\Omega]\!] & \stackrel{def}{=} & (\boldsymbol{\nu}\, q, r\,)\big(\overline{q}\langle x\rangle \mid \mathcal{P}_1[\![\Omega]\!](r) \mid q(z).\boldsymbol{\nu}\, v\, \overline{z}\langle v\rangle.r(y).\overline{v}\langle y, p\rangle\big) \\
& \approx^{c} & (\boldsymbol{\nu}\, r\,)\big(\mathbf{0} \mid \mathbf{0} \mid \boldsymbol{\nu}\, v\, \overline{x}\langle v\rangle.r(y).\overline{v}\langle y, p\rangle\big) \quad \xrightarrow{\boldsymbol{\nu}\, v\, \overline{x}\langle v\rangle}
\end{array}$$

□

Now we turn to the examination of the second difference between the encoding $\mathcal{C} \circ \mathcal{H}$ and Milner's $\mathcal{P}_i$'s (everything we shall say for the $\mathcal{P}_i$'s holds for their "rectified" $\mathcal{P}'_i$'s). We remind that the difference is the optimisations of

$$\boldsymbol{\nu}\, m\, \overline{v}\langle m, p\rangle.\,!\, m(w).\overline{y}\langle w\rangle.\mathbf{0} \quad \text{with} \quad \overline{v}\langle y, p\rangle.\mathbf{0}$$

in the application rule and, only for $\mathcal{P}_2$, the similar optimisation of

$$(\boldsymbol{\nu}\, y)\overline{p}\langle y\rangle.\,!\,y(w).\overline{x}\langle w\rangle.\mathbf{0} \quad \text{with} \quad \overline{p}\langle x\rangle.\mathbf{0}$$

in the variable rule. We call the former *optimisation 1* and the latter *optimisation 2*. Both of them can be recovered in $\mathcal{C}\circ\mathcal{H}$ by adopting the optimisation of $\mathcal{C}$ discussed in Section 5.2, namely the replacement of the rule

$$\mathcal{C}[\![\overline{a}\langle X\rangle.\mathbf{0}]\!] \stackrel{def}{=} \boldsymbol{\nu}\, m\left(\overline{a}\langle m\rangle.\mathbf{0} \mid\, !\, m(y).\overline{x}\langle y\rangle\right)$$

where X is a variable representing a first-order abstraction, with the simpler

$$\mathcal{C}[\![\overline{a}\langle X\rangle.\mathbf{0}]\!] \stackrel{def}{=} \overline{a}\langle x\rangle.\mathbf{0} \tag{$*$}$$

We showed in Section 5.2 that rule $(*)$ is not sound in general — it can break the full abstraction for $\mathcal{C}$. However $(*)$ is sound in some cases, and we believe that the mentioned optimisation 1 falls in this category. This would give us a factorisation for $\mathcal{P}_1$ (or better, for its rectified $\mathcal{P}_1'$) through the HOπ encoding and the compilation $\mathcal{C}$, up-to some code-optimisation. We defer the analysis of the soundness of optimisation 1, as well as of possible other optimisations of $\mathcal{C}$, for future research.

On the contrary, the use of $(*)$ in optimisation 2 is not sound. We show this by proving that β-conversion is not valid for $\mathcal{P}_2$ — whereas it is valid in $\mathcal{C}\circ\mathcal{H}$ by Lemma 6.4.1 and the full abstraction of $\mathcal{C}$.

Lemma 6.4.3 *There are β-convertible terms M and N s.t. $\mathcal{P}_2[\![M]\!] \not\approx^{c} \mathcal{P}_2[\![N]\!]$.*

Proof: Take $M = \big(\lambda x.(\lambda y.x)\big)(\lambda z.z)$ and $N = \lambda y.(\lambda z.z)$. With a β-conversion, M reduces to N. However, $\mathcal{P}_2[\![M]\!] \not\approx^{c} \mathcal{P}_2[\![N]\!]$, as we are going to prove. We exploit the characterisation of $\approx^{c}$ in terms of weak early congruence in π-calculus (Corollary 3.3.4). Since weak early congruence is the congruence induced by weak early bisimulation ($\approx_{e}$), it is enough to show that $\mathcal{P}_2[\![M]\!] \not\approx_{e} \mathcal{P}_2[\![N]\!]$.

Let $R_1 \stackrel{def}{=} x'(w').w'(y',r').\overline{r'}\langle y'\rangle$. By repeated use of the expansion law and simple application of the structural congruence relation, we get

$$\mathcal{P}_2[\![M]\!]\langle p\rangle \quad \approx_{e} \quad (\boldsymbol{\nu}\, x, x')\big(\overline{p}\langle x\rangle.\,!\,x(w).w(y,r).\overline{r}\langle x'\rangle \mid\, !\, R_1\big) \stackrel{def}{=} P$$

On the other hand, we have

$$\mathcal{P}_2[\![N]\!]\langle p\rangle \;=\; (\boldsymbol{\nu}\, x)\overline{p}\langle x\rangle.\,!\,x(w).w(y,r).(\boldsymbol{\nu}\, x')\overline{r}\langle x'\rangle.\,!\,R_1 \stackrel{def}{=} Q$$

We have to show that $P \not\approx_{\mathrm{e}} Q$. It is the different position of the restriction $\boldsymbol{\nu}\, x'$ which causes the difference between P and Q. Consider the sequence below of transitions from P. Here R_2 stands for $x(w).w(y,r).\overline{r}\langle x'\rangle$ and the underlined action or component is the one causing the successive action.

$$\begin{array}{lll}
P & \xrightarrow{(\boldsymbol{\nu}\, x)\overline{p}\langle x\rangle} & \boldsymbol{\nu}\, x'\big(\,!\,\underline{R_2} \mid\, !\,R_1\big) \\
 & \xrightarrow{x\langle w\rangle} & \boldsymbol{\nu}\, x'\big(\,!\,R_2 \mid \underline{w(y,r)}.\overline{r}\langle x'\rangle \mid\, !\,R_1\big) \\
 & \xrightarrow{w\langle y,r\rangle} & \boldsymbol{\nu}\, x'\big(\,!\,R_2 \mid \underline{\overline{r}\langle x'\rangle} \mid\, !\,R_1\big) \\
 & \xrightarrow{(\boldsymbol{\nu}\, x')\overline{r}\langle x'\rangle} & !\,\underline{R_2} \mid\, !\,R_1 \\
 & \xrightarrow{x\langle w''\rangle} & !\,R_2 \mid \underline{w''(y,r).\overline{r}\langle x'\rangle} \mid\, !\,R_1 \\
 & \xrightarrow{w''\langle y,r\rangle} & !\,R_2 \mid \underline{\overline{r}\langle x'\rangle} \mid\, !\,R_1 \\
 & \xrightarrow{\overline{r}\langle x'\rangle} & !\,R_2 \mid\, !\,R_1
\end{array}$$

In this sequence, the last action is a *free* output. Instead, in the corresponding sequence for Q, the last action is a *bound* output. Below, R_3 stands for $x(w).w(y,r).(\boldsymbol{\nu}\, x')\overline{r}\langle x'\rangle.\,!\,R_1$.

$$\begin{array}{lll}
Q & \xrightarrow{(\boldsymbol{\nu}\, x)\overline{p}\langle x\rangle} & !\,\underline{R_3} \\
 & \xrightarrow{x\langle w\rangle} & !\,R_3 \mid \underline{w(y,r)}.(\boldsymbol{\nu}\, x')\overline{r}\langle x'\rangle.\,!\,R_1 \\
 & \xrightarrow{w\langle y,r\rangle} & !\,R_3 \mid \underline{(\boldsymbol{\nu}\, x')\overline{r}\langle x'\rangle}.\,!\,R_1 \\
 & \xrightarrow{(\boldsymbol{\nu}\, x')\overline{r}\langle x'\rangle} & !\,\underline{R_3} \mid\, !\,R_1 \\
 & \xrightarrow{x\langle w''\rangle} & !\,R_3 \mid \underline{w''(y,r)}.(\boldsymbol{\nu}\, x')\overline{r}\langle x'\rangle.\,!\,R_1 \mid\, !\,R_1 \\
 & \xrightarrow{w''\langle y,r\rangle} & !\,R_3 \mid \underline{(\boldsymbol{\nu}\, x')\overline{r}\langle x'\rangle}.\,!\,R_1 \mid\, !\,R_1 \\
 & \xrightarrow{(\boldsymbol{\nu}\, x')\overline{r}\langle x'\rangle}\equiv & !\,R_3 \mid\, !\,R_1
\end{array}$$

□

So, we have found a deficiency in $\mathcal{P}_2$. Nevertheless it is true that $\mathcal{P}_2$ yields a precise operational correspondence between λ-terms and their process encodings, and intuitively one "expects" $\mathcal{P}_2$ to be correct. We leave for future research how to best formalise such a perception.[1] An alternative question is whether there is an encoding as efficient as $\mathcal{P}_2$ but which does not have the above deficiency.

6.5 A λ-model from process terms

Having shown that there is an exact operational correspondence between lazy λ-terms and their HOπ encodings (Proposition 6.3.6), it is legitimate to ask ourselves whether the lazy encoding $\mathcal{H}$ gives rise to a λ-model, and if yes, what kind of λ-model it represents.

We chose the lazy λ-calculus for this study because of its simplicity. For the time being, we prefer to leave the corresponding analysis for call-by-value as future work, together with the question whether a simpler encoding exists. We conduct our study in the HOπ; by Theorem 6.3.7 and full abstraction of the compilation $\mathcal{C}$, all results obtained can be transported to the π-calculus encoding. From now on up to the end of the chapter, the word encoding and the symbol $\mathcal{H}$ refer to the HOπ encoding of lazy λ-calculus.

6.5.1 λ-models

There are simple syntax-free definitions of λ-model (i.e. they do not mention λ-terms). But since we already have the mapping from λ to process terms, it is more convenient a definition where we can use such a mapping explicitly. A *valuation* is a function from the set of λ-variables to the domain D of the λ-model; $[\boldsymbol{d}/x]\rho$ is the valuation which maps x to $\boldsymbol{d}$ and which behaves like ρ on the remaining elements.

Definition 6.5.1 *A* λ-model *is a triple* $< D, \cdot, \mathcal{M} >$*, where* D *is a set with at least two elements, '*$\cdot$*' is a mapping from* $D \times D$ *to* D *and* $\mathcal{M}$ *is a mapping which assigns, to each* λ*-term* M *and evaluation* ρ*, a member* $\mathcal{M}[\![M]\!]_\rho \in D$ *s.t. :*

[1] By the time the review of this thesis was finished, a positive answer to this question has been obtained by Pierce and Sangiorgi in the paper "Typing and subtyping for mobile processes", to appear in the proceedings of Eigth Annual IEEE Symposium on Logics in Computer Science (LICS'93), Montreal, Canada, June 1993.

1. $\mathcal{M}[\![x]\!]_\rho = \rho(x)$
2. $\mathcal{M}[\![MN]\!]_\rho = \mathcal{M}[\![M]\!]_\rho \cdot \mathcal{M}[\![N]\!]_\rho$
3. $\mathcal{M}[\![\lambda x.M]\!]_\rho \cdot \boldsymbol{d} = \mathcal{M}[\![M]\!]_{[\boldsymbol{d}/x]\rho}$, $\forall \boldsymbol{d} \in D$
4. $\mathcal{M}[\![M]\!]_\rho = \mathcal{M}[\![M]\!]_\sigma$ *if* $\rho(x) = \sigma(x)\ \forall x \in fv(M)$
5. $\mathcal{M}[\![\lambda x.M]\!]_\rho = \mathcal{M}[\![\lambda y.M\{y/x\}]\!]_\rho$, *if* $y \notin fv(M)$
6. *if* $\forall \boldsymbol{d} \in D$, $\mathcal{M}[\![M]\!]_{[\boldsymbol{d}/x]\rho} = \mathcal{M}[\![N]\!]_{[\boldsymbol{d}/x]\rho}$, *then* $\mathcal{M}[\![\lambda x.M]\!]_\rho = \mathcal{M}[\![\lambda x.N]\!]_\rho$.

6.5.2 The model

Our definition λ-model should respect the semantic relation adopted in HOπ. So, let us denote by $[\,A\,]_{\approx^c}$ the equivalence class of the agent A, namely

$$\{A' \ :\ A' \text{ is an HO}\pi \text{ agent and } A \approx^c A'\}$$

The elements of the domain D of the model will be the equivalence classes of the closed HOπ agents with the same sort (s) as the agents encoding λ-terms.

$$D \stackrel{def}{=} \{[\,F\,]_{\approx^c} \ :\ F \in \text{HO}\pi \text{ and } F : (s)\}$$

The definition of application on these elements follow the translation of λ-application in $\mathcal{H}$:

$$[\,G\,]_{\approx^c} \cdot [\,F\,]_{\approx^c} \stackrel{def}{=} [\,(p)\boldsymbol{\nu}\, q\,(G\langle q\rangle \mid \overline{q}\langle F, p\rangle)\,]_{\approx^c} \quad \text{for } p, q \text{ not free in } G, F$$

Note that the definition of application is consistent: by the congruence properties of $\approx^c$, the result of the application does not depend upon the representatives G and F chosen from the equivalence classes. We are left with the definition of the mapping $\mathcal{M}[\![M]\!]_\rho$. The evaluation ρ maps λ-variables to equivalence classes of $\approx^c$. Given an evaluation ρ, we denote by ρ^H a substitution from HOπ variables to HOπ agents s.t.

$$\text{for each } \lambda\text{-variable } x, \quad \rho^H(\mathcal{H}[\![x]\!]) \in \rho(x)$$

Therefore, ρ^H is the "conversion" of ρ which operates on the HOπ variable $\mathcal{H}[\![x]\!]$ and which selects a representative out of the equivalence class of $\rho(x)$. Now, the mapping $\mathcal{M}$ of the λ-model is defined using $\mathcal{H}$ as follows:

$$\mathcal{M}[\![M]\!]_\rho \stackrel{def}{=} [\,\mathcal{H}[\![M]\!]\rho^H\,]_{\approx^c}$$

Note that since $\approx^c$ is a congruence, this definition is independent from the representatives of the equivalence classes selected by ρ^H. We denote by $\mathcal{D}$ be the triple $< D, \cdot, \mathcal{M} >$ so obtained.

Theorem 6.5.2 *$\mathcal{D}$ is a λ-model.*

PROOF: We consider each of the clauses (1-6), in the order.

(1) We have:

$$\mathcal{M}[\![x]\!]_\rho \stackrel{def}{=} [\,\mathcal{H}[\![x]\!]\rho^H\,]_{\approx^c} = [\,\rho^H(\mathcal{H}[\![x]\!])\,]_{\approx^c} = \rho(x)$$

where the last equality holds because, by definition, $\rho^H(\mathcal{H}[\![x]\!])$ belongs to the equivalence class $\rho(x)$.

(2) We have:

$$\begin{array}{rcl} \mathcal{M}[\![MN]\!]_\rho & \stackrel{def}{=} & \\ [\,\mathcal{H}[\![MN]\!]\rho^H\,]_{\approx^c} & = & \text{for } p, q \text{ fresh} \\ [\,(p)\boldsymbol{\nu}\, q\left(\mathcal{H}[\![M]\!]\rho^H\langle q\rangle \mid \overline{q}\langle \mathcal{H}[\![N]\!]\rho^H, p\rangle\right)]_{\approx^c} & = & \\ [\,\mathcal{H}[\![M]\!]\rho^H\,]_{\approx^c} \cdot [\,\mathcal{H}[\![N]\!]\rho^H\,]_{\approx^c} & \stackrel{def}{=} & \mathcal{M}[\![M]\!]_\rho \cdot \mathcal{M}[\![N]\!]_\rho \end{array}$$

(3) Let $\boldsymbol{d} = [\,F\,]_{\approx^c}$. We have to show that for p and q fresh

$$\mathcal{M}[\![\lambda x.M]\!]_\rho \cdot \boldsymbol{d} = [\,(p)\boldsymbol{\nu}\, q\left(\mathcal{H}[\![\lambda x.M]\!]\rho^H\langle q\rangle \mid \overline{q}\langle F, p\rangle\right)]_{\approx^c}$$

is the same equivalence class as

$$\mathcal{M}[\![M]\!]_{[\boldsymbol{d}/x]\rho} \stackrel{def}{=} [\,\mathcal{H}[\![M]\!]([\boldsymbol{d}/x]\rho)^H\,]_{\approx^c} = [\,\mathcal{H}[\![M]\!]\{F/X\}\rho^H\,]_{\approx^c}$$

For this, we show that for all p, with $p \neq q$,

$$\boldsymbol{\nu}\, q\left(\mathcal{H}[\![\lambda x.M]\!]\rho^H\langle q\rangle \mid \overline{q}\langle F, p\rangle\right) \approx^c \left(\mathcal{H}[\![M]\!]\{F/X\}\rho^H\right)\langle p\rangle$$

By α-conversion we can assume that the bound variable X is not modified by ρ^H and then we have:

$$
\begin{array}{rcl}
\boldsymbol{\nu}\, q \left(\mathcal{H}[\![\lambda x.M]\!]\rho^H\langle q\rangle \mid \overline{q}\langle F, p\rangle\right) & = & \\
\boldsymbol{\nu}\, q \left(q(X, r).\mathcal{H}[\![M]\!]\rho^H\langle r\rangle \mid \overline{q}\langle F, p\rangle\right) & \approx^{\mathrm{c}} & \text{(expansion law)} \\
\boldsymbol{\nu}\, q \left(\mathcal{H}[\![M]\!]\{F/X\}\rho^H\langle p\rangle\right) & \equiv & (q \text{ not free in the body}) \\
\left(\mathcal{H}[\![M]\!]\{F/X\}\rho^H\right)\langle p\rangle & &
\end{array}
$$

(4) If $\rho(x) = \sigma(x)$, then $\rho^H(X) \approx^{\mathrm{c}} \sigma^H(X)$; since this holds for all x and $\approx^{\mathrm{c}}$ is a congruence,

$$\mathcal{H}[\![M]\!]\rho^H \approx^{\mathrm{c}} \mathcal{H}[\![M]\!]\sigma^H$$

From this, we get

$$\mathcal{M}[\![M]\!]_\rho \stackrel{def}{=} [\,\mathcal{H}[\![M]\!]\rho^H\,]_{\approx^{\mathrm{c}}} = [\,\mathcal{H}[\![M]\!]\sigma^H\,]_{\approx^{\mathrm{c}}} \stackrel{def}{=} \mathcal{M}[\![M]\!]_\sigma$$

(5) An α-conversion on λ-terms is reflected on the α-conversion on their HOπ encodings.

(6) By hypothesis, for all F, if $\boldsymbol{d} = [\,F\,]_{\approx^{\mathrm{c}}}$, we have:

$$\mathcal{M}[\![M]\!]_{[\boldsymbol{d}/x]\rho} = \mathcal{M}[\![N]\!]_{[\boldsymbol{d}/x]\rho}$$

Then from the definition of $\mathcal{M}$, we infer that for all F,

$$\mathcal{H}[\![M]\!]\{F/X\}\rho^H \approx^{\mathrm{c}} \mathcal{H}[\![N]\!]\{F/X\}\rho^H$$

which means that

$$\mathcal{H}[\![M]\!]\rho^H \approx^{\mathrm{c}} \mathcal{H}[\![N]\!]\rho^H$$

Since $\approx^{\mathrm{c}}$ is a congruence, we get

$$(p)p(X, q).\mathcal{H}[\![M]\!]\rho^H\langle q\rangle \approx^{\mathrm{c}} (p)p(X, q).\mathcal{H}[\![N]\!]\rho^H\langle q\rangle$$

which, by definition of $\mathcal{M}$, gives

$$\mathcal{M}[\![\lambda x.M]\!]_\rho = \mathcal{M}[\![\lambda x.N]\!]_\rho$$

$\square$

Remark 6.5.3 One could have tried to be more selective in the definition of the domain $\mathcal{D}$, and take as domain $D^* = \{[\mathcal{H}[\![M]\!]]_{\approx^c} : M \in \Lambda\}$; then $\mathcal{D}^* =< D^*, \cdot, \mathcal{M} >$ represents the *interior* of $\mathcal{D}$ (see [HS86, pag.107]). But it turns out that $\mathcal{D}^*$ is *not* a λ-model: Clause (6) in Definition 6.5.1 fails. As counterexample, take the terms L_1 and L_2 as will be defined in Section 6.6. Following the proof in [AO89] that that $L_1 \cong L_2$, one can show that for all closed N, $\mathcal{H}[\![L_1\{N/x\}]\!] \approx^c \mathcal{H}[\![L_2\{N/x\}]\!]$; this means that

$$\text{for all } \boldsymbol{d} \in D^* \quad \mathcal{M}[\![L_1]\!]_{[\boldsymbol{d}/x]\rho} = \mathcal{M}[\![L_2]\!]_{[\boldsymbol{d}/x]\rho}$$

However,

$$\mathcal{M}[\![\lambda x.L_1]\!]_\rho \not\approx^c \mathcal{M}[\![\lambda x.L_2]\!]_\rho$$

which can be derived from the fact that $\mathcal{H}[\![L_1]\!] \not\approx^c \mathcal{H}[\![L_2]\!]$ (the difference between them appears when instantiating their free variable with an agent which is not the encoding of a λ-term, as discussed in Section 6.6). Therefore $\mathcal{D}$ is a λ-model whose interior is not a λ-model. See [HL80] for two more examples. $\square$

Having proved that $\mathcal{D}$ is a λ-model, we can infer for it all properties of λ-models. In particular, we get that:

- Every provable equation of $\lambda\beta$ is valid for the encoding, up-to $\approx^c$ (where $\lambda\beta$ is the formal theory given by α, β-conversion plus the rules of inference for equivalence and congruence).

- $< D, \cdot >$ is a combinatory algebra (and hence is combinatorially complete) where the two distinguished elements $\boldsymbol{k}$ and $\boldsymbol{s}$ can be defined as $\boldsymbol{k} \stackrel{def}{=} \mathcal{H}[\![\lambda xy.x]\!]$, and $\boldsymbol{s} \stackrel{def}{=} \mathcal{H}[\![\lambda xyz.xz(yz)]\!]$.

We now turn to examining the questions of extensionality and full abstraction for $\mathcal{D}$.

6.5.3 Extensionality

Model D is *not* extensional, i.e. it is not a $\lambda\eta$ model. As counterexample, take Ω and $\lambda x.\Omega x$. Then we have $\mathcal{H}[\![\Omega]\!]\langle p\rangle \not\approx^c \mathcal{H}[\![\lambda x.\Omega x]\!]\langle p\rangle$, since $\mathcal{H}[\![\Omega]\!]\langle p\rangle \approx^c \mathbf{0}$, whereas $\mathcal{H}[\![\lambda x.\Omega x]\!]\langle p\rangle$ can perform a visible action at p.

This failure is not too surprising, since our encoding mimics the lazy λ-calculus, in which the η-rule is not valid. However, in the lazy λ-calculus a weak form of η-rule holds, namely

$$M \Downarrow \text{ implies } \lambda x.Mx \cong M, \text{ for } x \notin fv(M).$$

The same weak η-rule holds for our encoding.

Theorem 6.5.4 (weak extensionality) *$M \Downarrow$ implies $\mathcal{H}[\![\lambda x.Mx]\!] \approx^{c} \mathcal{H}[\![M]\!]$, for $x \notin fv(M)$.*

PROOF: By exploiting the characterisation of $\approx^{c}$ in terms of weak normal congruence: For any p, $\mathcal{H}[\![\lambda x.Mx]\!]\langle p\rangle$ and $\mathcal{H}[\![M]\!]\langle p\rangle$ are processes willing to perform an input at p. Using Proposition 6.3.6 and the definition of the encoding, one can show that for each input provided by the environment, $\mathcal{H}[\![\lambda x.Mx]\!]\langle p\rangle$ and $\mathcal{H}[\![M]\!]\langle p\rangle$ evolve to equivalent derivations. $\square$

Remark 6.5.5 Only as a remark, and without reporting the proofs of our claims, let us mention *λ-transitions systems (lts)*: They are introduced in Section 6.3 of [Abr87] as general operational models of the lazy λ-calculus. Then Abramsky shows that every *lts*:

- is also a λ-model
- satisfies weak extensionality.

If we had proved that our model $\mathcal{D}$ is an *lts*, then we could have inferred directly the results in Sections 6.5.2 and 6.5.3. However this is *not* the case (and neither its interior $\mathcal{D}'$ is an *lts*). The reason is that the equivalence which has to be defined on an *lts* (see [Abr87, Definition 6.3.2]) would not be compatible with $\approx^{c}$; in other words, in order to obtain an *lts* we would have to quotient the HOπ agents with a relation which does not seem to be particularly appealing in a process calculus setting. $\square$

6.6 Full abstraction

The full abstraction problem was first studied by Milner [Mil77] and Plotkin [Plo77]. It is concerned with the problem of finding a denotational interpretation for a programming language s.t. the resulting semantic equality coincides with a notion of operational indistinguishableness. Simply typed λ-calculus is the classical setting in which the problem has been developed. With the introduction of the operational equivalence resulting from applicative bisimulation, it can be neatly transferred to the untyped λ-calculus and it has motivated elegant works by Abramsky, Ong, Boudol ([AO89,Bou91]), in the setting of lazy λ-calculus.

A denotational interpretation is said to be

- *sound* if it only equates operationally equivalent terms,
- *complete* if it equates all operational equivalent terms,
- *fully abstract* if it is sound and complete.

Let us consider what happens with the encoding $\mathcal{H}$ of lazy λ-calculus into HOπ. It is *sound*, since it is true that $\mathcal{H}[\![M]\!] \approx^{c} \mathcal{H}[\![N]\!]$ implies $M \cong N$. This can be established using Proposition 6.3.6 and the characterisation of $\approx^{c}$ in terms of weak context congruence. However, $\mathcal{H}$ is not *complete*. To prove this, consider the terms

$$L_1 = x(\lambda y.(x\Xi\Omega y))\Xi \qquad\qquad L_2 = x(x\Xi\Omega)\Xi$$

where Ξ is an always convergent term (that is, for all $\widetilde{N}$, it holds that $\Xi\widetilde{N}\Downarrow$), like $(\lambda x.\lambda y.(xx))(\lambda x.\lambda y.(xx))$. Terms L_1, L_2 are used by Abramsky and Ong [AO89] to show that their canonical model for lazy λ-calculus is not fully abstract. They show that $L_1 \cong L_2$ by induction on the order of the term which instantiates x. On the other hand, they show that L_1 and L_2 can be distinguished using a new combinator, called *convergence test* and denoted by ∇, which is definable in the canonical model but is not in lazy λ-calculus. The operational behaviour of ∇ is given by

$$(\nabla 1)\ \frac{M\Downarrow}{\nabla M \Longrightarrow I} \qquad\qquad (\nabla 2)\ \frac{M \Longrightarrow M'}{\nabla M \Longrightarrow \nabla M'}$$

Using ∇, we have $L_1\{\nabla/x\} \Downarrow$, whereas $\mathrm{L}_2\{\nabla/x\} \not\Downarrow$. We also have $\mathcal{H}[\![L_1]\!] \not\approx^{c} \mathcal{H}[\![L_2]\!]$. This follows from the corresponding result for the encoding into π-calculus which Milner obtains by implementing convergence test as a π-process, and Theorem 6.3.7. In terms of our model $\mathcal{D}$ this means that L_1 and L_2 have different denotations and that $\mathcal{D}$ is not a fully abstract model for the lazy λ-calculus.

Given a denotational interpretation which is not fully abstract, there are two natural directions to achieve full abstraction.

- To cut down the existing "over-generous" semantics domain (*restrictive approach*).
- To enrich the language (*expansive approach*).

These two approaches are exemplified by the solutions to the full abstraction problem for PCF (a typed λ-calculus extended with fixed points, boolean and arithmetic features) proposed by Milner [Mil77] and Plotkin [Plo77]; in the latter, PCF is augmented with a parallel 'or' operator. We shall see that in the case of our encoding $\mathcal{H}$ both these approaches lead to interesting constructions.

6.6.1 The restrictive approach

Such approach is based on the use of barbed bisimulation. More specifically, we exploit the possibility of parametrising this equivalence on a specific class of contexts. As λ-terms are *only* used in λ-calculus contexts, so we require that their encodings be used *only* in encodings of λ-contexts. The encoding $\mathcal{H}$ is extended to λ-contexts by mapping the λ hole to the HOπ hole, i.e. $\mathcal{H}[\![[\cdot]]\!] \stackrel{def}{=} [\cdot]$. Thus, the class of contexts we are interested in is

$$\mathcal{L} = \{\mathcal{H}[\![C_\lambda[\cdot]]\!] \ \ s.t.\ \ C_\lambda[\cdot] \text{ is a closed } \lambda\text{-context}\}$$

If M and N are closed λ-terms, we set $\mathcal{H}[\![M]\!] \approx_{\mathcal{L}} \mathcal{H}[\![N]\!]$ if for every $C[\cdot] \in \mathcal{L}$ and for every p, it holds that $C[\mathcal{H}[\![M]\!]]\langle p\rangle \stackrel{\bullet}{\approx} C[\mathcal{H}[\![N]\!]]\langle p\rangle$ (i.e. they are weak barbed bisimilar). The definition of $\approx_{\mathcal{L}}$ is extended to encodings of open λ-terms as expected: If $fv(\mathcal{H}[\![M]\!], \mathcal{H}[\![N]\!]) = \{X_1, \ldots, X_n\}$, then $\mathcal{H}[\![M]\!] \approx_{\mathcal{L}} \mathcal{H}[\![N]\!]$ if for all closed λ-terms $L_1, \ldots, L_n$, it holds that

$$\mathcal{H}[\![M]\!]\{\mathcal{H}[\![L_1]\!]/X_1, \ldots, \mathcal{H}[\![L_n]\!]/X_n\} \approx_{\mathcal{L}} \mathcal{H}[\![N]\!]\{\mathcal{H}[\![L_1]\!]/X_1, \ldots, \mathcal{H}[\![L_n]\!]/X_n\}.$$

We prove that $M \cong N$ iff $\mathcal{H}[\![M]\!] \approx_{\mathcal{L}} \mathcal{H}[\![N]\!]$ via the characterisation of $\cong$ given in Theorem 6.3.4. This characterisation used the relations $\dot{\approx}$ and $\approx^{c}$, which are the specialisation to the λ-calculus of barbed bisimulation and congruence and were introduced in Definition 6.3.3.

Lemma 6.6.1 *Let $M \in \Lambda$: then for each p, $M \Downarrow$ iff $\mathcal{H}[\![M]\!]\langle p\rangle \Downarrow$.*

PROOF: Use Proposition 6.3.6. □

Lemma 6.6.2 *For all $M, N \in \Lambda$ and for each name p, it holds that $M \dot{\approx} N$ iff $\mathcal{H}[\![M]\!]\langle p\rangle \dot{\approx} \mathcal{H}[\![N]\!]\langle p\rangle$.*

PROOF: Use Lemma 6.6.1 together with the operational correspondence shown in Proposition 6.3.6. □

Proposition 6.6.3 *If $M, N \in \Lambda^o$, then $M \cong N$ iff $\mathcal{H}[\![M]\!] \approx_{\mathcal{L}} \mathcal{H}[\![N]\!]$.*

PROOF: It is enough to prove the result for closed terms. By Theorem 6.3.4, $M \cong N$ iff for each closed λ-context $C_\lambda[\cdot]$, $C_\lambda[M] \dot{\approx} C_\lambda[N]$. Similarly, by definition of $\approx_{\mathcal{L}}$, $\mathcal{H}[\![M]\!] \approx_{\mathcal{L}} \mathcal{H}[\![N]\!]$ iff for each context $C[\cdot]$ encoding of a closed λ-context $C_\lambda[\cdot]$, and for each name p, it holds that

$$C[\mathcal{H}[\![M]\!]]\langle p\rangle = \mathcal{H}[\![C_\lambda[M]]\!]\langle p\rangle \dot{\approx} \mathcal{H}[\![C_\lambda[N]]\!]\langle p\rangle = C[\mathcal{H}[\![N]\!]]\langle p\rangle.$$

Then the assertion of the proposition follows from Lemma 6.6.2. □

Lemma 6.6.4 ($\approx_{\mathcal{L}}$ is preserved by encodings of λ-contexts) *Suppose that $C[\cdot] = \mathcal{H}[\![C_\lambda[\cdot]]\!]$, for some (possibly open) λ-context $C_\lambda[\cdot]$, and that $M, N \in \Lambda^o$. Then*

$$\mathcal{H}[\![M]\!] \approx_{\mathcal{L}} \mathcal{H}[\![N]\!] \text{ implies } C[\mathcal{H}[\![M]\!]] \approx_{\mathcal{L}} C[\mathcal{H}[\![N]\!]].$$

PROOF: We have $C[\mathcal{H}[\![M]\!]] = \mathcal{H}[\![C_\lambda[M]]\!]$ and $C[\mathcal{H}[\![N]\!]] = \mathcal{H}[\![C_\lambda[N]]\!]$. Then the result of the lemma follows from the analogous result for λ-terms in Corollary 6.3.5 and from the correspondence between $\cong$ on λ-terms and $\approx_{\mathcal{L}}$ on their HOπ encodings in Proposition 6.6.3. $\square$

This result allows us to construct a fully abstract model for the lazy λ-calculus. Let $[A]_{\approx_{\mathcal{L}}}$ be the equivalence class of A modulo $\approx_{\mathcal{L}}$, '$\cdot$' and $\mathcal{M}[\![M]\!]$ as defined in Section 6.5.2 but with $[\;]_{\approx_{\mathcal{L}}}$ in place of $[\;]_{\approx^c}$; then take

$$\begin{aligned} D' &\stackrel{def}{=} \{\mathcal{M}[\![M]\!]_\rho \;:\; M \in \Lambda\} \\ \mathcal{D}' &\stackrel{def}{=} < D', \cdot, \mathcal{M} > \end{aligned}$$

Theorem 6.6.5 (full abstraction) *$\mathcal{D}'$ is a fully abstract model for the lazy λ-calculus.*

PROOF: Full abstraction follows from Proposition 6.6.3. The proof that $\mathcal{D}'$ is a λ-model is analogous to the proof that $\mathcal{D}$ is a λ model in Theorem 6.5.2. (The proof of Theorem 6.5.2 used the congruence properties of $\approx^c$; in this case, we need the congruence properties of $\approx_{\mathcal{L}}$ showed in Lemma 6.6.4. We also need the fact that $\approx^c$ implies $\approx_{\mathcal{L}}$). $\square$

Indeed, the model $\mathcal{D}'$ is also *fully expressive*: all objects of the domain of interpretation are λ-definable. These results show that if from HOπ and π-calculus we discard everything which is extraneous to the encoding of the lazy λ-calculus, in particular adopting $\approx_{\mathcal{L}}$ as semantic equivalence, then the structure that we get back is the "same thing" as the lazy λ-calculus. In our view, this was really the decisive test for the correctness of these encodings.

The cut of the domain D' with respect to the domain D in Section 6.5 is in two dimensions: The behavioural equivalence is $\approx_{\mathcal{L}}$ rather than the more discriminanting $\approx^c$; and only the interior of D is taken into account. The first restriction is necessary to get full abstraction; the second to make consistent the definition of application. It is interesting to note the relationship between the choices of the classes of HOπ agents and of the behavioural equivalence in the definitions of the domains D and D'. In the former, we took the class $\mathcal{A}$ of *all* admissible agents, and then in the behavioural equivalence we had to use quantification over the class $\mathcal{C}nt$ of *all* admissible contexts (definition of $\approx^{c}$); in the latter we restrained ourselves to the *"interior"* of $\mathcal{A}$, and then in the behavioural equivalence we also had to restrain ourselves to the *"interior"* of $\mathcal{C}nt$ (definition of $\approx_{\mathcal{L}}$).

6.6.2 The expansive approach

In this section we study the equivalence induced on λ-terms by the encoding. We call it λ-*observational equivalence.*

Definition 6.6.6 (λ-observational equivalent) *We say that two Λ terms M and N are* λ-observational equivalent *if* $\mathcal{H}[\![M]\!] \approx^{c} \mathcal{H}[\![N]\!]$.

The main result will be a *direct* characterisation of λ-observational equivalence (i.e. a characterisation not mentioning the encoding). From the characterisation of $\approx$ (weak barbed equivalence) and $\approx^{c}$ (weak barbed congruence) in terms of, respectively, weak normal bisimulation and weak normal congruence (Corollaries 4.7.8 and 4.7.9), it follows that $\approx^{c}$ means "$\approx$ under all substitutions of names". Since the HOπ encoding of a λ-term does not have names free in it, the relations $\approx^{c}$ and $\approx$ coincide on such terms. In view of this fact, in this section we shall use $\approx$ to study λ-observational equivalence. We shall often exploit the characterisation of $\approx$ in terms of weak context bisimulation and weak normal bisimulation in the proofs.

λ-observational equivalence

To study λ-observational equivalence means to understand the effect on λ-terms of the use of "richer" contexts, in which also concurrent features may be present. There are also motivations from the theory itself of lazy λ-calculus.

As originally defined, applicative bisimulation carries some problems. It is based on the notion of termination, but you cannot "speak" in the pure λ-calculus about termination. For instance, you cannot define the convergence test [AO89]. Such an operator can be used to show that Abramsky's canonical model for lazy λ-calculus and the model $\mathcal{D}$ of Section 6.5.2 are not fully abstract. Now, what does this mean? Maybe Abramsky's canonical domain and the encoding $\mathcal{H}$ are not good enough for the lazy λ-calculus; but maybe the problem is just that the pure λ-calculus is too weak versus the predicate of convergence. Moreover, since applicative bisimulation is based on equivalence notions originally developed for frameworks of reactive and concurrent systems, one might consider appealing the introduction of parallel operators.

Various enrichments of the lazy λ-calculus with operators not λ-definable have already appeared in the literature. However either the operators themselves — as in the case of convergence test and parallel convergence in [AO89,Ong88] — or the semantics chosen — as for the non-deterministic choice and parallel operator in [Bou90] and [Bou91] — are rather ad hoc, aimed to achieve full abstraction for some canonical domain. Or at least, from a programming language point of view, they do not seem to be justified by the common practice. Moreover it remains unclear whether the equivalences induced are insensitive to the adoption of more operators.

λ-observational equivalence seems a *robust* equivalence. Firstly, it enjoys simple and nice operational and denotational characterisations. Secondly, it coincides with the equivalence obtained when λ-calculus is augmented with the whole class of *well-formed operators*, intuitively operators whose behaviour depends only on the semantics – not on the syntax – of its operands. Put in other words, the encoding into π-calculus/HOπ induces maximal observational discrimination on λ-terms. Further, the adoption of certain simple non-deterministic operators is sufficient and necessary to induce such a discrimination. These results are showed in [San92].

Direct characterisation

To derive a direct characterisation for λ-observational equivalence we have to enrich the λ-calculus with constants. The standard way to treat a constant is to introduce it together with some rules describing its operational behaviour. We call such symbols *operators*; an example is convergence test. When only operators are used, λ-abstraction remains the only sensible normal form for closed terms.

We keep the word *constant* to denote instead symbols which are added to the language without specifying any operational rule. Such a use of the constants can be found in the well-known technique of the *top-down specification and analysis*, where a system is developed through a series of refinement steps each representing a different level of abstraction; a lower level implements some details which at a higher level have been abstracted. A constant c is then an high-level primitive standing for some lower-level procedure K_c; at this stage we might want to explicitly abstract from the behaviour of K_c to facilitate the reasoning, or it might just be that we cannot make assumptions on the behaviour of K_c (for instance, we might be interested in refinments of c with different K_c's). Now, $c\widetilde{M}$ becomes a sensible normal form too. Operationally we really can see it as the *output* of the tuple $\widetilde{M}$ along the channel c and towards K_c. The definition of applicative bisimulation has been generalised according to such interpretation to the case where also constants are admitted.

Let C be a set containing an infinite, countable number of constants. The class of Λ^o_C terms, i.e. λ-terms enriched with constants in C, is defined from the grammar of Λ^o by adding constants among the constructs:

$$M = c \quad | \quad x \quad | \quad \lambda x.M \quad | \quad M_1 M_2, \qquad \text{where } c \in C.$$

When generalising applicative bisimulation to terms in Λ_C, the main question is which condition should be imposed on the equality between the terms $c\widetilde{M}$ and $c\widetilde{N}$. According to our interpretation of the constants given above, the most natural thing to do seems to require that the *ordered* sequence of arguments represented by $\widetilde{M}$ and $\widetilde{N}$ be equivalent; hence the clause (2) in the following definition.

Definition 6.6.7 *A relation $\mathcal{R} \subseteq \Lambda_C \times \Lambda_C$ is a $\cong_C$-simulation, if $(M, N) \in \mathcal{R}$ implies:*

1. *whenever $M \Longrightarrow \lambda x.M'$ then N' exists s.t. $N \Longrightarrow \lambda x.N'$ and for all $L \in \Lambda_C$ it holds that $(M'\{L/x\}, N'\{L/x\}) \in \mathcal{R}$,*
2. *whenever $M \Longrightarrow cM_1...M_n$, for some $n \geq 0$ and $c \in C$, then $N_1, \ldots N_n$ exist s.t. $N \Longrightarrow cN_1 \ldots N_n$ and $(M_i, N_i) \in \mathcal{R}$, $1 \leq i \leq n$.*

A relation $\mathcal{R}$ is an $\cong_C$-bisimulation if $\mathcal{R}$ and $\mathcal{R}^{-1}$ are $\cong_C$-simulations. M and N are applicative bisimilar over Λ_C*, written $M \cong_C N$, if $(M,N) \in \mathcal{R}$, for some $\cong_C$-bisimulation $\mathcal{R}$.* □

Remark 6.6.8 On pure λ-calculus we could have obtained the same equivalence as $\cong_C$ if in Definition 6.6.7 we had used variables instead of constants and accepted to work on open terms [San92]. Our choice is justified by essentially two reasons. First, variables and constants play distinct rôles in Definition 6.6.7 and therefore we think it is correct to keep them separated. Secondly, the use of constants is convenient to get easier and more intuitive proofs of the theorems which follow. The problem using variables is that open λ-terms are encoded into open HOπ terms, which have to be closed to perform transitions: This would make rather obscure the operational correspondence between λ and HOπ terms. □

EXTENSION OF THE ENCODING $\mathcal{H}$ TO Λ_C TERMS

We extend the encoding $\mathcal{H}$ to Λ_C terms and prove that $\cong_C$ is exactly the equivalence induced. We have to say what the encoding of a constant is. We use each constant $c \in C$ also as an HOπ name, whose sort is supposed to be different from those of the other names appearing in the encoding and we decree:

$$\mathcal{H}[\![c]\!] \stackrel{def}{=} Tr_c \quad \text{where } Tr_c \stackrel{def}{=} (p)\overline{c}\langle p\rangle.$$

Agents like Tr_c were introduced in Section 4.3.1 and called triggers. The results in Chapter 4 show the discriminating power given by triggers. It turns out that this corresponds exactly to the discriminating power that constants provide on λ-terms.

We shall adopt the following abbreviations, already used in various occasions in the previous chapters; here F represents a first-order abstraction:

$$\begin{aligned} \tau^n &= \underbrace{\tau \ldots \tau}_{n} \\ \tau^n \star F &= (p)\tau^n.F\langle p\rangle \end{aligned}$$

To prove that $M \cong_C N$ iff $\mathcal{H}[\![M]\!] \approx \mathcal{H}[\![N]\!]$, the implication form left to right is the hardest. To work it out, it is technically convenient to work with an encoding

$\mathcal{H}'$ which expands $\mathcal{H}$ by adding some τ-actions. The number of these extra τ-actions is not fixed, hence $\mathcal{H}'[\![M]\!]$ defines a *set* of agents rather than a *single* agent. We require that, however, at least one τ-action is added in the output action of the application rule; this τ-action is underlined for clearness in the definition of $\mathcal{H}'$ below. The set (of first-order abstractions) $\mathcal{H}'[\![M_1]\!]$ is defined inductively bottom-up on the structure of M_1; for every $F_M \in \mathcal{H}'[\![M]\!]$, $F_N \in \mathcal{H}'[\![N]\!]$ and $n \geq 0$ we have:

$$\begin{array}{rcl}
\mathcal{H}'[\![\lambda x.M]\!] & \ni & (p)\tau^n.p(X,q).F_M\langle q\rangle \\
\mathcal{H}'[\![x]\!] & \ni & \tau^n \star X \\
\mathcal{H}'[\![MN]\!] & \ni & (p)\tau^n.\boldsymbol{\nu}\, v\left(F_M\langle v\rangle \mid \overline{v}\left\langle \underline{\tau} \star F_N, p\right\rangle.\mathbf{0}\right) \\
\mathcal{H}'[\![c]\!] & \ni & \tau^n \star Tr_c
\end{array}$$

We denote by $\mathcal{H}'[\![M]\!]\langle p\rangle$ the set of processes

$$\{F\langle p\rangle \ : \ F \in \mathcal{H}'[\![M]\!]\}$$

We write $P \equiv\in \mathcal{H}'[\![M]\!]\langle p\rangle$ if $P \equiv P' \in \mathcal{H}'[\![M]\!]\langle p\rangle$, for some P'. Occasionally, we use these same notations, "$\in$" and "$\equiv\in$", for $\mathcal{H}$; therefore, for instance, we might write $P \in \mathcal{H}[\![M]\!]\langle p\rangle$ to mean that $P = \mathcal{H}[\![M]\!]\langle p\rangle$.

By Theorem 4.4.1 and Lemma 4.4.2, the τ-actions added in $\mathcal{H}'$ do not affect weak context congruence. Since by Corollary 4.7.8 weak context congruence coincides with weak barbed congruence and the latter implies weak barbed equivalence, we have:

Lemma 6.6.9 *If $M, N \in \Lambda_C$ and $F_M \in \mathcal{H}'[\![M]\!]$, $F_N \in \mathcal{H}'[\![N]\!]$, then*

$$\mathcal{H}[\![M]\!] \approx \mathcal{H}[\![N]\!] \quad \textit{iff} \quad F_M \approx F_N. \qquad \square$$

Now we present some properties of $\mathcal{H}$ (as previously extended to Λ_C) and of $\mathcal{H}'$ needed later in the proofs of the main theorems (6.6.17 and 6.6.19). We start with the operational correspondence between λ and process terms.

Lemma 6.6.10 (operational correspondence) *Let $M \in \Lambda_C$ and $\mathcal{K} \in \{\mathcal{H}, \mathcal{H}'\}$:*

1. *Suppose that $\mathcal{K}[\![M]\!]\langle p\rangle \ni P \xrightarrow{\mu}$, for $\mu \neq \tau$ and $sbj(\mu) \neq p$.*

   Then, for some $c, M_1, ..., M_n$, it must be $M = cM_1...M_n$ and $\mu \equiv (\boldsymbol{\nu}\, q)\overline{c}\langle q\rangle$

2. *Suppose that* $\mathcal{K}[\![M]\!]\langle p\rangle \ni P \xrightarrow{\mu}$, *for* $\mu \neq \tau$ *and* $sbj(\mu) = p$.

   Then for some M_1, *it must be* $M = \lambda x.M_1$ *and* μ *is an input action.*

3. *Suppose* $M \longrightarrow M'$. *Then for any* $P \in \mathcal{K}[\![M]\!]\langle p\rangle$, *there is* Q *s.t.* $P \Longrightarrow Q \equiv\in \mathcal{K}[\![M']\!]$.

4. *Suppose* $\mathcal{K}[\![M]\!]\langle p\rangle \ni P \longrightarrow Q$. *Then it holds that* $Q \in \mathcal{K}[\![M]\!]\langle p\rangle$ *or there is* M' *s.t.* $M \longrightarrow M'$ *and* $Q \equiv\in \mathcal{K}[\![M']\!]\langle p\rangle$.

PROOF: All cases can be proved by simple inductions on the structure of M, and are similar to the proof of Proposition 6.3.6. For $\mathcal{H}'$ one also needs the fact that if $F_1 \in \mathcal{H}'[\![M_1]\!]$ and $F_2 \in \mathcal{H}'[\![M_2]\!]$, for $M_1, M_2 \in \Lambda^o_C$, then $F_1\{F_2/X\} \in \mathcal{H}'[\![M_1\{M_2/x\}]\!]$; this property follows immediately from the compositional definition of $\mathcal{H}'$. $\square$

In the previous lemma, for $\mathcal{K} = \mathcal{H}$ the correspondence between λ-reduction and HOπ-reductions in cases (3) and (4) can be made one-to-one as in Proposition 6.3.6. However, for our purposes, the weaker version presented will suffice.

A process $P \in \mathcal{K}[\![M]\!]\langle p\rangle$ has only one possible reduction, that is, if $P \longrightarrow Q$ and $P \longrightarrow Q'$, then $Q = Q'$: This means that P can be equated to $\tau.Q$ and therefore, reasoning similarly as for Lemma 6.6.9, we have:

Corollary 6.6.11 *Let* $M \in \Lambda_C$ *and* $\mathcal{K} \in \{\mathcal{H}, \mathcal{H}'\}$. *If* $\mathcal{K}[\![M]\!]\langle p\rangle \ni P \longrightarrow Q$, *then* $P \approx Q$. $\square$

Now we introduce the agent

$$Out_n \stackrel{def}{=}$$
$$(q,p,X_1,...,X_n)(\boldsymbol{\nu}\, r_1,...,r_{n-1})\Big(\overline{q}\langle X_1,r_1\rangle|\overline{r}_1\langle X_2,r_2\rangle|...|\overline{r}_{n-2}\langle X_{n-1},r_{n-1}\rangle|\overline{r}_{n-1}\langle X_n,p\rangle\Big)$$

If s is the sort of names p,q,r, then $Out_n : (s,s,\underbrace{(s),...,(s)}_{n})$.

$Out_n\langle q,p,F_1,\ldots,F_n\rangle$ is a process which outputs $F_1,\ldots,F_n$ in the order, each along a different name. Such a name had been exported, as a private name, in the previous output of the process; exceptions are the first output, which takes place at q, and the last one, where the name emitted is p. We use Out_n to represent in a compact and manageable way encodings of terms with a constant in head position. In fact it holds that:

Lemma 6.6.12

1. *$\mathcal{H}[\![cM_1...M_n]\!]\langle p\rangle \equiv \boldsymbol{\nu}\, q\left(\overline{c}\langle q\rangle \mid Out_n\left\langle q, p, \mathcal{H}[\![M_1]\!], ..., \mathcal{H}[\![M_n]\!]\right\rangle\right)$*
2. *If $P \in \mathcal{H}'[\![cM_1...M_n]\!]\langle p\rangle$, then for some n, and m,*

$$P \equiv \tau^n \boldsymbol{\nu}\, q\left(\tau^m.\overline{c}\langle q\rangle \mid Out_n\left\langle q, p, \tau\star F_1, \ldots, \tau\star F_2\right\rangle\right)$$

where $F_i \in \mathcal{H}'[\![M_i]\!]$ for each $1 \leq i \leq n$.

PROOF: Simple inductions on n. $\square$

We recall that $\simeq_{\mathrm{Ct}}$ and $\simeq_{\mathrm{Nr}}$ represent strong context and normal bisimulation, introduced in Chapter 4, and that $\approxeq_{\mathrm{Ct}}$, $\approxeq_{\mathrm{Nr}}$ are their weak versions (by Corollary 4.7.8 all coincide with $\approx$). Lemmas 6.6.13 and 6.6.15 below show properties of the agent Out_n which will be needed in Proposition 6.6.16 and Theorem 6.6.19, respectively.

Lemma 6.6.13 *Let $\widetilde{m} = m_1, ..., m_n$ and suppose $\widetilde{m} \cap \left(fn(F_1, ..., F_n) \cup \{q, p\}\right) = \emptyset$. Then $Q_1 \simeq_{\mathrm{Nr}} Q_2$, where*

$$\begin{array}{lcl} Q_1 & \stackrel{def}{=} & Out_n\left\langle q, p, \tau\star F_1, ..., \tau\star F_n\right\rangle \\ Q_2 & \stackrel{def}{=} & \boldsymbol{\nu}\,\widetilde{m}\left(Out_n\left\langle q, p, Tr_{m_1}, \ldots, Tr_{m_n}\right\rangle \mid \prod_{i=1}^{n} !\, m_i(r).F_i\langle r\rangle\right) \end{array}$$

PROOF: Use Lemma 4.4.6 and the inclusion $\simeq_{\mathrm{Ct}} \subset \simeq_{\mathrm{Nr}}$ (the requirements in the definition of $\simeq_{\mathrm{Nr}}$ are a subset of those in the definition of $\simeq_{\mathrm{Ct}}$). $\square$

In the following lemma, the assertion (2) also holds when F and G are higher-order abstractions. We have not reported it because we shall not need it.

Lemma 6.6.14

1. *Let $\mathcal{N}_{Q_i} = \{a \,:\, Q_i \Downarrow_a\}$, $i = 1, 2$, and suppose $fn(P_1, P_2) \cap \left(\mathcal{N}_{Q_1} \cup \mathcal{N}_{Q_2}\right) = \emptyset$. Then $P_1 \mid Q_1 \approxeq_{\mathrm{Nr}} P_2 \mid Q_2$ implies $P_1 \approxeq_{\mathrm{Nr}} P_2$.*

2. Let $m \notin fn(F, G)$. Then $!\, m(x).F\langle x\rangle \cong_{\mathrm{Nr}} !\, m(x).G\langle x\rangle$ implies $F \cong_{\mathrm{Nr}} G$.

PROOF: We first prove (1). Take

$$\mathcal{R} = \{(P_1, P_2) \ : \ P_1 \mid Q_1 \cong_{\mathrm{Nr}} P_2 \mid Q_2 \text{ for some } Q_1, Q_2 \text{ with } fn(P_1, P_2) \cap \left(\mathcal{N}_{Q_1} \cup \mathcal{N}_{Q_2}\right) = \emptyset \ \}$$

Since $fn(P_1, P_2) \cap (\mathcal{N}_{Q_1} \cup \mathcal{N}_{Q_1}) = \emptyset$, we do not have to "test" P_1 and P_2 with names in $\mathcal{N}_{Q_1} \cup \mathcal{N}_{Q_2}$ and therefore the side condition in the definition of $\mathcal{R}$ can be preserved; moreover no interaction between P_i and Q_i is possible, $i = 1, 2$. Then the proof that $\mathcal{R}$ is a $\cong_{\mathrm{Nr}}$-bisimulation becomes straightforward.

Now the assertion (2) of the lemma. We have to show that if y ranges over the free names of F and G plus a fresh name, it holds that $F\langle y\rangle \cong_{\mathrm{Nr}} G\langle y\rangle$. Since m is not free in F and G, we can assume, without loss of generality, that y is different from m. Since $!\, m(x).F\langle x\rangle \cong_{\mathrm{Nr}} !\, m(x).G\langle x\rangle$ and $!\, m(x).F\langle x\rangle \xrightarrow{m\langle y\rangle} F\langle y\rangle \mid !\, m(x).F\langle x\rangle$, we have, for some P

$$!\, m(x).G\langle x\rangle \xrightarrow{m\langle y\rangle} P \cong_{\mathrm{Nr}} F\langle y\rangle \mid !\, m(x).F\langle x\rangle \tag{6.2}$$

Since m does not occur in $G\langle y\rangle$, no interaction between $G\langle y\rangle$ and $!\, m(x).G\langle x\rangle$ may have occurred; therefore P is of the form $P_G \mid !\, m(x).G\langle x\rangle$, for some P_G s.t.

$$G\langle y\rangle \Longrightarrow P_G. \tag{6.3}$$

Thus (6.2) can be written as $P_G \mid !\, m(x).G\langle x\rangle \cong_{\mathrm{Nr}} F\langle y\rangle \mid !\, m(x).F\langle x\rangle$. From this, we get

$$P_G \cong_{\mathrm{Nr}} F\langle y\rangle \tag{6.4}$$

using the assertion (1) of the lemma, since $\mathcal{N}_{!\, m(x).G\langle x\rangle} = \mathcal{N}_{!\, m(x).F\langle x\rangle} = \{m\}$ and m is not free in $F\langle y\rangle$ and P_G. In a symmetric way (just exchange F and G), we can derive, for some P_F s.t. $F\langle y\rangle \Longrightarrow P_F$,

$$P_F \cong_{\mathrm{Nr}} G\langle y\rangle \tag{6.5}$$

Now, we exploit (6.4) and (6.5) to show that $F\langle y\rangle \cong_{\mathrm{Nr}} G\langle y\rangle$: For this we prove that

$$\mathcal{R} = \{(F\langle y\rangle, G\langle y\rangle)\} \cup \cong_{\mathrm{Nr}}$$

is a $\approxeq_{\mathrm{Nr}}$-bisimulation. Suppose $F\langle y\rangle \xrightarrow{\mu} Q_F$ and μ is an input action (the cases in which μ is an output or a silent action are similar). We show how $G\langle y\rangle$ can match this move: Since $F\langle y\rangle \approxeq_{\mathrm{Nr}} P_G$, we have $P_G \stackrel{\mu}{\Longrightarrow} Q_G \approxeq_{\mathrm{Nr}} Q_F$; therefore, by (6.3), $G\langle y\rangle \Longrightarrow P_G \stackrel{\mu}{\Longrightarrow} Q_G \approxeq_{\mathrm{Nr}} Q_F$, which closes the bisimulation. The case in which $G\langle y\rangle$ moves first is analogous. $\square$

Lemma 6.6.15 *It holds that*

$$Out_n\Big\langle q, p, F_1, \ldots, F_n\Big\rangle \approxeq_{\mathrm{Ct}} Out_n\Big\langle q, p, G_1, \ldots, G_n\Big\rangle \text{ iff for all } 1 \le i \le n,\ F_i \approx G_i.$$

PROOF: The implication from right to left can be inferred from the congruence properties of $\approxeq_{\mathrm{Ct}}$ (Theorem 4.4.1). Thus, we only have to consider the implication from left to right. We exploit the characterisation of $\approxeq_{\mathrm{Ct}}$ in terms of $\approxeq_{\mathrm{Nr}}$ (Corollary 4.7.8) and we proceed by induction on n. We only consider the inductive case. Let

$$P \stackrel{def}{=} Out_n\Big\langle q, p, F_1, \ldots, F_n\Big\rangle \qquad\qquad Q \stackrel{def}{=} Out_n\Big\langle q, p, G_1, \ldots, G_n\Big\rangle$$

The only transitions they can perform are

$$\begin{array}{llll} P & \xrightarrow{(\boldsymbol{\nu} r_1)\overline{q}\langle F_1, r_1\rangle} & Out_{n-1}\Big\langle r_1, p, F_2, \ldots, F_n\Big\rangle & \stackrel{def}{=} \quad P_1 \\ Q & \xrightarrow{(\boldsymbol{\nu} r_1)\overline{q}\langle G_1, r_1\rangle} & Out_{n-1}\Big\langle r_1, p, G_2, \ldots, G_n\Big\rangle & \stackrel{def}{=} \quad Q_1 \end{array}$$

Then, by definition of $\approxeq_{\mathrm{Nr}}$ (see also the discussion at the end of Section 4.7.3), $P \approxeq_{\mathrm{Nr}} Q$ implies

$$!\, m(x).F_1\langle x\rangle \mid P_1 \approxeq_{\mathrm{Nr}} \,!\, m(x).G_1\langle x\rangle \mid Q_1$$

where m is a fresh name. Since $\{a \;:\; P_1 \Downarrow_a \;\; or \;\; Q_1 \Downarrow_a\} = \{r_1\}$ with $r_1 \notin fn(F_1, G_1, \{m\})$ and $m \notin fn(P_1, Q_1)$, using Lemma 6.6.14(1) twice we infer

$$!\, m(x).F_1\langle x\rangle \approxeq_{\mathrm{Nr}} !\, m(x).G_1\langle x\rangle \quad \text{and} \quad P_1 \approxeq_{\mathrm{Nr}} Q_1$$

Now from the former we get $F_1 \approxeq_{\mathrm{Nr}} G_1$ using Lemma 6.6.14(2); from the latter we get $F_i \approxeq_{\mathrm{Nr}} G_i$, $2 \le i \le n$ using the inductive assumption. $\square$

Proposition 6.6.16 *For every $M, N \in \Lambda_C$, and $F_M \in \mathcal{H}'[\![M]\!]$, $F_n \in \mathcal{H}'[\![N]\!]$, it holds that $M \cong_C N$ implies $F_M \approx F_N$.*

PROOF: We shall make some abuse of language, and use the same symbol $\mathcal{H}'[\![M]\!]\langle p\rangle$ to denote both the set of processes $\mathcal{H}'[\![M]\!]\langle p\rangle$ and a generic process of this set. Thus, for instance, $P = Q \mid \mathcal{H}'[\![M]\!]\langle p\rangle$ means that $P = Q \mid P_M$, for some $P_M \in \mathcal{H}'[\![M]\!]\langle p\rangle$.

Again, we use the characterisation of $\approx$ in terms of $\approxeq_{\mathrm{Nr}}$. Let $\mathcal{R}$ be the set of all the pairs (P, Q) s.t.

$$P = R_i \mid R_z \mid R_v \mid \prod_{j=1}^{n_j} !\, m_j(q).\mathcal{H}'[\![M_j]\!]\langle q\rangle \mid \prod_{r=1}^{n_r} \mathcal{H}'[\![M_r]\!]\langle q_r\rangle$$

$$Q = R_i \mid R_z \mid R_v \mid \prod_{j=1}^{n_j} !\, m_j(q).\mathcal{H}'[\![N_j]\!]\langle q\rangle \mid \prod_{r=1}^{n_r} \mathcal{H}'[\![N_r]\!]\langle q_r\rangle$$

where

$$R_i \stackrel{def}{=} \textstyle\prod_{i=1}^{n_i} \overline{q_i}\langle Tr_{c_i}, p_i\rangle, \quad R_z \stackrel{def}{=} \prod_{z=1}^{n_z} !\, m_z(q).Tr_{c_z}\langle q\rangle, \quad R_v \stackrel{def}{=} \prod_{v=1}^{n_v} \overline{c_v}\langle q_v\rangle$$

and where

- all free names and numerical variables appearing in the definition of P, Q are existentially quantified,
- for all j, z, it holds that $M_j \cong_C N_j$ and $M_z \cong_C N_z$,
- names m_z, m_j are different each other and different from all other names occurring free in P, Q.

We show that $\mathcal{R}$ is a $\approxeq_{\mathrm{Nr}}$-bisimulation up-to $\simeq_{\mathrm{Nr}}$ and up-to restriction. This would prove the theorem, because if $M \cong_C N$, then we have, for any p,

$$\mathcal{H}'[\![M]\!]\langle p\rangle \equiv \quad \mathcal{R} \quad \equiv \mathcal{H}'[\![N]\!]\langle p\rangle$$

and hence $\mathcal{H}'[\![M]\!] \approxeq_{\mathrm{Nr}} \mathcal{H}'[\![N]\!]$. Let (P, Q) be as in the definition of $\mathcal{R}$. We have to consider the possible moves by P and show how Q can match each of them. We proceed by a case analysis on the component(s) of P which have caused the action.

Case (a) The action has been caused by R_i.

This means that for some i, $P \overset{\overline{q_i}\langle Tr_{c_i}, p_i\rangle}{\longrightarrow} P'$. Then, using the same R_i, also $Q \overset{\overline{q_i}\langle Tr_{c_i}, p_i\rangle}{\longrightarrow} Q'$ and if m is a fresh name

$$!\, m(q).Tr_{c_i}\langle q\rangle \mid P' \equiv \ \mathcal{R} \ \equiv !\, m(q).Tr_{c_i}\langle q\rangle \mid Q'$$

as required by definition of $\approxeq_{\mathrm{Nr}}$ (see also the discussion in Section 4.7.3).

Case (b) The action has been caused by R_z or R_v.

Easy.

Case (c) The action has been caused by $\prod_{j=1}^{n_j} !\, m_j(a).\mathcal{H}'[\![M_j]\!]\langle a\rangle$.

Easy.

Case (d) The action has been caused by $\prod_{r=1}^{n_r} \mathcal{H}'[\![M_r]\!]\langle q_r\rangle$.

It is crucial here to use the operational correspondence of Lemma 6.6.10. One thing which follows from it is that processes encoding λ-terms cannot interact each other. Therefore, if $\prod_{r=1}^{n_r} \mathcal{H}'[\![M_r]\!]\langle q_r\rangle$ produces an action, then the action must have been originated by some specific $\mathcal{H}'[\![M_r]\!]\langle q_r\rangle$. We shall only show how $\mathcal{H}'[\![N_r]\!]\langle q_r\rangle$ can match this move. It will be easy to see from here how to infer the action of Q which matches the one by P, and we omit it.

Suppose $\mathcal{H}'[\![M_r]\!]\langle q_r\rangle$ can perform an input action. Then, M_r has to be an abstraction, say $M_r = \lambda x.M_r'$. By definition of $\approxeq_{\mathrm{Nr}}$, if c is a fresh constant, we have to check that for every q, $\mathcal{H}'[\![N_r]\!]\langle q_r\rangle$ can match the following action:

$$\mathcal{H}'[\![\lambda x.M_r']\!]\langle q_r\rangle \overset{q_r\langle Tr_c, q\rangle}{\longrightarrow} \mathcal{H}'[\![M_r']\!]\langle q\rangle\{Tr_c/X\} = \mathcal{H}'[\![M_r'\{c/x\}]\!]\langle q\rangle$$

Since $M_r \cong_C N_r$, for some N_r',

$$N_r \Longrightarrow \lambda x.N_r' \quad \text{with} \quad M_r'\{c/x\} \cong_C N_r'\{c/x\}.$$

Then, by Lemma 6.6.10(3), $\mathcal{H}'[\![N_r]\!]\langle q_r\rangle \Longrightarrow\equiv \mathcal{H}'[\![\lambda x.N_r']\!]\langle q_r\rangle$ and therefore, by definition of $\mathcal{H}'$,

$$\mathcal{H}'[\![\lambda x.N'_r]\!]\langle q_r\rangle \stackrel{q_r\langle Tr_c,q\rangle}{\Longrightarrow} \mathcal{H}'[\![N'_r\{c/x\}]\!]\langle q\rangle$$

which matches the action by $\mathcal{H}'[\![M_r]\!]\langle q_r\rangle$. Suppose now that $\mathcal{H}'[\![M_r]\!]\langle q_r\rangle$ can perform a silent action; by Lemma 6.6.10(4), such action must be of the form

$$\mathcal{H}'[\![M_r]\!]\langle q_r\rangle \longrightarrow\equiv \mathcal{H}'[\![M'_r]\!]\langle q_r\rangle, \text{ for some } M'_r \text{ s.t. } M'_r = M_r \text{ or } M_r \longrightarrow M'_r.$$

Since by Lemma 6.3.2 $M'_r \cong M_r \cong N_r$, the process $\mathcal{H}'[\![N_r]\!]\langle q_r\rangle$ does not need to move.

Finally, suppose that $\mathcal{H}'[\![M_r]\!]\langle q_r\rangle \stackrel{\mu}{\longrightarrow} P_1$ and μ is an output action. Then, using Lemma 6.6.10(1) and 6.6.12(2), we get that

$$\begin{array}{l} \mu = (\boldsymbol{\nu}\, q)\overline{c}\langle q\rangle, \qquad M_r = cM_1\ldots M_n \qquad \text{and} \\ P_1 \equiv Out_n\Big\langle q, q_r, \tau\star\mathcal{H}'[\![M_1]\!], ..., \tau\star\mathcal{H}'[\![M_n]\!]\Big\rangle \end{array}$$

Since $M_r \cong_C N_r$, there exist $N_1, ..., N_n$ s.t. $N \Longrightarrow cN_1...N_n$ with $M_i \cong_C N_i$, for $1 \le i \le n$. By Lemma 6.6.10(3) and again Lemma 6.6.12(2), also

$$\begin{array}{l} \mathcal{H}'[\![N_r]\!]\langle p\rangle \Longrightarrow\equiv \mathcal{H}'[\![cN_1...N_n]\!]\langle q_r\rangle \stackrel{(\boldsymbol{\nu}\, q)\overline{c}\langle q\rangle}{\Longrightarrow}\equiv \\ Out_n\Big\langle q, q_r, \tau\star\mathcal{H}'[\![N_1]\!], ..., \tau\star\mathcal{H}'[\![N_n]\!]\Big\rangle \stackrel{def}{=} Q_1 \end{array}$$

This is the point where we need Lemma 6.6.13. Using it we infer

$$\begin{array}{l} P_1 \simeq_{\mathrm{Nr}} \\ \;(\boldsymbol{\nu}\, m_1, ..., m_n)\Big(\prod_{i=1}^n !\, m_i(r).\mathcal{H}'[\![M_i]\!]\langle r\rangle \mid Out_n\Big\langle q, q_r, Tr_{m_1}, ..., Tr_{m_n}\Big\rangle\Big) \stackrel{def}{=} P_2 \end{array}$$

$$\begin{array}{l} Q_1 \simeq_{\mathrm{Nr}} \\ \;(\boldsymbol{\nu}\, m_1, ..., m_n)\Big(\prod_{i=1}^n !\, m_i(r).\mathcal{H}'[\![N_i]\!]\langle r\rangle \mid Out_n\Big\langle q, q_r, Tr_{m_1}, ..., Tr_{m_n}\Big\rangle\Big) \stackrel{def}{=} Q_2 \end{array}$$

Since $\mathcal{R}$ is a bisimulation up-to restriction, from P_2 and Q_2 we can discharge the outermost restrictions $m_1, \ldots, m_n$, as well as those inside the process $Out_n\Big\langle q, q_r, Tr_{m_1}, ..., Tr_{m_n}\Big\rangle$ (to do this, the restrictions have first to be pushed at the outermost level using $\equiv$). At the end, we are left