

A CPS Encoding of Name-Passing in Higher-Order Mobile Embedded Resources

Mikkel Bundgaard* Thomas Hildebrandt Jens Chr. Godskesen

*Department of Theoretical Computer Science
IT University of Copenhagen
Copenhagen, Denmark*

Abstract

We present an encoding of the synchronous π -calculus in the calculus of Higher-Order Mobile Embedded Resources (Homer), a pure higher-order calculus with mobile processes in nested locations, defined as a simple, conservative extension of the core process-passing subset of Thomsen's Plain CHOCS. We prove that our encoding is fully abstract with respect to barbed bisimulation and sound with respect to barbed congruence. Our encoding demonstrates that higher-order process-passing together with mobile resources in, possibly local, named locations are sufficient to express π -calculus name-passing. The encoding uses a novel continuation passing style to facilitate the encoding of synchronous communication.

Key words: π -calculus; name-passing encoding; process-passing; nested locations; continuation-passing; explicit substitutions

1 Introduction

The π -calculus [1,2] is, by most people, considered the classic process calculus for modelling mobile systems. Its most prominent features, compared to its predecessor CCS, are the communication of names as expressed by the reduction rule

$$n(m) . P \mid \bar{n}\langle o \rangle . Q \rightarrow_{\pi} \{o/m\}P \mid Q$$

* Corresponding author. Address: IT University of Copenhagen, Rued Langgaards Vej 7, DK-2300 Copenhagen S, Denmark.

Email addresses: mikkelbu@itu.dk (Mikkel Bundgaard), hilde@itu.dk (Thomas Hildebrandt), jcg@itu.dk (Jens Chr. Godskesen).

and the creation of local names with static scope. Combined these concepts provide the π -calculus with most of its expressive power. Notably, by representing the location of a process by its links, the ability to dynamically change the communication links between processes makes it possible to model mobile computing processes.

This account of mobility has been very successful for a decade, but it has its limitations. Recently, a number of calculi have been proposed, e.g. the Ambient calculus [3] and the Seal calculus [4], with an explicit representation of mobile computing resources in nested locations which is not easy to model in the π -calculus. Many of the proposed calculi include the name-passing capability of the π -calculus as well, which increases the complexity of the calculi. A natural question is if name-passing can be expressed using mobile computing resources in nested locations alone.

In this paper we present a compositional encoding of the synchronous π -calculus, and thus name passing, in a pure higher-order calculus with nested locations, obtained as a simple, conservative extension of the core process-passing subset of Thomsen's Plain CHOCS [5]. Thomsen demonstrated that the π -calculus could be encoded in Plain CHOCS by making crucial use of explicit name substitution to encode the dynamic linking. The calculus is a simplified variant of the Homer calculus of Higher-Order Mobile Embedded Resources presented in [6]. The Homer calculus does *not* have explicit name substitution, thus the encoding of [5] cannot be applied in Homer. Homer introduces mobile computing resources in, possibly local, named locations, and our encoding demonstrates that this, together with higher-order process-passing is sufficient to express π -calculus name-passing without relying on name substitution. The encoding uses a novel continuation passing style to facilitate the encoding of synchronous communication.

The main motivation for this paper is to examine the expressive power of Homer, in particular to relate it to the most examined calculus for mobile processes, the π -calculus, and hereby indirectly relate it to other calculi for mobile processes. As mentioned above Homer is a simple process-passing calculus augmented with an explicit representation of locations giving rise to a simple semantics, but with great expressive power. Specifically the ability to communicate with local nested named locations and the representation of locations as prefixes makes the encoding of the π -calculus easier.

To briefly recall, mobility of processes in Plain CHOCS is introduced by replacing the *name*-passing of the π -calculus with *process*-passing. We will represent this kind of interaction with the prefixes $\bar{n}(q)$ (*send*) and $n(x)$ (*receive*), respectively. Here x is a *process variable* for which the received process is substituted,

as expressed formally by the reaction rule

$$\bar{n}\langle q \rangle . p_1 \parallel n(x) . p_2 \searrow p_1 \parallel p_2[q/x] . \quad (1)$$

As usual, there may be any number of occurrences of x in p_2 meaning that processes may both be discarded and copied, making Plain CHOCS a *non-linear* higher-order calculus. However, as also remarked by Thomsen, the process q cannot start computing before it is moved, and once it has started computing, it cannot be moved again. This is known as *code* mobility or *weak* mobility, as opposed to *process* mobility or *strong* mobility, where processes may move *during* their computation.

In Homer strongly mobile computing resources in nested named locations are introduced by allowing an *additional* kind of interaction, given by two new complementary prefixes $n\langle q \rangle$ (*resource*) and $\bar{n}(x)$ (*take*). The process $n\langle q \rangle . p_1$ denotes a resource q residing at the location (or address) n which may be *moved* or *taken* by the complementary prefix, $\bar{n}(x) . p_2$. Just as for the previous interaction the synchronisation is expressed by the reaction

$$n\langle q \rangle . p_1 \parallel \bar{n}(x) . p_2 \searrow p_1 \parallel p_2[q/x] . \quad (2)$$

The important difference between the two types of interactions, is that the resource q in $n\langle q \rangle$ is able to interact with processes outside its location by allowing resources to be sent down to and taken up from q . In other words, the state of q may be modified by processes outside the location n . We introduce this kind of interaction, as in the Mobile Resources (MR) calculus [7], by allowing *sequences* as addresses in the downward prefixes *take* and *send*. For instance, using the sequence n_1n_2 in the address of the take prefix, a resource q may be taken from the address n_2 in a resource running at address n_1 , as in

$$n_1\langle n_2\langle q \rangle . q' \parallel q'' \rangle . p_1 \parallel \bar{n_1n_2}(x) . p_2 \searrow n_1\langle q' \parallel q'' \rangle . p_1 \parallel p_2[q/x] . \quad (3)$$

Dually, using the sequence n_1n_2 in the *send* prefix, a resource q may be sent to address n_2 in a resource running at address n_1 by

$$\bar{n_1n_2}\langle q \rangle . p_1 \parallel n_1\langle n_2(x) . p'_2 \parallel p''_2 \rangle . p_2 \searrow p_1 \parallel n_1\langle p'_2[q/x] \parallel p''_2 \rangle . p_2 . \quad (4)$$

We allow sequences of names in addresses of the *receive* and *resource* prefixes as well. This permits the physical nested structure of the address space to be different from the abstract structure

$$\bar{n_1n_2}\langle q \rangle . p_1 \parallel n_1n_2(x) . p_2 \searrow p_1 \parallel p_2[q/x] .$$

We will, however, only use this for an alternative encoding, where we do not

use any auxiliary names. For the main encoding of this paper this feature of Homer is superfluous.

To summarise, the two dual kinds of process movement allow us to express mobile resources in a nested location structure that may be moved (and copied) locally or upwards, and to send passive resources that may be received (and copied) by a local process or a sub-resource.

The interaction presented above is the only kind of interaction we need for the results presented in this paper. The only other feature is that of local names as found in the π -calculus and Plain CHOCS. We let $(n)p$ denote a process p in which the name n is local. The full Homer calculus [6] also allows for mobile resources that can make internal reactions. This however requires a more careful treatment of free names in the semantics.

Related Work

Thomsen demonstrated that the recursion and the name-passing of the π -calculus can be encoded in Plain CHOCS [5] by passing wires instead of names. An *a-wire* representing the π -calculus name a is defined as

$$i? . a?x . c!x . nil + o? . c?x . a!x . nil ,$$

where i and o are used to indicate whether the wire is used for input or output, and c is used as an auxiliary forwarder. Thomsen used an encoding scheme in two levels, resembling the encoding presented in this paper: a structurally defined encoding translating free names and names bound by an input prefix into process variables, and names bound by restriction into wires, and on top-level, an instantiation of the process variables representing free names with wires.

The most complex part of the encoding is the encoding of prefixes defined as

$$\begin{aligned} \llbracket x(y) . P \rrbracket_1 &= \left(x[c \mapsto c'][i \mapsto i'][o \mapsto o'] \mid i'! . c'?y . \llbracket P \rrbracket_1 \right) \backslash c' \backslash i' \backslash o' \\ \llbracket \bar{x}(y) . P \rrbracket_1 &= \left(x[c \mapsto c'][i \mapsto i'][o \mapsto o'] \mid o'! . c'!y . \llbracket P \rrbracket_1 \right) \backslash c' \backslash i' \backslash o' , \end{aligned}$$

where $(x[c \mapsto c'][i \mapsto i'][o \mapsto o'] \mid \dots) \backslash c' \backslash i' \backslash o'$ uses an explicit substitution $[c \mapsto c'][i \mapsto i'][o \mapsto o']$ to localise the wire input for x . This localisation is essential for the encoding, since the main problem of encoding a first-order calculi, like the π -calculus, is dynamic linking (the name substitution). The encoding presented in this paper differs in a crucial way from the encoding by Thomsen, since Homer does not include the explicit name substitution of Plain CHOCS. Instead we obtain dynamic linking from the possibility to communicate with strongly mobile resources in local locations.

Zimmer presented in [8] an encoding of the synchronous π -calculus into a restricted Ambient calculus containing only the mobility primitives and the hierarchical structure of the ambients, and therefore neither communication nor name substitution. Much like the presentation in this paper, Zimmer designed an intermediate calculus π_{esc} (π -calculus with Explicit Substitutions and Channels), but whereas π_{esc} has explicit variables and channels as part of the syntax, we follow the path of the $\pi\xi$ calculus [9], also a π -calculus with explicit substitutions, and consider our processes with respect to *one* global environment. Another clear difference between the approach of Zimmer and ours is that the mobility of Mobile ambients is subjective, and the mobility in Homer is objective.

The connection between first-order and higher-order calculi has been examined in several contexts, most notably in [10], where Sangiorgi shows how higher-order π -calculus, containing higher-order communication primitives, can be represented in first-order π -calculus. However the representability of π -calculus in higher-order π -calculus is not examined in [10].

In section 13.3 of [11] Sangiorgi and Walker give an encoding of the asynchronous localized π -calculus (AL π) in the asynchronous higher-order π -calculus (AHO π). Localized means that only output-capabilities of names can be communicated. Both the mentioned calculi are typed and the encoding from AL π to AHO π depends on the typing of the AL π process, but we have for clarity omitted the types in this presentation. Intuitively the encoding works by sending an abstraction instead of a name. For instance, consider the example where a process sends a name b to some receiving process p , which can then only use the name to send along. This is encoded by sending an abstraction $(z) . \bar{b}\langle z \rangle$ which, when applied to a value v , sends v along b . For example, the AL π -process

$$\bar{a}\langle b \rangle \mid a(x) . \bar{x}\langle c \rangle$$

is translated into the following AHO π -process

$$\bar{a}\langle (y) . \bar{b}\langle y \rangle \rangle \mid a(x) . x[(z) . \bar{c}\langle z \rangle] ,$$

where $x[(z) . \bar{c}\langle z \rangle]$ is the application between the abstraction, to be substituted in for x , and the value $(z) . \bar{c}\langle z \rangle$. This process can then react twice to become the encoding of the AL π -process $\bar{b}\langle c \rangle$

$$\bar{a}\langle (y) . \bar{b}\langle y \rangle \rangle \mid a(x) . x[(z) . \bar{c}\langle z \rangle] \rightarrow_{\pi} (y) . \bar{b}\langle y \rangle [(z) . \bar{c}\langle z \rangle] \rightarrow_{\pi} \bar{b}\langle (z) . \bar{c}\langle z \rangle \rangle .$$

The ability to send a process to a resource in a local named location in Homer is related to the ability to apply an abstraction to a process in HO π . That is, applying a process p to an abstraction a , $a[p]$, can be represented in Homer by

$$(n)(n\langle m(x) . a \rangle \parallel \overline{nm}\langle p \rangle) .$$

From this observation we can in a straightforward manner translate the encoding from $AL\pi$ to $AHO\pi$ to an encoding from $AL\pi$ to Homer.

It is noted that the encoding can be extended to the synchronous π -calculus, but that it increases the complexity due to the lack of continuations of applications. The extension to synchronous communication as well as a variant, where we only allow for the communication of input-capabilities (and thus treat dynamic binding), are left as so-called “more difficult” exercises. This is exactly the difficult parts of the encoding of the π -calculus into Homer. Note that a translation of the full π -calculus, thus combining the original encoding and the two exercises, is not addressed in [11].

We currently investigate if the ideas in the encoding into Homer can be transferred to an encoding into $HO\pi$. It seems quite straightforward to adapt the continuation passing style encoding of synchronous communication. However, it is not trivial to represent the dynamic binding necessary to allow communication of input-capabilities. We expect that we will have to use a $HO\pi$ -calculus with both product and unions, and hence both product and union types. We will probably need the products to encode the synchronous communication, and the unions to encode that a name can be used both for input and for output.

The Seal calculus [4] is a calculus with name-passing, non-linear process-passing and named, nested locations as Homer. But it is not immediately clear, though, if one can reduce name-passing to process-passing in Seal, since scope extension for mobile processes in the Seal semantics depends on name-passing.

In [12] Carbone and Maffei examines the expressive power of a π -calculus with *polyadic synchronisation*, which is a generalisation of the communication mechanism of the π -calculus that allows channel names to be composite, like the addresses in Homer. For instance they allow reactions like the following

$$x \cdot y(z) . P \mid \overline{x \cdot y} \langle w \rangle . Q \rightarrow_{\pi} \{w/z\}P \mid Q ,$$

where $x \cdot y$ is a vector of the names x and y . They show that the expressive power of a π -calculus with polyadic synchronisation depends on the degree of synchronisation. Compared to addresses in Homer they have no notion of locations as primitive in the calculi, and hence they do not allow for the possibility to break up an address in components, one part matching the location hierarchy and on part matching the actual prefix, as illustrated in (3) and (4).

We prove the correspondence between the π -calculus and its encoding in Homer following the same approach as [11], by proving an operational correspondence between π -calculus processes and their encoding in Homer, from which we can infer full abstractness with respect to barbed bisimulation and

soundness with respect to barbed congruence.

Outline

In Section 2 we present the syntax and reaction semantics of Homer. In Section 3 we do the same for the monadic synchronous π -calculus and introduce a π -calculus with explicit substitutions. We present the encoding, give some examples of the encoding, and present an extension to matching and an alternative encoding in Section 4. In Section 5 we prove the operational correspondence between π -calculus processes and their encoding in Homer, from which we infer full abstraction with respect to barbed bisimulation, and soundness with respect to barbed congruence.

2 Homer

We assume an infinite set of *names* \mathcal{N} ranged over by m and n , and let \tilde{n} range over finite sets of names. We let δ range over non-empty sequences of names, referred to as *addresses*, and let $|\delta|$ denote the length of the address δ . We assume an infinite set of *process variables* \mathcal{V} ranged over by x and y .

The set \mathcal{P} of *process expressions*, ranged over by p, q, r , is then defined by the grammar:

$$\begin{array}{lcl}
 p, q, r ::= & \mathbf{0} & \text{inactive process} \\
 & | & \\
 & p \parallel q & \text{parallel composition} \\
 & | & \\
 & (n)p & \text{let } n \text{ be local in } p \\
 & | & \\
 & \lambda . p & \text{action prefixing} \\
 & | & \\
 & x & \text{process variable}
 \end{array}$$

and the set of prefixes Λ , ranged over by λ , is defined by the grammar:

$$\begin{array}{lcl}
 \lambda ::= & \delta(x) & \text{receive a resource at } \delta \text{ and bind it to } x \\
 & | & \\
 & \bar{\delta}(x) & \text{take resource from } \delta \text{ and bind it to } x \\
 & | & \\
 & \bar{\delta}\langle r \rangle & \text{send a resource } r \text{ to } \delta \\
 & | & \\
 & \delta\langle r \rangle & \text{resource } r \text{ at } \delta
 \end{array}$$

The process constructors are the standard constructors from concurrent process calculi, extended with process variables as in the λ -calculus and Plain CHOCS. For an introduction to Homer, its semantics, and examples, see [6].

The prefixes $\bar{\delta}\langle r \rangle$ and $\delta(x)$ correspond to the send and receive prefixes in Plain CHOCS, except from paths and not only names being allowed as addresses. The new prefixes allowing strong mobility are the prefixes $\delta\langle r \rangle$ and $\bar{\delta}(x)$. We let the restriction operator (n) bind the name n and the prefixes $\delta(x)$ and $\bar{\delta}(x)$ bind the variable x . The sets $fn(\lambda)$, $fn(p)$ and $fv(\lambda)$, $fv(p)$ of *free names* and *free variables* are defined accordingly as usual.

We say that a process with no free variables is *closed* and let \mathcal{P}_c denote the set of closed processes. For any subset \mathcal{P}' of \mathcal{P} we let \mathcal{P}'/α denote the set of α -equivalence classes (with respect to both names and variables) of process expressions. We define the process $p[q/x]$ to be p with all free occurrences of x replaced by q , if necessary α -converting p such that no free names and variables in q are bound.

By convenience we omit trailing $\mathbf{0}$ s and hence write λ instead of $\lambda . \mathbf{0}$. We let prefixing and restriction be right associative and bind stronger than parallel composition. For a set of names $\tilde{n} = \{n_1, \dots, n_k\}$ we let $(\tilde{n})p$ denote $(n_1) \cdots (n_k)p$. We write $\tilde{m}\tilde{n}$ for $\tilde{m} \cup \tilde{n}$, always implicitly assuming $\tilde{m} \cap \tilde{n} = \emptyset$. Finally, we will write n for the singleton set $\{n\}$ when no confusion can occur.

2.1 Reductions

We provide Homer with a reduction semantics defined through the use of contexts, structural congruence, and reduction rules.

Structural congruence \equiv is the least congruence on \mathcal{P}/α satisfying the following rules.

$$\begin{array}{ll}
E_1. p \parallel \mathbf{0} \equiv p & E_2. (n)\mathbf{0} \equiv \mathbf{0} \\
E_3. p \parallel q \equiv q \parallel p & E_4. (n)(m)p \equiv (m)(n)p \\
E_5. (p \parallel p') \parallel p'' \equiv p \parallel (p' \parallel p'') & E_6. (n)p \parallel q \equiv (n)(p \parallel q), \text{ if } n \notin fn(q)
\end{array}$$

Equations E_1 , E_3 , and E_5 express that $(p, \parallel, \mathbf{0})$ is a commutative monoid, and equations E_2 , E_4 , and E_6 enforce the rules for scope.

Contexts \mathcal{C} are, as usual, terms with a hole $(-)$, and we write $\mathcal{C}(p)$ for the insertion of p in the hole of context \mathcal{C} . Note that free names (and variables) of p may get bound by insertion of p in the hole of a context. We define $fn(\mathcal{C})$ as $fn(\mathcal{C}(\mathbf{0}))$ and $fv(\mathcal{C})$ as $fv(\mathcal{C}(\mathbf{0}))$.

Evaluation contexts \mathcal{E} are contexts with no free variables, and whose hole is not guarded by a prefix, nor appear as the object of a send or resource prefix,

i.e.

$$\mathcal{E} ::= (-) \quad | \quad \mathcal{E} \parallel p \quad | \quad p \parallel \mathcal{E} \quad | \quad (n)\mathcal{E} \quad , \text{ for } p \in \mathcal{P}_c.$$

As our calculus allows actions involving terms at depths arbitrarily far apart, we define a family of *path contexts* $\mathcal{C}_\gamma^{\tilde{n}}$, indexed by a path address $\gamma \in \mathcal{N}^*$ and a set of names \tilde{n} . The path address γ indicates the path under which the hole of the context is found, and the set \tilde{n} indicates the bound names of the hole. We define the path contexts inductively in \tilde{n} and γ by $\mathcal{C}_\epsilon^\emptyset ::= (-)$ and whenever $p, q \in \mathcal{P}_c$,

$$\mathcal{C}_{\delta\gamma}^{\tilde{n}\tilde{m}} ::= \delta\langle(\tilde{n})(\mathcal{C}_\gamma^{\tilde{m}} \parallel p)\rangle . q \quad , \text{ where } \tilde{n} \cap \gamma = \emptyset.$$

The side condition ensures that none of the names in the path address are bound.

We need to handle scope extension when resources are taken up from sub-locations. For this purpose we define an *open* operator on path contexts $\mathcal{C}_\gamma^{\tilde{n}} \setminus \tilde{m}$ inductively by: $\mathcal{C}_\epsilon^\emptyset \setminus \tilde{m} = \mathcal{C}_\epsilon^\emptyset$ and

$$\mathcal{C}_{\delta\gamma}^{\tilde{n}_1\tilde{n}_2} \setminus \tilde{m} = \delta\langle(\tilde{n}_1 \setminus \tilde{m})(\mathcal{C}_\gamma^{\tilde{n}_2} \setminus \tilde{m} \parallel p)\rangle . q \quad ,$$

if $\mathcal{C}_{\delta\gamma}^{\tilde{n}_1\tilde{n}_2} = \delta\langle(\tilde{n}_1)(\mathcal{C}_\gamma^{\tilde{n}_2} \parallel p)\rangle . q$ and $\tilde{n}_1\tilde{n}_2 \cap \text{fn}(\mathcal{C}_{\delta\gamma}^{\tilde{n}_1\tilde{n}_2}) = \emptyset$. The side condition ensures that the opened names do not equal any free names outside the scope. When applied in the reduction rule, this condition can always be met by α -conversion, and ensures that we can extend the scope by using the open operator and place the restriction at top level.

We finally define \searrow as the least binary relation on $\mathcal{P}_{c/\alpha}$ satisfying the following rules and closed under all evaluation contexts \mathcal{E} and structural congruence.

$$\begin{aligned} (\textit{send}) \quad & \overline{\gamma\delta}\langle r \rangle . q \parallel \mathcal{C}_\gamma^{\tilde{m}}(\delta(x) . p) \searrow q \parallel \mathcal{C}_\gamma^{\tilde{m}}(p[r/x]) \quad , \\ & \text{if } \tilde{m} \cap (\text{fn}(r) \cup \delta) = \emptyset \end{aligned}$$

$$\begin{aligned} (\textit{take}) \quad & \mathcal{C}_\gamma^{\tilde{m}}(\delta\langle r \rangle . q) \parallel \overline{\gamma\delta}\langle x \rangle . p \searrow (\tilde{n} \cap \tilde{m})(\mathcal{C}_\gamma^{\tilde{m}} \setminus \tilde{n}(q) \parallel p[r/x]) \quad , \\ & \text{if } \tilde{n} = \text{fn}(r) \text{ and } \tilde{m} \cap (\delta \cup \text{fn}(p)) = \emptyset \end{aligned}$$

The *(send)* rule expresses how a passive resource r is sent down to the sub-location γ , where it is received at the address δ , and substituted into the receiving process p , possibly in several copies. The side condition guarantees that no free names of r may be bound by the path context. This can always be guaranteed by α -conversion, and thus will never block mobility.

The *(take)* rule captures that a resource r is taken from the sub-location γ , where it is running at the address δ and substituted into the process p , possibly

in several copies. The open operator is used to extend the scope of the local names defined in $\mathcal{C}_\gamma^{\tilde{m}}$ that occur free in r . Note that for $\gamma = \epsilon$ and taking the path δ to be of length 1, the two rules reduce to the two reduction rules (1) and (2) given in the introduction.

2.2 Encoding Replication

We may encode general recursion in Homer (up to weak equivalence) using copyability of resources [6]. The encoding of general recursion differs slightly from the encoding in Plain CHOCS [5], since recursion variables may appear at sub-locations, and since we use a local location instead of an explicit substitution. However, we only need recursion to encode replication, so by utilising a direct encoding of replication, we get the following simpler encoding

$$!p =_{def} (a)(!^a p) \text{ ,}$$

where $!^a p = \bar{a}\langle r \rangle \parallel r$ and $r = a(x) . (p \parallel \bar{a}\langle x \rangle \parallel x)$, for $a \notin fn(p)$. Intuitively, r places a new copy of p in parallel with $!^a p$ for each reaction step. Using this encoding we have the following reactions

$$!^a p \searrow p \parallel !^a p \searrow p \parallel p \parallel !^a p \text{ .}$$

3 The Pi-calculus

We present the monadic synchronous π -calculus without summations. We present its syntax, structural congruence relation, and the reaction relation. For a much more thorough introduction to and description of the π -calculus, see e.g. [1,2].

Even though some of the process constructors of Homer collide with the constructors of the π -calculus, we will nonetheless use the same symbols, since any ambiguity can easily be resolved from the context. We let N denote an infinite set of names and let m, n range over N . The set \mathcal{P}_π of *process expressions* is

then defined by the grammar

$P, Q ::=$	$\mathbf{0}$	inactive process
	$P \mid Q$	parallel composition
	$(\nu n)P$	restriction
	$!P$	replication
	$\bar{n}\langle m \rangle . P$	output m along n
	$n(m) . P$	receive along n and bind it to m

We consider π -calculus terms up to α -conversion and define *structural congruence* \equiv_π in the π -calculus, as for Homer, as the least congruence on $\mathcal{P}_{\pi/\alpha}$ satisfying the following rules.

$$\begin{array}{ll}
E_1. P \mid \mathbf{0} \equiv_\pi P & E_2. (\nu n)\mathbf{0} \equiv_\pi \mathbf{0} \\
E_3. P \mid Q \equiv_\pi Q \mid P & E_4. (\nu n)(\nu m)P \equiv_\pi (\nu m)(\nu n)P \\
E_5. (P \mid P') \mid P'' \equiv_\pi P \mid (P' \mid P'') & E_6. (\nu n)P \mid Q \equiv_\pi (\nu n)(P \mid Q), \text{ if } n \notin fn(Q)
\end{array}$$

Again, we define the reaction relation \rightarrow_π in terms of evaluation contexts

$$\mathcal{E}_\pi ::= (-) \quad \mid \quad \mathcal{E}_\pi \mid P \quad \mid \quad P \mid \mathcal{E}_\pi \quad \mid \quad (\nu n)\mathcal{E}_\pi, \text{ for } P \in \mathcal{P}_\pi.$$

\rightarrow_π is then the least binary relation over $\mathcal{P}_{\pi/\alpha}$ satisfying the following rules and closed under all evaluation contexts \mathcal{E}_π and structural congruence.

$$\begin{array}{c}
\text{(React)} \frac{}{n(m) . P \mid \bar{n}\langle m' \rangle . Q \rightarrow_\pi \{m'/m\}P \mid Q} \\
\text{(Repl)} \frac{}{!P \rightarrow_\pi P \mid !P}
\end{array}$$

As usual, we let $\{n/m\}P$ denote the process P with all free occurrences of m replaced by n , using α -conversion to avoid that n becomes bound in P . Note that we have chosen to use a form of guarded replication $!P$, for which it takes one reaction step to activate each instance of P .

3.1 *Pi-calculus with Explicit Substitutions*

In this subsection we introduce an intermediate calculus: π -calculus with *explicit substitutions* to ease the proof of our encoding and to make the intuition of our encoding clearer. The only substantial way this π -calculus differs from

traditional π -calculus, is that we record the substitution occurring in a synchronisation in a global environment. As mentioned in the introduction there already exist several variants of the π -calculus with explicit substitutions, see e.g. [8,9,13].

An *environment* (or *substitution*) σ is associated with a process P , giving rise to the judgement $\sigma \vdash P$. The substitution is a partial function from names to names, and we let $\text{dom } \sigma$ ($\text{codom } \sigma$) denote the domain (codomain) of the function σ , respectively. We require that if $\sigma n \in \text{dom } \sigma$ then $\sigma(\sigma n) = \sigma n$ (i.e. that the operator is idempotent). Furthermore, for a judgement $\sigma \vdash P$ we require that $\text{dom } \sigma \supseteq \text{fn}(P)$. We write id_A for the substitution that is the identity on A .

We let $P\sigma$ denote the process P , where all free names of P have been simultaneously substituted according to σ . We let $\sigma[m \mapsto n]$ denote the substitution σ extended such that m maps to n , and let $\sigma[\bar{m} \mapsto \bar{n}]$, where \bar{m} and \bar{n} are sequences of names of equal length k , denote the substitution σ extended such that m_i maps to n_i , for $1 \leq i \leq k$. Finally, we let $\mathcal{P}_{\pi\sigma}$ denote the set of process judgements.

We define the reaction relation $\rightarrow_{\pi\sigma}$ as the least binary relation satisfying the following rules and closed under all evaluation contexts $\mathcal{E}_{\pi\sigma}$ and structural equivalence $\equiv_{\pi\sigma}$. In the rule (React σ), the first part of the side-condition can always be satisfied by α -conversion. It guarantees that local names differ from the names already present in the substitution, in particular the free names of the process.

$$\text{(React}\sigma) \frac{\sigma \vdash n(m) . P \mid \bar{n}'\langle m' \rangle . Q \rightarrow_{\pi\sigma} \sigma[m \mapsto \sigma m'] \vdash P \mid Q}{\text{if } m \notin \text{dom } \sigma \cup \text{codom } \sigma \text{ and } \sigma n = \sigma n'}$$

The set of names $\{n \mid \sigma n \in \text{codom } \sigma \setminus \text{dom } \sigma\}$ corresponds to restricted names in the π -calculus, which will be made precise in the definition of structural equivalence below. Note that in general more than one name can map to a name $n \in \text{codom } \sigma \setminus \text{dom } \sigma$. The rule Rest σ then corresponds to the usual reaction under restriction.

$$\text{(Rest}\sigma) \frac{\sigma[n \mapsto n] \vdash P \rightarrow_{\pi\sigma} \sigma'[n \mapsto n] \vdash P'}{\sigma[m \mapsto n] \vdash P[m/n] \rightarrow_{\pi\sigma} \sigma'[m \mapsto n] \vdash P'[m/n]} ,$$

if $n \notin \text{dom } \sigma \cup \text{dom } \sigma'$ and $m \notin \{n\} \cup \text{dom } \sigma \cup \text{codom } \sigma$.

Last, the rule for replication is defined as for the π -calculus

$$\text{(Repl}\sigma) \frac{}{\sigma \vdash !P \rightarrow_{\pi\sigma} \sigma \vdash P \mid !P} .$$

Evaluation contexts for the π -calculus with explicit substitutions are defined with respect to an environment σ . We define $\mathcal{E}_{\pi\sigma}$ (with respect to σ) by the following grammar

$$\mathcal{E}_{\pi\sigma} ::= (-) \quad | \quad \mathcal{E}_{\pi\sigma} \mid P \quad | \quad P \mid \mathcal{E}_{\pi\sigma} , \text{ for } fn(P) \subseteq \text{dom } \sigma .$$

Note that we have removed restriction $(\nu n)\mathcal{E}_{\pi\sigma}$ as an evaluation context. The reason for this is the complications that arise when we combine restriction and a global environment, since we cannot record bound names in the environment.

In [9] they solve this problem by requiring that all bound names are pairwise distinct and differ from the free names.

We define structural equivalence of the π -calculus with explicit substitutions in terms of the structural congruence of the π -calculus. Structural equivalence $\equiv_{\pi\sigma}$ is defined as the least equivalence satisfying the following rules.

$$\frac{P \equiv_{\pi} P'}{\sigma \vdash P \equiv_{\pi\sigma} \sigma \vdash P'}$$

$$\sigma \vdash (\nu n)P \equiv_{\pi\sigma} \sigma[n \mapsto m] \vdash P , \quad (*)$$

if $n \neq m$ and $n, m \notin \text{dom } \sigma \cup \text{codom } \sigma$.

Note that this is only an equivalence and not a congruence, since we can only apply the rules at top-level. The intuition behind the last rule is that we can lift a top-level restriction to the environment, by making it map to a fresh name.

We end this section by noting that there is an operational correspondence between a traditional π -calculus term and the corresponding term with explicit substitutions. In the following propositions we use a standard form, where restrictions cannot occur at top-level (i.e. restriction can only occur under replication or behind a prefix). We let Q range over the standard form in the following propositions and define the standard form by the following grammar

$$Q ::= \mathbf{0} \quad | \quad Q \mid Q \quad | \quad !P \quad | \quad \bar{n}\langle m \rangle . P \quad | \quad n(m) . P ,$$

where P can be an arbitrary π -calculus process.

Proposition 1 *If $P \rightarrow_{\pi} P'$ and $P \equiv_{\pi} (\nu \tilde{n})Q\sigma$, where Q is in standard form and $\tilde{n} = \text{codom } \sigma \setminus \text{dom } \sigma$, then $\sigma \vdash Q \rightarrow_{\pi\sigma} \sigma' \vdash Q'$ and $P' \equiv_{\pi} (\nu \tilde{n}')Q'\sigma'$, where $\tilde{n}' = \text{codom } \sigma' \setminus \text{dom } \sigma'$.*

Proposition 2 *If $\sigma \vdash Q \rightarrow_{\pi\sigma} \sigma' \vdash Q'$, then there exists a P' such that $(\nu \tilde{n})(Q\sigma) \rightarrow_{\pi} P'$ and $P' \equiv_{\pi} (\nu \tilde{n}')(Q'\sigma')$, where $\tilde{n} = \text{codom } \sigma \setminus \text{dom } \sigma$ and $\tilde{n}' = \text{codom } \sigma' \setminus \text{dom } \sigma'$.*

4 The Encoding

We adopt the following shorthand for taking a copy of the resource at address n and placing the copy in the variable x , as the prefix $n \triangleright x$, defined as

$$n \triangleright x . p =_{def} \bar{n}(x) . (n\langle x \rangle \parallel p) .$$

We encode π -calculus processes with names in $\mathcal{N} \setminus \{v, c, s, r\}$ as Homer processes with names in $\mathcal{N} \uplus \mathcal{N}' \uplus \{v, c, s, r\}$, where \mathcal{N}' is ranged over by n', m' and we assume the mapping of $n \in \mathcal{N}$ to $n' \in \mathcal{N}'$ is a bijection. The names $\{v, c, s, r\}$ are used as auxiliary names in the encoding. In Section 4.3 we present a variation of the encoding, that does not depend on neither the set \mathcal{N}' nor the names $\{v, c, s, r\}$, we will however use the encoding given in this section in the rest of the paper due to readability.

We encode a π -calculus name n as a mobile resource $\llbracket n \rrbracket$ that can perform two tasks: sending and receiving along the name n .

$$\begin{aligned} Send_n &= v(x) . c(y) . \bar{n}\langle x \rangle . y \\ Receive_n &= v(x) . c(y) . n(z) . (a)(a\langle x \rangle \parallel \bar{a}\bar{v}\langle z \rangle . \bar{a}\langle z' \rangle) . (y \parallel z') \\ \llbracket n \rrbracket &= s\langle Send_n \rangle \parallel r\langle Receive_n \rangle \end{aligned}$$

The $Send_n$ process can be seen as taking two parameters on the locations v and c , respectively. On location v it takes the encoding of the name $\llbracket m \rrbracket$ to send, and on location c the encoding of the continuation $\llbracket P \rrbracket$, resulting in a process of the following form $\bar{n}\langle \llbracket m \rrbracket \rangle . \llbracket P \rrbracket$.

$Receive_n$ also takes two parameters: on location v it takes a process $Bind_b = v(x) . b'\langle x \rangle$ responsible for binding the received name, and on location c the encoding $\llbracket Q \rrbracket$ of the continuation, resulting in a process of the following form $n(z) . (a)(a\langle Bind_b \rangle \parallel \bar{a}\bar{v}\langle z \rangle . \bar{a}\langle z' \rangle) . (\llbracket Q \rrbracket \parallel z')$.

In parallel these two processes, $\bar{n}\langle \llbracket m \rrbracket \rangle . \llbracket P \rrbracket$ and $n(z) . (a)(a\langle Bind_b \rangle \parallel \bar{a}\bar{v}\langle z \rangle . \bar{a}\langle z' \rangle) . (\llbracket Q \rrbracket \parallel z')$, can perform a synchronisation on the location n , corresponding to the actual π -calculus synchronisation. After the synchronisation the received name $\llbracket m \rrbracket$ is sent to the $Bind_b$ process and finally placed in parallel with the continuations, resulting in $\llbracket P \rrbracket \parallel \llbracket Q \rrbracket \parallel b'\langle \llbracket m \rrbracket \rangle$.

The encoding of a π -calculus process $\sigma \vdash P$ is done at two levels: at top-level, we translate all names in σ . At the next, we give a compositional encoding of P . Let $\llbracket \sigma \vdash P \rrbracket$ denote the following

$$(\tilde{m})\left(\llbracket P \rrbracket \parallel \prod_{n \in \text{dom } \sigma} n'\langle \llbracket \sigma n \rrbracket \rangle\right) ,$$

where $\tilde{m} = \{n' \mid n \in \text{dom } \sigma \text{ and } n \neq \sigma n\} \cup \{n \mid n \in \text{codom } \sigma \setminus \text{dom } \sigma\}$. Note that the encoding of a name n (or more precisely, its image under the substitution) is kept as a resource at the address n' . The restriction \tilde{m} restricts the locations of substituted names, which intuitively are bound to local names and simulates the restrictions lifted to the environment. Then the encoding of a π -calculus process P is defined to be $\llbracket id_{fn(P)} \vdash P \rrbracket$. The encoding $\llbracket - \rrbracket$ of local names, parallel composition, replication, and the inactive process is defined inductively as

$$\begin{aligned} \llbracket (\nu n)P \rrbracket &= (n)(n')(\llbracket P \rrbracket \parallel n'\langle \llbracket n \rrbracket \rangle) \\ \llbracket P \mid Q \rrbracket &= \llbracket P \rrbracket \parallel \llbracket Q \rrbracket \\ \llbracket !P \rrbracket &= !\llbracket P \rrbracket \\ \llbracket \mathbf{0} \rrbracket &= \mathbf{0} \end{aligned}$$

In the encoding of a local name n we restrict both the address n' and the actual name n , such that we can match an α -conversion in the π -calculus. Note that we in the encoding of replication utilise our encoding of replication from Section 2.2. The translation is a homomorphism for the remaining constructs. We encode the output and input prefixes of the π -calculus as follows

$$\begin{aligned} \llbracket \bar{n}\langle m \rangle . P \rrbracket &= (a)(n' \triangleright x . (a\langle x \rangle \parallel m' \triangleright y . \overline{asv}\langle y \rangle . \overline{asc}\langle \llbracket P \rrbracket \rangle . \overline{as}(x'') . \bar{a}(z) . x'')) \\ \llbracket n(m) . P \rrbracket &= (a)(n' \triangleright x . (m')(a\langle x \rangle \parallel \overline{arv}\langle Bind_m \rangle . \overline{arc}\langle \llbracket P \rrbracket \rangle . \overline{ar}(x'') . \bar{a}(z) . x'')) \end{aligned}$$

where $Bind_m = v(x) . m'\langle x \rangle$. As described above, the process $Bind_m$ is responsible for binding the received name to the local name m . Intuitively, $\llbracket \bar{n}\langle m \rangle . P \rrbracket$ first takes a copy of the encoded name at location n' and keeps it at the local address a . Then a copy of the encoded name at location m' is taken which, together with the continuation $\llbracket P \rrbracket$, is sent to the *Send* part of the encoded name at address a . The *Send* part is then retrieved, and finally the rest of the encoded name at location a is discarded.

$\llbracket n(m) . P \rrbracket$ is encoded using the same template, but it is a bit more complicated due to the binding. First and foremost, we restrict the location, m' , of the formal parameter of the input prefix to ensure that this is only available to the continuation, and that it can be α -converted. Secondly, we use the $Bind_m$ process to create a new location to contain the received, encoded name.

There are several relevant observations to be made from the encoding. First, our encoding of name substitution relies heavily on the ability to communicate with resources in local, named locations in Homer, and on our ability to copy resources. We use the ability to communicate with resources in local, named locations to localise the communication, instead of using explicit name substitution as in the Plain CHOCS encoding. We make use of the non-linearity of

Homer to take a copy of the encoding of a name, each time we use it, and in the encoding of replication. These are the only places where we make use of the non-linearity of Homer, but in both cases we take *precisely* one copy, and in all other places the communication is linear. So we do not utilise the full expressive power of the calculus in the encoding. Hence the encoding would also work in a weaker sub-calculus, where we only have a clone construct and linear copying. Second, strongly mobile resources are utilised in several places in the encoding: in the encoding of $Receive_n$ and in the two prefixes, where we send resources to local addresses and let them compute, before taking the resources up again.

However, note that there is no need for using a resource prefix for holding the encoding of a name, $n'\langle\llbracket m \rrbracket\rangle$, we could just as well have used a send prefix, $\bar{n}\langle m \rangle$. But when we consider the encoding without auxiliary names in Section 4.3, this actually makes a big difference.

Also note that there is a subtle difference between reactions of π -calculus processes and reactions in our π -calculus with explicit substitutions (and in the encoding). Reaction in the π -calculus, $P \rightarrow_{\pi} P'$, can reduce the set of free names, while this set is preserved in the environment, and hence remains fixed in our encoding. Actually, the same situation occurs in Milner and Jensen's encoding of the asynchronous π -calculus (without replication and summation) as a bigraphical reactive system [14].

Finally, note how the continuation passing style of the encoding facilitates the encoding of synchronous communication. We send the continuation $\llbracket P \rrbracket$ down to the *Send* and the *Receive* part of a name in the encodings of the output and the input prefix, respectively. The received continuation is then placed behind the actual synchronisation.

4.1 Some Examples

In this section we present two examples of our encoding. First, a simple example showing how the encoding can simulate simple reactions in the π -calculus, followed by an example containing both restriction and replication.

Example 3 *As an example of how our encoding properly simulates reactions in the π -calculus with explicit substitutions, we look at the process*

$$\begin{aligned} id_A \vdash \bar{n}\langle m \rangle . m(d) . P \mid n(f) . \bar{f}\langle e \rangle . Q &\rightarrow_{\pi\sigma} \\ id_A[f \mapsto m] \vdash m(d) . P \mid \bar{f}\langle e \rangle . Q &\rightarrow_{\pi\sigma} \\ id_A[f \mapsto m][d \mapsto e] \vdash P \mid Q &, \end{aligned}$$

where A is the set of free names in $\bar{n}\langle m \rangle . m(d) . P \mid n(f) . \bar{f}\langle e \rangle . Q$. Letting $r = \Pi_{n \in A} n' \langle \llbracket n \rrbracket \rangle$, we get the following reductions in Homer

$$\begin{aligned}
& \llbracket \bar{n}\langle m \rangle . m(d) . P \rrbracket \parallel \llbracket n(f) . \bar{f}\langle e \rangle . Q \rrbracket \parallel r && \searrow^* \\
& \bar{n} \langle \llbracket m \rrbracket \rangle . \llbracket m(d) . P \rrbracket \parallel (f')(n(z) . (a)(\llbracket \bar{f}\langle e \rangle . Q \rrbracket_{z/f})) \parallel r && \searrow^* \\
& (f')(\llbracket m(d) . P \rrbracket \parallel \llbracket \bar{f}\langle e \rangle . Q \rrbracket \parallel f' \langle \llbracket m \rrbracket \rangle \parallel r) && \searrow^* \\
& (f')(d')(\llbracket P \rrbracket \parallel \llbracket Q \rrbracket \parallel d' \langle \llbracket e \rrbracket \rangle \parallel f' \langle \llbracket m \rrbracket \rangle \parallel r) = \llbracket id_A[f \mapsto m][d \mapsto e] \vdash P \mid Q \rrbracket ,
\end{aligned}$$

where $\llbracket \bar{f}\langle e \rangle . Q \rrbracket_{z/f} = a \langle Bind_f \rangle \parallel \bar{a}v(z) . \bar{a}(z') . (\llbracket \bar{f}\langle e \rangle . Q \rrbracket \parallel z')$ is the part of the $Receive_n$ process that, with help from $Bind_f$, binds the received name to the local name f and then runs the continuation $\llbracket \bar{f}\langle e \rangle . Q \rrbracket$.

Example 4 One of the strengths of the π -calculus is the creation of local names with static scope. Here we consider the following example, where we have a replicated output of a local name and two recipients

$$!(\nu n)\bar{m}\langle n \rangle . R \mid m(o) . P \mid m(i) . Q .$$

Letting A denote the set of free names and letting n_1, n_2, m_1 , and m_2 all be fresh names with respect to the entire expression, we can get the following reactions

$$\begin{aligned}
& id_A \vdash !(\nu n)\bar{m}\langle n \rangle . R \mid m(o) . P \mid m(i) . Q \rightarrow_{\pi\sigma}^* \\
& id_A[n_1 \mapsto m_1][o \mapsto m_1] \vdash !(\nu n)\bar{m}\langle n \rangle . R \mid R_1 \mid P \mid (\nu n)\bar{m}\langle n \rangle . R \mid m(i) . Q \rightarrow_{\pi\sigma} \\
& id_A' \vdash !(\nu n)\bar{m}\langle n \rangle . R \mid R_1 \mid P \mid R_2 \mid Q ,
\end{aligned}$$

where $id_A' = id_A[n_1 \mapsto m_1][o \mapsto m_1][n_2 \mapsto m_2][i \mapsto m_2]$, and R_1 and R_2 are R , where we have α -converted the previously bound name n to n_1 and n_2 , respectively.

So the processes, P and Q , each receive a local name, which they each share with an instance of the process R . Letting r denote the process $\Pi_{n \in A} n' \langle \llbracket n \rrbracket \rangle$, we can match this in Homer using several reactions. Below we interleave the reactions with explanations of the main points.

$$\begin{aligned}
& \llbracket !(\nu n)\bar{m}\langle n \rangle . R \rrbracket \parallel \llbracket m(o) . P \rrbracket \parallel \llbracket m(i) . Q \rrbracket \parallel r && \searrow^* \\
& \llbracket !(\nu n)\bar{m}\langle n \rangle . R \rrbracket \parallel (o')(m(z) . (a)(\llbracket P \rrbracket_{z/o})) \parallel (i')(m(z) . (a)(\llbracket Q \rrbracket_{z/i})) \parallel r && \searrow^* \\
& \llbracket !(\nu n)\bar{m}\langle n \rangle . R \rrbracket \parallel \llbracket (\nu n)\bar{m}\langle n \rangle . R \rrbracket \parallel \llbracket (\nu n)\bar{m}\langle n \rangle . R \rrbracket \\
& \parallel (o')(m(z) . (a)(\llbracket P \rrbracket_{z/o})) \parallel (i')(m(z) . (a)(\llbracket Q \rrbracket_{z/i})) \parallel r && =
\end{aligned}$$

First we use the encoding of replication and input prefixes, and make several

reactions to activate the input prefixes and to unfold the replication twice.

$$\begin{aligned}
& !\llbracket(\nu n)\overline{m}\langle n \rangle . R\rrbracket \parallel (n)(n')\left(\llbracket\overline{m}\langle n \rangle . R\rrbracket \parallel n'\langle\llbracket n \rrbracket\rangle\right) \parallel \llbracket(\nu n)\overline{m}\langle n \rangle . R\rrbracket \\
& \parallel (\sigma')(m(z) . (a)(\llbracket P \rrbracket_{z/o})) \parallel (i')(m(z) . (a)(\llbracket Q \rrbracket_{z/i})) \parallel r \quad \searrow^* \\
& !\llbracket(\nu n)\overline{m}\langle n \rangle . R\rrbracket \parallel (n)\left((n')(\llbracket R \rrbracket \parallel n'\langle\llbracket n \rrbracket\rangle) \parallel (\sigma')(\llbracket P \rrbracket \parallel \sigma'\langle\llbracket n \rrbracket\rangle)\right) \parallel \\
& \llbracket(\nu n)\overline{m}\langle n \rangle . R\rrbracket \parallel (i')(m(z) . (a)(\llbracket Q \rrbracket_{z/i})) \parallel r \quad \searrow^*
\end{aligned}$$

Then we utilise the encoding of π -calculus restriction, followed by several reactions to match the synchronisation between $\overline{m}\langle n_1 \rangle . R$ and $m(o) . P$, under the restriction of n_1 .

$$\begin{aligned}
& !\llbracket(\nu n)\overline{m}\langle n \rangle . R\rrbracket \parallel (n)\left((n')(\llbracket R \rrbracket \parallel n'\langle\llbracket n \rrbracket\rangle) \parallel (\sigma')(\llbracket P \rrbracket \parallel \sigma'\langle\llbracket n \rrbracket\rangle)\right) \parallel \\
& (n)\left((n')(\llbracket R \rrbracket \parallel n'\langle\llbracket n \rrbracket\rangle) \parallel (i')(\llbracket Q \rrbracket \parallel i'\langle\llbracket n \rrbracket\rangle)\right) \parallel r
\end{aligned}$$

Finally, we can match the same kind of synchronisation again, this time between $\overline{m}\langle n_2 \rangle . R$ and $m(i) . Q$, under the restriction of n_2 . By applying α -conversion to the four local names n and n' , we can achieve the desired correspondence.

$$\begin{aligned}
& !\llbracket(\nu n)\overline{m}\langle n \rangle . R\rrbracket \parallel (m_1)\left((n'_1)(\llbracket R_1 \rrbracket \parallel n'_1\langle\llbracket m_1 \rrbracket\rangle) \parallel (\sigma')(\llbracket P \rrbracket \parallel \sigma'\langle\llbracket m_1 \rrbracket\rangle)\right) \parallel \\
& (m_2)\left((n'_2)(\llbracket R_2 \rrbracket \parallel n'_2\langle\llbracket m_2 \rrbracket\rangle) \parallel (i')(\llbracket Q \rrbracket \parallel i'\langle\llbracket m_2 \rrbracket\rangle)\right) \parallel r
\end{aligned}$$

4.2 Encoding Matching

The π -calculus presented so far does not contain a matching operator. In this subsection, we describe the changes necessary to the encoding in order to encode a matching operator.

Recall, that the matching operator of the π -calculus, written $[n = m]P$, behaves as P if $n = m$, and as $\mathbf{0}$ otherwise. For simplicity, we assume that the grammar of process expressions has been extended with the clause $[n = m]P$, and that the reaction relation has been extended with the following rule

$$(\text{Match}\sigma) \frac{}{\sigma \vdash [n = m]P \rightarrow_{\pi\sigma} \sigma \vdash P} \quad , \text{ if } \sigma n = \sigma m.$$

We make two changes in the encoding to encode this. First, we augment the encoding of a name n with the following task:

$$\text{Match}_n = v(x) . c(y) . (a)(a\langle x \rangle \parallel \overline{an}(x') . \overline{a}(x'') . y)$$

The $Match_n$ task receives an encoding of a name and a continuation. The received name is placed at the local location a , in parallel with a process that wants to take up from location an , then discard the location a and continue as the received continuation.

In addition to the new task $Match_n$ we also add a marker, $n\langle\mathbf{0}\rangle$, to the encoding of a name n . We need the marker for testing if two names are equal. In the $Match_n$ task, where we wanted to take up from location an , we are actually testing if the marker is equal to n or not. After these additions the encoding of a name n has the following form:

$$\llbracket n \rrbracket = s\langle Send_n \rangle \parallel r\langle Receive_n \rangle \parallel m\langle Match_n \rangle \parallel n\langle \mathbf{0} \rangle .$$

Second, we also need to encode the matching operator

$$\llbracket [f = g]P \rrbracket = (a)(f' \triangleright x . (a\langle x \rangle \parallel g' \triangleright y . \overline{amv}\langle y \rangle . \overline{amc}\langle \llbracket P \rrbracket \rangle . \overline{am}(x') . \bar{a}(z) . x')) .$$

The encoding first takes a copy of the name at location f' and keeps the copy at the local address a . Then it takes a copy of the name at location g' , and sends this copy down to the $Match$ part of the name at a together with the continuation $\llbracket P \rrbracket$. The $Match$ part is then retrieved, and the location a is discarded. If the names f and g are equal, then the retrieved $Match$ part can make one synchronisation, and the continuation can be run. Otherwise, if f and g are distinct names, then the process will be blocked and hence behave as $\mathbf{0}$.

4.3 Encoding without Auxiliary Names

As mentioned in the beginning of Section 4 we can get rid of the set of names \mathcal{N}' and $\{v, c, s, r\}$ in the encoding, at the expense of readability.

A first observation is that the set of names \mathcal{N}' is superfluous. We can keep the encoding of a name $\llbracket n \rrbracket$ at the address n instead of the address n' . If one look closely at the occurrences of the *resource* prefixes, these prefixes only appear as addresses for encodings of names or as local addresses. Furthermore, the *take* prefixes only take from local addresses, and the addresses containing an encoding of a π -calculus name using the shorthand $n \triangleright x$.

Also note, that we cannot send anything down to the encoding of a name, except when the encoding is a copy kept in a local address. The only *send* prefix that is not restricted to local addresses is the synchronisation in Homer, representing the actual synchronisation in the π -calculus, but this send prefix cannot interfere with our encoding, since the address in the send prefix is only

of length 1. So this prefix cannot send anything down into a location, and hence cannot send anything down to the encoding of a name.

We need several observations and a single technical trick for removing the auxiliary names $\{v, c, s, r\}$. First, we observe that there is no need for both the names v and c . The reason why one name is sufficient is that the only communication with the encoding of a name happens sequentially, which is enforced by keeping the name in a local address only available to one process. So the merging of the names v and c will not change the behaviour of the encoding. We will from this point on only consider the auxiliary set of names $\{v, s, r\}$, where we have merged v and c into v .

We can encode these three remaining auxiliary names using sequences of different lengths of a single name occurring in the π -calculus expression. We represent s, r, v , so that $|\delta| \neq |\delta'|$ for $\delta \neq \delta' \in \{s, r, sv, rv\}$. One solution is to take $|s| = 2, |r| = 1$, and $|v| = 2$. So for example the encoding of an output prefix

$$\llbracket \bar{n}\langle m \rangle . P \rrbracket = (a)(n' \triangleright x . (a\langle x \rangle \parallel m' \triangleright y . \overline{asv}\langle y \rangle . \overline{asc}\langle \llbracket P \rrbracket \rangle . \overline{as}\langle x'' \rangle . \bar{a}\langle z \rangle . x''))$$

is changed into the following, using n as the element of our sequences, and changing n' and m' into n and m , respectively

$$\llbracket \bar{n}\langle m \rangle . P \rrbracket = (a)(n \triangleright x . (a\langle x \rangle \parallel m \triangleright y . \overline{annnn}\langle y \rangle . \overline{annnn}\langle \llbracket P \rrbracket \rangle . \overline{ann}\langle x'' \rangle . \bar{a}\langle z \rangle . x'')) ,$$

and the *Send* part of the encoded name $\llbracket n \rrbracket$ is changed into

$$Send_n = nn(x) . nn(y) . \bar{n}\langle x \rangle . y .$$

See Section A in the Appendix for the complete encoding without auxiliary names.

5 Proof of Correspondence

In this section we prove the soundness of the encoding. Following the approach in [11], we first show that there is an operational correspondence between a π -calculus process $\sigma \vdash P$ and its encoding $\llbracket \sigma \vdash P \rrbracket$ in Homer. From this we can deduce that the encoding is fully abstract with respect to barbed bisimulation, and sound with respect to the barbed congruence.

We define strong barbs in a π -calculus term P with explicit substitution σ as $\sigma \vdash P \downarrow n$, if P can perform an input action on a name, that maps to the free name n , and we define weak barbs in terms of strong barbs.

$$\begin{aligned} \sigma \vdash P \downarrow n, & \quad \text{if } \sigma \vdash P \equiv_{\pi\sigma} m(o) . P' \mid Q, \text{ where } \sigma m = n \text{ and } \sigma n = n, \text{ and} \\ \sigma \vdash P \Downarrow n, & \quad \text{if } \exists P', \sigma'. \sigma \vdash P \rightarrow_{\pi\sigma}^* \sigma' \vdash P' \text{ and } \sigma' \vdash P' \downarrow n . \end{aligned}$$

Correspondingly in Homer, we write $p \downarrow n$, if p can perform an unrestricted receive action $n(m)$ at top-level, and define $p \Downarrow n$ in terms of $p \downarrow n$.

$$\begin{aligned} p \downarrow n & \quad \text{if } p \equiv (\tilde{n})(n(m) . p' \parallel q), \text{ where } n \notin \tilde{n}, \text{ and} \\ p \Downarrow n & \quad \text{if } \exists p'. p \searrow^* p' \text{ and } p' \downarrow n . \end{aligned}$$

We then define a *matching barbed bisimilarity* between π -calculus and Homer terms.

Definition 5 Matching barbed bisimilarity is the largest relation $\dot{\approx} \subseteq \mathcal{P}_{\pi\sigma/\alpha} \times \mathcal{P}_{c/\alpha}$, if whenever $(\sigma \vdash P, q) \in \dot{\approx}$,

- if $\sigma \vdash P \rightarrow_{\pi\sigma} \sigma' \vdash P'$ then there exists q' such that $q \searrow^* q'$ and $\sigma' \vdash P' \dot{\approx} q'$
- if $q \searrow q'$ then there exists P' and σ' such that $\sigma \vdash P \rightarrow_{\pi\sigma}^* \sigma' \vdash P'$ and $\sigma' \vdash P' \dot{\approx} q'$
- if $\sigma \vdash P \downarrow n$ then $q \Downarrow n$
- if $q \downarrow n$ then $\sigma \vdash P \Downarrow n$

The main result of this paper is that there is a matching barbed bisimulation between the encoding and the encoded process.

Theorem 6 For all π -calculus processes P and substitutions σ , we have $\sigma \vdash P \dot{\approx} \llbracket \sigma \vdash P \rrbracket$.

Definition 7 Barbed bisimilarity in Homer is the largest symmetric relation $\dot{\approx}_H$ on $\mathcal{P}_{c/\alpha}$, such that whenever $p \dot{\approx}_H q$,

- if $p \downarrow n$ then $q \Downarrow n$
- if $p \searrow p'$ then there exists q' such that $q \searrow^* q'$ and $p' \dot{\approx}_H q'$

Definition 8 We define barbed congruence in Homer as $p \approx_H q$, if $\mathcal{C}_H(p) \dot{\approx}_H \mathcal{C}_H(q)$ for every Homer context \mathcal{C}_H .

From Theorem 6 it follows that the encoding is fully abstract with respect to barbed bisimulation.

Corollary 9 *Let \approx_π denote the standard barbed bisimilarity in π -calculus, then we have $\sigma \vdash P \approx_\pi \sigma \vdash Q$ if and only if $\llbracket \sigma \vdash P \rrbracket \approx_H \llbracket \sigma \vdash Q \rrbracket$.*

We can prove soundness with respect to barbed congruence from the compositionality of the encoding.

Theorem 10 *Letting \mathcal{C}_H denote a Homer context, \mathcal{C}_π a π -calculus context, we have that $\llbracket \sigma \vdash P \rrbracket \approx_H \llbracket \sigma \vdash Q \rrbracket$ implies $\forall \mathcal{C}_\pi \forall \sigma'. \sigma \sigma' \vdash \mathcal{C}_\pi(P) \approx_\pi \sigma \sigma' \vdash \mathcal{C}_\pi(Q)$, such that $\text{dom } \sigma' \supseteq \text{fn}(\mathcal{C}_\pi) \setminus \text{dom } \sigma$ and $\text{dom } \sigma' \cap \text{dom } \sigma = \emptyset$.*

6 Conclusions and Future Work

We have presented a novel encoding of π -calculus name-passing and name-substitution in the calculus Homer, using process-passing, mobile computing resources, named nested locations and local names. We have used a continuation passing style to give an elegant encoding of synchronous communication. We have introduced a π -calculus with explicit substitutions to maintain the set of free names under reaction, and for the purpose of making the correspondence intuitive. The encoding extends the one in [15] to include replication and name-matching.

Several interesting questions arise from the work done in this paper. First and foremost, a logical next step would be to see if it is possible to encode a version of Homer extended with name-passing in Homer. It is not clear at this point, how to make an encoding like this, or if it is possible at all. Second, it would be interesting to look for a completeness result for the encoding with respect to barbed congruence. As mentioned in [11], this is a difficult problem for synchronous calculi. A possible solution could be to use a labelled bisimulation characterisation of barbed congruence in both the π -calculus and in Homer. We explore labelled bisimulation congruence in Homer in [6].

Finally, we are currently examining an alternative encoding of the π -calculus in Homer, inspired by the encoding in section 13.3 of [11] by Sangiorgi and Walker. The encoding is direct, in the sense that it does not utilise an intermediate π -calculus with explicit substitutions. Instead we use that we can represent abstractions and applications in Homer as described in the introduction.

Acknowledgements Many thanks to the anonymous referees for their suggestions and comments from which this paper has benefited greatly.

References

- [1] R. Milner, J. Parrow, D. Walker, A calculus of mobile processes, parts I and II, *Journal of Information and Computation* 100 (1992) 1–40 and 41–77.
- [2] R. Milner, *Communicating and Mobile Systems: the π -calculus*, Cambridge University Press, 1999.
- [3] L. Cardelli, A. D. Gordon, Mobile ambients, *Theoretical Computer Science* 240 (1) (2000) 177–213.
URL [http://dx.doi.org/10.1016/S0304-3975\(99\)00231-5](http://dx.doi.org/10.1016/S0304-3975(99)00231-5)
- [4] G. Castagna, F. Z. Nardelli, The Seal calculus revisited: Contextual equivalence and bisimilarity, in: M. Agrawal, A. Seth (Eds.), *Proceedings of the 22nd Conference on the Foundations of Software Technology and Theoretical Computer Science (FSTTCS'02)*, Vol. 2556 of *Lecture Notes in Computer Science*, Springer Verlag, 2002, pp. 85–96.
- [5] B. Thomsen, Plain CHOCS: A second generation calculus for higher order processes, *Acta Informatica* 30 (1) (1993) 1–59.
- [6] T. Hildebrandt, J. C. Godskesen, M. Bundgaard, Bisimulation congruences for Homer — a calculus of higher order mobile embedded resources, *Tech. Rep. TR-2004-52*, IT University of Copenhagen (2004).
- [7] J. C. Godskesen, T. Hildebrandt, V. Sassone, A calculus of mobile resources, in: L. Brim, P. Jancar, M. Kretínský, A. Kucera (Eds.), *Proceedings of the 13th International Conference on Concurrency Theory (CONCUR'02)*, Vol. 2421 of *Lecture Notes in Computer Science*, Springer Verlag, 2002, pp. 272–287.
- [8] P. Zimmer, On the expressiveness of pure mobile ambients, *Journal of Mathematical Structures in Computer Science* 13 (5) (2004) 721–770.
- [9] G. Ferrari, U. Montanari, P. Quaglia, A π -calculus with explicit substitutions, *Theoretical Computer Science* 168 (1) (1996) 53–103.
- [10] D. Sangiorgi, Expressing mobility in process algebras: First-order and higher-order paradigms, Ph.D. thesis, Department of Computer Science, University of Edinburgh (1992).
- [11] D. Sangiorgi, D. Walker, *The π -calculus*, Cambridge University Press, 2001.
- [12] M. Carbone, S. Maffei, On the expressive power of polyadic synchronisation in π -calculus, *Nordic Journal of Computing* 10 (2) (2003) 70–98.
- [13] D. Hirschhoff, Handling substitutions explicitly in the π -calculus, in: *Proceedings of Second International Workshop on Explicit Substitutions: Theory and Applications to Programs and Proofs (WESTAPP'99)*, 1999, pp. 28–43.
URL <http://cermics.enpc.fr/~dh/sigma/full.ps.gz>

- [14] O. H. Jensen, R. Milner, Bigraphs and mobile processes (revised), Tech. Rep. UCAM-CL-TR-580, University of Cambridge, Computer Laboratory (2004).
- [15] M. Bundgaard, T. Hildebrandt, J. C. Godskesen, A CPS encoding of name-passing in higher-order mobile embedded resources, in: J. Baeten, F. Corradini (Eds.), Proceedings of the 11th International Workshop on Expressiveness in Concurrency (EXPRESS'04), Vol. 128 of Electronic Notes in Theoretical Computer Science, Elsevier, 2004, pp. 131–150.

A Encoding without Auxiliary Names

As before, we encode a π -calculus name n as a Homer process $\llbracket n \rrbracket$ that can perform two tasks: sending and receiving along the name n . As mentioned in Section 4.3 we encode the auxiliary name s as a sequence of n 's of length 2, r is encoded as a sequence of length 1, and c (and v) as a sequence of length 2.

$$\begin{aligned}
Send_n &= nn(x) . nn(y) . \bar{n}\langle x \rangle . y \\
Receive_n &= nn(x) . nn(y) . n(z) . (a)(a\langle x \rangle \parallel \overline{ann}\langle z \rangle . \bar{a}\langle z' \rangle . (y \parallel z')) \\
\llbracket n \rrbracket &= nn\langle Send_n \rangle \parallel n\langle Receive_n \rangle
\end{aligned}$$

We keep the name n as a resource at the address n instead of the address n' , in this variant of the encoding.

$$\begin{aligned}
\llbracket (\nu n)P \rrbracket &= (n)(\llbracket P \rrbracket \parallel n\langle \llbracket n \rrbracket \rangle) \\
\llbracket P \mid Q \rrbracket &= \llbracket P \rrbracket \parallel \llbracket Q \rrbracket \\
\llbracket !P \rrbracket &= !\llbracket P \rrbracket \\
\llbracket \mathbf{0} \rrbracket &= \mathbf{0}
\end{aligned}$$

We encode the output and input prefixes of the π -calculus as follows

$$\begin{aligned}
\llbracket \bar{n}\langle m \rangle . P \rrbracket &= (a)(n \triangleright x . (a\langle x \rangle \parallel m \triangleright y . \overline{annnn}\langle y \rangle . \overline{annnn}\langle \llbracket P \rrbracket \rangle . \overline{ann}\langle x'' \rangle . \bar{a}\langle z \rangle . x'')) \\
\llbracket n(m) . P \rrbracket &= (a)(n \triangleright x . (m)(a\langle x \rangle \parallel \overline{annn}\langle Bind_m \rangle . \overline{annn}\langle \llbracket P \rrbracket \rangle . \overline{an}\langle x'' \rangle . \bar{a}\langle z \rangle . x'')) ,
\end{aligned}$$

where $Bind_m = nn(x) . m\langle x \rangle$ and the shorthand $n \triangleright x$ is defined as before. As before, the process $Bind_m$ is responsible for binding the received name to the local name m .

B Reductions of Prefixes

As explained in section 4, our encoding requires several reaction steps in Homer in order to mimic a single reaction in the π -calculus. We call the step that corresponds to a reaction in the π -calculus the *reaction step*, and the additional steps *bookkeeping steps*.

From the encoding we see that for an output prefix there are only bookkeeping steps before the reaction step, whereas for an input prefix there are also bookkeeping steps after the reaction step, since we need to bind the received name.

We index our encoding of a π -calculus prefix to capture the intuition that a π -calculus prefix corresponds to a sequence of Homer processes. We take index 0 to be the original translation of the prefix (before any bookkeeping steps have occurred in the encoding), i.e. $\llbracket \bar{n}\langle m \rangle . P \rrbracket_\sigma^0 = \llbracket \bar{n}\langle m \rangle . P \rrbracket$, and then define $\llbracket \bar{n}\langle m \rangle . P \rrbracket_\sigma^k$ such that

$$\llbracket \bar{n}\langle m \rangle . P \rrbracket_\sigma^0 \searrow \dots \searrow \llbracket \bar{n}\langle m \rangle . P \rrbracket_\sigma^6 = \overline{\sigma n}\langle \llbracket \sigma m \rrbracket \rangle . \llbracket P \rrbracket .$$

More precisely, given a substitution σ , we define $\llbracket \bar{n}\langle m \rangle . P \rrbracket_\sigma^k$, for $0 \leq k \leq 6$, as

$$\begin{aligned} \llbracket \bar{n}\langle m \rangle . P \rrbracket_\sigma^0 &= \llbracket \bar{n}\langle m \rangle . P \rrbracket \\ \llbracket \bar{n}\langle m \rangle . P \rrbracket_\sigma^1 &= (a)(a\langle \llbracket \sigma n \rrbracket \rangle \parallel \underline{m'} \triangleright y . \underline{\overline{a s v}}\langle y \rangle . \underline{\overline{a s c}}\langle \llbracket P \rrbracket \rangle . \underline{\overline{a s}}(x'') . \underline{\bar{a}}(z) . x'') \\ \llbracket \bar{n}\langle m \rangle . P \rrbracket_\sigma^2 &= (a)(a\langle \llbracket \sigma n \rrbracket \rangle \parallel \underline{\overline{a s v}}\langle \llbracket \sigma m \rrbracket \rangle . \underline{\overline{a s c}}\langle \llbracket P \rrbracket \rangle . \underline{\overline{a s}}(x'') . \underline{\bar{a}}(z) . x'') \\ \llbracket \bar{n}\langle m \rangle . P \rrbracket_\sigma^3 &= (a)(a\langle \underline{s}\langle \underline{c}(y) . \underline{\overline{\sigma n}}\langle \llbracket \sigma m \rrbracket \rangle} . y \parallel r\langle \underline{Receive}_{\sigma n} \rangle \rangle \\ &\quad \parallel \underline{\overline{a s c}}\langle \llbracket P \rrbracket \rangle . \underline{\overline{a s}}(x'') . \underline{\bar{a}}(z) . x'') \\ \llbracket \bar{n}\langle m \rangle . P \rrbracket_\sigma^4 &= (a)(a\langle \underline{s}\langle \underline{\overline{\sigma n}}\langle \llbracket \sigma m \rrbracket \rangle} . \llbracket P \rrbracket \parallel r\langle \underline{Receive}_{\sigma n} \rangle \rangle \parallel \underline{\overline{a s}}(x'') . \underline{\bar{a}}(z) . x'') \\ \llbracket \bar{n}\langle m \rangle . P \rrbracket_\sigma^5 &= (a)(a\langle \underline{r}\langle \underline{Receive}_{\sigma n} \rangle \rangle \parallel \underline{\bar{a}}(z) . \underline{\overline{\sigma n}}\langle \llbracket \sigma m \rrbracket \rangle . \llbracket P \rrbracket) \\ \llbracket \bar{n}\langle m \rangle . P \rrbracket_\sigma^6 &= \underline{\overline{\sigma n}}\langle \llbracket \sigma m \rrbracket \rangle . \llbracket P \rrbracket , \end{aligned}$$

where we have underlined the synchronising prefixes and emphasised some of the resource boundaries for readability. Notice that in index 0 we need a location n' in the environment σ , in index 1 a location m' , and in index 6 we need a matching co-action in the environment, in order to be able to perform the reaction.

Similarly, we define $\llbracket n(m) . P \rrbracket_\sigma^k$, for $0 \leq k \leq 5$, as

$$\begin{aligned}
\llbracket n(m) . P \rrbracket_\sigma^0 &= \llbracket n(m) . P \rrbracket \\
\llbracket n(m) . P \rrbracket_\sigma^1 &= (a)(m')(a(\llbracket \sigma n \rrbracket) \parallel \overline{a\bar{v}}\langle \text{Bind}_m \rangle . \overline{a\bar{r}c}\langle \llbracket P \rrbracket \rangle . \overline{a\bar{r}}(x'') . \bar{a}(z) . x'') \\
\llbracket n(m) . P \rrbracket_\sigma^2 &= (a)(m')(a\langle s\langle \text{Send}_{\sigma n} \rangle \parallel r\langle \underline{c}(y) \rangle . \sigma n(z) . (a)(a\langle \text{Bind}_m \rangle \parallel \\
&\quad \overline{a\bar{v}}\langle z \rangle . \bar{a}(z') . (y \parallel z')) \rangle \parallel \overline{a\bar{r}c}\langle \llbracket P \rrbracket \rangle . \overline{a\bar{r}}(x'') . \bar{a}(z) . x'') \\
\llbracket n(m) . P \rrbracket_\sigma^3 &= (a)(m')(a\langle s\langle \text{Send}_{\sigma n} \rangle \parallel \\
&\quad \underline{r\langle \sigma n(z) . (a)(a\langle \text{Bind}_m \rangle \parallel \overline{a\bar{v}}\langle z \rangle . \bar{a}(z') . (\llbracket P \rrbracket \parallel z')) \rangle} \rangle \parallel \\
&\quad \overline{a\bar{r}}(x'') . \bar{a}(z) . x'') \\
\llbracket n(m) . P \rrbracket_\sigma^4 &= (a)(m')(a\langle s\langle \text{Send}_{\sigma n} \rangle \rangle \parallel \underline{\bar{a}(z) . \sigma n(z) . (a)(a\langle \text{Bind}_m \rangle \parallel \\
&\quad \overline{a\bar{v}}\langle z \rangle . \bar{a}(z') . (\llbracket P \rrbracket \parallel z'))}) \\
\llbracket n(m) . P \rrbracket_\sigma^5 &= (m')(\underline{\sigma n(z) . (a)(\llbracket P \rrbracket_{z/m})}) \\
\llbracket P \rrbracket_{e/m}^0 &= (m')(a)(a\langle \text{Bind}_m \rangle \parallel \overline{a\bar{v}}\langle \llbracket e \rrbracket \rangle . \bar{a}(z') . (\llbracket P \rrbracket \parallel z')) \\
\llbracket P \rrbracket_{e/m}^1 &= (m')(a)(a\langle m' \langle \llbracket e \rrbracket \rangle \rangle \parallel \underline{\bar{a}(z') . (\llbracket P \rrbracket \parallel z')}) ,
\end{aligned}$$

where $\llbracket P \rrbracket_{z/m} = a\langle \text{Bind}_m \rangle \parallel \overline{a\bar{v}}\langle z \rangle . \bar{a}(z') . (\llbracket P \rrbracket \parallel z')$. Again, notice that in index 0 we need a location n' in the environment, and in index 5 we need a matching co-action in the surrounding environment. $\llbracket P \rrbracket_{e/m}^0$ and $\llbracket P \rrbracket_{e/m}^1$ represent the steps needed for binding the received name (to the local name m). In this case the encoding of the name e has been input.

C Proof of Correspondence

We consider π -calculus processes in a (pre) normal form in order to ease the proof.

Definition 11 *A normal form for a π -calculus process P is defined as follows*

$$P = (\nu \tilde{n}) (I_1 \mid \cdots \mid I_k \mid O_1 \mid \cdots \mid O_m \mid !P_1'' \mid \cdots \mid !P_n'') ,$$

where each I_i is on the form $x(y) . P_i$ and each O_i is on the form $\bar{x}\langle y \rangle . P_i'$, both for some x, y , and where $P_1, \dots, P_k, P_1', \dots, P_m'$, and P_1'', \dots, P_n'' are in normal form.

As a standard result we have that every π -calculus term can be rewritten to a term in normal form using the rules for structural congruence.

Proposition 12 *Every π -calculus term P is structurally congruent to a term P' in normal form.*

However, we need to weaken our notion of normal form, since we in our encoding can have input prefixes which have participated in a synchronisation, but which have not performed the following booking with respect to binding. Hence we introduce a *prenormal form*, where we explicit represent these input prefixes.

Definition 13 *A prenormal form for a π -calculus process P is defined as follows*

$$P = (\nu\tilde{n})\left(I_1 \mid \cdots \mid I_k \mid O_1 \mid \cdots \mid O_m \mid !P_1 \mid \cdots \mid !P_n \mid P'_1 \mid \cdots \mid P'_l\right), \quad (\star)$$

where each I_i is on the form $x(y) . P''_i$, and each O_i is on the form $\bar{x}\langle y \rangle . P'''_i$, for some names x, y , and where $P''_1, \dots, P''_k, P'''_1, \dots, P'''_m, P_1, \dots, P_n$, and P'_1, \dots, P'_l are in normal form.

The processes P'_1, \dots, P'_l will correspond to input prefixes in the encoding which have received a value, but have not completed the following bookkeeping in the binding of a name. We can now define a prenormal form for our π -calculus with explicit substitutions.

Definition 14 *A prenormal form for a π -calculus process P on the prenormal form (\star) and an environment σ is defined as follows*

$$\sigma[\bar{n} \mapsto \bar{m}] \vdash P = I_1 \mid \cdots \mid I_k \mid O_1 \mid \cdots \mid O_m \mid !P_1 \mid \cdots \mid !P_n \mid P'_1 \mid \cdots \mid P'_l, \quad (\star\star)$$

where \bar{n} is the set \tilde{n} ordered as a sequence, and where all names m in the sequence \bar{m} are chosen fresh.

So a prenormal form for a π -calculus process with explicit substitutions $\sigma \vdash P$ is the underlying prenormal form for P , except that we have lifted the restrictions to the environment using structural congruence. We will write $[\tilde{n} \mapsto \bar{m}]$ for the result of lifting the restriction $(\nu\tilde{n})$ to the environment, where all elements in \bar{m} are chosen fresh.

It is easy to prove that if $\sigma \vdash P$ is on prenormal form and $\sigma \vdash P \rightarrow_{\pi\sigma} \sigma' \vdash P'$, then there exists a $\sigma'' \vdash P'' \equiv_{\pi\sigma} \sigma' \vdash P'$, and $\sigma'' \vdash P''$ is on prenormal form $(\star\star)$. Hence we need only to consider processes on prenormal form, both in π -calculus with explicit substitutions and their encodings in Homer, and we do not lose any behaviour because of this.

We then translate processes on prenormal form ($\star\star$) into the following Homer processes

$$\llbracket \sigma \vdash P \rrbracket^{(\bar{i}, \bar{o}, \bar{p}, \bar{e}, \bar{m})} = (\tilde{m}) \left(\llbracket I_1 \rrbracket_\sigma^{i_1} \parallel \cdots \parallel \llbracket I_k \rrbracket_\sigma^{i_k} \parallel \llbracket O_1 \rrbracket_\sigma^{o_1} \parallel \cdots \parallel \llbracket O_m \rrbracket_\sigma^{o_m} \parallel \right. \\ \left. \begin{array}{l} \llbracket P_1 \rrbracket_\sigma \parallel \cdots \parallel \llbracket P_n \rrbracket_\sigma \parallel \llbracket P_1 \rrbracket_{e_1/m_1}^{p_1} \parallel \cdots \parallel \llbracket P_l \rrbracket_{e_l/m_l}^{p_l} \parallel r \end{array} \right), \quad (\star\star\star)$$

where $r = \prod_{n \in \text{dom } \sigma} n' \langle \llbracket \sigma n \rrbracket \rangle$ and $\tilde{m} = \{m' \mid m \in \text{dom } \sigma \text{ and } m \neq \sigma m\} \cup \{n \mid n \in \text{codom } \sigma \setminus \text{dom } \sigma\}$, i.e. the restriction of the local names and the restricted names. We introduced the notation $\llbracket - \rrbracket^{(\bar{i}, \bar{o}, \bar{p}, \bar{e}, \bar{m})}$, where

- \bar{i} is the list of indexes of the input prefixes $\langle i_1, \dots, i_k \rangle$ (e.g. for each i_j , $0 \leq i_j \leq 6$),
- \bar{o} is the list of indexes of the output prefixes $\langle o_1, \dots, o_m \rangle$ (e.g. for each o_j , $0 \leq o_j \leq 5$),
- \bar{p} is the list of indexes of the preforms $\langle p_1, \dots, p_l \rangle$ (e.g. for each p_j , $0 \leq p_j \leq 1$), and the lists \bar{e} and \bar{m} are lists of names, consisting of the names received in the input and the local names, respectively. Notice that the length of the lists \bar{p} , \bar{e} , and \bar{m} must be the same, l .

Proof of Theorem 6 (Sketch) Define a relation \mathcal{R} , such that for all π -calculus processes P on the form of ($\star\star$), all substitutions σ , and for all possible lists $\bar{i}, \bar{o}, \bar{p}, \bar{e}, \bar{m}$ (with respect to P)

$$\sigma[\bar{m} \mapsto \bar{e}] \vdash P \mathcal{R} \llbracket \sigma \vdash P \rrbracket^{(\bar{i}, \bar{o}, \bar{p}, \bar{e}, \bar{m})}.$$

We show that \mathcal{R} is a matching barbed bisimulation. We prove each of the four conditions of Definition 5 separately. In the following cases assume that we have taken an arbitrary pair from \mathcal{R} (e.g. $\sigma[\bar{m} \mapsto \bar{e}] \vdash P \mathcal{R} \llbracket \sigma \vdash P \rrbracket^{(\bar{i}, \bar{o}, \bar{p}, \bar{e}, \bar{m})}$).

First condition: There are 6 possible cases to consider here. The only type of synchronisation that can occur is between an output and an input prefix, and either of these can be in preform, we can unfold one of the replications, and we can use the rule (Rest σ). We consider one of these cases: the synchronisation where neither of the prefixes are in preform.

- We have an input prefix I_i , $1 \leq i \leq k$, of the form $n(o) . P'$, where o is chosen fresh, and an output prefix O_j , $1 \leq j \leq m$, of the form $\bar{h}\langle f \rangle . Q'$. Without loss of generality, assume that the forms of Q' and P' are

$$Q' = (\nu \tilde{n}_Q) \left(I_1^Q \mid \cdots \mid I_{k'}^Q \mid O_1^Q \mid \cdots \mid O_{m'}^Q \mid !P_1^Q \mid \cdots \mid !P_{n'}^Q \right) \text{ and} \\ P' = (\nu \tilde{n}_P) \left(I_1^P \mid \cdots \mid I_{k''}^P \mid O_1^P \mid \cdots \mid O_{m''}^P \mid !P_1^P \mid \cdots \mid !P_{n''}^P \right),$$

and that $\sigma n = \sigma h$, $\sigma f = g$. After the synchronisation, we have the following

expression on prenormal form

$$\begin{aligned}
\sigma' \vdash & I_1 \mid \cdots \mid I_{i-1} \mid I_{i+1} \mid \cdots \mid I_k \mid I_1^Q \mid \cdots \mid I_{k'}^Q \mid I_1^P \mid \cdots \mid I_{k''}^P \mid \\
& O_1 \mid \cdots \mid O_{j-1} \mid O_{j+1} \mid \cdots \mid O_m \mid O_1^Q \mid \cdots \mid O_{m'}^Q \mid O_1^P \mid \cdots \mid \\
& O_{m''}^P \mid !P_1 \mid \cdots \mid !P_n \mid !P_1^Q \mid \cdots \mid !P_{n'}^Q \mid !P_1^P \mid \cdots \mid !P_{n''}^P \mid \\
& P_1 \mid \cdots \mid P_l ,
\end{aligned} \tag{C.1}$$

where $\sigma' = \sigma[\overline{m} \mapsto \overline{e}][o \mapsto g][\tilde{n}_P \mapsto \overline{n}_P][\tilde{n}_Q \mapsto \overline{n}_Q]$. In Homer, the output prefix is on the form $\llbracket \overline{h}\langle f \rangle . Q' \rrbracket_\sigma^{i'}$, so after $6 - i'$ reactions we have $\llbracket \overline{h}\langle f \rangle . Q' \rrbracket_\sigma^6 = \overline{\sigma h}\langle \llbracket \sigma f \rrbracket \rangle . \llbracket Q' \rrbracket$. For the input prefix we have $\llbracket n(o) . P' \rrbracket_\sigma^{j'}$, so after $5 - j'$ reactions we have $\llbracket n(o) . P' \rrbracket_\sigma^5 = (\sigma')(\sigma n(z) . (a)(\llbracket P' \rrbracket_{z/o}))$, and then we have the reactions

$$\begin{aligned}
& \llbracket \overline{h}\langle f \rangle . Q' \rrbracket_\sigma^6 \parallel \llbracket n(o) . P' \rrbracket_\sigma^5 \searrow \llbracket Q' \rrbracket \parallel \llbracket P' \rrbracket_{g/o}^0 \searrow \\
& \llbracket Q' \rrbracket \parallel \llbracket P' \rrbracket_{g/o}^1 \searrow (\sigma')(\llbracket Q' \rrbracket \parallel \llbracket P' \rrbracket \parallel \sigma' \langle \llbracket g \rrbracket \rangle) .
\end{aligned}$$

So we end up with the following Homer process

$$\begin{aligned}
(\tilde{m}) \Big(& \llbracket I_1 \rrbracket_\sigma \parallel \cdots \parallel \llbracket I_{i-1} \rrbracket_\sigma \parallel \llbracket I_{i+1} \rrbracket_\sigma \parallel \cdots \parallel \llbracket I_k \rrbracket_\sigma \parallel \llbracket I_1^Q \rrbracket_\sigma^0 \parallel \cdots \parallel \\
& \llbracket I_{k'}^Q \rrbracket_\sigma^0 \parallel \llbracket I_1^P \rrbracket_\sigma^0 \parallel \cdots \parallel \llbracket I_{k''}^P \rrbracket_\sigma^0 \parallel \llbracket O_1 \rrbracket_\sigma \parallel \cdots \parallel \llbracket O_{j-1} \rrbracket_\sigma \parallel \\
& \llbracket O_{j+1} \rrbracket_\sigma \parallel \cdots \parallel \llbracket O_m \rrbracket_\sigma \parallel \llbracket O_1^P \rrbracket_\sigma^0 \parallel \cdots \parallel \llbracket O_{m'}^P \rrbracket_\sigma^0 \parallel \llbracket O_1^Q \rrbracket_\sigma^0 \parallel \cdots \parallel \\
& \llbracket O_{m''}^Q \rrbracket_\sigma^0 \parallel \llbracket !P_1 \rrbracket_\sigma \parallel \cdots \parallel \llbracket !P_n \rrbracket_\sigma \parallel \llbracket !P_1^Q \rrbracket_\sigma \parallel \cdots \parallel \llbracket !P_{n'}^Q \rrbracket_\sigma \parallel \\
& \llbracket !P_1^P \rrbracket_\sigma \parallel \cdots \parallel \llbracket !P_{n''}^P \rrbracket_\sigma \parallel \llbracket [P_1]_{e_1/m_1} \rrbracket \parallel \cdots \parallel \llbracket [P_l]_{e_l/m_l} \rrbracket \parallel r \Big) ,
\end{aligned} \tag{C.2}$$

where \tilde{m} and r are defined as in $(\star\star\star)$, just with respect to σ' instead of σ . Note that we have left out the indices on the untouched prefixes, since these remain unchanged. The π -calculus process in (C.1) and the Homer process in (C.2) are related by \mathcal{R} .

Second condition: For the second condition there are again several cases to consider: either the reaction can come from an internal step of one of the components of the parallel composition (i.e. the bookkeeping steps of a prefix), the unfolding of one of the replications, or a synchronisation between an input on index 5 and an output on index 6. We present only the last of the cases.

- If the reduction came from a synchronisation between an input I_i on index 5 ($\llbracket l(m) . P' \rrbracket_\sigma^5$) and an output O_j on index 6 ($\llbracket \overline{l}\langle n \rangle . Q' \rrbracket_\sigma^6$), and assuming $\sigma l = \sigma l'$ and $\sigma n = g$. Then these can synchronise to become $\llbracket P' \rrbracket_{g/m}^0$ and $\llbracket Q' \rrbracket$, respectively. Assume without loss of generality, that the form of Q' is $(\nu \tilde{n}_Q)(I_1^Q \mid \cdots \mid I_{k'}^Q \mid O_1^Q \mid \cdots \mid O_{m'}^Q \mid !P_1^Q \mid \cdots \mid !P_{n'}^Q)$. In Homer the entire

expression will have this form after the synchronisation.

$$\begin{aligned}
(\tilde{m}\tilde{n}_Q') & \left(\llbracket I_1 \rrbracket_\sigma \parallel \cdots \parallel \llbracket I_{i-1} \rrbracket_\sigma \parallel \llbracket I_{i+1} \rrbracket_\sigma \parallel \cdots \parallel \llbracket I_k \rrbracket_\sigma \parallel \llbracket I_1^Q \rrbracket_\sigma^0 \parallel \cdots \parallel \right. \\
& \llbracket I_{k'}^Q \rrbracket_\sigma^0 \parallel \llbracket O_1 \rrbracket_\sigma \parallel \cdots \parallel \llbracket O_{j-1} \rrbracket_\sigma \parallel \llbracket O_{j+1} \rrbracket_\sigma \parallel \cdots \parallel \llbracket O_m \rrbracket_\sigma \parallel \\
& \llbracket O_1^Q \rrbracket_\sigma^0 \parallel \cdots \parallel \llbracket O_{m'}^Q \rrbracket_\sigma^0 \parallel \llbracket P_1^Q \rrbracket \parallel \cdots \parallel \llbracket P_{n'}^Q \rrbracket \parallel \llbracket P_1 \rrbracket_{e_1/m_1} \parallel \\
& \left. \cdots \parallel \llbracket P_l \rrbracket_{e_l/m_l} \parallel \llbracket P' \rrbracket_{g/m}^0 \parallel \Pi_{n \in \tilde{n}_Q} n' \langle \llbracket n \rrbracket \rangle \parallel r \right) , \tag{C.3}
\end{aligned}$$

where $\tilde{n}_Q' = \{n, n' \mid n \in \tilde{n}_Q\}$, and \tilde{m} and r are unchanged by the synchronisation. Again, we have left out the indices on the untouched prefixes, since these remain unchanged.

We can match this synchronisation in the π -calculus by performing a synchronisation between I_i ($l(m) . P'$) and O_j ($\bar{l}\langle n \rangle . Q'$) producing the following prenormal form

$$\begin{aligned}
\sigma' \vdash & I_1 \mid \cdots \mid I_{i-1} \mid I_{i+1} \mid \cdots \mid I_k \mid I_1^Q \mid \cdots \mid I_{k'}^Q \mid \\
& O_1 \mid \cdots \mid O_{j-1} \mid O_{j+1} \mid \cdots \mid O_m \mid O_1^Q \mid \cdots \mid O_{m'}^Q \mid \\
& !P_1^Q \mid \cdots \mid !P_{n'}^Q \mid P_1 \mid \cdots \mid P_l \mid P' , \tag{C.4}
\end{aligned}$$

where $\sigma' = \sigma[\bar{m} \mapsto \bar{e}][m \mapsto g][\tilde{n}_Q \mapsto \bar{n}_Q]$. Notice how the substitution $[m \mapsto g]$ together with the process P' correspond to the process in prenormal form $\llbracket P' \rrbracket_{g/m}^0$, as required by \mathcal{R} . Again, the π -calculus process in (C.4) is related to the Homer process in (C.3) by \mathcal{R} .

Third condition: Assume that we have a barb $\sigma \vdash P \downarrow n$, for some n . From the form of our π -calculus expressions, we know that this can either come from some input prefix I_i , for $1 \leq i \leq k$, or from one of the processes in preform P_i , for $1 \leq i \leq l$. Here we only consider the first case.

- If one of the input prefixes I_i is on the form $l(m) . P'$, and assuming that $\sigma l = n$ and $\sigma n = n$ (i.e. it is not restricted), then it gives rise to $\sigma \vdash P \downarrow n$. From the correspondence, we have $\llbracket l(m) . P' \rrbracket_\sigma^j$ for some j , where $0 \leq j \leq 5$. Hence $\llbracket l(m) . P' \rrbracket_\sigma^j$ can make $5 - j$ reactions and become $\llbracket l(m) . P' \rrbracket_\sigma^5$, and $\llbracket l(m) . P' \rrbracket_\sigma^5 \downarrow n$.

Fourth condition: We assume that $\llbracket \sigma \vdash P \rrbracket_{(\bar{i}, \bar{\sigma}, \bar{p}, \bar{e}, \bar{m})} \downarrow n$. Considering the induced prenormal form $(\star \star \star)$ and the encoding of prefixes, this can only occur, if an unrestricted π -calculus input prefix exists in index 5, so that we have a receive prefix in Homer ($\llbracket I_i \rrbracket_\sigma^5$, for some $1 \leq i \leq k$). Without loss of generality, we assume that I_i is of the form $l(m) . P'$ and that $\sigma l = n$. From the assumption we have that $\sigma n = n$. Hence, we have that $\sigma \vdash P \downarrow n$.