On Encoding the π-calculus in Higher-Order Calculi

Mikkel Bundgaard, Jens Chr. Godskesen, and Thomas Hildebrandt

IT University Technical Report Series TR-2008-106

ISSN 1600–6100 March 2008

ISSN 1600–6100

ISBN 978-87-7949-165-6

Copies may be obtained by contacting:

IT University of Copenhagen
Rued Langgaards Vej 7
DK-2300 Copenhagen S
Denmark

Telephone: +45 72 18 50 00
Telefax: +45 72 18 50 01
Web www.itu.dk

On Encoding the π-calculus in Higher-Order Calculi

Mikkel Bundgaard and Thomas Hildebrandt
{mikkelbu, hilde}itu.dk
The Programming, Logic, and Semantics group
IT University of Copenhagen

Jens Chr. Godskesen
jcg@itu.dk
The Computational Logic and Algorithms group
IT University of Copenhagen

Abstract

The connection between first-order calculi and higher-order calculi have been examined in many setting within the area of process calculi. In this paper we examine two existing encodings of the π-calculus in higher-order calculi: the encoding in HOπ-calculus by Sangiorgi and Walker and the encoding in Plain CHOCS by Thomsen. We propose a new encoding of the synchronous π-calculus in the calculus of Higher-Order Mobile Embedded Resources (Homer) inspired by the aforementioned encodings. Homer is 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. Our encoding demonstrates that non-linear higher-order process-passing together with mobile resources in, possibly local, named locations are sufficient to express π-calculus name-passing.

Introduction

The π-calculus [MPW92, Mil99] 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

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

where the name o is communicated along the name n, and the creation of local names with static scope. Combined these concepts provide the π-calculus with

most of its expressive power[1]. 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 [CG00] and the Seal calculus [CZN02], 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 an 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 [Tho93]. 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 used in the present paper is the calculus of Higher-Order Mobile Embedded Resources (Homer) introduced in [HGB04] and presented using type annotations as in [BH06, GH05, BHG07]. The Homer calculus does *not* have explicit name substitution, thus the encoding of [Tho93] cannot be *directly* applied in Homer, but the encoding presented in this paper has essentially the same structure as the encoding of Thomsen.

Homer introduce 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 explicit name substitution. Instead the encoding presented in this paper depends on the ability to represent abstractions and applications in Homer. The encoding uses a continuation-passing style to give an elegant encoding of synchronous communication.

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 $\overline{n}\langle q\rangle$ (*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

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

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.

[1] In [San96] Sangiorgi examines the expressivity of πI, a subcalculus of the π-calculus in which only private names may be communicated, and he comes to the conclusion that the expressive power of πI is close to that of the full π-calculus.

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 $n[q]$ (*resource*) and $\overline{n}(x)$ (*take*). The process $n[q].p_1$ denotes a resource q residing at the location (or address) n which may be *moved* or *taken* by the complementary prefix, $\overline{n}(x).p_2$. Just as for the previous interaction the synchronisation is expressed by the reaction

$$n[q] \,.\, p_1 \parallel \overline{n}(x) \,.\, p_2 \searrow p_1 \parallel p_2[q/x] \quad . \tag{2}$$

There are two important differences between the two types of interactions, First, in $n[q]$, but not in $\overline{n}\langle q\rangle$, the resource q may perform internal computations, i.e.

$$q \searrow q' \text{ implies } n[q] \,.\, p_1 \searrow n[q'] \,.\, p_1 \quad .$$

The encoding presented in the this paper, however, do not utilise that resources may perform internal computations. Secondly, the resource q in $n[q]$ 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 [GHS02], 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

$$n_1[n_2[q] \,.\, q' \parallel q''] \,.\, p_1 \parallel \overline{n_1n_2}(x) \,.\, p_2 \searrow n_1[q' \parallel q''] \,.\, p_1 \parallel p_2[q/x] \ . \tag{3}$$

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

$$\overline{n_1n_2}\langle q\rangle \,.\, p_1 \parallel n_1[n_2(x) \,.\, p_2' \parallel p_2''] \,.\, p_2 \searrow p_1 \parallel n_1[p_2'[q/x] \parallel p_2''] \,.\, p_2 \ . \tag{4}$$

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

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

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.

To handle the problems of local names and non-linear process passing we add type annotations, a set of names, to resource and send prefixes as in [BH06, GH05, BHG07], written $n[r : \tilde{n}]$ and $\overline{n}\langle r : \tilde{n}\rangle$. We use the type annotations instead of the free name operator used in [HGB04]. Intuitively, the

type annotation $\tilde{n}$ in $n[r : \tilde{n}]$ says that we consider the process r to have $\tilde{n}$ as the set of allocated names, assuming that $\tilde{n} \supseteq fn(r)$. We will make this formal using a type system. Besides the type annotations we only consider relations relating processes with the same type.

Related Work

In previous works, we presented an encoding of the π-calculus in Homer [BHG05, BHG06] utilising an intermediate π-calculus with explicit substitutions, which used a global environment to record all substitutions. Like the current encoding, the encoding depended on our ability to represent names as processes, but contrary to the current encoding we used indirection to encode the dynamic name-binding. We obtained indirection by placing the encoding of a name in a placeholder location. Concretely, we used an additional set of names $\mathcal{N}'$ which are in a one-to-one mapping with $\mathcal{N}$, the set of names in the π-calculus. Hence the names in $\mathcal{N}'$ only occur in the encoding and are responsible for the addresses of placeholder locations. We kept the encoding of a name n (or more precisely, its image under the substitution) as a resource at the address n'. For instance to represent that the name m was substituted for the name n we used the location n' to hold the encoding of m, $n'[[\![m]\!]]$. In this way we encoded the dynamic name-binding of the π-calculus, as we accessed the encoding of the name n through the location n' and therefore received $[\![m]\!]$, mimicking that the name m was substituted for the name n.

The encoding presented in this paper is more direct, in the sense that we do not use an intermediate π-calculus that records the substitutions, instead we use the ability to represent abstractions and applications in Homer, and we perform the substitution in the encoding, mimicking the actual synchronisation.

Zimmer presented in [Zim04] 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. To this end Zimmer designed an intermediate calculus π_{esc} (π-calculus with Explicit Substitutions and Channels).

The connection between first-order and higher-order calculi has been examined in several contexts, most notably in [San92], 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 [San92]. In Section 2.2 we present the encoding of π-calculus in HOπ by Sangiorgi and Walker presented in [SW01].

Sangiorgi notes in [San96] that the expressive power of Plain CHOCS (without relabelling) is comparable to that of CCS. The addition of locations, active process passing, and address paths to obtain the Homer calculus, thus adds expressive power to the calculus, since we are able to encode the synchronous π-calculus. We leave it for future investigation to examine the expressive power of Homer in more detail.

In [CM03] Carbone and Maffeis 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) above. They show that the degree of synchronisation of a language increases its expressive power by means of a separation result in the style of Palamidessi's result for mixed choice.

Outline

In Section 1 we present the syntax and reaction semantics of the monadic synchronous π-calculus. We present the encodings of Thomsen [Tho93] and of Sangiorgi and Walker in Section 2. In Section 3 we present the syntax and reaction semantics of Homer. We present the encoding and some examples in Section 4, and we prove the operational correspondence between π-calculus processes and their encoding in Homer.

1 The Pi-calculus

We begin by presenting 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. [MPW92, Mil99, SW01].

We will in this paper only consider a π-calculus without replication in order to make the presentation of the encoding and, in particular, the proof of the encoding succinct. But since we can encode general recursion in Homer and thereby replication, we can also encode the replication operator. Even though some of the process constructors of HOπ, Plain CHOCS, and Homer will conflict 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 $\mathcal{P}_\pi$ of *process expressions* is then defined by the grammar in Table 1.

We consider π-calculus terms up to α-conversion, let $\mathcal{P}_{\pi/\alpha}$ denote the set of α-equivalence classes of $\mathcal{P}_\pi$, and define *structural congruence* $\equiv_\pi$ in the π-calculus as the least congruence on $\mathcal{P}_{\pi/\alpha}$ satisfying the rules in Table 2.

We define the reaction relation $\rightarrow_\pi$ in terms of evaluation contexts

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

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

Table 1: The π-calculus

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.	$(\nu n)P \mid Q \equiv_\pi (\nu n)(P \mid Q)$, if $n \notin fn(Q)$		

Table 2: Structural congruence

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

$$\text{(React)}\ \frac{}{n(m) \,.\, P \mid \overline{n}\langle m'\rangle \,.\, Q \rightarrow_\pi \{m'/m\}P \mid Q}$$

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.

2 Existing Encodings

In this section we briefly review the encodings of Thomsen and of Sangiorgi and Walker. We will not present the formal semantics of neither Plain CHOCS nor HOπ and refer the reader to [Tho93] and [SW01], respectively.

2.1 Thomsen's Encoding

Thomsen demonstrated that the recursion and the name-passing of the π-calculus can be encoded in Plain CHOCS [Tho93] 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. It is assumed that the names i, o, and c are distinct names not used in the encoded processes. We will let a^{wire} denote the process representing a a-wire in the encoding.

Thomsen used an encoding scheme in two levels:

- a structurally defined encoding, $[\![\ ]\!]_1$, translating free names and names bound by an input prefix into process variables, and names bound by restriction into wires, and
- on top-level, $[\![\ ]\!]_2$, an instantiation of the process variables, representing free π-calculus names, with wires.

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

$$\begin{aligned}[\![x(y)\,.\,P]\!]_1 &= \big(x[c \mapsto c'][i \mapsto i'][o \mapsto o'] \mid i'!\,.\,c'?y\,.\,[\![P]\!]_1\big)\backslash c'\backslash i'\backslash o' \\ [\![\overline{x}\langle y\rangle\,.\,P]\!]_1 &= \big(x[c \mapsto c'][i \mapsto i'][o \mapsto o'] \mid o'!\,.\,c'!y\,.\,[\![P]\!]_1\big)\backslash c'\backslash i'\backslash o'\ ,\end{aligned}$$

where $(x[c \mapsto c'][i \mapsto i'][o \mapsto o'] \mid \ldots)\backslash c'\backslash i'\backslash o'$ uses an explicit substitution $[c \mapsto c'][i \mapsto i'][o \mapsto o']$ to localise the wire substituted 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 Section 4 differs 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 represent abstractions and applications in Homer. Furthermore, Homer does not include external choice, but we can encode this using local, named locations.

For the encoding of restriction, $(\nu x)P$, Thomsen encode the process under the restriction, substitute a fresh wire a^{wire} for the restricted name, and then use restriction of Plain CHOCS to restrict the wire

$$[\![(\nu x)P]\!]_1 = ([\![P]\!]_1[a^{\text{wire}}/x]) \setminus a\ \ \text{, where } a \notin \mathit{fn}(P).$$

The encoding is a homomorphism on the remaining constructs: the inactive process, τ-prefix, choice, and parallel composition.

As mentioned above, on top-level Thomsen instantiate the process variables, representing free names of the π-calculus process, with wires.

$$[\![P]\!]_2 = (\ldots([\![P]\!]_1[a_1^{\text{wire}}/x_1])\ldots)[a_n^{\text{wire}}/x_n]\ ,$$

where $\mathit{fv}([\![P]\!]_1) = \{x_1, \ldots, x_n\}$ and $a_1, \ldots, a_n$ are allocated by some $1-1$ mapping between V (the set of variables of Plain CHOCS) and N, usually established by the $1-1$ mapping between $\mathit{fn}(P)$ and $\mathit{fv}([\![P]\!]_1)$.

Example 2.1. Consider the π-calculus process

$$\overline{a}\langle b\rangle\,.\,b(d)\,.\,\overline{d}\langle d\rangle \mid a(x)\,.\,\overline{x}\langle c\rangle$$

which is structurally translated into the following Plain CHOCS process

$$\begin{array}{c}\big(a[c \mapsto c'][i \mapsto i'][o \mapsto o'] \mid o'! \,.\, c'!b \,.\, [\![b(d) \,.\, \overline{d}\langle d\rangle]\!]_1\big)\backslash c'\backslash i'\backslash o' \;\mid \\ \big(a[c \mapsto c'][i \mapsto i'][o \mapsto o'] \mid i'! \,.\, c'?x \,.\, [\![\overline{x}\langle c\rangle]\!]_1\big)\backslash c'\backslash i'\backslash o'\end{array}$$

and with the following top-level instantiation

$$[a^{\text{wire}}/a][b^{\text{wire}}/b][c^{\text{wire}}/c] \;\; .$$

So we end up with a Plain CHOCS process, which can make the following reactions, letting $\rightarrow_C$ denote the reaction relation in Plain CHOCS

$$\begin{array}{l}\big(i'? \,.\, a?x \,.\, c'!x + o'? \,.\, c'?x \,.\, a!x \mid o'! \,.\, c'!b^{\text{wire}} \,.\, [\![b(d) \,.\, \overline{d}\langle d\rangle]\!]_1[b^{\text{wire}}/b]\big)\backslash c'\backslash i'\backslash o' \mid \\ \big(i'? \,.\, a?x \,.\, c'!x + o'? \,.\, c'?x \,.\, a!x \mid i'! \,.\, c'?x \,.\, [\![\overline{x}\langle c\rangle]\!]_1[c^{\text{wire}}/c]\big)\backslash c'\backslash i'\backslash o' \rightarrow_C {}^2 \\ \big(a!b^{\text{wire}} \mid [\![b(d) \,.\, \overline{d}\langle d\rangle]\!]_1[b^{\text{wire}}/b]\big) \mid \big(a?x \,.\, c'!x \mid c'?x \,.\, [\![\overline{x}\langle c\rangle]\!]_1[c^{\text{wire}}/c]\big)\backslash c' \rightarrow_C {}^2 \\ [\![b(d) \,.\, \overline{d}\langle d\rangle]\!]_1[b^{\text{wire}}/b] \mid [\![\overline{x}\langle c\rangle]\!]_1[c^{\text{wire}}/c][b^{\text{wire}}/x] \rightarrow_C {}^3 \\ \big(b?x \,.\, c'!x \mid c'?d \,.\, [\![\overline{d}\langle d\rangle]\!]_1\big)\backslash c' \mid b!c^{\text{wire}} \rightarrow_C {}^2 \; [\![\overline{d}\langle d\rangle]\!]_1[c^{\text{wire}}/d] \;\; .\end{array}$$

Thomsen conjectures that strong ground bisimilar processes are translated to Plain CHOCS processes that are related by a suitable formulation of weak higher order applicative bisimulation. Thomsen also notes that for the opposite direction the implication does not hold, since the translation may introduce non-determinism that did not appear in the original process.

2.2 Sangiorgi and Walker's Encoding

In section 13.3 of [SW01] Sangiorgi and Walker give an encoding of the asynchronous localised π-calculus (ALπ) in the asynchronous higher-order π-calculus (AHOπ). Localised means that only output-capabilities of names can be communicated. Both ALπ and AHOπ are typed and the encoding depends on the typing of the ALπ process. For clarity we have omitted most of 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 b to send along. This is encoded by sending an abstraction $(z) \,.\, \overline{b}\langle z\rangle$ which, when applied to a value v, sends v along b.

Example 2.2. Consider the ALπ-process

$$\overline{a}\langle b\rangle \mid a(x) \,.\, \overline{x}\langle c\rangle \;\; .$$

The process is translated into the following AHOπ-process

$$\overline{a}\langle (y) \,.\, \overline{b}\langle y\rangle\rangle \mid a(x) \,.\, x\lfloor (z) \,.\, \overline{c}\langle z\rangle\rfloor \;\; ,$$

where $x\lfloor(z)\,.\,\overline{c}\langle z\rangle\rfloor$ is the application between the abstraction, to be substituted for x, and the value $(z)\,.\,\overline{c}\langle z\rangle$. This process can then react twice to become the encoding of the ALπ-process $\overline{b}\langle c\rangle$

$$\overline{a}\langle(y)\,.\,\overline{b}\langle y\rangle\rangle \mid a(x).x\lfloor(z)\,.\,\overline{c}\langle z\rangle\rfloor \rightarrow_{HO\pi} ((y)\,.\,\overline{b}\langle y\rangle)\lfloor(z)\,.\,\overline{c}\langle z\rangle\rfloor \rightarrow_{HO\pi} \overline{b}\langle(z)\,.\,\overline{c}\langle z\rangle\rangle.$$

In the translation Sangiorgi and Walker — among other — uses the type information to distinguish free names (and names bound by restriction) from names bound by input prefix. Below we present the most interesting parts of the encoding: the encoding of values (i.e. the entities to be communicated) and the encoding of processes[2].

Translation of values

$$[\![v]\!]^{\Gamma} \quad = \quad \begin{cases} (z)\,.\,\overline{v}\langle z\rangle & \text{if } \Gamma(v) \sim_{\text{type}} \#T, \text{ for some } T \\ (z)\,.\,v\lfloor z : [\![T]\!]\rfloor & \text{if } \Gamma(v) \sim_{\text{type}} oT \end{cases}$$

Translation of processes (only the cases for input and output)

$$[\![x(y)\,.\,P]\!]^{\Gamma} \quad = \quad x(y)\,.\,[\![P]\!]^{\Gamma,y:T} \qquad \text{if } \Gamma(x) \sim_{\text{type}} \#T$$

$$[\![\overline{x}\langle v\rangle]\!]^{\Gamma} \quad = \quad \begin{cases} \overline{x}\langle[\![v]\!]^{\Gamma}\rangle & \text{if } \Gamma(x) \sim_{\text{type}} \#T, \text{ for some } T \\ x\lfloor[\![v]\!]^{\Gamma} : [\![T]\!]\rfloor & \text{if } \Gamma(x) \sim_{\text{type}} oT \end{cases}$$

In both the translations we utilise the type information. Free names (and names bound by restriction) will have a *connection type*, written $\#T$, meaning that we can send names of type T along the name. On the other hand names bound by input prefix will only have the output capability type, written oT.

In the translation of values we have two cases, depending on the name is bound by input prefix or not. In both cases we translate the value into an abstraction, and if the name is bound by input prefix we furthermore use an application to handle the name-binding as described above. In the translation of processes we again use the type information. For the input prefix we note that we can only input along a name which is not bound by input prefix. For the output prefix we have two cases: if the name is *not* bound by input prefix we just use the name and translate the value; if the name is bound by input prefix we need to introduce an application, as described above.

The ability to send a process to a resource in a local, named location in Homer is closely related to the ability to apply an abstraction to a process in HOπ. That is, applying a process P to an abstraction A, i.e. $A\lfloor P\rfloor$ in HOπ, can be represented in Homer by

$$(n)(n[m(x)\,.\,A] \parallel \overline{nm}\langle P\rangle)\ .$$

In the expression we restrict the name n, and place the process $m(x)\,.\,A$ inside a location named n. The process can input along the name m and bind the

[2]We have simplified the translation a bit and ignored e.g. unit types.

received process to x in the process A. In parallel we have a process which can send P along the composite name nm. So P will end up being substituted for x in A. The localisation of the name n ensures that the only communication with the process inside the location n is performed by the process $\overline{nm}\langle P\rangle$. But note that the restriction of n do not (necessarily) bind any names in process residing inside the location n. So the process residing inside location n could have been received from another process, and we still could have localised the communication. Thus obtaining the same effect as a using an abstraction and application, but with added functionality that our applications can have continuations as apposed to traditional applications in i.e. the π-calculus. We can represent the explicit name substitution of Plain CHOCS in a similar manner by using a local, named location. From the representation of abstractions and applications we can in a straightforward manner translate the encoding from ALπ to AHOπ to an encoding from ALπ to Homer.

It is noted in [SW01] 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 encoding of the π-calculus into a higher-order calculus like Homer. Note that a translation of the full π-calculus, thus combining the original encoding and the two exercises, is not addressed in [SW01].

2.3 Comparing the Encodings

Common for all encodings of the first-order π-calculus into a higher-order calculus is the representation of a name using a process (or using a parametrised process, i.e. an abstraction), as the only form of communication is the communication of a process (or abstraction). In most cases the translation is a homomorphism on several of the constructs (i.e. the inactive process, parallel composition etc.), and the complexity of the translation is in encoding of the prefixes. In Plain CHOCS the complexity is due to the usage of wires representing names, and in AHOπ the complexity is due to the usage of type information in the translation to handle whether a name is bound by an input prefix.

The two encodings presented in this section also differ in other aspects. In the encoding in Plain CHOCS we use explicit name substitutions and restrictions compared to the abstractions and applications in AHOπ to localise communication between the encoding of a prefix and the processes representing the names used in the prefix.

A obvious difference between the two encodings is that Thomsen gives an encoding of the full π-calculus whereas Sangiorgi and Walker only consider a subset. On the other hand, Sangiorgi and Walker prove that their encoding is fully abstract with respect to barbed bisimulation and sound with respect to barbed congruence, whereas Thomsen only conjectures that strong ground bisimilar processes are translated to Plain CHOCS processes that are related by a suitable formulation of weak higher order applicative bisimulation

Processes:			
$p, q, r ::=$		$\mathbf{0}$	inactive process
	$\mid$	$\pi \,.\, p$	action prefixing
	$\mid$	$p \parallel q$	parallel composition
	$\mid$	$(n)p$	let n be local in p
	$\mid$	x	process variable
Prefixes:			
$\pi ::=$		$\delta(x)$	receive a resource at δ and bind it to x
	$\mid$	$\overline{\delta}(x)$	take computing resource from δ and bind it to x
	$\mid$	$\overline{\delta}\langle r : \tilde{n}\rangle$	send a passive resource r having type $\tilde{n}$ to δ
	$\mid$	$\delta[r : \tilde{n}]$	computing resource r at location δ having type $\tilde{n}$

Table 3: Higher-Order Mobile Embedded Resources

3 Homer

In this section we present the calculus of Higher-Order Mobile Embedded Resources (Homer), a non-linear, pure higher-order process calculi with local names and named, nested locations. As explained in the introduction Homer contains explicitly typed locations (and send prefixes). The type annotation is a set of names that include the free names of the process residing in the location (or send prefix) to handle the problems of scope extension as described in [BH06, GH05]. Furthermore, we will only consider relations that relate processes with the same top-level type.

3.1 Syntax and notation

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 γ range over (possibly empty) sequences of names, and let δ range over non-empty sequences of names referred to as *addresses*, also we let $\phi ::= \delta \mid \overline{\delta}$. We assume an infinite set of *process variables* $\mathcal{V}$ ranged over by x and y, and let $\tilde{x}$ range over finite sets of variables. The set $\mathcal{P}$ of *process expressions* is then defined by the grammar in Table 3.

The processes constructors are the usual process constructors from higher-order concurrent process calculi. As usual, we let the restriction operator (n) bind the name n and the prefixes $\phi(x)$ bind the variable x. Note that the restriction operator also can bind the names that occur in a type annotation.

The prefix $\delta(x)$ represents the possibility to receive a passive resource sent from a local processes or a processes in a parent-location, whereas the prefix $\overline{\delta}(x)$ represents the possibility to take an active resource from a local location or a sub-

$$\frac{}{\vdash \mathbf{0} : \tilde{n}} \qquad \frac{\vdash p : \tilde{n}_1 \qquad \vdash q : \tilde{n}_2}{\vdash p \parallel q : \tilde{n}_1 \cup \tilde{n}_2}$$

$$\frac{}{\vdash x : \tilde{n}} \qquad \frac{\vdash p : \tilde{n}n}{\vdash (n)p : \tilde{n}}$$

$$\frac{\vdash p : \tilde{n}}{\vdash \phi(x) \,.\, p : \tilde{n} \cup fn(\phi)} \qquad \frac{\vdash r : \tilde{m} \qquad \vdash p : \tilde{n}}{\vdash \phi[\!\langle r : \tilde{m} \rangle\!] \,.\, p : \tilde{m} \cup \tilde{n} \cup fn(\phi)}$$

Table 4: Typing rules for Homer

location. The prefixes $\overline{\delta}\langle r : \tilde{n}\rangle$ and $\delta[r : \tilde{n}]$ are responsible for sending a *passive* resource r locally (or down) to the address δ and providing an *active* resource r locally (or up) on the location δ, respectively. In both cases we explicitly annotate the prefix with a set containing the free names of the resource. The prefixes $\overline{\delta}\langle r : \tilde{n}\rangle$ and $\delta(x)$ are the usual prefixes of Plain CHOCS [Tho93], except for the type annotation and that we allow sequences of names as addresses instead of only a name. The prefixes $\delta[r : \tilde{n}]$ and $\overline{\delta}(x)$ are responsible for adding active process mobility to the calculus.

We define the *free names* and *free variables* as usual, with the exception of the free names of the prefixes $\phi[\!\langle r : \tilde{n}\rangle\!]$, which we define as $fn(\phi[\!\langle r : \tilde{n}\rangle\!]) = fn(\phi) \cup \tilde{n}$, so the type annotation of a send or a location prefix determines the free names of the resource in the prefix, under the assumption that the type annotation contains the free names of the resource. We will ensure this using a simple type system (see Definition 3.2). The sets $bn(p)$ and $bv(p)$ of *bound names* and *bound variables* are defined according as usual.

We define capture-free substitution in usual manner, though with the proper update of type annotations.

Definition 3.1 (substitutions)**.** We define the process $p[q : \tilde{n}/x]$ to be p with all free occurrences of x replaced by q of type $\tilde{n}$, where we have changed the annotations of all sub-terms $\phi[\!\langle r : \tilde{m}\rangle\!]$ in p to $\phi[\!\langle r : \tilde{m} \cup \tilde{n}\rangle\!]$, if and only if r contains a free occurrence of x, and if necessary α-converting p such that no free names and variables in q are bound.

Definition 3.2 (well-typed process)**.** We define the valid typing judgements of the form $\vdash p : \tilde{n}$ inductive by the rules in Table 4.

From now on we will only consider well-typed processes. Note that a process p is well-typed with respect to a finite set of names $\tilde{n}$, written $\vdash p : \tilde{n}$, if and only if the free names of p are included in the set $\tilde{n}$, and for every sub-term $\phi[\!\langle r : \tilde{m}\rangle\!]$ in p we have that r can be typed with the type $\tilde{m}$. We will say that the type annotations in a process are *valid* if for all sub-terms $\phi[\!\langle r : \tilde{m}\rangle\!]$ it is the case that $\tilde{m} \supseteq fn(r)$.

We say that a process with no free variables is *closed* and let $\mathcal{P}_c$ denote the set of closed processes. We write $p \equiv_\alpha q$, if p and q are α-convertible (with respect to both names and variables), we let $\mathcal{P}_{/\alpha}$ (and $\mathcal{P}_{c/\alpha}$) denote the set of α-equivalence classes of (closed) process expressions, and we consider processes up to α-equivalence.

We omit trailing $\mathbf{0}$s, and hence write π instead of $\pi \,.\, \mathbf{0}$. We let prefixing and restriction be right associative and bind stronger than parallel composition, hence writing e.g. $\pi \,.\, p \parallel (n)q \parallel r$ instead of $(\pi \,.\, p) \parallel ((n)q) \parallel r$. For a set of names $\tilde{n} = \{n_1, \ldots, n_k\}$ 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 $\tilde{m} \cap \tilde{n} = \emptyset$.

3.2 Reaction Semantics

We provide Homer with a reaction semantics defined in the Chemical Abstract Machine [BB90] style using structural congruence, evaluation contexts, and reaction rules.

Definition 3.3 (contexts and congruence)**.** A *context* $\mathcal{C}$ is defined by taking the grammar defined in Table 3 and augmenting the production of process expressions to also contain a special symbol called a *hole*

$$\mathcal{C} ::= \quad \ldots \quad | \quad (-)_{\tilde{n}}$$

and by requiring that the hole only occur once in the term. We annotate the hole with a type, meaning we can only place a process of type $\tilde{n}$ into the hole $(-)_{\tilde{n}}$. We write $\mathcal{C}(p)$ for the insertion of p into the hole of the context $\mathcal{C}$, assuming that the hole in $\mathcal{C}$ is annotated with the type $\tilde{n}$ and we have $\vdash p : \tilde{n}$. We extend $fn()$ to contexts by $fn(\mathcal{C}) = fn(\mathcal{C}(\mathbf{0}))$, and we extend $fv()$ accordingly. For typing contexts, we add the following rule to the typing rules of Table 4.

$$\frac{}{\vdash (-)_{\tilde{n}} : \tilde{n}}$$

A binary relation $\mathcal{R}$ on well-typed processes is called *well-typed* if and only if it only relates processes p and q with the same type $\tilde{n}$, written $\vdash p \;\mathcal{R}\; q : \tilde{n}$. We will only consider well-typed relations in this paper. A relation $\mathcal{R}$ is called a *congruence* if and only if it is a well-typed equivalence relation and it satisfies that $\vdash p \;\mathcal{R}\; q : \tilde{n}$ implies $\vdash \mathcal{C}(p) \;\mathcal{R}\; \mathcal{C}(q) : \tilde{n}'$ for all contexts $\mathcal{C}$, where the hole is annotated with the type $\tilde{n}$ and the type of the context is $\tilde{n}'$.

Structural congruence $\equiv$ is defined as the least congruence on well-typed processes relating $\vdash p \equiv q : \tilde{n}$, if $p \equiv q$ can be derived using the rules in Table 5 and $\vdash p : \tilde{n}$ and $\vdash q : \tilde{n}$, as structural congruence does not affect the typing of a process. The first row of the equations express that $(P, \parallel, \mathbf{0})$ is a commutative monoid, the next two rows enforce the rules of scope of name restriction.

As Homer permits reactions arbitrarily deep in the location hierarchy and also permits reactions between a process and an arbitrarily deeply nested subresource, we define in Definition 3.4 the concepts of evaluation and path contexts.

$$p \parallel \mathbf{0} \equiv p \qquad (p \parallel p') \parallel p'' \equiv p \parallel (p' \parallel p'') \qquad p \parallel q \equiv q \parallel p$$
$$(n)p \parallel q \equiv (n)(p \parallel q), \text{ if } n \notin fn(q) \qquad (n)(m)p \equiv (m)(n)p$$
$$(n)p \equiv p, \text{ if } n \notin fn(p)$$

Table 5: Structural congruence

Definition 3.4 (evaluation contexts and path contexts)**.** An *evaluation context* $\mathcal{E}$ is a context with no free variables and whose hole is not guarded by a prefix, nor does it occur as the object of a send prefix. We define evaluation contexts by the following grammar

$$\mathcal{E} ::= (-) \mid \mathcal{E} \parallel p \mid (n)\mathcal{E} \mid \delta[\mathcal{E} : \tilde{n}] \,.\, p, \text{ for } p \in \mathcal{P}_c \ .$$

We define a family of *path contexts* $\mathcal{C}^{\tilde{n}}_{\gamma}$, indexed by a path address $\gamma \in \mathcal{N}^*$ and a set of names $\tilde{n}$, inductively in $\tilde{n}$ and γ

$$\begin{aligned} \mathcal{C}^{\emptyset}_{\epsilon} &::= (-) \\ \mathcal{C}^{\tilde{n}\tilde{m}}_{\delta\gamma} &::= \delta[(\tilde{n})(\mathcal{C}^{\tilde{m}}_{\gamma} \parallel p) : \tilde{n}'] \,.\, q, \end{aligned}$$

whenever $p, q \in \mathcal{P}_c$ and $\tilde{n} \cap \gamma = \emptyset$.

Note that the evaluation context $\delta[\mathcal{E} : \tilde{n}].p$ enables internal reactions of active resources, and that for a path context $\mathcal{C}^{\tilde{n}}_{\gamma}$, the path address γ indicates the path under which the context's hole is found, and the set of names $\tilde{n}$ indicates the bound names of the hole. The side condition in the definition of path contexts ensures that none of the names in the path address of the hole are bound. The bound names $(\tilde{n})$ in the definition of path contexts are needed since the structural congruence does not permit vertical scope extension, as explained in [HGB04] or [BH06].

We handle the vertical scope extension and the update of type annotations of a location using an *open* operator, defined on path contexts.

Definition 3.5 (open operator on path contexts)**.** We define an *open* operator on path contexts $\tilde{m} \odot \mathcal{C}^{\tilde{n}}_{\gamma}$ inductively by:

$$\begin{aligned} \tilde{m} \odot \mathcal{C}^{\emptyset}_{\epsilon} &= \mathcal{C}^{\emptyset}_{\epsilon} \\ \tilde{m} \odot \mathcal{C}^{\tilde{n}_1\tilde{n}_2}_{\delta\gamma} &= \delta[(\tilde{n}_1 \setminus \tilde{m})(\tilde{m} \odot \mathcal{C}^{\tilde{n}_2}_{\gamma} \parallel p) : \tilde{m}' \cup \tilde{m}] \,.\, q \ , \end{aligned}$$

if $\mathcal{C}^{\tilde{n}_1\tilde{n}_2}_{\delta\gamma} = \delta[(\tilde{n}_1)(\mathcal{C}^{\tilde{n}_2}_{\gamma} \parallel p) : \tilde{m}'] \,.\, q$ and $\tilde{m} \cap \tilde{n}_1\tilde{n}_2 \cap fn(\mathcal{C}^{\tilde{n}_1\tilde{n}_2}_{\delta\gamma}) = \emptyset$.

Intuitively, the open operator in $\tilde{m} \odot \mathcal{C}^{\tilde{n}}_{\gamma}$ removes the names $\tilde{m}$ from the bound names of the hole and adds them to the type annotations of the locations that are part of the address path. When applied in the reaction rule, the latter

$$(send)\ \vdash \overline{\gamma\delta}\langle r : \tilde{n}\rangle \,.\, q \parallel \mathcal{C}^{\tilde{m}}_{\gamma}(\delta(x) \,.\, p) \ \searrow\ q \parallel \tilde{n} \odot \mathcal{C}^{\tilde{m}}_{\gamma}(p[r : \tilde{n}/x]) : \tilde{n}' \ , \quad \text{if } \tilde{m} \cap (\delta \cup \tilde{n}) = \emptyset$$

$$(take)\ \vdash \mathcal{C}^{\tilde{m}}_{\gamma}(\delta[r : \tilde{n}] \,.\, q) \parallel \overline{\gamma\delta}(x) \,.\, p \ \searrow\ (\tilde{n} \cap \tilde{m})\big(\tilde{n} \odot \mathcal{C}^{\tilde{m}}_{\gamma}(q) \parallel p[r : \tilde{n}/x]\big) : \tilde{n}' \ , \quad \text{if } \tilde{m} \cap (\delta \cup fn(p)) = \emptyset$$

Table 6: Reaction rules for Homer

condition of the open operator can always be met by α-conversion, the condition ensures us that we can extend the scope by using the open operator and place the restriction at top level, without any name captures.

As for the structural congruence, we define the reaction relation for Homer, written $\searrow$, as the least well-typed binary relation between well-typed processes satisfying the rules in Table 6 and closed under all evaluation contexts $\mathcal{E}$ and structural congruence. The rules are essentially the reaction rules of [HGB04] altered to use type annotations instead of the free name constructor.

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 where it is substituted for x in p, possibly in several copies, updating the type annotations as necessary. The side conditions ensure the location path is not bound in the context and that no free names of r get bound during movement. Note that the open operator only extends the type annotations of the locations constituting the location path and does not lift any restrictions, since $\tilde{m} \cap \tilde{n} = \emptyset$.

The (*take*) rule captures that a computing resource r is taken from the (sub) location γ, where it is running at the address δ, and is substituted for x in p, possibly in several copies. Again, the side conditions ensure that the location path is not bound in the context and that no free names are bound, when we lift the restriction. In this rule it is possible that the open operator both lifts restrictions and extends the type annotations of the locations.

The types ensure that no names can disappear from the free names of a location, a send prefix or from top-level during reaction. However, note that locations or send prefixes in the process that receives the moved resource r can get their type annotation extended by the type of r that do not already appear in their annotation.

4 The Encoding

Inspired by the encoding in section 13.3 of [SW01] by Sangiorgi and Walker and the encoding of Thomsen in [Tho93] we present an encoding of the π-calculus in the calculus of Homer. Our encoding crucially depends on the fact that we can send a process representing an abstraction. In the application of an abstraction to a process we use the full non-linearity of Homer. As illustrated in Section 2.2

we can encode an abstraction as a process wanting to receive on a name, for clarity we chose the name *abs*

$$[\![(x)P]\!]_1 = abs(x) \,.\, [\![P]\!]_1 \ .$$

Note that since we encode the application of an abstraction to a value using a local location we need an auxiliary reaction to retrieve the result from the locale location. So the representation of the π-calculus application $(x)Q\lfloor P \rfloor$ will be represented in the following manner[3] (assuming that n is fresh for $[\![P]\!]_1$ and $[\![Q]\!]_1$).

$$(n)\left(n[abs(x) \,.\, [\![Q]\!]_1] \parallel \overline{nabs}\langle [\![P]\!]_1 \rangle \,.\, \overline{n}(z) \,.\, z\right)$$

and after two reactions (and garbage collection of the idle restriction of n) we obtain

$$[\![Q]\!]_1[[\![P]\!]_1/x] \ .$$

Contrary to Thomsen's encoding in Plain CHOCS we do not need external choice to encode that a name can be used both for input and for output. Instead, we use nested locations, and the choice between the two branches can be represented by a parallel composition of two locations (containing the possibility for send and receiving along the name, respectively) and a process responsible for choosing the resource in one of the locations. It should be noted, however, that by using nested locations we need to explicitly garbage collect the rest of the name. We encode a π-calculus name n as a mobile resource $[\![n]\!]$ that can perform two tasks: *sending* and *receiving* along the name n.

$$\begin{aligned}
Send_n &= v(x) \,.\, c(y) \,.\, \overline{n}\langle x : \emptyset \rangle \,.\, y \\
Receive_n &= c(x) \,.\, n(y) \,.\, (a)(a[x : \emptyset] \parallel \overline{aabs}\langle y : \emptyset \rangle \,.\, \overline{a}(z) \,.\, z) \\
[\![n]\!] &= s[Send_n : \{v, c, n\}] \parallel r[Receive_n : \{c, n, b, s\}]
\end{aligned}$$

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 $[\![m]\!]$ to send, and on location c the encoding of the continuation $[\![P]\!]_1$, resulting in a process of the following form $\overline{n}\langle [\![m]\!] : \tilde{n} \rangle \,.\, [\![P]\!]_1$. $Receive_n$ expects to receive a process representing an abstraction $[\![(x)P']\!]_1$ along c and after performing the synchronisation on n, corresponding to the actual π-calculus synchronisation, it sends down the received process to the abstraction. In parallel these two processes can interact as follows

$$\begin{gathered}
\overline{n}\langle [\![m]\!] \rangle \,.\, [\![P]\!]_1 \parallel n(y) \,.\, (a)(a[[\![(x)P']\!]_1] \parallel \overline{aabs}\langle y \rangle \,.\, \overline{a}(z) \,.\, z) \searrow \\
[\![P]\!]_1 \parallel (a)(a[[\![(x)P']\!]_1] \parallel \overline{aabs}\langle [\![m]\!] \rangle \,.\, \overline{a}(z) \,.\, z) \searrow \\
[\![P]\!]_1 \parallel (a)(a[[\![P']\!]_1[[\![m]\!]/x]] \parallel \overline{a}(z) \,.\, z) \searrow \\
[\![P]\!]_1 \parallel [\![P']\!]_1[[\![m]\!]/x] \ .
\end{gathered}$$

For the input and output prefixes we use the same simple pattern.

$$\begin{aligned}
[\![\overline{n}\langle m \rangle \,.\, P]\!]_1 &= (a)(a[n' : \emptyset] \parallel \overline{asv}\langle m' : \emptyset \rangle \,.\, \overline{asc}\langle [\![P]\!]_1 : \emptyset \rangle \,.\, \overline{as}(z) \,.\, \overline{a}(z') \,.\, z) \\
[\![n(x) \,.\, P]\!]_1 &= (a)(a[n' : \emptyset] \parallel \overline{arc}\langle [\![(x)P]\!]_1 : \emptyset \rangle \,.\, \overline{ar}(z) \,.\, \overline{a}(z') \,.\, z)
\end{aligned}$$

[3] We will ignore the type annotations in the following examples to increase readability.

In both cases we use the free variable n' residing at the location a, this variable will on top-level be replaced by the process $[\![n]\!]$. We abuse notation and use the notation n' to emphasise that n' is a process variable and not a name. We assume that there exists a one-to-one mapping between the variables n' in Homer and the names n in the π-calculus. For the output prefix we also use the free variable m' which together with the encoding of the continuation are sent down to the $Send_n$ part of $[\![n]\!]$. For the input prefix we send down an abstraction, $[\![(x)P]\!]_1$, to the $Receive_n$ part of $[\![n]\!]$. Both cases conclude by retrieving their respective part of $[\![n]\!]$ and discarding the rest of the location a.

The encoding of the remaining process expressions is a homomorphism for most of the constructs of the π-calculus.

$$\begin{array}{lcl} [\![\mathbf{0}]\!]_1 & = & \mathbf{0} \\ [\![P \mid Q]\!]_1 & = & [\![P]\!]_1 \parallel [\![Q]\!]_1 \\ [\![(\nu n)P]\!]_1 & = & (n)([\![P]\!]_1[[\![n]\!] : \tilde{n}/n']) \end{array}$$

For the encoding of restriction we restrict the corresponding name in Homer, substitute the process $[\![n]\!]$ for the free variable n', and let $\tilde{n} = \{s, v, c, n, r, abs\}$.

At top-level we substitute the encoding of a name for the free variable representing the name. For a π-calculus-process P with free names $\{n_1, \ldots n_m\}$, the encoding of P, $[\![P]\!]_1$, will have the set $\{n'_1, \ldots n'_m\}$ as free variables. So for the free variable n' we substitute in the process $[\![n]\!]$, hence giving the following encoding at top-level, denoted $[\![\ ]\!]_2$,

$$[\![P]\!]_2 = [\![P]\!]_1[[\![n_1]\!] : \tilde{n}_1/n'_1, \ldots, [\![n_m]\!] : \tilde{n}_m/n'_m] \ ,$$

where we let $\tilde{n}_i = \{s, v, c, n_i, r, abs\}$.

We can now mimic π-calculus reactions using multiple reactions in Homer. Note that we both need some auxiliary reactions before the reaction corresponding to the actual reaction in the π-calculus to "activate" the input and output prefixes, and we need some auxiliary reactions after to perform the name binding (as part of the input prefix).

Example 4.1. Consider the following reaction $\overline{n}\langle m\rangle \mid n(x) \,.\, \overline{x}\langle x\rangle \rightarrow_\pi \overline{m}\langle m\rangle$ in the π-calculus. First we translate the process $P = \overline{n}\langle m\rangle \mid n(x) \,.\, \overline{x}\langle x\rangle$

$$\begin{array}{rcl} [\![P]\!]_1 & = & (a)\big(a[n'] \parallel \overline{asv}\langle m'\rangle \,.\, \overline{asc}\langle [\![\mathbf{0}]\!]_1\rangle \,.\, \overline{as}(z) \,.\, \overline{a}(z') \,.\, z\big) \parallel \\ & & (a)\big(a[n'] \parallel \overline{arc}\langle [\![(x)\overline{x}\langle x\rangle]\!]_1\rangle \,.\, \overline{ar}(z) \,.\, \overline{a}(z') \,.\, z\big) \ , \end{array}$$

and then we apply the top-level substitution $[[\![n]\!]/n', [\![m]\!]/m']$

$$\begin{array}{c} [\![P]\!]_1[[\![n]\!]/n', [\![m]\!]/m'] \searrow^* \\ \overline{n}\langle [\![m]\!]\rangle \parallel n(y) \,.\, (a)\big(a[[\![(x)\overline{x}\langle x\rangle]\!]_1] \parallel \overline{aabs}\langle y\rangle \,.\, \overline{a}(z) \,.\, z\big) \searrow \\ (a)\big(a[abs(x) \,.\, [\![\overline{x}\langle x\rangle]\!]_1] \parallel \overline{aabs}\langle [\![m]\!]\rangle \,.\, \overline{a}(z) \,.\, z\big) \searrow^* \\ [\![\overline{x}\langle x\rangle]\!]_1[[\![m]\!]/x'] = [\![\overline{m}\langle m\rangle]\!]_1[[\![m]\!]/m'] \ . \end{array}$$

The Homer reaction (over the name n and communicating the process $[\![m]\!]$) in the second row above corresponds to the reaction (again over the name n

and communicating the name m) in the π-calculus. To state the proof of the dynamic correspondence, in the following section, we will use the special symbol $\searrow_\pi$ to denote this reaction. Note that this is the only Homer reaction in the encoding where a synchronisation occurs between a send- and a receive-prefix both of length one (and both residing on top-level).

The encoding highlights several implicit observations. The only difference between free and restricted names in the encoding is the obvious restrictions (in Homer) of the restricted names, otherwise the two kinds of entities behave the same. Thomsen's encoding depends on explicit name substitutions to restrict some of the names of a received process, indirectly making the restriction operator a *dynamic* operator which captures some of the names of a received process. In a similar manner Sangiorgi and Walkers's encoding depends on the ability to communicate abstractions. We can encode this by the usage of local, nested locations, where we localise the location in which we place the received process. We can then communicate with the process by the usage of address paths.

In several places the encoding depends upon strongly mobile resources. We use strongly mobile resources when we localise communication (as describe above) where we place resources in local addresses and let them compute, before taking them up again. To encode the synchronous communication we only use code mobility by utilising a continuation-passing style where we pass the residual of a prefix as a continuation to the process responsible for performing the prefix.

4.1 Proof of the Correspondence

In this section we prove the correspondence between π-calculus processes and their encoding as Homer processes. First we prove that the encoding respects and reflects structural congruence. To this end we need to relate structural congruence with the two levels of the translation.

Lemma 4.2. $[\![P]\!]_1 \equiv [\![Q]\!]_1$ *if and only if* $[\![P]\!]_2 \equiv [\![Q]\!]_2$.

Proposition 4.3 (Static Correspondence)**.** $P \equiv_\pi Q$ *if and only if* $[\![P]\!]_2 \equiv [\![Q]\!]_2$.

To establish that the encoding preserves the reactions of the π-calculus we need that we can characterise reactions in the π-calculus up-to structural congruence.

Lemma 4.4. $P \rightarrow_\pi P'$ *if and only if* $P \equiv_\pi (\nu\tilde{n})(n(m) \,.\, Q \mid \overline{n}\langle o\rangle \,.\, Q' \mid Q'')$ *and* $P' \equiv_\pi (\nu\tilde{n})(\{o/m\}Q \mid Q' \mid Q'')$ *for some names* $\tilde{n}$, m, n, *and* o.

We also need a substitution lemma relating the translation and substitutions.

Lemma 4.5. *For a* π*-calculus process* P *we have* $[\![P]\!]_1[n'/m'] = [\![\{n/m\}P]\!]_1$.

Proposition 4.6. *If* $P \rightarrow_\pi P'$ *then* $[\![P]\!]_2 \searrow^7 \searrow_\pi \searrow^2 [\![P']\!]_2$.

Proof (Sketch). Assuming $P \rightarrow_\pi P'$ we know from Lemma 4.4 that P and P' are of the following forms:

$$\begin{array}{lll} P & \equiv_\pi & (\nu\tilde{n})(n(m) \,.\, Q \mid \overline{n}\langle o\rangle \,.\, Q' \mid Q'') \\ P' & \equiv_\pi & (\nu\tilde{n})(\{o/m\}Q \mid Q' \mid Q'') \ , \end{array}$$

for some names $\tilde{n}$, m, n, and o. It is straightforward to prove that

$$[\![(\nu\tilde{n})(n(m) \,.\, Q \mid \overline{n}\langle o\rangle \,.\, Q' \mid Q'')]\!]_2 \searrow^7 \searrow_\pi \searrow^2 [\![(\nu\tilde{n})(\{o/m\}Q \mid Q' \mid Q'')]\!]_2 \ .$$

using Lemma 4.5 and some simple reasoning about substitutions. The result follows by Proposition 4.3 and since both reaction relations are closed under structural congruence. □

Proposition 4.7. *If* $[\![P]\!]_2 \searrow^7 \searrow_\pi \searrow^2 [\![P']\!]_2$ *then* $P \rightarrow_\pi P'$.

Proof (Sketch). As a first observation we note that in the sequence $\searrow^7 \searrow_\pi \searrow^2$ the first seven reactions can only have been responsible for "activating" an input and an output prefix, respectively, as otherwise we would not be able to do a $\searrow_\pi$ reaction. In the same manner, the last two reactions can have only have been the reactions responsible for the name-binding of the input prefix as otherwise we would not end up in a Homer process which is in the image of the translation. Again, we need Lemma 4.5 and some simple reasoning about substitutions. □

Combining the above results we obtain the dynamic correspondence between π-calculus processes and their encoding as Homer processes.

Theorem 4.8 (Dynamic Correspondence)**.**

$$P \rightarrow_\pi P' \textit{ if and only if } [\![P]\!]_2 \searrow^7 \searrow_\pi \searrow^2 [\![P']\!]_2 \ .$$

5 Conclusions and Future Work

In this paper we have examined two existing encodings of the first-order π-calculus into a higher-order calculus: HOπ and Plain CHOCS, respectively. We have furthermore presented a direct 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. The encoding is direct, in the sense that it does not utilise an intermediate π-calculus with explicit substitutions as the prior encoding in Homer did (see [BHG05, BHG06]). Instead we use that we can represent abstractions and applications and external choice succinctly in Homer by using local, named locations and active process passing. The encoding in Homer owes a lot to the existing encodings by Thomsen [Tho93] and by Sangiorgi and Walker [SW01], but the encoding also show that active process-passing, named nested locations, and local names gives Homer great expressive power. Following the ideas of our previous encoding in [BHG05, BHG06] we have used a continuation passing style to give an elegant encoding of synchronous communication.

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 examine the expressive power of Homer in more detail. The hierarchy presented in [Gor06b] could be a promising starting point. Also we plan to examine whether the lengths of the addresses in Homer affects its expressive power, as it is the case in the π-calculus with polyadic synchronisation [CM03]. I.e. whether Homer where addresses are of length "at most k" is more expressive than Homer where addresses are of length "at most $k-1$". For small k the results are obvious, but for larger k the results are unclear. Third, we plan to examine a fully compositional encoding of an untyped π-calculus in a higher-order process calculus such as Homer. The encoding in [Tho93] is done in two levels, as the encoding in the present paper. The encoding in [SW01] is fully compositional, however it only covers a subset of the full π-calculus and furthermore it depends crucially on the typing of the π-calculus process being encoded.

In a different direction it could be interesting to examine the question of what constitutes a 'good' encoding. This area has largely been unexplored in the literature. In [Pal03] Palamidessi proposes uniform encodings to examine the expressive power of synchronous and asynchronous π-calculus. An encoding is uniform if it is compositional and preserve some "reasonable" semantics. Furthermore, she require that the encoding of parallel composition is a homomorphism, and that the encoding commutes with (name) substitutions. In [Gor06a] Gorla uses a similar condition on encodings to study the expressive power of eight different asynchronous communication primitives.

References

[BB90] Gérard Berry and Gérard Boudol. The chemical abstract machine. In *Proceedings of the 17th ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL'90)*, pages 81–94. ACM Press, 1990.

[BH06] Mikkel Bundgaard and Thomas Hildebrandt. Bigraphical semantics of higher-order mobile embedded resources with local names. In Arend Rensink, Reiko Heckel, and Barbara König, editors, *Proceedings of the Graph Transformation for Verification and Concurrency workshop (GT-VC'05)*, volume 154 of *Electronic Notes in Theoretical Computer Science*, pages 7–29. Elsevier, 2006.

[BHG05] Mikkel Bundgaard, Thomas Hildebrandt, and Jens Chr. Godskesen. A CPS encoding of name-passing in higher-order mobile embedded resources. In Jos Baeten and Flavio Corradini, editors, *Proceedings of the 11th International Workshop on Expressiveness in Concur-*

rency (EXPRESS'04), volume 128 of *Electronic Notes in Theoretical Computer Science*, pages 131–150. Elsevier, 2005.

[BHG06] Mikkel Bundgaard, Thomas Hildebrandt, and Jens Chr. Godskesen. A CPS encoding of name-passing in higher-order mobile embedded resources. *Theoretical Computer Science*, 356(3):422–439, 2006.

[BHG07] Mikkel Bundgaard, Thomas Hildebrandt, and Jens Chr. Godskesen. Modelling the security of smart cards by hard and soft types for higher-order mobile embedded resources. In Daniele Gorla and Catuscia Palamidessi, editors, *Proceedings of the 5th International Workshop on Security Issues in Concurrency (SecCo'07)*, volume 194 of *Electronic Notes in Theoretical Computer Science*, pages 23–38. Elsevier, 2007.

[CG00] Luca Cardelli and Andrew D. Gordon. Mobile ambients. *Theoretical Computer Science*, 240(1):177–213, 2000.

[CM03] Marco Carbone and Sergio Maffeis. On the expressive power of polyadic synchronisation in π-calculus. *Nordic Journal of Computing*, 10(2):70–98, 2003.

[CZN02] Giuseppe Castagna and Francesco Zappa Nardelli. The Seal calculus revisited: Contextual equivalence and bisimilarity. In Manindra Agrawal and Anil Seth, editors, *Proceedings of the 22nd Conference on the Foundations of Software Technology and Theoretical Computer Science (FSTTCS'02)*, volume 2556 of *Lecture Notes in Computer Science*, pages 85–96. Springer Verlag, 2002.

[GH05] Jens Chr. Godskesen and Thomas Hildebrandt. Extending Howe's method to early bisimulations for typed mobile embedded resources with local names. In *Proceedings of the 25th Conference on the Foundations of Software Technology and Theoretical Computer Science (FSTTCS'05)*, volume 3821 of *Lecture Notes in Computer Science*, pages 140–151. Springer Verlag, 2005.

[GHS02] Jens Christian Godskesen, Thomas Hildebrandt, and Vladimiro Sassone. A calculus of mobile resources. In Lubos Brim, Petr Jancar, Mojmír Kretínský, and Antonín Kucera, editors, *Proceedings of the 13th International Conference on Concurrency Theory (CONCUR'02)*, volume 2421 of *Lecture Notes in Computer Science*, pages 272–287. Springer Verlag, 2002.

[Gor06a] Daniele Gorla. On the relative expressive power of asynchronous communication primitives. In *Proceedings of the 9th International Conference of Foundations of Software Science and Computation Structures (FOSSACS'06)*, volume 3921 of *Lecture Notes in Computer Science*, pages 47–62. Springer Verlag, 2006.

[Gor06b] Daniele Gorla. On the relative expressive power of calculi for mobility. Technical Report 09/2006, University of Rome "La Sapienza", 2006.

[HGB04] Thomas Hildebrandt, Jens Chr. Godskesen, and Mikkel Bundgaard. Bisimulation congruences for Homer — a calculus of higher order mobile embedded resources. Technical Report TR-2004-52, IT University of Copenhagen, 2004.

[Mil99] Robin Milner. *Communicating and Mobile Systems: the π-calculus*. Cambridge University Press, 1999.

[MPW92] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, parts I and II. *Journal of Information and Computation*, 100:1–40 and 41–77, 1992.

[Pal03] Catuscia Palamidessi. Comparing the expressive power of the synchronous and asynchronous π-calculi. *Journal of Mathematical Structures in Computer Science*, 13(5):685–719, 2003.

[San92] Davide Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*. PhD thesis, Department of Computer Science, University of Edinburgh, 1992.

[San96] Davide Sangiorgi. Pi-calculus, internal mobility, and agent-passing calculi. *Theoretical Computer Science*, 167(1–2):235–274, 1996.

[SW01] Davide Sangiorgi and David Walker. *The π-calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001.

[Tho93] Bent Thomsen. Plain CHOCS: A second generation calculus for higher order processes. *Acta Informatica*, 30(1):1–59, 1993.

[Zim04] Pascal Zimmer. On the expressiveness of pure mobile ambients. *Journal of Mathematical Structures in Computer Science*, 13(5):721–770, 2004.