Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 242 (2009) 113–138
www.elsevier.com/locate/entcs

Decidable Fragments of a Higher Order Calculus with Locations

Mikkel Bundgaard[1,3] Jens Chr. Godskesen[2,4]

The Programming, Logic, and Semantics group
IT University of Copenhagen
Denmark

Bjørn Haagensen[5] Hans Hüttel[6]

Department of Computer Science
Aalborg University
Denmark

Abstract

Homer is a higher order process calculus with locations. In this paper we study Homer in the setting of the semantic finite control property, which is a finite reachability criterion that implies decidability of barbed bisimilarity. We show that strong and weak barbed bisimilarity are undecidable for Homer. We then identify and compare two distinct subcalculi of Homer that both satisfy the semantic finite control property. One subcalculus is obtained by using a type system bounding the size of process terms. The other subcalculus is obtained by considering the image of the encoding of the finite control π-calculus in Homer.

Keywords: Decidability, higher order process passing, locations, semantic finite control

1 Introduction

The calculus Homer [7] is a higher order process calculus with nested location hierarchies and active process mobility. Its syntax and semantics are inspired by calculi such as Plain CHOCS [17] and the higher order π-calculus [15]. Similar to these

[1] Supported by grant no. 274-06-0415 and 2059-03-0031 from the Danish Research Council for Technology and Production and the IT University of Copenhagen (the CosmoBiz and BPL projects).
[2] Supported by grant no. 272-05-0258 from the Danish Research Agency.
[3] Email: mikkelbu@itu.dk
[4] Email: jcg@itu.dk
[5] Email: bh@cs.aau.dk
[6] Email: hans@cs.aau.dk

doi:10.1016/j.entcs.2009.06.016

calculi we have the ability to send a *passive* resource r (along the name a),

$$\overline{a}\langle r\rangle.p \,|\, a(x).q \longrightarrow p \,|\, q\{r/x\} \ .$$

Active process mobility and nested location hierarchies are introduced in the calculus by the location prefix, $a\langle r\rangle.p$, where r is an *active* resource computing at the location a. A process can take an active resource and bind it to a process variable using the complementary prefix $\overline{a}(x).q$ according to the following reduction rule

$$a\langle r\rangle.p \,|\, \overline{a}(x).q \longrightarrow p \,|\, q\{r/x\} \ .$$

We can communicate with processes residing in locations by allowing sequences of names in the prefixes. E.g. we can take the resource r from the location b inside the location a using the composite address ab

$$a\langle b\langle r\rangle \,|\, p'\rangle.p \,|\, \overline{ab}(x).q \longrightarrow a\langle p'\rangle.p \,|\, q\{r/x\} \ .$$

In a similar manner we can send a passive resource to a receiver residing in a sublocation.

$$a\langle b(x).p' \,|\, p''\rangle.p \,|\, \overline{ab}\langle r\rangle.q \longrightarrow a\langle p'\{r/x\} \,|\, p''\rangle.p \,|\, q \ .$$

Homer can encode persistent locations [7], mobility as in the Seal calculus [9], and name passing as in the π-calculus [1,2], thus exemplifying some of its expressive power. The purpose of this work is to investigate decidability of barbed bisimilarity in Homer with all operators. Results of this type are useful as a basis for model- and equivalence-checking. Apart from the results mentioned below, few results of this type exist in the context of higher order calculi with locations, and the question is non-trivial since Homer can encode Turing machines.

Intuitively, a *finite control* calculus is a calculus where the control structure is finite. I.e. starting in any state, the number of states reachable via internal reduction steps are finite. This paper shows that in a full higher order calculus with locations, finite control [6] is a complicated issue. In the context of CCS [12] and the π-calculus [16] it has been shown that finite control can be obtained simply by prohibiting the use of the operator for parallel composition in recursive definitions [6]. The solution is not equally simple in higher order calculi such as Homer and $HO\pi$. There are several reasons for this. First, there is no explicit recursion or replication operator in Homer since recursion is a derived operator [9]. Moreover, process-variables may be instantiated with arbitrary processes. But most important is the observation that even without using parallel composition in recursion, one can define a process with infinitely many non-barbed bisimilar reducts. We can construct such a process in Homer by using that process variables can occur at sublocations as in $a(x).\overline{a}\langle n\langle x\rangle\rangle$, where an extra level of nesting (the location n) is added to the process received on channel a.

In order to find a decidable characterisation of a subcalculus of Homer for which barbed bisimilarity is decidable we explore two different approaches. The first approach is to use a type system which bounds the size of processes in terms of the

number of parallel components, sequential length, and nesting of locations. The resulting subcalculus of Homer is called HFC_Γ. The resulting calculus is too restrictive and does not allow for infinite reductions. Therefore a recursion operator is added to Homer. Since processes in HFC_Γ cannot acquire new free names, this ensures us that there are only finitely many different α-equivalence classes reachable from any process. The second approach is to consider an encoding of the π-calculus into Homer [1,2]. We apply it to the finite control π-calculus, $FC\pi$, and consider the image of the encoding as a subcalculus of Homer, HFC_π. It is shown that the finite control property is preserved by the encoding.

HFC_π as well as HFC_Γ are subcalculi of the full calculus Homer, and the finiteness results for HFC_π and HFC_Γ imply that the inclusions are strict. Moreover we show that there are HFC_Γ-processes which do not have semantically equivalent counterparts in HFC_π. For the converse, there are HFC_π-processes which are not well-typed as HFC_Γ-processes. However it is an open question as to whether there exists an semantically equivalent HFC_π-process which is well-typed.

Related work

The extent to which higher order communication adds to the expressiveness of the first order π-calculus has been studied in [16], where it is shown that one can encode the higher order π-calculus, $HO\pi$, in the first order π-calculus by passing "triggers" instead of passing processes. However the encoding breaks down, when we introduce locations to the calculi. So the results for the π-calculus and $HO\pi$ are not directly applicable in our setting. In the context of calculi with hierarchical locations, the work on the calculus of Mobile Ambient [4] is related. The expressive power of different of subcalculi of Mobile Ambients have been examined in [18], [3], and [11]. In [18] the expressive power of Pure Safe Ambient Calculus is examined by giving an encoding of the synchronous π-calculus. In [3] it is examined whether restriction and ambient movement can be removed from the pure mobile ambient calculus without losing expressive power. Building upon this [11] examines the connection between operators and minimal Turing-complete fragments. In the paper it is shown that Turing completeness can be achieved merely using the movement capabilities of ambients.

Closer related to the subject of the present paper is [5]. In the paper the authors examine a finite control fragment of the ambient calculus. Similar to one of the approaches examined in this paper the finite control fragment is obtained by the usage of a type system instead of, as usual, relying on syntactic restrictions. However, the Ambient Calculus cannot be considered be be *higher-order* in the sense that the values exchanged in communications contain processes. In the Ambient Calculus processes can move around in the location hierarchy, but they cannot be copied or discarded as part of a synchronisation, hence the communication is essential *linear*, as opposed to *non-linear* as in Homer or $HO\pi$.

Recent work in [10] considers a *minimal* variant of the $HO\pi$-calculus. Contrary to the full $HO\pi$-calculus this variant has asynchronous output and no name-restriction. The resulting calculus is shown to be Turing complete, hence its termi-

nation problem is undecidable. However, perhaps surprisingly, it holds that strong bisimilarity is decidable, which in turn implies that barbed congruence is decidable. It is also shown that if at least four static (i.e., top-level) restrictions are added to the calculus then strong bisimilarity becomes undecidable.

Structure of the paper

In Section 2 we present the syntax and the reduction semantics of Homer, and we define the notion of semantic finite control and give an indication of the expressiveness of calculi satisfying the semantic finite control property. In Section refsec:undec-results we prove that strong/weak barbed bisimilarity are undecidable. Therefore we present two fragments of Homer where barbed bisimilarity is decidable: in Section 4 we define HFC_Γ using a type system, and in Section 5 we define HFC_π using an encoding of the finite control π-calculus. We compare the calculi in Section 6. Finally in Section 7 we conclude and propose future work.

2 The Calculus Homer

The syntax and semantics of Homer as presented by Bundgaard et. al. in [1] are given as follows. Let $\mathcal{N}$ be an infinite set of names and let $\mathcal{N}^*$ denote the set of all sequences of names formed by using names from $\mathcal{N}$, let $\mathcal{N}^+ \subset \mathcal{N}^*$ denote the set of non-empty sequences of names, and let $\tilde{n}$ range over finite sets of names. Let $a, b, n, m, \ldots$ range over $\mathcal{N}$, γ over $\mathcal{N}^*$ and δ over $\mathcal{N}^+$. Let $\mathcal{V}$ be an infinite set of process variables ranged over by $x, y, z, \ldots$. Finally let $\mathcal{U}$ be a set of recursion variables ranged over by X and Y. The set of Homer processes is given by the following grammar.

$$\begin{array}{l} p ::= \mathbf{0} \quad | \quad \mathrm{rec}\, X.p \quad | \quad p \,|\, p' \quad | \quad (n)p \quad | \quad \pi.p \quad | \quad x \quad | \quad X \\ \pi ::= \delta(x) \quad | \quad \overline{\delta}(x) \quad | \quad \overline{\delta}\langle p \rangle \quad | \quad \delta\langle p \rangle \end{array}$$

The primitives for the inactive process, recursion, parallel composition, and restriction have the same meaning as in other higher order process calculi. There are two prefixes representing a resource at a location δ, where δ is a sequence of names enabling addressing at sub-locations as described in the introduction. The active $\delta\langle p \rangle$, and the passive $\overline{\delta}\langle p \rangle$ prefix. The process p can perform internal reactions in $\delta\langle p \rangle$, and the context can communicate with p; this is not the case for p in $\overline{\delta}\langle p \rangle$. In Homer names are bound by restriction, $(n)p$, and process variables are bound by $\delta(x).p$ or $\overline{\delta}(x).p$, and recursion variables are bound as in $\mathrm{rec}\, X.p$. For a process p the set of free and bound names and variables are defined accordingly and denoted $\mathrm{fn}(p), \mathrm{bn}(p), \mathrm{fv}(p)$ and $\mathrm{bv}(p)$. We will often omit trailing occurrences of $\mathbf{0}$ in e.g. $\delta\langle p \rangle.\mathbf{0}$. We will also let φ range over δ and $\overline{\delta}$.

Let $\equiv_\alpha$ denote α-equivalence both with respect to names and variables. If $\mathrm{fv}(p) = \emptyset$, then p is called a *closed* process. Let $\mathcal{P}$ denote the set of processes given by the grammar (up to α-equivalence) and let $p, q, r, \ldots$ range over $\mathcal{P}$. Furthermore let $\mathcal{P}_c \subset \mathcal{P}$ denote the set of closed processes ranged over by the same meta-variables as $\mathcal{P}$. Contexts are defined as process terms with a single hole.

Definition 2.1 (Contexts) *Homer* contexts $\mathscr{C}$ *and* evaluation contexts $\mathscr{E}$ *are given by the following grammars:*

$$\mathscr{C} ::= (-) \mid \operatorname{rec} X.\mathscr{C} \mid \mathscr{C} \mid p \mid (n)\mathscr{C} \mid \pi.\mathscr{C} \mid \delta\langle\mathscr{C}\rangle.p \mid \overline{\delta}\langle\mathscr{C}\rangle.p$$
$$\mathscr{E} ::= (-) \mid \mathscr{E} \mid p \mid (n)\mathscr{E} \mid \delta\langle\mathscr{E}\rangle.p'$$

A recursion variable X is said to be guarded in p if its occurces are always underneath some prefix π.

Definition 2.2 (Linearity and guarded) *Let $p \in \mathcal{P}_c$. Then p is* linear *if for every sub-process $\delta(x).q$ or $\overline{\delta}(x).q$ of p, x occurs free at most once in q. A process p is* guarded *if for every occurrence of* $\operatorname{rec} X.p'$ *in p, all occurrences of X in p' are guarded.*

From now on we only consider guarded processes. Structural congruence is the least equivalence relation on $\equiv \subseteq \mathcal{P} \times \mathcal{P}$ which is closed under application of process contexts and which satisfies the following axioms.

$$p \mid \mathbf{0} \equiv p \qquad p \mid q \equiv q \mid p \qquad p \mid (q \mid r) \equiv (p \mid q) \mid r$$
$$(n)\mathbf{0} \equiv \mathbf{0} \qquad (n)p \mid q \equiv (n)(p \mid q), \text{ where } n \notin \operatorname{fn}(q) \qquad (n)(m)p \equiv (m)(n)p$$
$$\operatorname{rec} X.X \equiv \mathbf{0} \qquad \operatorname{rec} X.p \equiv p\{\operatorname{rec} X.p/X\}$$

As usual we also identify processes up to α-conversion. In order to handle addressing at sublocations the reduction rules are given using so-called path indexed contexts, $\mathscr{C}^{\tilde{m}}_{\gamma}$, where γ is the path to the hole, and $\tilde{m}$ the names bound in the hole.

Definition 2.3 (Path-indexed contexts) *Let $p, q \in \mathcal{P}_c$ and $\delta \in \mathcal{N}^+$ and $\gamma \in \mathcal{N}^*$. Then inductively define* path-indexed contexts *by*

$$\mathscr{C}^{\emptyset}_{\epsilon} \stackrel{def}{=} (-) \qquad \mathscr{C}^{\tilde{n}\tilde{m}}_{\delta\gamma} \stackrel{def}{=} \delta\langle(\tilde{n})\mathscr{C}^{\tilde{m}}_{\gamma} \mid p)\rangle.q, \text{ where } \tilde{n} \cap \gamma = \emptyset \ .$$

Vertical scope extrusion is defined using an open operator *on path contexts.*

$$\mathscr{C}^{\emptyset}_{\epsilon} \setminus \tilde{o} \stackrel{def}{=} \mathscr{C}^{\emptyset}_{\epsilon} \qquad \mathscr{C}^{\tilde{n}\tilde{m}}_{\delta\gamma} \setminus \tilde{o} \stackrel{def}{=} \delta\langle(\tilde{n} \setminus \tilde{o})\mathscr{C}^{\tilde{m}}_{\gamma} \setminus \tilde{o} \mid p\rangle.q,$$
$$\text{if } \mathscr{C}^{\tilde{n}\tilde{m}}_{\delta\gamma} = \delta\langle(\tilde{n})\mathscr{C}^{\tilde{m}}_{\gamma} \mid p\rangle.q \text{ and } (\tilde{m} \cup \tilde{n}) \cap \operatorname{fn}(\mathscr{C}^{\tilde{m}\tilde{n}}_{\delta\gamma}) = \emptyset \ .$$

When a resource is moved from a location it may be necessary to extend the scope of a name vertical through the location boundary using the open operator. For instance given the path context $\mathscr{C} = n\langle(m, m')n'\langle(-)\rangle\rangle$, and for simplicity assume that all names are distinct, then we can apply the open operator on $\mathscr{C}$ with the argument m' (i.e. $\mathscr{C} \setminus m'$) obtaining the path context $n\langle(m)n'\langle(-)\rangle\rangle$.

Definition 2.4 (Reduction relation) *The* reduction relation *is the least binary relation on $\mathcal{P}_c$ which is closed under structural congruence and evaluation contexts*

and which satisfies the following axioms.

$$(\textsc{Send})\ \overline{\gamma\delta}\langle p\rangle.p' \mid \mathscr{C}^{\tilde{m}}_{\gamma}(\delta(x).q) \longrightarrow p' \mid \mathscr{C}^{\tilde{m}}_{\gamma}(q\{p/x\}) \quad \textit{where } \tilde{m} \cap (\mathrm{fn}(p) \cup \delta) = \emptyset$$

$$(\textsc{Take})\ \mathscr{C}^{\tilde{m}}_{\gamma}(\delta\langle p\rangle.p') \mid \overline{\gamma\delta}(x).q \longrightarrow (\tilde{m} \cap \tilde{n})\big((\mathscr{C}^{\tilde{m}}_{\gamma} \setminus \tilde{n})(p') \mid q\{p/x\}\big) \quad \textit{where } \tilde{n} = \mathrm{fn}(p), \tilde{m} \cap (\delta \cup \mathrm{fn}(q)) = \emptyset$$

In general, we allow interaction with arbitrarily deeply nested subprocesses. However, note that two processes that are neither locally parallel nor in the sub/super process relation in the location hierarchy need a common super process to act as a router that mediates communication. For instance, the processes at locations n and m in the process

$$n\langle b\langle r\rangle.q' \mid q''\rangle \mid m\langle b(x).p' \mid p''\rangle \ ,$$

cannot communicate synchronously with each other, and need the super process at the top level to first retrieve the process r from location b inside location n and then pass the resource on to the receiving prefix inside location m. As illustrated by the following sequence of reductions.

$$\begin{aligned} &n\langle b\langle r\rangle.q' \mid q''\rangle \mid \overline{nb}(x).\overline{mb}\langle x\rangle \mid m\langle b(x).p' \mid p''\rangle \longrightarrow \\ &n\langle q' \mid q''\rangle \mid \overline{mb}\langle r\rangle \mid m\langle b(x).p' \mid p''\rangle \longrightarrow \\ &n\langle q' \mid q''\rangle \mid m\langle p'\{r/x\} \mid p''\rangle \ . \end{aligned}$$

Note that in the (Take)-rule the names bound in the hole are vertically extruded if and only if they are actually free in p. For example in

$$a\langle (n)(b\langle r\rangle \mid p)\rangle \mid \overline{ab}(y).(y \mid y) \ ,$$

the scope of n is extruded (through the location boundary of a) iff n is free in r, so each copy of r will share the name n. Otherwise r leaves the scope of n. A detailed discussion of this choice is presented in [9]. The rest of the side conditions in Definition 2.3 and Definition 2.4 are standard and prevent free names from being captured. Let $\longrightarrow^*$ denote the transitive and reflexive closure of $\longrightarrow$.

We define strong and weak barbs in the usual manner, i.e. the possibility of observing a prefix (possibly residing in the location hierarchy).

Definition 2.5 (Strong and weak barbs) *Define:*

- $p \downarrow_{\mathtt{o}} \gamma\delta$ *if* $p \equiv \mathscr{C}^{\tilde{m}}_{\gamma}((\tilde{m}')(\delta\langle q\rangle.q' \mid q''))$, *where* $\delta \cap (\tilde{m} \cup \tilde{m}') = \emptyset$.
- $p \downarrow_{\mathtt{i}} \gamma\delta$ *if* $p \equiv \mathscr{C}^{\tilde{m}}_{\gamma}((\tilde{m}')(\delta(x).q' \mid q''))$, *where* $\delta \cap (\tilde{m} \cup \tilde{m}') = \emptyset$.
- $p \downarrow_{\mathtt{o}} \overline{\delta}$ *if* $p \equiv (\tilde{m}')(\overline{\delta}\langle q\rangle.q' \mid q'')$, *where* $\overline{\delta} \cap \tilde{m}' = \emptyset$.
- $p \downarrow_{\mathtt{i}} \overline{\delta}$ *if* $p \equiv (\tilde{m}')(\overline{\delta}(x).q' \mid q'')$, *where* $\overline{\delta} \cap \tilde{m}' = \emptyset$.

We will let $p \downarrow \varphi$ range over $p \downarrow_{\mathtt{o}} \varphi$ and $p \downarrow_{\mathtt{i}} \varphi$. (Recall, that we let φ range over δ and $\overline{\delta}$.) Weak barbs *are then defined in terms of strong barbs in the usual manner.*

- *$p \Downarrow \varphi$ if there is p' such that $p \longrightarrow^* p'$ and $p' \downarrow \varphi$.*

To illustrate our notion of barbs consider the following process.

$$p = (n)\left(m\langle n'\langle q\rangle\rangle \mid \overline{a}\langle q'''\rangle\rangle \mid (m')(n''(x).q'') \mid \overline{b}(x').p'\right)$$

The process p has the following barbs: $p \downarrow_{\mathtt{o}} m$, $p \downarrow_{\mathtt{o}} mn'$, $p \downarrow_{\mathtt{i}} n''$, and $p \downarrow_{\mathtt{i}} \overline{b}$ (assuming that $\{n\} \cap \{m, n', n'', b\} = \emptyset$ and $\{m'\} \cap \{n''\} = \emptyset$).

Definition 2.6 (Strong and weak barbed bisimilarity) *A binary symmetric relation $\mathcal{R} \subseteq \mathcal{P}_c \times \mathcal{P}_c$ is called a* strong barbed bisimulation *if whenever $(p, q) \in \mathcal{R}$ the following holds:*

(i) *If $p \downarrow \varphi$ then $q \downarrow \varphi$*

(ii) *If $p \longrightarrow p'$ then $q \longrightarrow q'$ and $(p', q') \in \mathcal{R}$.*

Processes p and q are called strong barbed bisimilar, denoted $p \sim q$, if there is a strong barbed bisimulation $\mathcal{R}$ such that $(p, q) \in \mathcal{R}$.

Weak barbed bisimilarity *is obtained by replacing (i) and (ii) with:*

(i) *If $p \downarrow \varphi$ then $q \Downarrow \varphi$*

(ii) *If $p \longrightarrow p'$ then $q \longrightarrow^* q'$ and $(p', q') \in \mathcal{R}$.*

Processes p and q are weakly bisimilar, denoted $p \approx q$, if there is a weak bisimulation *such that $(p, q) \in \mathcal{R}$.*

For rec-free well-formed processes the following finiteness property can be proven by observing that all reductions strictly reduce the size of processes. The corresponding result does not hold for full Homer.

Lemma 2.7 *If p is a linear process term built from the grammar without resorting to the* rec $X.p$ *construction, then $\{p' \mid p \longrightarrow^* p'\}/_{\equiv}$ is finite.*

Strong, respectively weak, barbed bisimilarity are too coarse for many purposes, specifically in most cases only the congruence versions coincide with strong and weak bisimilarity. Indeed this is the case for Homer [9]. The coarsest bisimulation equivalence is *reduction bisimilarity*, $\sim_r$. This holds since reduction bisimilarity only requires equivalent processes to match on τ-actions, whereas they, contrary to barbed bisimilarity, need not have equivalent barbs, i.e. observable actions. It is indeed necessary that $\sim_r$ is decidable for the congruences to be decidable. This is seen since decidability of reduction bisimilarity can be reduced to deciding barbed congruence, $\sim^c$. For any processes p and q let,

$$p \sim_r q \text{ if and only if } (\tilde{n})p \sim^c (\tilde{m})q, \text{ where } \tilde{n} = \mathrm{fn}(p) \text{ and } \tilde{m} = \mathrm{fn}(q) \ .$$

Therefore, even though our decidability results only applies to the non-congruence versions, they must indeed hold in any formalism for which the congruences are

decidable. For the positive results in this paper we will use the following finite control property.

Definition 2.8 (Semantic finite control (SFC)) *Let $\mathcal{A}$ be a process calculus and $\eqslantless$ be some decidable equivalence such that $\eqslantless \subseteq \approx, \sim$. Then $\mathcal{A}$ is called* semantic finite control *up to $\eqslantless$ if the set $\{p' \mid p \longrightarrow^* p'\}/_{\eqslantless}$ is finite for any process $p \in \mathcal{A}$.*

Lemma 2.9 *If p is SFC up to $\eqslantless$, then the set*

$$A = \{p' \mid p \longrightarrow^* p'\}/_{\eqslantless}$$

is computable.

Proof. Either p terminates, or p does not terminate. By assumption A is finite. Moreover $\longrightarrow$ is finitely branching up to $\eqslantless$. If $p_0 \longrightarrow p_1 \longrightarrow \cdots$ is an infinite reduction sequence the set A can be constructed in a breadth-first manner by considering all outgoing reductions from p_0, p_1, etc. Moreover there must be some p_j such that for some p_i, $i < j$ and $p_i \eqslantless p_j$. When such a p_j is found, noting that $\eqslantless$ is decidable, we know that any reduction from p_j must visit a previously seen state. □

By definition $p \eqslantless q$ implies that $p \sim q$ and $p \approx q$. Hence $\eqslantless$ is said to *respect barbs* in the sense that if $p \eqslantless q$, we have that $p \downarrow \phi$ iff $q \downarrow \phi$, and similarily $p \Downarrow \phi$ iff $q \Downarrow \phi$.

Lemma 2.10 *If p is SFC, then $p \downarrow \varphi$ and $p \Downarrow \varphi$ are decidable.*

Proof. We can decided $p \downarrow \varphi$ by inductively checking whether there are any top-level prefixes with φ in the subject position, or if we can decompose φ into an address and a prefix residing at this address, and that this φ is not restricted. By assumption the set,

$$\{p' \mid p \longrightarrow^* p'\}/_{\eqslantless} \ ,$$

is finite and computable by Lemma 2.9 Therefore, since $\eqslantless$ respects barbs we only need to check whether $p' \downarrow \varphi$ for finitely many p'. □

Proposition 2.11 *If $\mathcal{A}$ is SFC up to $\eqslantless$, then $\approx$ and $\sim$ are decidable.*

Proof. Let p and q be processes in $\mathcal{A}$ and related by either $\approx$ or $\sim$. From the SFC property we know that the sets,

$$\{p' \mid p \longrightarrow^* p'\}/_{\eqslantless} \text{ and } \{q' \mid q \longrightarrow^* q'\}/_{\eqslantless} \ ,$$

are both finite. By Proposition 2.10 strong and weak barbs are decidable. Moreover both sets are computable by Lemma 2.9 so deciding reduction bisimilarity is just a matter of checking whether the two sets contains the same equivalence classes. □

As an indication of the expressiveness of SFC calculi, the next result shows that the traces of processes from calculi satisfying the SFC property are simple in structure. Below we let `b` range over `i` and `o`.

Definition 2.12 (Barbed Trace) *Let $\mathcal{A}$ be a process calculus and let $p \in \mathcal{A}$ be a process. $\alpha = \varphi_1 \mathtt{b}_1 \cdots \varphi_k \mathtt{b}_k$ is a* barbed trace *of p ending in p' if there exists*

$$p \longrightarrow^* \downarrow_{\mathtt{b}_1} \varphi_1 \longrightarrow \longrightarrow^* \downarrow_{\mathtt{b}_2} \varphi_2 \cdots \longrightarrow \longrightarrow^* p' \downarrow_{\mathtt{b}_k} \varphi_k \ .$$

We let $p \xrightarrow{\alpha}{}^* p'$ denote such a reduction sequence. The set of barbed traces generated by p is denoted $BTrace(p)$.

Lemma 2.13 (Pumping Lemma) *Let $\mathcal{A}$ be a process calculus which is SFC up to $\eqsim$ and assume $\eqsim$ respects barbed traces, and let $p \in \mathcal{A}$. Then there is a number n such that if $\alpha \in BTrace(p)$ and $|\alpha| \geq n$, then there are α_1, α_2, and α_3 such that $\alpha = \alpha_1\alpha_2\alpha_3$ and for each $i \in \mathbb{N}$ the following holds:*

(i) $\alpha_1\alpha_2^i\alpha_3 \in BTrace(p)$

(ii) $|\alpha_2| > 0$.

Proof. Let $n = |\{p' \mid p \longrightarrow^* p'\}/_{\eqsim}| + 1$ and α a barbed trace, with $|\alpha| = n'$ and $n' \geq n$. Then there must exist a sequence of transitions visiting at least n' states. Thus by the pigeonhole principle one can now prove that some equivalence class, with respect to $\eqsim$, must repeat. Or more precisely, that at some point a state is reach which belongs to the same equivalence class as some previously seen state. Denote the first of occurrence of this state p_j and the second p_k. Then $p \xrightarrow{\alpha_1}{}^* p_j \xrightarrow{\alpha_2}{}^* p_k \xrightarrow{\alpha_3}{}^* p'$, for some p'. Then it easy to see that $\alpha_1\alpha_2^i\alpha_3 \in BTrace(p)$ for all $i \geq 0$, as we can loop via α_2. □

Corollary 2.14 *Let $\mathcal{A}$ be a process calculus which is SFC up to $\eqsim$ and assume that $\eqsim$ respects barbed traces. Then there is no $p \in \mathcal{A}$ and addresses φ and φ' and $\mathtt{b}$ and $\mathtt{b}'$ such that $BTrace(p) = \{(\varphi\mathtt{b})^i(\varphi'\mathtt{b}')^i \mid \text{for all } i \in N\}$.*

In the following sections we first present some undecidability results, then we characterise SFC processes in two different ways: first using a type system which bounds the size of processes, and second using an encoding of the (finite control) π-calculus into Homer.

3 Undecidability Results

In [1,2] it is shown that Homer can encode the π-calculus. From that result it follows that Homer is Turing-complete. Although Turing-completeness usually implies that semantic properties of processes are undecidable, the recent paper [10] shows that undecidability of barbed congruence does not follow from the ability to encode Minsky machines in a termination preserving manner.

Definition 3.1 *A $\approx$-property $\mathcal{S}$ is a set of $\approx$-equivalence classes. $\mathcal{S}$ is* non-trivial *if there exists a $\mathcal{C}_1 \in \mathcal{S}$ and $\mathcal{C}_2 \notin \mathcal{S}$.*

Theorem 3.2 *If $\mathcal{S}$ is a non-trivial $\approx$-property, then $L_{\mathcal{S}} = \{p \mid \exists\, \mathcal{C} \in \mathcal{S}.\ p \in \mathcal{C}\}$ is undecidable.*

Proof. Reduction from the halting problem for Turing machines. Since $\mathcal{S}$ is non-trivial, there exist equivalence classes $\mathcal{C}_1 \in \mathcal{S}$ and $\mathcal{C}_2 \notin \mathcal{S}$. We choose a process $p_1 \in \mathcal{C}_1$. Further, we assume without loss of generality that $p_2 \in \mathcal{C}_2$, where

$$p_2 = (a)(\mathrm{rec}\, X.a(z).X \mid \mathrm{rec}\, Y.\overline{a}\langle \mathbf{0}\rangle.Y) \ ,$$

is a non-terminating process.

Given a Turing machine M and an input x we can construct a Homer process $p_{M,x}$ whose only free name is w and such that w is only used to signal termination and such that $p_{M,x} \Downarrow_{\mathrm{o}} w$ iff M halts on input x. Now construct the process $p_0 = (w)(p_{M,x} \mid \overline{w}(y).p_1)$ with y fresh for p_1. Then we have that $p_0 \in \mathcal{C}_1$ if M halts on x and that $p_0 \in \mathcal{C}_2$ if M does not halt on x.

M halts on x: Then

$$Id \cup \{(p_0', p_1) \mid p_0 \longrightarrow^* p_0'\} \ ,$$

is a weak bisimulation up to $\equiv$ containing the pair (p_0, p_1).

M does not halt on x: Then

$$Id \cup \{(p_0', p_2) \mid p_0 \longrightarrow^* p_0'\} \ ,$$

is a weak bisimulation up to $\equiv$ containing the pair (p_0, p_2). □

Corollary 3.3 $\approx$ *is undecidable.*

The proof of Theorem 3.2 remains valid also for barbed congruence and even for reduction bisimilarity. Consequently Corollary 3.3 also remains true for both equivalences. The analogous result for strong barbed bisimilarity is obtained by an encoding of the λ-calculus in Homer, inspired by the encoding in Plain CHOCS [17]. Assuming that a and $i \notin \mathrm{fn}(\llbracket M \rrbracket) \cup \mathrm{fn}(\llbracket N \rrbracket)$.

$$\begin{aligned} \llbracket x \rrbracket &\stackrel{\mathrm{def}}{=} x \\ \llbracket \lambda x.M \rrbracket &\stackrel{\mathrm{def}}{=} i(x).\, \llbracket M \rrbracket \\ \llbracket MN \rrbracket &\stackrel{\mathrm{def}}{=} (a)\left(a\langle \llbracket M \rrbracket \rangle \mid \overline{a}i\langle \llbracket N \rrbracket \rangle.\overline{a}(x).x\right) \ . \end{aligned}$$

This encoding, while only being correct up to weak equivalence, is termination preserving, and moreover it takes exactly 2 reduciton-steps for the encoding to simulate a reduction in a λ-term. Therefore, for a λ-term M

$$(i)(\llbracket M \rrbracket) \sim \mathrm{rec}\, X.(a)(\overline{a} \mid a.X) \text{ iff } M \text{ diverges} \ ,$$

which is a reduction from the divergence problem for the λ-calculus to the problem of checking strong barbed bisimilarity. Note that we used CCS-prefixes in the previous statement. The prefixes a and $\overline{a}$ can easily be encoded by passing the inactive process. We will throughout the remaining paper use CCS-prefixes.

4 The Calculus HFC_Γ

In this section we present the typed subcalculus HFC_Γ of Homer. The basic purpose of the type system is to ensure that the size of a typeable process is bounded under reduction, i.e. that for any well-typed process p where $p \longrightarrow^* p'$ the size of p' is bounded by the size of p. Types in the type system are triples of natural numbers. In a type (d, w, s), d, w, and s are upper bounds on the depth of nested locations, width, i.e. the number of parallel processes structurally different from $\mathbf{0}$, and the sequential length of processes (in case of a recursive process it is the highest number of prefixes any recursive process will be able to pass through before a recursive call must be made).

Definition 4.1 (Types) Types *are triples* (d, w, s) *of non-negative natural numbers,* $\mathbb{N}$*, ranged over by* S *and* T*.*

We write $(d, w, s) \leq (d', w', s')$ if $d \leq d', w \leq w'$, and $s \leq s'$.

Definition 4.2 (Type environments) *A* type environment *is a finite partial function* $\Gamma : \mathcal{N} \cup \mathcal{V} \cup \mathcal{U} \hookrightarrow \mathbb{N} \times \mathbb{N} \times \mathbb{N}$.

Type environments can be regarded as finite sets of type assignments $a : T$, where $a \in \mathcal{N} \cup \mathcal{V} \cup \mathcal{U}$ and $T \in \mathbb{N} \times \mathbb{N} \times \mathbb{N}$ and an environment is written $\{a_1 : T_1, \ldots, a_n : T_n\}$ where $a_i \neq a_j$ when $i \neq j$. Type environments can be extended. This is written $\Gamma \cup \{a : T\}$, and is only defined if a is not defined in Γ. Recall that we let φ range over δ and $\overline{\delta}$. There are two type judgement relations for HFC_Γ.

Definition 4.3 *The* type relation for names, $\vdash_n$*, is given by the following rules*

$$(\textsc{TName}) \qquad \Gamma \cup \{a : T\} \vdash_n a : T$$

$$(\textsc{TSeqName}) \quad \frac{\Gamma \vdash_n \delta : T \qquad \Gamma \vdash_n a : S}{\Gamma \vdash_n \delta a : S}$$

Definition 4.4 *The* type relation for processes, $\vdash$*, is given by the following rules*

$$(\textsc{TProcVar}) \quad \Gamma \cup \{x : T\} \vdash x : T$$

$$(\textsc{TRecVar}) \quad \Gamma \cup \{X : T\} \vdash X : T$$

$$(\textsc{TNil}) \qquad \Gamma \vdash \mathbf{0} : (0, 0, 0)$$

$$(\textsc{TNew}) \quad \frac{\Gamma \cup \{n : S\} \vdash p : T}{\Gamma \vdash (n)p : T}$$

$$(\textsc{TPar}) \quad \frac{\Gamma \vdash p : (d, w, s) \qquad \Gamma \vdash q : (d', w', s')}{\Gamma \vdash p \,|\, q : (\max\{d, d'\}, w + w', s + s')}$$

$$(\textsc{TTake}|\textsc{TIn}) \quad \frac{\Gamma \cup \{x : S\} \vdash p : (d', w', s') \qquad \Gamma \vdash_n \varphi : T}{\Gamma \vdash \varphi(x).p : (d', \max\{w', 1\}, 1 + s')}, \text{ if } T \leq S$$

$$(\text{TSend}|\text{TOut})\quad \frac{\Gamma \vdash_n \varphi : (d', w', s') \qquad \Gamma \vdash p : (d', w', s') \qquad \Gamma \vdash q : (d, w, s)}{\Gamma \vdash \varphi\langle p\rangle.q : (\max\{d'+1, d\}, \max\{w', w, 1\}, \max\{s', s+1\})}$$

$$(\text{TSub})\quad \frac{\Gamma \vdash p : T}{\Gamma \vdash p : T'},\ \textit{if } T \leq T'$$

$$(\text{TRec})\quad \frac{\Gamma \cup \{X : (d, w, 0)\} \vdash p : (d, w, s)}{\Gamma \vdash \operatorname{rec} X.p : (d, w, s)}$$

The meaning of the rules are fairly self-explanatory. The (TRec) rule enforces, through its side-conditions, that the recursion variable cannot be placed freely in p. In particular, in $\operatorname{rec} X.p$, if the recursion variable occurs free in p, then there cannot be any occurrences of $\mid$ in p. This is similar to the finite control condition in $FC\pi$. Moreover, the recursion variable cannot be placed inside a location either. Hence a well-typed recursive process will have the following form $\operatorname{rec} X.\pi_1.\pi_2.\ldots.\pi_n.X$ (recalling that π ranges over prefixes).

The rules (TSend|TOut) express that a resource residing in a location should be typable with the type of the address. The rules (TTake|TIn) express that we should treat input as possibly having a larger type than the type of the address. Together these rules enforce that we cannot type processes such as

$$a(x).\overline{a}\langle n\langle x\rangle\rangle \quad \text{or} \quad a(x).\overline{a}\langle \pi.x\rangle \ ,$$

which could be used as part of counter-examples to SFC. In the following, the notation is overloaded so $\vdash$ denotes $\vdash$ as well as $\vdash_n$ relying on the context to make it clear which one is meant.

Definition 4.5 (Well-typedness) *A process p is* well-typed *if there is some Γ and T such that $\Gamma \vdash p : T$.*

We can prove the usual lemmas for (most) type systems: strengthening, weakening, and substitution. All the lemmas can be proven by induction in the derivation of the typing.

Lemma 4.6 (Strengthening) *If $\Gamma \cup \{x : T\} \vdash p : S$ and $x \notin \text{fv}(p)$, then $\Gamma \vdash p : S$.*

Lemma 4.7 (Weakening) *If $\Gamma \vdash p : S$ then $\Gamma \cup \{u : T\} \vdash p : S$ for any T and u such that $u \notin \text{dom}(\Gamma)$.*

Lemma 4.8 (Substitution lemma) *If $\Gamma \cup \{x : S\} \vdash p : T$ and $\Gamma \vdash q : S'$ with $S' \leq S$ then it holds that $\Gamma \vdash p\{q/x\} : T'$ where $T' \leq T$.*

Corollary 4.9 *Suppose $\Gamma \vdash \mathscr{C}^{\tilde{m}}_{\gamma}(\delta(x).q) : T$, $\Gamma \vdash \delta : S$, and $\Gamma \vdash p : S'$, where $S' \leq S$. Then $\Gamma \vdash \mathscr{C}^{\tilde{m}}_{\gamma}(q\{p/x\}) : T'$, where $T' \leq T$.*

The following results establish subject reduction for well-typed processes. First note that the type prescribed to some process need not be unique. For instance the process

$$\operatorname{rec} X.n\langle \mathbf{0}\rangle.X$$

can be typed with the type $(d, w, 1)$ for all d and w larger than 1, assuming that the type environment assigns $(0, 0, 0)$ to n. And in general we can apply subsumption in any typing derivation. Second the type prescribed to a process by the type system is not stable with respect to folding and unfolding of recursive definitions. Therefore we will be interested in the least type prescribed to a process satisfying certain structural properties which we will introduce below.

Definition 4.10 (Least type) *Let p be a process. If $\Gamma \vdash p : T$, then T is called the* least type *of p with respect to Γ if for all T' such that $\Gamma \vdash p : T'$ it holds that $T \leq T'$. We write $\Gamma \vdash_\mu p : T$ if $\Gamma \vdash p : T$ where T is the least type of p with respect to Γ.*

As a first step towards our normal form we first transform processes to a folded form, i.e. we fold all recursive definitions as much as possible, and we remove unnecessary restrictions and inactive processes.

Definition 4.11 (Folding) *We define the binary relation* fold*, written $>$, on process terms by the axioms*

$$p \,|\, \mathbf{0} > p \qquad \mathbf{0} \,|\, p > p \qquad (n)p > p, \text{ if } n \notin \mathrm{fn}(p)$$

$$(n)(p \,|\, q) > (n)p \,|\, q, \text{ if } n \notin \mathrm{fn}(q) \qquad (n)(p \,|\, q) > p \,|\, (n)q, \text{ if } n \notin \mathrm{fn}(p)$$

$$p\{\mathrm{rec}\, X.p/X\} > \mathrm{rec}\, X.p, \text{ if } X \in \mathrm{fv}(p)$$

and closure under process contexts. p is said to be on folded form *if $p \not>$.*

Let $>^*$ denote the transitive closure of $>$. It is easy to see that any process is either on folded form, or can be brought on folded form.

Lemma 4.12 *For any p either p is on folded form, or there is some p' such that $p >^* p'$ and p' is on folded form.*

Moreover the relation $>$ is convergent and a sub-relation of $\equiv$. Therefore we will often assume that processes are on folded form (or a variant of folded form) for the remaining part of this section.

By requiring that processes are on folded form we can avoid problems such as structural congruence unnecessarily unfolding recursive definitions, adding $\mathbf{0}$-processes in parallel, or adding restrictions. However it is not sufficient to consider processes on folded form as the following process illustrates

$$p \,|\, q, \text{ where } p = \mathrm{rec}\, X.b.b.X \text{ and } q = \mathrm{rec}\, Y.\overline{b}.\overline{b}.Y \ .$$

Now $p \,|\, q$ is on folded form, and the s-component of the least type of p and q is 2, so the s-component of the least type of $p \,|\, q$ is 4. But the two processes can perform one reduction as follows

$$p \,|\, q \longrightarrow b.\,\mathrm{rec}\, X.b.b.X \,|\, \overline{b}.\,\mathrm{rec}\, Y.\overline{b}.\overline{b}.Y \ .$$

In the reduct the two parallel components are both on folded form, but the s-component of their least type is 6. So it seems that we need to be able to let the s-component grow under reduction. However for any process p and reduct p' of p, there is a uniform upper bound on the s-component of p' which only depends on p and which is closely related to the unfolding of recursive definitions.

We define a function, OUF (once-unfold) from processes to processes. The purpose of the function is to unfold every recursive definition exactly once. To define OUF we use an auxiliary function OUF' from processes and sets of recursion variables to processes. OUF' is defined as follows.

$$\begin{aligned}
\text{OUF'}(p, \emptyset) &= p \\
\text{OUF'}(\mathbf{0}, \chi) &= \mathbf{0} \\
\text{OUF'}(p_1 \mid p_2, \chi) &= \text{OUF'}(p_1, \chi) \mid \text{OUF'}(p_2, \chi) \\
\text{OUF'}(\varphi\langle p_1\rangle.p_2, \chi) &= \varphi\langle \text{OUF'}(p_1, \chi)\rangle.\, \text{OUF'}(p_2, \chi) \\
\text{OUF'}(\varphi(x).p, \chi) &= \varphi(x).\, \text{OUF'}(p, \chi) \\
\text{OUF'}((n)p, \chi) &= (n)\, \text{OUF'}(p, \chi) \\
\text{OUF'}(\text{rec}\, X.p, \chi) &= \begin{cases} \text{OUF'}(p\{\text{rec}\, X.p/X\}, \chi \setminus X) & \text{, if } X \in \chi \\ \text{rec}\, X.\, \text{OUF'}(p, \chi) & \text{, if } X \notin \chi \end{cases}
\end{aligned}$$

Let p be a closed process. We can then make sure that all recursive definitions use distinct recursion-variables by α-converting, so assume for the rest of this document that all recursion variables are pairwise distinct. Note that as part of the unfolding of nested recursive definitions, e.g. $\text{rec}\, X.a\langle \text{rec}\, Y.\pi.Y\rangle.X$, we can obtain a process where there are two (or more) recursive definitions using the "same" recursion variable. However, this will not affect the following results. Now let Ξ denote the set of recursion variables occurring in p and let $\text{OUF}(p) = \text{OUF'}(p, \Xi)$.

Lemma 4.13 *Let p be a closed process, then there is p' such that $p' = \text{OUF}(p)$.*

Proof (Sketch). We argue that OUF' terminates by observing that in all cases exactly one of the following holds:

- the syntactical size of the remaining process decreases
- the size of the set χ of remaining recursion variables decreases
- the function terminates directly yielding a process.

□

Lemma 4.14 *If $p' = \text{OUF}(p)$, then $p \equiv p'$.*

Proof (Sketch). By examining all the cases defining OUF' one sees that each case yields a structurally equivalent process. □

Recall that we have shown that all processes can be brought on folded form where all occurrences of recursion are folded as much as possible.

Definition 4.15 *A process p is said to be on* once-unfolded-form *(*OUFF*) if there is some p' such that p' is on folded form and $p = \text{OUF}(p')$. Whenever $p = \text{OUF}(p')$*

for some p' on folded form we say that p is the OUFF of p'.

Definition 4.16 *p is* rec-guarded *if every occurrence of* $\mathrm{rec}\, X.p'$ *in p occurs underneath a prefix. If p is not rec-guarded, then p is* rec-exposed.

As stated above structural congruence does not preserve types due to the folding and unfolding of recursive definitions. Letting $\equiv_m$ denote structural congruence without the two axioms for recursive definitions we have the following weaker result.

Lemma 4.17 *Suppose $p \equiv_m q$ then $\Gamma \vdash_\mu p : T$ if and only if $\Gamma \vdash_\mu q : T$ and p rec-guarded if and only q rec-guarded.*

The idea of subject reduction is roughly that we will only unfold recursive definition in a process when the definition is rec-exposed. Moreover a process p is brought on a particular form (OUFF) before initially typed ensuring that the type of p is "large" enough to capture all reducts of p. With this type system we can show that the number of different well-typed processes reachable starting from any well-typed process is finite up to $\equiv$. We will call a reduction $p \longrightarrow p'$ for *consuming* if the unfolding axiom was not as part of the structural congruence used in inferring the reduction.

Theorem 4.18 (Subject reduction) *Suppose that p is rec-guarded and $\Gamma \vdash_\mu p : T$, then for all p' with $p \longrightarrow p'$, where $\longrightarrow$ is consuming, we have $\Gamma \vdash_\mu p' : T'$ for some T' such that $T' \leq T$.*

For readability we have placed the proof of this theorem in the appendix.

Corollary 4.19 *Let p be on OUFF and suppose $\Gamma \vdash_\mu p : T$ then for all p' on folded form where $p \longrightarrow^* p'$ it holds that $\Gamma \vdash_\mu p' : T$.*

Proof (Sketch). The idea behind the proof is to analyze how much the sequential coordinate of a type can change as the result of a series of reductions (note that both the depth and the width coordinate can only decrease); some additional bookkeeping (but essentially the same line of reasoning) is needed, if reductions happen within a location. Since we only consider process up to structural congruence in the following results we can consider a reduction sequence such as

$$p = p_0 \longrightarrow p_1 \longrightarrow \ldots \longrightarrow p_k \ldots \tag{1}$$

where p_0 is on OUFF and hence rec-guarded, and $\Gamma \vdash_\mu p_0 : (d, w, s)$. For each p_i, $0 \leq i \leq k$ we can apply Theorem 4.18 to get an upper bound on the type. Now suppose p_k is the first successor of p_0 that fails to be rec-guarded. p_k must have arisen as the result of a series of communications that have left the subprocesses in the set $E = \{q^1, \ldots, q^j\}$ rec-exposed. By the proof of Theorem 4.18 any process $q^i \in E$ must have a type (d^i, w^i, s^i) where $s^i < s$. Moreover, the sum $\sum_{i=1}^{j} s^i \leq s - 2k$, since all q^i's have contributed with a total of at least $2k$ prefixes to the reduction sequence. Let p'_k be the OUFF of p_k which we basically obtain by applying OUF on all processes in E. However, this can only increase the s-component of p'_k by $2k$

(since it took k steps to get from an OUFF to a rec-exposed form). Finally note that for any well-typed process its folded form has a smaller type. □

Lemma 4.20 *Let $p \longrightarrow^* p'$. Then $\mathrm{fn}(p') \subseteq \mathrm{fn}(p)$.*

Proposition 4.21 *Let $\tilde{n} \subset \mathcal{N}$ be a finite set of names. Then for all types T', there are only finitely many α-inequivalent processes p such that*

- *p is on folded form*
- *$\Gamma \vdash_\mu p : T$, where $T \leq T'$*
- *$\mathrm{fn}(p) \cup \mathrm{bn}(p) \subseteq \tilde{n}$*

Proposition 4.22 *The set of reachable configurations is finite, i.e.*

$$|\{p' \mid \Gamma \vdash_\mu p : T, \text{ and } p \longrightarrow^* p'\}|/_{\equiv} < \infty \ .$$

Proposition 4.22 says that for a well-typed process there are only finitely many reachable processes up to $\equiv$.

Lemma 4.23 *Assume $\Gamma \vdash_\mu p : T$. Then $p \downarrow \varphi$ and $p \Downarrow \varphi$ are decidable.*

We will write HFC_Γ for the subcalculus of Homer consisting of all well-typed processes.

Theorem 4.24 *Strong and weak barbed bisimilarity are decidable for HFC_Γ.*

We now show that within the current setting the types bounding the depth, width, and length of prefix sequences are all necessary to obtain finite control. In the following let HFC_Γ^{-d}, HFC_Γ^{-w}, and HFC_Γ^{-s} denote subcalculi of Homer defined in the same way as HFC_Γ, but with the type-system slightly modified by removing the i'th component of the types and adapting the rules accordingly. The corresponding typing judgements are $\Gamma^{-d} \vdash_\mu p$, $\Gamma^{-w} \vdash_\mu p$, and $\Gamma^{-s} \vdash_\mu p$ respectively.

Proposition 4.25 *Strong/weak barbed bisimilarity are undecidable for HFC_Γ^{-d}.*

Proof. The proposition is proven by showing that HFC_Γ^{-d} can encode Minsky machines [14] in a deterministic and termination preserving manner. A Minsky machine consists of a list of instructions $\{L_1, \ldots L_k\}$ (indexed by i) where the instructions operates on two counters c_1 and c_2. Each instruction L_i is either $\mathrm{Inc}(c_j, n)$, which increments the value of counter c_j and jumps to the next instruction n, or $\mathrm{Dec}(c_j, n, m)$ which jumps to instruction n if $c_j = 0$, otherwise the counter c_j is decremented by 1 followed by a jump to instruction m. A *program counter* (PC) keeps track of the current executing instruction. Execution starts with the first instruction and halts if the PC gets assigned a value outside the range $1, \ldots, k$. The semantics of a Minsky machine is a transition system over configurations (i, c_1, c_2), where i is the PC and c_i, the values of the counters. The transition system is generated by the

following rules (letting $\overline{1} = 2$ and $\overline{2} = 1$).

$$(\textsc{Inc}:) \quad \frac{L_i = \mathrm{Inc}(c_j, n) \qquad c'_j = c_j + 1 \qquad c'_{\overline{j}} = c_{\overline{j}}}{(i, c_1, c_2) \longrightarrow (n, c'_1, c'_2)}$$

$$(\textsc{Dec-1}:) \quad \frac{L_i = \mathrm{Dec}(c_j, n, m) \qquad c_j = 0}{(i, c_1, c_2) \longrightarrow (n, c_1, c_2)}$$

$$(\textsc{Dec-2}:) \quad \frac{L_i = \mathrm{Dec}(c_j, n, m) \qquad c_j \neq 0 \qquad c'_j = c_j - 1 \qquad c'_{\overline{j}} = c_{\overline{j}}}{(i, c_1, c_2) \longrightarrow (m, c'_1, c'_2)}$$

In the following we write $a\langle \mathbf{0} \rangle.\mathbf{0}$ as a. Numbers are encoded as $[\![0]\!] = z$ and $[\![n+1]\!] = n\langle [\![n]\!] \rangle$. In the encoding we use two persistent register locations, r_1 and r_2 from which the values of the counters c_1 and c_2 are read and saved. The encoding of persistent locations is described in more detail in [9]. Instructions are encoded as recursive processes as follows (below we assume that we are encoding the instruction indexed by m).

$$[\![\mathrm{Inc}(c_i, n)]\!] = \mathrm{rec}\, X.l_m.\overline{r_i}(x).\overline{r_i}\langle n\langle x \rangle\rangle.\overline{l'_n}.X \ .$$

The encoding of an instruction is guarded (the CCS prefix l_m above) to ensure that we only can execute the instruction if it has been activated. For the encoding of $\mathrm{Inc}(c_i, n)$ we first read the content of location r_i and then place this content inside a location n in r_i. Finally, we activate the instruction indexed by n through the forwarder process defined below.

For the encoding of $\mathrm{Dec}(c_i, n, m)$ we split the encoding into several parts.

$$\begin{aligned} \mathrm{Get}(c_i) &= \mathrm{rec}\, Y.l_m.\overline{r_i}(x).a\langle x \rangle.b.Y \\ \mathrm{Zero}(n) &= \mathrm{rec}\, X'.\overline{az}(y).\overline{r_i}\langle z \rangle.\overline{b}.\overline{l'_n}.X' \\ \mathrm{NonZero}(m) &= \mathrm{rec}\, Y'.\overline{an}(y).\overline{r_i}\langle y \rangle.\overline{b}.\overline{l'_m}.Y' \end{aligned}$$

Now the encoding of the if-then-else instruction is given as follows

$$[\![\mathrm{Dec}(c_i, n, m)]\!] = (a, b)(\mathrm{Get}(c_i) \mid \mathrm{Zero}(n) \mid \mathrm{NonZero}(m)) \ .$$

Again we guard the encoding of the instruction. For the encoding of $\mathrm{Dec}(c_i, n, m)$ we read the content of the location r_i and place the content in the local location a. Now either Zero or NonZero can synchronise with the content inside a using the address az or an respectively (depending on whether the encoded number inside a is zero or non-zero). If the number was zero we place zero inside r_i, signal the Get-process using $\overline{b}$, and activate the instruction indexed by n. If the number was non-zero we place the content of the outermost n-location, thus decrementing the number by one, inside r_i, signal the Get-process using $\overline{b}$, and activate the instruction indexed by m.

Finally, we need one "forwarder" process for each instruction, to be able to represent instructions that loop, e.g. $(i, c_1, c_2) \longrightarrow (i, c'_1, c'_2)$, hence in the encoding

the instruction indexed by i should be able to activate the instruction indexed by i. We obtain this by using a forwarder process defined as follows.

$$F_i = \mathrm{rec}\, Y.l'_i.\overline{l}_i.Y \quad .$$

The full encoding is defined by encoding the instructions in parallel with the encoding of the counters.

$$(\tilde{m})\,(\llbracket L_1 \rrbracket \mid \ldots \mid \llbracket L_k \rrbracket \mid F_1 \mid \ldots \mid F_k \mid r_1\langle \llbracket c_1 \rrbracket \rangle.R_1 \mid r_2\langle \llbracket c_2 \rrbracket \rangle.R_2) \quad ,$$

where $\tilde{m} = \{l_1, \ldots, l_k, l'_1, \ldots, l'_k, r_1, r_2, n, z\}$ and $R_i = \mathrm{rec}\, X.r_i(x).r_i\langle x \rangle.X$. To represent the PC (in instruction i) we just replace F_i with $\overline{l}_i.F_i$, so that we are ready to activate the process representing the instruction indexed by 1. Letting L denote the list of instructions $\{L_1, \ldots L_k\}$ we write this encoding of the Minsky machine in instruction i and with counters c_1 and c_2 as follows $\llbracket (L, i, c_1, c_2) \rrbracket$. We now state the close operational correspondence.

Lemma 4.26 *For a Minsky machine with instructions L if $(i, c_1, c_2) \longrightarrow (j, c'_1, c'_2)$, then we have $\llbracket (L, i, c_1, c_2) \rrbracket \longrightarrow^k \llbracket (L, j, c'_1, c'_2) \rrbracket$, where k is either 4 or 6.*

Lemma 4.27 *For a Minsky machine with instructions L.*

- *If $\llbracket (L, i, c_1, c_2) \rrbracket \longrightarrow^4 c$ and the instruction indexed by i is an* Inc *then $c = \llbracket (L, j, c'_1, c'_2) \rrbracket$ for some j, c'_1, and c'_2 and we have the reduction $(i, c_1, c_2) \longrightarrow (j, c'_1, c'_2)$.*
- *If $\llbracket (L, i, c_1, c_2) \rrbracket \longrightarrow^6 c$ and the instruction indexed by i is a* Dec *then $c = \llbracket (L, j, c'_1, c'_2) \rrbracket$ for some j, c'_1, and c'_2 and we have the reduction $(i, c_1, c_2) \longrightarrow (j, c'_1, c'_2)$.*

□

In a similar manner we can prove that strong and weak barbed bisimilarity are undecidable for HFC^{-s}_{Γ} by encoding numbers using sequencing instead of nesting of locations. Hence we represent the number 2 by $n.n.z$ in the encoding instead of $n\langle n\langle z \rangle\rangle$, and we change the encodings of Inc and Dec accordingly to handle this change in representation.

Proposition 4.28 *Strong/weak barbed bisimilarity are undecidable for HFC^{-s}_{Γ}.*

For HFC^{-w}_{Γ} we have the weaker result.

Proposition 4.29 *The calculus HFC^{-w}_{Γ} is not finite control.*

Proof. We prove the statement by providing a counter-example.

$$\overline{a}\langle (n)n\langle \mathbf{0} \rangle \rangle \mid \mathrm{rec}\, X.a(x).\overline{a}\langle x \rangle.X \mid \mathrm{rec}\, Y.a(x).\overline{a}\langle n\langle \mathbf{0} \rangle \mid x \rangle.Y \quad ,$$

which places an $n\langle \mathbf{0} \rangle$ process in parallel for each iteration of the two recursive definitions. □

Recalling the comments in Section 2, Theorem 4.24 is, in our setting, an upper bound on the expressivity of a calculus for which strong and weak barbed congruence can be decidable. We believe that strong and weak barbed bisimilarity are also undecidable for HFC^{-w}_{Γ}, but we have not been able to improve the result in Proposition 4.29.

5 The Calculus HFC_{π}

Contrast to Homer, the π-calculus is a first order calculus without a primitive notion of locations. The syntax and main reduction rule is reminiscent of Homer. However, whereas processes are passed over named channels in Homer, only names can be passed in the π-calculus. We briefly present the π-calculus and recommend [16,13] for details.

$$P ::= \mathbf{0} \quad | \quad P \,|\, Q \quad | \quad (\nu n)P \quad | \quad \mathrm{rec}\, X.P \quad | \quad X \quad | \quad \overline{n}\langle m\rangle.P \quad | \quad n(m).P$$

The finite control segment is obtained by imposing the following simple restrictions on the recursion operator, $\mathrm{rec}\, X.P$. First, all occurrences of X in P must occur under a prefix, $\overline{n}\langle m\rangle$ or $n(m)$. Second, there should be no parallel compositions in P. These two conditions are sufficient for obtaining finite control in the π-calculus [6]. Process contexts and evaluation contexts are defined as usual.

Definition 5.1 *π-calculus* contexts $\mathscr{C}_\pi$ *and* evaluation contexts $\mathscr{E}_\pi$ *are given by the following grammars:*

$$\begin{aligned} \mathscr{C}_\pi &::= (-) \quad | \quad \mathrm{rec}\, X.\mathscr{C}_\pi \quad | \quad \mathscr{C}_\pi \,|\, P \quad | \quad (\nu n)\mathscr{C}_\pi \quad | \quad \overline{n}\langle m\rangle.\mathscr{C}_\pi \quad | \quad n(m).\mathscr{C}_\pi \\ \mathscr{E}_\pi &::= (-) \quad | \quad \mathscr{E}_\pi \,|\, P \quad | \quad (\nu n)\mathscr{E}_\pi \ . \end{aligned}$$

Structural congruence $\equiv_\pi$ is obtained in the usual manner, i.e. as the smallest congruence satisfying the following axioms.

$$\begin{array}{lll} P \,|\, \mathbf{0} \equiv_\pi P & P \,|\, Q \equiv_\pi Q \,|\, P & P \,|\, (Q \,|\, R) \equiv_\pi (P \,|\, Q) \,|\, R \\ (\nu n)\mathbf{0} \equiv_\pi \mathbf{0} & (\nu n)P \,|\, Q \equiv_\pi (\nu n)(P \,|\, Q), \text{ where } n \notin \mathrm{fn}(Q) & (n)(m)P \equiv (m)(n)P \\ \mathrm{rec}\, X.X \equiv \mathbf{0} & \mathrm{rec}\, X.P \equiv P\{\mathrm{rec}\, X.P/X\} & \end{array}$$

The semantics of the finite control π-calculus is given as the least binary relation $\longrightarrow_\pi$ over π-calculus terms closed under evaluation contexts and $\equiv_\pi$ and satisfying the following axiom

$$(\textsc{React})\ n(m).P \,|\, \overline{n}\langle m'\rangle.Q \longrightarrow_\pi P\{m'/m\} \,|\, Q \ .$$

Next follows a brief account of the encoding of the π-calculus in Homer from [2] applied to $FC\pi$. The full encoding, $[\![\cdot]\!]_2$, is defined in terms of an encoding of names, $[\![\cdot]\!]$, and an encoding of processes, $[\![\cdot]\!]_1$. A π-calculus name n is encoded as a mobile

resource $[\![n]\!]$ that performs two tasks; sending and receiving.

$$\begin{aligned} \mathit{Send}_n &\stackrel{\text{def}}{=} v(x).c(y).\overline{n}\langle x\rangle.y \\ \mathit{Receive}_n &\stackrel{\text{def}}{=} c(x).n(y).(a)\big(a\langle x\rangle \mid \overline{ab}\langle y\rangle.\overline{a}(z).z\big) \\ [\![n]\!] &\stackrel{\text{def}}{=} s\langle \mathit{Send}_n\rangle \mid r\langle \mathit{Receive}_n\rangle \end{aligned}$$

Send_n expects the encoding of the name to be communicated on v, and the continuation of the prefix on c. $\mathit{Receive}_n$ expects the encoding of the continuation and is then ready to synchronise with the resulting Send_n prefix. The significant cases of the encoding are input, output, and restriction.

$$\begin{aligned} [\![\overline{n}\langle m\rangle.P]\!]_1 &\stackrel{\text{def}}{=} (a)\big(a\langle n'\rangle \mid \overline{asv}\langle m'\rangle.\overline{asc}\langle [\![P]\!]_1\rangle.\overline{as}(z).\overline{a}(z').z\big) \\ [\![n(x).P]\!]_1 &\stackrel{\text{def}}{=} (a)\big(a\langle n'\rangle \mid \overline{arc}\langle b(x).\,[\![P]\!]_1\rangle.\overline{ar}(z).\overline{a}(z').z\big) \\ [\![(\nu n)P]\!]_1 &\stackrel{\text{def}}{=} (n)\big(\,[\![P]\!]_1\,\{[\![n]\!]\,/n'\}\big) \end{aligned}$$

Note that the names n' and m' are free process variables which will be replaced by $[\![n]\!]$ and $[\![m]\!]$ on top-level in the encoding. The encoding of $\overline{n}\langle m\rangle.P$ sends $[\![m]\!]$ to the Send_n process followed by the encoding of P. The Send_n-process is now located in a and ready to send $[\![m]\!]$ on n after which it becomes $[\![P]\!]_1$. This Send_n process is now fetched from a and placed on top-level ready to communicate with $\mathit{Receive}_n$. The encoding of an input $n(x).P$ sends the encoding of the continuation prefixed with an input on which it can receive the $[\![m]\!]$ which was sent by Send_n. The actual π-calculus communication can now be executed before the result is finally fetched from a and placed at the top level. The $a(z')$ in both encodings garbage collects the unused part of the encoding of a name. It is assumed that there is a one-to-one mapping between π-calculus names n and process variables n'. The encoding is homomorphic on $\mathbf{0}$, $\mid$, and $\mathrm{rec}\,X.P$. The full encoding of a π-calculus process P with free names $n_1,\ldots,n_m$ is $[\![P]\!]_2 \stackrel{\text{def}}{=} [\![P]\!]_1\,\{[\![n_1]\!]\,/n'_1,\ldots[\![n_m]\!]\,/n'_m\}$, where $n'_1,\ldots n'_m$ are names in bijection with $n_1,\ldots,n_m$.

Example 5.2 The encoding of $P = \overline{n}\langle m\rangle \mid n(x).\overline{x}\langle x\rangle \longrightarrow_\pi \overline{m}\langle m\rangle$.

$$\begin{aligned} [\![P]\!]_2 &= \Big[(a)\big(a\langle n'\rangle \mid asv\langle m'\rangle.asv\langle [\![\mathbf{0}]\!]_1\rangle.\overline{as}(z).\overline{a}(z').z\big) \mid \\ &\qquad (a)\big(a\langle n'\rangle \mid arc\langle b(x).\,[\![\overline{x}\langle x\rangle]\!]_1\rangle.ar(z).\overline{a}(z').z\big)\Big]\{[\![n]\!]\,/n', [\![m]\!]\,/m'\} \\ &= (a)\big(a\langle [\![n]\!]\rangle \mid asv\langle [\![m]\!]\rangle.asv\langle [\![\mathbf{0}]\!]_1\rangle.as(z).\overline{a}(z').z\big) \mid \\ &\qquad (a)\big(a\langle [\![n]\!]\rangle \mid arc\langle b(x).\,[\![\overline{x}\langle x\rangle]\!]_1\rangle.ar(z).\overline{a}(z').z\big) \end{aligned}$$

Thus we have the reductions

$$\begin{aligned} [\![P]\!]_2 &\longrightarrow^* \overline{n}\langle [\![m]\!]\rangle \mid n(y).(a)\big(a\langle b(x).\,[\![\overline{x}\langle x\rangle]\!]_1\rangle \mid ab\langle y\rangle.\overline{a}(z).z\big) \\ &\longrightarrow^* (a)\big(a\langle b(x).\,[\![\overline{x}\langle x\rangle]\!]_1\rangle \mid ab\langle [\![m]\!]\rangle.\overline{a}(z).z\big) \\ &\longrightarrow^* [\![\overline{x}\langle x\rangle]\!]_1\,\{[\![m]\!]\,/x\} = [\![m\langle m\rangle]\!]_2 \end{aligned}$$

The following lemmas are direct consequences of the results in [2] for arbitrary, i.e. not finite control π-calculus processes.

Theorem 5.3 (Dynamic correspondence [2]) $P \longrightarrow_\pi P'$ *iff* $[\![P]\!]_2 \longrightarrow^{10} [\![P']\!]_2$.

Thus if there are only finitely many reducts in the π-calculus, this must be reflected in the encoded process.

Corollary 5.4 $|\{P' \mid P \longrightarrow_\pi^* P'\}| < \infty$ *implies* $|\{[\![P'']\!]_2 \mid [\![P]\!]_2 \longrightarrow^* [\![P'']\!]_2\}|/_\equiv < \infty$.

The next proposition generalises the preceding lemma to arbitrary reductions in the encoded process.

Proposition 5.5 $|\{P' \mid P \longrightarrow_\pi^* P'\}| < \infty$ *implies* $|\{c \mid [\![P]\!]_2 \longrightarrow^* c\}|/_\equiv < \infty$.

Let HFC_π denote the subset of Homer-processes obtained as the taking the encoding of all $FC\pi$ processes together with their reducts.

Lemma 5.6 *If p is a HFC_π-process. Then $p \downarrow \varphi$ and $p \Downarrow \varphi$ are decidable.*

Theorem 5.7 *Strong and weak barbed bisimilarity are decidable for HFC_π.*

Although Theorem 4.24 and Theorem 5.7 only gives us SFC up to $\equiv$, stronger statements can be obtained by using the labelled transition semantics without $\equiv$ instead [7].

6 Comparing Homer, HFC_Γ, and HFC_π

In this section we show that Homer, HFC_Γ, and HFC_π are different calculi with respect to weak bisimilarity. Let $A, B \in \{\text{Homer}, HFC_\Gamma, HFC_\pi\}$. We compare the calculi according to the following criteria.

$$A \lessapprox B \text{ if for all } p \in A \text{ there exists } q \in B \text{ such that } p \approx q$$

$$A \approx B \text{ if } A \lessapprox B \text{ and } B \lessapprox A. \qquad A \not\approx B \text{ if } A \not\lessapprox B \text{ and } B \not\lessapprox A \ .$$

First we show that both HFC_Γ and HFC_π are strictly contained in Homer.

Proposition 6.1 *(i) $HFC_\Gamma \lessapprox$ Homer and (ii) Homer $\not\lessapprox HFC_\Gamma$*

Proof. (i) holds since any HFC_Γ process is also a Homer-process. (ii) holds since one can easily construct a Homer-process which has an infinite sequence of reductions going through mutually non-equivalent states. This is not possible in HFC_Γ due to Proposition 4.22. □

In a similar manner we get.

Proposition 6.2 *(i) $HFC_\pi \lessapprox$ Homer and (ii) Homer $\not\lessapprox HFC_\pi$*

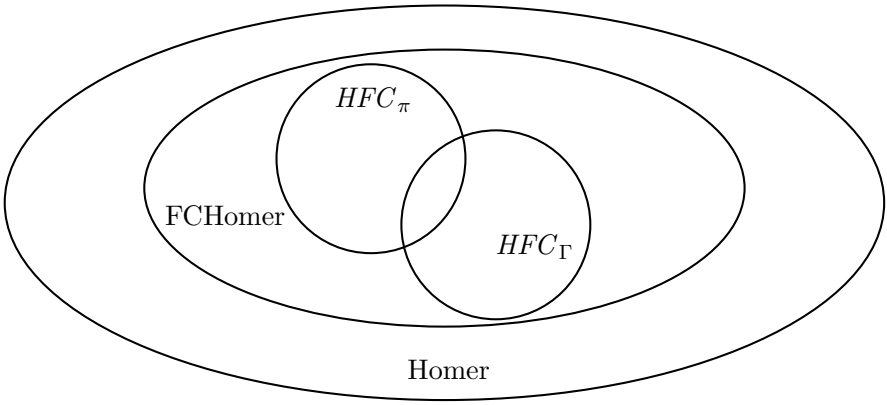

Fig. 1. Relationship between Homer, FCHomer, HFC_Γ, and HFC_π.

We have that the question of membership in HFC_Γ is decidable (i.e. type checking is decidable). At the moment we have not been able to prove a similar result for HFC_π.

Proposition 6.3 *Let $p \in Homer$. Then the question of whether $p \in HFC_\Gamma$ is decidable.*

We have yet to fully explore the relationship between HFC_Γ and HFC_π, but the following is presently known.

Proposition 6.4 *We have $HFC_\Gamma \not\lessapprox HFC_\pi$.*

Proof. Take any process p in HFC_Γ with a barb whose length is strictly greater than 3, i.e. $a\langle b\langle c\langle d\langle \mathbf{0}\rangle\rangle\rangle\rangle$ having the barb $abcd$. Such p cannot be equivalent to any process in HFC_π since the nesting depth of any HFC_π-process is at most 3. □

Letting FCHomer denote the full subcalculus of Homer where barbed bisimilarity is decidable we depict the calculi and inclusions with respect to $\approx$ in Figure 1. In Figure 1 the inclusion of HFC_Γ and HFC_π in Homer are strict with respect to $\approx$. Moreover we conjecture that that $HFC_\pi \not\lessapprox HFC_\Gamma$. Obviously HFC_Γ and HFC_π have a non-empty intersection since e.g. the $\mathbf{0}$-process is typeable as well as the encoding of $\mathbf{0}$. For the same reason we note that HFC_Γ and HFC_π are not closed with respect to $\approx$, as there are processes which are deadlocked but not typeable or in the encoding.

7 Conclusion

This paper deals with decidability of barbed bisimilarity in a higher order process calculus with locations called Homer. Since Homer is Turing-complete most semantic properties are undecidable. In particular barbs and hence barbed bisimilarity. The problem of decidability of barbed bisimilarity seems much more complicated than for CCS, the π-calculus, and Mobile Ambients.

These negative results lead us to pursue characterisations of subcalculi of Homer where bisimilarity is decidable. To fix a point of reference we defined semantic finite control up to a decidable relation $\div$. Semantic finite control then implies decidability of any relation containing $\div$. In this paper we provide two different characterisations. One characterisation is obtained by using a type system which bounds the size of processes. The typed calculus is a subcalculus of Homer which is semantic finite control. The other characterisation draws on results from the finite control π-calculus and a relatively recent published encoding of the π-calculus into Homer. Combining these results we again obtain a subcalculus of Homer which is semantic finite control.

In regards to future work the most pressing issue is whether our results extends to congruences. In Homer, early context bisimilarity characterises barbed congruence. We have not, so far, succeeded in extending the present results to barbed congruence, or equivalently early context bisimilarity. The reason for this is that in the presence of higher order communication the early (labelled) context bisimilarity relation does not get rid of the universal quantification over contexts. However recent work in [10] shows that in a certain case it is possible to derive a quantification free characterisation of barbed congruence. Whether the same approach is applicable in our setting would be interesting to examine.

We have shown that $HFC_\Gamma \not\lessapprox HFC_\pi$ and that Homer is strictly more expressive than both of these calculi. It is also clear that at the syntactic level, there are processes in HFC_π which are not in HFC_Γ and vice versa. E.g. there are processes in HFC_π (i.e. processes which are in the image of the encoding of the finite control π-calculus) which cannot be typed with the type system presented in this paper. Conversely, there are well-typed processes in HFC_Γ which are not under the image of the encoding.

We conjecture that $HFC_\pi \not\lessapprox HFC_\Gamma$. However we note that if the conjecture does not holds, then it is indeed possible that HFC_π could be embedded in HFC_Γ. Thus showing that despite the rather strict conditions imposed by the type-system, HFC_Γ would be at least as expressive as the finite control π-calculus. We would like to investigate more about the expressive power of HFC_Γ and HFC_π. In particular it will be interesting to examine whether some of the abstract approaches outlined in [8] are applicable in our setting. Finally, note that HFC_π is at least as expressive as the finite control π-calculus.

Also of interest is to find some more general notion of what decidable procedures could characterise semantic finite control. A natural extension of such work would then be to study the relationship between various notions of semantic finite control.

Finally, we would like to relax the type system for HFC_Γ. Currently the type system is quite strict, and it would be interesting to examine weaker variants of the type system which would allow us to type a larger class of processes, hopefully the entire subcalculus HFC_π.

References

[1] Bundgaard, M., T. Hildebrandt and J. C. Godskesen, *A CPS encoding of name-passing in higher-order mobile embedded resources*, Theoretical Computer Science **356** (2006), pp. 422–439.

[2] Bundgaard, M., T. Hildebrandt and J. C. Godskesen, *On encoding the π-calculus in higher-order calculi*, Technical Report TR-2008-106, IT University of Copenhagen (2008).

[3] Busi, N. and G. Zavattaro, *On the expressive power of movement and restriction in pure mobile ambients*, Theoretical Computer Science **322** (2004), pp. 477–515.

[4] Cardelli, L. and A. D. Gordon, *Mobile ambients*, Theoretical Computer Science **240** (2000), pp. 177–213.

[5] Charatonik, W., A. D. Gordon and J.-M. Talbot, *Finite-control mobile ambients*, in: *Proceedings of the 11th European Symposium on Programming (ESOP'02)*, Lecture Notes in Computer Science **2305** (2002), pp. 295–313.

[6] Dam, M., *On the decidability of process equivalences for the π-calculus*, Theoretical Computer Science **183** (1997), pp. 215–228.

[7] Godskesen, J. C. and T. 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)*, Lecture Notes in Computer Science **3821** (2005), pp. 140–151.

[8] Gorla, D., *Towards a unified approach to encodability and separation results for process calculi*, in: *Proceedings of the 19th International Conference on Concurrency Theory (CONCUR'08)*, Lecture Notes in Computer Science **5201** (2008), pp. 492–507.

[9] Hildebrandt, T., J. C. Godskesen and M. Bundgaard, *Bisimulation congruences for Homer — a calculus of higher order mobile embedded resources*, Technical Report TR-2004-52, IT University of Copenhagen (2004).

[10] Lanese, I., J. A. Pérez, D. Sangiorgi and A. Schmitt, *On the expressiveness and decidability of higher-order process calculi*, in: *Proceedings of the 23th Annual IEEE Symposium on Logic in Computer Science (LICS'08)* (2008), pp. 145–155.

[11] Maffeis, S. and I. Phillips, *On the computational strength of pure ambient calculi*, Theoretical Computer Science **330** (2005), pp. 501–551.

[12] Milner, R., "A Calculus of Communicating Systems," Lecture Notes in Computer Science **92**, Springer Verlag, 1980.

[13] Milner, R., "Communicating and Mobile Systems: the π-calculus," Cambridge University Press, 1999.

[14] Minsky, M., "Computation: Finite and infinite machines," Prentice-Hall, 1967.

[15] Sangiorgi, D., "Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms," Ph.D. thesis, University of Edinburgh, Dept. of Computer Science (1993).

[16] Sangiorgi, D. and D. Walker, "The π-Calculus: a Theory of Mobile Processes," Cambridge University Press, 2001.

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

[18] Zimmer, P., *On the expressiveness of pure safe ambients*, Mathematical Structures in Computer Science **13** (2004), pp. 721–770.

A Appendix

Proof of Theorem 4.18 (Subject Reduction). Induction in height of the derivation of $p \longrightarrow p'$.

(Send)*:* We have

$$\gamma\delta\langle p\rangle.p' \mid \mathscr{C}_\gamma^{\tilde{m}}(\delta(x).q) \longrightarrow p' \mid \mathscr{C}_\gamma^{\tilde{m}}(q\{p/x\}) \ ,$$

where $\tilde{m} \cap (\mathrm{fn}(p) \cup \delta) = \emptyset$. This sub-case is proven by induction in the length of the path γ in $\mathscr{C}_\gamma^{\tilde{m}}$.

Assume $\mathscr{C}^{\tilde{m}}_{\gamma} = (-)$. To carry out the proof of this case we perform a relatively tedious analysis of the typing of $\overline{\delta}\langle p\rangle.p' \mid \delta(x).q$ where the goal is to show that the type of $p' \mid q\{p/x\}$ is smaller than the type of the former process. The derivation bottom up starts with some applications of the (TSub)-rule followed by the (TPar)-rule. In general the (TSub)-rule could be applied anywhere in the derivation, as the rule is not syntax-directed. For simplicity we tacitly ignore the (TSub)-rule in this presentation of the proof, as the rule does not change the overall structure of the proof, and in some cases actually makes the proof easier.

So we have

$$\frac{\Gamma \vdash_\mu \overline{\delta}\langle p\rangle.p' : (k_1, k_2, k_3) \qquad \Gamma \vdash_\mu \delta(x).q : (l_1, l_2, l_3)}{\Gamma \vdash_\mu \overline{\delta}\langle p\rangle.p' \mid \delta(x).q : (n_1, n_2, n_3)} \tag{A.1}$$

with the following equations

$$n_1 = \max\{k_1, l_1\} \qquad n_2 = k_2 + l_2 \qquad n_3 = k_3 + l_3 \ . \tag{A.2}$$

The typing of the left premise of Inference (A.1) is

$$\frac{\Gamma \vdash_\mu \delta : (k_{2_1}, k_{2_2}, k_{2_3}) \qquad \Gamma \vdash_\mu p : (k_{2_1}, k_{2_2}, k_{2_3}) \qquad \Gamma \vdash_\mu p' : (k_{3_1}, k_{3_2}, k_{3_3})}{\Gamma \vdash_\mu \overline{\delta}\langle p\rangle.p' : (k_1, k_2, k_3)} \tag{A.3}$$

with the equations

$$k_1 = \max\{k_{2_1} + 1, k_{3_1}\} \quad k_2 = \max\{k_{2_2}, k_{3_2}, 1\} \quad k_3 = \max\{k_{2_3}, k_{3_3} + 1\} \ . \tag{A.4}$$

The typing of the right premise of Inference (A.1) is

$$\frac{\Gamma, x : (l_{2_1}, l_{2_2}, l_{2_3}) \vdash_\mu q : (l_{3_1}, l_{3_2}, l_{3_3}) \qquad \Gamma \vdash_\mu \delta : (l'_{2_1}, l'_{2_2}, l'_{2_3})}{\Gamma \vdash_\mu \delta(x).q : (l_1, l_2, l_3)} \tag{A.5}$$

with equations

$$l_1 = l_{3_1} \qquad l_2 = \max\{l_{3_2}, 1\} \qquad l_3 = l_{3_3} + 1 \ . \tag{A.6}$$

From Inference (A.3) and Inference (A.5) we get the equations

$$k_{2_1} \leq l_{2_1} \qquad k_{2_2} \leq l_{2_2} \qquad k_{2_3} \leq l_{2_3} \ .$$

Thus we can apply the substitution Lemma 4.8 to infer

$$\Gamma \vdash_\mu q\{p/x\} : (m_1, m_2, m_3) \ , \tag{A.7}$$

where $m_{1_i} \leq l_{3_i}$ for $i \in \{1, 2, 3\}$. Combining with the inference of p' in Inference (A.3) we obtain

$$\Gamma \vdash_\mu p' \mid q\{p/x\} : (m_{2_1}, m_{2_2}, m_{2_3}) \ ,$$

where

$$m_{2_1} = \max\{k_{3_1}, m_{1_1}\} \qquad m_{2_2} = k_{3_2} + m_{1_2} \qquad m_{2_3} = k_{3_3} + m_{1_3} \ . \tag{A.8}$$

Now the only thing left is to show that this type is less than the type of the original process. We know that $k_{3_i} \leq k_{1_i}$ from Equation (A.4) and $m_{1_i} \leq l_{3_i}$ from Equation (A.7) for $i \in \{1, 2, 3\}$. Hence the result follows from Equation (A.2) and Equation (A.8).

Proceeding with the inductive step, assume $\gamma = \delta'\gamma'$ for some non-empty sequence of names δ' and a possibly empty sequence of names γ', and $\tilde{m} = \tilde{n}\tilde{n}'$ for some names $\tilde{n}$ and $\tilde{n}'$ such that

$$\mathscr{C}^{\tilde{m}}_{\gamma} = \mathscr{C}^{\tilde{n}\tilde{n}'}_{\delta'\gamma'} = \delta'\langle(\tilde{n})\mathscr{C}^{\tilde{n}'}_{\gamma'} \mid q_2\rangle.q_3 \ .$$

As in the base-case we carry out an analysis of the typing of the process

$$\frac{\Gamma \vdash_\mu \gamma\delta\langle p\rangle.p' : (k_1, k_2, k_3) \qquad \Gamma \vdash_\mu \delta'\langle(\tilde{n})\mathscr{C}^{\tilde{n}'}_{\gamma'}(\delta(x).q) \mid q_2\rangle.q_3 : (l_1, l_2, l_3)}{\Gamma \vdash_\mu \gamma\delta\langle p\rangle.p' \mid \delta'\langle(\tilde{n})\mathscr{C}^{\tilde{n}'}_{\gamma'}(\delta(x).q) \mid q_2\rangle.q_3 : (n_1, n_2, n_3)} \tag{A.9}$$

where

$$n_1 = \max\{k_1, l_1\} \qquad n_2 = k_2 + l_2 \qquad n_3 = k_3 + l_3 \ . \tag{A.10}$$

The proof of the left premise of Inference (A.9) is

$$\frac{\Gamma \vdash_\mu \gamma\delta : (k_{2_1}, k_{2_2}, k_{2_3}) \qquad \Gamma \vdash_\mu p : (k_{2_1}, k_{2_2}, k_{2_3}) \qquad \Gamma \vdash_\mu p' : (k_{3_1}, k_{3_2}, k_{3_3})}{\Gamma \vdash_\mu \gamma\delta\langle p\rangle.p' : (k_1, k_2, k_3)} \tag{A.11}$$

where

$$k_{1_1} = \max\{k_{2_1} + 1, k_{3_1}\} \quad k_{1_2} = \max\{k_{2_2}, k_{3_2}, 1\} \quad k_{1_3} = \max\{k_{2_3}, k_{3_3} + 1\} \ . \tag{A.12}$$

By Corollary 4.9 the right premise of Inference (A.9) yields

$$\Gamma \vdash_\mu \delta'\langle(\tilde{n})\mathscr{C}^{\tilde{n}'}_{\gamma'}(q\{p/x\}) \,|\, q_2\rangle.q_3 : (l'_{1_1}, l'_{1_2}, l'_{1_3}) \ , \tag{A.13}$$

where $l'_{1_i} \leq l_{1_i}$ for $i \in \{1,2,3\}$. Thus by Inference (A.11) we get

$$\frac{\Gamma \vdash_\mu p' : (k_{3_1}, k_{3_2}, k_{3_3}) \qquad \Gamma \vdash_\mu \delta'\langle\mathscr{C}^{\tilde{n}'}_{\gamma'}(q\{p/x\}) \,|\, q_2\rangle.q_3 : (l'_{1_1}, l'_{1_2}, l'_{1_3})}{\Gamma \vdash_\mu p' \,|\, \delta'\langle\mathscr{C}^{\tilde{n}'}_{\gamma'}(q\{p/x\}) \,|\, q_2\rangle.q_3 : (m'_1, m'_2, m'_3)} \tag{A.14}$$

where

$$m'_1 = \max\{k_{3_1}, l'_{1_1}\} \qquad m'_2 = k_{3_2} + l'_{1_2} \qquad m'_3 = k_{3_3} + l'_{1_3} \ . \tag{A.15}$$

Now by Equation (A.12) we have $k_{3_i} \leq k_{1_i}$ and by Inference A.13 we know that $l'_{1_i} \leq l_{1_i}$ for $i \in \{1,2,3\}$, thus by Equation (A.10) and Equation (A.15) we have as needed $m'_i \leq n_i$ for $i \in \{1,2,3\}$. The (Take)-rule is handled in a similar manner, except that we need to handle the open operator for scope extension.

If the reduction was derived by closure under structural congruence (without using the unfolding axioms), i.e. we have

$$\begin{array}{c} p \equiv_m q \longrightarrow q' \equiv_m p' \\ p \longrightarrow p' \end{array} \ .$$

From $\Gamma \vdash_\mu p : T$ and p rec-guarded and using Lemma 4.17 we know that $\Gamma \vdash_\mu q : T$ and q rec-guarded. By the induction hypothesis we have that $\Gamma \vdash_\mu q' : S$ for some S such that $S \leq T$ and again by Lemma 4.17 we have that $\Gamma \vdash_\mu p' : S$ as needed.

Finally we consider the case where the reduction is closed under evaluation contexts. But in all the cases the result follows directly by the induction hypothesis. □