Termination in Higher-Order Concurrent Calculi*

Romain Demangeon[1], Daniel Hirschkoff[1], and Davide Sangiorgi[2]

[1] ENS Lyon, Université de Lyon, CNRS, INRIA, France
[2] Università di Bologna, Italy

Abstract. We study termination of programs in concurrent higher-order languages. A higher-order concurrent calculus combines features of the λ-calculus and of the message-passing concurrent calculi. However, in contrast with the λ-calculus, a simply-typed discipline need not guarantee termination; and, in contrast with message-passing calculi such as the π-calculus, divergence can be obtained even without a recursion (or replication) construct.
We first consider a higher-order calculus where only processes can be communicated. We propose a type system for termination that borrows ideas from termination in Rewriting Systems (and following the approach to termination in the π-calculus in [DS06]). We then show how this type system can be adapted to accommodate higher-order functions in messages. Finally, we address termination in a richer calculus, that includes localities and a passivation construct, as well as name-passing communication. We illustrate the expressiveness of the type systems on a few examples.

1 Introduction

A system is terminating when it cannot perform an infinite number of transition steps. Termination is a difficult property to ensure: for instance, the termination of a rewriting system is not decidable in the general case. The problem of termination has been widely studied in sequential languages, including higher-order ones such as the λ-calculus, employing static analysis and especially type systems.

Ensuring termination for concurrent and mobile systems is much more challenging, as such systems are rarely confluent. The presence of mobility, under the form of an evolving topology of communication (new servers can be created, information travels across the system along dynamically evolving connections), adds even more complexity to the task. Previous works on this subject [YBH04, San06, DS06] rely on type systems to ensure termination in a concurrent context, in the setting of the π-calculus (π). In some of these systems, weights are assigned to π-calculus channels, and typability guarantees that, at

* This work has been supported by the European Project FET-GC II IST-2005-16004 Sensoria, and by the french ANR projects "CHoCo" and "Complice".

each reduction step that involves the firing of a replicated term, the total weight associated to the process decreases.

In this work, we want to address the problem of termination in languages that include powerful primitives for distributed programming. The most important primitive that we focus on is *process passing*, that is, the ability to transmit an entity of computation along messages. We therefore study higher-order concurrent languages, and focus on the Higher-Order π-calculus, HOpi [San92], as working formalism to analyse termination in this setting.

To our knowledge, there exists no result on termination for higher-order concurrent processes. In some sense, formalisms like HOpi combine features from both the λ-calculus and the π-calculus, and ensuring termination in such a setting involves the control of difficulties related both to the higher-order aspects and to the concurrency aspects of the model.

In contrast with name-passing concurrent languages such as the π-calculus, where recursion (or a similar operator such as replication) is needed in order to have non-terminating programs, in HOpi, similarly to the λ-calculus, non-termination can show up already in the fragment without recursion. As an example, consider the following process:

$$Q_0 = P_0 \mid \overline{a}\langle P_0 \rangle, \qquad \text{where} \qquad P_0 = a(X).(X \mid \overline{a}\langle X \rangle)$$

(P receives a process on channel a, spawns the received process and emits a copy of this process on a again). Q_0 can only reduce to itself, giving rise to a divergence.

Also, in contrast with the λ-calculus, where termination is ensured by adopting a simple type discipline, such as that of the simply-typed λ-calculus, which rules out recursive types, the HOpi process Q_0 is typable without resorting to recursive types (Q_0 is a process of simply-typed HOpi, where name a is used to carry processes, and the variables used are process variables).

To sum up, calculi like HOpi put together ideas from π-calculus and λ-calculus, and in both these calculi termination has been studied (using type systems). We cannot however directly adapt existing ideas. On the one hand, the type systems for termination in the π-calculus essentially impose constraints on the recursion (or replication) operators; we cannot directly adopt the idea in HOpi because HOpi has no recursion. On the other hand, the type systems for termination in the λ-calculus put constraints on self-applications, notably by forbidding recursive types. We cannot directly adopt these either, because of non-terminating examples like the one above. Indeed, there is no explicit self-application in Q_0, and Q_0 is actually typable in the simplest of the type systems of HOpi, which corresponds to the simply-typed discipline of the λ-calculus, without recursive types.

The goal of this paper is to propose more refined type disciplines, that allow us to rule out non-terminating programs such as the one above while retaining a non-trivial expressiveness.

A solution could be to exploit the standard encoding of HOpi in π [SW01], that respects termination, and use it, together with existing type systems for π,

to infer termination in HOpi. However this would not be applicable in extensions of HOpi that are not encodable in π (or that appear difficult to encode), for instance, in distributed versions of the calculus. If one wishes to handle models for distributed computing (including explicit locations and mobility of locations), the techniques and type systems for termination should be directly formulated on HOpi. Further, a direct formulation would allow one to make enhancements of the techniques that are tailored to (and therefore more effective on) higher-order concurrency. We nevertheless analyse the approach via the encoding in the π-calculus in Sect. 2.3, to compare it with our system in terms of expressiveness.

In this paper, we first (Sect. 2) analyse termination in HOpi_2, a higher-order calculus where processes are the only values exchanged. We propose a type system for termination using techniques from term-rewriting, in which termination is guaranteed by a decreasing weight associated to processes. This is also the approach followed in [DS06] for termination in the π-calculus. The technical details and the proofs are however rather different, for the reasons outlined earlier (e.g., name-passing vs process passing, absence of replication or recursion). We present the basic type system, make some assessment of its expressiveness, and describe a few important refinements (though only briefly, due to lack of space).

The system for HOpi_2 is a starting point, from which we build a similar type system for HOpi_ω, a richer higher-order calculus where the values communicated also include higher-order functions (Sect. 3 – the names HOpi_2 and HOpi_ω are inspired from [SW01]). The additional constructs for functions have to be controlled in order to rule out diverging behaviours.

These results pave the way for the study of a further and much richer extension, the calculus we call PaPi (Sect. 4). PaPi is equipped with powerful primitives that are found in formalisms for global computing: in addition to standard name-passing (as in the π-calculus) and higher-order communication, we also handle explicit localities and *passivation*. Passivation is the operation of intrusively capturing a running computation, in order to be able to modify the process being executed (for instance to discard, duplicate or update it). We provide several examples to illustrate the expressive power given by the combination of primitives in PaPi. Analysing and controlling interaction in PaPi is a challenging task. We discuss how the ideas we developed to control process passing in HOpi_2 and HOpi_ω can be combined with the approach to name passing of [DS06] in order to guarantee termination.

2 HOpi_2

This section is dedicated to the study of HOpi_2, a basic higher-order process calculus, with processes as the only communication values.

2.1 The Calculus

We shall use symbols P, Q, R, S for processes, X, Y for process variables, and names a, b, c for channels.

$$\overline{\overline{a}\langle Q\rangle.P_1 \mid a(X).P_2 \to P_1 \mid P_2[Q/X]}$$

$$\frac{P_1 \to P_1'}{P_1 \mid P_2 \to P_1' \mid P_2} \qquad \frac{P \to P'}{(\nu c)P \to (\nu c)P'} \qquad \frac{Q \equiv P \quad P \to P' \quad P' \equiv Q'}{Q \to Q'}$$

Fig. 1. The Operational Semantics of HOpi_2

The grammar for processes of HOpi_2 is the following:

$$P ::= \mathbf{0} \mid P|P \mid \overline{a}\langle P\rangle.P \mid a(X).P \mid X \mid (\nu c)P \ .$$

Structural congruence ($\equiv$) is defined in the standard way on HOpi_2. We shall omit trailing occurrences of $\mathbf{0}$ in processes of the form $\overline{a}\langle P\rangle.\mathbf{0}$. Reduction is defined by the rules of Fig. 1. $Q[P/X]$ stands for the capture avoiding substitution of variable X with process P in Q. A process P is *terminating* if there exists no infinite sequence of reductions emanating from P. We suppose that all processes we shall manipulate obey a *Barendregt convention*: all bound names are pairwise distinct and different from all free names (similar notations and conventions will be adopted for the calculi we study in the next sections).

To see how HOpi_2 processes interact, consider $S_1 = \overline{a}\langle\overline{b}\langle\mathbf{0}\rangle.\mathbf{0}\rangle.\overline{a}\langle b(Z).\mathbf{0}\rangle$ and $S_2 = a(X).a(Y).(X \mid Y)$. S_1 is a process which sends on a the code of a process emitting $\mathbf{0}$ on b, and then sends on a the code of a process receiving on b. S_2 is a process which upon reception of two processes on a (in sequence) executes these in parallel. Process $S_1 \mid S_2$ performs two reductions to become $\overline{b}\langle\mathbf{0}\rangle.\mathbf{0} \mid b(Z).\mathbf{0}$, after which a synchronisation on b can take place.

As discussed above, recursive outputs ("self-emissions") can lead to diverging behaviours in HOpi_2: in process Q_0 from Sect. 1, a process containing an output on a is sent over channel a itself in $\overline{a}\langle P_0\rangle$. Our type system, in Sect. 2.2, puts constraints on self-emissions in order to control divergence.

2.2 A Type System to Ensure Termination in HOpi_2

The types for channels are of the form $Ch^n(\diamond)$, where $\diamond$ is interpreted as the type of processes (throughout the paper, we use the syntax $Ch(T)$ to denote the type of a channel carrying values of type T), and n is a natural number, called the *level* of the channel being typed. We use Γ to range over typing contexts, that are lists of typing hypotheses. If $a : Ch^n(\diamond)$ belongs to Γ, we write $\Gamma(a) = Ch^n(\diamond)$, and $lvl_\Gamma(a) = n$. Processes (and process variables in Γ) are typed using simply a natural number.

Figure 2 presents the rules of our type system for HOpi_2. (This system, and all systems we shall study in the paper, are *syntax directed*: there is one typing rule per syntactic construct. We shall exploit this when referring to the typing rules by only mentioning the construct they deal with.) The actual control takes

$$\frac{\Gamma(X) = n}{\Gamma \vdash X : n} \qquad \frac{\Gamma, c : Ch^k(\diamond) \vdash P : n}{\Gamma \vdash (\nu c)P : n} \qquad \frac{\Gamma \vdash P_1 : n_1 \qquad \Gamma \vdash P_2 : n_2}{\Gamma \vdash P_1 \mid P_2 : \max(n_1, n_2)}$$

$$\frac{}{\Gamma \vdash \mathbf{0} : 0} \qquad \frac{\Gamma \vdash P : k \qquad \Gamma \vdash Q : m \qquad lvl_\Gamma(a) = n \qquad k < n}{\Gamma \vdash \overline{a}\langle P \rangle.Q : \max(m, n)} \qquad \frac{\Gamma, X : k-1 \vdash P : n \qquad lvl_\Gamma(a) = k}{\Gamma \vdash a(X).P : n}$$

Fig. 2. HOpi$_2$: Typing Rules

place in the output rule, where we ensure that the level of the transmitted process is strictly smaller than the level of the carrying channel: this way, we exclude "self-emissions". This discipline is at the basis of the termination proof: when a communication is performed, an output of weight n is traded for possibly several new outputs appearing in the process, that all have a smaller weight.

We can check that process Q_0 from Sect. 1 is ruled out by our system: as P_0 contains an output on a, its level is at least the level of a. As a consequence, the output rule forbids P_0 to be sent on a itself, and Q_0 is not typable.

To establish soundness of our type system, we introduce a measure on typing derivations that decreases along reductions. We use notation $\mathcal{D} : (\Gamma \vdash P : n)$ to mean that $\mathcal{D}$ is a derivation of the typing judgment $\Gamma \vdash P : n$. Below and in the remainder of the paper, $\uplus$ will stand for multiset union.

Definition 1. *If $\mathcal{D} : (\Gamma \vdash P : n)$, we define $m(\mathcal{D})$ by induction over the structure of $\mathcal{D}$ as follows (to ease presentation, we take advantage of the fact that the type system is syntax-directed, and reason according to the shape of P):*

- $m_\mathcal{D}(\mathbf{0}) = m_\mathcal{D}(X) = \emptyset$;
- $m_\mathcal{D}(P_1 \mid P_2) = m_\mathcal{D}(P_1) \uplus m_\mathcal{D}(P_2)$
- $m_\mathcal{D}((\nu c)P) = m_\mathcal{D}(a(X).P) = m_\mathcal{D}(P)$;
- $m_\mathcal{D}(\overline{a}\langle P \rangle.Q) = m_\mathcal{D}(Q) \uplus \{n\}$ *if* $lvl(a) = n$ *according to* Γ.

We note that if $\mathcal{D} : (\Gamma \vdash P : n)$ and $P \equiv P'$, there exists $\mathcal{D}'$ s.t. $\mathcal{D}' : (\Gamma \vdash P' : n)$ and $m(\mathcal{D}) = m(\mathcal{D}')$. $<_{mul}$ denotes the multiset extension of the standard ordering on integers (e.g., $\{2, 2\} <_{mul} \{3\}$).

Lemma 1. *If $\mathcal{D} : (\Gamma \vdash P : n)$, then $m(\mathcal{D}) <_{mul} \{n + 1\}$.*

Proposition 1. *If $\mathcal{D} : (\Gamma \vdash P : n)$ and $P \to P'$ then there exist $\mathcal{D}'$ and $n' \leq n$ such that $\mathcal{D}' : (\Gamma \vdash P : n')$ and $m(\mathcal{D}') <_{mul} m(\mathcal{D})$.*

Proof (Sketch). We reason by induction on the derivation of the transition of P. The most interesting case is when $P = \overline{a}\langle P_1 \rangle.Q_1 \mid a(X).Q_2 \to P' = Q_1 \mid Q_2[P_1/X]$. By the typing hypothesis, we get $lvl(a) = n$ and $\mathcal{D}_0 : (\Gamma \vdash P_1 : k)$ for some $\mathcal{D}_0$ and k. We can build a typing derivation $\mathcal{D}'$ for P' such that there exists c satisfying $m(\mathcal{D}') = m(\mathcal{D}) \setminus \{n\} \uplus m(\mathcal{D}_0)^c$, where $m(\mathcal{D}_0)^c$ stands for the multiset union of c copies of $m(\mathcal{D}_0)$. Indeed, an output on a in P is erased along

the reduction, and there are possibly several copies of P_1 which appear in P': c is defined as the number of occurrences of X in Q_2 which do not appear inside messages. From typability of $\overline{a}\langle P_1\rangle.Q_1$, we get $k < n$, and from Lemma 1 we deduce $m(\mathcal{D}_0) <_{mul} \{n\}$. Thus $m(\mathcal{D}') <_{mul} m(\mathcal{D})$. □

Corollary 1. *If $\Gamma \vdash P : n$, then P terminates.*

Proof. Follows from Proposition 1 and the fact that the multiset extension of a terminating order is terminating. □

2.3 An Analysis of the Type System for HOpi$_2$

Typing via Encoding into π. We now compare the expressiveness of our type system with the expressiveness induced on HOpi$_2$ by the translation into π and the existing type system [DS06] for the π-calculus.

Translating HOpi$_2$ processes. We use (an adaption of) the standard encoding of HOpi$_2$ into the π-calculus [San92] (see also [Tho96]). The target calculus of our encoding is the simply typed monadic π-calculus, with `unit` as base type (the unique value of type `unit` is noted $\bigstar$), and where replication is allowed only on inputs. The encoding is rather standard – we only recall the clauses for input, output and process variables (an unambiguous correspondence between HOpi$_2$ process variables and their counterpart as π names is implicitly assumed):

$$[\![a(X).P]\!] = a(x).[\![P]\!] \qquad [\![X]\!] = \overline{x} \qquad [\![\overline{a}\langle Q\rangle.P]\!] = (\nu h_a)\ \overline{a}\langle h_a\rangle.(\ [\![P]\!] \mid\ !h_a.[\![Q]\!]\ )$$

A higher-order output action $\overline{a}\langle Q\rangle.P$ is translated into the emission of a new name (h_a), which intuitively is the address where process Q can be accessed. Interactions on h_a and x, noted using CCS prefixes, actually involve the transmission of the unique value of type `unit`.

Proposition 2. *For any HOpi$_2$ process P, P terminates iff $[\![P]\!]$ terminates.*

Typing the encoding. We rely on the first type system of [DS06] to type the encoding of a HOpi$_2$ process. This type system assigns levels to names, in order to control replicated processes. If we call $os(P)$ the multiset consisting of channel names that are used as subject of an output in a π process P, and where the output does not occur under a replication, then $!a(x).P$ is well-typed if the level of a is strictly greater than the level of all names in $os(P)$. We write $\Gamma \vdash_{\text{pi}} P$ for typability in the π-calculus according to [DS06]. All processes typable using this type system are terminating.

There exist HOpi$_2$ processes that can be proved to terminate using the type system for HOpi$_2$, but whose encoding fails to be typable using the type system for π. A very simple example is given by $R_0 = a(X).\overline{a}\langle X\rangle$. We indeed have

$$[\![R_0]\!] \quad = \quad a(h_X).(\nu h_a)\,\overline{a}\langle h_a\rangle.!h_a.\overline{h_X}\ ,$$

which is not typable: indeed, h_X and h_a necessarily have the same type (both are transmitted on a), which prevents subprocess $!h_a.\overline{h_X}$ from being typable.

This example suggests a way to establish a relationship between the type systems in HOpi_2 and in π. Consider for that the type system for HOpi_2 obtained by replacing rule (**In**) in Figure 2 with the following one, the other rules remaining unchanged (the typing judgment for this modified type system shall be written $\Gamma \vdash_{\mathbf{m}} P : n$):

$$(\mathbf{In}') \quad \frac{\Gamma, X : k \vdash_{\mathbf{m}} P : n \qquad lvl_\Gamma(a) = k}{\Gamma \vdash_{\mathbf{m}} a(X).P : n}$$

Clearly, the modified type system is more restrictive, that is, $\Gamma \vdash_{\mathbf{m}} P : n$ implies $\Gamma \vdash P : n$, but not the converse (cf. process R_0 seen above).

Using this system, we can establish the following property, that allows us to draw a comparison between typability in HOpi_2 and in the π-calculus:

Proposition 3. *Let S be a HOpi$_2$ process. If $\Gamma \vdash_{\mathbf{m}} S : m$, then there exists Δ, a typing context for π, such that $\Delta \vdash_{\text{pi}} [\![S]\!]$.*

Proof (Sketch). The crux of this proof is the correspondence between the typing of output actions in HOpi_2 and the typing of replications in π.

The encoding seen above induces a translation of HOpi_2 typing contexts as follows (type checking the encoding of a restricted term induces similar typing assumptions):

$$[\![\emptyset]\!] = \emptyset \qquad\qquad [\![\Gamma, X : n]\!] = [\![\Gamma]\!], x : Ch^n(\mathtt{unit})$$

$$[\![\Gamma, a : Ch^n(\diamond)]\!] = [\![\Gamma]\!], a : Ch^n(Ch^n(\mathtt{unit}))$$

To show that $\Gamma \vdash_{\mathbf{m}} S : m$ implies $[\![\Gamma]\!] \vdash_{\text{pi}} [\![S]\!]$, we focus on replicated terms in $[\![S]\!]$. Every replication appearing in $[\![S]\!]$ corresponds to the encoding of an output of S. It therefore appears in a context of the form $(\nu h_a)\ \overline{a}\langle h_a\rangle.(!h_a.[\![P]\!] \mid [\![Q]\!])$, corresponding to an output action $\overline{a}\langle P\rangle.Q$ occurring in S. As $\Gamma \vdash_{\mathbf{m}} S : m$, the rule for output gives $\Gamma' \vdash_{\mathbf{m}} a : Ch^n(\diamond)$ and $\Gamma' \vdash_{\mathbf{m}} P : m'$ for some Γ', with $m' < n$. This means that $[\![\Gamma']\!] \vdash_{\text{pi}} h_a : Ch^n(\mathtt{unit})$ and $\forall b \in os([\![P]\!]), [\![\Gamma']\!](b) = Ch^m(T)$ with $m \leq m'$. Thus $n > m$, and the replicated input at h_a is well-typed (in π). □

Remark 1 (The limits of our type system).

Symmetrically to the example process R_0 seen above, there exist terms that can be typed via the encoding, but that are rejected by our type system: consider

$$R_1 = \overline{a}\langle\overline{a}\langle\mathbf{0}\rangle\rangle \mid a(X).\mathbf{0} \quad \text{and} \quad R_2 = a(X).b(Y).X \mid \overline{a}\langle\overline{a}\langle\mathbf{0}\rangle\rangle \mid \overline{b}\langle\mathbf{0}\rangle \ .$$

None of these processes is typable, because they contain "self-emissions" (an output action on channel a occurring inside a process emitted on a). However, R_1 and R_2 are terminating. Their encodings in π are

$$[\![R_1]\!] = (\nu h_a)\ \overline{a}\langle h_a\rangle.!h_a.(\nu h'_a)\ \overline{a}\langle h'_a\rangle.!h'_a.\mathbf{0} \mid a(x).\mathbf{0} \qquad \text{and}$$
$$[\![R_2]\!] = a(x).b(y).\overline{x} \mid (\nu h_a)\ \overline{a}\langle h_a\rangle.!h_a.(\nu h'_a)\ \overline{a}\langle h'_a\rangle.!h'_a.\mathbf{0} \mid (\nu h_b)\ \overline{b}\langle h_b\rangle.!h_b.\mathbf{0} \ ,$$

which are both typable using the system of [DS06]. A suitable assignment for R_1 is, e.g., $lvl(a) = 1$, $lvl(h_a) = lvl(h'_a) = 2$; both replications are typed as the first one trades a name of level 2 for a name of level 1 and the second one has no output at all in its continuation. R_2 can be typed with the same level assignment, and $lvl(h_b) = lvl(h'_b) = 2$.

It thus appears that self-emissions can be innocuous, while they are systematically rejected by the system of Sect. 2. Self-emissions in R_1 and R_2 are reminiscent of recursive calls in continuations of replicated π processes, like, e.g., in $!a(x).b(y).\overline{a}\langle y\rangle$. It turns out that constructions like the one we find in R_2 show up in examples (see Remark 2 below).

As pointed out in the Introduction, a direct type system can be the basis for refinements and extensions. Indeed, both the refinements discussed at the end of this section, and the extensions presented in Sect. 4 allow us to handle processes that go well beyond those that can be treated via encodings into the pi-calculus.

Towards More Expressive Type Systems. We now discuss some possible refinements of our type system that allow us to overcome the limitations we have presented. Some of these refinements are inspired by the type systems developed in [DS06], some are specific to our higher-order setting.

A first enrichment consists in attaching two pieces of information to a channel, instead of simply a level. First, a channel a has a *weight*, which stands for the contribution of active outputs on a to the global weight of a process. For instance, in the process $P_1 = \overline{a_1}\langle P_2\rangle$, with $P_2 = \overline{b_1}\langle Q_1\rangle \mid \overline{b_2}\langle Q_2\rangle$, the global weight of P_2 is equal to the sum of the weights attached to names b_1 and b_2. Second, a channel a has a *capacity*, which is an upper bound on the weight of processes that may be sent on a: P_1 is well-typed provided the capacity of a_1 is greater than the weight of P_2. This approach can be related to the observations we have made above about $[\![R_1]\!]$ and $[\![R_2]\!]$, where the level of a (resp. h_a) plays the rôle of the weight (resp. the capacity).

As a second enrichment, we use multisets of natural numbers to represent the weight and the capacity attached to a channel, as well as the type attached to a process. The rules defining a type system that includes these two enrichments are presented on Fig. 3, where M, N denote multisets of natural numbers. In the rule for output, M_2 plays the rôle of the capacity (which must dominate the weight of Q), and M_1, the weight of the output on a, is combined with the weight (M) of the continuation process P. As an example, if the outputs on a (resp. on b) weight $\{1\}$ (resp. $\{2\}$), the process $\overline{a}\langle P\rangle \mid \overline{a}\langle P'\rangle \mid \overline{b}\langle Q\rangle$ has type $\{2, 1, 1\}$.

In the rule for input, $o(M, P, X)$ stands for the multiset obtained by computing the multiset union of as many copies of M as there are occurrences of X that do not appear in a message in P. Formally:

$$
\begin{array}{l}
o(M, \mathbf{0}, X) = \emptyset \\
o(M, X, X) = M, \qquad o(M, Y, X) = \emptyset,\ Y \neq X \\
o(M, P_1 \mid P_2, X) = o(M, P_1, X) \uplus o(M, P_2, X) \\
o(M, a(Y).P, X) = o(M, P, X),\ Y \neq X, \qquad o(M, a(X).P, X) = \emptyset \\
o(M, \overline{a}\langle Q\rangle.P, X) = o(M, (\nu c)\ P, X) = o(M, P, X)
\end{array}
$$

$$\frac{}{\Gamma \vdash \mathbf{0} : \emptyset} \qquad \frac{\Gamma(X) = M}{\Gamma \vdash X : M} \qquad \frac{\Gamma, c : Ch^{M_1, M_2}(\diamond) \vdash P : N}{\Gamma \vdash (\nu c)P : N}$$

$$\frac{\Gamma \vdash P_1 : M_1 \quad \Gamma \vdash P_2 : M_2}{\Gamma \vdash P_1 \mid P_2 : M_1 \uplus M_2} \qquad \frac{\Gamma \vdash P : M \quad \Gamma \vdash Q : N \quad \Gamma(a) = Ch^{M_1, M_2}(\diamond) \quad N <_{mul} M_2}{\Gamma \vdash \overline{a}\langle Q \rangle.P : M \uplus M_1}$$

$$\frac{\Gamma, X : M_2 \vdash P : M \quad \Gamma(a) = Ch^{M_1, M_2}(\diamond) \quad o(M_2, P, X) <_{mul} M_1}{\Gamma \vdash a(X).P : M}$$

Fig. 3. Typing Processes using Multisets

Computing $o(M, P, X)$ is necessary because communication of a process Q can have the effect of spawning several copies of Q. Accordingly, the weight associated to the channel transmitting Q must be strictly greater than the total weight of the processes spawned along consumption of the message.

To establish soundness of this type system, we rely as previously on a measure on terms. The measure of a process P is given by the multiset union of the weights associated to all names that are used in output, for occurrences that are not themselves within a message in P. We show that the type of P (i.e., the multiset given by the typing judgment for P) is always greater than the measure of P. Intuitively, this is the case because the process variables contribute to the type, but not to the measure. In a reduction of the form $\overline{a}.\langle Q \rangle.P_1 \mid a(X).P_2 \rightarrow P_1 \mid P_2[Q/X]$, with $\Gamma(a) = Ch^{M_1, M_2}(\diamond)$ and $\Gamma \vdash Q : N$, an output of type M_1 is consumed and a process Q of type N is spawned in P_2 for each occurrence of X in P_2. The typing rule for outputs enforces $M_2 >_{mul} N$ and the typing rule for input enforces $M_1 >_{mul} o(M_2, P_2, X)$. This entails that M_1 is greater than the multiset union of the measure of each process Q spawned in P_2. Thus the measure globally decreases, which guarantees termination.

We can check that using this type system, certain forms of "self-emission", like in $R_1 = \overline{a}\langle \overline{a}\langle \mathbf{0} \rangle \rangle \mid a(X).\mathbf{0}$, can be typed. If we assign weight $\{1\}$ and capacity $\{2\}$ to a, the output is well-typed because process $\overline{a}\langle \mathbf{0} \rangle$ has weight $\{1\} <_{mul} \{2\}$. The input is also well-typed as the weight of a is greater than $o(\{2\}, \mathbf{0}, X) = \emptyset$.

We have studied a third refinement of our type system, defined for a higher-order formalism with a primitive construct for replication. This in principle does not add expressiveness to the calculus, because replication is encodable in HOpi_2 (using a process similar to Q_0 from Sect. 1). However, in terms of typability, having a primitive replication, and a dedicated typing rule for it, helps in dealing with examples. The type system to handle replication in presence of higher-order communications controls divergences that can arise both from self-emissions and from recursion in replications (as they appear in the setting of [DS06]). More details about this analysis are given in an extended version of this paper [DHS09].

A further refinement: handling successive input prefixes. Inspired by the third type system of [DS06], we can treat sequences of input prefixes as a kind of 'single input action', that has the effect of decreasing the weight of the process being executed.

Let us sketch the main idea behind this approach. Consider a process of the form $a_1(X_1).\ldots.a_k(X_k).P$. We make sure that the weight associated to the sequence of inputs is strictly greater than the weight of the processes that are spawned in the continuation P. If $\Gamma(a_i) = Ch^{M_1^i, M_2^i}(\diamond)$, then the former quantity is equal to $M_1^1 \uplus \cdots \uplus M_1^k$. To compute the latter quantity, one has, like above, to take into account the multiplicity of the X_is (whose weight is given by typing) in P — again, we only consider occurrences of these process variables that are not within messages.

According to this approach, the overall weight of a process can temporarily increase along communications, before a sequence of inputs is consumed. However, each consumption of a whole sequence of inputs induces a global weight loss, thus ensuring termination (it can be shown that for a divergence to exist, there must be infintely many consumptions of whole sequences of inputs). Technically, the shape of the soundness proof follows the lines of the justification of a corresponding type system for the name-passing paradigm in [DHS08].

This way, process $R_2 = a(X).b(Y).X \mid \overline{a}\langle\overline{a}\langle\mathbf{0}\rangle\rangle \mid \overline{b}\langle\mathbf{0}\rangle$ can be typed by assigning type $Ch^{\{1\},\{1,1\}}(\diamond)$ to a and $Ch^{\{2\},\{2\}}(\diamond)$ to b. The input sequence is typed as the total weight of the sequence is $\{2,1\}$, and the total weight of the processes spawned is $\{1,1\}$ (one occurrence of X, no occurrence of Y). The outer output on a is typed as the weight of the object process is $\{1\}$ (an output on a) and the capacity of a is $\{1,1\}$. The inner output on a and the output on b are well-typed as the capacities of these two names are greater than $\emptyset$, the weight of $\mathbf{0}$.

Remark 2 (Encoding the choice operator). To illustrate the expressiveness of the resulting type system, we show in [DHS09] the typability of Nestmann and Pierce's protocol for modelling (separate) choice [NP00] — precisely, the protocol adapted to a higher-order calculus. From the termination viewpoint, this protocol is interesting because it is non-trivial, and because its termination is not a straightforward property to establish, as the protocol involves some forms of backtracking. Also, when rewritten in the higher-order paradigm (more precisely, in an extension of HOpi_ω^- see Sect. 3), the protocol makes use of some patterns or combinations of operators that are delicate for termination (in particular, a pattern similar to $a(X).b(Y).X$ in the above example).

3 HOPi$_\omega$: Transmitting Higher-order Functions

The Calculus. We now present HOpi_ω, a calculus inspired from $\text{HOPi}^{\mathtt{unit},\rightarrow,\diamond}$ in [SW01]. The main difference between HOpi_ω and HOpi_2 is that the values communicated in HOpi_ω can be $\bigstar$, the unique element of type `unit`, or functions (precisely parametrised processes) of arbitrarily high order (the order indicating the level of arrow nesting in the type). The grammar for processes and values

(we use metavariables v, w, not to be confused with names, to range over values, and x, y to range over variables) is the following:

$$P := \mathbf{0} \mid P|P \mid \overline{a}\langle v\rangle.P \mid v\lfloor v\rfloor \mid a(x).P \mid (\nu a)P \qquad\qquad v = \bigstar \mid x \mapsto P$$

$x \mapsto P$ is a function from values to processes, and $v\lfloor w\rfloor$ is the application of a function to its argument. We will restrict ourselves to meaningful usages of (higher-order) functions, which can be ensured by adopting a standard type discipline (see below).

The operational semantics of HOpi$_\omega$ is given by the rules below (rules for closure w.r.t. parallel composition, restriction, and structural congruence are omitted): communication involves the transmission of a value, and β-reduction takes place when a function is applied to a value.

$$\overline{\overline{a}\langle v\rangle.Q_1 \mid a(x).Q_2 \to Q_1 \mid Q_2[v/x]} \qquad\qquad \overline{(x \mapsto P)\lfloor v\rfloor \to P[v/x]}$$

HOpi$_2$ processes can be seen as HOpi$_\omega$ processes by replacing communication of processes with communication of values of type $\mathtt{unit} \to \diamond$, and, accordingly, usages of process variables with an application to $\bigstar$. For instance, the diverging example Q_0 in HOpi$_2$ becomes $\overline{a}\langle x \mapsto P\rangle \mid P$ where $P = a(y).(y\lfloor\bigstar\rfloor \mid \overline{a}\langle y\rangle)$.

The following is an example HOpi$_\omega$ process:

$$\begin{aligned} P \;=\; & \overline{a}\langle x \mapsto (x\lfloor\bigstar\rfloor \mid x\lfloor\bigstar\rfloor)\rangle \;\mid\; \overline{b_1}\langle x_1 \mapsto \overline{c}\langle\bigstar\rangle\rangle.\overline{b_2}\langle x_2 \mapsto c(z).\mathbf{0}\rangle \\ & \mid\; b_1(y_1).b_2(y_2).a(y_3).(y_3\lfloor y_1\rfloor \mid y_3\lfloor y_2\rfloor)\;. \end{aligned}$$

Channel c has type $Ch(\mathtt{unit})$, channels b_1, b_2 have type $Ch(\mathtt{unit} \to \diamond)$ (see the grammar for types below), and channel a has type $Ch((\mathtt{unit} \to \diamond) \to \diamond)$. P can do two communications on b_1 and b_2. Then, a function (in this case, a duplicator) can be transmitted on a, and successively applied to the functions sent on b_1 and b_2 (corresponding to processes emitting and receiving on c). After these three reductions, we obtain $\overline{c}\langle\bigstar\rangle \mid \overline{c}\langle\bigstar\rangle \mid c(z).\mathbf{0} \mid c(z').\mathbf{0}$, which can still do two synchronisations.

Type System. The grammar for types for HOpi$_\omega$ includes types for values, given by $T ::= \mathtt{unit} \mid (T \to^n \diamond)$, and channel types, of the form $Ch^n(T)$. We restrict ourselves to using only *well-formed* value types, defined as follows:

Definition 2 (Well-formed value types). *We say that T is a* well-formed value type *at level n w.r.t. a typing context Γ (written $Lvl_\Gamma(T) = n$ or simply $Lvl(T) = n$ when there is no ambiguity on Γ), whenever either $T = \mathtt{unit}$ and $n = 0$, or T' is a well-formed value type at level n', $T = T' \to^n \diamond$ and $n' < n$.*

The rules defining our type system for HOpi$_\omega$ are presented in Fig. 4. As in Sect. 2, types are annotated with a level, and the type assigned to a process is given by a natural number. The type of a process P is bound to dominate both the maximum level of outputs contained in P (not occurring inside a message), and the maximum level associated to function v_1, in applications $v_1\lfloor v_2\rfloor$ that occur in P not inside a message.

Typing values

$$\frac{}{\Gamma \vdash \bigstar : \mathtt{unit}} \qquad \frac{\Gamma, x : T \vdash P : n}{\Gamma \vdash x \mapsto P : T \rightarrow^{n+1} \diamond}$$

Typing processes

$$\frac{}{\Gamma \vdash \mathbf{0} : 0} \qquad \frac{\Gamma, a : Ch^k(T) \vdash P : n}{\Gamma \vdash (\nu a)P : n} \qquad \frac{\Gamma \vdash P_1 : n_1 \qquad \Gamma \vdash P_2 : n_2}{\Gamma \vdash P_1 \mid P_2 : \max(n_1, n_2)}$$

$$\frac{\Gamma \vdash v_1 : T \rightarrow^n \diamond \quad \Gamma \vdash v_2 : T}{\Gamma \vdash v_1 \lfloor v_2 \rfloor : n} \qquad \frac{\Gamma, x : T \vdash P : n \quad \Gamma(a) = Ch^k(T)}{\Gamma \vdash a(x).P : n} \qquad \frac{\Gamma \vdash v : T \qquad \Gamma \vdash P : n' \qquad \Gamma(a) = Ch^n(T) \qquad Lvl(T) = k \qquad n > k}{\Gamma \vdash \overline{a}\langle v \rangle.P : \max(n, n')}$$

Fig. 4. Typing Rules for HOpi_ω

Soundness. As before, we associate to a process a measure that decreases along reductions. Relying as above on $os(P)$, the multiset of names used in output subject position in P, does not work, because β-reduction may let $os(P)$ grow.

Definition 3 (Measure on processes in HOpi_ω). *Let P be a well-typed $HOpi_\omega$ process. We define $\mathcal{M}(P) = os(P) \uplus fun(P)$, where: (i) $os(P)$ is the multiset of the levels of the channel names that are used in an output in P, without this output occurring in message position. (ii) $fun(P)$ is defined as the multiset union of all $\{k\}$, for all $v_1 \lfloor v_2 \rfloor$ occurring in P not within a message, such that v_1 is of type $T \rightarrow^k \diamond$.*

Proposition 4 (Soundness). *If $\Gamma \vdash P : n$ for some $HOpi_\omega$ process P, then P terminates.*

Proposition 4 is established by observing that $\mathcal{M}(P)$ decreases at each step of transition:

- If the transition is a communication, the continuations of the processes involved in the communication contribute to the global measure the same way they did before communication, because a type preserving substitution is applied. $\mathcal{M}(P)$ decreases because an output has been consumed.
- If the transition is a β-reduction involving a function of level k, a process of level strictly smaller than k is spawned in P. Therefore, all new messages and active function applications that contribute to the measure are of a level strictly smaller than l, and $\mathcal{M}(P)$ decreases.

4 Controlling Communication and Passivation

PaPi: A Calculus with Locations and Passivation. The objective of this section is to study termination in presence of further constructs that are known to be challenging in the semantics of higher-order concurrent languages, notably

constructs of locations (i.e., explicit spatial distribution) and of passivation. We consider a calculus, which we refer to as PaPi (for '*Pa*ssivation *Pi*-calculus'), that integrates such constructs with the higher-order features of HOpi_2 and the name-passing capabilities of the π-calculus.

In PaPi, names belong to two sorts: they are either channels or locations. We use a, b, c to denote channels, and l to denote locations. We let n stand for any name, be it used as a channel or as a location, and names x, y, z will denote name variables. The syntax of PaPi is as follows:

$$\begin{aligned} P \;::=\; & \mathbf{0} \mid P|P \mid \overline{a}\langle n\rangle.P \mid a(x).P \mid !a(x).P \\ & \mid l[P] \mid l(X) \triangleright P \mid \overline{a}\langle P\rangle.P \mid a(X).P \mid (\nu n)P \;\; . \end{aligned}$$

Note that replication is allowed only on name-passing input prefixes. $l[P]$ stands for the process P running at location l (locations can be nested). The construct $l(X) \triangleright P$ corresponds to passivation, that is, the operation that consists in capturing a computation running at location l, calling it X, and proceeding according to P. Passivation can be found in calculi like Kells [SS05, HHH^+08] or Homer [HGB04].

The operational semantics of PaPi is described by the following reduction rules (we omit the rules for closure of reduction w.r.t. structural congruence, restriction and parallel composition):

$$\overline{\overline{a}\langle n\rangle.P \mid a(x).Q \to P \mid Q[n/x]} \qquad \overline{\overline{a}\langle P\rangle.Q_1 \mid a(X).Q_2 \to Q_1 \mid Q_2[P/X]}$$

$$\overline{l[Q] \mid l(X) \triangleright P \to P[Q/X]} \qquad \frac{P \to P'}{l[P] \to l[P']}$$

It has to be noted that we do not claim here that the combination of primitives provided in PaPi (first and higher-order message passing, localised interaction, passivation) makes this calculus a proposal for a model for distributed or component based programming. Indeed, important interaction mechanisms such as communication between distant locations, or subjective mobility, are not available in PaPi.

Our primary goal is instead to study how the constructs of PaPi, which have the advantage of being presented in a rather simple way, can be taken into account in our termination analysis. We believe that the way we handle these can be smoothly adapted to small variations: for instance, typing distant communication in kπ [HHH^+08] should be done pretty much like we type local communication in PaPi.

We now provide a few examples of PaPi processes to illustrate typical idioms that can be programmed using passivation.

$$\begin{array}{ccc} & \mathit{Dup} \quad c(r).l(X) \triangleright \big(l[X] \mid (\nu l')\,(\overline{r}\langle l'\rangle \,|\, l'[X]) \big) & \\ \mathit{Res} \quad c(l).l(X) \triangleright l[P_0] & & \mathit{DynUpd} \quad c(l).d(X).(\, l(Y) \triangleright l[X] \,) \\ & \mathit{Coloc} \quad l_1(X) \triangleright \big(l_2(Y) \triangleright (l_1[X|Y] \mid l_2[\mathbf{0}])\big) & \end{array}$$

Dup performs code duplication: when a message is received on channel c, the computation running at location l is duplicated, and the location of the new copy is sent back on r, the channel transmitted along c.

Process *Res* (reset): upon reception of a location name l along c, the computation taking place at l is replaced with P_0, that can be considered as a start state. Essentially the same "program" can be used when we want to replace the code running at l with a new version, that is transmitted along some channel d: this is a form of dynamic update (process *DynUpd*).

"Co-localisation": processes running at locations l_1 and l_2 are put together, and computation proceeds within location l_1. This might trigger new interactions between formerly separated processes. This is a form of *objective mobility* (running computations are being moved around).

Termination in PaPi. In PaPi, divergences arise both from recursion in usages of the passivation and process-passing mechanisms, and from recursive calls in the continuation of replicated (name-passing) inputs. We control the latter source of divergences by resorting to the type discipline of [DS06], while the former is controlled by associating levels to locations and to process-carrying channels.

However, the mere superposition of these two systems (of Sect. 2.2 and of [DS06]) does not work, as the two mechanisms can cooperate to produce divergences. Indeed, consider the non terminating process $P_1 = l(X) \triangleright !a.X \mid l[\overline{a}] \mid \overline{a}$ (channel a, which is used in a CCS-like fashion, carries values of type `unit`). The usages of passivation (which can be treated as a form of process passing) and name passing in P_1 are unfortunately compliant with the principles of the aforementioned type systems. In this particular case, we must take into account the fact that X can be instantiated by a process containing an output on a channel having the same level as a. More generally, we must understand how the two type systems can interact, in order to avoid diverging behaviours.

In PaPi, every entity (process, location, name-passing channel and process-passing channel) is given a level which is used to control both sources of divergences. The base types for names are `unit` and **loc** (for location names). The level of a name-passing channel a corresponds to the maximum level allowed for the continuation P in a replicated input of the form $!a(x).P$. The level of a process-passing channel corresponds to the maximum level of a process sent on this channel. The level of a location corresponds to the maximum level a process executing at this location can have. In turn, the level of a process P corresponds to the maximum level of messages and locations that occur in P neither within a higher-order output nor under a replication.

The rules defining the type system for termination in PaPi are given in Fig. 5. As far as typing termination is concerned, higher-order inputs (resp. outputs) are typed like passivations (resp. located processes). We can moreover remark that this type system subsumes the type system of [DS06] for the π-calculus: if a π process P is typable according to [DS06], then it is typable as a PaPi process.

Remark 3. It has to be noted that the type system we present can be made more expressive by exploiting ideas from Sect. 2.3. Indeed, what we control here

$$\frac{}{\Gamma \vdash \mathbf{0} : 0} \qquad \frac{\Gamma(X) = m}{\Gamma \vdash X : m} \qquad \frac{\Gamma, v : T \vdash P : m}{\Gamma \vdash (\nu v)\ P : m}$$

$$\frac{\Gamma \vdash P_1 : m_1 \qquad \Gamma \vdash P_2 : m_2}{\Gamma \vdash P_1 \mid P_2 : \max(m_1, m_2)} \qquad \frac{\Gamma, X : k-1 \vdash P : m \qquad \Gamma(l) = \mathbf{loc}^k}{\Gamma \vdash l(X) \triangleright P : m}$$

$$\frac{\Gamma(l) = \mathbf{loc}^k \qquad \Gamma \vdash Q : m' \qquad k > m'}{\Gamma \vdash l[Q] : k} \qquad \frac{\Gamma, X : k-1 \vdash P : m \qquad \Gamma(a) = Ch^k(\diamond)}{\Gamma \vdash a(X).P : m}$$

$$\frac{\begin{array}{c}\Gamma \vdash P : m \qquad \Gamma \vdash Q : m' \\ \Gamma(a) = Ch^k(\diamond) \qquad k > m'\end{array}}{\Gamma \vdash \overline{a}\langle Q \rangle.P : \max(k, m)} \qquad \frac{\Gamma, x : T \vdash P : m \qquad \Gamma(a) = Ch^k(T)}{\Gamma \vdash a(x).P : m}$$

$$\frac{\begin{array}{c}\Gamma \vdash P : m \qquad \Gamma \vdash v : T \\ \Gamma(a) = Ch^k(T)\end{array}}{\Gamma \vdash \overline{a}\langle v \rangle.P : \max(k, m)} \qquad \frac{\begin{array}{c}\Gamma, x : T \vdash P : m \\ \Gamma(a) = Ch^k(T) \qquad k > m\end{array}}{\Gamma \vdash !a(x).P : 0}$$

Fig. 5. Typing Rules for PaPi

using a unique level for names could be refined by associating channels with three natural numbers: one is its weight, and the other two are interpreted as capacities, to control the two sources of recursion: the weight of name passing outputs on one side, and the weight of process passing outputs and located processes on the other side. In what we have presented, these three components of the type of a name are merged into a single one.

Termination. For lack of space, we do not present the soundness proof of our type system for PaPi. It essentially follows the same strategy as in the previous sections. At its core is the definition of a measure on processes, that takes into account the contribution of locations and first- and higher-order messages that do not occur within a message. We then show that this measure decreases along reductions, which finally gives:

Proposition 5. *If $\Gamma \vdash P : m$ for a PaPi process P, then P terminates.*

Examples of typing. Process P_1 seen above cannot be typed. The typing rule for locations forces the level of the location l to be strictly greater than $lvl(a)$ when typing $l[\overline{a}]$. The typing rule of passivation forces the level of l to be equal to $1 + lvl(X)$. Thus $lvl(X) \leq lvl(a)$ and the typing rule for replicated inputs cannot be applied to $!a.X$.

For process *Coloc* to be typable, $lvl(l_1)$, the level assigned to l_1, should be greater than $lvl(l_2)$. In this case, we can observe that it is safe to take two processes running in separate locations and let them run in parallel, as *Coloc* does: while this might trigger new interactions (inter-locations communication is forbidden in PaPi), this is of no harm for termination.

5 Concluding Remarks

In this paper, we have analysed termination in higher-order concurrent languages, using the higher-order π-calculus as a core formalism to build the basis of our type systems. For future work, we plan to examine how the type systems we have presented can be adapted to existing process calculi in which processes can be exchanged in communications or can move among locations such as, e.g., Ambients [CG98], Homer [HGB04], Kells [SS05, HHH$^+$08]. Another question we would like to address is type inference; for this, [DHKS07] could serve as a starting point.

References

[CG98] L. Cardelli and A. D. Gordon. Mobile Ambients. In *Proc. of FoSSaCS'98*, volume 1378 of *LNCS*, pages 140–155. Springer, 1998.

[DHKS07] R. Demangeon, D. Hirschkoff, N. Kobayashi, and D. Sangiorgi. On the Complexity of Termination Inference for Processes. In *Proc. of TGC'07*, volume 4912 of *LNCS*, pages 140–155. Springer, 2007.

[DHS08] R. Demangeon, D. Hirschkoff, and D. Sangiorgi. Static and Dynamic Typing for the Termination of Mobile Processes. In *Proc. of IFIP TCS'08*. Springer, 2008.

[DHS09] R. Demangeon, D. Hirschkoff, and D. Sangiorgi. Termination in Higher-Order Concurrent Calculi. long version of this paper; in preparation, 2009.

[DS06] Y. Deng and D. Sangiorgi. Ensuring Termination by Typability. *Information and Computation*, 204(7):1045–1082, 2006.

[HGB04] T. Hildebrandt, J. C. Godskesen, and M. Bundgaard. Bisimulation Congruences for Homer — a Calculus of Higher Order Mobile Embedded Resources. Technical Report TR-2004-52, Univ. of Copenhagen, 2004.

[HHH$^+$08] D. Hirschkoff, T. Hirschowitz, S. Hym, A. Pardon, and D. Pous. Encapsulation and Dynamic Modularity in the Pi-calculus. In *Proc. of the PLACES'08 workshop*, ENTCS. Elsevier, 2008. to appear.

[NP00] U. Nestmann and B. C. Pierce. Decoding Choice Encodings. *Information and Computation*, 163(1):1–59, 2000.

[San92] D. Sangiorgi. Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms. *PhD Thesis, University of Edinburgh*, 1992.

[San06] D. Sangiorgi. Termination of Processes. *Mathematical Structures in Computer Science*, 16(1):1–39, 2006.

[SS05] A. Schmitt and J.-B. Stefani. The Kell Calculus: A Family of Higher-Order Distributed Process Calculi. In *Proc. of TGC'05*, volume 3267 of *LNCS*, pages 146–178. Springer, 2005.

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

[Tho96] B. Thomsen. Calculi for Higher Order Communication Systems. *PhD Thesis, University of London*, 1996.

[YBH04] N. Yoshida, M. Berger, and K. Honda. Strong Normalisation in the Pi-Calculus. *Information and Computation*, 191(2):145–202, 2004.