Leaf-First Zipper Semantics

Sergueï Lenglet, Alan Schmitt

Leaf-First Zipper Semantics

Sergueï Lenglet[1,2] and Alan Schmitt[3]

[1] Université de Lorraine, Nancy, France
[2] Université Sorbonne Paris Nord, Villetaneuse, France
[3] INRIA, Rennes, France

Abstract. Biernacka et al. recently proposed *zipper semantics*, a semantics format from which sound and complete abstract machines for non-deterministic languages can be automatically derived. We present a new style of zipper semantics, called *leaf-first*, in which we express the semantics of two extensions of HOπ, a higher-order version of the π-calculus: one with passivation and the other with join patterns. The leaf-first style is better suited than the original one to express phenomena occurring in process calculi semantics such as scope extrusion, which is observable with passivation and complex with join patterns.

Keywords: Process calculi · Abstract machines · Scope extrusion.

1 Introduction

In concurrency theory, abstract machines are used as an implementation model [9, 11, 19, 21, 27], in particular to study the distribution of computation [1, 12–14, 16, 22]. The design of these machines is often ad-hoc, following principles which apply to the considered calculus only. Besides, some of the machines are not complete, i.e., there exist reduction paths of the language which cannot be simulated by its abstract machine [9, 11, 13, 19, 27]. To overcome these issues, Biernacka et al. [2, 3] recently proposed Non-Deterministic Abstract Machines (NDAM), a generic design of abstract machines for non-deterministic languages. An NDAM explores the term in the search for a redex, making arbitrary choices when several paths are possible. The machine backtracks when it reaches a normal form, annotating it to not try it again, thus avoiding infinite loops during the search.

As writing the definition of an NDAM can be tedious, the authors also propose a sound and complete automatic derivation procedure from a *zipper semantics*, a format in between a Structural Operational Semantics (SOS) [23] and a context-based reduction semantics [7, 8]. Like in a SOS, a zipper semantics explores a term with structural rules, but it remembers its position in the term using an evaluation context, a syntactic object that represents a term with a hole [8]. A zipper semantics respecting some properties can then be augmented into an NDAM by adding the annotations and backtracking mechanisms.

Biernacka et al. [2] define a zipper semantics for several calculi including HOπ [24], a process calculus where messages are executable terms. Like the π-calculus [25], HOπ features name restriction, a construct which restricts the scope

of a communication channel, hiding it to the outside. When a communication happens between a sender and a receiver, it may be necessary to enlarge the scope of restricted names to include the receiver: this phenomenon is known as *scope extrusion*. Biernacka et al. consider a variant of HOπ with *eager* scope extrusion, where the scope of all name restrictions around the message output are extended to include the input. In contrast, in *lazy* scope extrusion, only the restrictions of the names occurring in the message contents are extended.

While the difference between lazy and eager scope extrusion cannot be observed in HOπ, it may lead to distinguishable behaviors in calculi with localities and *passivation* [1, 18], a feature used in component models to interrupt and capture the state of running components [26]. Lazy scope extrusion is also quite intricate in presence of *join patterns* [11], where a single receiver may receive messages from different senders at once. The scope of lazily extruded names should encompass the process sending the message and the receiver, and no more. With several messages emitted at once, the scopes of names restricting two distinct messages are different and depend on the respective output processes.

In this paper, we encode lazy scope extrusion in various settings by defining a new format of zipper semantics, called *leaf-first*. Our contributions are: we express the semantics of HOπ with passivation and lazy scope extrusion in our new leaf-first format, and we show that our approach can also handle an extension of HOπ with join patterns. Our format respects the zipper semantics requirements, so we can apply Biernacka et al.'s derivation procedure to obtain an NDAM from it, and we get for free abstract machines for these calculi.

The paper is organized as follows. Section 2 presents the syntax and lazy semantics of HOπ with passivation; we recall an example illustrating how we can distinguish lazy from eager scope extrusion in that calculus. We then present a leaf-first zipper semantics for this calculus, that we prove equivalent to the usual one. Section 3 follows the same structure for HOπ with join patterns. Section 4 discusses related work and concludes. The accompanying research report [17] contains the missing proofs.

2 HOπ with Passivation

We present the lazy semantics of HOπP (HOπ with passivation) [18], that we characterize using the new zipper semantics style we propose in this paper.

2.1 Syntax and Lazy Semantics

Syntax. HOπP [18] is a higher-order calculus—where messages are executable processes—extended with hierarchical localities and passivation. We assume countable sets of channel names, ranged over by lowercase letters a, b, etc, and of process variables ranged over by X, Y, Z. The syntax of processes is as follows.

$$P, Q, R, S, M, K ::= X \mid \mathbf{0} \mid P \parallel Q \mid a(X).R \mid \overline{a}\langle M \rangle K \mid \nu a.P \mid a[P]$$

Informally, $\mathbf{0}$ is the inactive process, the parallel composition $P \parallel Q$ is executing P and Q concurrently, $\nu a.P$ restricts the scope of channel a to P, an output $\overline{a}\langle M\rangle K$ is sending M on a and continues as K, while $a(X).R$ is waiting for a message M on a to continue as R where X is replaced by M. A process may run in a locality $a[P]$, which can be *passivated* (dissolved) at any time, interrupting the execution of P and saving its current state in a message on a.

For readability, we often use S to denote a sending process, R a receiving one, M a message, and K a continuation, but these all stand for processes. In examples, we also often omit $\mathbf{0}$ in output continuations or in parallel processes, abbreviating $\overline{a}\langle P\rangle\mathbf{0}$ as $\overline{a}\langle P\rangle$ and $P \parallel \mathbf{0}$ as P. The scope of name restriction extends to the right as much as possible, writing $\nu a.(P \parallel Q)$ as $\nu a.P \parallel Q$.

Scope extrusion. The scope of a in $\nu a.P$ is restricted to P, so that a communication on a is possible inside P only. It does not prevent communication on other names, but the scope of restricted names present in a message has to be enlarged to prevent them from escaping their binder. For example, we have

$$b(X).(X \parallel \overline{c}\langle\mathbf{0}\rangle) \parallel \nu ad.\overline{b}\langle\overline{a}\langle\mathbf{0}\rangle\rangle \parallel \overline{d}\langle\mathbf{0}\rangle \xrightarrow{\tau} \nu a.\overline{a}\langle\mathbf{0}\rangle \parallel \overline{c}\langle\mathbf{0}\rangle \parallel \nu d.\overline{d}\langle\mathbf{0}\rangle$$

The scope of a is extended to include the receiver, a phenomenon known as *scope extrusion*. The scope of d, however, remains the same: we only extend the scope of names occurring in the message. The scope extrusion is thus said to be *lazy*.

In $\nu a.P$, a is bound in P, and $a(X).R$ binds X in R. We write $\mathsf{fn}(P)$ for free names of P, defined as expected. To avoid unwanted captures (in particular during scope extrusion), we henceforth assume bound names and variables to be pairwise distinct, and distinct from the free names and variables of all the processes under consideration, using α-conversion if necessary. This convention matters for instance for the rules outPar, outLoc, and tauCom of Figure 1.

Lazy semantics. Given an entity ranged over by e, we write $\overrightarrow{e}$ for a sequence $(e_1 \dots e_n)$. We write $()$ for the empty sequence, $e' \cdot \overrightarrow{e}$ for the extension of a sequence to the left, and $\overrightarrow{e_1} \cdot \overrightarrow{e_2}$ for the concatenation of two sequences. We write $P\{Q/X\}$ for the usual capture-avoiding substitution of X by Q in P.

We define the labeled operational semantics with three judgments: $P \xrightarrow{\tau} P'$ for silent steps, $P \xrightarrow{a(M)} R$ for a process that inputs the message M on a, and $P \xrightarrow{\nu \overrightarrow{b}.\overline{a}\langle M\rangle} K$ for a process that outputs the message $\overline{a}\langle M\rangle$ while extruding the names in $\overrightarrow{b}$. We give the rules in Figure 1, except the symmetric counterpart of the rules marked with (s), a convention we follow from now on.

A transition $P \xrightarrow{\tau} P'$ can be decomposed as follows. First, we apply the rules tauPar, tauLoc, and tauNu to reach the parallel composition of the communication. At this point, rule tauCom is applied, with two premises. In one premise, output rules outPar, outLoc, outExtr, and outNu are applied until the messaging process. The last two rules are for name restriction, the former is used when the name needs to be extruded (it is free in the message contents), while the latter is used when no extrusion is needed. The message may

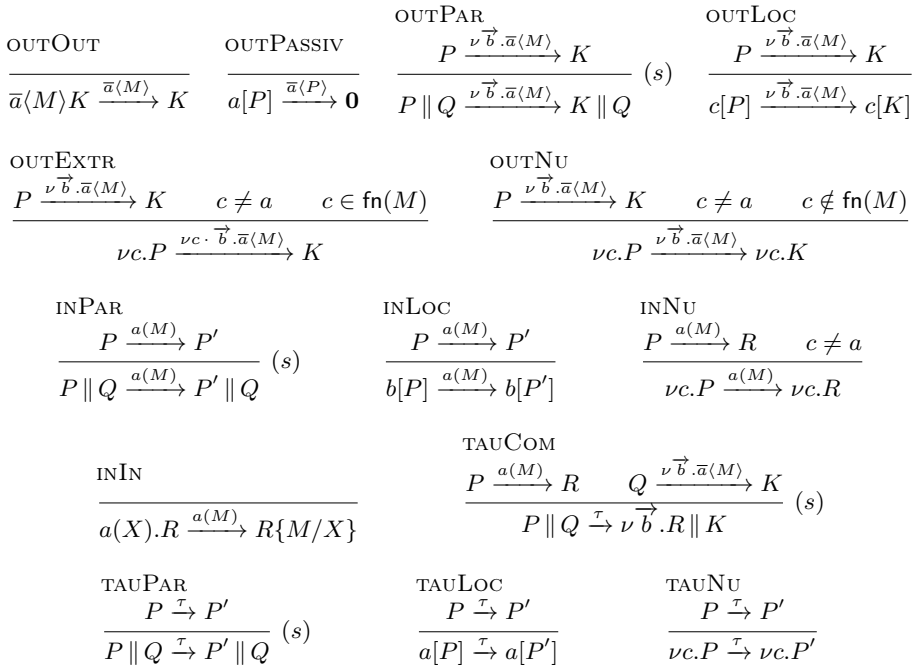

Fig. 1. Lazy Semantics of HOπP

be the result of an output (rule outOut) or a passivation (rule outPassiv). In the other premise of the tauCom rule, the input is reached through rules inPar, inLoc, and inNu, and the actual input is done in rule inIn. Finally, the conclusion of tauCom restores the name restrictions that have been extruded.

Example. We show the difference between lazy and eager scope extrusion in a calculus with passivation. We write $a(X).P$ as $a(\_).P$ if X is not free in P. Let $P = a[\nu c.\overline{b}\langle \mathbf{0}\rangle \overline{c}\langle \mathbf{0}\rangle \parallel c(\_).c(\_).\overline{d}\langle \mathbf{0}\rangle] \parallel b(\_).a(X).X \parallel X$. We communicate on b and then passivate the locality on a, duplicating its content. With eager scope extrusion, the scope of c is expanded outside a:

$$\begin{aligned} P &\xrightarrow{\tau} \nu c.a[\overline{c}\langle \mathbf{0}\rangle \parallel c(\_).c(\_).\overline{d}\langle \mathbf{0}\rangle] \parallel a(X).(X \parallel X) \\ &\xrightarrow{\tau} \nu c.\overline{c}\langle \mathbf{0}\rangle \parallel c(\_).c(\_).\overline{d}\langle \mathbf{0}\rangle \parallel \overline{c}\langle \mathbf{0}\rangle \parallel c(\_).c(\_).\overline{d}\langle \mathbf{0}\rangle \end{aligned}$$

The name c is then shared between the copies of the duplicated process, and a message output on d becomes possible after two communications on c. With lazy scope extrusion, the scope of c does not change:

$$\begin{aligned} P &\xrightarrow{\tau} a[\nu c.\overline{c}\langle \mathbf{0}\rangle \parallel c(\_).c(\_).\overline{d}\langle \mathbf{0}\rangle] \parallel a(X).X \parallel X \\ &\xrightarrow{\tau} \nu c.(\overline{c}\langle \mathbf{0}\rangle \parallel c(\_).c(\_).\overline{d}\langle \mathbf{0}\rangle) \parallel \nu c.(\overline{c}\langle \mathbf{0}\rangle \parallel c(\_).c(\_).\overline{d}\langle \mathbf{0}\rangle) \end{aligned}$$

Each duplicated process has its own copy of c, and it is no longer possible for the message output on d to trigger.

2.2 Root-First Zipper Semantics

Zipper semantics exhibits the step-by-step decomposition of a term into a context and a redex. Like a SOS [23], it traverses a term in the search for a redex using structural rules. It also makes explicit the current position in the term using an evaluation context, like in context-based reduction semantics [8]. We represent a context as a stack of elementary contexts $\mathfrak{F}$, called *frames*, of four kinds.

$$\mathbb{E}, \mathbb{F}, \mathbb{G}, \mathbb{H} ::= \bullet \mid \mathfrak{F} :: \mathbb{E} \qquad \mathfrak{F} ::= \parallel P \mid P \parallel \mid \nu a \mid a[]$$

Each frame corresponds to a set of similar structural rules in the definition of the lazy semantics. The decomposition process happening in a zipper semantics makes it more convenient to interpret a context *inside-out*: the topmost frame is the innermost one in the term. We define the operation of plugging P inside a frame $\mathfrak{F}[\![P]\!]$ and inside a context $\mathbb{E}[\![P]\!]$ as follows.

$$\bullet[\![P]\!] \triangleq P \qquad (\mathfrak{F} :: \mathbb{E})[\![P]\!] \triangleq \mathbb{E}[\![\mathfrak{F}[\![P]\!]]\!]$$
$$(\parallel Q)[\![P]\!] \triangleq P \parallel Q \qquad (Q \parallel)[\![P]\!] \triangleq Q \parallel P \qquad (\nu a)[\![P]\!] \triangleq \nu a.P \qquad (a[])[\![P]\!] \triangleq a[P]$$

We use contexts to indicate where the current focus is when exploring a term: for instance, $\mathbb{E}[\![P \parallel Q]\!]$ means that the current focus is on the parallel composition. To focus on P, we consider $(\parallel Q :: \mathbb{E})[\![P]\!]$: going further down the term consists in pushing a frame on top of the context. In contrast, going towards the root amounts to popping the top frame from the context.

In most semantics of process calculi based on labeled transitions, an input $a(X).R$ communicates with an output $\overline{a}\langle M\rangle K$ even if they are not directly in parallel. The zipper semantics therefore aims at decomposing a term as a redex in some evaluation context $\mathbb{E}$, the redex itself being of the form $\mathbb{F}[\![S]\!] \parallel \mathbb{G}[\![a(X).R]\!]$ or $\mathbb{G}[\![a(X).R]\!] \parallel \mathbb{F}[\![S]\!]$ with $S = \overline{a}\langle M\rangle K$ or $S = a[M]$. The strategy followed by Biernacka et al. [2, Appendix C] to decompose a HOπ process P proceeds in three steps: starting from the top of P, search for the parallel composition of the communication, building $\mathbb{E}$ at the same time. At that point, P is decomposed as $\mathbb{E}[\![P_1 \parallel P_2]\!]$. Then, look for the output in either P_1 or P_2, decomposing it as $\mathbb{F}[\![\overline{a}\langle M\rangle K]\!]$. At the same time, $\mathbb{F}$ is split into $\mathbb{F}_\parallel$ and $\mathbb{F}_\nu$, so that $\mathbb{F}_\parallel$ contains the parallel compositions and $\mathbb{F}_\nu$ the name restrictions of $\mathbb{F}$. Finally, look for the input in the other process, decomposing it as $\mathbb{G}[\![a(X).R]\!]$. Once all the ingredients have been found, the result of the communication is either $\mathbb{E}\Big[\!\Big[\mathbb{F}_\nu[\![\mathbb{F}_\parallel[\![K]\!] \parallel \mathbb{G}[\![R\{Q/X\}]\!]]\!]\Big]\!\Big]$ or $\mathbb{E}\Big[\!\Big[\mathbb{F}_\nu[\![\mathbb{G}[\![R\{Q/X\}]\!] \parallel \mathbb{F}_\parallel[\![K]\!]]\!]\Big]\!\Big]$. We refer to such an exploration strategy as *root-first*, because it starts the decomposition with the operator at the root of the redex (the parallel composition).

In the final result, the scope extrusion is *eager* because $\mathbb{F}_\nu$ collects all the enclosing name restrictions of $\mathbb{F}$, not only those restricting the free names of M.

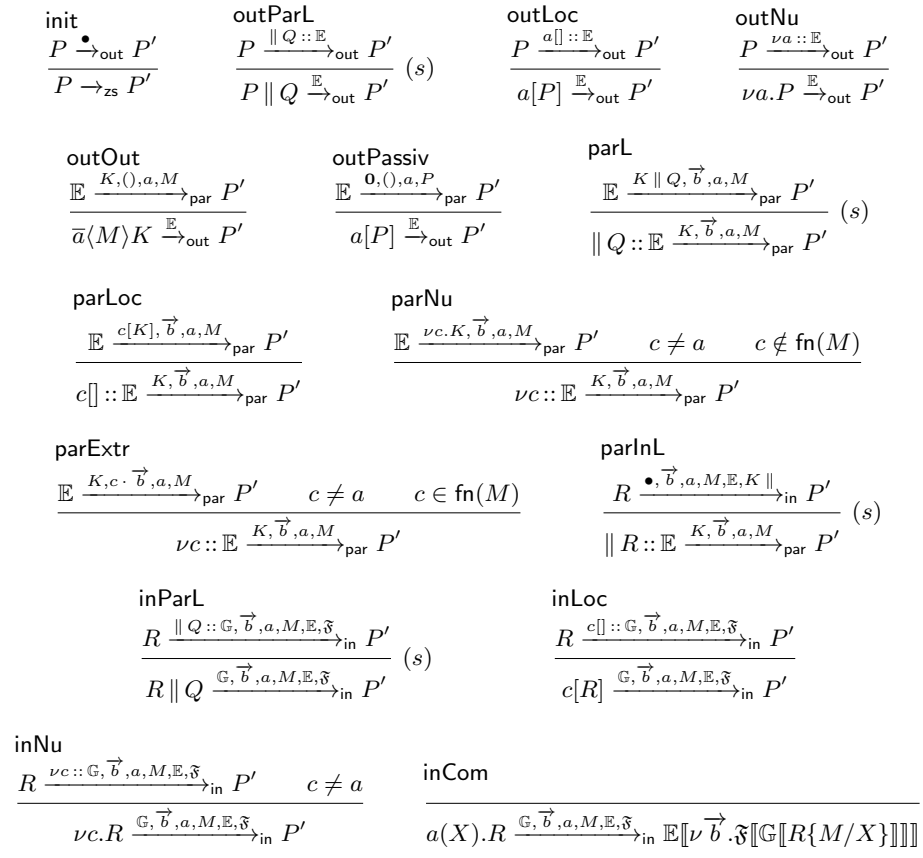

Fig. 2. Leaf-First Zipper semantics for lazy HOπP

The problem is that the decomposition of $\mathbb{F}$ happens while going through the sending process, while the message M is not yet known: when we reach it, the decomposition of $\mathbb{F}$ is already over. Therefore, while a root-first strategy is enough to express the λ-calculus or a process calculus with eager scope extrusion, it is not fit to represent the semantics of a calculus with lazy scope extrusion.

2.3 Leaf-First Zipper Semantics

As said before, finding the extruded names in $\mathbb{F}[\![\overline{a}\langle M\rangle K]\!]$ requires us to know M when going through $\mathbb{F}$. We therefore search for the output first, then for the parallel composition and the input. Because the output is a leaf of the redex, we refer to such a strategy as a *leaf-first* zipper semantics. It is defined in Figure 2.

A zipper transition $P \to_{\mathsf{zs}} P'$ is defined using three auxiliary transitions (or *modes*) out, par, and in. The out mode looks for a message either from an output

or a passivation. The rule init starts the search at the top of the term. The rules outParL, outParR, outLoc, and outNu change the current focus to a subterm. When we apply the rule outOut or outPassiv, the initial term is decomposed as $\mathbb{E}[\![\overline{a}\langle M\rangle K]\!]$ or $\mathbb{E}[\![a[P]]\!]$, and we switch to the par mode.

The par mode searches the parallel composition which separates the output and input processes, computing on the way the extruded names. The transition $\mathbb{E} \xrightarrow{K,\overrightarrow{b},a,M}_{\mathsf{par}} P'$ goes through $\mathbb{E}$ looking for the parallel composition, constructing K (the continuation of the output) and $\overrightarrow{b}$ (the extruded names) while doing so, and remembering that M is sent on a. The rules outOut and outPassiv initialize the par mode with the proper continuation and with an empty sequence of extruded names. If the context is of the form $\nu c :: \mathbb{E}$, the name restriction νc surrounds the message output, so we should check that it does not prevent the communication on a, hence the premise $c \neq a$ in rules parNu and parExtr. If c occurs free in the message M, it must be extruded, and it is added to the sequence $\overrightarrow{b}$ in the rule parExtr. Otherwise, it is added to K in the rule parNu.

A locality is moved from the context to the continuation (rule parLoc). If the context is of the shape $\| Q :: \mathbb{E}$ or $Q \| :: \mathbb{E}$, we consider a process of the form $\mathbb{E}[\![K \| Q]\!]$ or $\mathbb{E}[\![Q \| K]\!]$ and we have two possibilities. If Q is not the process receiving the message, then Q has to be added to the continuation, and we continue searching for the receiver in $\mathbb{E}$ (rules parL and parR). Otherwise, we have found the parallel composition where the communication takes place, which is where the extruded names $\overrightarrow{b}$ should be restricted. We remember this position with $\mathbb{E}$, and search for the input in Q with a transition $Q \xrightarrow{\mathbb{G},\overrightarrow{b},a,M,\mathbb{E},\mathfrak{F}}_{\mathsf{in}} P'$, where $\mathbb{G}$ is the context surrounding Q up to the parallel composition (initially $\bullet$), and $\mathfrak{F}$ records if the continuation K is to the left (then $\mathfrak{F} = K \|$) or to the right (then $\mathfrak{F} = \| K$) of Q (rules parInL and parInR).

The in mode is then going through the receiving process, pushing the constructs on the context $\mathbb{G}$ (rules inParL, inParR, parLoc, and inNu). Once the input $a(X).R$ has been found, all the pieces have been collected for the communication to happen, resulting in $\mathbb{E}\Big[\!\Big[\nu\,\overrightarrow{b}\,.\mathfrak{F}[\![\mathbb{G}[\![R\{M/X\}]\!]]\!]\Big]\!\Big]$. For instance, if we start from a process $\mathbb{E}\Big[\!\Big[\mathbb{F}[\![\overline{a}\langle M\rangle K]\!] \,\|\, \mathbb{G}[\![a(X).R]\!]\Big]\!\Big]$, then $\mathfrak{F} = \mathbb{F}'[\![K]\!] \|$ where $\mathbb{F}'$ is what is left from $\mathbb{F}$ after removing the restricted names $\overrightarrow{b}$.

2.4 Properties of the Leaf-First Semantics

Correspondence results. We state how the leaf-first zipper semantics relates to the lazy semantics of HOπP.

Theorem 1. *If* $R \xrightarrow{\mathbb{G},\overrightarrow{b},a,M,\mathbb{E},\mathfrak{F}}_{\mathsf{in}} P'$, *then* $R \xrightarrow{a(M)} R'$ *for some* R' *and* $P' = \mathbb{E}[\![\nu\,\overrightarrow{b}\,.\mathfrak{F}[\![\mathbb{G}[\![R']\!]]\!]]\!]$. *If* $\mathbb{E} \xrightarrow{K,\overrightarrow{b},a,M}_{\mathsf{par}} P'$, *then for all* $S \xrightarrow{\nu\,\overrightarrow{b}\,.\overline{a}\langle M\rangle} K$, *we have* $\mathbb{E}[\![S]\!] \xrightarrow{\tau} P'$. *If* $P \xrightarrow{\mathbb{E}}_{\mathsf{out}} P'$, *then* $\mathbb{E}[\![P]\!] \xrightarrow{\tau} P'$.

From the last item, we deduce that $P \to_{\mathsf{zs}} P'$ implies $P \xrightarrow{\tau} P'$. For the reverse implication, given $P \xrightarrow{\tau} P'$, we make explicit the contexts in P and P', writing

them as $P = \mathbb{E}[\![\mathbb{H}[\![\overline{a}\langle M\rangle K]\!] \parallel R]\!]$ and $P' = \mathbb{E}[\![\nu \overrightarrow{b}.\mathbb{G}[\![K]\!] \parallel R']\!]$, with $R \xrightarrow{a(M)} R'$, and $\mathbb{H}$ built from $\overrightarrow{b}$ and $\mathbb{G}$. From there, we can reconstruct the zipper derivation; the proof is in the appendix.

Theorem 2. *If $P \xrightarrow{\tau} P'$, then $P \rightarrow_{\mathsf{zs}} P'$.*

Derivation into a NDAM. The semantics of Figure 2 fits the zipper semantics generic format [3]: each traversing rule (distinct from init and inCom) has exactly one premise, and is decomposing its source term one operator at a time.

A zipper semantics can be derived into an NDAM if it satisfies some properties, like being *machine constructive*: in each rule, it should be possible to construct the terms in the premise from those of the conclusion. This property holds if the meta-variables (of processes, contexts, names, etc) of the premise are included in those of the conclusion, as one can check in Figure 2. The semantics should also be *reversible*: conversely, we can recreate the terms in the conclusion from the premise, to allow for backtracking. It is the case because for each rule, the meta-variables of the conclusion are included in those of the premise.

Finally, the zipper semantics must be *terminating*, ensuring that the search for a redex in the NDAM does not infinitely loop. To prove it, it is enough to exhibit a strictly decreasing size on transitions such that the size of the premise is smaller than its conclusion. For each rule, the source term of the premise is a subterm of the source term of the conclusion, except when changing mode. We therefore consider a lexicographic ordering on modes first, so that $\mathsf{out} > \mathsf{par} > \mathsf{in}$, and then on the size of the source term of the transition.

3 HOπ with Join Patterns

We consider HOπJ, an extension of HOπ with join patterns [11], where an input may receive an arbitrary number of messages simultaneously.

3.1 Syntax and Lazy Semantics

Syntax and informal semantics. We let ξ range over *message patterns*, which can be seen as multisets of elementary inputs $a_1(X_1) \mid \ldots \mid a_n(X_n)$ where the X_i are pairwise distinct. We change the input process with $\xi \triangleright R$.

$$\xi ::= a(X) \mid \xi \mid \xi \qquad P, Q, \ldots ::= X \mid \mathbf{0} \mid P \parallel Q \mid \xi \triangleright R \mid \overline{a}\langle M\rangle K \mid \nu a.P$$

A communication happens when enough message outputs are in parallel to fulfil the message pattern of an input. Like in HOπ or HOπP, name restriction delimits the scope of names, and lazy scope extrusion may be necessary during communication. However, to keep it as lazy as possible, restricted names should not be extruded further than the parallel composition enclosing the sender and receiver. Consider the following example, where $\xi = a_1(X_1) \mid a_2(X_2) \mid a_3(X_3) \mid a_4(X_4)$ and

we abbreviate $\overline{b}\langle\mathbf{0}\rangle\mathbf{0}$ as $\overline{b}$ for any b. We number the parallel compositions in the redex, which has the same parallel structure as in Figure 4.

$$P \parallel \Big(\big((\nu b.\overline{a_1}\langle\overline{b}\rangle K_1) \parallel^2 (\nu c.\nu d.\overline{a_2}\langle\mathbf{0}\rangle K_2 \parallel^3 \overline{a_3}\langle\overline{d}\rangle K_3) \big) \parallel^1 \big((\nu e.\overline{a_4}\langle\overline{e}\rangle K_4) \|^4 \xi \triangleright R \big) \Big)$$
$$\xrightarrow{\tau} P \parallel \nu b.\nu d.\Big(\big(K_1 \parallel^2 (\nu c.K_2 \parallel^3 K_3) \big) \parallel^1 \nu e.(K_4 \|^4 R') \Big)$$

The scope of c is unchanged, while b and d have been extruded to include $\parallel^1$, and e to include $\parallel^4$.

Lazy semantics. We let p range over pairs (a, M) of a message and its output channel. Given a sequence $\overrightarrow{p}$, we define the sequence of *communication names* inductively so that $\mathsf{cn}(()) \triangleq ()$ and $\mathsf{cn}((a, M) \cdot \overrightarrow{p}) \triangleq a \cdot \mathsf{cn}(\overrightarrow{p})$, and the set of *message names* as $\mathsf{mn}(()) \triangleq \emptyset$ and $\mathsf{mn}((a, M) \cdot \overrightarrow{p}) \triangleq \mathsf{fn}(M) \cup \mathsf{mn}(\overrightarrow{p})$.

We assume an operator $\xi \blacksquare \overrightarrow{p}$ which checks that there are as many messages in $\overrightarrow{p}$ as names in the pattern ξ and computes the possible matchings as substitutions. For example, $a(X) \mid a(Y) \blacksquare (a, P), (a, Q)$ generates $X \mapsto P, Y \mapsto Q$ and $X \mapsto Q, Y \mapsto P$. We let θ range over these substitutions, and write $\theta \in \xi \blacksquare \overrightarrow{p}$ when θ is a possible matching between ξ and $\overrightarrow{p}$, the latter considered as a multiset. Formally, we write $\uplus$ for the multiset union, and we define $a(X) \blacksquare (a, M) \triangleq \{x \mapsto M\}$ and $(\xi_1 \mid \xi_2) \blacksquare \overrightarrow{p} \triangleq \{\theta_1 \uplus \theta_2 \mid \theta_1 \in \xi_1 \blacksquare \overrightarrow{p_1}, \theta_2 \in \xi_2 \blacksquare \overrightarrow{p_2}, \overrightarrow{p} = \overrightarrow{p_1} \uplus \overrightarrow{p_2}\}$.[4]

We define a labeled semantics for HOπJ in Figure 3. Pure input transitions $P \xrightarrow{\overrightarrow{p}}_{\mathsf{i}} R$ describe a process P inputting the messages $\overrightarrow{p}$ to yield R. Pure output transitions $S \xrightarrow{\nu \overrightarrow{b}.\overrightarrow{p}}_{\mathsf{o}} K$ describe a process S emitting the messages $\overrightarrow{p}$ with extruded names $\overrightarrow{b}$ to yield K. Mixed input/output transitions $P \xrightarrow{\overrightarrow{p}}_{\mathsf{io}} P'$ describe a process P that emits some messages which, combined with messages $\overrightarrow{p}$, trigger an input and result in P'. Finally, silent transitions $P \xrightarrow{\tau} P'$ describe a process that does an internal communication: we define them as a notation, standing for $P \xrightarrow{()}_{\mathsf{io}} P'$, an input/output transition with no message received.

A derivation tree of a transition is best described following a path from the root of the source process to the receiver. Upon encountering a name restriction in input/output mode (rule IONU), we check that it does not prevent communication. For a parallel composition, there are two cases: either one of the process is not involved in the communication (rule IOPAR), or it is. In that case, one process must only contain outputs, while the other either contains both an input and outputs (rule IOBOTHIO), or only an input (rule IOBOTHIN). In both cases, the messages and extruded names are collected in the output-only premise, and these messages $\overrightarrow{p}$ are added to the ones $\overrightarrow{q}$ received up to this point in the other premise. The name restrictions are restored at this point, as it is the parallel composition closest to both the newly collected messages and the input.

On the output-only side, messages are collected by rule OUTOUT; name restrictions are either added to the label for extrusion with rule OUTEXTR, or

[4] As bound variables in a join pattern are distinct, θ_1 and θ_2 have disjoint domains.

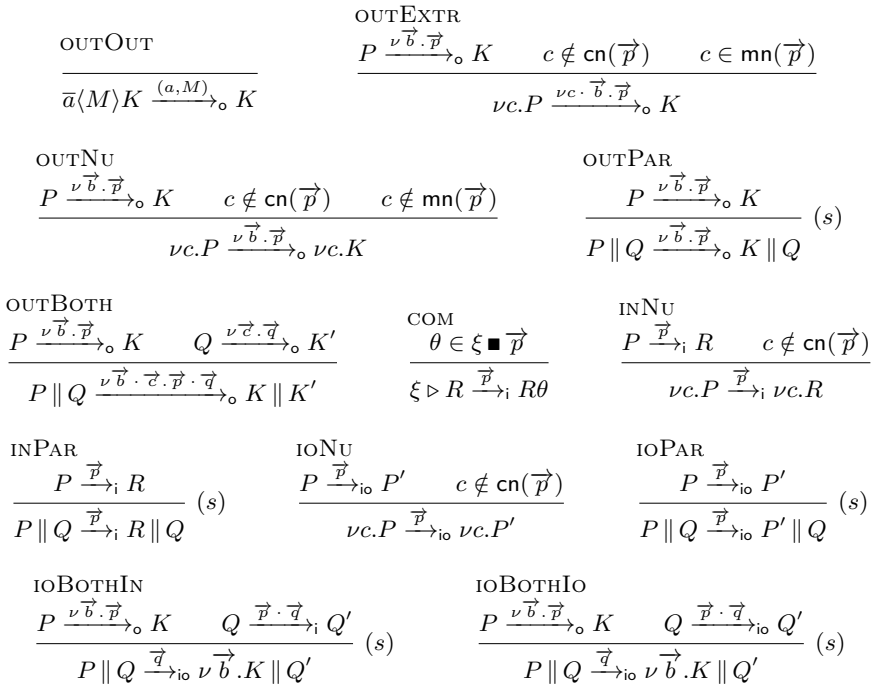

Fig. 3. Lazy Semantics for HOπJ

left in place in rule outNu; parallel composition either explores both processes, merging their results (rule outBoth), or only one of them (rule outPar). Note that output rules always collect at least one message.

On the input-only side, name restrictions are left in place (rule inNu), parallel processes only result in the exploration of one of them as there is a single input (rule inPar), and the actual communication occurs in rule com, where the collected messages are matched against the input pattern. Input/output rules and input-only rules are very similar and could be merged. We keep them separate as it simplifies the proof that the zipper and lazy semantics coincide.

3.2 Leaf-First Zipper semantics

Informal description. The HOπJ zipper semantics collects all the messages by alternating between the out and par modes, before looking for the input. During the search, we record the positions where the extruded names should be restricted using contexts stored in a *global stack*. We also use a *local stack* to record checkpoints when collecting several messages in a row. We illustrate how the search proceeds using the process in Figure 4, where we represent only the communicating outputs and input, and number the parallel compositions be-

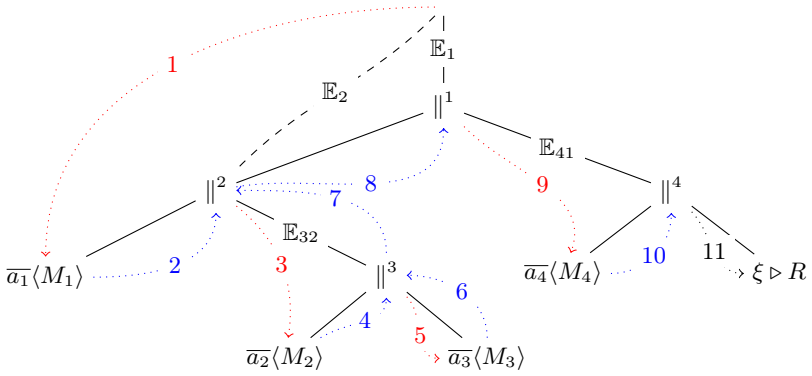

Fig. 4. Example of Process Exploration

tween them. The dotted lines show the exploration of the term within the out mode in red and the par mode in blue.

We start by looking for an output, say M_1 (step 1), and then switch to the par mode to look for $\|^2$, computing K_1 and extruded names $\overrightarrow{b_1}$ while doing so (step 2). We create a checkpoint $(\overrightarrow{b_1}, M_1, \mathbb{E}_2, K_1)$ on the local stack, where $\mathbb{E}_2$ represents the position of $\|^2$, and then look for the next message M_2. After finding it (step 3), we go back to $\|^3$ (step 4), computing K_2 and $\overrightarrow{b_2}$. We create a checkpoint $(\overrightarrow{b_2}, M_2, \mathbb{E}_{32}, K_2)$ on the local stack, where $\mathbb{E}_{32}$ represents the position of $\|^3$ relative to $\|^2$, before searching for M_3 (step 5). We then go back to $\|^3$, computing K_3 and $\overrightarrow{b_3}$ (step 6).

To continue exploring, we pop the local stack to get back $\mathbb{E}_{32}$. We also combine the continuations $K_{23} = K_2 \,\|\, K_3$ and the names $\overrightarrow{b_{23}} = \overrightarrow{b_2} \cdot \overrightarrow{b_3}$. We go through $\mathbb{E}_{32}$ (step 7) to get back to $\|^2$. We pop the local stack again to get $\mathbb{E}_2$ and create $K_{123} = K_1 \,\|\, K_{23}$ and $\overrightarrow{b_{123}} = \overrightarrow{b_1} \cdot \overrightarrow{b_{23}}$. The search goes through $\mathbb{E}_2$ to find $\|^1$ (step 8). Because $\|^1$ separates M_1, M_2, and M_3 from the input, it is where $\overrightarrow{b_{123}}$ should be restricted: we create a checkpoint $(\overrightarrow{b_{123}}, (M_1, M_2, M_3), \mathbb{E}_1, K_{123})$ on the global stack, where $\mathbb{E}_1$ records the position of $\|^1$.

We search for M_4 (step 9), then go back to $\|^4$, building K_4 and $\overrightarrow{b_4}$ (step 10). Because $\|^4$ is where we restrict $\overrightarrow{b_4}$, we add a checkpoint $(\overrightarrow{b_4}, M_4, \mathbb{E}_{41}, K_4)$ on the global stack, where $\mathbb{E}_{41}$ records the position of $\|^4$ relative to $\|^1$. We then look for the input R and compute the result $\mathbb{E}_1[\![\nu\overrightarrow{b_{123}}.K_{123} \,\|^1\, \mathbb{E}_{41}[\![\nu\overrightarrow{b_4}.K_4 \,\|^4\, R']\!]]\!]$ for some R' by repeatedly popping the global stack (step 11).

This derivation collects the messages in the order M_1, M_2, M_3, M_4. It is possible to collect them in the order M_1, M_3, M_2, M_4, or M_2, M_3, M_1, M_4, or M_3, M_2, M_1, M_4. However, the design of the zipper rules forbids the remaining orders. Roughly, when considering the topmost parallel composition (here $\|^1$),

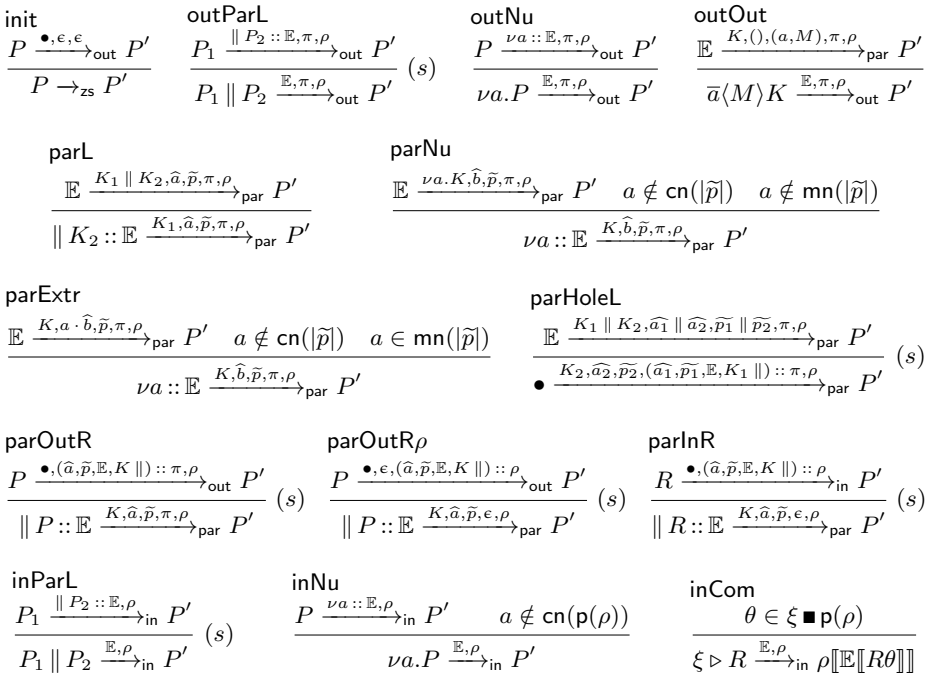

Fig. 5. Zipper semantics for HOπJ

the search starts on the other side of the input, so starting with M_4 is not possible. Besides, we go from one message to the nearest one, so starting from M_2, it is not possible to go to M_1 and then back to M_3.

Formal definitions. The zipper semantics of HOπJ is defined in Figure 5. To make it reversible, we need more complex data structures than just sequences to collect extruded names and messages; we explain why in Section 3.3. We define a structure of names $\widehat{a}$ and of pairs $\widetilde{p}$ as follows:

$$\widehat{a}, \widehat{b} ::= () \mid a \cdot \widehat{b} \mid \widehat{a} \parallel \widehat{b} \qquad \widetilde{p}, \widetilde{q} ::= (a, P) \mid \widetilde{p} \parallel \widetilde{q}$$

We abbreviate a structure $a_1 \cdot \ldots a_n \cdot ()$ as $(a_1, \ldots, a_n)$ or $\overrightarrow{a}$. These structures reflect the process from which they are generated. For example, the structures corresponding to $\nu a.(\overline{a_1}\langle P_1\rangle \parallel \nu b.\nu c.\overline{a_2}\langle P_2\rangle)$ are $a \cdot (() \parallel (b, c))$ and $(a_1, P_1) \parallel (a_2, P_2)$. We define a flattening $|\cdot|$ from structures back to sequences as follows:

$$|()| \triangleq () \qquad |\widehat{a} \parallel \widehat{b}| \triangleq |\widehat{a}| \cdot |\widehat{b}| \qquad |a \cdot \widehat{b}| \triangleq a \cdot |\widehat{b}|$$
$$|(a, P)| \triangleq (a, P) \qquad |\widetilde{a} \parallel \widetilde{b}| \triangleq |\widetilde{a}| \cdot |\widetilde{b}|$$

The syntax of stacks is $\gamma ::= \epsilon \mid (\widehat{a}, \widetilde{p}, \mathbb{E}, \mathfrak{F}) :: \gamma$ where $\mathfrak{F}$ is either $\parallel K$ or $K \parallel$ for some K. We let ρ range over global stacks and π over local ones. We extend

plugging to global stacks, and define $\mathsf{p}(\rho)$ as the pairs occurring in ρ.

$$\epsilon[\![P]\!] \triangleq P \qquad ((\widehat{a},\widetilde{p},\mathbb{E},\mathfrak{F})::\rho)[\![P]\!] \triangleq \rho[\![\mathbb{E}[\![\nu|\widehat{a}|.\mathfrak{F}[\![P]\!]]\!]]$$
$$\mathsf{p}(\epsilon) \triangleq () \qquad \mathsf{p}((\widehat{a},\widetilde{p},\mathbb{E},\mathfrak{F})::\rho) \triangleq |\widetilde{p}|\cdot\mathsf{p}(\rho)$$

The rules for the output $P \xrightarrow{\mathbb{E},\pi,\rho}_{\mathsf{out}} P'$ and input modes $R \xrightarrow{\mathbb{G},\rho}_{\mathsf{in}} P'$ are roughly the same as in HOπP. The stacks π, ρ are passed along without being modified; ρ is used in the axiom inCom to compute the final result. The transition $\mathbb{E} \xrightarrow{K,\widehat{a},\widetilde{p},\pi,\rho}_{\mathsf{par}} P'$ is looking for the nearest parallel composition which separates the last found output to the rest of the redex, building K and $\widehat{a}$. The rules, parNu, and parExtr are the same as in HOπP, only adapted to the multiple messages $\widetilde{p}$.

When the context is $\| P :: \mathbb{E}$, i.e., the term is $\mathbb{E}[\![K \| P]\!]$, we have several possibilities. If P is not part of the communication, it is added to the continuation (rule parL). If P contains at least one output but not the input, we apply the rule parOutR to look for the next output in P and create a checkpoint on the local stack. That checkpoint is popped when we apply parHoleL: after finding all the messages in P, we restore $\mathbb{E}$ to continue the search in that context.

The next possibility is that P contains at least one output and the input: we apply parOutRρ to look for the next output in P and create a checkpoint on the global stack to remember that $\mathbb{E}$ is where the names $\widehat{a}$ should be restricted. The rule parOutRρ applies only when the local stack is empty, meaning that we are done collecting messages on that side of the parallel composition. The final case is when P contains only the input: we switch to the in mode and creates the last checkpoint on the global stack. As in the previous case, the local stack must be empty. We illustrate these rules on the example of Figure 4.

Back to the example. We start with empty local and global stacks. After reaching $\overline{a_1}\langle M_1\rangle$, we go back to $\|^2$ to search for the next messages: at that point, the derivation looks like the rule on the left below. The process P_{23} contains $\overline{a_2}\langle M_2\rangle$ and $\overline{a_3}\langle M_3\rangle$: we use similar naming convention for processes henceforth. Let $\pi_1 \triangleq (\widehat{b_1},(a_1,M_1),\mathbb{E}_2,K_1 \|^2)::\epsilon$. We look for M_2 and then go back to $\|^3$, where we apply parOutR again to search for M_3 (right rule).

$$\text{parOutR}\ \frac{P_{23} \xrightarrow{\bullet,(\widehat{b_1},(a_1,M_1),\mathbb{E}_2,K_1\|^2)::\epsilon,\epsilon}_{\mathsf{out}} P'}{\|^2 P_{23} :: \mathbb{E}_2 \xrightarrow{K_1,\widehat{b_1},(a_1,M_1),\epsilon,\epsilon}_{\mathsf{par}} P'} \qquad \text{parOutR}\ \frac{P_3 \xrightarrow{\bullet,(\widehat{b_2},(a_2,M_2),\mathbb{E}_{32},K_2\|^3)::\pi_1,\epsilon}_{\mathsf{out}} P'}{\|^3 P_3 :: \mathbb{E}_{32} \xrightarrow{K_2,\widehat{b_2},(a_2,M_2),\pi_1,\epsilon}_{\mathsf{par}} P'}$$

After finding M_3, we go back to $\|^3$ again and pop the local stack to restore $\mathbb{E}_{32}$ and continue the exploration of the term.

$$\text{parHoleL}\ \frac{\mathbb{E}_{32} \xrightarrow{K_2\|^3 K_3,\widehat{b_2}\,\|\,\widehat{b_3},(a_2,M_2)\,\|(a_3,M_3),\pi_1,\epsilon}_{\mathsf{par}} P'}{\bullet \xrightarrow{K_3,\widehat{b_3},(a_3,M_3),(\widehat{b_2},(a_2,M_2),\mathbb{E}_{32},K_2\|^3)::\pi_1,\epsilon}_{\mathsf{par}} P'}$$

We go through $\mathbb{E}_{32}$ to reach $\|^2$, and apply parHoleL again to restore $\mathbb{E}_2$.

$$\textsf{parHoleL}\ \frac{\mathbb{E}_2 \xrightarrow{K_1 \|^2 K_{23}, \widehat{b_1} \| \widehat{b_{23}}, (a_1, M_1) \| \widetilde{p_{23}}, \epsilon, \epsilon}_{\textsf{par}} P'}{\bullet \xrightarrow{K_{23}, \widehat{b_{23}}, \widetilde{p_{23}}, (\widehat{b_1}, (a_1, M_1), \mathbb{E}_2, K_1 \|^2) :: \epsilon, \epsilon}_{\textsf{par}} P'}$$

The next step is when we get to $\|^1$, where we create a checkpoint on the global stack (left rule below). Let $\rho_1 \triangleq (\widehat{b_{123}}, \widetilde{p_{123}}, \mathbb{E}_1, K_{123} \|^1) :: \epsilon$. We then look for M_4, and go back to $\|^4$, where we create the final checkpoint (right rule).

$$\textsf{parOutR}\rho\ \frac{P_{4R} \xrightarrow{\bullet, \epsilon, (\widehat{b_{123}}, \widetilde{p_{123}}, \mathbb{E}_1, K_{123} \|^1) :: \epsilon}_{\textsf{out}} P'}{\|^1 P_{4R} :: \mathbb{E}_1 \xrightarrow{K_{123}, \widehat{b_{123}}, \widetilde{p_{123}}, \epsilon, \epsilon}_{\textsf{par}} P'} \qquad \textsf{parInR}\ \frac{P_R \xrightarrow{\bullet, (\widehat{b_4}, (a_4, M_4), \mathbb{E}_{41}, K_4 \|^4) :: \rho_1}_{\textsf{in}} P'}{\|^4 P_R :: \mathbb{E}_{41} \xrightarrow{K_4, \widehat{b_4}, (a_4, M_4), \epsilon, \rho_1}_{\textsf{par}} P'}$$

Let $\rho_2 \triangleq (\widehat{b_4}, (a_4, M_4), \mathbb{E}_{41}, K_4 \|^4) :: \rho_1$. Once we find the input, we can compute the final result $P' = \rho_2[\![R']\!]$ for some R', and one can check that $\rho_2[\![R']\!] = \mathbb{E}_1[\![\nu\overrightarrow{b_{123}}.K_{123} \|^1 \mathbb{E}_{41}[\![\nu\overrightarrow{b_4}.K_4 \|^4 R']\!]]\!]$, as wished.

3.3 Properties of the Zipper Semantics

Correspondence results. The correspondence proofs between the zipper and lazy semantics follow the same strategy as in HOπP: using the information stored in the label, we reconstruct the source and target terms of a zipper transition. For example, for each checkpoint $(\widehat{a}, \widetilde{p}, \mathbb{E}, K \|) :: \gamma$ of a local or global stack and for any $S \xrightarrow{\nu|\widehat{a}|.|\widetilde{p}|}_{\mathsf{o}} K$, we know that the source process at this point is of the form $\mathbb{E}[\![S \| R]\!]$ and the target process is $\mathbb{E}[\![\nu|\widehat{a}|.K \| R']\!]$, where R and R' are computed recursively from γ.

Theorem 3. *If $P \rightarrow_{\mathsf{zs}} P'$, then $P \xrightarrow{\tau} P'$.*

For the reverse implication, for any transition $P \xrightarrow{\tau} P'$, we can decompose P and P' with a global stack ρ and sending processes. We then reconstruct the zipper derivation starting with the input mode, then alternating between the par and out modes for each output, reasoning by induction on the size of ρ.

Theorem 4. *If $P \xrightarrow{\tau} P'$, then $P \rightarrow_{\mathsf{zs}} P'$.*

Derivation into an NDAM. The rules of the zipper semantics have been designed to be machine constructive and reversible. In particular, using complex data structures for extruded names and messages ensures that the rules parHoleL and parHoleR are reversible: a structure $\widehat{a_1} \| \widehat{a_2}$ uniquely decomposes into $\widehat{a_1}$ and $\widehat{a_2}$, while a sequence $\overrightarrow{a}$ can be split in two in many ways.

For termination, we consider a lexicographic ordering with three sizes on transitions. The first one is the number of pairs p in the label of the transitions,

including the stacks. It is constant on all rules except outOut, where it is strictly increasing between the conclusion and the premise, but bounded by the total number of outputs in the process. The second one is constantly equal to 0 for the out and in modes, and equal to the sum of the sizes of all contexts occurring in the transition (including in stacks) in par mode, where the size of a context is its number of frames plus one. It is designed to be strictly decreasing between the conclusion and the premise on all the rules defining the par mode. In particular for parHoleL and parHoleR, the size of the conclusion is equal to the size of the premise plus one. The last ordering is the subterm ordering for terms at the source of the transition, already used in HOπP. It is strictly decreasing on the rules of the out and in modes, except outOut which has already been covered. As a result we can derive a sound, complete, and terminating NDAM for HOπJ.

4 Related Work and Conclusion

Related work. Section 2.2 discusses the root-first zipper semantics [2] for HOπ with eager scope extrusion. As far as we know, its derived NDAM is the only machine proposed for a variant of HOπ. We discuss next the other machines for higher-order calculi and for calculi with join patterns that we are aware of.

The notion of join patterns comes from the join calculus [11], whose initial semantics takes the form of a *chemical abstract machine*, where messages can be seen as as molecules free to move around in a solution until all the messages required to trigger a join-pattern are present. The machines for the languages with passivation Kell [1] and M [13] (which also features join patterns) also rely on this chemical design, typically implemented as message queues. A major drawback of this approach is that it ignores the syntactic structure of processes, making proofs on such structure quite complex. Our approach, on the other hand, preserves the syntactic structure as much as possible, which allows us to prove an operational correspondence between the lazy and zipper semantics (and therefore the NDAM), and not only an observation-based equivalence [1]. This comes with a cost: as the syntactic structure of the term is preserved, the extruded names may enclose unrelated processes even with lazy scope extrusion. Structural congruence would be able to move these unrelated terms around, at the cost of the operational correspondence proof.

A strength and limitation of the join calculus is that it uses the same construct for restriction, reception, and replication. More precisely, definitions $\xi_1 \triangleright R_1 \wedge \ldots \wedge \xi_n \triangleright R_n$ are replicated inputs which restrict the names $\bigcup_i \mathsf{cn}(\xi_i)$. As a result, the receivers for a given name are fixed. This reduces the expressive power of the calculus [20], but it allows for efficient implementations as long as join patterns are linear [10]. HOπJ is more permissive, as name restriction and input are separate constructs, and although our approach is not as efficient as the join calculus, our use of zipper semantics lets us derive an abstract machine [3].

Conclusion. Zipper semantics decomposes a term into an evaluation context and a redex. Biernacka et al. [2, 3] define zipper semantics where the search for

the redex starts by the operator at its root; instead, we look for an operator at its leaves. This style is well-suited to express the semantics of calculi with lazy scope extrusion, as knowing the message first lets us limit scope extrusion to the sent names. In particular, we are able to express the semantics of a calculus with join patterns, where the scope of extruded names is not the same for all the messages. As a result, we get for free a sound and complete abstract machine for this language. It shows the expressiveness of leaf-first zipper semantics.

Leaf-first zipper semantics is well-suited when the result of reduction depends on information at the leaves of the redex, like message output w.r.t. scope extrusion. Other examples are distributed calculi which restrict communication through locality boundaries [4–6, 15, 26]. In such cases, we need to know the message contents to determine whether it can be sent outside of a locality, so a leaf-first semantics should be more appropriate.

References

1. Bidinger, P., Schmitt, A., Stefani, J.: An abstract machine for the kell calculus. In: Steffen, M., Zavattaro, G. (eds.) Formal Methods for Open Object-Based Distributed Systems, 7th IFIP WG 6.1 International Conference, FMOODS 2005, Athens, Greece, June 15-17, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3535, pp. 31–46. Springer (2005)
2. Biernacka, M., Biernacki, D., Lenglet, S., Schmitt, A.: Non-deterministic abstract machines. Tech. Rep. 9475, Inria (2022), available at `https://hal.inria.fr/hal-03545768`
3. Biernacka, M., Biernacki, D., Lenglet, S., Schmitt, A.: Non-deterministic abstract machines. In: Klin, B., Lasota, S., Muscholl, A. (eds.) 33rd International Conference on Concurrency Theory, CONCUR 2022, September 12-16, 2022, Warsaw, Poland. LIPIcs, vol. 243, pp. 7:1–7:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2022)
4. Bugliesi, M., Crafa, S., Merro, M., Sassone, V.: Communication and mobility control in boxed ambients. Inf. Comput. **202**(1), 39–86 (2005)
5. Cardelli, L., Gordon, A.D.: Mobile ambients. In: Nivat, M. (ed.) Foundations of Software Science and Computation Structure, First International Conference, FoSSaCS'98, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS'98, Lisbon, Portugal, March 28 - April 4, 1998, Proceedings. Lecture Notes in Computer Science, vol. 1378, pp. 140–155. Springer (1998)
6. Castagna, G., Vitek, J., Nardelli, F.Z.: The seal calculus. Inf. Comput. **201**(1), 1–54 (2005)
7. Danvy, O.: From reduction-based to reduction-free normalization. In: Koopman, P.W.M., Plasmeijer, R., Swierstra, S.D. (eds.) Advanced Functional Programming, 6th International School, AFP 2008, Heijen, The Netherlands, May 2008, Revised Lectures. Lecture Notes in Computer Science, vol. 5832, pp. 66–164. Springer (2008)
8. Felleisen, M., Hieb, R.: The revised report on the syntactic theories of sequential control and state. Theor. Comput. Sci. **103**(2), 235–271 (1992)
9. Fessant, F.L.: JoCaml: conception et implémentation d'un langage à agents mobiles. Ph.D. thesis, École polytechnique (2001)

10. Fessant, F.L., Maranget, L.: Compiling join-patterns. In: Nestmann, U., Pierce, B.C. (eds.) 3rd International Workshop on High-Level Concurrent Languages, HLCL 1998, Satellite Workshop of CONCUR 1998, Nice, France, September 12, 1998. pp. 205–224. No. 16(3) in Electronic Notes in Theoretical Computer Science, Elsevier (1998). https://doi.org/10.1016/S1571-0661(04)00143-4, https://www.sciencedirect.com/journal/electronic-notes-in-theoretical-computer-science/vol/16/issue/3
11. Fournet, C., Gonthier, G.: The reflexive CHAM and the join-calculus. In: Boehm, H., Jr., G.L.S. (eds.) Conference Record of POPL'96: The 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Papers Presented at the Symposium, St. Petersburg Beach, Florida, USA, January 21-24, 1996. pp. 372–385. ACM Press (1996)
12. Gardner, P., Laneve, C., Wischik, L.: The fusion machine. In: Brim, L., Jancar, P., Kretínský, M., Kucera, A. (eds.) CONCUR 2002 - Concurrency Theory, 13th International Conference, Brno, Czech Republic, August 20-23, 2002, Proceedings. Lecture Notes in Computer Science, vol. 2421, pp. 418–433. Springer (2002)
13. Germain, F., Lacoste, M., Stefani, J.: An abstract machine for a higher-order distributed process calculus. Electron. Notes Theor. Comput. Sci. **66**(3), 145–169 (2002)
14. Giannini, P., Sangiorgi, D., Valente, A.: Safe ambients: Abstract machine and distributed implementation. Sci. Comput. Program. **59**(3), 209–249 (2006)
15. Godskesen, J.C., Hildebrandt, T.T., Sassone, V.: A calculus of mobile resources. In: Brim, L., Jancar, P., Kretínský, M., Kucera, A. (eds.) CONCUR 2002 - Concurrency Theory, 13th International Conference, Brno, Czech Republic, August 20-23, 2002, Proceedings. Lecture Notes in Computer Science, vol. 2421, pp. 272–287. Springer (2002)
16. Hirschkoff, D., Pous, D., Sangiorgi, D.: An efficient abstract machine for safe ambients. J. Log. Algebraic Methods Program. **71**(2), 114–149 (2007)
17. Lenglet, S., Schmitt, A.: Leaf-first zipper semantics (2024), available at https://inria.hal.science/hal-04537440
18. Lenglet, S., Schmitt, A., Stefani, J.: Characterizing contextual equivalence in calculi with passivation. Inf. Comput. **209**(11), 1390–1433 (2011)
19. Lopes, L.M.B., Silva, F.M.A., Vasconcelos, V.T.: A virtual machine for a process calculus. In: Nadathur, G. (ed.) Principles and Practice of Declarative Programming, International Conference PPDP'99, Paris, France, September 29 - October 1, 1999, Proceedings. Lecture Notes in Computer Science, vol. 1702, pp. 244–260. Springer (1999)
20. Nestmann, U.: On the expressive power of joint input. In: Castellani, I., Palamidessi, C. (eds.) Fifth International Workshop on Expressiveness in Concurrency, EXPRESS 1998, Satellite Workshop of CONCUR 1998, Nice, France, September 7, 1998. pp. 145–152. No. 16(2) in Electronic Notes in Theoretical Computer Science, Elsevier (1998). https://doi.org/10.1016/S1571-0661(04)00123-9, https://doi.org/10.1016/S1571-0661(04)00123-9
21. Phillips, A., Cardelli, L.: A correct abstract machine for the stochastic pi-calculus. In: Concurrent Models in Molecular Biology (2004)
22. Phillips, A., Yoshida, N., Eisenbach, S.: A distributed abstract machine for boxed ambient calculi. In: Schmidt, D.A. (ed.) Programming Languages and Systems, 13th European Symposium on Programming, ESOP 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, March 29 - April 2, 2004, Proceedings. Lecture Notes in Computer Science, vol. 2986, pp. 155–170. Springer (2004)

23. Plotkin, G.D.: A structural approach to operational semantics. Tech. Rep. FN-19, DAIMI, Department of Computer Science, Aarhus University, Aarhus, Denmark (Sep 1981)
24. Sangiorgi, D.: Bisimulation in higher-order process calculi. In: Olderog, E. (ed.) Programming Concepts, Methods and Calculi, Proceedings of the IFIP TC2/WG2.1/WG2.2/WG2.3 Working Conference on Programming Concepts, Methods and Calculi (PROCOMET '94) San Miniato, Italy, 6-10 June, 1994. IFIP Transactions, vol. A-56, pp. 207–224. North-Holland (1994)
25. Sangiorgi, D., Walker, D.: The Pi-Calculus - a theory of mobile processes. Cambridge University Press (2001)
26. Schmitt, A., Stefani, J.: The kell calculus: A family of higher-order distributed process calculi. In: Priami, C., Quaglia, P. (eds.) Global Computing, IST/FET International Workshop, GC 2004, Rovereto, Italy, March 9-12, 2004, Revised Selected Papers. Lecture Notes in Computer Science, vol. 3267, pp. 146–178. Springer (2004)
27. Turner, D.: The Polymorphic Pi-calculus:Theory and Implementation. Ph.D. thesis, University of Edinburgh (1995)