From Complementary to Zipper Semantics

Sergueï Lenglet, Camille Noûs, Alan Schmitt

From Complementary to Zipper Semantics

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

[1] Université Sorbonne Paris Nord, Villetaneuse
[2] Laboratoire Cogitamus
[3] INRIA

Abstract. We present two serialized semantics format for process calculi based on labeled transition systems. *Complementary semantics* are useful to prove the congruence of bisimilarity. *Zipper semantics* are the first step toward the automatic derivation of an abstract machine. Both semantics are based on the same restrictions: inductive rules have at most one premise and some contextual information is kept in the label.

Keywords: Process calculus · Semantics · Labeled transition system · Bisimilarity · Congruence · Abstract machine.

1 Introduction

The semantics of process calculi can be defined using a *labeled transition system* (LTS), where a process performs a little bit of computation (it is a *small-step* semantics) while indicating as a label the interaction it is doing. Such a description simplifies semantics with interactions that are not local, such as the sending and receiving of a message between (syntactically) distant processes. Consider for instance the process $(\overline{a} \parallel P) \parallel (Q \parallel a.R)$, where $\overline{a}$ is the sending of (an empty) message on a, and $a.R$ is the receiving of an empty message before continuing as R. The $\parallel$ operator represents the parallel composition. This process may evolve to $P \parallel (Q \parallel R)$ after the communication on a. This can be witnessed by the fact that $\overline{a} \parallel P$ can emit on a to evolve to P, and $Q \parallel a.R$ can receive on a to evolve to $Q \parallel R$. The derivation as an LTS of this computation would typically be shown as follows, where τ denotes an internal step.

$$\frac{\dfrac{\cdots}{\overline{a} \parallel P \xrightarrow{\overline{a}} P} \qquad \dfrac{\cdots}{Q \parallel a.R \xrightarrow{a} Q \parallel R}}{(\overline{a} \parallel P) \parallel (Q \parallel a.R) \xrightarrow{\tau} P \parallel (Q \parallel R)}$$

This approach has two drawbacks. First, the communication rule has multiple inductive hypotheses, which can raise some transitivity issues when proving some properties of the calculus. This was first described in our work on behavioural equivalences in calculi with passivation [11–13] and is detailed in Section 2. Second, the decomposition of a term as a context around a redex is left implicit in the structure of the derivation, while it would be useful to have it as an explicit

label, for instance as a first step towards an abstract machine for the calculus as we more recently showed [2, 10]. This is described in Section 3. Although these questions are quite different, our proposal to address them is very similar: a labeled transition system where each rule has at most one inductive premise, and where contexts or parts of contexts are explicitly given in the labels.

2 Complementary Semantics

To be preserved by context is a desirable property of behavioural equivalences: it means that one can prove that two terms are equivalent by decomposing them into smaller equivalent subterms. Such a property turned out to be difficult to prove for some higher-order process calculi [11, 12]. To illustrate this, we use a simple calculus called HOcore [9], for which we define a context bisimilarity. We show what the problem is with its congruence proof, and how it can be solved using complementary semantics.

2.1 HOcore

We recall the syntax and semantics of HOcore, which we use to present complementary and zipper semantics (Section 3). HOcore is a minimal higher-order process calculus with no name restriction. We let a, b range over *channel names*, X, Y over *process variables*, and we define the syntax of processes as follows.

$$P, Q, R ::= X \mid \mathbf{0} \mid P \parallel Q \mid a(X).P \mid \overline{a}\langle P \rangle$$

The process $\mathbf{0}$ is the inactive process, $P \parallel Q$ runs P and Q in parallel, and a communication may happen between an input $a(X).P$ and an output $\overline{a}\langle Q \rangle$ that run in parallel. The communication is asynchronous because a message output does not have a continuation [16]. The notions of bound variables (the variable X in $a(X).P$) and free variables are defined as usual. We adopt the *Barendregt convention* that bound variables are always kept distinct from free variable through α-conversion. We write $P\{R/X\}$ for the usual capture-avoiding substitution of X by R in P, and $\widetilde{P}$ for a tuple of processes.

In spite of its minimal number of constructors, HOcore is Turing-complete [9].

We present the semantics of HOcore as a labeled transition system (LTS). A process may evolve towards a process (internal actions $P \xrightarrow{\tau} P'$), an *abstraction* (message input $P \xrightarrow{a} F = (X)Q$), or a *concretion* (message output $P \xrightarrow{\overline{a}} C = \langle R \rangle Q$). The transition $P \xrightarrow{a} (X)Q$ means that P may receive a process R on a to continue as $Q\{R/X\}$. The transition $P \xrightarrow{\overline{a}} \langle R \rangle Q$ means that P may send the process R on a and continue as Q. Note that Q is needed even though our calculus is asynchronous, as it not only contains the direct continuation of the message (which is absent in an asynchronous calculus), but also processes in parallel of the message sending.

Let *agents*, noted A, be the set of processes, abstractions, and concretions. We extend the parallel composition to all agents as follows, relying on the Barendregt

$$\textsf{Abstr}\quad a(X).P \xrightarrow{a} (X)P \qquad \textsf{Concr}\quad \overline{a}\langle Q\rangle \xrightarrow{\overline{a}} \langle Q\rangle \mathbf{0}$$

$$\textsf{Par}\quad \frac{P \xrightarrow{l} A}{P \parallel Q \xrightarrow{l} A \parallel Q} \qquad \textsf{HO}\quad \frac{P \xrightarrow{a} F \qquad Q \xrightarrow{\overline{a}} C}{P \parallel Q \xrightarrow{\tau} F \bullet C}$$

Fig. 1. Labeled Transition System for HOcore

convention to avoid capture.

$$(X)Q \parallel P \triangleq (X)(Q \parallel P) \qquad \langle Q\rangle R \parallel P \triangleq \langle Q\rangle(R \parallel P)$$
$$P \parallel (X)Q \triangleq (X)(P \parallel Q) \qquad P \parallel \langle Q\rangle R \triangleq \langle Q\rangle(P \parallel R)$$

A higher-order communication takes place when a concretion interacts with an abstraction. We define a pseudo-application operator $\bullet$ between an abstraction $(X)P$ and a concretion $\langle R\rangle Q$ as follows.

$$(X)P \bullet \langle R\rangle Q \triangleq P\{R/X\} \parallel Q$$

The rules of the LTS are given in Figure 1, where we omit the symmetric rules for Par and HO.

2.2 Context Bisimilarity and Howe's Method

Sangiorgi [15] defines *context* bisimilarity as a behavioural equivalence for higher-order process calculi. When two tested processes P and Q perform a partial action like receiving a message on a, a context may interact with them by sending a message on that channel. Context bisimilarity therefore requires the processes to remain bisimilar after communicating with any such recipient, represented by the abstraction F in the second clause of Definition 1. The principle is symmetric for outputting processes.

Definition 1. *Strong context bisimilarity $\sim$ is the largest symmetric relation on closed processes $\mathcal{R}$ such that $P\ \mathcal{R}\ Q$ implies:*

- *for all $P \xrightarrow{\tau} P'$, there exists Q' such that $Q \xrightarrow{\tau} Q'$ and $P'\ \mathcal{R}\ Q'$;*
- *for all $P \xrightarrow{a} F$, for all C, there exists F' such that $Q \xrightarrow{a} F'$ and $F \bullet C\ \mathcal{R}\ F' \bullet C$;*
- *for all $P \xrightarrow{\overline{a}} C$, for all F, there exists C' such that $Q \xrightarrow{\overline{a}} C'$ and $F \bullet C\ \mathcal{R}\ F \bullet C'$.*

Sangiorgi's proof technique to show that context bisimilarity is a congruence unfortunately does not scale to more complex calculi [11]. It explains why we turn to Howe's method [1,6,7], a systematic proof technique to show that a simulation $\mathcal{R}$ is a congruence. The method can be divided in three steps: first, prove

some basic properties on the *Howe's closure* $\mathcal{R}^\bullet$ of the relation. By construction, $\mathcal{R}^\bullet$ contains $\mathcal{R}$ and is a congruence. Second, prove a simulation-like property for $\mathcal{R}^\bullet$, and finally prove that $\mathcal{R}$ and $\mathcal{R}^\bullet$ coincide on closed processes. Since $\mathcal{R}^\bullet$ is a congruence, conclude that $\mathcal{R}$ is a congruence.

The definition of the Howe's closure relies on the open extension of $\mathcal{R}$, noted $\mathcal{R}^\circ$: it extends the definition of the relation $\mathcal{R}$ to *open processes*, that are processes with free process variables.

Definition 2. *Let P and Q be two open processes. We have $P\ \mathcal{R}^\circ\ Q$ iff $P\sigma\ \mathcal{R}\ Q\sigma$ for all process substitutions σ that close P and Q.*

Howe's closure is inductively defined as the smallest congruence which contains $\mathcal{R}^\circ$ and is closed under right composition with $\mathcal{R}^\circ$.

Definition 3. *Howe's closure $\mathcal{R}^\bullet$ of a relation $\mathcal{R}$ is the smallest relation verifying:*

- $\mathcal{R}^\circ \subseteq \mathcal{R}^\bullet$*;*
- $\mathcal{R}^\bullet\mathcal{R}^\circ \subseteq \mathcal{R}^\bullet$*;*
- *for all operators op of the language, if $\widetilde{P}\ \mathcal{R}^\bullet\ \widetilde{Q}$, then $op(\widetilde{P})\ \mathcal{R}^\bullet\ op(\widetilde{Q})$.*

By definition, $\mathcal{R}^\bullet$ is a congruence. The composition with $\mathcal{R}^\circ$ allows for some transitivity and gives some additional properties to the relation.

In our case, we want to prove that a bisimilarity $\mathcal{B}$ is a congruence. By definition, we have $\mathcal{B} \subseteq \mathcal{B}^\circ \subseteq \mathcal{B}^\bullet$. To have the reverse inclusion, we prove that $\mathcal{B}^\bullet$ is a bisimulation. To this end, we need the following classical properties of the Howe's closure.

Lemma 1. *Let $\mathcal{R}$ be a reflexive relation. If $P\ \mathcal{R}^\bullet\ Q$ and $R\ \mathcal{R}^\bullet\ S$, then we have $P\{R/X\}\ \mathcal{R}^\bullet\ Q\{S/X\}$.*

This lemma is typically used to establish the simulation-like result (second step of the method). We sketch the proof in order to give an idea on why the transitive item $\mathcal{R}^\bullet\mathcal{R}^\circ \subseteq \mathcal{R}^\bullet$ is needed in Definition 3. The proof is by induction on the derivation of $P\ \mathcal{R}^\bullet\ Q$. Suppose we have $P\ \mathcal{R}^\circ\ Q$. Since $R\ \mathcal{R}^\bullet\ S$ and $\mathcal{R}^\bullet$ is a congruence, we have $P\{R/X\}\ \mathcal{R}^\bullet\ P\{S/X\}$. Let σ be a substitution that closes P, Q, and S except for X; by open extension definition, we have $P\{S/X\}\sigma\ \mathcal{R}\ Q\{S/X\}\sigma$, i.e., we have $P\{S/X\}\ \mathcal{R}^\circ\ Q\{S/X\}$. Finally we have $P\{R/X\}\ \mathcal{R}^\bullet\mathcal{R}^\circ\ Q\{S/X\}$, hence we have $P\{R/X\}\ \mathcal{R}^\bullet\ Q\{S/X\}$. The other cases are easy using the induction hypothesis.

We cannot prove directly that $\mathcal{B}^\bullet$ is symmetric. Instead we use the following lemma.

Lemma 2. *Let $\mathcal{R}$ be an equivalence. Then the reflexive and transitive closure $(\mathcal{R}^\bullet)^*$ of $\mathcal{R}^\bullet$ is symmetric.*

Then one proves that the restriction of $(\mathcal{B}^\bullet)^*$ to closed terms is a bisimulation. Consequently we have $\mathcal{B} \subseteq \mathcal{B}^\bullet \subseteq (\mathcal{B}^\bullet)^* \subseteq \mathcal{B}$ on closed terms, and we

conclude that $\mathcal{B}$ is a congruence.

The main difficulty lies in the proof of the simulation-like property for Howe's closure. In the following subsection, we explain why we cannot directly use Howe's method with Definition 1.

2.3 Communication Problem

Proving that a congruence is a simulation raises transitivity issues [11]. In a nutshell, the problem occurs in the communication case, where we have two inductive premises (one for the message sending and one for the message receiving). Applying the induction hypothesis twice results in the final processes being related by $\mathcal{R}\mathcal{R}$ instead of directly $\mathcal{R}$. To avoid the problem, we establish a stronger result. Given a bisimilarity $\mathcal{B}$ based on a LTS $P \xrightarrow{\lambda} A$, the simulation-like result follows the pattern below.

Let $P\ \mathcal{B}^\bullet\ Q$. If $P \xrightarrow{\lambda} A$, then for all $\lambda\ \mathcal{B}^\bullet\ \lambda'$, there exists B such that $Q \xrightarrow{\lambda'} B$ and $A\ \mathcal{B}^\bullet\ B$.

Instantiated with Definition 1, the property would be as follows.

Conjecture 1. If $P\ \mathcal{R}^\bullet\ Q$, then:

- for all $P \xrightarrow{\tau} P'$, there exists Q' such that $Q \xrightarrow{\tau} Q'$ and $P'\ \mathcal{R}^\bullet\ Q'$;
- for all $P \xrightarrow{a} F$, for all $C\ \mathcal{R}^\bullet\ C'$, there exists F' such that $Q \xrightarrow{a} F'$ and $F \bullet C\ \mathcal{R}^\bullet\ F' \bullet C'$;
- for all $P \xrightarrow{\overline{a}} C$, for all $F\ \mathcal{R}^\bullet\ F'$ there exists C' such that $Q \xrightarrow{\overline{a}} C'$ and we have $F \bullet C\ \mathcal{R}^\bullet\ F' \bullet C'$.

The proof of conjecture 1 fails with higher-order communication. The reasoning is by induction on $P\ \mathcal{R}^\bullet\ Q$. Suppose we are in the parallel case, i.e., we have $P = P_1 \parallel P_2$ and $Q = Q_1 \parallel Q_2$, with $P_1\ \mathcal{R}^\bullet\ Q_1$ and $P_2\ \mathcal{R}^\bullet\ Q_2$. Suppose that we have $P \xrightarrow{\tau} P'$, and the transition comes from rule HO: we have $P_1 \xrightarrow{a} F$, $P_2 \xrightarrow{\overline{a}} C$ and $P' = F \bullet C$. We want to find Q' such that $Q \xrightarrow{\tau} Q'$ and $P'\ \mathcal{R}^\bullet\ Q'$. We also want to use the same rule HO, hence we have to find F', C' such that $Q \xrightarrow{\tau} F' \bullet C'$. However we cannot use the input clause of the induction hypothesis with P_1, Q_1: to have a F' such that $Q_1 \xrightarrow{a} F'$, we have to find first a concretion C' such that $C\ \mathcal{R}^\bullet\ C'$. We cannot use the output clause with P_2, Q_2 either: to have a C' such that $Q_2 \xrightarrow{\overline{a}} C'$, we have to find first an abstraction F' such that $F\ \mathcal{R}^\bullet\ F'$. We cannot bypass this mutual dependency and the inductive proof of conjecture 1 fails.

To break the circularity, we propose a change in the LTS: instead of requiring the abstraction before providing a matching output, we only require the process that will do the reception (that will reduce to the abstraction). This may seem a very small change, yet it is sufficient to break the symmetry. We return to the communication problem where $P_1 \parallel P_2$ is in relation with $Q_1 \parallel Q_2$. The concretion C' from Q_2 matching the $P_2 \xrightarrow{\overline{a}} C$ step depends only on Q_1, which is known,

$$\textsf{In}\ \frac{}{a(X).P \overset{a,R}{\longmapsto} P\{R/X\}} \qquad \textsf{Out}\ \frac{Q \overset{a,R}{\longmapsto} Q'}{\overline{a}\langle R\rangle \overset{\overline{a},Q}{\longmapsto} Q'} \qquad \textsf{Par}\ \frac{P_1 \overset{\lambda}{\mapsto} P_1'}{P_1 \parallel P_2 \overset{\lambda}{\mapsto} P_1' \parallel P_2} \qquad \textsf{HO}\ \frac{P \overset{\overline{a},Q}{\longmapsto} P'}{P \parallel Q \overset{\tau}{\mapsto} P'}$$

Fig. 2. Complementary LTS for HOcore

and not on some unknown abstraction. We can then obtain the abstraction F' from Q_1 that matches the $P_1 \xrightarrow{a} F$ step.

Technically, we do not use concretions and abstractions anymore. In the LTS, when a communication between P and Q occurs, this becomes a transition from P using Q as a label. In other words, we store part of the context of the communication in the label. Higher in the derivation, the actual output from P is discovered, and we switch to dealing with the input knowing exactly the output. The proof of the bisimulation property for the candidate relation relies on this serialization of the LTS and illustrates how symmetry is broken. On the other hand, we can prove that the resulting bisimilarity coincides with context bisimilarity.

2.4 Complementary Semantics and Bisimilarity

We define a LTS $P \overset{\lambda}{\mapsto} P'$ for HOcore where processes always evolve towards other processes. We have three kinds of transitions: internal actions $P \overset{\tau}{\mapsto} P'$, message input $P \overset{a,R}{\longmapsto} P'$, and message output $P \overset{\overline{a},R}{\longmapsto} P'$. We called this LTS *complementary* because in a transition $P \overset{\overline{a},R}{\longmapsto} P'$, we put in the label what is needed to complement P for a communication to happen (here, a receiving process R). Rules of this LTS can be found in Figure 2, except for the symmetric of rules Par and HO.

Rules for internal actions $P \overset{\tau}{\mapsto} P'$ are similar to the one for the contextual LTS $P \xrightarrow{\tau} P'$, except for higher-order communication since we change the message output judgement; we detail the rule HO later. Message input $P \overset{a,R}{\longmapsto} P'$ means that process P may receive the process R as a message on a and becomes P'. In the contextual style, it means that $P \xrightarrow{a} (X)P''$ and $P' = P''\{R/X\}$ for some P''.

The main difference is in the definition of output actions. The transition $P \overset{\overline{a},R}{\longmapsto} P'$ means that P may send a message on a, R may receive on a, and the communication on a between P and R results in P'. The transition $P \overset{\overline{a},R}{\longmapsto} P'$ means that there exists F, C such that $P \xrightarrow{\overline{a}} C$, $R \xrightarrow{a} F$, and $P' = F \bullet C$.

Rules of the LTS (Figure 2) are classic except rules HO and Out. In rule HO, the premise $P \overset{\overline{a},Q}{\longmapsto} P'$ means that P and Q can communicate on a name a and the result is P', i.e., $P \parallel Q \xrightarrow{\tau} P'$ (by communication on a in the classical LTS), which is exactly what the conclusion of the rule states. Rule Out has a premise

(unlike its equivalent rule Concr) since in the conclusion we need the result Q' of the input of R on a by Q.

Example 1. We show an example of derivation for a process close to the one given in the introduction.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{a(X).R \overset{a,\mathbf{0}}{\longmapsto} R\{\mathbf{0}/X\}}\,\mathsf{In}}{Q \parallel a(X).R \overset{a,\mathbf{0}}{\longmapsto} Q \parallel R\{\mathbf{0}/X\}}\,\mathsf{Par}}{\overline{a}\langle\mathbf{0}\rangle \overset{\overline{a},Q \parallel a(X).R}{\longmapsto} Q \parallel R\{\mathbf{0}/X\}}\,\mathsf{Out}}{\overline{a}\langle\mathbf{0}\rangle \parallel P \overset{\overline{a},Q \parallel a(X).R}{\longmapsto} (Q \parallel R\{\mathbf{0}/X\}) \parallel P}\,\mathsf{Par}}{(\overline{a}\langle\mathbf{0}\rangle \parallel P) \parallel (Q \parallel a(X).R) \overset{\tau}{\longmapsto} (Q \parallel R\{\mathbf{0}/X\}) \parallel P}\,\mathsf{HO}$$

The structure of parallel compositions is not preserved between the source and target processes: the correspondence between the complementary LTS and the LTS of Figure 1 is up to *structural congruence* [11], an equivalence on processes where parallel composition is commutative, associative, and with $\mathbf{0}$ as neutral. This discrepancy between the semantics is not a problem because our goal is to define a behavioural equivalence, and reordering processes does not change the behaviour of processes in HOcore. In more expressive calculi such as calculi with passivation, we can no longer reason up to structural congruence and the definition of the complementary semantics is more complex as a result [11].

We define complementary bisimilarity as the bisimilarity associated to the complementary LTS.

Definition 4. *Strong complementary bisimilarity $\sim_m$ is the largest symmetric relation on closed processes $\mathcal{R}$ such that $P\ \mathcal{R}\ Q$ and $P \overset{\lambda}{\longmapsto} P'$ implies $Q \overset{\lambda}{\longmapsto} Q'$ with $P'\ \mathcal{R}\ Q'$.*

As in context bisimilarity, in the message output case $P \overset{\overline{a},R}{\longmapsto} P'$, the matching transition $Q \overset{\overline{a},R}{\longmapsto} Q'$ still depends on a receiving entity (here R). However, instead of considering a context which directly receives the message (an abstraction F), we consider a process R which evolves toward an abstraction. This nuance allows us to use Howe's method to prove the soundness of $\sim_m$. We extend $\sim_m^\bullet$ to labels λ: we have $\lambda \sim_m^\bullet \lambda'$ iff $\lambda = \lambda' = \tau$, or $\lambda = (\gamma, R)$, $\lambda' = (\gamma, R')$ with $R \sim_m^\bullet R'$. We prove the following simulation-like property for $\sim_m^\bullet$:

Lemma 3. *Let P, Q be closed processes. If $P \sim_m^\bullet Q$ and $P \overset{\lambda}{\longmapsto} Q$, then for all $\lambda \sim_m^\bullet \lambda'$, there exists Q' such that $Q \overset{\lambda'}{\longmapsto} Q'$ and $P' \sim_m^\bullet Q'$.*

We do not have the same problem as in Section 2.3 with higher-order communication. We remind that in this case, we have $P_1 \parallel P_2 \sim_m^\bullet Q_1 \parallel Q_2$ with $P_1 \sim_m^\bullet Q_1$, $P_2 \sim_m^\bullet Q_2$ and $P_1 \overset{\overline{a},P_2}{\longmapsto} P'$. We can apply directly the message output clause of the induction hypothesis: there exists Q' such that $Q_1 \overset{\overline{a},Q_2}{\longmapsto} Q'$ and $P' \sim_m^\bullet Q'$. We conclude that $Q_1 \parallel Q_2 \overset{\tau}{\longmapsto} Q'$ (by rule HO) with $P' \sim_m^\bullet Q'$ as wished.

Theorem 1. *Relation $\sim_m$ is a congruence.*

We refer to our paper [11] for the proof of the theorem as well as for the proof that complementary and context bisimilarities coincide.

3 Zipper Semantics

Section 2 shows that serializing the LTS makes possible the congruence proof of bisimilarities for higher-order process calculi. The same principle also helps as a first step towards abstract machines for process calculi. We present *zipper semantics*, a small-step semantics inspired by context-based reduction semantics and which can be automatically transformed into an abstract machine [2, 10]. Indeed, each rule of a zipper semantics has at most one inductive premise, and the conclusion of the rule is only modified in the case of axioms.

3.1 Context-based Reduction Semantics

The semantics of process calculi is usually presented either with a structural congruence relation which reorders terms to make redexes appear, bringing input and output processes together, or with a LTS which preserves the structure of the term [16]. Instead, we present it here as a reduction semantics with explicit contexts [3, 5], which makes it easier to come up with (or translate into) the corresponding zipper semantics, introduced in the next section.

We represent a context $\mathbb{E}$ as a list of elementary contexts called *frames* $\mathfrak{F}$. The empty list is denoted as $\bullet$ and concatenation of $\mathfrak{F}$ to $\mathbb{E}$ as $\mathfrak{F} :: \mathbb{E}$. Each frame is either of the form $\parallel P$ or of the form $P \parallel$. Because it is more convenient for the definition of the zipper semantics, we interpret contexts inside-out [4]: the head of the context is the innermost frame. The definition of plugging a process in a context $\mathbb{E}[P]$ is therefore as follows.

$$\bullet[P] \triangleq P \qquad (\parallel Q :: \mathbb{E})[P] \triangleq \mathbb{E}[P \parallel Q] \qquad (Q \parallel :: \mathbb{E})[P] \triangleq \mathbb{E}[Q \parallel P]$$

A redex is a parallel composition with an input on one side and an output on the same name on the other side, both surrounded with contexts. The general formulation of such communication sites in a program can be expressed with the following reduction semantics.

$$\mathbb{E}[\mathbb{F}[\overline{a}\langle Q\rangle] \parallel \mathbb{G}[a(X).P]] \rightarrow_{\mathsf{rs}} \mathbb{E}[\mathbb{F}[\mathbf{0}] \parallel \mathbb{G}[P\{Q/X\}]]$$
$$\mathbb{E}[\mathbb{G}[a(X).P] \parallel \mathbb{F}[\overline{a}\langle Q\rangle]] \rightarrow_{\mathsf{rs}} \mathbb{E}[\mathbb{G}[P\{Q/X\}] \parallel \mathbb{F}[\mathbf{0}]]$$

Such rules can be read declaratively: if we find a redex in a context $\mathbb{E}$ built according to the given grammar of contexts, where the redex is itself the parallel composition of the sending and receiving of messages inside contexts $\mathbb{F}$ and $\mathbb{G}$, then we can reduce. Note that the structure of parallel compositions is preserved between the process being reduced and the result of the reduction: the contexts $\mathbb{E}$, $\mathbb{F}$, and $\mathbb{G}$ are left unchanged.

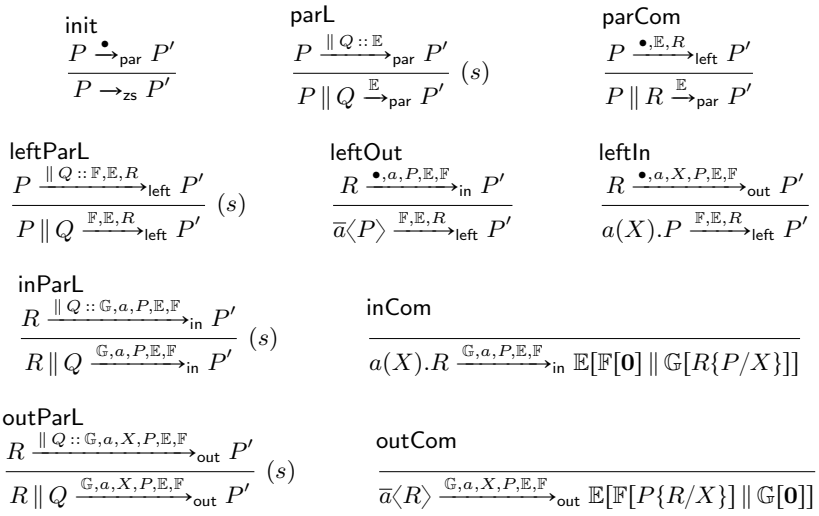

Fig. 3. Left-first Zipper Semantics for HOcore

This format of semantics does not make it apparent how to *decompose* a term to find a redex. On the other hand, structural operational semantics (SOS) offers another common semantic format that makes it more explicit how to navigate in a term to find a redex, but it does not store the traversed path. Zipper semantics is a middle ground between these two formats.

3.2 Left-first Zipper Semantics

A first step towards an abstract machine is to make explicit the step-by-step decomposition of a term into a context and a redex. To this end, we proposed zipper semantics, a combination of SOS and reduction semantics. Like a regular SOS, a zipper semantics goes through a term looking for a redex using structural rules, except the current position in the term is made explicit with a context as in reduction semantics.

Finding an HOcore redex requires us to recognize three constructs (parallel composition along with output and input on a shared name) and build the contexts $\mathbb{E}$, $\mathbb{F}$, and $\mathbb{G}$. Such a search can be conducted in many ways. A possibility is to search first for the parallel composition at the root of the redex $P \parallel Q$. From there, the communication rules of typical LTSs for process calculi [9,14,16] have two inductive premises looking for the output in P and the input in Q, or the opposite. To be closer to an abstract machine, we sequentialize the search by going left in P to find either an output or an input on some name a, and we then look for a complementary action on the same name in Q. Starting with the left process is an arbitrary choice—going right would produce a symmetric semantics.

Figure 3 presents such a left-first zipper semantics, where we omit the symmetric versions of the rules marked with the symbol (s)–a convention we follow from now on. There exist different kinds of transitions, or *modes*, to reuse a terminology used in abstract machines. We briefly explain the rules by describing them from the conclusion to their promise.

The init rule initializes the search with the par mode, the purpose of which is to decompose the initial process as $\mathbb{E}[P \parallel R]$, where P and R are the communicating processes. The rules parL and (omitted) parR are going through the initial process by focusing on the subprocess on the left or right of the parallel composition. We see why interpreting contexts inside-out is convenient: focusing on a subprocess amounts to pushing the corresponding frame on top of the current context.

Once we find the parallel composition $P \parallel R$ at the root of the redex, the parCom rule switches to the $\xrightarrow{\mathbb{F},\mathbb{E},R}_{\mathsf{left}}$ transition (with $\mathbb{F}$ set to $\bullet$), which looks for an input or an output in P, while building the context $\mathbb{F}$ and remembering $\mathbb{E}$ and R for later. As in the par mode, the rules leftParL and leftParR focus on the left or right subprocess and remembers the other in the context $\mathbb{F}$.

We eventually reach either an output (rule leftOut) or an input (rule leftIn) on some name a. It triggers the search for respectively an input or an output in R on the same name using the transitions $\xrightarrow{\mathbb{G},a,P,\mathbb{E},\mathbb{F}}_{\mathsf{in}}$ or $\xrightarrow{\mathbb{G},a,X,P,\mathbb{E},\mathbb{F}}_{\mathsf{out}}$, constructing the context $\mathbb{G}$ at the same time. The communication on a happens once the decomposition is complete with either rule inCom or outCom.

Example 2. We reuse the same example as before.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{a(X).R \xrightarrow{Q \parallel :: \bullet, a, \mathbf{0}, \bullet, \parallel P :: \bullet}_{\mathsf{in}} (\mathbf{0} \parallel P) \parallel (Q \parallel R\{\mathbf{0}/X\})}\,\mathsf{inCom}}{Q \parallel a(X).R \xrightarrow{\bullet, a, \mathbf{0}, \bullet, \parallel P :: \bullet}_{\mathsf{in}} (\mathbf{0} \parallel P) \parallel (Q \parallel R\{\mathbf{0}/X\})}\,\mathsf{inParR}}{\overline{a}\langle\mathbf{0}\rangle \xrightarrow{\parallel P :: \bullet, \bullet, Q \parallel a(X).R}_{\mathsf{left}} (\mathbf{0} \parallel P) \parallel (Q \parallel R\{\mathbf{0}/X\})}\,\mathsf{leftOut}}{\overline{a}\langle\mathbf{0}\rangle \parallel P \xrightarrow{\bullet, \bullet, Q \parallel a(X).R}_{\mathsf{left}} (\mathbf{0} \parallel P) \parallel (Q \parallel R\{\mathbf{0}/X\})}\,\mathsf{leftParL}}{(\overline{a}\langle\mathbf{0}\rangle \parallel P) \parallel (Q \parallel a(X).R) \xrightarrow{\bullet}_{\mathsf{par}} (\mathbf{0} \parallel P) \parallel (Q \parallel R\{\mathbf{0}/X\})}\,\mathsf{parCom}}{(\overline{a}\langle\mathbf{0}\rangle \parallel P) \parallel (Q \parallel a(X).R) \rightarrow_{\mathsf{zs}} (\mathbf{0} \parallel P) \parallel (Q \parallel R\{\mathbf{0}/X\})}\,\mathsf{init}$$

Unlike with complementary semantics, the structure of the initial process is preserved. Computation happens only when applying the axiom inCom; the other rules do not change the resulting process in any way.

3.3 Output-first Zipper Semantics

A different strategy to reach the same decomposition is as follows: after finding the parallel decomposition $P \parallel Q$ at the root of the redex, we look for the output in either P or Q, and then for the input in the other process. Again, the choice of looking for the output first is arbitrary, and we could start with the input instead. We give the rule of the output-first strategy in in Figure 4.

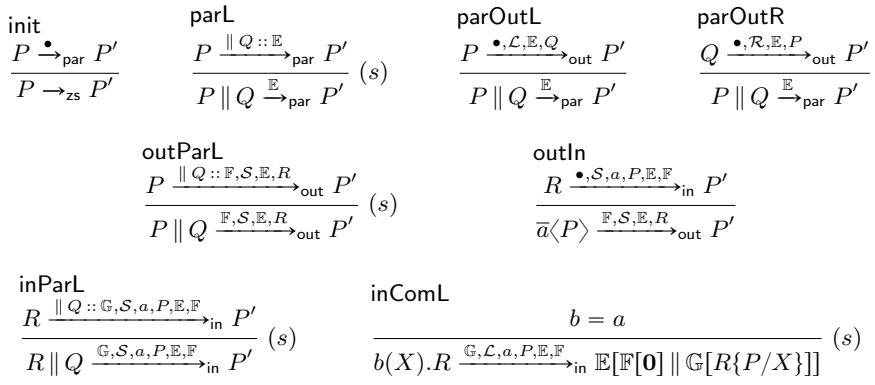

Fig. 4. Output-first Zipper Semantics for HOcore

The transition $\xrightarrow{\mathbb{E}}_{\mathsf{par}}$ is searching for the parallel composition and proceeds as in Section 3.2. Once we find it, we look for the output either on the left or on the right with respectively rules parOutL and parOutR. We record the side we pick with a parameter $\mathcal{S} ::= \mathcal{L} \mid \mathcal{R}$. For example, in rule parOutL, we look for an output in P on the left ($\mathcal{L}$), remembering that we should later search for a corresponding input in Q. We also initialize the context $\mathbb{F}$ surrounding the output with $\bullet$ and remember $\mathbb{E}$ as the context enclosing the whole redex.

The transition $\xrightarrow{\mathbb{F},\mathcal{S},\mathbb{E},R}_{\mathsf{out}}$ decomposes its source process to find an output, building $\mathbb{F}$ at the same time: the other parameters $\mathcal{S}$, $\mathbb{E}$, and R remain unchanged during the search. When we find the output $\overline{a}\langle P\rangle$ (rule outIn), we look for a corresponding input in R using $\xrightarrow{\mathbb{G},\mathcal{S},a,P,\mathbb{E},\mathbb{F}}_{\mathsf{in}}$, which builds the context $\mathbb{G}$ during the search. Once we find an input on a, we compute the result of the communication, which depends whether the output is on the left (rule inComL) or on the right (omitted rule inComR).

Example 3. The output-first derivation for our ongoing example is as follows.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{a(X).R \xrightarrow{Q \parallel ::\bullet,\mathcal{L},a,\mathbf{0},\bullet,\parallel P::\bullet}_{\mathsf{in}} (\mathbf{0} \parallel P) \parallel (Q \parallel R\{\mathbf{0}/X\})}\,\mathsf{inComL}}{Q \parallel a(X).R \xrightarrow{\bullet,\mathcal{L},a,\mathbf{0},\bullet,\parallel P::\bullet}_{\mathsf{in}} (\mathbf{0} \parallel P) \parallel (Q \parallel R\{\mathbf{0}/X\})}\,\mathsf{inParR}}{\overline{a}\langle\mathbf{0}\rangle \xrightarrow{\parallel P::\bullet,\mathcal{L},\bullet,Q \parallel a(X).R}_{\mathsf{out}} (\mathbf{0} \parallel P) \parallel (Q \parallel R\{\mathbf{0}/X\})}\,\mathsf{outIn}}{\overline{a}\langle\mathbf{0}\rangle \parallel P \xrightarrow{\bullet,\mathcal{L},\bullet,Q \parallel a(X).R}_{\mathsf{out}} (\mathbf{0} \parallel P) \parallel (Q \parallel R\{\mathbf{0}/X\})}\,\mathsf{outParL}}{(\overline{a}\langle\mathbf{0}\rangle \parallel P) \parallel (Q \parallel a(X).R) \xrightarrow{\bullet}_{\mathsf{par}} (\mathbf{0} \parallel P) \parallel (Q \parallel R\{\mathbf{0}/X\})}\,\mathsf{parOutL}}{(\overline{a}\langle\mathbf{0}\rangle \parallel P) \parallel (Q \parallel a(X).R) \rightarrow_{\mathsf{zs}} (\mathbf{0} \parallel P) \parallel (Q \parallel R\{\mathbf{0}/X\})}\,\mathsf{init}$$

In that particular example, the derivation remains close of the previous one because the output process is on the left of the redex. The difference is in the

extra information $\mathcal{L}$ recorded in the label, which tells us where should be the continuation of the output compared to the input.

3.4 Leaf-first Zipper Semantics

The exploration strategies described so far are called *root-first*, because they start the search with the operator at the root of the redex (the parallel composition). In contrast, a *leaf-first* strategy starts with an operator at the leaves of the redex, the input or the output. In the case of HOcore, it does not matter which one we choose. In a process calculus with name restriction [15], the communication is less symmetric because of the scope extrusion of the name restrictions surrounding the message output. In such a calculus, it is more convenient to start with the output [10], so we describe this strategy also in this paper.

The rules are given in Figure 5, using the same modes out, par, and in as in the output-first strategy, but with a different meaning. The rule init starts the search at the top of the term with the output mode out. The rules $\mathsf{outParL}$ and $\mathsf{outParR}$ change the current focus to a subterm. When we apply the rule outOut, the initial term is decomposed as $\mathbb{E}[\overline{a}\langle M\rangle]$, and we switch to the par mode.

The par mode searches the parallel composition which separates the output and input processes *from the inside out*, i.e., starting from the leaf (the message sending) and walking the context. The transition $\mathbb{E} \xrightarrow{K,a,M}_{\mathsf{par}} P'$ goes through $\mathbb{E}$ looking for the parallel composition, constructing K (the continuation of the output) while doing so, and remembering that M is sent on a. The rules outOut initializes the continuation of the par mode with $\mathbf{0}$.

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. We search for the input in Q with a transition $Q \xrightarrow{\mathbb{E},a,M}_{\mathsf{in}} P'$, where $\mathbb{E}$ is the context surrounding Q, initially either $K \| :: \mathbb{E}$ (rule parInL) or $\| K :: \mathbb{E}$ (rule parInR).

The in mode is then going through the receiving process (from the outside in), pushing the processes in parallel on the context $\mathbb{E}$ (rules inParL and inParR). Once an input $a(X).R$ has been found, the communication happens resulting in $\mathbb{E}[R\{M/X\}]$. The decomposition of the initial process into a redex is not as obvious in rule inCom as in the corresponding rules in Figure 3 or 4, but one can still prove the correspondence between the leaf-first zipper semantics and the context-based reduction semantics [10].

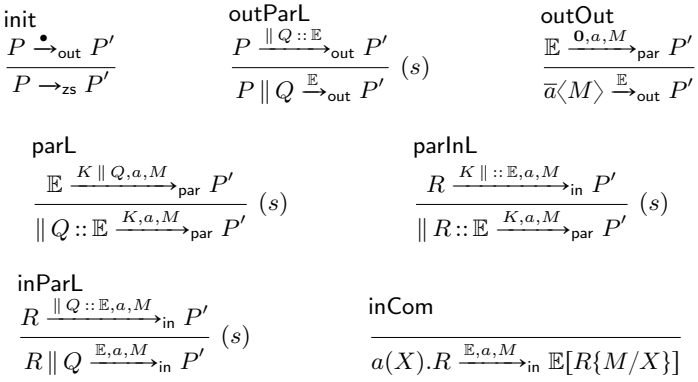

Fig. 5. Leaf-First Zipper Semantics for HOcore

Example 4. The leaf-first derivation is quite different from the previous ones.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{a(X).R \xrightarrow{Q \parallel :: (\mathbf{0} \parallel P) \parallel :: \bullet, a, \mathbf{0}}_{\mathsf{in}} (\mathbf{0} \parallel P) \parallel (Q \parallel R\{\mathbf{0}/X\})}\,\mathsf{inCom}}{Q \parallel a(X).R \xrightarrow{(\mathbf{0} \parallel P) \parallel :: \bullet, a, \mathbf{0}}_{\mathsf{in}} (\mathbf{0} \parallel P) \parallel (Q \parallel R\{\mathbf{0}/X\})}\,\mathsf{inParR}}{\parallel (Q \parallel a(X).R) :: \bullet \xrightarrow{\mathbf{0} \parallel P, a, \mathbf{0}}_{\mathsf{par}} (\mathbf{0} \parallel P) \parallel (Q \parallel R\{\mathbf{0}/X\})}\,\mathsf{parInL}}{\parallel P :: \parallel (Q \parallel a(X).R) :: \bullet \xrightarrow{\mathbf{0}, a, \mathbf{0}}_{\mathsf{par}} (\mathbf{0} \parallel P) \parallel (Q \parallel R\{\mathbf{0}/X\})}\,\mathsf{parL}}{\overline{a}\langle \mathbf{0} \rangle \xrightarrow{\parallel P :: \parallel (Q \parallel a(X).R) :: \bullet}_{\mathsf{out}} (\mathbf{0} \parallel P) \parallel (Q \parallel R\{\mathbf{0}/X\})}\,\mathsf{outOut}}{\overline{a}\langle \mathbf{0} \rangle \parallel P \xrightarrow{\parallel (Q \parallel a(X).R) :: \bullet}_{\mathsf{out}} (\mathbf{0} \parallel P) \parallel (Q \parallel R\{\mathbf{0}/X\})}\,\mathsf{outParL}}{(\overline{a}\langle \mathbf{0} \rangle \parallel P) \parallel (Q \parallel a(X).R) \xrightarrow{\bullet}_{\mathsf{out}} (\mathbf{0} \parallel P) \parallel (Q \parallel R\{\mathbf{0}/X\})}\,\mathsf{outParL}}{(\overline{a}\langle \mathbf{0} \rangle \parallel P) \parallel (Q \parallel a(X).R) \rightarrow_{\mathsf{zs}} (\mathbf{0} \parallel P) \parallel (Q \parallel R\{\mathbf{0}/X\})}\,\mathsf{init}$$

This style is where the analogy with the zipper data structure [8] is more obvious: we go up and down in the term by going through the context represented inside-out as a list.

4 Conclusion

We have presented two applications of a serialized semantics format to describe process calculi. First, the *complementary semantics* was introduced to prove congruence properties of higher-order process calculi with passivation. This was done during the PhD of Sergueï Lenglet, supervised by Jean-Bernard Stefani and Alan Schmitt. Many years later, Sergueï and Alan continued working on the semantics of process calculi, and we reused some of these fruitful ideas to define the *zipper semantics* of a calculus, from which an abstract machine (and an implementation) can be automatically derived. We are currently working

on the derivation of the zipper semantics itself from a context-based reduction semantics, which would simplify further the experimentation around process calculi.

References

1. M. Baldamus. *Semantics and Logic of Higher-Order Processes: Characterizing Late Context Bisimulation*. PhD thesis, Berlin University of Technology, 1998.
2. Małgorzata Biernacka, Dariusz Biernacki, Sergueï Lenglet, and Alan Schmitt. Non-Deterministic Abstract Machines. In Bartek Klin, Sławomir Lasota, and Anca Muscholl, editors, *33rd International Conference on Concurrency Theory (CONCUR 2022)*, volume 243 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 7:1–7:24, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.
3. Olivier Danvy. From reduction-based to reduction-free normalization. In Pieter W. M. Koopman, Rinus Plasmeijer, and S. Doaitse Swierstra, editors, *Advanced Functional Programming, 6th International School, AFP 2008, Heijen, The Netherlands, May 2008, Revised Lectures*, volume 5832 of *Lecture Notes in Computer Science*, pages 66–164. Springer, 2008.
4. Matthias Felleisen, Robert Bruce Findler, and Matthew Flatt. *Semantics Engineering with PLT Redex*. The MIT Press, 2009.
5. Matthias Felleisen and Robert Hieb. The revised report on the syntactic theories of sequential control and state. *Theor. Comput. Sci.*, 103(2):235–271, 1992.
6. Andrew D. Gordon. Bisimilarity as a theory of functional programming. *Electronic Notes in Theoretical Computer Science*, 1:232–252, 1995.
7. Douglas J. Howe. Proving congruence of bisimulation in functional programming languages. *Information and Computation*, 124(2):103–112, 1996.
8. Gérard P. Huet. The zipper. *J. Funct. Program.*, 7(5):549–554, 1997.
9. Ivan Lanese, Jorge A Pérez, Davide Sangiorgi, and Alan Schmitt. On the expressiveness and decidability of higher-order process calculi. *Information and Computation*, 209(2):198–226, February 2011. Extended abstract in *Logic in Computer Science (LICS)*, 2008. Coq formalization available at `http://www.irisa.fr/celtique/aschmitt/research/hocore/`.
10. Sergueï Lenglet and Alan Schmitt. Leaf-First Zipper Semantics. In *FORTE 2024 - 44th International Conference on Formal Techniques for Distributed Objects, Components, and Systems*, Groningen, Netherlands, June 2024.
11. Sergueï Lenglet, Alan Schmitt, and Jean-Bernard Stefani. Characterizing contextual equivalence in calculi with passivation. *Information and Computation*, 209(11):1390–1433, 2011.
12. Sergueï Lenglet, Alan Schmitt, and Jean-Bernard Stefani. Howe's method in calculi with passivation. In *CONCUR '09*, volume 5710 of *LNCS*, pages 448–462. Springer, 2009.
13. Sergueï Lenglet, Alan Schmitt, and Jean-Bernard Stefani. Normal bisimulations in process calculi with passivation. In *FoSSaCS '09*, volume 5504 of *LNCS*, pages 257–271. Springer, 2009.
14. Davide Sangiorgi. Bisimulation in higher-order process calculi. In Ernst-Rüdiger Olderog, editor, *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*, volume A-56 of *IFIP Transactions*, pages 207–224. North-Holland, 1994.

15. Davide Sangiorgi. Bisimulation for higher-order process calculi. *Information and Computation*, 131(2):141–178, 1996.
16. Davide Sangiorgi and David Walker. *The Pi-Calculus: A Theory of Mobile Processes*. Cambridge University Press, 2001.