Substructural Observed Communication Semantics

Ryan Kavanagh

Computer Science Department
Carnegie Mellon University
Pittsburgh, Pennsylvania, 15213-3891, USA
`rkavanagh@cs.cmu.edu`

Session-types specify communication protocols for communicating processes, and session-typed languages are often specified using substructural operational semantics given by multiset rewriting systems. We give an observed communication semantics [2] for a session-typed language with recursion, where a process's observation is given by its external communications. To do so, we introduce *fair executions* for multiset rewriting systems, and extract observed communications from fair process executions. This semantics induces an intuitively reasonable notion of observational equivalence that we conjecture coincides with semantic equivalences induced by denotational semantics [15], bisimulations [13], and barbed congruences [16, 27] for these languages.

1 Introduction

A proofs-as-processes correspondence between linear logic and the session-typed π-calculus is the basis of many programming languages for message-passing concurrency [4, 5, 26, 28]. Session types specify communication protocols, and all communication with session-typed processes must respect these protocols. If we take seriously the idea that we can only interact with processes through session-typed communication, then the only thing we can observe about them is their communications. Indeed, timing differences in communication are not meaningful due to the non-deterministic scheduling of process reductions, and "forwarding" or "linking" of channels renders process termination meaningless, even in the presence of recursion. It follows that processes should be observationally indistinguishable only if they always send the same output given the same input.

These ideas underlie Atkey's [2] novel *observed communication semantics* (OCS) for Wadler's Classical Processes [28]. Atkey's OCS uses a big-step evaluation semantics to observe communications on channels deemed "observable". Processes are then observationally equivalent whenever they have the same observed communications in all contexts.

Building on these ideas, **we give an OCS for session-typed languages that are specified using substructural operational semantics** (SSOS), a form of multiset rewriting. Our work differs from Atkey's on several key points. First, we assume that communication is asynchronous rather than synchronous. This assumption costs us nothing, for synchronous communication can be encoded in asynchronous systems [20], and it simplifies the semantics by eliminating the need for "configurations" and "visible" cuts. More importantly, **our OCS supports recursive and non-terminating processes**. To do so, we observe communications from process traces in (a conservative extension of) the usual SSOSs, instead of defining a separate big-step semantics.

To ensure that observed communications are well-defined in the presence of non-termination, we require that process executions be *fair*. Intuitively, fairness ensures that if a process can make progress, then it eventually does so. Fairness is also motivated by ongoing efforts to relate existing SSOSs to domain-theoretic semantics for this style of language [15]. There, processes denote continuous functions

between domains of session-typed communications, and fairness is built-in. To this end, **we introduce** *fair executions* **of multiset rewriting systems** (MRS) and give sufficient conditions for an MRS to have fair executions. We also introduce a new notion of trace equivalence, *union-equivalence*, that is key to defining our OCS.

We study fair executions of MRSs and their properties in section 2. In section 3, we give an SSOS for a session-typed language arising from a proofs-as-processes interpretation of intuitionistic linear logic. It supports recursive processes and types. Though it is limited, it represents the core of other SSOS-specified session-typed languages [3, 13, 15, 20, 25], and the techniques presented in this paper scale to their richer settings. In section 4, we give our observed communication semantics, where we use a coinductively defined judgment to extract observations from fair executions.

2 Fair Executions of Multiset Rewriting Systems

In this section, we introduce *fairness* and *fair executions* for multiset rewriting systems. We begin by revisiting (first-order) multiset rewriting systems, as presented by Cervesato et al. [9]. We present a notion of fairness for sequences of rewriting steps, and constructively show that under reasonable hypotheses, all fair sequences from the same multiset are permutations of each other. We introduce a new notion of trace equivalence, "union-equivalence", and give sufficient conditions for traces to be union-equivalent. Fairness and union-equivalence will be key ingredients for defining the observed communication semantics of section 4.

A **multiset** M is a pair (S, m) where S is a set (the *underlying set*) and $m : S \to \mathbb{N}$ is a function. It is finite if $\sum_{s \in S} m(s)$ is finite. We say s is an **element** of M, $s \in M$, if $m(s) > 0$. When considering several multisets, we assume without loss of generality that they have equal underlying sets. The **sum** M_1, M_2 of multisets $M_1 = (S, m_1)$ and $M_2 = (S, m_2)$ is the multiset $(S, \lambda s \in S. m_1(s) + m_2(s))$. Their **intersection** $M_1 \cap M_2$ is the multiset $(S, \lambda s \in S. \min(m_1(s), m_2(s)))$. Their **difference** $M_1 \setminus M_2$ is the multiset $(S, \lambda s \in S. \max(0, m_1(s) - m_2(s)))$. We say that M_1 is **included** in M_2, written $M_1 \subseteq M_2$, if $m_1(s) \leq m_2(s)$ for all $s \in S$.

Consider finite multisets M of first-order atomic formulas over some signature whose constants are drawn from some countably infinite set. We call closed formulas **judgments**. Judgments represent facts, some of which we may deem to be persistent. To this end, we partition formulas as **persistent** (indicated by bold face, **p**) and **ephemeral** (indicated by sans serif face, p). We write $M(\vec{x})$ to mean that the formulas in M draw their variables from $\vec{x}$. A **multiset rewrite rule** r is an ordered pair of multisets $F(\vec{x})$ and $G(\vec{x}, \vec{n})$, where the multiset $\pi(\vec{x})$ of persistent formulas in $F(\vec{x})$ is included in $G(\vec{x}, \vec{n})$. We interpret the variables $\vec{x}$ as being universally quantified and the variables $\vec{n}$ as being existentially quantified. This relation is made explicit using the syntax

$$r : \forall \vec{x}. F(\vec{x}) \to \exists \vec{n}. G(\vec{x}, \vec{n}).$$

In practice, we often elide $\forall \vec{x}$ and do not repeat the persistent formulas $\pi(\vec{x}) \subseteq F(\vec{x})$ on the right side of the arrow. A **multiset rewriting system** (MRS) is a set $\mathscr{R}$ of multiset rewrite rules.

Multiset rewrite rules describe localized changes to multisets of judgments. Given a rule $r : \forall \vec{x}. F(\vec{x}) \to \exists \vec{n}. G(\vec{x}, \vec{n})$ in $\mathscr{R}$ and some choice of constants $\vec{c}$ for $\vec{x}$, we say that the **instantiation** $r(\vec{c}) : F(\vec{c}) \to \exists \vec{n}. G(\vec{c}, \vec{n})$ is **applicable** to a multiset M of judgments if there exists a multiset M' such that $M = F(\vec{c}), M'$. The rule r is applicable to M if $r(\vec{c})$ is applicable to M for some $\vec{c}$. In these cases, the **result** of applying $r(\vec{c})$ to M is the multiset $G(\vec{c}, \vec{d}), M'$, where $\vec{d}$ is a choice of fresh constants. In particular, we assume that the constants $\vec{d}$ do not appear in M or in $\mathscr{R}$. We call $\theta = [\vec{c}/\vec{x}]$ the **matching substitution** and $\xi = [\vec{d}/\vec{n}]$

the **fresh-constant substitution**. The **instantiating substitution** for r relative to M is the composite substitution $\delta = (\theta, \xi)$. We capture this relation using the syntax

$$F(\vec{c}), M' \xrightarrow{(r;\delta)} G(\vec{c}, \vec{n}), M'.$$

For conciseness, we often abuse notation and write $r(\theta)$, $F(\theta)$, and $G(\theta, \xi)$ for $r(\vec{c})$, $F(\vec{c})$, and $G(\vec{c}, \vec{d})$. We call $F(\vec{c})$ the **active** multiset and M' the **stationary** multiset.

Given an MRS $\mathscr{R}$ and a multiset M_0, a **trace** from M_0 is a countable sequence of steps

$$M_0 \xrightarrow{(r_1;\delta_1)} M_1 \xrightarrow{(r_2;\delta_2)} M_2 \xrightarrow{(r_3;\delta_3)} \cdots \tag{1}$$

such that, where $\delta_i = (\theta_i, \xi_i)$,

1. for all i, ξ_i is one-to-one;

2. for all $i < j$, the constants in M_i and ξ_j are disjoint.

The notation $(M_0, (r_i; \delta_i)_{i \in I})$ abbreviates the trace (1), where I always ranges over $\mathbb{N}^+$ or $\mathbf{n} = \{1, \ldots, n\}$ for some $n \in \mathbb{N}$. An **execution** is a maximally long trace.

Example 1. *We model queues using an MRS. Let the judgment* $\mathsf{que}(q, \$)$ *mean that q is the empty queue, and let* $\mathsf{que}(q, v \to q')$ *mean that the queue q has value v at its head and that its tail is the queue q'. Then the multiset* $Q = \mathsf{que}(q, 0 \to q'), \mathsf{que}(q', \$)$ *describes a one-element queue containing 0. The following two rules capture enqueuing values on empty and non-empty queues, respectively, where the formula* $\mathsf{enq}(q, v)$ *is used to enqueue v onto the queue q:*

$$e_1 : \forall x, y. \, \mathsf{enq}(x, y), \mathsf{que}(x, \$) \to \exists z. \, \mathsf{que}(x, y \to z), \mathsf{que}(z, \$),$$
$$e_2 : \forall x, y, z, w. \, \mathsf{enq}(x, y), \mathsf{que}(x, z \to w) \to \mathsf{que}(x, z \to w), \mathsf{enq}(w, y).$$

The following sequence is an execution from $Q, \mathsf{enq}(q, 1)$*, and it captures enqueuing 1 on the queue q:*

$$Q, \mathsf{enq}(q, 1) \xrightarrow{(e_2;([q,1,0,q'/x,y,z,w],\emptyset))} Q, \mathsf{enq}(q', 1) \xrightarrow{(e_1;([q',1/x,y],[a/z]))} \mathsf{que}(q, 0 \to q'), \mathsf{que}(q', 1 \to a), \mathsf{que}(a, \$).$$

The constants in fresh-constant substitutions are not semantically meaningful, so we identify traces up to refreshing substitutions. A **refreshing substitution** for a trace $T = (M_0, (r_i; (\theta_i, \xi_i))_i)$ is a collection of fresh-constant substitutions $\eta = (\eta_i)_i$ such that $[\eta]T = (M_0, (r_i; (\theta_i, \eta_i))_i)$ is also a trace. Explicitly, we identify traces T and T' if there exists a refreshing substitution η such that $T' = [\eta]T$.

Given rules $r_i : \forall \vec{x}_i. F_i(\vec{x}_i) \to \exists \vec{n}_i. G_i(\vec{x}_i, \vec{n}_i)$ and matching substitutions θ_i for $i = 1, 2$, we say that the instantiations $r_1(\theta_1)$ and $r_2(\theta_2)$ are **equivalent**, $r_1(\theta_1) \equiv r_2(\theta_2)$, if both $F_1(\theta_1) = F_2(\theta_2)$ and (up to renaming of bound variables) $\exists \vec{n}_1. G_1(\theta_1, \vec{n}_1) = \exists \vec{n}_2. G_2(\theta_2, \vec{n}_2)$; otherwise they are **distinct**. Application does not distinguish between equivalent instantiations: if $r_1(\theta_1) \equiv r_2(\theta_2)$ are applicable to M_0, then applying each to M_0 gives the same result up to refreshing substitution.

Given an MRS $\mathscr{R}$, we say that an execution $(M_0, (r_i; \delta_i)_{i \in I})$ is **fair** if for all $i \in I$, $r \in \mathscr{R}$, and θ, whenever $r(\theta)$ is applicable to M_i, there exists a $j > i$ such that $r_j(\theta_j) \equiv r(\theta)$. Given a fair trace T, we write $\phi_T(i, r, \theta)$ for the least such j. In the case of MRSs specifying SSOSs of session-typed languages, this notion of fairness implies *strong process fairness* [11, 12, 18], which guarantees that if a process can take a step infinitely often, then it does so infinitely often. In particular, it implies that if a process can take a step, then it eventually does so.
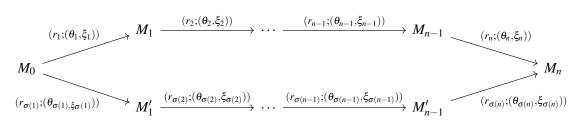
Example 2. *The execution of example 1 is fair.*

Proposition 1 (Fair Tail Property). *If $(M_0, (t_i; \delta_i)_{i \in I})$ is fair, then so is $(M_n, (t_i; \delta_i)_{n < i, i \in I})$ for all $n \in I$.*

We consider various criteria that imply fairness. The first will be interference-freedom, which roughly means that at any given point, the order in which we apply applicable rules does not matter. It will hold whenever the rules do not "overlap". In general, given an MRS $\mathscr{R}$ and a property P, we say P holds **from** M_0 if for all traces $(M_0, (r_i; \delta_i)_{i \in I})$, P holds for M_0 and for M_i for all $i \in I$.

Write S_I for the group of bijections on I; its elements are called permutations. A permutation $\sigma \in S_I$ acts on a trace $T = (M_0, (t_i; \delta_i)_{i \in I})$ to produce a sequence $\sigma \cdot T = (M_0, (t_{\sigma(i)}; \delta_{\sigma(i)})_{i \in I})$. This sequence $\sigma \cdot T$ is a **permutation of** T whenever it is also a trace. We adopt group-theoretic notation for cyclic permutations and write $(x, \sigma(x), \sigma(\sigma(x)), \dots)$ for a cyclic permutation $\sigma : I \to I$; implicit is that all elements not in the orbit of x are fixed by σ. Cycles of length two are called transpositions.

Consider an MRS $\mathscr{R}$ and let $r_1(\theta_1), \dots, r_n(\theta_n)$ enumerate all distinct instantiations of rules in $\mathscr{R}$ applicable to M_0. We say that $\mathscr{R}$ **commutes** on M_0 or is **interference-free** on M_0 if for all corresponding pairwise-disjoint fresh-constant substitutions ξ_i, the following diagram commutes for all permutations $\sigma \in S_{\mathbf{n}}$, and both paths around it are traces:

We note that interference-freedom is only defined if the enumeration of distinct applicable instantiations is finite. The following proposition is an immediate consequence of the definition of commuting rules:

Proposition 2. *Let $\mathscr{R}$ commute on M_0, and let $r_i(\theta_i)$ with $1 \leq i \leq n$ be the distinct instantiations applicable on M_0. If $M_0 \xrightarrow{(r_1;(\theta_1,\xi_1))} M_1$, then $r_2(\theta_2), \dots, r_n(\theta_n)$ are applicable to and commute on M_1.*

Interference-freedom implies the existence of fair executions:

Proposition 3 (Fair Scheduler). *Assume the axiom of countable choice. If $\mathscr{R}$ is interference-free from M_0, then there is a fair execution from M_0.*

Proof (Sketch). Let Q be a queue of rule instantiations applicable to M_0. Given M_n, dequeue a rule $r_{n+1}(\theta_{n+1})$ from Q and use the axiom of countable choice to choose a suitably disjoint fresh-constant substitution ξ_{n+1}. By interference-freedom, it is applicable to M_n, and let M_{n+1} be the result of doing so. Enqueue all newly-applicable rule instantiations. If Q is ever empty, then the trace is finite but maximally long. In all cases, the trace gives a fair execution: every distinct applicable rule instantiation is enqueued and then applied after some finite number of steps. $\square$

Though interference-freedom simplifies fair scheduling, it is primarily of interest for reasoning about executions. For example, it is useful for showing confluence properties. It also lets us safely permute certain steps in a trace without affecting observations for session-typed processes (see section 4). This can simplify process equivalence proofs, because it lets us assume that related steps in an execution happen one after another.

Interference-freedom is a strong property, but it arises frequently in nature. This is because many systems can be captured using rules whose active multisets do not overlap, and rules whose active multisets are non-overlapping commute. In fact, even if their active multisets overlap, the rules do not disable each other so long as they preserve these overlaps.

To make this intuition explicit, consider multisets $M_i \subseteq M$ for $1 \leq i \leq n$. Their **overlap** in M is $\Omega_M(M_1, \ldots, M_n) = M_1, \ldots, M_n \setminus M$. Consider an MRS $\mathcal{R}$ and let $r_i(\theta_i) : F_i(\theta_i) \rightarrow \exists \vec{n}_i. G_i(\theta_i, \vec{n}_i)$, $1 \leq i \leq n$, enumerate all distinct instantiations of rules in $\mathcal{R}$ applicable to M. We say that $\mathcal{R}$ is **non-overlapping** on M if for all $1 \leq i \leq n$ and fresh-constant substitutions ξ_i, $F_i(\theta_i) \cap \Omega_M(F_1(\theta_1), \ldots, F_n(\theta_n)) \subseteq G_i(\theta_i, \xi_i)$.

Example 3. *The MRS given by example 1 is non-overlapping from any multiset of the form Q, E where Q is a queue rooted at q, and E contains at most one judgment of the form $\mathsf{enq}(q, v)$.*

Proposition 4 characterizes the application of non-overlapping rules, while proposition 5 characterizes the relationship between commuting and non-overlapping rules.

Proposition 4. *Let $\mathcal{R}$ be non-overlapping on M_0 and let $r_i(\theta_i) : F_i(\theta_i) \rightarrow \exists \vec{n}_i. G(\theta_i, \vec{n}_i)$ with $1 \leq i \leq n$ be the distinct instantiations applicable to M_0. If $M_0 \xrightarrow{(r_1 ; (\theta_1, \xi_1))} M_1$ and $r_1, \ldots, r_n$ are non-overlapping on M_0, then $r_2(\theta_2), \ldots, r_n(\theta_n)$ are applicable to and non-overlapping on M_1.*

In particular, set $O = \Omega_{M_0}(F_1, \ldots, F_n) \cap F_1$. There exist F_1' and G_1' be such that $F_1 = O, F_1'$ and $G_1 = O, G_1'$, and there exists an M such that $M_0 = O, F_1', M$ and $M_1 = O, G_1', M$. The instantiations $r_2(\theta_2), \ldots, r_n(\theta_n)$ are all applicable to $O, M \subseteq M_1$.

Proposition 5. *An MRS commutes on M_0 if it is non-overlapping on M_0; the converse is false.*

For the remainder of this section, assume that if $(M_0, (r_i; \delta_i)_i)$ is a fair trace, then its MRS is interference-free from M_0. Interference-freedom implies the ability to safely permute finitely many steps that do not depend on each other. However, it is not obvious that finite permutations, let alone infinite permutations, preserve fairness. To show that they do, we use the following lemma to reduce arguments about infinite permutations to arguments about finite permutations:

Lemma 1. *For all $n \in \mathbb{N}$ and permutations $\sigma : \mathbb{N} \rightarrow \mathbb{N}$, set $\chi_\sigma(n) = \sup_{k \leq n} \sigma^{-1}(k)$. Then there exist permutations $\tau, \rho : \mathbb{N} \rightarrow \mathbb{N}$ such that $\sigma = \rho \circ \tau$, $\tau(k) = k$ for all $k > \chi_\sigma(n)$, and $\rho(k) = k$ for all $k \leq n$.*

The following proposition shows that permutations of prefixes of traces preserve fairness. Its proof uses a factorization of permutations into cycles permuting adjacent steps, where each cycle preserves fairness.

Proposition 6. *Consider an MRS $\mathcal{R}$ that is interference-free from M_0 and let $T = (M_0, (r_i; (\theta_i, \xi_i))_{i \in I})$ be a trace, an execution, or a fair execution. Let $\sigma \in S_I$ be such that for some $n \in I$, $\sigma(i) = i$ for all $i > n$. Then $\sigma \cdot T$ is respectively a trace, an execution, or a fair execution.*

Corollary 1. *Fairness is invariant under permutation, that is, if $\mathcal{R}$ is interference-free from M_0, T is a fair trace from M_0, and $\Sigma = \sigma \cdot T$ is a permutation of T, then Σ is also fair.*

Proof. Let $T = (M_0, (t_i; \delta_i)_i)$ and $\delta_i = (\theta_i, \xi_i)$, and let Σ be the trace $M_0 = \Sigma_0 \xrightarrow{(t_{\sigma(1)} ; \delta_{\sigma(1)})} \Sigma_1 \xrightarrow{(t_{\sigma(2)} ; \delta_{\sigma(2)})} \cdots$. Consider some rule $r \in \mathcal{R}$ such that $\Sigma_i \xrightarrow{(r ; (\theta, \xi))} \Sigma_i'$. We must show that there exists a j such that $\sigma(j) > \sigma(i)$, $t_{\sigma(j)}(\theta_{\sigma(j)}) \equiv r(\theta)$.

Let the factorization $\sigma = \rho \circ \tau$ be given by lemma 1 for $n = \sigma(i)$. By proposition 6, we get that $\tau \cdot T$ is fair. Moreover, by construction of τ, $\tau \cdot T$ and Σ agree on the first n steps and $n + 1$ multisets. By fairness, there exists a $k > \sigma(i)$ such that the k-th step in $\tau \cdot T$ is $r(\theta)$. By construction of ρ, $\rho(k) > \sigma(i)$, so this step appears after Σ_i in Σ as desired. We conclude that Σ is fair. $\qquad\square$

Corollary 1 established that permutations preserve fairness. Relatedly, all fair traces from a given multiset are permutations of each other. To do show this, we construct a potentially infinite sequence of permutations and use the following lemma to compose them:

Lemma 2. *Let $(\sigma_n)_{n\in I}$ be a family of bijections on I such that for all $m < n$,*

$$(\sigma_n \circ \cdots \circ \sigma_1)(m) = (\sigma_m \circ \cdots \circ \sigma_1)(m).$$

Let $\sigma : I \to I$ be given by $\sigma(m) = (\sigma_m \circ \cdots \circ \sigma_1)(m)$. Then σ is injective, but need not be surjective.

Lemma 3. *Let $\mathcal{R}$ be interference-free from M_0. Consider a fair execution $T = (M_0, (r_i; (\theta_i, \xi_i))_{i\in I})$ and a step $M_0 \xrightarrow{(t;(\tau,\rho))} M_1'$. Set $n = \phi_T(0, t, \tau)$ (so $t(\tau) \equiv r_n(\theta_n)$). Then $(1, \ldots, n) \cdot T$ is a permutation of T with $(t; (\tau, \xi_n))$ as its first step, and it is a fair execution.*

Proposition 7. *If $\mathcal{R}$ is interference-free from M_0, then all fair executions from M_0 are permutations of each other.*

Proof (Sketch). Consider traces $R = (R_0, (r_i; (\theta_i, \xi_i))_{i\in I})$ and $T = (T_0, (t_j; (\tau_j, \zeta_j))_{j\in J})$ where $R_0 = M_0 = T_0$. We construct a sequence of permutations $\sigma_0, \sigma_1, \ldots$, where $\Phi_0 = R$ and the step $\Phi_{n+1} = \sigma_{n+1} \cdot \Phi_n$ is given by lemma 3 such that Φ_{n+1} agrees with T on the first $n+1$ steps. We then assemble these permutations σ_n into an injection σ using lemma 2; fairness ensures that it is a surjection. We have $T = \sigma \cdot R$ by construction.   □

Let the support of a multiset $M = (S, m)$ be the set $\mathrm{supp}(M) = \{s \in S \mid m(s) > 0\}$. We say that two traces $T = (M_0; (r_i, \delta_i)_I)$ and T' are **union-equivalent** if T' can be refreshed to a trace $(N_0; (s_j, \rho_j)_j)$ such that the unions of the supports of the multisets in the traces are equal, i.e., such that

$$\bigcup_{i \geq 0} \mathrm{supp}(M_i) = \bigcup_{j \geq 0} \mathrm{supp}(N_j)$$

Lemma 4. *Consider an MRS and assume T is a permutation of S. Then T and S are union-equivalent.*

Proof. Consider a trace $(M_0, (r_i; \delta_i)_i)$. For all n, each judgment in M_n appears either in M_0 or in the result of some rule r_i with $i \leq n$. Traces T and S start from the same multiset and have the same rules. It follows that they are union-equivalent.   □

Corollary 2 will be key in section 4 to showing that processes have unique observations.

Corollary 2. *If $\mathcal{R}$ is interference-free from M, then all fair executions from M are union-equivalent.*

3   Session-Typed Processes

Session types specify communication protocols between communicating processes. In this section, we present a session-typed language arising from a proofs-as-programs interpretation of intuitionistic linear logic [4] extended to support recursive processes and recursive types.

We let A, B, C range over session types and a, b, c range over channel names. A process P provides a distinguished service A_0 over some channel c_0, and may use zero or more services A_i on channels c_i. In this sense, a process P is a server for the service A_0, and a client of the services A_i. The channels $c_1 : A_1, \ldots, c_n : A_n$ form a linear context Δ. We write $\Delta \vdash P :: c_0 : A_0$ to capture these data. We also allow P to depend on process variables p_i of type $\{b : B \leftarrow \Delta\}$. Values of type $\{b : B \leftarrow \Delta\}$ are processes Q such that $\Delta \vdash Q :: b : B$. We write Π for structural contexts of process variables $p_i : \{a_i : A_i \leftarrow \Delta_i\}$. These data are captured by the judgment $\Pi ; \Delta \vdash P :: c_0 : A_0$, and we say that P is closed if Π is empty.

At any given point in a computation, communication flows in a single direction on a channel $c : A$. The direction of communication is determined by the *polarity* of the type A, where session types are

partitioned as positive or negative [20]. Consider a process judgment $\Pi \mathbin{;} \Delta \vdash P :: c_0 : A_0$. Communication on positively-typed channels flows from left-to-right in this judgment: if A_0 is positive, then P can only send output on c_0, while if A_i is positive for $1 \leq i \leq n$, then P can only receive input on c_i. Symmetrically, communication on negatively-typed channels flows from right-to-left in the judgment. Bidirectional communication arises from the fact that the type of a channel evolves over the course of a computation, sometimes becoming positive, sometimes becoming negative.

Most session types have a polar dual, where the direction of communication is reversed. With one exception, we only consider positive session types here. Negative session types pose no difficulty and can be added by dualizing the constructions. To illustrate this dualization, we also consider the (negative) external choice type $\&\{l : A_l\}_{l \in L}$, the polar dual of the (positive) internal choice type $\oplus\{l : A_l\}_{l \in L}$.

The operational behaviour of closed processes is given by a substructural operational semantics (SSOS) in the form of a multiset rewriting system. The judgment $\mathsf{proc}(c, P)$ means that the closed process P provides a channel c. The judgment $\mathsf{msg}(c, m)$ means the channel c is carrying a message m. Process communication is asynchronous: processes send messages without synchronizing with recipients. To ensure that messages on a given channel are received in order, the $\mathsf{msg}(c, m)$ judgment encodes a queue-like structure similar to the queues of example 1, and we ensure that each channel name c is associated with at most one $\mathsf{msg}(c, m)$ judgment. For example, the multiset $\mathsf{msg}(c_0, m_0; c_0 \leftarrow c_1), \mathsf{msg}(c_1, m_0; c_1 \leftarrow c_2), \ldots$ captures the queue of messages $m_0, m_1, \ldots$ on c_0. There is no global ordering on sent messages: messages sent on different channels can be received out of order. We extend the usual SSOS with a new persistent judgment, $\mathbf{type}(c : A)$, which means that channel c has type A.

The **initial configuration** of $\cdot \mathbin{;} c_1 : A_1, \ldots, c_n : A_n \vdash P :: c_0 : A_0$ is the multiset

$$\mathsf{proc}(c_0, P), \mathbf{type}(c_0 : A_0), \ldots, \mathbf{type}(c_n : A_n).$$

A **process trace** is a trace from the initial configuration of a process, and a multiset in it is a **configuration**. A **fair execution** of $\cdot \mathbin{;} \Delta \vdash P :: c : A$ is a fair execution from its initial configuration.

We give the typing rules and the substructural operational semantics in section 3.1. In section 3.2, we study properties of process traces and fair executions. In particular, we show that each step in these traces preserves various invariants, that the MRS of section 3.1 is non-interfering from initial process configurations, and that every process has a fair execution.

3.1 Statics and Dynamics

The process $a \to b$ forwards all messages from the channel a to the channel b; it assumes that both channel have the same positive type. It is formed by ($\textsc{Fwd}^+$) and its operational behaviour is given by (2).

$$\frac{}{\Pi \mathbin{;} a : A \vdash a \to b :: b : A} \ (\textsc{Fwd}^+)$$

$$\mathsf{msg}(a, m), \mathsf{proc}(b, a \to b) \to \mathsf{msg}(b, m) \tag{2}$$

Process composition $a : A \leftarrow P; Q$ spawns processes P and Q that communicate over a shared private channel a of type A. It captures Milner's "parallel composition plus hiding" operation [17, pp. 20f.]. To ensure that the shared channel is truly private, we generate a globally fresh channel b for P and Q to communicate over.

$$\frac{\Pi \mathbin{;} \Delta_1 \vdash P :: a : A \quad \Pi \mathbin{;} a : A, \Delta_2 \vdash Q :: c : C}{\Pi \mathbin{;} \Delta_1, \Delta_2 \vdash a : A \leftarrow P; Q :: c : C} \ (\textsc{Cut})$$

$$\mathsf{proc}(c, a : A \leftarrow P; Q) \to \exists b.\mathsf{proc}(b, [b/a]P), \mathsf{proc}(c, [b/a]Q), \mathbf{type}(b : A) \tag{3}$$

The process close a closes a channel a of type $\mathbf{1}$ by sending the "close message" $*$ over a. Dually, the process wait $a;P$ blocks until it receives the close message on the channel a, and then continues as P.

$$\frac{}{\Pi\,;\cdot\vdash \mathsf{close}\ a :: a : \mathbf{1}}\ (\mathbf{1}R) \qquad \frac{\Pi\,;\Delta\vdash P :: c : C}{\Pi\,;\Delta,a : \mathbf{1}\vdash \mathsf{wait}\ a;P :: c : C}\ (\mathbf{1}L)$$

$$\mathsf{proc}(a,\mathsf{close}\ a) \to \mathsf{msg}(a,*) \tag{4}$$

$$\mathsf{msg}(a,*),\mathsf{proc}(c,\mathsf{wait}\ a;P) \to \mathsf{proc}(c,P) \tag{5}$$

Processes can communicate channels over channels of type $B\otimes A$, where the transmitted channel has type B and subsequent communication has type A. The process send $a\ b;\ P$ sends a channel b over channel a and then continues as P. To ensure a queue-like structure for messages on a, we generate a fresh channel name d for the "continuation channel" that will carry subsequent communications. The process $b \leftarrow$ recv $a;\ P$ blocks until it receives a channel over a, binds it to the name b, and continues as P. Operationally, we rename a in P to the continuation channel d carrying the remainder of the communications.

$$\frac{\Pi\,;\Delta\vdash P :: a : A}{\Pi\,;\Delta,b : B\vdash \mathsf{send}\ a\ b;\ P :: a : B\otimes A}\ (\otimes R^*) \qquad \frac{\Pi\,;\Delta,a : A,b : B\vdash P :: c : C}{\Pi\,;\Delta,a : B\otimes A\vdash b \leftarrow \mathsf{recv}\ a;\ P :: c : C}\ (\otimes L)$$

$$\mathsf{proc}(a,\mathsf{send}\ a\ b;\ P),\mathbf{type}(a : B\otimes A) \to \exists d.\mathsf{proc}(d,[d/a]P),\mathsf{msg}(a,\mathsf{send}\ a\ b;\ a \leftarrow d),\mathbf{type}(d : A) \tag{6}$$

$$\mathsf{msg}(a,\mathsf{send}\ a\ e;\ a \leftarrow d),\mathsf{proc}(c,b \leftarrow \mathsf{recv}\ a;\ P) \to \mathsf{proc}(c,[e,d/b,a]Q) \tag{7}$$

The internal choice type $\oplus\{l : A_l\}_{l\in L}$ offers a choice of services A_l. The process $a.k;P$ sends a label k on a to signal its choice to provide the service A_k on a. The process case $a\ \{l \Rightarrow P_l\}_{l\in L}$ blocks until it receives a label $k \in L$ on a, and then continues as P_k.

$$\frac{\Pi\,;\Delta\vdash P :: a : A_k \quad (k \in L)}{\Pi\,;\Delta\vdash a.k;P :: a : \oplus\{l : A_l\}_{l\in L}}\ (\oplus R_k) \qquad \frac{\Pi\,;\Delta,a : A_l\vdash P_l :: c : C \quad (\forall l \in L)}{\Pi\,;\Delta,a : \oplus\{l : A_l\}_{l\in L}\vdash \mathsf{case}\ a\ \{l \Rightarrow P_l\}_{l\in L} :: c : C}\ (\oplus L)$$

$$\mathsf{proc}(a,a.k;P),\mathbf{type}(a : \oplus\{l : A_l\}_{l\in L}) \to \exists d.\mathsf{msg}(a,a.k;a \leftarrow d),\mathsf{proc}(d,[d/a]P),\mathbf{type}(d : A_k) \tag{8}$$

$$\mathsf{msg}(a,a.k;a \leftarrow d),\mathsf{proc}(c,\mathsf{case}\ a\ \{l \Rightarrow P_l\}_{l\in L}) \to \mathsf{proc}(c,[d/a]P_k) \tag{9}$$

To illustrate the duality between positive and negative types, we consider the (negative) external choice type. It is the polar dual of the (positive) internal choice type. The external choice type $\&\{l : A_l\}_{l\in L}$ provides a choice of services A_l. The process case $a\ \{l \Rightarrow P_l\}_{l\in L}$ blocks until it receives a label $k \in L$ on a, and then continues as P_k. The process $a.k;P$ sends a label k on a to signal its choice to use the service A_k on a. Observe that, where a provider of an internal choice type *sends* a label in (8), a provider of the external choice type *receives* a label in (10). Analogously, a client of an internal choice type receives *receives* a label in (9), and a client of an external choice type *sends* a label in (11).

$$\frac{\Psi\,;\Delta\vdash P_l :: a : A_l \quad (\forall l \in L)}{\Psi\,;\Delta\vdash \mathsf{case}\ a\ \{l \Rightarrow P_l\}_{l\in L} :: a : \&\{l : A_l\}_{l\in L}}\ (\&R) \qquad \frac{\Psi\,;\Delta,a : A_k\vdash P :: c : C \quad (k \in L)}{\Psi\,;\Delta,a : \&\{l : A_l\}_{l\in L}\vdash a.k;P :: c : C}\ (\&L_k)$$

$$\mathsf{msg}(a,a.k;a \leftarrow d),\mathsf{proc}(a,\mathsf{case}\ a\ \{l \Rightarrow P_l\}_{l\in L}) \to \mathsf{proc}(d,[d/a]P_k) \tag{10}$$

$$\mathsf{proc}(c,a.k;P),\mathbf{type}(a : \oplus\{l : A_l\}_{l\in L}) \to \exists d.\mathsf{msg}(a,a.k;a \leftarrow d),\mathsf{proc}(c,[d/a]P),\mathbf{type}(d : A_k) \tag{11}$$

A communication of type $\rho\alpha.A$ is an unfold message followed by a communication of type $[\rho\alpha.A/\alpha]A$. The process send a unfold$;P$ sends an unfold message and continues as P. The process unfold $\leftarrow$ recv $a;P$

blocks until it receives the unfold message on a and continues as P.

$$\frac{\Pi \,;\Delta \vdash P :: a : [\rho\alpha.A/\alpha]A}{\Pi \,;\Delta \vdash \mathsf{send}\ a\ \mathsf{unfold};P :: a : \rho\alpha.A}\ (\rho^+R) \qquad \frac{\Pi \,;\Delta, a : [\rho\alpha.A/\alpha]A \vdash P :: c : C}{\Pi \,;\Delta, a : \rho\alpha.A \vdash \mathsf{unfold} \leftarrow \mathsf{recv}\ a;P :: c : C}\ (\rho^+L)$$

$$\mathsf{proc}(a, \mathsf{send}\ a\ \mathsf{unfold};P), \mathbf{type}(a : \rho\alpha.A) \rightarrow$$
$$\exists d.\mathsf{msg}(a, \mathsf{send}\ a\ \mathsf{unfold}; a \leftarrow d), \mathsf{proc}(d, [d/a]P), \mathbf{type}(d : [\rho\alpha.A/\alpha]A) \tag{12}$$

$$\mathsf{msg}(a, \mathsf{send}\ a\ \mathsf{unfold}; a \leftarrow d), \mathsf{proc}(c, \mathsf{unfold} \leftarrow \mathsf{recv}\ a;P) \rightarrow \mathsf{proc}(c, [d/a]P) \tag{13}$$

   Finally, recursive processes are formed in the standard way. The SSOS is only defined on closed processes, so there are no rules for process variables. Recursive processes step by unfolding.

$$\frac{}{\Pi, p : \{c : C \leftarrow \Delta\} \,;\Delta \vdash p :: c : C}\ (\text{VAR}) \qquad \frac{\Pi, p : \{c : C \leftarrow \Delta\} \,;\Delta \vdash P :: c : C}{\Pi \,;\Delta \vdash \mathsf{fix}\ p.P :: c : C}\ (\text{REC})$$

$$\mathsf{proc}(c, \mathsf{fix}\ p.P) \rightarrow \mathsf{proc}(c, [\mathsf{fix}\ p.P/p]P) \tag{14}$$

Example 4. *The protocol* $\mathsf{conat} = \rho\alpha.(\mathsf{z} : \mathbf{1}) \oplus (\mathsf{s} : \alpha)$ *encodes conatural numbers. Indeed, a communication is either an infinite sequence of successor labels* s, *or some finite number of* s *labels followed by the zero label* z *and termination. The following process receives a conatural number i and outputs its increment on o:*

$$\cdot \,; i : \mathsf{conat} \vdash \mathsf{send}\ o\ \mathsf{unfold}; s.o; o \rightarrow i :: o : \mathsf{conat}.$$

It works by outputting a successor label on o, and then forwarding the conatural number i to o. It has the following fair execution, where we elide $\mathbf{type}(c : A)$ *judgments and annotations on the arrows:*

$$\mathsf{proc}(o, \mathsf{send}\ o\ \mathsf{unfold}; s.o; o \rightarrow i) \longrightarrow \mathsf{msg}(c, \mathsf{send}\ o\ \mathsf{unfold}; o \leftarrow o_1), \mathsf{proc}(o_1, s.o_1; o_1 \rightarrow i) \longrightarrow$$
$$\mathsf{msg}(o, \mathsf{send}\ o\ \mathsf{unfold}; o \leftarrow o_1), \mathsf{msg}(o_1, s.o_1; s \leftarrow o_2), \mathsf{proc}(o_2, o_2 \leftarrow i).$$

The following recursive process outputs the infinite conatural number $s(s(s(\cdots)))$ *on o:*

$$\cdot \,; \cdot \vdash \mathsf{fix}\ \omega.\mathsf{send}\ o\ \mathsf{unfold}; s.o; \omega :: o : \mathsf{conat}.$$

It has an infinite fair execution where for $n \geq 1$, *the rules* r_{3n-2}, r_{3n-1}, *and* r_{3n} *are respectively instantiations of* (14), (12), *and* (8).

3.2   Properties of Process Traces

Let $\mathscr{P}$ be MRS given by the above rules. We prove various invariants maintained by process traces.

   Let $\mathsf{fc}(P)$ be the set of free channel names in P. The following result follows by an induction on n and a case analysis on the rule used in the last step:

Proposition 8. *Let* $T = (M_0, (r_i; \delta_i)_i)$ *be a process trace. For all n, if* $\mathsf{proc}(c_0, P) \in M_n$, *then*

   1. $c_0 \in \mathsf{fc}(P)$;

   2. *for all* $c_i \in \mathsf{fc}(P)$, *there exists an* A_i *such that* $\mathbf{type}(c_i : A_i) \in M_n$; *and*

   3. *where* $\mathsf{fc}(P) = \{c_0, \ldots, c_m\}$, *we have* $\cdot \,; c_1 : A_1, \ldots, c_m : A_m \vdash P :: c_0 : A_0$.

If $\mathsf{msg}(c, m) \in M_n$, *then*

   • *if* $m = \mathsf{msg}(c, *)$, *then* $\mathbf{type}(c : \mathbf{1}) \in M_n$;

- *if $m = c.l_j; c \leftarrow d$, then either* $\mathbf{type}(c : \oplus\{l_i : A_i\}_{i \in I}) \in M_n$ *or* $\mathbf{type}(c : \&\{l_i : A_i\}_{i \in I}) \in M_n$ *for some A_i ($i \in I$), and* $\mathbf{type}(d : A_j) \in M_n$ *for some $j \in I$.*

- *if $m = $ send c a; $c \leftarrow b$, then* $\mathbf{type}(c : A \otimes B), \mathbf{type}(a : A), \mathbf{type}(b : B) \in M_n$ *for some A and B;*

- *if $m = $ send c unfold; $c \leftarrow d$, then* $\mathbf{type}(c : \rho\alpha.A), \mathbf{type}(d : [\rho\alpha.A/\alpha]A) \in M_n$ *for some $\rho\alpha.A$.*

The MRS $\mathscr{P}$ differs from the usual MRSs given for this style session-typed languages [13, 20, 25] in the addition of $\mathbf{type}(c : A)$ judgments. Corollary 3 shows that their addition does not change the operational behaviour of the semantics. Let $|M|$, $|\mathscr{P}|$, $|T|$, etc., be the result of erasing all $\mathbf{type}(c : A)$ judgments.

Corollary 3. *Consider a process $\cdot \, ; \Delta \vdash P :: c : A$ with initial state M_0. If T is a trace from M_0 under $\mathscr{P}$, then $|T|$ is a trace from $|M_0|$ under $|\mathscr{P}|$. If T is a trace from $|M_0|$ under $|\mathscr{P}|$, then there exists a trace T' from M_0 under $\mathscr{P}$ such that $|T'| = T$.*

Proposition 8 showed that there were enough $\mathbf{type}(c : A)$ judgments in a trace. Proposition 9 shows that there are not too many:

Proposition 9. *Let $(M_0, (r_i; \delta_i)_i)$ be a process trace. For all channels c and all $i, j \geq 0$, if $\mathbf{type}(c : A_i)$ appears in M_i and $\mathbf{type}(c : A_j)$ appears in M_j, then $A_i = A_j$.*

We show an analogous uniqueness result for $\mathsf{msg}(c, m)$ judgments. It implies that each channel name in an execution carries at most one message. To prove it, we begin by partitioning a process's free channels into "input" and "output" channels and show that at all times, a channel is an output channel of at most one process. Given a process P, let $\mathrm{oc}(P)$ be the subset of $\mathrm{fc}(P)$ recursively defined by:

$$\mathrm{oc}(a \to b) = \{b\} \qquad\qquad \mathrm{oc}(a \leftarrow P; \; Q) = (\mathrm{oc}(P) \cup \mathrm{oc}(Q)) \setminus \{a\}$$
$$\mathrm{oc}(\mathsf{close}\; a) = \{a\} \qquad\qquad \mathrm{oc}(\mathsf{wait}\; a; P) = \mathrm{oc}(P)$$

$$\mathrm{oc}(a.k; P) = \{a\} \cup \mathrm{oc}(P) \qquad \mathrm{oc}(\mathsf{case}\; a\; (l \Rightarrow P_l)_{l \in L}) = \left(\bigcup_{l \in L} \mathrm{oc}(P_l) \right) \setminus \{a\}$$

$$\mathrm{oc}(\mathsf{send}\; a\; b; \; P) = \{a\} \cup \mathrm{oc}(P) \qquad \mathrm{oc}(b \leftarrow \mathsf{recv}\; a; \; P) = \mathrm{oc}(P) \setminus \{a, b\}$$
$$\mathrm{oc}(\mathsf{send}\; a\; \mathsf{unfold}; P) = \{a\} \cup \mathrm{oc}(P) \qquad \mathrm{oc}(\mathsf{unfold} \leftarrow \mathsf{recv}\; a; P) = \mathrm{oc}(P) \setminus \{a\}$$
$$\mathrm{oc}(p) = \emptyset \qquad\qquad \mathrm{oc}(\mathsf{fix}\; p.P) = \mathrm{oc}(P)$$

Intuitively, $c \in \mathrm{oc}(P)$ if the next time P communicates on c, P sends a message on c. Given a configuration $\mathscr{C}$, let $\mathrm{oc}(\mathscr{C})$ be the union of the sets $\mathrm{oc}(P)$ for $\mathrm{proc}(c, P)$ in $\mathscr{C}$. Analogously, let $\mathrm{ic}(P)$ and $\mathrm{ic}(\mathscr{C})$ be the set of input channels of P and of $\mathscr{C}$.

Lemma 5. *If $F(\vec{k}) \xrightarrow{(r; (\vec{k}, \vec{a}))} G(\vec{k}, \vec{a})$ by a rule r of section 3.1, then*

- *if $\mathsf{msg}(c, m) \in F(\vec{k})$, then $c \in \mathrm{ic}(F(\vec{k}))$;*

- *if $\mathsf{msg}(c, m) \in G(\vec{k}, \vec{a})$, then $c \in \mathrm{oc}(F(\vec{k}))$;*

- *if $\mathsf{msg}(c, m; c \leftarrow d) \in G(\vec{k}, \vec{a})$, then $d \in \vec{a}$ and $d \in \mathrm{fc}(G(\vec{k}, \vec{a}))$; and*

- *$\mathrm{oc}(G(\vec{k}, \vec{a})) \subseteq \mathrm{oc}(F, \vec{k}) \cup \vec{a}$ and $\mathrm{ic}(G(\vec{k}, \vec{a})) \subseteq \mathrm{ic}(F, \vec{k}) \cup \vec{a}$.*

Proof. Immediate by a case analysis on the rules. □

An induction with lemma 5 implies the desired disjointness result:

Lemma 6. *Let $(M_0, (r_i; \delta_i)_i)$ be a process trace. For all n, if $\mathsf{proc}(c, P)$ and $\mathsf{proc}(d, Q)$ appear in M_n, then $\mathrm{oc}(P) \cap \mathrm{oc}(Q) = \emptyset$ and $\mathrm{ic}(P) \cap \mathrm{ic}(Q) = \emptyset$.*

The following lemma shows that processes do not send messages on channels c already associated with a $\mathsf{msg}(c, m)$ judgment:

Lemma 7. *Let $(M_0, (r_i; \delta_i)_i)$ be a process trace. For all $n \leq k$, if $\mathsf{msg}(c, m) \in M_n$ and $\mathsf{proc}(d, P) \in M_k$, then $c \notin \mathrm{oc}(P)$.*

The desired result then follows by induction and the above results:

Corollary 4. *Let $(M_0, (r_i; \delta_i)_i)$ be a process trace. For all channels c and all $i, j \geq 0$, if $\mathsf{msg}(c, m_i)$ appears in M_i and $\mathsf{msg}(c, m_j)$ appears in M_j, then $m_i = m_j$.*

We now turn our attention to showing that all well-typed, closed processes have fair executions. This fact will follow easily from the following proposition:

Proposition 10. *The MRS $\mathscr{P}$ is non-overlapping from the initial configuration of $\cdot \,; \Delta \vdash P :: c : A$ for all $\cdot \,; \Delta \vdash P :: c : A$.*

Proof. Consider a trace $(M_0, (r_i; (\theta_i, \xi_i)))$ from the initial configuration of $\cdot \,; \Delta \vdash P :: c : A$ and some arbitrary n. It is sufficient to show that if $s_1(\phi_1)$ and $s_2(\phi_2)$ are distinct instantiations applicable to M_n, then $F_1(\phi_1)$ and $F_2(\phi_2)$ are disjoint multisets: $F_1(\phi_1) \cap F_2(\phi_2) = \emptyset$. Indeed, if this is the case and $s_1(\phi_1), \ldots, s_k(\phi_k)$ are the distinct rule instantiations applications to M_n, then $F_1(\theta_1), \ldots, F_k(\phi_k) \subseteq M_n$, so $\Omega_{M_n}(F_1(\phi_1), \ldots, F_k(\phi_k)) = \emptyset$.

We proceed by case analysis on the possible judgments in $F_1(\phi_1) \cap F_2(\phi_2)$.

Case $\mathsf{msg}(c, m)$. Then $c \in \mathrm{ic}(F_1(\phi_1))$ and $c \in \mathrm{ic}(F_2(\phi_2))$ by lemma 5. This is a contradiction by lemma 6.

Case $\mathsf{proc}(c, P)$. Then $s_1 = s_2$ by a case analysis on the rules. We show that $\phi_1 = \phi_2$. If s_1 is one of (2) to (6), (8), (11), (12) and (14), then we have $\phi_1 = \phi_2$, because all constants matched by ϕ_1 and ϕ_2 appear in $\mathsf{proc}(c, P)$. If s_1 is one of (7), (9), (10) and (13), then $F_i(\phi_i)$ contain a judgment $\mathsf{msg}(d, m_i)$ where there is a constant $e_i \in m_i$ that appears in ϕ_i, but not in $\mathsf{proc}(c, P)$ (explicitly, e_i is the name of the continuation channel). By corollary 4, $m_1 = m_2$, so $e_1 = e_2$. All other channel names in ϕ_i appear in $\mathsf{proc}(c, P)$, so $\phi_1 = \phi_2$. So $s_1(\phi_1)$ and $s_2(\phi_2)$ are not distinct rule instantiations, a contradiction.

Case $\mathsf{type}(c : A)$. By case analysis on the rules, $s_1 = s_2$ and there exist judgments $\mathsf{proc}(d_i, P_i) \in F_i(\phi_i)$. Suppose to the contrary that $P_1 \neq P_2$. By case analysis on the rules, s_1 is one of (6), (8), (11) and (12). This implies that $c \in \mathrm{oc}(P_1) \cap \mathrm{oc}(P_2)$, a contradiction of lemma 6. So $P_1 = P_2$. Because all constants in ϕ_1 and ϕ_2 appear in P_1, we conclude that $\phi_1 = \phi_2$. So $s_1(\phi_1)$ and $s_2(\phi_2)$ are not distinct rule instantiations, a contradiction. $\square$

Corollary 5. *Every process $\cdot \,; \Delta \vdash P :: c : A$ has a fair execution. Its fair executions are all permutations of each other and they are all union-equivalent.*

Proof. By proposition 10, $\mathscr{P}$ is non-overlapping from the initial configuration M_0 of $\cdot \,; \Delta \vdash P :: c : A$. It is then interference-free from M_0 by proposition 5, so a fair execution exists by proposition 3. All of its fair executions are permutations of each other by proposition 7. They are union-equivalent by corollary 2. $\square$

4   Observed Communications

Consider a closed process $\cdot \, ; c_1 : A_1, \ldots, c_n : A_n \vdash P :: c_0 : A_0$. In this section, we will define the observation of P to be a tuple $(c_i : v_i)_{0 \leq i \leq n}$, where v_i is the communication of type A_i observed on channel c_i in a fair execution of P. We extract communications from fair executions using a coinductively defined judgment. We colour-code the modes of judgments, where inputs to a judgment are in blue and outputs are in red.

We begin by defining session-typed communications. Let a communication v be a (potentially infinite) tree generated by the following grammar, where k and l_i range over labels. We explain these communications v below when we associate them with session types. For convenience, we also give a grammar generating the session types A of section 3.1. Session types are always finite expressions, and we treat $\rho \alpha . A$ as a binding operator.

$$v, v' := \bot_A \mid * \mid (k, v) \mid (v, v') \mid (\mathsf{unfold}, v)$$
$$A, A_i, B := \alpha \mid \mathbf{1} \mid A \otimes B \mid \oplus(l_1 : A_1, \ldots, l_n : A_n) \mid \&(l_1 : A_1, \ldots, l_n : A_n) \mid \rho \alpha . A.$$

As in section 3.1, we abbreviate $\oplus(l_1 : A_1, \ldots, l_n : A_n)$ and $\&(l_1 : A_1, \ldots, l_n : A_n)$ by $\oplus\{l : A_l\}_{l \in L}$ and $\&\{l : A_l\}_{l \in L}$, respectively, where L is the finite set of labels.

Next, we associate communications with session types. The judgment $v \, \varepsilon \, A$ means that the syntactic communication v has type A. It is coinductively defined by the following rules, where A is assumed to have no unbound occurrences of α. The rule forming $(k, v_k) \, \varepsilon \, \oplus\{l : A_l\}_{l \in L}$ has the side condition $k \in L$.

$$\frac{}{\bot_\mathbf{1} \, \varepsilon \, \mathbf{1}} \qquad \frac{}{* \, \varepsilon \, \mathbf{1}} \qquad \frac{}{\bot_{A \otimes B} \, \varepsilon \, A \otimes B} \qquad \frac{v \, \varepsilon \, A \quad v' \, \varepsilon \, B}{(v, v') \, \varepsilon \, A \otimes B} \qquad \frac{}{\bot_{\rho \alpha . A} \, \varepsilon \, \rho \alpha . A} \qquad \frac{v \, \varepsilon \, [\rho \alpha . A / \alpha]A}{(\mathsf{unfold}, v) \, \varepsilon \, \rho \alpha . A}$$

$$\frac{}{\bot_{\oplus\{l : A_l\}_{l \in L}} \, \varepsilon \, \oplus\{l : A_l\}_{l \in L}} \qquad \frac{v_k \, \varepsilon \, A_k}{(k, v_k) \, \varepsilon \, \oplus\{l : A_l\}_{l \in L}} \qquad \frac{}{\bot_{\&\{l : A_l\}_{l \in L}} \, \varepsilon \, \&\{l : A_l\}_{l \in L}} \qquad \frac{v_k \, \varepsilon \, A_k}{(k, v_k) \, \varepsilon \, \&\{l : A_l\}_{l \in L}}$$

Every closed session type A has an empty communication $\bot_A$ representing the absence of communication of that type. The communication $*$ represents the close message. A communication of type $\oplus\{l : A_l\}_{l \in L}$ or $\oplus\{l : A_l\}_{l \in L}$ is a label $k \in L$ followed by a communication v_k of type A_k, whence the communication (k, v_k). Though by itself the communication (k, v_k) does not capture the direction in which the label k travelled, this poses no problem to our development: we never consider communications without an associated session type, and the polarity of the type specifies the direction in which k travels. We cannot directly observe channels, but we can observe communications over channels. Consequently, we observe a communication of type $A \otimes B$ as a pair (v, v') of communications v of type A and v' of type B. A communication of type $\rho \alpha . A$ is an unfold message followed by a communication of type $[\rho \alpha . A / \alpha]A$.

Given a trace $T = (M_0, (r_i; (\theta_i, \xi_i))_i)$, we write $\mathscr{T}$ for the set-theoretic union of the M_i, that is, $x \in \mathscr{T}$ if and only if $x \in \mathrm{supp}(M_i)$ for some i. Write $T \vdash c : A$ if $\mathbf{type}(c : A) \in \mathscr{T}$. This judgment is defined on all channel names c that appear in T by proposition 8 and it is a function by proposition 9.

Assuming the channel c appears in T, the judgment $T \rightsquigarrow v \, \varepsilon \, A \, / \, c$ means that we observed a communication v of type A on the channel c during T. We will show below that whenever $T \rightsquigarrow v \, \varepsilon \, A \, / \, c$, we also have $T \vdash c : A$ and $v \, \varepsilon \, A$. Fixing T, the judgment $T \rightsquigarrow v \, \varepsilon \, A \, / \, c$ is coinductively defined by the following rules, i.e., $T \rightsquigarrow v \, \varepsilon \, A \, / \, c$ is the largest set of triples (v, c, A) closed under the following rules.

We observe no communications on a channel c if and only if $\mathsf{msg}(c, m)$ does not appear in the trace for any m. Subject to the side condition that for all m, $\mathsf{msg}(c, m) \notin \mathscr{T}$, we have the rule

$$\frac{T \vdash c : A}{T \rightsquigarrow \bot_A \, \varepsilon \, A \, / \, c} \; (\mathrm{O}\text{-}\bot)$$

We observe a close message on c if and only if the close message was sent on c:

$$\frac{\mathsf{msg}(c, *) \in \mathscr{T}}{T \rightsquigarrow * \; \varepsilon \; \mathbf{1} \, / \, c} \; \text{(O-1)}$$

We observe label transmission as labelling communications on the continuation channel. We rely on the judgment $T \vdash c : \oplus\{l : A_l\}_{l \in L}$ or $T \vdash c : \&\{l : A_l\}_{l \in L}$ to determine the type of c:

$$\frac{\mathsf{msg}(c, c.l; c \leftarrow d) \in \mathscr{T} \quad T \rightsquigarrow v \; \varepsilon \; A_l \, / \, d \quad T \vdash c : \oplus\{l : A_l\}_{l \in L}}{T \rightsquigarrow (l, v) \; \varepsilon \; \oplus\{l : A_l\}_{l \in L} \, / \, c} \; \text{(O-$\oplus$)}$$

$$\frac{\mathsf{msg}(c, c.l; c \leftarrow d) \in \mathscr{T} \quad T \rightsquigarrow v \; \varepsilon \; A_l \, / \, d \quad T \vdash c : \&\{l : A_l\}_{l \in L}}{T \rightsquigarrow (l, v) \; \varepsilon \; \&\{l : A_l\}_{l \in L} \, / \, c} \; \text{(O-\&)}$$

As described above, we observe channel transmission as pairing of communications:

$$\frac{\mathsf{msg}(c, \mathsf{send} \; c \; a; \; c \leftarrow d) \in \mathscr{T} \quad T \rightsquigarrow u \; \varepsilon \; A \, / \, a \quad T \rightsquigarrow v \; \varepsilon \; B \, / \, d}{T \rightsquigarrow (u, v) \; \varepsilon \; A \otimes B \, / \, c} \; \text{(O-$\otimes$)}$$

Finally, we observe the unfold message as an unfold message:

$$\frac{\mathsf{msg}(c, \mathsf{send} \; c \; \mathsf{unfold}; c \leftarrow d) \in \mathscr{T} \quad T \rightsquigarrow v \; \varepsilon \; [\rho\alpha.A/\alpha]A \, / \, d}{T \rightsquigarrow (\mathsf{unfold}, v) \; \varepsilon \; \rho\alpha.A \, / \, c} \; \text{(O-ρ)}$$

The following three propositions imply that for any T, $T \rightsquigarrow v \; \varepsilon \; A \, / \, c$ is a total function from channel names c in T to session-typed communications $v \; \varepsilon \; A$.

Proposition 11. *If $T \rightsquigarrow v \; \varepsilon \; A \, / \, c$, then $v \; \varepsilon \; A$.*

Proof. Immediate by rule coinduction. □

Proposition 12. *If T is a process trace, then for all c, if $T \vdash c : A$, then $T \rightsquigarrow v \; \varepsilon \; A \, / \, c$ for some v.*

Proof (Sketch). Let S be the set of all triples (v, A, c) for session-typed communications $v \; \varepsilon \; A$ and channel names c. Let $\Phi : \wp(S) \rightarrow \wp(S)$ be the rule functional defining $T \rightsquigarrow v \; \varepsilon \; A \, / \, c$. Then the judgment $T \rightsquigarrow v \; \varepsilon \; A \, / \, c$ is given by the greatest fixed point $\mathrm{gfp}(\Phi)$ of Φ on the complete lattice $\wp(S)$, where $T \rightsquigarrow v \; \varepsilon \; A \, / \, c$ if and only if $(v, A, c) \in \mathrm{gfp}(\Phi)$. The functional Φ is cocontinuous by [23, Theorem 2.9.4], so $\mathrm{gfp}(\Phi) = \bigcap_{n \geq 0} \Phi^n(S)$ by [23, Theorem 2.8.5]. It is sufficient to show that if $T \vdash c : A$, then there exists a v such that $(c, v, A) \in \Phi^n(S)$ for all n. This v can be constructed using a coinductive argument and a case analysis on $\mathsf{msg}(c, m) \in \mathscr{T}$. □

Proposition 13. *If T is a trace from the initial configuration of a process, then for all c, if $T \rightsquigarrow v \; \varepsilon \; A \, / \, c$ and $T \rightsquigarrow w \; \varepsilon \; B \, / \, c$, then $v = w$ and $A = B$.*

Proof (Sketch). Let $R = \{(T \rightsquigarrow v \; \varepsilon \; A \, / \, c, T \rightsquigarrow w \; \varepsilon \; B \, / \, c) \mid \exists v, w, c, A, B.T \rightsquigarrow v \; \varepsilon \; A \, / \, c \wedge T \rightsquigarrow w \; \varepsilon \; B \, / \, c\}$. We claim that R is a bisimulation. Indeed, let $(T \rightsquigarrow v \; \varepsilon \; A \, / \, c, T \rightsquigarrow w \; \varepsilon \; B \, / \, c) \in R$ be arbitrary. By corollary 4, at most one rule is applicable to form a judgment of the form $T \rightsquigarrow u \; \varepsilon \; C \, / \, c$ (with c fixed), so $T \rightsquigarrow v \; \varepsilon \; A \, / \, c$ and $T \rightsquigarrow w \; \varepsilon \; B \, / \, c$ were both formed by the same rule. A case analysis shows on this rule shows that R satisfies the definition of a bisimulation.

Consider arbitrary $T \rightsquigarrow v \; \varepsilon \; A \, / \, c$ and $T \rightsquigarrow w \; \varepsilon \; B \, / \, c$. They are related by R, so they are bisimilar. By [14, Theorem 2.7.2], bisimilar elements of the terminal coalgebra are equal, so $v = w$ and $A = B$. □

Corollary 6 gives the converse of proposition 12:

Corollary 6. *If T is a process trace, then for all c, if $T \leadsto v \, \varepsilon \, A \, / \, c$, then $T \vdash c : A$.*

Proof. We show by case analysis on the rules that if $T \leadsto v \, \varepsilon \, A \, / \, c$, then $T \vdash c : B$ for some B. The case (O-$\bot$) is obvious, while for each other case, if $T \leadsto v \, \varepsilon \, A \, / \, c$, then $\mathsf{msg}(c, m) \in \mathscr{T}$ for some m. For each of these cases, proposition 8 implies **type**$(c : B) \in \mathscr{T}$ for some B, i.e., $T \vdash c : B$.

Assume $T \leadsto v \, \varepsilon \, A \, / \, c$. By the claim, $T \vdash c : B$ for some B. By proposition 12, there exists a w such that $T \leadsto w \, \varepsilon \, B \, / \, c$. By proposition 13, $A = B$, so $T \vdash c : A$. □

Theorem 1. *Let T be a fair execution of $\cdot \, ; c_1 : A_1, \ldots, c_n : A_n \vdash P :: c_0 : A_0$. For all $0 \le i \le n$, there exist unique v_i such that $v_i \, \varepsilon \, A_i$ and $T \leadsto v_i \, \varepsilon \, A_i \, / \, c_i$.*

Proof. By definition of fair execution, we have **type**$(c_i : A_i) \in \mathscr{T}$ for all $0 \le i \le n$, i.e., $T \vdash c_i : A_i$ for all $0 \le i \le n$. By proposition 12, for all $0 \le i \le n$, there exists a v_i such that $T \leadsto v_i \, \varepsilon \, A_i \, / \, c_i$, and $v_i \, \varepsilon \, A_i$ by proposition 11. Each v_i is unique by proposition 13. □

The following theorem captures the confluence property typically enjoyed by SILL-style languages:

Theorem 2. *Let T and T' be a fair executions of $\cdot \, ; c_1 : A_1, \ldots, c_n : A_n \vdash P :: c_0 : A_0$. For all $0 \le i \le n$, if $T \leadsto v_i \, \varepsilon \, A_i \, / \, c_i$ and $T' \leadsto w_i \, \varepsilon \, A_i \, / \, c_i$, then $v_i = w_i$.*

Proof. Assume $T \leadsto v_i \, \varepsilon \, A_i \, / \, c_i$ and $T' \leadsto w_i \, \varepsilon \, A_i \, / \, c_i$. By corollary 5, traces T and T' are union-equivalent, i.e., $\mathscr{T} = \mathscr{T}'$. It immediately follows that $T' \leadsto w_i \, \varepsilon \, A_i \, / \, c_i$ if and only if $T \leadsto w_i \, \varepsilon \, A_i \, / \, c_i$. So $v_i = w_i$ by proposition 13. □

We use theorems 1 and 2 to define the **operational observation** $(\!| \cdot \, ; c_1 : A_1, \ldots, c_n : A_n \vdash P :: c_0 : A_0 |\!)$ of $\cdot \, ; c_1 : A_1, \ldots, c_n : A_n \vdash P :: c_0 : A_0$. It is the tuple of observed communications

$$(\!| \cdot \, ; c_1 : A_1, \ldots, c_n : A_n \vdash P :: c_0 : A_0 |\!) = (c_0 : v_0, \ldots, c_n : v_n)$$

where $T \leadsto v_i \, \varepsilon \, A_i \, / \, c_i$ for $0 \le i \le n$ for some fair execution T of $\cdot \, ; c_1 : A_1, \ldots, c_n : A_n \vdash P :: c_0 : A_0$. Such a T exists by corollary 5, and $(\!| \cdot \, ; c_1 : A_1, \ldots, c_n : A_n \vdash P :: c_0 : A_0 |\!)$ does not depend on the choice of T by theorem 2. The v_i such that $T \leadsto v_i \, \varepsilon \, A_i \, / \, c_i$ exist by proposition 9, and they are unique by proposition 13.

Uniqueness of operational observations and theorem 2 crucially depend on fairness. Indeed, without fairness a process can have infinitely many observations. To see this, let Ω and B respectively be given by

$$\cdot \, ; \cdot \vdash \mathsf{fix} \, \omega.\omega :: a : \mathbf{1}$$
$$\cdot \, ; a : \mathbf{1} \vdash \mathsf{fix} \, p.\mathsf{send} \, b \, \mathsf{unfold}; b.l; p :: b : \rho\beta. \oplus \{l : \beta\}$$

Rule (3) is the first step of any execution of their composition $\cdot \, ; \cdot \vdash a : \mathbf{1} \leftarrow \Omega; B :: b : \rho\beta. \oplus \{l : \beta\}$. It spawns Ω and B as separate processes. Without fairness, an execution could then consist exclusively of applications of rule (14) to Ω. This would give the observation $(b : \bot)$. Alternatively, B could take finitely many steps, leading to observations where b is a tree of correspondingly finite height. Fairness ensures that B and Ω both take infinitely many steps, leading to the unique observation $(b : (\mathsf{unfold}, (l, (\mathsf{unfold}, \ldots))))$.

Operational observation does not take into account the order in which a process sends on channels. For example, the following processes have the same operational observation $(a : (l, \bot_{\mathbf{1}}), b : (r, \bot_{\mathbf{1}}))$, even though they send on a and on b in different orders:

$$\cdot \, ; a : \&\{l : \mathbf{1}\} \vdash a.l; b.r; a \to b :: b : \oplus\{r : \mathbf{1}\}$$
$$\cdot \, ; a : \&\{l : \mathbf{1}\} \vdash b.r; a.l; a \to b :: b : \oplus\{r : \mathbf{1}\}.$$

The order in which channels are used does not matter for several reasons. First, messages are only ordered on a per-channel basis, and messages sent on different channels can arrive out of order. Second, each channel has a unique pair of endpoints, and the (CUT) rule organizes processes in a tree-like structure. This means that two processes communicating with a process R cannot at the same time also directly communicate with each other to compare the order in which R sent them messages. In other words, the ordering cannot be distinguished by other processes.

Our notion of operational observation scales to support language extensions. Indeed, for each new session type one first defines its corresponding session-typed communications. Then, one specifies how to observe message judgments $\mathsf{msg}(c,m)$ in a trace as communications. Informally, it seems desirable to ensure that if two message judgments $\mathsf{msg}(c,m)$ can be distinguished by a receiving process, then they are observed as different session-typed communications.

A **typed context** $\cdot\,; \Delta \vdash C[\cdot]_{a:A}^{\Delta'} :: b : B$ is a context derived using the process typing rules of section 3.1, plus exactly one instance of the axiom

$$\frac{}{\cdot\,; \Delta' \vdash [\cdot]_{a:A}^{\Delta'} :: a : A} \ (\text{HOLE})$$

Given a context $\cdot\,; \Delta \vdash C[\cdot]_{a:A}^{\Delta'} :: b : B$ and a process $\cdot\,; \Delta' \vdash P :: a : A$, we let $\cdot\,; \Delta \vdash C[P] :: b : B$ be the result of "plugging" P into the hole, that is, of replacing the axiom (HOLE) by the derivation $\Delta' \vdash P :: a : A$ in the derivation $\Delta \vdash C[\cdot]_{a:A}^{\Delta'} :: b : B$.

We say that processes $\cdot\,; \Delta \vdash P :: c : C$ and $\cdot\,; \Delta \vdash Q :: c : C$ are **observationally congruent**, $P \approx Q$, if $(\!|\,\cdot\,; \Delta' \vdash C[P] :: b : B\,|\!) = (\!|\,\cdot\,; \Delta' \vdash C[Q] :: b : B\,|\!)$ for all typed contexts $\cdot\,; \Delta' \vdash C[\cdot]_{c:C}^{\Delta} :: b : B$. Intuitively, this means that no context C can differentiate processes P and Q.

To illustrate observational congruence, we show that process composition is associative:

Proposition 14. *We have* $c_1 : C_1 \leftarrow P_1; \ (c_2 : C_2 \leftarrow P_2; \ P_3) \approx c_2 : C_2 \leftarrow (c_1 : C_1 \leftarrow P_1; \ P_2); \ P_3$ *for all* $\cdot\,; \Delta_1 \vdash P_1 :: c_1 : C_1$, *all* $\cdot\,; c_1 : C_1, \Delta_2 \vdash P_2 :: c_2 : C_2$, *and all* $\cdot\,; c_2 : C_2, \Delta_3 \vdash P_3 :: c_3 : C_3$.

Proof (Sketch). Let $L = c_1 : C_1 \leftarrow P_1; \ (c_2 : C_2 \leftarrow P_2; \ P_3)$ and $R = c_2 : C_2 \leftarrow (c_1 : C_1 \leftarrow P_1; \ P_2); \ P_3$. Consider some arbitrary observation context $C[\cdot]$ and a fair execution T of $C[L]$. It is sufficient to show that T agrees on message judgments with a fair execution $C[R]$. Union-equivalence of process traces is invariant under permutation, so we can assume without loss of generality that whenever $\mathsf{proc}(c_3, L)$ appears in some M_n of T, then the next two steps are applications (3) to decompose L:

$$\mathsf{proc}(c_3, L) \longrightarrow \mathsf{proc}(c_1', [c_1'/c_1]P_1), \mathsf{proc}(c_3, [c_1'/c_1](c_2 : C_2 \leftarrow P_2; \ P_3)) \longrightarrow$$
$$\mathsf{proc}(c_1', [c_1'/c_1]P_1), \mathsf{proc}(c_2', [c_1', c_2'/c_1, c_2]P_2), \mathsf{proc}(c_3', [c_2'/c_2]P_3)$$

(For conciseness, we elide the **type**$(c : A)$ judgments.) There exists a fair execution T' of $C[R]$ that agrees with T on all steps, except for those involving R, where we make the same assumption:

$$\mathsf{proc}(c_3, R) \longrightarrow \mathsf{proc}(c_2', [c_2'/c_2](c_1 : C_1 \leftarrow P_1; \ P_2)), \mathsf{proc}(c_3, [c_2'/c_2]P_3) \longrightarrow$$
$$\mathsf{proc}(c_1', [c_1'/c_1]P_1), \mathsf{proc}(c_2', [c_1', c_2'/c_1, c_2]P_2), \mathsf{proc}(c_3', [c_2'/c_2]P_3)$$

So traces T and T' agree on all message judgments, whence $(\!|C[L]|\!) = (\!|C[R]|\!)$. $\qquad\square$

5 Related Work

Multiset rewriting systems with existential quantification were first introduced by Cervesato et al. [8]. They were used to study security protocols and were identified as the first-order Horn fragment of linear

logic. Since, MRSs have modelled other security protocols, and strand spaces [7, 9]. Cervesato and Scedrov [10] studied the relationship between MRSs and linear logic. These works do not explore fairness.

Weak and strong fairness were first introduced by Apt and Olderog [1] and Park [18] in the context of do-od languages, and were subsequently adapted to process calculi, e.g., by Costa and Stirling [11] for Milner's CCS. Our novel notion of fairness for multiset rewriting systems in section 2 implies strong process fairness (so also weak process fairness) for the session-typed processes of section 3. We conjecture that this notion of fairness is stronger than required for many applications. In future work, we intend to explore other formulations of fairness for MRSs and their impact on applications.

Substructural operational semantics [24] based on multiset rewriting are widely used to specify the operational behaviour of session-typed languages arising from proofs-as-processes interpretations of linear logic and adjoint logic. Examples include functional languages with session-typed concurrency [25], languages with run-time monitoring [13], message-passing interpretations of adjoint logic [22], and session-typed languages with sharing [3]. The fragment of section 3.1 illustrates some of the key ideas of this approach, and extends to these richer settings.

Some of these languages are already equipped with observational equivalences. For example, Pérez et al. [19] introduced *typed context bisimilarity*, a labelled bisimilarity for session-typed processes. It does not support recursive processes or recursive session types. Toninho [27] explored barbed congruence for session-typed processes and shows that it coincides with logical equivalence. Kokke, Montesi, and Peressotti [16] showed that the usual notions of bisimilarity and barbed congruence carry over from the π-calculus. They also gave a denotational semantics using Brzozowski derivatives to "hypersequent classical processes" that built on Atkey's denotational semantics for CP, and showed that all three notions of equivalence agreed on well-typed programs. In future work, we intend to show that our observational congruence agrees with barbed congruence. Gommerstadt, Jia, and Pfenning [13] define a bisimulation-style observational equivalence on multisets in process traces. It deems two configurations equivalent if whenever both configurations send an externally visible message, then the messages are equivalent. It is easy to adapt this bisimulation to also require that one configuration sends an externally visible message if and only if the other does. We conjecture that this modified observational equivalence coincides with the one defined in section 4.

Session-typed languages enjoy other notions of process equivalence. Several session-typed languages are equipped with denotational semantics, and denotational semantics induce a compositional notion of program equivalence. For example, Castellan and Yoshida [6] gave a game semantics to a session-typed π-calculus with recursion, where session types denote event structures that encode games, and processes denote maps that encode strategies. Kavanagh [15] gave a domain-theoretic semantics to a full-featured functional language with session-typed message passing concurrency, where session types denote domains of communications and processes are continuous functions between these.

Atkey's observed communication semantics [2] for Wadler's CP [28] was motivated by two problems. Because CP uses a synchronous communication semantics, processes need partners to communicate with and get stuck if they try to communicate on a free channel. On the one hand, if processes have partners, then their communication are hidden by the (CUT) rule and cannot be observed, while on the other hand, if we leave the channels free, then we need to introduce reduction rules ("commuting conversions") for stuck processes, and these rules do not correspond to operationally justified communication steps. Atkey's elegant solution to this tension was to give communication partners to processes with free channels via closing "configurations", and then observing communications on these channels. Our task in section 4 is made easier by the fact that we use an asynchronous communication semantics. In our setting, a process can send messages on free channels, and we can observe these without having to provide it with communication partners via configurations. Atkey's observational equivalence and ours suffer from the

same weakness: to reason about observational equivalence, we must quantify over all observation contexts. Atkey addresses this by relating his semantics to a denotational semantics for CP and showing that they induce the same notion of equivalence. We are actively working on relating our OCS to Kavanagh's domain semantics [15]. Indeed, our OCS is largely motivated by efforts to relate denotational semantics of session-typed languages to their existing substructural operational semantics. We believe that our results on fair executions and their permutations should also simplify reasoning about observational equivalence.

6   Conclusion and Acknowledgements

We studied fair executions of multiset rewriting systems, and gave various conditions for an MRS to have fair executions. We used these results to define an observed communication semantics for session-typed languages that are defined by substructural operational semantics: the observation of a process is its communications on its free channels. Processes are then observationally equivalent if they cannot be distinguished through communication. We believe this work lays the foundation for future work on the semantics of session-typed processes, and in particular, we hope that it will be useful for exploring other notions of process equivalence.

The author thanks Stephen Brookes, Iliano Cervesato, Frank Pfenning, and the anonymous reviewers for their helpful comments.

References

[1]   Krzysztof R. Apt & Ernst-Rüdiger Olderog (1982): *Proof Rules Dealing With Fairness*. Extended Abstract. In Dexter Kozen, editor: *Logics of Programs*, Logics of Programs Workshop, *Lecture Notes in Computer Science* 131, Springer-Verlag Berlin Heidelberg, pages 1–8, doi: `10.1007/BFb0025770`.

[2]   Robert Atkey (2017): *Observed Communication Semantics for Classical Processes*. In Hongseok Yang, editor: *Programming Languages and Systems*, 26th European Symposium on Programming (ESOP 2017), *Lecture Notes in Computer Science* 10201, Springer Berlin Heidelberg, Berlin, pages 56–82, doi: `10.1007/978-3-662-54434-1`.

[3]   Stephanie Balzer & Frank Pfenning (2017): *Manifest Sharing With Session Types*. *Proceedings of the ACM on Programming Languages* 1(ICFP):37, 29 pages, doi: `10.1145/3110281`.

[4]   Luís Caires & Frank Pfenning (2010): *Session Types as Intuitionistic Linear Propositions*. In Paul Gastin & François Laroussinie, editors: *CONCUR 2010 — Concurrency Theory*, 21st International Conference, CONCUR 2010, *Lecture Notes in Computer Science* 6269, Springer-Verlag Berlin Heidelberg, pages 222–236, doi: `10.1007/978-3-642-15375-4_16`.

[5]   Luís Caires, Frank Pfenning & Bernardo Toninho (2016): *Linear Logic Propositions As Session Types*. *Mathematical Structures in Computer Science. Behavioural Types Part 2* 26(3), pages 367–423, doi: `10.1017/s0960129514000218`. Luís Caires & Frank Pfenning (2010): *Session Types as Intuitionistic Linear Propositions*. In Paul Gastin & François Laroussinie, editors: *CONCUR 2010 — Concurrency Theory*, 21st International Conference, CONCUR 2010, *Lecture Notes in Computer Science* 6269, Springer-Verlag Berlin Heidelberg, pages 222–236, doi: `10.1007/978-3-642-15375-4_16`.

[6]   Simon Castellan & Nobuko Yoshida (2019): *Two Sides of the Same Coin: Session Types and Game Semantics: A Synchronous Side and an Asynchronous Side. Proceedings of the ACM on Programming Languages* 3(POPL):27, 29 pages, doi: 10.1145/3290340.

[7]   I. Cervesato, N. Durgin, M. Kanovich & A. Scedrov (2000): *Interpreting Strands in Linear Logic*. In *2000 Workshop on Formal Methods and Computer Security*.

[8]   I. Cervesato, N. A. Durgin, P. D. Lincoln, J. C. Mitchell & A. Scedrov (1999): *A Meta-Notation for Protocol Analysis*. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop*, 12th IEEE Computer Security Foundations Workshop (CSFW'99), IEEE Computer Society, Los Alamitos, California, pages 55–69, doi: 10.1109/CSFW.1999.779762.

[9]   Iliano Cervesato, Nancy A. Durgin, Patrick D. Lincoln, John C. Mitchell & Andre Scedrov (2005): *A Comparison Between Strand Spaces and Multiset Rewriting for Security Protocol Analysis. Journal of Computer Security* 13(2), pages 265–316, doi: 10.3233/JCS-2005-13203.

[10] Iliano Cervesato & Andre Scedrov (2009): *Relating State-Based and Process-Based Concurrency Through Linear Logic (full-Version). Information and Computation. 13th Workshop on Logic, Language, Information and Computation (WoLLIC 2006)* 207(10), pages 1044–1077, doi: 10.1016/j.ic.2008.11.006.

[11] Gerardo Costa & Colin Stirling (1987): *Weak and Strong Fairness in CCS. Information and Computation* 73(3), pages 207–244, doi: 10.1016/0890-5401(87)90013-7.

[12] Nissim Francez (1986): *Fairness*, xiii+295 pages. *Texts and Monographs in Computer Science*, Springer-Verlag New York Inc. doi: 10.1007/978-1-4612-4886-6.

[13] Hannah Gommerstadt, Limin Jia & Frank Pfenning (2018): *Session-Typed Concurrent Contracts*. In Amal Ahmed, editor: *Programming Languages and Systems*, 27th European Symposium on Programming (ESOP 2018), *Lecture Notes in Computer Science* 10801, Springer, Cham, pages 771–798, doi: 10.1007/978-3-319-89884-1.

[14] Bart Jacobs & Jan Rutten (2012): *An Introduction to (Co)algebra and (Co)induction*. In Davide Sangiorgi & Jan Rutten, editors: *Advanced Topics in Bisimulation and Coinduction*, pages 38–99. *Cambridge Tracts in Theoretical Computer Science* 52, Cambridge University Press, Cambridge, United Kingdom, doi: 10.1017/CBO9780511792588.003.

[15] Ryan Kavanagh (2020): *A Domain Semantics for Higher-Order Recursive Processes*, arXiv: 2002.01960v3 [cs.PL].

[16] Wen Kokke, Fabrizio Montesi & Marco Peressotti (2019): *Better Late Than Never: A Fully-Abstract Semantics for Classical Processes. Proceedings of the ACM on Programming Languages* 4(POPL):24, 29 pages, doi: 10.1145/3290337.

[17] Robin Milner (1980): *A Calculus of Communicating Systems*, vi+171 pages. *Lecture Notes in Computer Science* 92, Springer-Verlag Berlin Heidelberg. doi: 10.1007/978-3-540-38311-6.

[18] David Park (1982): *A Predicate Transformer for Weak Fair Iteration. RIMS Kôkyûroku* 454, pages 211–228, ISSN: 1880-2818, HDL: 2433/103001. Also appears in [21].

[19] Jorge A. Pérez, Luís Caires, Frank Pfenning & Bernardo Toninho (2014): *Linear Logical Relations and Observational Equivalences for Session-Based Concurrency. Information and Computation* 239, pages 254–302, doi: 10.1016/j.ic.2014.08.001.

[20] Frank Pfenning & Dennis Griffith (2015): *Polarized Substructural Session Types*. In Andrew Pitts, editor: *Foundations of Software Science and Computation Structures*, 18th International Conference, FOSSACS 2015, *Lecture Notes in Computer Science* 9034, Springer-Verlag GmbH Berlin Heidelberg, Berlin Heidelberg, pages 3–32, doi: 10.1007/978-3-662-46678-0_1.

[21] (1981): *Proceedings of the Sixth IBM Symposium on Mathematical Foundations of Computer Science: Logic Aspects of Programs*, 6th IBM Symposium on Mathematical Foundations of Computer Science, Corporate & Scientific Programs, IBM Japan, Tokyo, Japan.

[22] Klaas Pruiksma & Frank Pfenning (2019): *A Message-Passing Interpretation of Adjoint Logic*. In Francisco Martins & Dominic Orchard, editors: *Proceedings: Programming Language Approaches to Concurrency- and Communication-cEntric Software*, Programming Language Approaches to Concurrency- and Communication-cEntric Software (PLACES), *Electronic Proceedings in Theoretical Computer Science* 291, European Joint Conferences on Theory and Practice of Software, pages 60–79, doi: 10.4204/EPTCS.291.6, arXiv: 1904.01290v1 [cs.PL].

[23] Davide Sangiorgi (2012): *Introduction to Bisimulation and Coinduction*, xii+247 pages. Cambridge University Press, Cambridge, United Kingdom, doi: 10.1017/CBO9780511777110.

[24] Robert J. Simmons (2012): *Substructural Logical Specifications*, PhD thesis, xvi+300 pages. Computer Science Department, Carnegie Mellon University, Pittsburgh, Pennsylvania.

[25] Bernardo Toninho, Luis Caires & Frank Pfenning (2013): *Higher-Order Processes, Functions, and Sessions: A Monadic Integration*. In Matthias Felleisen & Philippa Gardner, editors: *Programming Languages and Systems*, 22nd European Symposium on Programming, ESOP 2013, *Lecture Notes in Computer Science* 7792, Springer Berlin Heidelberg, Berlin, Heidelberg, pages 350–369, doi: 10.1007/978-3-642-37036-6_20.

[26] Bernardo Toninho, Luís Caires & Frank Pfenning (2011): *Dependent Session Types Via Intuitionistic Linear Type Theory*. In *PPDP'11*, 13th International ACM SIGPLAN Symposium on Principles and Practices of Declarative Programming (PPDP'11), Association for Computing Machinery, Inc., New York, New York, pages 161–172, doi: 10.1145/2003476.2003499.

[27] Bernardo Parente Coutinho Fernandes Toninho (2015): *A Logical Foundation for Session-based Concurrent Computation*, PhD thesis, xviii+178 pages. Universidade Nova de Lisboa.

[28] Philip Wadler (2014): *Propositions As Sessions*. *Journal of Functional Programming* 24(2-3), pages 384–418, doi: 10.1017/s095679681400001x.