Submitted to MFPS 2024

Typed Non-determinism in Concurrent Calculi: The Eager Way

Bas van den Heuvel[a] Daniele Nantes-Sobrinho[b] Joseph W.N. Paulus [c] Jorge A. Pérez[d]

[a] *Karlsruhe University of Applied Sciences, Karlsruhe, and University of Freiburg, Freiburg, Germany*

[b] *Department of Computing, Imperial College London, London, UK*

[c] *University of Oxford, Oxford, UK*

[d] *University of Groningen, Groningen, The Netherlands*

Abstract

We consider the problem of designing typed concurrent calculi with *non-deterministic choice* in which types leverage *linearity* for controlling resources, thereby ensuring strong correctness properties for processes. This problem is constrained by the delicate tension between non-determinism and linearity. Prior work developed a session-typed π-calculus with standard non-deterministic choice; well-typed processes enjoy type preservation and deadlock-freedom. Central to this typed calculus is a *lazy* semantics that gradually discards branches in choices. This lazy semantics, however, is complex: various technical elements are needed to describe the non-deterministic behavior of typed processes. This paper develops an entirely new approach, based on an *eager* semantics, which more directly represents choices and commitment. We present a π-calculus in which non-deterministic choices are governed by this eager semantics and session types. We establish its key correctness properties, including deadlock-freedom, and demonstrate its expressivity by correctly translating a typed resource λ-calculus.

Keywords: Concurrency, process calculi, linear type systems, session types, intersection types, non-determinism.

1 Introduction

This paper addresses the problem of designing concurrent calculi with *non-deterministic choice* in which types leverage *linearity* for controlling resources. Specifically, our interest is in variants of the π-calculus, the paradigmatic calculus of concurrency and interaction [7,12]; here the resources are the names (or channels) that communicating processes use to perform protocols described by *session types* [5,6]. This is a challenging design problem, due to the delicate tension between non-determinism and linearity. On the one hand, resource control based on linearity is essential to statically enforce important correctness properties for processes: *protocol fidelity* (processes respect their protocols), *communication safety* (processes never incur into message mismatches), and *deadlock-freedom* (processes never get stuck). On the other hand, implementing the usual (non-confluent) semantics of non-deterministic choice is at odds with linearity: a careless handling of discarded branches in choices can jeopardize resources meant to be used exactly once.

To better understand the problem, it is instructive to recall the reduction rule for the (untyped) π-calculus (e.g., [12]):

$$(\overline{x}[z]; P_1 + M_1) \mid (x(y); P_2 + M_2) \longrightarrow P_1 \mid P_2\{z/y\} \qquad (1)$$

Rule (1) specifies the interaction between two (binary) choices: it coalesces a synchronization along name x (whereby name z is passed around) with the commitment to retaining the branches involved in the exchange. Indeed, after reduction the two branches not involved in the synchronization, M_1 and M_2,

are discarded. While appropriate in the untyped setting, it would be unwise to adopt this rule in a linearly-typed setting: clearly, M_1 and M_2 could as well denote resources that must be used exactly once.

The core technical problem is then how to devise formulations for non-determinism that preserve the non-confluent character captured by Rule (1)—which effectively expresses commitment in specifications—while respecting the principles of linearity-based resource control.

As an answer to this problem, our prior work [14] introduced a typed process model with a new non-deterministic choice operator, denoted $P \mathbin{+\!\!\!\!+} Q$, in which P and Q act upon the same (linear) resources—they are branches that denote different implementations of the same session protocols. This choice operator is governed by a *lazy* semantics that minimizes commitment as much as possible. Roughly speaking, the lazy semantics distinguishes between "possible" and "impossible" branches, depending on the synchronizations enabled in a process and its context. This distinction allows us to reduce the set of branches under consideration: the actual choice (as in Rule (1)) is enacted at the level of possible branches.

The lazy semantics meets our desiderata: it expresses commitment and respects linearity of resources. Also, the resulting typed process model enforces the three correctness properties given above, and is also expressive enough to precisely encode a resource λ-calculus with non-deterministic behavior and explicit failures. Still, the lazy approach is not entirely satisfactory: its definition is complex and so the behavior of non-deterministic choices cannot be easily discerned. The lazy semantics rests upon a pre-order on processes (which captures the intermediate distinction between possible and impossible branches) and a compatibility relation on prefixes; also, it needs to be indexed by the names involved in the synchronization. This required machinery is not ideal, in particular if one contrasts it with the compact and effective Rule (1).

In this paper, we devise a formulation of non-determinism that is simpler and more direct than the lazy semantics of [14]. We propose an *eager* semantics that enforces commitment by examining the contexts under which reductions occur. This is an economical solution, as it rests upon a simple definition that is arguably easier to understand and reason about than the lazy semantics. Perhaps more importantly, our new eager semantics still meets our desiderata on commitment and linearity as enforced by typing.

Clearly, giving an alternative eager semantics for the typed process model in [14] immediately raises the question of its positioning with respect to preceding developments. Several interesting issues arise. Does the eager semantics fit well with the session type system in [14]? Because the typed process model with the lazy semantics was shown to precisely encode a resource λ-calculus with non-determinism, we may also ask: does moving to a simpler operational setting affect expressivity? Moreover, how does the eager semantics compare to the lazy semantics, independently from the ability of encoding advanced λ-calculi?

This paper's goal is to provide technical answers to these questions. The base language for our new eager semantics is $\mathsf{s}\pi^!$, the extension of the session π-calculus in [14] with unrestricted behaviors (client and server constructs). This way, our developments are based on a richer setting than in [14] (where only linear behaviors were considered). Having defined the eager semantics, we move to consider the associated session type system. We actually consider the exact same type system as in [14] and establish that well-typed processes satisfy the same properties (type preservation and deadlock-freedom). These results are reassuring: they confirm that our eager semantics does not break properties derived from typing, and that the eager/lazy distinction remains an operational concern, which does not transpire at the level of typing.

We then assess the expressiveness of the eager process model by giving a process interpretation of λ_{C}, a resource λ-calculus with non-determinism. Also in this case, we consider an extension of the language considered in [14]: the calculus λ_{C} features both linear and unrestricted resources, which requires several innovations, in particular for the associated intersection type system (the λ-calculus in [14] is the sub-language of λ_{C} with linear resources only). The translation of λ_{C} into $\mathsf{s}\pi^!$ we present here also features innovations: while its linear portion (i.e., the translation of terms with linear resources into linear processes) is the same as in [14], the translation of terms with unrestricted resources into client/server processes is new to this presentation. Again, this corroborates that the eager/lazy distinction is not relevant at the static level given by the translation. The salient differences appear at the *dynamic* level, i.e., in the operational correspondence properties that relate the computations of a term in λ_{C} with the behavior of its corresponding process in $\mathsf{s}\pi^!$ (and vice versa). Indeed, it turns out that the eager semantics of $\mathsf{s}\pi^!$ induces operational correspondences that are "looser" than in the lazy regime. That is, the lazy semantics provides a tighter account of the dynamics of terms and their corresponding translations.

In summary, our paper extends and complements the results in [14] with the following contributions:

$P, Q ::= \mathbf{0}$	inaction	$\mid [x \leftrightarrow y]$	forwarder	$\mid P \mid Q$	parallel
$\mid (\boldsymbol{\nu} x)(P \mid Q)$	connect	$\mid P \mathbin{+\!\!\!+} Q$	non-determinism	$\mid \overline{x}.\texttt{some}; P$	available
$\mid \overline{x}[y]; (P \mid Q)$	output	$\mid x(y); P$	input	$\mid \overline{x}.\texttt{none}$	unavailable
$\mid \overline{x}.\ell; P$	select	$\mid x.\texttt{case}\{i : P\}_{i \in I}$	branch	$\mid x.\texttt{some}_{w_1, \ldots, w_n}; P$	expect
$\mid ?\overline{x}[y]; P$	client request	$\mid !x(y); P$	server		
$\mid \overline{x}[]$	close	$\mid x(); P$	wait		

$$P \equiv P' \; [P \equiv_\alpha P'] \qquad [x \leftrightarrow y] \equiv [y \leftrightarrow x] \qquad P \mid \mathbf{0} \equiv P$$
$$(P \mid Q) \mid R \equiv P \mid (Q \mid R) \qquad P \mid Q \equiv Q \mid P \qquad (\boldsymbol{\nu} x)(P \mid Q) \equiv (\boldsymbol{\nu} x)(Q \mid P)$$
$$P \mathbin{+\!\!\!+} P \equiv P \qquad P \mathbin{+\!\!\!+} Q \equiv Q \mathbin{+\!\!\!+} P \qquad (P \mathbin{+\!\!\!+} Q) \mathbin{+\!\!\!+} R \equiv P \mathbin{+\!\!\!+} (Q \mathbin{+\!\!\!+} R)$$

$$(\boldsymbol{\nu} x)((P \mid Q) \mid R) \equiv (\boldsymbol{\nu} x)(P \mid R) \mid Q \qquad [x \notin fn(Q)]$$
$$(\boldsymbol{\nu} x)((\boldsymbol{\nu} y)(P \mid Q) \mid R) \equiv (\boldsymbol{\nu} y)((\boldsymbol{\nu} x)(P \mid R) \mid Q) \qquad [x \notin fn(Q), y \notin fn(R)]$$
$$(\boldsymbol{\nu} x)(!x(y); P \mid Q) \equiv Q \qquad [x \notin fn(Q)]$$

Fig. 1. $\mathsf{s}\pi^!$: syntax (top) and structural congruence (bottom).

(i) A new eager semantics for $\mathsf{s}\pi^!$, the session π-calculus with non-deterministic choice that extends the calculus introduced in [14] with client/server behaviors (Section 2.3).

(ii) A type system for $\mathsf{s}\pi^!$, which ensures type preservation and deadlock-freedom for well-typed processes governed by the eager semantics (Section 2.4).

(iii) The resource calculus λ_{C}, which extends the one presented in [14] with unrestricted resources. Governed by intersection types, we establish subject reduction and subject expansion results (Section 3).

(iv) A typed translation of λ_{C} into $\mathsf{s}\pi^!$, with an analysis of its static and dynamic correctness (Section 4), and a comparison between our new eager semantics and the lazy semantics presented (Section 5).

Omitted material can be found in [13], which contains technical details for both lazy and eager semantics. Throughout the paper we use different colors (such as red and green) to improve readability. However, the paper can be followed in black-and-white.

2 A Typed π-calculus with Non-deterministic Choice

In this section, we start by giving the syntax of $\mathsf{s}\pi^!$, a session-typed π-calculus with non-deterministic choice. Following the linear calculus $\mathsf{s}\pi^+$ given in [14], the key feature in $\mathsf{s}\pi^!$ is the non-deterministic choice operator '$P \mathbin{+\!\!\!+} Q$'. The key novelty is the eagerly committing semantics for '$\mathbin{+\!\!\!+}$', which is compatible with linearity (Section 2.3). Following its predecessors [2,14], we give a session type system for $\mathsf{s}\pi^!$; intuitively, session types express protocols to be executed along channels. We prove that well-typed processes under the new eager semantics satisfy two key properties: *type preservation* and *deadlock-freedom*.

2.1 Syntax

We use $P, Q, \ldots$ to denote processes, and $x, y, z, \ldots$ to denote *names* representing channels. Figure 1 (top) gives the syntax of processes. $P\{y/z\}$ denotes the capture-avoiding substitution of y for z in P. Process $\mathbf{0}$ denotes inaction, and $[x \leftrightarrow y]$ is a forwarder: a bidirectional link between x and y. There are two forms of parallel composition: while the process $P \mid Q$ denotes communication-free concurrency, process $(\boldsymbol{\nu} x)(P \mid Q)$ uses restriction $(\boldsymbol{\nu} x)$ to express that P and Q communicate on x and do not share any other names.

Process $P \mathbin{+\!\!\!+} Q$ denotes the non-deterministic choice between P and Q: intuitively, if one choice can perform a synchronization, the other option may be discarded if it cannot. Since $\mathbin{+\!\!\!+}$ is associative, we often omit parentheses. Also, we write $\mathop{+\!\!\!+}_{i \in I} P_i$ for the non-deterministic choice between each P_i for $i \in I$.

Our output construct integrates parallel composition and restriction: process $\overline{x}[y]; (P \mid Q)$ sends a fresh name y along x and then continues as $P \mid Q$. Types will ensure that behaviors on y and x are implemented

by P and Q, respectively, which do not share any names; this is a form of communication-free concurrency that is key to avoiding deadlocks. The input process $x(y); P$ receives a name z along x and continues as $P\{z/y\}$. Process $x.\mathtt{case}\{i : P_i\}_{i \in I}$ denotes a branch with labeled choices indexed by the finite set I: it awaits a choice on x with continuation P_j for each $j \in I$. The process $\overline{x}.\ell; P$ selects on x the choice labeled ℓ before continuing as P. Processes $\overline{x}[]$ and $x(); P$ are dual actions for closing the session on x.

Our language has server and client processes, not considered in [14]. The server process $!x(y); P$ accepts requests from clients, receiving a name z along x to spawn $P\{z/y\}$; the server process remains available for further requests. A client request $?\overline{x}[y]; P$ sends a fresh name y along x and continues as P. Both client and server prefixes bind y in P.

The remaining constructs define non-deterministic sessions which may provide a protocol or fail [2]. Process $\overline{x}.\mathtt{some}; P$ confirms the availability of a session on x and continues as P. Process $\overline{x}.\mathtt{none}$ signals the failure to provide the session on x. Process $x.\mathtt{some}_{w_1,\ldots,w_n}; P$ specifies a dependency on a non-deterministic session on x (names $w_1, \ldots, w_n$ implement sessions in P). This process can either (i) synchronize with a '$\overline{x}.\mathtt{some}$' and continue as P, or (ii) synchronize with a '$\overline{x}.\mathtt{none}$', discard P, and propagate the failure to $w_1, \ldots, w_n$. To reduce eye strain, in writing $x.\mathtt{some}$ we freely combine names and sets of names. This way, e.g., we write $x.\mathtt{some}_{y,\mathit{fn}(P),\mathit{fn}(Q)}$ rather than $x.\mathtt{some}_{\{y\} \cup \mathit{fn}(P) \cup \mathit{fn}(Q)}$.

Name y is bound in $(\boldsymbol{\nu} y)(P \mid Q)$, $\overline{x}[y]; (P \mid Q)$, and $x(y); P$. We write $\mathit{fn}(P)$ and $\mathit{bn}(P)$ for the free and bound names of P, respectively. The sets $\mathit{fln}(P)$ and $\mathit{fpn}(P)$ contain the free linear and non-linear names of P, respectively. Note that $\mathit{fpn}(P) = \mathit{fn}(P) \setminus \mathit{fln}(P)$. We adopt Barendregt's convention.

As usual, we shall use a *structural congruence* ($\equiv$), the least congruence relation on processes induced by the rules in Figure 1 (bottom). Like the syntax of processes, the definition of $\equiv$ is aligned with the type system (defined next), such that $\equiv$ preserves typing (subject congruence, cf. Theorem 2.5). Notice that non-deterministic choice does not distribute over parallel and restriction. The position of a non-deterministic choice in a process determines how it may commit, so changing its position affects commitment.

2.2 The Lazy Semantics, by Example

Reduction defines the steps that a process performs on its own. A key contribution of our paper is an eager semantics for $\mathsf{s}\pi^!$, which we present in Section 2.3. Before going into details, we find it instructive to illustrate the key ideas of the lazy semantics from [14].

The lazy semantics relies on a precongruence on processes, denoted $\succeq_S$, where S is a set that contains the names involved in a reduction: $S = \{x\}$ for a synchronization on x, and $S = \{x, y\}$ when a forwarder process $[x \leftrightarrow y]$ reduces. Building upon $\succeq_S$, the lazy semantics is then denoted $\rightsquigarrow_S$. We omit the curly braces; this way, e.g., we write '$\rightsquigarrow_{x,y}$' instead of '$\rightsquigarrow_{\{x,y\}}$'. The following example illustrates $\succeq_S$ and $\rightsquigarrow_S$:

Example 2.1 Consider a server that offers watching a movie's trailer or buying a movie using card or cash. We define the process $\mathsf{MovieServer}_s := s(\mathit{title}); \mathsf{Movies}_s$, where s is a name and Movies_s is as follows:

$$\begin{aligned}
\mathsf{Movies}_s &:= s.\mathtt{case}\{\mathsf{buy} : \mathsf{MoviesBuy}_s, \mathsf{peek} : \mathsf{MoviesPeek}_s\} \\
\mathsf{MoviesBuy}_s &:= s.\mathtt{case}\{\mathsf{card} : \mathsf{MoviesBuyCard}_s, \mathsf{cash} : \mathsf{MoviesBuyCash}_s\} \\
\mathsf{MoviesBuyCard}_s &:= s(\mathit{info}); \overline{s}[\mathtt{movie}]; \overline{s}[] \\
\mathsf{MoviesBuyCash}_s &:= \overline{s}[\mathtt{movie}]; \overline{s}[] \\
\mathsf{MoviesPeek}_s &:= \overline{s}[\mathtt{trailer}]; \overline{s}[]
\end{aligned}$$

Now consider a client, Eve, undecided between buying 'Barbie' or watching its trailer. If she decides to buy the movie, she cannot decide between paying with card or cash. We model Eve as follows, again using s to communicate with the movie server, and modeling her indecisiveness with non-deterministic choices:

$$\begin{aligned}
\mathsf{MovieClient}_s &:= \overline{s}[\mathtt{Barbie}]; \mathsf{Eve}_s \\
\mathsf{Eve}_s &:= \overline{s}.\mathsf{buy}; \overline{s}.\mathsf{card}; \mathsf{EveBuyCard}_s \mathbin{+\!\!\!\!+} \overline{s}.\mathsf{buy}; \overline{s}.\mathsf{cash}; \mathsf{EveBuyCash}_s \mathbin{+\!\!\!\!+} \overline{s}.\mathsf{peek}; \mathsf{EvePeek}_s \\
\mathsf{EveBuyCard}_s &:= \overline{s}[\mathtt{visa}]; s(\mathit{movie}); s(); \mathbf{0} \\
\mathsf{EveBuyCash}_s &:= s(\mathit{movie}); s(); \mathbf{0} \\
\mathsf{EvePeek}_s &:= s(\mathit{link}); s(); \mathbf{0}
\end{aligned}$$

$$[\rightarrow_{\text{ID}}]\ (\boldsymbol{\nu} x)(\mathtt{N}[[x \leftrightarrow y]] \mid Q) \longrightarrow (\!|\mathtt{N}|\!)[Q\{y/x\}] \qquad [\rightarrow_{\mathbf{1}\bot}]\ (\boldsymbol{\nu} x)(\mathtt{N}[\overline{x}[]] \mid \mathtt{N}'[x(); Q]) \longrightarrow (\!|\mathtt{N}|\!)[\mathbf{0}] \mid (\!|\mathtt{N}'|\!)[Q]$$

$$[\rightarrow_{\otimes ⅋}]\ (\boldsymbol{\nu} x)(\mathtt{N}[\overline{x}[y]; (P \mid Q)] \mid \mathtt{N}'[x(z); R]) \longrightarrow (\!|\mathtt{N}|\!)[(\boldsymbol{\nu} x)(Q \mid (\boldsymbol{\nu} y)(P \mid (\!|\mathtt{N}'|\!)[R\{y/z\}]))]$$

$$[\rightarrow_{\oplus \&}]\ \forall k' \in K.\ (\boldsymbol{\nu} x)(\mathtt{N}[\overline{x}.k'; P] \mid \mathtt{N}'[x.\mathtt{case}\{k : Q^k\}_{k \in K}]) \longrightarrow (\boldsymbol{\nu} x)((\!|\mathtt{N}|\!)[P] \mid (\!|\mathtt{N}'|\!)[Q^{k'}])$$

$$[\rightarrow_{?!}]\ (\boldsymbol{\nu} x)(\mathtt{N}[?\overline{x}[y]; P] \mid \mathtt{N}'[!x(z); Q]) \longrightarrow (\!|\mathtt{N}'|\!)\big[(\boldsymbol{\nu} x)\big((\boldsymbol{\nu} y)((\!|\mathtt{N}|\!)[P] \mid Q\{y/z\}) \mid !x(z); Q\big)\big]$$

$$[\rightarrow_{\mathtt{some}}]\ (\boldsymbol{\nu} x)(\mathtt{N}[\overline{x}.\mathtt{some}; P] \mid \mathtt{N}'[x.\mathtt{some}_{w_1,\ldots,w_n}; Q]) \longrightarrow (\boldsymbol{\nu} x)((\!|\mathtt{N}|\!)[P] \mid (\!|\mathtt{N}'|\!)[Q])$$

$$[\rightarrow_{\mathtt{none}}]\ (\boldsymbol{\nu} x)(\mathtt{N}[\overline{x}.\mathtt{none}] \mid \mathtt{N}'[x.\mathtt{some}_{w_1,\ldots,w_n}; Q]) \longrightarrow (\!|\mathtt{N}|\!)[\mathbf{0}] \mid (\!|\mathtt{N}'|\!)[\overline{w_1}.\mathtt{none} \mid \ldots \mid \overline{w_n}.\mathtt{none}]$$

$$[\rightarrow_{\equiv}]\ \frac{P \equiv P' \quad P' \longrightarrow Q' \quad Q' \equiv Q}{P \longrightarrow Q} \qquad [\rightarrow_{\nu}]\ \frac{P \longrightarrow P'}{(\boldsymbol{\nu} x)(P \mid Q) \longrightarrow (\boldsymbol{\nu} x)(P' \mid Q)} \qquad [\rightarrow_{\mid}]\ \frac{P \longrightarrow P'}{P \mid Q \longrightarrow P' \mid Q}$$

$$[\rightarrow_{+\!\!\!+}]\ \frac{P \longrightarrow P'}{P \mathbin{+\!\!\!+} Q \longrightarrow P' \mathbin{+\!\!\!+} Q}$$

Fig. 2. Eager reduction semantics for $\mathsf{s}\pi^!$.

We compose our movie server and Eve on s. Initially, the movie title is sent without non-determinism; notice how the reduction is annotated with 's' to indicate that the exchange occurred on s:

$$(\boldsymbol{\nu} s)(\mathsf{MovieServer}_s \mid \mathsf{MovieClient}_s) \rightsquigarrow_s (\boldsymbol{\nu} s)(\mathsf{Movies}_s \mid \mathsf{Eve}_s)$$

At this point, two communications on s are available: the selection of buy and the selection of peek. We model Eve's choice using the precongruence $\succeq_s$:

$$\mathsf{Eve}_s \succeq_s \overline{s}.\mathsf{buy}; \overline{s}.\mathsf{card}; \mathsf{EveBuyCard}_s \mathbin{+\!\!\!+} \overline{s}.\mathsf{buy}; \overline{s}.\mathsf{cash}; \mathsf{EveBuyCash}_s \quad \text{and} \quad \mathsf{Eve}_s \succeq_s \overline{s}.\mathsf{peek}; \mathsf{EvePeek}_s.$$

Notice how in the buy-case the precongruence preserves the choice between the two methods of payment, because both choices start with the same selection. As such, performing the buy-selection preserves this choice, whereas performing the peek-selection results in a single alternative:

$$(\boldsymbol{\nu} s)(\mathsf{Movies}_s \mid \mathsf{Eve}_s) \rightsquigarrow_s (\boldsymbol{\nu} s)\big(\mathsf{MoviesBuy}_s \mid (\overline{s}.\mathsf{card}; \mathsf{EveBuyCard}_s \mathbin{+\!\!\!+} \overline{s}.\mathsf{cash}; \mathsf{EveBuyCash}_s)\big)$$

and $(\boldsymbol{\nu} s)(\mathsf{Movies}_s \mid \mathsf{Eve}_s) \rightsquigarrow_s (\boldsymbol{\nu} s)(\mathsf{MoviesPeek}_s \mid \mathsf{EvePeek}_s)$.

After the buy-selection, the choice cannot be preserved, because the branches start with different selections; this is reflected by the precongruence on Eve's process:

$$\overline{s}.\mathsf{card}; \mathsf{EveBuyCard}_s \mathbin{+\!\!\!+} \overline{s}.\mathsf{cash}; \mathsf{EveBuyCash}_s \succeq_s \overline{s}.\mathsf{card}; \mathsf{EveBuyCard}_s$$

and $\overline{s}.\mathsf{card}; \mathsf{EveBuyCard}_s \mathbin{+\!\!\!+} \overline{s}.\mathsf{cash}; \mathsf{EveBuyCash}_s \succeq_s \overline{s}.\mathsf{cash}; \mathsf{EveBuyCash}_s$.

As such, after the buy-selection, two further reduction paths are possible. We give one of them:

$$(\boldsymbol{\nu} s)\big(\mathsf{MoviesBuy}_s \mid (\overline{s}.\mathsf{card}; \mathsf{EveBuyCard}_s \mathbin{+\!\!\!+} \overline{s}.\mathsf{cash}; \mathsf{EveBuyCash}_s)\big) \rightsquigarrow_s (\boldsymbol{\nu} s)(\mathsf{MoviesBuyCard}_s \mid \mathsf{EveBuyCard}_s)$$

2.3 The New Eager Semantics

The eager reduction semantics, denoted $\longrightarrow$, is given in Figure 2. Intuitively, we follow the principles of Rule (1), discussed earlier: a reduction step simultaneously expresses (i) the intended interaction (say, a synchronization) and (ii) the transformation of the involved contexts so as to express commitment. To this end, we rely on *ND-contexts*, denoted $\mathtt{N}, \mathtt{M}, \ldots$, and their *commitment*, denoted $(\!|\mathtt{N}|\!), (\!|\mathtt{M}|\!), \ldots$, respectively.

Definition 2.2 We define *ND-contexts* ($\mathtt{N}, \mathtt{M}$) as follows:

$$\mathtt{N}, \mathtt{M} ::= [\cdot] \;\big|\; \mathtt{N} \mid P \;\big|\; (\boldsymbol{\nu} x)(\mathtt{N} \mid P) \;\big|\; \mathtt{N} \mathbin{+\!\!\!+} P$$

The process obtained by replacing $[\cdot]$ in $\mathtt{N}$ with P is denoted $\mathtt{N}[P]$. We refer to ND-contexts that do not use the clause '$\mathtt{N} \mathbin{+\!\!\!+} P$' as *D-contexts*, denoted $\mathtt{C}, \mathtt{D}$.

This semantics implements the expected commitment of non-deterministic choices by transforming ND-contexts to D-contexts as follows:

Definition 2.3 The *commitment* of an ND-context $\mathtt{N}$, denoted $(\!|\mathtt{N}|\!)$, is defined as follows:

$$(\!|[\cdot]|\!) := [\cdot] \qquad (\!|\mathtt{N} \mid P|\!) := (\!|\mathtt{N}|\!) \mid P \qquad (\!|(\boldsymbol{\nu} x)(\mathtt{N} \mid P)|\!) := (\boldsymbol{\nu} x)((\!|\mathtt{N}|\!) \mid P) \qquad (\!|\mathtt{N} \mathbin{+\!\!\!+} P|\!) := (\!|\mathtt{N}|\!)$$

Barring non-deterministic choice, the reduction rules in Figure 2 arise as directed interpretations of proof transformations in the underlying linear logic. We follow Caires and Pfenning [3] and Wadler [15] in interpreting cut-elimination in linear logic as synchronization in $\mathsf{s}\pi^+$. As such, reduction rules are standard but extended with non-determinism: the synchronizing subprocesses appear under ND-contexts (Def. 2.2) before reductions, and under collapsed ND-contexts (Def. 2.3) after reductions. For example, in Rule $[\rightarrow_{\otimes ⅋}]$, the send and receive appear under ND-contexts $\mathtt{N}$ and $\mathtt{N}'$, respectively. After reduction, these contexts collapse to $(\!|\mathtt{N}|\!)$ and $(\!|\mathtt{N}'|\!)$, respectively; notice how the scope of $(\!|\mathtt{N}|\!)$ extends to the entire process after the reduction, to avoid freeing names that were bound before by $\mathtt{N}$.

Example 2.4 We revisit Example 2.1 now under the eager semantics. We can express $(\boldsymbol{\nu} s)(\mathsf{Movies}_s \mid \mathsf{Eve}_s)$ using ND-contexts. There is only one way to do so for Movies_s, since it is deterministic:

$$\mathsf{Movies}_s = \mathtt{N}[s.\mathsf{case}\{\mathsf{buy} : \mathsf{MoviesBuy}_s, \mathsf{peek} : \mathsf{MoviesPeek}_s\}] \qquad \text{where} \qquad \mathtt{N} := [\cdot].$$

There are three ways to do so for Eve_s, since there are three non-deterministic branches; for example:

$$\mathsf{Eve}_s = \mathtt{N}'[\overline{s}.\mathsf{buy}; \overline{s}.\mathsf{cash}; \mathsf{EveBuyCash}_s] \qquad \text{where} \qquad \mathtt{N}' := \overline{s}.\mathsf{buy}; \overline{s}.\mathsf{card}; \mathsf{EveBuyCard}_s \mathbin{+\!\!\!+} [\cdot] \mathbin{+\!\!\!+} \overline{s}.\mathsf{peek}; \mathsf{EvePeek}_s.$$

With $(\!|\mathtt{N}|\!) = (\!|\mathtt{N}'|\!) = [\cdot]$, we have, e.g., $(\boldsymbol{\nu} s)(\mathsf{Movies}_s \mid \mathsf{Eve}_s) \longrightarrow (\boldsymbol{\nu} s)(\mathsf{MoviesBuy}_s \mid \overline{s}.\mathsf{cash}; \mathsf{EveBuyCash}_s)$. (This is one of three possible reductions.) Note: the first reduction immediately determines the payment method.

2.4 Resource Control for $\mathsf{s}\pi^!$ via Session Types

We define a session type system for $\mathsf{s}\pi^!$, following 'propositions-as-sessions' [3,15]. As already mentioned, in a session type system, resources are names that perform protocols: the *type assignment* $x : A$ says that x should conform to the protocol specified by the session type A. We give the syntax of types:

$$A, B ::= \mathbf{1} \;\big|\; \bot \;\big|\; A \otimes B \;\big|\; A ⅋ B \;\big|\; \oplus\{i : A\}_{i \in I} \;\big|\; ?A \;\big|\; !A \;\big|\; \&\{i : A\}_{i \in I} \;\big|\; \&A \;\big|\; \oplus A$$

The units $\mathbf{1}$ and $\bot$ type closed sessions. $A \otimes B$ types a name that first outputs a name of type A and then proceeds as B. Similarly, $A ⅋ B$ types a name that inputs a name of type A and then proceeds as B. Types $\oplus\{i : A_i\}_{i \in I}$ and $\&\{i : A_i\}_{i \in I}$ are given to names that can select and offer a labeled choice, respectively.

Type $?A$ is assigned to a name that performs a request for a session of type A, and type $!A$ is assigned to a name that offers a server of type A: the modalities '?' and '!' define *non-linear* (i.e., persistent) types. Then, $\&A$ is the type of a name that *may produce* a behavior of type A, or fail; dually, $\oplus A$ types a name that *may consume* a behavior of type A.

For any type A we denote its *dual* as $\overline{A}$. Intuitively, duality of types serves to avoid communication errors: the type at one end of a channel is the dual of the type at the opposite end. Duality is an involution, defined as follows:

$$\begin{array}{lllll} \overline{\mathbf{1}} = \bot & \overline{A \otimes B} = \overline{A} ⅋ \overline{B} & \overline{\oplus\{i : A_i\}_{i \in I}} = \&\{i : \overline{A_i}\}_{i \in I} & \overline{\&A} = \oplus\overline{A} & \overline{?A} = !\overline{A} \\ \overline{\bot} = \mathbf{1} & \overline{A ⅋ B} = \overline{A} \otimes \overline{B} & \overline{\&\{i : A_i\}_{i \in I}} = \oplus\{i : \overline{A_i}\}_{i \in I} & \overline{\oplus A} = \&\overline{A} & \overline{!A} = ?\overline{A} \end{array}$$

$$[\text{Tcut}]\ \frac{P \vdash \Gamma, x{:}A \qquad Q \vdash \Delta, x{:}\overline{A}}{(\boldsymbol{\nu} x)(P \mid Q) \vdash \Gamma, \Delta} \qquad [\text{Tmix}]\ \frac{P \vdash \Gamma \qquad Q \vdash \Delta}{P \mid Q \vdash \Gamma, \Delta} \qquad [\text{T}\nparallel]\ \frac{P \vdash \Gamma \qquad Q \vdash \Gamma}{P \nparallel Q \vdash \Gamma} \qquad [\text{Tempty}]\ \frac{}{\mathbf{0} \vdash \emptyset}$$

$$[\text{Tid}]\ \frac{}{[x \leftrightarrow y] \vdash x{:}A, y{:}\overline{A}} \qquad [\text{T}\mathbf{1}]\ \frac{}{\overline{x}[] \vdash x{:}\mathbf{1}} \qquad [\text{T}\bot]\ \frac{P \vdash \Gamma}{x(); P \vdash \Gamma, x{:}\bot} \qquad [\text{T}\otimes]\ \frac{P \vdash \Gamma, y{:}A \qquad Q \vdash \Delta, x{:}B}{\overline{x}[y]; (P \mid Q) \vdash \Gamma, \Delta, x{:}A \otimes B}$$

$$[\text{T}\parr]\ \frac{P \vdash \Gamma, y{:}A, x{:}B}{x(y); P \vdash \Gamma, x{:}A \parr B} \qquad [\text{T}\oplus]\ \frac{P \vdash \Gamma, x{:}A_j \qquad j \in I}{\overline{x}.j; P \vdash \Gamma, x{:}\oplus\{i : A_i\}_{i \in I}} \qquad [\text{T}\&]\ \frac{\forall i \in I.\ P_i \vdash \Gamma, x{:}A_i}{x.\mathtt{case}\{i : P_i\}_{i \in I} \vdash \Gamma, x{:}\&\{i : A_i\}_{i \in I}}$$

$$[\text{T}\&\mathtt{some}]\ \frac{P \vdash \Gamma, x{:}A}{\overline{x}.\mathtt{some}; P \vdash \Gamma, x{:}\& A} \qquad [\text{T}\&\mathtt{none}]\ \frac{}{\overline{x}.\mathtt{none} \vdash x{:}\& A} \qquad [\text{T}\oplus\mathtt{some}]\ \frac{P \vdash \&\Gamma, x{:}A}{x.\mathtt{some}_{dom(\Gamma)}; P \vdash \&\Gamma, x{:}\oplus A}$$

$$[\text{T}?]\ \frac{P \vdash \Gamma, y{:}A}{?\overline{x}[y]; P \vdash \Gamma, x{:}?A} \qquad [\text{T}!]\ \frac{P \vdash ?\Gamma, y{:}A}{!x(y); P \vdash ?\Gamma, x{:}!A} \qquad [\text{Tweaken}]\ \frac{P \vdash \Gamma}{P \vdash \Gamma, x{:}?A} \qquad [\text{Tcontract}]\ \frac{P \vdash \Gamma, x{:}?A, x'{:}?A}{P\{x/x'\} \vdash \Gamma, x{:}?A}$$

Fig. 3. Typing rules for $\mathsf{s}\pi^!$.

Judgments are of the form $P \vdash \Gamma$, where P is a process and Γ is a context, a collection of type assignments. In writing $\Gamma, x : A$, we assume $x \notin dom(\Gamma)$. We write $dom(\Gamma)$ to denote the set of names appearing in Γ. We write $\&\Gamma$ to denote that $\forall x : A \in \Gamma.\ \exists A'.\ A = \& A'$.

Figure 3 gives the typing rules: they correspond to the rules in Curry-Howard interpretations of classical linear logic as session types (cf. Wadler [15]), with the rules for $\& A$ and $\oplus A$ extracted from [2], and the additional Rule [T$\nparallel$] for non-confluent non-deterministic choice, which modifies the confluent rule in [2].

We discuss selected rules. Rule [T&some] types a process with a name whose behavior can be provided, while Rule [T&none] types a name whose behavior cannot. Rule [T⊕some] types a process with a name x whose behavior may not be available. If the behavior is not available, all the sessions in the process must be canceled; hence, the rule requires all names to be typed under the $\& A$ monad. Rule [T$\nparallel$] types our non-deterministic choice operator; the branches must be typable under the same typing context. Hence, all branches denote the same sessions, which may be implemented differently. In context of a synchronization, branches that are kept are able to synchronize, whereas the discarded branches are not; nonetheless, the remaining branches still represent different implementations of the same sessions.

Our type system ensures *session fidelity* (processes correctly follow their ascribed session protocols) *communication safety* (no communication errors/mismatches occur). Both properties follow from the fact that typing is consistent across structural congruence and reduction.

We state the main results of this section: $\mathsf{s}\pi^!$ with $\longrightarrow$ satisfies type preservation and deadlock-freedom.

Theorem 2.5 (Type Preservation) *If $P \vdash \Gamma$, then both $P \equiv Q$ and $P \longrightarrow Q$ imply $Q \vdash \Gamma$.*

Theorem 2.6 (Deadlock-freedom) *If $P \vdash \emptyset$ and $P \not\equiv \mathbf{0}$, then there is R such that $P \longrightarrow R$.*

3 A non-deterministic λ-calculus with unrestricted resources

We present $\lambda_{\mathtt{C}}$, a resource λ-calculus with non-determinism and lazy evaluation. Our calculus features two kinds of resources: linear (to be used exactly once) and unrestricted (usable zero or more times). It extends the resource λ-calculus studied in [14], which does not support unrestricted resources. In $\lambda_{\mathtt{C}}$, non-determinism is non-confluent and *implicit*, as it arises from the fetching of terms from bags of linear and unrestricted resources. (In contrast, the choice operator '$\nparallel$' in $\mathsf{s}\pi^!$ specifies non-determinism *explicitly*.) A mismatch between the number of variable occurrences and the size of the bag induces *failure*.

3.1 Syntax

Figure 4 gives the syntax of $\lambda_{\mathtt{C}}$-terms (M, N, L) and bags (A, B) and contexts $(\mathcal{C}, \mathcal{C}')$. Variables $x, y, z, \ldots$ have linear and unrestricted occurrences.

$M, N, L ::= x[*]$	variable	$\mid M[\widetilde{x} \leftarrow x]$	sharing
$\mid \lambda x.M$	abstraction	$\mid M\langle\!\langle {}^{B}\!/x \rangle\!\rangle$	intermediate substitution
$\mid (M\ B)$	application	$\mid M\langle\!\vert C/x_1, \ldots, x_k \vert\!\rangle$	linear substitution
$\mid \mathtt{fail}^{\tilde{x}}$	failure	$\mid M\lfloor\!\lfloor U/x \rceil\!\rceil$	unrestricted substitution
$[*] ::= [\mathtt{l}] \mid [i] \quad i \in \mathbb{N}$	annotations	$A, B ::= C \star U$	bag
$U, V ::= \mathtt{1}^! \mid \wr M \int^! \mid U \diamond V$	unrestricted bag	$C, D ::= \mathtt{1} \mid \wr M \int \cdot C$	linear bag
$\mathcal{C} ::= \ \mid (\mathcal{C}\ B) \mid \mathcal{C}\langle\!\vert C/\widetilde{x} \vert\!\rangle \mid \mathcal{C}\lfloor\!\lfloor U/x \rceil\!\rceil \mid \mathcal{C}[\widetilde{x} \leftarrow x]$			context

Fig. 4. Syntax of $\lambda_{\mathtt{C}}$.

- Notation $x[\mathtt{l}]$ denotes a *linear* occurrence of x; we often omit the annotation '$[\mathtt{l}]$', and a sequence $\widetilde{x}$ (finite sequence of pairwise distinct x_i's, with length $|\widetilde{x}|$) always involves linear occurrences.
- Notation $x[i]$ denotes an *unrestricted* occurrence of x, explicitly referencing the i-th element of an unrestricted (ordered) bag. We write $x^!$ whenever the i in $x[i]$ is unimportant.

A bag is split into a linear and an unrestricted part: linear resources in bags cannot be duplicated, but unrestricted resources are always duplicated when consumed. The empty linear bag is denoted $\mathtt{1}$. We use C_i to denote the i-th term in the linear bag C; also, $\mathsf{size}(C)$ denotes the number of elements in C. To ease readability, we often write, e.g., $\wr N_1, N_2 \int$ as a shorthand notation for $\wr N_1 \int \cdot \wr N_2 \int$. The empty unrestricted bag is denoted $\mathtt{1}^!$. Notation U_i denotes the singleton bag at the i-th position in U; if there is no i-th position in U, then U_i defaults to $\mathtt{1}^!$. We use '$\star$' to combine a linear and an unrestricted bag, and unrestricted bags are joined via the non-commutative '$\diamond$'.

An abstraction $\lambda x.M$ binds occurrences of x in M. Application $(M\ C)$ is as usual. The term $M\langle\!\langle {}^{B}\!/x \rangle\!\rangle$ is an intermediate substitution, which involves a bag B with linear and unrestricted parts. To distinguish between linear and unrestricted occurrences of variables, we have two forms of explicit substitution:

- A *linear* substitution of a bag C for $\widetilde{x}$ in M is denoted $M\langle\!\vert C/\widetilde{x} \vert\!\rangle$. We require $\mathsf{size}(C) = |\widetilde{x}|$ and for each $x_i \in \widetilde{x}$: (i) x_i occurs in M; (ii) x_i is not a sharing variable; (iii) x_i cannot occur in another explicit substitution in M.
- An *unrestricted* substitution, denoted $M\lfloor\!\lfloor U/x \rceil\!\rceil$, concerns the substitution of a bag U for a variable $x^!$ in M, with the assumption that $x^!$ does not appear in another unrestricted substitution in M.

The *sharing* construct $M[x_1, \ldots, x_n \leftarrow x]$, expresses that x may be used in M under "aliases" $x_1, \ldots, x_n$. Hence, it atomizes n occurrences of x in M, via an explicit pointer to n variables. In $M[\widetilde{x} \leftarrow x]$, we say that $\widetilde{x}$ are the *shared variables* and that x is the *sharing variable*. We require for each $x_i \in \widetilde{x}$: (i) x_i occurs exactly once in M; (ii) x_i is not a sharing variable. The sequence $\widetilde{x}$ can be empty: $M[\leftarrow x]$ means that x does not share any variables in M. Sharing binds the shared variables in the term. This way, e.g., the λ-term $\lambda x.(x\ x)$ is expressed in $\lambda_{\mathtt{C}}$ as $\lambda x.(x_1 \wr x_2 \int [x_1, x_2 \leftarrow x])$, where $\wr x_2 \int$ is a bag containing x_2.

The term $\mathtt{fail}^{\widetilde{x}}$ denotes failure; the variables in $\widetilde{x}$ are "dangling" resources, which cannot be accounted for after failure. We write $\mathsf{fv}(M)$ to denote the free variables of M. Term M is *closed* if $\mathsf{fv}(M) = \emptyset$.

3.2 Reduction Semantics

Figure 5 gives the reduction semantics for $\lambda_{\mathtt{C}}$, denoted $\longrightarrow$. Rule $[\mathtt{RS:Beta}]$ induces an intermediate explicit substitution. Rule $[\mathtt{RS:Ex\text{-}Sub}]$ reduces an intermediate substitution to an explicit substitution that will manage the two-component format of bags. An explicit substitution $(M[\widetilde{x} \leftarrow x])\langle\!\langle {}^{C \star U}\!/x \rangle\!\rangle$ reduces to a term in which the linear and unrestricted parts of the bag are separated into their explicit substitutions $M\langle\!\vert C/\widetilde{x} \vert\!\rangle\lfloor\!\lfloor U/x \rceil\!\rceil$. This only occurs when the size of the bag equals the number of shared variables. The fetching of linear/unrestricted resources from their corresponding bags is done by the appropriated fetch rules $[\mathtt{RS:Fetch}^{\ell}]$ or $[\mathtt{RS:Fetch}^{!}]$. In case of a mismatch, the term evolves into failure via Rule $[\mathtt{RS:Fail}^{\ell}]$.

An explicit substitution $M\langle\!\vert C/\widetilde{x} \vert\!\rangle$, where the head variable of M is $x_j \in \widetilde{x}$, reduces via Rule $[\mathtt{RS:Fetch}^{\ell}]$. The rule extracts a C_i from C (for some $0 < i \leq \mathsf{size}(C)$) and substitutes it for x_j in M; this is how fetching induces a non-deterministic choice between $\mathsf{size}(C)$ possible reductions. The

$$\texttt{[RS:Beta]}\ \frac{}{(\lambda x.M)\ B \longrightarrow M\langle\!\langle B/x\rangle\!\rangle} \qquad \texttt{[RS:Ex-Sub]}\ \frac{\mathsf{size}(C) = |\widetilde{x}| \qquad M \neq \texttt{fail}^{\widetilde{y}}}{(M[\widetilde{x} \leftarrow x])\langle\!\langle C \star U/x\rangle\!\rangle \longrightarrow M\langle\!| C/\widetilde{x} |\!\rangle \lfloor\!\lfloor U/x \rceil\!\rceil}$$

$$\texttt{[RS:Fetch}^{\ell}\texttt{]}\ \frac{\mathsf{head}(M) = x_j \qquad 0 < i \leq \mathsf{size}(C)}{M\langle\!| C/\widetilde{x}, x_j |\!\rangle \longrightarrow (M\{C_i/x_j\})\langle\!| (C \setminus C_i)/\widetilde{x} |\!\rangle}$$

$$\texttt{[RS:Fail}^{\ell}\texttt{]}\ \frac{\mathsf{size}(C) \neq |\widetilde{x}| \qquad \widetilde{y} = (\mathsf{lfv}(M) \setminus \{\widetilde{x}\}) \cup \mathsf{lfv}(C)}{(M[\widetilde{x} \leftarrow x])\langle\!\langle C \star U/x\rangle\!\rangle \longrightarrow \texttt{fail}^{\widetilde{y}}} \qquad \texttt{[RS:Fetch}^{!}\texttt{]}\ \frac{\mathsf{head}(M) = x[i] \qquad U_i = \wr N \int^{!}}{M\lfloor\!\lfloor U/x \rceil\!\rceil \longrightarrow M\{N/x[i]\}\lfloor\!\lfloor U/x \rceil\!\rceil}$$

$$\texttt{[RS:Fail}^{!}\texttt{]}\ \frac{\mathsf{head}(M) = x[i] \qquad U_i = \mathtt{1}^{!}}{M\lfloor\!\lfloor U/x \rceil\!\rceil \longrightarrow M\{\texttt{fail}^{\emptyset}/x[i]\}\lfloor\!\lfloor U/x \rceil\!\rceil} \qquad \texttt{[RS:Cons}_1\texttt{]}\ \frac{\widetilde{y} = \mathsf{lfv}(C)}{\texttt{fail}^{\widetilde{x}}\ (C \star U) \longrightarrow \texttt{fail}^{\widetilde{x}\ \widetilde{y}}}$$

$$\texttt{[RS:Cons}_2\texttt{]}\ \frac{\mathsf{size}(C) = |\widetilde{x}| \qquad \widetilde{z} = \mathsf{lfv}(C)}{(\texttt{fail}^{\widetilde{x}\cup\widetilde{y}}[\widetilde{x} \leftarrow x])\langle\!\langle C \star U/x\rangle\!\rangle \longrightarrow \texttt{fail}^{\widetilde{y}\cup\widetilde{z}}}$$

$$\texttt{[RS:Cons}_3\texttt{]}\ \frac{\widetilde{z} = \mathsf{lfv}(C)}{\texttt{fail}^{\widetilde{y}\cup\widetilde{x}}\langle\!| C/\widetilde{x} |\!\rangle \longrightarrow \texttt{fail}^{\widetilde{y}\cup\widetilde{z}}} \qquad \texttt{[RS:Cons}_4\texttt{]}\ \frac{}{\texttt{fail}^{\widetilde{y}}\lfloor\!\lfloor U/x \rceil\!\rceil \longrightarrow \texttt{fail}^{\widetilde{y}}} \qquad \texttt{[RS : TCont]}\ \frac{M \longrightarrow N}{\mathcal{C}[M] \longrightarrow \mathcal{C}[N]}$$

where $\mathsf{head}(M)$ is defined as follows:

$$\begin{array}{llll}
\mathsf{head}(x) = x & \mathsf{head}(x[i]) = x[i] & \mathsf{head}(\lambda x.M) = \lambda x.M & \mathsf{head}((M\ C)) = \mathsf{head}(M) \\
\mathsf{head}(\texttt{fail}^{\widetilde{x}}) = \texttt{fail}^{\widetilde{x}} & \mathsf{head}(M\langle\!\langle C/x\rangle\!\rangle) = M\langle\!\langle C/x\rangle\!\rangle & \mathsf{head}(M\langle\!| C/\widetilde{x} |\!\rangle) = \mathsf{head}(M) & \mathsf{head}(M\lfloor\!\lfloor U/x \rceil\!\rceil) = \mathsf{head}(M)
\end{array}$$

$$\mathsf{head}(M[\widetilde{x} \leftarrow x]) = \begin{cases} x & \mathsf{head}(M) = y \text{ and } y \in \widetilde{x} \\ \mathsf{head}(M) & \text{otherwise} \end{cases}$$

Fig. 5. Reduction rules for λ_{C}.

reduction of an unrestricted substitution $M\lfloor\!\lfloor U/x \rceil\!\rceil$, where the head variable of M is $x[i]$, depends on U_i:

- If $U_i = \wr N \int^{!}$, then the term reduces via Rule $[\texttt{R} : \texttt{Fetch}^{!}]$ by substituting the head occurrence of $x[i]$ in M with N, denoted $M\{N/x[i]\}$; note that U_i remains available after this reduction.
- If $U_i = \mathtt{1}^{!}$, the head variable is instead substituted with failure via Rule $[\texttt{R} : \texttt{Fail}^{!}]$.

Rules $[\texttt{RS} : \texttt{Cons}_\texttt{j}]$ for $j \in \{1, 2, 3\}$ consume terms when they meet failure. Finally, Rule $[\texttt{RS} : \texttt{TCont}]$ closes reduction under contexts. The following example illustrates reduction.

Example 3.1 Consider the term $M_0 = (\lambda x.x_1 \wr x_2 \wr x_3\ \mathtt{1} \int \int [\widetilde{x} \leftarrow x])\ \wr \texttt{fail}^{\emptyset}, y, I \int$, where $I = \lambda x.(x_1[x_1 \leftarrow x])$ and $\widetilde{x} = x_1, x_2, x_3$. Here we simplify the notation of M_0 by omitting the empty unrestricted bag, i.e., we write $M \wr M' \int$ to denote $M \wr M' \int \star \mathtt{1}^{!}$. First, M_0 evolves into an intermediate substitution (2). The bag can provide for all shared variables, so it then evolves into an explicit substitution (3):

$$\begin{aligned} M_0 &\longrightarrow (x_1 \wr x_2 \wr x_3\ \mathtt{1} \int \int [\widetilde{x} \leftarrow x])\langle\!\langle \wr \texttt{fail}^{\emptyset}, y, I \int / x\rangle\!\rangle \lfloor\!\lfloor \mathtt{1}^{!}/x \rceil\!\rceil & (2) \\ &\longrightarrow (x_1 \wr x_2 \wr x_3\ \mathtt{1} \int \int)\langle\!| \wr \texttt{fail}^{\emptyset}, y, I \int / \widetilde{x} |\!\rangle \lfloor\!\lfloor \mathtt{1}^{!}/x \rceil\!\rceil = M & (3) \end{aligned}$$

Since $\mathsf{head}(M) = x_1$, one of the three elements of the bag will be substituted. M represents a non-deterministic choice between the following three reductions:

$$M \begin{array}{l} \nearrow (\texttt{fail}^{\emptyset} \wr x_2 \wr x_3\ \mathtt{1} \int \int)\langle\!| \wr y, I \int / x_2, x_3 |\!\rangle \lfloor\!\lfloor \mathtt{1}^{!}/x \rceil\!\rceil = N_1 \\ \longrightarrow (y \wr x_2 \wr x_3\ \mathtt{1} \int \int)\langle\!| \wr \texttt{fail}^{\emptyset}, I \int / x_2, x_3 |\!\rangle \lfloor\!\lfloor \mathtt{1}^{!}/x \rceil\!\rceil = N_2 \\ \searrow (I \wr x_2 \wr x_3\ \mathtt{1} \int \int)\langle\!| \wr \texttt{fail}^{\emptyset}, y \int / x_2, x_3 |\!\rangle \lfloor\!\lfloor \mathtt{1}^{!}/x \rceil\!\rceil = N_3 \end{array}$$

There are no rules for garbage collection; therefore, the normal forms maintain the explicit substitution for empty unrestricted bags.

3.3 Resource Control for $\lambda_{\mathtt{C}}$ via Intersection Types

Our type system for $\lambda_{\mathtt{C}}$ is based on non-idempotent intersection types which, As in [8,1], intersection types account for available resources in bags, which are unordered and have all the same type. Because we admit the term $\mathtt{fail}^{\tilde{x}}$ as typable, we say that our system enforces *well-formedness* rather than *well-typedness*. As we will see, well-typed terms form the sub-class of well-formed terms that does not include $\mathtt{fail}^{\tilde{x}}$.

Strict types (σ, τ, δ) and multiset types (π, ζ) are defined as follows:

$$\sigma,\tau,\delta ::= \ \mathbf{unit} \mid (\pi,\eta) \to \sigma \qquad\qquad \pi,\zeta ::= \bigwedge_{i\in I}\sigma_i \mid \omega$$

$$\eta,\epsilon ::= \ \sigma \mid \epsilon \diamond \eta \quad \text{list} \qquad\qquad (\pi,\eta) \quad \text{tuple}$$

Given a non-empty I, multiset types $\bigwedge_{i\in I}\sigma_i$ are given to bags of size $|I|$. This operator is associative, commutative, and non-idempotent (i.e., $\sigma \wedge \sigma \neq \sigma$), with identity ω. Notation σ^k stands for $\sigma \wedge \cdots \wedge \sigma$ (k times, if $k > 0$) or ω (if $k = 0$).

The list type $\epsilon \diamond \eta$ types the concatenation of unrestricted bags. It can be recursively unfolded into a finite composition of strict types $\sigma_1 \diamond \ldots \diamond \sigma_n$, for some $n \geq 1$, with length n and σ_i its i-th strict type ($1 \leq i \leq n$). We write $x^! : \eta$ to denote for $x[1] : \eta_1, \ldots, x[k] : \eta_k$ where η has length k. The tuple type (π, η) types concatenation of a linear bag of type π with an unrestricted bag of type η. Finally strict types are amended to allow for unrestricted functional types which go from tuple types to strict types $(\pi, \eta) \to \sigma$ rather then multiset types to strict types.

Definition 3.2 $[\eta \sim \epsilon]$ Let ϵ and η be two list types, with the length of ϵ greater or equal to that of η. We say that ϵ *embraces* η, denoted $\eta \sim \epsilon$, whenever there exist ϵ' and ϵ'' such that: i) $\epsilon = \epsilon' \diamond \epsilon''$; ii) the size of ϵ' is that of η; iii) for all i, $\epsilon'_i = \eta_i$.

We separate contexts into two parts: linear $(\Gamma, \Delta, \ldots)$ and unrestricted $(\Theta, \Upsilon, \ldots)$:

$$\Gamma,\Delta ::= \text{-} \mid \Gamma, x:\pi \mid \Gamma, x:\sigma \qquad\qquad \Theta,\Upsilon ::= \text{-} \mid \Theta, x^! : \eta$$

where - denotes the empty context. Both linear and unrestricted occurrences of variables may occur at most once in a context. Judgments have the form $\Theta; \Gamma \vDash M : \tau$. We write $\vDash M : \tau$ to denote -;- $\vDash M : \tau$.

We write $dom(\Gamma)$ for the set of variables in Γ. For $\Gamma, x : \pi$, we assume $x \notin dom(\Gamma)$. To avoid ambiguities, we write $x : \sigma^1$ to denote that the assignment involves a multiset type, rather than a strict type. Given Γ, its *core context* $\Gamma^{\downarrow}$ concerns variables with types different from ω; it is defined as $\Gamma^{\downarrow} = \{x : \pi \in \Gamma \mid \pi \neq \omega\}$.

Definition 3.3 [Well-formedness in $\lambda_{\mathtt{C}}$] A $\lambda_{\mathtt{C}}$-term M is *well-formed* if there exists a context Θ and Γ and a type τ such that the rules in Figure 6 entail $\Theta; \Gamma \vDash M : \tau$.

In Figure 6, Rule $[\mathtt{FS}:\mathtt{var}^{\ell}]$ types variables. Rule $[\mathtt{FS}:\mathtt{1}^{\ell}]$ types the empty bag with ω. Rule $[\mathtt{FS}:\mathtt{bag}^{\ell}]$ types the concatenation of bags. Rule $[\mathtt{FS}:\mathtt{fail}]$ types the term $\mathtt{fail}^{\widetilde{x}}$ with a strict type τ, provided that the domain of the core context coincides with $\widetilde{x}$ (i.e., no variable in $\widetilde{x}$ is typed with ω). Rule $[\mathtt{FS}:\mathtt{weak}]$ types $M[\leftarrow x]$ by weakening the context with $x : \omega$. Rule $[\mathtt{FS}:\mathtt{shar}]$ types $M[\widetilde{x} \leftarrow x]$ with τ, provided that there are assignments to the shared variables in $\widetilde{x}$.

Rule $[\mathtt{FS}:\mathtt{abs\text{-}sh}]$ types an abstraction $\lambda x.(M[\widetilde{x} \leftarrow x])$ with $\sigma^k \to \tau$, provided that $M[\widetilde{x} \leftarrow x] : \tau$ can be entailed from an assignment $x : \sigma^k$. Rule $[\mathtt{FS}:\mathtt{app}]$ types $(M\ C)$, provided that M has type $\sigma^j \to \tau$ and C has type σ^k. Note that, unlike usual intersection type systems, j and k may differ. Rule $[\mathtt{FS}:\mathtt{Esub}]$ types the intermediate substitution of a bag C of type σ^k, provided that x has type σ^j; again, j and k may differ. Rule $[\mathtt{FS}:\mathtt{Esub}^{\ell}]$ types $M \langle\!| C/\widetilde{x} |\!\rangle$ as long as C has type $\sigma^{|\widetilde{x}|}$, and each $x_i \in \widetilde{x}$ is of type σ.

Well-formed terms satisfy subject reduction (SR):

Theorem 3.4 (SR in $\lambda_{\mathtt{C}}$) *If $\Theta; \Gamma \vDash M : \tau$ and $M \longrightarrow M'$ then $\Theta; \Gamma \vDash M' : \tau$.*

From our system for well-formedness we can extract a system for *well-typed* terms, which do not include $\mathtt{fail}^{\tilde{x}}$. Judgments for well-typedness are denoted $\Gamma \vdash M : \tau$, with rules adapted from Figure 6 (the rule

$$\texttt{[FS:var}^{\ell}\texttt{]}\ \frac{}{\Theta; x : \sigma \models x : \sigma} \qquad \texttt{[FS:var}^{!}\texttt{]}\ \frac{\Theta, x^! : \eta; x : \eta_i, \Delta \models x : \sigma}{\Theta, x^! : \eta; \Delta \models x[i] : \sigma} \qquad \texttt{[FS:1}^{\ell}\texttt{]}\ \frac{}{\Theta; \text{-} \models \mathtt{1} : \omega}$$

$$\texttt{[FS:bag}^{\ell}\texttt{]}\ \frac{\Theta; \Gamma \models M : \sigma \qquad \Theta; \Delta \models C : \sigma^k}{\Theta; \Gamma, \Delta \models \wr M \int \cdot C : \sigma^{k+1}} \qquad \texttt{[FS:1}^{!}\texttt{]}\ \frac{}{\Theta; \text{-} \models \mathtt{1}^! : \sigma}$$

$$\texttt{[FS:bag}^{!}\texttt{]}\ \frac{\Theta; \text{-} \models U : \epsilon \qquad \Theta; \text{-} \models V : \eta}{\Theta; \text{-} \models U \diamond V : \epsilon \diamond \eta} \qquad \texttt{[FS:bag]}\ \frac{\Theta; \Gamma \models C : \sigma^k \qquad \Theta; \text{-} \models U : \eta}{\Theta; \Gamma \models C \star U : (\sigma^k, \eta)}$$

$$\texttt{[FS:fail]}\ \frac{dom(\Gamma^{\downarrow}) = \widetilde{x}}{\Theta; \Gamma^{\downarrow} \models \mathtt{fail}^{\widetilde{x}} : \tau} \qquad \texttt{[FS:weak]}\ \frac{\Theta; \Gamma \models M : \tau}{\Theta; \Gamma, x : \omega \models M[\leftarrow x] : \tau}$$

$$\texttt{[FS:shar]}\ \frac{\Theta; \Gamma, x_1 : \sigma, \cdots, x_k : \sigma \models M : \tau \qquad x \notin dom(\Gamma) \qquad k \neq 0}{\Theta; \Gamma, x : \sigma^k \models M[x_1, \ldots, x_k \leftarrow x] : \tau}$$

$$\texttt{[FS:abs-sh]}\ \frac{\Theta, x^! : \eta; \Gamma, x : \sigma^k \models M[\widetilde{x} \leftarrow x] : \tau \qquad x \notin dom(\Gamma)}{\Theta; \Gamma \models \lambda x.(M[\widetilde{x} \leftarrow x]) : (\sigma^k, \eta) \to \tau}$$

$$\texttt{[FS:app]}\ \frac{\Theta; \Gamma \models M : (\sigma^j, \eta) \to \tau \qquad \Theta; \Delta \models B : (\sigma^k, \epsilon) \qquad \eta \sim \epsilon}{\Theta; \Gamma, \Delta \models (M\ B) : \tau}$$

$$\texttt{[FS:Esub}^{!}\texttt{]}\ \frac{\Theta, x^! {:} \eta; \Gamma \models M : \tau \qquad \Theta; \text{-} \models U : \epsilon \qquad \eta \sim \epsilon}{\Theta; \Gamma \models M \llbracket U/x \rrbracket : \tau}$$

$$\texttt{[FS:Esub]}\ \frac{\Theta, x^! : \eta; \Gamma, x : \sigma^j \models M[\widetilde{x} \leftarrow x] : \tau \qquad \Theta; \Delta \models B : (\sigma^k, \epsilon) \qquad \eta \sim \epsilon}{\Theta; \Gamma, \Delta \models (M[\widetilde{x} \leftarrow x])\langle\!\langle B/x \rangle\!\rangle : \tau}$$

$$\texttt{[FS:Esub}^{\ell}\texttt{]}\ \frac{\Theta; \Gamma, x_1 : \sigma, \cdots, x_k : \sigma \models M : \tau \qquad \Theta; \Delta \models C : \sigma^k}{\Theta; \Gamma, \Delta \models M \langle\!| C/x_1, \cdots, x_k |\!\rangle : \tau}$$

Fig. 6. Well-Formedness Rules for $\lambda_{\mathtt{C}}$.

name prefix FS is replaced with TS), with the following modifications: (i) There is no rule [TS:fail]; (ii) Rules [TS:app] and [TS:Esub] do not allow a mismatch between variables and resources. This way, e.g.,

$$\texttt{[TS:Esub]}\ \frac{\Theta, x^! : \eta; \Gamma, x : \sigma^j \vdash M[\widetilde{x} \leftarrow x] : \tau \qquad \Theta; \Delta \vdash B : (\sigma^k, \eta)}{\Theta; \Gamma, \Delta \vdash (M[\widetilde{x} \leftarrow x])\langle\!\langle B/x \rangle\!\rangle : \tau}$$

For the sake of completeness, the full set of rules is in Figure 7. Well-typed terms are also well-formed, and thus satisfy SR. Moreover, well-typed terms also satisfy subject expansion (SE).

Theorem 3.5 (SE in $\lambda_{\mathtt{C}}$) *If $\Theta; \Gamma \vdash M' : \tau$ and $M \longrightarrow M'$ then $\Theta; \Gamma \vdash M : \tau$.*

4 Translating $\lambda_{\mathtt{C}}$ into $\mathsf{s}\pi^!$

Clearly, $\mathsf{s}\pi^!$ and $\lambda_{\mathtt{C}}$ are different models. In particular, $\lambda_{\mathtt{C}}$ is a programming calculus in which implicit non-determinism implements fetching of linear and unrestricted resources. To illustrate the potential of $\mathsf{s}\pi^!$ to precisely model non-determinism as found in realistic programs/protocols using an eager approach, we follow and extend the approach in [14], and give a translation of $\lambda_{\mathtt{C}}$ into $\mathsf{s}\pi^!$ which preserves types (Theorem 4.2) and respects well-known criteria for dynamic correctness (Definition 4.3).

4.1 The translation

Given a $\lambda_{\mathtt{C}}$-term M, its translation into $\mathsf{s}\pi^!$ is denoted $\llbracket M \rrbracket_u$ and given in Figure 8. As in Milner's classical translation: every variable x in M becomes a name x in process $\llbracket M \rrbracket_u$, where name u provides the behavior of M. To handle failures in $\lambda_{\mathtt{C}}$, u is a non-deterministically session: the translated term can be available or not, as signaled by prefixes $\overline{u}.\mathtt{some}$ and $\overline{u}.\mathtt{none}$, respectively. As a result, reductions from $\llbracket M \rrbracket_u$ include synchronizations that codify M's behavior but also synchronizations that confirm a session's availability.

We discuss Figure 8, focusing on constructs related to unrestricted resources, not considered in [14]. The translation of an unrestricted variable $x[j]$ first connects to a server along channel x via a request $?\overline{x^!}[x_i]$ followed by a selection on $\overline{x_i}.j$. Process $\llbracket \lambda x.(M[\widetilde{x} \leftarrow x]) \rrbracket_u$ first confirms its behavior along u, followed by the reception of a channel x. The channel x provides a linear channel x^{ℓ} and an unrestricted

$$\frac{}{\Theta; x:\sigma \vdash x:\sigma}\ [\mathtt{TS{:}var}^{\ell}] \qquad \frac{\Theta, x^{!}:\eta; x:\eta_i, \Delta \vdash x:\sigma}{\Theta, x^{!}:\eta; \Delta \vdash x[i]:\sigma}\ [\mathtt{TS{:}var}^{!}] \qquad \frac{}{\Theta; \text{-} \vdash \mathtt{1}:\omega}\ [\mathtt{TS{:}1}^{\ell}] \qquad \frac{\Theta; \Gamma \vdash M:\tau}{\Theta; \Gamma, x:\omega \vdash M[\leftarrow x]:\tau}\ [\mathtt{TS{:}weak}]$$

$$\frac{\Theta, x^{!}:\eta; \Gamma, x:\sigma^{k} \vdash M[\widetilde{x} \leftarrow x]:\tau \qquad x \notin dom(\Gamma)}{\Theta; \Gamma \vdash \lambda x.(M[\widetilde{x} \leftarrow x]):(\sigma^{k},\eta) \rightarrow \tau}\ [\mathtt{TS{:}abs\text{-}sh}] \qquad \frac{\Theta; \Gamma \vdash M:\sigma \qquad \Theta; \Delta \vdash C:\sigma^{k}}{\Theta; \Gamma, \Delta \vdash \wr M \int \cdot C:\sigma^{k+1}}\ [\mathtt{TS{:}bag}^{\ell}]$$

$$\frac{\Theta; \Gamma \vdash M:(\sigma^{j},\eta) \rightarrow \tau \qquad \Theta; \Delta \vdash B:(\sigma^{j},\eta)}{\Theta; \Gamma, \Delta \vdash M\ B:\tau}\ [\mathtt{TS{:}app}] \qquad \frac{\Theta; \text{-} \vdash U:\epsilon \qquad \Theta; \text{-} \vdash V:\eta}{\Theta; \text{-} \vdash U \diamond V:\epsilon \diamond \eta}\ [\mathtt{TS{:}bag}^{!}]$$

$$\frac{\Theta; \Gamma, x_1:\sigma, \ldots, x_k:\sigma \vdash M:\tau \qquad x \notin dom(\Gamma) \qquad k \neq 0}{\Theta; \Gamma, x:\sigma^{k} \vdash M[x_1, \ldots, x_k \leftarrow x]:\tau}\ [\mathtt{TS{:}shar}] \qquad \frac{\Theta; \Gamma \vdash C:\sigma^{k} \qquad \Theta; \text{-} \vdash U:\eta}{\Theta; \Gamma \vdash C \star U:(\sigma^{k},\eta)}\ [\mathtt{TS{:}bag}]$$

$$\frac{\Theta; \Gamma, x_1:\sigma, \ldots, x_k:\sigma \vdash M:\tau \quad \Theta; \Delta \vdash C:\sigma^{k}}{\Theta; \Gamma, \Delta \vdash M \langle\!| C/x_1, \ldots, x_k |\!\rangle:\tau}\ [\mathtt{TS{:}Esub}^{\ell}] \qquad \frac{\Theta, x^{!}{:}\eta; \Gamma \vdash M:\tau \qquad \Theta; \text{-} \vdash U:\eta}{\Theta; \Gamma \vdash M \llbracket U/x^{!} \rrbracket:\tau}\ [\mathtt{TS{:}Esub}^{!}]$$

$$\frac{\Theta, x^{!}:\eta; \Gamma, x:\sigma^{j} \vdash M[\widetilde{x} \leftarrow x]:\tau \qquad \Theta; \Delta \vdash B:(\sigma^{k},\eta)}{\Theta; \Gamma, \Delta \vdash (M[\widetilde{x} \leftarrow x])\langle\!\langle B/x \rangle\!\rangle:\tau}\ [\mathtt{TS{:}Esub}]$$

Fig. 7. Well-Typed Rules for $\lambda_{\mathtt{C}}$.

channel $x^{!}$ for dedicated substitutions of the linear and unrestricted bag components. This separation is also present in the translation of $\llbracket M\langle\!\langle B/x \rangle\!\rangle \rrbracket_u$, for the same reason.

Process $\llbracket M\ (C \star U)_u \rrbracket$ consists of synchronizations between the translation of $\llbracket M \rrbracket_v$ and $\llbracket C \star U \rrbracket_x$: the translation of $C \star U$ evolves when M is an abstraction, say $\lambda x.(M'[\widetilde{x} \leftarrow x])$. The channel x^{ℓ} provides the linear behavior of the bag C while $x^{!}$ provides the behavior of U. This is done by guarding the translation of U with a server connection: every time a channel synchronizes with it a fresh copy of U is spawned.

As in [14], non-deterministic choices occur in the translations of $M\langle\!| C/\widetilde{x} |\!\rangle$ (explicit substitutions) and $M[\widetilde{x} \leftarrow x]$ (non-empty sharing). Roughly speaking, the position of $\mathbin{+\!\!\!+}$ in the translation of $M\langle\!| C/\widetilde{x} |\!\rangle$ represents the most desirable way of mimicking the fetching of terms from a bag. This use of $\mathbin{+\!\!\!+}$ is a central idea in our translation: as we explain below, it allows for appropriate commitment in non-deterministic choices, but also for *delayed* commitment when necessary.

For simplicity, we consider explicit substitutions $M\langle\!| C/\widetilde{x} |\!\rangle$ where $C = \wr N_1, N_2 \int$ and $\widetilde{x} = x_1, x_2$. The translation $\llbracket M\langle\!| C/\widetilde{x} |\!\rangle \rrbracket_u$ uses the processes $\llbracket N_i \rrbracket_{z_i}$, where each z_i is fresh. First, each bag item confirms its behavior. Then, a variable $x_i \in \widetilde{x}$ is chosen non-deterministically; we ensure that these choices consider all variables. Note that writing $\mathbin{+\!\!\!+}_{x_i \in \{x_1,x_2\}} \mathbin{+\!\!\!+}_{x_j \in \{x_1,x_2\} \setminus x_i}$ is equivalent to non-deterministically assigning x_i, x_j to each permutation of x_1, x_2. The resulting choice involves $\llbracket M \rrbracket_u$ with x_i, x_j substituted by z_1, z_2. Commitment here is triggered only via synchronizations along z_1 or z_2; synchronizing with $z_i.\mathtt{some}_{\mathsf{fv}(N_i)}; \llbracket N_i \rrbracket_{z_i}$ then represents fetching N_i from the bag.

The process $\llbracket M[\widetilde{x} \leftarrow x] \rrbracket_u$ first confirms its behavior along x. Then it sends a name y_i on x, on which a failed reduction may be handled. Next, the translation confirms again its behavior along x and non-deterministically receives a reference to an $x_i \in \widetilde{x}$. Each branch consists of $\llbracket M[(\widetilde{x} \setminus x_i) \leftarrow x] \rrbracket_u$. The possible choices are permuted, represented by $\mathbin{+\!\!\!+}_{x_i \in \widetilde{x}}$. Synchronizations with $\llbracket M[(\widetilde{x} \setminus x_i) \leftarrow x] \rrbracket_u$ and bags delay commitment in this choice (we return to this point below). The process $\llbracket M[\leftarrow x] \rrbracket_u$ is similar but simpler: here the name x fails, as it cannot take further elements to substitute.

Process $\llbracket M \llbracket U/x \rrbracket \rrbracket_u$ consists of the composition of the translation of M and a server guarding the

$$
\begin{aligned}
\llbracket x \rrbracket_u &= \overline{x}.\mathtt{some}; [x \leftrightarrow u] \qquad\qquad \llbracket x[j] \rrbracket_u = ?\overline{x^!}[x_i]; \overline{x_i}.j; [x_i \leftrightarrow u] \\
\llbracket \lambda x.M \rrbracket_u &= \overline{u}.\mathtt{some}; u(x); \overline{x}.\mathtt{some}; x(x^\ell); x(x^!); x(); \llbracket M \rrbracket_u \\
\llbracket M \langle\!\langle C \star U / x \rangle\!\rangle \rrbracket_u &= (\boldsymbol{\nu} x)(\overline{x}.\mathtt{some}; x(x^\ell); x(x^!); x(); \llbracket M \rrbracket_u \mid \llbracket C \star U \rrbracket_x) \\
\llbracket M (C \star U) \rrbracket_u &= (\boldsymbol{\nu} v)(\llbracket M \rrbracket_v \mid v.\mathtt{some}_{u,\mathsf{lfv}(C)}; \overline{v}[x]; (\llbracket C \star U \rrbracket_x \mid [v \leftrightarrow u])) \\
\llbracket C \star U \rrbracket_x &= x.\mathtt{some}_{\mathsf{lfv}(C)}; \overline{x}[x^\ell]; \big(\llbracket C \rrbracket_{x^\ell} \mid \overline{x}[x^!]; (!x^!(x_i); \llbracket U \rrbracket_{x_i} \mid \overline{x}[])\big) \\
\llbracket \wr M_j \int \cdot C \rrbracket_{x^\ell} &= x^\ell.\mathtt{some}_{\mathsf{lfv}(C)}; x(y_i); x^\ell.\mathtt{some}_{y_i,\mathsf{lfv}(C)}; \overline{x^\ell}.\mathtt{some}; \\
&\qquad \overline{x^\ell}[z_i]; (z_i.\mathtt{some}_{\mathsf{lfv}(M_j)}; \llbracket M_j \rrbracket_{z_i} \mid (\llbracket (C \setminus M_j) \rrbracket_{x^\ell} \mid \overline{y_i}.\mathtt{none})) \\
\llbracket \mathtt{1} \rrbracket_{x^\ell} &= x^\ell.\mathtt{some}_\emptyset; x(y_n); (\overline{y_n}.\mathtt{some}; \overline{y_n}[] \mid x^\ell.\mathtt{some}_\emptyset; \overline{x^\ell}.\mathtt{none}) \\
\llbracket \mathtt{1}^! \rrbracket_x = \overline{x}.\mathtt{none} \qquad & \llbracket \wr N \int^! \rrbracket_x = \llbracket N \rrbracket_x \qquad \llbracket U \rrbracket_x = x.\mathtt{case}\{i : \llbracket U_i \rrbracket_x\}_{U_i \in U} \\
\llbracket M \langle\!| \wr M_1 \int \cdot \wr M_2 \int / x_1, x_2 |\!\rangle \rrbracket_u &= (\boldsymbol{\nu} z_1)(z_1.\mathtt{some}_{\mathsf{lfv}(M_1)}; \llbracket M_1 \rrbracket_{z_1} \mid (\boldsymbol{\nu} z_2)(z_2.\mathtt{some}_{\mathsf{lfv}(M_2)}; \llbracket M_2 \rrbracket_{z_2} \\
&\qquad \mid \textstyle\bigparallel_{x_{i_1} \in \{x_1, x_2\}} \bigparallel_{x_{i_2} \in \{x_1, x_2 \setminus x_{i_1}\}} \llbracket M \rrbracket_u \{z_1 / x_{i_1}\}\{z_2 / x_{i_2}\}) \ldots) \\
\llbracket M \llbracket U / x^! \rrbracket \rrbracket_u &= (\boldsymbol{\nu} x^!)(\llbracket M \rrbracket_u \mid !x^!(x_i); \llbracket U \rrbracket_{x_i}) \\
\llbracket M[\leftarrow x] \rrbracket_u &= \overline{x^\ell}.\mathtt{some}; \overline{x^\ell}[y_i]; (y_i.\mathtt{some}_{u,\mathsf{lfv}(M)}; y_i(); \llbracket M \rrbracket_u \mid \overline{x^\ell}.\mathtt{none}) \\
\llbracket M[\widetilde{x} \leftarrow x] \rrbracket_u &= \overline{x^\ell}.\mathtt{some}; \overline{x^\ell}[y_i]; \big(y_i.\mathtt{some}_\emptyset; y_i(); \mathbf{0} \\
&\qquad \mid \overline{x^\ell}.\mathtt{some}; x^\ell.\mathtt{some}_{u,\mathsf{lfv}(M) \setminus \widetilde{x}}; \textstyle\bigparallel_{x_i \in \widetilde{x}} x^\ell(x_i); \llbracket M[(\widetilde{x} \setminus x_i) \leftarrow x] \rrbracket_u\big) \\
\llbracket \mathtt{fail}^{x_1, \ldots, x_k} \rrbracket_u &= \overline{u}.\mathtt{none} \mid \overline{x_1}.\mathtt{none} \mid \ldots \mid \overline{x_k}.\mathtt{none}
\end{aligned}
$$

Fig. 8. Translation of λ_{C} into $\mathsf{s}\pi^!$.

translation of U: in order for $\llbracket M \rrbracket_u$ to gain access to $\llbracket U \rrbracket_{x_i}$ it must first synchronize with the server channel $x^!$ to spawn a fresh copy of the translation of U. In case of a failure (i.e., a mismatch between the size of the bag C and the number of variables in M), our translation ensures that the confirmations of C will not succeed. This is how failure in λ_{C} is correctly translated to failure in $\mathsf{s}\pi^!$.

4.2 Static Correctness: Type Preservation

Our translation preserves types: intersection types in λ_{C} are translated to session types in $\mathsf{s}\pi^!$ (Figure 9). The translation of types describes how non-deterministic fetches are codified as non-deterministic session protocols. As discussed in Section 2.4, session types effectively abstract away from the behavior of processes, as all branches of a non-deterministic choice use the same typing context. Thus, it is expected that the translation of types remains unchanged w.r.t. those in [9,14]. Our translation enjoys *static correctness*: well-formed terms in λ_{C} translate to well-typed processes in $\mathsf{s}\pi^!$. We need the following definition.

Definition 4.1 Let $\Gamma = x_1 : \sigma_1, \ldots, x_m : \sigma_m, v_1 : \pi_1, \ldots, v_n : \pi_n$ be a linear context. Also, consider the unrestricted context $\Theta = x^![1] : \eta_1, \ldots, x^![k] : \eta_k$. Translation $\llbracket \cdot \rrbracket__$ in Figure 9 extends to Γ, Θ as follows:

$$
\begin{aligned}
\llbracket \Gamma \rrbracket &= x_1 : \&\overline{\llbracket \sigma_1 \rrbracket}, \ldots, x_m : \&\overline{\llbracket \sigma_m \rrbracket}, v_1 : \overline{\llbracket \pi_1 \rrbracket_{(\sigma, i_1)}}, \ldots, v_n : \overline{\llbracket \pi_n \rrbracket_{(\sigma, i_n)}} \\
\llbracket \Theta \rrbracket &= x^![1] : \overline{\llbracket \eta_1 \rrbracket}, \ldots, x^![k] : \overline{\llbracket \eta_k \rrbracket}
\end{aligned}
$$

Theorem 4.2 (Type Preservation) *Let B and M be a bag and an term in λ_{C}, respectively.*

(i) *If $\Theta; \Gamma \vDash B : (\sigma^k, \eta)$ then $\llbracket B \rrbracket_u \vdash \llbracket \Gamma \rrbracket, u : \llbracket (\sigma^k, \eta) \rrbracket_{(\sigma, i)}, \llbracket \Theta \rrbracket$.*

(ii) *If $\Theta; \Gamma \vDash M : \tau$ then $\llbracket M \rrbracket_u \vdash \llbracket \Gamma \rrbracket, u : \llbracket \tau \rrbracket, \llbracket \Theta \rrbracket$.*

$$
\begin{aligned}
[\![\mathbf{unit}]\!] &= \&\mathbf{1} & [\![\eta]\!] &= !\&_{\eta_i \in \eta}\{i : [\![\eta_i]\!]\} \\
[\![(\sigma^k, \eta) \to \tau]\!] &= \&(\overline{[\![(\sigma^k, \eta)]\!]_{(\sigma,i)}} \invamp [\![\tau]\!]) & [\![(\sigma^k, \eta)]\!]_{(\sigma,i)} &= \oplus(([\![\sigma^k]\!]_{(\sigma,i)}) \otimes (([\![\eta]\!]) \otimes (\mathbf{1}))) \\
[\![\sigma \wedge \pi]\!]_{(\sigma,i)} &= \oplus((\&\mathbf{1}) \invamp (\oplus\&((\oplus[\![\sigma]\!]) \otimes ([\![\pi]\!]_{(\sigma,i)})))) \\
[\![\omega]\!]_{(\sigma,i)} &= \begin{cases} \oplus((\&\mathbf{1}) \invamp (\oplus\&\mathbf{1})) & \text{if } i = 0 \\ \oplus((\&\mathbf{1}) \invamp (\oplus\&((\oplus[\![\sigma]\!]) \otimes ([\![\omega]\!]_{(\sigma,i-1)})))) & \text{if } i > 0 \end{cases}
\end{aligned}
$$

Fig. 9. Translation of intersection types into session types (cf. Def. 4.1).

4.3 Dynamic Correctness Under the Eager Semantics

To state *dynamic correctness*, we rely on established notions that (abstractly) characterize *correct translations* [4,10,11]. A language $\mathcal{L} = (L, \to)$ consists of a set of terms L and a reduction relation $\to$ on L. Each language $\mathcal{L}$ is assumed to contain a success constructor $\checkmark$. A term $T \in L$ has *success*, denoted $T \Downarrow \checkmark$, when there is a sequence of reductions (using $\to$) from T to a term satisfying success criteria.

Given $\mathcal{L}_1 = (L_1, \to_1)$ and $\mathcal{L}_2 = (L_2, \to_2)$, we seek translations $[\![\cdot]\!] : L_1 \to L_2$ that are correct, i.e., translations that satisfy well-known correctness criteria. The next definition formulates such criteria.

Definition 4.3 (Correct Translation) Let $\mathcal{L}_1 = (\mathcal{M}, \to_1)$ and $\mathcal{L}_2 = (\mathcal{P}, \to_2)$ be two languages. Let $\asymp_2$ be an equivalence over $\mathcal{L}_2$. We use M, M' (resp. P, P') to range over terms in $\mathcal{M}$ (resp. $\mathcal{P}$). Given a translation $[\![\cdot]\!] : \mathcal{M} \to \mathcal{P}$, we define:

Completeness: For every M, M' such that $M \to_1^* M'$, there exists P such that $[\![M]\!] \to_2^* P \asymp_2 [\![M']\!]$.

Weak Soundness: For every M and P such that $[\![M]\!] \to_2^* P$, there exist M', P' such that $M \to_1^* M'$ and $P \to_2^* P' \asymp_2 [\![M']\!]$.

Success Sensitivity: For every M, we have $M \Downarrow \checkmark$ if and only if $[\![M]\!] \Downarrow \checkmark$.

Hence, to prove that our translation of λ_{C} into $\mathsf{s}\pi^!$ is correct, we need to instantiate Definition 4.3 appropriately. It turns out that to prove that our translation is correct, we need to instantiate $\asymp_2$ with a pre-congruence on processes, denoted $\succeq_{\mathbin{\#}}$, defined as follows:

$$
\frac{}{P \succeq_{\mathbin{\#}} P} \qquad \frac{P_i \succeq_{\mathbin{\#}} P_i' \quad {\scriptstyle i \in \{1,2\}}}{P_1 \mathbin{\#} P_2 \succeq_{\mathbin{\#}} P_i'} \qquad \frac{P \succeq_{\mathbin{\#}} P' \quad Q \succeq_{\mathbin{\#}} Q'}{P \mid Q \succeq_{\mathbin{\#}} P' \mid Q'} \qquad \frac{P \succeq_{\mathbin{\#}} P'}{(\boldsymbol{\nu} x)P \succeq_{\mathbin{\#}} (\boldsymbol{\nu} x)P'}
$$

Intuitively, $P \succeq_{\mathbin{\#}} Q$ says that P has at least as many branches as Q. We have the following properties, which are instances of those stated in Definition 4.3:

Theorem 4.4 (Loose Completeness) *If $N \longrightarrow M$ for a well-formed closed λ_{C}-term N, then there exists Q such that $[\![N]\!]_u \longrightarrow^* Q$ and $[\![M]\!]_u \succeq_{\mathbin{\#}} Q$.*

Theorem 4.5 (Loose Weak Soundness) *If $[\![N]\!]_u \longrightarrow^* Q$ for a well-formed closed λ_{C}-term N, then there exist N' and Q' such that (i) $N \longrightarrow^* N'$ and (ii) $Q \longrightarrow^* Q'$ with $[\![N']\!]_u \succeq_{\mathbin{\#}} Q'$.*

Theorem 4.6 (Success Sensitivity) *$M \Downarrow \checkmark_\lambda$ iff $[\![M]\!]_u \Downarrow \checkmark_\pi$ for well-formed closed terms M.*

Translation correctness up to $\succeq_{\mathbin{\#}}$ thus means that $\longrightarrow$ is "too eager", as it prematurely commits to branches. In contrast, the translation correctness properties established in [14] under the lazy semantics hold by instantiating $\asymp_2$ simply with $\equiv$. We thus conclude that moving from the (complex) lazy semantics of [14] to the (simpler) eager semantics of Section 2.3 has a concrete effect in encodability properties: while translation correctness in the lazy regime is *tight*, it becomes *loose* in the eager regime.

Example 4.7 To further contrast commitment in eager and lazy semantics (and their effect on the translation's correctness), recall from Example 3.1 the term M (3) and the three branching reductions from M to N_1, N_2 and N_3. Figure 10 depicts a side-by-side comparison of the reductions of $[\![M]\!]_u$ under the lazy

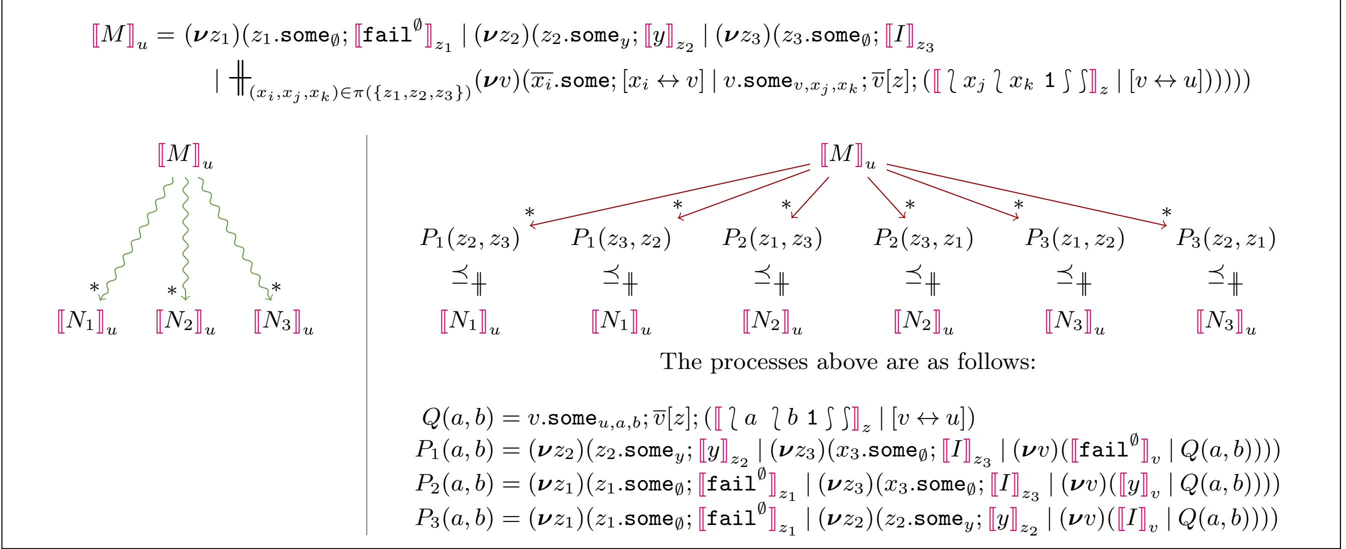

Fig. 10. Example 4.7: Reductions of $[\![M]\!]_u$ under $\rightsquigarrow_S$ and $\longrightarrow$. We write '$\pi(X)$' for the permutations of set X.

($\rightsquigarrow$) and eager ($\longrightarrow$) semantics. We omit the translation of the empty unrestricted bag $[\![\lfloor 1^! / x \rceil]\!]_u$ in the translations of $[\![M]\!]_u$ and each of the $[\![N_i]\!]_u$, since it does not add any insight. In the figure, $\rightsquigarrow^*$ and $\longrightarrow^*$ denote the reflexive, transitive closures of $\rightsquigarrow$ and $\longrightarrow$, respectively.

Under $\rightsquigarrow$ there are three reduction paths, each resulting directly in the translation of one of N_1, N_2, N_3: after the first choice, the following choices are preserved. In contrast, under $\longrightarrow$ there are six reduction paths, each resulting in a process that relates to the translation of one of N_1, N_2, N_3 through $\succeq_{\#}$: after the first choice for an item from the bag is made, the semantics commits to choices for the other items.

5 Comparing Lazy and Eager Semantics via Behavioral Equivalences

We now compare $\rightsquigarrow$ and $\longrightarrow$ *independently from* $\lambda_{\mathtt{C}}$ by resorting to *behavioral equivalences*. We define a simple behavioral notion of equivalence on $\mathsf{s}\pi^!$ processes, parametric in $\rightsquigarrow$ or $\longrightarrow$; then, we prove that there are classes of processes that are equal with respect to $\rightsquigarrow$, but incomparable with respect to $\longrightarrow$ (Theorem 5.4). A key ingredient is the following notion of observable on processes:

Definition 5.1 A process P has a *ready-prefix* α, denoted $P \downarrow_\alpha$, iff there exist $\mathtt{N}, P'$ such that $P \equiv \mathtt{N}[\alpha; P']$.

Definition 5.2 (Ready-Prefix Bisimilarity) A relation $\mathbb{B}$ on $\mathsf{s}\pi^!$ processes is a *(strong) ready-prefix bisimulation with respect to* $\rightsquigarrow$ if and only if, for every $(P, Q) \in \mathbb{B}$,

(i) For every P' such that $P \rightsquigarrow P'$, there exists Q' such that $Q \rightsquigarrow Q'$ and $(P', Q') \in \mathbb{B}$;

(ii) For every Q' such that $Q \rightsquigarrow Q'$, there exists P' such that $P \rightsquigarrow P'$ and $(P', Q') \in \mathbb{B}$;

(iii) For every $\alpha \bowtie \beta$, $P \downarrow_\alpha$ if and only if $Q \downarrow_\beta$.

P and Q are *ready-prefix bisimilar with respect to* $\rightsquigarrow$, denoted $P \sim_{\mathtt{L}} Q$, if there exists a relation $\mathbb{B}$ that is a ready-prefix bisimulation with respect to $\rightsquigarrow$ such that $(P, Q) \in \mathbb{B}$.

A *(strong) ready-prefix bisimulation with respect to* $\longrightarrow$ is defined by replacing every occurrence of '$\rightsquigarrow$' by '$\longrightarrow$' in the definition above. We write $P \sim_{\mathtt{E}} Q$ if P and Q are *ready-prefix bisimilar with respect to* $\longrightarrow$.

Ready-prefix bisimulation can highlight a significant difference between the behavior induced the lazy and eager semantics. To illustrate this, we consider session-typed implementations of a vending machine.

Example 5.3 (Two Vending Machines) Consider vending machines VM_1 and VM_2 consisting of three parts: (1) an interface, which interacts with the user to send money and choose between coffee (c) and tea (t); (2) a brewer, which produces either beverage; (3) a system, which collects the money and forwards

the user's choice to the brewer. An $\mathsf{s}\pi^!$ specification follows (below € and €2 stand for names):

$$\mathsf{VM}_1 := (\boldsymbol{\nu}x)\big(\mathsf{IF}_1 \mid (\boldsymbol{\nu}y)(\mathsf{Brewer} \mid \mathsf{System})\big) \qquad \mathsf{VM}_2 := (\boldsymbol{\nu}x)\big(\mathsf{IF}_2 \mid (\boldsymbol{\nu}y)(\mathsf{Brewer} \mid \mathsf{System})\big)$$

$$\mathsf{IF}_1 := \overline{x}[€2]; \big(\overline{€2}[] \mid (\overline{x}.\mathsf{c}; \overline{x}[] \mathbin{+\!\!\!\!+} \overline{x}.\mathsf{t}; \overline{x}[])\big) \qquad \mathsf{IF}_2 := \overline{x}[€2]; \big(\overline{€2}[] \mid \overline{x}.\mathsf{c}; \overline{x}[]\big) \mathbin{+\!\!\!\!+} \overline{x}[€2]; \big(\overline{€2}[] \mid \overline{x}.\mathsf{t}; \overline{x}[]\big)$$

$$\mathsf{System} := x(€); x.\mathsf{case}\begin{Bmatrix}\mathsf{c} : \overline{y}.\mathsf{c}; x(); €(); \overline{y}[], \\ \mathsf{t} : \overline{y}.\mathsf{t}; x(); €(); \overline{y}[]\end{Bmatrix} \qquad \mathsf{Brewer} := y.\mathsf{case}\{\mathsf{c} : y(); \mathsf{Brew}_\mathsf{c}, \mathsf{t} : y(); \mathsf{Brew}_\mathsf{t}\}$$

where $\mathsf{Brew}_\mathsf{c} \vdash \emptyset$, $\mathsf{Brew}_\mathsf{t} \vdash \emptyset$, such that $\mathsf{VM}_1 \vdash \emptyset$, $\mathsf{VM}_2 \vdash \emptyset$.

The machines VM_1 and VM_2 are based on two different implementations of the interface. Machine VM_1 uses IF_1, which sends the money and then chooses coffee or tea. Machine VM_2 uses IF_2, which chooses sending the money and then requesting coffee, or sending the money and then requesting tea.

We have $\mathsf{VM}_1 \not\sim_{\mathtt{E}} \mathsf{VM}_2$: the eager semantics distinguishes the machines; e.g., IF_1 has a single money slot, a button for coffee, and another button for tea, whereas IF_2 has two money slots, one for coffee, and another for tea. In contrast, the machines are indistinguishable under the lazy semantics: $\mathsf{VM}_1 \sim_{\mathtt{L}} \mathsf{VM}_2$.

Example 5.3 highlights a difference in behavior between $\rightsquigarrow$ and $\longrightarrow$ when a moment of choice is subtly altered. The following theorem captures this distinction; we need an auxiliary definition. Let $\bowtie$ denote the least relation on prefixes defined by: (i) $\overline{x}[y] \bowtie \overline{x}[z]$, (ii) $x(y) \bowtie x(z)$, and (iii) $\alpha \bowtie \alpha$ otherwise.

Theorem 5.4 *Take $R \equiv \mathtt{N}[\alpha_1; (P \mathbin{+\!\!\!\!+} Q)] \vdash \emptyset$ and $S \equiv \mathtt{N}[\alpha_2; P \mathbin{+\!\!\!\!+} \alpha_3; Q] \vdash \emptyset$, where $\alpha_1 \bowtie \alpha_2 \bowtie \alpha_3$ and $\alpha_1, \alpha_2, \alpha_3$ require a continuation. Suppose that $P \not\sim_{\mathtt{L}} Q$ and $P \not\sim_{\mathtt{E}} Q$. Then (i) $R \sim_{\mathtt{L}} S$ but (ii) $R \not\sim_{\mathtt{E}} S$.*

6 Conclusion

We studied how to reconcile non-deterministic choice and linearity in a session-typed π-calculus. We extended and complemented the results in [14], which were based on a lazy semantics for expressing commitment in non-deterministic choices. Our central contribution is an eager semantics that more directly expresses such commitment while respecting linearity. We confirmed that this eager semantics fits well with the session type discipline: both type preservation and deadlock-freedom properties are still ensured in the eager regime. We also showed the expressivity of our typed model by giving a correct translation of a resource λ-calculus, which extends the analogous results in [14] with unrestricted resources. We compared the two semantics in two ways: (i) via the correctness properties they induce for the translation of resource λ-calculi, and (ii) via a simple behavioral equivalence that captures different moments of choice. Based on our results, we conclude that eager and lazy semantics are both worth studying, as each of them has merits and shortcomings: the lazy semantics is complex but admits fine-grained observations; the new eager semantics has a simpler definition but induces commitment that can be sometimes too premature.

Acknowledgments

We are grateful to the anonymous reviewers for useful comments on previous versions of this paper. This research has been supported by the Dutch Research Council (NWO) under project No. 016.Vidi.189.046 ('Unifying Correctness for Communicating Software') and the EPSRC Fellowship 'VeTSpec: Verified Trustworthy Software Specification' (EP/R034567/1).

References

[1] Boudol, G. and C. Laneve, *Lambda-calculus, multiplicities, and the pi-calculus*, in: *Proof, Language, and Interaction, Essays in Honour of Robin Milner*, pages 659–690 (2000).

[2] Caires, L. and J. A. Pérez, *Linearity, control effects, and behavioral types*, in: H. Yang, editor, *Programming Languages and Systems - 26th European Symposium on Programming, ESOP 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, volume 10201 of *Lecture Notes in Computer Science*, pages 229–259, Springer (2017), ISBN 978-3-662-54433-4.
https://doi.org/10.1007/978-3-662-54434-1_9

[3] Caires, L. and F. Pfenning, *Session types as intuitionistic linear propositions*, in: *CONCUR 2010 - Concurrency Theory, 21th International Conference, CONCUR 2010, Paris, France, August 31-September 3, 2010. Proceedings*, pages 222–236 (2010).
https://doi.org/10.1007/978-3-642-15375-4_16

[4] Gorla, D., *Towards a unified approach to encodability and separation results for process calculi*, Inf. Comput. **208**, pages 1031–1053 (2010).
https://doi.org/10.1016/j.ic.2010.05.002

[5] Honda, K., *Types for dyadic interaction*, in: E. Best, editor, *CONCUR '93, 4th International Conference on Concurrency Theory, Hildesheim, Germany, August 23-26, 1993, Proceedings*, volume 715 of *Lecture Notes in Computer Science*, pages 509–523, Springer (1993).
https://doi.org/10.1007/3-540-57208-2_35

[6] Honda, K., V. T. Vasconcelos and M. Kubo, *Language primitives and type discipline for structured communication-based programming*, in: C. Hankin, editor, *Programming Languages and Systems - ESOP'98, 7th European Symposium on Programming, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS'98, Lisbon, Portugal, March 28 - April 4, 1998, Proceedings*, volume 1381 of *Lecture Notes in Computer Science*, pages 122–138, Springer (1998), ISBN 3-540-64302-8.
https://doi.org/10.1007/BFb0053567

[7] Milner, R., J. Parrow and D. Walker, *A calculus of mobile processes, I*, Inf. Comput. **100**, pages 1–40 (1992).
https://doi.org/10.1016/0890-5401(92)90008-4

[8] Pagani, M. and S. Ronchi Della Rocca, *Solvability in resource lambda-calculus*, in: C. L. Ong, editor, *Foundations of Software Science and Computational Structures, 13th International Conference, FOSSACS 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings*, volume 6014 of *Lecture Notes in Computer Science*, pages 358–373, Springer (2010), ISBN 978-3-642-12031-2.
https://doi.org/10.1007/978-3-642-12032-9_25

[9] Paulus, J. W. N., D. Nantes-Sobrinho and J. A. Pérez, *Non-deterministic functions as non-deterministic processes*, in: N. Kobayashi, editor, *6th International Conference on Formal Structures for Computation and Deduction, FSCD 2021, July 17-24, 2021, Buenos Aires, Argentina (Virtual Conference)*, volume 195 of *LIPIcs*, pages 21:1–21:22, Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021). Extended version on https://arxiv.org/abs/2104.14759.
https://doi.org/10.4230/LIPIcs.FSCD.2021.21

[10] Peters, K., *Translational Expressiveness. Comparing Process Calculi using Encodings*, Ph.D. thesis, Berlin Institute of Technology (2012).
https://doi.org/10.14279/depositonce-3416

[11] Peters, K., *Comparing process calculi using encodings*, in: J. A. Pérez and J. Rot, editors, *Proceedings Combined 26th International Workshop on Expressiveness in Concurrency and 16th Workshop on Structural Operational Semantics, EXPRESS/SOS 2019, Amsterdam, The Netherlands, 26th August 2019*, volume 300 of *EPTCS*, pages 19–38 (2019).
https://doi.org/10.4204/EPTCS.300.2

[12] Sangiorgi, D. and D. Walker, *The Pi-Calculus - a theory of mobile processes*, Cambridge University Press (2001), ISBN 978-0-521-78177-0.

[13] van den Heuvel, B., J. W. N. Paulus, D. Nantes-Sobrinho and J. A. Pérez, *Typed non-determinism in functional and concurrent calculi (extended version)*, CoRR **abs/2205.00680** (2022). 2205.00680.
https://doi.org/10.48550/arXiv.2205.00680

[14] van den Heuvel, B., J. W. N. Paulus, D. Nantes-Sobrinho and J. A. Pérez, *Typed non-determinism in functional and concurrent calculi*, in: C. Hur, editor, *Programming Languages and Systems - 21st Asian Symposium, APLAS 2023, Taipei, Taiwan, November 26-29, 2023, Proceedings*, volume 14405 of *Lecture Notes in Computer Science*, pages 112–132, Springer (2023).
https://doi.org/10.1007/978-981-99-8311-7_6

[15] Wadler, P., *Propositions as sessions*, in: P. Thiemann and R. B. Findler, editors, *ACM SIGPLAN International Conference on Functional Programming, ICFP'12, Copenhagen, Denmark, September 9-15, 2012*, pages 273–286, ACM (2012).
https://doi.org/10.1145/2364527.2364568