Typed Non-determinism in Functional and Concurrent Calculi

van den Heuvel, Bas; Paulus, Joseph W.N.; Nantes-Sobrinho, Daniele; Pérez, Jorge A.

Published in:
Programming Languages and Systems - 21st Asian Symposium, APLAS 2023, Proceedings

DOI:
10.1007/978-981-99-8311-7_6

Typed Non-determinism in Functional and Concurrent Calculi

Bas van den Heuvel[1], Joseph W. N. Paulus[1,2], Daniele Nantes-Sobrinho[3,4], and Jorge A. Pérez[1(✉)]

[1] University of Groningen, Groningen, The Netherlands
j.a.perez@rug.nl
[2] University of Oxford, Oxford, UK
[3] University of Brasília, Brasília, Brazil
[4] Imperial College London, London, UK

Abstract. We study functional and concurrent calculi with non-determinism, along with type systems to control resources based on linearity. The interplay between non-determinism and linearity is delicate: careless handling of branches can discard resources meant to be used exactly once. Here we go beyond prior work by considering non-determinism in its standard sense: once a branch is selected, the rest are discarded. Our technical contributions are three-fold. First, we introduce a π-calculus with non-deterministic choice, governed by session types. Second, we introduce a resource λ-calculus, governed by intersection types, in which non-determinism concerns fetching of resources from bags. Finally, we connect our two typed non-deterministic calculi via a correct translation.

1 Introduction

In this paper, we present new formulations of typed programming calculi with *non-determinism*. A classical ingredient of models of computation, non-determinism brings flexibility and generality in specifications. In process calculi such as CCS and the π-calculus, one source of non-determinism is choice, which is typically *non-confluent*: that is, given $P + Q$, we have either $P + Q \longrightarrow P$ or $P + Q \longrightarrow Q$. Thus, committing to a branch entails discarding the rest.

We study non-determinism as a way of increasing the expressivity of typed calculi in which resource control is based on *linearity*. The interplay between non-determinism and linearity is delicate: a careless discarding of branches can jeopardize resources meant to be used exactly once. On the concurrent side, we consider the π-calculus, the paradigmatic model of concurrency [27]. We focus on π-calculi with *session types* [14,15], in which linear logic principles ensure communication correctness: here the resources are names that perform session protocols; they can be *unrestricted* (used multiple times) and *linear* (used exactly once). To properly control resources, non-confluent non-determinism is confined to unrestricted names; linear names can only perform deterministic choices.

In this context, considering *confluent* forms of non-determinism can be appealing. Intuitively, such formulations allow all branches to proceed independently: given $P_1 \longrightarrow Q_1$ and $P_2 \longrightarrow Q_2$, then $P_1 + P_2 \longrightarrow Q_1 + P_2$ and

C.-K. Hur (Ed.): APLAS 2023, LNCS 14405, pp. 112–132, 2023.
https://doi.org/10.1007/978-981-99-8311-7_6

$P_1 + P_2 \longrightarrow P_1 + Q_2$. Because confluent non-determinism does not discard branches, it is compatible with a resource-conscious view of computation.

Confluent non-determinism has been studied mostly in the functional setting; it is present, e.g., in Pagani and Ronchi della Rocca's resource λ-calculus [21] and in Ehrhard and Regnier's differential λ-calculus [10]. In [21], non-determinism resides in the application of a term M to a *bag* of available resources C; a β-reduction applies M to a resource *non-deterministically fetched* from C. Confluent non-deterministic choice is also present in the session-typed π-calculus by Caires and Pérez [5], where it expresses a choice between different implementations of the same session protocols, which are all *non-deterministically available*—they may be available but may also *fail*. In their work, a Curry-Howard correspondence between linear logic and session types ('*propositions-as-sessions*' [6,31]) ensures confluence, protocol fidelity, and deadlock-freedom. Paulus et al. [22] relate functional and concurrent calculi with confluent non-determinism: they give a translation of a resource λ-calculus into the session π-calculus from [5], in the style of Milner's '*functions-as-processes*' [17].

Although results involving confluent non-determinism are most significant, usual (non-confluent) non-determinism remains of undiscussed convenience in formal modeling; consider, e.g., specifications of distributed protocols [2,20] in which commitment is essential. Indeed, non-confluent non-deterministic choice is commonplace in verification frameworks such as mCRL2 [12]. It is also relevant in functional calculi; a well-known framework is De'Liguoro and Piperno's (untyped) non-deterministic λ-calculus [8] (see also [9] and references therein).

To further illustrate the difference between confluent and non-confluent non-determinism, we consider an example adapted from [5]: a movie server that offers a choice between buying a movie or watching its trailer. In $\mathsf{s}\pi^+$, the typed π-calculus that we present in this paper, this server can be specified as follows:

$$\mathsf{Server}_s = s.\mathtt{case}\left\{\begin{array}{l}\mathsf{buy} : s(\mathit{title}); s(\mathit{paym}); \overline{s}[\mathtt{movie}]; \overline{s}[]\ , \\ \mathsf{peek} : s(\mathit{title}); \overline{s}[\mathtt{trailer}]; \overline{s}[]\end{array}\right\} \begin{array}{l}-\mathsf{Server}_s^{\mathsf{buy}} \\ -\mathsf{Server}_s^{\mathsf{peek}}\end{array}$$

where $s(-)$ and $\overline{s}[-]$ denote input and output prefixes on a name/channel s, respectively, and '`movie`' and '`trailer`' denote references to primitive data. Also, the free names of a process are denoted with subscripts. Process Server_s offers a choice on name s ($s.\mathtt{case}\{-\}$) between labels buy and peek. If buy is received, process $\mathsf{Server}_s^{\mathsf{buy}}$ is launched: it receives the movie's title and a payment method, sends the movie, and closes the session on s ($\overline{s}[]$). If peek is received, it proceeds as $\mathsf{Server}_s^{\mathsf{peek}}$: the server receives the title, sends the trailer, and closes the session.

Using the non-deterministic choice operator of $\mathsf{s}\pi^+$, denoted '$\mathbin{+\!\!+}$', we can specify a process for a client Alice who is interested in the movie 'Jaws' but is undecided about buying the film or just watching its trailer for free:

$$\begin{array}{rll}\mathsf{Alice}_s := & \overline{s}.\mathsf{buy};\ \overline{s}[\mathtt{Jaws}]; \overline{s}[\mathtt{mcard}]; s(\mathit{movie}); s(); \mathbf{0} & -\mathsf{Alice}_s^{\mathsf{buy}} \\ \mathbin{+\!\!+} & \overline{s}.\mathsf{peek}; \overline{s}[\mathtt{Jaws}]; s(\mathit{trailer}); s(); \mathbf{0} & -\mathsf{Alice}_s^{\mathsf{peek}}\end{array}$$

If Alice_s selects the label buy ($\overline{s}.\mathsf{buy}$), process $\mathsf{Alice}_s^{\mathsf{buy}}$ is launched: it sends title and payment method, receives the movie, waits for the session to close ($s()$),

and then terminates ($\mathbf{0}$). If Alice_s selects peek, process $\mathsf{Alice}_s^{\mathsf{peek}}$ is launched: it sends a title, receives the trailer, waits for the session to close, and terminates. Then, process $\mathsf{Sys} := (\boldsymbol{\nu} s)(\mathsf{Server}_s \mid \mathsf{Alice}_s)$ denotes the composition of client and server, connected along s (using $(\boldsymbol{\nu} s)$). Our semantics for $\mathsf{s}\pi^+$, denoted $\rightsquigarrow$, enforces non-confluent non-determinism, as Sys can reduce to separate processes, as expected:

$$\mathsf{Sys} \rightsquigarrow (\boldsymbol{\nu} s)(\mathsf{Server}_s^{\mathsf{buy}} \mid \mathsf{Alice}_s^{\mathsf{buy}}) \qquad \text{and} \qquad \mathsf{Sys} \rightsquigarrow (\boldsymbol{\nu} s)(\mathsf{Server}_s^{\mathsf{peek}} \mid \mathsf{Alice}_s^{\mathsf{peek}})$$

In contrast, the confluent non-deterministic choice from [5], denoted '$\oplus$', behaves differently: in their confluent semantics, Sys reduces to a *single* process including *both alternatives*, i.e., $\mathsf{Sys} \longrightarrow (\boldsymbol{\nu} s)(\mathsf{Server}_s^{\mathsf{buy}} \mid \mathsf{Alice}_s^{\mathsf{buy}}) \oplus (\boldsymbol{\nu} s)(\mathsf{Server}_s^{\mathsf{peek}} \mid \mathsf{Alice}_s^{\mathsf{peek}})$.

Contributions. We study new concurrent and functional calculi with usual (non-confluent) forms of non-determinism. Framed in the typed (resource-conscious) setting, we strive for definitions that do not exert a too drastic discarding of branches (as in the non-confluent case) but also that do not exhibit a lack of commitment (as in the confluent case). Concretely, we present:

(Section 2) $\mathsf{s}\pi^+$, a variant of the session-typed π-calculus in [5], now with non-confluent non-deterministic choice. Its semantics adapts to the typed setting the usual semantics of non-deterministic choice in the untyped π-calculus [27]. Well-typed processes enjoy type preservation and deadlock-freedom (Theorems 1 and 2).

(Section 3) $\lambda_{\mathsf{C}}^{\ell}$, a resource λ-calculus with non-determinism, enhanced with constructs for expressing resource usage and failure. Its non-idempotent intersection type system provides a quantitative measure of the need/usage of resources. Well-typed terms enjoy subject reduction and subject expansion (Theorems 3 and 4).

(Section 4) A typed translation of $\lambda_{\mathsf{C}}^{\ell}$ into $\mathsf{s}\pi^+$, which provides further validation for our non-deterministic calculi, and casts them in the context of 'functions-as-processes'. We prove that our translation is *correct*, i.e., it preserves types and satisfies tight operational correspondences (Theorems 5 and 6).

Moreover, Sect. 5 closes by discussing related works. Omitted technical material can be found in the full version of the paper [13].

2 A Typed π-calculus with Non-deterministic Choice

We introduce $\mathsf{s}\pi^+$, a session-typed π-calculus with non-deterministic choice. Following [5], session types express protocols to be executed along channels. These protocols can be *non-deterministic*: sessions may succeed but also fail. The novelty in $\mathsf{s}\pi^+$ is the non-deterministic choice operator '$P \parallel Q$', whose *lazily committing semantics* is compatible with linearity. We prove that well-typed processes satisfy two key properties: *type preservation* and *deadlock-freedom*.

$$
\begin{array}{rlrl}
P, Q ::= \mathbf{0} & \text{inaction} & \mid [x \leftrightarrow y] & \text{forwarder} \\
\mid (\boldsymbol{\nu} x)(P \mid Q) & \text{connect} & \mid P \parallel Q & \text{non-determinism} \\
\mid \overline{x}[y]; (P \mid Q) & \text{output} & \mid x(y); P & \text{input} \\
\mid \overline{x}.\ell; P & \text{select} & \mid x.\mathtt{case}\{i : P\}_{i \in I} & \text{branch} \\
\mid \overline{x}[] & \text{close} & \mid x(); P & \text{wait} \\
\mid x.\mathtt{some}_{w_1,\ldots,w_n}; P & \text{expect} & \mid \overline{x}.\mathtt{some}; P & \text{available} \\
\mid P \mid Q & \text{parallel} & \mid \overline{x}.\mathtt{none} & \text{unavailable}
\end{array}
$$

$$
\begin{array}{ccc}
P \equiv P' \ [P =_\alpha P'] & [x \leftrightarrow y] \equiv [y \leftrightarrow x] & P \mid \mathbf{0} \equiv P \\
(P \mid Q) \mid R \equiv P \mid (Q \mid R) & P \mid Q \equiv Q \mid P & (\boldsymbol{\nu} x)(P \mid Q) \equiv (\boldsymbol{\nu} x)(Q \mid P) \\
P \parallel P \equiv P & P \parallel Q \equiv Q \parallel P & (P \parallel Q) \parallel R \equiv P \parallel (Q \parallel R)
\end{array}
$$

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

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

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

2.1 Syntax and Semantics

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. Parallel composition appears in two forms: 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 implement complementary behaviors on x and do not share any other names.

Process $P \parallel Q$ 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 $\parallel$ is associative, we often omit parentheses. Also, we write $\parallel_{i \in I} P_i$ for the non-deterministic choice between each P_i for $i \in I$.

Our output construct integrates parallel composition and restriction: process $\overline{x}[y]; (P|Q)$ sends a fresh name y along x and then continues as $P|Q$. The type system will ensure that behaviors on y and x are implemented by P and Q, respectively, which do not share any names—this separation defines communication-free concurrency and is key to ensuring deadlock-freedom. The input process $x(y); P$ receives a name z along x and continues as $P\{z/y\}$, which does not require the separation present in the output case. 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. We omit replicated servers $!x(y); P$ and corresponding client requests $?\overline{x}[y]; P$, but they can be easily added (cf. [13]).

The remaining constructs define non-deterministic sessions which may provide a protocol or fail, following [5]. Process $\overline{x}.\texttt{some}; P$ confirms the availability of a session on x and continues as P. Process $\overline{x}.\texttt{none}$ signals the failure to provide the session on x. Process $x.\texttt{some}_{w_1,\ldots,w_n}; P$ 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 '$\overline{x}.\texttt{some}$' and continue as P, or (ii) synchronize with a '$\overline{x}.\texttt{none}$', discard P, and propagate the failure to $w_1, \ldots, w_n$. To reduce eye strain, in writing $x.\texttt{some}$ we freely combine names and sets of names. This way, e.g., we write $x.\texttt{some}_{y,fn(P),fn(Q)}$ rather than $x.\texttt{some}_{\{y\}\cup fn(P)\cup fn(Q)}$.

Name y is bound in $(\boldsymbol{\nu} y)(P \mid Q)$, $\overline{x}[y]; (P \mid Q)$, and $x(y); P$. The set $fn(P)$ includes the names in P that are not bound. We adopt Barendregt's convention.

Structural Congruence. Reduction defines the steps that a process performs on its own. It relies on *structural congruence* ($\equiv$), the least congruence relation on processes induced by the rules in Fig. 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 1). Differently from [5], we do not allow distributing non-deterministic choice over parallel and restriction. As shown in [13], the position of a non-deterministic choice in a process determines how it may commit, so changing its position affects commitment.

Reduction: Intuitions and Prerequisites. Barring non-deterministic choice, our reduction rules arise as directed interpretations of proof transformations in the underlying linear logic. We follow Caires and Pfenning [6] and Wadler [31] in interpreting cut-elimination in linear logic as synchronization in $\mathsf{s}\pi^+$.

Before delving into our reduction rules (Fig. 2), it may be helpful to consider the usual reduction axiom for the (untyped) π-calculus (e.g., [19,27]):

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

This axiom captures the interaction of two (binary) choices: it integrates the commitment of choice in synchronization; after the reduction step, the two branches not involved in the synchronization, M_1 and M_2, are discarded. Our semantics of $\mathsf{s}\pi^+$ is defined similarly: when a prefix within a branch of a choice synchronizes with its dual, that branch reduces and the entire process commits to it.

The key question at this point is: when and to which branches should we commit? In (1), a communication commits to a single branch. For $\mathsf{s}\pi^+$, we define a *lazy semantics* that minimizes commitment as much as possible.

The intuitive idea is that multiple branches of a choice may contain the same prefix, and so all these branches represent possibilities for synchronization ("possible branches"). Other branches with different prefixes denote different possibilities ("impossible branches"). When one synchronization is chosen, the possible branches are maintained while the impossible ones are discarded.

Example 1. To distinguish possible and impossible branches, consider:

$$P := (\boldsymbol{\nu} s)\big(s.\mathsf{case}\{\mathsf{buy} : \ldots, \mathsf{peek} : \ldots\} \mid (\overline{s}.\mathsf{buy}; \ldots + \overline{s}.\mathsf{buy}; \ldots + \overline{s}.\mathsf{peek}; \ldots)\big)$$

The branch construct (case) provides the context for the non-deterministic choice. When the case synchronizes on the 'buy' label, the two branches prefixed by '$\overline{s}.\mathsf{buy}$' are possible, whereas the branch prefixed by '$\overline{s}.\mathsf{peek}$' becomes impossible, and can be discarded. The converse occurs when the 'peek' label is selected. ▽

To formalize these intuitions, our reduction semantics (Fig. 2) relies on some auxiliary definitions. First, we define contexts.

Definition 1. *We define* ND-contexts *($\mathtt{N},\mathtt{M}$) as follows:*

$$\mathtt{N},\mathtt{M} ::= [\cdot] \mid \mathtt{N} \mid P \mid (\boldsymbol{\nu} x)(\mathtt{N} \mid P) \mid \mathtt{N} \nparallel P$$

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 '$\mathtt{N} \nparallel P$' as D-contexts*, denoted $\mathtt{C},\mathtt{D}$.*

Using D-contexts, we can express that, e.g., $\nparallel_{i\in I} \mathtt{C}_i[\overline{x}[]]$ and $\nparallel_{j\in J} \mathtt{D}_j[x(); Q_j]$ should match. To account for reductions with impossible branches, we define a precongruence on processes, denoted $\succeq_S$, where the parameter S denotes the subject(s) of the prefix in the possible branches. Our semantics is closed under $\succeq_S$. Hence, e.g., anticipating a reduction on x, the possible branch $\mathtt{C}_1[x(y); P]$ can be extended with an impossible branch to form $\mathtt{C}_1[x(y); P] \nparallel \mathtt{C}_2[z(); Q]$.

Before defining $\succeq_S$ (Definition 3), we first define prefixes (and their subjects). Below, we write $\widetilde{x}$ to denote a finite tuple of names $x_1, \ldots, x_k$.

Definition 2. *Prefixes are defined as follows:*

$$\alpha,\beta ::= \overline{x}[y] \mid x(y) \mid \overline{x}.\ell \mid x.\mathsf{case} \mid \overline{x}[] \mid x() \mid \overline{x}.\mathsf{some} \mid \overline{x}.\mathsf{none} \mid x.\mathsf{some}_{\widetilde{w}} \mid [x \leftrightarrow y]$$

The subjects of α, denoted $subj\{\alpha\}$, are $\{x, y\}$ in case of $[x \leftrightarrow y]$, or $\{x\}$. By abuse of notation, we write $\alpha; P$ even when α takes no continuation (as in $\overline{x}[]$, $\overline{x}.\mathsf{none}$, and $[x \leftrightarrow y]$) and for $\overline{x}[y]$ which takes a parallel composition as continuation.

Definition 3. *Let $\bowtie$ denote the least relation on prefixes (Definition 2) defined by:*
(i) $\overline{x}[y] \bowtie \overline{x}[z]$, (ii) $x(y) \bowtie x(z)$, and (iii) $\alpha \bowtie \alpha$ otherwise.

Given a non-empty set $S \subseteq \{x, y\}$, the precongruence $P \succeq_S Q$ holds when both following conditions hold:

1. *$S = \{x\}$ implies*
   $P = \left(\nparallel_{i\in I} \mathtt{C}_i[\alpha_i; P_i]\right) \nparallel \left(\nparallel_{j\in J} \mathtt{C}_j[\beta_j; Q_j]\right)$ and $Q = \nparallel_{i\in I} \mathtt{C}_i[\alpha_i; P_i]$, where
   (i) $\forall i, i' \in I.\, \alpha_i \bowtie \alpha_{i'}$ and $subj\{\alpha_i\} = \{x\}$, and
   (ii) $\forall i \in I. \forall j \in J.\, \alpha_i \not\bowtie \beta_j \wedge x \in fn(\beta_j; Q_j)$;
2. *$S = \{x, y\}$ implies*
   $P = \left(\nparallel_{i\in I} \mathtt{C}_i[[x \leftrightarrow y]]\right) \nparallel \left(\nparallel_{j\in J} \mathtt{C}_j[[x \leftrightarrow z_j]]\right) \nparallel \left(\nparallel_{k\in K} \mathtt{C}_k[\alpha_k; P_k]\right)$
   and $Q = \nparallel_{i\in I} \mathtt{C}_i[[x \leftrightarrow y]]$, where
   (i) $\forall j \in J.\, z_j \neq y$, and (ii) $\forall k \in K.\, x \in fn(\alpha_k; P_k) \wedge \forall z.\, \alpha_k \not\bowtie [x \leftrightarrow z]$.

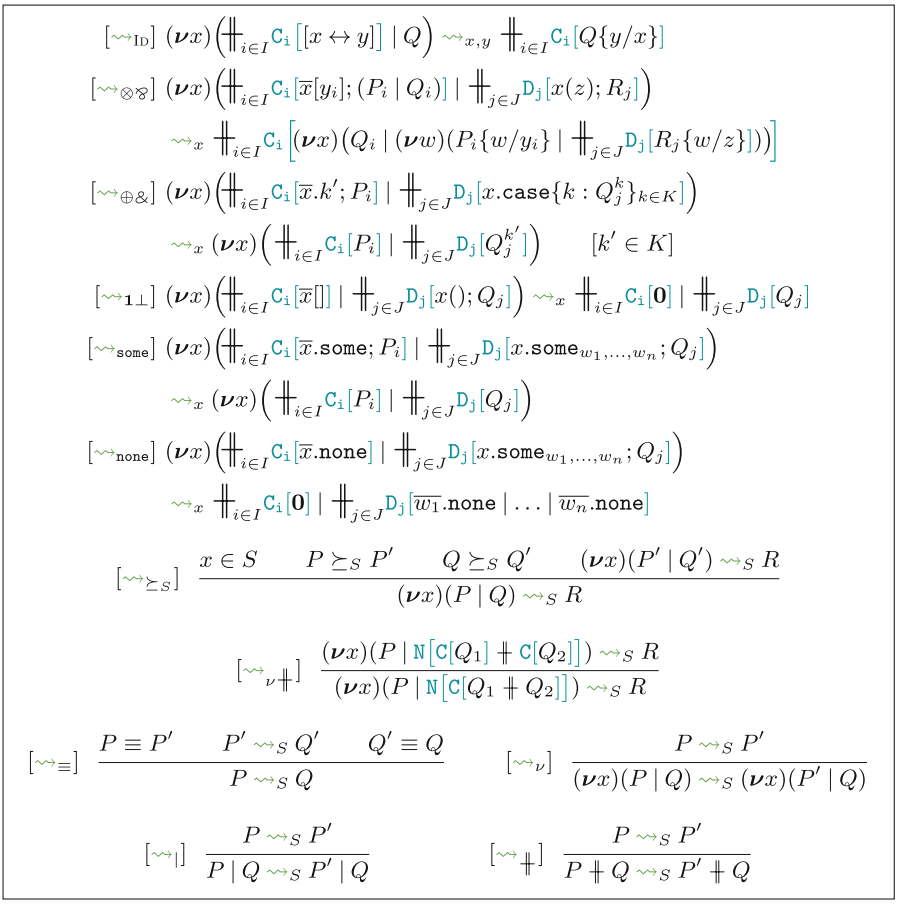

Fig. 2. Reduction semantics for $\mathsf{s}\pi^+$.

Intuitively, $\bowtie$ allows us to equate output/input prefixes with the same subject (but different object). The rest of Definition 3 accounts for two kinds of reduction, using S to discard "impossible" branches. In case S is $\{x\}$ (Item 1), it concerns a synchronization on x; in case S is $\{x, y\}$, it concerns forwarding on x and y (Item 2). In both cases, P and Q contain matching prefixes on x, while P may contain additional branches with different or blocked prefixes on x; x must appear in the hole of the contexts in the additional branches in P (enforced with $x \in \mathit{fn}(\ldots)$), to ensure that no matching prefixes are discarded.

Example 2. Recall process P from Example 1. To derive a synchronization with the 'buy' alternative of the case, we can use $\succeq_S$ to discard the 'peek' alternative, as follows: $\overline{s}.\mathsf{buy}; \ldots \Vdash \overline{s}.\mathsf{buy}; \ldots \Vdash \overline{s}.\mathsf{peek}; \ldots \;\succeq_s\; \overline{s}.\mathsf{buy}; \ldots \Vdash \overline{s}.\mathsf{buy}; \ldots$ ▽

Reduction Rules. Figure 2 gives the rules for the (lazy) reduction semantics, denoted $\rightsquigarrow_S$, where the set S contains the names involved in the interaction. We

omit the curly braces in this annotation; this way, e.g., we write '$\rightsquigarrow_{x,y}$' instead of '$\rightsquigarrow_{\{x,y\}}$'. Also, we write $\rightsquigarrow_S^k$ to denote a sequence of $k \geq 0$ reductions.

The first six rules in Fig. 2 formalize forwarding and communication: they are defined on choices containing different D-contexts (cf. Definition 1), each with the same prefix but possibly different continuations; these rules preserve the non-deterministic choices. Rule [$\rightsquigarrow_{\text{ID}}$] fixes S to the forwarder's two names, and the other rules fix S to the one involved name. In particular, Rule [$\rightsquigarrow_{\otimes\parr}$] formalizes name communication: it involves multiple senders and multiple receivers (grouped in choices indexed by I and J, respectively). Because they proceed in lock-step, reduction leads to substitutions involving the same (fresh) name w; also, the scopes of the choice and the contexts enclosing the senders is extended.

Rule [$\rightsquigarrow_{\succeq_S}$] is useful to derive a synchronization that discards groups of choices. Rule [$\rightsquigarrow_{\nu\parallel}$] allows inferring reductions when non-deterministic choices are not top-level: e.g., $(\boldsymbol{\nu} x)\big(\overline{x}[] \mid (\boldsymbol{\nu} y)((x(); Q_1 \parallel x(); Q_2) \mid R)\big) \rightsquigarrow_x (\boldsymbol{\nu} y)(Q_1 \mid R) \parallel (\boldsymbol{\nu} y)(Q_2 \mid R)$. The last four rules formalize that reduction is closed under structural congruence, restriction, parallel composition, and non-deterministic choice.

As mentioned earlier, a key motivation for our work is to have non-deterministic choices that effectively enforce commitment, without a too drastic discarding of alternatives. Next we illustrate this intended form of *gradual commitment*.

Example 3. (A Modified Movie Server). Consider the following variant of the movie server from the introduction, where the handling of the payment is now modeled as a branch:

$$\mathsf{NewServer}_s := s(\mathit{title}); s.\mathtt{case}\left\{\begin{array}{l}\mathsf{buy} : s.\mathtt{case}\left\{\begin{array}{l}\mathsf{card} : s(\mathit{info}); \overline{s}[\mathtt{movie}]; \overline{s}[],\\ \mathsf{cash} : \overline{s}[\mathtt{movie}]; \overline{s}[]\end{array}\right\},\\ \mathsf{peek} : \overline{s}[\mathtt{trailer}]; \overline{s}[]\end{array}\right\}$$

Consider a client, Eve, who cannot decide between buying 'Oppenheimer' or watching its trailer. In the former case, she has two options for payment method:

$$\mathsf{Eve}_s := \overline{s}[\mathtt{Oppenheimer}]; \left(\begin{array}{l}\overline{s}.\mathsf{buy}; \overline{s}.\mathsf{card}; \overline{s}[\mathtt{visa}]; s(\mathit{movie}); s(); \mathbf{0}\\ \parallel \overline{s}.\mathsf{buy}; \overline{s}.\mathsf{cash}; s(\mathit{movie}); s(); \mathbf{0}\\ \parallel \overline{s}.\mathsf{peek}; s(\mathit{link}); s(); \mathbf{0}\end{array}\right)$$

Let $\mathsf{Sys}^* := (\boldsymbol{\nu} s)(\mathsf{NewServer}_s \mid \mathsf{Eve}_s)$. After sending the movie's title, Eve's choice (buying or watching the trailer) enables gradual commitment. We have:

$$\mathsf{Sys}^* \rightsquigarrow_s^2 (\boldsymbol{\nu} s)\big(s.\mathtt{case}\{\mathsf{card} : \ldots, \mathsf{cash} : \ldots\} \mid (\overline{s}.\mathsf{card}; \ldots \parallel \overline{s}.\mathsf{cash}; \ldots)\big) =: \mathsf{Sys}_1^*$$

$$\text{and} \qquad \mathsf{Sys}^* \rightsquigarrow_s^2 (\boldsymbol{\nu} s)(\overline{s}[\mathtt{trailer}]; \ldots \mid s(\mathit{trailer}); \ldots) =: \mathsf{Sys}_2^*$$

Process Sys_1^* represents the situation for Eve after selecting buy, in which case the third alternative ($\overline{s}.\mathsf{peek}; \ldots$) can be discarded as an impossible branch. Process Sys_2^* represents the dual situation. From Sys_1^*, the selection of payment method completes the commitment to one alternative; we have: $\mathsf{Sys}_1^* \rightsquigarrow_s (\boldsymbol{\nu} s)(s(\mathit{info}); \ldots \mid \overline{s}[\mathtt{visa}]; \ldots)$ and $\mathsf{Sys}_1^* \rightsquigarrow_s (\boldsymbol{\nu} s)(\overline{s}[\mathtt{movie}]; \ldots \mid s(\mathit{movie}); \ldots)$. ▽

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

$$[\text{T}\parallel]\ \frac{P \vdash \Gamma \qquad Q \vdash \Gamma}{P \parallel Q \vdash \Gamma} \qquad [\text{Tempty}]\ \frac{}{\mathbf{0} \vdash \emptyset} \qquad [\text{Tid}]\ \frac{}{[x \leftrightarrow y] \vdash x{:}A, y{:}\overline{A}}$$

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

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

$$[\text{T}\&]\ \frac{\forall i \in I.\ P_i \vdash \Gamma, x{:}A_i}{x.\mathtt{case}\{i : P_i\}_{i \in I} \vdash \Gamma, x{:}\&\{i : A_i\}_{i \in I}} \qquad [\text{T}\&\mathtt{some}]\ \frac{P \vdash \Gamma, x{:}A}{\overline{x}.\mathtt{some}; P \vdash \Gamma, x{:}\&A}$$

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

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

2.2 Resource Control for $\mathsf{s}\pi^+$ via Session Types

We define a session type system for $\mathsf{s}\pi^+$, following 'propositions-as-sessions' [6, 31]. 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:

$$A, B ::= \mathbf{1} \mid \bot \mid A \otimes B \mid A \parr B \mid \oplus\{i : A\}_{i \in I} \mid \&\{i : A\}_{i \in I} \mid \&A \mid \oplus A$$

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, $A \parr B$ 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. 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, dual types serve 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:

$$\overline{\mathbf{1}} = \bot \qquad \overline{A \otimes B} = \overline{A} \parr \overline{B} \qquad \overline{\oplus\{i : A_i\}_{i \in I}} = \&\{i : \overline{A_i}\}_{i \in I} \qquad \overline{\&A} = \oplus\overline{A}$$

$$\overline{\bot} = \mathbf{1} \qquad \overline{A \parr B} = \overline{A} \otimes \overline{B} \qquad \overline{\&\{i : A_i\}_{i \in I}} = \oplus\{i : \overline{A_i}\}_{i \in I} \qquad \overline{\oplus A} = \&\overline{A}$$

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 [31]), with the rules for $\&A$ and $\oplus A$ extracted from [5], and the additional Rule $[T\!\!+\!\!]$ for non-confluent non-deterministic choice, which modifies the confluent rule in [5].

Most rules follow [31], so we focus on those related to non-determinism. 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 $[T\!\!+\!\!]$ types our new 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. Compared to the rule for non-determinism in [5], we do not require processes to be typable under the $\&A$ monad.

Example 4. Consider again process Eve_s from Example 3. The three branches of the non-deterministic choice give *different implementations of the same session*: assuming primitive, self-dual data types $\mathtt{C}$, $\mathtt{M}$, and $\mathtt{L}$, all three branches on s are typable by $\oplus\{\mathsf{buy} : \oplus\{\mathsf{card} : \mathtt{C} \otimes \mathtt{M} \parr \bot, \mathsf{cash} : \mathtt{M} \parr \bot\}, \mathsf{peek} : \mathtt{L} \parr \bot\}$. ▽

Example 5 (Unavailable Movies). Consider now a modified movie server, which offers movies that may not be yet available. We specify this server using non-deterministic choice and non-deterministically available sessions:

$$\mathsf{BuyServ}_s := s(\mathit{title}); (\overline{s}.\mathtt{none} + \overline{s}.\mathtt{some}; s(\mathit{paym}); \overline{s}[\mathtt{movie}]; \overline{s}[]) \vdash s : \mathtt{T} \parr (\&(\mathtt{P} \parr \mathtt{M} \otimes \mathbf{1})),$$

where $\mathtt{T}, \mathtt{P}, \mathtt{M}$ denote primitive, self-dual data-types. While the branch '$\overline{s}.\mathtt{none}$' signals that the movie is not available, the branch '$\overline{s}.\mathtt{some}; \ldots$' performs the expected protocol. We now define a client Ada who buys a movie for Tim, using session s; Ada only forwards it to him (using session u) if it is actually available:

$$\begin{aligned} \mathsf{Ada}_{s,u} &:= \overline{s}[\mathtt{Barbie}]; s.\mathtt{some}_u; \overline{s}[\mathtt{visa}]; s(\mathit{movie}); s(); \overline{u}.\mathtt{some}; \overline{u}[\mathit{movie}]; \overline{u}[] \\ &\quad \vdash s : \mathtt{T} \otimes \big(\oplus (\mathtt{P} \otimes \mathtt{M} \parr \bot) \big), u : \&(\mathtt{M} \otimes \mathbf{1}) \\ \mathsf{Tim}_u &:= u.\mathtt{some}; u(\mathit{movie}); u(); \mathbf{0} \vdash u : \oplus(\mathtt{M} \parr \mathbf{1}) \end{aligned}$$

Let $\mathsf{BuySys} := (\boldsymbol{\nu} s)\big(\mathsf{BuyServ}_s \mid (\boldsymbol{\nu} u)(\mathsf{Ada}_{s,u} \mid \mathsf{Tim}_u)\big)$. Depending on whether the server has the movie "Barbie" available, we have the following reductions:

$$\mathsf{BuySys} \rightsquigarrow_s^2 (\boldsymbol{\nu} u)(\overline{u}.\mathtt{none} \mid \mathsf{Tim}_u) \quad \text{or} \quad \mathsf{BuySys} \rightsquigarrow_s^5 (\boldsymbol{\nu} u)(\overline{u}.\mathtt{some}; \ldots \mid \mathsf{Tim}_u)$$

▽

Our type system ensures *session fidelity* and *communication safety*, but not confluence: the former says that processes correctly follow their ascribed session protocols, and the latter that no communication errors/mismatches occur. Both properties follow from the fact that typing is consistent across structural congruence and reduction. See [13] for details.

$$
\begin{array}{rrlrl}
M, N, L ::= & x & \text{variable} & \mid M\langle\!\langle C/x \rangle\!\rangle & \text{intermediate subst.} \\
& \mid (M\ C) & \text{application} & \mid M\langle\!\lfloor C/\widetilde{x} \rfloor\!\rangle & \text{explicit subst.} \\
& \mid \lambda x.M & \text{abstraction} & \mid \mathtt{fail}^{\widetilde{x}} & \text{failure} \\
& \mid M[\widetilde{x} \leftarrow x] & \text{sharing} & & \\
C, D ::= & \mathtt{1} \mid \wr M \int \cdot C & & & \text{bag} \\
\mathcal{C} ::= & [\cdot] \mid \mathcal{C}[\widetilde{x} \leftarrow x] \mid (\mathcal{C}\ C) \mid \mathcal{C}\langle\!\lfloor C/\widetilde{x} \rfloor\!\rangle & & & \text{context}
\end{array}
$$

Fig. 4. Syntax of $\lambda^{\ell}_{\mathsf{C}}$: terms, bags, and contexts.

Theorem 1 (Type Preservation). *If $P \vdash \Gamma$, then both $P \equiv Q$ and $P \rightsquigarrow_S Q$ (for any Q and S) imply $Q \vdash \Gamma$.*

Another important, if often elusive, property in session types is *deadlock-freedom*, which ensures that processes can reduce as long as they are not inactive. Our type system satisfies deadlock-freedom for processes with fully connected names, i.e., typable under the empty context. See [13] for details.

Theorem 2 (Deadlock-freedom). *If $P \vdash \emptyset$ and $P \not\equiv \mathbf{0}$, then there are Q and S such that $P \rightsquigarrow_S Q$.*

3 A Non-deterministic Resource λ-calculus

We present $\lambda^{\ell}_{\mathsf{C}}$, a resource λ-calculus with non-determinism and lazy evaluation. In $\lambda^{\ell}_{\mathsf{C}}$, non-determinism is non-confluent and *implicit*, as it arises from the fetching of terms from bags of *linear* resources. This is different from $\mathsf{s}\pi^{+}$, where the choice operator '$\mid\!\mid$' specifies non-determinism *explicitly*. A mismatch between the number of variable occurrences and the size of the bag induces *failure*.

In $\lambda^{\ell}_{\mathsf{C}}$, 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. This way, e.g., the λ-term $\lambda x.(x\ x)$ is expressed in $\lambda^{\ell}_{\mathsf{C}}$ 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.

3.1 Syntax and Reduction Semantics

Syntax. We use $x, y, z, \ldots$ for variables, and write $\widetilde{x}$ to denote a finite sequence of pairwise distinct x_i's, with length $|\widetilde{x}|$. Figure 4 gives the syntax of terms (M, N, L) and bags (C, D). The empty bag is denoted $\mathtt{1}$. We use C_i to denote the i-th term in C, and $\mathsf{size}(C)$ 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 $\wr N_1 \int \cdot \wr N_2 \int$.

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.

$$[\texttt{RS:Beta}]\quad \frac{}{(\lambda x.M)C \longrightarrow M\langle\!\langle C/x\rangle\!\rangle}$$

$$[\texttt{RS:Ex-Sub}]\quad \frac{\mathsf{size}(C) = |\widetilde{x}| \qquad M \neq \texttt{fail}^{\widetilde{y}}}{(M[\widetilde{x} \leftarrow x])\langle\!\langle C/x\rangle\!\rangle \longrightarrow M\langle\!|C/\widetilde{x}|\!\rangle}$$

$$[\texttt{RS} : \texttt{TCont}]\quad \frac{M \longrightarrow N}{\mathcal{C}[M] \longrightarrow \mathcal{C}[N]}$$

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

$$[\texttt{RS:Fail}^{\ell}]\quad \frac{\mathsf{size}(C) \neq |\widetilde{x}| \qquad \widetilde{y} = (\mathsf{fv}(M) \setminus \{\widetilde{x}\}) \cup \mathsf{fv}(C)}{(M[\widetilde{x} \leftarrow x])\langle\!\langle C/x\rangle\!\rangle \longrightarrow \texttt{fail}^{\widetilde{y}}}$$

$$[\texttt{RS:Cons}_1]\quad \frac{\widetilde{y} = \mathsf{fv}(C)}{\texttt{fail}^{\widetilde{x}}\ C \longrightarrow \texttt{fail}^{\widetilde{x} \cup \widetilde{y}}}$$

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

$$[\texttt{RS:Cons}_3]\quad \frac{\widetilde{z} = \mathsf{fv}(C)}{\texttt{fail}^{\widetilde{y} \cup \widetilde{x}}\langle\!|C/\widetilde{x}|\!\rangle \longrightarrow \texttt{fail}^{\widetilde{y} \cup \widetilde{z}}}$$

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

$$\mathsf{head}(x) = x \qquad \mathsf{head}(\lambda x.M) = \lambda x.M \qquad \mathsf{head}((M\ C)) = \mathsf{head}(M)$$

$$\mathsf{head}(\texttt{fail}^{\widetilde{x}}) = \texttt{fail}^{\widetilde{x}} \qquad \mathsf{head}(M\langle\!\langle C/x\rangle\!\rangle) = M\langle\!\langle C/x\rangle\!\rangle \qquad \mathsf{head}(M\langle\!|C/\widetilde{x}|\!\rangle) = \mathsf{head}(M)$$

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

Fig. 5. Reduction rules for $\lambda^{\ell}_{\mathsf{C}}$.

An abstraction $\lambda x.M$ binds occurrences of x in M. Application $(M\ C)$ is as usual. The term $M\langle\!|C/\widetilde{x}|\!\rangle$ is the *explicit substitution* of a bag C for $\widetilde{x}$ in M. 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. The term $M\langle\!\langle C/x\rangle\!\rangle$ denotes an intermediate explicit substitution that does not (necessarily) satisfy the conditions for explicit substitutions.

The term $\texttt{fail}^{\widetilde{x}}$ 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, defined as expected. Term M is *closed* if $\mathsf{fv}(M) = \emptyset$.

Semantics. Figure 5 gives the reduction semantics, denoted $\longrightarrow$, and the *head variable* of term M, denoted $\mathsf{head}(M)$. Rule $[\texttt{RS} : \texttt{Beta}]$ induces an intermediate substitution. Rule $[\texttt{RS} : \texttt{Ex-Sub}]$ reduces an intermediate substitution to an explicit substitution, provided the size of the bag equals the number of shared variables. In case of a mismatch, the term evolves into failure via Rule $[\texttt{RS} : \texttt{Fail}^{\ell}]$.

An explicit substitution $M\langle\!|C/\widetilde{x}|\!\rangle$, where the head variable of M is $x_j \in \widetilde{x}$, reduces via Rule $[\texttt{R} : \texttt{Fetch}^{\ell}]$. 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. Rules $[\mathtt{RS} : \mathtt{Cons}_j]$ for $j \in \{1,2,3\}$ consume terms when they meet failure. Finally, Rule $[\mathtt{RS} : \mathtt{TCont}]$ closes reduction under contexts. The following example illustrates reduction.

Example 6. Consider the term $M_0 = (\lambda x.x_1 \wr x_2 \wr x_3\ \mathtt{1} \int \int [\widetilde{x} \leftarrow x])\ \wr \mathtt{fail}^{\emptyset}, y, I \int$, where $I = \lambda x.(x_1[x_1 \leftarrow x])$ and $\widetilde{x} = x_1, x_2, x_3$. 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):

$$M_0 \longrightarrow (x_1 \wr x_2 \wr x_3\ \mathtt{1} \int \int [\widetilde{x} \leftarrow x]) \langle\!\langle \wr \mathtt{fail}^{\emptyset}, y, I \int / x \rangle\!\rangle \tag{2}$$

$$\longrightarrow (x_1 \wr x_2 \wr x_3\ \mathtt{1} \int \int) \langle\!\langle \wr \mathtt{fail}^{\emptyset}, y, I \int / \widetilde{x} \rangle\!\rangle = M \tag{3}$$

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:

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

▽

3.2 Resource Control for $\lambda^{\ell}_{\mathsf{c}}$ via Intersection Types

Our type system for $\lambda^{\ell}_{\mathsf{c}}$ is based on non-idempotent intersection types. As in prior works [4,21], 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 $\mathtt{fail}^{\tilde{x}}$ (see the text after Theorem 3).

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

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

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 $k = 0$).

Judgments have the form $\Gamma \vDash M : \tau$, with contexts defined as follows:

$$\Gamma, \Delta ::= \text{-} \mid \Gamma, x : \pi \mid \Gamma, x : \sigma$$

where - denotes the empty context. 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 $\Gamma^{\downarrow} = \{x : \pi \in \Gamma \mid \pi \neq \omega\}$.

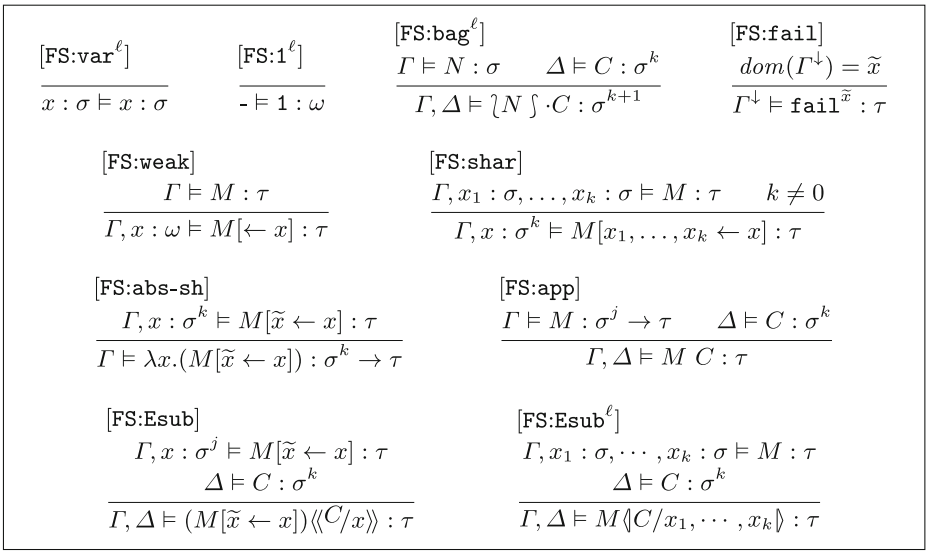

Fig. 6. Well-Formedness Rules for $\lambda^{\ell}_{\mathtt{c}}$.

Definition 4 (Well-formedness in $\lambda^{\ell}_{\mathtt{c}}$). *A term M is* well-formed *if there exists a context Γ and a type τ such that the rules in Fig. 6 entail $\Gamma \vDash M : \tau$.*

In Fig. 6, Rule [FS : var$^\ell$] types variables. Rule [FS : 1$^\ell$] types the empty bag with ω. Rule [FS : bag$^\ell$] types the concatenation of bags. Rule [FS : 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 [FS : weak] types $M[\leftarrow x]$ by weakening the context with $x : \omega$. Rule [FS : shar] types $M[\widetilde{x} \leftarrow x]$ with τ, provided that there are assignments to the shared variables in $\widetilde{x}$.

Rule [FS : abs-sh] types an abstraction $\lambda x.(M[\widetilde{x} \leftarrow x])$ with $\sigma^k \rightarrow \tau$, provided that $M[\widetilde{x} \leftarrow x] : \tau$ can be entailed from an assignment $x : \sigma^k$. Rule [FS : app] types $(M\ C)$, provided that M has type $\sigma^j \rightarrow \tau$ and C has type σ^k. Note that, unlike usual intersection type systems, j and k may differ. Rule [FS : 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 [FS : 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), whereas *well-typed* terms, defined below, satisfy also subject expansion (SE). See [13] for details.

Theorem 3 (SR in $\lambda^{\ell}_{\mathtt{c}}$). *If $\Gamma \vDash M : \tau$ and $M \longrightarrow M'$, then $\Gamma \vDash M' : \tau$.*

From our system for well-formedness we can extract a system for *well-typed* terms, which do not include $\mathtt{fail}^{\widetilde{x}}$. Judgments for well-typedness are denoted $\Gamma \vdash M : \tau$, with rules copied from Fig. 6 (the rule name prefix FS is replaced with TS), with the following modifications: (i) Rule [TS:fail] is removed; (ii) Rules [TS:app] and [TS:Esub] are modified to disallow a mismatch between

$$
\begin{aligned}
[\![x]\!]_u &= \overline{x}.\texttt{some}; [x \leftrightarrow u] \\
[\![\lambda x.M]\!]_u &= \overline{u}.\texttt{some}; u(x); [\![M]\!]_u \\
[\![(M\ C)]\!]_u &= (\boldsymbol{\nu} v)([\![M]\!]_v \mid v.\texttt{some}_{u,\mathsf{lfv}(C)}; \overline{v}[x]; ([\![C]\!]_x \mid [v \leftrightarrow u])) \\
[\![M\langle\!\langle C/x \rangle\!\rangle]\!]_u &= (\boldsymbol{\nu} x)([\![M]\!]_u \mid [\![C]\!]_x) \\
[\![\wr N \int \cdot\ C]\!]_x &= x.\texttt{some}_{\mathsf{fv}(C),\mathsf{fv}(N)}; x(y_i); x.\texttt{some}_{y_i,\mathsf{fv}(C),\mathsf{fv}(N)}; \overline{x}.\texttt{some}; \overline{x}[z_i]; \\
&\qquad (z_i.\texttt{some}_{\mathsf{fv}(N)}; [\![N]\!]_{z_i} \mid [\![C]\!]_x \mid \overline{y_i}.\texttt{none}) \\
[\![\mathtt{1}]\!]_x &= x.\texttt{some}_{\emptyset}; x(y_n); (\overline{y_n}.\texttt{some}; \overline{y_n}[] \mid x.\texttt{some}_{\emptyset}; \overline{x}.\texttt{none}) \\
[\![M\langle\!\langle \wr N_1, N_2 \int / x_1, x_2 \rangle\!\rangle]\!]_u &= (\boldsymbol{\nu} z_1)(z_1.\texttt{some}_{\mathsf{fv}(N_1)}; [\![N_1]\!]_{z_1} \mid (\boldsymbol{\nu} z_2)(z_2.\texttt{some}_{\mathsf{fv}(N_2)}; [\![N_2]\!]_{z_2} \\
&\qquad \mid \mathbin{\Vert}_{x_i \in \{x_1,x_2\}} \mathbin{\Vert}_{x_j \in \{x_1,x_2 \setminus x_i\}} [\![M]\!]_u\{z_1/x_i\}\{z_2/x_j\})) \\
[\![M[\leftarrow x]]\!]_u &= \overline{x}.\texttt{some}; \overline{x}[y_i]; (y_i.\texttt{some}_{u,\mathsf{fv}(M)}; y_i(); [\![M]\!]_u \mid \overline{x}.\texttt{none}) \\
[\![M[\widetilde{x} \leftarrow x]]\!]_u &= \overline{x}.\texttt{some}; \overline{x}[y_i]; \big(y_i.\texttt{some}_{\emptyset}; y_i(); \mathbf{0} \mid \overline{x}.\texttt{some}; x.\texttt{some}_{u,\mathsf{fv}(M)\setminus\widetilde{x}}; \\
&\qquad \mathbin{\Vert}_{x_i \in \widetilde{x}} x(x_i); [\![M[(\widetilde{x} \setminus x_i) \leftarrow x]]\!]_u\big) \\
[\![\texttt{fail}^{x_1,\ldots,x_k}]\!]_u &= \overline{u}.\texttt{none} \mid \overline{x_1}.\texttt{none} \mid \ldots \mid \overline{x_k}.\texttt{none}
\end{aligned}
$$

Fig. 7. Translation of $\lambda^{\ell}_{\mathsf{C}}$ into $\mathsf{s}\pi^{+}$.

variables and resources, i.e., multiset types should match in size. Well-typed terms are also well-formed, and thus satisfy SR. Moreover, as a consequence of adopting (non-idempotent) intersection types, they also satisfy SE:

Theorem 4 (SE in $\lambda^{\ell}_{\mathsf{C}}$). *If $\Gamma \vdash M' : \tau$ and $M \longrightarrow M'$, then $\Gamma \vdash M : \tau$.*

4 A Typed Translation of $\boldsymbol{\lambda^{\ell}_{\mathsf{C}}}$ into $\mathbf{s}\boldsymbol{\pi}^{+}$

While $\mathsf{s}\pi^{+}$ features non-deterministic choice, $\lambda^{\ell}_{\mathsf{C}}$ is a prototypical programming language in which implicit non-determinism implements fetching of resources. Resources are controlled using different type systems (session types in $\mathsf{s}\pi^{+}$, intersection types in $\lambda^{\ell}_{\mathsf{C}}$). To reconcile these differences and illustrate the potential of $\mathsf{s}\pi^{+}$ to precisely model non-determinism as found in realistic programs/protocols, we give a translation of $\lambda^{\ell}_{\mathsf{C}}$ into $\mathsf{s}\pi^{+}$. This translation preserves types (Theorem 5) and respects well-known criteria for dynamic correctness [11,23,24] (Theorem 6).

The Translation. Given a $\lambda^{\ell}_{\mathsf{C}}$-term M, its translation into $\mathsf{s}\pi^{+}$ is denoted $[\![M]\!]_u$ and given in Fig. 7. As usual, every variable x in M becomes a name x in process $[\![M]\!]_u$, where name u provides the behavior of M. A peculiarity is that, to handle failures in $\lambda^{\ell}_{\mathsf{C}}$, u is a non-deterministically available session: the translated term can be available or not, as signaled by prefixes $\overline{u}.\texttt{some}$ and $\overline{u}.\texttt{none}$, respectively. As a result, reductions from $[\![M]\!]_u$ include synchronizations that codify M's behavior but also synchronizations that confirm a session's availability.

At its core, our translation follows Milner's. This way, e.g., the process $[\![(\lambda x.M)\ C]\!]_u$ enables synchronizations between $[\![\lambda x.M]\!]_v$ and $[\![C]\!]_x$ along name v,

$$[\![\mathbf{unit}]\!] = \&\mathbf{1} \qquad [\![\sigma^k \to \tau]\!] = \&(\overline{[\![\sigma^k]\!]_{(\sigma,i)}} \parr [\![\tau]\!])$$

$$[\![\sigma \wedge \pi]\!]_{(\tau,i)} = \oplus((\&\mathbf{1}) \parr (\oplus\&((\oplus[\![\sigma]\!]) \otimes ([\![\pi]\!]_{(\tau,i)}))))$$

$$[\![\omega]\!]_{(\sigma,i)} = \begin{cases} \oplus((\&\mathbf{1}) \parr (\oplus\&\mathbf{1})) & \text{if } i = 0 \\ \oplus((\&\mathbf{1}) \parr (\oplus\,\&((\oplus[\![\sigma]\!]) \otimes ([\![\omega]\!]_{(\sigma,i-1)})))) & \text{if } i > 0 \end{cases}$$

Fig. 8. Translation of intersection types into session types (cf. Definition 5).

resulting in the translation of an intermediate substitution. The *key novelty* is the role and treatment of non-determinism. Accommodating non-confluent non-determinism is non-trivial, as it entails translating explicit substitutions and sharing in $\lambda^{\ell}_{\mathsf{C}}$ using the non-deterministic choice operator $\shortparallel$ in $\mathsf{s}\pi^{+}$. Next we discuss these novel aspects, while highlighting differences with respect to a translation by Paulus et al. [22], which is given in the confluent setting (see Section 5).

In Fig. 7, 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 $\shortparallel$ 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 $\shortparallel$ 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 $[\![M\langle\!|C/\widetilde{x}|\!\rangle]\!]_u$ uses the processes $[\![N_i]\!]_{z_i}$, 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 $\shortparallel_{x_i \in \{x_1,x_2\}} \shortparallel_{x_j \in \{x_1,x_2\} \setminus x_i}$ is equivalent to non-deterministically assigning x_i, x_j to each permutation of x_1, x_2. The resulting choice involves $[\![M]\!]_u$ 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 $z_i.\mathtt{some}_{\mathsf{fv}(N_i)}$; $[\![N_i]\!]_{z_i}$ then represents fetching N_i from the bag. The size of the translated term $[\![M\langle\!|C/\widetilde{x}|\!\rangle]\!]_u$ is exponential with respect to the size of C.

The process $[\![M[\widetilde{x} \leftarrow x]]\!]_u$ proceeds as follows. First, it 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 $[\![M[(\widetilde{x} \backslash x_i) \leftarrow x]]\!]_u$. The possible choices are permuted, represented by $\shortparallel_{x_i \in \widetilde{x}}$. Synchronizations with $[\![M[(\widetilde{x} \backslash x_i) \leftarrow x]]\!]_u$ and bags delay commitment in this choice (we return to this point below). The process $[\![M[\leftarrow x]]\!]_u$ is similar but simpler: here the name x fails, as it cannot take further elements to substitute.

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 $\lambda^{\ell}_{\mathsf{C}}$ is correctly translated to failure in $\mathsf{s}\pi^{+}$.

Translation Correctness. The translation is typed: intersection types in $\lambda^{\ell}_{\mathsf{C}}$ are translated into session types in $\mathsf{s}\pi^{+}$ (Fig. 8). This translation of types abstractly describes how non-deterministic fetches are codified as non-deterministic session protocols. It is worth noting that this translation of types is the same as in [22]. This is not surprising: as we have seen, session types effectively abstract away from the behavior of processes, as all branches of a non-deterministic choice use the same typing context. Still, it is pleasant that the translation of types remains unchanged across different translations with our (non-confluent) non-determinism (in Fig. 7) and with confluent non-determinism (in [22]).

To state *static* correctness, we require the following definition:

Definition 5. *Let $\Gamma = x_1 : \sigma_1, ..., x_m : \sigma_m, v_1 : \pi_1, ..., v_n : \pi_n$ be a context. The translation $[\![\cdot]\!]_{\_}$ in Fig. 8 extends to contexts as follows:*

$$[\![\Gamma]\!] = x_1 : \&\overline{[\![\sigma_1]\!]}, \cdots, x_m : \&\overline{[\![\sigma_m]\!]}, v_1 : \overline{[\![\pi_1]\!]_{(\sigma,i_1)}}, \cdots, v_n : \overline{[\![\pi_n]\!]_{(\sigma,i_n)}}$$

Well-formed terms translate into well-typed processes:

Theorem 5. *If $\Gamma \vDash M : \tau$, then $[\![M]\!]_u \vdash [\![\Gamma]\!], u : [\![\tau]\!]$.*

To state *dynamic* correctness, we rely on established notions that (abstractly) characterize *correct translations*. A language $\mathcal{L} = (L, \rightarrow)$ consists of a set of terms L and a reduction relation $\rightarrow$ 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 $\rightarrow$) from T to a term satisfying success criteria.

Given $\mathcal{L}_1 = (L_1, \rightarrow_1)$ and $\mathcal{L}_2 = (L_2, \rightarrow_2)$, we seek translations $[\![\cdot]\!] : L_1 \rightarrow L_2$ that are correct: they satisfy well-known correctness criteria [11,23,24]. We state the set of correctness criteria that determine the correctness of a translation.

Definition 6 (Correct Translation). *Let $\mathcal{L}_1 = (\mathcal{M}, \rightarrow_1)$ and $\mathcal{L}_2 = (\mathcal{P}, \rightarrow_2)$ 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 $[\![\cdot]\!] : \mathcal{M} \rightarrow \mathcal{P}$, we define:*

Completeness: *For every M, M' such that $M \rightarrow_1^* M'$, there exists P such that $[\![M]\!] \rightarrow_2^* P \asymp_2 [\![M']\!]$.*
Weak Soundness: *For every M and P such that $[\![M]\!] \rightarrow_2^* P$, there exist M', P' such that $M \rightarrow_1^* M'$ and $P \rightarrow_2^* P' \asymp_2 [\![M']\!]$.*
Success Sensitivity: *For every M, we have $M \Downarrow \checkmark$ if and only if $[\![M]\!] \Downarrow \checkmark$.*

Let us write Λ to denote the set of well-formed $\lambda^{\ell}_{\mathsf{C}}$ terms, and Π for the set of all well-typed $\mathsf{s}\pi^{+}$ processes, both including $\checkmark$. We have our final result:

Theorem 6 (Translation correctness under $\rightsquigarrow$). *The translation $[\![\cdot]\!]_{\_} : (\Lambda, \longrightarrow) \rightarrow (\Pi, \rightsquigarrow)$ is correct (cf. Definition 6) using equivalence $\equiv$ (Fig. 1).*

The proof of Theorem 6 involves instantiating/proving each of the parts of Definition 6. Among these, *weak soundness* is the most challenging to prove. Prior work on translations of typed λ into π with confluent non-determinism [22] rely

critically on confluence to match a behavior in π with a corresponding behavior in λ. Because in our setting confluence is lost, we must resort to a different proof.

As already discussed, our translation makes the implicit non-determinism in a $\lambda^{\ell}_{\mathsf{C}}$-term M explicit by adding non-deterministic choices in key points of $[\![M]\!]_u$. Our reduction $\rightsquigarrow$ preserves those branches that simultaneously have the same prefix available (up to $\bowtie$). In proving weak soundness, we exploit the fact that reduction entails delayed commitment. To see this, consider the following terms:

$$(\boldsymbol{\nu} x)((\alpha_1; P_1 \parallel \alpha_2; P_2) \mid Q) \tag{4}$$

$$(\boldsymbol{\nu} x)(\alpha_1; P_1 \mid Q) \parallel (\boldsymbol{\nu} x)(\alpha_2; P_2 \mid Q) \tag{5}$$

In (4), commitment to a choice relies on whether $\alpha_1 \bowtie \alpha_2$ holds (cf. Definition 3). If $\alpha_1 \not\bowtie \alpha_2$, a choice is made; otherwise, commitment is delayed, and depends on P_1 and P_2. Hence, in (4) the possibility of committing to either branch is kept open. In contrast, in (5) commitment to a choice is independent of $\alpha_1 \bowtie \alpha_2$.

Our translation exploits the delayed commitment of non-determinism illustrated by (4) to mimic commitment to non-deterministic choices in $\lambda^{\ell}_{\mathsf{C}}$, which manifests in fetching resources from bags. The fact that this delayed commitment preserves information about the different branches (e.g., P_1 and P_2 in (4)) is essential to establish weak soundness, i.e., to match a behavior in $\mathsf{s}\pi^{+}$ with a corresponding step in $\lambda^{\ell}_{\mathsf{C}}$. In contrast, forms of non-determinism in $[\![N]\!]_u$ that resemble (5) are useful to characterize behaviors different from fetching.

5 Summary and Related Work

We studied the interplay between resource control and non-determinism in typed calculi. We introduced $\mathsf{s}\pi^{+}$ and $\lambda^{\ell}_{\mathsf{C}}$, two calculi with non-confluent non-determinism, both with type systems for resource control. Inspired by the untyped π-calculus, non-determinism in $\mathsf{s}\pi^{+}$ is lazy and explicit, with session types defined following 'propositions-as-sessions' [5]. In $\lambda^{\ell}_{\mathsf{C}}$, non-determinism arises in the fetching of resources, and is regulated by intersection types. A correct translation of $\lambda^{\ell}_{\mathsf{C}}$ into $\mathsf{s}\pi^{+}$ precisely connects their different forms of non-determinism.

Related Work. Integrating (non-confluent) non-determinism within session types is non-trivial, as carelessly discarding branches would break typability. Work by Caires and Pérez [5], already mentioned, develops a confluent semantics by requiring that non-determinism is only used inside the monad $\&A$; our non-confluent semantics drops this requirement. This allows us to consider non-deterministic choices not possible in [5], such as, e.g., selections of different labels. We stress that linearity is not jeopardized: the branches of '$\parallel$' do not represent *different sessions*, but *different implementations* of the same sessions.

Atkey et al. [1] and Kokke et al. [16] extend 'propositions-as-sessions' with non-determinism. Their approaches are very different (conflation of the additives and bounded linear logic, respectively) and support non-determinism for unrestricted names only. Also, [1,16] do not connect with typed λ-calculi, as we do.

Rocha and Caires also consider non-determinism, relying on confluence in [25] and on unrestricted names in [26]. Casal et al. [7,30] develop a type system for *mixed sessions* (sessions with mixed choices), which can express non-determinism but does not ensure deadlock-freedom. Ensuring deadlock-freedom by typing is a key feature of the 'propositions-as-sessions' approach that we adopt for $\mathsf{s}\pi^{+}$.

Our language $\lambda^{\ell}_{\mathsf{C}}$ is most related to calculi by Boudol [3], Boudol and Laneve [4], and by Pagani and Ronchi Della Rocca [21]. Non-determinism in the calculi in [3,4] is committing and implicit; their linear resources can be consumed *at most* once, rather than *exactly* once. The work [21] considers non-committing non-determinism that is both implicit (as in $\lambda^{\ell}_{\mathsf{C}}$) and explicit (via a sum operator on terms). Both [3,21] develop (non-idempotent) intersection type systems to regulate resources. In our type system, all terms in a bag have the same type; the system in [21] does not enforce this condition. Unlike these type systems, our system for well-formedness can type terms with a lack or an excess of resources.

Boudol and Laneve [4] and Paulus et al. [22] give translations of resource λ-calculi into π. The translation in [4] is used to study the semantics induced upon λ-terms by a translation into π; unlike ours, it does not consider types. As already mentioned in Sect. 4, Paulus et al. [22] relate calculi with *confluent* non-determinism: a resource λ-calculus with sums on terms, and the session π-calculus from [5]. Our translation of terms and that in [22] are very different: while here we use non-deterministic choice to mimic the sharing construct, the translation in [22] uses it to translate bags. Hence, our Theorem 6 cannot be derived from [22].

The last decade of work on 'propositions-as-sessions' has delivered insightful connections with typed λ-calculi—see, e.g., [28,29,31]. Excepting [22], already discussed, none of these works consider non-deterministic λ-calculi.

Acknowledgments. We are grateful to the anonymous reviewers for useful comments on previous versions of this paper. We are also grateful to Mariangiola Dezani for her encouragement and suggestions. 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. Atkey, R., Lindley, S., Morris, J.G.: Conflation confers concurrency. In: Lindley, S., McBride, C., Trinder, P., Sannella, D. (eds.) A List of Successes That Can Change the World. LNCS, vol. 9600, pp. 32–55. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-30936-1_2
2. Berger, M., Honda, K.: The two-phase commitment protocol in an extended pi-calculus. In: Aceto, L., Victor, B. (eds.) 7th International Workshop on Expressiveness in Concurrency, EXPRESS 2000, Satellite Workshop of CONCUR 2000, State College, PA, USA, 21 August 2000. Electronic Notes in Theoretical Computer Science, vol. 39, pp. 21–46. Elsevier (2000). https://doi.org/10.1016/S1571-0661(05)82502-2

3. Boudol, G.: The lambda-calculus with multiplicities. In: Best, E. (ed.) CONCUR 1993. LNCS, vol. 715, pp. 1–6. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-57208-2_1
4. Boudol, G., Laneve, C.: Lambda-calculus, multiplicities, and the pi-calculus. In: Proof, Language, and Interaction, Essays in Honour of Robin Milner, pp. 659–690 (2000)
5. Caires, L., Pérez, J.A.: Linearity, control effects, and behavioral types. In: Yang, H. (ed.) ESOP 2017. LNCS, vol. 10201, pp. 229–259. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54434-1_9
6. Caires, L., Pfenning, F.: Session types as intuitionistic linear propositions. In: Gastin, P., Laroussinie, F. (eds.) CONCUR 2010. LNCS, vol. 6269, pp. 222–236. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15375-4_16
7. Casal, F., Mordido, A., Vasconcelos, V.T.: Mixed sessions. Theor. Comput. Sci. **897**, 23–48 (2022). https://doi.org/10.1016/j.tcs.2021.08.005
8. de'Liguoro, U., Piperno, A.: Non deterministic extensions of untyped lambda-calculus. Inf. Comput. **122**(2), 149–177 (1995). https://doi.org/10.1006/inco.1995.1145
9. Dezani-Ciancaglini, M.: Logical semantics for concurrent lambda-calculus. Ph.D. thesis, Nijmegen University (1996). https://www.di.unito.it/~dezani/papers/tesi.ps
10. Ehrhard, T., Regnier, L.: The differential lambda-calculus. Theor. Comput. Sci. **309**(1–3), 1–41 (2003). https://doi.org/10.1016/S0304-3975(03)00392-X
11. Gorla, D.: Towards a unified approach to encodability and separation results for process calculi. Inf. Comput. **208**(9), 1031–1053 (2010). https://doi.org/10.1016/j.ic.2010.05.002
12. Groote, J.F., Mousavi, M.R.: Modeling and Analysis of Communicating Systems. MIT Press (2014). https://mitpress.mit.edu/books/modeling-and-analysis-communicating-systems
13. van den Heuvel, B., Paulus, J.W.N., Nantes-Sobrinho, D., Pérez, J.A.: Typed non-determinism in functional and concurrent calculi (extended version). CoRR abs/2205.00680 (2022). https://doi.org/10.48550/arXiv.2205.00680
14. Honda, K.: Types for dyadic interaction. In: Best, E. (ed.) CONCUR 1993. LNCS, vol. 715, pp. 509–523. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-57208-2_35
15. Honda, K., Vasconcelos, V.T., Kubo, M.: Language primitives and type discipline for structured communication-based programming. In: Hankin, C. (ed.) ESOP 1998. LNCS, vol. 1381, pp. 122–138. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0053567
16. Kokke, W., Morris, J.G., Wadler, P.: Towards races in linear logic. Log. Meth. Comput. Sci. **16**(4) (2020). https://doi.org/10.23638/LMCS-16(4:15)2020
17. Milner, R.: Functions as processes. Research Report 1154, INRIA, Sophia Antipolis (1990). Final version appeared as [18]
18. Milner, R.: Functions as processes. Math. Struct. Comput. Sci. **2**(2), 119–141 (1992). https://doi.org/10.1017/S0960129500001407
19. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes. I. Inf. Comput. **100**(1), 1–40 (1992). https://doi.org/10.1016/0890-5401(92)90008-4
20. Nestmann, U., Fuzzati, R., Merro, M.: Modeling consensus in a process calculus. In: Amadio, R., Lugiez, D. (eds.) CONCUR 2003. LNCS, vol. 2761, pp. 399–414. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45187-7_26

21. Pagani, M., della Rocca, S.R.: Solvability in resource lambda-calculus. In: Ong, L. (ed.) FoSSaCS 2010. LNCS, vol. 6014, pp. 358–373. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12032-9_25
22. Paulus, J.W.N., Nantes-Sobrinho, D., Pérez, J.A.: Non-deterministic functions as non-deterministic processes. In: Kobayashi, N. (ed.) 6th International Conference on Formal Structures for Computation and Deduction, FSCD 2021, Buenos Aires, Argentina (Virtual Conference), 17–24 July 2021. LIPIcs, vol. 195, pp. 21:1–21:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021). https://doi.org/10.4230/LIPIcs.FSCD.2021.21, Extended version on https://arxiv.org/abs/2104.14759
23. Peters, K.: Translational expressiveness. Comparing process calculi using encodings. Ph.D. thesis, Berlin Institute of Technology (2012). https://doi.org/10.14279/depositonce-3416
24. Peters, K.: Comparing process calculi using encodings. In: Pérez, J.A., Rot, J. (eds.) 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. EPTCS, vol. 300, pp. 19–38 (2019). https://doi.org/10.4204/EPTCS.300.2
25. Rocha, P., Caires, L.: Propositions-as-types and shared state. Proc. ACM Program. Lang. **5**(ICFP), 79:1–79:30 (2021). https://doi.org/10.1145/3473584
26. Rocha, P., Caires, L.: Safe session-based concurrency with shared linear state. In: Wies, T. (ed.) Programming Languages and Systems. LNCS, vol. 2072, pp. 421–450. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-30044-8_16
27. Sangiorgi, D., Walker, D.: The Pi-Calculus - A Theory of Mobile Processes. Cambridge University Press, Cambridge (2001)
28. Toninho, B., Caires, L., Pfenning, F.: Functions as session-typed processes. In: Birkedal, L. (ed.) FoSSaCS 2012. LNCS, vol. 7213, pp. 346–360. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28729-9_23
29. Toninho, B., Yoshida, N.: On polymorphic sessions and functions. In: Ahmed, A. (ed.) ESOP 2018. LNCS, vol. 10801, pp. 827–855. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-89884-1_29
30. Vasconcelos, V.T., Casal, F., Almeida, B., Mordido, A.: Mixed sessions. In: ESOP 2020. LNCS, vol. 12075, pp. 715–742. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-44914-8_26
31. Wadler, P.: Propositions as sessions. In: Thiemann, P., Findler, R.B. (eds.) ACM SIGPLAN International Conference on Functional Programming, ICFP 2012, Copenhagen, Denmark, 9–15 September 2012, pp. 273–286. ACM (2012). https://doi.org/10.1145/2364527.2364568