




Types and Terms Translated: Unrestricted Resources in Encoding Functions as Processes

Joseph W. N. Paulus   
University of Groningen, The Netherlands

Daniele Nantes-Sobrinho   
University of Brasília, Brazil
Imperial College London, UK

Jorge A. Pérez   
University of Groningen, The Netherlands

Abstract

Type-preserving translations are effective rigorous tools in the study of core programming calculi. In this paper, we develop a new typed translation that connects sequential and concurrent calculi; it is governed by type systems that control *resource consumption*. Our main contribution is the source language, a new resource λ -calculus with non-collapsing non-determinism and failures, dubbed $u\lambda_{\oplus}^{\xi}$. In $u\lambda_{\oplus}^{\xi}$, resources are split into linear and unrestricted; failures are explicit and arise from this distinction. We define a type system based on intersection types to control resources and fail-prone computation. The target language is $s\pi$, an existing session-typed π -calculus that results from a Curry-Howard correspondence between linear logic and session types. Our typed translation subsumes our prior work; interestingly, it treats unrestricted resources in $u\lambda_{\oplus}^{\xi}$ as client-server session behaviours in $s\pi$.

2012 ACM Subject Classification Theory of computation \rightarrow Type structures; Theory of computation \rightarrow Process calculi

Keywords and phrases Resource λ -calculus, intersection types, session types, process calculi

Digital Object Identifier 10.4230/LIPIcs.TYPES.2021.11

Related Version Online appendix with omitted proofs and further examples:

Full Version: <http://arxiv.org/abs/2112.01593> [18]

Funding Research partially supported by the Dutch Research Council (NWO) under project No. 016.Vidi.189.046 (Unifying Correctness for Communicating Software).

Acknowledgements We are grateful to the anonymous reviewers for their constructive feedback.

1 Introduction

Context. *Type-preserving translations* are effective rigorous tools in the study of core programming calculi. They can be seen as an abstract counterpart to the type-preserving compilers that enable key optimisations in the implementation of programming languages. The goal of this paper is to develop a new typed translation that connects sequential and concurrent calculi, and is governed by type systems that control *resource consumption*.

A central idea in the resource λ -calculus is to consider that in an application $M N$ the argument N is a *resource* of possibly limited availability. This generalisation of the λ -calculus triggers many fascinating questions, such as typability, solvability, expressiveness power, etc., which have been studied in different settings (see, e.g., [1, 3, 16, 7]). In established resource λ -calculi, such as those by Boudol [1] and by Pagani and Ronchi della Rocca [16], a more general form of application is considered: a term can be applied to a bag of resources $B = \{N_1\} \cdot \dots \cdot \{N_k\}$, where N_1, \dots, N_k denote terms; then, an application $M B$ must take into account that each N_i may be reusable or not. Thus, non-determinism is natural in



© Joseph W. N. Paulus, Daniele Nantes-Sobrinho, and Jorge A. Pérez;
licensed under Creative Commons License CC-BY 4.0

27th International Conference on Types for Proofs and Programs (TYPES 2021).

Editors: Henning Basold, Jesper Cockx, and Silvia Ghilezan; Article No. 11; pp. 11:1–11:24

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

resource λ -calculi, because a term has now multiple ways of consuming resources from the bag. This bears a strong resemblance with process calculi such as the π -calculus [15], in which concurrent interactions are intrinsically non-deterministic.

There are different flavors of non-determinism. Over two decades ago, Boudol and Laneve [2, 3] explored connections between a resource λ -calculus and the π -calculus. In their setting, an application $M B$ would branch, i.e., M could consume a resource N_j in B (with $j \in \{1, \dots, k\}$) and discard the other $k - 1$ resources in a non-confluent manner; this is what we call a *collapsing* approach to non-determinism. On a different direction, Pagani and Ronchi della Rocca [16] proposed λ^r , a resource λ -calculus that implements *non-collapsing* non-determinism, whereby all the possible alternatives for resource consumption are retained together in a sum, ensuring confluence. They investigated typability and characterisations of solvability in λ^r , but no connection with the π -calculus was established. In an attempt to address this gap, our previous work [17] identified λ_{\oplus}^{ζ} , a resource λ -calculus with non-collapsing non-determinism, explicit failure, and *linear* resources (to be used exactly once), and developed a correct typed translation into a session typed π -calculus [5]. The calculus λ_{\oplus}^{ζ} , however, does not include *unrestricted* resources (to be used zero or many times).

This Paper. Here we introduce a new λ -calculus, dubbed $u\lambda_{\oplus}^{\zeta}$, its intersection type system, and its translation into session-typed processes. Our motivation is twofold: to elucidate the status of unrestricted resources in a functional setting with non-collapsing non-determinism, and to characterise unrestricted resources within a translation of functions into processes. Unlike its predecessors, $u\lambda_{\oplus}^{\zeta}$ distinguishes between linear and unrestricted resources. This distinction determines the semantics of terms and especially the deadlocks (*failures*) that arise due to mismatches in resources. This way, $u\lambda_{\oplus}^{\zeta}$ subsumes λ_{\oplus}^{ζ} , which is purely linear and cannot express failures related to unrestricted resources.

Distinguishing linear and unrestricted resources is not a new insight. This idea goes back to Boudol’s λ -calculus with multiplicities [1], where arguments can be tagged as unrestricted. What is new about $u\lambda_{\oplus}^{\zeta}$ is that the distinction between linear and unrestricted resources leads to two main differences. First, occurrences of a variable can be linear or unrestricted, depending on the kind of resources they should be substituted with. This way, e.g., a linear occurrence of variable must be substituted with a linear resource. In $u\lambda_{\oplus}^{\zeta}$, a variable can have linear and unrestricted occurrences in the same term. (Notice that we use the adjective ‘linear’ in connection to resources used exactly once, and not to the number of occurrences of a variable in a term.) Second, failures depend on the nature of the involved resource(s). In $u\lambda_{\oplus}^{\zeta}$, a linear failure arises from a mismatch between required and available (linear) resources; an unrestricted failure arises when a specific (unrestricted) resource is not available.

Accordingly, the syntax of $u\lambda_{\oplus}^{\zeta}$ incorporates linear and unrestricted resources, enabling their consistent separation, within non-collapsing non-determinism. The calculus allows for linear and unrestricted occurrences of variables, as just discussed; bags comprise two separate zones, linear and unrestricted; and the *failure term* $\mathbf{fail}^{x_1, \dots, x_n}$ explicitly mentions the linear variables x_1, \dots, x_n . The (lazy) reduction semantics of $u\lambda_{\oplus}^{\zeta}$ includes two different rules for “fetching” terms from bags, and for consistently handling the failure term.

We equip $u\lambda_{\oplus}^{\zeta}$ with non-idempotent intersection types, extending the approach in [17]: in $u\lambda_{\oplus}^{\zeta}$, intersection types account for more than resource multiplicity, since the elements of the unrestricted bag can have different types. Using intersection types, we define a class of *well-formed* $u\lambda_{\oplus}^{\zeta}$ expressions, which includes terms that correctly consume resources but also terms that may reduce to the failure term. Well-formed expressions thus subsume the *well-typed* expressions that can be defined in a sub-language of $u\lambda_{\oplus}^{\zeta}$ without the failure term.

The calculus $u\lambda_{\oplus}^{\zeta}$ can express terms whose dynamic behaviour is not captured by prior works. This way, e.g., the identity function \mathbf{I} admits two formulations, depending on whether the variable occurrence is linear or unrestricted. One can have $\lambda x.x$, as usual, but also the unrestricted variant $\lambda x.x[i]$, where “[i]” is an index annotation (similar to a qualifier or a tag), which indicates that x should be replaced by the i -th element of the unrestricted zone of the bag. The behaviour of these functions will depend on the bags that are provided as their arguments. Similarly, we can express variants of $\Delta = \lambda x.xx$ and $\Omega = \Delta \Delta$ whose behaviours again depend on linear or unrestricted occurrences of variables and bags. Consider the term $\Delta_7 = \lambda x.(x[1](1 \star \wr x[1] \wr^! \diamond \wr x[2] \wr^!))$, where we use “ \star ” to separate linear and unrestricted resources in the bag, and “ \diamond ” denotes concatenation of unrestricted resources. Term Δ_7 is an abstraction on x of an application of an unrestricted occurrence of x , which aims to consume the first component of an unrestricted bag, to a bag with an empty linear zone (denoted 1) and an unrestricted zone with resources $\wr x[1] \wr^!$ and $\wr x[2] \wr^!$. The self-application $\Delta_7 \Delta_7$ produces a non-terminating behaviour and yet Δ_7 itself is well-formed (see Example 20).

Both $u\lambda_{\oplus}^{\zeta}$ and λ_{\oplus}^{ζ} are *logically motivated* resource λ -calculi, in the following sense: their design has been strongly influenced by $s\pi$, a typed π -calculus resulting from the Curry-Howard correspondence between linear logic and session types in [5], where proofs correspond to processes and cut elimination to process communication. As demonstrated in [5], providing primitive support for explicit failures is key to expressing many useful programming idioms (such as exceptions); this insight is a leading motivation in our design for $u\lambda_{\oplus}^{\zeta}$.

To attest to the logical underpinnings of $u\lambda_{\oplus}^{\zeta}$, we develop a typed translation (or *encoding*) of $u\lambda_{\oplus}^{\zeta}$ into $s\pi$ and establish its correctness with respect to well-established criteria [9, 14]. As in [17], we encode λ_{\oplus}^{ζ} into $s\pi$ by relying on an intermediate language with *sharing* constructs [10, 8, 13]. A key idea in encoding $u\lambda_{\oplus}^{\zeta}$ is to codify the behaviour of unrestricted occurrences of a variable and their corresponding resources in the bag as *client-server connections*, leveraging the copying semantics for the exponential “ $!$ ” induced by the Curry-Howard correspondence. This typed encoding into $s\pi$ justifies the semantics of $u\lambda_{\oplus}^{\zeta}$ in terms of precise session protocols (i.e., linear logic propositions, because of the correspondence).

In summary, the **main contributions** of this paper are: (1) The resource calculus $u\lambda_{\oplus}^{\zeta}$ of linear and unrestricted resources, and its associated intersection type system. (2) A typed encoding of $u\lambda_{\oplus}^{\zeta}$ into $s\pi$, which connects well-formed expressions (disciplined by intersection types) and well-typed concurrent processes (disciplined by session types, under the Curry-Howard correspondence with linear logic), subsuming the results in [17].

2 $u\lambda_{\oplus}^{\zeta}$: Unrestricted Resources, Non-Determinism, and Failure

Syntax. We shall use x, y, \dots to range over *variables*, and i, j, \dots , as positive integers, to range over *indices*. Variable occurrences will be *annotated* to distinguish the kind of resource they should be substituted with (linear or unrestricted). With a slight abuse of terminology, we may write “linear variable” and “unrestricted variable” to refer to linear and unrestricted occurrences of a variable. As we will see, a variable’s annotation will be inconsequential for binding purposes. We write \tilde{x} to abbreviate x_1, \dots, x_n , for $n \geq 1$ and each x_i distinct.

► **Definition 1** ($u\widehat{\lambda}_{\oplus}^{\ell}$). We define terms (M, N) , bags (A, B) , and expressions (\mathbb{M}, \mathbb{N}) as:

(Annotations)	$[*] ::= [i] \mid [\ell] \quad i \in \mathbb{N}$
(Terms)	$M, N ::= x[*] \mid \lambda x.M \mid (M B) \mid M\langle\langle B/x \rangle\rangle \mid \mathbf{fail}^{\tilde{x}}$
(Linear Bags)	$C, D ::= 1 \mid \wr M \wr \cdot C$
(Unrestricted Bags)	$U, V ::= 1^! \mid \wr M \wr^! \mid U \diamond V$
(Bags)	$A, B ::= C \star U$
(Expressions)	$\mathbb{M}, \mathbb{N} ::= M \mid \mathbb{M} + \mathbb{N}$

To lighten up notation, we shall omit the annotation for linear variables. This way, e.g., we write $(\lambda x.x)B$ rather than $(\lambda x.x[\ell])B$.

Definition 1 introduces three syntactic categories: *terms* (in functional position); *bags* (multisets of resources, in argument position), and *expressions*, which are finite formal sums that denote possible results of a computation. Below we describe each category in details.

- Terms (unary expressions):
 - Variables: We write $x[\ell]$ to denote a *linear* occurrence of x , i.e., an occurrence that can only be substituted for linear resources. Similarly, $x[i]$ denotes an *unrestricted* occurrence of x , i.e., an occurrence that can only be substituted for a resource located at the i -th position of an unrestricted bag.
 - Abstractions $\lambda x.M$ of a variable x in a term M , which may have contain linear or unrestricted occurrences of x . This way, e.g., $\lambda x.x$ and $\lambda x.x[i]$ are linear and unrestricted versions of the identity function. Notice that the scope of x is M , as usual, and that $\lambda x.(\cdot)$ binds both linear and unrestricted occurrences of x .
 - Applications of a term M to a bag B (written $M B$) and the explicit substitution of a bag B for a variable x (written $\langle\langle B/x \rangle\rangle$) are as expected (cf. [1, 3]). Notice that in $M\langle\langle B/x \rangle\rangle$ the occurrences of x in M , linear and unrestricted, are bound. Some conditions apply to B : this will be evident later on, after we define our operational semantics (cf. Fig. 1).
 - The failure term $\mathbf{fail}^{\tilde{x}}$ denotes a term that will result from a reduction in which there is a lack or excess of resources, where \tilde{x} denotes a multiset of free linear variables that are encapsulated within failure.
- A bag B is defined as $C \star U$: the concatenation of a bag of linear resources C with a bag (actually, a list) of unrestricted resources U . We write $\wr M \wr$ to denote the linear bag that encloses term M , and use $\wr M \wr^!$ in the unrestricted case.
 - Linear bags (C, D, \dots) are multisets of terms. The empty linear bag is denoted 1 . We write $C_1 \cdot C_2$ to denote the concatenation of C_1 and C_2 ; this is a commutative and associative operation, where 1 is the identity.
 - Unrestricted bags (U, V, \dots) are ordered lists of terms. The empty unrestricted bag is denoted as $1^!$. The concatenation of U_1 and U_2 is denoted by $U_1 \diamond U_2$; this operation is associative but not commutative. Given $i \geq 1$, we write U_i to denote the i -th element of the unrestricted (ordered) bag U .
- Expressions are sums of terms, denoted as $\sum_i^n N_i$, where $n > 0$. Sums are associative and commutative; reordering of the terms in a sum is performed silently.

► **Example 2.** Consider the term $M := \lambda x.(x[1] \wr x \wr \star \wr y[1] \wr^!)$, which has linear and unrestricted occurrences of the same variable. This is an abstraction of an application that contains two bound occurrences of x (one unrestricted with index 1, and one linear) and one free unrestricted occurrence of $y[1]$, occurring in an unrestricted bag. As we will see, in $M (C \star U)$, the unrestricted occurrence “ $x[1]$ ” should be replaced by the first element of U .

The salient features of $u\lambda_{\oplus}^{\dagger}$ – the explicit construct for failure, the index annotations on unrestricted variables, the ordering of unrestricted bags – are *design choices* that will be responsible for interesting behaviours, as the following examples illustrate.

► **Example 3.** As already mentioned, $u\lambda_{\oplus}^{\dagger}$ admits different variants of the usual λ -term $\mathbf{I} = \lambda x.x$. We could have one in which x is a linear variable (i.e., $\lambda x.x$), but also several possibilities if x is unrestricted (i.e., $\lambda x.x[i]$, for some positive integer i). Interestingly, because $u\lambda_{\oplus}^{\dagger}$ supports failures, non-determinism, and the consumption of arbitrary terms of the unrestricted bag, these two variants of \mathbf{I} can have behaviours that may differ from the usual interpretation of \mathbf{I} . In Example 11 we will show that the six terms below give different behaviours:

$$\begin{array}{ll} \blacksquare M_1 = (\lambda x.x)(\lambda N \int \star U) & \blacksquare M_4 = (\lambda x.x[1])(1 \star \lambda N \int^! \diamond U) \\ \blacksquare M_2 = (\lambda x.x)(\lambda N_1 \int \cdot \lambda N_2 \int \star U) & \blacksquare M_5 = (\lambda x.x[1])(1 \star 1^! \diamond U) \\ \blacksquare M_3 = (\lambda x.x[1])(\lambda N \int \star 1^!) & \blacksquare M_6 = (\lambda x.x[i])(C \diamond U) \end{array}$$

We will see that M_1, M_4, M_6 reduce without failures, whereas M_2, M_3, M_5 reduce to failure.

► **Example 4.** Similarly, $u\lambda_{\oplus}^{\dagger}$ allows for several forms of the standard λ -terms such as $\Delta := \lambda x.xx$ and $\Omega := \Delta\Delta$, depending on whether the variable x is linear or unrestricted:

1. $\Delta_1 := \lambda x.(x(\lambda x \int \star 1^!))$ consists of an abstraction of a linear occurrence of x applied to a linear bag containing another linear occurrence of x . There are two forms of self-applications of Δ_1 , namely: $\Delta_1(\lambda \Delta_1 \int \star 1^!)$ and $\Delta_1(1 \star \lambda \Delta_1 \int^!)$.
2. $\Delta_4 := \lambda x.(x[1](\lambda x \int \star 1^!))$ consists of an unrestricted occurrence of x applied to a linear bag (containing a linear occurrence of x) that is composed with an empty unrestricted bag. Similarly, there are two self-applications of Δ_4 , namely: $\Delta_4(\lambda \Delta_4 \int \star 1^!)$ and $\Delta_4(1 \star \lambda \Delta_4 \int^!)$.
3. We show applications of an unrestricted variable occurrence ($x[2]$ or $x[1]$) applied to an empty linear bag composed with a non-empty unrestricted bag (of size two):

$\blacksquare \Delta_3 := \lambda x.(x[1](1 \star \lambda x[1] \int^! \diamond \lambda x[1] \int^!))$	$\blacksquare \Delta_6 := \lambda x.(x[1](1 \star \lambda x[1] \int^! \diamond \lambda x[2] \int^!))$
$\blacksquare \Delta_5 := \lambda x.(x[2](1 \star \lambda x[1] \int^! \diamond \lambda x[2] \int^!))$	$\blacksquare \Delta_7 := \lambda x.(x[2](1 \star \lambda x[1] \int^! \diamond \lambda x[1] \int^!))$

Applications between these terms express behaviour, similar to a lazy evaluation of Ω :

$$\begin{array}{ll} \blacksquare \Omega_5 := \Delta_5(1 \star \lambda \Delta_5 \int^! \diamond \lambda \Delta_5 \int^!) & \blacksquare \Omega_{6,5} := \Delta_6(1 \star \lambda \Delta_5 \int^! \diamond \lambda \Delta_6 \int^!) \\ \blacksquare \Omega_{5,6} := \Delta_5(1 \star \lambda \Delta_5 \int^! \diamond \lambda \Delta_6 \int^!) & \blacksquare \Omega_7 := \Delta_7(1 \star \lambda \Delta_7 \int^! \diamond \lambda \Delta_7 \int^!) \end{array}$$

The behaviour of these terms will be made explicit later on (see Examples 13 and 14).

Semantics. The semantics of $u\lambda_{\oplus}^{\dagger}$ captures that linear resources can be used only once, and that unrestricted resources can be used *ad libitum*. Thus, the evaluation of a function applied to a multiset of linear resources produces different possible behaviours, depending on the way these resources are substituted for the linear variables. This induces non-determinism, which we formalise using a *non-collapsing* approach, in which expressions keep all the different possibilities open, and do not commit to one of them. This is in contrast to *collapsing* non-determinism, in which selecting one alternative discards the rest.

We define a reduction relation \longrightarrow , which operates lazily on expressions. Informally, a β -reduction induces an explicit substitution of a bag $B = C \star U$ for a variable x , denoted $\langle\langle B/x \rangle\rangle$, in a term M . This explicit substitution is then expanded depending on whether the head of M has a linear or an unrestricted variable. Accordingly, in $u\lambda_{\oplus}^{\dagger}$ there are *two sources of failure*: one concerns mismatches on linear resources (required vs available resources); the other concerns the unavailability of a required unrestricted resource (an empty bag $1^!$).

To formalise reduction, we require a few auxiliary notions.

► **Definition 5.** *The multiset of free linear variables of \mathbb{M} , denoted $\text{mlfv}(\mathbb{M})$, is defined below. We denote by $[x]$ the multiset containing the linear variable x and $[x_1, \dots, x_n]$ denotes the multiset containing x_1, \dots, x_n . We write $\tilde{x} \uplus \tilde{y}$ to denote the multiset union of \tilde{x} , and \tilde{y} and $\tilde{x} \setminus y$ to express that every occurrence of y is removed from \tilde{x} .*

$$\begin{array}{ll}
 \text{mlfv}(x) = [x] & \text{mlfv}(x[i]) = \text{mlfv}(1) = \emptyset \\
 \text{mlfv}(C \star U) = \text{mlfv}(C) & \text{mlfv}(M B) = \text{mlfv}(M) \uplus \text{mlfv}(B) \\
 \text{mlfv}(\lambda M) = \text{mlfv}(M) & \text{mlfv}(\lambda x.M) = \text{mlfv}(M) \setminus \{x\} \\
 \text{mlfv}(M \langle\langle B/x \rangle\rangle) = (\text{mlfv}(M) \setminus \{x\}) \uplus \text{mlfv}(B) & \text{mlfv}(\lambda M \int \cdot C) = \text{mlfv}(M) \uplus \text{mlfv}(C) \\
 \text{mlfv}(\mathbb{M} + \mathbb{N}) = \text{mlfv}(\mathbb{M}) \uplus \text{mlfv}(\mathbb{N}) & \text{mlfv}(\mathbf{fail}^{x_1, \dots, x_n}) = [x_1, \dots, x_n]
 \end{array}$$

A term M (resp. expression \mathbb{M}) is called linearly closed if $\text{mlfv}(M) = \emptyset$ (resp. $\text{mlfv}(\mathbb{M}) = \emptyset$).

► **Notation 6.** We shall use the following notations.

- $N \in \mathbb{M}$ means that N occurs in the sum \mathbb{M} . Also, we write $N_i \in C$ to denote that N_i occurs in the linear bag C , and $C \setminus N_i$ to denote the linear bag obtained by removing one occurrence of N_i from C .
- $\#(x, M)$ denotes the number of (free) linear occurrences of x in M . Also, $\#(x, \tilde{y})$ denotes the number of occurrences of x in the multiset \tilde{y} .
- $\text{PER}(C)$ is the set of all permutations of a linear bag C and $C_i(n)$ denotes the n -th term in the (permuted) C_i .
- $\text{size}(C)$ denotes the number of terms in a linear bag C . That is, $\text{size}(1) = 0$ and $\text{size}(\lambda M \int \cdot C) = 1 + \text{size}(C)$. Given a bag $B = C \star U$, we define $\text{size}(B)$ as $\text{size}(C)$.

► **Definition 7 (Head).** Given a term M , we define $\text{head}(M)$ inductively as:

$$\begin{array}{lll}
 \text{head}(x) = x & \text{head}(M B) = \text{head}(M) & \text{head}(\lambda x.M) = \lambda x.M \\
 \text{head}(x[i]) = x[i] & \text{head}(\mathbf{fail}^{\tilde{x}}) = \mathbf{fail}^{\tilde{x}} & \text{head}(M \langle\langle B/x \rangle\rangle) = \begin{cases} \text{head}(M) & \text{if } \#(x, M) = \text{size}(B) \\ \mathbf{fail}^\emptyset & \text{otherwise} \end{cases}
 \end{array}$$

► **Definition 8 (Head Substitution).** Let M be a term such that $\text{head}(M) = x$. The head substitution of a term N for x in M , denoted $M\{N/x\}$, is inductively defined as follows (where $x \neq y$):

$$x\{N/x\} = N \quad (M B)\{N/x\} = (M\{N/x\}) B \quad (M \langle\langle B/y \rangle\rangle)\{N/x\} = (M\{N/x\}) \langle\langle B/y \rangle\rangle$$

When $\text{head}(M) = x[i]$, the head substitution $M\{N/x[i]\}$ works as expected: $x[i]\{N/x[i]\} = N$ as the base case of the definition. Finally, we define contexts for terms and expressions:

► **Definition 9 (Evaluation Contexts).** Contexts for terms ($CTerm$) and expressions ($CExpr$) are defined by the following grammar:

$$(CTerm) \quad C[\cdot], C'[\cdot] ::= ([\cdot])B \mid ([\cdot])\langle\langle B/x \rangle\rangle \quad (CExpr) \quad D[\cdot], D'[\cdot] ::= M + [\cdot]$$

Reduction is defined by the rules in Fig. 1. Rule $[R : \text{Beta}]$ induces explicit substitutions. Resource consumption is implemented by two fetch rules, which open up explicit substitutions:

- Rule $[R : \text{Fetch}^\ell]$, the *linear fetch*, ensures that the number of required resources matches the size of the linear bag C . It induces a sum of terms with head substitutions, each denoting the partial evaluation of an element from C . Thus, the size of C determines the summands in the resulting expression.
- Rule $[R : \text{Fetch}^!]$, the *unrestricted fetch*, consumes a resource occurring in a specific position of the unrestricted bag U via a linear head substitution of an unrestricted variable occurring in the head of the term. In this case, reduction results in an explicit substitution with U kept unaltered. Note that we check for the size of the linear bag C : in the case $\#(x, M) \neq \text{size}(C)$, the term evolves to a linear failure via Rule $[R : \text{fail}^\ell]$ (see Example 12). This is another design choice: linear failure is prioritised in $u\lambda_{\oplus}^{\ell}$.

$$\begin{array}{c}
\text{[R : Beta]} \frac{}{(\lambda x.M)B \longrightarrow M\langle\langle B/x \rangle\rangle} \\
\text{[R : Fetch}^\ell] \frac{\text{head}(M) = x \quad C = \{N_1\} \cdots \{N_k\}, k \geq 1 \quad \#(x, M) = k}{M\langle\langle C \star U/x \rangle\rangle \longrightarrow M\{N_1/x\}\langle\langle (C \setminus N_1) \star U/x \rangle\rangle + \cdots + M\{N_k/x\}\langle\langle (C \setminus N_k) \star U/x \rangle\rangle} \\
\text{[R : Fetch}^!] \frac{\text{head}(M) = x[i] \quad \#(x, M) = \text{size}(C) \quad U_i = \{N\}^!}{M\langle\langle C \star U/x \rangle\rangle \longrightarrow M\{N/x[i]\}\langle\langle C \star U/x \rangle\rangle} \\
\text{[R : Fail}^\ell] \frac{\#(x, M) \neq \text{size}(C) \quad \tilde{y} = (\text{mlfv}(M) \setminus x) \uplus \text{mlfv}(C)}{M\langle\langle C \star U/x \rangle\rangle \longrightarrow \sum_{\text{PER}(C)} \text{fail}^{\tilde{y}}} \\
\text{[R : Fail}^!] \frac{\#(x, M) = \text{size}(C) \quad U_i = \mathbf{1}^! \quad \text{head}(M) = x[i]}{M\langle\langle C \star U/x \rangle\rangle \longrightarrow M\{\text{fail}^0/x[i]\}\langle\langle C \star U/x \rangle\rangle} \\
\text{[R : Cons}_1] \frac{\tilde{y} = \text{mlfv}(C)}{(\text{fail}^x) C \star U \longrightarrow \sum_{\text{PER}(C)} \text{fail}^{x \uplus \tilde{y}}} \\
\text{[R : Cons}_2] \frac{\#(z, \tilde{x}) = \text{size}(C) \quad \tilde{y} = \text{mlfv}(C)}{\text{fail}^x \langle\langle C \star U/z \rangle\rangle \longrightarrow \sum_{\text{PER}(C)} \text{fail}^{(x \setminus z) \uplus \tilde{y}}} \\
\text{[R : ECont]} \frac{\mathbb{M} \longrightarrow \mathbb{M}'}{D[\mathbb{M}] \longrightarrow D[\mathbb{M}']} \quad \text{[R : TCont]} \frac{M \longrightarrow \sum_{i=1}^k M'_i}{C[M] \longrightarrow \sum_{i=1}^k C[M'_i]}
\end{array}$$

■ **Figure 1** Reduction rules for $u\lambda_{\oplus}^{\ddagger}$.

Four rules show reduction to failure terms, and accumulate free variables involved in failed reductions. Rules $\text{[R : Fail}^\ell]$ and $\text{[R : Fail}^!]$ formalise the failure to evaluate an explicit substitution $M\langle\langle C \star U/x \rangle\rangle$. The former rule targets a linear failure, which occurs when the size of C does not match the number of occurrences of x . The multiset \tilde{y} preserves all free linear variables in M and C . The latter rule targets an *unrestricted failure*, which occurs when the head of the term is $x[i]$ and U_i (i.e., the i -th element of U) is empty. In this case, failure preserves the free linear variables in M and C excluding the head unrestricted occurrence $x[i]$ which is replaced by fail^0 .

Rules $\text{[R : Cons}_1]$ and $\text{[R : Cons}_2]$ describe reductions that lazily consume the failure term, when a term has fail^x at its head position. The former rule consumes bags attached to it whilst preserving all its free linear variables; the latter rule consumes explicit substitution attached to it whilst also preserving all its free linear variables. The side condition $\#(z, \tilde{x}) = \text{size}(C)$ is necessary in Rule $\text{[R : Cons}_2]$ to avoid a clash with the premise of Rule $\text{[R : Fail}^\ell]$. Finally, Rules [R : ECont] and [R : TCont] state closure by the C and D contexts (cf. Def. 9).

Notice that the left-hand sides of the reduction rules in $u\lambda_{\oplus}^{\ddagger}$ do not interfere with each other. As a result, reduction in $u\lambda_{\oplus}^{\ddagger}$ satisfies a *diamond property*: for all $\mathbb{M} \in u\lambda_{\oplus}^{\ddagger}$, if there exist $\mathbb{M}_1, \mathbb{M}_2 \in u\lambda_{\oplus}^{\ddagger}$ such that $\mathbb{M} \longrightarrow \mathbb{M}_1$ and $\mathbb{M} \longrightarrow \mathbb{M}_2$, then there exists $\mathbb{N} \in u\lambda_{\oplus}^{\ddagger}$ such that $\mathbb{M}_1 \longrightarrow \mathbb{N} \longleftarrow \mathbb{M}_2$ (see [18] for more details).

► **Notation 10.** As usual, \longrightarrow^* denotes the reflexive-transitive closure of \longrightarrow . We write $\mathbb{N} \longrightarrow_{\text{[R]}} \mathbb{M}$ to denote that [R] is the last (non-contextual) rule used in the step from \mathbb{N} to \mathbb{M} .

► **Example 11** (Cont. Example 3). We illustrate different reductions for $\lambda x.x$ and $\lambda x.x[i]$.

1. $M_1 = (\lambda x.x)(\lambda N \int \star U)$ concerns a linear variable x with an linear bag containing one element. This is similar to the usual meaning of applying an identity function to a term:
 $(\lambda x.x)(\lambda N \int \star U) \xrightarrow{[\mathbf{R}:\mathbf{Beta}]} x \langle \langle \lambda N \int \star U / x \rangle \rangle \xrightarrow{[\mathbf{R}:\mathbf{Fetch}^\ell]} x \{ \{ N/x \} \} \langle \langle 1 \star U / x \rangle \rangle = N \langle \langle 1 \star U / x \rangle \rangle$,
 with a “garbage collector” that collects unused unrestricted resources.

2. $M_2 = (\lambda x.x)(\lambda N_1 \int \cdot \lambda N_2 \int \star U)$ concerns the case in which a linear variable x has a single occurrence but the linear bag has size two. Term M_2 reduces to a sum of failure terms:
 $(\lambda x.x)(\lambda N_1 \int \cdot \lambda N_2 \int \star U) \xrightarrow{[\mathbf{R}:\mathbf{Beta}]} x \langle \langle \lambda N_1 \int \cdot \lambda N_2 \int \star U / x \rangle \rangle \xrightarrow{[\mathbf{R}:\mathbf{Fail}^\ell]} \sum_{\text{PER}(C)} \mathbf{fail}^{\tilde{y}}$

for $C = \lambda N_1 \int \cdot \lambda N_2 \int$ and $\tilde{y} = \text{mlfv}(C)$.

3. $M_3 = (\lambda x.x[1])(\lambda N \int \star 1^!)$ represents an abstraction of an unrestricted variable, which aims to consume the first element of the unrestricted bag. Because this bag is empty, M_3 reduces to failure:

$$(\lambda x.x[1])(\lambda N \int \star 1^!) \xrightarrow{[\mathbf{R}:\mathbf{Beta}]} x[1] \langle \langle \lambda N \int \star 1^! / x \rangle \rangle \xrightarrow{[\mathbf{R}:\mathbf{fail}^\ell]} \mathbf{fail}^{\tilde{y}},$$

for $\tilde{y} = \text{mlfv}(N)$. Notice that $0 = \#(x, x[1]) \neq \text{size}(\lambda N \int) = 1$, since there are no linear occurrences of x in $x[1]$.

► **Example 12.** To illustrate the need to check “size(C)” in $[\mathbf{R}:\mathbf{Fail}^!]$, consider the term $x[1] \langle \langle \lambda M \int \star 1^! / x \rangle \rangle$, which features both a mismatch of linear bags for the linear variables to be substituted and an empty unrestricted bag with the need for the first element to be substituted. We check the size of the linear bag because we wish to prioritise the reduction of Rule $[\mathbf{R}:\mathbf{Fail}^\ell]$. Hence, in case of a mismatch of linear resources we wish not to perform a reduction via Rule $[\mathbf{R}:\mathbf{Fail}^!]$. This is a design choice: our semantics collapses linear failure at the earliest moment it arises.

► **Example 13** (Cont. Example 4). Self-applications of Δ_1 do not behave as an expected variation of a lazy reduction from Ω . Both $\Delta_1(\lambda \Delta_1 \int \star 1^!)$ and $\Delta_1(1 \star \lambda \Delta_1 \int^!)$ reduce to failure since the number of linear occurrences of x does not match the number of resources in the linear bag: $\Delta_1(\lambda \Delta_1 \int \star 1^!) \xrightarrow{(\lambda x.(x \int \star 1))} \langle \langle \lambda \Delta_1 \int \star 1^! / x \rangle \rangle \xrightarrow{(\lambda x.(x \int \star 1))} \mathbf{fail}^\emptyset$.

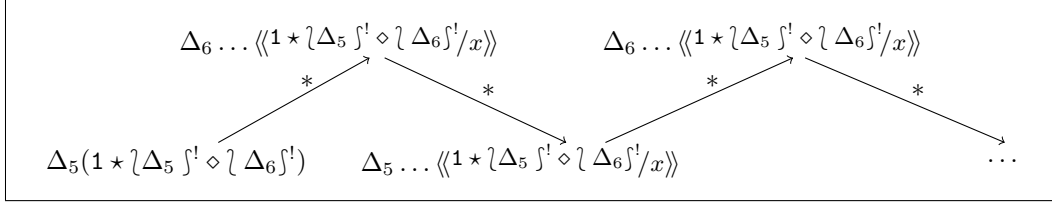
The term $\Delta_4(1 \star \lambda \Delta_4 \int^!)$ also fails: the linear bag is empty and there is one linear occurrence of x in Δ_4 . Note that $\Delta_4(\lambda \Delta_4 \int \star \lambda \Delta_4 \int^!)$ reduces to another application of Δ_4 before failing:

$$\begin{aligned} \Delta_4(\lambda \Delta_4 \int \star \lambda \Delta_4 \int^!) &= (\lambda x.(x[1](\lambda x \int \star 1^!)))(\lambda \Delta_4 \int \star \lambda \Delta_4 \int^!) \\ &\xrightarrow{[\mathbf{R}:\mathbf{Beta}]} (x[1](\lambda x \int \star 1^!)) \langle \langle \lambda \Delta_4 \int \star \lambda \Delta_4 \int^! / x \rangle \rangle \\ &\xrightarrow{[\mathbf{R}:\mathbf{Fetch}^!]} (\Delta_4(\lambda x \int \star 1^!)) \langle \langle \lambda \Delta_4 \int \star \lambda \Delta_4 \int^! / x \rangle \rangle \\ &\xrightarrow{*} \mathbf{fail}^\emptyset \langle \langle \lambda x \int \star 1^! / y \rangle \rangle \langle \langle \lambda \Delta_4 \int \star \lambda \Delta_4 \int^! / x \rangle \rangle \end{aligned}$$

Differently from [17], there are terms in $u\lambda_{\oplus}^{\zeta}$ that when applied to each other behave similarly to Ω , namely $\Omega_{5,6}$, $\Omega_{6,5}$, and Ω_7 (Example 4).

► **Example 14** (Cont. Example 4). The following reductions illustrate different behaviours provided that subtle changes are made within $u\lambda_{\oplus}^{\zeta}$ -terms:

- An interesting behaviour of $u\lambda_{\oplus}^{\zeta}$ is that variations of Δ can be applied to each other and appear alternately (highlighted in blue) in the functional position throughout the computation – this behaviour is illustrated in Fig. 2:



■ **Figure 2** An Ω -like behaviour in $u\lambda_{\oplus}^{\zeta}$ (cf. Example 14).

$$\begin{aligned}
\Omega_{5,6} &= \Delta_5(1 \star \Delta_5 \mathfrak{f}^! \diamond \wr \Delta_6 \mathfrak{f}^!) \\
&= (\lambda x.(x[2](1 \star \Delta_5 \mathfrak{f}^! \diamond \wr \Delta_6 \mathfrak{f}^!))) (1 \star \Delta_5 \mathfrak{f}^! \diamond \wr \Delta_6 \mathfrak{f}^!) \\
&\rightarrow_{[\text{R:Beta}]} (x[2](1 \star \Delta_5 \mathfrak{f}^! \diamond \wr \Delta_6 \mathfrak{f}^!)) \langle\langle 1 \star \Delta_5 \mathfrak{f}^! \diamond \wr \Delta_6 \mathfrak{f}^! / x \rangle\rangle \\
&\rightarrow_{[\text{R:Fetch}^!]} (\Delta_6(1 \star \Delta_5 \mathfrak{f}^! \diamond \wr \Delta_6 \mathfrak{f}^!)) \langle\langle 1 \star \Delta_5 \mathfrak{f}^! \diamond \wr \Delta_6 \mathfrak{f}^! / x \rangle\rangle \\
&\rightarrow_{[\text{R:Beta}]} (y[1](1 \star \Delta_5 \mathfrak{f}^! \diamond \wr \Delta_6 \mathfrak{f}^!)) \langle\langle (1 \star \Delta_5 \mathfrak{f}^! \diamond \wr \Delta_6 \mathfrak{f}^!) / y \rangle\rangle \langle\langle 1 \star \Delta_5 \mathfrak{f}^! \diamond \wr \Delta_6 \mathfrak{f}^! / x \rangle\rangle \\
&\rightarrow_{[\text{R:Fetch}^!]} (x[1](1 \star \Delta_5 \mathfrak{f}^! \diamond \wr \Delta_6 \mathfrak{f}^!)) \langle\langle (1 \star \Delta_5 \mathfrak{f}^! \diamond \wr \Delta_6 \mathfrak{f}^!) / y \rangle\rangle \langle\langle 1 \star \Delta_5 \mathfrak{f}^! \diamond \wr \Delta_6 \mathfrak{f}^! / x \rangle\rangle \\
&\rightarrow_{[\text{R:Fetch}^!]} (\Delta_5(1 \star \Delta_5 \mathfrak{f}^! \diamond \wr \Delta_6 \mathfrak{f}^!)) \langle\langle (1 \star \Delta_5 \mathfrak{f}^! \diamond \wr \Delta_6 \mathfrak{f}^!) / y \rangle\rangle \langle\langle 1 \star \Delta_5 \mathfrak{f}^! \diamond \wr \Delta_6 \mathfrak{f}^! / x \rangle\rangle \\
&\rightarrow \dots
\end{aligned}$$

- Applications of Δ_7 into two unrestricted copies of Δ_7 behave as Ω producing a non-terminating behaviour. Letting $B = 1 \star \Delta_7 \mathfrak{f}^! \diamond \wr \Delta_7 \mathfrak{f}^!$, we have:

$$\begin{aligned}
\Omega_7 &= (\lambda x.(x[2](1 \star \Delta_7 \mathfrak{f}^! \diamond \wr \Delta_7 \mathfrak{f}^!))) (1 \star \Delta_7 \mathfrak{f}^! \diamond \wr \Delta_7 \mathfrak{f}^!) \\
&\rightarrow_{[\text{R:Beta}]} (x[2](1 \star \Delta_7 \mathfrak{f}^! \diamond \wr \Delta_7 \mathfrak{f}^!)) \langle\langle 1 \star \Delta_7 \mathfrak{f}^! \diamond \wr \Delta_7 \mathfrak{f}^! / x \rangle\rangle \\
&\rightarrow_{[\text{R:Fetch}^!]} (\Delta_7(1 \star \Delta_7 \mathfrak{f}^! \diamond \wr \Delta_7 \mathfrak{f}^!)) \langle\langle 1 \star \Delta_7 \mathfrak{f}^! \diamond \wr \Delta_7 \mathfrak{f}^! / x \rangle\rangle \\
&\rightarrow_{[\text{R:Beta}]} (y[2](1 \star \Delta_7 \mathfrak{f}^! \diamond \wr \Delta_7 \mathfrak{f}^!)) \langle\langle B / y \rangle\rangle \langle\langle 1 \star \Delta_7 \mathfrak{f}^! \diamond \wr \Delta_7 \mathfrak{f}^! / x \rangle\rangle \\
&\rightarrow_{[\text{R:Fetch}^!]} (x[1](1 \star \Delta_7 \mathfrak{f}^! \diamond \wr \Delta_7 \mathfrak{f}^!)) \langle\langle B / y \rangle\rangle \langle\langle 1 \star \Delta_7 \mathfrak{f}^! \diamond \wr \Delta_7 \mathfrak{f}^! / x \rangle\rangle \\
&\rightarrow_{[\text{R:Fetch}^!]} (\Delta_7(1 \star \Delta_7 \mathfrak{f}^! \diamond \wr \Delta_7 \mathfrak{f}^!)) \langle\langle B / y \rangle\rangle \langle\langle 1 \star \Delta_7 \mathfrak{f}^! \diamond \wr \Delta_7 \mathfrak{f}^! / x \rangle\rangle \\
&\rightarrow \dots
\end{aligned}$$

Later on we will show that this term is well-formed (see Example 20) with respect to the intersection type system introduced in § 3.

3 Well-Formed Expressions via Intersection Types

We define *well-formed* $u\lambda_{\oplus}^{\zeta}$ -expressions by relying on a non-idempotent intersection type system, based on the system by Bucciarelli et al. [4]. Our system for well-formed expressions subsumes the one in [17]: it uses *strict* and *multiset* types to check linear bags; moreover, it uses *list* and *tuple* types to check unrestricted bags. As in [17], we write “well-formedness” (of terms, bags, and expressions) to stress that, unlike usual type systems, our system can account for terms that may reduce to the failure term (cf. Remark 22).

- **Definition 15** (Types for $u\lambda_{\oplus}^{\zeta}$). We define *strict*, *multiset*, *list*, and *tuple* types.

$$\begin{array}{ll}
(\textit{Strict}) & \sigma, \tau, \delta ::= \mathbf{unit} \mid (\pi, \eta) \rightarrow \sigma \\
(\textit{Multiset}) & \pi, \zeta ::= \bigwedge_{i \in I} \sigma_i \mid \omega \\
(\textit{List}) & \eta, \epsilon ::= \sigma \mid \epsilon \diamond \eta \\
(\textit{Tuple}) & (\pi, \eta)
\end{array}$$

11:10 Unrestricted Resources in Encoding Functions as Processes

A strict type can be the **unit** type or a functional type $(\pi, \eta) \rightarrow \sigma$, where (π, η) is a tuple type and σ is a strict type. Multiset types can be either the empty type ω or an intersection of strict types $\bigwedge_{i \in I} \sigma_i$, with I non-empty. The operator \wedge is commutative, associative, non-idempotent, that is, $\sigma \wedge \sigma \neq \sigma$, with identity ω . The intersection type $\bigwedge_{i \in I} \sigma_i$ is the type of a linear bag; the cardinality of I corresponds to its size.

A list type can be either a strict type σ or the composition $\epsilon \diamond \eta$ of two list types ϵ and η . We use the list type $\epsilon \diamond \eta$ to type the concatenation of two unrestricted bags. A tuple type (π, η) types the concatenation of a linear bag of type π with an unrestricted bag of type η . Notice that a list type $\epsilon \diamond \eta$ can be recursively unfolded into a finite composition of strict types $\sigma_1 \diamond \dots \diamond \sigma_n$, for some $n \geq 1$. In this case the length of $\epsilon \diamond \eta$ is n and that σ_i is its i -th strict type, for $1 \leq i \leq n$.

► **Notation 16.** Given $k \geq 0$, we write σ^k to stand for $\sigma \wedge \dots \wedge \sigma$ (k times, if $k > 0$) or for ω (if $k = 0$). Similarly, $\hat{x} : \sigma^k$ stands for $x : \sigma, \dots, x : \sigma$ (k times, if $k > 0$) or for $x : \omega$ (if $k = 0$). Given $k \geq 1$, we write $x^1 : \eta$ to stand for $x[1] : \eta_1, \dots, x[k] : \eta_k$.

► **Notation 17** ($\eta \propto \epsilon$). Let ϵ and η be two list types, with the length of ϵ greater or equal to that of η . Let us write ϵ_i and η_i to denote the i -th strict type in ϵ and η , respectively. We write $\eta \propto \epsilon$ meaning the initial sublist, 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 , $\epsilon'_i = \eta_i$.

Linear contexts range over Γ, Δ, \dots and unrestricted contexts range over Θ, Υ, \dots . They are defined by the following grammar:

$$\Gamma, \Delta ::= - \mid x : \sigma \mid \Gamma, x : \sigma \quad \Theta, \Upsilon ::= - \mid x^1 : \eta \mid \Theta, x^1 : \eta$$

The empty linear/unrestricted type assignment is denoted “-”. Linear variables can occur more than once in a linear context; they are assigned only strict types. For instance, $x : (\tau, \sigma) \rightarrow \tau, x : \tau$ is a valid context: it means that x can be of both type $(\tau, \sigma) \rightarrow \tau$ and τ . In contrast, unrestricted variables can occur at most once in unrestricted contexts; they are assigned only list types. The multiset of linear variables in Γ is denoted as $\text{dom}(\Gamma)$; similarly, $\text{dom}(\Theta)$ denotes the set of unrestricted variables in Θ .

Judgements are of the form $\Theta; \Gamma \models \mathbb{M} : \sigma$, where the left-hand side contexts are separated by “;” and $\mathbb{M} : \sigma$ means that \mathbb{M} has type σ . We write $\models \mathbb{M} : \sigma$ to denote $-; - \models \mathbb{M} : \sigma$.

► **Definition 18** (Well-formed $u\lambda_{\oplus}^{\downarrow}$ expressions). An expression \mathbb{M} is well-formed (wf, for short) if there exist Γ, Θ and τ such that $\Theta; \Gamma \models \mathbb{M} : \tau$ is entailed via the rules in Fig. 3.

We describe the well-formedness rules in Fig. 3.

- Rules $[\mathbf{F} : \text{var}^{\ell}]$ and $[\mathbf{F} : \text{var}^1]$ assign types to linear and unrestricted variables, respectively.
- Rule $[\mathbf{F} : \text{var}^1]$ resembles the *copy* rule [6] where we use a linear copy of an unrestricted variable $x[i]$ of type σ , typed with $x^1 : \eta$, and type the linear copy with the corresponding strict type η_i which in this case the linear copy x would have type equal to σ .
- Rules $[\mathbf{F} : 1^{\ell}]$ and $[\mathbf{F} : 1^1]$ assign types to the empty linear/unrestricted bag: 1 has type ω , whereas 1^1 has an arbitrary strict type σ . Arbitrariness is allowed since the substitution of an unrestricted variable for 1^1 leads to a **fail** term (Rule $[\mathbf{R} : \text{Fail}^1]$), which has an arbitrary strict type.
- Rule $[\mathbf{F} : \text{abs}]$ assigns type $(\sigma^k, \eta) \rightarrow \tau$ to an abstraction $\lambda z.M$, provided that the unrestricted occurrences of z may be typed by the unrestricted context containing $z^1 : \eta$, the linear occurrences of z are typed with the linear context containing $\hat{z} : \sigma^k$, for some $k \geq 0$, and there are no other linear occurrences of z in the linear context Γ .

$\text{[F:var}^\ell] \frac{}{\Theta; x : \sigma \models x : \sigma}$	$\text{[F:var}^!] \frac{\Theta, x^! : \eta; x : \eta_i, \Delta \models x : \sigma}{\Theta, x^! : \eta; \Delta \models x[i] : \sigma}$	$\text{[F:1}^\ell] \frac{}{\Theta; - \models 1 : \omega}$
$\text{[F:1}^!] \frac{}{\Theta; - \models 1^! : \sigma}$	$\text{[F:abs]} \frac{\Theta, z^! : \eta; \Gamma, \hat{z} : \sigma^k \models M : \tau \quad z \notin \text{dom}(\Gamma)}{\Theta; \Gamma \models \lambda z.M : (\sigma^k, \eta) \rightarrow \tau}$	
$\text{[F:app]} \frac{\Theta; \Gamma \models M : (\sigma^j, \eta) \rightarrow \tau \quad \Theta; \Delta \models B : (\sigma^k, \epsilon) \quad \eta \propto \epsilon}{\Theta; \Gamma, \Delta \models M B : \tau}$		$\text{[F:ex-sub]} \frac{\Theta, x^! : \eta; \Gamma, \hat{x} : \sigma^j \models M : \tau \quad \Theta; \Delta \models B : (\sigma^k, \epsilon) \quad \eta \propto \epsilon}{\Theta; \Gamma, \Delta \models M \langle\langle B/x \rangle\rangle : \tau}$
$\text{[F:bag]} \frac{\Theta; \Gamma \models C : \sigma^k \quad \Theta; - \models U : \eta}{\Theta; \Gamma \models C \star U : (\sigma^k, \eta)}$	$\text{[F:bag}^\ell] \frac{\Theta; \Gamma \models M : \sigma \quad \Theta; \Delta \models C : \sigma^k}{\Theta; \Gamma, \Delta \models \{M\} \cdot C : \sigma^{k+1}}$	
$\text{[F:bag}^!] \frac{\Theta; - \models M : \sigma}{\Theta; - \models \{M\}^! : \sigma}$	$\text{[F:} \diamond \text{bag}^!] \frac{\Theta; - \models U : \epsilon \quad \Theta; - \models V : \eta}{\Theta; - \models U \diamond V : \epsilon \diamond \eta}$	$\text{[F:fail]} \frac{\text{dom}(\Gamma) = \tilde{x}}{\Theta; \Gamma \models \text{fail}^{\tilde{x}} : \tau}$
$\text{[F:sum]} \frac{\Theta; \Gamma \models M : \sigma \quad \Theta; \Gamma \models N : \sigma}{\Theta; \Gamma \models M + N : \sigma}$		$\text{[F:weak]} \frac{\Theta; \Gamma \models M : \sigma \quad x \notin \text{dom}(\Gamma)}{\Theta; \Gamma, x : \omega \models M : \sigma}$

■ **Figure 3** Well-formedness rules for $u\lambda_{\oplus}^{\ddagger}$ (cf. Def. 18). In Rules [F:app] and [F:ex-sub]: $k, j \geq 0$.

- Rules [F:app] and [F:ex-sub] (for application and explicit substitution, resp.) use the condition $\eta \propto \epsilon$ (cf. Notation 17), which captures the portion of the unrestricted bag that is effectively used in a term: it ensures that ϵ can be decomposed into some ϵ' and ϵ'' , such that each type component ϵ'_i matches with η_i . If this requirement is satisfied, Rule [F:app] types an application $M B$ given that M has a functional type in which the left of the arrow is a tuple type (σ^j, η) whereas the bag B is typed with tuple (σ^k, ϵ) . Similarly, Rule [F:ex-sub] types the term $M \langle\langle B/x \rangle\rangle$ provided that B has the tuple type (σ^k, ϵ) and M is typed with the variable x having linear type assignment σ^j and unrestricted type assignment η .

► **Remark 19.** Differently from intersection type systems [4, 16], in Rules [F:app] and [F:ex-sub] there is no equality requirement between j and k , as we would like to capture terms that fail due to a mismatch in resources: we only require that the linear part of the tuples are composed of the same strict type, say σ . As a term can take an unrestricted bag with arbitrary size we only require that the elements of the unrestricted bag that are used have a “consistent” type, i.e., the type of the unrestricted bag satisfies the relation \propto with the unrestricted fragment of the corresponding tuple type.

There are four rules for bags:

- Rule [F:bag[!]] types an unrestricted bag $\{M\}^!$ with the type σ of M . Note that $\{x\}^!$, an unrestricted bag containing a linear variable x , is not well-formed, whereas $\{x[i]\}^!$ is well-formed.
- Rule [F:bag] assigns the tuple type (σ^k, η) to the concatenation of a linear bag of type σ^k with an unrestricted bag of type η .
- Rules [F:bag^ℓ] and [F:◊bag[!]] type the concatenation of linear and unrestricted bags.
- Rule [F:1[!]] allows an empty unrestricted bag to have an arbitrary σ type since it may be referred to by a variable for substitution: we must be able to compare its type with the type of unrestricted variables that may consume the empty bag (this reduction would inevitably lead to failure).

11:12 Unrestricted Resources in Encoding Functions as Processes

As in [17], Rule [F:fail] handles the failure term, and is the main difference with respect to standard type systems. Rules for sums and weakening ([F : sum] and [F : weak]) are standard.

► **Example 20** (Cont. Example 14). Term $\Delta_7 := \lambda x.x[2](1 \star \lambda x[1] \mathfrak{f}^! \diamond \lambda x[1] \mathfrak{f}^!)$ is well-formed, as ensured by the judgement $\Theta; - \models \Delta_7 : (\omega, \sigma' \diamond (\sigma^j, \sigma' \diamond \sigma') \rightarrow \tau) \rightarrow \tau$, whose derivation is given below:

- Π_3 is the derivation of $\Theta, x^! : \eta; - \models \lambda x[1] \mathfrak{f}^! : \sigma'$, for $\eta = \sigma' \diamond (\sigma^j, \sigma' \diamond \sigma') \rightarrow \tau$.
- Π_4 is the derivation: $\Theta, x^! : \eta; - \models x[2] : (\sigma^j, \sigma' \diamond \sigma') \rightarrow \tau$
- Π_5 is the derivation: $\Theta, x^! : \eta; x : \omega \models (1 \star \lambda x[1] \mathfrak{f}^! \diamond \lambda x[1] \mathfrak{f}^!) : (\omega, \sigma' \diamond \sigma')$

Therefore,

$$\frac{\frac{[\text{F:app}]}{\Theta, x^! : \eta; x : \omega \models x[2](1 \star \lambda x[1] \mathfrak{f}^! \diamond \lambda x[1] \mathfrak{f}^!) : \tau} \quad \frac{\Pi_5 \quad \Pi_4 \quad \sigma' \diamond \sigma' \propto \sigma' \diamond \sigma'}{\Theta; - \models \underbrace{\lambda x.(x[2](1 \star \lambda x[1] \mathfrak{f}^! \diamond \lambda x[1] \mathfrak{f}^!))}_{\Delta_7} : (\omega, \eta) \rightarrow \tau}}{[\text{F:abs}]} \quad \Theta; - \models \lambda x.(x[2](1 \star \lambda x[1] \mathfrak{f}^! \diamond \lambda x[1] \mathfrak{f}^!)) : (\omega, \eta) \rightarrow \tau$$

Well-formed expressions satisfy subject reduction (SR); see [18] for a full proof.

► **Theorem 21** (SR in $u\lambda_{\oplus}^!$). *If $\Theta; \Gamma \models \mathbb{M} : \tau$ and $\mathbb{M} \rightarrow \mathbb{M}'$ then $\Theta; \Gamma \models \mathbb{M}' : \tau$.*

Proof. By structural induction on the reduction rules. We proceed by analysing the rule applied in \mathbb{M} . An interesting case occurs when the rule is [F : Fetch[!]]: Then $\mathbb{M} = M \langle\langle C \star U/x \rangle\rangle$, where $U = \lambda N_1 \mathfrak{f}^! \diamond \dots \diamond \lambda N_l \mathfrak{f}^!$ and $\text{head}(M) = x[i]$. The reduction is as follows:

$$[\text{R : Fetch}^!] \frac{\text{head}(M) = x[i] \quad U_i = \lambda N_i \mathfrak{f}^!}{M \langle\langle C \star U/x \rangle\rangle \rightarrow M \{\{N_i/x[i]\}\} \langle\langle C \star U/x \rangle\rangle}$$

By hypothesis, one has the derivation:

$$[\text{F:ex-sub}] \frac{\Theta, x^! : \eta; \Gamma', \hat{x} : \sigma^j \models M : \tau \quad \frac{[\text{F:bag}] \frac{\Theta; \cdot \models U : \epsilon \quad \Theta; \Delta \models C : \sigma^k}{\Theta; \Delta \models C \star U : (\sigma^k, \epsilon)} \quad \Pi}{\Theta; \Gamma', \Delta \models M \langle\langle C \star U/x \rangle\rangle : \tau} \quad \eta \propto \epsilon}$$

where Π has the form

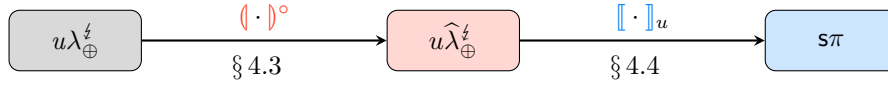
$$[\text{F} : \diamond \text{bag}^!] \frac{\frac{[\text{F:bag}^!] \frac{\Theta; \cdot \models N_1 : \epsilon_1}{\Theta; \cdot \models \lambda N_1 \mathfrak{f}^! : \epsilon_1} \quad \dots \quad [\text{F:bag}^!] \frac{\Theta; \cdot \models N_l : \epsilon_l}{\Theta; \cdot \models \lambda N_l \mathfrak{f}^! : \epsilon_l}}{\Theta; \cdot \models \lambda N_1 \mathfrak{f}^! \diamond \dots \diamond \lambda N_l \mathfrak{f}^! : \epsilon}}$$

with $\Gamma = \Gamma', \Delta$. Notice that if $\epsilon_i = \delta$ and $\eta \propto \epsilon$ then $\eta_i = \delta$. It can be shown that there exists a derivation Π_1 of $\Theta, x^! : \eta; \Gamma', \hat{x} : \sigma^j \models M \{\{N_i/x[i]\}\} : \tau$. Therefore, we have:

$$[\text{F:ex-sub}] \frac{\Theta, x^! : \eta; \Gamma', \hat{x} : \sigma^j \models M \{\{N_i/x[i]\}\} : \tau \quad \frac{[\text{F:bag}] \frac{\Theta; \cdot \models U : \epsilon \quad \Theta; \Delta \models C : \sigma^k}{\Theta; \Delta \models C \star U : (\sigma^k, \epsilon)} \quad \eta \propto \epsilon}{\Theta; \Gamma', \Delta \models M \{\{N_i/x[i]\}\} \langle\langle C \star U/x \rangle\rangle : \tau}}$$

◀

► **Remark 22** (Well-Formed vs Well-Typed Expressions). Our type system (and Theorem 21) can be specialised to the case of *well-typed* expressions that do not contain (and never reduce to) the failure term. In particular, Rules [F:app] and [F:ex-sub] would need to check that $\sigma^k = \sigma^j$, as failure can be caused due to a mismatch of linear resources. A difference between well typed and well formed expressions is that the former satisfy subject expansion, but the latter do not: expressions that lead to failure can be ill-typed yet failure itself is well-formed.



■ **Figure 4** Our two-step approach to encode $u\lambda_{\oplus}^{\zeta}$ into $s\pi$.

4 A Typed Encoding of $u\lambda_{\oplus}^{\zeta}$ into Concurrent Processes

We encode $u\lambda_{\oplus}^{\zeta}$ into $s\pi$, a session π -calculus that stands on a Curry-Howard correspondence between linear logic and session types (§ 4.1). We extend the two-step approach that we devised in [17] for the sub-calculus λ_{\oplus}^{ζ} (with linear resources only) (cf. Fig. 4). First, in § 4.3, we define an encoding $(\cdot)^{\circ}$ from well-formed expressions in $u\lambda_{\oplus}^{\zeta}$ to well-formed expressions in a variant of $u\lambda_{\oplus}^{\zeta}$ with *sharing*, dubbed $u\widehat{\lambda}_{\oplus}^{\zeta}$ (§ 4.2). Then, in § 4.4, we define an encoding $[\cdot]_u$ (for a name u) from well-formed expressions in $u\widehat{\lambda}_{\oplus}^{\zeta}$ to well-typed processes in $s\pi$.

We prove that $(\cdot)^{\circ}$ and $[\cdot]_u$ satisfy well-established correctness criteria [9, 14]: *type preservation*, *operational completeness*, *operational soundness*, and *success sensitiveness* (cf. [18]). Because $u\lambda_{\oplus}^{\zeta}$ includes unrestricted resources, the results given here strictly generalise those in [17].

4.1 $s\pi$: A Session-Typed π -Calculus

$s\pi$ is a π -calculus with *session types* [11, 12], which ensure that the endpoints of a channel perform matching actions. We consider the full process framework in [5], including constructs for specifying labelled choices and client/server connections; they will be useful to codify unrestricted resources and variables in $u\lambda_{\oplus}^{\zeta}$. Following [6, 19], $s\pi$ stands on a Curry-Howard correspondence between session types and a linear logic with dual modalities/types ($\&A$ and $\oplus A$), which define *non-deterministic* session behaviour. As in [6, 19], in $s\pi$, cut elimination corresponds to communication, proofs to processes, and propositions to session types.

Syntax. Names $x, y, z, w \dots$ denote the endpoints of protocols specified by session types. We write $P\{x/y\}$ for the capture-avoiding substitution of x for y in process P .

► **Definition 23** (Processes). *The syntax of $s\pi$ processes is given by the grammar below.*

$$\begin{aligned}
 P, Q ::= & \mathbf{0} \mid \bar{x}(y).P \mid x(y).P \mid x.l_i; P \mid x.\text{case}_{i \in I}\{l_i : P_i\} \mid x.\overline{\text{close}} \mid x.\text{close}; P \\
 & \mid (P \mid Q) \mid [x \leftrightarrow y] \mid (\nu x)P \mid !x(y).P \mid \bar{x}?(y).P \\
 & \mid x.\overline{\text{some}}; P \mid x.\overline{\text{none}} \mid x.\text{some}_{(w_1, \dots, w_n)}; P \mid (P \oplus Q)
 \end{aligned}$$

Process $\mathbf{0}$ denotes inaction. Process $\bar{x}(y).P$ sends a fresh name y along x and then continues as P . Process $x(y).P$ receives a name z along x and then continues as $P\{z/y\}$. Process $x.\text{case}_{i \in I}\{l_i : P_i\}$ is a branching construct, with labelled alternatives indexed by the finite set I : it awaits a choice on x with continuation P_j for each $j \in I$. Process $x.l_i; P$ selects on x the alternative indexed by i before continuing as P . Processes $x.\overline{\text{close}}$ and $x.\text{close}; P$ are complementary actions for closing session x . We sometimes use the shorthand notations $\bar{y}[]$ and $y[]; P$ to stand for $y.\overline{\text{close}}$ and $y.\text{close}; P$, respectively. Process $P \mid Q$ is the parallel execution of P and Q . The forwarder process $[x \leftrightarrow y]$ denotes a bi-directional link between sessions x and y . Process $(\nu x)P$ denotes the process P in which name x is kept private (local) to P . Process $!x(y).P$ defines a server that spawns copies of P upon requests on x . Process $\bar{x}?(y).P$ denotes a client that connects to a server by sending the fresh name y on x .

$\bar{x}(y).Q \mid x(y).P \longrightarrow (\nu y)(Q \mid P)$	$x.\overline{\text{some}}; P \mid x.\text{some}_{(w_1, \dots, w_n)}; Q \longrightarrow P \mid Q$
$Q \longrightarrow Q' \Rightarrow P \oplus Q \longrightarrow P \oplus Q'$	$x.\overline{\text{close}} \mid x.\text{close}; P \longrightarrow P$
$x.1_i; Q \mid x.\text{case}_{i \in I} \{1_i : P_i\} \longrightarrow Q \mid P_i$	$!x(y).Q \mid \bar{x}(y).P \longrightarrow (\nu x)(!x(y).Q \mid (\nu y)(Q \mid P))$
$(\nu x)([x \leftrightarrow y] \mid P) \longrightarrow P\{y/x\} \quad (x \neq y)$	$P \equiv P' \wedge P' \longrightarrow Q' \wedge Q' \equiv Q \Rightarrow P \longrightarrow Q$
$Q \longrightarrow Q' \Rightarrow P \mid Q \longrightarrow P \mid Q'$	$P \longrightarrow Q \Rightarrow (\nu y)P \longrightarrow (\nu y)Q$
$x.\overline{\text{none}} \mid x.\text{some}_{(w_1, \dots, w_n)}; Q \longrightarrow w_1.\overline{\text{none}} \mid \dots \mid w_n.\overline{\text{none}}$	

■ **Figure 5** Reduction for $\mathfrak{s}\pi$.

The remaining constructs come from [5] and introduce non-deterministic sessions which *may* provide a session protocol *or* fail. Process $x.\overline{\text{some}}; P$ confirms that the session on x will execute and continues as P . Process $x.\overline{\text{none}}$ signals the failure of implementing the session on x . Process $x.\text{some}_{(w_1, \dots, w_n)}; P$ specifies a dependency on a non-deterministic session x . This process can either (i) synchronise with an action $x.\overline{\text{some}}$ and continue as P , or (ii) synchronise with an action $x.\overline{\text{none}}$, discard P , and propagate the failure on x to (w_1, \dots, w_n) , which are sessions implemented in P . When x is the only session implemented in P , there is no tuple of dependencies (w_1, \dots, w_n) and so we write simply $x.\text{some}; P$. Finally, process $P \oplus Q$ denotes a non-deterministic choice between P and Q . We shall often write $\bigoplus_{i \in \{1, \dots, n\}} P_i$ to stand for $P_1 \oplus \dots \oplus P_n$. In $(\nu y)P$ and $x(y).P$ the occurrence of name y is binding, with scope P . The set of free names of P is denoted by $fn(P)$.

Semantics. The *reduction relation* of $\mathfrak{s}\pi$ specifies the computations that a process performs on its own (cf. Fig. 5). It is closed by *structural congruence*, denoted \equiv , which expresses basic identities for processes and the non-collapsing nature of non-determinism (cf. [18]).

The first reduction rule formalises communication, which concerns bound names only (internal mobility), as y is bound in $\bar{x}(y).Q$ and $x(y).P$. Reduction for the forwarder process leads to a substitution. The reduction rule for closing a session is self-explanatory, as is the rule in which prefix $x.\overline{\text{some}}$ confirms the availability of a non-deterministic session. When a non-deterministic session is not available, $x.\overline{\text{none}}$ triggers this failure to all dependent sessions w_1, \dots, w_n ; this may in turn trigger further failures (i.e., on sessions that depend on w_1, \dots, w_n). The remaining rules define contextual reduction with respect to restriction, composition, and non-deterministic choice.

Type System. Session types govern the behaviour of the names of a process. An assignment $x : A$ enforces the use of name x according to the protocol specified by A .

► **Definition 24** (Session Types). *Session types are given by*

$$A, B ::= \perp \mid \mathbf{1} \mid A \otimes B \mid A \wp B \mid \bigoplus_{i \in I} \{1_i : A_i\} \mid \&_{i \in I} \{1_i : A_i\} \mid !A \mid ?A \mid \&A \mid \oplus A$$

The multiplicative units \perp and $\mathbf{1}$ are used to type closed session endpoints. We use $A \otimes B$ to type a name that first outputs a name of type A before proceeding as specified by B . Similarly, $A \wp B$ types a name that first inputs a name of type A before proceeding as specified by B . Then, $!A$ types a name that repeatedly provides a service specified by A . Dually, $?A$ is the type of a name that can connect to a server offering A . Types $\bigoplus_{i \in I} \{1_i : A_i\}$ and $\&_{i \in I} \{1_i : A_i\}$ are assigned to names that can select and offer a labelled choice, respectively. Then we have the two modalities introduced in [5]. We use $\&A$ as the type of a (non-deterministic) session that *may produce* a behaviour of type A . Dually, $\oplus A$ denotes the type of a session that *may consume* a behaviour of type A .

$\text{[Tid]} \frac{}{[x \leftrightarrow y] \vdash x:A, y:\bar{A}; \Theta}$	$\text{[T1]} \frac{}{x.\text{close} \vdash x : \mathbf{1}; \Theta}$	$\text{[T}\perp\text{]} \frac{P \vdash \Delta; \Theta}{x.\text{close}; P \vdash x:\perp, \Delta; \Theta}$
$\text{[T}\otimes\text{]} \frac{P \vdash \Delta, y : A; \Theta \quad Q \vdash \Delta', x : B; \Theta}{\bar{x}(y).(P \mid Q) \vdash \Delta, \Delta', x : A \otimes B; \Theta}$	$\text{[T}\wp\text{]} \frac{P \vdash \Delta, y : C, x : D; \Theta}{x(y).P \vdash \Delta, x : C \wp D; \Theta}$	
$\text{[T}\oplus_{\tilde{w}}^x\text{]} \frac{P \vdash \tilde{w} : \&\Delta, x : A; \Theta}{x.\text{some}_{\tilde{w}}; P \vdash \tilde{w}:\&\Delta, x : \oplus A; \Theta}$	$\text{[T}\&_{\tilde{d}}^x\text{]} \frac{P \vdash \Delta, x : A; \Theta}{x.\overline{\text{some}}; P \vdash \Delta, x : \&A; \Theta}$	
$\text{[T}\&^x\text{]} \frac{}{x.\overline{\text{none}} \vdash x : \&A; \Theta}$	$\text{[T}\oplus\text{]} \frac{P \vdash \&\Delta; \Theta \quad Q \vdash \&\Delta; \Theta}{P \oplus Q \vdash \&\Delta; \Theta}$	
$\text{[T}\oplus_i\text{]} \frac{P \vdash \Delta, x : A_i; \Theta}{x.\mathbf{1}_i; P \vdash \Delta, x : \oplus_{i \in I} \{\mathbf{1}_i : A_i\}; \Theta}$	$\text{[T}\&\text{]} \frac{P_i \vdash \Delta, x : A_i; \Theta \quad (\forall i \in I)}{x.\text{case}_{i \in I} \{\mathbf{1}_i : P_i\} \vdash \Delta, x : \&_{i \in I} \{\mathbf{1}_i : A_i\}; \Theta}$	
$\text{[T?]} \frac{P \vdash \Delta; x : A, \Theta}{P \vdash \Delta, x : ?A; \Theta}$	$\text{[T!]} \frac{P \vdash y : A; \Theta}{!x(y).P \vdash x : !A; \Theta}$	$\text{[Tcopy]} \frac{P \vdash \Delta, y : A; x : A, \Theta}{\bar{x}?(y).P \vdash \Delta; x : A, \Theta}$

■ **Figure 6** Typing rules for $\mathcal{S}\pi$.

The two endpoints of a session should be *dual* to ensure absence of communication errors. The dual of a type A is denoted \bar{A} . Duality corresponds to negation $(\cdot)^\perp$ in linear logic [5].

► **Definition 25** (Duality). *Duality on types is given by:*

$$\begin{array}{llllll} \bar{\perp} = \perp & \bar{\mathbf{1}} = \mathbf{1} & \overline{A \otimes B} = \bar{A} \wp \bar{B} & \overline{\oplus_{i \in I} \{\mathbf{1}_i : A_i\}} = \&_{i \in I} \{\mathbf{1}_i : \bar{A}_i\} & \overline{\&A} = \oplus \bar{A} \\ \overline{!A} = ?A & \overline{?A} = !A & \overline{A \wp B} = \bar{A} \otimes \bar{B} & \overline{\&_{i \in I} \{\mathbf{1}_i : A_i\}} = \oplus_{i \in I} \{\mathbf{1}_i : \bar{A}_i\} & \overline{\oplus A} = \& \bar{A} \end{array}$$

Judgements are of the form $P \vdash \Delta; \Theta$, where P is a process, Δ is the linear context, and Θ is the unrestricted context. Both Δ and Θ contain assignments of types to names, but satisfy different substructural principles: while Θ satisfies weakening, contraction and exchange, Δ only satisfies exchange. The empty context is denoted “.”. We write $\&\Delta$ to denote that all assignments in Δ have a non-deterministic type, i.e., $\Delta = w_1:\&A_1, \dots, w_n:\&A_n$, for some A_1, \dots, A_n . The typing judgement $P \vdash \Delta$ corresponds to the logical sequent for classical linear logic, which can be recovered by erasing processes and name assignments.

Typing rules for processes in Fig. 6 correspond to proof rules in linear logic; we discuss some of them. Rule [Tid] interprets the identity axiom using the forwarder process. Rules [T1] and [T \perp] type the process constructs for session termination. Rules [T \otimes] and [T \wp] type output and input of a name along a session, resp. The last four rules are used to type process constructs related to non-determinism and failure. Rules [T $\&_{\tilde{d}}^x$] and [T $\&^x$] introduce a session of type $\&A$, which may produce a behaviour of type A : while the former rule covers the case in which $x : A$ is indeed available, the latter rule formalises the case in which $x : A$ is not available (i.e., a failure). Given a sequence of names $\tilde{w} = w_1, \dots, w_n$, Rule [T $\oplus_{\tilde{w}}^x$] accounts for the possibility of not being able to consume the session $x : A$ by considering sessions different from x as potentially not available. Rule [T \oplus] expresses non-deterministic choice of processes P and Q that implement non-deterministic behaviours only. Finally, Rule [T \oplus_i] and [T $\&$] correspond, resp., to selection and branching: the former provides a selection of behaviours along x as long as P is guarded with the i -th behaviour; the latter offers a labelled choice where each behaviour A_i is matched to a corresponding P_i .

The type system enjoys type preservation, a result that follows from the cut elimination property in linear logic; it ensures that the observable interface of a system is invariant under reduction. The type system also ensures other properties for well-typed processes (e.g. global progress, strong normalisation, and confluence); see [5] for details.

► **Theorem 26** (Type Preservation [5]). *If $P \vdash \Delta; \Theta$ and $P \longrightarrow Q$ then $Q \vdash \Delta; \Theta$.*

4.2 $u\widehat{\lambda}_{\oplus}^{\downarrow}$: An Auxiliary Calculus With Sharing

To facilitate the encoding of $u\lambda_{\oplus}^{\downarrow}$ into $\mathfrak{s}\pi$, we define $u\widehat{\lambda}_{\oplus}^{\downarrow}$: an auxiliary calculus whose constructs are inspired by the work of Gundersen et al. [10], Ghilezan et al. [8], and Kesner and Lengrand [13]. The syntax of $u\widehat{\lambda}_{\oplus}^{\downarrow}$ only modifies the syntax of terms, which is defined by the grammar below; variables $x[*]$, bags B , and expressions \mathbb{M} are as in Definition 1.

$$\begin{aligned} \text{(Terms)} \quad M, N, L ::= & x[*] \mid \lambda x.(M[\tilde{x} \leftarrow x]) \mid (M B) \mid M \langle N/x \rangle \mid M \llbracket U/x \rrbracket \\ & \mid \mathbf{fail}^{\tilde{x}} \mid M[\tilde{x} \leftarrow x] \mid (M[\tilde{x} \leftarrow x]) \langle\langle B/x \rangle\rangle \end{aligned}$$

We consider the *sharing construct* $M[\tilde{x} \leftarrow x]$ and two kinds of explicit substitutions: the *explicit linear substitution*, written $M \langle N/x \rangle$, and the *explicit unrestricted substitution*, written $M \llbracket U/x \rrbracket$. The term $M[\tilde{x} \leftarrow x]$ defines the sharing of variables \tilde{x} occurring in M using the linear variable x . We shall refer to x as *sharing variable* and to \tilde{x} as *shared variables*. A linear variable is only allowed to appear once in a term. Notice that \tilde{x} can be empty: $M[\leftarrow x]$ expresses that x does not share any variables in M . As in $u\lambda_{\oplus}^{\downarrow}$, the term $\mathbf{fail}^{\tilde{x}}$ explicitly accounts for failed attempts at substituting the variables in \tilde{x} .

We summarise some requirements. In $M[\tilde{x} \leftarrow x]$, we require: (i) every $x_i \in \tilde{x}$ occurs exactly once in M and that (ii) x_i is not a sharing variable. The occurrence of x_i can appear within the fail term $\mathbf{fail}^{\tilde{y}}$, if $x_i \in \tilde{y}$. In the explicit linear substitution $M \langle N/x \rangle$, we require: the variable x has to occur in M ; x cannot be a sharing variable; and x cannot be in an explicit linear substitution occurring in M ; all free *linear* occurrences of x in M are bound. In the explicit unrestricted substitution $M \llbracket U/x \rrbracket$, we require: all free *unrestricted* occurrences of x in M are bound; x cannot be in an explicit unrestricted substitution occurring in M . This way, e.g., $M' \langle L/x \rangle \langle N/x \rangle$ and $M' \langle U'/x \rangle \langle U/x \rangle$ are not valid terms in $u\widehat{\lambda}_{\oplus}^{\downarrow}$.

The following congruence will be important when proving encoding correctness.

► **Definition 27.** *The congruence \equiv_{λ} for $u\widehat{\lambda}_{\oplus}^{\downarrow}$ on terms and expressions is given by the identities below.*

$$\begin{aligned} M \llbracket U/x \rrbracket & \equiv_{\lambda} M, x \notin M \\ (MB) \langle N/x \rangle & \equiv_{\lambda} (M \langle N/x \rangle) B, x \notin \mathbf{fv}(B) \\ (MB) \llbracket U/x \rrbracket & \equiv_{\lambda} (M \llbracket U/x \rrbracket) B, x \notin \mathbf{fv}(B) \\ (MA)[\tilde{x} \leftarrow x] \langle\langle B/x \rangle\rangle & \equiv_{\lambda} (M[\tilde{x} \leftarrow x] \langle\langle B/x \rangle\rangle) A, x_i \in \tilde{x} \Rightarrow x_i \notin \mathbf{fv}(A) \\ M[\tilde{y} \leftarrow y] \langle\langle A/y \rangle\rangle [\tilde{x} \leftarrow x] \langle\langle B/x \rangle\rangle & \equiv_{\lambda} (M[\tilde{x} \leftarrow x] \langle\langle B/x \rangle\rangle) [\tilde{y} \leftarrow y] \langle\langle A/y \rangle\rangle, \\ & \quad x_i \in \tilde{x} \Rightarrow x_i \notin \mathbf{fv}(A), y_i \in \tilde{y} \Rightarrow y_i \notin \mathbf{fv}(B) \\ M \langle N_2/y \rangle \langle N_1/x \rangle & \equiv_{\lambda} M \langle N_1/x \rangle \langle N_2/y \rangle, x \notin \mathbf{fv}(N_2), y \notin \mathbf{fv}(N_1) \\ M \llbracket U_2/y \rrbracket \llbracket U_1/x \rrbracket & \equiv_{\lambda} M \llbracket U_1/x \rrbracket \llbracket U_2/y \rrbracket, x \notin \mathbf{fv}(U_2), y \notin \mathbf{fv}(U_1) \\ C[M] & \equiv_{\lambda} C[M'], \text{ with } M \equiv_{\lambda} M' \\ D[\mathbb{M}] & \equiv_{\lambda} D[\mathbb{M}'], \text{ with } \mathbb{M} \equiv_{\lambda} \mathbb{M}' \end{aligned}$$

The first rule states that we may remove unneeded unrestricted substitutions when the variable in concern does not appear within the term. The next three identities enforce that bags can always be moved in and out of all forms of explicit substitution, which are useful to manipulate expressions and to form a redex for Rule [R : Beta]. The other rules deal with permutation of explicit substitutions and contextual closure.

Well-formedness for $u\widehat{\lambda}_{\oplus}^{\downarrow}$, based on intersection types, is defined as in §3; see [18].

$$\begin{array}{l}
(x)^\bullet = x \qquad (x[i])^\bullet = x[i] \qquad (1)^\bullet = 1 \\
(1^!)^\bullet = 1^! \qquad (\mathbf{fail}^x)^\bullet = \mathbf{fail}^x \qquad (M B)^\bullet = (M)^\bullet (B)^\bullet \\
(\lambda M \int^!)^\bullet = \lambda M \int^! \qquad (\lambda M \int \cdot C)^\bullet = \lambda (M)^\bullet \int \cdot (C)^\bullet \qquad (C \star U)^\bullet = (C)^\bullet \star (U)^\bullet \\
(U \diamond V)^\bullet = U \diamond V \qquad (M \langle N/x \rangle)^\bullet = (M)^\bullet \langle (N)^\bullet /x \rangle \qquad (M \ll U/x \rrbracket)^\bullet = (M)^\bullet \ll (U)^\bullet /x \rrbracket \\
(\lambda x.M)^\bullet = \lambda x.((M \langle x_1, \dots, x_n/x \rangle)^\bullet [x_1, \dots, x_n \leftarrow x]) \quad \#(x, M) = n, \text{ each } x_i \text{ is fresh} \\
(M \langle C \star U/x \rangle)^\bullet = \begin{cases} \sum_{C_i \in \text{PER}(C)^\bullet} (M \langle \tilde{x}/x \rangle)^\bullet \langle C_i(1)/x_1 \rangle \cdots \langle C_i(k)/x_k \rangle \ll U/x \rrbracket, & \text{if } \#(x, M) = \text{size}(C) = k \\ (M \langle x_1, \dots, x_k/x \rangle)^\bullet [x_1, \dots, x_k \leftarrow x] \ll (C \star U)^\bullet /x \rrbracket, & \text{if } \#(x, M) = k \geq 0 \end{cases}
\end{array}$$

■ **Figure 7** Auxiliary Encoding: $u\lambda_{\oplus}^{\frac{k}{\oplus}}$ into $u\widehat{\lambda}_{\oplus}^{\frac{k}{\oplus}}$.

4.3 Encoding $u\lambda_{\oplus}^{\frac{k}{\oplus}}$ into $u\widehat{\lambda}_{\oplus}^{\frac{k}{\oplus}}$

We define an encoding $(\cdot)^\circ$ from well-formed terms in $u\lambda_{\oplus}^{\frac{k}{\oplus}}$ into $u\widehat{\lambda}_{\oplus}^{\frac{k}{\oplus}}$. This encoding relies on an intermediate encoding $(\cdot)^\bullet$ on $u\lambda_{\oplus}^{\frac{k}{\oplus}}$ -terms.

► **Notation 28.** Given a term M such that $\#(x, M) = k$ and a sequence of pairwise distinct fresh variables $\tilde{x} = x_1, \dots, x_k$ we write $M \langle \tilde{x}/x \rangle$ or $M \langle x_1, \dots, x_k/x \rangle$ to stand for $M \langle x_1/x \rangle \cdots \langle x_k/x \rangle$, i.e., a simultaneous linear substitution whereby each distinct linear occurrence of x in M is replaced by a distinct $x_i \in \tilde{x}$. Notice that each x_i has the same type as x . We use (simultaneous) linear substitutions to force all bound linear variables in $u\lambda_{\oplus}^{\frac{k}{\oplus}}$ to become shared variables in $u\widehat{\lambda}_{\oplus}^{\frac{k}{\oplus}}$.

► **Definition 29** (From $u\lambda_{\oplus}^{\frac{k}{\oplus}}$ to $u\widehat{\lambda}_{\oplus}^{\frac{k}{\oplus}}$). Let $M \in u\lambda_{\oplus}^{\frac{k}{\oplus}}$. Suppose $\Theta; \Gamma \models M : \tau$, with $\text{dom}(\Gamma) = \text{lfv}(M) = \{x_1, \dots, x_k\}$ and $\#(x_i, M) = j_i$. We define $(M)^\circ$ as

$$(M)^\circ = (M \langle \tilde{x}_1/x_1 \rangle \cdots \langle \tilde{x}_k/x_k \rangle)^\bullet [\tilde{x}_1 \leftarrow x_1] \cdots [\tilde{x}_k \leftarrow x_k]$$

where $\tilde{x}_i = x_{i_1}, \dots, x_{j_i}$ and the encoding $(\cdot)^\bullet : u\lambda_{\oplus}^{\frac{k}{\oplus}} \rightarrow u\widehat{\lambda}_{\oplus}^{\frac{k}{\oplus}}$ is defined in Fig. 7 on $u\lambda_{\oplus}^{\frac{k}{\oplus}}$ -terms. The encoding $(\cdot)^\circ$ extends homomorphically to expressions.

The encoding $(\cdot)^\circ$ converts n occurrences of x in a term into n distinct variables x_1, \dots, x_n . The sharing construct coordinates them by constraining each to occur exactly once within a term. We proceed in two stages. First, we share all linear free linear variables using $(\cdot)^\circ$: this ensures that free variables are replaced by shared variables which are then bound by the sharing construct. Second, we apply the encoding $(\cdot)^\bullet$ on the corresponding term. The encoding is presented in Fig. 7: $(\cdot)^\bullet$ maintains $x[i]$ unaltered, and acts homomorphically over concatenation of bags and explicit substitutions. The encoding renames bound variables with bound shared variables. As we will see, this will enable a tight operational correspondence result with $s\pi$. In [18] we establish the correctness of $(\cdot)^\circ$.

► **Example 30.** We apply the encoding $(\cdot)^\bullet$ in some of the $u\lambda_{\oplus}^{\frac{k}{\oplus}}$ -terms from Example 3: for simplicity, we assume that N and U have no free variables.

$$\begin{aligned}
((\lambda x.x) \lambda N \int \star U)^\bullet &= ((\lambda x.x)^\bullet)^\bullet (\lambda N \int \star U)^\bullet = \lambda x.x_1 [x_1 \leftarrow x] \lambda (N)^\bullet \int \star (U)^\bullet \\
((\lambda x.x[1]) \mathbf{1} \star \lambda N \int^! \diamond U)^\bullet &= ((\lambda x.x[1])^\bullet)^\bullet (\mathbf{1} \star \lambda N \int^! \diamond U)^\bullet = (\lambda x.x[1] [\leftarrow x]) \mathbf{1} \star \lambda (N)^\bullet \int^! \diamond (U)^\bullet
\end{aligned}$$

4.4 Encoding $u\widehat{\lambda}_{\oplus}^{\frac{k}{\oplus}}$ into $s\pi$

We now define our encoding of $u\widehat{\lambda}_{\oplus}^{\frac{k}{\oplus}}$ into $s\pi$, and establish its correctness.

► **Notation 31.** To help illustrate the behaviour of the encoding, we use the names x , x^ℓ , and $x^!$ to denote three distinct channel names: while x^ℓ is the channel that performs the linear substitution behaviour of the encoded term, channel $x^!$ performs the unrestricted behaviour.

► **Definition 32** (From $u\widehat{\lambda}_{\oplus}^{\downarrow}$ into $s\pi$: Expressions). Let u be a name. The encoding $\llbracket \cdot \rrbracket_u : u\widehat{\lambda}_{\oplus}^{\downarrow} \rightarrow s\pi$ is defined in Fig. 8.

Every (free) variable x in an $u\widehat{\lambda}_{\oplus}^{\downarrow}$ expression becomes a name x in its corresponding $s\pi$ process. As customary in encodings of λ into π , we use a name u to provide the behaviour of the encoded expression. In our case, u is a non-deterministic session: the encoded expression can be effectively available or not; this is signalled by prefixes $u.\overline{\text{some}}$ and $u.\overline{\text{none}}$, respectively.

We discuss the most salient aspects of the encoding in Fig. 8.

- While linear variables are encoded as in [17], the encoding of an unrestricted variable $x[j]$, not treated in [17], is much more interesting: it first connects to a server along channel x via a request $x^!(x_i)$ followed by a selection on $x_i.l_j$, which takes the j -th branch.
- The encoding of $\lambda x.M[\tilde{x} \leftarrow x]$ confirms its behaviour first followed by the receiving of a channel x . The channel x provides a linear channel x^ℓ and an unrestricted channel $x^!$ for dedicated substitutions of the linear and unrestricted bag components.
- We encode $M(C \star U)$ as a non-deterministic sum: an application involves a choice in the order in which the elements of C are substituted.
- The encoding of $C \star U$ synchronises with the encoding of $\lambda x.M[\tilde{x} \leftarrow x]$. The channel x^ℓ provides the linear behaviour of the bag C while $x^!$ provides the behaviour of U ; this is done by guarding the encoding of U with a server connection such that every time a channel synchronises with $!x^!(x_i)$ a fresh copy of U is spawned.
- The encoding of $\wr M \wr \cdot C$ synchronises with the encoding of $M[\tilde{x} \leftarrow x]$, just discussed. The name y_i is used to trigger a failure in the computation if there is a lack of elements in the encoding of the bag.
- The encoding of $M[\tilde{x} \leftarrow x]$ first confirms the availability of the linear behaviour along x^ℓ . Then it sends a name y_i , which is used to collapse the process in the case of a failed reduction. Subsequently, for each shared variable, the encoding receives a name, which will act as an occurrence of the shared variable. At the end, a failure prefix on x is used to signal that there is no further information to send over.
- The encoding of U synchronises with the last half encoding of $x[j]$; the name x_i selects the j -th term in the unrestricted bag.
- The encoding of $M \langle N/x \rangle$ is the composition of the encodings of M and N , where we await a confirmation of a behaviour along the variable that is being substituted.
- $M \llbracket U/x \rrbracket$ is encoded as the composition of the encoding of M and a server guarding the encoding of U : in order for $\llbracket M \rrbracket_u$ to gain access to $\llbracket U \rrbracket_{x_i}$ it must first synchronise with the server channel $x^!$ to spawn a fresh copy of U .
- The encoding of $\mathbb{M} + \mathbb{N}$ homomorphically preserves non-determinism. Finally, the encoding of $\text{fail}^{x_1, \dots, x_k}$ simply triggers failure on u and on each of x_1, \dots, x_k .

► **Example 33.** [Cont. Example 3] We illustrate the encoding $\llbracket \cdot \rrbracket$ on the $u\widehat{\lambda}_{\oplus}^{\downarrow}$ -terms/bags occurring in $M_1 = \lambda x.x_1[x_1 \leftarrow x](\wr \langle N \rangle \wr \cdot \star \langle U \rangle \cdot)$ as below:

$$\begin{aligned} \llbracket \lambda x.x_1[x_1 \leftarrow x] \rrbracket_v &= v.\overline{\text{some}}; v(x).x.\overline{\text{some}}; x(x^\ell).x(x^!).x[]; \llbracket x_1[x_1 \leftarrow x] \rrbracket_v \\ \llbracket \wr \langle N \rangle \wr \cdot \star \langle U \rangle \cdot \rrbracket_x &= x.\text{some}_{\text{fv}(\wr \langle N \rangle \cdot \wr)}; \overline{x}(x^\ell).(\llbracket \langle N \rangle \rrbracket_{x^\ell} \mid \overline{x}(x^!).(!x^!(x_i).\llbracket \langle U \rangle \rrbracket_{x_i} \mid \overline{x}[])) \end{aligned}$$

$$\begin{aligned}
\llbracket x \rrbracket_u &= x.\overline{\text{some}}; [x \leftrightarrow u] \\
\llbracket x[j] \rrbracket_u &= \overline{x^!}(x_i).x_i.l_j; [x_i \leftrightarrow u] \\
\llbracket \lambda x.M[\tilde{x} \leftarrow x] \rrbracket_u &= u.\overline{\text{some}}; u(x).x.\overline{\text{some}}; x(x^\ell).x(x^!).x.\text{close}; \llbracket M[\tilde{x} \leftarrow x] \rrbracket_u \\
\llbracket M[\tilde{x} \leftarrow x] \llbracket C \star U/x \rrbracket \rrbracket_u &= \bigoplus_{C_i \in \text{PER}(C)} (\nu x)(x.\overline{\text{some}}; x(x^\ell).x(x^!).x.\text{close}; \llbracket M[\tilde{x} \leftarrow x] \rrbracket_u \mid \llbracket C_i \star U \rrbracket_x) \\
\llbracket M(C \star U) \rrbracket_u &= \bigoplus_{C_i \in \text{PER}(C)} (\nu v)(\llbracket M \rrbracket_v \mid v.\text{some}_{u, \text{fv}(C)}; \overline{v}(x).([v \leftrightarrow u] \mid \llbracket C_i \star U \rrbracket_x)) \\
\llbracket C \star U \rrbracket_x &= x.\text{some}_{\text{fv}(C)}; \overline{x}(x^\ell).(\llbracket C \rrbracket_{x^\ell} \mid \overline{x}(x^!).(!x^!(x_i).\llbracket U \rrbracket_{x_i} \mid x.\overline{\text{close}})) \\
\llbracket \llbracket M \rrbracket \cdot C \rrbracket_{x^\ell} &= x^\ell.\text{some}_{\text{fv}(\llbracket M \rrbracket \cdot C)}; x^\ell(y_i).x^\ell.\text{some}_{y_i, \text{fv}(\llbracket M \rrbracket \cdot C)}; x^\ell.\overline{\text{some}}; \overline{x^\ell}(x_i). \\
&\quad (x_i.\text{some}_{\text{fv}(M)}; \llbracket M \rrbracket_{x_i} \mid \llbracket C \rrbracket_{x^\ell} \mid y_i.\overline{\text{none}}) \\
\llbracket \mathbf{1} \rrbracket_{x^\ell} &= x^\ell.\text{some}_\emptyset; x^\ell(y_n).(y_n.\overline{\text{some}}; y_n.\overline{\text{close}} \mid x^\ell.\text{some}_\emptyset; x^\ell.\overline{\text{none}}) \\
\llbracket \mathbf{1}^! \rrbracket_x &= x.\overline{\text{none}} \\
\llbracket \llbracket N \rrbracket^! \rrbracket_x &= \llbracket N \rrbracket_x \\
\llbracket U \rrbracket_x &= x.\text{case}_{U_i \in U} \{ \mathbf{1}_i : \llbracket U_i \rrbracket_x \} \\
\llbracket M \llbracket N/x \rrbracket \rrbracket_u &= (\nu x)(\llbracket M \rrbracket_u \mid x.\text{some}_{\text{fv}(N)}; \llbracket N \rrbracket_x) \\
\llbracket M \llbracket U/x \rrbracket \rrbracket_u &= (\nu x^!)(\llbracket M \rrbracket_u \mid !x^!(x_i).\llbracket U \rrbracket_{x_i}) \\
\llbracket M[\leftarrow x] \rrbracket_u &= x^\ell.\overline{\text{some}}.\overline{x^\ell}(y_i).(y_i.\text{some}_{u, \text{fv}(M)}; y_i.\text{close}; \llbracket M \rrbracket_u \mid x^\ell.\overline{\text{none}}) \\
\llbracket M[x_1, \dots, x_n \leftarrow x] \rrbracket_u &= x^\ell.\overline{\text{some}}.\overline{x^\ell}(y_1).(y_1.\text{some}_\emptyset; y_1.\text{close}; \mathbf{0} \mid \\
&\quad x^\ell.\overline{\text{some}}; x^\ell.\text{some}_{u, (\text{fv}(M) \setminus \{x_1, \dots, x_n\})}; x^\ell(x_1).\llbracket M[x_2, \dots, x_n \leftarrow x] \rrbracket_u) \\
\llbracket M + N \rrbracket_u &= \llbracket M \rrbracket_u \oplus \llbracket N \rrbracket_u \\
\llbracket \text{fail}^{x_1, \dots, x_k} \rrbracket_u &= u.\overline{\text{none}} \mid x_1.\overline{\text{none}} \mid \dots \mid x_k.\overline{\text{none}}
\end{aligned}$$

■ **Figure 8** Encoding $u\widehat{\lambda}_{\oplus}^{\xi}$ into $s\pi$ (cf. Def. 32).

$$\begin{aligned}
\llbracket \llbracket M_1 \rrbracket^\bullet \rrbracket_u &= \llbracket \lambda x.x_1[x_1 \leftarrow x] \llbracket \llbracket N \rrbracket^\bullet \rrbracket \star \llbracket \llbracket U \rrbracket^\bullet \rrbracket_x \rrbracket_u \\
&= (\nu v)(\llbracket \lambda x.x_1[x_1 \leftarrow x] \rrbracket_v \mid v.\text{some}_{u, \text{fv}(\llbracket N \rrbracket^\bullet)}; \overline{v}(x).([v \leftrightarrow u] \mid \llbracket \llbracket \llbracket N \rrbracket^\bullet \rrbracket \star \llbracket \llbracket U \rrbracket^\bullet \rrbracket_x \rrbracket_x)) \\
&= (\nu v)(v.\overline{\text{some}}; v(x).x.\overline{\text{some}}; x(x^\ell).x(x^!).x[]; x^\ell.\overline{\text{some}}.\overline{x^\ell}(y_1).(y_1.\text{some}_\emptyset; y_1[]; \mathbf{0} \mid \\
&\quad x^\ell.\overline{\text{some}}; x^\ell.\text{some}_u; x^\ell(x_1).x^\ell.\overline{\text{some}}.\overline{x^\ell}(y_2).(y_2.\text{some}_{u, x_1}; y_2[]; \llbracket x_1 \rrbracket_v \mid x^\ell.\overline{\text{none}})) \mid \\
&\quad v.\text{some}_{u, \text{fv}(\llbracket N \rrbracket^\bullet)}; \overline{v}(x).([v \leftrightarrow u] \mid \\
&\quad x.\text{some}_{\text{fv}(\llbracket N \rrbracket^\bullet)}; \overline{x}(x^\ell).(x^\ell.\text{some}_{\text{fv}(\llbracket N \rrbracket^\bullet)}; x^\ell.\text{some}_{y_1, \text{fv}(\llbracket \llbracket N \rrbracket^\bullet \rrbracket)}); \\
&\quad x^\ell.\overline{\text{some}}; \overline{x^\ell}(x_1).(x_1.\text{some}_{\text{fv}(\llbracket N \rrbracket^\bullet)}; \llbracket \llbracket N \rrbracket^\bullet \rrbracket_{x_1} \mid y_1.\overline{\text{none}} \mid x^\ell.\text{some}_\emptyset; x^\ell(y_2). \\
&\quad (y_2.\overline{\text{some}}; \overline{y_2}[] \mid x^\ell.\text{some}_\emptyset; x^\ell.\overline{\text{none}})) \mid \overline{x}(x^!).(!x^!(x_i).\llbracket \llbracket U \rrbracket^\bullet \rrbracket_{x_i} \mid \overline{x}[])))
\end{aligned}$$

We now encode intersection types (for $u\widehat{\lambda}_{\oplus}^{\xi}$) into session types (for $s\pi$).

► **Definition 34** (From $u\widehat{\lambda}_{\oplus}^{\xi}$ into $s\pi$: Types). *The translation $\llbracket \cdot \rrbracket$ in Figure 9 extends as follows to a context $\Gamma = x_1:\sigma_1, \dots, x_m:\sigma_m, v_1:\pi_1, \dots, v_n:\pi_n$ and a context $\Theta = x_1^!:\eta_1, \dots, x_k^!:\eta_k$:*

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

This encoding formally expresses how non-deterministic session protocols (typed with “&”) capture linear and unrestricted resource consumption in $u\widehat{\lambda}_{\oplus}^{\xi}$. Notice that the encoding of the multiset type π depends on two arguments (a strict type σ and a number $i \geq 0$) which are left unspecified above. This is crucial to represent failures in $u\widehat{\lambda}_{\oplus}^{\xi}$ as typable processes

$$\begin{aligned}
\llbracket \mathbf{unit} \rrbracket &= \& \mathbf{1} \\
\llbracket \eta \rrbracket &= \&_{\eta_i \in \eta} \{ \mathbf{1}_i; \llbracket \eta_i \rrbracket \} \\
\llbracket (\sigma^k, \eta) \rightarrow \tau \rrbracket &= \&(\overline{\llbracket (\sigma^k, \eta) \rrbracket}_{(\sigma, i)} \wp \llbracket \tau \rrbracket) \\
\llbracket (\sigma^k, \eta) \rrbracket_{(\sigma, i)} &= \oplus((\llbracket \sigma^k \rrbracket_{(\sigma, i)} \otimes (!\llbracket \eta \rrbracket) \otimes (\mathbf{1}))) \\
\llbracket \sigma \wedge \pi \rrbracket_{(\sigma, i)} &= \overline{\&((\oplus \perp) \otimes (\& \oplus ((\& \overline{\llbracket \sigma \rrbracket}) \wp (\overline{\llbracket \pi \rrbracket}_{(\sigma, i)}))))} \\
&= \oplus((\& \mathbf{1}) \wp (\oplus \&((\oplus \llbracket \sigma \rrbracket) \otimes (\llbracket \pi \rrbracket_{(\sigma, i)})))) \\
\llbracket \omega \rrbracket_{(\sigma, i)} &= \begin{cases} \overline{\&((\oplus \perp) \otimes (\& \oplus \perp))} & \text{if } i = 0 \\ \overline{\&((\oplus \perp) \otimes (\& \oplus ((\& \overline{\llbracket \sigma \rrbracket}) \wp (\overline{\llbracket \omega \rrbracket}_{(\sigma, i-1)}))))} & \text{if } i > 0 \end{cases}
\end{aligned}$$

■ **Figure 9** Encoding of intersection types into session types (cf. Def. 34).

in $\mathfrak{s}\pi$. For instance, given $(\sigma^j, \eta) \rightarrow \tau$ and (σ^k, η) , the well-formedness rule for application admits a mismatch ($j \neq k$, see [18]). In our proof of type preservation, the two arguments of the encoding are instantiated appropriately. Notice also how the client-server behaviour of unrestricted resources appears as “ $!\llbracket \eta \rrbracket$ ” in the encoding of the tuple type (σ^k, η) . With our encodings of expressions and types in place, we can now define our encoding of judgements:

► **Definition 35.** *If \mathbb{M} is an $u\hat{\lambda}_{\oplus}^{\ddagger}$ expression such that $\Theta; \Gamma \models \mathbb{M} : \tau$ then we define the encoding of the judgement to be: $\llbracket \mathbb{M} \rrbracket_u \vdash \llbracket \Gamma \rrbracket, u : \llbracket \tau \rrbracket; \llbracket \Theta \rrbracket$.*

The correctness of our encoding $\llbracket \cdot \rrbracket_u : u\hat{\lambda}_{\oplus}^{\ddagger} \rightarrow \mathfrak{s}\pi$, stated in Theorem 37 (and detailed in [18]), relies on a notion of *success* for both $u\hat{\lambda}_{\oplus}^{\ddagger}$ and $\mathfrak{s}\pi$, given by the \checkmark construct:

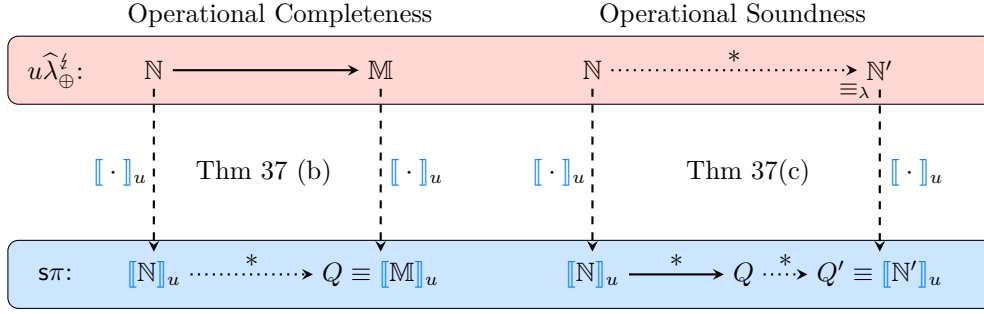
- **Definition 36.** *We extend the syntax of terms for $u\hat{\lambda}_{\oplus}^{\ddagger}$ and processes for $\mathfrak{s}\pi$ with \checkmark :*
- **(In $u\hat{\lambda}_{\oplus}^{\ddagger}$)** $\mathbb{M} \Downarrow_{\checkmark}$ *iff there exist M_1, \dots, M_k such that $\mathbb{M} \rightarrow^* M_1 + \dots + M_k$ and $\text{head}(M_j) = \checkmark$, for some $j \in \{1, \dots, k\}$ and term M'_j such that $M_j \equiv_{\lambda} M'_j$.*
 - **(In $\mathfrak{s}\pi$)** $P \Downarrow_{\checkmark}$ *holds whenever there exists a P' such that $P \rightarrow^* P'$ and P' contains an unguarded occurrence of \checkmark (i.e., an occurrence that does not occur behind a prefix).*

We now state operational correctness. Fig. 10 illustrates the relation between completeness and soundness that the encoding satisfies: solid arrows denote reductions assumed, dashed arrows denote the application of $\llbracket \cdot \rrbracket_u$, and dotted arrows denote the existing reductions that can be implied from the results.

We remark that since $u\hat{\lambda}_{\oplus}^{\ddagger}$ satisfies the diamond property, it suffices to consider completeness based on a single reduction ($\mathbb{N} \rightarrow \mathbb{M}$). Soundness uses the congruence \equiv_{λ} in Def. 27. We write $N \rightarrow_{\equiv_{\lambda}} N'$ iff $N \equiv_{\lambda} N_1 \rightarrow N_2 \equiv_{\lambda} N'$, for some N_1, N_2 . Then, $\rightarrow_{\equiv_{\lambda}}^*$ is the reflexive, transitive closure of $\rightarrow_{\equiv_{\lambda}}$. For success sensitivity, we decree $\llbracket \checkmark \rrbracket_u = \checkmark$. We have:

► **Theorem 37 (Operational Correctness).** *Let \mathbb{N} and \mathbb{M} be well-formed $u\hat{\lambda}_{\oplus}^{\ddagger}$ closed expressions.*

- (a) *(Type Preservation) Let B be a bag. We have:*
- (i) *If $\Theta; \Gamma \models B : (\sigma^k, \eta)$ then $\llbracket B \rrbracket_u \vdash \llbracket \Gamma \rrbracket, u : \llbracket (\sigma^k, \eta) \rrbracket_{(\sigma, i)}; \llbracket \Theta \rrbracket$.*
 - (ii) *If $\Theta; \Gamma \models \mathbb{M} : \tau$ then $\llbracket \mathbb{M} \rrbracket_u \vdash \llbracket \Gamma \rrbracket, u : \llbracket \tau \rrbracket; \llbracket \Theta \rrbracket$.*
- (b) *(Completeness) If $\mathbb{N} \rightarrow \mathbb{M}$ then there exists Q such that $\llbracket \mathbb{N} \rrbracket_u \rightarrow^* Q \equiv_{\lambda} \llbracket \mathbb{M} \rrbracket_u$.*
- (c) *(Soundness) If $\llbracket \mathbb{N} \rrbracket_u \rightarrow^* Q$ then $Q \rightarrow^* Q'$, $\mathbb{N} \rightarrow_{\equiv_{\lambda}}^* \mathbb{N}'$ and $\llbracket \mathbb{N}' \rrbracket_u \equiv Q'$, for some Q', \mathbb{N}' .*
- (d) *(Success Sensitivity) $\mathbb{M} \Downarrow_{\checkmark}$ if, and only if, $\llbracket \mathbb{M} \rrbracket_u \Downarrow_{\checkmark}$.*



■ **Figure 10** An overview of operational soundness and completeness for $\llbracket \cdot \rrbracket_u$.

Proof. All items are proven by structural induction; a detailed proof can be found in [18]. Below we present the most interesting case in the proof of *soundness*: the case when $\mathbb{N} = M(C \star U)$. Then,

$$\llbracket \mathbb{N} \rrbracket_u = \llbracket M(C \star U) \rrbracket_u = \bigoplus_{C_i \in \text{PER}(C)} (\nu v)(\llbracket M \rrbracket_v \mid v.\text{some}_{u, \text{fv}(C)}; \bar{v}(x).([v \leftrightarrow u] \mid \llbracket C_i \star U \rrbracket_x)).$$

The proof then proceeds by induction on the number of reduction steps k that can be taken from $\llbracket \mathbb{N} \rrbracket_u$, i.e., $\llbracket \mathbb{N} \rrbracket_u \rightarrow^k Q$. We will consider the case when $k \geq 1$, where for some process R and non-negative integers n, m such that $k = n + m$, we have the following:

$$\llbracket \mathbb{N} \rrbracket_u \rightarrow^m \bigoplus_{C_i \in \text{PER}(C)} (\nu v)(R \mid v.\text{some}_{u, \text{fv}(C)}; \bar{v}(x).([v \leftrightarrow u] \mid \llbracket C_i \star U \rrbracket_x)) \rightarrow^n Q$$

There are several cases to analyse depending on the values of m and n , and the shape of M . We consider $m = 0, n \geq 1$ and $M = (\lambda x.(M'[\tilde{x} \leftarrow x])) \langle N_1/y_1 \rangle \cdots \langle N_p/y_p \rangle \llbracket U_1/z_1 \rrbracket \cdots \llbracket U_q/z_q \rrbracket$, where $p, q \geq 0$. Then, $\llbracket \mathbb{N} \rrbracket_u$ can perform the following reduction:

$$\llbracket \mathbb{N} \rrbracket_u \rightarrow^* \bigoplus_{C_i \in \text{PER}(C)} (\nu \tilde{y}, \tilde{z}, x)(x.\overline{\text{some}}; x(x^\ell).x(x^1).x[]; \llbracket M'[\tilde{x} \leftarrow x] \rrbracket_u \mid Q'' \mid \llbracket C_i \star U \rrbracket_x) \quad (:= Q_3)$$

where Q'' defines the encoding of explicit substitutions within the encoded subterm M . Notice that:

$$\begin{aligned} \mathbb{N} &= (\lambda x.(M'[\tilde{x} \leftarrow x])) \langle N_1/y_1 \rangle \cdots \langle N_p/y_p \rangle \llbracket U_1/z_1 \rrbracket \cdots \llbracket U_q/z_q \rrbracket (C \star U) \\ &\equiv_{\lambda} (\lambda x.(M'[\tilde{x} \leftarrow x])) (C \star U) \langle N_1/y_1 \rangle \cdots \langle N_p/y_p \rangle \llbracket U_1/z_1 \rrbracket \cdots \llbracket U_q/z_q \rrbracket \\ &\rightarrow M'[\tilde{x} \leftarrow x] \langle \langle C \star U \rangle / x \rangle \langle N_1/y_1 \rangle \cdots \langle N_p/y_p \rangle \llbracket U_1/z_1 \rrbracket \cdots \llbracket U_q/z_q \rrbracket = \mathbb{M} \end{aligned}$$

where the congruence holds assuming the necessary α -renaming of variables. Finally, one can verify that $\llbracket \mathbb{M} \rrbracket_u = Q_3$, and the result follows. ◀

► **Example 38.** Recall again term M_1 from Example 3. It can be shown that $(M_1)^\bullet \rightarrow^* (\langle N \rangle)^\bullet \llbracket \langle U \rangle / x^1 \rrbracket$. To illustrate operational completeness, we can verify preservation of reduction, via $\llbracket \cdot \rrbracket$: reductions below use the rules for $s\pi$ in Figure 5 – see Figure 11.

5 Concluding Remarks

Summary. We have extended the line of work we developed in [17], on resource λ -calculi with firm logical foundations via typed concurrent processes. We presented $u\lambda_{\oplus}^{\xi}$, a resource calculus with non-determinism and explicit failures, with dedicated treatment for linear

$$\begin{aligned}
 & \llbracket (M_1)^\bullet \rrbracket = \\
 & (\nu v)(v.\overline{\text{some}}; v(x).x.\overline{\text{some}}; x(x^\ell).x(x^1).x[]; x^\ell.\overline{\text{some}}.\overline{x^\ell}(y_1).(y_1.\text{some}_\emptyset; y_1[]; \mathbf{0} | \\
 & \quad x^\ell.\overline{\text{some}}; x^\ell.\text{some}_u; x^\ell(x_1).x^\ell.\overline{\text{some}}.\overline{x^\ell}(y_2).(y_2.\text{some}_{u,x_1}; y_2.[]; \llbracket x_1 \rrbracket_v | x^\ell.\overline{\text{none}})) | \\
 & \quad v.\text{some}_{u,\text{fv}(\langle N \rangle^\bullet)}; \overline{v}(x).(\llbracket v \leftrightarrow u \rrbracket | x.\text{some}_{\text{fv}(\langle N \rangle^\bullet)}; \overline{x}(x^\ell).(x^\ell.\text{some}_{\text{fv}(\langle N \rangle^\bullet)}; x^\ell(y_1). \\
 & \quad x^\ell.\text{some}_{y_1,\text{fv}(\langle N \rangle^\bullet)}); x^\ell.\overline{\text{some}};\overline{x^\ell}(x_1).(x_1.\text{some}_{\text{fv}(\langle N \rangle^\bullet)}; \llbracket \langle N \rangle^\bullet \rrbracket_{x_1} | y_1.\overline{\text{none}} | x^\ell.\text{some}_\emptyset; \\
 & \quad x^\ell(y_2).(y_2.\overline{\text{some}}; \overline{y_2}[] | x^\ell.\text{some}_\emptyset; x^\ell.\overline{\text{none}})) | \overline{x}(x^1).(!x^1(x_i).\llbracket \langle U \rangle^\bullet \rrbracket_{x_i} | \overline{x}[]))) \\
 & \longrightarrow^3 (\nu x)(x.\overline{\text{some}}; x(x^\ell).x(x^1).x[]; x^\ell.\overline{\text{some}}.\overline{x^\ell}(y_1).(y_1.\text{some}_\emptyset; y_1[]; \mathbf{0} | x^\ell.\overline{\text{some}}; x^\ell.\text{some}_u; \\
 & \quad x^\ell(x_1).x^\ell.\overline{\text{some}}.\overline{x^\ell}(y_2).(y_2.\text{some}_{u,x_1}; y_2.[]; \llbracket x_1 \rrbracket_u | x^\ell.\overline{\text{none}})) | (x.\text{some}_{\text{fv}(\langle N \rangle^\bullet)}; \overline{x}(x^\ell). \\
 & \quad (x^\ell.\text{some}_{\text{fv}(\langle N \rangle^\bullet)}; x^\ell(y_1).x^\ell.\text{some}_{y_1,\text{fv}(\langle N \rangle^\bullet)}); x^\ell.\overline{\text{some}};\overline{x^\ell}(x_1).(x_1.\text{some}_{\text{fv}(\langle N \rangle^\bullet)}; \llbracket \langle N \rangle^\bullet \rrbracket_{x_1} | \\
 & \quad y_1.\overline{\text{none}} | x^\ell.\text{some}_\emptyset; x^\ell(y_2).(y_2.\overline{\text{some}}; \overline{y_2}[] | x^\ell.\text{some}_\emptyset; x^\ell.\overline{\text{none}})) | \overline{x}(x^1).(!x^1(x_i).\llbracket \langle U \rangle^\bullet \rrbracket_{x_i} | \overline{x}[]))) \\
 & \longrightarrow^2 (\nu x, x^\ell)(x(x^1).x[]; x^\ell.\overline{\text{some}}.\overline{x^\ell}(y_1).(y_1.\text{some}_\emptyset; y_1[]; \mathbf{0} | x^\ell.\overline{\text{some}}; x^\ell.\text{some}_u; x^\ell(x_1). \\
 & \quad x^\ell.\overline{\text{some}}.\overline{x^\ell}(y_2).(y_2.\text{some}_{u,x_1}; y_2.[]; \llbracket x_1 \rrbracket_u | x^\ell.\overline{\text{none}})) | (x^\ell.\text{some}_{\text{fv}(\langle N \rangle^\bullet)}; x^\ell(y_1). \\
 & \quad x^\ell.\text{some}_{y_1,\text{fv}(\langle N \rangle^\bullet)}; x^\ell.\overline{\text{some}};\overline{x^\ell}(x_1).(x_1.\text{some}_{\text{fv}(\langle N \rangle^\bullet)}; \llbracket \langle N \rangle^\bullet \rrbracket_{x_1} | y_1.\overline{\text{none}} | x^\ell.\text{some}_\emptyset; x^\ell(y_2). \\
 & \quad (y_2.\overline{\text{some}}; \overline{y_2}[] | x^\ell.\text{some}_\emptyset; x^\ell.\overline{\text{none}})) | \overline{x}(x^1).(!x^1(x_i).\llbracket \langle U \rangle^\bullet \rrbracket_{x_i} | \overline{x}[]))) \\
 & \longrightarrow (\nu x, x^\ell, x^1)(x[]; x^\ell.\overline{\text{some}}.\overline{x^\ell}(y_1).(y_1.\text{some}_\emptyset; y_1[]; \mathbf{0} | x^\ell.\overline{\text{some}}; x^\ell.\text{some}_u; x^\ell(x_1).x^\ell.\overline{\text{some}}. \\
 & \quad \overline{x^\ell}(y_2).(y_2.\text{some}_{u,x_1}; y_2.[]; \llbracket x_1 \rrbracket_u | x^\ell.\overline{\text{none}})) | (x^\ell.\text{some}_{\text{fv}(\langle N \rangle^\bullet)}; x^\ell(y_1).x^\ell.\text{some}_{y_1,\text{fv}(\langle N \rangle^\bullet)}; \\
 & \quad x^\ell.\overline{\text{some}};\overline{x^\ell}(x_1).(x_1.\text{some}_{\text{fv}(\langle N \rangle^\bullet)}; \llbracket \langle N \rangle^\bullet \rrbracket_{x_1} | y_1.\overline{\text{none}} | \\
 & \quad x^\ell.\text{some}_\emptyset; x^\ell(y_2).(y_2.\overline{\text{some}}; \overline{y_2}[] | x^\ell.\text{some}_\emptyset; x^\ell.\overline{\text{none}})) | !x^1(x_i).\llbracket \langle U \rangle^\bullet \rrbracket_{x_i} | \overline{x}[])) \\
 & \longrightarrow (\nu x^\ell, x^1)(x^\ell.\overline{\text{some}}.\overline{x^\ell}(y_1).(y_1.\text{some}_\emptyset; y_1.[]; \mathbf{0} | x^\ell.\overline{\text{some}}; x^\ell.\text{some}_u; x^\ell(x_1). \\
 & \quad x^\ell.\overline{\text{some}}.\overline{x^\ell}(y_2).(y_2.\text{some}_{u,x_1}; y_2.[]; \llbracket x_1 \rrbracket_u | x^\ell.\overline{\text{none}})) | (x^\ell.\text{some}_{\text{fv}(\langle N \rangle^\bullet)}; x^\ell(y_1). \\
 & \quad x^\ell.\text{some}_{y_1,\text{fv}(\langle N \rangle^\bullet)}; x^\ell.\overline{\text{some}};\overline{x^\ell}(x_1).(x_1.\text{some}_{\text{fv}(\langle N \rangle^\bullet)}; \llbracket \langle N \rangle^\bullet \rrbracket_{x_1} | y_1.\overline{\text{none}} | \\
 & \quad x^\ell.\text{some}_\emptyset; x^\ell(y_2).(y_2.\overline{\text{some}}; \overline{y_2}[] | x^\ell.\text{some}_\emptyset; x^\ell.\overline{\text{none}})) | !x^1(x_i).\llbracket \langle U \rangle^\bullet \rrbracket_{x_i})) \\
 & \longrightarrow (\nu x^\ell, y_1, x^1)(y_1.\text{some}_\emptyset; y_1.[]; \mathbf{0} | x^\ell.\overline{\text{some}}; x^\ell.\text{some}_u; x^\ell(x_1).x^\ell.\overline{\text{some}}.\overline{x^\ell}(y_2). \\
 & \quad (y_2.\text{some}_{u,x_1}; y_2.[]; \llbracket x_1 \rrbracket_u | x^\ell.\overline{\text{none}})) | (x^\ell.\text{some}_{y_1,\text{fv}(\langle N \rangle^\bullet)}; x^\ell.\overline{\text{some}};\overline{x^\ell}(x_1).(x_1.\text{some}_{\text{fv}(\langle N \rangle^\bullet)}; \\
 & \quad \llbracket \langle N \rangle^\bullet \rrbracket_{x_1} | y_1.\overline{\text{none}} | x^\ell.\text{some}_\emptyset; x^\ell(y_2).(y_2.\overline{\text{some}}; \overline{y_2}[] | x^\ell.\text{some}_\emptyset; x^\ell.\overline{\text{none}})) | !x^1(x_i).\llbracket \langle U \rangle^\bullet \rrbracket_{x_i})) \\
 & \longrightarrow^* (\nu x_1, x^1)(x_1.\overline{\text{some}}; [x_1 \leftrightarrow u] | x_1.\text{some}_{\text{fv}(\langle N \rangle^\bullet)}; \llbracket \langle N \rangle^\bullet \rrbracket_{x_1} | !x^1(x_i).\llbracket \langle U \rangle^\bullet \rrbracket_{x_i}) \\
 & \longrightarrow^* (\nu x^1)(\llbracket \langle N \rangle^\bullet \rrbracket_u | !x^1(x_i).\llbracket \langle U \rangle^\bullet \rrbracket_{x_i}) \\
 & = \llbracket \langle N \rangle^\bullet \llbracket \langle U \rangle^\bullet / x^1 \rrbracket_u
 \end{aligned}$$

■ **Figure 11** Illustrating operational correspondence, following Example 38.

and unrestricted resources. By means of examples, we illustrated the expressivity, (lazy) semantics, and design decisions underpinning $u\lambda_{\oplus}^{\zeta}$, and introduced a class of well-formed expressions based on intersection types, which includes fail-prone expressions. To bear witness to the logical foundations of $u\lambda_{\oplus}^{\zeta}$, we defined and proved correct a typed encoding into the concurrent calculus $\mathfrak{s}\pi$, which subsumes the one in [17]. We plan to study key properties for $u\lambda_{\oplus}^{\zeta}$ (such as solvability and normalisation) by leveraging our typed encoding into $\mathfrak{s}\pi$.

Related Work. With respect to previous resource calculi, a distinctive feature of $u\lambda_{\oplus}^{\dagger}$ is its support of explicit failures, which may arise depending on the interplay between (i) linear and unrestricted occurrences of variables in a term and (ii) associated resources in the bag. This feature allows $u\lambda_{\oplus}^{\dagger}$ to express variants of usual λ -terms (\mathbf{I} , Δ , Ω) not expressible in other resource calculi.

Related to $u\lambda_{\oplus}^{\dagger}$ is Boudol's work on a λ -calculus in which multiplicities can be infinite [1, 3]. An intersection type system is used to prove *adequacy* with respect to a testing semantics. However, failing behaviours as well as typability are not explored. Multiplicities can be expressed in $u\lambda_{\oplus}^{\dagger}$: a linear resource is available m times when the linear bag contains m copies of it; the term fails if the corresponding number of linear variables is different from m .

Also related is the resource λ -calculus by Pagani and Ronchi della Rocca [16], which includes linear and reusable resources; the latter are available in multisets, also called bags. In their setting, $M[N^!]$ denotes an application of a term M to a resource N that can be used *ad libitum*. Standard terms such as \mathbf{I} , Δ and Ω are expressed as $\lambda x.x$, $\Delta := \lambda x.x[x^!]$, and $\Omega := \Delta[\Delta^!]$, respectively; different variants are possible but cannot express the desired behaviour. A lazy reduction semantics is based on *baby* and *giant* steps: whereas the first consume one resource at each time, the second comprises several baby steps; combinations of the use of resources (by permuting resources in bags) are considered. A (non-idempotent) intersection type system is proposed: normalisation and a characterisation of solvability are investigated. Unlike our work, encodings into the π -calculus are not explored in [16].

References

- 1 Gérard Boudol. The lambda-calculus with multiplicities (abstract). In Eike 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 1–6. Springer, 1993. doi:10.1007/3-540-57208-2_1.
- 2 Gérard Boudol and Cosimo Laneve. The discriminating power of multiplicities in the lambda-calculus. *Inf. Comput.*, 126(1):83–102, 1996. doi:10.1006/inco.1996.0037.
- 3 Gérard Boudol and Cosimo Laneve. lambda-calculus, multiplicities, and the pi-calculus. In *Proof, Language, and Interaction, Essays in Honour of Robin Milner*, pages 659–690, 2000.
- 4 Antonio Bucciarelli, Delia Kesner, and Daniel Ventura. Non-idempotent intersection types for the lambda-calculus. *Logic Journal of the IGPL*, 25(4):431–464, 2017.
- 5 Luís Caires and Jorge A. Pérez. Linearity, control effects, and behavioral types. In Hongseok 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. doi:10.1007/978-3-662-54434-1_9.
- 6 Luís Caires and Frank 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. doi:10.1007/978-3-642-15375-4_16.
- 7 Maurizio Dominici, Simona Ronchi Della Rocca, and Paolo Tranquilli. Standardization in resource lambda-calculus. In *Proceedings 2nd International Workshop on Linearity, LINEARITY 2012, Tallinn, Estonia, 1 April 2012.*, pages 1–11, 2012. doi:10.4204/EPTCS.101.1.
- 8 Silvia Ghilezan, Jelena Ivetic, Pierre Lescanne, and Silvia Likavec. Intersection types for the resource control lambda calculi. In *Theoretical Aspects of Computing - ICTAC 2011 - 8th International Colloquium, Johannesburg, South Africa, August 31 - September 2, 2011. Proceedings*, pages 116–134, 2011. doi:10.1007/978-3-642-23283-1_10.

- 9 Daniele Gorla. Towards a unified approach to encodability and separation results for process calculi. *Inf. Comput.*, 208(9):1031–1053, 2010. doi:10.1016/j.ic.2010.05.002.
- 10 Tom Gundersen, Willem Heijltjes, and Michel Parigot. Atomic lambda calculus: A typed lambda-calculus with explicit sharing. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*, pages 311–320, 2013. doi:10.1109/LICS.2013.37.
- 11 Kohei Honda. Types for dyadic interaction. In Eike 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. doi:10.1007/3-540-57208-2_35.
- 12 Kohei Honda, Vasco Thudichum Vasconcelos, and Makoto Kubo. Language primitives and type discipline for structured communication-based programming. In Chris 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. doi:10.1007/BFb0053567.
- 13 Delia Kesner and Stéphane Lengrand. Resource operators for lambda-calculus. *Inf. Comput.*, 205(4):419–473, 2007. doi:10.1016/j.ic.2006.08.008.
- 14 Dimitrios Kouzapas, Jorge A. Pérez, and Nobuko Yoshida. On the relative expressiveness of higher-order session processes. *Inf. Comput.*, 268, 2019. doi:10.1016/j.ic.2019.06.002.
- 15 Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, I. *Inf. Comput.*, 100(1):1–40, 1992. doi:10.1016/0890-5401(92)90008-4.
- 16 Michele Pagani and Simona Ronchi Della Rocca. Solvability in resource lambda-calculus. In C.-H. Luke 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. doi:10.1007/978-3-642-12032-9_25.
- 17 Joseph W. N. Paulus, Daniele Nantes-Sobrinho, and Jorge A. Pérez. Non-deterministic functions as non-deterministic processes. In Naoki 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. doi:10.4230/LIPICs.FSCD.2021.21.
- 18 Joseph W. N. Paulus, Daniele Nantes-Sobrinho, and Jorge A. Pérez. Types and Terms Translated: Unrestricted Resources in Encoding Functions as Processes (Extended Version). *CoRR*, abs/2112.01593, 2021. arXiv:2112.01593.
- 19 Philip Wadler. Propositions as sessions. In Peter Thiemann and Robby Bruce Findler, editors, *ACM SIGPLAN International Conference on Functional Programming, ICFP'12, Copenhagen, Denmark, September 9-15, 2012*, pages 273–286. ACM, 2012. doi:10.1145/2364527.2364568.