sMALL CaPS

An Infinitary Linear Logic for a Calculus of Pure Sessions

Francesco Dagnino
University of Genova
Italy

Luca Padovani
University of Camerino
Italy

Abstract

We present an infinitary version of Multiplicative Additive Linear Logic (sMALL) that serves as logical foundation for a Calculus of Pure Sessions (CaPS). sMALL is infinitary not only because proof derivations may be infinite, but also because propositions themselves may be infinite. In this sense, sMALL differs from other related extensions of Linear Logic based on least and greatest fixed points. Also, all sMALL derivations are valid proofs by construction. sMALL enables the description and implementation in CaPS of recursive communication protocols – like authentication, coordination, consensus – in which termination is not decided autonomously by a single process, but results from some *negotiation* involving two or more interacting processes. We prove that sMALL is sound and that it enjoys cut elimination. We also prove a relative completeness result showing that a certain class of well-behaving CaPS processes are well typed in sMALL. Finally, we show that sMALL can be easily extended to address a broader class of *fairly terminating* processes, those that terminate under a suitable fairness assumption.

CCS Concepts

• **Theory of computation** → **Linear logic**; **Type theory**; *Process calculi*; *Program analysis*; *Program specifications*.

Keywords

infinitary linear logic, session types, termination, cut elimination

ACM Reference Format:
Francesco Dagnino and Luca Padovani. 2023. sMALL CaPS: An Infinitary Linear Logic for a Calculus of Pure Sessions. In *International Symposium on Principles and Practice of Declarative Programming (PPDP 2023), October 22–23, 2023, Lisboa, Portugal.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3610612.3610614

1 Introduction

Session types [25, 26, 28] are communication protocols in the form of types that enable the verification and the enforcement of various safety and liveness properties through a session type system. Since the seminal work of Honda [25], session type systems have always manifested some connections with linear logic [23], although it was only in the paper of Caires and Pfenning [4] and subsequent works [5, 36] that these connections have been developed into a proper Curry-Howard correspondence. According to this correspondence, (linear logic) propositions are session types, proofs are well-typed processes and cut reductions are communication steps among processes. The fact that session type systems rest on such logical foundations has many positive consequences. In particular, desirable safety and liveness properties of sessions such as the absence of communication errors, deadlock and livelock freedom, and termination become straightforward consequences of well-known properties of the logic, most notably cut elimination. However, there are cases in which the logic "gets in the way", in the sense that it limits the class of communication protocols that can be described and/or implemented and for which the aforementioned properties can be guaranteed. In this work we focus on some of these limitations concerning *recursive communication protocols*, those describing unbounded interactions between processes.

Extensions of linear logic that support (co)recursion have been actively studied for quite some time [1–3, 21]. In fact, some of these extensions have already been used as logical foundations for session type systems featuring (co)recursive session types [8, 18, 19, 27, 30]. All of these systems share the representation of recursive propositions/types by means of *least and greatest fixed point operators*, each being the dual of the other. Least and greatest fixed points fit naturally in the setting of linear logic and substantially enhance the expressiveness of its multiplicative additive fragment. For example, (co)exponentials of linear logic can be encoded using fixed points [3, 32] and small fragments of the logic become Turing complete when enriched with fixed points [20]. Despite these expressiveness results, there exist remarkably simple interaction patterns whose implementation does not correspond to a valid proof in any of the known extensions of linear logic with fixed points.

To illustrate this limitation, imagine the interaction between a player and a slot machine: the player may decide how many games are played, but has no control over which games are won; the slot machine may decide which games are won by the player, but has no control over the number of played games. We can describe the interaction protocol between the player and the slot machine (from the standpoint of the player) as a recursive session type A such that

$$A \cong \oplus\{\mathsf{play} : \&\{\mathsf{win} : A, \mathsf{lose} : A\}, \mathsf{stop} : \bot\} \tag{1}$$

where, for now, the $\cong$ relation denotes some unspecified equivalence between A on the left-hand side and the (partially unfolded) session type on the right-hand side of eq. (1). The $\oplus$ connective (which we adopt in n-ary and tagged form for generality and readability) describes a choice performed *internally* by the player. The $\&$ connective (also in n-ary and tagged form) describes a choice performed *externally* by the slot machine. After each notification from the slot machine, the protocol restarts. Once the player has decided to stop playing, the protocol ends ($\bot$).

PPDP 2023, October 22–23, 2023, Lisboa, Portugal

ACM ISBN 979-8-4007-0812-1/23/10
https://doi.org/10.1145/3610612.3610614

When we define A in terms of fixed points in the aforementioned logics, we have to decide whether A is a least or greatest solution of eq. (1). This choice affects the set of proofs of $\vdash A, \mathbf{1}$ (which correspond to well-typed players of the slot machine) and of $\vdash A^\perp$ (which correspond to well-typed slot machines). If we use a *least fixed point* for defining A, then each proof of $\vdash A, \mathbf{1}$ encodes *a priori* a finite upper bound to the number of games the player is willing to play. Although this limitation appears to be generally sensible, it prevents for example the modeling of a player who plays relentlessly until a game is won. If we use a *greatest fixed point* for defining A, then the proof of $\vdash A, \mathbf{1}$ corresponding to the relentless player becomes valid, but then $\vdash A^\perp$ is no longer provable: by duality, $A^\perp$ would be defined by a least fixed point, thus preventing the slot machine to offer an arbitrary number of games. Note that the combination of the relentless player and of the slot machine still enjoys termination, at least under the assumption that the slot machine is not completely unfair.

In general, the use of least and greatest fixed points for defining (co)recursive session types leads to a rough classification of well-typed processes into two disjoint categories: those typed according to a least fixed point *internally* encode the maximum number of iterations they are willing to perform; those typed according to a greatest fixed point let the other party to *externally* decide how many iterations they perform. The space in between these two categories, however, is not empty. It is populated by processes where the (maximum) number of iterations is not fixed *a priori*, but depends on some *negotiations* that take place when they execute. Besides the relentless player and the slot machine, other typical examples of negotiation arise in protocols for authentication, for the orchestration and coordination of services, for the agreement and consensus in distributed systems. Currently, these processes fall outside the scope of all session type systems based on linear logics with least and greatest fixed points.

The main contribution of this paper is an infinitary version of Multiplicative Additive Linear Logic (sMALL) serving as logical foundation for a Calculus of Pure Sessions (CaPS) in which we can model (among other things) terminating interactions that may involve negotiations. We dub the logic sMALL to emphasize the fact that it is tailored towards the description of *session-based communication protocols*, where the aforementioned behaviours are of central interest. Unlike other logics with explicit support for (co)recursion [1–3, 18, 19, 21], sMALL does *not* make use of fixed points in propositions. Instead, sMALL propositions are *possibly infinite trees* built with the usual multiplicative additive connectives and units of linear logic. In other words, we take the equivalence relation $\cong$ of eq. (1) to be equality between possibly infinite trees. Similarly, sMALL proofs are *possibly infinite derivations* built using the usual proof rules for the multiplicative additive connectives and units of linear logic. Not surprisingly, as it is well known from other infinitary logics like *e.g.* μMALL$^\infty$ [2, 3, 21], in order to retain the consistency of the logic not all infinite derivations can be considered valid proofs. For example, both the relentless player and the unfair slot machine always sending lose represent infinite derivations for $\vdash A, \mathbf{1}$ and $\vdash A^\perp$, respectively, but their composition through a cut results in a non-terminating process, that is a "proof" for which the cut elimination property does not hold. In other infinitary logics with fixed points like μMALL$^\infty$ [2, 3, 21], this problem is prevented by imposing some *global validity conditions* on derivations/processes aimed at constraining their "intensional" behavior. In particular, these conditions ensure that a process behaving according to a least fixed point must terminate after a fixed number of steps, encoded by the process itself. In sMALL we adopt a different approach: instead of imposing *a posteriori* some global validity conditions on derivations, we let the intensional information about processes emerge at the level of propositions and judgements by extending them with appropriate modalities and annotations. In this way, all sMALL derivations (including the infinite ones) are valid by construction and enjoy cut elimination. These modalities and annotations have been inspired by the line of works on *resource-aware session types* [12, 13, 15, 16], which enrich session types with quantitative information for reasoning about the amount of work required for executing the protocol they describe.

In general, the expressiveness of sMALL is not comparable to that of other related logics: there exist valid infinite derivations in other logics that are not valid sMALL proofs; there are sMALL proofs describing negotiations that do not correspond to valid proofs in other logics. We substantiate the relative expressiveness of sMALL in two ways: (1) we prove that if a process P yields a (productively) terminating system when immersed in every context supposed to be filled with a process of a given type, then there exists a sMALL typing derivation assigning P with that type; (2) we show a simple, conservative extension of sMALL that can serve as foundation for a session type system that ensures the *fair termination* [22, 24] of binary sessions, a weaker form of termination whereby only the *fair* (*i.e.* "realistic" or "reasonable") executions of a process are taken into account.

Structure of the paper. Section 2 presents syntax and semantics of CaPS, along with the termination properties ensured by sMALL. Section 3 presents sMALL as a type system for CaPS. Section 4 states the key soundness properties of sMALL with respect to CaPS and Section 5 establishes a cut elimination result of sMALL that largely builds on the results in Section 4. We present the relative completeness results of sMALL in Section 6. Section 7 discusses a few more examples of session-based systems modeled in CaPS and typed in sMALL. In Section 8 we show how to relax both CaPS and sMALL to enable the description and typing of non-deterministic processes and show that well-typed processes are fairly terminating under a suitable fairness assumption. Section 9 discusses closely related work more in detail and Section 10 concludes outlining some future research directions.

Acknowledgments. We are grateful to the anonymous PPDP'24 reviewers for their comments and suggestions of improvements.

2 CaPS: A Calculus of Pure Sessions

The syntax of CaPS (Table 1) makes use of an infinite set of *channel names* (or simply channels) ranged over by x, y and z and of a set of *tags* ranged over by a, b, ... CaPS is very similar to other session calculi based on classical linear logic such as CP [36] or μCP [30]. The term $x \leftrightarrow y$ denotes a process that links x and y by forwarding each message sent on one channel to the other. There are two terms $x[]$ and $x().P$ to describe the closing of sessions, the first one denoting a process that sends a termination signal on x

Table 1: Syntax of CaPS (infinitary).

P, Q	::=	$x \leftrightarrow y$	(link)
	|	$x[]$	(close)
	|	$x().P$	(wait)
	|	$x[y](P, Q)$	(send)
	|	$x(y).P$	(receive)
	|	$x \triangleleft \mathsf{a}.P$	(select)
	|	$x \triangleright \{\mathsf{a}_i : P_i\}_{i \in I}$	(offer)
	|	$(x)(P \mid Q)$	(cut)

and the second one denoting a process that waits for a termination signal from x and then continues as P. Next are the terms $x[y](P, Q)$ and $x(y).R$ respectively describing the output and input of a fresh channel y on channel x. The sender *forks* into two processes P and Q where P uses one end of y and Q continues using x. The chosen notation highlights the asymmetry in the behavior of this process on the channels x and y. The receiver uses both x and the received end of y in the continuation R. The terms $x \triangleleft \mathsf{a}.P$ and $x \triangleright \{\mathsf{a}_i : Q_i\}_{i \in I}$ respectively denote processes sending and receiving a tag. The sender process elects a single tag a to be sent on x and then continues as P. The receiver process waits for a tag a_k from the set $\{\mathsf{a}_i\}_{i \in I}$ and then continues as P_k. Finally, the term $(x)(P \mid Q)$ denotes the parallel composition of P and Q connected by a fresh channel x. We consider both $x \leftrightarrow y$ and $y \leftrightarrow x$ and $(x)(P \mid Q)$ and $(x)(Q \mid P)$ to be syntactically equal, so that we do not have to explicitly account for the commutativity of links and of parallel compositions. We will often refer to a process of the form $(x)(P \mid Q)$ as a *cut* (on x) and we call *thread* any process that is not a cut. Sometimes we use the term x-thread to emphasize that x is the channel on which the thread is acting first. Note that $x \leftrightarrow y$ is both an x-thread and a y-thread.

Note the lack of any concrete support for recursion. In this respect we depart from other related logics [3, 18, 19, 21, 30] where recursion is expressed with fixed points in propositions and dedicated forms in processes. In sMALL we work directly with possibly infinite propositions. Likewise, in CaPS we work directly with possibly infinite processes generated by the productions in Table 1. To define (infinite) processes we make use of equations of the form $Foo\langle \overline{x} \rangle = P$ where $\overline{x}$ is a list of channel names and P is a process term within which occurrences of $Foo\langle \overline{y} \rangle$ stand for $P\{\overline{y}/\overline{x}\}$, namely a copy of P with suitable substitutions applied. We require these equations to be contractive, in the sense that $Foo\langle \overline{y} \rangle$ can only occur within threads. Finally, there is no explicit form denoting a "failed process". Since sMALL makes use of n-ary (as opposed to binary) additive connectives, the failed process is conveniently represented by the term $x \triangleright \{\,\}$, which offers no continuations.

The notions of free and bound channel names are defined in the expected way, with the proviso that $x[y](P, Q)$ binds y in P but not in Q. We write $\mathsf{fn}(P)$ (respectively, $\mathsf{bn}(P)$) for the sets of free (respectively, bound) channel names occurring in P. For each equation $C\langle \overline{x} \rangle = P$ we also assume that $\mathsf{fn}(P) \subseteq \{\overline{x}\}$.

The operational semantics of CaPS is given in terms of two relations called *reduction* $\longrightarrow$ and *extraction* $\sqsupseteq$, both defined in Table 2. Reductions are standard. The rule [R-LINK] reduces a link by substituting the restricted channel with the linked one. The rules [R-CLOSE], [R-SEND] and [R-SELECT] describe the usual interactions between dual actions. Then, [R-CUT] and [R-EXT] close reductions under parallel compositions and extractions. Extractions pull the topmost action of a y-thread out of a cut on x when $x \neq y$ so as to expose the action and possibly enable interactions with another y-thread in parallel with the cut. Operationally, extractions play the role of structural congruence in standard process calculi, allowing a process to be rearranged so as to place interacting threads next to each other and enable further reductions. From a logical standpoint, reductions correspond to the elimination of a principal cut, whereas extractions push a cut past an introduction rule. Reductions and extractions are respectively called *internal reductions* and *external reductions* in μMALL$^\infty$ [3, 21]. We prefer not to count extractions as proper reductions because in general they do not reduce the complexity of the process. The extraction relation is (implicitly) a precongruence with respect to all language constructs.

Hereafter we write $\Longrightarrow$ for the reflexive, transitive closure of $\longrightarrow$. We write $P \longrightarrow$ if $P \longrightarrow Q$ for some Q. We write $P \not\longrightarrow$ if not $P \longrightarrow$. In this case, we say that P is *irreducible*.

We conclude this section by introducing some standard terminology on the behavior of processes and the notions of termination that we are targeting. A *reduction sequence* of P is a (non-empty) sequence $P_0, P_1, \ldots$ such that $P = P_0$ and $P_i \longrightarrow P_{i+1}$ whenever $i + 1$ is a valid index of the sequence. A *run* of P is a reduction sequence of P that is either infinite or such that the last process in it, say P_n, is irreducible (*i.e.* $P_n \not\longrightarrow$). The *length* of a finite run is the number of processes in it minus one. That is, the length counts the number of reductions in a run rather than the number of processes in it. We say that a run of P is *productive* if it ends with some Q such that $Q \sqsupseteq R$ and R is a thread. We say that a run of P is *successful* if it ends with $x[]$ for some x. Note that productive and successful runs are also finite. We write $P{\downarrow^m}$ if every run of P is not longer than m. In this case we say that P is *terminating*. We write $P{\downarrow\!\!\downarrow^m}$ if every run of P is productive and not longer than m. In this case we say that P is *productively terminating*. We write $P{\Downarrow^m}$ if every run of P is successful and not longer than m. In this case we say that P is *successfully terminating*. Note that ${\Downarrow^m} \subsetneq {\downarrow\!\!\downarrow^m} \subsetneq {\downarrow^m}$. For example, $(x)(x[] \mid y[])$ is terminating but not productively terminating and $(x)(x[] \mid y \triangleright \{\,\})$ is productively terminating but not successfully terminating.

Example 2.1 (buyer-seller). Consider the processes defined by the following equations

$$\begin{aligned} \mathit{Seller}\langle x \rangle &= x \triangleright \{\mathsf{add} : \mathit{Seller}\langle x \rangle, \mathsf{pay} : x[]\} \\ \mathit{Buyer}_0\langle x, y \rangle &= x \triangleleft \mathsf{pay}.x().y[] \\ \mathit{Buyer}_{k+1}\langle x, y \rangle &= x \triangleleft \mathsf{add}.\mathit{Buyer}_k\langle x, y \rangle \\ \mathit{Buyer}_\infty\langle x, y \rangle &= x \triangleleft \mathsf{add}.\mathit{Buyer}_\infty\langle x, y \rangle \end{aligned}$$

modeling an e-commerce scenario. The process $\mathit{Seller}\langle x \rangle$ models an online store offering its clients to add items into a shopping cart and to pay for the purchased items. A process $\mathit{Buyer}_k\langle x, y \rangle$ models a client purchasing k items as it interacts with the seller on session x and then sending a termination signal on y. The process $\mathit{Buyer}_\infty\langle x, y \rangle$ models a client that adds an infinite number of items into the shopping cart and never pays. It is easy to see that $(x)(\mathit{Buyer}_k\langle x, y \rangle \mid \mathit{Seller}\langle x \rangle)$ is (successfully) terminating for every k, whereas $(x)(\mathit{Buyer}_\infty\langle x, y \rangle \mid \mathit{Seller}\langle x \rangle)$ is not terminating. ■

Table 2: Operational semantics of CaPS.

Reductions

[R-LINK]	$(x)(x \leftrightarrow y \mid P) \to P\{y/x\}$	$y \notin \mathsf{fn}(P)$
[R-CLOSE]	$(x)(x[] \mid x().P) \to P$	$x \notin \mathsf{fn}(P)$
[R-SEND]	$(x)(x[y](P,Q) \mid x(y).R) \to (y)(P \mid (x)(Q \mid R))$	
[R-SELECT]	$(x)(x \triangleleft \mathsf{a}_k.P \mid x \triangleright \{\mathsf{a}_i : Q_i\}_{i\in I}) \to (x)(P \mid Q_k)$	$k \in I$
[R-CUT]	$(x)(P \mid R) \to (x)(Q \mid R)$	$P \to Q$
[R-EXT]	$P \to R$	$P \sqsupseteq Q \to R$

Extractions

[E-SEND-L]	$(x)(y[z](P,Q) \mid R) \sqsupseteq y[z](P,(x)(Q \mid R))$	$x \in \mathsf{fn}(Q) \setminus (\mathsf{fn}(P) \cup \{y,z\})$
[E-SEND-R]	$(x)(y[z](P,Q) \mid R) \sqsupseteq y[z]((x)(P \mid R),Q)$	$x \in \mathsf{fn}(P) \setminus (\mathsf{fn}(Q) \cup \{y,z\})$
[E-RECV]	$(x)(y(z).P \mid Q) \sqsupseteq y(z).(x)(P \mid Q)$	$x \notin \{y,z\}, z \notin \mathsf{fn}(Q)$
[E-OFFER]	$(x)(y \triangleright \{\mathsf{a}_i : P_i\}_{i\in I} \mid Q) \sqsupseteq y \triangleright \{\mathsf{a}_i : (x)(P_i \mid Q)\}_{i\in I}$	$x \neq y$
[E-SELECT]	$(x)(y \triangleleft \mathsf{a}.P \mid Q) \sqsupseteq y \triangleleft \mathsf{a}.(x)(P \mid Q)$	$x \neq y$
[E-WAIT]	$(x)(y().P \mid Q) \sqsupseteq y().(x)(P \mid Q)$	$x \neq y$
[E-FAIL]	$(x)(y \triangleright \{\,\} \mid P) \sqsupseteq y \triangleright \{\,\}$	$x \neq y$

Example 2.2 (slot machine). Consider the processes

$$\begin{aligned}
\mathit{Slot}_0\langle x\rangle &= x \triangleright \{\mathsf{play} : x \triangleleft \mathsf{win}.\mathit{Slot}_1\langle x\rangle, \mathsf{stop} : x[]\} \\
\mathit{Slot}_1\langle x\rangle &= x \triangleright \{\mathsf{play} : x \triangleleft \mathsf{lose}.\mathit{Slot}_0\langle x\rangle, \mathsf{stop} : x[]\} \\
\mathit{UnfairSlot}\langle x\rangle &= x \triangleright \{\mathsf{play} : x \triangleleft \mathsf{lose}.\mathit{UnfairSlot}\langle x\rangle, \mathsf{stop} : x[]\} \\
\mathit{Play}\langle x,y\rangle &= x \triangleleft \mathsf{play}.x \triangleright \left\{\begin{array}{l}\mathsf{win} : x \triangleleft \mathsf{stop}.x().y[] \\ \mathsf{lose} : \mathit{Play}\langle x,y\rangle\end{array}\right\}
\end{aligned}$$

modeling the hypothetical behaviors of three slot machines and of one player, connected by a channel x. Each slot machine offers to play an unbounded number of games and the player chooses to play or to stop. After each game, the slot machine notifies the player by sending a tag win or lose. The process Slot_0 (respectively, Slot_1) models a slot machine in which the player wins every even (respectively, odd) game. The process *UnfairSlot* models a slot machine in which the player never wins. The player modeled by *Play* relentlessly plays until one game is won. It is easy to see that $P_i = (x)(\mathit{Play}\langle x,y\rangle \mid \mathit{Slot}_i\langle x\rangle)$ is (successfully) terminating for every $i = 0, 1$. In particular, we have $P_0 \Downarrow^4$ but not $P_0 \Downarrow^3$ and $P_1 \Downarrow^6$ but not $P_1 \Downarrow^5$. On the other hand, $(x)(\mathit{Play}\langle x,y\rangle \mid \mathit{UnfairSlot}\langle x\rangle)$ is not terminating. ∎

3 sMALL: A Type System for CaPS

In this section we present sMALL as a type system for CaPS. We start from the description of the type system directly related to classical linear logic (Section 3.1) and then its extension with features for the amortized analysis of processes (Section 3.2).

3.1 Basic Type System

We let A, B and C range over propositions/types. Types are (possibly infinite) terms generated by the following productions:

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

The constants $\mathbf{1}$ and $\bot$ describe channels used for sending and receiving a termination signal. The types $A \otimes B$ and $A \parr B$ describe channels used for sending and receiving another channel of type A and then continuing according to B. Finally, the types $\oplus\{\mathsf{a}_i : A_i\}_{i\in I}$ and $\&\{\mathsf{a}_i : A_i\}_{i\in I}$ describe channels used for sending and receiving a tag a_k and then continuing according to A_k. We assume that in each occurrence of these types the set I is finite and the tags a_i are pairwise disjoint.

The analogies between types and propositions of multiplicative additive linear logic are obvious, with only two major differences. First, the additive connectives $\oplus$ and $\&$ are n-ary instead of binary and each branch is identified by an annotated tag. Also, the types $\oplus\{\,\}$ and $\&\{\,\}$ (with no branches) play the role of the additive constants $\mathbf{0}$ and $\top$. Second, types may be infinite so as to enable the description of unbounded communication protocols. We define possibly infinite types as solutions of equations $A = B$ where occurrences of the metavariable A in B are guarded by a connective.

The notion of *duality* is defined in the expected (but corecursive) way. In particular, we have:

$$\mathbf{1}^\bot = \bot \qquad \oplus\{\mathsf{a}_i : A_i\}_{i\in I}^\bot = \&\{\mathsf{a}_i : A_i^\bot\}_{i\in I} \qquad (A \otimes B)^\bot = A^\bot \parr B^\bot$$
$$\bot^\bot = \mathbf{1} \qquad \&\{\mathsf{a}_i : A_i\}_{i\in I}^\bot = \oplus\{\mathsf{a}_i : A_i^\bot\}_{i\in I} \qquad (A \parr B)^\bot = A^\bot \otimes B^\bot$$

Typing judgments have the form $P \vdash^l \Gamma$ where P is the process being judged, $l \in \mathbb{N}$ is a *measure* associated with P and Γ is a *typing context* used to track the types of the channels occurring free in P, as usual. Intuitively, the measure of P is an upper bound to the number of reductions that are necessary in order to eliminate all the bottom-most applications of the cut rule in a typing derivation for P. Since this quantity is finite by nature, the derivability of a judgment $P \vdash^l \Gamma$ will entail that P is productively terminating. Hereafter we will use l, m, n to range over natural numbers.

We model typing contexts as partial maps from channel names to types. We let Γ and Δ range over typing contexts, we write $x : A$ for the singleton typing context that associates x with A and we write Γ, Δ for the union of Γ and Δ when they have disjoint domains.

The typing rules for CaPS and the proof rules of sMALL are shown in Table 3. Structurally the typing rules are standard in the sense that they are modeled after the proof rules of MALL and they are found in similar forms in most session type systems based on linear logic. Only two aspects are worth noting. First, the rules are meant to be interpreted *coinductively*, namely $P \vdash^l \Gamma$ is

Table 3: Typing rules for CaPS and proof rules of sMALL.

$$[\text{cut}]\ \frac{P \vdash^{l} \Gamma, x : A \qquad Q \vdash^{m} \Delta, x : A^{\perp}}{(x)(P \mid Q) \vdash^{l+m} \Gamma, \Delta} \qquad [\text{link}]\ \frac{}{x \leftrightarrow y \vdash^{1+l} x : A^{\perp}, y : A}$$

$$[\mathbf{1}]\ \frac{}{x[] \vdash^{1+l} x : \mathbf{1}} \qquad [\perp]\ \frac{P \vdash^{l} \Gamma}{x().P \vdash^{l} \Gamma, x : \perp}$$

$$[\otimes]\ \frac{P \vdash^{l} \Gamma, y : A \qquad Q \vdash^{m} \Delta, x : B}{x[y](P, Q) \vdash^{1+l+m} \Gamma, \Delta, x : A \otimes B} \qquad [\parr]\ \frac{P \vdash^{l} \Gamma, y : A, x : B}{x(y).P \vdash^{l} \Delta, x : A \parr B}$$

$$[\oplus]\ \frac{P \vdash^{l} \Gamma, x : A_k}{x \triangleleft \mathrm{a}_k.P \vdash^{1+l} \Gamma, x : \oplus\{\mathrm{a}_i : A_i\}_{i \in I}}\ k \in I$$

$$[\&]\ \frac{\forall i \in I : P_i \vdash^{l} \Gamma, x : A_i}{x \triangleright \{\mathrm{a}_i : P_i\}_{i \in I} \vdash^{l} \Gamma, x : \&\{\mathrm{a}_i : A_i\}_{i \in I}}$$

derivable if there is an arbitrary (finite or infinite) derivation built using the rules in Table 3. Second, the rules $[\oplus]$ and $[\&]$ are n-ary generalizations of the rules for the binary additive connectives of linear logic. Note that no rule allows the introduction of a type $\oplus\{\,\}$ (as expected, there is no introduction rule for $\mathbf{0}$) and the n-ary form of $[\&]$ covers the standard introduction rule for $\top$ when $I = \emptyset$: we can always derive $x \triangleright \{\,\} \vdash^{l} \Gamma, x : \&\{\,\}$ for every l and Γ.

The key aspect of the type system is the presence of measures in typing judgments. As we have anticipated, the basic idea is that the measure l found in a judgment $P \vdash^{l} \Gamma$ is meant to (over)estimate the number of reductions needed to terminate a (well-typed) process. Since each reduction stems from the interaction between a positive unit/connective ($\mathbf{1}$, $\otimes$, $\oplus$) and its dual ($\perp$, $\parr$, $\&$), the measure only accounts for each introduction of a positive unit/connective. This is the reason why only the rules $[\mathbf{1}]$, $[\otimes]$ and $[\oplus]$ increase the measure in the conclusion, while the rules $[\perp]$, $[\parr]$ and $[\&]$ do not. Let us see how each rule determines the measure in detail, starting from [cut]. The measure of a cut $(x)(P \mid Q)$ combines that of P and Q. This is consistent with the idea that terminating a parallel composition means terminating both components running in parallel. The processes $x \leftrightarrow y$ and $x[]$ can have any strictly positive measure. The 1 unit accounts for their reduction by [r-link] and [r-close], respectively. The fact that the measure can be greater than 1 is consistent with the idea that the measure may overestimate the actual effort needed to terminate a process. This flexibility can be useful in typing some processes, as we will see in Example 3.3. The measure of processes of the form $x().P$ or $x(y).P$ in inherited from the continuation P. Since these processes are threads, and therefore they are terminated, one could expect their measure to be arbitrary. However, the continuation P may require further reductions if the process is put in appropriate contexts. For this reason, the measure cannot be lower than that of the continuation P. The measure of a process $x[y](P, Q)$ is $1 + l + m$ where l and m are the measures of P and Q respectively and the 1 term accounts for the [r-send] reduction. The measure of a process $x \triangleleft \mathrm{a}.P$ is 1 plus the measure of the continuation of P where, again, the 1 term accounts for the reduction [r-select]. Finally, the measure of a process of the form $x \triangleright \{\mathrm{a}_i : P_i\}_{i \in I}$ coincides with that of its branches. Note that $x \triangleright \{\,\}$ can be given any measure.

Example 3.1. In order to find typing derivations for *Seller* and $Buyer_k$ in Example 2.1, consider the type A that satisfies the equation $A = \oplus\{\mathrm{add} : A, \mathrm{pay} : \perp\}$. We show that $Buyer_k\langle x, y\rangle$ is well typed by induction on k. In the base case we derive

$$\dfrac{\dfrac{\dfrac{}{y[] \vdash^{1} y : \mathbf{1}}\,[\mathbf{1}]}{x().y[] \vdash^{1} x : \perp, y : \mathbf{1}}\,[\perp]}{Buyer_0\langle x, y\rangle \vdash^{2} x : A, y : \mathbf{1}}\,[\oplus]$$

and in the inductive case we derive

$$\dfrac{\dfrac{\vdots}{Buyer_k\langle x, y\rangle \vdash^{k+2} x : A, y : \mathbf{1}}}{Buyer_{k+1}\langle x, y\rangle \vdash^{k+3} x : A, y : \mathbf{1}}\,[\oplus]$$

Concerning *Seller* we derive

$$\dfrac{\dfrac{\vdots}{Seller\langle x\rangle \vdash^{1} x : A^{\perp}} \qquad \dfrac{}{x[] \vdash^{1} x : \mathbf{1}}\,[\mathbf{1}]}{Seller\langle x\rangle \vdash^{1} x : A^{\perp}}\,[\&]$$

therefore we have $(x)(Buyer_k\langle x, y\rangle \mid Seller\langle x\rangle) \vdash^{k+3} y : \mathbf{1}$ for every k. Note that there is no k such that $Buyer_\infty\langle x, y\rangle \vdash^{k} x : A$ is derivable, which is consistent with the observation made in Example 2.1 that $(x)(Buyer_\infty\langle x, y\rangle \mid Seller\langle x\rangle)$ is not terminating. ■

It should be quite obvious that none of the processes $Slot_n$ and *Play* in Example 2.2 is well typed. According to the typing rules in Table 3, the measure is monotonically nondecreasing when reading the rules from top to bottom. Moreover, each output action strictly increases the measure. Since every infinite branch in both $Slot_n$ and *Play* goes through infinitely many output actions, it is clear that neither process can be given a finite measure. This is a serious limitation of the type system as it stands. Hence, we now introduce a mechanism to make measure management more flexible.

3.2 Amortized Measure Analysis

If we look carefully at the typing rules in Table 3, it is quite obvious that the lack of accuracy in estimating the number of reductions needed to terminate a process is mainly due to the rule $[\&]$, requiring every branch P_i of a process $x \triangleright \{\mathrm{a}_i : P_i\}_{i \in I}$ to be given the same measure l. The reason for this formulation of the rule is that we do not know which tag a_i is going to be received from the channel x. Without this information, the best we can do to compute the number of reductions needed to terminate the process is to pretend that every continuation P_i has the same, possibly overapproximated measure.

The basic idea to compute measures in a more precise way is to realize some form of *amortized measure analysis*, by allowing some dependency between the received tag a_i and the measure of the corresponding branch P_i. We do so following some recent works

on RAST [12, 13, 15, 16], and augmenting processes and types with constructs that allow the transfer of measure within a session. These constructs are not meant to alter the semantics of processes or the structure of the protocols they implement. Rather, they should be merely considered as a form of code and type instrumentation that improves the accuracy of the measure analysis performed by the type system. At the level of processes we introduce two prefixes $\blacktriangleleft x$ and $\blacktriangleright x$ for respectively *sending* and *receiving* some measure through the session x

$$P, Q ::= \cdots \mid \blacktriangleleft x.P \mid \blacktriangleright x.P$$

and, at the level of types, we introduce two corresponding modalities $\blacktriangleleft^n$ and $\blacktriangleright^n$ describing the effect of these prefixes:

$$A, B ::= \cdots \mid \blacktriangleleft^n A \mid \blacktriangleright^n A$$

The two modalities are meant to be dual of one another. That is, $(\blacktriangleleft^n A)^\perp = \blacktriangleright^n A^\perp$ and $(\blacktriangleright^n A)^\perp = \blacktriangleleft^n A^\perp$. Note that the amount of transferred measure is not affected by duality. This is important to ensure that the overall measure of a session is not altered despite such transfers. Unlike the works on RAST where $\blacktriangleleft x.P$ and $\blacktriangleright x.P$ have been introduced, we do not specify the amount of transferred measure in the processes, since such amount is not operationally relevant in our case. This is clear by looking at the additional reduction and structural precongruence rules that concern these prefixes:

$$\begin{array}{lll}
[\text{E-PUT}] & (x)(\blacktriangleleft y.P \mid Q) \sqsupseteq \blacktriangleleft y.(x)(P \mid Q) & x \neq y \\
[\text{E-GET}] & (x)(\blacktriangleright y.P \mid Q) \sqsupseteq \blacktriangleright y.(x)(P \mid Q) & x \neq y \\
[\text{R-PUT}] & (x)(\blacktriangleleft x.P \mid \blacktriangleright x.Q) \to (x)(P \mid Q) &
\end{array}$$

The rules [E-PUT] and [E-GET] respectively pull $\blacktriangleleft x$ and $\blacktriangleright x$ prefixes outside of unrelated cuts in order to enable further synchronizations. The new [R-PUT] reduction expresses a mere synchronization between two processes without any actual exchange of meaningful information. In fact, we could have dispensed CaPS from these forms and simply augmented types and the typing rules with the measure transfer modalities. However, having explicit process forms that correspond to such modalities simplifies the proofs because the type system remains syntax directed.

Below are the typing rules for the newly introduced measure transfer forms in processes and modalities in types:

$$[\text{PUT}]\ \frac{P \vdash^l \Gamma, x : A}{\blacktriangleleft x.P \vdash^{1+l+n} \Gamma, x : \blacktriangleleft^n A} \qquad [\text{GET}]\ \frac{P \vdash^{l+n} \Gamma, x : A}{\blacktriangleright x.P \vdash^l \Gamma, x : \blacktriangleright^n A}$$

These rules formalize the effect of transferring some amount n of measure through a session x: [PUT] charges that amount on the sender, while [GET] discharges the same amount from the continuation of the receiver. We make the assumption that $\blacktriangleleft^n$ is a positive connective and therefore we charge an additional unit of measure on the sender to account for the [R-PUT] reduction.

Example 3.2. To illustrate the kind of amortized analysis enabled by the machinery introduced in this section, consider the processes

$$P = x \triangleleft \mathsf{a}.\blacktriangleleft x.x[] \qquad Q = x \triangleright \{\mathsf{a} : \blacktriangleright x.x().y[], \mathsf{b} : \blacktriangleright x.R\}$$

where R is some well-typed process with a large measure $n \gg 4$, that is $R \vdash^n x : A, y : \mathbf{1}$. Note that P selects the a-tagged branch of Q which quickly leads to termination without requiring the execution of R. For the process P we derive

$$\dfrac{\dfrac{\dfrac{}{x[] \vdash^1 x : \mathbf{1}}\,[1]}{\blacktriangleleft x.x[] \vdash^3 x : \blacktriangleleft^1 \mathbf{1}}\,[\text{PUT}]}{x \triangleleft \mathsf{a}.\blacktriangleleft x.x[] \vdash^4 x : \oplus\{\mathsf{a} : \blacktriangleleft^1 \mathbf{1}, \mathsf{b} : \blacktriangleleft^n A^\perp\}}\,[\oplus]$$

and for the process Q we derive

$$\dfrac{\dfrac{\dfrac{\dfrac{}{y[] \vdash^1 y : \mathbf{1}}\,[1]}{x().y[] \vdash^1 x : \perp, y : \mathbf{1}}\,[\perp]}{\blacktriangleright x.x().y[] \vdash^0 x : \blacktriangleright^1 \perp, y : \mathbf{1}}\,[\text{GET}] \qquad \dfrac{\dfrac{\vdots}{R \vdash^n x : A^\perp, y : \mathbf{1}}}{\blacktriangleright x.R \vdash^0 x : \blacktriangleright^n A^\perp, y : \mathbf{1}}\,[\text{GET}]}{x \triangleright \{\mathsf{a} : \blacktriangleright x.x().y[], \mathsf{b} : \blacktriangleright x.R\} \vdash^0 x : \&\{\mathsf{a} : \blacktriangleright^1 \perp, \mathsf{b} : \blacktriangleright^n A^\perp\}, y : \mathbf{1}}\,[\&]$$

We have been able to assign Q a null measure thanks to the $\blacktriangleright x$ prefixes in its body. Basically, the actual measure of Q is "received" from x *after* the process at the other end of the session x has selected the desired branch to be taken. In this way, the amount of received measure can match exactly what is needed for typing the selected continuation of Q, namely 1 in the case of the a-tagged branch and n in the case of the b-tagged branch. Since P selects the a-tagged branch of Q, the amount of measure discharged from Q and charged on P is just 1, as witnessed by the typing derivation for P, and the overall measure of the composition $(x)(P \mid Q)$ is 4 which is much smaller than n by assumption. ■

Example 3.3. Let us consider again the processes *Play* and $Slot_i$ defined in Example 2.2. In order to find a typing derivation for them we consider the following instrumented versions, where we have inserted appropriate $\blacktriangleleft x$ and $\blacktriangleright x$ operations to transfer measure depending on the actual branches selected by the processes:

$$\begin{array}{rcl}
Slot_0\langle x\rangle & = & x \triangleright \{\mathsf{play} : \blacktriangleright x.x \triangleleft \mathsf{win}.Slot_1\langle x\rangle, \mathsf{stop} : x[]\} \\
Slot_1\langle x\rangle & = & x \triangleright \{\mathsf{play} : \blacktriangleright x.x \triangleleft \mathsf{lose}.\blacktriangleleft x.Slot_0\langle x\rangle, \mathsf{stop} : x[]\} \\
Play\langle x, y\rangle & = & x \triangleleft \mathsf{play}.\blacktriangleleft x.x \triangleright \left\{\begin{array}{l}\mathsf{win} : x \triangleleft \mathsf{stop}.x().y[] \\ \mathsf{lose} : \blacktriangleright x.Play\langle x, y\rangle\end{array}\right\}
\end{array}$$

We also consider the types A and B that satisfy the equations

$$A = \oplus\{\mathsf{play} : \blacktriangleleft^5 B, \mathsf{stop} : \perp\} \quad \text{and} \quad B = \&\{\mathsf{win} : A, \mathsf{lose} : \blacktriangleright^7 A\}$$

Now we derive

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{y[] \vdash^1 y : \mathbf{1}}\,[1]}{x().y[] \vdash^1 x : \perp, y : \mathbf{1}}\,[\perp]}{x \triangleleft \mathsf{stop} \ldots \vdash^2 x : A, y : \mathbf{1}}\,[\oplus] \qquad \dfrac{\vdots}{Play\langle x, y\rangle \vdash^9 x : A, y : \mathbf{1}}}{\blacktriangleright x \ldots \vdash^2 x : \blacktriangleright^7 A, y : \mathbf{1}}\,[\text{GET}]}{x \triangleright \{\mathsf{win} : \ldots, \mathsf{lose} : \ldots\} \vdash^2 x : B, y : \mathbf{1}}\,[\&]$$
$$\dfrac{\blacktriangleleft x.x \triangleright \{\mathsf{win} : \ldots, \mathsf{lose} : \ldots\} \vdash^8 x : \blacktriangleleft^5 B, y : \mathbf{1}}{Play\langle x, y\rangle \vdash^9 x : A, y : \mathbf{1}}\,[\oplus]\ [\text{PUT}]$$

as well as

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{Slot_1\langle x\rangle \vdash^{5} x : A^{\perp}}}{x \triangleleft \mathsf{win}.Slot_1\langle x\rangle \vdash^{6} x : B^{\perp}}\,[\oplus]}{\blacktriangleright x.x \triangleleft \mathsf{win}.Slot_1\langle x\rangle \vdash^{1} x : \blacktriangleright^{5} B^{\perp}}\,[\textsc{get}] \quad \dfrac{}{x[] \vdash^{1} x : \mathbf{1}}\,[\mathbf{1}]}{Slot_0\langle x\rangle \vdash^{1} x : A^{\perp}}\,[\&]}{\blacktriangleleft x.Slot_0\langle x\rangle \vdash^{9} x : \blacktriangleleft^{7} A^{\perp}}\,[\textsc{put}]}{x \triangleleft \mathsf{lose}.\blacktriangleleft x.Slot_0\langle x\rangle \vdash^{10} x : B^{\perp}}\,[\oplus]}{\blacktriangleright x.x \triangleleft \mathsf{lose}.\blacktriangleleft x.Slot_0\langle x\rangle \vdash^{5} x : \blacktriangleright^{5} B^{\perp}}\,[\textsc{get}] \quad \dfrac{}{x[] \vdash^{5} x : \mathbf{1}}\,[\mathbf{1}]}{Slot_1\langle x\rangle \vdash^{5} x : A^{\perp}}\,[\&]$$

therefore we conclude $(x)(Play\langle x, y\rangle \mid Slot_1\langle x\rangle) \vdash^{14} y : \mathbf{1}$ with one application of [cut].

It is far from obvious that the measure annotations in A and B are the "right ones" in order to obtain a well-typed composition of *Play* and $Slot_i$. However, we can make sense of those annotations by accounting carefully the "cost" in terms of measure of performing the actions that occur in types. For instance, we know that a slot machine implementing the protocol $A^{\perp}$ cannot send lose tags forever: performing each such action charges 9 units on the measure of the slot machine in contrast to only 5 units gained when it receives play. Insisting on sending lose tags would eventually exhaust the slot machine's balance. This is also the reason why a process like *Play* is well typed: we know that sooner or later *Play* will receive the tag win, thus reaching successful termination. ■

The instrumentation of the processes in Example 2.2 into those in Example 3.3 may seem like wizardry at first. In fact, there is not a unique way in which a process can be instrumented in order to make it well typed. However, recalling the role of the modalities $\blacktriangleleft^n$ and $\blacktriangleright^n$ in enabling the amortized measure analysis of a process, there is an effective rule of thumb that works well in a large number of cases: insert $\blacktriangleleft x$ *right after* a selection $x \triangleleft \mathsf{a}$ and insert $\blacktriangleright x$ *right after every case* $x \triangleright \{\mathsf{a}_i : P_i\}_{i\in I}$. This strategy can be improved in a few ways: there is usually no need to instrument finite processes; when the amount of transferred measure turns out to be 0 the modalities can be omitted altogether. Another family of (infinite) processes that may require the careful placement of instrumentation in order to be finitely measurable are those modeling infinite streams. We will discuss an instance of this case in Section 7.1.

4 Soundness

Here we present the soundness properties of the type system. We start with two standard results stating that extractions and reductions preserve typing. For extractions we have:

Lemma 4.1. *If* $P \vdash^{l} \Gamma$ *and* $P \sqsupseteq Q$, *then* $Q \vdash^{l} \Gamma$.

Note that extractions preserve not only typing, but also the measure. This is somewhat expected since the measure only accounts for the number of actions that need to be performed and extractions preserve this number except for the relation $(x)(y \triangleright \{\,\} \mid P) \sqsupseteq y \triangleright \{\,\}$. However, the rule [&] is formulated in such a way that a process $y \triangleright \{\,\}$ can be given any measure, therefore the measure is preserved also in this case.

Next is typing preservation by reductions:

Theorem 4.2. *If* $P \vdash^{l} \Gamma$ *and* $P \to Q$, *then* $Q \vdash^{m} \Gamma$ *and* $m < l$.

This result clearly shows that the measure of a process decreases after each reduction. Then, the fact that every well-typed processes is terminating follows as a simple corollary.

Corollary 4.3. *If* $P \vdash^{l} \Gamma$, *then* $P{\downarrow^{l}}$.

Corollary 4.3 assures us that if we keep reducing a well-typed process P we eventually reach an irreducible one $Q \not\to$. In principle, Q might be irredubile because it is somewhat ill formed (for example, Q could model a deadlocked network configuration like $(x)(x[] \mid y[])$). The next result shows that this is not the case.

Theorem 4.4. *If* $P \vdash^{l} \Gamma$, *then* $P{\downarrow^{l}}$.

That is, once P has reached an irreducible process, it always exposes some observable behavior in the form of a thread. This property is key both for the successful termination result below and also for the cut elimination property that we discuss in Section 5.

Corollary 4.5. *If* $P \vdash^{l} x : \mathbf{1}$, *then* $P{\Downarrow^{l}}$.

5 Cut Elimination

Cut elimination means that the rule [cut] is admissible: every sequent that can be proved with applications of [cut] can be proved also without [cut]. In the case of sMALL, a sequent is just a typing context. It is easy to see that sMALL enjoys cut elimination using the results of Section 4. Of course, since sMALL derivations may be infinite, there are cases in which all cuts are eliminated only "in the limit", that is after an infinite number of (cut) reductions. However, we can argue that each layer of the cut-free derivation can be obtained in finite time. Then, a corecursive application of the same argument allows us to build the whole cut-free derivation.

Let us describe the cut elimination procedure more in detail. We start by defining a function C that computes the set of continuations of a process. The function is defined by the equations

$$\begin{aligned} C(x().P) = C(x(y).P) = C(x \triangleleft \mathsf{a}.P) &= \{P\} \\ C(\blacktriangleleft x.P) = C(\blacktriangleright x.P) &= \{P\} \\ C(x[y](P, Q)) &= \{P, Q\} \\ C(x \triangleright \{\mathsf{a}_i : P_i\}_{i\in I}) &= \{P_i\}_{i\in I} \end{aligned}$$

and $C(P) = \emptyset$ for any other form P.

Now suppose that $P \vdash^{l} \Gamma$ is derivable. In order to find a cut-free derivation for the sequent Γ, we reduce P as much as we can until we reach an irreducible process Q such that $P \Rightarrow Q \not\to$. We know that such Q exists from Corollary 4.3 and we know that $Q \vdash^{m} \Gamma$ with $m \leq l$ is also derivable from Theorem 4.2. Moreover, Theorem 4.4 and Lemma 4.1 assure us that $R \vdash^{m} \Gamma$ is also derivable for some thread R. Thus, we have found a derivation for the sequent Γ that necessarily ends with the application of a rule other than [cut]. In other words, we have built the first layer of the cut-free proof of Γ. In order to iterate this argument and obtain more layers of the cut-free proof of Γ, the only technical difficulty may arise from the fact that the judgment $R \vdash^{m} \Gamma$ may have a measure (m) that is strictly smaller than (hence different from) the one (l) in the original judgment $P \vdash^{l} \Gamma$. Conveniently, we can always *increase* the measure of a judgment without altering the structure of the proof.

Lemma 5.1. *If* $P \vdash^{l} \Gamma$ *and* $l \leq m$, *then* $P \vdash^{m} \Gamma$.

Now let $C(R) = \{P_1, \dots, P_n\}$ where $P_1, \dots, P_n$ are all the continuations of R. For each continuation there is a premise $P_i \vdash^{l_i} \Gamma_i$ above the line of the rule that derives $R \vdash^m \Gamma$. We iterate the same procedure for each continuation P_i, where each iteration builds one more layer of the derivation tree for the original sequent Γ.

Example 5.2. Consider the type $A = \&\{\mathsf{a} : \blacktriangleright^1 A, \mathsf{b} : \blacktriangleright^1 \mathbf{1}\}$ and the processes $Foo\langle x\rangle = x \triangleright \{\mathsf{a} : \blacktriangleright x.(y)(y[] \mid y().Foo\langle x\rangle), \mathsf{b} : \blacktriangleright x.x[]\}$ and $Bar\langle x\rangle = x \triangleright \{\mathsf{a} : \blacktriangleright x.Foo\langle x\rangle, \mathsf{b} : \blacktriangleright x.x[]\}$. We derive

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{y[] \vdash^1 y : \mathbf{1}}\,[\mathbf{1}] \quad \dfrac{\dfrac{\vdots}{Foo\langle x\rangle \vdash^0 x : A}}{y() \ldots \vdash^0 x : A, y : \bot}\,[\bot]}{(y)(y[] \mid y().Foo\langle x\rangle) \vdash^1 x : A}\,[\textsc{cut}]}{\blacktriangleright x.(y)(y[] \mid y().Foo\langle x\rangle) \vdash^0 x : \blacktriangleright^1 A}\,[\textsc{get}] \quad \dfrac{\dfrac{\vdots}{x[] \vdash^1 x : \mathbf{1}}\,[\mathbf{1}]}{\blacktriangleright x.x[] \vdash^0 x : \blacktriangleright^1 \mathbf{1}}\,[\textsc{get}]}{Foo\langle x\rangle \vdash^0 x : A}\,[\&]$$

which contains infinitely many cuts and is cut free up to depth 2. At the bottom of the derivation we find a thread with two continuations $\blacktriangleright x.(y)(y[] \mid y().Foo\langle x\rangle)$ and $\blacktriangleright x.x[]$, where the second one is cut-free and the first one is a thread with one continuation derived with a [cut]. By applying the procedure outlined above we obtain

$$\dfrac{\dfrac{\dfrac{\vdots}{Foo\langle x\rangle \vdash^1 x : A}}{\blacktriangleright x.Foo\langle x\rangle \vdash^0 x : \blacktriangleright^1 A}\,[\textsc{get}] \quad \dfrac{\dfrac{\vdots}{x[] \vdash^1 x : \mathbf{1}}\,[\mathbf{1}]}{\blacktriangleright x.x[] \vdash^0 x : \blacktriangleright^1 \mathbf{1}}\,[\textsc{get}]}{Bar\langle x\rangle \vdash^0 x : A}\,[\&]$$

which is cut-free up to depth 4. Note that it is necessary to use Lemma 5.1 to turn the derivation of $Foo\langle x\rangle \vdash^0 x : A$ into a derivation of $Foo\langle x\rangle \vdash^1 x : A$ to be able to apply the rule [get]. ∎

6 A relative completeness result

In this section we provide a relative completeness result for sMALL. More precisely, we characterize the typeability of processes in CaPS in terms of their operational behavior using the notion of productive termination as reference property. In other words, we provide sufficient conditions to invert a suitable generalization of Theorem 4.4. This result is quite challenging because, in order to build a typing derivation for $P \vdash^l \Gamma$, it is not enough to observe the (essentially unique) reduction that leads P to termination, but rather, we must take into account all the contexts in which P can be used and, in doing so, we must be able to infer properties of l and Γ solely by performing *observations* on the behavior of P in such contexts. For these reasons, we prove a completeness result under the following assumptions and leave its generalization to future work.

- We assume that P is cut-free and link-free.
- We only consider types built using $\mathbf{1}$, $\bot$, the additive connectives $\oplus$ and $\&$ and modalities $\blacktriangleleft^l$ and $\blacktriangleright^l$.
- We assume that Γ is *inhabited* in a sense that will be made precise shortly. This key assumption allows us to synthetize a suitable set of *tests* to observe the behavior of P.

Let us introduce some terminology that helps us defining the notion of inhabited context.

Definition 6.1. We say that A is a *server type* (and that $A^\bot$ is a *client type*) if there exist P, Q, l and m such that $P \vdash^l x : A$ and $Q \vdash^m x : A^\bot, y : \mathbf{1}$. We say that Γ is *server context* if $\Gamma = x : A, y_1 : B_1, \dots, y_k : B_k, z_1 : \top, \dots, z_h : \top$ where A is a server type and B_i is a client type for $1 \le i \le k$. We say that Γ is $\top$*-free* if $h = 0$.

Intuitively, a client/server type is a type that corresponds to a protocol according to which at least two processes can communicate. From a logical standpoint, a server type is *logically equivalent* to $\mathbf{1}$. Indeed, from the definition it follows immediately that the sequents $\vdash \bot, A$ and $\vdash A^\bot, \mathbf{1}$ are provable. Note that $\top$ and $\mathbf{0}$ are neither server nor client types. We say that P is a *server* if there exists a derivation for $P \vdash^l \Gamma$ in which every context (including Γ) is a server context. We can prove that server contexts are inhabited.

Proposition 6.2. *If Γ is a server context, then there exist l and a server P such that $P \vdash^l \Gamma$.*

We now formalize the notion of *observation* that we perform on a process P to build a derivation for $P \vdash^l \Gamma$. We start by introducing a refined notion of reduction context, where the hole is decorated with the expected measure (l) and typing context (Γ) of the process being observed (P). Also, we require that other processes occurring in the reduction context are servers. A $(l;\Gamma)$*-context* is generated by the following grammar

$$C ::= [\,]^l_\Gamma \mid (x)(P \mid C)$$

where Γ is a $\top$-free server context and P is a server. We write $C \Vdash^m \Delta$ when the $(l;\Gamma)$-context C is well typed in Δ and has measure m according to the following inductive rules:

$$\dfrac{}{[\,]^l_\Gamma \Vdash^l \Gamma}\,[\textsc{c-hole}] \qquad \dfrac{P \vdash^l \Gamma, x : A \qquad C \Vdash^m \Delta, x : A^\bot}{(x)(P \mid C) \Vdash^{l+m} \Gamma, \Delta}\,[\textsc{c-cut}]$$

We say that a $(l;\Gamma)$-context C is an $(m;l;\Gamma)$*-test* if $C \Vdash^m \Delta$ for some Δ and $\mathrm{dom}(\Gamma) \subseteq \mathrm{bn}(C)$. Intuitively, a $(m;l;\Gamma)$-test provides a way to observe the behavior of a process P through all channels in Γ using well-behaved processes, *i.e.* well-typed servers. We are now ready to state our relative completeness result with respect to productive termination. Recall that $P{\downarrow^m}$ means that P is productively terminating in at most m reductions (*cf.* Section 2).

Theorem 6.3. *Let P be a cut-free and link-free process. If $C[P]{\downarrow^m}$ for every $(m;l;\Gamma)$-test C, then $P \vdash^l \Gamma$.*

Note that the inverse of this result is an immediate consequence of Theorem 4.4, since when $P \vdash^l \Gamma$ holds, we can easily derive $C[P] \vdash^m \Delta$ for every $(m;l;\Gamma)$-test C. Therefore, Theorem 6.3 says that a cut-free and link-free process P is well typed in a context Γ with measure l if and only if $C[P]$ is productively terminating in at most m steps for all $(m;l;\Gamma)$-tests C. FD questo vale sotto le assunzioni

Remark 6.4. Theorem 6.3 holds under several assumptions. We believe that including the multiplicative connectives is doable, even though we have not worked out all of the details yet, whereas including cuts and axioms seems to be more challenging. For example, the theorem as it stands does not hold for the axiom $x \leftrightarrow y$: it is easy to see that $C[x \leftrightarrow y]{\downarrow^m}$ holds for every $(m;1;\Gamma)$-test C, where $\Gamma = x : \oplus\{\mathsf{a} : \mathbf{1}, \mathsf{b} : \mathbf{1}\}, y : \&\{\mathsf{a} : \bot\}$, but clearly $x \leftrightarrow y \vdash^1 \Gamma$ is not derivable, as the types in the context are not dual to each other. This suggests that extending Theorem 6.3 may require a relaxation of

sMALL to incorporate some form of subtyping, whose development is an interesting direction *per sé*.

7 Examples in sMALL CaPS

7.1 Stream Processing

In this section we illustrate an example of stream processing, which is a paradigmatic application domain for corecursive types. In particular, we aim at defining a (well-typed) process that negates the (infinitely many) elements of a stream of boolean values.

We start by encoding the type of boolean values as $\mathsf{Bool} = \oplus\{\mathsf{ff} : \mathbf{1}, \mathsf{tt} : \mathbf{1}\}$. This type describes a process that sends either ff or tt on a channel and then closes the channel. For this reason, Bool is the type of a boolean value "being produced" and its dual $\mathsf{Bool}^\perp$ describes a boolean value "being consumed".

Now, the process $Neg\langle x, y\rangle$ defined by the equation

$$Neg\langle x, y\rangle = x \triangleright \begin{Bmatrix} \mathsf{ff} : y \triangleleft \mathsf{tt}.x().y[] \\ \mathsf{tt} : y \triangleleft \mathsf{ff}.x().y[] \end{Bmatrix}$$

consumes a boolean value from x and produces its negation on y. It is easy to come up with a typing derivation for the judgment $Neg\langle x, y\rangle \vdash^2 x : \mathsf{Bool}^\perp, y : \mathsf{Bool}$ showing that $Neg\langle x, y\rangle$ is well typed. The measure 2 accounts for the messages sent on channel y.

We now model a stream of boolean values. This is an entity providing infinitely many boolean values, thus it is naturally modelled by the type $S = \mathsf{Bool} \otimes S$. However, any non-trivial process P adhering to S would have an infinite measure. This is reasonable, because its dual $S^\perp$ is inhabited by a process Q with 0 measure and so the composition $(x)(P \mid Q)$ would be non-terminating.

In order to turn S into a type that is inhabited by an interesting process P, the idea is that P may charge the cost of producing each element on the consumer of the stream. Therefore we consider the type of streams $\mathsf{BoolS}_n = \blacktriangleright^n(\mathsf{Bool} \otimes \mathsf{BoolS}_n)$ where each produced element of the stream requires a transfer of n units of measure from the consumer.It is now easy to find well-typed processes that produce streams of type BoolS_n. For instance we have $Trues\langle x\rangle \vdash^0 x : \mathsf{BoolS}_3$ where $Trues\langle x\rangle = \blacktriangleright x.x[y](y \triangleleft \mathsf{tt}.y[], Trues\langle x\rangle)$ is the stream providing infinitely many tts. Now we can define the process $NegS\langle x, y\rangle$ that negates a stream of boolean values as follows

$$NegS\langle x, y\rangle = \blacktriangleright y.\blacktriangleleft x.x(u).y[v](Neg\langle u, v\rangle, NegS\langle x, y\rangle)$$

and it is a simple exercise to find a typing derivation for $NegS\langle x, y\rangle \vdash^0 x : \mathsf{BoolS}_n^\perp, y : \mathsf{BoolS}_{n+4}$ showing that $NegS\langle x, y\rangle$ is well typed and has null measure. Note that the output stream y requires 4 more units of measure to produce each element compared to input stream x. This additional measure accounts for the cost required to produce each single element of the stream.

It is interesting to observe that the problem discussed above in finding a non-trivial inhabitant for the tentative stream type S now resurfaces for the type $\mathsf{BoolS}_n^\perp = \blacktriangleleft^n(\mathsf{Bool}^\perp \parr \mathsf{BoolS}_n^\perp)$. Every non-trivial process adhering to this type has an infinite measure. This makes sense: if we were able to finitely measure a "pure stream consumer" (a process that consumes a stream without producing another one), then we would also be able to obtain a non-terminating composition by cutting this process against *e.g.* $Trues\langle x\rangle$.

7.2 Consensus

In this example we model a simple distributed system in which an arbiter process $Ar\langle x, y, z\rangle$ orchestrates a consensus protocol between two agents which try to agree on some value (either a or b). The arbiter is modeled thus:

$$Ar\langle x, y, z\rangle = x \triangleright \left\{ \begin{array}{l} \mathsf{a} : \blacktriangleright x.y \triangleright \begin{Bmatrix} \mathsf{a} : \blacktriangleright y.x \triangleleft \mathsf{tt}.y \triangleleft \mathsf{tt}.x().y().z[] \\ \mathsf{b} : \blacktriangleright y.x \triangleleft \mathsf{ff}.\blacktriangleleft x.y \triangleleft \mathsf{ff}.\blacktriangleleft y.Ar\langle x, y, z\rangle \end{Bmatrix} \\ \mathsf{b} : \blacktriangleright x.y \triangleright \begin{Bmatrix} \mathsf{a} : \blacktriangleright y.x \triangleleft \mathsf{ff}.\blacktriangleleft x.y \triangleleft \mathsf{ff}.\blacktriangleleft y.Ar\langle x, y, z\rangle \\ \mathsf{b} : \blacktriangleright y.x \triangleleft \mathsf{tt}.y \triangleleft \mathsf{tt}.x().y().z[] \end{Bmatrix} \end{array} \right\}$$

The two agents are connected to the arbiter by means of the channels x and y and they repeatedly propose a value to the arbiter. In doing so, they also transfer a certain amount of measure to allow the arbiter to work. The arbiter compares the two proposed values: if they coincide then consensus is reached, the arbiter notifies the agents with tt and it signals termination on z; if the proposed values differ, then the arbiter notifies the agents with ff and the protocol is iterated. Note that in the latter case the arbiter also refunds the agents with some measure, so that the protocol can continue.

For this example we consider the agents $Voter_\mathsf{a}$ and $Voter_\mathsf{b}$ below such that $Voter_\mathsf{a}$ always sends a and $Voter_\mathsf{b}$ sends b once before behaving as $Voter_\mathsf{a}$:

$$\begin{aligned} Voter_\mathsf{a}\langle x\rangle &= x \triangleleft \mathsf{a}.\blacktriangleleft x.Poller\langle x\rangle \\ Voter_\mathsf{b}\langle y\rangle &= y \triangleleft \mathsf{b}.\blacktriangleleft y.Poller\langle y\rangle \\ Poller\langle z\rangle &= z \triangleright \{\mathsf{tt} : z[], \mathsf{ff} : \blacktriangleright z.Voter_\mathsf{a}\langle z\rangle\} \end{aligned}$$

It is easy to see that the composition of $Ar\langle x, y, z\rangle$ with $Voter_\mathsf{a}\langle x\rangle$ and $Voter_\mathsf{b}\langle y\rangle$ successfully terminates. In order to come up with a typing derivation for this composition we have to find the appropriate amount of measure to be exchanged between the arbiter and the agents. Consider for instance the type $A = \oplus\{\mathsf{a} : \blacktriangleleft^0 B, \mathsf{b} : \blacktriangleleft^8 B\}$ where $B = \&\{\mathsf{tt} : \mathbf{1}, \mathsf{ff} : \blacktriangleright^2 A\}$ which describes the behavior of an agent interacting with the arbiter. By looking at the annotations within A we can argue that A describes an agent that cannot vote a infinitely many times since the loop within A that goes through the output of b is not sustainable forever in the economy of measures: each vote b "costs" 10 units of measure against a "gain" of just 2 units if consensus is not reached. On the contrary, the loop that goes through the output of a is perfectly balanced: each vote a costs 2 units, which are completely "refunded" in case consensus is not reached. Indeed, we obtain the derivation

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{x[] \vdash^1 x : \mathbf{1}}\,[1] \quad \dfrac{\dfrac{\vdots}{Voter_\mathsf{a}\langle x\rangle \vdash^3 x : A}\,[\oplus]}{\blacktriangleright x.Voter_\mathsf{a}\langle x\rangle \vdash^1 x : \blacktriangleright^2 A}\,[\textsc{get}]}{Poller\langle x\rangle \vdash^1 x : B}\,[\&]}{\blacktriangleleft x.Poller \vdash^2 x : \blacktriangleleft^0 B}\,[\textsc{put}]}{Voter_\mathsf{a}\langle x\rangle \vdash^3 x : A}\,[\oplus]$$

as well as the derivation

$$\dfrac{\dfrac{\dfrac{\vdots}{Poller\langle y\rangle \vdash^1 y : B}\,[\&]}{\blacktriangleleft y.Poller\langle y\rangle \vdash^{10} y : \blacktriangleleft^8 B}\,[\textsc{put}]}{Voter_\mathsf{b}\langle y\rangle \vdash^{11} y : A}\,[\oplus]$$

showing that both $Voter_\mathsf{a}\langle x\rangle$ and $Voter_\mathsf{b}\langle y\rangle$ are well typed.

It is easy to obtain a typing derivation for $Ar\langle x, y, z\rangle \vdash^{3} x : A^{\perp}, y : A^{\perp}, z : \mathbf{1}$. With this we finally derive $(x)(Voter_{\mathsf{a}}\langle x\rangle \mid (y)(Ar\langle x, y, z\rangle \mid Voter_{\mathsf{b}}\langle y\rangle)) \vdash^{17} z : \mathbf{1}$.

We conclude this example with a few observations. First of all, note that A describes a communication protocol where termination is decided by an external entity (the arbiter), not by the agent behaving as A. In the type A, this translates to the fact that the only constant $\mathbf{1}$ is found as immediate leaf of a & type. If we were to seek for a derivation for $\vdash A$ in one of the logical systems based on dual fixed points, A would have to be modeled by a largest fixed point. But then the typing derivation for the arbiter would be invalid, since its branches would only unfold least fixed points.

Second, observe that different voting policies will usually result in different annotations in types which identify a "consensus behavior" to which the agents have to eventually adhere. Above, A requires the agents to eventually agree on a.

Finally, in the given example we have instrumented the arbiter uniformly so that measure transfers take place after each vote, even if they are not really needed. For instance, for the voters above it is clear that after sending a we could avoid sending some measure, as witnessed by the 0 annotation in the type A. However, figuring out *a priori* which instrumentation is really necessary can be challenging. The given implementation of the arbiter can be used in combination with voters adhereing to different voting policies, although the amount of measure exchanged between the arbiter and the agents will be different in each case.

8 Fair Termination

Some of the processes that we have discussed so far, like the slot machines in Examples 2.2 and 3.3 or the agents participating to a consensus protocol in Section 7.2, exhibit a purely deterministic behavior that we do not expect to experience in practice. Our modeling of these processes would benefit from the ability to describe a *non-deterministic choice* between winning and losing a game or between sending a and sending b, without necessarily describing in full detail how these choices are made. In this section we study a simple extension of CAPS and a corresponding conservative extension of sMALL that allow us to address a larger family of processes that can perform internal non-deterministic choices.

The basic idea is to extend the syntax of CAPS in this way

$$P, Q := \cdots \mid P \oplus Q$$

where $P \oplus Q$ is a non-deterministic choice among the behaviors P and Q that reduces according to the rule [R-CHOICE] defined as

$$P_1 \oplus P_2 \longrightarrow P_i$$

for every $i = 1, 2$. We can now model a non-deterministic slot machine that emits win and lose messages according to some undisclosed algorithm:

$$Slot_?\langle x\rangle = x \triangleright \left\{ \begin{array}{l} \mathsf{play} : \blacktriangleright x.(x \triangleleft \mathsf{win}.Slot_?\langle x\rangle \oplus x \triangleleft \mathsf{lose}.\blacktriangleleft x.Slot_?\langle x\rangle) \\ \mathsf{stop} : x[] \end{array} \right\}$$

The addition of non-deterministic choices to CAPS has a profound impact on its metatheory. To see why, consider the composition $(x)(Play\langle x, y\rangle \mid Slot_?\langle x\rangle)$ where $Play\langle x, y\rangle$ is the same player defined in Example 2.2. Even though both $Slot_1\langle x\rangle$ and $Slot_?\langle x\rangle$ allow for infinitely many outputs of both win and lose tags, the composition $(x)(Play\langle x, y\rangle \mid Slot_1\langle x\rangle)$ is terminating whereas the composition $(x)(Play\langle x, y\rangle \mid Slot_?\langle x\rangle)$ is not. Indeed, there exists an infinite run of this composition in which the slot machine always sends lose. This is rather unfortunate since it is reasonable to assume that no implementation of the slot machine should behave like this. Yet, from a purely technical standpoint, the system modeled using the non-deterministic slot machine is no longer terminating and therefore no longer in the scope of sMALL, even in light of the completeness result of Section 6.

This problem has received considerable attention in the literature and has lead to the definition of *fair termination* [22, 24], a weaker form of termination in which only the runs that are considered to be "realistic" (technically, "fair") should be taken into account. We now show how to extend sMALL in such a way that well-typed processes are guaranteed to be *fairly terminating*. A similar extension has already been studied for μMALL$^{\infty}$ [10].

The first step is to formalize the fairness assumption that we make, which turns out to be an instance of *full fairness* [35]. We say that P is *weakly (productively/successfully) terminating* if it has a (productive/successful) run of length at most l. In these cases we write $P\downarrow^{l}_{\mathsf{weak}}$ (respectively, $P\downarrow^{l}_{\mathsf{weak}}$ and $P\Downarrow^{l}_{\mathsf{weak}}$). We say that P is *diverging* if it is not weakly terminating. We say that a run is *fair* if it contains finitely many weakly terminating processes. In words, a run is fair either if it is finite or if it is infinite and contains a diverging process. If this description is still uninspiring, it may be useful to think of its complement: a run is *unfair* if it consists of infinitely many weakly terminating processes. That is, along an unfair run the process has always the chance to terminate, but it stubbornly refuses to do so.

Now, the notions of fair termination, productive fair termination and successful fair termination follow from their counterparts of Section 2 by considering fair runs only. We write $P\downarrow^{l}_{\mathsf{fair}}$ if every fair run of P is finite and $P\downarrow^{l}_{\mathsf{weak}}$. In this case we say that P is *fairly terminating*. We write $P\downarrow^{l}_{\mathsf{fair}}$ if every fair run of P is productive and $P\downarrow^{l}_{\mathsf{weak}}$. In this case we say that P is *fairly productively terminating*. We write $P\Downarrow^{l}_{\mathsf{fair}}$ if every fair run of P is successful and $P\Downarrow^{l}_{\mathsf{weak}}$. In this case, we say that P is *fairly successfully terminating*. Note that the measure annotation in the various fair termination predicates has a rather different meaning compared to the non-fair versions of the same predicates. The point is that in general there is no upper bound to the length of fair runs (see Example 8.7). However, that annotation plays an important role in connection with the measure annotation in typing judgments (Theorem 8.5). Occasionally we omit the measure annotation in the weak and fair termination predicates. In these cases, the annotation is existentially quantified.

It has been observed that the above notion of fair termination can be characterized without mentioning fair runs at all [6, 10].

THEOREM 8.1. *$P\downarrow_{\mathsf{fair}}$ if and only if $P \Rightarrow Q$ implies $Q\downarrow_{\mathsf{weak}}$.*

Similar results can be proved for productive and successful fair termination. The relevance of Theorem 8.1 in our setting is that its right-to-left implication provides a proof principle for proving the soundness of sMALL extended with non-deterministic choices. Indeed, if we manage to extend sMALL in such a way that well-typed processes are guaranteed to be *weakly* terminating, then by virtue of subject reduction and Theorem 8.1 we can conclude that well-typed processes are guaranteed to be *fairly* terminating.

An extension of sMALL with this property makes use of the following typing rule for non-deterministic choices

$$\frac{\forall i = 1,2 : P_i \vdash^{l_i} \Gamma}{P_1 \oplus P_2 \vdash^{1+l_k} \Gamma}\ [\text{CHOICE}]\ k \in \{1,2\}$$

where we measure a non-deterministic choice with the successor of the measure of one of its branches. Since we strive to look for derivations in which the measure is finite, in general this amounts to choosing k as the index of the branch with the least measure. In the rest of this section we write $P \vdash^{l}_{\text{fair}} \Gamma$ whenever the judgment $P \vdash^{l} \Gamma$ is derivable by the rules in Table 3 extended with [CHOICE].

Let us now discuss the properties of this extension. While subject extraction (Lemma 4.1) holds unchanged since extractions have not been modified, subject reduction (Theorem 4.2) no longer guarantees that the measure strictly decreases because of [CHOICE]. Instead we have:

Theorem 8.2. *If $P \vdash^{l}_{\text{fair}} \Gamma$ and $P \to Q$, then $Q \vdash^{m} \Gamma$ for some m.*

However, it is possible to show that every *reducible* process may be reduced in such a way so as to decrease the measure.

Proposition 8.3. *If $P \vdash^{l}_{\text{fair}} \Gamma$ and $P \not\to$, then $Q \vdash^{m}_{\text{fair}} \Gamma$ for some Q and $m < l$ such that $P \to Q$.*

As a consequence, well-typed processes are weakly terminating and their measure is an upper-bound to the length of at least one of their runs.

Proposition 8.4. *If $P \vdash^{l}_{\text{fair}} \Gamma$, then $P{\downarrow}^{l}_{\text{weak}}$.*

Now, by combining Theorem 8.2 and Proposition 8.4 we can prove the soundness of (non-deterministic) sMALL with respect to fair termination using the characterization of fair termination given in Theorem 8.1.

Theorem 8.5. *If $P \vdash^{l}_{\text{fair}} \Gamma$, then $P{\downarrow}^{l}_{\text{fair}}$.*

For the particular context $\Gamma = x : \mathbf{1}$, Theorem 8.5 can be strengthened to entail fair successful termination.

Example 8.6. Consider again the types A and B defined in Example 3.3. We derive

$$\dfrac{\dfrac{\dfrac{\vdots}{Slot_?\langle x\rangle \vdash^{4} x : A^{\perp}}}{x \triangleleft \mathsf{win}.Slot_?\langle x\rangle \vdash^{5} x : B^{\perp}} \quad \dfrac{\dfrac{\vdots}{Slot_?\langle x\rangle \vdash^{1} x : A^{\perp}}}{\blacktriangleleft x.Slot_?\langle x\rangle \vdash^{9} x : \blacktriangleleft^{7} A^{\perp}}}{x \triangleleft \mathsf{win}\ldots \oplus x \triangleleft \mathsf{lose}\ldots \vdash^{6} x : B^{\perp}}}{\blacktriangleright x.x \triangleleft \mathsf{win}\ldots \oplus x \triangleleft \mathsf{lose}\ldots \vdash^{1} x : \blacktriangleright^{5} B^{\perp}} \quad \dfrac{}{x[] \vdash^{1} x : \mathbf{1}}}{Slot_?\langle x\rangle \vdash^{1} x : A^{\perp}}$$

and therefore $(x)(Play\langle x, y\rangle \mid Slot_?\langle x\rangle) \vdash^{10} y : \mathbf{1}$ with one application of [CUT]. By Theorem 8.5 we deduce that $(x)(Play\langle x, y\rangle \mid Slot_?\langle x\rangle)$ is fairly successfully terminating. Note that, in the derivation above, the right branch of the application of [CHOICE] has a measure that is strictly larger than that of the choice itself. Each time the reduction of $(x)(Play\langle x, y\rangle \mid Slot_?\langle x\rangle)$ follows that branch, its measure increases as the path that leads to termination must go through (at least) one more game. ∎

Example 8.7. In this example we show that there exist well-typed processes for which there is no finite upper bound to their measure. If we consider the definition $Walk(x) \triangleq x[] \oplus (y)(Walk\langle y\rangle \mid y().Walk\langle x\rangle)$ we derive

$$\dfrac{\dfrac{}{x[] \vdash^{1} x : \mathbf{1}}\ [1] \quad \dfrac{\dfrac{\vdots}{Walk\langle y\rangle \vdash^{2} y : \mathbf{1}} \quad \dfrac{\dfrac{\vdots}{Walk\langle x\rangle \vdash^{2} x : \mathbf{1}}}{y().Walk\langle x\rangle \vdash^{2} x : \mathbf{1}, y : \perp}\ [\perp]}{(y)(Walk\langle y\rangle \mid y().Walk\langle x\rangle) \vdash^{4} x : \mathbf{1}}\ [\text{CUT}]}{Walk\langle x\rangle \vdash^{2} x : \mathbf{1}}\ [\text{CHOICE}]$$

and now for every $n \in \mathbb{N}$ there exists Q_n such that $Walk\langle x\rangle \to^{n} Q_n$ such that $Q_n \vdash^{2n}_{\text{fair}} x : \mathbf{1}$. ∎

We conclude this section by observing that the cut elimination result of Section 5 can be easily extended to non-deterministic sMALL. The only aspect one has to be careful about is that, in reducing a process as much as possible, choices should be reduced by picking the k-tagged branch in accordance with [CHOICE]. The obtained derivation is not only cut free but also choice free.

9 Related Work

The literature on session types and on their connection with Linear Logic is vaste. Here we focus on two areas more closely related to the present work: proof systems for Linear Logic with fixed points and session type systems enforcing termination-related properties.

The common way of enabling forms of recursion within a logical system, the (modal) μ-calculus being a typical example, is by adding (least and greatest) fixed point operators. The study of the proof theory of such extensions for classical Linear Logic goes back to Baelde [1] and has been subsequently developed in an infinitary system called μMALL$^{\infty}$ by Baelde et al. [3] and Doumane [21]. μMALL$^{\infty}$ allows possibly infinite proof derivations built using standard rules of MALL together with two additional rules for unfolding fixed point formulas. Allowing arbitrary infinite proofs in general leads to inconsistency. For this reason, μMALL$^{\infty}$ imposes a *validity condition* on derivations which, roughly speaking, requires all infinite branches of the derivation to be supported by the infinite unfolding of the same greatest fixed point. This validity condition suffices to prove cut elimination, thus ensuring consistency, but it does not interact well with cuts and axioms. In particular, validity is not necessarily preserved by cut introduction. To overcome these limitations, Baelde et al. [2] propose a more sophisticated (but undecidable in general) validity condition that takes cuts and axioms into account. The resulting proof system still enjoys cut elimination and proves the same sequents as the original one, but enlarges the set of valid proofs. This is important from a computational point of view because, more valid proofs means more well-typed programs.

Derakhshan [18] and Derakhshan and Pfenning [19] study an analogous extension for the subsingleton fragment of intuitionistic Linear Logic. Instead of using fixed point operators in formulas, they represent fixed point formulas by means of a set of mutually recursive equations, each of them interpreted either inductively or coinductively. Equations need to be appropriately stratified in order to avoid mutual recursion between inductive and coinductive equations. This is achieved by labelling equations with natural numbers representing priorities. Infinite derivations in this system

still need a validity condition to preserve cut elimination but, thanks to the aforementioned labelling, this condition can be checked locally, resulting in better compositionality compared to μMALL$^\infty$.

The logic proposed in this paper differs from the above systems in several ways. First of all, sMALL abandons the approach based on fixed point operators in favor of infinite propositions. This choice grants the maximum flexibility, especially in the session-based setting that sMALL is targeting, where there is a relevant class of protocols whose semantics eludes the crude classification into least and greatest fixed points. Typical examples of such protocols are those involving a negotiation between two (or more) parties, like the slot machine (Examples 2.2 and 3.3) or the consensus (Section 7.2) protocols. Second, previous logical systems require validity conditions on infinite derivations in order to retain cut elimination and therefore consistency. Also, the proof of cut elimination of μMALL$^\infty$ is quite complex and requires a careful setup [21]. In contrast, sMALL derivations are valid by construction and cut elimination is straightforward, but the heavy use of annotations in formulas and of corresponding prefixes in processes makes them more difficult to understand (see *e.g.* Section 7.2).

We achieve these results by leveraging on well-established techniques from *resource-aware session types* [12, 13], which provide the foundation for the RAST language [15] and its extensions [16]. Resource-aware session types are designed for analyzing the total work done by a process: they associate a cost to certain actions and introduce annotations for trasferring such costs between the two sides of a communication channel. These annotations can be either attached to each type constructor [13] or treated themselves as constructors with their corresponding term formers [12]. These systems are proved to be sound with respect to a *cost semantics* that keeps track of the total work performed by a process. However, these soundness results cannot be used, as they are formulated, to infer any general termination property for well-typed processes. Another somewhat related technique that enriches session types with quantitative information is *arithmetic refinements* [14], which extends types with expressions and assertions from a decidable arithemtics to measure typing judgements. Somayyajula and Pfenning [34] apply this technique to ensure termination of a process calculus extracted from the semi-axiomatic sequent calculus, which models a form of concurrency based on futures.

Despite the fact that the ability to describe "infinite computations" is considered to be a key testbench for many models of concurrent systems, termination remains a key property since, in combination with deadlock freedom, it entails valuable liveness properties akin to productivity such as lock freedom [29] and strong progress [19]. Many session type systems based on Linear Logic [9, 30, 36] ensure termination-related properties as direct consequences of cut elimination. As these type systems make use of least and greatest fixed points, they are unable to capture some communication patterns involving negotiations. Other type-based techniques to enforce termination in π-calculi have been proposed by Deng and Sangiorgi [17], Sangiorgi [33], Yoshida et al. [37]. In particular, Deng and Sangiorgi [17], Sangiorgi [33] propose a technique based on annotated types which superficially look similar to our own, although their annotations are priority levels used to constraint the order in which different channels are used and are not directly related to the measure of a process. Paulus et al. [31] compare (a variant of) this approach with those based on intuitionistic Linear Logic, showing that it can capture more terminating processes.

As we have seen in Section 8, non-determinism may have a disruptive effect on an otherwise well-behaved model in the sense that diverging but unrealistic computations may easily compromise termination. To tame this phenomenon, weaker versions of termination like *fair termination* [22, 24] have been proposed. Ciccone and Padovani [8] and Ciccone et al. [6, 7] propose session type systems ensuring the fair termination of respectively binary and multiparty sessions. These type systems share with sMALL the idea of annotating judgements with measures. However, they are not as closely related to Linear Logic as sMALL and they do not make use of measure annotations in types. As a result, many processes that are well-typed in sMALL cannot be typed in such systems (the non-deterministic slot machine in Example 8.6 being one such example). Ciccone and Padovani [9] propose an interpretation of μMALL$^\infty$ proofs as processes in the linear π-calculus, into which a session calculus like CaPS can be encoded [11]. They present an extension of μMALL$^\infty$ with non-deterministic choices and a consequent refinement of μMALL$^\infty$'s validity condition proving that well-typed processes are fairly terminating. Being based on μMALL$^\infty$, their type system is affected by the limitations that we have described in Section 1 and that have motivated this paper.

10 Concluding Remarks

sMALL is an infinitary version of multiplicative-additive linear logic without fixed points but with possibly infinite propositions. This feature finds natural applications in the description of session-based communication protocols, particularly those based on negotiations which are out of scope of other logical foundations for sessions.

We have already pointed out that sMALL is incomparable to other systems based on linear logic in terms of valid proofs. However, the exact relationship between sMALL and other extensions of Linear Logic with fixed points requires further investigations to be fully appreciated. On the one hand, the reasoning on positively/negatively balanced loops in types (Section 7.2) suggests that measure annotations may provide enough information to interpret (some) types as least or greatest fixed points. On the other hand, it seems that measure annotations intrinsically limit the class of typeble processes, by imposing constraints on their complexity. For instance, we conjecture that a process implementing the multiplication of Peano natural numbers, which has a non-linear complexity, cannot be typed. Intuitively, this would be due to the fact that this process must duplicate one of the provided natural numbers and use one of the copies in the recursive call; this copy necessarily stores less measure than the original number as the duplication process has to split the measure between the two copies, which conflicts with the fact that it is used in a recursive call. It should also be pointed out that, while it is relatively easy to find examples of "non-linear" processes that are difficult or apparently impossible to type, they tend not to correspond to the typical communication protocols we expect to be able to model in CaPS.

Looking ahead, we are confident that some of the assumptions needed by the completeness result in Section 6 can be relaxed (Remark 6.4) and we think that a similar completeness result should be provable with respect to fair productive termination (Section 8).

References

[1] David Baelde. 2012. Least and Greatest Fixed Points in Linear Logic. *ACM Trans. Comput. Log.* 13, 1 (2012), 2:1–2:44. https://doi.org/10.1145/2071368.2071370

[2] David Baelde, Amina Doumane, Denis Kuperberg, and Alexis Saurin. 2022. Bouncing Threads for Circular and Non-Wellfounded Proofs: Towards Compositionality with Circular Proofs. In *LICS '22: 37th Annual ACM/IEEE Symposium on Logic in Computer Science, Haifa, Israel, August 2 - 5, 2022*, Christel Baier and Dana Fisman (Eds.). ACM, 63:1–63:13. https://doi.org/10.1145/3531130.3533375

[3] David Baelde, Amina Doumane, and Alexis Saurin. 2016. Infinitary Proof Theory: the Multiplicative Additive Case. In *25th EACSL Annual Conference on Computer Science Logic, CSL 2016, August 29 - September 1, 2016, Marseille, France (LIPIcs, Vol. 62)*, Jean-Marc Talbot and Laurent Regnier (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 42:1–42:17. https://doi.org/10.4230/LIPIcs.CSL.2016.42

[4] Luís Caires and Frank Pfenning. 2010. Session Types as Intuitionistic Linear Propositions. In *CONCUR 2010 - Concurrency Theory, 21th International Conference, CONCUR 2010, Paris, France, August 31-September 3, 2010. Proceedings (Lecture Notes in Computer Science, Vol. 6269)*, Paul Gastin and François Laroussinie (Eds.). Springer, 222–236. https://doi.org/10.1007/978-3-642-15375-4_16

[5] Luís Caires, Frank Pfenning, and Bernardo Toninho. 2016. Linear logic propositions as session types. *Math. Struct. Comput. Sci.* 26, 3 (2016), 367–423. https://doi.org/10.1017/S0960129514000218

[6] Luca Ciccone, Francesco Dagnino, and Luca Padovani. 2022. Fair Termination of Multiparty Sessions. In *36th European Conference on Object-Oriented Programming, ECOOP 2022, June 6-10, 2022, Berlin, Germany (LIPIcs, Vol. 222)*, Karim Ali and Jan Vitek (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 26:1–26:26. https://doi.org/10.4230/LIPICS.ECOOP.2022.26

[7] Luca Ciccone, Francesco Dagnino, and Luca Padovani. 2024. Fair termination of multiparty sessions. *Journal of Logical and Algebraic Methods in Programming* 139 (2024), 100964. https://doi.org/10.1016/j.jlamp.2024.100964

[8] Luca Ciccone and Luca Padovani. 2022. Fair termination of binary sessions. *Proc. ACM Program. Lang.* 6, POPL (2022), 1–30. https://doi.org/10.1145/3498666

[9] Luca Ciccone and Luca Padovani. 2022. Inference Systems with Corules for Combined Safety and Liveness Properties of Binary Session Types. *Log. Methods Comput. Sci.* 18, 3 (2022). https://doi.org/10.46298/LMCS-18(3:27)2022

[10] Luca Ciccone and Luca Padovani. 2022. An Infinitary Proof Theory of Linear Logic Ensuring Fair Termination in the Linear π-Calculus. In *33rd International Conference on Concurrency Theory, CONCUR 2022, September 12-16, 2022, Warsaw, Poland (LIPIcs, Vol. 243)*, Bartek Klin, Slawomir Lasota, and Anca Muscholl (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 36:1–36:18. https://doi.org/10.4230/LIPICS.CONCUR.2022.36

[11] Ornela Dardha, Elena Giachino, and Davide Sangiorgi. 2017. Session types revisited. *Inf. Comput.* 256 (2017), 253–286. https://doi.org/10.1016/j.ic.2017.06.002

[12] Ankush Das, Stephanie Balzer, Jan Hoffmann, Frank Pfenning, and Ishani Santurkar. 2021. Resource-Aware Session Types for Digital Contracts. In *34th IEEE Computer Security Foundations Symposium, CSF 2021, Dubrovnik, Croatia, June 21-25, 2021*. IEEE, 1–16. https://doi.org/10.1109/CSF51468.2021.00004

[13] Ankush Das, Jan Hoffmann, and Frank Pfenning. 2018. Work Analysis with Resource-Aware Session Types. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, Anuj Dawar and Erich Grädel (Eds.). ACM, 305–314. https://doi.org/10.1145/3209108.3209146

[14] Ankush Das and Frank Pfenning. 2020. Session Types with Arithmetic Refinements. In *31st International Conference on Concurrency Theory, CONCUR 2020, September 1-4, 2020, Vienna, Austria (Virtual Conference) (LIPIcs, Vol. 171)*, Igor Konnov and Laura Kovács (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 13:1–13:18. https://doi.org/10.4230/LIPICS.CONCUR.2020.13

[15] Ankush Das and Frank Pfenning. 2022. Rast: A Language for Resource-Aware Session Types. *Log. Methods Comput. Sci.* 18, 1 (2022). https://doi.org/10.46298/LMCS-18(1:9)2022

[16] Ankush Das, Di Wang, and Jan Hoffmann. 2023. Probabilistic Resource-Aware Session Types. *Proc. ACM Program. Lang.* 7, POPL (2023), 1925–1956. https://doi.org/10.1145/3571259

[17] Yuxin Deng and Davide Sangiorgi. 2004. Ensuring Termination by Typability. In *Exploring New Frontiers of Theoretical Informatics, IFIP 18th World Computer Congress, TC1 3rd International Conference on Theoretical Computer Science (TCS2004), 22-27 August 2004, Toulouse, France (IFIP, Vol. 155)*, Jean-Jacques Lévy, Ernst W. Mayr, and John C. Mitchell (Eds.). Kluwer/Springer, 619–632. https://doi.org/10.1007/1-4020-8141-3_47

[18] Farzaneh Derakhshan. 2021. *Session-Typed Recursive Processes and Circular Proofs*. Ph. D. Dissertation. Carnegie Mellon University. https://www.andrew.cmu.edu/user/fderakhs/publications/Dissertation_Farzaneh.pdf

[19] Farzaneh Derakhshan and Frank Pfenning. 2022. Circular Proofs as Session-Typed Processes: A Local Validity Condition. *Log. Methods Comput. Sci.* 18, 2 (2022). https://doi.org/10.46298/LMCS-18(2:8)2022

[20] Henry DeYoung and Frank Pfenning. 2016. Substructural Proofs as Automata. In *Programming Languages and Systems - 14th Asian Symposium, APLAS 2016, Hanoi, Vietnam, November 21-23, 2016, Proceedings (Lecture Notes in Computer Science, Vol. 10017)*, Atsushi Igarashi (Ed.). Springer, 3–22. https://doi.org/10.1007/978-3-319-47958-3_1

[21] Amina Doumane. 2017. *On the infinitary proof theory of logics with fixed points. (Théorie de la démonstration infinitaire pour les logiques à points fixes)*. Ph. D. Dissertation. Paris Diderot University, France. https://tel.archives-ouvertes.fr/tel-01676953

[22] Nissim Francez. 1986. *Fairness*. Springer. https://doi.org/10.1007/978-1-4612-4886-6

[23] Jean-Yves Girard. 1987. Linear Logic. *Theor. Comput. Sci.* 50 (1987), 1–102. https://doi.org/10.1016/0304-3975(87)90045-4

[24] Orna Grumberg, Nissim Francez, and Shmuel Katz. 1984. Fair Termination of Communicating Processes. In *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing* (Vancouver, British Columbia, Canada) *(PODC '84)*. Association for Computing Machinery, New York, NY, USA, 254–265. https://doi.org/10.1145/800222.806752

[25] Kohei Honda. 1993. Types for Dyadic Interaction. In *CONCUR '93, 4th International Conference on Concurrency Theory, Hildesheim, Germany, August 23-26, 1993, Proceedings (Lecture Notes in Computer Science, Vol. 715)*, Eike Best (Ed.). Springer, 509–523. https://doi.org/10.1007/3-540-57208-2_35

[26] Kohei Honda, Vasco Thudichum Vasconcelos, and Makoto Kubo. 1998. Language Primitives and Type Discipline for Structured Communication-Based Programming. In *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 (Lecture Notes in Computer Science, Vol. 1381)*, Chris Hankin (Ed.). Springer, 122–138. https://doi.org/10.1007/BFb0053567

[27] Ross Horne and Luca Padovani. 2023. A Logical Account of Subtyping for Session Types. In *Proceedings 14th Workshop on Programming Language Approaches to Concurrency and Communication-cEntric Software, PLACES@ETAPS 2023, Paris, France, 22 April 2023 (EPTCS, Vol. 378)*, Ilaria Castellani and Alceste Scalas (Eds.). Open Publishing Association, 26–37. https://doi.org/10.4204/EPTCS.378.3

[28] Hans Hüttel, Ivan Lanese, Vasco T. Vasconcelos, Luís Caires, Marco Carbone, Pierre-Malo Deniélou, Dimitris Mostrous, Luca Padovani, António Ravara, Emilio Tuosto, Hugo Torres Vieira, and Gianluigi Zavattaro. 2016. Foundations of Session Types and Behavioural Contracts. *ACM Comput. Surv.* 49, 1 (2016), 3:1–3:36. https://doi.org/10.1145/2873052

[29] Naoki Kobayashi and Davide Sangiorgi. 2010. A hybrid type system for lock-freedom of mobile processes. *ACM Trans. Program. Lang. Syst.* 32, 5 (2010), 16:1–16:49. https://doi.org/10.1145/1745312.1745313

[30] Sam Lindley and J. Garrett Morris. 2016. Talking bananas: structural recursion for session types. In *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming, ICFP 2016, Nara, Japan, September 18-22, 2016*, Jacques Garrigue, Gabriele Keller, and Eijiro Sumii (Eds.). ACM, 434–447. https://doi.org/10.1145/2951913.2951921

[31] Joseph W. N. Paulus, Jorge A. Pérez, and Daniele Nantes-Sobrinho. 2023. Termination in Concurrency, Revisited. In *International Symposium on Principles and Practice of Declarative Programming, PPDP 2023, Lisboa, Portugal, October 22-23, 2023*, Santiago Escobar and Vasco T. Vasconcelos (Eds.). ACM, 3:1–3:14. https://doi.org/10.1145/3610612.3610615

[32] Zesen Qian, G. A. Kavvos, and Lars Birkedal. 2021. Client-server sessions in linear logic. *Proc. ACM Program. Lang.* 5, ICFP (2021), 1–31. https://doi.org/10.1145/3473567

[33] Davide Sangiorgi. 2006. Termination of processes. *Math. Struct. Comput. Sci.* 16, 1 (2006), 1–39. https://doi.org/10.1017/S0960129505004810

[34] Siva Somayyajula and Frank Pfenning. 2022. Type-Based Termination for Futures. In *7th International Conference on Formal Structures for Computation and Deduction, FSCD 2022, August 2-5, 2022, Haifa, Israel (LIPIcs, Vol. 228)*, Amy P. Felty (Ed.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 12:1–12:21. https://doi.org/10.4230/LIPICS.FSCD.2022.12

[35] Rob van Glabbeek and Peter Höfner. 2019. Progress, Justness, and Fairness. *ACM Comput. Surv.* 52, 4 (2019), 69:1–69:38. https://doi.org/10.1145/3329125

[36] Philip Wadler. 2014. Propositions as sessions. *J. Funct. Program.* 24, 2-3 (2014), 384–418. https://doi.org/10.1017/S095679681400001X

[37] Nobuko Yoshida, Martin Berger, and Kohei Honda. 2001. Strong Normalisation in the pi-Calculus. In *16th Annual IEEE Symposium on Logic in Computer Science, Boston, Massachusetts, USA, June 16-19, 2001, Proceedings*. IEEE Computer Society, 311–322. https://doi.org/10.1109/LICS.2001.932507