TECHNICAL REPORT

Report No. CS2012-02
Date: September 2012

Compositional Reasoning for Channel-Based Concurrent Resource Management

Adrian Francalanza
Edsko de Vries
Matthew Hennessy

Department of Computer Science
University of Malta
Msida MSD 06
MALTA

Tel: +356-2340 2519
Fax: +356-2132 0539
http://www.cs.um.edu.mt

Compositional Reasoning for Channel-Based Concurrent Resource Management

Adrian Francalanza
CS Dept., University of Malta
adrian.francalanza@um.edu.mt

Edsko de Vries
Well-Typed LLP, UK
edsko@well-typed.com

Matthew Hennessy
CS Dept., Trinity College Dublin, Ireland
matthew.hennessy@cs.tcd.ie

Abstract: *We define a π-calculus variant with a costed semantics where channels are treated as resources that must explicitly be allocated before they are used and can be deallocated when no longer required. We use a substructural type system tracking permission transfer to construct compositional proof techniques for comparing behaviour and resource usage efficiency of concurrent processes.*

Compositional Reasoning for Explicit Resource Management in Channel-Based Concurrency *

Adrian Francalanza
CS Dept., University of Malta
adrian.francalanza@um.edu.mt

Edsko de Vries
Well-Typed LLP, UK
edsko@well-typed.com

Matthew Hennessy
CS Dept., Trinity College Dublin, Ireland
matthew.hennessy@cs.tcd.ie

Abstract: *We define a π-calculus variant with a costed semantics where channels are treated as resources that must explicitly be allocated before they are used and can be deallocated when no longer required. We use a substructural type system tracking permission transfer to construct compositional proof techniques for comparing behaviour and resource usage efficiency of concurrent processes.*

1 Introduction

We investigate the *behaviour* and *space efficiency* of concurrent programs with *explicit resource-management*. In particular, our study focusses on *channel-passing concurrent programs*: we define a π-calculus variant, called $R\pi$, where the only resources available are channels; these channels must explicitly be allocated before they can be used, and can be deallocated when no longer required. As part of the operational model of the language, channel allocation and deallocation have costs associated with them, reflecting the respective resource usage.

Explicit resource management is typically desirable in settings where resources are *scarce*. Resource management constructs, such as explicit deallocation, provide fine-grained control over how these resources are used and recycled. By comparison, in automated mechanisms such as garbage collection, unused resources (in this case, memory) tend to remain longer in an unreclaimed state [Jon96]. Explicit resource management constructs such as memory deallocation also carry advantages over automated mechanism such as garbage collection techniques when it comes to *interactive* and *real-time* programs [BM00, Jon96]. In particular, garbage collection techniques require additional computation to determine otherwise explicit information as to which parts of the memory to reclaim and at what stage of the computation; the associated overheads may lead to uneven performance and intolerable pause periods where the system becomes unresponsive [BM00].

In the case of channel-passing concurrency with explicit memory-management, the analysis of the relative behaviour and efficiency of programs is non-trivial for a number of reasons. Explicit

*Research partially funded by SFI project SFI 06 IN.1 1898.

memory-management introduces the risk of either premature or multiple deallocation of resources along separate threads of executions; these are more difficult to detect than in single-threaded programs and potentially result in problems such as wild pointers or corrupted heaps which may, in turn, lead to unpredictable, even catastrophic, behaviour [Jon96]. It also increases the possibility of memory leaks, which are often not noticeable in short-running, terminating programs but subtly eat up resources over the course of long-running programs. In a concurrent settings such as ours, complications relating to the assessment and comparison of resource consumption is further compounded by the fact that the runtime execution of channel-passing concurrent programs can have *multiple interleavings*, is sometimes *non-deterministic* and often *non-terminating*.

1.1 Scenario

Consider a setting with two servers, S_1 and S_2, which repeatedly listen for service requests on channels $\mathtt{srv}_1$ and $\mathtt{srv}_2$, respectively. Requests send a *return* channel on $\mathtt{srv}_1$ or $\mathtt{srv}_2$ which is then used by the servers to service the requests and send back answers, v_1 and v_2. A possible implementation for these servers is given in (1) below, where $\mathtt{rec}\, w.P$ denotes a process P recursing at w, $\mathtt{c}?x.P$ denotes a process inputting on channel $\mathtt{c}$ some value that is bound to the variable x in the continuation P, and $\mathtt{c}!v.P$ outputs a value v on channel $\mathtt{c}$ and continues as P:

$$S_i \triangleq \mathtt{rec}\, w.\ \mathtt{srv}_i?x.\ x!v_i.\ X \qquad \text{for } i \in \{1,2\} \tag{1}$$

Clients that need to request service from *both* servers, so as to report back the outcome of both server interactions on some channel, $\mathtt{ret}$, can be programmed in a variety of ways:

$$\begin{aligned} C_0 &\triangleq \mathtt{rec}\, w.\ \mathtt{alloc}\, x_1.\mathtt{alloc}\, x_2.\ \mathtt{srv}_1!x_1.\, x_1?y.\ \mathtt{srv}_2!x_2.\, x_2?z.\ \mathtt{ret}!(y,z).\ w \\ C_1 &\triangleq \mathtt{rec}\, w.\ \mathtt{alloc}\, x.\ \mathtt{srv}_1!x.\, x?y.\ \mathtt{srv}_2!x.\, x?z.\mathtt{ret}!(y,z).\ w \\ C_2 &\triangleq \mathtt{rec}\, w.\mathtt{alloc}\, x.\ \mathtt{srv}_1!x.\, x?y.\ \mathtt{srv}_2!x.\, x?z.\ \mathtt{free}\, x.\ \mathtt{ret}!(y,z).\ w \end{aligned} \tag{2}$$

C_0 corresponds to an idiomatic π-calculus client. In order to ensure that it is the sole recipient of the service requests, it creates *two* new return channels to communicate with S_1 and S_2 on $\mathtt{srv}_1$ and $\mathtt{srv}_2$, using the command $\mathtt{alloc}\, x.P$; this command allocates a *new* channel $\mathtt{c}$ and binds it to the variable x in the continuation P. Thus, allocating a new channel for each service request ensures that the return channel used between the client and server is *private* for the duration of the service, preventing interferences from other parties executing in parallel.

One important difference between the computational model considered in this paper and that of the standard π-calculus is that channel allocation is an expensive operation *i.e.,* it incurs an additional *(spatial)* cost compared to the other operations. Client C_1 attempts to address the inefficiencies of C_0 by allocating only *one* additional new channel, and *reusing* this channel for both interactions with the servers. Intuitively, this channel reuse is valid, *i.e.,* it preserves the client-server behaviour C_0 had with servers S_1 and S_2, because the server implementations above use the received return-channels *only once*. This single channel usage guarantees that return channels remain private during the duration of the service, despite the reuse from client C_1.

Client C_2 attempts to be more efficient still. More precisely, since our computational model does not assume implicit resource reclamation, the previous two clients can be deemed as having *memory leaks*: at every iteration of the client-server interaction sequence, C_0 and C_1 allocate new channels that are not disposed, even though these channels are never used again in subsequent iterations. By contrast, C_2 deallocates unused channels at the end of each iteration using the construct $\texttt{free}\,\texttt{c}.P$.

In this work we develop a formal framework for comparing the behaviour of concurrent processes that explicitly allocate and deallocate channels. For instance, processes consisting of the servers S_1 and S_2 together with any of the clients C_0, C_1 or C_2 should be *related*, on the basis that they exhibit the same behaviour. In addition, we would like to *order* these systems, based on their relative efficiencies *wrt.* the (channel) resources used. Thus, we would intuitively like to develop a framework yielding the following preorder, where $\sqsubseteq$ reads "more efficient than":

$$S_1 \parallel S_2 \parallel C_2 \quad \sqsubseteq \quad S_1 \parallel S_2 \parallel C_1 \quad \sqsubseteq \quad S_1 \parallel S_2 \parallel C_0 \tag{3}$$

A pleasing property of this preorder would be *compositionality*, which implies that orderings are preserved under larger contexts, *i.e.,* for all (valid) contexts $C[-]$, $P \sqsubseteq Q$ implies $C[P] \sqsubseteq C[Q]$. Dually, compositionality would also improve the scalability of our formal framework since, to show that $C[P] \sqsubseteq C[Q]$ (for some context $C[-]$), it suffices to obtain $P \sqsubseteq Q$. For instance, in the case of (3), compositionality would allow us to factor out the common code, *i.e.,* the servers S_1 and S_2 as the context $S_1 \parallel S_2 \parallel [-]$, and focus on showing that

$$C_2 \sqsubseteq C_1 \sqsubseteq C_0 \tag{4}$$

1.2 Main Challenges

The details are however not as straightforward. To begin with, we need to asses relative program cost over potentially infinite computations. Thus, rudimentary aggregate measures such as adding up the total computation cost of processes and comparing this total at the end of the computation is insufficient for system comparisons such as (3). In such cases, a preliminary attempt at a solution would be to compare the *relative cost* for *every* server interaction (action): in the sense of [AKH92], the preorder would then ensure that every *costed* interaction by the inefficient clients must be matched by a corresponding *cheaper* interaction by the more efficient client (and viceversa).

$$C_3 \triangleq \texttt{rec}\, w.\texttt{alloc}\, x_1.\texttt{alloc}\, x_2.\ \texttt{srv}_1!x_1.\, x_1?y.\ \texttt{srv}_2!x_2.\, x_2?z.\ \texttt{free}\, x_1.\texttt{free}\, x_2.\texttt{ret}!(y,z).w \tag{5}$$

There are however problems with this approach. Consider, for instance, C_3 defined in (5). Even though this client allocates two channels for every iteration of server interactions, it does not exhibit any memory leaks since it deallocates them both at the end of the iteration. It may therefore be sensible for our preorder to equate C_3 with client C_2 of (2) by having $C_2 \sqsubseteq C_3$ as well as $C_3 \sqsubseteq C_2$. However showing $C_3 \sqsubseteq C_2$ would not be possible using the preliminary strategy discussed above, since, C_3 must engage in more expensive computation (allocating two channels as opposed to 1) by the time the interaction with the first server is carried out.

Worse still, an analysis strategy akin to [AKH92] would also not be applicable for a comparison involving the clients C_1 and C_3. In spite of the fact that over the course of its entire computation C_3 requires less resources than C_1 , *i.e.,* it is more efficient, after the interaction with the first server on channel $\mathtt{srv}_1$, client C_3 appears to be *less efficient* than C_1 since, at that stage, it has allocated two new channels as opposed to one. However, C_1 becomes less efficient for the remainder of the iteration since it never deallocates the channel it allocates whereas C_3 deallocates both channels. To summarise, for comparisons $C_3 \sqsubseteq C_2$ and $C_3 \sqsubseteq C_1$, we need our analysis to allow a process to be *temporarily inefficient* as long as it can recover later on.

In this paper, we use a costed semantics to define an efficiency preorder to reason about the relative cost of processes over potentially infinite computation, based on earlier work by [KAK05, LV06]. In particular, we adapt the concept of *cost amortisation* to our setting, used by our preorders to compare processes that are eventually more efficient that others over the course of their entire computation, but are temporarily less efficient at certain stages of the computation.

Issues concerning cost assessment are however not the only obstacles tackled in this work; there are also complications associated with the compositionality aspects of our proposed framework. More precisely, we want to limit our analysis to *safe* contexts, *i.e.,* contexts that use resources in a sensible way, *e.g.,* not deallocating channels while they are still in use. In addition, we also want to consider behaviour *wrt.* a subset of the possible safe contexts. For instance, our clients from (2) only exhibit the same behaviour *wrt.* servers that (*i*) accept *(any number of)* requests on channels $\mathtt{srv}_1$ and $\mathtt{srv}_2$ containing a return channel, which then (*ii*) use this channel at most *once* to return the requested answer. We can characterise the interface between the servers and the clients using fairly standard channel type descriptions adapted from [KPT99] in (6), where $[\mathbf{T}]^{\omega}$ describes a channel than can be used *any* number of times (*i.e.,* the channel-type attribute ω) to communicate values of type $\mathbf{T}$, whereas $[\mathbf{T}]^{\mathbf{1}}$ denotes an *affine* channel (*i.e.,* a channel type with attribute $\mathbf{1}$) that can be used *at most* once to communicate values of type $\mathbf{T}$:

$$\mathtt{srv}_1 : [[\mathbf{T}_1]^{\mathbf{1}}]^{\omega}, \quad \mathtt{srv}_2 : [[\mathbf{T}_2]^{\mathbf{1}}]^{\omega} \tag{6}$$

In the style of [YHB07, HR04], we could then use this interface to abstract away from the actual server implementations described in (1) and state that, *wrt.* contexts that observe the channel mappings of (6), client C_2 is more efficient than C_1 which is, in turn, more efficient than C_0. These can be expressed as:

$$\mathtt{srv}_1 : [[\mathbf{T}_1]^{\mathbf{1}}]^{\omega}, \mathtt{srv}_2 : [[\mathbf{T}_2]^{\mathbf{1}}]^{\omega} \models C_2 \sqsubseteq C_1 \tag{7}$$

$$\mathtt{srv}_1 : [[\mathbf{T}_1]^{\mathbf{1}}]^{\omega}, \mathtt{srv}_2 : [[\mathbf{T}_2]^{\mathbf{1}}]^{\omega} \models C_1 \sqsubseteq C_0 \tag{8}$$

Unfortunately, the machinery of [YHB07, HR04] cannot be extended forthrightly to our costed analysis because of two main reasons. First, in order to limit our analysis to safe computation, we would need to show that clients C_0, C_1 and C_2 adhere to the channel usage stipulated by the type associations in (6). However, the channel reuse in C_1 and C_2 (an essential feature to attain space efficiency) requires our analysis to associate potentially different types (*i.e.,* $[\mathbf{T}_1]^{\mathbf{1}}$ and $[\mathbf{T}_2]^{\mathbf{1}}$) to the same return channel; this channel reuse at different types amounts to a form of *strong update*, a degree of flexibility not supported by [YHB07, HR04].

Second, the equivalence reasoning mechanisms used in [YHB07, HR04] would be substantially limiting for processes with channel reuse. More specifically, consider the slightly tweaked client implementation of C_2 below:

$$C_2' \triangleq \mathtt{rec}\, w.\mathtt{alloc}\, x.(\mathtt{srv}_1!x \parallel x?y.(\mathtt{srv}_2!x \parallel x?z.\mathtt{free}\, x.\mathtt{c}!(y,z).X)) \tag{9}$$

The only difference between the client in (9) and the original one in (2) is that C_2 *sequences* the service requests before the service inputs, *i.e.,* $\ldots \mathtt{srv}_1!x.\, x?y.\ldots$ and $\ldots \mathtt{srv}_2!x.\, x?z.\ldots$, whereas C_2' parallelises them, *i.e.,* $\ldots \mathtt{srv}_1!x \parallel x?y.\ldots$ and $\ldots \mathtt{srv}_2!x \parallel x?z.\ldots$. It turns out that the two client implementations exhibit the same behaviour: the return channel used by both clients is private, *i.e.,* newly allocated, and as a result the servers cannot answer the service on that channel before it is receives it on either $\mathtt{srv}_1$ or $\mathtt{srv}_2$[1]. Through *scope extrusion*, [YHB07, HR04] can reason adequately about the first server interaction, and relate $\ldots \mathtt{srv}_1!x.\, x?y.\ldots$ of C_2 with $\ldots \mathtt{srv}_1!x \parallel x?y.\ldots$ of C_2. However, they have no mechanism for tracking channel locality post scope extrusion, thereby recovering the information that the return channel *becomes private again* to the client after the first server interaction. This prohibits [YHB07, HR04] from determining that the second server interaction is just an instance of the first server interaction thus failing to relate these two implementations.

In [DFH12] we developed a substructural type system based around a type attribute describing channel *uniqueness*, and this was used to statically ensure safe computations for $R\pi$. In this work, we weave this type information into our framework, imbuing it with an operational permission-semantics to reason compositionally about the costed behaviour of (safe) process. More specifically, in (2), when C_2 allocates channel x, no other process knows about x: from a typing perspective, but also operationally, x is *unique* to C_2. Client C_2 then sends x on $\mathtt{srv}_1$ at an *affine* type, which (by definition) limits the server to use x at most once. At this point, from an operational perspective, x is to C_2, the entity previously "owning" it, *unique-after-1* (communication) use. This means that after one communication step on x, (the derivative of) C_2 recognises that all the other processes apart from it must have used up the single affine permission for x, and hence x becomes once again *unique* to C_2. This also means that C_2 can safely *reuse* x, possibly at a different object type (strong update), or else safely deallocate it.

The concept of affinity is well-known in the process calculus community. By contrast, uniqueness and its duality to affinity is much less used. In a compositional framework, uniqueness can be used to record the guarantee at one end of a channel corresponding to the restriction associated with affine channel usage at the other; an operation semantics can be defined, tracking the *permission transfer* of affine permissions back and forth between processes as a result of communication, addressing the aforementioned complications associated with idioms such as channel reuse. We employ such an operational (costed) semantics to define our efficiency preorders for concurrent processes with explicit resource management, based around the notion of amortised cost discussed above.

[1] Analogously, in the π-calculus, $\mathtt{new}\, d.(c!d \parallel d?x.P)$ is indistinguishable from $\mathtt{new}\, d.(c!d.d?x.P)$

$$
\begin{array}{lllll}
P, Q ::= & u!\vec{v}.P & \textit{(output)} & \mid\ u?\vec{x}.P & \textit{(input)} \\
\mid & \texttt{nil} & \textit{(nil)} & \mid\ \texttt{if}\ u = v\ \texttt{then}\ P\ \texttt{else}\ Q & \textit{(match)} \\
\mid & \texttt{rec}\, w.P & \textit{(recursion)} & \mid\ x & \textit{(process variable)} \\
\mid & P \parallel Q & \textit{(parallel)} & \mid\ \texttt{alloc}\, x.P & \textit{(allocate)} \\
\mid & \texttt{free}\, u.P & \textit{(deallocate)} &
\end{array}
$$

Figure 1: $R\pi$ Syntax

1.3 Paper Structure

Section 2 introduces our language with constructs for explicit memory management and defines a costed semantics for it. Section 3 develops a labelled-transition system for our language that takes into consideration some representation of the observer and the permissions that are exchanged between the program and the observer. Based on this transition system, the section also defines a coinductive cost-based preorder and proves a number of properties about it. Section 4 justifies the cost-based preorder by relating it with a behavioural contextual preorder that is defined in terms of the reduction semantics of Section 2. Section 5 applies the theory of Section 3 to reason about the efficiency of two implementations of an unbounded buffer. Finally, Section 6 surveys related work and Section 7 concludes.

2 The Language

Fig. 1 shows the syntax for our language, the resource π-calculus, or $R\pi$ for short. It has the standard π-calculus constructs with the exception of scoping, which is replaced with primitives for explicit channel allocation, $\texttt{alloc}\, x.P$, and deallocation, $\texttt{free}\, x.P$. The syntax assumes two separate denumerable sets of channel names $c, d \in \textsc{Chan}$, and variables $x, y, z, w \in \textsc{Var}$, and lets identifiers u, v range over both sets, $\textsc{Chan} \cup \textsc{Var}$. The input construct, $c?x.\mathrm{P}$, recursion construct, $\texttt{rec}\, w.P$, and channel allocation construct, $\texttt{alloc}\, x.P$, are binders whereby free occurrences of the variable x in P are bound.

$R\pi$ processes run in a resource environment, ranged over by M, N, denoting predicates over channel names stating whether a channel is allocated or not. We find it convenient to represent this function as a list of channels representing the set channels that are allocated, *e.g.,* the list c, d denotes the set $\{c, d\}$, representing the resource environment returning *true* for channels c and d and *false* otherwise - in this representation, the order of the channels in the list is, unimportant, but duplicate channels are disallowed; as shorthand, we also write M, c to denote $M \cup \{c\}$ whenever $c \notin M$.

In this paper we consider only resource environments with an *infinite* number of deallocated channels, *i.e., M* is a total function. Models with finite resources can be easily accommodated by making M partial; this also would entail a slight change in the semantics of the allocation construct, which could either block or fail whenever there are no deallocated resources left. Although interesting in its own right, we focus on settings with infinite resources as it lends itself better to the analysis of

Contexts

$$C \quad ::= \quad [-] \quad | \quad C \parallel P \quad | \quad P \parallel C$$

Structural Equivalence

$$\textsc{sCom} \quad P \parallel Q \equiv Q \parallel P \qquad \textsc{sAss} \quad P \parallel (Q \parallel R) \equiv (P \parallel Q) \parallel R \qquad \textsc{sNil} \quad P \parallel \mathtt{nil} \equiv P$$

Reduction Rules

$$\frac{}{M, c \triangleright c!\vec{d}.P \parallel c?\vec{x}.Q \longrightarrow_0 M, c \triangleright P \parallel Q\{\vec{d}/\vec{x}\}}\ \textsc{rCom}$$

$$\frac{}{M, c \triangleright \mathtt{if}\ c = c\ \mathtt{then}\ P\ \mathtt{else}\ Q \longrightarrow_0 M, c \triangleright P}\ \textsc{rThen}$$

$$\frac{}{M, c, d \triangleright \mathtt{if}\ c = d\ \mathtt{then}\ P\ \mathtt{else}\ Q \longrightarrow_0 M, c, d \triangleright Q}\ \textsc{rElse}$$

$$\frac{}{M \triangleright \mathtt{rec}\ w.P \longrightarrow_0 M \triangleright P\{\mathtt{rec}\ w.P/w\}}\ \textsc{rRec} \qquad \frac{P \equiv P' \quad M \triangleright P' \longrightarrow_c M \triangleright Q' \quad Q' \equiv Q}{M \triangleright P \longrightarrow_c M \triangleright Q}\ \textsc{rStr}$$

$$\frac{}{M \triangleright \mathtt{alloc}\ x.P \longrightarrow_{+1} M, c \triangleright P\{c/x\}}\ \textsc{rAll} \qquad \frac{}{M, c \triangleright \mathtt{free}\ c.P \longrightarrow_{-1} M \triangleright P}\ \textsc{rFree}$$

Reflexive Transitive Closure

$$\frac{}{M \triangleright P \longrightarrow_0^* M \triangleright P} \qquad \frac{M \triangleright P \longrightarrow_k^* M' \triangleright P' \qquad M' \triangleright P' \longrightarrow_l M'' \triangleright P''}{M \triangleright P \longrightarrow_{k+l}^* M'' \triangleright P''}$$

Figure 2: $R\pi$ Reduction Semantics

resource efficiency that follows.

We refer to a pair $M \triangleright P$ of a resource environment and a process as a *system*. The reduction relation is defined as the least *contextual* relation over systems satisfying the rules in Fig. 2; contexts consist of parallel composition of processes (see Fig. 2). Rule (rStr) extends reductions to structurally equivalent processes, $P \equiv Q$, *i.e.,* processes that are identified up to superfluous `nil` processes, and commutativity/associativity of parallel composition (see Fig. 2).

Most rules follow those of the standard π-calculus, *e.g.,* (rRec), with the exception of those which involve resource handling. For instance, the rule for communication (rCom) requires the communicating channel to be allocated. Allocation (rAll) chooses a deallocated channel, allocates it, and substitutes it for the bound variable of the allocation construct.[2] Deallocation (rFree) changes the states of a channel from allocated to deallocated, making it available for future allocations. The rules are annotated with a cost reflecting resource usage; allocation has a cost of +1, deallocation has a (negative) cost of −1 while the other reductions carry no cost, *i.e.,* 0. Fig. 2 also shows the natural definition of the reflexive transitive closure of the costed reduction relation. In what follows, we use $k, l \in \mathbb{Z}$ as integer metavariables to range over costs.

Example 1. *The following reduction sequence illustrates potential unwanted behaviour resulting from resource mismanagement:*

$$M, c \triangleright \texttt{free}\, c.(c!1 \parallel c?x.P) \parallel \texttt{alloc}\, y.(y!42 \parallel y?z.Q) \longrightarrow_{-1} \quad (10)$$

$$M \quad \triangleright c!1 \parallel c?x.P \parallel \texttt{alloc}\, y.(x!42 \parallel x?z.Q) \longrightarrow_{+1} \quad (11)$$

$$M, c \triangleright c!1 \parallel c?x.P \parallel c!42 \parallel c?z.Q \quad (12)$$

Intuitively, allocation should yield "fresh" channels i.e., *channels that are not in use by any active process. This assumption is used by the right process in system* (10)*,* $\texttt{alloc}\, y.(y!42 \parallel y?z.Q)$*, to carry out a* local *communication, sending the value* `42` *on some local channel* y *that no other process is using. However, the premature deallocation of the channel* c *by the left process in* (10)*,* $\texttt{free}\, c.(c!1 \parallel c?x.P)$*, allows channel* c *to be reallocated by the right process in the subsequent reduction,* (11)*. This may then lead to unintended behaviour since we may end up with interferences when communicating on* c *in the residuals of the left and right processes,* (12)*.*[3] □

In [DFH12] we defined a type system that precludes unwanted behaviour such as in Ex. 1. The type syntax is shown in Fig. 3. The main type entities are *channel types*, denoted as $[\vec{\mathbf{U}}]^a$, where *type attributes* a range over

- $\mathbf{1}$, for affine, imposing a restriction/obligation on usage;
- $(\bullet, i)$, for unique-after-i usages ($i \in \mathbb{N}$), providing guarantees on usage;
- ω, for unrestricted channel usage without restrictions or guarantees.

[2] The expected side-condition $c \notin M$ is implicit in the notation (M, c) used in the system $M, c \triangleright P\{^c/_x\}$ that is reduced to.

[3] Operationally, we do not describe erroneous computation that may result from attempted communications on deallocated channels. This may occur after reduction (10), if the residual of the left process communicate on channel c.

$$
\begin{array}{llllllll}
a & ::= & \omega & \text{(unrestricted)} & \mid \mathbf{1} & \text{(affine)} & \mid (\bullet, i) & \text{(unique after } i \text{ steps)} \\
\mathbf{T} & ::= & \mathbf{U} & \text{(channel type)} & \mid \mathbf{proc} & \text{(process type)} & & \\
\mathbf{U} & ::= & [\vec{\mathbf{U}}]^a & \text{(channel)} & \mid \mu X.\mathbf{U} & \text{(recursion)} & \mid X & \text{(variable)}
\end{array}
$$

Figure 3: Type Attributes and Types

Uniqueness typing can be seen as dual to affine typing [Har06], and in [DFH12] we make use of this duality to keep track of uniqueness across channel-passing parallel processes: an attribute $(\bullet, i)$ typing a endpoint of a channel c accounts for (at most) i instances of affine attributes typing endpoints of that same channel.

A channel type $[\vec{\mathbf{U}}]^a$ also describes the type of the values that can be communicated on that channel, $\vec{\mathbf{U}}$, which denotes a list of types $\mathbf{U}_1, \ldots, \mathbf{U}_n$ for $n \in \text{Nat}$; when $n = 0$, the type list is an empty list and we simply write $[]^a$. Note the difference between $[\vec{\mathbf{U}}]^{\mathbf{1}}$, *i.e.*, a channel with an affine usage restriction, and $[\vec{\mathbf{U}}]^{(\bullet,1)}$, *i.e.*, a channel with a unique-after-1 usage guarantee. We denote fully unique channels as $[\vec{\mathbf{U}}]^{\bullet}$ in lieu of $[\vec{\mathbf{U}}]^{(\bullet,0)}$.

The type syntax also assumes a denumerable set of type variables X, Y, bound by the recursive type construct $\mu X.\mathbf{U}$. In what follows, we restrict our attention to *closed, contractive* types, where every type variable is bound and appears within a channel constructor $[-]^a$; this ensures that channel types such as $\mu X.X$ are avoided. We assume an equi-recursive interpretation for our recursive types [Pie02] (see tEq in Fig. 4), characterised as the least type-congruence satisfying rule eRec in Fig. 4.

$$
\frac{\Gamma \vdash P \qquad d\text{om}(\Gamma) \subseteq M \qquad \Gamma \text{ is consistent}}{\Gamma \vdash M \triangleright P}\ \text{tSys}
$$

The rules for typing processes are given in Fig. 4 and take the usual shape $\Gamma \vdash P$ stating that process P is well-typed with respect to the environment Γ. Systems are typed according to (tSys) above: a system $M \triangleright P$ is well-typed under Γ if P is well-typed *wrt.* Γ, $\Gamma \vdash P$, and Γ only contains assumptions for channels that have been allocated, $d\text{om}(\Gamma) \subseteq M$. This restricts channel usage in P to allocated channels and is key for ensuring safety. In [DFH12], typing environments are multisets of pairs of identifiers and types; we do not require them to be partial functions. However, the (top-level) typing rule for systems (tSys) requires that the typing environment is *consistent*. A typing environment is consistent if whenever it contains multiple assumptions about a channel, then these assumptions can be derived from a single assumption using the structural rules of the type system (see the structural rule tCon and the splitting rule pUnq in Fig. 4). Intuitively, the consistency condition ensures that there is no mismatch in the duality between the guarantees of unique types and the restrictions of affine types, which allows sound compositional type-checking by our type system. For instance, consistency rules out environments such as

$$
c : [\mathbf{U}]^{\bullet}, c : [\mathbf{U}]^{\mathbf{1}} \tag{13}
$$

Logical rules

$$\frac{\Gamma, u:[\vec{\mathbf{T}}]^{a-1} \vdash P}{\Gamma, u:[\vec{\mathbf{T}}]^{a}, \overrightarrow{v:\mathbf{T}} \vdash u!\vec{v}.P}\ \textsc{tOut} \qquad \frac{\Gamma, u:[\vec{\mathbf{T}}]^{a-1}, \overrightarrow{x:\mathbf{T}} \vdash P}{\Gamma, u:[\vec{\mathbf{T}}]^{a} \vdash u?\vec{x}.P}\ \textsc{tIn} \qquad \frac{\Gamma_1 \vdash P \quad \Gamma_2 \vdash Q}{\Gamma_1, \Gamma_2 \vdash P \parallel Q}\ \textsc{tPar}$$

$$\frac{u, v \in \Gamma \quad \Gamma \vdash P \quad \Gamma \vdash Q}{\Gamma \vdash \texttt{if}\ u = v\ \texttt{then}\ P\ \texttt{else}\ Q}\ \textsc{tIf} \qquad \frac{\Gamma^{\omega}, x:\mathbf{proc} \vdash P}{\Gamma^{\omega} \vdash \texttt{rec}\ w.P}\ \textsc{tRec} \qquad \frac{}{x:\mathbf{proc} \vdash x}\ \textsc{tVar}$$

$$\frac{\Gamma \vdash P}{\Gamma, u:[\vec{\mathbf{T}}]^{\bullet} \vdash \texttt{free}\ u.P}\ \textsc{tFree} \qquad \frac{\Gamma, x:[\vec{\mathbf{T}}]^{\bullet} \vdash P}{\Gamma \vdash \texttt{alloc}\ x.P}\ \textsc{tAll} \qquad \frac{}{\emptyset \vdash \texttt{nil}}\ \textsc{tNil}$$

$$\frac{\Gamma' \vdash P \qquad \Gamma \prec \Gamma'}{\Gamma \vdash P}\ \textsc{tStr}$$

where Γ^{ω} can only contain unrestricted assumptions and all bound variables are fresh.

Structural rules ($\prec$) is the least reflexive transitive relation satisfying

$$\frac{\mathbf{T} = \mathbf{T}_1 \circ \mathbf{T}_2}{\Gamma, u:\mathbf{T} \prec \Gamma, u:\mathbf{T}_1, u:\mathbf{T}_2}\ \textsc{tCon} \qquad \frac{\mathbf{T} = \mathbf{T}_1 \circ \mathbf{T}_2}{\Gamma, u:\mathbf{T}_1, u:\mathbf{T}_2 \prec \Gamma, u:\mathbf{T}}\ \textsc{tJoin} \qquad \frac{\mathbf{T}_1 \sim \mathbf{T}_2}{\Gamma, u:\mathbf{T}_1 \prec \Gamma, u:\mathbf{T}_2}\ \textsc{tEq}$$

$$\frac{}{\Gamma, u:\mathbf{T} \prec \Gamma}\ \textsc{tWeak} \qquad \frac{\mathbf{T}_1 \prec_s \mathbf{T}_2}{\Gamma, u:\mathbf{T}_1 \prec \Gamma, u:\mathbf{T}_2}\ \textsc{tSub} \qquad \frac{}{\Gamma, u:[\vec{\mathbf{T}}_1]^{\bullet} \prec \Gamma, u:[\vec{\mathbf{T}}_2]^{\bullet}}\ \textsc{tRev}$$

Equi-Recursion

$$\frac{}{\mu X.\mathbf{U} \sim \mathbf{U}\{\mu X.\mathbf{U}/X\}}\ \textsc{eRec}$$

Counting channel usage

$$c:[\vec{\mathbf{T}}]^{a-1} \stackrel{\text{def}}{=} \begin{cases} \varepsilon \quad \textit{(empty list)} & \text{if } a = \mathbf{1} \\ c:[\vec{\mathbf{T}}]^{\omega} & \text{if } a = \omega \\ c:[\vec{\mathbf{T}}]^{(\bullet, i)} & \text{if } a = (\bullet, i+1) \end{cases}$$

Type splitting

$$\frac{}{[\vec{\mathbf{T}}]^{\omega} = [\vec{\mathbf{T}}]^{\omega} \circ [\vec{\mathbf{T}}]^{\omega}}\ \textsc{pUnr} \qquad \frac{}{\mathbf{proc} = \mathbf{proc} \circ \mathbf{proc}}\ \textsc{pProc} \qquad \frac{}{[\vec{\mathbf{T}}]^{(\bullet,i)} = [\vec{\mathbf{T}}]^{\mathbf{1}} \circ [\vec{\mathbf{T}}]^{(\bullet,i+1)}}\ \textsc{pUnq}$$

Subtyping

$$\frac{}{(\bullet, i) \prec_s (\bullet, i+1)}\ \textsc{sIndx} \qquad \frac{}{(\bullet, i) \prec_s \omega}\ \textsc{sUnq} \qquad \frac{}{\omega \prec_s \mathbf{1}}\ \textsc{sAff} \qquad \frac{a_1 \prec_s a_2}{[\vec{\mathbf{T}}]^{a_1} \prec_s [\vec{\mathbf{T}}]^{a_2}}\ \textsc{sTyp}$$

Figure 4: Typing processes

where a process typed under the guarantee that a channel c is unique now, $c : [\mathbf{U}]^{\bullet}$, contradicts the fact that some other process may be typed under the affine usage allowed by the assumption $c : [\mathbf{U}]^{\mathbf{1}}$. For similar reasons, consistency also rules out environments such as

$$c : [\mathbf{U}]^{\bullet}, c : [\mathbf{U}]^{\omega} \tag{14}$$

However, it does not rule out environments such as (15) even though the guarantee provided by $c : [\mathbf{U}]^{(\bullet,2)}$ is too conservative—it states that channel c will become unique after two uses, whereas it will in fact become unique after one use, since this would consume the only affine type assumption $c : [\mathbf{U}]^{\mathbf{1}}$.

$$c : [\mathbf{U}]^{(\bullet,2)}, c : [\mathbf{U}]^{\mathbf{1}} \tag{15}$$

Definition 1 (Consistency)**.** *A typing environment* Γ *is* consistent *if there is a partial map* Γ' *such that* $\Gamma' \prec \Gamma$.

The type system is *substructural*, implying that typing assumptions can be used *only once* during typechecking [Pie04]. This is clearly manifested in the output and input rules, TOUT and TIN in Fig. 4. In fact, rule TOUT collapses three output rules (listed below) into one, hiding discrepancies using the operation $c : [\vec{\mathbf{T}}]^{a-1}$ (see Fig. 4.)

$$\frac{\Gamma \vdash P}{\Gamma,\, u : [\vec{\mathbf{T}}]^{\mathbf{1}},\, \overrightarrow{v : \mathbf{T}} \;\vdash\; u!\vec{v}.P}\ \textsc{tOutA} \qquad \frac{\Gamma,\, u : [\vec{\mathbf{T}}]^{\omega} \vdash P}{\Gamma,\, u : [\vec{\mathbf{T}}]^{\omega},\, \overrightarrow{v : \mathbf{T}} \;\vdash\; u!\vec{v}.P}\ \textsc{tOutW}$$
$$\frac{\Gamma,\, u : [\vec{\mathbf{T}}]^{(\bullet,i)} \vdash P}{\Gamma,\, u : [\vec{\mathbf{T}}]^{(\bullet,i+1)},\, \overrightarrow{v : \mathbf{T}} \;\vdash\; u!\vec{v}.P}\ \textsc{tOutU} \tag{16}$$

Rule TOUTA states that an output of values $\vec{v}$ on channel u is allowed if the type environment has an *affine* channel-type assumption for that channel, $u : [\vec{\mathbf{T}}]^{\mathbf{1}}$, and the corresponding type assumptions for the values communicated, $\overrightarrow{v : \mathbf{T}}$, match the object type of the affine channel-type assumption, $\vec{\mathbf{T}}$; in the rule premise, the continuation P must also be typed *wrt.* the *remaining* assumptions in the environment, *without* the assumptions consumed by the conclusion. Rule TOUTW is similar, but permits outputs on u for environments with an *unrestricted* channel-type assumption for that channel, $u : [\vec{\mathbf{T}}]^{\omega}$. The continuation P is typechecked *wrt.* the remaining assumptions and a *new* assumption, $u : [\vec{\mathbf{T}}]^{\omega}$; this assumption is identical to the one consumed in the conclusion, so as to model the fact that uses of channel u are unrestricted. Rule TOUTU is again similar, but it allows outputs on channel u for a *"unique after* $i+1$*"* channel-type assumption; in the premise of the rule, P is typechecked *wrt.* the remaining assumptions and a *new* assumption $u : [\vec{\mathbf{T}}]^{(\bullet,i)}$, where u is now *unique after* i uses. Analogously, the input rule, TIN, also encodes three input cases (listed below):

$$\frac{\Gamma, \overrightarrow{x : \mathbf{T}} \vdash P}{\Gamma, u : [\vec{\mathbf{T}}]^{\mathbf{1}} \vdash u?\vec{x}.P}\ \textsc{tInO} \qquad \frac{\Gamma, u : [\vec{\mathbf{T}}]^{\omega}, \overrightarrow{x : \mathbf{T}} \vdash P}{\Gamma, u : [\vec{\mathbf{T}}]^{\omega} \vdash u?\vec{x}.P}\ \textsc{tInW} \qquad \frac{\Gamma, u : [\vec{\mathbf{T}}]^{(\bullet,i)}, \overrightarrow{x : \mathbf{T}} \vdash P}{\Gamma, u : [\vec{\mathbf{T}}]^{(\bullet,i+1)} \vdash u?\vec{x}.P}\ \textsc{tInU} \tag{17}$$

Parallel composition (TPAR) enforces the substructural treatment of type assumptions, by ensuring that type assumptions are used by either the left process or the right, but not by both. However,

some type assumption can be *split* using contraction, *i.e.*, rules (tStr) and (tCon). For example, an assumption $c : [\vec{\mathbf{T}}]^{(\bullet,i)}$ can be split as $c : [\vec{\mathbf{T}}]^{\mathbf{1}}$ and $c : [\vec{\mathbf{T}}]^{(\bullet,i+1)}$—see (pUnq).

The rest of the rules are fairly straightforward. The consistency requirement of the type system ensures that whenever a process is typed *wrt.* a unique assumption for a channel, $[\vec{\mathbf{T}}]^{\bullet}$, no other process has access to that channel. It can therefore safely deallocate it (tFree), or change the object type of the channel (tRev). Dually, when a channel is newly allocated it is assumed unique (tAll). Note also that name matching is only permitted when channel permissions are owned, $u, v \in \Gamma$ in (tIf). Uniqueness can therefore also be thought of as "freshness", a claim we substantiate further when we discuss bisimulation (Sect. 3.2).

In [DFH12] we prove the usual subject reduction and progress lemmas for this type system, given an (obvious) error relation.

3 A Cost-Based Preorder

We define our cost-based preorder as a bisimulation relation that relates two systems $M \triangleright P$ and $N \triangleright Q$ whenever they have equivalent behaviour and when, in addition, $M \triangleright P$ is more efficient than $N \triangleright Q$. We are interested in reasoning about *safe* computations, aided by the type system described in Section 2. For this reason, we limit our analysis to instances of $M \triangleright P$ and $N \triangleright Q$ that are *well-typed*, as defined in [DFH12]. In order to preserve safety, we also need to reason under the assumption of safe contexts. Again, we employ the type system described in Section 2 and characterise the (safe) context through a type environment that typechecks it, Γ_{obs}. Thus our bisimulation relations take the form of a typed relation, indexed by type environments [HR04]:

$$\Gamma_{obs} \models (M \triangleright P)\ \mathcal{R}\ (N \triangleright Q) \tag{18}$$

Behavioural reasoning for safe systems is achieved by ensuring that the overall type environment $(\Gamma_{sys}, \Gamma_{obs})$, consisting of the environment typing $M \triangleright P$ and $N \triangleright Q$, say Γ_{sys}, and the observer environment Γ_{obs}, is *consistent* according to Definition 1. This means that there exists a global environment, Γ_{global}, which can be decomposed into Γ_{obs} and Γ_{sys}; it also means that the observer process, which is universally quantified by our semantic interpretation (18), typechecks when composed in parallel with P, *resp.* Q (see tPar of Fig. 4).

There is one other complication worth highlighting about (18): although both systems $M \triangleright P$ and $N \triangleright Q$ are related *wrt.* the same *observer*, Γ_{obs}, they can each be typed under *different* typing environments. For instance, consider the two clients C_0 and C_1 we would like to relate from the introduction:

$$\begin{aligned} \mathrm{C}_0 &\triangleq \mathtt{rec}\, w.\ \mathtt{alloc}\, x_1.\mathtt{alloc}\, x_2.\ \mathtt{srv}_1!x_1.\ x_1?y.\mathtt{srv}_2!x_2.\ x_2?z.\mathtt{c}!(y,z).w \\ \mathrm{C}_1 &\triangleq \mathtt{rec}\, w.\ \mathtt{alloc}\, x.\ \mathtt{srv}_1!x.\ x?y.\mathtt{srv}_2!x.\ x?z.\mathtt{c}!(y,z).w \end{aligned} \tag{19}$$

Even though, initially, they may be typed by the same type environment, after a few steps, the derivatives of C_0 and C_1 must be typed under different typing environments, because C_0 allocates

two channels, while C_1 only allocates a single channel. Our typed relations allows for this by *existentially quantifying* over the type environments typing the respective systems. All this is achieved indirectly through the use of *configurations*.

Definition 2 (Configuration)**.** *The triple $\Gamma \triangleleft M \triangleright P$ is a configuration if and only if $dom(\Gamma) \subseteq M$ and there exist some Δ such that (Γ, Δ) is consistent and $\Delta \vdash M \triangleright P$.*

Note that, in a configuration $\Gamma \triangleleft M \triangleright P$:

- $c \in (dom(\Gamma) \cup names(P))$ implies $c \in M$.
- $c \in M$ and $c \notin (dom(\Gamma) \cup names(P))$ denotes a resource leak for channel c.
- $c \notin dom(\Gamma)$ implies that channel c is not known to the observer.

Definition 3 (Typed Relation)**.** *A type-indexed relation $\mathcal{R}$ relates systems under a observer characterized by a context Γ; we write*

$$\Gamma \vDash M \triangleright P \; \mathcal{R} \; N \triangleright Q$$

if $\mathcal{R}$ relates $\Gamma \triangleleft M \triangleright P$ and $\Gamma \triangleleft N \triangleright Q$, and both $\Gamma \triangleleft M \triangleright P$ and $\Gamma \triangleleft N \triangleright Q$ are configurations.

3.1 Labelled Transition System

In order to be able to reason coinductively over our typed relations, we define a labelled transition system (LTS) over configurations. Apart from describing the behaviour of the system $M \triangleright P$ in a configuration $\Gamma \triangleleft M \triangleright P$, the LTS also models interactions between the system and the observer characterised by Γ. Our LTS is also *costed*, assigning a cost to each form of transition.

The costed LTS, denoted as $\xrightarrow{\mu}_k$, is defined in Fig. 5, in terms of a top-level rule, lREN, and a pre-LTS, denoted as $\xrightarrow{\mu}_k$. The rule lREN allows us to rename channels for transitions derived in the pre-LTS, as long as this renaming is invisible to the observer, and is comparable to alpha-renaming of scoped bound names in the standard π-calculus. It relies on the renaming-modulo (observer) type environments given in Definition 4.

Definition 4 (Renaming Modulo Γ)**.** *Let σ_Γ : NAME $\mapsto$ NAME range over bijective name substitutions satisfying the constraint that $c \in dom(\Gamma)$ implies $c\sigma_\Gamma = c\sigma_\Gamma^{-1} = c$.*

The renaming introduced by lREN allows us to relate clients the C_0 and C_1 of (19) *wrt.* an observer environment such as $\mathtt{srv}_1 : [[\mathbf{T}_1]^{\mathbf{1}}]^\omega, \mathtt{srv}_2 : [[\mathbf{T}_2]^{\mathbf{1}}]^\omega$ of (6) and some appropriate common M, even when, after the initial channel allocations, the two clients communicate potentially different (newly allocated) channels on $\mathtt{srv}_1$. The rule is particularly useful when, later on, we need to also match the output of a new allocated channel on $\mathtt{srv}_2$ from C_0 with the output on the previously allocated channel from C_1 on $\mathtt{srv}_2$. The renaming-modulo observer environments function can be used for C_1 at that stage — even though the client reuses a channel previously communicated to the observer — because the respective observer information relating to that channel is lost, *i.e.*, it is not in the

domain of the observer environment; see discussion for lOut and lIn below for an explanation of how observers loose information. This mechanism differs from standard scope-extrusion techniques for π-calculus which assume that, once a name has been extruded, it remains forever known to the observer. As a result, there are more opportunities for renaming in our calculus than there are in the standard π-calculus.

To limit itself to safe interactions, the (pre-)LTS must be able to reason compositionally about resource usage between the process, P, and the observer, Γ. We therefore imbue our type assumptions from Sec. 2 with a *permission semantics*, in the style of [TA08, FRS11]. Under this interpretation, type assumptions constitute *permissions* describing the respective usage of resources. Permissions are woven into the behaviour of configurations giving them an *operational* role: they may either restrict usage or privilege processes to use resources in special ways. In a configuration, the observer and the process each *own* a set of permissions and may *transfer* them to one another during communication. The consistency requirement of a configuration ensures that the guarantees given by permissions owned by the observer are not in conflict with those given by permissions owned by the configuration process, and viceversa.

To understand how the pre-LTS deals with permission transfer and compositional resource usage, consider the rule for output, (lOut). Since we employ the type system of Sec. 2 to ensure safety, this rule models the typing rule for output (tOut) on the part of the process, and the typing rule for input (tIn) on the part of the observer. Thus, apart from describing the communication of values $\vec{d}$ from the configuration process to the observer on channel c, it also captures permission transfer between the two parties, mirroring the type assumption usage in tOut and tIn. More specifically, rule (lOut) employs the operation $c:[\vec{\mathbf{T}}]^{a-1}$ of Fig. 4 so as to concisely describe the three variants of the output rule:

$$\frac{}{\Gamma, c:[\vec{\mathbf{T}}]^{(\bullet,i+1)} \triangleleft M \triangleright c!\vec{d}.P \xrightarrow{c!\vec{d}}_0 \Gamma, c:[\vec{\mathbf{T}}]^{(\bullet,i)}, \vec{d}:\vec{\mathbf{T}} \triangleleft M \triangleright P}\ \textsc{lOutU}$$

$$\frac{}{\Gamma, c:[\vec{\mathbf{T}}]^{\mathbf{1}} \triangleleft M \triangleright c!\vec{d}.P \xrightarrow{c!\vec{d}}_0 \Gamma, \vec{d}:\vec{\mathbf{T}} \triangleleft M \triangleright P}\ \textsc{lOutA} \tag{20}$$

$$\frac{}{\Gamma, c:[\vec{\mathbf{T}}]^{\omega} \triangleleft M \triangleright c!\vec{d}.P \xrightarrow{c!\vec{d}}_0 \Gamma, c:[\vec{\mathbf{T}}]^{\omega}, \vec{d}:\vec{\mathbf{T}} \triangleleft M \triangleright P}\ \textsc{lOutW}$$

The first output rule variant, lOutU, deals with the case where the observer owns a unique-after-$(i+1)$ permission for channel c. Def. 2 implies that the process in the configuration is well-typed (*wrt.* some environment) and, since the process is in a position to output on channel c, rule tOut must have been used to type it. This typing rule, in turn, states that the type assumptions relating to the values communicated, $\vec{d}:\vec{\mathbf{T}}$, must have been owned by the process and consumed by the output operation. Dually, since the observer is capable of inputting on c, rule tIn must have been used to type it,[4] which states that the continuation (after the input) assumes the use the assumptions $\vec{d}:\vec{\mathbf{T}}$. Rule lOutU models these two usages operationally as the *explicit transfer* of the permissions $\vec{d}:\vec{\mathbf{T}}$ from the process to the observer.

[4] More specifically, tInU of (17).

Costed Transitions and pre-Transitions

$$\frac{\Gamma \triangleleft (M \triangleright P)\sigma_\Gamma \xrightarrow{\mu}_k \Gamma' \triangleleft M' \triangleright P'}{\Gamma \triangleleft M \triangleright P \xrightarrow{\mu}_k \Gamma' \triangleleft M' \triangleright P'}\ \textsc{lRen}$$

$$\frac{}{\Gamma, c:[\vec{\mathbf{T}}]^a \triangleleft M \triangleright c!\vec{d}.P \xrightarrow{c!\vec{d}}_0 \Gamma, c:[\vec{\mathbf{T}}]^{a-1}, \vec{d}:\vec{\mathbf{T}} \triangleleft M \triangleright P}\ \textsc{lOut}$$

$$\frac{}{\Gamma, c:[\vec{\mathbf{T}}]^a, \vec{d}:\vec{\mathbf{T}} \triangleleft M \triangleright c?\vec{x}.P \xrightarrow{c?\vec{d}}_0 \Gamma, c:[\vec{\mathbf{T}}]^{a-1} \triangleleft M \triangleright P\{\vec{d}/\vec{x}\}}\ \textsc{lIn}$$

$$\frac{\Gamma_1 \triangleleft M \triangleright P \xrightarrow{c!\vec{d}}_0 \Gamma_1' \triangleleft M \triangleright P' \qquad \Gamma_2 \triangleleft M \triangleright Q \xrightarrow{c?\vec{d}}_0 \Gamma_2' \triangleleft M \triangleright Q'}{\Gamma \triangleleft M \triangleright P \parallel Q \xrightarrow{\tau}_0 \Gamma \triangleleft M \triangleright P' \parallel Q'}\ \textsc{lCom-L}$$

$$\frac{\Gamma \triangleleft M \triangleright P \xrightarrow{\mu}_k \Gamma' \triangleleft M' \triangleright P'}{\Gamma \triangleleft M \triangleright P \parallel Q \xrightarrow{\mu}_k \Gamma' \triangleleft M' \triangleright P' \parallel Q}\ \textsc{lPar-L}$$

$$\frac{\Gamma < \Gamma'}{\Gamma \triangleleft M \triangleright P \xrightarrow{\text{env}}_0 \Gamma' \triangleleft M \triangleright P}\ \textsc{lStr} \qquad \frac{}{\Gamma \triangleleft M \triangleright \texttt{rec}\, w.P \xrightarrow{\tau}_0 \Gamma \triangleleft M \triangleright P\{\texttt{rec}\, w.P/w\}}\ \textsc{lRec}$$

$$\frac{}{\Gamma \triangleleft M, c \triangleright \texttt{if}\ c = c\ \texttt{then}\ P\ \texttt{else}\ Q \xrightarrow{\tau}_0 \Gamma \triangleleft M \triangleright P}\ \textsc{lThen}$$

$$\frac{}{\Gamma \triangleleft M, c, d \triangleright \texttt{if}\ c = d\ \texttt{then}\ P\ \texttt{else}\ Q \xrightarrow{\tau}_0 \Gamma \triangleleft M \triangleright Q}\ \textsc{lElse}$$

$$\frac{}{\Gamma \triangleleft M \triangleright \texttt{alloc}\, x.P \xrightarrow{\tau}_{+1} \Gamma \triangleleft M, c \triangleright P\{c/x\}}\ \textsc{lAll} \qquad \frac{}{\Gamma \triangleleft M \triangleright P \xrightarrow{\texttt{alloc}}_{+1} \Gamma, c:[\vec{\mathbf{T}}]^\bullet \triangleleft M, c \triangleright P}\ \textsc{lAllE}$$

$$\frac{}{\Gamma \triangleleft M, c \triangleright \texttt{free}\, c.P \xrightarrow{\tau}_{-1} \Gamma \triangleleft M \triangleright P}\ \textsc{lFree} \qquad \frac{}{\Gamma, c:[\mathbf{T}]^\bullet \triangleleft M, c \triangleright P \xrightarrow{\texttt{free}\, c}_{-1} \Gamma \triangleleft M \triangleright P}\ \textsc{lFreeE}$$

Weak (Cost-Accumulating) Transitions

$$\frac{\Gamma \triangleleft M \triangleright P \xrightarrow{\mu}_k \Delta \triangleleft N \triangleright Q}{\Gamma \triangleleft M \triangleright P \overset{\mu}{\Rightarrow}_k \Delta \triangleleft N \triangleright Q} \qquad \frac{\Gamma \triangleleft M \triangleright P \xrightarrow{\tau}_l \Gamma' \triangleleft M' \triangleright P \overset{\mu}{\Rightarrow}_k \Gamma'' \triangleleft N \triangleright Q}{\Gamma \triangleleft M \triangleright P \overset{\mu}{\Rightarrow}_{(l+k)} \Gamma'' \triangleleft N \triangleright Q}$$

$$\frac{\Gamma \triangleleft M \triangleright P \overset{\mu}{\Rightarrow}_l \Gamma' \triangleleft M' \triangleright P \xrightarrow{\tau}_k \Gamma'' \triangleleft N \triangleright Q}{\Gamma \triangleleft M \triangleright P \overset{\mu}{\Rightarrow}_{(l+k)} \Gamma'' \triangleleft N \triangleright Q}$$

Figure 5: LTS Process Moves

The rule also models the *implicit transfer* of permissions between the observer and the output process. More precisely, Def. 2 requires that the process is typed *wrt.* an environment that *does not conflict with* the observer environment, which implies that the process environment must have (necessarily) used an affine permission, $c : [\vec{\mathbf{T}}]^{\mathbf{1}}$, for outputting on channel c,[5] since any other type of permission would conflict with the unique-after-$(i+1)$ permission for channel c owned by the observer. Moreover, through the guarantee given by the permission used, $c : [\vec{\mathbf{T}}]^{(\bullet,i+1)}$, the observer knows that after the communication it is one step closer towards gaining exclusive permission for channel c. Rule lOutU models this as the (implicit) transfer of the affine permission $c : [\vec{\mathbf{T}}]^{\mathbf{1}}$ from the process to the observer, updating the observer's permission for c to $[\vec{\mathbf{T}}]^{(\bullet,i)}$—two permissions $c : [\vec{\mathbf{T}}]^{(\bullet,i+1)}, c : [\vec{\mathbf{T}}]^{\mathbf{1}}$ can be consolidated as $c : [\vec{\mathbf{T}}]^{(\bullet,i)}$ using the structural rules tJoin and pUnq of Fig. 4.

The second output rule variant of (20), lOutA, is similar to the first when modelling the explicit transfer of permissions $\vec{d} : \vec{\mathbf{T}}$ from the process to the observer. However, it describes a different implicit transfer of permissions, since the observer uses an affine permission to input from the configuration process on channel c. The rule caters for two possible subcases. In the first case, the process could have used a unique-after-$(i+1)$ permission when typed using tOut: this constitutes a dual case to that of rule lOutU, and the rule models the implicit transfer of the affine permission $c : [\vec{\mathbf{T}}]^{\mathbf{1}}$ in the *opposite* direction, *i.e.,* from the observer to the process. In the second case, the process could have used an affine or an unrestricted permission instead, which does not result in any implicit permission transfer, but merely the consumption of affine permissions. Since the environment on the process side is existentially quantified in a configuration, this difference is abstracted away and the two subcases are handled by the same rule variant. Note that, in the extreme case where the observer affine permission is the only one relating to channel c, the observer loses all knowledge of channel c.

The explicit permission transfer for lOutW of (20), is identical to the other two rule variants. The use of an unrestricted permission for c from the part of the observer, $c : [\vec{\mathbf{T}}]^{\omega}$, implies that the output process could have either used an affine or an unrestricted permission—see (14). In either case, there is no implicit permission transfer involved. Moreover, the observer permission is not consumed since it is unrestricted.

The pre-LTS rule lIn can also be expanded into three rule variants, and models analogous permission transfer between the observer and the input process. Importantly, however, the explicit permission transfer described is in the opposite direction to that of lOut, namely from the observer to the input process. As in the case of lOutA of (20), the permission transfer from the observer to the input process may result in the observer losing all knowledge relating to the channels communicated, $\vec{d}$.

In order to allow an internal communication step through either lCom-L, or its dual lCom-R (elided), the left process should be considered to be part of the "observer" of the right process, and vice versa. However, it is not necessary to be quite so precise; we can follow [Hen08] and consider an arbitrary observer instead. This is sufficient to enable the communication action, and we do not loose any information since an internal communication step within the process does not affect the (global) observer.

[5] This implies that tOutA of (16) was used when typing the process

In our LTS, both the process (lAll, lFree) and the observer (lAllE, lFreeE) can allocate and deallocate memory. Finally, since the observer is modelled exclusively by the permissions it owns, we must allow the observer to split these permissions when necessary (lStr). The only rules that may alter the observer environment are those corresponding to external actions *i.e.,* lIn, lOut, lAllE, lFreeE and lStr. The remaining axioms in the pre-LTS model reduction rules from Figure 2 and should be self-explanatory; note that, as in the reduction semantics, the only actions to carry a cost are allocation and deallocation.

Technically, the pre-LTS is defined over triples Γ, M, P rather than configurations $\Gamma \triangleleft M \triangleright P$, but we can prove that the pre-LTS rules preserve the requirements for such triples to be configurations.

Lemma 1 (Transition and Structure)**.** $\Gamma \triangleleft M \triangleright P \xrightarrow{\mu}_k \Gamma_1 \triangleleft N \triangleright Q$ *and for* Δ *consistent* $\Delta \vdash M \triangleright P$ *implies the cases:*

$\mu = c!\vec{d}$**:** $M = N$, $k = 0$, $P \equiv c!\vec{d}.P_1 \parallel P_2$, $Q \equiv P_1 \parallel P_2$ *and* $\Gamma = (\Gamma', c : [\vec{\mathbf{T}}]^a)$, $\Gamma_1 = (\Gamma', c : [\vec{\mathbf{T}}]^{a-1}, d : \vec{\mathbf{T}})$ *and* $\Delta \prec (\Delta', c : [\vec{\mathbf{T}}]^b, d : \vec{\mathbf{T}})$, $(\Delta', c : [\vec{\mathbf{T}}]^{b-1}) \vdash Q$

$\mu = c?\vec{d}$**:** $M = N$, $k = 0$, $P \equiv c?\vec{x}.P_1 \parallel P_2$, $Q \equiv P_1\{\vec{d}/\vec{x}\} \parallel P_2$ *and* $\Gamma = (\Gamma', c : [\vec{\mathbf{T}}]^a, d : \vec{\mathbf{T}}), \Gamma_1 = (\Gamma', c : [\vec{\mathbf{T}}]^{a-1})$ *and* $\Delta \prec (\Delta', c : [\vec{\mathbf{T}}]^b)$, $(\Delta', c : [\vec{\mathbf{T}}]^{b-1}, d : \vec{\mathbf{T}}) \vdash Q$

$\mu = \tau$**:** *Either of three cases:*

- $M = N$, $k = 0$ *and* $\Gamma = \Gamma_1$ *and* $\Delta \vdash Q$ *or;*
- $M = M', c : \top$, $N = M', c : \bot$, $k = -1$ *and* $P \equiv \texttt{free}\, c.P_1 \parallel P_2$, $Q \equiv P_1 \parallel P_2$ *and* $\Gamma = \Gamma_1$ *and* $\Delta \prec \Delta', c : [\vec{\mathbf{T}}]^\bullet$ *and* $\Delta' \vdash Q$ *or;*
- $M = M', c : \bot$, $N = M', c : \top$, $k = +1$ *and* $P \equiv \texttt{alloc}\, x.P_1 \parallel P_2$, $Q \equiv P_1\{c/x\} \parallel P_2$ *and* $\Gamma = \Gamma_1$ *and* $\Delta \prec \Delta'$ *and* $\Delta', c : [\vec{\mathbf{T}}]^\bullet \vdash Q$

$\mu = \texttt{free}\, c$**:** $M = M', c : \top$, $N = M', c : \bot$, $k = -1$ *and* $\Gamma = \Gamma_1, c : [\vec{\mathbf{T}}]^\bullet$ *and* $P = Q$

$\mu = \texttt{alloc}$**:** $M = M', c : \bot$, $N = M', c : \top$, $k = +1$ *and* $\Gamma, c : [\vec{\mathbf{T}}]^\bullet = \Gamma_1$ *and* $P = Q$

$\mu = \texttt{env}$**:** $\Gamma \prec \Gamma_1$, $M = N$, $k = 0$ *and* $P = Q$

Proof. By rule induction on $\Gamma \triangleleft M \triangleright P \xrightarrow{\mu}_k \Gamma_1 \triangleleft N \triangleright Q$ □

Lemma 2 (Subject reduction)**.** *If* $\Gamma \triangleleft M \triangleright P$ *is a configuration and* $\Gamma \triangleleft M \triangleright P \xrightarrow{\mu}_k \Delta \triangleleft N \triangleright Q$ *then* $\Delta \triangleleft N \triangleright Q$ *is also a configuration.*

Proof. We assume that $\Gamma \vdash M$ and that there exists Δ such that Γ, Δ is consistent and that $\Delta \vdash M \triangleright P$. The rest of the proof follows from Lemma 1 (Transition and Structure), by case analysis of μ. □

3.2 Costed Bisimulation

We define a cost-based preorder over systems as a *typed relation*, *cf.* Definition 3, ordering systems that exhibit the same external behaviour at a less-than-or-equal-to cost. We require the preorder to consider client C_1 as more efficient than C_0 *wrt.* an appropriate resource environment M and observers characterised by the type environment stated in (6) but also that, *wrt.* the same resource and observer environments, client C_3 of (5) is more efficient than C_1. This latter ordering is harder to establish since client C_1 is at times *temporarily* more efficient than C_3.

In order to handle this aspect we define our preorder as an *amortized* bisimulation [KAK05]. Amortized bisimulation uses a *credit n* to compare a system $M \triangleright P$ with an less efficient system $N \triangleright Q$ while allowing $M \triangleright P$ to do a more expensive action than $N \triangleright Q$, as long as the credit can make up for the difference. Conversely, whenever $M \triangleright P$ does a cheaper action than $N \triangleright Q$, then the difference gets *added* to the credit.[6] In general, we refine Definition 3 to amortized typed relations with the following structure:

Definition 5 (Amortised Typed Relation)**.** *An amortized type-indexed relation $\mathcal{R}$ relates systems under a observer characterized by a context Γ, with credit n ($n \in$ Nat); we write*

$$\Gamma \vDash M \triangleright P \; \mathcal{R}^n \; N \triangleright Q$$

if $\mathcal{R}^n$ relates $\Gamma \triangleleft M \triangleright P$ and $\Gamma \triangleleft N \triangleright Q$, and both $\Gamma \triangleleft M \triangleright P$ and $\Gamma \triangleleft N \triangleright Q$ are configurations.

Definition 6 (Amortised Typed Bisimulation)**.** *An amortized type-indexed relation over processes $\mathcal{R}$ is a bisimulation at Γ with credit n if, whenever $\Gamma \vDash (M \triangleright P)\,\mathcal{R}^n\,(N \triangleright Q)$,*

- *If $\Gamma \triangleleft M \triangleright P \xrightarrow{\mu}_k \Gamma' \triangleleft M' \triangleright P'$ then there exist N' and Q' such that $\Gamma \triangleleft N \triangleright Q \stackrel{\hat{\mu}}{\Longrightarrow}_l \Gamma' \triangleleft N' \triangleright Q'$ where $\Gamma' \vDash (M' \triangleright P')\,\mathcal{R}^{n+l-k}\,(N' \triangleright Q')$*
- *If $\Gamma \triangleleft N \triangleright Q \xrightarrow{\mu}_l \Gamma' \triangleleft N' \triangleright Q'$ then there exist M' and P' such that $\Gamma \triangleleft M \triangleright P \stackrel{\hat{\mu}}{\Longrightarrow}_k \Gamma' \triangleleft M' \triangleright P'$ where $\Gamma' \vDash (M' \triangleright P')\,\mathcal{R}^{n+l-k}\,(N' \triangleright Q')$*

where $\hat{\mu}$ is the empty string if $\mu = \tau$ and μ otherwise.

Bisimilarity at Γ with credit n, denoted $\Gamma \vDash M \triangleright P \sqsubseteq^n_{\approx bis} N \triangleright Q$, is the largest amortized typed bisimulation at Γ with credit n. We sometimes existentially quantify over the credit and write $\Gamma \vDash M \triangleright P \sqsubseteq_{\approx bis} N \triangleright Q$. We also write $\Gamma \vDash M \triangleright P \simeq_{bis} N \triangleright Q$ whenever we have both $\Gamma \vDash M \triangleright P \sqsubseteq_{\approx bis} N \triangleright Q$ and $\Gamma \vDash N \triangleright Q \sqsubseteq_{\approx bis} M \triangleright P$, and $\Gamma \vDash M \triangleright P \sqsubset_{bis} N \triangleright Q$ whenever $\Gamma \vDash M \triangleright P \sqsubseteq_{\approx bis} N \triangleright Q$ but $\Gamma \vDash N \triangleright Q \not\sqsubseteq_{bis} M \triangleright P$.

Example 2 (Assessing Client Efficiency)**.** *For*

$$\Gamma_1 \stackrel{def}{=} \mathtt{srv}_1 : [[\mathbf{T}_1]^{\mathbf{1}}]^{\omega},\ \mathtt{srv}_2 : [[\mathbf{T}_2]^{\mathbf{1}}]^{\omega},\ c : [\mathbf{T}_1, \mathbf{T}_2]^{\omega} \tag{21}$$

[6] Stated otherwise, $M \triangleright P$ can do a more expensive action than $N \triangleright Q$ now, as long as it makes up for it later.

we can show that $\Gamma_1 \vDash (M \triangleright \mathrm{C}_1) \sqsubseteq_{bis} (M \triangleright \mathrm{C}_0)$ *by constructing the witness bisimulation (family of) relation(s)* $\mathcal{R}$ *for* $\Gamma_1 \vDash (M \triangleright \mathrm{C}_1) \sqsubseteq^0_{bis} (M \triangleright \mathrm{C}_0)$ *stated below:*[7]

$$\mathcal{R} \stackrel{def}{=} \left\{ \begin{array}{l|l}
\langle \Gamma,\ n,\ M' \triangleright \mathrm{C}_1,\ N' \triangleright \mathrm{C}_0 \rangle & \\
\left\langle \begin{array}{l} \Gamma, n, M' \triangleright \mathtt{alloc}\, x.\ \mathtt{srv}_1!x.\, x?y.\mathtt{srv}_2!x.\, x?z.\mathtt{c}!(y,z).\mathrm{C}_1 \\ \quad , N' \triangleright \mathtt{alloc}\, x_1.\mathtt{alloc}\, x_2.\ \mathtt{srv}_1!x_1.\, x_1?y.\mathtt{srv}_2!x_2.\, x_2?z.\mathtt{c}!(y,z).\mathrm{C}_0 \end{array} \right\rangle & n \geq 0 \\
\left\langle \begin{array}{l} \Gamma,\ n,\ (M',d) \triangleright \mathtt{srv}_1!d.\, d?y.\mathtt{srv}_2!d.\, d?z.\mathtt{c}!(y,z).\mathrm{C}_1 \\ \quad , (N',d') \triangleright \mathtt{alloc}\, x_2.\ \mathtt{srv}_1!d'.\, d'?y.\mathtt{srv}_2!x_2.\, x_2?z.\mathtt{c}!(y,z).\mathrm{C}_0 \end{array} \right\rangle & d \notin \mathrm{dom}(\Gamma) \\
\left\langle \begin{array}{l} \Gamma,\ n+1,\ (M',d) \triangleright \mathtt{srv}_1!d.\, d?y.\mathtt{srv}_2!d.\, d?z.\mathtt{c}!(y,z).\mathrm{C}_1 \\ \quad , (N',d',d'') \triangleright \mathtt{srv}_1!d'.\, d'?y.\mathtt{srv}_2!d''.\, d''?z.\mathtt{c}!(y,z).\mathrm{C}_0 \end{array} \right\rangle & d'' \notin \mathrm{dom}(\Gamma) \\
\left\langle \begin{array}{l} (\Gamma, d:[\mathbf{T}_1]^{\mathbf{1}}),\ n+1,\ (M',d) \triangleright d?y.\mathtt{srv}_2!d.\, d?z.\mathtt{c}!(y,z).\mathrm{C}_1 \\ \quad , (N',d,d'') \triangleright d?y.\mathtt{srv}_2!d''.\, d''?z.\mathtt{c}!(y,z).\mathrm{C}_0 \end{array} \right\rangle & M' \subseteq N' \\
\left\langle \begin{array}{l} \Gamma,\ n+1,\ (M',d) \triangleright \mathtt{srv}_2!d.\, d?z.\mathtt{c}!(v,z).\mathrm{C}_1 \\ \quad , (N',d,d'') \triangleright \mathtt{srv}_2!d''.\, d''?z.\mathtt{c}!(v,z).\mathrm{C}_0 \end{array} \right\rangle & \mathrm{dom}(\Gamma) \subseteq M' \\
\langle (\Gamma, d:[\mathbf{T}_2]^{\mathbf{1}}),\ n+1,\ (M',d) \triangleright d?z.\mathtt{c}!(v,z).\mathrm{C}_1,\ (N',d',d) \triangleright d?z.\mathtt{c}!(v,z).\mathrm{C}_0 \rangle & \\
\langle \Gamma,\ n+1,\ (M',d) \triangleright \mathtt{c}!(v,v').\mathrm{C}_1,\ (N',d',d) \triangleright \mathtt{c}!(v,v').\mathrm{C}_0 \rangle &
\end{array} \right\}$$

It is not hard to see that $\mathcal{R}$ *contains the quadruple* $\langle \Gamma_1, 0, M \triangleright \mathrm{C}_1, M \triangleright \mathrm{C}_0 \rangle$*. One can also show that it is closed* wrt. *the transfer property of Definition 6. The key moves are:*

- *a single channel allocation by* C_1 *is matched by two channel allocations by* C_0 *-from the second up to the fourth quadruple in the definition of* $\mathcal{R}$*. Since channel allocations carry a positive cost, the amortisation credit increases from* n *to* $n + 2 - 1$*, i.e.,* $n + 1$*, but this still yields a quadruple that is in the relation. One thing to note is that the first channel allocated by both systems is allowed to be different, e.g.,* d *and* d'*, as long as it is not allocated already.*
- *Even though the internal channels allocated may be different, rule* lRen *allows us to match the channels communicated on* $\mathtt{srv}_1$ *(fourth and fifth quadruples) since these channels are not known to the observer, i.e., they are not in* $\mathrm{dom}(\Gamma)$*.*
- *Communicating on the previously communicated channel on* $\mathtt{srv}_1$ *consumes all of the observer's permissions for that channel (fifth quadruple), which allows rule* lRen *to be applied again so as to match the channels communicated on* $\mathtt{srv}_2$ *(sixth quadruple).*

We cannot however prove that $\Gamma_1 \vDash (M \triangleright \mathrm{C}_0) \sqsubseteq^n_{bis} (M \triangleright \mathrm{C}_1)$ *for any* n *because we would need an infinite amortisation credit to account for additional cost incurred by* C_0 *when it performs the channel extra allocation at every iteration; recall that this credit cannot become negative, and thus no finite credit is large enough to cater for all the additional cost incurred by* C_0 *over sufficiently large transition sequences.*

Similarly, we can show that $\Gamma_1 \vDash (M \triangleright \mathrm{C}_2) \sqsubset_{bis} (M \triangleright \mathrm{C}_1)$ *but also , from* (5)*, that* $\Gamma_1 \vDash (M \triangleright \mathrm{C}_3) \sqsubset_{bis} (M \triangleright \mathrm{C}_1)$*. In particular, we can show* $\Gamma_1 \vDash (M \triangleright \mathrm{C}_3) \sqsubseteq_{bis} (M \triangleright \mathrm{C}_1)$ *even though* $M \triangleright \mathrm{C}_1$ *is temporarily more efficient than* $M \triangleright \mathrm{C}_3$*, i.e., during the course of the first iteration. Our framework handles this through*

[7] In families of relations ranging over systems indexed by type environments and amortisation credits, such as $\mathcal{R}$, we represent $\Gamma \vDash (M \triangleright P) \sqsubseteq^n_{\mathrm{bis}} (\Delta \triangleright Q)$ as the quadruple $\langle \Gamma, n, (M \triangleright P), (\Delta \triangleright Q) \rangle$.

the use of the amortisation credit whereby, in this case, it suffices to use a credit of value 1 *and show* $\Gamma_1 \vDash (M \triangleright \mathrm{C}_3) \sqsubseteq^1_{\approx bis} (M \triangleright \mathrm{C}_1)$*; we leave the details to the interested reader. Using an amortisation credit of* 1 *we can also show* $\Gamma_1 \vDash (M \triangleright \mathrm{C}_3) \sqsubseteq^1_{\approx bis} (M \triangleright \mathrm{C}_2)$ *through the bisimulation family-of-relations* $\mathcal{R}'$ *below — it is easy to check that it observes the transfer property of Definition 6. We just note that in* $\mathcal{R}'$*, the amortisation credit* n *can be capped* $0 \leq n \leq 1$*; we will touch again on this point in Section 3.3.*

$$
\mathcal{R}' \stackrel{def}{=} \left\{ \begin{array}{l|l}
\langle \Gamma,\ 1,\ M \triangleright \mathrm{C}_3,\ M \triangleright \mathrm{C}_2 \rangle & \\
\left\langle \begin{array}{l} \Gamma,\ 1,\ M \triangleright \left(\begin{array}{l} \mathtt{alloc}\, x_1.\mathtt{alloc}\, x_2.\ \mathtt{srv}_1!x_1.\, x_1?y. \\ \quad \mathtt{srv}_2!x_2.\, x_2?z.\mathtt{free}\, x_1.\mathtt{free}\, x_2.\mathtt{c}!(y,z).\mathrm{C}_3 \end{array} \right), \\ \qquad M \triangleright \mathtt{alloc}\, x.\ \mathtt{srv}_1!x.\, x?y.\mathtt{srv}_2!x.\, x?z.\mathtt{free}\, x.\mathtt{c}!(y,z).\mathrm{C}_2 \end{array} \right\rangle & \\
\left\langle \begin{array}{l} \Gamma,\ 1,\ (M,d) \triangleright \left(\begin{array}{l} \mathtt{alloc}\, x_2.\ \mathtt{srv}_1!d.\, d?y. \\ \quad \mathtt{srv}_2!x_2.\, x_2?z.\mathtt{free}\, d.\mathtt{free}\, x_2.\mathtt{c}!(y,z).\mathrm{C}_3 \end{array} \right), \\ \qquad (M,d') \triangleright\ \mathtt{srv}_1!d'.\, d'?y.\mathtt{srv}_2!d'.\, d'?z.\mathtt{free}\, d'.\mathtt{c}!(y,z).\mathrm{C}_2 \end{array} \right\rangle & d \notin d\mathrm{om}(\Gamma) \\
\left\langle \begin{array}{l} \Gamma,\ 0,\ (M,d,d'') \triangleright \left(\begin{array}{l} \mathtt{srv}_1!d.\, d?y.\mathtt{srv}_2!d''.\, d''?z. \\ \quad \mathtt{free}\, d.\mathtt{free}\, d''.\mathtt{c}!(y,z).\mathrm{C}_3 \end{array} \right), \\ \qquad (M,d') \triangleright\ \mathtt{srv}_1!d'.\, d'?y.\mathtt{srv}_2!d'.\, d'?z.\mathtt{free}\, d'.\mathtt{c}!(y,z).\mathrm{C}_2 \end{array} \right\rangle & d' \notin d\mathrm{om}(\Gamma) \\
\left\langle \begin{array}{l} (\Gamma, d:[\mathbf{T}_1]^{\mathbf{1}}),\ 0,\ (M,d,d'') \triangleright \left(\begin{array}{l} d?y.\mathtt{srv}_2!d''.\, d''?z. \\ \quad \mathtt{free}\, d.\mathtt{free}\, d''.\mathtt{c}!(y,z).\mathrm{C}_3 \end{array} \right), \\ \qquad (M,d) \triangleright\ d?y.\mathtt{srv}_2!d.\, d?z.\mathtt{free}\, d.\mathtt{c}!(y,z).\mathrm{C}_2 \end{array} \right\rangle & d'' \notin d\mathrm{om}(\Gamma) \\
\left\langle \begin{array}{l} \Gamma,\ 0,\ (M,d,d'') \triangleright\ \mathtt{srv}_2!d''.\, d''?z.\mathtt{free}\, d.\mathtt{free}\, d''.\mathtt{c}!(v,z).\mathrm{C}_3, \\ \qquad (M,d) \triangleright\ \mathtt{srv}_2!d.\, d?z.\mathtt{free}\, d.\mathtt{c}!(v,z).\mathrm{C}_2 \end{array} \right\rangle & \\
\left\langle \begin{array}{l} (\Gamma, d':[\mathbf{T}_2]^{\mathbf{1}}),\ 0,\ (M,d,d') \triangleright\ d'?z.\mathtt{free}\, d.\mathtt{free}\, d'.\mathtt{c}!(v,z).\mathrm{C}_3, \\ \qquad (M,d') \triangleright\ d'?z.\mathtt{free}\, d'.\mathtt{c}!(v,z).\mathrm{C}_2 \end{array} \right\rangle & d\mathrm{om}(\Gamma) \subseteq M \\
\left\langle \begin{array}{l} \Gamma,\ 0,\ (M,d,d') \triangleright\ \mathtt{free}\, d.\mathtt{free}\, d'.\mathtt{c}!(v,z).\mathrm{C}_3, \\ \qquad (M,d') \triangleright\ \mathtt{free}\, d'.\mathtt{c}!(v,v').\mathrm{C}_2 \end{array} \right\rangle & \\
\left\langle \Gamma,\ 0,\ (M,d') \triangleright\ \mathtt{free}\, d'.\mathtt{c}!(v,z).\mathrm{C}_3,\ M \triangleright\ \mathtt{c}!(v,v').\mathrm{C}_2 \right\rangle & \\
\left\langle \Gamma,\ 1,\ M \triangleright\ \mathtt{c}!(v,z).\mathrm{C}_3,\ M \triangleright\ \mathtt{c}!(v,v').\mathrm{C}_2 \right\rangle &
\end{array} \right\}
$$

□

3.3 Properties of $\sqsubseteq_{\mathbf{bis}}$

We show that our bisimulation relation of Definition 6 observes a number of properties that are useful when reasoning about resource efficiency; see Example 3. Lemmas 3 and 4 prove that the relation is in fact a preorder, whereas Lemma 5 outlines conditions where symmetry can be recovered. Finally, Theorem 1 shows that this preorder is preserved under (valid) context; this is the main result of the section.

We show that $\sqsubseteq_{\mathrm{bis}}$ is a preorder through Lemma 3 (where σ_Γ would be the identity) and Lemma 4.

Lemma 3 (Reflexivity upto Renaming)**.** *Whenever the triple* $\Gamma \triangleleft M \triangleright P$ *is a configuration, then* $\Gamma \vDash (M \triangleright P)\sigma_\Gamma \simeq_{bis} M \triangleright P$

Proof. By coinduction, by showing that the family of relations

$$\{\langle \Gamma, 0, (M \triangleright P)\sigma_\Gamma, M \triangleright P\rangle \mid \Gamma \triangleleft M \triangleright P \text{ is a configuration}\}$$

is a bisimulation. □

Lemma 4 (Transitivity)**.** *Whenever* $\Gamma \models M \triangleright P \precsim_{bis} M' \triangleright P'$ *and* $\Gamma \models M' \triangleright P' \precsim_{bis} M'' \triangleright P''$ *then* $\Gamma \models M \triangleright P \precsim_{bis} M'' \triangleright P''$

Proof. $\Gamma \models M \triangleright P \precsim_{\text{bis}} M' \triangleright P'$ implies that there exists some $n \geq 0$ and corresponding bisimulation relation justifying $\Gamma \models M \triangleright P \precsim^{n}_{\text{bis}} M' \triangleright P'$. The same applies for $\Gamma \models M' \triangleright P' \precsim_{\text{bis}} M'' \triangleright P''$ and some $m \geq 0$. From these two relations, one can construct a corresponding bisimulation justifying $\Gamma \models M \triangleright P \precsim^{n+m}_{\text{bis}} M'' \triangleright P''$. □

Corollary 1 (Preorder)**.** $\precsim_{bis}$ *is a preorder.*

Proof. Follows from Lemma 3 and Lemma 4. □

We can define a restricted form of amortised typed bisimulation, in analogous fashion to Definition 6, whereby the credit is *capped at some upperbound*, *i.e.*, some natural number m. We refer to such relations as *Bounded Amortised Typed-Bisimulations* and write

$$\Gamma \models^{m} M \triangleright P \precsim^{n}_{\text{bis}} N \triangleright Q$$

to denote that $\Gamma \triangleleft M \triangleright P$and $\Gamma \triangleleft N \triangleright Q$ are related by some amortised typed-indexed bisimulation at index Γ and credit n, and where every credit in this relation is less than or equal to m; whenever the precise credit n is not important we elide it and simply write $\Gamma \models^{m} M \triangleright P \precsim_{\text{bis}} N \triangleright Q$. We can show that bounded amortised typed-bisimulations are symmetric.

Lemma 5 (Symmetry)**.** $\Gamma \models^{m} M \triangleright P \precsim_{bis} N \triangleright Q$ *implies* $\Gamma \models^{m} N \triangleright Q \precsim_{bis} M \triangleright P$

Proof. If $\mathcal{R}$ is the bounded amortised typed relation justifying $\Gamma \models^{m} M \triangleright P \precsim_{\text{bis}} N \triangleright Q$, we define the amortised typed relation

$$\mathcal{R}_{\text{sym}} = \{\langle \Gamma, (m-n), N \triangleright Q, M \triangleright P\rangle \mid \langle \Gamma, n, M \triangleright P, N \triangleright Q\rangle \in \mathcal{R}\}$$

and show that it is a bounded amortised typed bisimulation as well. Consider an arbitrary pair of configurations $\Gamma \models N \triangleright Q \, \mathcal{R}^{m-n}_{\text{sym}} \, M \triangleright P$:

- Assume $\Gamma \triangleleft N \triangleright Q \xrightarrow{\mu}_{l} \Gamma' \triangleleft N' \triangleright Q'$. From the definition of $\mathcal{R}_{\text{sym}}$, it must be the case that $\langle \Gamma, n, M \triangleright P, N \triangleright Q\rangle \in \mathcal{R}$. Since $\mathcal{R}$ is a bounded amortised typed bisimulation, we know that $\Gamma \triangleleft M \triangleright P \overset{\hat{\mu}}{\Rightarrow}_{l} \Gamma' \triangleleft M' \triangleright P'$ where $\langle \Gamma', n + l - k, M' \triangleright P', N' \triangleright Q'\rangle \in \mathcal{R}$. We however need to show that $\langle \Gamma', ((m-n) + k - l), N' \triangleright Q', M' \triangleright P'\rangle \in \mathcal{R}_{\text{sym}}$, which follows from the definition of $\mathcal{R}_{\text{sym}}$ and the fact that $(m - (n + l - k)) = (m - n) + k - l$.

What is left to show is that $\mathcal{R}_{\text{sym}}$ is an amortised typed bisimulation bounded by m, *i.e.,* we need to show that $0 \leq (m-n)+k-l \leq m$. Since $\mathcal{R}$ is an m-bounded amortised typed bisimulation, we know that $0 \leq (n+l-k) \leq m$ from which we can drive $-m \leq -(n+l-k) \leq 0$ and, by adding m throughout we obtain $0 \leq (m-(n+l-k) = (m-n)+k-l) \leq m$ as required.

- The dual case for $\Gamma \triangleleft M \triangleright P \xrightarrow{\mu}_l \Gamma' \triangleleft M' \triangleright P'$ is analogous.

□

Two systems $M \triangleright P$ and $N \triangleright Q$ related by $\precsim_{\text{bis}}$ under Γ, remain related when extended with an additional process, R, whenever this process runs safely over the respective resource environments M and N, and observes the type restrictions and guarantees assumed by Γ (and dually, those of the respective existentially-quantified type environments for $M \triangleright P$ and $N \triangleright Q$). Following Definition 2, for these conditions to hold, contextuality requires R to typecheck *wrt.* a sub-environment of Γ, say Γ_1 where $\Gamma = \Gamma_1, \Gamma_2$, and correspondingly strengthens the relation of $M \triangleright P \parallel R$ and $N \triangleright Q \parallel R$ in $\precsim_{\text{bis}}$ under the remaining sub-environment, Γ_2.Thus, contextuality requires us to transfer the respective permissions associated with the observer sub-process R from the observer environment Γ; this is crucial in order to preserve consistency in the respective configurations, thus safety.

Theorem 1 relies on a list of lemmas outlined below.

Lemma 6 (Weakening)**.** *If* $\Gamma \triangleleft M \triangleright P \xrightarrow{\mu}_k \Gamma' \triangleleft M' \triangleright P'$ *then* $(\Gamma, \Delta) \triangleleft M \triangleright P \xrightarrow{\mu}_k (\Gamma', \Delta) \triangleleft M' \triangleright P'$. *(These may or may not be configurations.)*

Proof. By rule induction on $\Gamma \triangleleft M \triangleright P \xrightarrow{\mu}_k \Gamma' \triangleleft M' \triangleright P'$. Note that, in the case of `alloc`, the action can still be performed. □

Lemma 7 (Strengthening)**.** *If* $(\Gamma, \Delta) \triangleleft M \triangleright P \xrightarrow{\mu}_k (\Gamma', \Delta) \triangleleft M' \triangleright P'$ *then* $\Gamma \triangleleft M \triangleright P \xrightarrow{\mu}_k \Gamma' \triangleleft M' \triangleright P'$.

Proof. By rule induction on $\Gamma, \Delta \triangleleft P \xrightarrow{\mu}_k \Gamma', \Delta \triangleleft P'$. Note that strengthening is restricted to the part of the environment that remains unchanged (Δ is the same on the left and right hand side) — otherwise the property does not hold for actions $c!\vec{d}$ and $c?\vec{d}$. □

Lemma 8. *If* Γ, Δ *is consistent and* $\Delta \prec \Delta'$ *then* Γ, Δ' *is consistent and* $\Gamma, \Delta \prec \Gamma, \Delta'$

Proof. As in [dVFH10]. □

Lemma 9 (Typing Preserved by $\equiv$)**.** $\Gamma \vdash P$ *and* $P \equiv Q$ *implies* $\Gamma \vdash Q$

Proof. As in [dVFH10]. □

Lemma 10 (Environment Structural Manipulation Preserves bisimulation)**.**

$$\Gamma \vDash S \precsim^n_{bis} T \text{ and } \Gamma \prec \Delta \text{ implies } \Delta \vDash S \precsim^n_{bis} T$$

Proof. By coinduction. □

Lemma 11 (Bisimulation and Structural Equivalence)**.**

$$P \equiv Q \text{ and } \Gamma \triangleleft M \triangleright P \xrightarrow{\mu}_k \Delta \triangleleft M' \triangleright P' \text{ implies } \Gamma \triangleleft M \triangleright Q \xrightarrow{\mu}_k \Delta \triangleright M' \triangleright Q' \text{ and } P' \equiv Q'$$

Proof. By rule induction on $P \equiv Q$ and then a case analysis of the rules permitting $\Gamma \triangleleft M \triangleright P \xrightarrow{\mu}_k \Delta \triangleleft M' \triangleright P'$. □

Corollary 2 (Structural Equivalence and Bisimilarity)**.** $P \equiv Q$ *implies* $\Gamma \vDash M \triangleright P \sqsubseteq^n_{\approx bis} M \triangleright Q$ *for arbitrary n and* Γ *where* $\Gamma \triangleleft M \triangleright P$ *and* $\Gamma \triangleleft M \triangleright Q$ *are configurations.*

Proof. By coinduction and Lemma 11. □

Lemma 12 (Renaming)**.** *If* $\Gamma, \Delta \vDash (M \triangleright P) \sqsubseteq^n_{\approx bis} (N \triangleright Q)$ *then* $\Gamma, (\Delta\sigma_\Gamma) \vDash (M \triangleright P)\sigma_\Gamma \sqsubseteq^n_{\approx bis} (N \triangleright Q)\sigma_\Gamma$

Proof. By coinduction. □

Theorem 1 (Contextuality)**.** *If* $\Gamma, \Delta \vDash (M \triangleright P) \sqsubseteq^n_{\approx bis} (N \triangleright Q)$ *and* $\Delta \vdash R$ *then*

$$\Gamma \vDash (M \triangleright P \parallel R) \sqsubseteq^n_{\approx bis} (N \triangleright Q \parallel R) \quad \text{and} \quad \Gamma \vDash (M \triangleright R \parallel P) \sqsubseteq^n_{\approx bis} (N \triangleright R \parallel Q)$$

Proof. We define the family of relations $\mathcal{R}^{\Gamma,n}$ to be the least one satisfying the rules

$$\frac{\Gamma \vDash (M \triangleright P) \sqsubseteq^n_{\approx \text{bis}} (N \triangleright Q)}{\Gamma \vDash (M \triangleright P)\ \mathcal{R}^n\ (N \triangleright Q)} \qquad \frac{\Gamma, \Delta \vDash (M \triangleright P)\ \mathcal{R}^n\ (N \triangleright Q) \quad \Delta \vdash R}{\Gamma \vDash (M \triangleright P \parallel R)\ \mathcal{R}^n\ (N \triangleright Q \parallel R)} \qquad \frac{\Gamma, \Delta \vDash (M \triangleright P)\ \mathcal{R}^n\ (N \triangleright Q) \quad \Delta \vdash R}{\Gamma \vDash (M \triangleright R \parallel P)\ \mathcal{R}^n\ (N \triangleright R \parallel Q)}$$

and then show that $\mathcal{R}^{\Gamma,n}$ is a costed typed bisimulation at Γ,n (up to ≡). Note that the premise of the first rule implies that both $\Gamma, \Delta \triangleleft M \triangleright P$ and $\Gamma, \Delta \triangleleft N \triangleright Q$ are configurations. We consider only the transitions of the left hand configurations for second case of the relation; the first is trivial and the third is analogous to the second. Although the relation is not symmetric, the transition of the right hand configurations are analogous to those of the left hand configurations.

1. Case

$$\dfrac{\dfrac{\Gamma \triangleleft (M \triangleright P)\sigma_\Gamma \xrightarrow{\mu}_l \Gamma' \triangleleft M' \triangleright P'}{\Gamma \triangleleft (M \triangleright P \parallel R)\sigma_\Gamma \xrightarrow{\mu}_l \Gamma' \triangleleft M' \triangleright P' \parallel R\sigma_\Gamma}\ \textsc{lPar-L}}{\Gamma \triangleleft M \triangleright P \parallel R \xrightarrow{\mu}_l \Gamma' \triangleleft M' \triangleright P' \parallel R}\ \textsc{lRen} \tag{22}$$

   By Lemma 6 (Weakening), lRen and (22) we obtain

$$\Gamma, (\Delta\sigma_\Gamma) \triangleleft (M \triangleright P)\sigma_\Gamma \xrightarrow{\mu}_l \Gamma', (\Delta\sigma_\Gamma) \triangleleft M' \triangleright P'$$

   From the case assumption $\Gamma, \Delta \vDash M \triangleright P\, \mathcal{R}^n\, N \triangleright Q$ (defining $\mathcal{R}^{\Gamma,n}$) and Lemma 12 we obtain

$$\Gamma, (\Delta\sigma_\Gamma) \vDash (M \triangleright P)\sigma_\Gamma \mathcal{R}^n (N \triangleright Q)\sigma_\Gamma \tag{23}$$

Hence by I.H. and $\Gamma, \Delta \vDash (M \triangleright P)\ \mathcal{R}\ (N \triangleright Q)$ there exists a $N' \triangleright Q'$ such that

$$\Gamma, (\Delta\sigma_\Gamma) \triangleleft (N \triangleright Q)\sigma_\Gamma \overset{\hat{\mu}}{\Rightarrow}_k \Gamma', (\Delta\sigma_\Gamma) \triangleleft N' \triangleright Q' \tag{24}$$

$$\text{where} \quad \Gamma', (\Delta\sigma_\Gamma) \vDash (M' \triangleright P')\ \mathcal{R}^{n+k-l}\ (N' \triangleright Q') \tag{25}$$

By (24) and lRen, were $\Gamma_1 = \Gamma, (\Delta\sigma_\Gamma)$, we obtain

$$\Gamma, (\Delta\sigma_\Gamma) \triangleleft ((N \triangleright Q)\sigma_\Gamma))\sigma'_{\Gamma_1} \quad (\xrightarrow{\tau}{}^*_{k_1}) \xrightarrow{\hat{\mu}}_{k_2} (\xrightarrow{\tau}{}^*_{k_3}) \quad \Gamma', (\Delta\sigma_\Gamma) \triangleleft N' \triangleright Q' \tag{26}$$

where $k = k_1 + k_2 + k_3$. By, lPar Lemma 7 (Strengthening) and (26) we deduce

$$\Gamma \triangleleft ((N \triangleright Q)\sigma_\Gamma))\sigma'_{\Gamma_1} \parallel R\sigma_\Gamma \quad (\xrightarrow{\tau}{}^*_{k_1}) \xrightarrow{\hat{\mu}}_{k_2} (\xrightarrow{\tau}{}^*_{k_3}) \quad \Gamma' \triangleleft N' \triangleright Q' \parallel R\sigma_\Gamma \tag{27}$$

From $\Delta \vdash R$ we know

$$\Delta\sigma_\Gamma \vdash R\sigma_\Gamma \tag{28}$$

and, from $\Gamma_1 = \Gamma, (\Delta\sigma_\Gamma)$ and Def. 4 (Renaming Modulo Environments), we know that $(R\sigma_\Gamma)\sigma'_{\Gamma_1} = R\sigma_\Gamma$ since the renaming does not modify any of the names in the domain of Γ_1, hence of $\Delta\sigma_\Gamma$. Also, from Def. 4, σ'_{Γ_1} is also a substitution modulo Γ and can therefore refer to it as σ'_Γ, thereby rewriting (27) as

$$\Gamma \triangleleft (N \triangleright Q \parallel R)\sigma_\Gamma\sigma'_\Gamma \quad (\xrightarrow{\tau}{}^*_{k_1}) \xrightarrow{\hat{\mu}}_{k_2} (\xrightarrow{\tau}{}^*_{k_3}) \quad \Gamma' \triangleleft N' \triangleright Q' \parallel R\sigma_\Gamma \tag{29}$$

From (29) and lRen we thus obtain

$$\Gamma \triangleleft N \triangleright Q \parallel R \quad \overset{\hat{\mu}}{\Rightarrow}_k \quad \Gamma' \triangleleft N' \triangleright (Q' \parallel R\sigma_\Gamma)$$

This is our matching move since and by (25), (28) and the definition of $\mathcal{R}$ we obtain $\Gamma' \vDash (M' \triangleright P' \parallel R\sigma_\Gamma)\ \mathcal{R}^{n+l-k}\ (N' \triangleright Q' \parallel R\sigma_\Gamma)$.

2. Case

$$\dfrac{\dfrac{\Gamma \triangleleft (M \triangleright R)\sigma_\Gamma \quad \xrightarrow{\mu}_l \quad \Gamma' \triangleleft M' \triangleright R'}{\Gamma \triangleleft (M \triangleright P \parallel R)\sigma_\Gamma \quad \xrightarrow{\mu}_l \quad \Gamma' \triangleleft M' \triangleright P \parallel R'}\ \textsc{lPar-R}}{\Gamma \triangleleft M \triangleright P \parallel R \quad \xrightarrow{\mu}_l \quad \Gamma' \triangleleft M' \triangleright P \parallel R'}\ \textsc{lRen} \tag{30}$$

The proof proceeds by case analysis of μ whereby the most interesting cases are when $l = +1$ or $l = -1$. We here show the case for when $l = -1$ (the other case is analogous). By Lemma 1 we know

$$M\sigma_\Gamma = M', c \tag{31}$$

$$R\sigma_\Gamma \equiv \mathtt{free}\, c.R_1 \parallel R_2 \ \text{ and } \ R' \equiv R_1 \parallel R_2 \tag{32}$$

$$\Gamma = \Gamma' \tag{33}$$

$$\Delta\sigma_\Gamma \prec \Delta', c : [\vec{\mathbf{T}}]^\bullet \ \text{ and } \ \Delta' \vdash R'. \tag{34}$$

By Lemma 12 we know $\Gamma, \Delta\sigma_\Gamma \vDash (M \triangleright P)\sigma_\Gamma \; \mathcal{R}^n \; (N \triangleright Q)\sigma_\Gamma$, and by (34) and Lemma 10 we obtain

$$\Gamma, \Delta', c : [\vec{\mathbf{T}}]^\bullet \vDash (M \triangleright P)\sigma_\Gamma \; \mathcal{R}^n \; (N \triangleright Q)\sigma_\Gamma \tag{35}$$

and by (31) and lFree we deduce

$$\Gamma, \Delta', c : [\vec{\mathbf{T}}]^\bullet \triangleleft (M \triangleright P)\sigma_\Gamma \xrightarrow{\texttt{free}\, c}_{-1} \Gamma, \Delta' \triangleleft (M' \triangleright P)\sigma_\Gamma$$

and by (35) and I.H. there exists a matching move

$$\Gamma, \Delta', c : [\vec{\mathbf{T}}]^\bullet \triangleleft (N \triangleright Q)\sigma_\Gamma \overset{\texttt{free}\, c}{\Longrightarrow}_k \Gamma, \Delta' \triangleleft N' \triangleright Q' \tag{36}$$

$$\text{and } \Gamma, \Delta' \vDash M' \triangleright P' \; \mathcal{R}^{n+k-(-1)} \; N' \triangleright Q' \tag{37}$$

By (36) and lRen, for $k = k_1 - 1 + k_2$, we know

$$\Gamma, \Delta', c : [\vec{\mathbf{T}}]^\bullet \triangleleft ((N \triangleright Q)\sigma_\Gamma)\sigma'_{\Gamma_2} \xrightarrow{\tau}{}^*_{k_1} \Gamma, \Delta', c : [\vec{\mathbf{T}}]^\bullet \triangleleft N'' \triangleright Q'' \tag{38}$$

$$\text{where } \Gamma_2 = \Gamma, \Delta', c : [\vec{\mathbf{T}}]^\bullet \text{ (used in } \sigma'_{\Gamma_2} \text{ above)} \tag{39}$$

$$\Gamma, \Delta', c : [\vec{\mathbf{T}}]^\bullet \triangleleft N'' \triangleright Q'' \xrightarrow{\texttt{free}\, c}_{-1} \Gamma, \Delta' \triangleleft N''' \triangleright Q'' \tag{40}$$

$$\Gamma, \Delta' \triangleleft N''' \triangleright Q'' \xrightarrow{\tau}{}^*_{k_2} \Gamma, \Delta' \triangleleft N' \triangleright Q' \tag{41}$$

From (38), (41), lPar-L and Lemma 7(Strengthening) we obtain:

$$\Gamma \triangleleft ((N \triangleright Q)\sigma_\Gamma)\sigma'_{\Gamma_2} \parallel R\sigma_\Gamma \xrightarrow{\tau}{}^*_{k_1} \Gamma \triangleleft N'' \triangleright Q'' \parallel (R\sigma_\Gamma) \tag{42}$$

$$\Gamma \triangleleft N''' \triangleright Q'' \parallel R' \xrightarrow{\tau}{}^*_{k_2} \Gamma \triangleleft N' \triangleright Q' \parallel R' \tag{43}$$

Also, from (40) and Lemma 1(Transition and Structure) we deduce that $M'' = M''', c$ and thus, from (32), lFree, lPar-R we obtain:

$$\Gamma \triangleleft N'' \triangleright Q'' \parallel R\sigma_\Gamma \xrightarrow{\tau}_{-1} \Gamma \triangleleft N''' \triangleright Q'' \parallel R' \tag{44}$$

By (34) and (39), we know that we can find an alternative renaming function σ''_{Γ_3}, where $\Gamma_3 = \Gamma, (\Delta\sigma_\Gamma)$, in a way that, from (42), we can obtain

$$\Gamma \triangleleft ((N \triangleright Q)\sigma_\Gamma)\sigma''_{\Gamma_3} \parallel R\sigma_\Gamma \xrightarrow{\tau}{}^*_{k_1} \Gamma \triangleleft N'' \triangleright Q'' \parallel (R\sigma_\Gamma) \tag{45}$$

Now, by $\Delta \vdash R$ we know that $\Delta\sigma_\Gamma \vdash R\sigma_\Gamma$ and subsequently, by Def. 4 and (39) we know $(R\sigma_\Gamma)\sigma''_{\Gamma_3} = R\sigma_\Gamma$. Thus, we can rewrite $((N \triangleright Q)\sigma_\Gamma)\sigma''_{\Gamma_3} \parallel R\sigma_\Gamma$ in (45) as $((N \triangleright Q \parallel R)\sigma_\Gamma)\sigma''_{\Gamma_3}$. Merging (45), (44) and (43) we obtain:

$$\Gamma \triangleleft ((N \triangleright Q \parallel R)\sigma_\Gamma)\sigma''_{\Gamma_3} \xrightarrow{\tau}{}^*_{k_1} \xrightarrow{\tau}_{-1} \xrightarrow{\tau}{}^*_{k_2} \Gamma \triangleleft N' \triangleright Q' \parallel R'$$

By Def. 4 we know that σ''_{Γ_3} can be rewritten as σ''_Γ and thus by lRen we obtain the matching move

$$\Gamma \triangleleft N \triangleright Q \parallel R \overset{\tau}{\Longrightarrow}_k \Gamma \triangleleft N' \triangleright Q' \parallel R'$$

because by (37), (34) and the definition of $\mathcal{R}$ we know that

$$\Gamma \vDash M' \triangleright P' \parallel R' \; \mathcal{R}^{n+k-(-1)} \; N' \triangleright Q' \parallel R'.$$

3. Case

$$\dfrac{\dfrac{\Gamma_1 \triangleleft (M \triangleright P)\sigma_\Gamma \xrightarrow{c!\vec{d}}_0 \Gamma_1' \triangleleft M' \triangleright P' \qquad \Gamma_2 \triangleleft (M \triangleright R)\sigma_\Gamma \xrightarrow{c?\vec{d}}_0 \Gamma_2' \triangleleft M' \triangleright R'}{\Gamma \triangleleft (M \triangleright P \parallel R)\sigma_\Gamma \xrightarrow{\tau}_0 \Gamma \triangleleft M' \triangleright P' \parallel R'}\ \textsc{lCom-L}}{\Gamma \triangleleft M \triangleright P \parallel R \xrightarrow{\tau}_0 \Gamma \triangleleft M' \triangleright P' \parallel R'}\ \textsc{lRen} \tag{46}$$

By the two top premises of (46) and Lemma 1 we know

$$M\sigma_\Gamma = M' \tag{47}$$

$$P\sigma_\Gamma \equiv c!\vec{d}.P_1 \parallel P_2 \qquad P' \equiv P_1 \parallel P_2 \tag{48}$$

$$R\sigma_\Gamma \equiv c?\vec{x}.R_1 \parallel R_2 \qquad R' \equiv R_1\{\vec{d}/\vec{x}\} \parallel R_2 \tag{49}$$

From $\Delta \vdash R$ we obtain $\Delta\sigma_\Gamma \vdash R\sigma_\Gamma$, and by (49), $\Delta \vdash R$ and Inversion we obtain

$$\Delta\sigma_\Gamma \prec \Delta_1, \Delta_2, c : [\vec{\mathbf{U}}]^a \tag{50}$$

$$\Delta_1, c : [\vec{\mathbf{U}}]^{a-1}, \vec{x} : \vec{\mathbf{U}} \vdash R_1 \tag{51}$$

$$\Delta_2 \vdash R_2 \tag{52}$$

Note that through (51) we know that

$$c : [\vec{\mathbf{U}}]^{a-1} \text{ is defined.} \tag{53}$$

By (51), Substitution Lemma (from [dVFH10]) and (52) we obtain

$$\Delta_1, \Delta_2, c : [\vec{\mathbf{U}}]^{a-1}, \vec{d} : \vec{\mathbf{U}} \vdash R_1\{\vec{d}/\vec{x}\} \parallel R_2 \tag{54}$$

From the assumption defining $\mathcal{R}$, and Lemma 12 we obtain

$$\Gamma, (\Delta\sigma_\Gamma) \ \vDash\ (M \triangleright P)\sigma_\Gamma \ \mathcal{R}^n\ (N \triangleright Q)\sigma_\Gamma, \tag{55}$$

and by (50) and Proposition 8 we know that $\Gamma, (\Delta\sigma_\Gamma) \prec \Gamma, \Delta_1, \Delta_2, c : [\vec{\mathbf{U}}]^a$ and also that $\Gamma, \Delta_1, \Delta_2, c : [\vec{\mathbf{U}}]^a$ is consistent. Thus by (55) and Lemma 10 we deduce

$$\Gamma, \Delta_1, \Delta_2, c : [\vec{\mathbf{U}}]^a \ \vDash\ (M \triangleright P)\sigma_\Gamma \ \mathcal{R}^n\ (N \triangleright Q)\sigma_\Gamma \tag{56}$$

Now by (53),(48),(47), lOut, lPar-L, lRen and Lemma 11 we deduce

$$\Gamma, \Delta_1, \Delta_2, c : [\vec{\mathbf{U}}]^a \triangleleft (M \triangleright P)\sigma_\Gamma \xrightarrow{c!\vec{d}}_0 \Gamma, \Delta_1', \Delta_2', c : [\vec{\mathbf{U}}]^{a-1}, \vec{d} : \vec{\mathbf{U}} \triangleleft M' \triangleright P' \tag{57}$$

and hence by (56) and I.H. we obtain

$$\Gamma, \Delta_1, \Delta_2, c : [\vec{\mathbf{U}}]^a \triangleleft (N \triangleright Q)\sigma_\Gamma \xRightarrow{c!\vec{d}}_k \Gamma, \Delta_1', \Delta_2', c : [\vec{\mathbf{U}}]^{a-1}, \vec{d} : \vec{\mathbf{U}} \triangleleft N' \triangleright Q' \tag{58}$$

$$\text{such that } \Gamma, \Delta_1, \Delta_2, c : [\vec{\mathbf{U}}]^{a-1}, \vec{d} : \vec{\mathbf{U}} \vDash (M' \triangleright P')\ \mathcal{R}^{n+k-0}\ (N' \triangleright Q') \tag{59}$$

From (58) and lRen we know

$$\Gamma, \Delta_1, \Delta_2, c : [\vec{\mathbf{U}}]^a \triangleleft ((N \triangleright Q)\sigma_\Gamma)\sigma'_{\Gamma_4} \xrightarrow{\tau}{}^*_{k_1} \Gamma, \Delta_1, \Delta_2, c : [\vec{\mathbf{U}}]^a \triangleleft N'' \triangleright Q'' \tag{60}$$

$$\Gamma, \Delta_1, \Delta_2, c : [\vec{\mathbf{U}}]^a \triangleleft N'' \triangleright Q'' \xrightarrow{c!\vec{d}}_0 \Gamma, \Delta_1, \Delta_2, c : [\vec{\mathbf{U}}]^{a-1}, \vec{d} : \vec{\mathbf{U}} \triangleleft N'' \triangleright Q''' \tag{61}$$

$$\Gamma, \Delta_1, \Delta_2, c : [\vec{\mathbf{U}}]^{a-1}, \vec{d} : \vec{\mathbf{U}} \triangleleft N'' \triangleright Q''' \xrightarrow{\tau}{}^*_{k_2} \Gamma, \Delta_1, \Delta_2, c : [\vec{\mathbf{U}}]^{a-1}, \vec{d} : \vec{\mathbf{U}} \triangleleft N' \triangleright Q' \tag{62}$$

$$\text{where } k = k_1 + k_2 \text{ and } \Gamma_4 = \Gamma, \Delta_1, \Delta_2, c : [\vec{\mathbf{U}}]^a \ \ (\Gamma_4 \text{ is used in (60)}) \tag{63}$$

From (60), (62), lPar-L and Lemma 7(Strengthening) we obtain:

$$\Gamma \triangleleft ((N \triangleright Q)\sigma_\Gamma)\sigma'_{\Gamma_4} \parallel R\sigma_\Gamma \xrightarrow{\tau}{}^*_{k_1} \Gamma \triangleleft N'' \triangleright Q'' \parallel R\sigma_\Gamma \tag{64}$$

$$\Gamma \triangleleft N'' \triangleright Q''' \parallel R' \xrightarrow{\tau}{}^*_{k_2} \Gamma \triangleleft N' \triangleright Q' \parallel R' \tag{65}$$

By (49), lIn and lPar-L we can construct (for some Γ_6, Γ_7)

$$\Gamma_6 \triangleleft N'' \triangleright R\sigma_\Gamma \xrightarrow{c?\vec{d}}_0 \Gamma_7 \triangleleft N'' \triangleright R' \tag{66}$$

and by (61), (66) and lCom-L we obtain

$$\Gamma \triangleleft N'' \triangleright Q'' \parallel R\sigma_\Gamma \xrightarrow{\tau}_0 \Gamma \triangleleft N'' \triangleright Q''' \parallel R' \tag{67}$$

By (50) and (63), we know that we can find an alternative renaming function σ''_{Γ_5}, where $\Gamma_5 = \Gamma, (\Delta\sigma_\Gamma)$, in a way that, from (64), we can obtain

$$\Gamma \triangleleft ((N \triangleright Q)\sigma_\Gamma)\sigma''_{\Gamma_5} \parallel R\sigma_\Gamma \xrightarrow{\tau}{}^*_{k_1} \Gamma \triangleleft N'' \triangleright Q'' \parallel R\sigma_\Gamma \tag{68}$$

By Def. 4, $\Delta\sigma_\Gamma \vdash R\sigma_\Gamma$, (50), (63) we know that $(R\sigma_\Gamma)\sigma''_{\Gamma_5} = R\sigma_\Gamma$, and also that σ''_{Γ_5} is also a renaming modulo Γ, so we can denote it as σ''_Γ and rewrite $((N \triangleright Q)\sigma_\Gamma)\sigma''_{\Gamma_5} \parallel R\sigma_\Gamma$ as $((N \triangleright Q \parallel R)\sigma_\Gamma)\sigma''_\Gamma$ in (68). Thus, by (68), (67), (65), (63) and lRen we obtain the matching move

$$\Gamma \triangleleft N \triangleright Q \parallel R \ \ \stackrel{\tau}{\Longrightarrow}_k \ \ \Gamma \triangleleft N' \triangleright Q' \parallel R'$$

since by (59), (54), (49) and the definition of $\mathcal{R}$we obtain

$$\Gamma \vDash (M' \triangleright P' \parallel R') \ \mathcal{R}^{n+k-0} \ (N' \triangleright Q' \parallel R')$$

□

Example 3 (Properties of $\sqsubseteq_{\text{bis}}$)**.** *From the proved statements* $\Gamma_1 \vDash (M \triangleright \mathrm{C}_1) \sqsubseteq_{bis} (M \triangleright \mathrm{C}_0)$ *and* $\Gamma_1 \vDash (M \triangleright \mathrm{C}_2) \sqsubseteq_{bis} (M \triangleright \mathrm{C}_1)$ *of Example 2, and by Corollary 1(Preorder), we may conclude that*

$$\Gamma_1 \vDash (M \triangleright \mathrm{C}_2) \sqsubseteq_{bis} (M \triangleright \mathrm{C}_0) \tag{69}$$

without the need to provide a bisimulation relation justifying (69)*. We also note that* $\mathcal{R}'$ *of Example 2, justifying* $\Gamma_1 \vDash (M \triangleright \mathrm{C}_3) \sqsubseteq_{bis} (M \triangleright \mathrm{C}_2)$ *is a* bounded *amortised typed-bisimulation, and by Lemma 5 we can also conclude*

$$\Gamma_1 \vDash (M \triangleright \mathrm{C}_2) \sqsubseteq_{bis} (M \triangleright \mathrm{C}_3)$$

and thus $\Gamma_1 \vDash (M \triangleright \mathrm{C}_3) \simeq_{bis} (M \triangleright \mathrm{C}_2)$. *Finally, by Theorem 1, in order to show that*

$$c : [\mathbf{T}_1, \mathbf{T}_2]^{\omega} \vDash (M \triangleright \mathrm{S}_1 \parallel \mathrm{S}_2 \parallel \mathrm{C}_1) \sqsubset_{bis} (M \triangleright \mathrm{S}_1 \parallel \mathrm{S}_2 \parallel \mathrm{C}_0)$$

it suffices to abstract away from the common code, $\mathrm{S}_1 \parallel \mathrm{S}_2$*, and show* $\Gamma_1 \vDash (M \triangleright \mathrm{C}_1) \sqsubset_{bis} (M \triangleright \mathrm{C}_0)$*, as proved already in Example 2.*

3.4 Alternatives

The cost model we adhere to so far, which follows that introduced by the costed reductions of Section 2, is not the only plausible one. There may be other valid alternatives, some of which can be easily accommodated through minor tweaking in our existing framework.

For instance, an alternative cost model may focus on assessing the runtime execution of programs, whereby operations that access memory such as $\mathtt{alloc}\, x.P$ and $\mathtt{free}\, c.P$ have a runtime cost that far exceeds that of other operations. We can model this by considering an LTS that assigns a cost of 1 to both of these operations, which can be attained as a derived LTS from our existing LTS of Section 3.1 through the rule

$$\frac{\Gamma \triangleleft M \triangleright P \xrightarrow{\mu}_{k} \Gamma' \triangleleft M' \triangleright P'}{\Gamma \triangleleft M \triangleright P \xrightarrow{\mu}\!\!\!\!\rightarrow_{|k|} \Gamma' \triangleleft M' \triangleright P'}\ \textsc{lDer1}$$

where $|k|$ returns the absolute value of an integer. Def. 6 extends in straightforward fashion to work with the derived costed LTS $\xrightarrow{\mu}\!\!\!\!\rightarrow_{k}$. The new preorder would allow us to conclude $\Gamma_1 \vDash (M \triangleright \mathrm{C}_1) \sqsubseteq_{\mathrm{bis}} (M \triangleright \mathrm{C}_2)$ because according to the new cost model, for every server-interaction iteration, client C_1 uses less expensive memory operations than C_2.

Another cost model may require us to refine our existing preorder: at present we are able to equate the following clients

$$\Gamma_1 \vDash (M \triangleright \mathrm{C}_4) \simeq_{\mathrm{bis}} (M \triangleright \mathrm{C}_3) \simeq_{\mathrm{bis}} (M \triangleright \mathrm{C}_2)$$

on the basis that neither client carries any memory leaks. Client C_4, defined below, creates a single channel and keeps on reusing it for all iterations:

$$\mathrm{C}_4 \triangleq \mathtt{alloc}\, x.\, \mathtt{rec}\, w.\, \mathtt{srv}_1!x.\, x?y.\, \mathtt{srv}_2!x.\, x?z.\mathtt{ret}!(y, z).\, w$$

However, we may want a finer preorder where C_4 is considered to be (strictly) more efficient than C_2, which is in turn more efficient than C_3. The underlying reasoning for this would be that C_4 uses the least amount of expensive operations; by contrast C_2 keeps on allocating (and deallocating) new channels for each iteration, and C_3 allocates (and deallocates) two new channels for every iteration. We can characterise this preorder as follows. First we generate the derived costed LTS using the rule lDer2 below — $\lfloor k \rfloor$ maps all negative integers to 0, leaving positive integers unaltered.

$$\frac{\Gamma \triangleleft M \triangleright P \xrightarrow{\mu}_{k} \Gamma' \triangleleft M' \triangleright P'}{\Gamma \triangleleft M \triangleright P \xrightarrow{\mu}\!\!\!\!\rightarrow_{\lfloor k \rfloor} \Gamma' \triangleleft M' \triangleright P'}\ \textsc{lDer2}$$

Then, after adapting Def. 6 to this derived LTS, denoting such a bisimulation relation as $\sqsubseteq_{\text{bis2}}$, we can define the refined preorder, denoted as $\sqsubseteq_{\text{bis3}}$, as follows:

$$\Gamma \vDash M \triangleright P \sqsubseteq_{\text{bis3}} N \triangleright Q \quad \stackrel{\text{def}}{=} \quad \begin{cases} \Gamma \vDash M \triangleright P \sqsubseteq_{\text{bis}} N \triangleright Q \text{ and} \\ \Gamma \vDash N \triangleright Q \sqsubseteq_{\text{bis}} M \triangleright P \text{ implies } \Gamma \vDash M \triangleright P \sqsubseteq_{\text{bis2}} N \triangleright Q \end{cases}$$

4 Characterisation

In this section we give a sound and complete characterization of bisimilarity in terms of the reduction semantics of Section 2, justifying the bisimulation relation and the respective LTS as a proof technique for reasoning about the behaviour of $R\pi$ processes. Our touchstone behavioural preorder is based on a costed version of families of reduction closed barbed congruences along similar lines to [Hen09]. In order to limit behaviour to safe computations, these congruences are defined as typed relation (Definition 3), where systems are subject to common observers typed by environments.

The observer type-environment delineates the observations that can be made: the observer can only make distinctions for channels that it has a permission for, *i.e.*, at least an affine typing assumption. The observations that can be made in our touchstone behavioural preorder are described as *barbs* [HT92] that take into account the permissions owned by the observer. We require systems related by our behavioral preorder to exhibit the same barbs *wrt.* a common observer.

Definition 7 (Barb)**.** $(\Gamma \triangleleft M \triangleright P) \Downarrow_c^{barb} \stackrel{def}{=} (M \triangleright P) \longrightarrow_k^* \equiv (M' \triangleright P' \parallel c!\vec{d}.P'')$ *and* $c \in \text{dom}(\Gamma)$.

Definition 8 (Barb Preservation)**.** *A typed relation* $\mathcal{R}$ *is barb preserving if and only if*

$$\Gamma \vDash M \triangleright P \,\mathcal{R}\, N \triangleright Q \text{ implies } \left(\Gamma \triangleleft M \triangleright P \Downarrow_c^{barb} \text{ iff } \Gamma \triangleleft N \triangleright Q \Downarrow_c^{barb}\right).$$

Our behavioural preorder takes cost into consideration; it is defined in terms of families of amortised typed relations that are closed under costed reductions.

Definition 9 (Cost Improving)**.** *An amortized type-indexed relation* $\mathcal{R}$ *is* cost improving at credit n *iff whenever* $\Gamma \vDash (M \triangleright P)\ \mathcal{R}^n\ (N \triangleright Q)$ *and*

1. *if* $M \triangleright P \longrightarrow_k M' \triangleright P'$ *then* $N \triangleright Q \longrightarrow_l^* N' \triangleright Q'$ *such that* $\Gamma \vDash (M' \triangleright P')\ \mathcal{R}^{n+l-k}\ (N' \triangleright Q')$*;*
2. *if* $N \triangleright Q \longrightarrow_l N' \triangleright Q'$ *then* $M \triangleright P \longrightarrow_k^* M' \triangleright P'$ *such that* $\Gamma \vDash (M' \triangleright P')\ \mathcal{R}^{n+l-k}\ (N' \triangleright Q')$.

Related processes must be related under arbitrary (parallel) contexts; moreover, these contexts must be allowed to allocate new channels. We note that the second clause of our contextuality definition, Definition 10, is similar to that discussed earlier in Section 3.3, where we *transfer* the respective permissions held by the observer along with the test R placed in parallel with the processes. This is essential in order to preserve consistency (see Def. 1) thus limiting our analysis to safe computations.

Definition 10 (Contextuality)**.** *An amortized type-indexed relation* $\mathcal{R}$ *is contextual at environment* Γ *and credit* n *iff whenever* $\Gamma \vDash (M \triangleright P)\ \mathcal{R}^n\ (N \triangleright Q)$*:*

1. $\Gamma, c : [\vec{\mathbf{T}}]^\bullet \vDash (M, c \triangleright P)\ \mathcal{R}^n\ (N, c \triangleright Q)$
2. *If* $\Gamma \prec \Gamma_1, \Gamma_2$ *where* $\Gamma_2 \vdash R$ *then*
   - $\Gamma_1 \vDash (M \triangleright P \parallel R)\ \mathcal{R}^n\ (N \triangleright Q \parallel R)$ *and*
   - $\Gamma_1 \vDash (M \triangleright R \parallel P)\ \mathcal{R}^n\ (N \triangleright R \parallel Q)$

We can now define the preorder defining our notion of observational system efficiency:

Definition 11 (Behavioral Contextual Preorder)**.** $\lesssim_{beh}^{\Gamma,n}$ *is the largest family of amortized typed relations that is:*

- *Barb Preserving;*
- *Cost Improving at credit n;*
- *Contextual at environment* Γ.

A system $M \triangleright P$ *is said to be behaviourally as efficient as another system* $N \triangleright Q$ *wrt. an observer* Γ*, denoted as* $\Gamma \vDash M \triangleright P \lesssim_{beh} N \triangleright Q$*, whenever there exists an amortisation credit n such that* $\Gamma \vDash M \triangleright P \lesssim_{beh}^{n} N \triangleright Q$*. Similarly, we can lift our preorder to processes: a process P is said to be as efficient as Q wrt. M and* Γ *whenever there exists an n such that* $\Gamma \vDash M \triangleright P \lesssim_{beh}^{n} M \triangleright Q$

4.1 Soundness for $\sqsubseteq_{\mathbf{bis}}$

Through Definition 11 we are able to articulate why clients C_2 and C_2' should be deemed to be behaviourally equally efficient *wrt.* Γ_1 of (21): for an appropriate M, it turns out that we cannot differentiate between the two processes under any context allowed by Γ. Unfortunately, the universal quantification of contexts of Definition 10 makes it hard to verify such a statement. Through Theorem 2 we can however establish that our bisimulation preorder of Definition 6 provides a sound technique for determining behavioural efficiency. This Theorem, in turn, relies on the lemmas we outline below. In particular, Lemma 15 and Lemma 16 prove that bisimulations are barb-preserving and cost-improving, whereas Lemma 17 proves that bisimulations are preserved under resource extensions. The required result then follows from Theorem 1 of Section 3.3.

Lemma 13 (Reduction and Silent Transitions)**.**

1. $M \triangleright P \longrightarrow_k M' \triangleright P'$ *implies* $\Gamma \triangleleft M \triangleright P \xrightarrow{\tau}_k \Gamma \triangleleft M' \triangleright P''$ *for arbitrary* Γ *where* $P'' \equiv P'$.
2. $\Gamma \triangleleft M \triangleright P \xrightarrow{\tau}_k \Delta \triangleleft M' \triangleright P'$ *implies* $(M \triangleright P)\sigma_\Gamma \longrightarrow_k M' \triangleright P'$ *for some* σ_Γ.

Proof. By rule induction on $M \triangleright P \longrightarrow_k M' \triangleright P'$ and $\Gamma \triangleleft M \triangleright P \xrightarrow{\tau}_k \Delta \triangleleft M' \triangleright P'$. □

Lemma 14 (Reductions and Bijective Renaming)**.** *For any bijective renaming* σ*,* $(M \triangleright P)\sigma \longrightarrow_k (M' \triangleright P')\sigma$ *implies* $M \triangleright P \longrightarrow_k M \triangleright P$

Proof. By rule induction on $(M \triangleright P)\sigma \longrightarrow_k (M' \triangleright P')\sigma$. □

Lemma 15 (Barb Preservation)**.**

$$\Gamma \vDash M \triangleright P \sqsubseteq_{\approx bis} N \triangleright Q \text{ and } \Gamma \triangleleft M \triangleright P \Downarrow_c^{barb} \text{ implies } \Gamma \triangleleft N \triangleright Q \Downarrow_c^{barb}$$

Proof. By Definition 7 we know $M \triangleright P \longrightarrow_l^* \equiv (M' \triangleright P' \parallel c!\vec{d}.P'')$ where $c \in d\mathrm{om}(\Gamma)$. By Lemma 13(1) we obtain $\Gamma \triangleleft M \triangleright P \Longrightarrow_l \Gamma \triangleleft M' \triangleright P'''$ where $P''' \equiv (P' \parallel c!\vec{d}.P'')$. Moreover, by LOUT, LPAR-R and Lemma 11 we deduce $\Gamma \triangleleft M \triangleright P \overset{c!\vec{d}}{\Longrightarrow}_l \equiv \Gamma' \triangleleft P' \parallel P''$ By $\Gamma \vDash M \triangleright P \sqsubseteq_{\approx \mathrm{bis}} N \triangleright Q$ we know that there exists a move $\Gamma \triangleleft N \triangleright Q \overset{c!\vec{d}}{\Longrightarrow}_k \Gamma' \triangleleft N' \triangleright Q'$ and from this matching move, Lemma 13(2) (for the initial τ moves of the weak action) and Lemma 1 we obtain $(N \triangleright Q)\sigma_\Gamma \longrightarrow_{k_1}^* \equiv (N'' \triangleright Q'' \parallel c!\vec{d}.Q''')\sigma_\Gamma$, which, together with $c \in d\mathrm{om}(\Gamma)$ and Lemma 14, implies $N \triangleright Q \longrightarrow_{k_1}^* \equiv N'' \triangleright Q'' \parallel c!\vec{d}.Q'''$ *i.e.*, c is unaffected by the renaming σ_Γ, and thus $\Gamma \triangleleft N \triangleright Q \Downarrow_c^{\mathrm{barb}}$. □

Lemma 16 (Cost Improving)**.** $\Gamma \vDash M \triangleright P \sqsubseteq_{\approx bis}^n N \triangleright Q$ *and* $M \triangleright P \longrightarrow_l M' \triangleright P'$ *then there exist some* $N' \triangleright Q'$ *such that* $N \triangleright Q \longrightarrow_k^* N' \triangleright Q'$ *and* $\Gamma \vDash M' \triangleright P' \sqsubseteq_{\approx bis}^{n+k-l} N' \triangleright Q'$

Proof. By $M \triangleright P \longrightarrow_l M' \triangleright P'$ and Lemma 13(1) we know $\Gamma \triangleleft M \triangleright P \xrightarrow{\tau}_l \Gamma \triangleleft M \triangleright P''$ where $P'' \equiv P'$. By Def. 6 and assumption $\Gamma \vDash M \triangleright P \sqsubseteq_{\approx \mathrm{bis}}^n N \triangleright Q$, this implies that $\Gamma \triangleleft N \triangleright Q \Longrightarrow_k \Gamma \triangleleft M' \triangleright Q'$ where

$$\Gamma \vDash M' \triangleright P'' \sqsubseteq_{\approx \mathrm{bis}}^{n+k-l} N' \triangleright Q'. \tag{70}$$

By Lemma 13(2) we deduce $(N \triangleright Q)\sigma_\Gamma \longrightarrow_k^* N' \triangleright Q'$ and by Lemma 14 we obtain $N \triangleright Q \longrightarrow^* N'' \triangleright Q''$ where $N'' \triangleright Q'' = (N' \triangleright Q')\sigma_\Gamma$. The required result follows from $\Gamma \vDash M' \triangleright P' \sqsubseteq_{\approx \mathrm{bis}}^0 M' \triangleright P''$, which we obtain from $P' \equiv P''$ and Corollary 2 (Structural Equivalence and Bisimilarity), (70), $\Gamma \vDash N'' \triangleright Q'' \sqsubseteq_{\approx \mathrm{bis}}^0 N' \triangleright Q'$ which we obtain from Lemma 3 (Reflexivity upto Renaming) and $N'' \triangleright Q'' = (N' \triangleright Q')\sigma_\Gamma$, and Lemma 4. □

Lemma 17 (Resource Extensions)**.**

$$\Gamma \vDash M \triangleright P \sqsubseteq_{\approx bis}^n N \triangleright Q \text{ implies } \Gamma, c : [\vec{\mathbf{T}}]^\bullet \vDash (M, c) \triangleright P \sqsubseteq_{\approx bis}^n (N, c) \triangleright Q$$

Proof. By coinduction. □

Theorem 2 (Soundness)**.** $\Gamma \vDash (M \triangleright P) \sqsubseteq_{\approx bis}^n (N \triangleright Q)$ *implies* $\Gamma \vDash (M \triangleright P) \precsim_{beh}^n (N \triangleright Q)$.

Proof. Follows from Lemma 15 (Barb Preservation), Lemma 16 (Cost Improving), Lemma 17 (Environment Weakening) and Theorem 1 (Contextuality). □

Corollary 3 (Soundness)**.** $\Gamma \vDash (M \triangleright P) \sqsubseteq_{\approx bis} (N \triangleright Q)$ *implies* $\Gamma \vDash (M \triangleright P) \precsim_{beh} (N \triangleright Q)$.

4.2 Full Abstraction of $\precsim_{\mathbf{beh}}$

Completeness, *i.e.,* for every behavioural contextual preorder there exists a corresponding amortised typed-bisimulation, relies on the adapted notion of *action definability* [Hen08, HR04], which intuitively means that every action (label) used by our LTS can, in some sense, be simulated (observed) by a specific test context. In our specific case, two important aspects need to be taken into consideration:

- the *typeability* of the testing context *wrt.* our substructural type system;
- the *cost* of the action simulation, which has to correspond to the cost of the action being observed.

These aspects are formalised in Definition 12, which relies on the functions definitions *d*oml and *c*odl, taking a substructural type environment and returning respectively a *list* of channel names and a *list* of types:

$$\begin{array}{rr} \mathit{d}\mathrm{oml}(\epsilon) \stackrel{\text{def}}{=} \epsilon & \mathit{c}\mathrm{odl}(\epsilon) \stackrel{\text{def}}{=} \epsilon \\ \mathit{d}\mathrm{oml}(\Gamma, c:\mathbf{T}) \stackrel{\text{def}}{=} \mathit{d}\mathrm{oml}(\Gamma), c & \mathit{c}\mathrm{odl}(\Gamma, c:\mathbf{T}) \stackrel{\text{def}}{=} \mathit{c}\mathrm{odl}(\Gamma), \mathbf{T} \end{array}$$

Before stating cost-definability for actions, Definition 12, we prove the technical Lemma 18 which allows us to express transitions in a convenient format for the respective definition without loss of generality.

Lemma 18 (Transitions and Renaming)**.** $\Gamma \triangleleft M \triangleright P \xrightarrow{\mu}_k \Gamma' \triangleleft M' \triangleright P'$ *if and only if* $\Gamma \triangleleft M \triangleright P \xrightarrow{\mu}_k (\Gamma'' \triangleleft M'' \triangleright P'')\sigma_\Gamma$ *for some* $\sigma_\Gamma, \Gamma'', M'', P''$ *where* $\Gamma' = \Gamma''\sigma_\Gamma$, $M' = M''\sigma_\Gamma$ *and* $P' = P''\sigma_\Gamma$.

Proof. The *if* case is immediate. The proof for the *only-if* is complicated by actions that perform channel allocation (see lAll and lAllE from Fig. 5) because, in such cases, the renaming used in lRen's premise cannot be used directly. More precisely, from the premise we know:

$$\frac{\Gamma \triangleleft (M \triangleright P)\sigma_\Gamma \xrightarrow{\mu}_k \Gamma' \triangleleft M' \triangleright P'}{\Gamma \triangleleft M \triangleright P \xrightarrow{\mu}_k \Gamma' \triangleleft M' \triangleright P'}\ \textsc{lRen}$$

and the required result follows if we prove the (slightly more cumbersome) sublemma:

Sublemma (Transition and Renaming)**.** $\Gamma \triangleleft (M \triangleright P)\sigma_\Gamma \xrightarrow{\mu}_k \Gamma' \triangleleft M' \triangleright P'$ *where* $\mathrm{fn}(P) \subseteq M$ *implies* $\Gamma \triangleleft (M \triangleright P)\sigma_\Gamma \xrightarrow{\mu}_k (\Gamma'' \triangleleft M'' \triangleright P'')\sigma'_\Gamma$ *for some* $\sigma'_\Gamma, \Gamma'', M'', P''$ *where*

- $\Gamma' = \Gamma''\sigma'_\Gamma$, $M' = M''\sigma'_\Gamma$ *and* $P' = P''\sigma'_\Gamma$;
- $c \in \mathit{d}\mathrm{om}(M)$ *implies* $\sigma_\Gamma(c) = \sigma'_\Gamma(c)$

The above sublemma is proved by rule induction on $\Gamma \triangleleft (M \triangleright P)\sigma_\Gamma \xrightarrow{\mu}_k \Gamma' \triangleleft M' \triangleright P'$. We show one of the main cases:

lAll We have $\Gamma \triangleleft (M \triangleright \texttt{alloc}\, x.P)\sigma_\Gamma \xrightarrow{\tau}_{+1} \Gamma \triangleleft ((M)\sigma_\Gamma, c) \triangleright ((P)\sigma_\Gamma\{{}^{c}/_{x}\})$. From the fact that $c \notin (M\sigma_\Gamma)$ — it follows because $((M)\sigma_\Gamma, c)$ is defined — we know that $\sigma_\Gamma^{-1}(c) \notin M$. We thus choose some fresh channel d, *i.e.*, $d \notin (M \cup (M\sigma_\Gamma) \cup d\mathrm{om}(\Gamma))$[8], and define σ'_Γ as σ_Γ, except that it maps d to c and (in order to preserve bijectivity) $\sigma_\Gamma^{-1}(c)$, *i.e.*, the channel name that mapped to c in σ_Γ, to $\sigma_\Gamma(d)$, since this channel is not mapped to by d anymore:

$$\sigma'_\Gamma(x) \stackrel{\text{def}}{=} \begin{cases} c & \text{if } x = d \\ \sigma_\Gamma(d) & \text{if } x = \sigma_\Gamma^{-1}(c) \\ \sigma_\Gamma(x) & \text{otherwise} \end{cases}$$

We subsequently define

- Γ'' as Γ since $\Gamma\sigma'_\Gamma = \Gamma\sigma_\Gamma = \Gamma$;
- M'' as M, d since $(M, d)\sigma'_\Gamma = ((M)\sigma_\Gamma, c)$; and
- P'' as $P\{{}^{d}/_{x}\}$ since $P\{{}^{d}/_{x}\}\sigma'_\Gamma = P\sigma_\Gamma\{{}^{c}/_{x}\}$

□

Definition 12 (Cost Definable Actions)**.** *An action μ is cost-definable iff for any pair of type environments*[9] *Γ and Γ', a corresponding substitution σ_Γ, set of channel names $C \in$ Chan, and channel names $\texttt{succ}, \texttt{fail} \notin C$, there exists a test R such that $\Gamma, \texttt{succ}:[\mathrm{codl}(\Gamma')]^{\mathbf{1}}, \texttt{fail}:[]^{\mathbf{1}}, \texttt{fail}:[]^{\mathbf{1}} \vdash R$ and whenever $M \in C$:*

1. *$\Gamma \triangleleft M \triangleright P \xrightarrow{\mu}_k (\Gamma' \triangleleft M' \triangleright P')\sigma_\Gamma$ implies $M, \texttt{succ}, \texttt{fail} \triangleright P \longrightarrow_k^* M', \texttt{succ}, \texttt{fail} \triangleright P' \parallel \texttt{succ}!(d\mathrm{oml}(\Gamma'))$.*
2. *$M, \texttt{succ}, \texttt{fail} \triangleright P \parallel R \longrightarrow_k^* M'' \triangleright P''$ where $\texttt{succ}:[\mathrm{codl}(\Gamma')]^a, \texttt{fail}:[]^a \triangleleft M'' \triangleright P'' \Downarrow^{barb}_{\texttt{fail}}$ and $\texttt{succ}:[\mathrm{codl}(\Gamma')]^a, \texttt{fail}:[]^a \triangleleft M'' \triangleright P'' \Downarrow^{barb}_{\texttt{succ}}$ implies $\Gamma \triangleleft M \triangleright P \stackrel{\mu}{\Longrightarrow}_k (\Gamma' \triangleleft M' \triangleright P')\sigma_\Gamma$ where $M'' = M', \texttt{succ}, \texttt{fail}$ and $P'' \equiv P'' \parallel \texttt{succ}!(d\mathrm{oml}(\Gamma'))$.*

Lemma 19 (Action Cost-Definability)**.** *External actions $\mu \in \left\{c!\vec{d}, c?\vec{d}, \texttt{alloc}, \texttt{free}\, c \mid c, \vec{d} \subset \text{Chan}\right\}$ are cost-definable.*

Proof. The witness tests for $c!\vec{d}$ and $c?\vec{d}$ are reasonably standard (see [Hen08]), but need to take into account permission transfer. For instance, for the specific case of the action $c!d$ where $d \notin d\mathrm{oml}(\Gamma)$, if the transition $\Gamma \triangleleft M \triangleright P \xrightarrow{\mu}_k (\Gamma' \triangleleft M' \triangleright P')\sigma_\Gamma$ holds then we we know that, for some Γ_1 and $[\mathbf{T}]^a$:

[8]The condition that $d \notin d\mathrm{om}(\Gamma)$ is required since we do not state whether the triple $\Gamma \triangleleft M \triangleright P$ is a configuration; otherwise, it is redundant — see comments succeeding Definition 2.

[9]Cost Definability cannot be defined *wrt.* the first environment only in the case of action `alloc`, since it non-deterministically allocates a fresh channel name and adds it to the residual environment - see lAllE in Fig. 5.

- $\Gamma = \Gamma_1, c : [\mathbf{T}]^a$;
- $\Gamma'\sigma_\Gamma = \Gamma_1, c : [\mathbf{T}]^{a-1}, d : \mathbf{T}$

In particular, when $a = \mathbf{1}$ (affine), using the permission to input on c implicitly transfers the permission to process P (see Section 3.1), potentially revoking the test's capability to perform name matching on channel name c (see TIF in Fig. 4) — this happens if $c \notin \mathit{dom}(\Gamma_1)$. For this reason, when $a = \mathbf{1}$ the test is defined as

$$\texttt{fail!} \parallel c?x.\texttt{if}\ (x \in \mathit{doml}(\Gamma_1))\ \texttt{then nil else fail?.succ!}(\mathit{doml}(\Gamma'))$$

where $x \in \mathit{doml}(\Gamma_1)$ is shorthand for a sequence of name comparisons as in [Hen08]. Otherwise, the respective type assumption is not consumed from the observer environment and the test is defined as

$$\texttt{fail!} \parallel c?x.\texttt{if}\ (x \in \mathit{doml}(\Gamma))\ \texttt{then nil else fail?.succ!}(\mathit{doml}(\Gamma'))$$

Note that name comparisons on freshly acquired names are typeable since we also obtain the respective permissions upon input, *i.e.*, the explicit permission transfer (see Section 3.1.) The reader can verify that these tests typecheck *wrt.* the environment $\Gamma, \texttt{succ} : [\mathit{codl}(\Gamma')]^{\mathbf{1}}, \texttt{fail} : []^{\mathbf{1}}, \texttt{fail} : []^{\mathbf{1}}$ and that they observe clauses (1) and (2) of Definition 12. In the case of clause (2), we note that from the typing of the tests above, we know that $c \in \mathit{doml}(\Gamma)$ must hold (because both tests use channel c for input); this is is a key requirement for the transition to fire — see LOUT of Fig. 5.

The witness tests for `alloc` and `free` c involve less intricate permission transfer and are respectively defined as:

$$\texttt{fail!} \parallel \texttt{alloc}\ x.\texttt{fail?.succ!}(\mathit{doml}(\Gamma), x)$$

and

$$\texttt{fail!} \parallel \texttt{free}\ c.\texttt{fail?.succ!}(\mathit{doml}(\Gamma'))$$

We here focus on `alloc` and leave the analogous proof for `free` c for the interested reader:

1. If $\Gamma \triangleleft M \triangleright P \xrightarrow{\texttt{alloc}}_k (\Gamma' \triangleleft M' \triangleright P')\sigma_\Gamma$ we know that, for some $d \notin M$ and $c \notin M\sigma_\Gamma$ where $\sigma_\Gamma(d) = c$, we have $(\Gamma')\sigma_\Gamma = (\Gamma, d : [\mathbf{T}]^\bullet)\sigma_\Gamma = \Gamma, c : [\mathbf{T}]^\bullet$, $M' = (M, d)$ and $P' = P$. We can therefore simulate this action by the following sequence of reductions:

$$M \triangleright P \parallel \texttt{fail!} \parallel \texttt{alloc}\ x.\texttt{fail?.succ!}(\mathit{doml}(\Gamma), x) \longrightarrow$$
$$M, d \triangleright P \parallel \texttt{fail!} \parallel \texttt{fail?.succ!}(\mathit{doml}(\Gamma), d) \longrightarrow M, d \triangleright P \parallel \texttt{succ!}(\mathit{doml}(\Gamma), d)$$

2. From the structure of R and the assumption that $\texttt{fail}, \texttt{succ} \notin \mathit{fn}(P)$, we know that, if $\texttt{succ} : [\mathit{codl}(\Delta)]^a, \texttt{fail} : []^a \triangleleft M' \triangleright P' \not\Downarrow^{\mathrm{barb}}_{\texttt{fail}}$ and $\texttt{succ} : [\mathit{codl}(\Delta)]^a, \texttt{fail} : []^a \triangleleft M' \triangleright P' \Downarrow^{\mathrm{barb}}_{\texttt{succ}}$, then it must be the case that, for some $d \notin M$, $P' = P'' \parallel \texttt{succ!}(\mathit{doml}(\Gamma), d)$ where $M'' = (M', \texttt{succ}, \texttt{fail}, d)$ for some M'.

   Since P and R do not share common channels there could not have been any interaction between the two processes in the reduction sequence $M, \texttt{succ}, \texttt{fail} \triangleright P \parallel R \longrightarrow^*_k M' \triangleright P'$.

Within this reduction sequence, from every reduction $M_i \triangleright P_i \parallel R' \longrightarrow_{k_i} M_{i+1} \triangleright P_{i+1} \parallel R'$ resulting from derivatives of P, *i.e.,* $M_i \triangleright P_i \longrightarrow_{k_i} M_{i+1} \triangleright P_{i+1}$ that happened before the allocation of channel d, we obtain a corresponding silent transition

$$\Gamma_i \triangleleft (M_i \setminus \{\mathtt{succ}, \mathtt{fail}\}) \triangleright P_i \xrightarrow{\tau}_{k_i} \Gamma_i \triangleleft (M_{i+1} \setminus \{\mathtt{succ}, \mathtt{fail}\}) \triangleright P_{i+1} \tag{71}$$

by Lemma 13(1) and an appropriate lemma that uses the fact $\{\mathtt{succ}, \mathtt{fail}\} \cap fn(P) = \emptyset$ to allows us to shrink the allocated resources from M_i to $(M_i \setminus \{\mathtt{succ}, \mathtt{fail}\})$. A similar procedure can be carried out for reductions that happened after the allocation of d as a result of reductions from P derivatives, and by applying renaming σ_Γ we can obtain

$$(\Gamma_i \triangleleft (M_i \setminus \{\mathtt{succ}, \mathtt{fail}\}) \triangleright P_i)\sigma_\Gamma \xrightarrow{\tau}_{k_i} (\Gamma_i \triangleleft (M_{i+1} \setminus \{\mathtt{succ}, \mathtt{fail}\}) \triangleright P_{i+1})\sigma_\Gamma \tag{72}$$

The reduction

$$M_i, \mathtt{succ}, \mathtt{fail} \triangleright P_i \parallel \mathtt{alloc}\, x.\mathtt{fail}?.\mathtt{succ}!(d\mathrm{oml}(\Gamma), x) \longrightarrow_{+1} M_i, \mathtt{succ}, \mathtt{fail}, d \triangleright P_i \parallel \mathtt{fail}?.\mathtt{succ}!(d\mathrm{oml}(\Gamma), d)$$

can be substituted by the transition

$$\Gamma_i \triangleleft M_i \triangleright P_i \xRightarrow{\mathtt{alloc}}_{+1} \Gamma_i, (d)\sigma_\Gamma : [\mathbf{T}]^\bullet \triangleleft ((M_i)\sigma_\Gamma, (d)\sigma_\Gamma) \triangleright (P_i)\sigma_\Gamma \tag{73}$$

This follows from the fact that $d \notin M_i$ and the fact that σ_Γ is a bijection, which implies that $(d)\sigma_\Gamma \notin (M_i)\sigma_\Gamma$ (necessary for $((M_i)\sigma_\Gamma, (d)\sigma_\Gamma)$ to be a valid resource environment). By joining together the transitions from (71), (73) and (72) in the appropriate sequence we obtain the required weak transition.

□

The proof of Theorem 3(Completeness) relies on Lemma 19 to simulate a costed action by the appropriate test and is, for the most part, standard. As stated already, one novel aspect is that the cost semantics requires the simulation to incur the same cost as that of the costed action. Through Reduction Closure, Lemma 19 again and then finally the Extrusion Lemma 20 we then obtain the matching bisimulation move which preserves the relative credit index. Another novel aspect of the proof for Theorem 3 is that the name matching in the presence of our substructural type environment requires a reformulation of the Extrusion Lemma. More precisely, in the case of the output actions, the simulating test requires all of the environment permissions to perform all the necessary name comparisons. We then make sure that these permissions are not lost by communicating them all again on succ; this passing on of permissions then allows us to show contextuality in Lemma 20.

Lemma 20 (Extrusion)**.** *Whenever* $\Gamma \triangleleft M \triangleright P$ *and* $\Gamma \triangleleft N \triangleright Q$ *are configurations and* $\vec{d} \notin d\mathrm{om}(\Gamma)$*:*

$$succ : [\mathrm{codl}(\Gamma)]^{(\bullet,1)} \vDash (M, succ, \vec{d}) \triangleright P \parallel succ!(d\mathrm{oml}(\Gamma)) \lesssim^n_{beh} (N, succ, \vec{d}) \triangleright Q \parallel succ!(d\mathrm{oml}(\Gamma))$$

implies $\Gamma \vdash M \triangleright P \lesssim^n_{beh} N \triangleright Q$

Proof. By coinduction we show that a family of amortized typed relations $\Gamma \vdash M \triangleright P \; \mathcal{R}^n \; N \triangleright Q$ observes the required properties of Def. 11. Note that the environment $\mathtt{succ} : [\mathrm{codl}(\Gamma)]^{(\bullet,1)}$ ensures that $\mathtt{succ} \notin n\mathrm{ames}(P, Q)$ since both $P \parallel \mathtt{succ}!(d\mathrm{oml}(\Gamma))$ and $Q \parallel \mathtt{succ}!(d\mathrm{oml}(\Gamma))$ must typecheck *wrt.* a type environment that is consistent with $\mathtt{succ} : [\mathrm{codl}(\Gamma)]^{(\bullet,1)}$. Cost improving is straightforward and Barb Preserving and Contextuality follow standard techniques; see [Hen08].

For instance, for barb preservation we are required to show that $\Gamma \triangleleft M \triangleright P \Downarrow_c^{\mathrm{barb}}$ implies $\Gamma \triangleleft N \triangleright Q \Downarrow_c^{\mathrm{barb}}$ (and viceversa). From $\Gamma \triangleleft M \triangleright P \Downarrow_c^{\mathrm{barb}}$ and Definition 7 we know that $c : [\vec{\mathbf{T}}]^a \in \Gamma$ at some index i. We can therefore define the process $R \triangleq succ?\vec{x}.x_i?\vec{y}.ok!$ where $|\vec{\mathbf{T}}| = |\vec{y}|$; this test process typechecks *wrt.* $succ : [\mathrm{codl}(\Gamma)]^{(\bullet,1)}, ok : []^{\mathbf{1}}$. Now by Definition 10(1) we know

$$succ : [\mathrm{codl}(\Gamma)]^{(\bullet,1)}, ok : []^{\bullet} \vDash (M, succ, \vec{d}, ok) \triangleright P \parallel succ!(d\mathrm{oml}(\Gamma)) \lesssim^n_{\mathrm{beh}} (N, succ, \vec{d}, ok) \triangleright Q \parallel succ!(d\mathrm{oml}(\Gamma))$$

and thus, by Definition 10(2) and $succ : [\mathrm{codl}(\Gamma)]^{(\bullet,1)}, ok : []^{\mathbf{1}} \vdash R$

$$ok : []^{(\bullet,1)} \vDash (M, succ, \vec{d}, ok) \triangleright P \parallel succ!(d\mathrm{oml}(\Gamma)) \parallel R \lesssim^n_{\mathrm{beh}} (N, succ, \vec{d}, ok) \triangleright Q \parallel succ!(d\mathrm{oml}(\Gamma)) \parallel R \tag{74}$$

Clearly, if $\Gamma \triangleleft M \triangleright P \Downarrow_c^{\mathrm{barb}}$ then $(ok : []^{(\bullet,1)} \triangleleft (M, succ, \vec{d}, ok) \triangleright (P \parallel succ!(d\mathrm{oml}(\Gamma)) \parallel R)) \Downarrow_{ok}^{\mathrm{barb}}$. By (74) and Definition 8 we must have $(ok : []^{(\bullet,1)} \triangleleft (N, succ, \vec{d}, ok) \triangleright (Q \parallel succ!(d\mathrm{oml}(\Gamma)) \parallel R)) \Downarrow_{ok}^{\mathrm{barb}}$ as well, which can only happen if $N \triangleright Q \longrightarrow^* \equiv Q' \parallel c!\vec{d}.Q''$. This means that $\Gamma \triangleleft N \triangleright Q \Downarrow_c^{\mathrm{barb}}$. □

Lemma 21. *$\Gamma \vDash M \triangleright P \lesssim^n_{beh} N \triangleright Q$ and $\Gamma \prec \Gamma'$ implies $\Gamma' \vDash M \triangleright P \lesssim^n_{beh} N \triangleright Q$*

Proof. By coinduction. □

Lemma 22. *$\Gamma \vDash M \triangleright P \lesssim^n_{beh} N \triangleright Q$ and σ is a bijective renaming implies $\Gamma\sigma \vDash (M \triangleright P)\sigma \lesssim^n_{beh} (N \triangleright Q)\sigma$*

Proof. By coinduction. □

Theorem 3 (Completeness)**.** *$\Gamma \vDash (M \triangleright P) \lesssim^n_{beh} (N \triangleright Q)$ implies $\Gamma \vDash (M \triangleright P) \precsim^n_{bis} (N \triangleright Q)$.*

Proof. By coinduction, we show that for arbitrary Γ, n, the family of relations included in $\Gamma \vDash M \triangleright P \lesssim^n_{\mathrm{beh}} N \triangleright Q$ observes the transfer properties of Def. 6 at Γ, n. Assume

$$\Gamma \triangleleft M \triangleright P \xrightarrow{\mu}_k (\Gamma' \triangleleft M' \triangleright P')\sigma_\Gamma \tag{75}$$

If $\mu = \tau$, the matching move follows from Lemma 13, Definition 9 and Definition 11.

If $\mu \in \left\{c!\vec{d}, c?\vec{d}, \mathtt{alloc}, \mathtt{free}\, c \mid c, \vec{d} \in \textsc{Chan}\right\}$, by Lemma 19 we know that there exists a test process that can simulate it; we choose one such test R with channel names $\mathtt{succ}, \mathtt{fail} \notin M, N$. By Definition 10(1) we know

$$\Gamma, \mathtt{succ} : [\mathrm{codl}(\Gamma)]^{\bullet}, \mathtt{fail} : []^{\bullet} \vDash M, \mathtt{succ}, \mathtt{fail} \triangleright P \lesssim^n_{\mathrm{beh}} N, \mathtt{succ}, \mathtt{fail} \triangleright Q$$

and by Definition 10(2) and $\Gamma, \mathtt{succ}:[\mathit{codl}(\Delta)]^{\mathbf{1}}, \mathtt{fail}:[]^{\mathbf{1}}, \mathtt{fail}:[]^{\mathbf{1}} \vdash R$ (Definition 12) we obtain

$$\mathtt{succ}:[\mathit{codl}(\Gamma)]^{(\bullet,1)}, \mathtt{fail}:[]^{(\bullet,2)} \vDash (M, \mathtt{succ}, \mathtt{fail}) \triangleright P \parallel R \lesssim^{n}_{\mathrm{beh}} (N, \mathtt{succ}, \mathtt{fail}) \triangleright Q \parallel R \quad (76)$$

From (75) and Definition 12(1), we know

$$(M, \mathtt{succ}, \mathtt{fail}) \triangleright P \parallel R \longrightarrow^{*}_{k} (M', \mathtt{succ}, \mathtt{fail}) \triangleright P' \parallel \mathtt{succ}!\, \mathit{doml}(\Gamma')$$

By (76) and Definition 9 (Cost Improving) we know

$$(N, \mathtt{succ}, \mathtt{fail}) \triangleright Q \parallel R \longrightarrow^{*}_{l} N'' \triangleright Q''$$

where

$$\mathtt{succ}:[\mathit{codl}(\Gamma)]^{(\bullet,1)}, \mathtt{fail}:[]^{(\bullet,2)} \vDash (M', \mathtt{succ}, \mathtt{fail}) \triangleright P' \parallel \mathtt{succ}!\, \mathit{doml}(\Gamma') \lesssim^{n+l-k}_{\mathrm{beh}} N'' \triangleright Q'' \quad (77)$$

By Definition 8 (Barb Preservation), this means that $\mathtt{succ}:[\mathit{codl}(\Gamma)]^{(\bullet,1)}, \mathtt{fail}:[]^{(\bullet,2)} \triangleleft N' \triangleright Q' \Downarrow^{\mathrm{barb}}_{\mathtt{fail}}$ and also that $\mathtt{succ}:[\mathit{codl}(\Gamma)]^{(\bullet,1)}, \mathtt{fail}:[]^{(\bullet,2)} \triangleleft N' \triangleright Q' \Downarrow^{\mathrm{barb}}_{\mathtt{succ}}$. By Definition 12(2) we obtain

$$Q'' \equiv Q' \parallel \mathtt{succ}!\, \mathit{doml}(\Gamma') \text{ and } N'' = (N', \mathtt{succ}, \mathtt{fail}) \quad (78)$$

$$\Gamma \triangleleft N \triangleright Q \stackrel{\mu}{\Longrightarrow}_{l} (\Gamma' \triangleleft N' \triangleright Q')\sigma_{\Gamma} \quad (79)$$

Transition (79) is the matching move because by (77) and Lemma 21 we obtain

$$\mathtt{succ}:[\mathit{codl}(\Gamma)]^{(\bullet,1)} \vDash (M', \mathtt{succ}, \mathtt{fail}) \triangleright P' \parallel \mathtt{succ}!\, \mathit{doml}(\Gamma') \lesssim^{n+l-k}_{\mathrm{beh}} N'' \triangleright Q''$$

By (78), and Lemma 20 we obtain $\Gamma' \vDash M' \triangleright P' \lesssim^{n+l-k}_{\mathrm{beh}} N' \triangleright Q'$ and subsequently by Lemma 22 we get

$$\Gamma'\sigma_{\Gamma} \vDash (M' \triangleright P')\sigma_{\Gamma} \lesssim^{n+l-k}_{\mathrm{beh}} (N' \triangleright Q')\sigma_{\Gamma}$$

□

5 Case Study

Resource management is particularly relevant to programs manipulating (unbounded) regular structures. We consider the concurrent implementation of an unbounded buffer, Buff, receiving values to queue on channel `in` and dequeuing values by outputting on channel `out`.

$$\begin{aligned}
\mathrm{Buff} &\stackrel{\mathbf{def}}{=} \mathtt{in}?y.\ \mathtt{alloc}\, z.\ (\mathrm{Frn} \parallel b!z \parallel c_1!(y,z)) \quad \parallel \quad c_1?(y,z).\ \mathtt{out}!y.\ (\mathrm{Bck} \parallel d!z) \\
\mathrm{Frn} &\stackrel{\mathbf{def}}{=} \mathtt{rec}\, w.\ b?x.\ \mathtt{in}?y.\mathtt{alloc}\, z.\ (w \parallel b!z \parallel x!(y,z)) \\
\mathrm{Bck} &\stackrel{\mathbf{def}}{=} \mathtt{rec}\, w.\ d?x.\ x?(y,z).\ \mathtt{out}!y.\ (w \parallel d!z)
\end{aligned}$$

In order to decouple input requests from output requests while still preserving the order of inputted values, the process handling inputs in Buff, $\mathtt{in}?y.\mathtt{alloc}\, z.(\mathrm{Frn} \parallel b!z \parallel c_1!(y,z))$, stores inputted values $v_1, \ldots, v_n$ as a queue of interconnected outputs

$$c_1!(v_1, c_2) \parallel \ldots \parallel c_n!(v_n, c_{n+1}) \quad (80)$$

on the internal channels $c_1, \ldots, c_{n+1}$. The process handling the outputs , $c_1?(y,z).\mathtt{out}!y.(\text{Bck} \parallel d!z)$, then reads from the head of this queue, *i.e.,* the output on channel c_1, so as to obtain the first value inputted, v_1, and the next head of the queue, c_2. The input and output processes are defined in terms of the recursive processes, Frn and Bck *resp.*, which are parameterised by the channel to output (*resp.* input) on next through the channels b and d.[10]

Since the buffer is unbounded, the number of internal channels used for the queue of interconnected outputs, (80), is not fixed and these channels cannot therefore be created up front. Instead, they are created on demand by the input process for every value inputted, using the $R\pi$ construct $\mathtt{alloc}\, z.P$. The newly allocated channel z is then passed on the next iteration of Frn through channel b, $b!z$, and communicated as the next head of the queue when adding the subsequent queue item; this is received by the output process when it inputs the value at the head of the chain and passed on the next iteration of Bck through channel d, $d!z$.

5.1 Typeability and behaviour of the Buffer

Our unbounded buffer implementation, Buff, can be typed *wrt.* the type environment

$$\Gamma_{\text{int}} \stackrel{\text{def}}{=} \mathtt{in}:[\mathbf{T}]^{\omega},\ \mathtt{out}:[\mathbf{T}]^{\omega},\ b:[\mathbf{T}_{\text{rec}}]^{\omega},\ d:[\mathbf{T}_{\text{rec}}]^{\omega},\ c_1:[\mathbf{T},\mathbf{T}_{\text{rec}}]^{\bullet} \tag{81}$$

where $\mathbf{T}$ is the type of the values stored in the buffer and $\mathbf{T}_{\text{rec}}$ is a recursive type defined as

$$\mathbf{T}_{\text{rec}} \stackrel{\text{def}}{=} \mu X.[\mathbf{T}, X]^{(\bullet,1)}.$$

This recursive type is used to type the internal channels $c_1, \ldots, c_{n+1}$ — recall that in (80) these channels carry channels of the same kind in order to link to one another as a chain of outputs. In particular, using the typing rules of Sec. 2 we can prove the following typing judgements:

$$\mathtt{in}:[\mathbf{T}]^{\omega},\ b:[\mathbf{T}_{\text{rec}}]^{\omega},\ c_1:[\mathbf{T},\mathbf{T}_{\text{rec}}]^{\mathbf{1}} \vdash \mathtt{in}?y.\ \mathtt{alloc}\, z.\ (\text{Frn} \parallel b!z \parallel c_1!(y,z)) \tag{82}$$

$$\mathtt{out}:[\mathbf{T}]^{\omega},\ d:[\mathbf{T}_{\text{rec}}]^{\omega},\ c_1:[\mathbf{T},\mathbf{T}_{\text{rec}}]^{(\bullet,1)} \vdash c_1?(y,z).\ \mathtt{out}!y.\ (\text{Bck} \parallel d!z) \tag{83}$$

From the perspective of user of unbounded buffer, Buff implements the interface defined by the environment

$$\Gamma_{\text{ext}} \stackrel{\text{def}}{=} \mathtt{in}:[\mathbf{T}]^{\omega}, \mathtt{out}:[\mathbf{T}]^{\omega}$$

abstracting away from the implementation channels b, d and c_1.

Example 4. *The transition rules of Fig. 5 allow us to derive the following behaviour for the config-*

[10] This models parametrisable process definitions Frn (x) and Bck (x) within our language.

uration $\Gamma_{ext} \triangleleft M, c_1 \triangleright \text{Buff}$ *(where* $\texttt{in}, \texttt{out}, b, d \in M$*):*

$$\Gamma_{ext} \triangleleft M, c_1 \triangleright \text{Buff} \xrightarrow{\texttt{in}?v_1}_0 \Gamma_{ext} \triangleleft M, c_1 \triangleright \left(\begin{array}{l} \texttt{alloc}\, z.(\text{Frn} \parallel b!z \parallel c_1!(v_1, z)) \\ \parallel \ c_1?(y,z).\texttt{out}!y.(\text{Bck} \parallel d!z) \end{array} \right) \tag{84}$$

$$\xrightarrow{\tau}_{+1} \Gamma_{ext} \triangleleft M, c_1, c_2 \triangleright \left(\begin{array}{l} (\text{Frn} \parallel b!c_2 \parallel c_1!(v_1, c_2)) \\ \parallel \ c_1?(y,z).\texttt{out}!y.(\text{Bck} \parallel d!z) \end{array} \right) \tag{85}$$

$$= \Gamma_{ext} \triangleleft M, c_1, c_2 \triangleright \left(\begin{array}{l} \texttt{rec}\, w.\ b?x.\ \texttt{in}?y.\texttt{alloc}\, z.\ (w \parallel b!z \parallel x!(y,z)) \\ \parallel \ b!c_2 \ \parallel \ c_1!(v_1, c_2) \\ \parallel \ c_1?(y,z).\texttt{out}!y.(\text{Bck} \parallel d!z) \end{array} \right)$$

$$\xrightarrow{\tau}_0 \Gamma_{ext} \triangleleft M, c_1, c_2 \triangleright \left(\begin{array}{l} b?x.\ \texttt{in}?y.\texttt{alloc}\, z.\ (\text{Frn} \parallel b!z \parallel x!(y,z)) \\ \parallel \ b!c_2 \ \parallel \ c_1!(v_1, c_2) \\ \parallel \ c_1?(y,z).\texttt{out}!y.(\text{Bck} \parallel d!z) \end{array} \right) \tag{86}$$

$$\xrightarrow{\tau}_0 \Gamma_{ext} \triangleleft M, c_1, c_2 \triangleright \left(\begin{array}{l} \texttt{in}?y.\texttt{alloc}\, z.(\text{Frn} \parallel b!z \parallel c_2!(y,z)) \\ \parallel \ c_1!(v_1, c_2) \\ \parallel \ c_1?(y,z).\texttt{out}!y.(\text{Bck} \parallel d!z) \end{array} \right) \tag{87}$$

$$\xRightarrow{\texttt{in}?v_2}_{+1} \Gamma_{ext} \triangleleft M, c_1, c_2, c_3 \triangleright \left(\begin{array}{l} \texttt{in}?y.\texttt{alloc}\, z.(\text{Frn} \parallel b!z \parallel c_3!(y,z)) \\ \parallel \ c_1!(v_1, c_2) \parallel c_2!(v_2, c_3) \\ \parallel \ c_1?(y,z).\texttt{out}!y.(\text{Bck} \parallel d!z) \end{array} \right) \tag{88}$$

$$\xRightarrow{\texttt{out}!v_1}_0 \Gamma_{ext} \triangleleft M, c_1, c_2, c_3 \triangleright \left(\begin{array}{l} \texttt{in}?y.\texttt{alloc}\, z.(\text{Frn} \parallel b!z \parallel c_3!(y,z)) \\ \parallel \ c_2!(v_2, c_3) \\ \parallel \ c_2?(y,z).\texttt{out}!y.(\text{Bck} \parallel d!z) \end{array} \right) \tag{89}$$

Transition (84) *describes an input from the user whereas* (85) *allocates a new internal channel,* c_2*, followed by a recursive process unfolding,* (86)*, and the instantiation of the unfolded process with the newly allocated channel* c_2*,* (87)*, through a communication on channel* b*. The weak transition* (88) *is an aggregation of 4 analogous transitions to the ones just presented, this time relating to a second input of value* v_2*. This yields an internal output chain of length 2, i.e.,* $c_1!(v_1, c_2) \parallel c_2!(v_2, c_3)$*. Finally,* (89) *is an aggregation of 4 transitions relating to the consumption of the first item in the chain,* $c_1!(v_1, c_2)$*, the subsequent output of* v_1 *on channel* $\texttt{out}$*, and the unfolding and instantiation of the recursive process* Bck *with* c_2 *— see definition for* Bck*.*

5.2 A resource-conscious Implementation of the Buffer

It is worth highlighting at (89) that, whereas channel c_1 is not present in either the free channels used in by the configuration process or the channel names available to observer in the configuration environment, the channel is still allocated in the resource environment, *i.e.*, $c_1 \in (M, c_1, c_2, c_3)$. This fact will repeat itself for every value that is stored in, and later retrieved from, the unbounded buffer and amounts to the equivalent of a *"memory leak"*. A more resource-conscious implementation of the unbounded buffer is eBuff, defined in terms of the previous input process used for Buff, and a modified output process, $c_1?(y,z).\texttt{free}\, c.\texttt{out}!y.(\text{eBk} \parallel d!z)$, which uses the tweaked recursive

process, eBk.

$$\text{eBuff} \stackrel{\text{def}}{=} \texttt{in}?y.\texttt{alloc}\, z.(\text{Frn} \parallel b!z \parallel c_1!(y,z)) \parallel c_1?(y,z).\texttt{free}\, c_1.\texttt{out}!y.(\text{eBk} \parallel d!z)$$
$$\text{eBk} \stackrel{\text{def}}{=} \texttt{rec}\, w.\ d?x.\ x?(y,z).\ \texttt{free}\, x.\ \texttt{out}!y.(w \parallel d!z)$$

The main difference between Buff and eBuff is that the latter deallocates the channel at the head of the internal chain once it is consumed. We can typecheck eBuff as safe since no other process uses the internal channels making up the chain after deallocation. More specifically, the typeability of eBuff *wrt.* Γ_{int} of (81) follows from (82) and the type judgement below:

$$\texttt{out}:[\mathbf{T}]^{\omega},\ d:[\mathbf{T}_{\text{rec}}]^{\omega},\ c_1:[\mathbf{T},\mathbf{T}_{\text{rec}}]^{(\bullet,1)} \ \vdash\ c_1?(y,z).\ \texttt{free}\, c_1.\ \texttt{out}!y.\ (\text{Bck} \parallel d!z)$$

5.3 Proofs of Buffer Efficiency

We can formally express that eBuff is more efficient than Buff in terms of the reduction semantics outlined in Section 2 through the following statements:

$$\Gamma_{\text{ext}} \ \models\ \ M \triangleright \text{eBuff} \ \lesssim_{\text{beh}}\ M \triangleright \text{Buff} \tag{90}$$
$$\Gamma_{\text{ext}} \ \models\ \ M \triangleright \text{Buff} \ \not\lesssim_{\text{beh}}\ M \triangleright \text{eBuff} \tag{91}$$

In order to show that the second statement (91) holds, we need to prove that *there is no amortisation credit n* for which $\Gamma_{\text{ext}} \ \models\ M \triangleright \text{Buff} \ \lesssim^{n}_{\text{beh}}\ M \triangleright \text{eBuff}$. By choosing the set of inductively defined contexts R_n where[11]:

$$R_0 \triangleq \texttt{nil} \qquad\qquad R_{n+1} \triangleq \texttt{in}!v.\texttt{out}?x.R_n$$

we can argue by analysing the reduction graph of the respective systems that, for any $n \geq 0$:

$$\Gamma_{\text{ext}} \ \models\ M \triangleright (\text{Buff} \parallel R_{n+1}) \ \not\lesssim^{n}_{\text{beh}}\ M \triangleright (\text{eBuff} \parallel R_{n+1})$$

since it violates the Cost Improving property of Definition 11.

Another way how to prove (91) is by exploiting completeness of our bisimulation proof technique *wrt.* our behavioural preorder, Theorem 3, and work at the level of the transition system of Section 3 showing that, for all $n \geq 0$, the following holds:

$$\Gamma_{\text{ext}} \ \models\ M \triangleright \text{Buff} \ \not\mathrel{\mathbb{F}}^{n}_{\text{bis}}\ M \triangleright \text{eBuff} \tag{92}$$

We prove the above statement as Theorem 4 of Section 5.4.

Property (90), *prima facie*, seems even harder to prove than (91), because we are required to show that Barb Preservation and Cost Improving hold under every possible valid context interacting with

[11] Note that $\Gamma_{\text{ext}} \vdash R_n$ for any n.

the two buffer implementations. Once again, we use the transition system of Section 3 and show instead that:

$$\Gamma_{\text{ext}} \;\models\; M \triangleright \text{eBuff} \sqsubseteq^{0}_{\approx\text{bis}} M \triangleright \text{Buff} \tag{93}$$

The required result then follows from Theorem 2. The proof for this statement is presented in Section 5.5.

In order to make the presentation of these proofs more manageable, we define the following macro definitions for sub-processes making up the derivatives of $\Gamma_{\text{ext}} \triangleleft M \triangleright \text{Buff}$ and $\Gamma_{\text{ext}} \triangleleft M \triangleright \text{eBuff}$.

$$\text{Frn'} \stackrel{\text{def}}{=} b?x.\texttt{in}?y.\texttt{alloc}\,z.(\text{Frn} \parallel b!z \parallel x!(y,z)) \qquad \text{Bck'} \stackrel{\text{def}}{=} d?x.x?(y,z).\texttt{out}!y.(\text{Bck} \parallel d!z)$$

$$\text{Frn''}(x) \stackrel{\text{def}}{=} \texttt{in}?y.\texttt{alloc}\,z.(\text{Frn} \parallel b!z \parallel x!(y,z)) \qquad \text{Bck''}(x) \stackrel{\text{def}}{=} x?(y,z).\texttt{out}!y.(\text{Bck} \parallel d!z)$$

$$\text{Frn'''}(x,y) \stackrel{\text{def}}{=} \texttt{alloc}\,z.(\text{Frn} \parallel b!z \parallel x!(y,z)) \qquad \text{Bck'''}(y,z) \stackrel{\text{def}}{=} \texttt{out}!y.(\text{Bck} \parallel d!z)$$

$$\text{eBk'} \stackrel{\text{def}}{=} d?x.x?(y,z).\texttt{free}\,x.\texttt{out}!y.(\text{eBk} \parallel d!z) \qquad \text{eBk''}(x) \stackrel{\text{def}}{=} x?(y,z).\texttt{free}\,x.\texttt{out}!y.(\text{eBk} \parallel d!z)$$

$$\text{eBk'''}(x,y,z) \stackrel{\text{def}}{=} \texttt{free}\,x.\texttt{out}!y.(\text{eBk} \parallel d!z) \qquad \text{eBk''''}(y,z) \stackrel{\text{def}}{=} \texttt{out}!y.(\text{eBk} \parallel d!z)$$

We can thus express the definitions for Buff and eBuff as:

$$\text{Buff} \stackrel{\text{def}}{=} \text{Frn''}(c_1) \parallel \text{Bck''}(c_1) \qquad \text{eBuff} \stackrel{\text{def}}{=} \text{Frn''}(c_1) \parallel \text{eBk''}(c_1) \tag{94}$$

5.4 Proving Strict Inefficiency

In order to prove (94), we do not need to explore the entire state space for $\Gamma_{\text{ext}} \triangleleft M \triangleright \text{Buff}$ and $\Gamma_{\text{ext}} \triangleleft M \triangleright \text{eBuff}$. Instead, it suffices to limit external interactions with the observer to traces of the form $(\stackrel{\texttt{in}?v \cdot \texttt{out}!v}{\Longrightarrow})^*$, which simulate interactions with the observing processes R_n discussed in Section 5.3. It is instructive to visualise the transition graphs for both $\Gamma_{\text{ext}} \triangleleft M \triangleright \text{Buff}$ and $\Gamma_{\text{ext}} \triangleleft M \triangleright \text{eBuff}$ for a single iteration $\stackrel{\texttt{in}?v \cdot \texttt{out}!v}{\Longrightarrow}$ as depicted in Fig. 6: due to lack of space, the nodes in these graphs abstract away from the environment Γ_{ext} and appropriate resource environments $M, N, \ldots$ containing internal channels $c_1, c_2, \ldots$ as required.[12] For instance the first node of the first graph in Fig. 6, $\text{Frn''}(c_1) \parallel \text{Bck''}(c_1)$, stands for $\Gamma_{\text{ext}} \triangleleft M \triangleright (\text{Frn''}(c_1) \parallel \text{Bck''}(c_1))$, where $c_1 \in M$, whereas the third node in the same graph, $\text{Frn} \parallel b!c_2 \parallel c_1!(v,c_2) \parallel \text{Bck''}(c_1)$, stands for $\Gamma_{\text{ext}} \triangleleft N \triangleright (\text{Frn} \parallel b!c_2 \parallel c_1!(v,c_2) \parallel \text{Bck''}(c_1))$, where $c_1, c_2 \in N$.

Theorem 4, which proves (92), relies on two lemmas. The main one is Lemma 24, which establishes that a number of derivatives from the configurations $\Gamma_{\text{ext}} \triangleleft M \triangleright \text{Buff}$ and $\Gamma_{\text{ext}} \triangleleft M \triangleright \text{eBuff}$ cannot be related for *any* amortisation credit. This Lemma, in turn, relies on Lemma 23, which establishes that, for a particular amortisation credit n, if some pair of derivatives of the configurations $\Gamma_{\text{ext}} \triangleleft M \triangleright \text{Buff}$ and $\Gamma_{\text{ext}} \triangleleft M \triangleright \text{eBuff}$ *resp.* cannot be related, then other pairs of derivatives cannot be related

[12] The transition graph also abstracts away from environment moves.

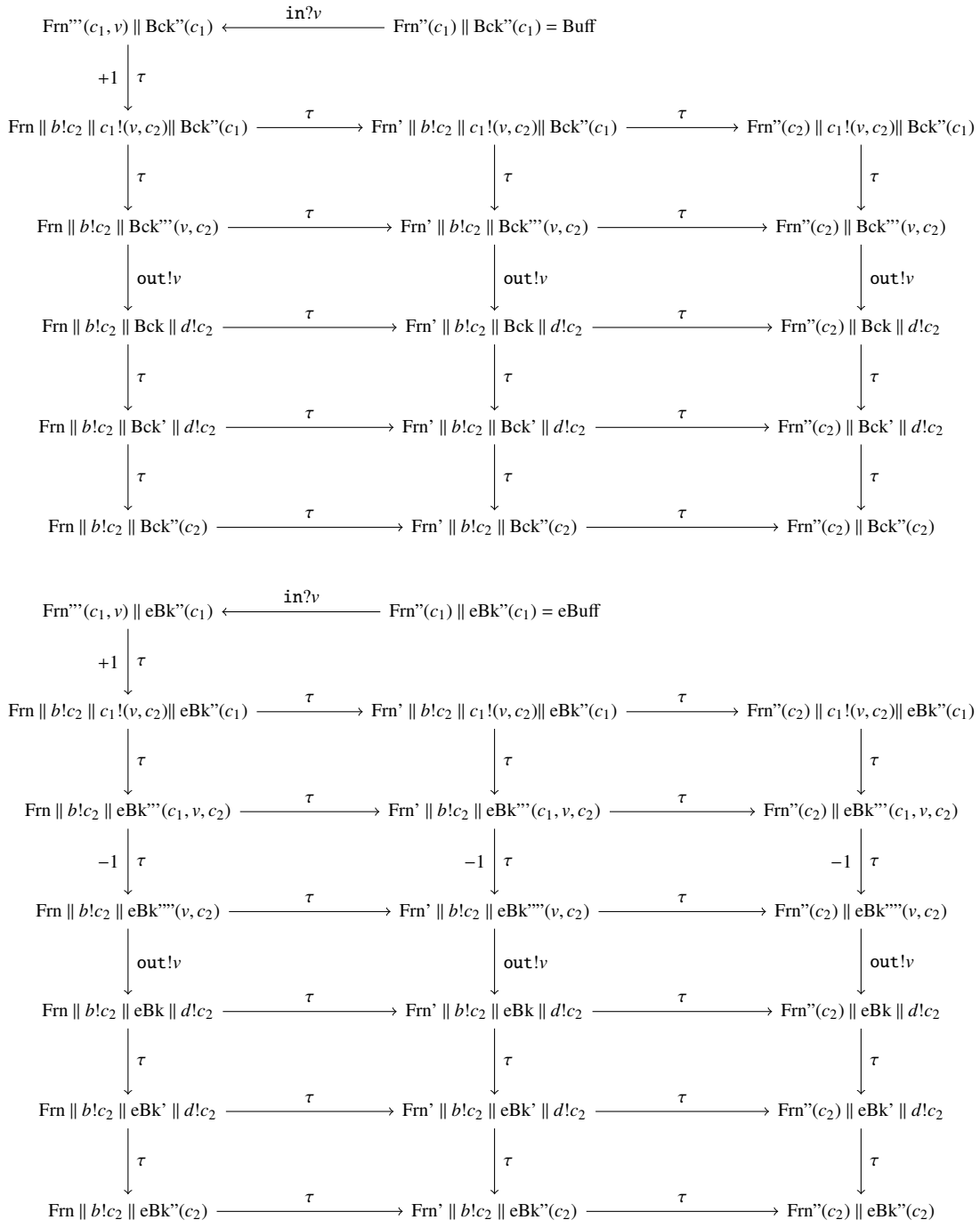

Figure 6: Transition graphs for $\Gamma_{\text{ext}} \triangleleft M \triangleright$ Buff and $\Gamma_{\text{ext}} \triangleleft M \triangleright$ eBuff restricted to $\xRightarrow{\texttt{in?}v \cdot \texttt{out!}v}$

either. Lemma 23 is used again by Theorem 4 to derive that, from the unrelated pairs identified by Lemma 24, the required pair of configurations $\Gamma_{\text{ext}} \triangleleft M \triangleright \text{Buff}$ and $\Gamma_{\text{ext}} \triangleleft M \triangleright \text{eBuff}$ cannot be related for any amortisation credit. Upon first reading, the reader who is only interested in the eventual result may safely skip to the statement of Theorem 4 and treat Lemma 24 and Lemma 23 as black-boxes.

In order to be able to state Lemma 23 and Lemma 24 more succinctly, we find it convenient to delineate groups of processes relating to derivatives of Buff and eBuff. For instance, we can partition the processes depicted in the second transition graph of Fig. 6 (derivatives of eBuff) into three sets:

$$\text{Prc}_{\text{A}} \stackrel{\text{def}}{=} \left\{ \begin{array}{l} (\text{Frn} \| b!c_2 \| c_1!(v,c_2) \| \text{eBk"}(c_1)), \\ (\text{Frn'} \| b!c_2 \| c_1!(v,c_2) \| \text{eBk"}(c_1)), \\ (\text{Frn"}(c_2) \| c_1!(v,c_2) \| \text{eBk"}(c_1)),\ (\text{Frn} \| b!c_2 \| \text{eBk"'}(c_1,v,c_2)), \\ (\text{Frn'} \parallel b!c_2 \parallel \text{eBk"'}(c_1,v,c_2)),\ (\text{Frn"}(c_2) \parallel \text{eBk"'}(c_1,v,c_2)) \end{array} \middle| \begin{array}{l} c_1 \neq c_2 \in \\ \textsc{Chan} \setminus \{\texttt{in}, \texttt{out}, b, d\} \end{array} \right\}$$

$$\text{Prc}_{\text{B}} \stackrel{\text{def}}{=} \left\{ \begin{array}{l} (\text{Frn} \| b!c_2 \| \text{eBk""}(v,c_2)),\ (\text{Frn'} \| b!c_2 \| \text{eBk""}(v,c_2)), \\ (\text{Frn"}(c_2) \| \text{eBk""}(v,c_2)) \end{array} \middle| \ c_2 \in \textsc{Chan} \setminus \{\texttt{in}, \texttt{out}, b, d\} \right\}$$

$$\text{Prc}_{\text{C}} \stackrel{\text{def}}{=} \left\{ \begin{array}{l} (\text{Frn} \parallel b!c_2 \parallel \text{eBk} \parallel d!c_2),\ (\text{Frn'} \parallel b!c_2 \parallel \text{eBk} \parallel d!c_2), \\ (\text{Frn"}(c_2) \parallel \text{eBk} \parallel d!c_2),\ (\text{Frn} \parallel b!c_2 \parallel \text{eBk'} \parallel d!c_2), \\ (\text{Frn'} \parallel b!c_2 \parallel \text{eBk'} \parallel d!c_2),\ (\text{Frn"}(c_2) \parallel \text{eBk'} \parallel d!c_2), \\ (\text{Frn} \parallel b!c_2 \parallel \text{eBk"}(c_2)),\ (\text{Frn'} \parallel b!c_2 \parallel \text{eBk"}(c_2)), \\ (\text{Frn"}(c_2) \parallel \text{eBk"}(c_2)) \end{array} \middle| \ c_2 \in \textsc{Chan} \setminus \{\texttt{in}, \texttt{out}, b, d\} \right\}$$

With respect to the second transition graph of Fig. 6, Prc_{A} groups the processes *after the allocation* of an (arbitrary) internal channel c_2 but not before any deallocation, *i.e.*, the second and third rows of the graph. The set Prc_{B} groups the processes *after the deallocation* of the (arbitrary) internal channel c_1, *i.e.*, the fourth row of the graph. Finally, the set Prc_{C} groups processes *after the output action* $\texttt{out}!v$ is performed (before an input action is performed), *i.e.*, the last three rows of the graph.

Lemma 23 (Related Negative Results)**.**

1. *For any amortisation credit n and appropriate M, N, whenever:*

   - $\Gamma_{ext} \models M \triangleright \text{Frn"'}(c_1,v) \| \text{Bck"}(c_1) \ \not\approx^n_{bis} \ N \triangleright \text{Frn"'}(c'_1,v) \| \text{eBk"}(c'_1)$
   - *For any* $Q \in Prc_{\text{A}}$ *we have* $\Gamma_{ext} \models M \triangleright \text{Frn"'}(c_1,v) \| \text{Bck"}(c_1) \ \not\approx^{n+1}_{bis} \ N \triangleright Q$
   - *For any* $Q \in Prc_{\text{B}}$ *we have* $\Gamma_{ext} \models M \triangleright \text{Frn"'}(c_1,v) \| \text{Bck"}(c_1) \ \not\approx^n_{bis} \ N \triangleright Q$

   then, for any $P \in Prc_{\text{C}}$*, we have* $\Gamma_{ext} \models M \triangleright \text{Frn"}(c_1) \| \text{Bck"}(c_1) \ \not\approx^n_{bis} \ N \triangleright P$.

2. *For any amortisation credit n and appropriate M, N, and for any* $P, Q \in Prc_{\text{C}}$*:*

   (a) $\Gamma_{ext} \models M \triangleright \text{Frn"}(c_1) \| \text{Bck"}(c_1) \ \not\approx^n_{bis} \ N \triangleright Q$ *implies*
   $\Gamma_{ext} \models M \triangleright \text{Frn'} \| b!c_1 \| \text{Bck"}(c_1) \ \not\approx^n_{bis} \ N \triangleright P$

   (b) $\Gamma_{ext} \models M \triangleright \text{Frn'} \| b!c_1 \| \text{Bck"}(c_1) \ \not\approx^n_{bis} \ N \triangleright Q$ *implies*
   $\Gamma_{ext} \models M \triangleright \text{Frn} \| b!c_1 \| \text{Bck"}(c_1) \ \not\approx^n_{bis} \ N \triangleright P$

(c) $\Gamma_{ext} \models M \triangleright \mathrm{Frn} \| b!c_1 \| \mathrm{Bck}''(c_1) \not\approx^n_{bis} N \triangleright Q$ *implies*
$\Gamma_{ext} \models M \triangleright \mathrm{Frn} \| b!c_1 \| \mathrm{Bck}' \| d!c_1 \not\approx^n_{bis} N \triangleright P$

(d) $\Gamma_{ext} \models M \triangleright \mathrm{Frn} \| b!c_1 \| \mathrm{Bck}' \| d!c_1 \not\approx^n_{bis} N \triangleright Q$ *implies*
$\Gamma_{ext} \models M \triangleright \mathrm{Frn} \| b!c_1 \| \mathrm{Bck} \| d!c_1 \not\approx^n_{bis} N \triangleright P$

3. *For any amortisation credit n and appropriate M, N, and for any* $P, R \in Prc_{\mathrm{B}}$, $Q \in Prc_{\mathrm{C}}$*:*

(a) $\Gamma_{ext} \models M \triangleright \mathrm{Frn} \| b!c_2 \| \mathrm{Bck} \| d!c_2 \not\approx^n_{bis} N \triangleright Q$ *implies*
$\Gamma_{ext} \models M \triangleright \mathrm{Frn} \| b!c_2 \| \mathrm{Bck}'''(v, c_2) \not\approx^n_{bis} N \triangleright P$

(b) $\Gamma_{ext} \models M \triangleright \mathrm{Frn} \| b!c_2 \| \mathrm{Bck}'''(v, c_2) \not\approx^n_{bis} N \triangleright R$ *implies*
$\Gamma_{ext} \models M \triangleright \mathrm{Frn} \| b!c_2 \| c_1!(v, c_2) \| \mathrm{Bck}''(c_1) \not\approx^n_{bis} N \triangleright P$

4. *For any amortisation credit n and appropriate M, N, and for any* $P \in Prc_{\mathrm{A}}$, $Q \in Prc_{\mathrm{C}}$*:*

(a) $\Gamma_{ext} \models M \triangleright \mathrm{Frn} \| b!c_1 \| \mathrm{Bck} \| d!c_1 \not\approx^n_{bis} N \triangleright Q$ *implies*
$\Gamma_{ext} \models M \triangleright \mathrm{Frn} \| b!c_1 \| \mathrm{Bck}'''(v, c_1) \not\approx^{n+1}_{bis} N \triangleright P$

(b) $\Gamma_{ext} \models M \triangleright \mathrm{Frn} \| b!c_1 \| \mathrm{Bck} \| d!c_1 \not\approx^n_{bis} N \triangleright Q$ *implies*
$\Gamma_{ext} \models M \triangleright \mathrm{Frn} \| b!c_1 \| \mathrm{Bck}'''(v, c_1) \not\approx^n_{bis} N \triangleright \mathrm{Frn}'''(c_1', v) \parallel \mathrm{eBk}''(c_1')$

(c) $\Gamma_{ext} \models M \triangleright \mathrm{Frn} \| b!c_1 \| \mathrm{Bck} \| d!c_1 \not\approx^n_{bis} N \triangleright Q$ *implies*
$\Gamma_{ext} \models M \triangleright \mathrm{Frn} \| b!c_2 \| c_1!(v, c_2) \| \mathrm{Bck}''(c_1) \not\approx^n_{bis} N \triangleright \mathrm{Frn}'''(c_1', v) \parallel \mathrm{eBk}''(c_1')$

Proof. Each case is proved by contradiction:

1. Assume the premises together with the inverse of the conclusion, *i.e.*,

$$\Gamma_{\mathrm{ext}} \models M \triangleright \mathrm{Frn}''(c_1) \| \mathrm{Bck}''(c_1) \not\precsim^n_{\mathrm{bis}} N \triangleright P.$$

Consider the transition from the left-hand configuration:

$$\Gamma_{\mathrm{ext}} \triangleleft M \triangleright \mathrm{Frn}''(c_1) \| \mathrm{Bck}''(c_1) \xrightarrow{\mathtt{in}?v}_0 \Gamma_{\mathrm{ext}} \triangleleft M \triangleright \mathrm{Frn}'''(c_1, v) \| \mathrm{Bck}''(c_1).$$

For any $P \in \mathrm{Prc}_{\mathrm{C}}$, this can only be matched by the right-hand configuration, $\Gamma_{\mathrm{ext}} \triangleleft N \triangleright P$, through either of the following cases:

(a) $\Gamma_{\mathrm{ext}} \triangleleft N \triangleright P \stackrel{\mathtt{in}?v}{\Longrightarrow}_0 \Gamma_{\mathrm{ext}} \triangleleft N \triangleright \mathrm{Frn}'''(c_1', v) \| \mathrm{eBk}''(c_1')$, *i.e.*, a weak input action without trailing τ-moves after the external action $\mathtt{in}?v$ — see first row of the second graph in Fig. 6. But we know $\Gamma_{\mathrm{ext}} \models M \triangleright \mathrm{Frn}'''(c_1, v) \| \mathrm{Bck}''(c_1) \not\approx^n_{\mathrm{bis}} N \triangleright \mathrm{Frn}'''(c_1', v) \| \mathrm{eBk}''(c_1')$ from the first premise.

(b) $\Gamma_{\mathrm{ext}} \triangleleft N \triangleright P \stackrel{\mathtt{in}?v}{\Longrightarrow}_{+1} \Gamma_{\mathrm{ext}} \triangleleft N \triangleright Q$ for some $Q \in \mathrm{Prc}_{\mathrm{A}}$. However, from the second premise we know that $\Gamma_{\mathrm{ext}} \models M \triangleright \mathrm{Frn}'''(c_1, v) \| \mathrm{Bck}''(c_1) \not\approx^{n+1}_{\mathrm{bis}} N \triangleright Q$

(c) $\Gamma_{\mathrm{ext}} \triangleleft N \triangleright P \stackrel{\mathtt{in}?v}{\Longrightarrow}_0 \Gamma_{\mathrm{ext}} \triangleleft N \triangleright Q$ for some $Q \in \mathrm{Prc}_{\mathrm{B}}$. Again, from the third premise we know that $\Gamma_{\mathrm{ext}} \models M \triangleright \mathrm{Frn}'''(c_1, v) \| \mathrm{Bck}''(c_1) \not\approx^n_{\mathrm{bis}} N \triangleright Q$

Since $\Gamma_{\text{ext}} \triangleleft N \triangleright P$ cannot perform a matching move, we obtain a contradiction.

2. We here prove case (a). The other cases are analogous.

   Assume $\Gamma_{\text{ext}} \models M \triangleright \text{Frn'} \,\|\, b!c_1 \,\|\, \text{Bck"}(c_1) \sqsubseteq^n_{\text{bis}} N \triangleright P$ and consider the action

   $$\Gamma_{\text{ext}} \triangleleft M \triangleright \text{Frn'} \,\|\, b!c_1 \,\|\, \text{Bck"}(c_1) \xrightarrow{\tau}_0 \Gamma_{\text{ext}} \triangleleft M \triangleright \text{Frn"}(c_1) \,\|\, \text{Bck"}(c_1).$$

   For our assumption to hold, $\Gamma_{\text{ext}} \triangleleft N \triangleright P$ would need to match this move by a (weak) silent action leading to a configuration that can match $\Gamma_{\text{ext}} \triangleleft M \triangleright \text{Frn"}(c_1) \parallel \text{Bck"}(c_1)$. The only matching move can be

   $$\Gamma_{\text{ext}} \triangleleft N \triangleright P \Longrightarrow_0 \Gamma_{\text{ext}} \triangleleft N \triangleright Q \qquad \text{for some } Q \in \text{Prc}_{\text{C}}.$$

   However, from our premise we know $\Gamma_{\text{ext}} \models M \triangleright \text{Frn"}(c_1) \parallel \text{Bck"}(c_1) \not\sqsubseteq^n_{\text{bis}} N \triangleright Q'$ for any amortisation credit n and $Q' \in \text{Prc}_{\text{C}}$ and therefore conclude that the move cannot be matched, thereby obtaining a contradiction.

3. We here prove case (a). Case (b) is analogous.

   Assume $\Gamma_{\text{ext}} \models M \triangleright \text{Frn} \,\|\, b!c_2 \,\|\, \text{Bck"'}(v, c_2) \sqsubseteq^n_{\text{bis}} N \triangleright P$ and consider the action

   $$\Gamma_{\text{ext}} \triangleleft M \triangleright \text{Frn} \,\|\, b!c_2 \,\|\, \text{Bck"'}(v, c_2) \xrightarrow{\texttt{out}!v}_0 \Gamma_{\text{ext}} \triangleleft M \triangleright \text{Frn} \,\|\, b!c_2 \,\|\, \text{Bck} \,\|\, d!c_2$$

   This action can only be matched by a transition of the form

   $$\Gamma_{\text{ext}} \triangleleft N \triangleright P \xRightarrow{\texttt{out}!v}_0 \Gamma_{\text{ext}} \triangleleft N \triangleright Q \qquad \text{for some } Q \in \text{Prc}_{\text{C}}.$$

   However, from our premise we know $\Gamma_{\text{ext}} \models M \triangleright \text{Frn} \,\|\, b!c_2 \,\|\, \text{Bck} \,\|\, d!c_2 \not\sqsubseteq^n_{\text{bis}} N \triangleright Q$ for any amortisation credit n and $Q \in \text{Prc}_{\text{C}}$. Thus we conclude that the move cannot be matched, thereby obtaining a contradiction.

4. Cases (a) and (b) are analogous to 3(a) and 3(b). We here outline the proof for case (c).

   First we note that from the premise $\Gamma_{\text{ext}} \models M \triangleright \text{Frn} \,\|\, b!c_1 \,\|\, \text{Bck} \,\|\, d!c_1 \not\sqsubseteq^n_{\text{bis}} N \triangleright Q$ (for any $Q \in \text{Prc}_{\text{C}}$) and Lemma 23.3$(a)$, Lemma 23.4$(a)$ and Lemma 23.4(b) *resp.* we obtain:

   $$\Gamma_{\text{ext}} \models M \triangleright \text{Frn} \,\|\, b!c_2 \,\|\, \text{Bck"'}(v, c_2) \not\sqsubseteq^n_{\text{bis}} N \triangleright P \qquad \text{for any } P \in \text{Prc}_{\text{B}} \tag{95}$$
   $$\Gamma_{\text{ext}} \models M \triangleright \text{Frn} \,\|\, b!c_1 \,\|\, \text{Bck"'}(v, c_1) \not\sqsubseteq^{n+1}_{\text{bis}} N \triangleright P \qquad \text{for any } P \in \text{Prc}_{\text{A}} \tag{96}$$
   $$\Gamma_{\text{ext}} \models M \triangleright \text{Frn} \,\|\, b!c_1 \,\|\, \text{Bck"'}(v, c_1) \not\sqsubseteq^n_{\text{bis}} N \triangleright \text{Frn"'}(c_1', v) \parallel \text{eBk"}(c_1') \tag{97}$$

   We assume $\Gamma_{\text{ext}} \models M \triangleright \text{Frn} \,\|\, b!c_2 \,\|\, c_1!(v, c_2) \,\|\, \text{Bck"}(c_1) \sqsubseteq^n_{\text{bis}} N \triangleright \text{Frn"'}(c_1', v) \parallel \text{eBk"}(c_1')$ and then showing that this leads to a contradiction. Consider the move

   $$\Gamma_{\text{ext}} \triangleleft M \triangleright \text{Frn} \,\|\, b!c_2 \,\|\, c_1!(v, c_2) \,\|\, \text{Bck"}(c_1) \xrightarrow{\tau}_0 \Gamma_{\text{ext}} \triangleleft M \triangleright \text{Frn} \,\|\, b!c_2 \,\|\, \text{Bck"'}(v, c_2)$$

   This can be matched by $\Gamma_{\text{ext}} \triangleleft N \triangleright \text{Frn"'}(c_1', v) \parallel \text{eBk"}(c_1')$ using either of the following moves:

- $\Gamma_{\text{ext}} \triangleleft N \triangleright \text{Frn}'''(c_1', v) \parallel \text{eBk}''(c_1') \implies_0 \Gamma_{\text{ext}} \triangleleft N \triangleright \text{Frn}'''(c_1', v) \parallel \text{eBk}''(c_1')$. But (97) prohibits this from being the matching move.
- $\Gamma_{\text{ext}} \triangleleft N \triangleright \text{Frn}'''(c_1', v) \parallel \text{eBk}''(c_1') \implies_{+1} \Gamma_{\text{ext}} \triangleleft N \triangleright Q$ for some $Q \in \text{Prc}_\text{A}$. But (96) prohibits this from being the matching move.
- $\Gamma_{\text{ext}} \triangleleft N \triangleright \text{Frn}'''(c_1', v) \parallel \text{eBk}''(c_1') \implies_0 \Gamma_{\text{ext}} \triangleleft N \triangleright Q$ for some $Q \in \text{Prc}_\text{B}$. But (95) prohibits this from being the matching move.

This contradicts our earlier assumption.

□

Lemma 24. *For all $n \in \text{Nat}$ and appropriate M, N:*

1. *For any $Q \in Prc_\text{A}$ we have* $\Gamma_{ext} \models M \triangleright \text{Frn} \parallel b!c_2 \parallel \text{Bck}'''(v, c_2) \;\; \not\sqsubseteq^n_{bis} N \triangleright Q$
2. *For any $Q \in Prc_\text{A}$ we have* $\Gamma_{ext} \models M \triangleright \text{Frn} \parallel b!c_2 \parallel c_1!(v, c_2) \parallel \text{Bck}''(c_1) \;\; \not\sqsubseteq^n_{bis} N \triangleright Q$
3. *For any $Q \in Prc_\text{A}$ we have* $\Gamma_{ext} \models M \triangleright \text{Frn}'''(c_1, v) \parallel \text{Bck}''(c_1) \;\; \not\sqsubseteq^{n+1}_{bis} N \triangleright Q$
4. *For any $Q \in Prc_\text{B}$ we have* $\Gamma_{ext} \models M \triangleright \text{Frn}'''(c_1, v) \parallel \text{Bck}''(c_1) \;\; \not\sqsubseteq^n_{bis} N \triangleright Q$
5. $\Gamma_{ext} \models M \triangleright \text{Frn}'''(c_1, v) \parallel \text{Bck}''(c_1) \;\; \not\sqsubseteq^n_{bis} N \triangleright \text{Frn}'''(c_1', v) \parallel \text{eBk}''(c_1')$

Proof. By induction on n

$n = 0$ We prove each clause by contradiction:

1. Assume $\Gamma_{\text{ext}} \models M \triangleright \text{Frn} \parallel b!c_2 \parallel \text{Bck}'''(v, c_2) \sqsubseteq^0_{\sim\text{bis}} N \triangleright Q$ for some $Q \in \text{Prc}_\text{A}$ and consider the transition

$$\Gamma_{\text{ext}} \triangleleft M \triangleright \text{Frn} \parallel b!c_2 \parallel \text{Bck}'''(v, c_2) \xrightarrow{\texttt{out}!v}_0 \Gamma_{\text{ext}} \triangleleft M \triangleright \text{Frn} \parallel b!c_2 \parallel \text{Bck} \parallel d!c_2$$

For any $Q \in \text{Prc}_\text{A}$, this cannot be matched by any move from $\Gamma_{\text{ext}} \triangleleft N \triangleright Q$ since output actions must be preceded by a channel deallocation, which incurs a *negative* cost — see second and third rows of the second graph in Fig. 6. Stated otherwise, every matching move can only be of the form

$$\Gamma_{\text{ext}} \triangleleft N \triangleright Q \xRightarrow{\texttt{out}!v}_{-1} \Gamma_{\text{ext}} \triangleleft N' \triangleright Q'$$

where $N = (N', c_1')$ for some c_1' and $Q' \in \text{Prc}_\text{C}$. However, since the amortisation credit cannot be negative, we can never have $\Gamma_{\text{ext}} \models M \triangleright \text{Frn} \parallel b!c_2 \parallel \text{Bck} \parallel d!c_2 \sqsubseteq^{-1}_{\sim\text{bis}} N' \triangleright Q'$. We therefore obtain a contradiction.

2. Assume $\Gamma_{\text{ext}} \models M \triangleright \text{Frn} \parallel b!c_2 \parallel c_1!(v, c_2) \parallel \text{Bck"}(c_1) \sqsubseteq^0_{\text{bis}} N \triangleright Q$ for some $Q \in \text{Prc}_\text{A}$ and consider the transition

$$\Gamma_{\text{ext}} \triangleleft M \triangleright \text{Frn} \parallel b!c_2 \parallel c_1!(v, c_2) \parallel \text{Bck"}(c_1) \xrightarrow{\tau}_0 \Gamma_{\text{ext}} \triangleleft M \triangleright \text{Frn} \parallel b!c_2 \parallel \text{Bck"'}(v, c_2)$$

Since the amortisation credit can never be negative, the matching move can only be of the form

$$\Gamma_{\text{ext}} \triangleleft N \triangleright Q \Longrightarrow_0 \Gamma_{\text{ext}} \triangleleft N \triangleright Q'$$

for some $Q' \in \text{Prc}_\text{A}$. But then we get a contradiction since, from the previous clause, we know that $\Gamma_{\text{ext}} \models M \triangleright \text{Frn} \parallel b!c_2 \parallel \text{Bck"'}(v, c_2) \not\sqsubseteq^0_{\text{bis}} N \triangleright Q'$.

3. Assume $\Gamma_{\text{ext}} \models M \triangleright \text{Frn"'}(c_1, v) \parallel \text{Bck"}(c_1) \sqsubseteq^{n+1}_{\text{bis}} N \triangleright Q$ for some $Q \in \text{Prc}_\text{A}$ and consider the transition

$$\Gamma_{\text{ext}} \triangleleft M \triangleright \text{Frn"'}(c_1, v) \parallel \text{Bck"}(c_1) \xrightarrow{\tau}_{+1} \Gamma_{\text{ext}} \triangleleft M, c_2 \triangleright \text{Frn} \parallel b!c_2 \parallel c_1!(v, c_2) \parallel \text{Bck"}(c_1)$$

for some newly allocated channel c_2. As in the previous case, since the amortisation credit can never be negative, the matching move can only be of the form

$$\Gamma_{\text{ext}} \triangleleft N \triangleright Q \Longrightarrow_0 \Gamma_{\text{ext}} \triangleleft N \triangleright Q'$$

for some $Q' \in \text{Prc}_\text{A}$. But then we get a contradiction since, from the previous clause, we know that $\Gamma_{\text{ext}} \models (M, c_2) \triangleright \text{Frn} \parallel b!c_2 \parallel c_1!(v, c_2) \parallel \text{Bck"}(c_1) \not\sqsubseteq^n_{\text{bis}} N \triangleright Q'$.

4. Analogous to the previous case.

5. Assume $\Gamma_{\text{ext}} \models M \triangleright \text{Frn"'}(c_1, v) \parallel \text{Bck"}(c_1) \sqsubseteq^0_{\text{bis}} N \triangleright \text{Frn"'}(c'_1, v) \parallel \text{eBk"}(c'_1)$ and consider the transition

$$\Gamma_{\text{ext}} \triangleleft M \triangleright \text{Frn"'}(c_1, v) \parallel \text{Bck"}(c_1) \xrightarrow{\tau}_{+1} \Gamma_{\text{ext}} \triangleleft M, c_2 \triangleright \text{Frn} \parallel b!c_2 \parallel c_1!(v, c_2) \parallel \text{Bck"}(c_1)$$

Since the transition incurred a cost of $+1$ and the current amortisation credit is 0, the matching weak transition must also incur a cost of $+1$ and thus $\Gamma_{\text{ext}} \triangleleft N \triangleright \text{Frn"'}(c'_1, v) \parallel \text{eBk"}(c'_1)$ can only match this by the move

$$\Gamma_{\text{ext}} \triangleleft N \triangleright \text{Frn"'}(c'_1, v) \parallel \text{eBk"}(c'_1) \overset{\tau}{\Longrightarrow}_{+1} \Gamma_{\text{ext}} \triangleleft N, c'_2 \triangleright Q$$

for some $Q \in \text{Prc}_\text{A}$. But then we still get a contradiction since, from clause (2), we know $\Gamma_{\text{ext}} \models M \triangleright \text{Frn} \parallel b!c_2 \parallel c_1!(v, c_2) \parallel \text{Bck"}(c_1) \not\sqsubseteq^0_{\text{bis}} N \triangleright Q$.

$n = k + 1$ We prove each clause by contradiction. However before we tackle each individual clause, we note that from clauses (3), (4) and (5) of the I.H. we know

For any $Q \in \text{Prc}_\text{A}$ we have $\Gamma_{\text{ext}} \models M \triangleright \text{Frn"'}(c_1, v) \parallel \text{Bck"}(c_1) \not\sqsubseteq^{k+1}_{\text{bis}} N \triangleright Q$
For any $Q \in \text{Prc}_\text{B}$ we have $\Gamma_{\text{ext}} \models M \triangleright \text{Frn"'}(c_1, v) \parallel \text{Bck"}(c_1) \not\sqsubseteq^{k}_{\text{bis}} N \triangleright Q$
$\Gamma_{\text{ext}} \models M \triangleright \text{Frn"'}(c_1, v) \parallel \text{Bck"}(c_1) \not\sqsubseteq^{k}_{\text{bis}} N \triangleright \text{Frn"'}(c_1, v) \parallel \text{eBk"}(c_1)$

By Lemma 23.1 we obtain, for any $Q' \in \mathrm{Prc_C}$ and appropriate N':

$$\Gamma_{\mathrm{ext}} \models M \triangleright \mathrm{Frn}"(c_1) \| \mathrm{Bck}"(c_1) \;\; \not\sqsubseteq^{k}_{\mathrm{bis}} N' \triangleright Q'$$

and by Lemma 23.2(a), Lemma 23.2(b), Lemma 23.2(c) and Lemma 23.2(d) we obtain, for any $Q' \in \mathrm{Prc_C}$ and appropriate N':

$$\Gamma_{\mathrm{ext}} \models M \triangleright \mathrm{Frn} \parallel b!c_2 \parallel \mathrm{Bck} \parallel d!c_2 \;\; \not\sqsubseteq^{k}_{\mathrm{bis}} N \triangleright Q' \tag{98}$$

Also, by (98), Lemma 23.3(a) and Lemma 23.3(b) we obtain, for any $Q'' \in \mathrm{Prc_B}$:

$$\Gamma_{\mathrm{ext}} \models M \triangleright \mathrm{Frn} \| b!c_2 \| \mathrm{Bck}"'(v, c_2) \;\; \not\sqsubseteq^{k}_{\mathrm{bis}} N \triangleright Q'' \tag{99}$$

$$\Gamma_{\mathrm{ext}} \models M \triangleright \mathrm{Frn} \| b!c_2 \| c_1!(v, c_2) \| \mathrm{Bck}"(c_1) \;\; \not\sqsubseteq^{k}_{\mathrm{bis}} N \triangleright Q'' \tag{100}$$

Moreover, by (98), Lemma 23.4(a), Lemma 23.4(b) and Lemma 23.4(c) we obtain:

$$\Gamma_{\mathrm{ext}} \models M \triangleright \mathrm{Frn} \| b!c_2 \| c_1!(v, c_2) \| \mathrm{Bck}"(c_1) \;\; \not\sqsubseteq^{k}_{\mathrm{bis}} N \triangleright \mathrm{Frn}"'(c_1', v) \parallel \mathrm{eBk}"(c_1') \tag{101}$$

The proofs for each clause are are follows:

1. Assume $\Gamma_{\mathrm{ext}} \models M \triangleright \mathrm{Frn} \parallel b!c_2 \parallel \mathrm{Bck}"'(v, c_2) \sqsubseteq^{k+1}_{\mathrm{bis}} N \triangleright Q$ for some $Q \in \mathrm{Prc_A}$ and consider the transition

   $$\Gamma_{\mathrm{ext}} \triangleleft M \triangleright \mathrm{Frn} \parallel b!c_2 \parallel \mathrm{Bck}"'(v, c_2) \;\xrightarrow{\mathtt{out}!v}_0\; \Gamma_{\mathrm{ext}} \triangleleft M \triangleright \mathrm{Frn} \parallel b!c_2 \parallel \mathrm{Bck} \parallel d!c_2$$

   For any $Q \in \mathrm{Prc_A}$, this can (only) be matched by any move of the form

   $$\Gamma_{\mathrm{ext}} \triangleleft N \triangleright Q \;\Longrightarrow^{\mathtt{out}!v}_{-1}\; \Gamma_{\mathrm{ext}} \triangleleft N' \triangleright Q'$$

   where $N = (N', c_1')$ for some c_1', $Q' \in \mathrm{Prc_C}$, and the external action $\mathtt{out}!v$ is preceded by a τ-move deallocating c_1'. For our initial assumption to hold we need to show that *at least one* of these configurations $\Gamma_{\mathrm{ext}} \triangleleft N' \triangleright Q'$ satisfies the property

   $$\Gamma_{\mathrm{ext}} \models M \triangleright \mathrm{Frn} \parallel b!c_2 \parallel \mathrm{Bck} \parallel d!c_2 \sqsubseteq^{k}_{\mathrm{bis}} N' \triangleright Q'.$$

   But by (98) we know that no such configuration exists, thereby contradicting our initial assumption.

2. Assume $\Gamma_{\mathrm{ext}} \models M \triangleright \mathrm{Frn} \parallel b!c_2 \parallel c_1!(v, c_2) \parallel \mathrm{Bck}"(c_1) \sqsubseteq^{k+1}_{\mathrm{bis}} N \triangleright Q$ for some $Q \in \mathrm{Prc_A}$ and consider the transition

   $$\Gamma_{\mathrm{ext}} \triangleleft M \triangleright \mathrm{Frn} \parallel b!c_2 \parallel c_1!(v, c_2) \parallel \mathrm{Bck}"(c_1) \;\xrightarrow{\tau}_0\; \Gamma_{\mathrm{ext}} \triangleleft M \triangleright \mathrm{Frn} \parallel b!c_2 \parallel \mathrm{Bck}"'(v, c_2)$$

   This transition can be matched by $\Gamma_{\mathrm{ext}} \triangleleft N \triangleright Q$ through either of the following moves:

   (a) $\Gamma_{\mathrm{ext}} \triangleleft N \triangleright Q \;\Longrightarrow_0\; \Gamma_{\mathrm{ext}} \triangleleft N \triangleright Q'$ for some $Q' \in \mathrm{Prc_A}$. However, from the previous clause, *i.e.*, clause (1) when $n = k + 1$, we know that this cannot be the matching move since $\Gamma_{\mathrm{ext}} \models M \triangleright \mathrm{Frn} \parallel b!c_2 \parallel \mathrm{Bck}"'(v, c_2) \;\not\sqsubseteq^{k+1}_{\mathrm{bis}} N \triangleright Q'$.

(b) $\Gamma_{\text{ext}} \triangleleft N \triangleright Q \implies_{-1} \Gamma_{\text{ext}} \triangleleft N' \triangleright Q'$ for some $Q' \in \text{Prc}_\text{B}$ and $N = (N', c_1')$. However, from (99), we know that this cannot be the matching move since $\Gamma_{\text{ext}} \models M \triangleright \text{Frn} \,\|\, b!c_2 \,\|\, \text{Bck}'''(v, c_2) \not\approx^k_{\text{bis}} N' \triangleright Q'$.

Thus, we obtain a contradiction.

3. Assume $\Gamma_{\text{ext}} \models M \triangleright \text{Frn}'''(c_1, v) \parallel \text{Bck}''(c_1) \not\approx^{k+2}_{\text{bis}} N \triangleright Q$, where $Q \in \text{Prc}_\text{A}$, and consider the transition:

$$\Gamma_{\text{ext}} \triangleleft M \triangleright \text{Frn}'''(c_1, v) \parallel \text{Bck}''(c_1) \xrightarrow{\tau}_{+1} \Gamma_{\text{ext}} \triangleleft M, c_2 \triangleright \text{Frn} \parallel b!c_2 \parallel c_1!(v, c_2) \parallel \text{Bck}''(c_1)$$

for some newly allocated channel c_2. This can be matched by $\Gamma_{\text{ext}} \triangleleft N \triangleright Q$ through either of the following moves:

(a) $\Gamma_{\text{ext}} \triangleleft N \triangleright Q \implies_0 \Gamma_{\text{ext}} \triangleleft N \triangleright Q'$ for some $Q' \in \text{Prc}_\text{A}$. However, from the previous clause, *i.e.,* clause (2) when $n = k + 1$, we know that this cannot be the matching move since $\Gamma_{\text{ext}} \models M, c_2 \triangleright \text{Frn} \parallel b!c_2 \parallel c_1!(v, c_2) \parallel \text{Bck}''(c_1) \not\approx^{k+1}_{\text{bis}} N \triangleright Q'$.

(b) $\Gamma_{\text{ext}} \triangleleft N \triangleright Q \implies_{-1} \Gamma_{\text{ext}} \triangleleft N' \triangleright Q'$ for some $Q' \in \text{Prc}_\text{B}$ and $N = (N', c_1')$. However, from (100), we know that this cannot be the matching move since $\Gamma_{\text{ext}} \models M \triangleright \text{Frn} \,\|\, b!c_2 \,\|\, c_1!(v, c_2) \,\|\, \text{Bck}''(c_1) \not\approx^k_{\text{bis}} N' \triangleright Q'$.

Thus, we obtain a contradiction.

4. Analogous to the proof for the previous clause and relies on (100) again.

5. Assume $\Gamma_{\text{ext}} \models M \triangleright \text{Frn}'''(c_1, v) \parallel \text{Bck}''(c_1) \sqsubseteq^{k+1}_{\sim\text{bis}} N \triangleright \text{Frn}'''(c_1', v) \parallel \text{eBk}''(c_1')$ and consider the transition

$$\Gamma_{\text{ext}} \triangleleft M \triangleright \text{Frn}'''(c_1, v) \parallel \text{Bck}''(c_1) \xrightarrow{\tau}_{+1} \Gamma_{\text{ext}} \triangleleft M, c_2 \triangleright \text{Frn} \parallel b!c_2 \parallel c_1!(v, c_2) \parallel \text{Bck}''(c_1)$$

for some newly allocated channel c_2. This can be matched by the right-hand configuration $\Gamma_{\text{ext}} \triangleleft N \triangleright \text{Frn}'''(c_1', v) \parallel \text{eBk}''(c_1')$ through either of the following moves:

(a) $\Gamma_{\text{ext}} \triangleleft N \triangleright \text{Frn}'''(c_1', v) \parallel \text{eBk}''(c_1') \implies_0 \Gamma_{\text{ext}} \triangleleft N \triangleright \text{Frn}'''(c_1', v) \parallel \text{eBk}''(c_1')$, *i.e.,* no transitions. However, from (101), this cannot be the matching move since $\Gamma_{\text{ext}} \models M \triangleright \text{Frn} \,\|\, b!c_2 \,\|\, c_1!(v, c_2) \,\|\, \text{Bck}''(c_1) \not\approx^k_{\text{bis}} N \triangleright \text{Frn}'''(c_1', v) \parallel \text{eBk}''(c_1')$.

(b) $\Gamma_{\text{ext}} \triangleleft N \triangleright \text{Frn}'''(c_1', v) \parallel \text{eBk}''(c_1') \overset{\tau}{\Longrightarrow}_{+1} \Gamma_{\text{ext}} \triangleleft N, c_2' \triangleright Q'$ for some $Q' \in \text{Prc}_\text{A}$ and $c_2' \notin N$. However, from clause (2) when $n = k + 1$, this cannot be the matching move since $\Gamma_{\text{ext}} \models M, c_2 \triangleright \text{Frn} \parallel b!c_2 \parallel c_1!(v, c_2) \parallel \text{Bck}''(c_1) \not\approx^{k+1}_{\text{bis}} N, c_2' \triangleright Q'$.

(c) $\Gamma_{\text{ext}} \triangleleft N \triangleright \text{Frn}'''(c_1', v) \parallel \text{eBk}''(c_1') \overset{\tau}{\Longrightarrow}_0 \Gamma_{\text{ext}} \triangleleft (N', c_2') \triangleright Q'$ for some $Q' \in \text{Prc}_\text{B}$, $N = (N', c_1')$ and $c_2' \notin N$. However, from (100), this cannot be the matching move since $\Gamma_{\text{ext}} \models M \triangleright \text{Frn} \,\|\, b!c_2 \,\|\, c_1!(v, c_2) \,\|\, \text{Bck}''(c_1) \not\approx^k_{\text{bis}} (N', c_2') \triangleright Q'$.

□

Theorem 4 (Strict Inefficiency)**.** *For all $n \geq 0$ and appropriate M we have*

$$\Gamma_{ext} \models M \triangleright \text{Buff} \not\approx^n_{bis} M \triangleright \text{eBuff}$$

Proof. Since:

$$\text{Buff} \stackrel{\text{def}}{=} \text{Frn''}(c_1) \parallel \text{Bck''}(c_1) \qquad\qquad \text{eBuff} \stackrel{\text{def}}{=} \text{Frn''}(c_1) \parallel \text{eBk''}(c_1)$$

we need to show that

$$\Gamma_{\text{ext}} \;\models\; M \triangleright \text{Frn''}(c_1) \parallel \text{Bck''}(c_1) \;\not\approx^{n}_{\text{bis}}\; M \triangleright \text{Frn''}(c_1) \parallel \text{eBk''}(c_1)$$

for any arbitrary n. By Lemma 24.3, Lemma 24.4 and Lemma 24.5 we know that for any n:

$$\text{For any } Q \in \text{Prc}_{\text{A}} \text{ we have } \Gamma_{\text{ext}} \models M \triangleright \text{Frn'''}(c_1, v) \parallel \text{Bck''}(c_1) \;\not\approx^{n+1}_{\text{bis}}\; M \triangleright Q \tag{102}$$

$$\text{For any } Q \in \text{Prc}_{\text{B}} \text{ we have } \Gamma_{\text{ext}} \models M \triangleright \text{Frn'''}(c_1, v) \parallel \text{Bck''}(c_1) \;\not\approx^{n}_{\text{bis}}\; M \triangleright Q \tag{103}$$

$$\Gamma_{\text{ext}} \models M \triangleright \text{Frn'''}(c_1, v) \parallel \text{Bck''}(c_1) \;\not\approx^{n}_{\text{bis}}\; M \triangleright \text{Frn'''}(c_1, v) \parallel \text{eBk''}(c_1) \tag{104}$$

Since $(\text{Frn''}(c_1) \parallel \text{eBk''}(c_1)) \in \text{Prc}_{\text{C}}$, by Lemma 23.1, (102), (103) and (104) we conclude

$$\Gamma_{\text{ext}} \;\models\; M \triangleright \text{Frn''}(c_1) \parallel \text{Bck''}(c_1) \;\not\approx^{n}_{\text{bis}}\; M \triangleright \text{Frn''}(c_1) \parallel \text{eBk''}(c_1)$$

□

5.5 Proving Relative Efficiency

As opposed Theorem 4, the proof for (93) requires us to consider the entire state-space of $\Gamma_{\text{ext}} \triangleleft M \triangleright \text{Buff}$ and $\Gamma_{\text{ext}} \triangleleft M \triangleright \text{eBuff}$. Fortunately, we can apply the compositionality result of Theorem 1 to prove (90) and focus on a subset of this state-space. More precisely, we recall from (94) that

$$\text{Buff} \stackrel{\text{def}}{=} \text{Frn''}(c_1) \parallel \text{Bck''}(c_1) \qquad\qquad \text{eBuff} \stackrel{\text{def}}{=} \text{Frn''}(c_1) \parallel \text{eBk''}(c_1)$$

where both buffer implementation share the common sub-process $\text{Frn''}(c_1)$. We also recall from (82) that this common sub-process was typed *wrt.* the type environment

$$\Gamma_{\text{Frn}} = \texttt{in}\!:\![\mathbf{T}]^{\omega},\, b\!:\![\mathbf{T}_{\text{rec}}]^{\omega},\, c_1\!:\![\mathbf{T}, \mathbf{T}_{\text{rec}}]^{\mathbf{1}}.$$

Theorem 1 thus states that in order to prove (90), it suffices abstract away from this common code and prove Theorem 5

Theorem 5 (Relative Efficiency)**.** $(\Gamma_{ext}, \Gamma_{Frn}) \models M \triangleright \text{eBk''}(c_1) \sqsubseteq^{0}_{\approx bis} M \triangleright \text{Bck''}(c_1)$

Proof. We prove $\Gamma_{\text{ext}}, \Gamma_{\text{Frn}} \models M \triangleright \text{eBk''}(c_1) \sqsubseteq^{0}_{\approx \text{bis}} M \triangleright \text{Bck''}(c_1)$ through the family of relations $\mathcal{R}$ defined below, which includes the required quadruple $\langle(\Gamma_{\text{ext}}, \Gamma_{\text{Frn}}), 0, (M \triangleright \text{eBk''}(c_1)), (M \triangleright \text{Bck''}(c_1))\rangle$.

$$\mathcal{R} \stackrel{\text{def}}{=} \left\{ \begin{array}{l|l} \langle (\Gamma, \Delta), n, (M' \triangleright \text{eBk''}(c)) \;\;, (N' \triangleright \text{Bck''}(c)) \;\;\rangle & \\ \langle (\Gamma, \Delta), n, (M' \triangleright \text{eBk'''}(c, v, c')), (N' \triangleright \text{Bck'''}(v, c'))\rangle & (\Gamma_{\text{ext}}, \Gamma_{\text{Frn}}) \prec \Gamma \\ \langle (\Gamma, \Delta), n, (M'' \triangleright \text{eBk''''}(v, c')) \;, (N' \triangleright \text{Bck'''}(v, c'))\rangle & n \geq 0,\; M' \subseteq N' \\ \langle (\Gamma, \Delta), n, (M'' \triangleright \text{eBk} \parallel d!c') \;\;, (N' \triangleright \text{Bck} \parallel d!c') \;\rangle & c \notin M'',\; M'' \subset N'',\; c \in N'' \\ \langle (\Gamma, \Delta), n, (M'' \triangleright \text{eBk'} \parallel d!c') \;, (N' \triangleright \text{Bck'} \parallel d!c') \rangle & \end{array} \right\}$$

$\mathcal{R}$ observes the transfer property of Definition 6. We here go over some key transitions:

- Consider a tuple from the first clause of the relation, for some Γ, Δ, n and c *i.e.*,

$$(\Gamma, \Delta) \models (M' \triangleright \text{eBk''}(c))\ \mathcal{R}^n\ (N' \triangleright \text{Bck''}(c))$$

We recall from the macros introduced in Section 5.3 that

$$\begin{aligned}\text{eBk''}(c) &= c?(y,z).\texttt{free}\, c.\texttt{out}!y.(\text{eBk} \parallel d!z)\\ \text{Bck''}(c) &= c?(y,z).\texttt{out}!y.(\text{Bck} \parallel d!z)\end{aligned}$$

Whenever (Γ, Δ) allows it, the left hand configuration can perform an input transitions

$$(\Gamma, \Delta) \triangleleft M' \triangleright \text{eBk''}(c) \xrightarrow{c?(v,c')}_0 (\Gamma', \Delta') \triangleleft M' \triangleright \text{eBk'''}(c, v, c')$$

where $\Gamma = \Gamma', c : [\mathbf{T}, \mathbf{T}_{\text{rec}}]^{\mathbf{1}}$ and $\Delta = \Delta', v : \mathbf{T}, c' : \mathbf{T}_{\text{rec}}$. This can be matched by the transition

$$(\Gamma, \Delta) \triangleleft N' \triangleright \text{Bck''}(c) \xrightarrow{c?(v,c')}_0 (\Gamma', \Delta') \triangleleft N' \triangleright \text{Bck'''}(v, c')$$

where we have $(\Gamma', \Delta') \models (M' \triangleright \text{eBk'''}(c, v, c'))\ \mathcal{R}^n\ (N' \triangleright \text{Bck'''}(v, c'))$ from the second clause of $\mathcal{R}$. The matching move for an input action from the right-hand configuration is dual to this. Matching moves for `env`, `alloc` and `free` c actions are analogous.

- Consider a tuple from the first clause of the relation, for some Γ, Δ, n, c, v and c' *i.e.*,

$$(\Gamma, \Delta) \models (M' \triangleright \text{eBk'''}(c, v, c'))\ \mathcal{R}^n\ (N' \triangleright \text{Bck'''}(v, c'))$$

Since $\text{eBk'''}(c, v, c') = \texttt{free}\, c.\texttt{out}!v.(\text{eBk} \parallel d!c')$, a possible transition by the left-hand configuration is the deallocation of channel c:

$$(\Gamma, \Delta) \triangleleft M' \triangleright \text{eBk'''}(c, v, c') \xrightarrow{\tau}_{-1} (\Gamma, \Delta) \triangleleft M'' \triangleright \text{eBk''''}(v, c')$$

where $M' = M'', c$. In this case, the matching move is the empty (weak) transition, since we have $(\Gamma, \Delta) \models (M'' \triangleright \text{eBk''''}(v, c'))\ \mathcal{R}^{n+1}\ (N' \triangleright \text{Bck'''}(v, c'))$ by the third clause of $\mathcal{R}$. Dually, if (Γ, Δ) allows it, the right hand configuration may perform an output action

$$(\Gamma, \Delta) \triangleleft N' \triangleright \text{Bck'''}(v, c') \xrightarrow{\texttt{out}!v}_0 (\Gamma, \Delta, v : \mathbf{T}) \triangleleft N' \triangleright \text{Bck} \parallel d!c'$$

This can be matched by the weak output action

$$(\Gamma, \Delta) \triangleleft M' \triangleright \text{eBk'''}(c, v, c') \xRightarrow{\texttt{out}!v}_{-1} (\Gamma, \Delta, v : \mathbf{T}) \triangleleft M'' \triangleright \text{eBk} \parallel d!c'$$

where $M' = M'', c$; by the fourth clause of $\mathcal{R}$, we know that this a matching move because $(\Gamma, \Delta, v : \mathbf{T}) \models (M'' \triangleright \text{eBk} \parallel d!c')\ \mathcal{R}^{n+1}\ (N' \triangleright \text{Bck} \parallel d!c')$.

□

6 Related Work

A note on terminology: From a logical perspective, a *linear* assumption is one that cannot be weakened nor contracted, while an *affine* assumption cannot be contracted but can be weakened. This leads to a reading of linear as "used exactly once" and of affine as "used at most once". However, in the presence of divergence or deadlock, most linear type systems do not in fact guarantee that a linear resource will be used exactly once. In the discussion below, we will classify such type systems as affine instead.

Linear logic was introduced by Girard [Gir87]; its use as a type system was pioneered by Wadler [Wad91]. Uniqueness typing was introduced by Barendsen and Smetsers [BS96]; the relation to linear logic has since been discussed in a number of papers (see [HHM07]).

Although there are many substructural (linear or affine) type systems for process calculi [AB07, AB08, ABL03, IK04, KS10, Yos04, and others], some specifically for resources [KSW06], the literature on *behaviour* of processes typed under such type systems is much smaller.

Kobayashi *et al.* [KPT99] introduce an affine type system for the π-calculus. Their channels have a polarity (input, output, or input/output) as well as a multiplicity (unrestricted or affine), and an affine input/output can be split as an affine input and an affine output channel. Communication on an affine input/affine output channel is necessarily deterministic, like communication on an affine/unique-after-1 channel in our calculus; however, both processes lose the right to use the channel after the communication, limiting reuse. Although the paper gives a definition of reduction closed barbed congruence, no compositional proof methods are presented.

Yoshida *et al* [YHB07, Hon04] define a linear type system, which uses "action types" to rule out deadlock. The use of action types means that the type system can provide some guarantees that we cannot; this is however an orthogonal aspect of the type system and it would be interesting to see if similar techniques can be applied in our setting. The type system does not have any type that corresponds to uniqueness; instead, the calculus is based on πI to control dynamic sharing of names syntactically, thereby limiting channel reuse. The authors give compositional proof techniques for their behavioural equivalence, but give no complete characterization.

Teller [Tel04] introduces a π-calculus variant with "finalizers", processes that run when a resource has been deallocated. The deallocation itself however is performed by a garbage collector. The calculus comes with a type system that provides bounds on the resources that are used, although the scope of channel reuse is limited in the absence of some sort of uniqueness information. Although the paper defines a bisimulation relation, this relation does not take advantage of type information, and no compositionality results or characterization is given.

Hoare and O'Hearn [HO08] give a trace semantics for a variant of CSP with point-to-point communication and explicit allocation and deallocation of channels, which relies on separation of permissions. However, they do not consider any behavioural theories. Pym and Tofts [PT06] similarly give a semantics for SCCS with a generic notion of resource, based on separation of permissions; they do however consider behaviour. They define a bisimulation relation, and show that it can be characterized by a modal logic. These approaches do not use a type system but opt for an operational

interpretation of permissions, where actions may block due to lack of permissions. Nevertheless, our consistency requirements for configurations (Def. 2) can be seen as separation criteria for permission environments. A detailed comparison between this untyped approach and our typed approach would be worthwhile.

Our unique-after-i type is related to fractional permissions, introduced in [Boy03] and used in settings such as separation logic for shared-state concurrency [BCOP05]. A detailed survey of this field is however beyond the scope of this paper.

The use of substitutions in our LTS (Def. 4) is reminiscent of the name-bijections carried around in spi-calculus bisimulations [BNP01]. In the spi-calculus however this substitution is carried through the bisimulation, and must remain a bijection throughout. Since processes may loose the permission to use channels in our calculus, this approach is too restrictive for us.

Finally, amortisation for coinductive reasoning was originally developed by Keihn *et al.*, [KAK05] and Lüttgen *et al.* [LV06]. It is investigated further by Hennessy in [Hen09], whereby a correspondence with (an adaptation of) reduction-barbed congruences is established. However, neither work considers aspects of resource misuse and the corresponding use of typed analysis in their behavioural and coinductive equivalences.

7 Conclusion

We study a compositional behavioural theory for $R\pi$, a π-calculus variant with mechanisms for explicit resource management. The theory allows us to compare the efficiency of concurrent channel-passing programs *wrt.* their resource usage. We integrate the theory with a substructural type system so as to limit our comparisons to safe programs. In particular, we interpret the type assertions of the type system as permissions, and use this to model (explicit and implicit) permission transfer between the systems being compared and the observer during compositional reasoning. Our contributions are as follows:

1. We define a costed semantic theory that orders systems of safe $R\pi$ programs, based on their costed extensional behaviour when deployed in the context of larger systems; Definition 11. Apart from cost, formulations relating to contextuality are different from those of typed congruences such as [HR04], because of the kind of type system used *i.e.*, substructural.

2. We define a bisimulation-based proof technique that allows us to order $R\pi$ programs coinductively, without the need to universally quantify over the possible contexts that these programs may be deployed in; Definition 6. As far as we are aware, the combination of actions-in-context and costed semantics, used in unison with implicit and explicit transfer of permissions so as to limit the efficiency analysis to safe programs, is new.

3. We prove a number of properties for our bisimulation preorder of Definition 6, facilitating the proof constructions for related programs. Whereas Corollary 1 follows [KAK05, Hen09], Theorem 1 extends the property of compositionality for amortised bisimulations to a typed

setting. Lemma 5, together with the concept of bounded amortisation, appears to be novel altogether.

4. We prove that the bisimulation preorder of Definition 6 is a sound and complete proof technique for the costed behavioural preorder of Definition 11; Theorem 2 and Theorem 3. In order to obtain completeness, the LTS definitions employ non-standard mechanisms for explicit renaming of channel names not known to the context. Also, the concept of (typed) action definability [HR04, Hen08] is different because it needs to take into consideration cost and typeability *wrt.* a substructural type system; the latter aspect also complicates the respective Extrusion Lemma — see Lemma 20.

5. We demonstrate the utility of the semantic theory and its respective proof technique by applying them to reason about the client-server systems outlined in the Introduction and a case study, discussed in Section 5.

Future Work

The extension of our framework to a higher-order setting seems worthwhile. Also, the amalgamation of our uniqueness types with modalities for input and output would give scope for richer notions of subtyping, affecting the respective behavioural theory; it would be interesting to explore how our notions of permission transfer extend to such a setting.

References

[AB07] Lucia Acciai and Michele Boreale. Type abstractions of name-passing processes. In *FSEN'07*, pages 302–317, 2007.

[AB08] Lucia Acciai and Michele Boreale. Responsiveness in process calculi. *Theor. Comput. Sci.*, 409(1):59–93, 2008.

[ABL03] Roberto M. Amadio, Gérard Boudol, and Cédric Lhoussaine. The receptive distributed π-calculus. *ACM Trans. Program. Lang. Syst.*, 25(5):549–577, 2003.

[AKH92] S. Arun-Kumar and Matthew Hennessy. An efficiency preorder for processes. *Acta Inf.*, 29(9):737–760, December 1992.

[BCOP05] Richard Bornat, Cristiano Calcagno, Peter O'Hearn, and Matthew Parkinson. Permission accounting in separation logic. *SIGPLAN Not.*, 40(1):259–270, 2005.

[BM00] Dov Bulka and David Mayhew. *Efficient C++: performance programming techniques*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.

[BNP01] Michele Boreale, Rocco De Nicola, and Rosario Pugliese. Proof techniques for cryptographic processes. *SIAM J. Comput.*, 31(3):947–986, 2001.

[Boy03] John Boyland. Checking interference with fractional permissions. In R. Cousot, editor, *Static Analysis: 10th International Symposium*, volume 2694 of *LNCS*, pages 55–72. Springer, 2003.

[BS96] Erik Barendsen and Sjaak Smetsers. Uniqueness typing for functional languages with graph rewriting semantics. *Mathematical Structures in Computer Science*, 6:579–612, 1996.

[DFH12] Edsko DeVries, Adrian Francalanza, and Matthew Hennessy. Uniqueness typing for resource management in message-passing concurrency. *Journal of Logic and Computation*, 2012. To Appear.

[dVFH10] Edsko de Vries, Adrian Francalanza, and Matthew Hennessy. Uniqueness typing for resource management in message-passing concurrency. *CoRR*, abs/1003.5513, 2010.

[FRS11] Adrian Francalanza, Julian Rathke, and Vladimiro Sassone. Permission-based separation logic for message-passing concurrency. *Logical Methods in Computer Science*, 7(3), 2011.

[Gir87] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–102, 1987.

[Har06] Dana Harrington. Uniqueness logic. *Theoretical Computer Science*, 354(1):24–41, 2006.

[Hen08] Matthew Hennessy. *A Distributed Picalculus*. Cambridge University Proess, Cambridge, UK., 2008.

[Hen09] Matthew Hennessy. A calculus for costed computations, 2009. To appear in LMCS.

[HHM07] Jurriaan Hage, Stefan Holdermans, and Arie Middelkoop. A generic usage analysis with subeffect qualifiers. In *Proceedings of the 12th ACM SIGPLAN International Conference on Functional Programming (ICFP)*, pages 235–246. ACM, 2007.

[HO08] Tony Hoare and Peter O'Hearn. Separation logic semantics for communicating processes. *ENTCS*, 212:3–25, 2008.

[Hon04] Kohei Honda. From process logic to program logic. In *ICFP '04*, pages 163–174, 2004.

[HR04] Matthew Hennessy and Julian Rathke. Typed behavioural equivalences for processes in the presence of subtyping. *Mathematical Structures in Computer Science*, 14:651–684, 2004.

[HT92] Kohei Honda and Mario Tokoro. On asynchronous communication semantics. In Mario Tokoro, Oscar Nierstrasz, and Peter Wegner, editors, *Proceedings of the ECOOP'91 Workshop on Object-Based Concurrent Computing*, volume 612 of *LNCS*, pages 21–51. Springer-Verlag, 1992.

[IK04] Atsushi Igarashi and Naoki Kobayashi. A generic type system for the pi-calculus. *Theor. Comput. Sci.*, 311(1-3):121–163, 2004.

[Jon96] Richard Jones. *Garbage Collection: Algorithms for Automatic Dynamic Memory Management*. John Wiley and Sons, July 1996. With a chapter on Distributed Garbage Collection by Rafael Lins. Reprinted 1997 (twice), 1999, 2000.

[KAK05] Astrid Kiehn and S. Arun-Kumar. Amortised bisimulations. In *FORTE 2005*, volume 3731 of *LNCS*, pages 320–334, 2005.

[KPT99] Naoki Kobayashi, Benjamin C. Pierce, and David N. Turner. Linearity and the pi-calculus. *ACM Trans. Program. Lang. Syst.*, 21(5):914–947, 1999.

[KS10] Naoki Kobayashi and Davide Sangiorgi. A hybrid type system for lock-freedom of mobile processes. *ACM Trans. Program. Lang. Syst.*, 32(5):1–49, 2010.

[KSW06] Naoki Kobayashi, Kohei Suenaga, and Lucian Wischik. Resource usage analysis for the pi-calculus. *CoRR*, abs/cs/0608035, 2006.

[LV06] Gerald Lüttgen and Walter Vogler. Bisimulation on speed: A unified approach. *Theor. Comput. Sci.*, 360(1-3):209–227, 2006.

[Pie02] Benjamin C. Pierce. *Types and programming languages*. MIT Press, Cambridge, MA, USA, 2002.

[Pie04] Benjamin C. Pierce. *Advanced Topics in Types and Programming Languages*. The MIT Press, Cambridge, MA, USA, 2004.

[PT06] David Pym and Chris Tofts. A calculus and logic of resources and processes. *Form. Asp. Comput.*, 18(4):495–517, 2006.

[TA08] T. Terauchi and A. Aiken. A capability calculus for concurrency and determinism. *TOPLAS*, 30(5):1–30, 2008.

[Tel04] David Teller. Recollecting resources in the pi-calculus. In *Proceedings of IFIP TCS 2004*, pages 605–618. Kluwer Academic Publishing, 2004.

[Wad91] Philip Wadler. Is there a use for linear logic? In *PEPM*, pages 255–273, 1991.

[YHB07] Nobuko Yoshida, Kohei Honda, and Martin Berger. Linearity and bisimulation. *Journal of Logic and Algebraic Programming*, 72(2):207 – 238, 2007.

[Yos04] Nobuko Yoshida. Channel dependent types for higher-order mobile processes. *SIGPLAN Not.*, 39(1):147–160, 2004.