Reasoning about Explicit Resource Management* (Abstract)

Edsko de Vries
Trinity College Dublin, Ireland
devriee@cs.tcd.ie

Adrian Francalanza
University of Malta
afra1@um.edu.mt

Matthew Hennessy
Trinity College Dublin, Ireland
Matthew.Hennessy@cs.tcd.ie

1 Introduction

We investigate the behaviour and efficiency of concurrent processes with explicit resource management. Our study is based on a π-calculus variant called $R\pi$ [4] where the only resources available are channels, which must be explicitly allocated before they can be used and can be deallocated when no longer required. A substructural type system guarantees the safe allocation and deallocation of channels, as well as safe channel reuse through strong updates. In this paper we use this type system to give compositional proof techniques for reasoning about the behaviour and efficiency of $R\pi$ processes.

Suppose two servers listen on channels $\mathtt{srv}_1$ and $\mathtt{srv}_2$ to receive a channel which they will use *once* to send a reply back to the client. Consider the following clients:

$$\mathrm{C}_0 \triangleq \mathtt{rec}\, X.\mathtt{alloc}\, x_1.\mathtt{alloc}\, x_2.\mathtt{srv}_1!x_1.x_1?y.\mathtt{srv}_2!x_2.x_2?z.\mathtt{c}!(y,z).X$$
$$\mathrm{C}_1 \triangleq \mathtt{rec}\, X.\mathtt{alloc}\, x.\mathtt{srv}_1!x.x?y.\mathtt{srv}_2!x.x?z.\mathtt{c}!(y,z).X$$
$$\mathrm{C}_2 \triangleq \mathtt{rec}\, X.\mathtt{alloc}\, x.\mathtt{srv}_1!x.x?y.\mathtt{srv}_2!x.x?z.\mathtt{free}\, x.\mathtt{c}!(y,z).X$$

C_0 is an idiomatic π-calculus client. It creates one private "reply" channel to communicate with $\mathtt{srv}_1$ and another to communicate with $\mathtt{srv}_2$. C_1 is more efficient and reuses the same channel for both servers. C_2 is more efficient still and deallocates the channel before recursing. Using our theory we can prove that these clients are equivalent, with C_2 being more efficient than C_1 and C_1 being more efficient than C_0.

We rely on type information to prove this. When C_1 allocates channel x, no other process knows x: from a typing perspective, x is *unique* to C_1. C_1 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 c is *unique-after-1* to C_1: after one communication step, C_1 is once again the only process that knows about x (x once again becomes unique). This means that C_1 can *reuse* x, possibly for values of a different type (strong update), because the type system guarantees that a unique channel is indistinguishable from a freshly allocated channel.

Uniqueness thus records the "positive" information (the guarantee) at one end of a channel corresponding to the "negative" information of an affine permission (a restriction) at the other. Our theory tracks *permission transfer*: in the above example, this manifests itself both as the *explicit* transfer of an affine permission when C_i sends its channel to the server, as well as the *implicit* transfer of this permission back from the server to the client after the communication, allowing us to recover channel uniqueness.

Our contributions are : (1) a compositional coinductive proof method for comparing the behaviour and efficiency of $R\pi$ processes that are well-behaved *wrt.* a uniqueness type system ensuring safe deallocations and strong updates. (2) the definition of a typed, costed, amortized efficiency preorder relation, which embodies the permission transfer discussed above, and a proof that the coinductive proof method is sound and complete with respect to this preorder.

2 Language

Fig. 1 shows the syntax and semantics of $R\pi$. It has the standard π-calculus constructs with the exception of scoping, which is replaced with primitives for explicit channel allocation and deallocation. The syntax

*Supported by SFI project SFI 06 IN.1 1898.

Syntax

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

Reduction Rules

$$\frac{}{M, c:\top \triangleright c!\vec{d}.P \parallel c?\vec{x}.Q \ \rightarrow_0\ M, c:\top \triangleright P \parallel Q\{\vec{d}/\vec{x}\}}\ \textsc{rCom}$$

$$\frac{}{M, c:\top \triangleright \texttt{if}\ c = c\ \texttt{then}\ P\ \texttt{else}\ Q \ \rightarrow_0\ M, c:\top \triangleright P}\ \textsc{rThen}$$

$$\frac{}{M, c:\top, d:\top \triangleright \texttt{if}\ c = d\ \texttt{then}\ P\ \texttt{else}\ Q \ \rightarrow_0\ M, c:\top, d:\top \triangleright Q}\ \textsc{rElse}$$

$$\frac{}{M \triangleright \texttt{rec}\,X.P \ \rightarrow_0\ M \triangleright P\{^{\texttt{rec}\,X.P}/_X\}}\ \textsc{rRec} \qquad \frac{P \equiv P' \quad M \triangleright P' \ \rightarrow_k\ M \triangleright Q' \quad Q' \equiv Q}{M \triangleright P \ \rightarrow_k\ M \triangleright Q}\ \textsc{rStr}$$

$$\frac{}{M, c:\bot \triangleright \texttt{alloc}\,x.P \ \rightarrow_{+1}\ M, c:\top \triangleright P\{^c/_x\}}\ \textsc{rAll} \qquad \frac{}{M, c:\top \triangleright \texttt{free}\,c.P \ \rightarrow_{-1}\ M, c:\bot \triangleright P}\ \textsc{rFree}$$

Figure 1: $R\pi$ Syntax and Reduction Semantics

assumes two separate denumerable sets of channel names c, d and variables x, y, and lets identifiers u, v range over both. The input and channel allocation constructs are binders. The syntax also assumes a denumerable set of process variables X, Y, bound by the recursion construct; we use k, l as metavariables to range over costs. $R\pi$ processes run in a *resource environment* M represented as a function from resource names to $\{\top, \bot\}$, recording whether channels are allocated ($\top$) or deallocated ($\bot$). We refer to a pair $M \triangleright P$ of a resource environment and a process as a *system*.

In order to measure resource usage, we annotate the reduction semantics with a cost k: allocation has a cost of $+1$, deallocation has a cost of -1, and the other reductions carry no cost. We define the reflexive transitive closure $\mathcal{R}^*$ of a costed relation $\mathcal{R}$ as follows:

$$\frac{}{P\ \mathcal{R}_0^*\ P} \qquad \frac{P\ \mathcal{R}_l\ P' \qquad P'\ \mathcal{R}_k^*\ P''}{P\ \mathcal{R}_{l+k}^*\ P''}$$

Example 1. Resource mismanagement in $R\pi$ may result in unexpected behaviour:

$$M, c:\top \triangleright \texttt{free}\,c.(c!1 \parallel c?x.P) \ \parallel\ \texttt{alloc}\,y.(y!42 \parallel y?z.Q) \tag{1}$$

$$\rightarrow_{-1} M, c:\bot \triangleright c!1 \parallel c?x.P \ \parallel\ \texttt{alloc}\,y.(y!42 \parallel y?z.Q) \ \rightarrow_{+1}\ M, c:\top \triangleright c!1 \parallel c?x.P \ \parallel\ c!42 \parallel c?z.Q \tag{2}$$

In this example premature deallocation of channel c (1) allows c to be reallocated by the right process (2). This leads to unintended behaviour through interferences when communicating on c: these interferences are unintended because, intuitively, allocation should yield "fresh" channels. □

In [4] we defined a type system with judgements $\Gamma \vdash P$ and $\Gamma \vdash M$ precluding such unwanted behaviour; the type syntax and the typing rules for processes are shown in Fig. 2. Channel types are denoted as $[\vec{\mathbf{T}}]^a$, where *type attributes* a range over "1" for affine, "$(\bullet, i)$" for unique-after-i ($i \in \mathbb{N}$), and "ω" for unrestricted channels (no usage restrictions or guarantees). The type system is substructural,

Types and Type Attributes

$$\begin{array}{rcll} \mathbf{T} & ::= & [\vec{\mathbf{T}}]^a & \text{(channel type)} \\ & | & \mathbf{proc} & \text{(process)} \end{array} \qquad \begin{array}{rcll} a & ::= & 1 & \text{(affine)} \\ & | & \omega & \text{(unrestricted)} \\ & | & (\bullet, i) & \text{(unique after } i \text{ steps, } i \in \mathbb{N}\text{)} \end{array}$$

Logical rules

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

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

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

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

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

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

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

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

Counting channel usage

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

Type splitting

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

Subtyping

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

Figure 2: Typing processes

$$\frac{\Gamma \vdash P \qquad \Gamma \vdash M \qquad \Gamma \text{ is consistent}}{\Gamma \vdash M \triangleright P}\ \textsc{tSys} \qquad \frac{\forall c \in \mathrm{dom}(\Gamma).\, M(c) = \top}{\Gamma \vdash M}$$

Figure 3: Typing systems

so that typing assumptions can be interpreted as *permissions*. This is especially evident in the rule for parallel composition (tPar) where a permission can be used by either the left process or the right, but not by both. Some permissions can be *split*, however, using contraction (tStr, tCon). For example, an assumption $c : [\vec{\mathbf{T}}]^{(\bullet,i)}$ can be split as $c : [\vec{\mathbf{T}}]^{1}$ and $c : [\vec{\mathbf{T}}]^{(\bullet,i+1)}$ (pUnq). Dually, $c : [\vec{\mathbf{T}}]^{1}$ and $c : [\vec{\mathbf{T}}]^{(\bullet,i+1)}$ can be consolidated again into $c : [\vec{\mathbf{T}}]^{(\bullet,i)}$ (tJoin). Permission splitting and merging plays a major role in the behaviour equivalences we study in Sections 3 and 4.

The type system guarantees that when a process is typed *wrt.* a unique assumption for a channel, $[\vec{\mathbf{T}}]^{(\bullet,0)}$, no other process has access to that channel. This means that deallocation and strong update (changing the object type of a channel) are safe for unique channels. In this paper we take advantage of this type system to provide a behavioural theory for $R\pi$.

3 Labelled Transition System

We introduce a labelled transition system (LTS) dealing with permission ownership and transfer to enable compositional reasoning about resource management in $R\pi$ processes. The LTS is defined over triples of the form $\Gamma \triangleleft M \triangleright P$, where Γ is the typing environment that types the observer. $\Gamma \triangleleft M \triangleright P$ is a *configuration* if there is a global set of permissions Γ_{global} which can be distributed as the permission Γ of the observer and some (existentially quantified) permissions Δ of the process. Both the observer and the process can only have typing assumptions for channels which have been allocated.

The LTS is defined in Fig. 4. Consider rule lOut, describing process output to the observer, which captures the two forms of permission transfer discussed in the introduction. First we have the explicit transfer where the observer, characterized by the type environment $\Gamma, c : [\vec{\mathbf{T}}]^{a}$, gains the permissions $\vec{d} : \vec{\mathbf{T}}$ for the channels received from the process; the process loses these permissions, although this is implicit in the rule (it follows from the typing of the output process). Second, implicit transfer occurs in one of two ways. If the observer has a unique-after-i permission for c, it gets one step closer to recovering full permission on c after the communication, since the process will have lost the corresponding (necessarily affine) permission. Dually, if the observer has an affine permission, this permission is transferred in the opposite direction. In the extreme case where this affine permission is the only one in Γ, the observer loses all knowledge of channel c. This is formalized by the operation $\Gamma, c : [\vec{\mathbf{T}}]^{a-1}$ $(a \neq \bullet)$, given in Fig. 2.

Rule lStr allows the observer to apply structural operations to its permissions, such as splitting a unique permission into a unique-after-1 and an affine permission; see Fig. 2 for the definition of $(\prec)$.

4 Costed Bisimulation and Characterization

We define an amortized bisimulation [6] to compare the efficiency and behaviour of systems $M \triangleright P$ and $N \triangleright Q$ at some *credit n*, where $M \triangleright P$ and $N \triangleright Q$ exhibit the same behaviour but $M \triangleright P$ is more efficient; the credit allows $M \triangleright P$ to do a more expensive action than $N \triangleright Q$ as long as the credit can make up for the difference. The amortized bisimulation is typed *wrt.* to an environment Γ characterizing the observer [5]. Novel in the definition of this bisimulation is that we explicitly allow local bijective renamings σ_Γ

Process moves

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

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

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

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

$$\frac{}{\Gamma \triangleleft M, c:\bot \triangleright \mathtt{alloc}\,x.P \xrightarrow{\tau}_{+1} \Gamma \triangleleft M, c:\top \triangleright P\{c/x\}}\ \textsc{lAll} \qquad \frac{}{\Gamma \triangleleft M, c:\top \triangleright \mathtt{free}\,c.P \xrightarrow{\tau}_{-1} \Gamma \triangleleft M, c:\bot \triangleright P}\ \textsc{lFree}$$

Environment moves

$$\frac{}{\Gamma \triangleleft M, c:\bot \triangleright P \xrightarrow{\mathtt{alloc}}_{+1} \Gamma, c:[\vec{\mathbf{T}}]^\bullet \triangleleft M, c:\top \triangleright P}\ \textsc{lAllE} \qquad \frac{}{\Gamma, c:[\mathbf{T}]^\bullet \triangleleft M, c:\top \triangleright P \xrightarrow{\mathtt{free}\,c}_{-1} \Gamma \triangleleft M, c:\bot \triangleright P}\ \textsc{lFreeE}$$

$$\frac{\Gamma \prec \Gamma'}{\Gamma \triangleleft M \triangleright P \xrightarrow{\mathtt{env}}_0 \Gamma' \triangleleft M \triangleright P}\ \textsc{lStr}$$

Figure 4: LTS Process Moves

of names that are not known to the observer (that is, we must have $c \in \mathrm{dom}(\Gamma)$ implies $c\sigma_\Gamma = c\sigma_\Gamma^{-1} = c$). Bijective renamings are comparable to alpha-renaming of scoped bound names in the standard π-calculus. In our calculus however, processes may regain uniqueness of a channel after "extruding" it, as illustrated in the next example.

Example 2. Consider clients C_0 and C_1 from the introduction and an observer characterized by $\Gamma = \mathtt{srv}_1 : [[\mathbf{Bool}]^1]^\omega, \mathtt{srv}_2 : [[\mathbf{Int}]^1]^\omega$. Since these two clients have different memory behaviour, they need to be typed under different typing environments. The bisimulation relation handles this by relating *configurations*, which existentially quantify over process type environments.

When the clients interact with Γ on $\mathtt{srv}_1$, they both send channels that are not known to Γ (the channel allocations for x_1 and x respectively) and bijective renamings allow us to match these actions. More importantly however, the same situation repeats itself when the clients interact with the observer on $\mathtt{srv}_2$. C_0 sends the yet unused channel allocated for x_2 whereas C_1 reuses the channel allocated for x. From the point of view of the observer, however, the situation is indistinguishable from the first interaction on $\mathtt{srv}_1$, as the server will have lost all permissions for the channel it received from the client (x or x_1) after using it to send a reply. Our LTS allows us to track these permission consumptions and, once again, bijective renamings allow us to match these actions.

It is important that these renamings are *locally* bijective: we rename x to x_1 during the first interaction, and x to x_2 in the second. □

Definition 1 (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 σ_Γ, N' and Q' such that*

$\Gamma \triangleleft (N \triangleright Q)\sigma_\Gamma \overset{\hat{\mu}}{\Longrightarrow}_l \Gamma' \triangleright N' \triangleright Q'$ *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* σ_Γ, M' *and* P' *such that* $\Gamma \triangleleft (M \triangleright P)\sigma_\Gamma \overset{\hat{\mu}}{\Longrightarrow}_k \Gamma' \triangleright M' \triangleright P'$ *where* $\Gamma' \vDash (M' \triangleright P')\,\mathcal{R}^{n+l-k}\,(N' \triangleright Q')$

where $\overset{\hat{\mu}}{\Longrightarrow}_l$ *is* $\overset{\tau}{\rightarrow}_l^*$ *if* $\mu = \tau$ *and* $\overset{\tau}{\rightarrow}_{l_1}\overset{\mu}{\rightarrow}_{l_2}\overset{\tau}{\rightarrow}_{l_3}^*$ $(l = l_1 + l_2 + l_3)$ *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.*

Example 3. *For an observer characterized by* $\Gamma = \mathtt{srv}_1 : [[\mathbf{Bool}]^1]^\omega, \mathtt{srv}_2 : [[\mathbf{Int}]^1]^\omega$*, we can prove* $\Gamma \vDash (M \triangleright \mathrm{C}_1) \sqsubseteq^0_{\approx bis} (N \triangleright \mathrm{C}_0)$ *and* $\Gamma \vDash (M \triangleright \mathrm{C}_2) \sqsubseteq^0_{\approx bis} (N \triangleright \mathrm{C}_1)$*. We however can neither prove that* $\Gamma \vDash (M \triangleright \mathrm{C}_0) \sqsubseteq^n_{\approx bis} (N \triangleright \mathrm{C}_1)$ *nor that* $\Gamma \vDash (M \triangleright \mathrm{C}_1) \sqsubseteq^n_{\approx bis} (N \triangleright \mathrm{C}_2)$ *for any n.*

An important theoretical result is that our amortized bisimulation admits compositional analysis.

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

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

This theorem allows us to abstract away from common code when exhibiting bisimulations by extending the observer environment with the permission environment that characterizes this code.

We also give a sound and complete characterization of our amortized bisimulation in terms of our reduction semantics through a costed version of reduction closed barbed congruence which takes into account the transfer of permissions. This congruence relies on a novel definition of contextuality, which tracks permission transfer in a similar way to Theorem 1.

Definition 2 (Contextuality)**.** *An amortized type-indexed relation* $\mathcal{R}$ *is contextual at environment* Γ *iff whenever* $\Gamma \vDash (M \triangleright P_1)\,\mathcal{R}^n\,(N \triangleright P_2)$*:*

1. *If* $M = M', c : \bot$ *and* $N = N', c : \bot$ *then* $\Gamma, c : [\vec{\mathbf{T}}]^\bullet \vDash (M', c : \top \triangleright P_1)\,\mathcal{R}^n\,(N', c : \top \triangleright P_2)$
2. *If* $\Gamma \prec \Gamma_1, \Gamma_2$ *where* $\Gamma_2 \vdash Q$ *then* $\Gamma_1 \vDash (M \triangleright P_1 \parallel Q)\,\mathcal{R}^n\,(N \triangleright P_2 \parallel Q)$

Definition 3 (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* $M \triangleright P \rightarrow_k M' \triangleright P'$ *then* $N \triangleright Q \rightarrow_l^* N' \triangleright Q'$ *such that* $\Gamma \vDash (M' \triangleright P')\,\mathcal{R}^{n+l-k}\,(N' \triangleright Q')$*;*
2. *if* $N \triangleright Q \rightarrow_l N' \triangleright Q'$ *then* $M \triangleright P \rightarrow_k^* M' \triangleright P'$ *such that* $\Gamma \vDash (M' \triangleright P')\,\mathcal{R}^{n+l-k}\,(N' \triangleright Q')$*.*

Definition 4 (Barb)**.** $(\Gamma \triangleleft M \triangleright P) \Downarrow^{barb}_c \overset{def}{=} c \in \mathrm{dom}(\Gamma)$ *and* $(M \triangleright P) \rightarrow_k^* \equiv (M' \triangleright P' \parallel c!\vec{d})$*.*

Definition 5 (Barb Preservation)**.** *A typed relation* $\mathcal{R}$ *is barb preserving if and only if*

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

Definition 6 (Behavioral Contextual Preorder)**.** $\lesssim^{\Gamma,n}_{beh}$ *is the largest family of amortized typed relations that is barb preserving, cost improving at credit n and contextual at environment* Γ*.*

Theorem 2 (Full Abstraction)**.** $\Gamma \vDash (M \triangleright P) \sqsubseteq^n_{\approx bis} (N \triangleright Q)$ *iff* $\Gamma \vDash (M \triangleright P) \lesssim^n_{beh} (N \triangleright Q)$*.*

Example 4. *For any observer characterized by* $\Gamma = \mathtt{srv}_1 : [[\mathbf{Bool}]^1]^\omega, \mathtt{srv}_2 : [[\mathbf{Int}]^1]^\omega$*, we can choose an appropriate context to show that, for any n, we cannot have* $\Gamma \vDash (M \triangleright \mathrm{C}_0) \lesssim^n_{beh} (N \triangleright \mathrm{C}_1)$ *or* $\Gamma \vDash (M \triangleright \mathrm{C}_1) \lesssim^n_{beh} (N \triangleright \mathrm{C}_2)$*. Despite the quantification over all possible contexts in Definition 6, we can also show that* $\Gamma \vDash (M \triangleright \mathrm{C}_1) \lesssim^0_{beh} (N \triangleright \mathrm{C}_0)$ *and* $\Gamma \vDash (M \triangleright \mathrm{C}_2) \lesssim^0_{beh} (N \triangleright \mathrm{C}_1)$ *following Example 3 and Theorem 2.*

5 Conclusions and Related Work

We outlined how to take advantage of uniqueness information to construct compositional proof techniques for comparing behaviour and amortized resource usage efficiency in $R\pi$. We gave a sound and complete characterization of the proof method in terms of a costed preorder based on the reduction semantics, which relies essentially on a novel definition of contextuality that takes care of permission transfer.

Kobayashi *et al.* [7] introduce an affine type system for the π-calculus and a definition of reduction closed barbed congruence, but no compositional proof methods. Yoshida *et al.* [9] define a linear type system for a calculus based on πI in which dynamic sharing is controlled dynamically, limiting channel reuse. They give compositional proof techniques for their behavioural equivalence, but no complete characterization.

Our unique-after-i type is related to fractional permissions, introduced in [3] and used in settings such as separation logic for shared-state concurrency [2]. A detailed survey of this field is however beyond the scope of this paper.

Pym and Tofts [8] give a behavioural theory for SCCS with resources based on separation of permissions. They define a bisimulation relation, and show that it can be characterized by a modal logic. A comparison between their untyped approach and our typed approach would be worthwhile.

The use of substitutions in our bisimulation is reminiscent of the name-bijections used in spi [1]. In the spi calculus however these bijections are carried through the bisimulation, whereas we use local renamings (one per action). This is essential to enable more channel reuse.

References

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

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

[3] 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.

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

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

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

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

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

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