A resource analysis of the π-calculus

Aaron Turon Mitchell Wand

College of Computer and Information Science
Northeastern University
Boston MA, USA

Abstract

We give a new treatment of the π-calculus based on the semantic theory of separation logic, continuing a research program begun by Hoare and O'Hearn. Using a novel resource model that distinguishes between public and private ownership, we refactor the operational semantics so that sending, receiving, and allocating are commands that influence owned resources. These ideas lead naturally to two denotational models: one for safety and one for liveness. Both models are fully abstract for the corresponding observables, but more importantly both are very simple. The close connections with the model theory of separation logic (in particular, with Brookes's action trace model) give rise to a logic of processes and resources.

Keywords: separation logic, pi-calculus, ownership, resources, scope extrusion, full abstraction

Names play a leading role in the π-calculus [12]: they are both the means of communication, and the data communicated. This paper presents a study of the π-calculus based on a new mechanism for name management, which is in turn rooted in separation logic. The main benefit of this study is a very simple—but fully abstract—denotational semantics for the π-calculus.

Traditionally, the use of names in the π-calculus is governed by lexical, but dynamically-expandable, scope. In the composite process $P|\mathsf{new}\ x.Q$ for example, the channel x is by virtue of scope initially *private* to Q. The prefix $\mathsf{new}\ x$ is not an imperative allocation. It is a binder that remains fixed as Q evolves—a constant reminder that x is private—until Q sends x in a message. At that point, the binder is lifted to cover both P and Q, dynamically "extruding" the scope of x. The π-calculus relies on α-renaming and side conditions about freshness to ensure that its privacy narrative is borne out.

In contrast, work on separation logic has led to models of dynamically-structured concurrency based on resources and ownership, rather than names and scoping [3,5]. From this perspective, programs consist of imperative commands that use certain resources (their "footprint") while leaving any additional resources unchanged. Concurrent processes must divide resources amongst themselves, with each process using only those resources it owns. Ownership makes it possible to constrain concurrent interference, and thereby to reason compositionally about process behavior.

In this paper, we reanalyze the π-calculus in terms of resources and ownership, establishing a clear connection with models of separation logic. The analysis hinges on the use of resources to specify not just that a process can do something, but that other processes cannot.[1] Concretely, channels are resources that can be owned either publicly or privately. Public ownership asserts only that a channel can be used by the owning process. Private ownership asserts moreover that a channel cannot be used by other processes. And the prefix $\mathsf{new}\ x$ becomes an imperative action, allocating an initially private channel.

Armed with this simple resource model, we give a new operational semantics for the π-calculus (§1). The semantics is factored into two layers. The first layer generates the basic labeled transitions, without regard to their global plausibility. The second layer then uniformly interprets those labels as resource transformers, filtering out implausible steps. The two-layer setup is reminiscent of Brookes's semantics for concurrent separation logic [3,2], and allows us to blend message passing and imperative interpretations of actions.

More importantly, the resource model also enables a very simple denotational treatment of the π-calculus. We give two denotational interpretations, both trace-theoretic. The first (§2) captures safety properties only, while the second (§3) is also sensitive to divergence and some branching behavior, along the lines of the failures/divergences model with infinite traces [18]. We prove that each model is fully abstract with respect to appropriate observables.

The semantic foundation reconciles the model theory of separation logic with the π-calculus; what about the proof theory? We sketch an integration of separation logic with refinement calculus for processes (§4). Refinement is justified by the denotational semantics, so the calculus is sound for contextual approximation. Resource reasoning allows us to derive an *interference-free expansion law* that uses privacy assertions to rule out interference on a channel.

To provide an accurate model of the π-calculus, public/private resources must be *conservative* in a certain sense: once a resource has been made public, it is impossible to make it private again. Work in separation logic has shown the usefulness of more "aggressive" resource models that capture not just what can and cannot be done, but assert that certain things *may* not be done. We sketch a few such aggressive resource models (§5.1), including an interpretation

[1] Such a reading of resources has already appeared in *e.g.* deny-guarantee reasoning [6].

of fractional permissions [1] and of session types [10].

Hoare and O'Hearn initiated a study of a π-calculus-like language in terms of separation logic semantics [9]. That study provided the impetus for our work, which goes farther by (1) handling the full calculus, (2) handling liveness, (3) proving full abstraction and (4) building a logic on the semantics. There have also been several fully abstract models of the π-calculus [20,8,7] based on functor categories for modeling scope. Our models complement these by providing an elementary account of behavior, structured around resources and abstract separation logic. A full discussion of related work is in §5.2.

1 A resource-driven operational semantics

There are many variants of the π-calculus; here's ours:

$$P \;::=\; \sum \pi_i.P_i \;\mid\; P \oplus Q \;\mid\; \mathsf{new}\ x.P \;\mid\; P|Q \;\mid\; \mathsf{rec}\ X.P \;\mid\; X$$

$$\pi \;::=\; \bar{e}e' \;\mid\; e(x) \qquad\qquad e \;::=\; x \;\mid\; c$$

We distinguish between external choice (+) and internal choice ($\oplus$), which simplifies the liveness semantics (§3) but is not essential. We also distinguish between channels (c, d) and channel variables (x, y, z) and include a simple grammar of channel expressions (e) ranging over both. A *closed* process has no unbound channel or process variables. Closed processes may, however, refer to channel constants and thereby communicate with their environment.

We write 0 for an empty summation, which is an inert process.

1.1 Generating actions

The operational semantics of closed processes is given in two layers, via two labelled transition systems. In both systems, the labels are (syntactic) *actions*, given by the following grammar:

$$\alpha ::= c!d \;\mid\; c?d \;\mid\; \nu c \;\mid\; \tau \;\mid\; ↯ \qquad (\textsc{Action})$$

Actions record the concrete channels involved in sending, receiving, and allocating, respectively. The action τ, as usual, represents an internal (unobservable) step on the part of the process. The action ↯ represents a fault, caused by using an unowned channel (§1.2). Communication actions are dual: $\overline{c!d} = c?d$ and $\overline{c?d} = c!d$, while $\overline{\nu c}$, $\overline{\tau}$, and $\overline{↯}$ are undefined.

The first transition system generates all conceivable actions associated with a process, without considering whether those actions are globally plausible:

Operational semantics: action generation $P \xrightarrow{\alpha} Q$

$$\cdots + \overline{c}d.P + \cdots \xrightarrow{c!d} P$$

$$\cdots + c(x).P + \cdots \xrightarrow{c?d} P\{d/x\}$$

$$P_1 \oplus P_2 \xrightarrow{\tau} P_i$$

$$\mathsf{new}\ x.P \xrightarrow{\nu c} P\{c/x\}$$

$$\mathsf{rec}\ X.P \xrightarrow{\tau} P\{\mathsf{rec}\ X.P/X\}$$

$$\frac{P \xrightarrow{\alpha} P'}{P|Q \xrightarrow{\alpha} P'|Q} \qquad \frac{Q \xrightarrow{\alpha} Q'}{P|Q \xrightarrow{\alpha} P|Q'}$$

$$\frac{P \xrightarrow{\alpha} P' \qquad Q \xrightarrow{\overline{\alpha}} Q'}{P|Q \xrightarrow{\tau} P'|Q'}$$

According to this semantics, we will have transitions like

$$\mathsf{new}\ x.\mathsf{new}\ y.\overline{x}y.0 \xrightarrow{\nu c} \mathsf{new}\ y.\overline{c}y.0 \xrightarrow{\nu c} \overline{c}c.0 \xrightarrow{c!c} 0$$

where c is allocated twice, and used to communicate with an environment that cannot know it. To filter out such executions, we use resources.

1.2 Resources and action semantics

The execution above is intuitively impossible because, after the first νc action, the process *already owns* the channel c. Similarly, for the process $\mathsf{new}\ x.\overline{x}x.0$ the trace

$$\mathsf{new}\ x.\overline{x}x.0 \xrightarrow{\nu c} \overline{c}c.0 \xrightarrow{c!c} 0$$

is impossible because the channel c, having just been allocated, is unknown to the environment—so no parallel process could possibly be on the other side of the communication, receiving along c.

Formally, resources are elements σ of the domain $\Sigma \triangleq \textsc{Chan} \rightharpoonup \{\mathsf{pub}, \mathsf{pri}\}$, where pub and pri are distinct atoms. If a process is executing with resources σ, it owns the channels $\mathrm{dom}(\sigma)$, and $\sigma(c)$ tells, for each c, whether that ownership is exclusive. Therefore, if $c \in \mathrm{dom}(\sigma)$, the action νc is impossible. Likewise, if $\sigma(c) = \mathsf{pri}$, the action $c!c$ is impossible.

The resources owned at a particular point in time determine not only what is *possible*, but also what is *permissible*. For example, the process $\overline{c}d.0$ immediately attempts a communication along the channel c. If this channel is not allocated (*i.e.*, not owned, *i.e.*, not in $\mathrm{dom}(\sigma)$) then the process is *faulty*: it is attempting to use a dangling pointer.

We interpret actions α as *resource transformers* of type $\Sigma \to \Sigma_\bot^\top$.[2] Since all nondeterminism is resolved during the generation of actions, these transformers are deterministic. A result of $\top$ or $\bot$ represents that an action is not permissible or not possible, respectively.

[2] The notation $\Sigma_\bot^\top$ denotes the set $\{\Sigma, \top, \bot\}$ and implies an ordering $\bot \le \sigma \le \top$ for all $\sigma \in \Sigma$. The order structure follows abstract separation logic [5], and is related to locality (§2).

Given the semantics $\llparenthesis\alpha\rrparenthesis : \Sigma \to \Sigma_\bot^\top$ of actions (defined below), we can define a transition system that *executes* actions according to the currently-owned resources:

Operational semantics: resource sensitivity $\qquad P, \sigma \xrightarrow{\alpha} P', \sigma'$

$$\frac{P \xrightarrow{\alpha} P' \qquad \llparenthesis\alpha\rrparenthesis\sigma = \sigma'}{P, \sigma \overset{\alpha}{\rightarrowtriangle} P', \sigma'} \qquad \frac{P \xrightarrow{\alpha} P' \qquad \llparenthesis\alpha\rrparenthesis\sigma = \top}{P, \sigma \overset{\lightning}{\rightarrowtriangle} 0, \sigma}$$

Successful actions proceed normally, updating the owned resources—note that if $\llparenthesis\alpha\rrparenthesis\sigma = \sigma'$ then in particular $\llparenthesis\alpha\rrparenthesis\sigma \neq \top, \bot$. Impermissible actions noisily fail, generating the faulting label $\lightning$. Impossible actions silently fail to occur.

The semantics of actions is as follows:

Action semantics $\qquad \llparenthesis\alpha\rrparenthesis : \Sigma \to \Sigma_\bot^\top$

$$\llparenthesis c!d\rrparenthesis\sigma \triangleq \begin{cases} \top & \{c, d\} \nsubseteq \mathrm{dom}(\sigma) \\ \sigma[d\ \mathsf{pub}] & \sigma(c) = \mathsf{pub} \\ \bot & \text{otherwise} \end{cases} \qquad \llparenthesis c?d\rrparenthesis\sigma \triangleq \begin{cases} \top & c \notin \mathrm{dom}(\sigma) \\ \sigma[d\ \mathsf{pub}] & \sigma(c) = \mathsf{pub}, \\ & \quad \sigma(d) \neq \mathsf{pri} \\ \bot & \text{otherwise} \end{cases}$$

$$\llparenthesis \nu c\rrparenthesis\sigma \triangleq \begin{cases} \sigma[c\ \mathsf{pri}] & c \notin \mathrm{dom}(\sigma) \\ \bot & \text{otherwise} \end{cases} \qquad \llparenthesis\tau\rrparenthesis\sigma \triangleq \sigma \qquad \llparenthesis\lightning\rrparenthesis\sigma \triangleq \top$$

Allocation is always permitted, but is not possible if the channel is already allocated. Allocated channels are initially private. Sending a channel publicizes it, but the communication is only possible if performed over an already public channel, and only permitted over an allocated channel. A locally-unknown channel received from the environment is known to the environment, and hence public; a locally-known channel received from the environment cannot possibly have been private.

Examples

Consider the process $\mathsf{new}\ x.0$. We have

$$\mathsf{new}\ x.0 \quad \xrightarrow{\nu c} \quad 0$$

for every channel c. It follows that

$$\mathsf{new}\ x.0,\ \varnothing \quad \overset{\nu c}{\rightarrowtriangle} \quad 0,\ [c \mapsto \mathsf{pri}]$$

for every channel c, while executing with more resources

$$\mathsf{new}\ x.0,\ [c \mapsto \mathsf{pri}] \quad \overset{\nu d}{\rightarrowtriangle} \quad 0,\ [c \mapsto \mathsf{pri}] \uplus [d \mapsto \mathsf{pri}]$$

results in constrained allocation: the $\uplus$ here denotes disjoint union, meaning that $c \neq d$. The fact that c was already allocated pruned one trace (preventing it from taking an impossible step), but introduced no new traces. Similarly,

$$\mathsf{new}\ x.\overline{x}x.0 \ \xrightarrow{\nu c} \overline{c}c.0 \ \xrightarrow{c!c} 0$$

but, taking resources into account, we have

$$\mathsf{new}\ x.\overline{x}x.0,\ \varnothing \quad \overset{\nu c}{\rightarrowtriangle} \quad \overline{c}c.0,\ [c \mapsto \mathsf{pri}]$$

at which point the process is stuck: the action $c!c$ is prevented from occurring, because $\llparenthesis c!c \rrparenthesis[c \mapsto \mathsf{pri}] = \bot$. This deadlock is exactly what we expect to see when a process attempts to communicate along a private channel. Finally, we have

$$\mathsf{new}\ x.(\overline{x}x.0|x(y).\overline{y}x.0) \quad \xrightarrow{\nu c} \quad \overline{c}c.0|c(y).\overline{y}c.0 \quad \xrightarrow{\tau} \quad 0|\overline{c}c.0 \quad \xrightarrow{c!d} \quad 0|0$$

which, with resources, yields

$$\mathsf{new}\ x.(\overline{x}x.0|x(y).\overline{y}x.0),\ \varnothing \quad \overset{\nu c}{\rightarrowtriangle} \quad \overline{c}c.0|c(y).\overline{y}c.0,\ [c \mapsto \mathsf{pri}] \quad \overset{\tau}{\rightarrowtriangle} \quad 0|\overline{c}c.0,\ [c \mapsto \mathsf{pri}]$$

Here we see that *internal* communication along a private channel is both possible and permitted: such internal steps appear as τ actions to the resource-sensitive stepping relation, and hence always pass through. On the other hand, the internal communication also leaves the ownership of c unchanged. Because it remains private, the final communication $\overline{c}c$ is stuck, as it should be.

1.3 Process safety

With the simple public/private resource model, faulting occurs only when using an unallocated channel. Our semantic framework can accommodate deallocation, but doing so complicates the full abstraction result, and we wish to focus on the standard π-calculus. Avoiding deallocation allows us to easily characterize "safe" processes: we say $\sigma \vdash P\checkmark$ iff P is closed and all channel constants in P are in $\mathrm{dom}(\sigma)$, and have:

Lemma 1.1 *If $\sigma \vdash P\checkmark$ then $P, \sigma \overset{\lightning}{\not\rightarrow}$, and if furthermore $P, \sigma \xrightarrow{\alpha} P', \sigma'$ then $\sigma' \vdash P'\checkmark$.*

2 Denotational semantics: safety traces

Resources provide an intriguing refactoring of the operational semantics for π-calculus, but their real payoff comes in the elementary denotational model they support. We begin with a simple trace model capturing only (some) safety properties, which allows us to focus on the role of resources. Afterwards we incorporate liveness (§3) and its interaction with resources.

For the safety model, we have traces t, trace sets T and behaviors B:

$$\textsc{Trace} \triangleq \textsc{Action}^* \qquad \textsc{Beh} \triangleq \Sigma \to \textsc{TraceSet}$$
$$\textsc{TraceSet} \triangleq \{T : \varnothing \subset T \subseteq \textsc{Trace},\ T \text{ prefix-closed}\}$$

Processes will denote behaviors: sets of action traces determined by the initially-available resources. Not every action is observable. We follow standard treatments of π-calculus [19,8] in considering τ steps unobservable, and eliding νc steps until just before the allocated channel c is sent over a public channel (a "bound send"). Our denotational semantics shows that the operators of the π-calculus are congruent for these observables, and the cited works prove that similar observables are fully abstract for yet coarser notions of observation. The observables of an action α are a (possibly empty) trace, depending on the available resources:

Action observables $\quad |\alpha|_\sigma : \textsc{Trace}$

$$|\tau|_\sigma \triangleq \epsilon \qquad |\lightning|_\sigma \triangleq \lightning \qquad |c!d|_\sigma \triangleq \begin{cases} \nu d \cdot c!d & \sigma(d) = \mathsf{pri} \\ c!d & \text{otherwise} \end{cases}$$
$$|\nu c|_\sigma \triangleq \epsilon \qquad |c?d|_\sigma \triangleq c?d$$

We write $t{\cdot}u$ or tu for trace concatenation, and ϵ for the empty trace. Although νc is not immediately observable, taking a νc step affects the resources owned by the process, so exposing c later will cause the νc step to visibly reemerge.

The safety behavior of a process can be read determined operationally:

Safety observation $\quad \mathcal{O}[\![P]\!] : \textsc{Beh}$

$$\frac{}{\epsilon \in \mathcal{O}[\![P]\!]\sigma} \qquad \frac{P, \sigma \overset{\alpha}{\rightarrowtail} P', \sigma' \qquad t \in \mathcal{O}[\![P']\!]\sigma'}{|\alpha|_\sigma t \in \mathcal{O}[\![P]\!]\sigma}$$

The goal of the denotational semantics is to calculate the same traces compositionally over process structure.

$\textsc{TraceSet}$ is a complete lattice under the subset order, and behaviors inherit this order structure pointwise: we write $B \sqsubseteq B'$ if $B(\sigma) \subseteq B'(\sigma)$ for all σ and have $(B \sqcup B')(\sigma) = B(\sigma) \cup B'(\sigma)$. The semantic operators are monotonic (in fact, continuous), so we are justified in defining rec as a fixpoint. For the safety semantics, which is based on finite observation, it is the least fixpoint.

The safety trace model is insensitive to branching behavior of processes [21], so internal and external choice are indistinguishable. We interpret both forms of choice using $\sqcup$, merging behaviors from all the alternatives. For empty summations, $\sqcup$ yields the smallest behavior: $\lambda\sigma.\{\epsilon\}$.

The denotation function is parameterized by an environment ρ, here taking channel variables x to channels c, and process variables X to behaviors B. It uses two additional operators, $\rhd$ and $\parallel$, which we will define shortly.

Denotational semantics (safety) $[\![P]\!] : \textsc{Env} \to \textsc{Beh}$

$$\begin{array}{rcl@{\qquad}rcl}
[\![\overline{e}e'.P]\!]^\rho & \triangleq & \rho e!\rho e' \rhd [\![P]\!]^\rho & [\![\sum \pi_i.P_i]\!]^\rho & \triangleq & \bigsqcup_i [\![\pi_i.P_i]\!]^\rho \\
[\![e(x).P]\!]^\rho & \triangleq & \bigsqcup_c \rho e?c \rhd [\![P]\!]^{\rho[x\mapsto c]} & [\![P \oplus Q]\!]^\rho & \triangleq & [\![P]\!]^\rho \sqcup [\![Q]\!]^\rho \\
[\![\mathsf{new}\ x.P]\!]^\rho & \triangleq & \bigsqcup_c \nu c \rhd [\![P]\!]^{\rho[x\mapsto c]} & [\![P|Q]\!]^\rho & \triangleq & [\![P]\!]^\rho \parallel [\![Q]\!]^\rho \\
[\![\mathsf{rec}\ X.P]\!]^\rho & \triangleq & \mu B.\, [\![P]\!]^{\rho[X\mapsto B]} & [\![X]\!]^\rho & \triangleq & \rho(X)
\end{array}$$

The interpretation of prefixed processes resembles the operational semantics: each clause of the denotational semantics generates all locally-reasonable actions, without immediately checking global plausibility. We use $\sqcup$ to join the behaviors arising from each action—once more reflecting nondeterminism—and we update the environment as necessary.

The operator $\alpha \rhd B$ prefixes an action α to a behavior B in a resource-sensitive way, playing a role akin to the second layer of the operational semantics:

Semantic prefixing $\alpha \rhd B : \textsc{Beh}$

$$(\alpha \rhd B)(\sigma) \;\triangleq\; \{\alpha t \;:\; (\!|\alpha|\!)\sigma = \sigma',\; t \in B(\sigma')\} \;\cup\; \{\lightning \;:\; (\!|\alpha|\!)\sigma = \top\} \;\cup\; \{\epsilon\}$$

To maintain prefix-closure, we include ϵ as a possible trace. A quick example:

$$[\![\mathsf{new}\ x.\overline{x}x.0]\!]^\varnothing \;=\; \bigsqcup_c \nu c \rhd [\![\overline{x}x.0]\!]^{x\mapsto c} \;=\; \bigsqcup_c \nu c \rhd c!c \rhd [\![0]\!]^{x\mapsto c} \;=\; \bigsqcup_c \nu c \rhd c!c \rhd \lambda\sigma.\{\epsilon\}$$

This expansion of the definition resembles the traces we see from the first layer of the operational semantics, without taking resources into account. The denotation, recall, is a *behavior*: to extract its set of traces, we must apply it to some particular resource σ. If we use the empty resource, we see that

$$\begin{aligned}
\left(\bigsqcup_c \nu c \rhd c!c \rhd \lambda\sigma.\{\epsilon\}\right)(\varnothing) &= \{\epsilon\} \cup \bigcup_c \{\nu c \cdot t \;:\; t \in (c!c \rhd \lambda\sigma.\{\epsilon\})\,[c \mapsto \mathsf{pri}]\} \\
&= \{\epsilon\} \cup \bigcup_c \{\nu c \cdot t \;:\; t \in \{\epsilon\}\}
\end{aligned}$$

in other words, we have $[\![\mathsf{new}\ x.\overline{x}x.0]\!]^\varnothing(\varnothing) = \{\epsilon\} \cup \bigcup_c \{\nu c\}$. Just as in the operational semantics, the fact that $(\!|c!c|\!)[c \mapsto \mathsf{pri}] = \bot$ prevents the $c!c$ step from being recorded. Here, the prefix closure (in particular, the inclusion of ϵ in every application of $\rhd$) ensures that we see the trace up to the point that we attempt an impossible action.

Finally, we have parallel composition—the most interesting semantic operator. Here we must ask a crucial question for the denotational semantics: if σ is the resource belonging to $P|Q$, what resources do we provide to P and Q? The question does not come up in the operational semantics, which maintains a single, global resource state, but a compositional semantics must answer it.

Consider the process $\mathsf{new}\ x.(\overline{x}c \mid x(z))$. When the process reaches the parallel composition, x will still be private. The privacy of x means that the subprocesses can only communicate with each other (yielding τ), not with the external environment of the process. But the *sub*processes *are* communicating with environments external to themselves—namely, each other. That is, x is private to $\overline{x}c \mid x(z)$, which cannot communicate along it externally, but it is *public* to the *subprocesses* $\overline{x}c$ and $x(z)$, which can.

Formally, we capture this narrative as follows:

Semantic parallel composition $B_1 \parallel B_2 : \textsc{Beh}$

$$(B_1 \parallel B_2)(\sigma) \triangleq \bigcup_{t_i \in B_i(\widehat{\sigma})} (t_1 \parallel t_2)(\sigma) \quad \text{where } \widehat{\sigma}(c) \triangleq \begin{cases} \mathsf{pub} & c \in \mathrm{dom}(\sigma) \\ \text{undefined} & \text{otherwise} \end{cases}$$

The resource σ given to a parallel composition of behaviors is fed in *public-lifted* form ($\widehat{\sigma}$) to the composed behaviors, yielding two sets of traces. For each pair of traces t_1 and t_2 from these sets, we calculate all interleavings $t_1 \| t_2$:

Trace interleavings $t \parallel u : \textsc{Beh}$

$$\begin{aligned} t \parallel u \triangleq\ & \lambda\sigma.\{\epsilon\} && \text{if } t = \epsilon = u \\ \sqcup\ & \alpha \rhd (t' \parallel u) && \text{if } t = \alpha t' \\ \sqcup\ & \alpha \rhd (t \parallel u') && \text{if } u = \alpha u' \\ \sqcup\ & t' \parallel u' && \text{if } t = \alpha t',\ u = \overline{\alpha} u' \end{aligned}$$

Interleaving at first glance appears standard, but note the use of semantic prefixing $\rhd$: *the interleavings are not simply another set of traces, they are given as a* behavior *that must be evaluated.* We evaluate with the *original* resources σ. The effect is that each interleaving is checked with respect to the resources held by the *combined* process. This additional check is the key to making the "declare everything public" approach work, allowing us to take into account channels that are private from the point of view of the combined process, but public between the subprocesses.

An example helps illuminate the definitions: take the process $\overline{d}c \mid d(z)$ with resources $\sigma = [c \mapsto \mathsf{pub}][d \mapsto \mathsf{pri}]$. It is easy to calculate that

$$\begin{aligned} [\![\overline{d}c]\!]^{\varnothing}(\widehat{\sigma}) &= \{\epsilon, d!c\} \\ [\![d(z)]\!]^{\varnothing}(\widehat{\sigma}) &= \{\epsilon\} \cup \{d?e \ :\ e \in \textsc{Chan}\} \\ d!c \parallel d?c &= (d!c \rhd d?c \rhd \lambda\sigma.\{\epsilon\}) \ \sqcup\ (d?c \rhd d!c \rhd \lambda\sigma.\{\epsilon\}) \ \sqcup\ (\lambda\sigma.\{\epsilon\}) \end{aligned}$$

The interleaving $d!c \parallel d?c$ includes the case that $d!c$ and $d?c$ are two sides of the same communication (yielding $\lambda\sigma.\{\epsilon\}$) and the two possible orderings if they are not. From the point of view of $\widehat{\sigma}$, which has lost the information that d is private to the combined process, this is the most we can say. However, the interleaving is built using the prefixing operation $\rhd$, so when we evaluate it with respect to the original σ, some traces will be silently dropped:

$$\begin{array}{rl} & (d!c \parallel d?c)(\sigma) \\ = & (d!c \rhd d?c \rhd \lambda\sigma.\{\epsilon\})(\sigma) \cup (d?c \rhd d!c \rhd \lambda\sigma.\{\epsilon\})(\sigma) \cup (\lambda\sigma.\{\epsilon\})(\sigma) \\ = & \{\epsilon\} \cup \{\epsilon\} \cup \{\epsilon\} \end{array}$$

In particular, for any B we have $(d!c \rhd B)(\sigma) = (d?c \rhd B)(\sigma) = \{\epsilon\}$ because $\sigma(d) = \mathsf{pri}$. We are left only with traces that could arise from internal communication, as expected. That is, $[\![\mathsf{new}\ x.(\overline{x}c|x(y))]\!]^{\varnothing}\,[c \mapsto \mathsf{pub}] = \{\epsilon\}$. More generally, we can show $[\![\mathsf{new}\ x.(\overline{x}c|x(y))]\!]^{\varnothing}\,\sigma = [\![0]\!]^{\varnothing}\,\sigma$ whenever $c \in \mathrm{dom}(\sigma)$.

Because $(\!|\lightning|\!)\sigma = \top$, we have $\lightning \rhd B = \lambda\sigma.\{\lightning, \epsilon\}$ for any B. Thus, when a $\lightning$ action is interleaved, the interleaving is terminated with that action.

In summary, we calculate the traces of $P|Q$ by calculating the traces of P and Q under conservatively public-lifted resources, then evaluating the interleavings with complete information about what resources $P|Q$ actually owns.

Example calculations

Before proving full abstraction, we briefly examine a few of the expected laws. For example, why does $[\![\mathsf{new}\ x.0]\!] = [\![0]\!]$? Expanding the former, we get $\bigsqcup_c \nu c \rhd \lambda\sigma.\{\epsilon\}$. When applied to a particular σ, this behavior yields the simple set $\{\epsilon\}$, because $|\nu c|_\sigma = \epsilon$. This simple example sheds light on the importance of action observation $|-|$: it is crucial for ignoring when, or in some cases whether, channels are allocated.

A more complex example is the following:

$$\begin{array}{rl} [\![\mathsf{new}\ x.\mathsf{new}\ y.P]\!]^\rho & = \bigsqcup_c \nu c \rhd [\![\mathsf{new}\ y.P]\!]^{\rho[x\mapsto c]} \\ & = \bigsqcup_c \nu c \rhd \bigsqcup_d \nu d \rhd [\![P]\!]^{\rho[x\mapsto c, y\mapsto d]} \\ & = \bigsqcup_{c,d} \nu c \rhd \nu d \rhd [\![P]\!]^{\rho[x\mapsto c, y\mapsto d]} \\ & = \bigsqcup_{c,d} \nu d \rhd \nu c \rhd [\![P]\!]^{\rho[x\mapsto c, y\mapsto d]} \\ & = \bigsqcup_d \nu d \rhd \bigsqcup_c \nu c \rhd [\![P]\!]^{\rho[x\mapsto c, y\mapsto d]} \\ & = \bigsqcup_d \nu d \rhd [\![\mathsf{new}\ x.P]\!]^{\rho[y\mapsto d]} \;=\; [\![\mathsf{new}\ y.\mathsf{new}\ x.P]\!]^\rho \end{array}$$

The key step is swapping νc and νd, which relies on the lemma $\nu c \rhd \nu d \rhd$

$B = \nu d \rhd \nu c \rhd B$. The validity of this lemma, again, relies on observability: $|\nu c|_\sigma = |\nu d|_\sigma = \epsilon$ for all σ.

2.1 Congruence for the basic operators

We prove full abstraction by proving a *congruence* result for each operator in the language. For the operators other than parallel composition, we show:

Lemma 2.1 (Core congruences) *All of the following equivalences on closed processes hold:*

(i) $\mathcal{O}[\![0]\!] = [\![0]\!]^{\varnothing}$

(ii) $\mathcal{O}[\![\bar{c}d.P]\!] = c!d \rhd \mathcal{O}[\![P]\!]$

(iii) $\mathcal{O}[\![c(x).P]\!] = \bigsqcup_d c?d \rhd \mathcal{O}[\![P\{d/x\}]\!]$

(iv) $\mathcal{O}[\![\textsf{new}\ x.P]\!] = \bigsqcup_c \nu c \rhd \mathcal{O}[\![P\{c/x\}]\!]$

(v) $\mathcal{O}[\![\sum_i P_i]\!] = \bigsqcup_i \mathcal{O}[\![P_i]\!]$

(vi) $\mathcal{O}[\![P \oplus Q]\!] = \mathcal{O}[\![P]\!] \sqcup \mathcal{O}[\![Q]\!]$

These equivalences are straightforward to show; we prove each by showing containment in both directions. For illustration, we give the proof that $\mathcal{O}[\![c(x).P]\!] \subseteq \bigsqcup_d c?d \rhd \mathcal{O}[\![P\{d/x\}]\!]$:

Proof. Let $\sigma \in \Sigma$ and $t \in \mathcal{O}[\![c(x).P]\!]\sigma$. We analyze cases on the derivation of $t \in \mathcal{O}[\![c(x).P]\!]\sigma$:

Case: $$\boxed{\frac{}{\epsilon \in \mathcal{O}[\![c(x).P]\!]\sigma}}$$

Let d be a channel. Then $t = \epsilon \in c?d \rhd \mathcal{O}[\![P\{d/x\}]\!]$ by definition of $\rhd$. The result follows by monotonicity of $\sqcup$.

Case: $$\boxed{\frac{c(x).P, \sigma \xrightarrow{\alpha} P', \sigma' \qquad t' \in \mathcal{O}[\![P']\!]\sigma'}{|\alpha|_\sigma t' \in \mathcal{O}[\![c(x).P]\!]\sigma}}$$

Reasoning by inversion, we see that there are two subcases:

Subcase: $$\boxed{\exists d.\ \alpha = c?d,\ \llparenthesis c?d \rrparenthesis\sigma = \sigma',\ P' = P\{d/x\}}$$

Then $t = \alpha t' \in \bigsqcup_d c?d \rhd \mathcal{O}[\![P\{d/x\}]\!]$ trivially by the definition of $\rhd$.

Subcase: $$\boxed{\alpha = \lightning,\ c \notin \mathrm{dom}(\sigma),\ P' = 0}$$

Then $t = \alpha t' = \lightning$ because $\mathcal{O}[\![0]\!]\sigma' = \{\epsilon\}$. That $\lightning \in \bigsqcup_d c?d \rhd \mathcal{O}[\![P\{d/x\}]\!]$ again follows easily by the definition of $\rhd$. □

2.2 Congruence for parallel composition

The justification of our treatment of parallel composition goes back to the intuitions from the beginning of the paper: concurrent process must divide resources amongst themselves, with each process using only those resources it owns. We say σ separates into σ_1 and σ_2 if the following conditions hold:

Parallel separation $(\sigma_1 \parallel \sigma_2) \subseteq \Sigma$

$$\sigma \in (\sigma_1 \parallel \sigma_2) \;\triangleq\; \begin{cases} \mathrm{dom}(\sigma) = \mathrm{dom}(\sigma_1) \cup \mathrm{dom}(\sigma_2) \\ \sigma_1(c) = \mathsf{pri} \implies \sigma(c) = \mathsf{pri},\; c \notin \mathrm{dom}(\sigma_2) \\ \sigma_2(c) = \mathsf{pri} \implies \sigma(c) = \mathsf{pri},\; c \notin \mathrm{dom}(\sigma_1) \end{cases}$$

We understand this definition as saying: if σ_1 and σ_2 are resources separately held by P and Q respectively, then σ is *possibly* the resource held by $P|Q$. The subresources σ_i do not uniquely determine a combination σ because resources public to the subprocess may, or may not, be private to the combined process.[3] Separation crisply captures the desired meaning of public and private ownership: if one subprocess owns a resource privately ($\sigma_1(c) = \mathsf{pri}$), then the other subprocess does not own the resource at all ($c \notin \mathrm{dom}(\sigma_2)$), but both processes may own a resource publicly.

To show that that $\mathcal{O}[\![P_1|P_2]\!] = \mathcal{O}[\![P_1]\!] \parallel \mathcal{O}[\![P_2]\!]$, we must show that our strategy of interleaving traces from publicly-lifted resources agrees with the global operational semantics. A key idea is that $\sigma \in \sigma_1 \parallel \sigma_2$ constitutes an invariant relationship between the resources owned by subprocesses (in the denotational semantics) and those owned by the composite process (in the operational semantics). The invariant holds initially because $\sigma \in \widehat{\sigma} \parallel \widehat{\sigma}$.

The unobservability of νc steps complicates matters somewhat: it means there is an additional perspective on resources—call it σ_{den}—owned by a composite process. Generally, σ_{den} underestimates the true resources σ of the operational semantics. Consider the denotational interleaving of two traces t_1 and t_2 from subprocesses P_1 and P_2 respectively. If P_1 allocates a channel, that allocation does not appear immediately in t_1, and hence does not appear immediately in the resources σ_{den} of the interleaving, while it *would* immediately appear in σ, operationally. During denotational interleaving, the same channel can even be owned privately in *both* σ_1 and σ_2. The key observation here is that either both subprocesses eventually reveal a given private channel—in which case the denotational interleaving is filtered out—or at least one subprocess does not—in which case its choice of channel is irrelevant. Altogether,

[3] This means that Σ with $\parallel$ does not form a separation algebra [5]; see §5.1.

the four resources—σ_{op}, σ_{den}, σ_1, and σ_2—can always be related:

$$\mathcal{I}(\sigma_{\mathrm{op}}, \sigma_{\mathrm{den}}, \sigma_1, \sigma_2) \;\triangleq\; \sigma_{\mathrm{op}} \in \sigma_1 \parallel \sigma_2,\;\; \sigma_{\mathrm{den}} = \sigma_{\mathrm{op}} \setminus \{c \;:\; \sigma_1(c) = \mathsf{pri} \vee \sigma_2(c) = \mathsf{pri}\}$$

provided that, within the proof, we apply appropriate channel renamings to avoid conflicts.

Validating parallel composition requires another important lemma, *locality* from abstract separation logic [5]. [4]

Lemma 2.2 (Locality) *If $\sigma \in \sigma_1 \parallel \sigma_2$ then*

- *if $(\!|\alpha|\!)\sigma = \top$ then $(\!|\alpha|\!)\sigma_1 = \top$, and*
- *if $(\!|\alpha|\!)\sigma = \sigma'$ then $(\!|\alpha|\!)\sigma_1 = \top$ or $(\!|\alpha|\!)\sigma_1 = \sigma_1'$ for some σ_1' with $\sigma' \in \sigma_1' \parallel \sigma_2$.*

The lemma characterizes the transformations an action can make given some composite resources σ in terms of its behavior on subresources σ_1. Providing additional resources can never introduce new faults, and if the action does not fault given just σ_1 resources, then the changes it makes to σ must only change the σ_1 portion (framing).

Locality was introduced to characterize the frame rule of separation logic [5], but we use it here to characterize interleaving steps in parallel composition. We have a related lemma for internal communication steps:

Lemma 2.3 (Communication) *If $\sigma \in \sigma_1 \parallel \sigma_2$, $(\!|\alpha|\!)\sigma_1 = \sigma_1'$ and $(\!|\overline{\alpha}|\!)\sigma_2 = \sigma_2'$ then $\sigma \in \sigma_1' \parallel \sigma_2'$.*

We prove each direction of congruence separately:

Lemma 2.4 *If $\mathcal{I}(\sigma_{op}, \sigma_{den}, \sigma_1, \sigma_2)$, $\sigma_i \vdash P_i\checkmark$ and $t \in \mathcal{O}[\![P_1|P_2]\!]\sigma_{op}$ then $t \in (t_1 \parallel t_2)(\sigma_{den})$ for some $t_i \in \mathcal{O}[\![P_i]\!]\sigma_i$.*

Lemma 2.5 *If $\mathcal{I}(\sigma_{op}, \sigma_{den}, \sigma_1, \sigma_2)$, $\sigma_i \vdash P_i\checkmark$, $t_i \in \mathcal{O}[\![P_i]\!]\sigma_i$, and $t \in (t_1 \parallel t_2)(\sigma_{den})$ then $t \in \mathcal{O}[\![P_1|P_2]\!]\sigma_{op}$.*

The first of these two lemmas is easier to prove, because we are given a trace t derived from the operational semantics of the composite processes. This means that the subprocesses are guaranteed not to independently allocate the same channel. The second lemma requires more care, using the insights mentioned above about renaming unexposed channels.

The assumptions $\sigma_i \vdash P_i\checkmark$ are needed to ensure that the processes we are working with do not fault. The reason that faulting is problematic is seen in

[4] For simplicity we avoid the order-theoretic definition here, which requires lifting some of our constructions to 2^{Σ} in a way that is not otherwise useful.

the following example:

$$\begin{array}{rl}
& \mathsf{new}\ x.\overline{c}x.0 \mid c(y).\overline{c}y.\overline{d}y.0),\ [c \mapsto \mathsf{pub}] \\
\xrightarrow{\nu d} & \overline{c}d.0 \mid c(y).\overline{c}y.\overline{d}y.0,\ [c \mapsto \mathsf{pub}, d \mapsto \mathsf{pri}] \\
\xrightarrow{\tau} & 0 \mid \overline{c}d.\overline{d}c.0,\ [c \mapsto \mathsf{pub}, d \mapsto \mathsf{pri}] \\
\xrightarrow{c!d} & 0 \mid \overline{d}c.0,\ [c \mapsto \mathsf{pub}, d \mapsto \mathsf{pub}] \\
\xrightarrow{d!c} & 0 \mid 0,\ [c \mapsto \mathsf{pub}, d \mapsto \mathsf{pub}]
\end{array}$$

The uncomfortable aspect of this derivation is that the channel d occurred in the process initially, even though it was not owned. As a result, the process was able to *allocate* d, in a sense falsely capturing the constant d that initially appeared. In cases where the process allocates a different channel than d, it will fault when it attempts to communicate along the constant channel d. But in this "lucky" case, the operational semantics allows communication along the constant channel.

The denotational semantics, however, *always* generates a fault. It computes the traces compositionally, meaning that a channel d allocated by one subprocess is not immediately available for use by a parallel subprocess.

Our full abstraction result applies only to nonfaulty processes, which, fortunately, is a trivial syntactic check. However, this does limit its applicability to languages that include features like deallocation, which makes checking for safety more difficult.

2.3 Full abstraction

To complete the proof of full abstraction, we must deal with recursion. We begin with the usual unwinding lemma, proved in the standard syntactic way:

Lemma 2.6 (Unwinding) *We have* $\mathcal{O}[\![\mathsf{rec}\ X.P]\!] = \bigsqcup_n \mathcal{O}[\![\mathsf{rec}_n X.P]\!]$*, where* $\mathsf{rec}_0 X.P \triangleq \mathsf{rec}\ X.X$ *and* $\mathsf{rec}_{n+1} X.P \triangleq P\{\mathsf{rec}_n X.P/X\}$*.*

We also have the standard substitution lemmas:

Lemma 2.7 (Substitution) *We have* $[\![P[Q/X]]\!]^\rho = [\![P]\!]^{\rho[X \mapsto Q]}$ *and* $[\![P[c/x]]\!]^\rho = [\![P]\!]^{\rho[x \mapsto c]}$*.*

Combined these lemmas with the previous congruence results, it is straightforward to show the following theorem relating the observed operational traces to those calculated denotationally:

Theorem 2.8 (Congruence) *If* P *is closed,* $\sigma \vdash P\checkmark$ *then* $\mathcal{O}[\![P]\!]\sigma = [\![P]\!]^{\varnothing}\sigma$*.*

To prove this theorem, we must generalize it to deal with open terms. We do this by introducing a *syntactic environment* η as a finite map taking channel variables to channels and process variables to closed processes. Given a syntactic environment η the corresponding semantic environment $\widehat{\eta}$ is given by:

$$(\widehat{\eta})(x) \triangleq \eta(x) \qquad (\widehat{\eta})(X) \triangleq \mathcal{O}[\![\eta(X)]\!]$$

We write ηP for the application of η as a syntactic substitution on P. The needed induction hypothesis for congruence is then

$$\text{if } \sigma \vdash \eta P\checkmark \text{ then } \mathcal{O}[\![\eta P]\!]\sigma = [\![P]\!]^{\widehat{\eta}}\sigma.$$

Define $P =_{\text{DEN}} Q$ iff $[\![P]\!]^{\rho}\sigma = [\![Q]\!]^{\rho}\sigma$ for all σ such that $\sigma \vdash P\checkmark$ and $\sigma \vdash Q\checkmark$. Likewise, let $P =_{\text{OP}} Q$ iff $\mathcal{O}[\![C[P]]\!]\sigma = \mathcal{O}[\![C[Q]]\!]\sigma$ for all contexts C with $\sigma \vdash C[P]\checkmark$ and $\sigma \vdash C[Q]\checkmark$. Full abstraction follows by compositionality:

Theorem 2.9 (Full abstraction) $P =_{\text{DEN}} Q$ *iff* $P =_{\text{OP}} Q$.

3 Denotational semantics: adding liveness

To round out our study of π-calculus, we must account for liveness properties. Liveness in process algebra appears under diverse guises, differing in sensitivity to branching behavior and divergence [21]. Each account of liveness corresponds to some choice of basic observable: given a process P and a context C, what behavior of $C[P]$ matters?

The standard observable for the π-calculus is barbed bisimilarity [13], which sits quite far on the branching side of the linear-branching time spectrum [21]. Here, we choose a treatment more in the spirit of linear time: an adaptation of acceptance traces [8]. This choice is partly a matter of taste, but it also allows us to stick with a purely trace-theoretic semantics, which keeps the domain theory to a minimum. We do not see any immediate obstacles to applying our resource-based handling of names to a branching-time semantics. Branching sensitivity and resource-sensitivity seem largely orthogonal, though of course branches may be pruned when deemed impossible given the owned resources.

3.1 Liveness observables

We say that a process *diverges* if it *can* perform an infinite sequence of unobservable (*i.e.*, internal) steps without any intervening interactions with its environment—which is to say, the process can livelock. On the other hand, a process that can make *no* further unobservable steps is blocked (waiting for interaction from its environment) or deadlocked.

The basic observables in our liveness model are:

- A finite sequence of interactions, after which the process diverges or faults;
- A finite sequence of interactions, after which the process is blocked, along with which channels it is blocked on (none for deadlock); and

- An infinite sequence of interactions.

Notice that we have conflated divergence and faulting: we view both as erroneous behavior. In particular, we view any processes that are capable of immediately diverging or faulting as equivalent, regardless of their other potential behavior. This perspective is reasonable—meaning that it yields a congruence—because such behavior is effectively uncontrollable. For example, if P can immediately diverge, so can $P|Q$ for any Q.

Formally, we add a new action δ_Δ which records that a process is blocked attempting communication along the finite set of *directions* Δ:

$$\alpha \;::=\; \cdots \;\mid\; \delta_\Delta \qquad \Delta \subseteq_{\text{fin}} \textsc{Dir} \triangleq \{c! \;:\; c \in \textsc{Chan}\} \cup \{c? \;:\; c \in \textsc{Chan}\}$$

We then define

$$\textsc{LTrace} \;\triangleq\; \textsc{NTAction}^*; \{\lightning, \delta_\Delta\} \;\cup\; \textsc{NTAction}^\omega \qquad \textsc{LBeh} \;\triangleq\; \Sigma \to 2^{\textsc{LTrace}}$$

where NTAction (for “non-terminating action”) refers to all actions except for $\lightning$ or blocking actions δ_Δ. Thus finite liveness traces must end with either a δ_Δ action or a $\lightning$ action, whereas neither of these actions can appear in an infinite trace.

Each liveness trace encompasses a *complete* behavior of the process: either the process continues interacting indefinitely, yielding an infinite trace, or diverges, faults or gets stuck after a finite sequence of interactions. Therefore, sets of liveness traces are not prefixed-closed.

As with the safety traces, we can observe liveness traces from the operational semantics. However, we do so using the *greatest* fixpoint of the following rules:

Liveness observation $\qquad \mathcal{LO}[\![P]\!] : \textsc{LBeh}$

$$\frac{\begin{array}{c} P,\sigma \xrightarrow{\alpha} P',\sigma' \\ \alpha \neq \lightning \qquad t \in \mathcal{LO}[\![P']\!]\sigma' \end{array}}{|\alpha|_\sigma t \in \mathcal{LO}[\![P]\!]\sigma}\,\text{gfp} \qquad \frac{P,\sigma \xrightarrow{\lightning}}{\lightning \in \mathcal{LO}[\![P]\!]\sigma}\,\text{gfp} \qquad \frac{P,\sigma \text{ blocked } \Delta}{\delta_\Delta \in \mathcal{LO}[\![P]\!]\sigma}\,\text{gfp}$$

where P,σ blocked Δ means that P,σ can only take communication steps, and Δ contains precisely the directions of available communication. Since the owned resources influence which communications are possible, they also influence the directions on which a process is blocked:

$$\delta_{\{c!\}} \in \mathcal{LO}[\![\bar{c}c.0]\!][c \mapsto \mathsf{pub}] \qquad \delta_\varnothing \in \mathcal{LO}[\![\bar{c}c.0]\!][c \mapsto \mathsf{pri}]$$

The action $\delta_\varnothing$ reflects a completely deadlocked process, and is for example the sole trace of the inert process 0.

Defining the observations via a greatest fixpoint allows for infinite traces to be observed, but also means that if a process diverges after a trace t, its behavior will contain all traces tu, in particular $t\lightning$. For example, suppose $P, \sigma \xrightarrow{\tau} P, \sigma$. If t is any liveness trace whatsoever, we can use the first inference rule to show, coinductively, that $t \in \mathcal{LO}[\![P]\!]\sigma$. We merely assume that $t \in \mathcal{LO}[\![P]\!]\sigma$, and derive that $|\tau|_\sigma t = t \in \mathcal{LO}[\![P]\!]\sigma$. Thus, divergence is "catastrophic" (as in failures/divergences [4]).

An important step toward making these observables coherent is the notion of *refinement*. In general, saying that P refines Q (or P "implements" Q) is to say that every behavior of P is a possible behavior of Q. In other words, P is a more deterministic version of Q. We define a refinement order on traces:

$$t \sqsubseteq t \qquad t\delta_\Delta \sqsubseteq t\delta_{\Delta'} \text{ if } \Delta' \subseteq \Delta \qquad tu \sqsubseteq t\lightning$$

which we lift to sets of traces as: $T \sqsubseteq U$ iff $\forall t \in T.\ \exists u \in U.\ t \sqsubseteq u$. This notion of refinement, which closely follows that of acceptance traces [8], says that an implementation must allow at least the external choices that its specification does. It also treats faulting as the most permissive specification: if Q faults, then any P will refine Q. Moreover, any two immediately-faulting processes are equivalent. Since faulting and divergence are treated identically, the same holds for divergent processes. Thus, the simple refinement ordering on traces has an effect quite similar to the closure conditions imposed in failures/divergences semantics.

The ordering on trace sets inherits the complete lattice structure of 2^{LTrace}, as does the pointwise order on LBeh. We again exploit this fact when interpreting recursion.

3.2 Liveness semantics

To complete the semantic story, we need to interpret blocking actions. We define

$$(\!|\delta_\Delta|\!)\sigma \triangleq \begin{cases} \top & \exists c.\ (c! \in \Delta \vee c? \in \Delta) \wedge c \notin \mathrm{dom}(\sigma) \\ \sigma & \text{otherwise} \end{cases}$$
$$|\delta_\Delta|_\sigma \triangleq \delta_{\Delta'} \text{ where } \Delta' = \Delta \upharpoonright \{c \ :\ \sigma(c) = \mathsf{pub}\}$$

which shows the interaction between resources and blocking: blocking on a private resource is possible, but unobservable (*cf.* projection on δ in [2]). For example, we have

$$(\!|\delta_{\{c!\}}|\!)[c \mapsto \mathsf{pub}] = [c \mapsto \mathsf{pub}] \qquad |\delta_{\{c!\}}|_{[c\mapsto\mathsf{pub}]} = \delta_{\{c!\}}$$
$$(\!|\delta_{\{c!\}}|\!)[c \mapsto \mathsf{pri}] = [c \mapsto \mathsf{pri}] \qquad |\delta_{\{c!\}}|_{[c\mapsto\mathsf{pri}]} = \delta_{\varnothing}$$

The denotational semantics for liveness, $\mathcal{L}[\![-]\!]$, is largely the same as that for safety, except for the following clauses:

$$\begin{aligned}\mathcal{L}[\![\mathsf{rec}\ X.P]\!]^{\rho} &\triangleq\ \nu B.\mathcal{L}[\![P]\!]^{\rho[X\mapsto B]}\\ \mathcal{L}\Big[\!\!\Big[\textstyle\sum \pi_i.P_i\Big]\!\!\Big]^{\rho} &\triangleq \big(\bigsqcup \mathcal{L}[\![\pi_i.P_i]\!]^{\rho}\big) \sqcup \big(\delta_{\{\mathrm{dir}(\rho\pi_i)\}} \rhd \lambda\sigma.\varnothing\big)\end{aligned}$$

Recursion is given by a greatest fixpoint, as expected. A summation of prefixed actions now generates a corresponding blocking set, recording the external choice (where dir extracts the direction of a prefix). The blocking action is "executed" using the prefixing operator $\rhd$ so that the actual observed action corresponds to the available resources, as in the example above.

Finally, we use the following definition of interleaving:

$$\begin{array}{rlll} t \parallel u \triangleq_{\mathrm{gfp}} & \alpha \rhd (t' \parallel u) & \text{if } t = \alpha t', & \alpha \text{ not blocking}\\ \sqcup & \alpha \rhd (t \parallel u') & \text{if } u = \alpha u', & \alpha \text{ not blocking}\\ \sqcup & \delta_{\Delta\cup\Delta'} & \text{if } t = \delta_{\Delta},\ u = \delta_{\Delta'}, & \overline{\Delta} \pitchfork \Delta'\\ \sqcup & t' \parallel u' & \text{if } t = \alpha t',\ u = \overline{\alpha} u' & \end{array}$$

Liveness interleaving is given by a greatest fixpoint. An infinite sequence of internal communications (operationally, an infinite sequence of τ moves) therefore yields *all* possible traces, including faulting ones, as it should. An interleaved trace is blocked only when both underlying traces are, and only when they do not block in opposite directions ($\overline{\Delta}$ is Δ with directions reversed, and $\pitchfork$ denotes empty intersection). If two processes are blocked in opposite directions, then their parallel composition is in fact *not* blocked, since they are willing to communicate with each other (*cf* stability [4]).

3.3 Full abstraction

The proof of full abstraction is structured similarly to the proof for the safety semantics. Congruence proofs must take into account blocking actions, which is straightforward in all cases except for parallel composition. There, we require a lemma:

Lemma 3.1 (Blocking congruence) *Suppose $\mathcal{I}(\sigma_{op}, \sigma_{den}, \sigma_1, \sigma_2)$. Then*

- *If $\delta_{\Delta_i} \in \mathcal{LO}[\![P_i]\!]\sigma_i$ and $\Delta_1 \pitchfork \overline{\Delta_2}$ then $|\delta_{\Delta_1\cup\Delta_2}|_{\sigma_{den}} \in \mathcal{LO}[\![P_1|P_2]\!]\sigma_{op}$.*
- *If $\delta_{\Delta} \in \mathcal{LO}[\![P_1|P_2]\!]\sigma_{op}$ then $\delta_{\Delta_i} \in \mathcal{LO}[\![P_i]\!]\sigma_i$ for some Δ_1, Δ_2 with $\Delta_1 \pitchfork \overline{\Delta_2}$ and $|\delta_{\Delta_1\cup\Delta_2}|_{\sigma_{den}} = \delta_{\Delta}$.*

Defining $=_{\mathrm{LDen}}$ and $=_{\mathrm{LOp}}$ analogously to the safety semantics, we again have full abstraction:

Theorem 3.2 (Full abstraction) *$P =_{\mathrm{LDen}} Q$ iff $P =_{\mathrm{LOp}} Q$.*

4 Logic

We now sketch a logic for reasoning about the safety semantics of processes. The logic proves *refinement* between open processes—denotationally, trace containment; operationally, contextual approximation. The refinements are qualified by assertions about owned resources, which is what makes the logic interesting. The basic judgment of the logic is $\Gamma \vdash p \blacktriangleright P \sqsubseteq Q$, which says the traces of P are traces of Q, as long as the initial resources and environment, respectively, satisfy assertions p and Γ (defined below).

Resource assertions p are as follows:

$$p \ ::= \ \mathsf{true} \ \mid \ \mathsf{false} \ \mid \ p \wedge q \ \mid \ p \vee q \ \mid \ p * q \ \mid \ x\ \mathsf{pub} \ \mid \ x\ \mathsf{pri} \ \mid \ x = y \ \mid \ x \neq y$$

and we let $x\ \mathsf{known} \triangleq x\ \mathsf{pub} \vee x\ \mathsf{pri}$. Satisfaction of assertions depends on both the environment and resources, as in these illustrative cases:

$$\rho, \sigma \vDash x\ \mathsf{pub} \ \triangleq \ \sigma(\rho(x)) = \mathsf{pub}$$

$$\rho, \sigma \vDash p_1 * p_2 \triangleq \exists \sigma_1, \sigma_2 . \sigma = \sigma_1 \uplus \sigma_2 \text{ and } \rho, \sigma_i \vDash p_i$$

Resource assertions like $x\ \mathsf{pub}$ are intuitionistic [17]; without deallocation there is no reason to use the classical reading, which can assert nonownership. We are using the standard interpretation of separation logic's $*$ as disjoint separation to enable *sequential* reasoning about resource transformers in our logic. Action interpretations $(\!|\alpha|\!)$ are local with respect to $*$, just as they were for $\|$.

Environment assertions Γ constrain process variables:

$$\Gamma \ ::= \ \varnothing \ \mid \ \Gamma, \ (p \blacktriangleright X \sqsubseteq P)$$

$$\rho \vDash (p \blacktriangleright X \sqsubseteq P) \quad \triangleq \quad \forall \sigma.\ (\rho, \sigma \vDash p) \implies \rho(X)(\sigma) \subseteq [\![P]\!]^{\rho}\, \sigma$$

The definition of entailment is thus:

$$\Gamma \vDash p \blacktriangleright P \sqsubseteq Q \quad \triangleq \quad \forall \rho, \sigma.\ (\rho \vDash \Gamma \ \wedge \ \rho, \sigma \vDash p) \implies [\![P]\!]^{\rho}\, \sigma \subseteq [\![Q]\!]^{\rho}\, \sigma$$

By qualifying refinements by resource assertions we can incorporate Hoare logic-like reasoning. Take, for example, the rule

$$\frac{\Gamma \vdash p * (x\ \mathsf{pub} \wedge y\ \mathsf{pub}) \blacktriangleright P \sqsubseteq Q}{\Gamma \vdash p * (x\ \mathsf{pub} \wedge y\ \mathsf{known}) \blacktriangleright \overline{x}y.P \sqsubseteq \overline{x}y.Q}$$

for sending over a public channel. It is a kind of congruence rule, but we shift resource assumptions for the subprocesses, corresponding to the Hoare triple

$$\{p * (x\ \mathsf{pub} \wedge y\ \mathsf{known})\}\ \overline{x}y\ \{p * (x\ \mathsf{pub} \wedge y\ \mathsf{pub})\}$$

The syntactic structure of prefixes (rather than sequential composition) prevents a clean formulation of the logic using Hoare triples. This is why the frame p is included, rather than added via a separate frame rule; we are using "large" rather than "small" axioms [15]. A better treatment is possible if we semantically interpret prefixing as sequential composition, which requires a variables-as-resources model [16].

For sending over a private channel, we have an axiom: $\overline{x}y.P$ refines *any* process when x is private, because $\overline{x}y.P$ is stuck. The corresponding Hoare triple is $\{x\ \mathsf{pri} \wedge y\ \mathsf{known}\}\ \overline{x}y\ \{\mathsf{false}\}$.

Here is a fragment of the logic, focusing on resource-sensitive rules:

A selection of logical rules for safety behavior $\Gamma \vdash p \blacktriangleright P \sqsubseteq Q$

$$\frac{\Gamma \vdash p * (x\ \mathsf{pub} \wedge y\ \mathsf{pub}) \blacktriangleright P \sqsubseteq Q}{\Gamma \vdash p * (x\ \mathsf{pub} \wedge y\ \mathsf{known}) \blacktriangleright \overline{x}y.P \sqsubseteq \overline{x}y.Q} \qquad \frac{}{\Gamma \vdash x\ \mathsf{pri} \wedge y\ \mathsf{known} \blacktriangleright \overline{x}y.P \sqsubseteq Q}$$

$$\frac{\Gamma \vdash (p * x\ \mathsf{pub}) \wedge y\ \mathsf{pub} \blacktriangleright P \sqsubseteq Q \qquad y \notin \mathrm{fv}(p, \Gamma)}{\Gamma \vdash p * x\ \mathsf{pub} \blacktriangleright x(y).P \sqsubseteq x(y).Q} \qquad \frac{}{\Gamma \vdash x\ \mathsf{pri} \blacktriangleright x(y).P \sqsubseteq Q}$$

$$\frac{\Gamma \vdash p * x\ \mathsf{pri} \blacktriangleright P \sqsubseteq Q \qquad x \notin \mathrm{fv}(p, \Gamma)}{\Gamma \vdash p \blacktriangleright \mathsf{new}\ x.P \sqsubseteq \mathsf{new}\ x.Q} \qquad \frac{\Gamma \vdash \widehat{p} \blacktriangleright P_i \sqsubseteq Q_i}{\Gamma \vdash p \blacktriangleright P_1 | P_2 \sqsubseteq Q_1 | Q_2}$$

$$\frac{p \blacktriangleright X \sqsubseteq P \in \Gamma}{\Gamma \vdash p \blacktriangleright X \sqsubseteq P} \qquad \frac{\Gamma, p \blacktriangleright X \sqsubseteq Q \vdash p \blacktriangleright P \sqsubseteq Q}{\Gamma \vdash p \blacktriangleright \mathsf{rec}\ X.P \sqsubseteq Q} \qquad \frac{p \vDash p' \qquad \Gamma \vdash p' \blacktriangleright P \sqsubseteq Q}{\Gamma \vdash p \blacktriangleright P \sqsubseteq Q}$$

The congruence rule for parallel composition performs public-lifting $\widehat{p}$ on resource assertions (by replacing pri by pub in the assertion).

Fixpoint induction is resource-qualified as well. We reason about the body P of a recursive definition $\mathsf{rec}\ X.P$ using a hypothetical bound on X as the induction hypothesis. That hypothesis, however, is only applicable under the *same* resource assumptions p that were present when it was introduced—making p the loop invariant.

In addition to these resource-sensitive rules, we have the usual laws of process algebra, including the expansion law. Combining those laws with the ones we have shown, we can derive an *interference-free* expansion law, as in this simplified version: $\Gamma \vdash x\ \mathsf{pri} \wedge y\ \mathsf{known} \blacktriangleright \overline{x}y.P | x(z).Q \equiv P | Q\{y/z\}$.

5 Discussion

5.1 Future work: richer resources

Our resource model captures exactly the guarantees provided by the π-calculus: until a channel is exposed, it is unavailable to the environment; afterwards, all bets are off. This property is reflected in the fact that Σ is not a separation algebra, since $c\ \mathsf{pub} \parallel c\ \mathsf{pub}$ can result in $c\ \mathsf{pub}$ or $c\ \mathsf{pri}$. No amount of public ownership adds up definitively to private ownership.

Rather than using resources to model the guarantees of a language, we can instead use them to enforce guarantees we intend of programs, putting ownership "in the eye of the asserter" [14]. We can then recover privacy just as Boyland showed [1] how to recover write permissions from read permissions: via a fractional model of ownership, $\Sigma_{\textsc{Frac}} \triangleq \textsc{Chan} \to [0,1]$. Unlike traditional fractional permissions, owning a proper fraction of a channel does not limit what can be done with the channel—instead, it means that the environment is *also* allowed to communicate on the channel. The fractional model yields a separation algebra, using (bounded) summation for resource addition. An easy extension is distinguishing send and receive permissions, so that interference can be ruled out in a direction-specific way.

One can also imagine encoding a session-type discipline [10] as a kind of resource: $\Sigma_{\textsc{Sess}} \triangleq \textsc{Chan} \rightharpoonup \textsc{Session}$ where

$$s \in \textsc{Session} \ ::= \ \ell.s \oplus \ell.s \ \mid \ \ell.s \mathbin{\&} \ell.s \ \mid \ !.s \ \mid \ ?.s \ \mid \ \mathsf{end}$$

Separation of session resources corresponds to matching up dual sessions, and actions work by consuming the appropriate part of the session. Ultimately, such resource models could yield rely-guarantee reasoning for the π-calculus, borrowing ideas from deny-guarantee [6]. A challenge for using these models is managing the ownership protocol in a logic: how are resources consistently attached to channels, and how are resources split when reasoning about parallel composition? We are far from a complete story, but believe our semantics and logic can serve as a foundation for work in this direction.

5.2 Related work

Hoare and O'Hearn's work [9] introduced the idea of connecting the model theory of separation logic with the π-calculus, and provided the impetus for the work presented here. Their work stopped short of the full π-calculus, modelling only point-to-point communication and only safety properties. Our liveness semantics, full abstraction results, and refinement calculus fill out the rest of the story, and they all rely on our new resource model. In addition, our semantics has clearer connections to both Brookes's action trace model [2] and abstract separation logic [5].

Previous fully abstract models of the π-calculus are based on functor categories [20,8,7], faithfully capturing the traditional role of scope for privacy in the π-calculus. Those models exploit general, abstract accounts of recursion, nondeterminism, names and scoping in a category-theoretic setting. We have similarly sought connections with a general framework, but have chosen resources, separation and locality as our foundation.

An immediate question is: why do we get away with so much less mathematical scaffolding? This question is particularly pertinent in the comparison with Hennessy's work [8], which uses a very similar notion of observation. Hennessy's full abstraction result is proved by extracting, from his functor-categorical semantics, a set of acceptance traces, and showing that this extraction is injective and order preserving. The force of this "internal full abstraction" is that the functor-categorical meaning of processes is completely determined by the corresponding acceptance traces. But note, these traces are *not* given directly via a compositional semantics: they are extracted only after the compositional, functor-categorical semantics has been applied. What we have shown, in a sense, is that something like acceptance traces for a process can be calculated directly, and compositionally, from process syntax.

Beyond providing a new perspective on the π-calculus, we believe the resource-oriented approach will yield new reasoning techniques, as argued above. We have also emphasized concreteness, giving an elementary model theory based on sets of traces.

Finally, it is worth noting that substructural type systems have been used to derive strong properties (like confluence) in the π-calculus [11], just as we derived interference-free expansion. Here, we have used a resource theory to explain the π-calculus as it is, rather than to enforce additional discipline. But the ideas of §5.1 take us very much into the territory of discipline enforcement. More work is needed to see what that territory looks like for the resource-based approach.

Acknowledgements

We are grateful to Paul Stansifer and Tony Garnock-Jones for feedback on drafts of this paper, and to the anonymous reviewers who provided guidance on presentation. The first author has been generously supported by a grant from Microsoft Research (Cambridge).

References

[1] Boyland, J., *Checking Interference with Fractional Permissions*, in: *SAS*, 2003.

[2] Brookes, S., *Traces, Pomsets, Fairness and Full Abstraction for Communicating Processes*, in: *CONCUR*, 2002, pp. 45 –71.

[3] Brookes, S., *A semantics for concurrent separation logic*, TCS **375** (2007), pp. 227–270.

[4] Brookes, S. D. and A. W. Roscoe, *An Improved Failures Model for Communicating Processes*, in: *Seminar on Concurrency*, 1984.

[5] Calcagno, C., P. W. O'Hearn and H. Yang, *Local Action and Abstract Separation Logic*, in: *LICS*, 2007.

[6] Dodds, M., X. Feng, M. Parkinson and V. Vafeiadis, *Deny-guarantee reasoning*, in: *ESOP*, 736 (2009), pp. 363–377.

[7] Fiore, M., E. Moggi and D. Sangiorgi, *A fully-abstract model for the pi-calculus*, in: *LICS*, December (1996).

[8] Hennessy, M., *A fully abstract denotational semantics for the pi-calculus*, TCS **278** (2002), pp. 53–89.

[9] Hoare, T. and P. O'Hearn, *Separation Logic Semantics for Communicating Processes*, Electronic Notes in Theoretical Computer Science (ENTCS) (2008).

[10] Honda, K., V. T. Vasconcelos and M. Kubo, *Language Primitives and Type Discipline for Structured Communication-Based Programming*, in: *ESOP*, 1998, pp. 122–138.

[11] Kobayashi, N., B. Pierce and D. Turner, *Linearity and the pi-calculus*, ACM Transactions on Programming Languages and Systems (TOPLAS) **21** (1999), pp. 914–947.

[12] Milner, R., J. Parrow and D. Walker, *A calculus of mobile processes, parts I and II*, Information and computation **100** (1992).

[13] Milner, R. and D. Sangiorgi, *Barbed bisimulation*, in: *Automata, Languages and Programming*, Lecture Notes in Computer Science **623**, 1992 pp. 685–695.

[14] O'Hearn, P., *Resources, concurrency, and local reasoning*, TCS **375** (2007), pp. 271–307.

[15] O'Hearn, P., J. Reynolds and H. Yang, *Local Reasoning about Programs that Alter Data Structures*, in: *Computer Science Logic*, 2001.

[16] Parkinson, M., R. Bornat and C. Calcagno, *Variables as Resource in Hoare Logics*, in: *LICS* (2006).

[17] Reynolds, J., *Separation logic: a logic for shared mutable data structures*, in: *LICS*, 2002.

[18] Roscoe, A. W. and G. Barrett, *Unbounded Non-determinism in CSP*, in: *MFPS*, 1989.

[19] Sangiorgi, D. and D. Walker, "The pi-calculus: a Theory of Mobile Processes," Cambridge University Press, 2001.

[20] Stark, I., *A fully abstract domain model for the pi-calculus*, in: *LICS* (1996), pp. 36–42.

[21] Van Glabbeek, R., *The linear time-branching time spectrum*, CONCUR'90 Theories of Concurrency: Unification and Extension (1990), pp. 278–297.