DRAFT

A separation logic for the π-calculus

Aaron Turon Mitchell Wand

Northeastern University

{turon, wand}@ccs.neu.edu

Abstract

Reasoning about concurrent processes requires distinguishing communication from interference, and is especially difficult when the means of interaction change over time. We present a new logic for the π-calculus that combines temporal and separation logic, and treats channels as resources that can be gained and lost by processes. The resource model provides a lightweight way to constrain interference. By interpreting process terms as formulas, our logic directly supports compositional reasoning.

1. Introduction

Concurrency is notoriously hard. What makes it hard is what makes it powerful: the ability of concurrent processes to influence each other. Whether this influence is interference or communication is a question of perspective, but the distinction matters: few processes behave correctly under arbitrary influence by their environment.

How do we specify and reason about the influence processes that have on each other? There is a huge literature addressing this question, in the settings of both shared-state and message-passing concurrency, making use of both logical [AL95, Jon83] and type-theoretic [IK01, PS93] tools. Compositionality is a central goal.

Compositional reasoning requires environmental assumptions that constrain the influence of one subsystem on another. Processes influence each other through shared resources, *e.g.*, a shared memory, or shared message channels. When these resources are fixed, environmental assumptions are usually fixed as well. When these resources change dynamically, environmental assumptions must evolve to match. Techniques for reasoning about fixed resources are relatively well-understood [Roe01], but dynamic resources remain a focus of current research.

For shared-state concurrency with dynamic allocation and pointers, concurrent separation logic (CSL) has recently gained a foothold [O'H07]. CSL is oriented around resource ownership, where the resources in question are shared heap cells. The logic provides a concise and elegant means of describing dynamic resources, and tracking them as they flow between processes. Its ownership paradigm provides a simple kind of environmental assumption: the environment of a process is assumed not to use resources owned by the process. Because it is a Hoare logic, CSL deals with input-output behavior of programs.

This paper presents a separation logic for the π-calculus, a core calculus for message-passing concurrency in which channels can be allocated and communicated. Broadly, our aim is to adapt ideas from CSL to the π-calculus. The π-calculus opens the door to non-terminating, reactive programs, which interact over time rather than computing a single result. Thus we move to a temporal logic capable of expressing behavior over time. Integrating separation and temporal logic while maintaining compositional reasoning poses a significant challenge, and is a central goal of this paper.

In the π-calculus, interference can arise through competition for communication, for example when multiple processes attempt to send messages over the same channel. Consider the process $P \triangleq \overline{c}(3).\mathbf{end}|c(n).P'$, consisting of a process sending 3 along channel c in parallel with a process receiving a number n along c. If the communication between the two processes succeeds, P behaves like $P'\{3/n\}$. However, the environment in which the P executes can interfere with the intended communication: if we place P in parallel with the process $\overline{c}(2).\mathbf{end}$, we can no longer say for sure what value n will take.

The π-calculus has a mechanism for controlling such interference: private channels. A channel private to a process cannot be used by its environment. But it is not always possible to make channels private, especially when a channel represents a resource shared between several processes. What we seek is an analog to privacy within a temporal logic, allowing us to make assertions such as "right now, the environment will not interfere along channel c." The thesis of this paper is that, by treating channels as resources, we can exclude the possibility of interference for public channels.

Contributions and outline

- We show how to view the π-calculus in terms of resource ownership and transfer (Section 2). In particular, permission to send or receive on a channel is a resource that can (conceptually) be transferred during communication. This perspective on permission is not new, but in the past it was usually restricted to point-to-point communication: because the send and receive permissions were owned by only one process, communication at any point in time was only allowed between one sender and one receiver [TA08, HO08].

  In contrast, we develop a novel interpretation of fractional permissions [Boy03], allowing multiple processes to share a given resource. Thus, competition over communicating on a channel is allowed, but it can still be controlled by tracking permissions.

- Using our resource interpretation, we define a novel operational semantics for a variant of the π-calculus (Section 3). The distinctive feature of the semantics is its lack of scope extrusion, which is the mechanism most often used to model the channel mobility of the π-calculus. We instead use our resource model to constrain possible communication.

  To incorporate separation logic we explicitly model *faults*, which occur when a process attempts to use resources it does not own. We allow a process to be annotated with local expectations about the behavior of its environment, *e.g.*, if the environment sends a number n along channel c, then n is even. These assumptions are checked when the process is composed with an actual environment, and a violation causes a fault.

- We propose a temporal separation logic (Section 4). It is not a Hennessy-Milner logic—it does not express predicates on labelled transition systems. Instead, the logic is interpreted over complete traces, and its observables are similar to those of the failures/divergences model over infinite traces [RB90]. The temporal and separation aspects of the logic interact through strongest postcondition reasoning. Faulting plays an important role in our treatment of refinement, and in particular leads to our logic being intuitionistic.

  We focus primarily on the syntax and model theory of the logic, giving only a cursory treatment of its proof theory.

- We include parallel composition as a connective in our logic, which provides the decisive step toward compositional reasoning. Its inclusion allows us to view processes as temporal formulas. Thus, the formulas of the logic cover a spectrum from concrete (pure processes) to abstract (pure specifications). Because the logic is defined compositionally, it gives a denotational semantics for our variant of the π-calculus (Section 5).

  Because processes are formulas, we can express refinement through implication: if $P \Rightarrow \varphi$ is a valid formula, then the process P refines the specification φ. The logic supports compositional reasoning in two directions: by reasoning about subformulas, and by reasoning about chains of implications. For example, if $P \Rightarrow \varphi$ and $Q \Rightarrow \psi$, we deduce that $P|Q \Rightarrow \varphi|\psi$. Moreover, if $\varphi|\psi \Rightarrow \theta$, we conclude that $P|Q \Rightarrow \theta$.

- We give examples in Section 6 illustrating resource-based, process-algebraic, and temporal reasoning.

We are not the first to propose the application of separation logic to the π-calculus. Hoare and O'Hearn took the first step in a paper that provided key inspiration for our work [HO08]. They developed a semantics for the π-calculus based on separation logic, but left open several questions, including

- how to allow more than point-to-point communication,
- how to incorporate liveness properties, and
- how to incorporate faulting and distinguish it from other forms of failure, such as deadlock.

We answer all of these questions, in service of our goal: combining temporal and separation logic for compositional, resource-based reasoning. We discuss Hoare and O'Hearn's paper in more detail, along with other related work, in Section 7.

2. A resource analysis of the π-calculus

We begin with a treatment of the π-calculus based on resources. This section develops our resource model and process language, and lays the technical foundation for the semantics and logic.

2.1 Resources

The resources we wish to track are permissions for sending and receiving along a channel, modeled as follows:

Resource model

$$\mathsf{c},\mathsf{d} \in \mathsf{Chan} \qquad \mathsf{f} \in \mathsf{Perm} \triangleq [0,1] \qquad \delta \in \mathsf{Dir} \triangleq \{!,?\}$$
$$\sigma \in \Sigma \triangleq \mathsf{Chan} \times \mathsf{Dir} \to \mathsf{Perm}$$

We assume an infinite collection of channel names, Chan. A resource σ maps each channel-direction pair to a permission quantity; $\mathrm{dom}(\sigma)$ is the set of channel-direction pairs mapped to a nonzero quantity. We allow permissions to be *fractionally* owned [Boy03]. The idea is simple. Suppose a process owns resources σ.

- If $\sigma(\mathsf{c}!) = 0$, the process is not permitted to send on channel c.
- If $\sigma(\mathsf{c}!) = 1$, the process is permitted to send on channel c, and is the only process that may do so.
- If $0 < \sigma(\mathsf{c}!) < 1$, the process is permitted to send on channel c, but other processes owning fractions of $\mathsf{c}!$ may do so as well.

and similarly for $\mathsf{c}?$. The motivation for tracking permissions quantitatively is *addition*: when reasoning about multiple processes, we add their resources to determine what they collectively own. Consider the processes $P \triangleq \bar{\mathsf{c}}(3).P'$ and $Q \triangleq \mathsf{c}(x).Q'$. The process P first attempts to send the value 3 on channel c, then continues on as P'. Likewise Q attempts to receive a value on c and continue as Q'. If we place them in parallel with another process R, yielding $P|Q|R$, what happens? In general, the outcome depends on R, which could interfere with the attempted communication.

We can use resources to reason about the situation. If P and Q are each assumed to own resources $\frac{1}{2}\mathsf{c}!$ and $\frac{1}{2}\mathsf{c}?$, then $P|Q$ owns $1\mathsf{c}!$ and $1\mathsf{c}?$—and therefore, R is assumed not to communicate along c. In the logic we present later, this will allow us to conclude that $P|Q|R \Rightarrow P'|Q'\{3/x\}|R$: the communication between P and Q succeeds even in the presence of R, as long as the ownership assumptions are satisfied.

The basic operations on resource are given below. Note that $\oplus$ and $\ominus$ are *partial*: if $\sigma_1(\mathsf{c}\delta) + \sigma_2(\mathsf{c}\delta) \notin [0,1]$ for any $\mathsf{c}\delta$, then $\sigma_1 \oplus \sigma_2$ is undefined, and similarly for $\ominus$.

Resource operations $\oplus, \ominus : \Sigma \times \Sigma \rightharpoonup \Sigma$

$$\begin{array}{rcl}
(\sigma_1 \oplus \sigma_2)(\mathsf{c}\delta) & \triangleq & \sigma_1(\mathsf{c}\delta) + \sigma_2(\mathsf{c}\delta) \\
(\sigma_1 \ominus \sigma_2)(\mathsf{c}\delta) & \triangleq & \sigma_1(\mathsf{c}\delta) - \sigma_2(\mathsf{c}\delta) \\
\sigma_1 \leq \sigma_2 & \Leftrightarrow & \forall \mathsf{c}\delta \,.\, \sigma_1(\mathsf{c}\delta) \leq \sigma_2(\mathsf{c}\delta) \\
\sigma_1 \# \sigma_2 & \Leftrightarrow & \sigma_1 \oplus \sigma_2 \text{ defined}
\end{array}$$

We work with resources through a language of *resource predicates* p, which includes *separating conjunction* $p * q$ and *septraction* $p \circledast\!- q$ from separation logic. Resource predicates denote subsets $\mathsf{p} \subseteq \Sigma$, and we use them extensively in the syntax, semantics, and logic for processes. We use italic, serifed font for metavariables ranging over syntactic expressions, and use the same letter sans serif to range over the domain of denotations.

Channel and fraction expressions, resource predicates

$$\begin{array}{rcl}
c & ::= & \mathsf{c} \mid x \\
f & ::= & \mathsf{f} \mid \iota \mid f + f \mid \cdots \\
p & ::= & \mathsf{emp} \mid fc\delta \mid p * q \mid p \circledast\!- q \mid \mathsf{tt} \mid p \vee q \\
& \mid & \neg p \mid \exists x.p \mid \exists \iota.p \mid c_1 = c_2 \mid f_1 \leq f_2 \mid \delta_1 = \delta_2
\end{array}$$

In defining resource predicates, we use both channel and fraction expressions. Channel expressions are either channel constants or channel variables. For fraction expressions, we do not specify a complete syntax, but we do assume that fraction constants $\mathsf{f} \in [0,1]$, variables ι, and addition are included. The operators $*$ and $\circledast\!-$ bind tighter than the other operators. We treat tt, ff, $\wedge$, $\Rightarrow$, and $\forall$ as derived constructs in the usual way.

Figure 1 gives the semantics of resource predicates: σ satisfies p in environment ρ if $\sigma \models_\rho p$, where ρ maps channel variables to channels and fraction variables to fractions. We drop ρ when it is empty. We write $[\![p]\!]^\rho$ for the subset $\{\sigma : \sigma \models_\rho p\}$ of Σ. We take as given a map $[\![-]\!]^\rho$ from fraction expressions to fractions and from channel expressions to channels.

$\sigma \models_\rho \mathrm{emp}$	iff	$\mathrm{dom}(\sigma) = \emptyset$		$\sigma \models_\rho \neg p$	iff	$\sigma \not\models_\rho p$
$\sigma \models_\rho fc\delta$	iff	$\mathrm{dom}(\sigma) = \{[\![c]\!]^\rho\},\ \sigma([\![c]\!]^\rho\,\delta) = [\![f]\!]^\rho$		$\sigma \models_\rho \exists x.p$	iff	$\sigma \models_{\rho[x\mapsto \mathsf{c}]} p$ for some c
$\sigma \models_\rho p * q$	iff	$\sigma_1 \models_\rho p,\ \sigma_2 \models_\rho q$ for some $\sigma_1 \oplus \sigma_2 = \sigma$		$\sigma \models_\rho \exists\iota.p$	iff	$\sigma \models_{\rho[\iota\mapsto \mathsf{f}]} p$ for some f
$\sigma \models_\rho p \circledast\!\!- q$	iff	$\sigma = \sigma_1 \ominus \sigma_2$ for some $\sigma_1 \models_\rho p,\ \sigma_2 \models_\rho q$		$\sigma \models_\rho c_1 = c_2$	iff	$[\![c_1]\!]^\rho = [\![c_2]\!]^\rho$
$\sigma \models_\rho \mathrm{tt}$	iff	always		$\sigma \models_\rho f_1 \le f_2$	iff	$[\![f_1]\!]^\rho \le [\![f_2]\!]^\rho$
$\sigma \models_\rho p \vee q$	iff	$\sigma \models_\rho p$ or $\sigma \models_\rho q$		$\sigma \models_\rho \delta_1 = \delta_2$	iff	$\delta_1 = \delta_2$

Figure 1. Resource predicate semantics

The predicates emp and $fc\delta$ both uniquely determine a resource σ. For emp, the resource must contain no permissions at all, while for $fc\delta$, it must contain exactly the permission, at quantity f, for communication along $c\delta$. On the other hand, a resource satisfies $p * q$ if it can be decomposed by $\oplus$ into resources satisfying p and q respectively. Finally, the predicate $p \circledast\!\!- q$ represents subtraction of resources: it is satisfied by σ if there are resources σ_1 satisfying p and σ_2 satisfying q such that $\sigma = \sigma_1 \ominus \sigma_2$.[1]

As a simple illustration, consider the difference between the predicates 1c! and 1c! $*$ tt. Resources satisfying the former must have domain exactly $\{\mathsf{c}!\}$, while resources satisfying the latter must have domain at least $\{\mathsf{c}!\}$. In fact, we have $[\![1\mathsf{c}!]\!] \subseteq [\![1\mathsf{c}! * \mathrm{tt}]\!]$.

A predicate p is:

- *valid* if $[\![p]\!]^\rho = \Sigma$ for all closing ρ,
- *pure* if, for each closing ρ, either $[\![p]\!]^\rho = \Sigma$ or $[\![p]\!]^\rho = \emptyset$,
- *intuitionistic* if $\sigma \in [\![p]\!]^\rho$ and $\sigma \le \sigma'$ implies $\sigma' \in [\![p]\!]^\rho$ [Rey02].

Valid predicates capture tautologies. For example, the predicate $\iota_1 x! * \iota_2 x! \ \Leftrightarrow\ (\iota_1 + \iota_2)x!$, which relates separating conjunction and fraction addition, is valid. Another useful example stems from the partiality of resource addition: $1x? * 1y? \ \Rightarrow\ x \ne y$ is valid because it is impossible to add resources resulting in a permission quantity larger than 1. In this way, resources can be used to draw conclusions about aliasing.

Pure predicates are primarily used to make assertions about fraction or permission variables, without asserting anything about resources.

Intuitionistic predicates serve as lower bounds on resources. For example, 1c! $*$ tt is intuitionistic, while 1c! is not.

2.2 Processes

We study a variant of the π-calculus with explicit resource annotations. These annotations describe the flow of resources during communication and split resources for parallel composition. We align resource transfers with standard communication, so that only resources associated with the channel being sent or received can be transferred. This is a choice of convenience, not necessity. In principle, arbitrary resources can be exchanged during synchronization.

Before defining the syntax of processes, we illustrate the main ideas with the following process term:

$$\begin{aligned}&\textbf{fix}\ X.z(n).z(\iota x! : \iota = 1 \wedge x = z).\overline{c}(n).\\ &\quad\big(X \ {}_{1z?*1c!*\mathrm{tt}}\|_{1z!*\mathrm{tt}}\ \overline{z}(n+1).\overline{z}(1z!).\textbf{end}\big)\end{aligned}$$

Informally, the behavior of this process is to receive some number n along z, and then send $n, n+1, \ldots$ along c—but this behavior is only guaranteed if the environment of the process does not interfere along the channel z. We will capture the noninterference assumption through resource predicates.

[1] Septraction is dual to separating implication: $q \circledast\!\!- p \ \Leftrightarrow\ \neg(p -\!\!* \neg q)$ [VP07].

Suppose the process begins execution while owning resources satisfying the predicate $1z? * 1c! * \mathrm{tt}$. It then

- begins a recursion, binding X,
- receives a number n on channel z,
- receives resources $\iota x!$, with the assumption that $\iota = 1$, $x = z$. The variables ι and x are bound in the rest of the process, but their values are completely determined by the assumption made about them. Thus, the process is receiving x not to learn a new channel, but to gain additional resources for a channel it already knows.
- then, the process sends the number n on channel c.

After this step, the process owns resources that satisfy

$$(\iota = 1 \wedge x = z \wedge \iota x!) * 1z? * 1c! * \mathrm{tt}$$

and hence satisfy $1z! * 1z? * 1c! * \mathrm{tt}$.

Next, the process breaks into two subprocesses, which are given resources satisfying $1z? * 1c! * \mathrm{tt}$ and $1z! * \mathrm{tt}$ respectively:

- The first subprocess simply behaves again like X. Notice that, when the new version of X is invoked, it owns resources $1z? * 1c! * \mathrm{tt}$—the same resources held when X was bound. The predicate $1z? * 1c! * \mathrm{tt}$ can be seen as a *loop invariant* for this example.
- The second subprocess begins by sending $n+1$ along z, which does not alter its resource ownership. Afterwards, however, the process sends the $1z!$ permission along z, thereby relinquishing ownership of z. The process is left with resources that satisfy the predicate tt, about which nothing can be assumed.

As the example illustrates, we can use resource annotations to calculate resource ownership at each point in the execution of a process. Why is such calculation useful? Just before forking into two subprocesses, the example process owns *all* resources associated with z. This is useful information: it captures our assumption that the environment will not interfere with internal communication along z even if the environment knows z. In Section 6, we will see how to deduce that the example process, in any well-behaved environment, sends an increasing sequence of numbers on c.

We now define our process syntax, as follows:

Process syntax

$$\begin{array}{rcl}\pi & ::= & \overline{c}(fc\delta)\\ & | & c(\iota x\delta : p) \quad \text{with } p \text{ pure}\\ P & ::= & \textbf{end} \ \mid\ \textbf{new}\ x.P \ \mid\ \sum \pi_i.P_i \ \mid\ P \vee Q \ \mid\ \textbf{fix}\ X.P \ \mid\ X\\ & | & P\ {}_p\|_q\ Q \quad \text{with } p, q \text{ intuitionistic}\end{array}$$

Prefixes π represent the two directions of basic interaction in the π-calculus: sending and receiving. In both cases what is being communicated is not just a channel, but also some *resources* associated with a channel. In examples, we also allow numbers to be communicated. When sending ($\overline{c}(fc\delta)$), the resources transferred are determined by the predicate $fc\delta$. When receiving ($c(\iota x\delta : p)$), the resources transferred satisfy the predicate $\iota x\delta \wedge p$; the pure predicate p constrains ι and x.

The process layer P is largely standard. The inert process **end** represents successful termination. The process **new** $x.P$ allocates a new channel and binds it to the channel variable x in P. External choice $\sum \pi_i.P_i$ offers each communication π_i to the environment; only one communication takes place, after which the process behaves as the corresponding P_i. In contrast, internal choice $P \vee Q$ allows the process to behave as either P or Q, arbitrarily. Recursive processes are written **fix** $X.P$, which binds the process variable X in P. Finally, the parallel composition of two processes $P\ {}_p\|_q\ Q$ is annotated with predicates p, q describing how resources are split between the two subprocesses. The predicates must be intuitionistic because they describe lower bounds on the expected resources.

Ultimately, annotations are meant to describe, rather than determine, the behavior of processes, but we have embedded annotations directly into the syntax of processes—why? The annotations are needed to give the semantics of *open* processes, where the environment of the process is unknown. Resource annotations are not necessary for a closed system (*e.g.* a *testing scenario* [NH84]), where all interacting processes are known. Thus, although resources are explicitly represented in our operational semantics, they are a logical device and need no representation in an implementation.

2.3 Actions

At heart of our semantic model is a distinction between two kinds of failure to perform an action: *noisy* and *silent*. Failure is connected to resource ownership. If an action is not *permitted* given the owned resources, attempting to perform it causes a noisy failure. If an action is not *possible* given the owned resources, it silently fails to be performed.

Take the process $\overline{c}(3)$.**end**, which attempts to send the number 3 over c. If the process owns resources emp, the attempt to send fails noisily, resulting in a fault. If the process owns $1c! * 1c?$, the attempt to send fails silently, resulting in deadlock. The silent failure arises because the process owns all receiving permissions for c, hence its environment owns none. If the environment attempted to receive along c, it would noisily fail.

Noisy and silent failure are dual in other respects as well. In the logic of Section 4, an action that noisily fails satisfies only the trivial specification tt, while silently failing actions satisfy every specification. It is important that noisy failures satisfy no interesting specifications, because noisy failure represents the breaking of an assumption. Noisy failure arises through owning too few resources, and silent failure too many.

Our semantics, both denotational and operational, is based on Brookes's *action traces* [Bro02]. In action trace semantics, the behavior of a process is given as a set of traces, each of which is a (possibly infinite) sequence of actions. Actions play two roles: they describe the observable steps a process takes, and they act as resource transformers. Actions α for our calculus are:

Action syntax

$$\begin{array}{rcl} \alpha & ::= & \mathcal{C}\langle \mathsf{c}\delta\mathsf{m}\rangle \mid \mathcal{A}\langle \mathsf{c?p}\rangle \mid \mathcal{B}\langle\Delta\rangle \mid \mathcal{N}\langle \mathsf{c}\rangle \\ \Delta & \in & \mathbb{P}_{\mathsf{fin}}(\mathsf{Chan} \times \mathsf{Dir}) \end{array}$$

The metavariable m, for *message*, ranges over triples $\mathsf{f c}\delta$. We view messages as resources mapping $\mathsf{c}\delta$ to f and every other channel-direction pair to 0. We drop set brackets for $\mathcal{B}\langle\Delta\rangle$, writing $\mathcal{B}\langle \mathsf{c!}, \mathsf{d?}, \dots\rangle$ rather than $\mathcal{B}\langle\{\mathsf{c!}, \mathsf{d?}, \dots\}\rangle$.

The action $\mathcal{C}\langle \mathsf{c}\delta\mathsf{m}\rangle$ represents a *communication* of m along channel c in direction δ. An *assumption* action $\mathcal{A}\langle \mathsf{c?p}\rangle$ records a communication that fails to satisfy the predicate p given by an annotation in a receive prefix. *Blocking* actions $\mathcal{B}\langle\Delta\rangle$ record the potential for communication. They arise when a process cannot make progress until it communicates with its environment; Δ gives the finite set of channel-direction pairs along which the process is offering to communicate. A *new channel* action $\mathcal{N}\langle \mathsf{c}\rangle$ records the allocation of channel c.

We view actions as resource transformers. A *resource transformer* is a function of type $\Sigma \to \Sigma_\perp^\top$. The constants $\top$ and $\perp$ represent noisy and silent failure, respectively. Thus, an action in a given resource context can either transform the resources, noisily fail, or silently fail. We place a partial order $\sqsubseteq$ on $\Sigma_\perp^\top$: for all σ, $\perp \sqsubseteq \sigma \sqsubseteq \top$. The order is lifted pointwise to resource transformers.

We have the following basic resource transformers:

Basic resource transformers

$$\mathsf{add}(\sigma_0)(\sigma) \triangleq \begin{cases} \sigma \oplus \sigma_0 & \text{if defined} \\ \perp & \text{otherwise} \end{cases} \qquad \mathsf{rem}(\sigma_0)(\sigma) \triangleq \begin{cases} \sigma \ominus \sigma_0 & \text{if defined} \\ \top & \text{otherwise} \end{cases}$$

$$\mathsf{per}(\mathsf{c}\delta)(\sigma) \triangleq \begin{cases} \top & \sigma(\mathsf{c}\delta) = 0 \\ \sigma & \sigma(\mathsf{c}\delta) > 0 \end{cases} \qquad \mathsf{pos}(\mathsf{c}\delta)(\sigma) \triangleq \begin{cases} \sigma & \sigma(\mathsf{c}\overline{\delta}) < 1 \\ \perp & \sigma(\mathsf{c}\overline{\delta}) = 1 \end{cases}$$

$$(g; h)(\sigma) \triangleq \begin{cases} h(g(\sigma)) & g(\sigma) \in \Sigma \\ g(\sigma) & g(\sigma) = \top \text{ or } g(\sigma) = \perp \end{cases}$$

The combinator $\mathsf{add}(\sigma_0)$ attempts to add σ_0 to the given resource, failing silently when this is impossible; likewise $\mathsf{rem}(\sigma_0)$ removes σ_0 or fails noisily. The combinator $\mathsf{per}(\mathsf{c}\delta)$ checks whether it is *permitted* to communicate in direction δ on channel c, while $\mathsf{pos}(\mathsf{c}\delta)$ checks whether it is *possible* (*i.e.*, whether the environment can communicate in the opposite direction). Note: $\overline{!} = ?$ and $\overline{?} = !$. The sequential composition of two resource transformers inherits the failure behavior of the first transformer.

Using the combinators, we define:

Actions as resource transformers $\quad (\!|-|\!) : \mathsf{Act} \to \Sigma \to \Sigma_\perp^\top$

$$\begin{array}{rcl} (\!|\mathcal{N}\langle \mathsf{c}\rangle|\!) & \triangleq & \mathsf{add}(1\mathsf{c}! \oplus 1\mathsf{c}?) \\ (\!|\mathcal{C}\langle \mathsf{c!m}\rangle|\!) & \triangleq & \mathsf{per}(\mathsf{c}!); \mathsf{pos}(\mathsf{c}!); \mathsf{rem}(\mathsf{m}) \\ (\!|\mathcal{C}\langle \mathsf{c?m}\rangle|\!) & \triangleq & \mathsf{per}(\mathsf{c}?); \mathsf{pos}(\mathsf{c}?); \mathsf{add}(\mathsf{m}) \\ (\!|\mathcal{A}\langle \mathsf{c?p}\rangle|\!) & \triangleq & \mathsf{per}(\mathsf{c}?); \mathsf{pos}(\mathsf{c}?) \\ (\!|\mathcal{B}\langle\emptyset\rangle|\!) & \triangleq & \mathsf{id} \\ (\!|\mathcal{B}\langle\Delta\rangle|\!) & \triangleq & \bigsqcup_{\mathsf{c}\delta\in\Delta} \mathsf{per}(\mathsf{c}\delta); \bigsqcap_{\mathsf{c}\delta\in\Delta} \mathsf{pos}(\mathsf{c}\delta) \qquad (\Delta \neq \emptyset) \end{array}$$

The order in which combinators are used is important. For example, the action $\mathcal{C}\langle \mathsf{c!m}\rangle$ fails noisily if sending on c is not permitted, otherwise fails silently if sending on c is not possible, and only after both these checks attempts to remove the resources m. Likewise the blocking action on Δ fails noisily if any $\mathsf{c}\delta \in \Delta$ is not permitted, and only then fails silently if some $\mathsf{c}\delta \in \Delta$ is not possible. For blocking actions we use the fact that resource transformers, ordered by $\sqsubseteq$, form a complete lattice.

3. Operational semantics

The operational semantics for our process language is built around a labeled transition system (LTS) in the tradition of process calculus. It breaks from tradition, however, by using resources rather than scoping to constrain communication.

$$\frac{\llparenthesis \mathcal{N}\langle \mathsf{c}\rangle \rrparenthesis\sigma \neq \top, \bot}{\textbf{new}\ x.P, \sigma \xrightarrow{\mathcal{N}\langle \mathsf{c}\rangle} P\{\mathsf{c}/x\}, \llparenthesis \mathcal{N}\langle \mathsf{c}\rangle \rrparenthesis\sigma}$$

$$\frac{\pi_j.P_j \xrightarrow{\alpha} Q \quad \llparenthesis\alpha\rrparenthesis\sigma \neq \top, \bot}{\sum \pi_i.P_i, \sigma \xrightarrow{\alpha} Q, \llparenthesis\alpha\rrparenthesis\sigma} \qquad \frac{\pi_j.P_j \xrightarrow{\alpha} Q \quad \llparenthesis\alpha\rrparenthesis\sigma = \top}{\sum \pi_i.P_i, \sigma \xrightarrow{\top} \bullet}$$

$$\mathrm{L}\frac{}{P \vee Q, \sigma \xrightarrow{\tau} P, \sigma} \qquad \frac{}{\textbf{fix}\ X.P, \sigma \xrightarrow{\tau} P\{\textbf{fix}\ X.P/X\}, \sigma}$$

$$\frac{\sigma_1 \in [\![p]\!] \quad \sigma_2 \in [\![q]\!]}{P\ {}_p\|_q\ Q, \sigma_1 \oplus \sigma_2 \xrightarrow{\tau} (P, \sigma_1)|(Q, \sigma_2)} \qquad \frac{\sigma \notin [\![p * q]\!]}{P\ {}_p\|_q\ Q, \sigma \xrightarrow{\top} \bullet}$$

$$\mathrm{L}\frac{\kappa_1 \xrightarrow{\alpha} \kappa_1' \quad \llparenthesis\alpha\rrparenthesis(\mathrm{res}(\kappa_1|\kappa_2)) \neq \top, \bot}{\kappa_1|\kappa_2 \xrightarrow{\alpha} \kappa_1'|\kappa_2}$$

$$\mathrm{L}\frac{\kappa_1 \xrightarrow{\tau} \kappa_1'}{\kappa_1|\kappa_2 \xrightarrow{\tau} \kappa_1'|\kappa_2} \qquad \frac{\kappa_1 \xrightarrow{\alpha} \kappa_1' \quad \kappa_2 \xrightarrow{\overline{\alpha}} \kappa_2'}{\kappa_1|\kappa_2 \xrightarrow{\tau} \kappa_1'|\kappa_2'}$$

$$\mathrm{L}\frac{\kappa_1 \xrightarrow{\top} \bullet}{\kappa_1|\kappa_2 \xrightarrow{\top} \bullet} \qquad \mathrm{L}\frac{\kappa_1 \xrightarrow{\mathcal{C}\langle \mathsf{c!m}\rangle} \kappa_1' \quad \kappa_2 \xrightarrow{\mathcal{A}\langle \mathsf{c?p}\rangle} \kappa_2' \quad \mathsf{m} \notin \mathsf{p}}{\kappa_1|\kappa_2 \xrightarrow{\top} \bullet}$$

Figure 2. Process semantics: labeled transition system (L designates Left versions of symmetric rules)

Before giving the LTS, we define an auxiliary relation below, whose role is to generate the actions corresponding to prefixes. Send prefixes are straightforward. For receive prefixes, however, there are several possible outcomes. First, for each message satisfying the resource predicate, a corresponding communication action is generated, with the contents of the message substituted into the process body. In addition, each receive prefix generates an assumption action, recording its resource predicate. After an assumption action, no further actions are observed of a process.

Process semantics: prefix steps

$$\begin{array}{lcll}
\overline{\mathsf{c}}(fd\delta).P & \xrightarrow{\mathcal{C}\langle \mathsf{c!fd}\delta\rangle} & P & \mathsf{f} = [\![f]\!],\ \mathsf{d} = [\![d]\!] \\
\mathsf{c}(\iota x\delta : p).P & \xrightarrow{\mathcal{C}\langle \mathsf{c?fd}\delta\rangle} & P\{\mathsf{f}/\iota, \mathsf{d}/x\} & \mathsf{fd}\delta \in [\![\exists \iota x.\iota x\delta \wedge p]\!] \\
\mathsf{c}(\iota x\delta : p).P & \xrightarrow{\mathcal{A}\langle \mathsf{c?q}\rangle} & \textbf{end} & \mathsf{q} = [\![\exists \iota x.\iota x\delta \wedge p]\!]
\end{array}$$

The LTS in Figure 2 is defined not on processes, but on *configurations* κ:

$$\kappa \ ::= \ P, \sigma \ | \ \kappa|\kappa \ | \ \bullet$$

Configurations track resources belonging to each process when multiple processes are running in parallel. The configuration $\bullet$ represents *unsuccessful* termination, caused by faulting. The resources owned by a configuration are given by the function res:

$$\mathrm{res}(P, \sigma) = \sigma \quad \mathrm{res}(\kappa_1|\kappa_2) = \mathrm{res}(\kappa_1) \oplus \mathrm{res}(\kappa_2) \quad \mathrm{res}(\bullet) = \emptyset$$

The LTS ensures that res is defined on any reachable configuration.

Labels are drawn from the syntax $\ell ::= \alpha \ | \ \top \ | \ \tau$. Here, $\top$ represents a faulting outcome and τ, as usual, represents an internal, unobservable step of a configuration. Each rule with an L ("Left") adjoined has an elided, symmetric counterpart. Note that $\overline{\alpha} = \mathcal{C}\langle \mathsf{c}\overline{\delta}\mathsf{m}\rangle$ if $\alpha = \mathcal{C}\langle \mathsf{c}\delta\mathsf{m}\rangle$, and is undefined otherwise.

The rules are fairly straightforward, but a few deserve comment:

- Allocation provides an example of silent failure: every channel c could potentially appear in the label $\mathcal{N}\langle \mathsf{c}\rangle$, but if $\sigma(\mathsf{c!}) > 0$ or $\sigma(\mathsf{c?}) > 0$ then $\llparenthesis \mathcal{N}\langle \mathsf{c}\rangle \rrparenthesis\sigma = \bot$, preventing the rule from firing.
- Likewise, the actions generated by prefixes are checked against the resources held by the configuration, and may silently cause the rule to fail to apply, or noisily cause a $\top$ label to appear. For example, if $\sigma(\mathsf{c!}) = \sigma(\mathsf{c?}) = 1$, then $\overline{\mathsf{c}}(\mathsf{m}).P, \sigma$ silently fails to take a $\mathcal{C}\langle \mathsf{c!m}\rangle$ step, because the environment cannot offer to receive a message on c.
- A parallel composition ${}_p\|_q$ noisily fails if its resources cannot be split into parts satisfying p and q respectively.
- When $\kappa_1 \xrightarrow{\alpha} \kappa_1'$ then $\kappa_1|\kappa_2 \xrightarrow{\alpha} \kappa_1'|\kappa_2$, but only if the additional resources provided by κ_2 do not rule out α.[2] Note that, since α did not fail noisily with only the resources of κ_1, adding the resources of κ_2 cannot cause α to fail noisily. But it can cause a silent failure, thereby preventing, for example, κ_1 from allocating a channel that κ_2 owns.
- The final rule demonstrates the purpose of assumption actions. If κ_1 is willing to send on c and κ_2 to receive, but the message being sent breaks the assumption of κ_2, the result is a noisy failure. By making assumptions observable, we are able to detect when they are violated.

3.1 Observation

We have given a semantics for configurations, but there are configurations with different transition graphs whose behavior we wish to equate. Thus, we define an *observation* function $\mathcal{O}[\![-]\!]$ from configurations to sets of traces, which extracts from the LTS the observable behavior we wish to reason about with our logic. Observational equivalence for us is much weaker than bisimilarity, but the inclusion of blocking actions allows some of the braching structure of the LTS to be observed. Our observables are similar to those of the failures/divergences model over infinite traces [RB90].

Let $\mathcal{A}$ be the set of all communication actions, $\mathcal{B}$ all blocking actions, *etc*. The set of all traces is defined as

$$\mathsf{Trace} \triangleq (\mathcal{C} \cup \mathcal{N})^\omega \cup (\mathcal{C} \cup \mathcal{N})^*((\mathcal{A} \cup \mathcal{B})^?)^\top$$

That is, a trace is either an infinite sequence of communications and allocations, or else a finite sequence possibly followed by $\mathcal{A}$, $\mathcal{B}$, or $\top$. We write ϵ for the empty trace.

Before defining $\mathcal{O}[\![-]\!]$, we give the *intended* output for a few example configurations. First, $\mathcal{O}[\![\textbf{end}, \sigma]\!] = \{\epsilon\}$ for any resource σ: the empty trace ϵ represents successful termination.

Next, consider the behavior of a process that attempts to send along c, as we vary its resources:

$$\begin{aligned}
\mathcal{O}[\![\overline{\mathsf{c}}(1\mathsf{c!}).\textbf{end}, 1\mathsf{c!}]\!] &= \{\mathcal{B}\langle \mathsf{c!}\rangle, \mathcal{C}\langle \mathsf{c!1c!}\rangle\} \\
\mathcal{O}[\![\overline{\mathsf{c}}(1\mathsf{c!}).\textbf{end}, 1\mathsf{c!} \oplus 1\mathsf{c?}]\!] &= \{\mathcal{B}\langle\rangle\} \\
\mathcal{O}[\![\overline{\mathsf{c}}(1\mathsf{c!}).\textbf{end}, 1\mathsf{c?}]\!] &= \{\top\}
\end{aligned}$$

When the process is given resources 1c!, two traces are possible: one recording that the process is blocking along c!, and the other recording that, after sending along c, the process successfully terminates (an implicit ϵ). If the process also owns 1c?, its behavior changes drastically. It is no longer thought of as blocking along c!,

[2] Compare this side-condition to the free variable check in the π-calculus LTS [SW01].

because no environment could possibly receive on c. Instead it is deadlocked, signified by $\mathcal{B}\langle\rangle$. Likewise the trace representing communication has disappeared. Finally, if the process is not given any resources for sending along c, it faults, producing a trace $\top$.

As a final example, we show how blocking actions allow internal and external nondeterminism to be distinguished. Let $\sigma = 1\mathsf{c}! \oplus 1\mathsf{d}!$, and consider the following processes:

$$\mathcal{O}[\![\bar{\mathsf{c}}(1\mathsf{c}!).\mathbf{end} \vee \bar{\mathsf{d}}(1\mathsf{c}!).\mathbf{end}, \sigma]\!] = \left\{ \begin{array}{l} \mathcal{B}\langle \mathsf{c}!\rangle, \mathcal{C}\langle \mathsf{c}!1\mathsf{c}!\rangle, \\ \mathcal{B}\langle \mathsf{d}!\rangle, \mathcal{C}\langle \mathsf{d}!1\mathsf{c}!\rangle \end{array} \right\}$$

$$\mathcal{O}[\![\bar{\mathsf{c}}(1\mathsf{c}!).\mathbf{end} + \bar{\mathsf{d}}(1\mathsf{c}!).\mathbf{end}, \sigma]\!] = \{\mathcal{B}\langle \mathsf{c}!, \mathsf{d}!\rangle,\ \mathcal{C}\langle \mathsf{c}!1\mathsf{c}!\rangle,\ \mathcal{C}\langle \mathsf{d}!1\mathsf{c}!\rangle\}$$

The first process internally decides whether to send on c or d, whereas the second process offers both communications and allows the environment to choose. Thus the first process nondeterministically blocks on *either* c! or d!—but not both. The communication traces are the same for both.

Note that the sets of traces being produced are not prefix-closed. Finite traces always end with an observation about the state of the process—it is about to fault, it is blocked, it is about to make an assumption, or it has successfully terminated.

The trace semantics of configurations uses the following definitions, borrowed from the failures/divergences model of CSP [BR84]:

Stability, termination, acceptances

$$\begin{array}{rcl} \kappa \text{ stable} & \triangleq & \kappa \xrightarrow{\ell} \kappa' \implies \ell \in \mathcal{C} \cup \mathcal{A} \\ \kappa \text{ term} & \Leftrightarrow & \kappa = \mathbf{end}, \sigma \text{ or } \kappa = \kappa_1|\kappa_2,\ \kappa_1, \kappa_2 \text{ term} \\ \mathrm{acc}(\kappa) & \triangleq & \{\mathsf{c}\delta \ :\ \kappa \xrightarrow{\mathcal{C}\langle \mathsf{c}\delta -\rangle}\} \cup \{\mathsf{c}?\ :\ \kappa \xrightarrow{\mathcal{A}\langle \mathsf{c}? -\rangle}\} \end{array}$$

A configuration is *stable* if it is unable to take any internal steps, such as τ or allocation steps; we use $\mathcal{C}$ and $\mathcal{A}$ to stand for the set of communication and assumption actions, respectively. A configuration has (successfully) *terminated* if every process term within it is inert. The *acceptances* of a configuration are the channel-direction pairs along which it is willing to communicate.

The observation function $\mathcal{O}[\![-]\!]$ is defined by the inference rules below. Because traces may be infinite, we interpret these rules *coinductively*, which we designate by placing a ∞ marker next to each rule. The effect is that infinite derivations are permitted.

Observation *greatest fixpoint of:*

$$\frac{\mathsf{t} \in \mathcal{O}[\![\kappa']\!] \quad \kappa \xrightarrow{\alpha} \kappa' \quad \alpha \notin \mathcal{A}}{\alpha\mathsf{t} \in \mathcal{O}[\![\kappa]\!]}\infty \qquad \frac{\mathsf{t} \in \mathcal{O}[\![\kappa']\!] \quad \kappa \xrightarrow{\tau} \kappa'}{\mathsf{t} \in \mathcal{O}[\![\kappa]\!]}\infty \qquad \frac{\kappa \xrightarrow{\mathcal{A}\langle \mathsf{c?p}\rangle} \kappa'}{\mathcal{A}\langle \mathsf{c?p}\rangle \in \mathcal{O}[\![\kappa]\!]}\infty$$

$$\frac{\kappa \xrightarrow{\top} \bullet}{\top \in \mathcal{O}[\![\kappa]\!]}\infty \qquad \frac{\kappa \text{ stable} \quad \neg(\kappa \text{ term})}{\mathcal{B}\langle \mathrm{acc}(\kappa)\rangle \in \mathcal{O}[\![\kappa]\!]}\infty \qquad \frac{\kappa \text{ term}}{\epsilon \in \mathcal{O}[\![\kappa]\!]}\infty$$

The definition of $\mathcal{O}[\![-]\!]$ echos the failures/divergences model [BR84] in two important ways. The first is the requirement that a configuration be stable before a blocking action is generated. The importance of stability is to ensure that the configuration is *consistently* offering a specific set of communications: internal steps can alter the communications a configuration offers its environment.

The second is the handling of *divergence*: a configuration may never reach a stable state, instead following an infinite sequence of internal computation steps $\kappa_1 \xrightarrow{\tau} \kappa_2 \xrightarrow{\tau} \cdots$. In this case, *every* trace $\mathsf{t} \in \mathsf{Trace}$ is an observable of κ_1, because we can derive $\mathsf{t} \in \mathcal{O}[\![\kappa_1]\!]$ by applying the rule for τ steps infinitely, progressing from κ_1 to κ_2 and so on. In particular, this means that $\top \in \mathcal{O}[\![\kappa_1]\!]$, *i.e.*, divergence leads to noisy failure.

4. A temporal separation logic

The purpose of our temporal logic is to express and check specifications for processes. To do this, we consider the set of traces observed of a process for given resources σ, and ask whether each trace is permitted by the specification. What it means for an observation to be permitted is determined by our notion of *refinement*.

4.1 Refinement

Refinement is closely related to (internal) nondeterminism: roughly speaking, P refines Q (written $P \sqsubseteq Q$) if every behavior P exhibits is a behavior Q *might* exhibit (sometimes glossed as: P is "more deterministic" than Q). Let

$$P \triangleq \mathsf{c}(k).\bar{\mathsf{d}}(k).\mathbf{end} \qquad Q \triangleq \mathsf{c}(k).(\bar{\mathsf{d}}(k) \vee \bar{\mathsf{d}}(k+1)).\mathbf{end}$$

Then $P \sqsubseteq Q$: an environment expecting to interact with Q will be satisfied by interacting with P instead. In this sense, viewing Q as a specification, P is an implementation of Q. Refinement will be our primary tool for relating processes to specifications.

Our notion of refinement begins with actions and traces:

Action refinement *smallest partial order such that:*

$$\begin{array}{rcll} \mathcal{B}\langle\Delta\rangle & \sqsubseteq & \mathcal{B}\langle\Delta'\rangle & \text{if } \Delta' \subseteq \Delta \\ \mathcal{A}\langle \mathsf{c?p}\rangle & \sqsubseteq & \mathcal{A}\langle \mathsf{c?q}\rangle & \text{if } \mathsf{q} \subseteq \mathsf{p} \end{array}$$

Trace refinement *smallest partial order such that*

$$\begin{array}{rcll} \mathsf{tu} & \sqsubseteq & \mathsf{t}\top & \\ \mathsf{t}\alpha & \sqsubseteq & \mathsf{t}\beta & \alpha \sqsubseteq \beta \\ \mathsf{t}\mathcal{C}\langle \mathsf{c?m}\rangle\mathsf{u} & \sqsubseteq & \mathsf{t}\mathcal{A}\langle \mathsf{c?p}\rangle & \mathsf{m} \notin \mathsf{p} \end{array}$$

For sets of traces T and U, we say that

$$T \sqsubseteq U \triangleq \forall \mathsf{t} \in T\ .\ \exists \mathsf{u} \in U\ .\ \mathsf{t} \sqsubseteq \mathsf{u}$$

Refinement on trace sets is a preorder, and if $\top \in U$, then $T \sqsubseteq U$ for any T. Informally, if $\mathcal{O}[\![\kappa]\!] \sqsubseteq \mathcal{O}[\![\kappa']\!]$, then any sequence of interactions with κ must be a possible sequence of interactions with κ', until:

- κ' faults, in which case κ may behave arbitrarily,
- κ blocks, in which case κ' must block on fewer directions,
- κ makes an assumption, in which case κ' must make a stronger assumption, or
- κ allows a communication that κ' assumes cannot occur, in which case κ may go on to behave arbitrarily.

The blocking case is particularly important, because it allows the specification of liveness properties: if an environment expects an offered communication to eventually be accepted by κ' and $\kappa \sqsubseteq \kappa'$ then the offer must likewise be accepted by κ.

4.2 Logic

We now have all the tools we need to define our logic:

Syntax of formulas

$$\begin{array}{l} \varphi ::= p \mid \mathbf{end} \mid \mathcal{C}\langle c\delta m\rangle\varphi \mid \mathcal{N}\langle c\rangle\varphi \mid \mathcal{A}\langle c?p\rangle \mid \mathcal{B}\langle\Delta\rangle \\ \quad\ \mid \varphi|\psi \mid X \mid \nu X.\varphi \mid \mu X.\varphi \mid \varphi \vee \psi \mid \varphi \wedge \psi \mid \varphi \Rightarrow \psi \\ \quad\ \mid \forall x.\varphi \mid \forall \iota.\varphi \mid \exists x.\varphi \mid \exists \iota.\varphi \end{array}$$

Specification variables X may only occur in positive positions. Message expressions $m ::= fc\delta$ denote messages as follows: $[\![fc\delta]\!]^\rho \triangleq [\![f]\!]^\rho\ [\![c]\!]^\rho\ \delta$. There is potential ambiguity between operators $\wedge$, $\vee$, *etc.* as used in formulas and as used in resource predicates (which can appear in formulas). Fortunately, the interpretations always agree. Separating connectives bind tighter than other logical connectives, so $p * q \wedge \varphi$ is read as $(p * q) \wedge \varphi$.

The semantics of formulas is given by a satisfaction relation:

Semantics of formulas $\mathsf{t} \models_\rho^\sigma \varphi$

$$\begin{array}{llll}
\mathsf{t} & \models_\rho^\sigma p & \text{iff} & \sigma \models_\rho p \\
\mathsf{t} & \models_\rho^\sigma \textbf{end} & \text{iff} & \mathsf{t} = \epsilon \\
\mathsf{t} & \models_\rho^\sigma \mathcal{C}\langle c\delta m\rangle\varphi & \text{iff} & \mathsf{t} \sqsubseteq \alpha\mathsf{u},\ \mathsf{u} \models_\rho^{(\!|\alpha|\!)\sigma} \varphi,\ \alpha = \mathcal{C}\langle [\![c]\!]^\rho\, \delta\, [\![m]\!]^\rho\rangle \\
\mathsf{t} & \models_\rho^\sigma \mathcal{N}\langle c\rangle\varphi & \text{iff} & \mathsf{t} \sqsubseteq \alpha\mathsf{u},\ \mathsf{u} \models_\rho^{(\!|\alpha|\!)\sigma} \varphi,\ \alpha = \mathcal{N}\langle [\![c]\!]^\rho\rangle \\
\mathsf{t} & \models_\rho^\sigma \mathcal{A}\langle c?p\rangle & \text{iff} & \mathsf{t} \sqsubseteq \alpha,\ (\!|\alpha|\!)\sigma \neq \top, \bot,\ \ \alpha = \mathcal{A}\langle [\![c]\!]^\rho ?\, [\![p]\!]^\rho\rangle \\
\mathsf{t} & \models_\rho^\sigma \mathcal{B}\langle \Delta\rangle & \text{iff} & \mathsf{t} \sqsubseteq \alpha,\ (\!|\alpha|\!)\sigma \neq \top, \bot,\ \ \alpha = \mathcal{B}\langle [\![\Delta]\!]^\rho\rangle \\
\mathsf{t} & \models_\rho^\sigma \varphi|\psi & \text{iff} & \mathsf{t} \sqsubseteq \mathsf{u},\ \text{with}\ \ \sigma \vdash \mathsf{u} \in \mathsf{u}_1|\mathsf{u}_2,\ \sigma = \sigma_1 \oplus \sigma_2, \\
 & & & \qquad\qquad \mathsf{u}_1 \models_\rho^{\sigma_1} \varphi,\ \mathsf{u}_2 \models_\rho^{\sigma_2} \psi \\
\mathsf{t} & \models_\rho^\sigma \varphi \vee \psi & \text{iff} & \mathsf{t} \models_\rho^\sigma \varphi \text{ or } \mathsf{t} \models_\rho^\sigma \psi \\
\mathsf{t} & \models_\rho^\sigma \varphi \wedge \psi & \text{iff} & \mathsf{t} \models_\rho^\sigma \varphi \text{ and } \mathsf{t} \models_\rho^\sigma \psi \\
\mathsf{t} & \models_\rho^\sigma \varphi \Rightarrow \psi & \text{iff} & \text{for all } \mathsf{u} \sqsubseteq \mathsf{t},\ \mathsf{u} \models_\rho^\sigma \varphi \text{ implies } \mathsf{u} \models_\rho^\sigma \psi \\
\mathsf{t} & \models_\rho^\sigma \forall x.\varphi & \text{iff} & \mathsf{t} \models_{\rho[x\mapsto \mathsf{c}]}^\sigma \varphi \text{ for all } \mathsf{c} \\
\mathsf{t} & \models_\rho^\sigma \forall \iota.\varphi & \text{iff} & \mathsf{t} \models_{\rho[\iota\mapsto \mathsf{f}]}^\sigma \varphi \text{ for all } \mathsf{f} \\
\mathsf{t} & \models_\rho^\sigma \exists x.\varphi & \text{iff} & \mathsf{t} \models_{\rho[x\mapsto \mathsf{c}]}^\sigma \varphi \text{ for some } \mathsf{c} \\
\mathsf{t} & \models_\rho^\sigma \exists \iota.\varphi & \text{iff} & \mathsf{t} \models_{\rho[\iota\mapsto \mathsf{f}]}^\sigma \varphi \text{ for some } \mathsf{f} \\
\mathsf{t} & \models_\rho^\sigma X & \text{iff} & \mathsf{t} \in \rho(X)(\sigma) \\
\mathsf{t} & \models_\rho^\sigma \nu X.\varphi & \text{iff} & \mathsf{t} \in \left(\bigsqcup\{\mathsf{B} \ :\ \mathsf{B} \sqsubseteq [\![\varphi]\!]^{\rho[X\mapsto \mathsf{B}]}\}\right)(\sigma) \\
\mathsf{t} & \models_\rho^\sigma \mu X.\varphi & \text{iff} & \mathsf{t} \in \left(\bigsqcap\{\mathsf{B} \ :\ \mathsf{B} \sqsupseteq [\![\varphi]\!]^{\rho[X\mapsto \mathsf{B}]}\}\right)(\sigma)
\end{array}$$

The satisfaction relation is parameterized by a resource σ and environment ρ. Environments map specification variables X, channel variables c, and permission variables ι to behaviors B, channels c, and permission quantities f respectively. We define behaviors in the discussion on fixpoints below.

Trace refinement is used in the semantics of several formula constructors. Applying refinement ensures the property that satisfaction is *downward-closed* under refinement: if $\mathsf{u} \models_\rho^\sigma \varphi$ and $\mathsf{t} \sqsubseteq \mathsf{u}$ then $\mathsf{t} \models_\rho^\sigma \varphi$. Downward-closure matches the intuition of a more refined trace being "better" than a less refined one for specification purposes. In particular, a specification that permits faulting permits every observation: if $\top \models_\rho^\sigma \varphi$ then $\mathsf{t} \models_\rho^\sigma \varphi$ for all t.

In the definition of the modalities $\mathcal{C}\langle c\delta m\rangle$ and $\mathcal{N}\langle c\rangle$, the use of action semantics $(\!|\alpha|\!)\sigma$ as a parameter to the satisfaction relation requires, in particular, that $(\!|\alpha|\!)\sigma \neq \top, \bot$. All action modalities require the action to be both possible and permitted. For example, no trace satisfies $\mathsf{emp} \wedge \mathcal{C}\langle \mathsf{c}!1\mathsf{c}!\rangle\varphi$ for any φ, because communication on c is not permitted with resources emp. Likewise no trace satisfies $1\mathsf{c}! * 1\mathsf{c}? \wedge \mathcal{C}\langle \mathsf{c}!1\mathsf{c}!\rangle\varphi$, because communication on c is impossible when all resources for c are owned.

Parallel composition of formulas is defined in terms of parallel composition of traces, below. We overload pos so that $\mathsf{pos}(\sigma) = \{\mathsf{c}\delta \ :\ \sigma(\mathsf{c}\overline{\delta}) < 1\}$ is the set of possible communication directions. The judgment $\sigma \vdash \mathsf{t} \in \mathsf{u}|\mathsf{v}$ is read: t is a possible interleaving of u and v, given the total resources σ. The definition is largely a translation of the operational semantics into denotational terms. Take, for example, the rule for interleaving:

$$\mathrm{L}\,\frac{(\!|\alpha|\!)\sigma \vdash \mathsf{t} \in \mathsf{u}|\mathsf{v}}{\sigma \vdash \alpha\mathsf{t} \in \alpha\mathsf{u}|\mathsf{v}}\,\infty$$

This rule mirrors the corresponding one in the operational semantics: an action α from one of the two traces is consumed, but is checked against the resources σ collectively held, to ensure that it represents a possible allocation or interaction with the environment.

Parallel trace composition $\sigma \vdash \mathsf{t} \in \mathsf{u}|\mathsf{v}$ *greatest fixpoint of:*

$$\frac{}{\sigma \vdash \epsilon \in \epsilon|\epsilon}\,\infty \qquad \mathrm{L}\,\frac{}{\sigma \vdash \top \in \top|\mathsf{v}}\,\infty \qquad \mathrm{L}\,\frac{(\!|\alpha|\!)\sigma \vdash \mathsf{t} \in \mathsf{u}|\mathsf{v}}{\sigma \vdash \alpha\mathsf{t} \in \alpha\mathsf{u}|\mathsf{v}}\,\infty$$

$$\frac{\sigma \vdash \mathsf{t} \in \mathsf{u}|\mathsf{v}}{\sigma \vdash \mathsf{t} \in \alpha\mathsf{u}|\overline{\alpha}\mathsf{v}}\,\infty \qquad \mathrm{L}\,\frac{\mathsf{m} \notin \mathsf{p}}{\sigma \vdash \top \in \mathcal{C}\langle \mathsf{c}!\mathsf{m}\rangle\mathsf{u}|\mathcal{A}\langle \mathsf{c}?\mathsf{p}\rangle}\,\infty$$

$$\mathrm{L}\,\frac{(\!|\mathcal{A}\langle \mathsf{c}?\mathsf{p}\rangle|\!)\sigma \neq \top, \bot}{\sigma \vdash \mathcal{A}\langle \mathsf{c}?\mathsf{p}\rangle \in \mathcal{A}\langle \mathsf{c}?\mathsf{p}\rangle|\mathsf{v}}\,\infty \qquad \mathrm{L}\,\frac{\Delta' = \Delta \cap \mathsf{pos}(\sigma)}{\sigma \vdash \mathcal{B}\langle \Delta'\rangle \in \mathcal{B}\langle \Delta\rangle|\epsilon}\,\infty$$

$$\frac{\{\mathsf{c}\delta \ :\ \mathsf{c}\delta \in \Delta_1, \mathsf{c}\overline{\delta} \in \Delta_2\} = \emptyset \qquad \Delta = (\Delta_1 \cup \Delta_2) \cap \mathsf{pos}(\sigma)}{\sigma \vdash \mathcal{B}\langle \Delta\rangle \in \mathcal{B}\langle \Delta_1\rangle|\mathcal{B}\langle \Delta_2\rangle}\,\infty$$

Operationally, a process is blocked if it can only make progress by communicating with its environment. By placing two blocked processes in parallel, they become part of each other's environment. If they are attempting to communicate in opposite directions, then they *are* able to make progress without external communication. Thus, two blocking actions in parallel produce a blocking trace only if they do not block in any opposite directions.

When a parallel composition results in a blocking action, the blocking action is trimmed to reflect possible communications given the total resources owned. To understand why, consider that parallel composition $\varphi|\psi$ splits resources σ into σ_1 and σ_2, and interleaves traces satisfying φ and ψ under resources σ_1 and σ_2 respectively. Suppose that $\sigma_1 = 1\mathsf{c}!$ and $\sigma_2 = 1\mathsf{c}?$, and that φ permits blocking along $\mathsf{c}!$ but $\psi = \textbf{end}$. The total resources $1\mathsf{c}! \oplus 1\mathsf{c}?$ rule out any interaction along c by the environment of $\varphi|\psi$. Thus, the attempt by φ to send on c cannot succeed, and the parallel composition is deadlocked. Formally, this is reflected by taking the intersection of the blocking set $\{\mathsf{c}!\}$ with the set $\mathsf{pos}(1\mathsf{c}! \oplus 1\mathsf{c}?)$, which does not include $\mathsf{c}!$.

Our logic contains as a fragment first-order logic, the semantics of which is standard except for implication. The definition of implication is forced by the structure of the logic—especially the use of refinement—and is justified as follows:

Definition 1. A formula φ is *valid* (or *tautological*) if, for all t, σ, ρ, we have $\mathsf{t} \models_\rho^\sigma \varphi$.

Proposition 2. $\varphi \Rightarrow \psi$ is valid iff for all t, σ, ρ, we have $\mathsf{t} \models_\rho^\sigma \varphi$ implies $\mathsf{t} \models_\rho^\sigma \psi$.

Proposition 3. The axioms of intuitionistic first-order logic are valid, and its inference rules are sound.

Recall that tt,ff are resource predicates and hence formulas. We define $\neg\varphi \triangleq \varphi \Rightarrow \mathrm{ff}$. Our logic is intuitionistic because $\varphi \vee \neg\varphi$ is not valid for all φ.[3]

Temporal properties are expressed in the usual way for a μ-calculus (see [BS01] for a concise introduction). The fixpoint operators μ and ν provide distinct ways to interpret recursion. Roughly speaking, μ only allows the recursion to be unfolded a finite number of times, whereas ν allows infinite unfolding. For example, the formula $\mu X.\mathcal{C}\langle \mathsf{c}!n\rangle\textbf{end} \vee \exists x.\mathcal{N}\langle x\rangle X$ allows only traces that perform a finite number of allocations, then finally communicate, whereas using ν would also allow traces that allocate infinitely without communicating.

[3] Order-theoretically, refinement is a complete Heyting algebra, but not a Boolean algebra.

In order to define the fixpoint operators, we need the notion of a *behavior* B, which is a function from resources Σ to sets of traces. Behaviors are ordered pointwise: $\mathsf{B}_1 \sqsubseteq \mathsf{B}_2$ iff $\mathsf{B}_1(\sigma) \sqsubseteq \mathsf{B}_2(\sigma)$ for all σ. This ordering is a complete lattice.

We define the behavior $\llbracket\varphi\rrbracket^\rho(\sigma) \triangleq \{\mathsf{t} \ : \ \mathsf{t} \models^\sigma_\rho \varphi\}$. Environments ρ map specification variables to behaviors. Formulas are monotonic in the usual sense: if $\mathsf{B}_1 \sqsubseteq \mathsf{B}_2$ then

$$\llbracket\varphi\rrbracket^{\rho[X\mapsto \mathsf{B}_1]}(\sigma) \sqsubseteq \llbracket\varphi\rrbracket^{\rho[X\mapsto \mathsf{B}_2]}(\sigma)$$

It follows, by the Knaster-Tarski fixpoint theorem, that our definitions of ν and μ do in fact compute the greatest and least fixpoints, respectively. Therefore the following inference rules, which capture induction and coinduction, are sound:

$$\frac{\varphi\{\psi/X\} \Rightarrow \psi}{(\mu X.\varphi) \Rightarrow \psi} \qquad \frac{\varphi \Rightarrow \psi\{\varphi/X\}}{\varphi \Rightarrow \nu X.\psi}$$

There are also valid unfolding axioms for both fixpoints:

$$(\mu X.\varphi) \Leftrightarrow \varphi\{\mu X.\varphi/X\} \qquad (\nu X.\varphi) \Leftrightarrow \varphi\{\nu X.\varphi/X\}$$

The satisfaction relation is lifted to closed processes as follows:

$$P \models \varphi \ \triangleq\ \sigma \in \Sigma \text{ and } \mathsf{t} \in \mathcal{O}\llbracket P, \sigma\rrbracket \ \text{ implies } \mathsf{t} \models^\sigma_\emptyset \varphi$$

We can characterize lifted satisfaction in terms of refinement:[4]

$$P \models \varphi \text{ iff } \sigma \in \Sigma \text{ implies } \mathcal{O}\llbracket P, \sigma\rrbracket \sqsubseteq \llbracket\varphi\rrbracket^\emptyset(\sigma)$$

Our temporal logic allows assertions about both resource states and actions, which raises an important question: how do such assertions interact? The following formulas, which are valid in the logic, provide the answer:

Strongest (liberal) postcondition reasoning

$$\begin{array}{rcl}
p \wedge \mathcal{N}\langle c\rangle\varphi & \Rightarrow & \mathcal{N}\langle c\rangle(p * 1c! * 1c? \wedge \varphi) \\
p \wedge \mathcal{C}\langle c?m\rangle\varphi & \Rightarrow & \mathcal{C}\langle c?m\rangle(p * m \wedge \varphi) \\
p \wedge \mathcal{C}\langle c!m\rangle\varphi & \Rightarrow & \mathcal{C}\langle c!m\rangle(p \circledast\!\!- m \wedge \varphi)
\end{array}$$

For each prefix action, we can prove that the postcondition axiom transforms an assertion about the resource state before the action to the strongest one holding after the action—assuming that the action can occur from the initial state. When receiving, the message resources are added to the initial resource assertion, and when sending, they are subtracted.

The postcondition axioms are crucial to doing separation logic-style reasoning within our temporal logic. The final step is to interpret processes as formulas, which we do next.

Section 6 discusses additional inference rules for the logic, capturing an expansion law and its interference-free variant.

5. Denotational semantics

The operational semantics gives us one way to compare processes to formulas: we can ask whether $\mathcal{O}\llbracket P, \sigma\rrbracket \sqsubseteq \llbracket\varphi\rrbracket^\emptyset(\sigma)$. In this section, we give a compositional translation of processes into formulas, allowing us to instead ask whether the formula $P \Rightarrow \varphi$ is valid.

Taking this step has several ramifications. It immediately yields compositional proof rules, because the operators of the logic are monotonic. For example, the soundness of the following inference rule follows from the monotonicity of $\vee$:

$$\frac{P \Rightarrow \varphi \quad Q \Rightarrow \psi}{P \vee Q \Rightarrow \varphi \vee \psi}$$

Moreover, the logical interpretation of processes provides a denotational semantics for them. Our formulas can be seen as *heterogeneous specifications* [CL02], containing operators both from process algebra and temporal logic. Finally, we have a notion of *implementability*: φ is implementable if there is some process P such that $P \Rightarrow \varphi$ is valid. No process P implements ff.

Much of the syntax of processes purposefully overlaps with that of formulas. The constructors that do not—allocation, recursion, annotated parallel composition, and external choice—we define as syntactic sugar for formula constructors:

Derived forms: process syntax

$$\begin{array}{rcl}
\textbf{new}\ x.\varphi & \triangleq & \exists x.\mathcal{N}\langle x\rangle\varphi \\
\textbf{fix}\ X.\varphi & \triangleq & \nu X.\varphi \\
\varphi\ {}_p\|_q\ \psi & \triangleq & (p * q) \Rightarrow (p \wedge \varphi \mid q \wedge \psi) \\
\sum_i \pi_i.\varphi_i & \triangleq & \bigwedge_i \mathrm{pre}(\pi_i) \ \Rightarrow\ \left(\bigvee_i \mathrm{acts}(\pi_i, \varphi_i) \vee \bigwedge_i \mathcal{B}\langle\pi_i\rangle\right)
\end{array}$$

Allocation is simple enough: we existentially bind the allocated channel variable, and use the $\mathcal{N}\langle - \rangle\varphi$ modality to capture the allocation itself. The translation of recursion is similarly thin: we simply interpret it as taking the greatest fixpoint, a choice justified by the theorem below.

Parallel composition and external choice both have the overall shape $p \Rightarrow \varphi$. The resource predicate in the implication acts as a *precondition* on the resource state. If $\sigma \not\models_\rho p$, then for *every* trace t we have $\mathsf{t} \models^\sigma_\rho (p \Rightarrow \varphi)$. In particular, the faulting trace $\top$ satisfies the implication. Thus we can read a formula $p \Rightarrow \varphi$ as "if p is satisfied, behave as φ; otherwise, fault."[5]

For an annotated parallel composition, the initial resources must be separable according to the annotations. The subformulas are placed in parallel, each assuming its annotation.

External choice is more complicated. The precondition $\mathrm{pre}(\pi)$ for each prefix in the choice must be satisfied. Moreover, each prefix contributes both communication/assumption actions $\mathrm{acts}(\pi, \varphi)$ and blocking actions $\mathcal{B}\langle\pi\rangle$, because operationally, prefixes generate both communication steps and blocking observations.

Definitions for external choice

$$\begin{array}{rcl}
\mathrm{pre}(\overline{c}(m)) & \triangleq & (c! * \mathrm{tt}) \wedge (m * \mathrm{tt}) \\
\mathrm{pre}(c(\iota x\delta{:}p)) & \triangleq & c? * \mathrm{tt} \\
c\delta & \triangleq & \exists\iota.\iota > 0 \wedge \iota c\delta \\
\\
\mathrm{acts}(\overline{c}(m), \varphi) & \triangleq & \mathcal{C}\langle c!m\rangle\varphi \\
\mathrm{acts}(c(\iota x\delta{:}p), \varphi) & \triangleq & \exists\iota.\exists x.\mathcal{C}\langle c?(\iota x\delta)\rangle(p \wedge \varphi) \\
 & \vee & \mathcal{A}\langle c?(\exists\iota.\exists x.\iota x\delta \wedge p)\rangle \\
\\
\mathcal{B}\langle\overline{c}(m)\rangle & \triangleq & \mathcal{B}\langle c!\rangle \vee (1c? \wedge \mathcal{B}\langle\rangle) \\
\mathcal{B}\langle c(\iota x\delta{:}p)\rangle & \triangleq & \mathcal{B}\langle c?\rangle \vee (1c! \wedge \mathcal{B}\langle\rangle)
\end{array}$$

The preconditions for prefixes are straightforward. The actions they generate, given by the $\mathrm{acts}(\pi, \varphi)$ function, are precisely those given by the prefix stepping relation on page 5.

The blocking set for a prefix reflects the attempted communication, but is empty if the resources owned by the process prevent its environment from communicating in the opposite direction. Because of the definition of refinement for blocking actions, the formulas $\mathcal{B}\langle\Delta_1\rangle \wedge \mathcal{B}\langle\Delta_2\rangle$ and $\mathcal{B}\langle\Delta_1 \cup \Delta_2\rangle$ are equivalent. Thus, when blocking actions from prefixes are conjoined in an external choice, the effect is that the choice blocks along *all* channel-direction pairs.

The processes-as-formulas interpretation is justified as follows.

[4] In fact, because $\llbracket - \rrbracket$ is downward-closed, we can replace $\sqsubseteq$ by $\subseteq$.

[5] This approach is related to specification statements in [COY07].

Theorem 4 (Adequacy). If P closed then, for all $\sigma \in \Sigma$,

- $\mathcal{O}[\![P, \sigma]\!] \sqsupseteq [\![P]\!]^{\emptyset}(\sigma)$, and
- $\mathcal{O}[\![P, \sigma]\!] \sqsubseteq [\![P]\!]^{\emptyset}(\sigma)$.

The proof of this theorem is given in our forthcoming technical report. Because our language is first-order (channels, not processes, are passed as messages), elementary domain-theoretic notions suffice: monotonicity, complete lattices, and the Knaster-Tarski theorem. To prove adequacy, we generalize the result to open terms. Soundness ($\sqsupseteq$) is proved by coinduction, because $\mathcal{O}[\![-]\!]$ is defined as a greatest fixpoint. Completeness ($\sqsubseteq$) is proved by induction on P, but we use coinduction within the recursion case. The proofs are largely straightforward. The most difficult case is parallel composition for completeness, for which we prove the following lemma:

Lemma 5. If $\mathsf{t} \in \mathcal{O}[\![\kappa_1 | \kappa_2]\!]$ then $\mathrm{res}(\kappa_1|\kappa_2) \vdash \mathsf{t}' \in \mathsf{u}|\mathsf{v}$ for some $\mathsf{t}', \mathsf{u}, \mathsf{v}$ with $\mathsf{t} \sqsubseteq \mathsf{t}'$, $\mathsf{u} \in \mathcal{O}[\![\kappa_1]\!]$ and $\mathsf{v} \in \mathcal{O}[\![\kappa_2]\!]$.

The proof of this lemma requires a three-way coinduction: one for proving $\mathrm{res}(\kappa_1|\kappa_2) \vdash \mathsf{t}' \in \mathsf{u}|\mathsf{v}$, for $\mathsf{u} \in \mathcal{O}[\![\kappa_1]\!]$ and for $\mathsf{v} \in \mathcal{O}[\![\kappa_2]\!]$.

6. Examples

We have seen a number of examples so far that illustrate the resource model and process semantics. We now revisit these examples from the perspective of the logic, illustrating resource-based, process-algebraic, and temporal reasoning within the logic. We close with an example tying together all three reasoning styles.

6.1 Resource-based reasoning

We begin with the examples from Section 3, in particular with the process term $\overline{\mathsf{c}}(1\mathsf{c}!).\mathbf{end}$. Using the definitions of the previous section, the term is syntactic sugar for the formula

$$\begin{array}{rcl} \varphi & \triangleq & (\mathsf{c}! * \mathrm{tt} \wedge 1\mathsf{c}! * \mathrm{tt}) \Rightarrow (\mathcal{C}\langle \mathsf{c}!1\mathsf{c}!\rangle.\mathbf{end} \;\vee\; \mathcal{B}\langle \overline{\mathsf{c}}(1\mathsf{c}!)\rangle) \\ & \Leftrightarrow & (1\mathsf{c}! * \mathrm{tt}) \Rightarrow (\mathcal{C}\langle \mathsf{c}!1\mathsf{c}!\rangle.\mathbf{end} \;\vee\; \mathcal{B}\langle \overline{\mathsf{c}}(1\mathsf{c}!)\rangle) \end{array}$$

As before, we consider the behavior of φ in resource contexts $1\mathsf{c}!$, $1\mathsf{c}! \oplus 1\mathsf{c}?$ and emp, resulting in communication, deadlock, and noisy failure respectively. To do this within the logic, we conjoin φ with corresponding resource predicates. We highlight in gray the portion of the formula being transformed. For $1\mathsf{c}!$, we reason:

$$\begin{array}{rl} 1\mathsf{c}! \wedge \varphi & \Rightarrow\; 1\mathsf{c}! \wedge (\mathcal{C}\langle \mathsf{c}!1\mathsf{c}!\rangle.\mathbf{end} \vee \mathcal{B}\langle \overline{\mathsf{c}}(1\mathsf{c}!)\rangle) \\ & \Rightarrow\; (\,1\mathsf{c}! \wedge \mathcal{C}\langle \mathsf{c}!1\mathsf{c}!\rangle.\mathbf{end}\,) \vee (1\mathsf{c}! \wedge \mathcal{B}\langle \overline{\mathsf{c}}(1\mathsf{c}!)\rangle) \\ & \Rightarrow\; \mathcal{C}\langle \mathsf{c}!1\mathsf{c}!\rangle.(\,1\mathsf{c}! \circledast\!\!- 1\mathsf{c}! \wedge \mathbf{end}\,) \vee (1\mathsf{c}! \wedge \mathcal{B}\langle \overline{\mathsf{c}}(1\mathsf{c}!)\rangle) \\ & \Rightarrow\; \mathcal{C}\langle \mathsf{c}!1\mathsf{c}!\rangle.(\mathrm{emp} \wedge \mathbf{end}) \vee (\,1\mathsf{c}! \wedge \mathcal{B}\langle \overline{\mathsf{c}}(1\mathsf{c}!)\rangle\,) \\ & \Rightarrow\; \mathcal{C}\langle \mathsf{c}!1\mathsf{c}!\rangle.(\mathrm{emp} \wedge \mathbf{end}) \vee \mathcal{B}\langle \mathsf{c}!\rangle \end{array}$$

The first step follows from *modus ponens*: the assumption $1\mathsf{c}!$ fulfills the precondition of φ (note that we retain the resource assumption). We then distribute conjunction over disjunction. Next, we apply postcondition reasoning to the communication. We simplify, using the valid formula $(1\mathsf{c}! \circledast\!\!- 1\mathsf{c}!) \Leftrightarrow \mathrm{emp}$ (*i.e.*, subtracting $1\mathsf{c}!$ from $1\mathsf{c}!$ yields emp). Finally, we apply $1\mathsf{c}! \wedge \mathcal{B}\langle \overline{\mathsf{c}}(1\mathsf{c}!)\rangle \Rightarrow \mathcal{B}\langle \mathsf{c}!\rangle$, which can be shown valid by propositional reasoning.

Similarly, we calculate for $1\mathsf{c}! * 1\mathsf{c}?$:

$$\begin{array}{rl} 1\mathsf{c}! * 1\mathsf{c}? \wedge \varphi & \Rightarrow\; 1\mathsf{c}! * 1\mathsf{c}? \wedge (\mathcal{C}\langle \mathsf{c}!1\mathsf{c}!\rangle.\mathbf{end} \vee \mathcal{B}\langle \overline{\mathsf{c}}(1\mathsf{c}!)\rangle) \\ & \Rightarrow\; (\,1\mathsf{c}! * 1\mathsf{c}? \wedge \mathcal{C}\langle \mathsf{c}!1\mathsf{c}!\rangle.\mathbf{end}\,) \\ & \qquad \vee (1\mathsf{c}! * 1\mathsf{c}? \wedge \mathcal{B}\langle \overline{\mathsf{c}}(1\mathsf{c}!)\rangle) \\ & \Rightarrow\; \mathrm{ff} \vee (\,1\mathsf{c}! * 1\mathsf{c}? \wedge \mathcal{B}\langle \overline{\mathsf{c}}(1\mathsf{c}!)\rangle\,) \\ & \Rightarrow\; \mathrm{ff} \vee \mathcal{B}\langle\rangle \\ & \Rightarrow\; \mathcal{B}\langle\rangle \end{array}$$

Thus, in the presence of resources $1\mathsf{c}!$ and $1\mathsf{c}?$, φ is deadlocked. Here, we use that $1\mathsf{c}! * 1\mathsf{c}? \wedge \mathcal{C}\langle \mathsf{c}!1\mathsf{c}!\rangle\psi \Rightarrow \mathrm{ff}$ is valid for any ψ, which we justified in Section 4.2. We simplify using the valid formula $(1\mathsf{c}! * 1\mathsf{c}? \wedge \mathcal{B}\langle \overline{\mathsf{c}}(1\mathsf{c}!)\rangle) \Rightarrow \mathcal{B}\langle\rangle$.

What can we say for resources emp? Because $\neg(\mathrm{emp} \wedge 1\mathsf{c}! * \mathrm{tt})$ is valid, $\mathrm{emp} \wedge 1\mathsf{c}! * \mathrm{tt} \;\Rightarrow\; (\mathcal{C}\langle \mathsf{c}!1\mathsf{c}!\rangle.\mathbf{end} \vee \mathcal{B}\langle \overline{\mathsf{c}}(1\mathsf{c}!)\rangle)$ is valid. Hence $\mathrm{emp} \Rightarrow \varphi$. Because $\top \models^{\emptyset}_{\emptyset} \mathrm{emp}$, it follows that $\top \models^{\emptyset}_{\emptyset} \varphi$. Thus we have deduced that, given no resources, the process faults. Generally, when a precondition is not satisfied, the result is a fault.

We can apply similar techniques to reason about the term $\overline{\mathsf{c}}(1\mathsf{c}!).\mathbf{end} + \overline{\mathsf{d}}(1\mathsf{c}!).\mathbf{end}$ with resources $1\mathsf{c}! \oplus 1\mathsf{c}? \oplus 1\mathsf{d}!$. Let $p \triangleq 1\mathsf{c}! * 1\mathsf{c}? * 1\mathsf{d}!$ and $\psi \triangleq \mathcal{B}\langle \overline{\mathsf{c}}(1\mathsf{c}!)\rangle \wedge \mathcal{B}\langle \overline{\mathsf{d}}(1\mathsf{c}!)\rangle$. We have:

$$\begin{array}{rl} & p \wedge (\,\overline{\mathsf{c}}(1\mathsf{c}!).\mathbf{end} + \overline{\mathsf{d}}(1\mathsf{c}!).\mathbf{end}\,) \\ \Rightarrow & p \wedge \big(\mathcal{C}\langle \mathsf{c}!1\mathsf{c}!\rangle\mathbf{end} \;\vee\; \mathcal{C}\langle \mathsf{d}!1\mathsf{c}!\rangle\mathbf{end} \;\vee\; \psi\big) \\ \Rightarrow & (\,p \wedge \mathcal{C}\langle \mathsf{c}!1\mathsf{c}!\rangle\mathbf{end}\,) \;\vee\; (p \wedge \mathcal{C}\langle \mathsf{d}!1\mathsf{c}!\rangle\mathbf{end}) \;\vee\; (p \wedge \psi) \\ \Rightarrow & \mathrm{ff} \;\vee\; (\,p \wedge \mathcal{C}\langle \mathsf{d}!1\mathsf{c}!\rangle\mathbf{end}\,) \;\vee\; (p \wedge \psi) \\ \Rightarrow & \mathrm{ff} \;\vee\; \mathcal{C}\langle \mathsf{d}!1\mathsf{c}!\rangle(\,p \circledast\!\!- 1\mathsf{c}! \;\wedge \mathbf{end}) \;\vee\; (p \wedge \psi) \\ \Rightarrow & \mathrm{ff} \;\vee\; \mathcal{C}\langle \mathsf{d}!1\mathsf{c}!\rangle(1\mathsf{c}? * 1\mathsf{d}! \wedge \mathbf{end}) \;\vee\; (\,p \wedge \psi\,) \\ \Rightarrow & \mathrm{ff} \vee \; \mathcal{C}\langle \mathsf{d}!1\mathsf{c}!\rangle(1\mathsf{c}? * 1\mathsf{d}! \wedge \mathbf{end}) \;\vee\; \mathcal{B}\langle \mathsf{d}!\rangle \\ \Rightarrow & \mathcal{C}\langle \mathsf{d}!1\mathsf{c}!\rangle(1\mathsf{c}? * 1\mathsf{d}! \wedge \mathbf{end}) \;\vee\; \mathcal{B}\langle \mathsf{d}!\rangle \end{array}$$

The steps are: *modus ponens*, distribution, impossibility, postcondition reasoning, and three steps of simplification. The example demonstrates how knowledge about resources can be used to resolve an external choice: because the process owns $1\mathsf{c}?$, its environment can be assumed not to receive on c.

6.2 Process-algebraic reasoning

In process algebra, a fundamental reasoning technique is *expansion*, which allows parallel composition to be replaced by nondeterministic interleaving. Our logic supports expansion as well, both at the level of the basic observables $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, and $\mathcal{N}$, and of process prefixes π. The inference rule below provides a sound expansion law for two processes offering to communicate with each other. The rule says that either (1) the process attempting to send communicates with the external environment, (2) the process attempting to receive does, or (3) the two processes communicate with each other. The rule only applies when the assumption r made by the receiving process is satisfied by the sending one.

$$\frac{p \Rightarrow c! * \mathrm{tt} \wedge fd\delta * \mathrm{tt} \qquad q \Rightarrow c? * \mathrm{tt} \qquad r\{f/\iota, d/x\}}{\begin{array}{l} p * q \wedge (\overline{c}(fd\delta).P \;{}_p\|_q\; c(\iota x\delta : r).Q) \;\Rightarrow \\ \left(\begin{array}{ll} \quad\;\; \overline{c}(fd\delta) & .\,\big(P \;{}_{p \circledast\!- fd\delta}\|_q\; c(\iota x\delta : r).Q\big) \\ \vee\; c(\iota x\delta : r) & .\,\big(\overline{c}(fd\delta).P \;{}_p\|_{q * \iota x\delta}\; (r \wedge Q)\big) \\ \vee \quad P \;{}_{p \circledast\!- fd\delta}\|_{q * fd\delta}\; (r \wedge Q)\{f/\iota, d/x\} & \end{array}\right) \end{array}}$$

The assumptions of the rule include the preconditions for both directions of communication (constraints on p and q). In addition, the rule calculates postconditions, which appear in the annotations for the parallel compositions.

If we add an additional assumption, that the two processes together own all of the resources for the channel of communication, we arrive at the following *interference-free expansion law*:

$$\frac{\begin{array}{cc} p \Rightarrow c! * \mathrm{tt} \wedge fd\delta * \mathrm{tt} & q \Rightarrow c? * \mathrm{tt} \\ p * q \Rightarrow 1c! * 1c? * \mathrm{tt} & r\{f/\iota, d/x\} \end{array}}{\begin{array}{l} p * q \wedge (\overline{c}(fd\delta).P \;{}_p\|_q\; c(\iota x\delta : r).Q) \;\Rightarrow \\ \qquad \big(P \;{}_{p \circledast\!- fd\delta}\|_{q * fd\delta}\; (r \wedge Q)\{f/\iota, d/x\}\big) \end{array}}$$

This rule is derivable from the previous one using resource-based reasoning. It reflects the central idea of this paper: owning all resources for a channel excludes interference from the environment. To illustrate the rule, we apply it to an example from Section 2.1:

$$
\begin{aligned}
&1\mathsf{c}! * 1\mathsf{c}? * \mathsf{tt} \wedge \big(\overline{\mathsf{c}}(3).P \;{}_{1\mathsf{c}!*\mathsf{tt}}\|_{1\mathsf{c}?*\mathsf{tt}}\; \mathsf{c}(x).Q\big)\\
\Rightarrow\;& \big(P \;{}_{1\mathsf{c}!*\mathsf{tt}}\|_{\mathsf{c}?*\mathsf{tt}}\; Q\{3/x\}\big)
\end{aligned}
$$

Because a data value rather than a channel is being communicated, there is no resource transfer between the two processes.

Because of the compositional nature of the logic, we have the following corollary:

$$
\begin{aligned}
&\big(1\mathsf{c}! * 1\mathsf{c}? * \mathsf{tt} \wedge \big(\overline{\mathsf{c}}(3).P \;{}_{1\mathsf{c}!*\mathsf{tt}}\|_{1\mathsf{c}?*\mathsf{tt}}\; \mathsf{c}(x).Q\big)\big) \;{}_{p}\|_{q}\; R\\
\Rightarrow\;& \big(\big(P \;{}_{1\mathsf{c}!*\mathsf{tt}}\|_{\mathsf{c}?*\mathsf{tt}}\; Q\{3/x\}\big) \;{}_{p}\|_{q}\; R\big)
\end{aligned}
$$

for any choice of p, q, and R. In short: we have deduced that communication between the two original processes succeeds in the presence of an arbitrary environment.

What if R breaks the resource assumptions we have made, by attempting to communicate along c? It will fault, which in the logic means it will be equivalent to the formula tt. The parallel composition of tt with any other formula is equivalent to tt, so the entire composition will fault. Thus, for such an R, the reasoning above collapses to a proof that tt $\Rightarrow$ tt.

6.3 Putting it all together: temporal reasoning

As a final example, we show that a recursive process satisfies a temporal specification. The process $\mathsf{counter}$ sends successive integers along a fixed channel c, starting from 0:

$$
\begin{aligned}
\mathsf{counter} &\triangleq \textbf{new}\; z.P_0\\
P_i &\triangleq P \;{}_{1z?*1\mathsf{c}!*\mathsf{tt}}\|_{1z!*\mathsf{tt}}\; \overline{z}(i).\overline{z}(1z!).\textbf{end}\\
P &\triangleq \nu X.z(i).z(\iota x! : \iota = 1 \wedge x = z).\overline{\mathsf{c}}(i).\\
&\qquad\quad (X \;{}_{1z?*1\mathsf{c}!*\mathsf{tt}}\|_{1z!*\mathsf{tt}}\; \overline{z}(i+1).\overline{z}(1z!).\textbf{end})
\end{aligned}
$$

The specification sorted asserts that an infinite, ordered stream of integers is sent along c:

$$
\begin{aligned}
\mathsf{sorted} &\triangleq \nu X.\exists i.\overline{\mathsf{c}}(i).(X \wedge \mathsf{above}_i)\\
\mathsf{above}_i &\triangleq \nu X.\exists j \geq i.\overline{\mathsf{c}}(j).X
\end{aligned}
$$

In order to carry out the proof, we need a formula representing a loop invariant. We define

$$
\varphi_i \triangleq 1z! * 1z? * 1\mathsf{c}! * \mathsf{tt} \wedge \exists j \geq i.P_j
$$

and prove in Figure 3 that $\varphi_i \Rightarrow \exists j \geq i.\overline{\mathsf{c}}(j).\varphi_i$. It follows by coinduction that $\varphi_i \Rightarrow \nu X.\exists j \geq i.\overline{\mathsf{c}}(j).X$, *i.e.*, that $\varphi_i \Rightarrow \mathsf{above}_i$.

When we refer to the numbered lines of the proof, we are referring to the implication *ending* at that line number. Lines 1-4 of the proof are simple expansions of definitions and predicate logic calculations. In line 5, we unroll the recursion once. In line 6, we replace the previously highlighted subformula by P_{k+1}, which is definitionally equivalent. We then use the interference-free inference rule from the previous subsection to match communications in lines 7 and 8. Line 9 uses the fact that **end** is a unit for parallel composition (when annotated with tt). In line 10, we use postcondition reasoning (in a trivial way, because $\overline{\mathsf{c}}(j)$ has no effect on resources). Line 11 replaces a subformula by the equivalent φ_j. Finally, line 12 uses that $\varphi_j \Rightarrow \varphi_i$ when $j \geq i$.

Now we prove that $\varphi_i \Rightarrow \mathsf{sorted}$ for all i, again by coinduction:

$$
\begin{aligned}
\varphi_i &\Rightarrow \exists j \geq i\,.\,\overline{\mathsf{c}}(j).\varphi_j\\
&\Rightarrow \exists j\,.\,\overline{\mathsf{c}}(j).(\varphi_i \wedge \mathsf{above}_j)
\end{aligned}
$$

The first line of this proof is just line 11 from Figure 3. The second line uses that $\varphi_j \Rightarrow \varphi_i$ when $j \geq i$, and that $\varphi_j \Rightarrow \mathsf{above}_j$.

Figure 3: proof that $\varphi_i \Rightarrow \exists j \geq i.\overline{\mathsf{c}}(j).\varphi_i$

$$
\begin{aligned}
\varphi_i \;=\;& 1z! * 1z? * 1\mathsf{c}! * \mathsf{tt} \wedge \exists j \geq i\,.\,P_j && (1)\\
\Rightarrow\;& \exists j \geq i\,.\,1z! * 1z? * 1\mathsf{c}! * \mathsf{tt} \wedge P_j && (2)\\
\Rightarrow\;& \exists j \geq i\,.\,1z! * 1z? * 1\mathsf{c}! * \mathsf{tt} \wedge \big(&& (3)\\
& \quad P \;{}_{1z?*1\mathsf{c}!*\mathsf{tt}}\|_{1z!*\mathsf{tt}}\; \overline{z}(j).\overline{z}(1z!).\textbf{end}\big) &&\\
\Rightarrow\;& \exists j \geq i\,.\,1z! * 1z? * 1\mathsf{c}! * \mathsf{tt} \wedge \big(&& (4)\\
& \quad \nu X.z(k).z(\iota x! : \iota = 1 \wedge x = z).\overline{\mathsf{c}}(k). &&\\
& \qquad (X \;{}_{1z?*1\mathsf{c}!*\mathsf{tt}}\|_{1z!*\mathsf{tt}}\; \overline{z}(k+1).\overline{z}(1z!).\textbf{end}) &&\\
& \quad {}_{1z?*1\mathsf{c}!*\mathsf{tt}}\|_{1z!*\mathsf{tt}}\; \overline{z}(j).\overline{z}(1z!).\textbf{end}\big) &&\\
\Rightarrow\;& \exists j \geq i\,.\,1z! * 1z? * 1\mathsf{c}! * \mathsf{tt} \wedge \big(&& (5)\\
& \quad z(k).z(\iota x! : \iota = 1 \wedge x = z).\overline{\mathsf{c}}(k). &&\\
& \qquad (P \;{}_{1z?*1\mathsf{c}!*\mathsf{tt}}\|_{1z!*\mathsf{tt}}\; \overline{z}(k+1).\overline{z}(1z!).\textbf{end}) &&\\
& \quad {}_{1z?*1\mathsf{c}!*\mathsf{tt}}\|_{1z!*\mathsf{tt}}\; \overline{z}(j).\overline{z}(1z!).\textbf{end}\big) &&\\
\Rightarrow\;& \exists j \geq i\,.\,1z! * 1z? * 1\mathsf{c}! * \mathsf{tt} \wedge \big(&& (6)\\
& \quad z(k).z(\iota x! : \iota = 1 \wedge x = z).\overline{\mathsf{c}}(k).P_{k+1} &&\\
& \quad {}_{1z?*1\mathsf{c}!*\mathsf{tt}}\|_{1z!*\mathsf{tt}}\; \overline{z}(j).\overline{z}(1z!).\textbf{end}\big) &&\\
\Rightarrow\;& \exists j \geq i\,.\,1z! * 1z? * 1\mathsf{c}! * \mathsf{tt} \wedge \big(&& (7)\\
& \quad z(\iota x! : \iota = 1 \wedge x = z).\overline{\mathsf{c}}(j).P_{j+1} &&\\
& \quad {}_{1z?*1\mathsf{c}!*\mathsf{tt}}\|_{1z!*\mathsf{tt}}\; \overline{z}(1z!).\textbf{end}\big) &&\\
\Rightarrow\;& \exists j \geq i\,.\,1z! * 1z? * 1\mathsf{c}! * \mathsf{tt} \wedge && (8)\\
& \quad \big(\overline{\mathsf{c}}(j).P_{j+1} \;{}_{1z?*1z!*1\mathsf{c}!*\mathsf{tt}}\|_{\mathsf{tt}}\; \textbf{end}\big) &&\\
\Rightarrow\;& \exists j \geq i\,.\,1z! * 1z? * 1\mathsf{c}! * \mathsf{tt} \wedge \overline{\mathsf{c}}(j).P_{j+1} && (9)\\
\Rightarrow\;& \exists j \geq i\,.\,\overline{\mathsf{c}}(j).\,(1z! * 1z? * 1\mathsf{c}! * \mathsf{tt} \wedge P_{j+1}) && (10)\\
\Rightarrow\;& \exists j \geq i\,.\,\overline{\mathsf{c}}(j).\varphi_j && (11)\\
\Rightarrow\;& \exists j \geq i\,.\,\overline{\mathsf{c}}(j).\varphi_i && (12)
\end{aligned}
$$

Finally, using postcondition reasoning, we have

$$
\begin{aligned}
1\mathsf{c}! * \mathsf{tt} \wedge \mathsf{counter} &= 1\mathsf{c}! * \mathsf{tt} \wedge \textbf{new}\; z.P_0\\
&\Rightarrow \exists z.\mathcal{N}\langle z\rangle.(1z! * 1z? * 1\mathsf{c}! * \mathsf{tt} \wedge P_0)\\
&\Rightarrow \exists z.\mathcal{N}\langle z\rangle.\varphi_0\\
&\Rightarrow \exists z.\mathcal{N}\langle z\rangle.\mathsf{sorted}
\end{aligned}
$$

7. Related work

Our work lies at the intersection of several different approaches to concurrency, so there is related literature along several axes.

7.1 Separation logic

As mentioned in the introduction, Hoare and O'Hearn developed a semantics for a variant of the π-calculus [HO08] that served as an inspiration for this paper. Their semantics is related to separation logic in an abstract way: through *ternary relations*. As they explain, any ternary relation (such as our $\oplus$) gives rise to a separating conjunction (such as our $*$). In light of this, they introduce separating conjunctions for both parallel and sequential composition. These operators suffice to give a denotational semantics for the π-calculus, restricted to point-to-point communication. There is an underlying ownership model, in which channel-direction pairs are owned by exactly one process and ownership can be transferred during synchronization.

The paper gives an insightful model of point-to-point π-calculus, but only hints at how the model might be used to reason about processes in a new way, and does not show how reasoning techniques from separation logic might apply. Aside from our technical contributions, an intellectual contribution of our work is recognizing that resource-based reasoning can be applied to the problem of interference in the π-calculus. As Hoare and O'Hearn say, "the general hope is that models of circumscribed resources can lead to modular and tractable methods of reasoning about concurrent processes." We see our paper as a step in this direction.

Pym and Tofts give a calculus, SCRP, and a logic, MBI, for reasoning about message-passing processes and resources [PT06]. Their work is parametrized over a resource model, and is thus quite general. Resources in their model can, in our jargon, only cause silent failure; processes do not fault for lack of resources. SCRP is based on synchronous CCS, and thus does not (directly) support passing channels as messages. A significant difference from our approach is the role of the separating conjunction $*$ in the logic. For us, $*$ can only be used to conjoin resource predicates, which are assertions about a fixed point in time. In MBI, $*$ conjoins arbitrary temporal formulas, and plays a role similar to parallel composition in our logic. It is therefore somewhat difficult to relate our model and logic to SCRP and MBI. In particular, it is not clear whether SCRP and MBI can be adapted to reason about channel ownership and interference in the style of this paper.

Several papers in the separation logic literature influenced the construction of our model and logic. Calcagno, O'Hearn, and Yang developed an account of separation logic based on an abstract notion of resources and local actions [COY07]. Their work clarified the notion of faulting and gave it an order-theoretic treatment, which we adopted in our model. We take these ideas a step further by highlighting the importance of silent failure as well. Boyland's fractional permissions [Boy03], and its adaptation to separation logic [BCOP05], are of central importance to this paper, although our interpretation of channel ownership is new. Finally, Brookes's fair action traces [Bro02, Bro07], which provided the first model for concurrent separation logic, strongly influenced our trace model.

7.2 Type systems for mobile processes

There are a substantial number of type systems for the π-calculus, aimed at a variety of goals. Early on, it was recognized that types permitting or denying processes from sending or receiving along a specific channel are useful. The challenge has been to treat such capabilities in a dynamic way, so that processes can gain (and perhaps lose) capabilities over time. To do this, process types must reflect at least some amount of process behavior [PS93, HVK98]. In the extreme, types resemble temporal logic formulas [IK01, Cai07].

The type system most closely related to our logic is Terauchi and Aiken's capability calculus [TA08]. While the goal of their type system is to ensure that processes behave deterministically—a very strong kind of noninterference—the mechanisms they use to do this resemble our permissions model to some degree. In particular, capabilities can be transferred during synchronization. However, channel capabilities cannot be fractionally owned.

Session types [HVK98] also embody strong noninterference assumptions, by treating session channels *linearly*. In such a type discipline, session channels must be used exactly once in a given context, which immediately excludes competition along session channels. Multiparty session types [HYC08] allow multiple processes to coordinate usage of session channels, but still exclude competition. Our logic, in contrast, allows fine-grained control over when competition for a channel is allowed and when it is not.

Because the purpose of a type system is usually to provide a tractable abstraction of behavior, type systems are at a disadvantage when compared to a logic that can express the precise behavior of any process. A natural question is whether a type system can be built on our notion of resources, abstracting away from full temporal logic. Caires has recently explored the relationship between logics and type systems for process calculi [Cai07], and we hope to use his insights to answer this question.

7.3 Compositional temporal logic

Temporal logics are defined using a satisfaction relation $\mathcal{S} \models \varphi$ between "structures" and formulas. The satisfaction relation is defined compositionally in the syntax of formulas. Structures vary from logic to logic, but are generally either graphs or traces. Usually we think of structures as arising from the operational semantics of a programming language, so that we might write $[\![P]\!] \models \varphi$, where P is a program and $[\![-]\!]$ gives its semantics as a structure. Thus, there are *two* syntactic structures at play: that of programs and that of formulas.

It was recognized early on that temporal logic is not compositional in the syntax of programs. In most temporal logics, knowing the formulas that $[\![P_1]\!]$ and $[\![P_2]\!]$ satisfy is not sufficient to say anything about the formulas that $[\![P_1; P_2]\!]$ satisfies, for example. This fact is not surprising, because temporal logic captures "global" properties of programs. But compositionality is very desirable because it allows verification to be done in a modular way.

Barringer, Kuiper, and Pnueli developed an early approach to compositional temporal reasoning by giving a *temporal semantics* for each program operator [BKP84]. The temporal semantics capture the strongest temporal logic formula true of each construct in a compositional way. Lamport uses a similar approach in his *temporal logic of actions*, which is designed to support compositional, refinement-based reasoning [Lam94, AL95]. In both cases, the problem of proving $[\![P]\!] \models \varphi$ is reduced to the problem of proving the validity of $\lfloor P \rfloor \Rightarrow \varphi$, where $\lfloor - \rfloor$ gives the temporal semantics of P. This is very similar to way we interpret processes as formulas. The most significant difference is that we have internalized parallel composition into the logic, relieving us from the burden of specifying it as a temporal formula, which in turn allows us to use a simpler temporal logic. There has also been work from the opposite direction, starting with a process calculus and adding temporal operators. This approach has been carried out both operationally [CL02] and denotationally [BVK95].

An important line of work in compositional logics for concurrency is *assume-guarantee* reasoning [Jon83, AL95, Roe01]. In this style of reasoning, specifications make guarantees about the behavior of a process relative to assumptions about the behavior of its environments. In the most general case, both the assumptions and the guarantees are arbitrary temporal formulas, making the technique extremely expressive. Assume-guarantee reasoning has recently been combined with separation logic [Fen09, VP07], which greatly decreases the annotation burden. In general, assume-guarantee reasoning provides a much more powerful way of describing and ruling out interference than the logic we have presented. The downside of this expressive power, however, is that assume-guarantee specifications are more complex than the resource-based reasoning we have proposed. For common cases of interference, we believe resource reasoning provides a lighter-weight approach.

7.4 Logics for mobile processes

Most logics for process calculi are *Hennessy-Milner logics* [HM80]: they are interpreted directly over the LTS given by an operational semantics and characterize bisimilarity for the LTS. It is possible, but difficult, to give compositional proof rules for such logics when mobility is involved [Dam03].

Caires and Cardelli's spatial/temporal logic for the π-calculus does not deal with a notion of resources, but it does deal with mobility [CC01]. The logic has operators for parallel composition and name restriction and includes a fresh name quantifier. Parallel composition in the logic is intensional: a process satisfies $\varphi|\psi$ only if it is (structurally equivalent to) a parallel composition of processes satisfying φ and ψ respectively. Our logic, on the other hand, treats parallel composition extensionally, so that a process satisfies $\varphi|\psi$ only if its behavior refines $[\![\varphi|\psi]\!]$. The combination of the fresh name quantifier and the hiding operator allow Cardelli and Caires's logic to express channel privacy and mobility. However, as in traditional π-calculus, once a channel has been exposed to the environment, interference is always possible, and hence must be explicitly proscribed. Our use of resources allows interference to be implicitly ruled out. It is not clear whether the traditional, scope-based treatment of channels can be reconciled with the resource-based treatment of this paper.

8. Conclusion and future work

The π-calculus provides a fundamental model of concurrency in which the means of communication between processes can evolve over time. In this paper, we have presented a logic for the π-calculus that treats channels as resources and can use resource ownership to rule out interference between processes. To accomplish this, we have combined ideas from separation and temporal logic in a way that allows compositional reasoning about processes. As a byproduct of this effort, we have also given a denotational semantics for a variant of the π-calculus.

There are many questions left open by this work. What is the appropriate proof theory for our logic? Is it possible to build a type system based on our notion of resources? How does our technique interact with other approaches, such as assume-guarantee reasoning, that deal with environmental assumptions? Can our model be adapted to give a denotational semantics for the standard π-calculus? We also hope to adapt our work to other process calculi, in particular the join calculus [FG96]. More broadly, we have chosen to view channels as resources, but there are many other possibilities. One promising direction is to examine session types [HVK98] from the perspective of resources.

Acknowledgments

Thanks to Alec Heller, Sam Tobin-Hochstadt, and Dimitris Vardoulakis for feedback on drafts of this paper, and particularly to Dan Brown for his careful reading and suggestions. Thanks also to Riccardo Pucella and Panagiotis Manolios for helpful discussions.

References

[AL95] Martin Abadi and Leslie Lamport. Conjoining specifications. *TOPLAS*, 1995.

[BCOP05] Richard Bornat, Cristiano Calcagno, Peter O'Hearn, and Matthew Parkinson. Permission accounting in separation logic. In *POPL*, 2005.

[BKP84] Howard Barringer, Ruurd Kuiper, and Amir Pnueli. Now you may compose temporal logic specifications. In *STOC*, 1984.

[Boy03] John Boyland. Checking interference with fractional permissions. In *SAS*, 2003.

[BR84] Stephen D. Brookes and A. W. Roscoe. An improved failures model for communicating processes. In *Seminar on Concurrency*, volume 197, 1984.

[Bro02] Stephen Brookes. Traces, pomsets, fairness and full abstraction for communicating processes. In *CONCUR*, pages 45 –71, 2002.

[Bro07] Stephen Brookes. A semantics for concurrent separation logic. *TCS*, 375:227–270, May 2007.

[BS01] Julian Bradfield and Colin Stirling. *Modal logics and mu-calculi: an introduction*. Elsevier, 2001.

[BVK95] Rudolf Berghammer and Burghard Von Karger. Formal derivation of csp programs from temporal specifications. In *MPC*, pages 180 – 196, 1995.

[Cai07] Luís Caires. Logical semantics of types for concurrency. In *CALCO*, pages 16 – 35, 2007.

[CC01] Luís Caires and Luca Cardelli. A spatial logic for concurrency (part i). In *TACS*, pages 1 – 37, 2001.

[CL02] Rance Cleaveland and Gerald Luttgen. A logical process calculus. In *EXPRESS*, 2002.

[COY07] Cristiano Calcagno, Peter W. O'Hearn, and Hongseok Yang. Local action and abstract separation logic. In *LICS*, 2007.

[Dam03] Mads Dam. *Proof systems for π-calculus logics*. Kluwer Academic Publishers, 2003.

[Fen09] Xinyu Feng. Local rely-guarantee reasoning. In *POPL*, page 12, 2009.

[FG96] Cedric Fournet and Georges Gonthier. The reflexive cham and the join-calculus. In *POPL*, pages 372–385, 1996.

[HM80] Matthew Hennessy and Robin Milner. On observing nondeterminism and concurrency. In *ICALP*, 1980.

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

[HVK98] Kohei Honda, Vasco Thudichum Vasconcelos, and Makoto Kubo. Language primitives and type discipline for structured communication-based programming. In *ESOP*, 1998.

[HYC08] Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multi-party asynchronous session types. In *POPL*, 2008.

[IK01] Atsushi Igarashi and Naoki Kobayashi. A generic type system for the pi-calculus. In *POPL*, 2001.

[Jon83] C. B. Jones. Tentative steps toward a development method for interfering programs. *TOPLAS*, 5, 1983.

[Lam94] Leslie Lamport. The temporal logic of actions. *TOPLAS*, 16, 1994.

[NH84] R. De Nicola and M. C. B. Hennessy. Testing equivalences for processes. In *TCS*, pages 83–133, 1984.

[O'H07] Peter O'Hearn. Resources, concurrency, and local reasoning. In *TCS*, volume 375, pages 271–307, May 2007.

[PS93] B. Pierce and D. Sangiorgi. Typing and subtyping for mobile processes. In *LICS*, volume 24, pages 376 – 385, 1993.

[PT06] David Pym and Chris Tofts. A calculus and logic of resources and processes. *FAC*, 18:495 – 517, 2006.

[RB90] A. Roscoe and Geoff Barrett. Unbounded nondeterminism in csp. In *MFPS*, pages 160 – 193, 1990.

[Rey02] J.C. Reynolds. Separation logic: a logic for shared mutable data structures. In *LICS*, 2002.

[Roe01] Willem-Paul De Roever. *Concurrency verification: introduction to compositional and noncompositional methods*. Cambridge University Press, 2001.

[SW01] Davide Sangiorgi and David Walker. *The Pi Calculus*. Cambridge University Press, 2001.

[TA08] Tachio Terauchi and Alex Aiken. A capability calculus for concurrency and determinism. *TOPLAS*, 30, August 2008.

[VP07] Viktor Vafeiadis and Matthew Parkinson. A marriage of rely/guarantee and separation logic. In *CONCUR*, 2007.